МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Многоядерные процессоры, параллельное программирование, ПЛИС, системы обработки сигналов

Сборник статей II региональной научно-практической конференции

Барнаул 2012

УДК 004.272.4(07); 004.318

В сборнике статей II региональной научно-практической конференции «Многоядерные процессоры, параллельное программирование, ПЛИС, системы обработки сигналов», проведенной 28 февраля 2012г. на базе ФГБОУ ВПО «Алтайский государственный университет», представлены материалы, посвященные системным и практическим проблемам параллельных вычислений, разработке высокопроизводительных систем на базе ПЛИС и многоядерных процессоров, системным и практическим проблемам применения ПЛИС и сигнальных процессоров в технике приема и обработки сигналов.

В конференции приняли участие технические специалисты предприятий, преподаватели, аспиранты и студенты высших учебных заведений Российской Федерации. Среди участников конференции представители ведущих мировых компаний – Intel Inc. и Xilinx.

Официальным спонсором конференции выступила ведущая компания Алтайского края в области информационных технологий **ООО "НТЦ Галэкс".**

Составители: А.В. Калачев, В.В. Белозерских

СОДЕРЖАНИЕ

Часть1. Параллельные вычисления

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ С ИСПОЛЬЗОВАНИЕМ MS NETFRAMEWORK 4.0 PARALLEL EXTENSIONS НА ПРИМЕРЕ БПФ

Д.В. Попов, А.Н. Куликов

ФГБОУ ВПО «Алтайский государственный университет»

Многопроцессорные машины становятся стандартом по мере того, как скорость отдельных процессоров растет всё медленнее и медленнее. Следовательно, чтобы повысить производительность, программа должна работать параллельно на нескольких процессорах. Но единой технологии раскрывающей потенциал многопроцессорных систем, к сожалению, нет, и большинство приложений попрежнему используют только одно ядро процессора.

Чтобы существенно облегчить написание сопровождаемого кода и автоматически использовать несколько процессоров компанией Microsoft был разработан набор инструментов параллельного выполнения заданий Parallel Extensions, входящих в состав набора Microsoft Net Framework 4. Используя этот инструментарий, можно выделять делегаты (блоки кода), пригодные для распараллеливания, в существующем последовательном коде. Выделенные таким образом параллельные задания будут выполняться одновременно на всех доступных процессорах.

Parallel Extensions состоит из трех основных компонентов:

Task Parallel Library (TPL) — предоставляет такие императивные методы как:

- Parallel.For() - метод позволяет выполнить делегат параллельно в несколько потоков заданное количество итераций;

- Parallel.ForEach() - параллельно выполняет делегат над элементами коллекции IEnumerable<TSource>;
- Parallel.Invoke() - принимает набор делегатов Action и обеспечивает их одновременное выполнение.

Parallel LINQ (PLINQ) — надстройка над LINQ to Objects и LINQ to XML, позволяющая выполнять параллельные запросы. В большинстве случаев достаточно в начале запроса написать AsParallel() для того, чтобы все последующие операторы выполнялись параллельно.

Coordination Data Structures (CDS) — набор структур, который используется для синхронизации и координации выполнения параллельных задач.

Новый пул потоков был переработан и оптимизирован для управления большим числом рабочих потоков, и реализация очереди задания для каждого из них изменена. Теперь кроме глобальной очереди задач каждый поток имеет свою локальную очередь. Это позволяет более эффективно управлять задачами. Например, первый поток выполняет очередную задачу, а второй успел опустошить свою очередь. Тогда он может "украсть" задачу из очереди первого потока, тем самым распределив нагрузку. При этом не будет обращения к глобальной очереди задач, что уменьшит нагрузку на неё. Эти действия будут прозрачны для прикладного программиста.

Создание и завершение потоков выполняется библиотекой автоматически. [2-3]

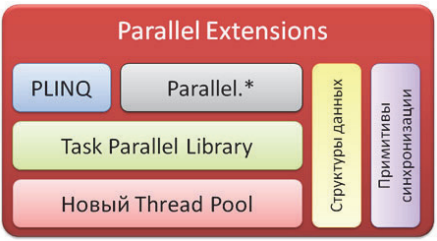


Рисунок 1. Структура Parallel Extensions.

Продемонстрируем работу расширения Parallel Extensions на примере метода спектрального анализа, в основе которого лежит алгоритм быстрого преобразования Фурье, способом прореживания по времени.

Считая, что количество отсчетов N делится на 2, представим двумя суммами, соответствующими четным и нечетным значениям n:

$$A(k) = \sum_{n=0}^{N-1} a(n) e^{-j2\pi nk/N} = \sum_{n=0}^{N/2-1} a(2n) e^{-j2\pi 2nk/N} + \sum_{n=0}^{N/2-1} a(2n+1) e^{-j2\pi(2n+1)k/N} = \sum_{n=0}^{N/2-1} a(2n) W^{2nk} +$$

$$+ \sum_{n=0}^{N/2-1} a(2n+1) W^{2nk} = B(k) + W^k C(k) \qquad (1)$$

где *B(k)* и *C(k)* — суммы соответственно первого и второго слагаемых.

Таким образом, вычисление N-точечного преобразования A(k), k=0,1,..., N — 1, можно произвести путем вычисления двух N/2-точечных преобразований: B(k), k=0,2, ..., N—2 и C(k), k=1,3, ..., N—1, с последующим их объединением по формуле (1).

Если N/2 в свою очередь делится на 2, то вычисление каждого из преобразований B(k) и C(k) можно также свести к двум N/4-точечным преобразованиям, что вызовет дополнительное уменьшение требуемого количества операций умножения и т.д.

Если N представляется целой степенью двух ($N = 2^m$), то вычисления разбиваются на $\tau = \log_2 N$ этапов, в каждом из которых требуется N/2 умножений. Таким образом, общее количество умножений равно $\frac{N}{2}\log_2 N$. Например, при $N = 2^{10} = 1024$-точечном преобразовании число умножений окажется равным $0{,}5 * 1024 * 10 \approx 10^4/2$, в то время как при N - точечном ДПФ потребовалось бы $N^2 \approx 10^6$ операций умножения. Как видим, БПФ обеспечивает существенное сокращение объема вычислений.[1]

Ниже представлены блок схемы БПФ способом прореживания по времени:

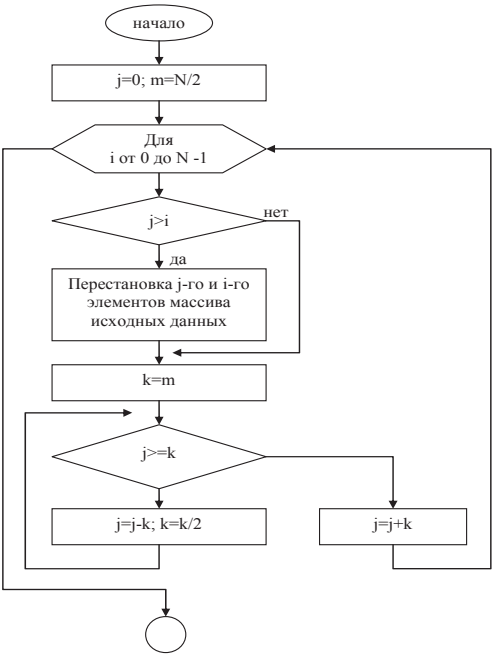


Рисунок 2. Быстрый алгоритм двоично-инверсных перестановок.

В отличие от алгоритма двоично-инверсных перестановок, реализованного с итерационной зависимостью по данным и не позволяющей применить библиотеку параллельных расширений, алгоритм БПФ выполняется параллельно с использованием метода Parallel.For() (см. рис. 4) со следующими условиями ($m=\log_2 N$):

- при i = 0,..., m/2 (целая часть) выполняется алгоритм с параллельным циклом 2 уровня вложенности;

при i = m/2,..., m (целая часть) выполняется алгоритм с параллельным циклом 3 уровня вложенности.

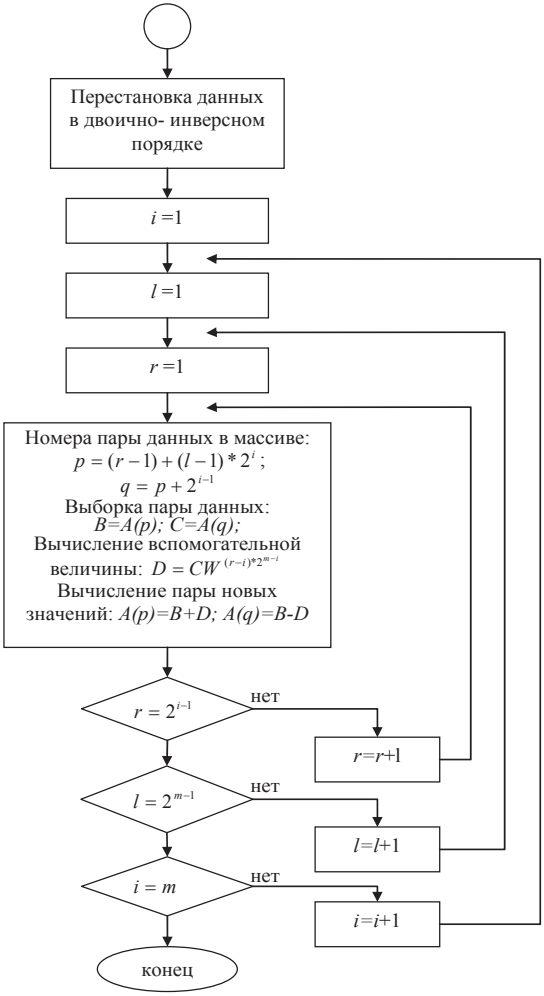


Рисунок 3. Схема алгоритма БПФ.

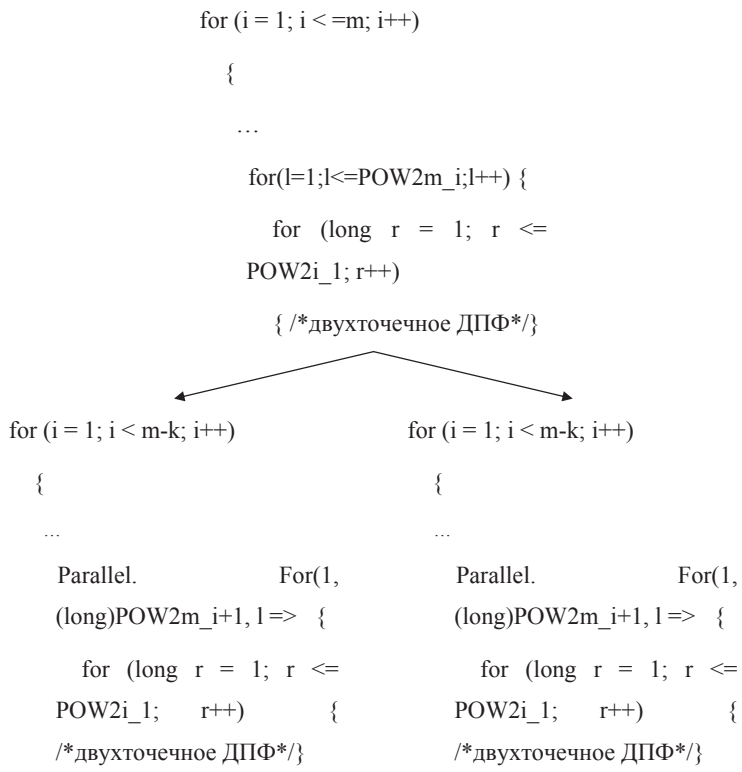


Рисунок 4. Применение метода Parallel.For() в быстром преобразовании Фурье.

В первую очередь это обусловлено тем, что на каждом шаге внешнего цикла, количество итерации цикла 2 уровня уменьшается, а цикла 3 уровня увеличивается в 2 раза. В результате алгоритм позволяет максимально задействовать вычислительные ресурсы на начальных и конечных этапах БПФ. Процесс параллельного выполнения для 8 отсчетов отражен на рисунке 5. Блоки, выделенные одним цветом, на каждом из этапов соответствуют одному из параллельно выполняемых процессов (потоков).

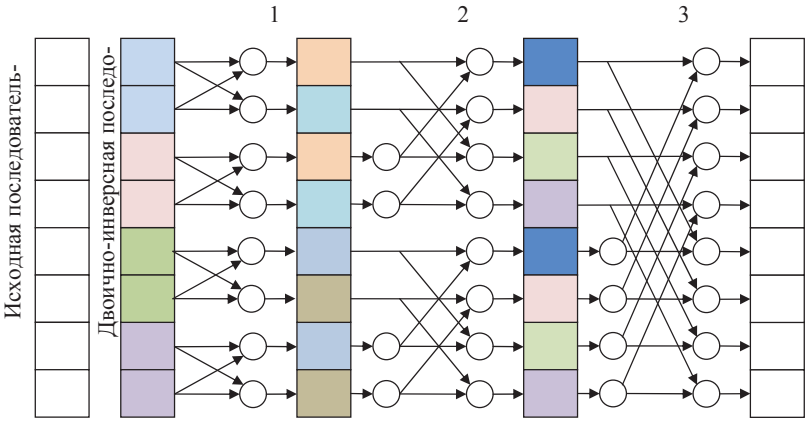


Рисунок 5. Параллельное выполнение БПФ.

Следует сразу заметить, что выигрыш при использовании расширения оправдывается при большом количестве вычислений, поскольку часть ресурсов тратиться на создание параллельно выполняемых потоков. Исходя из этого ограничения, был проведен тест производительности алгоритма при последовательном и параллельном режимах, и вычислен коэффициент, характеризующий степень увеличения производительности вычислений.

Библиотека Parallel Extensions предлагает все внимание разработчика целиком сосредоточить на задаче, упрощая распараллеливание программ, и предоставляет уже готовый, до некоторой степени универсальный подход. Это уменьшает затраты на распараллеливание, что в конечном счёте позволяет сделать разработку дешевле по экономическим и временным показателям.

Таблица 1

Результаты тестирования последовательного и параллельного БПФ

№	Процессор	Время последовательного преобразования (мс.)	Время параллельного преобразования (мс.)	Коэффициент.
1	Intel Core 2 Duo T5800 2.0 ГГц	671	420	1.6
2	Intel Pentium P6200 2.13 ГГц	600	400	1.5
3	Intel Core 2 Duo T7500 2.2 ГГц	600	380	1.6
4	Intel Atom N450 1.66 ГГц	1950	1370	1.4
5	Intel Core i3 2350M 2.3ГГц	570	250	2.3

(При тестировании использовалось 524288 отсчетов исходного сигнала.)

Литература

1. Калабеков Б.А. Микропроцессоры и их применение в системах обработки и передачи сигналов. – М.: Радио и связь, 1988. – 366 с.
2. Журнал MSDN Magazine. [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/ru-ru/magazine/cc163340.aspx свободный. – Загл. С экрана – Яз. Рус.
3. Интерфейс [Электронный ресурс] – Режим доступа: http://www.interface.ru/home.asp?artId=23095 свободный. – Загл. С экрана – Яз. Рус.

НОВЫЙ МЕТОД ПОИСКА ПРОСТЫХ ОСНОВАНИЙ МОДУЛЯРНОЙ СИСТЕМЫ СЧИСЛЕНИЯ

В.В. Чернобровкин

Сургутский государственный университет, факультет информатики и вычислительной техники

Для разрешения проблем вычислений в больших и сверхбольших компьютерных диапазонах необходимо применять специальные машинные арифметики, созданные на базе модулярных представлениях данных [1].

Одним из таких представлений является нормированный ранг Z_A модулярной величины $A(mod\ P)$, диапазон которого лежит в пределах $2^{30}, ..., 2^{31}$. Где A-натуральное число, делящееся с остатком на основание P - которое является простым числом.

В настоящее время имеется множество различных методов поиска простых оснований, иначе говоря, простых чисел, лучшими, из которых являются:

- *p*-Метод Полларда
- Метод квадратичного решета
- Метод решета числового поля

p-Метод Полларда базируется на малой теореме Ферма. Суть метода Полларда состоит в том, что достаточно каждый раз вычислять новое значение x, по формуле: $x = x^y\ mod\ n$, где y - произвольные натуральные числа. А по истечении некоторого количества итераций, проверять значение $a = (x - 1, n)$. Если результат окажется $1 < a < n$, то значит a - нетривиальный делитель числа n. Метод очень эффективен тогда, когда хотя бы один из делителей n , уменьшенный на 1, разлагается на малые простые сомножители.

Основная идея метода квадратичного решета:

В качестве факторной базы B берется множество простых чисел, состоящее из $p = 2$ и всех таких нечетных простых чисел p, не превосходящих заданной границы P (которая выбирается из соображений оптимальности), что n — квадра-

тичный вычет по модулю p. Множество S целых чисел, в котором ищутся B-числа (B-число — целое число, делящееся только на простые числа из B) выглядит следующим образом:

$$S = \left\{ t^2 - n \middle| [\sqrt{n}] + 1 \le t \le [\sqrt{n}] + A \right\}$$

Далее, вместо того, чтобы брать одно за другим, и делить его на простые числа из B, берутся одно за другим каждое, и проверяется делимость на p (и его степени) одновременно для всех $s \in S$.

Метод решета числового поля заключается в переходе в некоторое алгебраическое расширение $Q(\xi)$ поля рациональных чисел Q и перебором a, $b \in Z$, $(a,b)=1$ ищутся элементы $a+\xi b$, которые разлагаются на множители в некоторой факторной базе. Перебор a и b делается с помощью специального просеивания, наподобие метода квадратичного решета. Затем с помощью одного из методов исключения находят соотношение $x^2 \equiv y^2 (mod\ n)$. Тогда если $1<(x-\varepsilon y, n)<n$ для некоторого $\varepsilon \in \{-1, 1\}$, то $(x-\varepsilon y, n)$ - искомый нетривиальный делитель n[3].

Все выше перечисленные методы имеют эффективность. Но для некоторых из них есть один недостаток. Заключается он в том, что идет перебор всех чисел (четных и нечетных) подряд. На что в свою очередь идут временные затраты.

Метод распараллеливания поиска простых оснований модулярной системы по их окончаниям

В настоящее время известно, что все простые числа независимо от их разрядности оканчиваются на одну из четырех цифр – *1, 3, 7* и *9* кроме *2*-ки и *5*-ки. Суть метода в том, что в некотором заданном диапазоне $Z^n, \ldots, Z^{n+1}$ параллельно находятся числа, младшие разряды которых $i_0 = 1, 3, 7, 9$. Далее создаются файлы для чисел каждого окончания. Во всех файлах задается один и тот же диапазон, в котором происходит поиск простых чисел. Т.е. к примеру, в файле, где все числа, у которых $i_0 = 1$, ищутся простые числа также оканчивающиеся на 1-цу.

Алгоритм метода распараллеливания поиска:

Шаг 1. Для каждого из четырех файлов задать диапазон $[Z^n, \ldots, Z^{n+1}]$

Шаг 2. Создать файлы для чисел с каждым из окончаний: *(F1, F3, F7, F9)*

Шаг 3. В каждом из файлов формировать числа, где i_0= *1; 3; 7; 9*

Шаг 4. Задать поиск простых чисел.

Шаг 5. В “контрольный” файл записать все полученные результаты

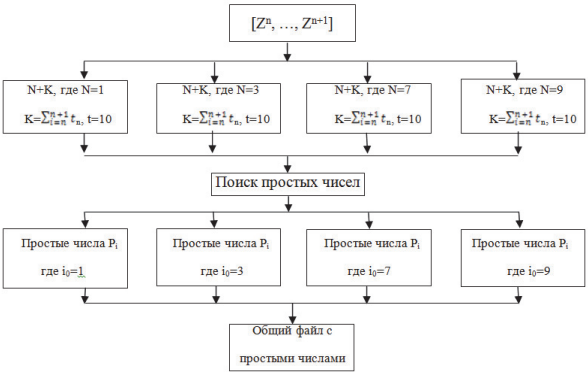


Рисунок 1. Схема метода распараллеливания поиска.

Из вышесказанного видно, что в натуральном ряду $Z=0,\ldots,\infty$, всех нечетных чисел с такими окончаниями всего 40 процентов, а цикл нахождения равен N+10, где N = 1; 3; 7; 9, например: 21, 31, 41 и т.д. Что значительно (в 4 раза) облегчает поиск простых чисел в заданном диапазоне. Так как нахождение простых чисел идет параллельно в каждом из четырех файлов. Из этого вытекает эффективность алгоритмов отыскивающих простые числа.

На основе вышеприведенного метода для создания будущего программного инструментария были отысканы простые основания из диапазона $2^{30}, \ldots, 2^{31}$. Ниже приведены таблицы с простыми числами (основаниями) вычисленные ближе к верхней границе этого диапазона. В точности во всех таблицах представлены последние 288 из 1000 найденных простых чисел. Р-означает простое число, а цифра – его порядковый номер

Таблица 1

Простые числа, оканчивающиеся на 1-цу

2147477021	P719	2147478481	P774	2147478911	P803	2147479601	P841	2147480651	P880	2147482021	P937
2147477201	P724	2147478521	P778	2147478961	P807	2147479681	P845	2147480921	P890	2147482081	P939
2147477531	P738	2147478581	P781	2147479031	P811	2147479751	P846	2147480971	P894	2147482121	P940
2147477681	P742	2147478601	P782	2147479121	P816	2147479781	P849	2147481031	P897	2147482231	P942
2147477701	P745	2147478611	P783	2147479171	P819	2147480011	P858	2147481071	P899	2147482361	P947
2147477851	P750	2147478661	P788	2147479231	P821	2147480161	P860	2147481151	P901	2147482481	P951
2147477861	P751	2147478701	P790	2147479361	P827	2147480311	P867	2147481311	P910	2147482501	P952
		2147482801	P963	2147482811	P964	2147482921	P971	2147482951	P974	2147483171	P995

68 простых чисел

Таблица 2

Простые числа, оканчивающиеся на 3-ку

2147476663	P700	2147477513	P737	2147478373	P772	2147479133	P818	2147481053	P898	2147482063	P938
2147476763	P704	2147477533	P739	2147478503	P776	2147479273	P823	2147481173	P902	2147482223	P941
2147476823	P709	2147477833	P749	2147478563	P779	2147479403	P829	2147481263	P907	2147482273	P944
2147476963	P717	2147477873	P752	2147478653	P786	2147479513	P834	2147481283	P909	2147482583	P955
2147477113	P722	2147477933	P755	2147478673	P789	2147479573	P839	2147481353	P913	2147482663	P958
2147477203	P725	2147477953	P756	2147478703	P791	2147479753	P847	2147481373	P916	2147482763	P962
2147477273	P730	2147478013	P758	2147478763	P797	2147479823	P852	2147481563	P921	2147482873	P969
2147477323	P731	2147478083	P762	2147478863	P800	2147480623	P879	2147481673	P924	2147483033	P988
2147477473	P735	2147478133	P765	2147479013	P810	2147480683	P882	2147481863	P928	2147483053	P989
2147477503	P736	2147478253	P767	2147479063	P813	2147480723	P884	2147481883	P929	2147483123	P993
										2147483323	P1000

61 простое число

Таблица 3

Простые числа, оканчивающиеся на 7-ку

2147476687	P701	2147478017	P759	2147479507	P833	2147480297	P865	2147481317	P911	2147482417	P950
2147476777	P706	2147478127	P764	2147479517	P835	2147480327	P868	2147481337	P912	2147482507	P953
2147476897	P711	2147478497	P775	2147479547	P837	2147480437	P871	2147481367	P915	2147482577	P954
2147476927	P713	2147478517	P777	2147479637	P843	2147480507	P874	2147481487	P917	2147482697	P960
2147476937	P715	2147478647	P784	2147479657	P844	2147480527	P876	2147481797	P925	2147482817	P965
2147477107	P721	2147478727	P794	2147479757	P848	2147480677	P881	2147481827	P927	2147482867	P968
2147477207	P726	2147478937	P805	2147479787	P850	2147480707	P883	2147481907	P932	2147482877	P970
2147477237	P728	2147478967	P808	2147479897	P854	2147480747	P885	2147481937	P933	2147482937	P972
2147477467	P734	2147478997	P809	2147479907	P855	2147480837	P886	2147481967	P935	2147483077	P992
2147477627	P741	2147479057	P812	2147479937	P856	2147480897	P888	2147481997	P936	2147483137	P994
2147477687	P743	2147479097	P815	2147480197	P861	2147480927	P891	2147482237	P943	2147483237	P997
2147477737	P746	2147479307	P824	2147480207	P862	2147480957	P892	2147482327	P945		
2147477807	P747	2147479447	P831	2147480227	P864	2147481247	P906	2147482367	P948		

76 простых чисел

Таблица 4

Простые числа, оканчивающиеся на 9-ку

2147476699	P702	2147477599	P740	2147478569	P780	2147479339	P825	2147480429	P870	2147481359	P914
2147476739	P703	2147477699	P744	2147478649	P785	2147479349	P826	2147480459	P872	2147481499	P918
2147476769	P705	2147477809	P748	2147478659	P787	2147479489	P832	2147480519	P875	2147481509	P919
2147476789	P707	2147477879	P753	2147478719	P792	2147479549	P838	2147480849	P887	2147481529	P920
2147476819	P708	2147477989	P757	2147478859	P799	2147479589	P840	2147480899	P889	2147481629	P923
2147476899	P712	2147478049	P760	2147478889	P801	2147479619	P842	2147480969	P893	2147481899	P930
2147476979	P718	2147478079	P761	2147478899	P802	2147479819	P851	2147480989	P895	2147481949	P934
2147477029	P720	2147478089	P763	2147478919	P804	2147479879	P853	2147481019	P896	2147482349	P946
2147477159	P723	2147478149	P766	2147478959	P806	2147480009	P857	2147481139	P900	2147482409	P949
2147477209	P727	2147483029	P975	2147479079	P814	2147480039	P859	2147481179	P903	2147482739	P961
2147477249	P729	2147478259	P768	2147479129	P817	2147480219	P863	2147481199	P904	2147482819	P966
2147477399	P732	2147478299	P769	2147477189	P820	2147480299	P866	2147481209	P905	2147482859	P967
2147477419	P733	2147478349	P771	2147479259	P822	2147480369	P869	2147481269	P908	2147482949	P973
				2147483059	P990	2147483069	P991	2147483179	P996	2147483249	P998

83 простых числа.

Как видно из таблиц распределение простых чисел идет неравномерно. Больше всего чисел, у которых i_0=9. Но некоторую закономерность удалось найти. Если взять весь диапазон 2^{30}, . . ., 2^{31} поделить на 50 то в каждом промежутке, состоящем из этих натуральных 50-ти десятиразрядных чисел имеется как минимум одно простое.

Ниже приведен общий график появления простых чисел в заданном диапазоне.

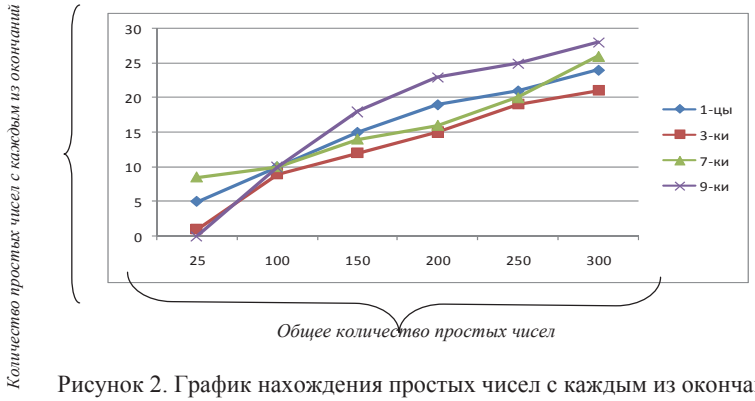


Рисунок 2. График нахождения простых чисел с каждым из окончаний.

Поиск простых чисел производился в диапазоне, состоящем из 14400 десятиразрядных натуральных чисел из которого были исключены все четные числа и числа, у которых i_0=5. В итоге получилось 5760 десятиразрядных чисел. Если взять соотношение итогового числа и чисел с каждым из окончаний, то получаться следующие результаты: 5760/68 ≈ 1 простое число из каждых 211, где i_0=1; 5760/61 ≈ 1 из 236, где i_0=3; 5760/76 ≈ 1 из 189, где i_0=7; 5760/83 ≈ 1 из 173, где i_0=9. Всего в 5760 десятиразрядных натуральных числах - 288 простых. Из полученных результатов можно проследить некие закономерности, по которым можно искать простые числа.

Литература

1. Инютин С.А. Исследование критичности при вычислении позиционных характеристик модулярной величины. – Сургут: Научный сборник СурГУ, 2011г.
2. Василенко О.Н. Теоретико-числовые алгоритмы в криптографии. -Москва: МЦНМО,2003 г.
3. Ноден П., Китте К. Алгебраическая алгоритмика – Москва: МИР, 1999 г.

ПРИМЕНЕНИЕ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ В ЗАДАЧЕ РАСПРОСТРАНЕНИЯ ЭЛЕКТРОМАГНИТНОЙ ВОЛНЫ ОТ НИТЕВИДНОГО ИСТОЧНИКА

П.Н. Уланов, П.М. Зацепин

ФГБОУ ВПО «Алтайский государственный университет»

В статье описано решение волнового уравнения нитевидного импульсного источника в свободном пространстве, приведено сравнение способов и методов распараллеливания расчетной программы, произведено сравнение расчетных времен расчетной программы в зависимости от количества расчетных потоков.

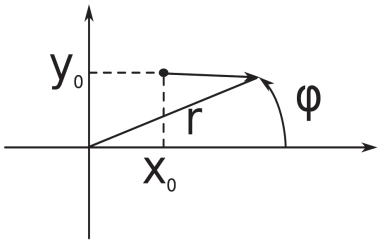


Рисунок 1. Геометрия задачи.

Ключевые слова: импульс, волновое уравнение, вейвлет-перобразование, MPI.

Геометрия задачи изображена на рисунке 1. Источник в виде нити электрического или магнитного тока, ориентированного вдоль оси Z находится в пространстве с координатами (x_0, y_0). Функции плотности тока имеют вид $j_z(x, y, z, t) = \delta(x - x_0, y - y_0) f(t)$. Временная зависимость функции плотности имеет импульсный характер. Задача является двухмерной, то есть $\frac{\partial}{\partial z} = 0$. Выполняются следующие начальные условия: $A_{z,t=0} = 0, \frac{\partial A_{z,t=0}}{\partial t} = 0$. Исходя из геометрии задачи, решение удобно искать в цилиндрической системе координат (r, φ, z), уравнение задачи в этом случае имеет вид:

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial A_z}{\partial r}\right)+\frac{1}{r^2}\frac{\partial^2 A_z}{\partial \varphi^2}-\frac{1}{c^2}\frac{\partial A_z^2}{\partial^2 t}=-\mu_0\delta(\vec{r}-\vec{r}_0)f(t) \quad (1)$$

В силу аксиальной симметрии задачи относительно источника, возможно преобразование исходной геометрии типа сдвиг-поворот, то есть параллельный перенос источника в начало координат. В результате уравнение (1) преобразуется к виду:

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial A_z}{\partial r}\right)-\frac{1}{c^2}\frac{\partial A_z^2}{\partial^2 t}=-\mu_0\delta(r)f(t) \quad (2)$$

Для решения таких задач обычно применяют либо разностные численные схемы, либо интегральные преобразования. Применение разностных схем приводит к непроизводительному расходу вычислительных ресурсов и в работе не рассматривается. Применение преобразования Фурье в данном случае требует аналитического продолжения искомой функции на отрицательную область по временной оси, что приводит к появлению в решении мнимой части, которая физически не оправдана, так как решения уравнений Максвелла являются вещественными. Применение преобразования Лапласа более оправдано, так как описывает решение начальной задачи. Кроме того, решение не дает физически спорных мнимых артефактов. Однако одним из существенных недостатков преобразования Лапласа является то, что изображения многих функций по Лапласу не существует. Эта ситуация характерна для многих практически значимых функций источников в задачах излучения, в том числе и для функций гауссова семейства.

Описанные выше недостатки отсутствуют у непрерывных вейвлет-преобразований, которые весьма популярны в последние пару десятков лет. Применим интегральное вейвлет-преобразование к рассматриваемой задаче. Возьмем $te^{-t^2/2}$ в качестве материнского вейвлета. После применения вейвлет-преобразования решение задачи примет следующий вид:

$$A_z(r,t) = \frac{1}{C_\psi}\int_{-\infty}^{\infty}\frac{ds}{s^2\sqrt{s}}\int_0^{\infty}dt_0 K_0\left(\frac{r}{c}\sqrt{\left(\frac{t-t_0}{s^2}\right)^2-\frac{3}{s^2}}\right)\frac{t-t_0}{s}e^{\frac{-(t-t_0)^2}{2s^2}}W(s,t_0) \qquad (3)$$

Здесь $W(s,t_0)$ — вейвлет-образ исходного импульсного источника. Как видно, для получения численного решения исходной задачи в виде зависимостей от времени и расстояния от источника необходимо рассчитать двумерный массив двойных интегралов с различными параметрами. Работа расчетной программы требует значительного расхода вычислительных ресурсов поэтому возникает проблема их сокращения. Совершенствование расчетных алгоритмов для однопроцессорных систем не приводит к заметному сокращению расхода ресурсов, зато использование многопоточности современных процессоров и многопроцессорных систем и применение параллельных алгоритмов может дать значительный прирост производительности. В настоящее время наиболее популярны алгоритмы распараллеливания OpenMP и MPI. В силу универсальности MPI (способность работать не только на многоядерных процессорах) в данной работе для расчетов применен этот алгоритм. Основные особенности программирования с использованием этой технологии следующие:

- На каждом потоке запускается своя версия программы, которая может определить общее количество потоков, а также номер своего потока.
- Потоки могут обмениваться сообщениями с использованием специальных синтаксических конструкций.

Обсудим способ распараллеливания расчетной программы. Согласно формуле (3), основной расчетной задачей является двукратный несобственный интеграл, следовательно, можно выделить несколько основных способов построения параллельного алгоритма:

1) Различные потоки рассчитывают искомый интеграл каждый в своей области параметров, в результате каждый поток получает часть результирующего массива решений интеграла, которую пересылает в главный поток. Главный поток собирает из частей результирующий массив.

2) Разделение на потоки областей интегрирования, в конце расчетов каждый поток имеет результирующий массив полного размера, который пересылается в главный поток. Главный поток получает результирующий массив путем поэлементного сложения полученных массивов.

3) Комбинированный способ.

Достоинство первого метода — наименьшее количество накладных расходов при пересылке массивов, достоинство второго метода в том, что распараллеливать необходимо только процедуру интегрирования, достоинство третьего способа в возможности распределения задачи по наибольшему количеству потоков. Недостатки любого способа — возможность возникновения дисбаланса в расчетных потоках. Второй способ наихудший с точки зрения увеличения доли последовательной части программы — пересылки данных. Недостаток третьего метода — сложность реализации. Таким образом, как наиболее предпочтительный, был выбран первый способ.

Отдельной проблемой для выбранного способа распараллеливания является распределение потоков программы по потокам системы. Здесь можно выделить два способа:

1) Количества потоков программы и системы совпадают, потоки программы распределяются соответственно по потокам системы.

2) Количество потоков программы намного превышает количество потоков системы. Главный поток системы динамически распределяет потоки программы по потокам системы, при этом, не участвуя в расчетах.

Первый способ хорошо работает и прост в реализации. Но для эффективной работы потоки должны быть хорошо сбалансированы. Второй способ может быть применен в том случае, если исходная задача разлагается на множество достаточно мелких подзадач, времена, выполнения которых существенно различны, то есть задача плохо сбалансирована.

В рассматриваемом случае задача может быть легко разложена на сбалансированные по времени выполнения подзадачи и, следовательно, более предпочтительным является первый метод. По классификации Флинна задача относится к

классу МКМД(MIMD) — множественный поток команд для получения множественного потока данных, а также к классу SIMD — одна программа для получения множественного потока данных.

Результаты

Тестовые расчеты производились на модели параллельного вычислителя, имеющей 24 расчетных потока, состоящей из шести двухъядерных процессоров с использованием технологии HyperThreading (“Нанокластер”). Результаты в виде зависимостей времени расчета от количества потоков изображены на рисунке 2.

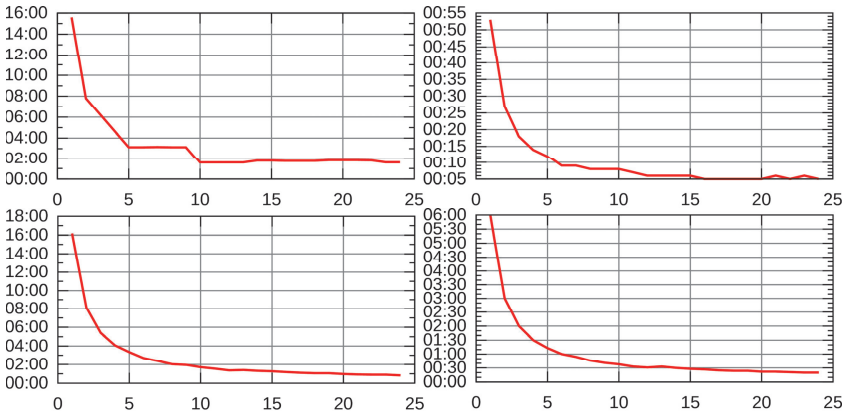


Рисунок 2. Зависимости расчетного времени от количества потоков.

На графиках четко прослеживается гиперболический характер зависимости расчетного времени от числа потоков, что соответствует описанным в законах Амдала и Густавсона-Барсиса зависимостям. На первом графике изображена зависимость для случая, когда задача может считаться лишь на 10 возможных потоках, остальные если даже и будут запущены, будут простаивать. Незначительное увеличение расчетного времени на более чем десяти потоках является погрешностью — это занятое операционной системой время на служебные опера-

ции. На втором графике ситуация та же, но для 16 потоков, эти случаи показывают, что бесконечное увеличение количества потоков не приведет к желаемому росту производительности параллельной программы для определенных плохих случаев.

На графиках 3 и 4 максимальное количество потоков задачи значительно превышает количество потоков системы, поэтому зависимости имеют боле гладкий вид. Расчетное время задачи при увеличении количества потоков не сразу выходит на постоянный результат, что говорит о небольшом вкладе накладных расходов расчетной программы. Это позволяет запускать ее на расчетных кластерах, имеющих достаточно большое количество расчетных потоков. Как видно из третьего графика, максимальный выигрыш по времени составил порядка 20 раз, что доказывает наличие влияющей на результат последовательной части расчетной программы.

Заключение

Рассмотрена задача расчета амплитуды компоненты векторного потенциала излучения импульсного нитевидного источника в свободном пространстве. Получена расчетная формула в виде обратного вейвлет-преобразования, представляющего собой повторный интеграл. Описаны методы и варианты распараллеливания расчетной программы. Разработана параллельная программа для численного расчета полученного интеграла. Произведено сравнение расчетных времен программы в зависимости от количества потоков системы. По сравнению с последовательной программой получен двадцатикратный выигрыш по времени расчета, что доказывает перспективность применения распараллеливания для уменьшения расчетного времени в подобных задачах.

ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ПОТОКА ИЗОБРАЖЕНИЙ БЫСТРОПРОТЕКАЮЩИХ ЗАПЫЛЕННЫХ ЧАСТИЦАМИ ПЛАЗМЕННЫХ СТРУЙ

В.И. Иордан, И.К. Рябченко

ФГБОУ ВПО «Алтайский государственный университет»

Введение

В настоящее время одной из перспективных технологий в области материаловедения является плазменное напыление покрытий с помощью микрокапель расплавов различных порошковых материалов. Для получения материалов с заданными свойствами и прогнозирования качества получаемого покрытия необходимо производить измерение параметров струи в технологической цепочке «плазмотрон-струя-покрытие». Наиболее важными параметрами струи напыления являются распределения скоростей и температур частиц в поперечных сечениях потока струи, а также и распределение объемной концентрации частиц (плотности частиц) не только в центре, но и на периферии струи.

Современные высокоскоростные цифровые камеры, основанные на современных матричных фотодиодных (ФД) приемниках или приемниках на основе приборов с зарядовой связью (ПЗС), представляющие собой сверхбольшие интегральные схемы (СБИС) с внутрикристальным процессором, позволяют регистрировать оптическое излучение нагретых частиц металлических порошков, движущихся в потоке струи со скоростью в диапазоне примерно от 100 и до 500 м/с. В силу скоротечности струи и сложной динамики изменения ее структуры (рис. 1) необходимо вести высокоскоростную съемку с высокой частотой кадров (порядка 500 кадров в секунду и более) и успевать обрабатывать поток огромного числа кадров изображений. Процессорную «систему на кристалле» можно реализовать с помощью ПЛИС архитектуры FPGA и реализовать в ней прошивку параллельной программы для обработки потока изображений с целью определения основных характеристик потока частиц в плазменной струе (распределения скоростей и плотности частиц).

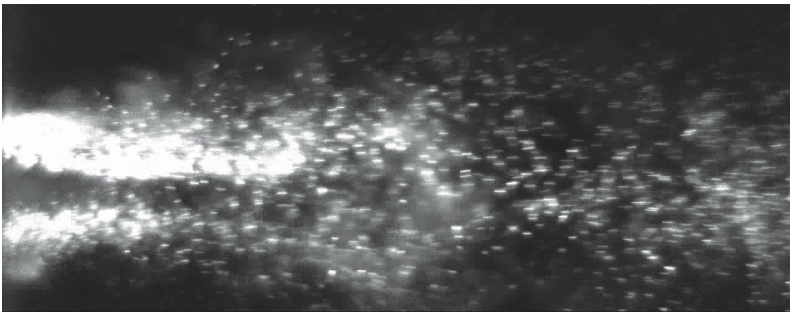


Рисунок 1. Изображение потока частиц в запыленной плазменной струе, полученное высокоскоростной цифровой камерой на основе ПЗС-матрицы.

Увеличивая время экспозиции кадра, на изображении можно получить более длинные треки (рис. 2), для которых достаточно просто оценивать скорость частиц в различных поперечных сечениях потока.

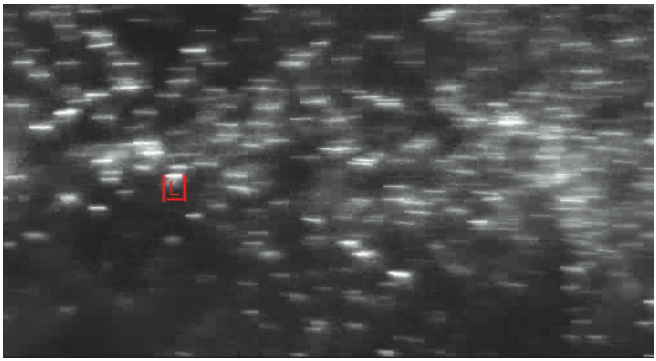


Рисунок 2. Изображение треков частиц (время экспозиции τ=10 мкс).

Скорость частицы определяется как отношение длины изображения трека L (рис. 2) к времени экспозиции кадра τ и умноженное на масштабный коэффициент, преобразующий расстояние на экране изображения в реальный пройденный частицей путь:

$$v = \frac{\mu \cdot L}{\tau} \quad . \qquad (1)$$

Структурная схема физического эксперимента с использованием высокоскоростной съемки приведена на рис. 3.

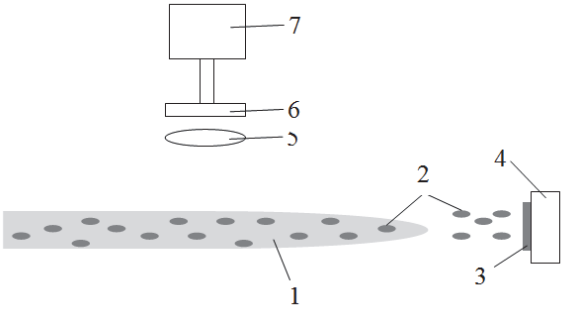


Рисунок 3. Структурная схема физического эксперимента: 1– плазменный поток, 2 – частицы порошка, 3 – покрытие, 4 – основа, 5– оптическая система, 6 – матричный фотоприемник цифровой камеры, 7 – блок управления видеосенсором и обработки потока изображений.

Сфокусированное оптической системой (5) изображение потока частиц (2) плазменной струи регистрируется цифровой камерой в виде изображения треков частиц. В блоке управления и обработки изображения (7) производится фильтрация изображения на шумоподавление изображения, выделение на полученном изображении треков частиц и измерение их параметров.

Описание алгоритмов обработки изображения

Обработка цифрового изображения состоит из следующих этапов:

1. нормировка отсчетов сигналов согласно калибровочной таблице;
2. фильтрация на шумоподавление;
3. выделение на изображении треков частиц и измерение их параметров.

В случае неоднородности чувствительности и других «геометрических» искажений сигналов ячеек матричного фотоприемника производится калибровка отсчетов яркостного сигнала каждой ячейки:

$$z = x * g(x), \tag{2}$$

где: x – отсчет яркостного сигнала ячейки матричного фотоприемника; z – нормированный отсчет; $g(x)$ – значение калибровочного коэффициента из таблицы «калибровки» отсчетов изображения, полученной на этапе калибровки видеорегистратора по эталонным источникам.

Для увеличения резкости (контрастности) изображения и уменьшения уровня «остаточных» шумов производится «высокочастотная» фильтрация изображения с помощью нерекурсивного фильтра с конечной импульсной характеристикой (КИХ-фильтр). Входные и выходные сигналы КИХ-фильтра связаны через операцию свертки [1]:

$$y(n) = \sum_{k=0}^{N-1} h(k) z(n-k), \tag{3}$$

где: $z(n-k)$ – отсчет входного нормированного сигнала, сдвинутого на k отсчетов относительно текущего n-го отсчета; $h(k)$ – коэффициенты импульсной характеристики фильтра, захватывающего в «окно сканирования» отсчеты в количестве N штук.

Выделение треков частиц осуществляется путем сегментации изображения на основе среднего уровня яркости [2]. Для пикселей, входящих в состав трека частицы, задается среднее значение яркости m. На изображении производится поиск однородных («сплошной связности») областей по порогу T, отвечающих условию:

$$\max_{P \in R} \left| y(P) - m \right| < T, \tag{4}$$

где: R – однородная область; $y(P)$ – яркость пикселя изображения, координаты которого связаны с величиной P. Среди полученных однородных областей отбра-

сываются те области, которые не являются треками частиц. Для оставшихся областей вычисляются длины треков, а затем и скорости согласно формуле (1).

Схема распараллеливания анализа и обработки потока кадров

Обработка одного кадра, регистрируемого цифровой камерой высокого разрешения, требует большого объема вычислений с учетом большого формата матричного фотоприемника (более миллиона пикселей). Кроме того, учитывая скоротечность регистрируемого процесса и его явно выраженную динамику изменчивости структуры потока, необходимо производить скоростную съемку с большой частотой кадров, что порождает очень большой поток видеокадров, который желательно было бы анализировать и обрабатывать в режиме реального времени. Решение задачи возможно только с помощью параллельной обработки потока изображений. При этом необходимо разделять каждый кадр на сегменты, которые можно обрабатывать параллельно.

Так как частицы движутся практически в горизонтальном направлении, отклонения треков частиц от горизонтали мало, поэтому можно ввести пороговое значение Y для разности координат между началом и концом трека, при превышении которого данный трек обрабатываться не будет. С учетом данного допущения, кадр можно разделить на перекрывающиеся сегменты с величиной перекрытия Y.

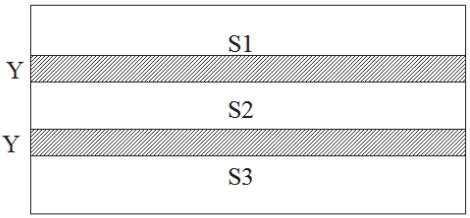


Рисунок 4. Разделение кадра на сегменты с перекрывающимися областями.

Функциональная блок-схема блока управления и обработки изображения приведена на рис. 5, двойными стрелками отмечены потоки данных, одинарными – потоки управляющих команд.

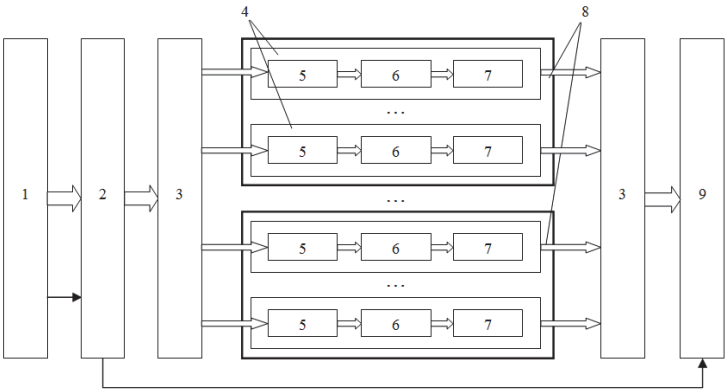


Рисунок 5. Функциональная блок-схема блока управления и обработки изображения: 1 – контроллер управления фотоприемником, 2 – менеджер кадров, 3 – разделяемая память, 4 – конвейер обработки сегмента кадра, 5 – IP-модуль нормировки отсчетов сигналов, 6 – IP-модуль ФВЧ, 7 – IP-модуль выделения и обработки треков на изображении, 8 – блок обработки кадра, 9 – контроллер ввода-вывода.

Каждая операция по обработке сегмента реализуется в виде отдельного IP-модуля (5, 6, 7), соединенных последовательно и образующих конвейер (4). Каждый конвейер обрабатывает свой сегмент кадра. Конвейеры, обрабатывающие определенные сегменты кадра, объединяются в блоки (8), каждый из которых обрабатывает отдельно взятый кадр, что позволяет выполнять обработку нескольких кадров одновременно под управлением менеджера кадров (2).

Литература

1. Айфичер Э., Джервис Б., Барри У. Цифровая обработка сигналов: Практический подход, 2е издание.: Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 992 с.
2. Павлидис Т. Алгоритмы машинной графики и обработки изображений. – М.: Радио и связь, 1986 – 394 с.

РАЗРАБОТКА БИБЛИОТЕКИ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ ДЛЯ СРЕДЫ DELPHI НА ОСНОВЕ ТЕХНОЛОГИИ ОБМЕНА СООБЩЕНИЯМИ

В.А. Бледнов, В.И. Иордан

ФГБОУ ВПО «Алтайский государственный университет»

В настоящее время трудоемкие задачи решаются с помощью сложных суперкомпьютеров, которые представляют собой объединение большого числа отдельных компьютеров и процессоров. Такой принцип организации вычислительных систем позволяет экономить средства на разработке современных сложных ЭВМ, т.к. создание суперкомпьютера в виде отдельного кристалла требует огромных затрат.

Суперкомпьютеры работают над массивной задачей по принципу «улья» отдельных машин, работающих параллельно и каждая из которых решает определенный блок (подзадачу) задачи. В некоторые моменты времени две машины организовывают взаимный диалог, посылая друг другу сообщения в виде команд или промежуточных результатов расчетов.

На сегодняшний момент существует несколько бесплатных и доступных широкому классу пользователей программных библиотек для организации работы суперкомпьютеров, доминирующими из которых являются реализации стандарта обмена сообщениями MPI [1]. На наш взгляд существенной недоработкой является направленность библиотек MPI только на работу с языками C/C++ и Фортран.

Язык Фортран, разработанный в 50-х годах прошлого века, завоевал большую популярность в среде научных работников. Вычислительные достоинства данного языка неоспоримы и общепризнанны. Однако в настоящее время сложно говорить о лидирующих позициях языка во многом из-за определенных сложностей при написании достаточно крупных программных продуктов. Недостаток в развитии средств создания визуального оформления, удобной работы с аппарат-

ными ресурсами и взаимодействия с ОС заставили разработчиков искать альтернативы среди других языков. Необычайную популярность завоевал C/C++, обеспечивающий возможность разрабатывать большие объемы программного кода, близкого к машинному. Кроме того, для языка имеется масса библиотек с широким спектром назначения. По сути, параллельная реализация C/C++ появилась как определенное расширение языка и заняла свою нишу. Однако, несмотря на широкое применение, язык C/C++ изначально не был адаптирован для решения крупных вычислительных задач, что породило массу проблем. Например, слишком «гибкая» и развитая система конвертации чисел разных типов требует со стороны программиста бдительности по контролю корректности преобразования типов и, как следствие, зачастую приходится перестраховываться, специально указывая тип результата арифметического выражения. В качестве другого неудобства можно указать на необходимость отслеживать моменты переполнения разрядности, так как, складывая два больших целых положительных числа, можно получить отрицательное значение. Кроме того, программируя на языке C/C++, необходимо учитывать при объявлении идентификаторов регистр переменных (прописная или строчная), что вносит путаницу при разработке и отладке громоздкого ПО. Некоторые языковые средства не являются встроенными средствами компилятора. Например, множественный тип в языке отсутствует, а «строковый» тип реализован только лишь с помощью библиотеки функций. Таким образом, именно для класса физико-математических задач с достаточно сложными функциями и формулами язык C/C++ требует очень высокой профессиональной подготовки на этапе отладки программы.

Наряду с популярной системой программирования C/C++ практически не менее популярной является среда визуального программирования Delphi (объектный язык Паскаль), которая также довольно часто используется для задач разработки ПО, в том числе и по взаимодействию с базами данных. Данный язык не уступает C/C++ по скорости вычислений, однако имеет ряд достоинств. В первую очередь удобства в написании программ и их отладке. Более «жесткая» система преобразования типов переменных позволяет быстро обнаруживать и устранять

различные ошибки. Кроме того, синтаксис данного языка ближе к «обычному» математическому языку, что позволяет привлекать его к разработке программ для решения сложных вычислительных задач.

Например, язык Pascal-XSC как одно из расширений стандартного языка Pascal имеет существенное преимущество перед языком C/C++ для сложных задач вычислительной математики и численного программирования. А именно: модульная структура программы, межмодульный экспорт-импорт данных различных типов, возможность введения собственных и переопределения уже существующих знаков операций с последующим их использованием в обычных выражениях, разрешение совмещать имена процедур, а также знаки операций (так, что они становятся применимыми к объектам различных типов), распространение основных операций на все структурированные типы, добавление комплексных чисел и интервалов, векторов и матриц в набор средств их размещения и переразмещения, управление округлениями для всех числовых операций и операций ввода-вывода, максимальная точность таких операций (которая оценивается и улучшается в процессе вычислений за счет «интервальной арифметики»), поддержка высокоточного вычисления операций типа скалярных произведений и т.д. [2]. Однако большим недостатком языка Pascal и его расширений является тот факт, что среди множества его библиотек пока нет библиотеки функций (расширения языка), поддерживающей распараллеливание программы. Поэтому авторы предприняли попытку разработки и реализации библиотеки, которая предоставит возможность пользователям использовать Delphi для разработки параллельных программ.

Основу систем параллельного программирования составляют библиотеки функций, оформленные в удобном для вызова виде. Библиотеки MPI оформлены в виде статических библиотек (*.lib), выполненных на компиляторе MS VC2003. Использовать такие библиотеки в «сторонних» компиляторах, в том числе и в компиляторах Borland, невозможно. Кроме того, не существует универсальных конверторов, позволяющих преобразовывать скомпилированный код библиотек в удобный вид. Для работы под управлением ОС Windows наиболее удобно ис-

пользовать динамические библиотеки функций (*.dll), так как их можно использовать с любым компилятором с помощью определенных API-функций самой ОС [3]. Т.е. код, реализующий библиотеку функций для работы параллельной программы Delphi, не нужно будет перекомпилировать и включать в код самой программы, они могут подключаться в процессе работы по мере необходимости.

В составе системы параллельного программирования предполагается создание специальной сетевой службы. Данная служба должна заниматься непосредственно приемом и отправкой сообщений (данных), т.е. представлять собой сам механизм обмена сообщениями, а dll-библиотека будет выполнять функции интерфейса взаимодействия с ней. Основное назначение службы состоит в том, чтобы «отслеживать» определенный порт на предмет приема нового сообщения от удаленной машины. Также служба отвечает за передачу блока сообщения машине с определенным номером и должна запускать множественные копии программы в момент начала вычислений. Данную службу целесообразнее разработать на Delphi с целью обеспечения большей совместимости с системой программирования Delphi и интегрировать ее в список служб ОС.

Пользователь обращается к службе на одной из машин и указывает перечень машин, на которых необходимо запустить экземпляр программы. Служба посылает всем указанным в списке машинам команду на запуск копии приложения. Дальнейшие действия координируются самой вычислительной процедурой.

Условно функции библиотеки можно разделить на три категории.

К первой категории относится функция, производящая первичную инициализацию взаимодействия с рабочей службой на локальной машине, а также функция «завершения», имеющая обратное назначение.

Вторая категория включает функции, возвращающие служебную информацию о номере процесса, времени начала вычислений, общего числа рабочих процессов и т.д. Данная категория необходима для того, чтобы определить для каждого процесса ту часть работы программы, которая должна быть выполнена этим процессом.

Третья категория используется наиболее часто и позволяет переслать опре-

деленный блок данных заданного пользователем размера. Сама функция не анализирует содержимое, поэтому посылать можно данные любых типов. В свою очередь функции приема и отправки могут быть: буферными, блокирующими, неблокирующими.

Буферные функции используют специальный буфер для сохранения переданных/принятых данных, в котором информация находится в очереди до момента, подходящего для приема/отправки. В случае если другой процесс занят, выполнение программы не прекращается, данные считываются и записываются в буфер службы по мере необходимости путем вызова соответствующей функции.

Блокирующие функции останавливают процесс вычислений и ожидают завершения приема/отправки и тем самым гарантируют факт приема/отправки данных. Затем программа продолжит работу.

Неблокирующие функции похожи на буферные, с разницей в том, что данные в буфер не сохраняются, а передаются в программу. Если в этот момент принять их некому, прием/передача пропускаются и программа продолжает вычисления, т.е. когда данные не представляют особой ценности, и нет необходимости в синхронизации процессов.

Для обмена сообщениями между машинами используется стек протоколов TCP/IP. Для работы этого используется механизм сокетов, реализованный в ОС Windows. Функции работы с сетью стандартизированы и работают во всех версиях системы.

Если функция передачи сообщения блокирующая, то она ожидает ответа от службы, остальные функции продолжают работу, не дождавшись уведомлений о передачи сообщения. Функция приема получает сообщение от службы. При этом, если тип переданного в службу сообщения буферный, тогда данные считываются из буфера в порядке очереди, остальные функции вызывают обмен данными по сети через службу.

Для реализации блокировки процесса используются API-функция Sleep, останавливающая процесс на определенное время. Далее производится проверка состояний и принятие решений о повторной блокировке процесса на постоянный

интервал (например, в 10 мс).

В настоящее время продолжается активная разработка библиотеки базовых функций, позволяющих производить надстройку, например, реализовав механизм «обмена теневыми гранями» [1].

Литература

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2004. – 608 с.: ил.
2. Клатте Р., Кулиш У., Неага М., Рац Д., Ульрих Х. PASCAL-XSC. Язык численного программирования: Пер. с англ. – 2-е изд., испр. и доп.-М.: ДМК Пресс, 2000. – 352 с.
3. Джонсон М. Харт. Системное программирование в среде Windows, 3-е издание. – М.: Издательский дом «Вильямс», 2005. – 592 с.

ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИИ NVIDIA CUDA ДЛЯ РЕАЛИЗАЦИИ АЛГОРИТМОВ FDTD

И.П. Молостов

ФГБОУ ВПО «Алтайский государственный университет»

Введение

Моделирование электромагнитных (ЭМ) процессов является важным этапом в проектировании сложных СВЧ-устройств. Оно позволяет разработчику получить хорошее представление о разрабатываемом устройстве, принимать более обоснованные решения и, следовательно, приведет к повышению качества разрабатываемого устройства.

Как известно, электромагнитные процессы описываются системой уравнений Максвелла. При работе с короткими импульсами не удается использовать квазистационарное приближение и приходится применять для расчетов нестационарные уравнения Максвелла. За исключением нескольких частных случаев, они не имеют аналитического решения. Трудности в решении этих уравнений, главным образом, обусловлены тем, что для граничных условий, встречающихся в реальных задачах, не удается получить аналитическое решение.

Одним из методов численного решения этих уравнений является FDTD (Finite Difference Time Domain). Теоретически, этот метод позволяет получать решение для наиболее общего случая, но он очень требователен к аппаратным ресурсам вычислительного устройства.

В данной работе мы рассмотрим применение технологии NVIDIA CUDA к решению задачи о распространении ЭМ волны. CUDA (Compute Unified Device Architecture) - это программно-аппаратная архитектура, позволяющая применять графический процессор для неграфических расчетов. Высокоуровневый CUDA Runtime API представляет собой расширение к языку C/C++. Аппаратной особенностью архитектуры является то, что все операции производятся одновременно над группами из 32 потоков (warp). Несколько «варпов» (warp) составляют «блоки» (block). Размер «блока» ограничен аппаратными ресурсами

устройства. На обработку данные поступают «блоками». Несколько блоков составляют «сетку» (grid). Такая иерархия позволяет автоматически масштабировать задачу для выполнения ее на GPU с различным количеством вычислительных ядер.

Постановка задачи

Рассмотрим реализацию алгоритма FDTD на примере задачи о распространении продольно-поперечной электромагнитной (TM) волны внутри двумерной прямоугольной области с идеально проводящей границей. Для решения системы уравнений Максвелла используем конечно-разностную схему, предложенную K. Yee[1].

В этом случае уравнения для TM волны имеют вид:

$$E_z^{n+1}(i,j) + E_z^{n+1}(i,j) + Z\frac{\Delta\tau}{\Delta x}\left[H_y^{n+\frac{1}{2}}\left(i+\tfrac{1}{2},j\right) - H_y^{n+\frac{1}{2}}\left(i-\tfrac{1}{2},j\right)\right] - \quad (1)$$

$$-Z\frac{\Delta\tau}{\Delta x}\left[H_x^{n+\frac{1}{2}}\left(i,j+\frac{1}{2}\right) - H_x^{n+\frac{1}{2}}\left(i,j-\frac{1}{2}\right)\right],$$

$$H_x^{n+\frac{1}{2}}\left(i,j+\frac{1}{2}\right) = H_x^{n-\frac{1}{2}}\left(i,j+\frac{1}{2}\right) - \frac{1}{Z}\frac{\Delta\tau}{\Delta y}[E_z^{n}(i,j+1) - E_z^{n}(i,j)], \quad (2)$$

$$H_y^{n+\frac{1}{2}}\left(i+\frac{1}{2},j\right) = H_y^{n-\frac{1}{2}}\left(i+\frac{1}{2},j\right) + \frac{1}{Z}\frac{\Delta\tau}{\Delta x}[E_z^{n}(i+1,j) - E_z^{n}(i,j)], \quad (3)$$

где $\tau = ct = \sqrt{\frac{1}{\mu\epsilon}}t$, $Z = \sqrt{\frac{\mu}{\varepsilon}} = 376{,}7$.

Верхний индекс у компонент ЭМ поля указывает на время ($t = \Delta t n$), нижний - на ориентацию данной компоненты поля в пространстве. Индексы i, j указывают на координату, в которой поле имеет соответствующую напряженность ($x = \Delta x i$, y$= \Delta y j$).

Для простоты будем рассматривать квадратную область со стороной $a = 1026\Delta x$. Положим, $\Delta x = \Delta y = \Delta z$ и $\frac{\Delta\tau}{\Delta x} = \frac{1}{2}$. Пусть в начальный момент

времени при $78 < i < 178$ и $78 < j < 178$ компонента поля E_z имеет вид:

$$E_z{}^0(i,j) = C\cos\left(\pi\frac{i-128}{100}\right)\cos\left(\pi\frac{j-128}{100}\right), \quad (4)$$

где C – некоторая константа. Иначе, $E_z{}^0(i,j) = 0$.

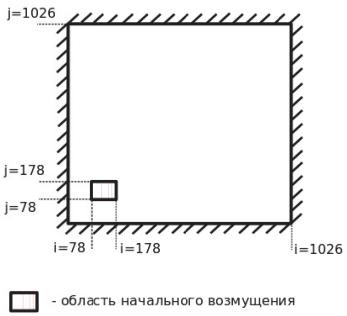


Рисунок 1. Геометрия задачи.

Особенности численного решения задачи

На основе формул (1), (2), (3) была написана расчетная программа на языке C с использованием CUDA Runtime API. Вычисления происходят параллельно для всех точек поля, так как в формулы входят только значения предыдущих итераций.

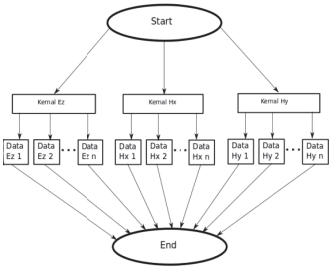
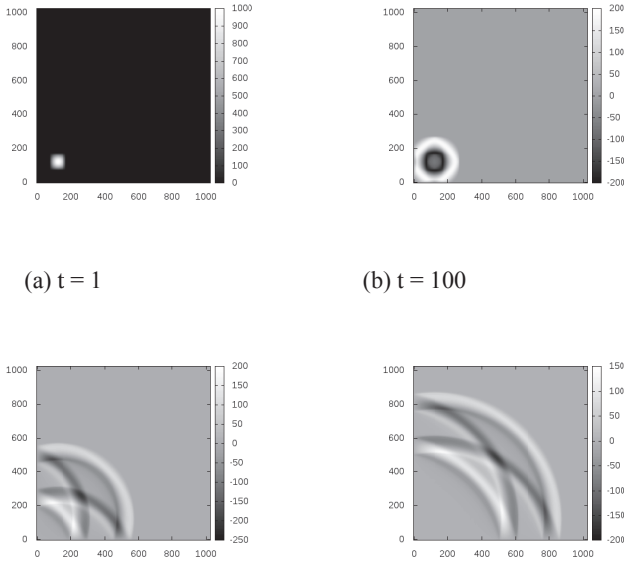


Рисунок 2. Блок-схема программы.

В настоящее время очень мало моделей GPU от NVIDIA поддерживают вычисления с двойной точностью, и они относительно дороги. Это обязывает программиста быть очень внимательным при разработке алгоритмов и написании программ, с целью избегания накопления ошибок округления.

Исследования показали, что при слишком малых приращениях компоненты поля E_z относительно соседних точек, возникали серьезные погрешности при расчете компонент магнитного поля, вызванные ошибкой округления. Это накладывает дополнительные ограничения на входные параметры задачи, помимо ограничений, обусловленных физикой самого процесса.

Численные результаты



(a) t = 1 (b) t = 100

(c) t = 400 (d) t = 700

Рисунок 3. Распределение напряженности поля E_z в плоскости XY.

Полученные результаты хорошо согласуются с результатами, полученными в работе [1]. Для расчетов использовались видеокарта NVIDIA GTS 450 и процессор Intel i5 2500K. Выигрыш в производительности, относительно расчета на одном ядре CPU, можно увидеть в таблице ниже.

Таблица 1

Сравнение времени выполнения программы на GPU и CPU

n	Время выполнения на GPU, с	Время выполнения на одном ядре CPU, с
100	1.460	4.010
400	3.800	15.500
700	6.190	27.280
1500	12.290	57.470

Заключение

Реализация FDTD алгоритмов с помощью CUDA API дает очень хорошие показатели производительности. При распараллеливании нашей программы на CPU мы не получили бы более чем четырехкратный выигрыш (CPU Intel i5 2500K имеет 4 ядра) относительно производительности на одном ядре. Это позволило бы процессору догнать GPU по производительности, но стоит учитывать еще и тот факт, что рыночная стоимость процессора, использованного нами при расчетах, почти в два раза превышает стоимость использованного графического ускорителя.

Литература

1. K. Yee Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media // IEEE Trans. Antennas Propagat. -- May 1966 -- vol. 14, No. 3, P. 302-307

ПРИМЕНЕНИЕ ТЕХНОЛОГИИ CUDA ДЛЯ ДЕКОДИРОВАНИЯ СКРЫТЫХ МАРКОВСКИХ МОДЕЛЕЙ

Д.А. Гефке, П.М. Зацепин

ФГБОУ ВПО «Алтайский государственный университет»

Введение

Математический аппарат Скрытых Марковских Моделей (СММ) представляет собой универсальный инструмент моделирования стохастических процессов, для описания которых не существует точных математических моделей, а их свойства меняются с течением времени в соответствии с некоторыми статистическими законами. Наиболее широкое применение СММ нашли при решении таких задач, как распознавание речи, анализа последовательностей ДНК и ряда других [1].

Современные системы распознавания речи предполагают наличие нескольких десятков, а то и сотен тысяч Скрытых Марковских моделей и их сочетаний, поэтому решение данной задачи связано со значительными вычислительными затратами, как на этапе обучения, так и при декодировании, что обусловлено необходимостью обработки огромного массива речевых данных. Применение современных технологий параллельного программирование, в частности, графических мультипроцессоров, позволяет получить значительный прирост производительности и перейти на качественно более высокий уровень в решении задач распознавания речи.

Целью данной работы является реализация алгоритма декодирования Скрытых Марковских Моделей (обобщенный алгоритм Витерби) с помощью графического процессора (CUDA) и оценка прироста производительности относительно центрального процессора (CPU).

Структура Скрытой Марковской Модели

В основе Скрытой Марковской Модели лежит конечный автомат, состоящий из **N** состояний. Переходы между состояниями в каждый дискретный мо-

мент времени **t** не являются детерминированными, а происходят в соответствии с некоторым вероятностным законом и описываются матрицей вероятностейпереходов A_{NN}. Схематическое изображение диаграммы переходов между состояниями в СММ приведено на рисунке 1.

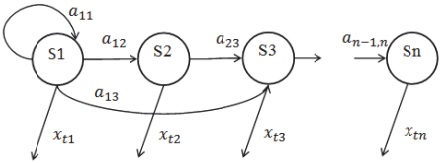


Рисунок 1. Структурная схема переходов в СММ.

При каждом переходе в новое состояние **i**в момент времени **t**происходит генерация выходного значения x_tв соответствии с функцией распределения f_i. Результатом работы СММ является последовательность векторов $\{x_1, x_2, \dots, x_T\}$ длиной Т.Достоинством СММ является возможность работать с последовательностями и сигналами разной длины, что затруднено при работе с искусственными нейронными сетями, в частности.

На практике, как правило, решается обратная задача: при известной структуре Марковской Модели требуется определить – какова вероятность, что наблюдаемая последовательность $\{x_1, x_2, \dots, x_T\}$ может быть сгенерирована данной СММ.

Таким образом, основными параметрами СММ являются:

1. N – количество состояний;
2. матрица вероятностей переходов A_{NN} между состояниями;
3. для каждого состояния i – функция плотности вероятности $f_i(x)$.

Функция плотности вероятности $f_i(x)$ описывается, как правило, взвешенной Гауссовой смесью:

$$f(x) = \sum_{i=1}^{M} w_i p_i(x),$$

где M-количество компонент смеси, w_i–вес компонента смеси, а $p_i(x)$–нормальное распределение вероятностей для D-мерного случая:

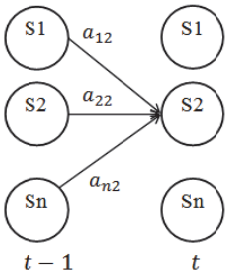
$$p_i(x) = \frac{1}{(2\pi)^{D/2}|\sigma_i|^{1/2}} exp[-1/2(x-\mu_i)^T \sigma_i^{-1}(x-\mu_i)],$$

где μ_i – вектор математического ожидания, σ_i–матрица ковариации.

Работа со Скрытыми Марковскими Моделями, как и с любой другой экспертной системой, осуществляется в 2 этапа: обучение – алгоритм Баума-Велча (Baum-Welchre-estimation), декодирование - алгоритм максимума правдоподобия (Витерби). Более подробно с этими алгоритмами можно ознакомиться в соответствующей литературе [2].

Алгоритм максимума правдоподобия (Витерби)

Суть алгоритма декодирования заключается в поиске последовательности состояний, при прохождении которой наблюдаемая входная последовательность имела бы максимальную вероятность. Схема переходов и выбора цепочки состояний в каждый момент времени **t**изображена на рисунке 2.



$$p_{21}(t) = p_1(t-1) + a_{12}$$

$$p_{22}(t) = p_2(t-1) + a_{22}$$

$$p_{2n}(t) = p_n(t-1) + a_{n2}$$

$$p_2(t) = \max[p_{21}(t), p_{22}(t), \dots, p_{2n}(t)]$$

Рисунок 2. Алгоритм декодирования Витерби.

В момент времени **t** осуществляется переход в состояние **i** из *всех* предыдущих состояний, после чего выбирается последовательность, имеющая максимальную суммарную вероятность в моменты времени **(t-1)** и **t**. Время выполнения

алгоритма пропорционально длине последовательности **L**и квадрату количества состояний **N**. Алгоритм имеет сложность $O(N^2L)$.

Особенности параллельного программирования на CUDA

Применение современных графических процессоров позволяет получить многократных прирост производительности при решении ряда научных и технических задач. Однако для достижения наилучших результатов необходимо учитывать ряд особенностей:

1) графический процессор (GPU) состоит из нескольких мультипроцессоров, которые, в свою очередь, состоят из ядер. Каждое ядро одновременно выполняет 32 потока (warp). Например, NVidia GeForceGTX 480 состоит из 15x32 = 480 ядер и параллельно может выполнять до 15360 легковесных потоков.
2) максимальной производительности удается достичь при выполнении *однотипных* действий над большим числом *обрабатываемых единиц* данных.
3) потоки объединяются в блоки и сетки блоков. Каждый поток имеет идентифицирующие его координаты.
4) архитектура памяти имеет сложную организацию: глобальная память (объемная, но медленная), локальная память, разделяемая память (быстрая), память констант и т.д.
5) особенности доступа к памяти: для получения максимальной пропускной способности все запросы к памяти должны быть выровнены.

Подробное описание и особенности применения графических процессоров можно найти в соответствующих руководствах [3].

Реализация декодера CMM на CUDA

Пусть $N = 32$ – количество состояний модели;

$L - 32$ – длина последовательности;

$C = 32$ – число одновременно декодируемых последовательностей одной CMM;

A_{NN} – матрица вероятностей переходов между состояниями;

B_{NL} – матрица вероятности наблюдения вектора последовательности X_L в состоянии N (Сматриц рассчитываются заранее).

Для декодирования одной последовательности используются Nпараллельных потоков (1 warp), каждый из которых декодирует свое состояние 1…N. Данная схема изображена на рисунке 3.Итоговая вероятность выбирается, как максимум вероятностей, полученных каждым потоком. Сложность алгоритма для одного потока - $O(N * L)$.

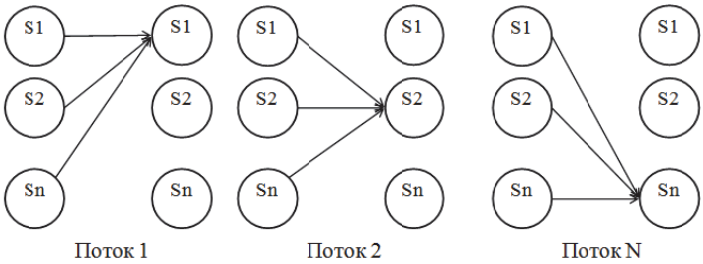


Рисунок 3. Параллельная схема декодирования.

Один вычислительный блок CUDA одновременно декодирует Споследовательностей (для одной СММ). Таким образом, задействовано N*C = 1024 потока, что соответствует максимальному количеству потоков на один мультипроцессор для CUDAверсии 2.0. Каждый мультипроцессор используется для декодирования отдельной СММ. В итоге, на GPUпараллельно происходит декодирование 32 * 15 = 480 последовательностей для видеопроцессора NVidia GeForceGTX 480.

Разделяемая (быстрая) память используется для хранения матрицы вероятностей Аи промежуточных вероятностей для Nсостояний. Объем необходимой памяти - $O(N^2 + N * C)$.

Тестирование

Для тестирования алгоритма использовалась машина следующей конфигурации:

1) Центральный процессор – Intel Core i7 930.
2) Операционная система – Windows 7 64-Bit.
3) Оперативная память - Kingston 12 Gb RAM DDR3.
4) Видеокарта – Nvidia GeForce GTX 480 (1536 Mb RAM, 480 ядер CUDA).
5) Версия CUDA – 3.2.
6) Компилятор Microsoft Visual Studio 2008.

Результаты приведены в следующей таблице:

Таблица 1

Процессор	Количество СММ	Количество декодируемых последовательностей	Число итераций	Время
CPU	1	32	1000	1260 мс.
GPU	16	32	1000	451,4 мс.

Общий прирост производительности составил: 44,7 раз.

Заключение

Применение современных графических процессоров позволило получить значительный прирост производительности при декодировании Скрытых Марковских Моделей – для 32 состояний был получен прирост 44,7 раз. В дальнейшем планируется реализовать алгоритм обучения (Baum-Welchre-estimation) с помощью GPU и, тем самым, полностью перенести аппарат СММ на программно-аппаратную платформу GPU.

Повышение итоговой производительности на несколько порядков позволит увеличить количество состояний в модели в несколько раз, и, следовательно, увеличить объем материала, используемого при обучении, что представляется перспективным с точки зрения реализации более качественных систем распознавания речи.

Литература

1. Vaseghi, Saeed V. Advanced digital signal processing and noise reduction. – 3rd ed. – Chichester, England:John Wiley & Sons,2006. – 453 p.

2. Hidden Markov Model Toolkit Book. - Cambridge University Engineering Department, 2001-2009.
3. NVIDIA CUDA C Programming Guide and CUDA SDK 3.2

ПРИМЕНЕНИЕ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ ДЛЯ ОБУЧЕНИЯ НЕЙРОННЫХ СЕТЕЙ

А.В. Демченко

ФГБОУ ВПО «Алтайский государственный университет»

Искусственные нейронные сети прочно вошли в нашу жизнь и в настоящее время широко используются при решении самых разных задач и активно применяются там, где обычные алгоритмические решения оказываются неэффективными или вовсе невозможными. В числе задач, решение которых доверяют искусственным нейронным сетям, можно назвать следующие:

- распознавание текстов;
- игра на бирже;
- контекстная реклама в Интернете;
- фильтрация спама;
- проверка проведения подозрительных операций по банковским картам;
- системы безопасности и видеонаблюдения.

Искусственные нейронные сети, подобно биологическим, являются вычислительной системой с огромным числом параллельно функционирующих простых процессоров с множеством связей. Несмотря на то, что при построении таких сетей обычно делается ряд допущений и значительных упрощений, отличающих их от биологических аналогов, искусственные нейронные сети демонстрируют удивительное число свойств, присущих мозгу, — это обучение на основе опыта, обобщение, извлечение существенных данных из избыточной информации.

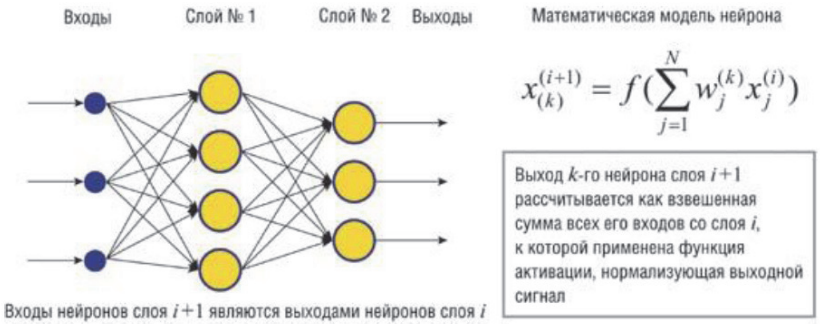


Рисунок 1. Структура слоистой нейронной сети.

Существует большое число алгоритмов обучения, ориентированных на решение разных задач. Среди них выделяет алгоритм обратного распространения ошибки, который является одним из наиболее успешных современных алгоритмов. Его основная идея заключается в том, что изменение весов синапсов происходит с учетом локального градиента функции ошибки. Разница между реальными и правильными ответами нейронной сети, определяемыми на выходном слое, распространяется в обратном направлении (рис. 2) — навстречу потоку сигналов. В итоге каждый нейрон способен определить вклад каждого своего веса в суммарную ошибку сети. Простейшее правило обучения соответствует методу наискорейшего спуска, то есть изменения синоптических весов пропорционально их вкладу в общую ошибку.

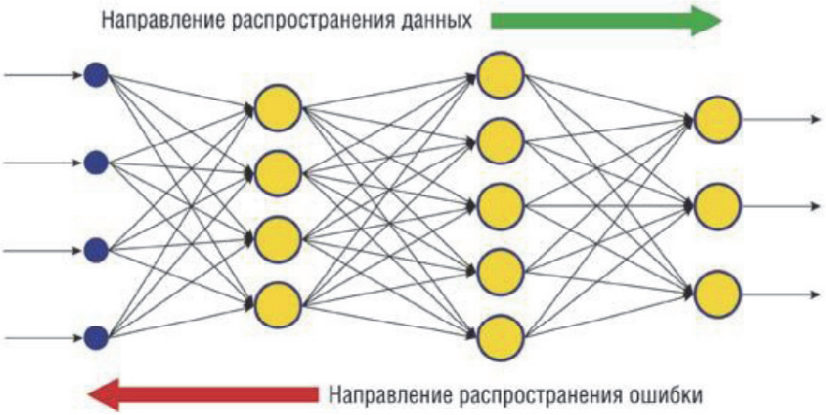


Рисунок 2. Сети с обратным распространением ошибки.

Многие вычислительные задачи требуют значительного количества операций, и, поэтому, даже на современной технике занимают немало вычислительных ресурсов. Более того, можно с уверенностью считать, что какой бы производительности ни достигла вычислительная техника, всегда найдутся задачи, на решение которых требуются значительные временные затраты. При этом для успешного решения некоторых задач требуется затратить строго ограниченное время, в противном случае, они теряют свою актуальность. К таким задачам относится прогнозирование погоды, обработка изображений, распознание образов при управлении техникой, а также моделирование каких-либо объектов.

Очевидным способом увеличения скорости вычислений было бы применение не одного вычислительного устройства, а нескольких, работающих совместно над решением одной задачи. Такой подход носит название параллельных вычислений. Несмотря на кажущуюся простоту решения, оно является весьма нетривиальной задачей. Первая проблема заключается в том, что для того, чтобы задачу можно было решить с помощью параллельных вычислений алгоритм ее решения должен допускать распараллеливание. Другой же, не менее важной проблемой является построение системы, на которой была бы возможна реализация параллельных вычислений. Решением последней проблемы стало появление кластерных систем.

Здесь важно отметить, что на уменьшение времени вычисления влияет не только увеличение количества вычислительных узлов на кластере и подготовка программного окружения, но и способ соединения узлов. Так, например, кластерные системы, в которых вычислительные узлы соединены с помощью высокоскоростного оптоволокна, дают больший временной выигрыш, чем в случае соединения витой парой.

Использование суперкомпьютера для подбора весовых коэффициентов значительно увеличивает скорость обучения и возникает вопрос во сколько раз кластер с n вычислительными узлами быстрее однопроцессорного компьютера (соответствующей мощности). Можно предположить, что кластер быстрее в n раз, но это не так, на самом деле время необходимое для обучения на многопроцессорной машине можно выразить формулой:

$$\tau = \frac{t}{n} + \Psi ;$$

где t - время обучения на однопроцессорной машине, n - количество вычислительных узлов, τ - время обучения на многопроцессорной машине. Таким образом, существует некая величина Ψ, которая представляет собой разницу между тем ускорением, которое можно было бы ожидать (в n раз) и тем, которое существует в реальности. Можно рассмотреть, из чего складывается эта величина:

$$\Psi = \eta + \psi + \upsilon + \gamma ;$$

где η - время межпроцессорной передачи данных, ψ- время, необходимое на подготовку данных, υ- время ожидания одних вычислительных узлов другими, γ - неучтенные временные затраты.

Наличие этих величин обусловлено тем, что параллельные алгоритмы отличаются от последовательных. Очень редко удается распределить нагрузку на вычислительные узлы равномерно, поэтому обычно на некоторых процессорах происходит ожидание окончания выполнения работы какого-либо другого процессора. Кроме того, большое влияние оказывает аппаратная платформа. Средства межпроцессорной коммуникации имеют различную скорость передачи и по-

этому выбор такого средства (оптоволокно, витая пара, wifi) также влияет на значение величины Ψ.

Для эксперимента была взята библиотека FANN, написанная на языке C. Была создана нейронная сеть, которая состояла из 2-х слоев на входе и 1 на выходе. Та же был один скрытый слой с количеством нейронов 2.

Нейронной сети была поставлена задача «предсказывания» курса доллара, соответственно в обучающей выборке участвовал год, месяц и среднее значение курса за этот период.

Но основная проблема заключалась в том, что FANN не поддерживает многопоточное обучение. Выходом из ситуации стало разбитие обучающей выборки на несколько интервалов, которые были сохранены в разные файлы.

Далее был создан цикл, который по очереди загружал данные из файлов в нашу ИНС, а уже этот цикл мы распараллелили с помощью библиотеки OpenMP.

Результатом данного эксперимента стало снижение времени обучения примерно в 3 раза.

Весь процесс моделировался на виртуальной машине, на 4-х ядерном процессоре AMD Phenom II X4. Использование данного метода позволило без особых трудностей менять количество процессоров в нашей системе.

На рисунке 3 приведен график зависимости времени выполнения от количества процессоров.

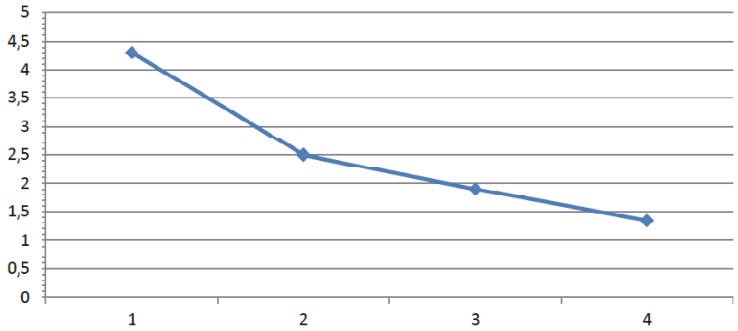


Рисунок 3. График зависимости времени обучения от количества процессоров.

В таблице 1 показано время обучения нейронной сети в зависимости от количества процессоров.

Таблица 1

Время обучения нейронной сети в зависимости от количества процессоров.

Количество процессоров	Время обучения, с
1	4.3
2	2.5
3	1.9
4	1.34

Заключение

После проведения экспериментов, мы можем с уверенностью заявить, что принципы параллельного вычисления очень эффективны, при решении современных задач обучения нейронных сетей. В ходе работы мы так же получили значение Ψ (таблица 2).

Таблица 2

Количество процессоров	Время обучения, с
2	0.35
3	0.47
4	0.26

ПРИМЕНЕНИЕ МЕТРИК КАЧЕСТВА ПРОГРАММНОГО КОДА ДЛЯ ВЫБОРА БЕЗОПАСНОГО МЕТОДА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

С.Л. Зефиров, А.Ю. Колобанов

ФГБОУ ВПО «Пензенский государственный университет», факультет приборостроения, информационных технологий и систем

Для целей создания безопасного программного кода созданы различные методы разработки. Однако в связи со значительным числом подобных методов, разницы в областях их применения, эффективности, гибкости немаловажным является вопрос выбора метода для реализации конкретного проекта. Критерием для подобного выбора могут являться метрики качества программного кода, получаемого с использованием метода разработки. Среди существующих метрик качества есть универсальные метрики, которые применимы практически ко всем видам программного обеспечения и могут использоваться практически в любом процессе разработки.

В международных стандартах даются следующие определения программного обеспечения - это степень, в которой оно обладает требуемой комбинацией свойств [1].

Для оценки качества программного кода возможно использование международных стандартов ISO, в частности описанию характеристик и метрик качества ПО посвящен международный стандарт ISO 9126 [2]. Четвертая часть стандарта (ISO 9126-4) предназначена для покупателей, поставщиков, разработчиков, сопровождающих, пользователей и менеджеров качества программных средств. В ней обосновываются и комментируются выделенные показатели сферы (контекста) использования программных средств и группы выбранных метрик для пользователей.

Методологии и стандартизации оценки характеристик качества готовых программных средств и их компонентов (программного продукта) на различных

этапах жизненного цикла посвящен международный стандарт ISO 14598 [3]. Рекомендуется следующая общая схема процессов оценки характеристик качества программ:

- селекция метрик качества, установление рейтингов и уровней приоритета метрик субхарактеристик и атрибутов, выделение критериев для проведения экспертиз и измерений;
- планирование и проектирование процессов оценки характеристик и атрибутов качества в жизненном цикле программного средства;
- выполнение измерений для оценки, сравнение результатов с критериями и требованиями, обобщение и оценка результатов.

ГОСТ 28195-89 «Оценка качества программных средств. Общие положения» [4] устанавливает общие положения по оценке качества программных средств. Достоинством методики, описанной в данном стандарте, является наиболее завершенная модель системы характеристик качества. К недостаткам следует отнести то, что большинство из приведенных оценочных элементов (порядка двухсот сорока из двухсот пятидесяти) рекомендуется оценивать экспертно, что в значительной степени снижает объективность полученной оценки. К тому же не указаны пути определения весовых коэффициентов.

Для обеспечения полноты измерения качества согласно стандарту IEEE 1061 требуется на ранних стадиях проекта на основе анализа целей проекта, области применения, ограничений и характеристик разработать проектно-ориентированные (design-oriented metrics) или структурные метрики (structural metrics) качества. Измерение качества в соответствии с данными метриками состоит в вычислении отклонения фактических характеристик продукта от норм и правил. Преимущества методологии качества IEEE основано на возможности обеспечения полноты управления качеством в проектах разработки программного обеспечения. Важной особенностью подхода IEEE является возможность управлять качеством на всех этапах жизненного цикла разработки программного обеспечения, поскольку разработанные правила и нормы для всех факторов каче-

ства являются не только метриками, но и инструкциями, выполнение которых может планироваться до исполнения и контролироваться в процессе работы.

Выбор методов безопасной разработки предлагается осуществлять с помощью оценки требуемого качества разрабатываемого программного кода. Для этого необходимо также, чтобы каждый метод разработки, из множества которых осуществляется выбор, был ранее уже применен группой разработчиков текущего проекта в одном или нескольких предыдущих процессах разработки. На основе данных сведений производится оценка получаемого качества программного продукта с использованием метода разработки. Следует также учесть, что если в процессе разработки были использованы несколько методов, то их следует рассматривать как один, комбинированный метод разработки.

Первым этапом процесса выбора является определение исполнителями отношений критериев с точки зрения оптимальности для рассматриваемого процесса разработки. Помимо непосредственных оценок качества программного кода, полученного в результате выполнения предыдущего процесса, эксперты должны учитывать стоимость и сроки разработки с применением рассматриваемых методов. Так, если при значительной разнице в сроках разработки, разница в качестве программного кода является несущественной, то оптимальным выбором является выбор набора методов, обеспечивающих меньший срок разработки. Однако, подобные отношения, являются сугубо индивидуальными для каждого процесса разработки и должны подбираться в каждом конкретном случае, ориентируясь на техническое задание и цели разработки.

На основе данных о предыдущих проектах данной группе разработчиков (технического задания, проектной документации, исходных кодов) производится оценка качества ранее разработанных проектов. Соответствующая оценка принимается равной оценке применяемого при этом метода разработки. При использовании метода в двух и более проектах оценка качества вычисляется на основе оценок каждого из этих проектов, причем большим весом обладают проекты, законченные недавно, а также проекты наиболее соответствующие и коррелирующие с текущей разработкой.

Выбор метода разработки осуществляется на основе полученных оценок качества. Для этих целей предлагается использование метода экспертных оценок. Экспертами производится оценивание (на основе данных технического задания) необходимого уровня качества разрабатываемого программного кода. Далее для выбора в ходе экспертного оценивания отбираются только те методы разработки (одиночные или комбинированные) для которых показатели качества программного кода, разработанного с их применением не ниже, чем требуемое для текущего проекта. При оценивании эксперты должны основываться на оценке качества, уровне безопасности, сроках и стоимости разработки, при этом учитывая отношения данных критериев, полученные на предыдущем этапе. После завершения оценки и обработки полученных результатов выбирается используемый для данного процесса разработки метод безопасной разработки.

Литература

1. 1061-1998 IEEE Standard for Software Quality Metrics Methodology
2. ISO/IEC 9126:1991. Information technology – Software product evaluation – Quality characteristics and guidelines for their use.
3. ISO/IEC 14598:1999 Information technology -- Software product evaluation
4. ГОСТ 28195 – 89. Оценка качества программных средств. Общие положения

СРЕДСТВА ПРОГРАММИРОВАНИЯ НЕТРАДИЦИОННОЙ МНОГОЯДЕРНОЙ АРХИТЕКТУРЫ И ПЕРСПЕКТИВЫ ИХ РАЗВИТИЯ

Д.В. Хилько

Федеральное государственное бюджетное учреждение науки.
Институт проблем информатики РАН (ИПИ РАН)

Аннотация

В статье рассматриваются ключевые аспекты реализации многоядерной потоковой рекуррентной архитектуры в виде VHDL-модели. Приводится описание существующих средств программирования архитектуры, их достоинства и недостатки. Описываются перспективы развития средств программирования на примере имитационной модели.

Ключевые слова: многоядерность, рекуррентность, параллелизм, потоковая архитектура

Введение

В ИПИ РАН ведутся работы по созданию нетрадиционной многоядерной потоковой рекуррентной архитектуры (МПРА)в области сигнальной обработки. Для экспериментальной апробации предлагаемой архитектуры разрабатывается рекуррентный обработчик сигналов (РОС), VHDL-модель которого представлена в гибридном, двухуровневом варианте с ведущим фон-неймановским процессором на управляющем (верхнем) уровне (УУ) и рядом потоковых процессоров на нижнем уровне – рекуррентном операционном устройстве (РОУ) [1].

Характерным отличием архитектуры РОУ от классической и нетрадиционной – является наличие одного, общего потока тегируемых, *самодостаточных*(рекуррентно разворачивающихся) данных. В этом потоке, помимо собственно обрабатываемых данных, содержится и необходимая для их обработки управляющая и служебная информация (в существующих архитектурах кодируемая в форме инструкций) [2]. Теги содержат некоторую начальную сжатую информацию, обес-

печивающую выполнение требуемой процедуры. Каждый следующий шаг процедуры рекуррентно самоопределяется, в том числе с учетом результата предыдущего шага.

В разрабатываемой архитектуре РОУ в состав центрального процессорного устройства (ЦПУ) включено автономное устройство преобразования тегов (ПТ), которое обладает возможностью саморазвертки рекуррентного вычислительного процесса. В памяти есть только набор операндов и начальных значений их функциональных полей, которые динамически подвергаются рекуррентной саморазвертке устройствомПТ. Данное представление программы называется *капсула*.

В настоящем исполнении VHDL-модель РОС имеет четыре процессорных ядра (ПЯ), способных функционировать параллельно. Кроме того, специфика исполнения модели позволяет говорить о различных уровнях параллелизма в архитектуре [3].На рис. 1 приведена структура двухуровневой архитектуры РОС, подробное описание которой содержится в работе [1].

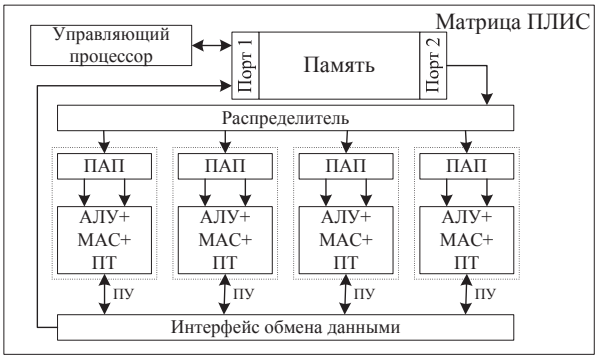


Рисунок 1. Структура РОС.

Существующие средства программирования РОС

Специфика архитектуры РОС находит свое отражение в способе его программирования. Интеграция потоков команд и данных ведет к аналогии с классами и объектами в объектно-ориентированном подходе. Следуя принципу инкапсуляции – программы в среде РОС представляют собой капсулы. Капсула содержит набор самоопределяющихся данных (операндов). В свою очередь, функциональные поля

операндов подвергаются рекуррентному и функциональному преобразованиям в ходе развертки. Данный подход позволяет добиться эффективной реализации рекуррентных вычислений в параллельных архитектурах, где эта проблема проявляется особенно остро.

Для программирования капсул был разработан и постоянно усовершенствуется специализированный язык, названный *капсульным языком программирования*. Данный язык используется в специализированной среде программирования СКАТ [4]. Предлагаемому языку в чистом виде (без вспомогательных утилит и подходов) свойственны те же недостатки, что и другим языкам ассемблерного типа:

1) Проследить ход вычислительного процесса практически невозможно в отсутствие специальных навыков или инструментальных средств.Алгоритм является параллельным и рекуррентно свернутым, что значительно усложняет его реализацию и отладку.

2) По структуре и размеру капсулы невозможно заранее оценить время ее исполнения (в условных вычислительных шагах). При создании архитектуры и капсульного языка преследовалась цель минимизировать размер потока самодостаточных данных, что и привело к возникновению указанной проблемы.

3) Низкая степень наглядности и читаемости программ в целом. Данный недостаток затрудняет отладку готовых капсул и их модификацию.

Несмотря на указанные недостатки, капсульный язык программирования обладает некоторыми достоинствами:

1) Капсула разрабатывается таким образом, что имеет минимальную длину и, следовательно, минимальный объем пересылок данных между УУ и РОУ. С одной стороны, это достигается благодаря рекуррентной свертке, а с другой – при помощи специальных конструкций языка.

2) Высокая степень защищенности данных от вмешательства извне (данное свойство обусловлено принципом инкапсуляции).

Поиск возможных путей устранения указанных недостатков привел к необходимости построения моделей программ. Развитие точных методов в програм-

мировании вызвало возникновение различных формальных моделей программ, в том числе и моделей параллельных программ [5]. Эти модели позволяют представить алгоритм в виде графовых структур, демонстрирующих последовательные и параллельные участки как вычислительного, так и управляющего процессов.

Развитием моделей параллельных программ для РОС стала концепция графодинамики [6]. Согласно этой концепции программа представляется в виде генерируемой последовательности графов вычислительного процесса. Содержание графов зависит от настройки преобразователя тегов (ПТ) на предметную область. В РОС задано жесткое ограничение на функциональность ПТ, благодаря чему возможна обратная трассировка от конечного графа (развернутого алгоритма) к свернутому представлению (капсуле). Такая концепция определяет последовательность разработки капсул: построение динамических графов с последующей сверткой конечного графа в капсулу.

На рисунке 2 приведены фрагменты капсулы и динамического графавычисления алгоритма быстрого преобразования Фурье (БПФ), являющегося одним из основных в области цифровой обработки сигналов. Более подробно о реализации БПФ в РОС можно узнать в работе [3].

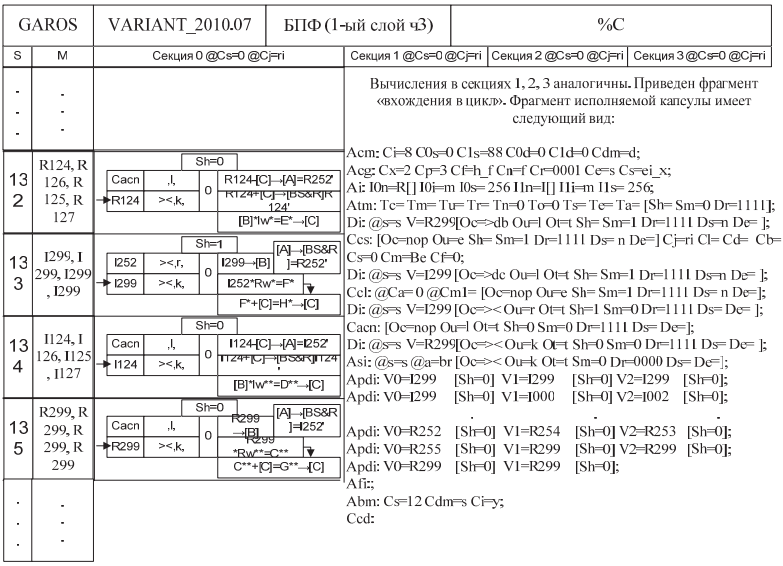


Рисунок 2. Фрагмент капсулы вычисления БПФ.

Перспективы развития средств программирования. Имитационная модель РОС

Графо–динамический подход построения капсул – это безусловно мощный инструмент, однако и его не достаточно для организации эффективного процесса разработки программ для РОС. Данное обстоятельство, а также отсутствие полнофункционального инструментария разработки ПО послужили причинами поиска других путей решения проблемы.

Предлагаемый вариант архитектуры РОС требует представления множества алгоритмов разрабатываемого программного обеспечения в виде двухуровневой структуры. К первому уровню можно отнести те алгоритмы, реализация которых нецелесообразна в среде РОУ (например, алгоритмы управления данными, алгоритмы управления вычислительным процессом и др.). Ко второму уровню относятся алгоритмы, реализующие вычислительные задачи высокой степени параллелизма и сложности.

Первый шаг разработки инструментария программирования - это создание

имитационной модели РОС, предназначенной для разработки и отладки капсул в условиях постоянно совершенствующейся спецификации архитектуры, а также для совершенствования VHDL-модели РОС.

Модель максимально соответствует спецификации РОС на логическом уровне. Данный подход позволяет использовать модель для исследования поведения функциональных блоков и локализации объектов архитектуры, для которых может потребоваться доработка. На текущем этапе разработки модель обладает следующими основными функциональными возможностями:

1) мультипроектность – возможность работы сразу с несколькими проектами отлаживаемых программных систем (например, с несколькими различными проектами распознавателя слов);

2) создание, редактирование и исполнение программ (капсул) на языке капсульного программирования;

3) подключение и использование специализированного компилятора для языка капсульного программирования (в настоящее время находится в разработке);

4) автоматическое и пошаговое моделирование процесса исполнения капсулы или последовательности капсул; эта функция позволит также осуществлять полную трассировку вычислительного процесса, а при необходимости и корректировать его с сохранением всех изменений в лог;

5) автоматизированное самотестирование и контроль результатов вычисления капсул.

Концептуальная структура модели, содержащая шесть основных компонентов, представлена на рис. 3.

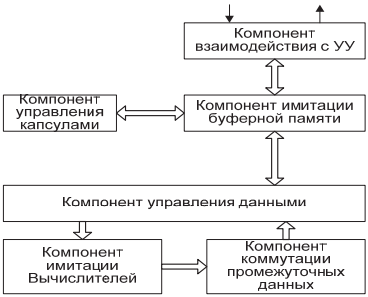


Рисунок 3. Структура имитационной модели РОС.

Компонент взаимодействия с УУ обеспечивает интерфейс, совместимый с программным обеспечением управляющего уровня. Введение этого компонента позволяет подключать имитационную модель к реальному управляющему процессору.Компонент имитации буферной памяти совместно с компонентом управления капсулами реализует блок «Память» (см. рис. 1): обеспечивает хранение внутреннего представления капсулы в памяти модели и организует доступ к нему. Компонент управления капсулами предоставляет сервисные функции для работы с ними (создание, редактирование), а также преобразует исходную капсулу во внутреннее представление модели.

Компонент управления данными - ключевое логическое звено модели - реализуется в виде блока «Распределитель» и четырех блоков «ПАП» (см. рис.1). Он обеспечивает рассылку операндов в четыре параллельных потока вычислений и образование пар операндов. Компонент имитации Вычислителей, содержащий четыре блока «МАС+АЛУ+ПТ» (см. рис. 1), организует дешифрацию и вычисления в каждом блоке «Вычислитель» в псевдо-параллельном режиме (за один такт синхронизации модели последовательно отрабатывают четыре вычислителя).Компонент коммутации промежуточных данных (блок «Интерфейс обмена данными» рис. 1) осуществляет сбор и перераспределение промежуточных данных между выходными буферами модели и компонентом управления данными.

В работе [7] была показана необходимость комбинирования сильных сто-

рон существующих подходов, с целью организации максимально эффективного процесса разработки ПО для РОС. Дальнейшее развитие средств программирования РОС основывается на данном подходе.

В перспективе планируется развитие представления динамических графов до уровня метаязыка программирования значительно более наглядного и понятного программисту-математику и создание соответствующих утилит компиляции программ в текущий формат графа с дальнейшей сверткой их в капсулы. Предлагаемый метаязык впоследствии будет развит до языка программирования высокого уровня, наследующего сильные стороны технологии MPIи языка потокового программирования Sisal.

Заключение

В результате анализа текущих инструментов программирования РОС стала очевидна необходимость их развития, с целью увеличения эффективности организации процесса разработки ПО. Один из первых шагов в данном направлении – создание имитационной модели РОС, позволяющей осуществлять всестороннюю отладку программ-капсул, а также совершенствовать экспериментальную архитектуру.

Дальнейшее развитие средств программирования РОС, создание методологии, а на их основе и технологии программирования позволит разработать полнофункциональную интегрированную среду разработки ПО. Данная среда станет мощным инструментом продвижения всего экспериментального проекта по созданию новой архитектуры.

Литература

1. Степченков Ю.А., Петрухин В.С. Особенности гибридного варианта реализации на ПЛИС рекуррентного обработчика сигналов // Системы и средства информатики: Доп. Вып. – М.: ИПИ РАН, 2008. – С. 118-129.
2. Степченков Ю.А., Петрухин В.С., Филин А.В. Рекуррентное операционное устройство для процессоров обработки сигналов // Системы и средства информатики. Вып. 11. М.: Наука, 2001. – С. 283-315.

3. Степченков Ю.А., Волчек В.Н., Петрухин В.С., Прокофьев А.А., Зеленов Р.А. Механизмы обеспечения поддержки алгоритмов цифровой обработки речевых сигналов в рекуррентном обработчике сигналов // Системы и средства информатики. Вып. 20, №1. – М.: ТОРУС ПРЕСС, 2010. – С. 31-47.
4. Зеленов Р.А., Степченков Ю.А., Волчек В.Н., Хилько Д.В., Шнейдер А.Ю., Прокофьев А.А. Система капсульного программирования и отладки // Системы и средства информатики. Вып. 20, №1. – М.: ТОРУС ПРЕСС, 2010. – С. 25-30.
5. Питерсон Дж. Теория сетей Петри и моделирование систем: Пер. с англ. – М.: Мир, 1984. – 264 с., ил.
6. Филин А.В. Динамический подход к выбору архитектуры вычислительных устройств обработки сигналов // Системы и средства информатики: Вып. 11 – М.:Наука, 2001. – С. 247-261.
7. Хилько Д.В., Степченков Ю.А. Вопросы программируемости многоядерной вычислительной архитектуры с единым потоком для эффективной реализации рекуррентных вычислений // Многоядерные процессоры и параллельное программирование; Системы обработки сигналов на базе ПЛИС и цифровых сигнальных процессоров: сб. ст. регион. науч.-практ. конф. / отв. ред. А.В. Калачев. – Барнаул : Изд-во Алт. Ун-та, 2011. – С. 86-92.

РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОВ В ПРИЛОЖЕНИЯХ ДЛЯ РАЗЛИЧНЫХ ВЫЧИСЛИТЕЛЬНЫХ УСТРОЙСТВ

Д.С. Кузьмин, В.А. Мали

ФГБОУ ВПО «Пензенский государственный университет», Факультет приборостроения, информационных технологий и систем

В процессе решения прикладных задач необходимо совершать переход от последовательного выполнения команд и программ к параллельному. Для многопроцессорных и многоядерных систем такая задача не представляет труда, так они и созданы для разбиения приложений на составные части и их параллельное выполнение. Это ускоряет выполнение программ, а также очень часто упрощает разработку приложения.

В системах с одним ядром ситуация следующая. Одновременно может выполняться лишь один процесс, поэтому необходимо точно распределить между процессами как процессорное время для каждого из них (приоритеты), так и ресурсы (программы и самой системы).

Часто практическую задачу бывает очень сложно или даже невозможно решить без использования параллелизма. Типичный пример – когда приложение должно регистрировать несколько асинхронных событий, возникновение которых не зависит ни от самой регистрирующей системы, ни от других процессов. Приложение, выполняемое на однопроцессорном устройстве, не может находиться сразу в нескольких состояниях, чтобы обрабатывать все одновременно возникающие события. Методам, позволяющим разрешить эту задачу, и посвящена данная статья.

В качестве исследуемых одноядерных систем выступают микроконтроллер MSP430f5438A фирмы Texas Instruments и обычный персональный компьютер. Микроконтроллер имеет одно 16-разрядное RISC ядро и работает на частотах вплоть до 60МГц при использовании внешнего кварцевого резонатора. В компь-

ютере стоит обычный процессор Intel Pentiun 2800 CISC-архитектуры с частотой работы 2,8ГГц.

Микроконтроллер служит для управления 2 периферийными устройствами: LED-дисплеем и клавиатурой. Контроллер связан через последовательный порт с компьютером, с которого ему приходят управляющие команды для дисплея. В то же время сообщения о событиях клавиатуры передаются от контроллера в компьютер. В итоге получается система, в которой нет однозначного разделения на «ведущий-ведомый» («master-slave»). Процессы принятия сообщения от компьютера контроллером и регистрация им изменений состояния клавиатуры – процессы асинхронные, поэтому без реализации параллельных процессов на микроконтроллере не обойтись. Для компьютера отправка команд контроллеру и принятие сообщений от него также асинхронны и требуют введения в приложение нескольких процессов. Архитектура системы показана на рисунке 1.

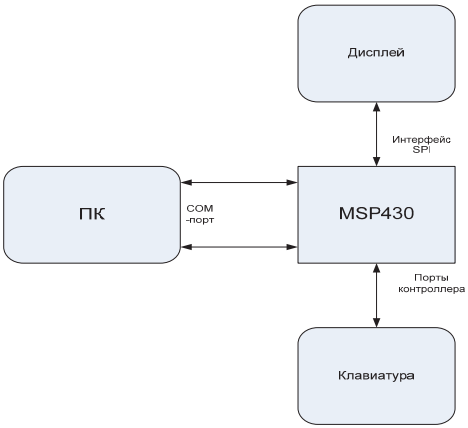


Рисунок 1. Схема исследуемой системы.

Реализация такого принципа для MSP430 затруднена следующими обстоятельствами. Первая – у микроконтроллера только одно ядро, поэтому параллельная обработка невозможна в принципе. Вторая – при написании приложений для контроллера нет возможности пользоваться приемами, как в «больших» операционных системах (механизм потоков и нитей в Windows и прочее). Третье – ско-

рость работы процессора контроллера специально ограничена и составляет всего 1,048 МГц.

Обойти эти проблемы и решить задачу позволяет, во-первых, само устройство микроконтроллера и его периферии, а также использование особой структуры приложения. У контроллера есть система аппаратных прерываний, которые следят за состоянием работы периферийных устройств и сообщают об их изменении приложению. Для обработки сообщений от компьютера служит модуль USCI, который представляет собой универсальный передатчик по интерфейсам UART и SPI. Для связи с COM-портом компьютера как раз и используется интерфейс UART. При получении байта от компьютера происходит прерывание от модуля USCI и байт фиксируется приложением. Для обработки сообщений от клавиатуры служат цифровые порты контроллера, прерывание от которых генерируется при изменении уровня сигнала на их входах. Теперь, когда есть средства регистрации событий, необходимо правильно ими распорядиться. Для этого используется принцип псевдопараллельного выполнения процессов.

Приложение представляет собой 3 псевдопараллельных процесса, работающих последовательно, но имеющие возможность обрабатывать асинхронные события за счет системы прерываний. Первый процесс – принимает сообщение от компьютера, анализирует его, отправляет команду на дисплей и формирует ответное сообщение компьютеру. Затем первый процесс передает управление второму, который осуществляет передачу сообщения. Обработку событий от клавиатуры осуществляет третий процесс, который фиксирует сигнал от клавиатуры и формирует сообщение к ПК. Затем третий процесс передает управление второму и тот передает сообщение. Передача управления между процессами осуществляется с помощью специального объекта типа «сообщение», в котором хранятся данные для передачи, их размер и флаг, символизирующий, какой из процессов должен выполняться следующим. В результате получилось, что встроенная система прерываний вместе с подобной архитектурой программы позволяет решать проблему с асинхронной обработкой событий.

Теперь рассмотрим решение задачи на ПК. Компьютер имеет быстрое вычислительное ядро процессора по сравнению с MSP430, и есть возможность использования всего арсенала программных средств для создания многопоточного приложения. Из минусов ПК – отсутствие системы прерываний (при программировании на высоком уровне), как у контроллера.

Приложение решено писать с использованием библиотеки классов Qt, так как она предоставляет доступ к классам работы с последовательным портом и к классам поддержки многопоточности.

Ядро у процессора одно, поэтому все процессы будут также выполняться псевдопараллельно, но скорость работы в 2,8ГГц позволяет этого не замечать. Источниками асинхронных сообщений будут являться пользователь компьютера, который используя графический интерфейс приложения, шлет команды контроллеру, и сам контроллер, который по последовательному порту сообщает ПК о событиях клавиатуры.

Структуру приложения решено сделать максимально приближенным к тому, что уже реализовано на контроллере. Но сделать это не удается из-за различия источников асинхронных сообщений. В приложении для ПК все сообщения асинхронные отправляются через последовательный порт, в то время как для приложения котроллера это последовательный порт и цифровые порты. В результате структура программы принимает вид, показанный на рисунке 2.

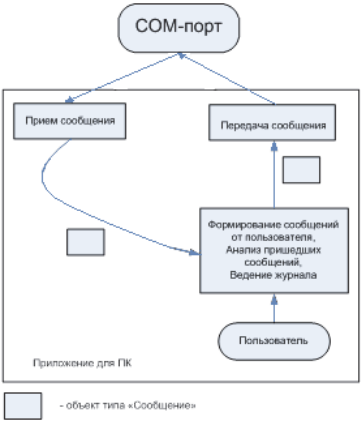


Рисунок 2. Структура приложения для ПК.

Один процесс программы разбивается на 3 потока. Основной поток – управляет пользовательским интерфейсом, анализирует пришедшие сообщения, выводит информацию о сообщениях и командах в журнал. Второй поток, как и в приложении для MSP430 передает готовые сообщения через последовательный порт. Третий поток, принимает сообщения от контроллера, в том числе и асинхронные основному потоку, в результате изменения состояния клавиатуры. Взаимодействие между процессами осуществляется при помощи такого же объекта, как и в приложении для микроконтроллера. Использование высокоуровневых средств создания процессов и потоков позволяет назначить каждому из потоков соответствующий приоритет. Ввиду того, что потоки приема/передачи данных используются гораздо реже, чем основной поток и операций они выполняют гораздо меньше, им назначен приоритет гораздо более низкий, чем потоку взаимодействия с интерфейсом.

Подводя итоги работы, можно сделать определенные выводы. Приложение, осуществляющее одновременную обработку асинхронных событий, можно реализовать и на однопроцессорном устройстве. Для решения задачи использовалась система аппаратных прерываний у микроконтроллера и высокоуровневые средства программирование на ПК вместе с высокой скоростью его работы. Но, глав-

ным здесь является, конечно же, еще и продуманная архитектура самого приложения. Причем, как показала практика – она может быть практически неизменной, даже при использовании различных аппаратных платформ.

Часть 2. ПЛИС

ОПЫТ ПРИМЕНЕНИЯ АППАРАТНОГО УЧЕБНОГО КОМПЛЕКСА НА БАЗЕ ПЛИС

К.Ф. Лысаков, М.Ю. Шадрин

Институт автоматики и электрометрии СО РАН,

Новосибирский государственный университет

Аннотация

Сегодня наблюдается тенденция все большего использования программируемой логики FPGA в самых разнообразных сферах применения. Особенно успешно решения на базе FPGA применяются в системах оперативной обработки данных, высокопроизводительной обработке больших потоков данных и реализации трудоемких математических алгоритмов. При этом особенности архитектуры и реализации на FPGA различных алгоритмов, диктуют очень высокие требования к квалификации разработчиков.

В Новосибирском государственном университете разработан и внедрен в учебный процесс курс, в рамках которого студенты знакомятся с архитектурой ПЛИС и особенностями реализации под нее различных задач. В основном выделяется два направления, которые последовательно изучаются в течение курса:

- особенности построения конвейеров для обработки данных в режиме поступления и распараллеливания исполнения для повышения производительности;
- разработка базовых контроллеров внешних интерфейсов: сигментных индикаторов, матричных клавиатур, звуковых ЦАП/АПЦ и так далее.

Для курса был разработан учебный комплекс SLED (ЗАО «СофтЛаб-НСК») на базе FPGA Spartan3E, который успешно применен в образовательном процессе. На нем студенты могут решать следующие практические задачи:

1. Реализация контроллера RS-232

2. Реализация видеоконтроллера VG
3. Реализация звукового контроллера PCM3001, ввод и вывод звука
4. Реализация ввода текста с использованием ограниченного количества кнопок
5. Реализация калькулятора с использованием матричной клавиатуры и индикаторов
6. Генерация и воспроизведение различной звуковой частоты.
7. Реализация контроллера динамической памяти SDRAM.

Описание аппаратного решения

Аппаратное решение SLED (Education Device от компании SoftLab-NSK) разработано для обучения программированию ПЛИС на примере кристалла Xilinx семейства Spartan3E XC3S500E.

Ниже приведена блок-схема внутреннего устройства SLED (рис. 1) и общий вид готового решения, предлагаемого студентам (рис. 2).

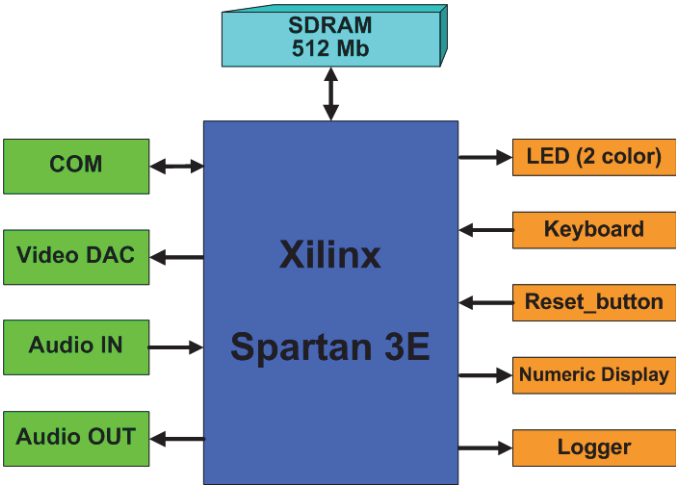


Рисунок 1. Блок-схема структурного решения SLED.

Рисунок 2. Общий вид аппаратного решения SLED.

Такое структурное решения позволяет осуществлять следующие действия с использование SLED:

- передачи данных в ПК и прием данных из ПК через COM-порт;
- возможность хранения до 512 Мб данных;
- прием звука в оцифрованном виде;
- воспроизведения звука в цифровом виде;
- вывод информации на монитор с использованием видео-ЦАП;
- индикация внутреннего состояния устройства;
- ввод данных со встроенной клавиатуры;
- возможность сброса в начальное состояние;
- вывод отладочной информации на логический анализатор (логгер).

Темы курса

В течение семестра, в рамках разработанного курса рассматриваются следующие темы:

1. Виды программируемой логики. Отличие архитектур PLD и FPGA: Устройство CLB. Механизм реконфигурирования.
2. RTL-модели. Реализация последовательных и параллельных сумматоров. Коды Хэмминга. Схемы ускоренного умножения.
3. Поведенческие модели на языке VHDL. Модули и компоненты.
4. Проект в понятиях ISE. Этапы синтеза, имплементации.
5. Примитивы: сумматоры, счетчики, сдвиговые регистры, умножители.
6. Принципы построения вычислительных конвейеров на базе программируемой логики. Реализация линейных и двумерных фильтров.
7. Интерфейс JTAG. Логические анализаторы уровней.
8. Последовательный порт COM. Контроллер через USB. Особенности реализации контроллеров на языке VHDL.
9. Звуковые контроллеры. Фильтры с откликом. Эффект реверберации.
10. Память статическая и динамическая. Базовая архитектура и принципы работы. Запоминающие элементы. Архитектура динамической памяти повышенной производительности DDR.
11. Вывод видеоинформации на монитор. Принципы работа стандарта VGA. Предельные пропускные способности.
12. Возможные приемы для повышения производительности вычислительной системы. Использование возможностей FPGA.
13. Работа с клавиатурой. Матричный принцип организации клавиатуры. Частота опроса нажатия клавиш. Борьба с «дребезгом».
14. Обзор существующих параллельных вычислительных систем. СБИС. Матричные процессоры для обработки сигналов и изображений. Принципы разработки СБИС-архитектур.

АППАРАТНО-ПРОГРАММНОЕ РЕШЕНИЕ ДЛЯ ОБРАБОТКИ ПОТОКОВ ВИДЕОДАННЫХ В ФОРМАТЕ HD-SDI В СОСТАВЕ ПК

К.Ф. Лысаков, М.Ю. Шадрин

Институт автоматики и электрометрии СО РАН,

Новосибирский государственный университет

Аннотация

Разработанный аппаратно-программный комплекс на базе FPGA Virtex5 позволяет в режиме реального времени обрабатывать одновременно до 4 потоков HD-SDI с передачей их в ПК через интерфейс PCI-E x4, и до 2 потоков из ПК. Суммарно, обрабатываемой поток составляет около 12 Гбит/с.

Такой поток достигается за счет использования оригинальной архитектуры программных модулей внутри FPGA, а также за счет использования в качестве контроллеров внутренней динамической памяти и контроллера шины PCI-E самого кристалла FPGA.

Решение имеет практическое применение и на сегодняшний день используется в линейке продуктов Forward (ЗАО «СофтЛаб-НСК») для обеспечения обработки шести потоков формата HD-SDI в виртуальной студии «Фокус» и системе замедленных повторов и многоканальной записи «Forward Goalkeeper».

Описание аппаратного решения

Спецвычислитель HDG (High Definition Grabber – устройство для сбора видеоданных высокого разрешения) для использования в области профессионального видео, требующей передачи больших потоков данных.

Особенностью использования профессионального видеооборудования является использование для передачи некомпрессированного видео и звука интерфейса SDI (Serial Digital Interface), известного также как SMPTE-259M (Society of Motion Picture and Television Engineers). Передача происходит однонаправлено по обычному коаксиальному кабелю. В качестве разъемов чаще всего используют BNC (см. рис. 1).

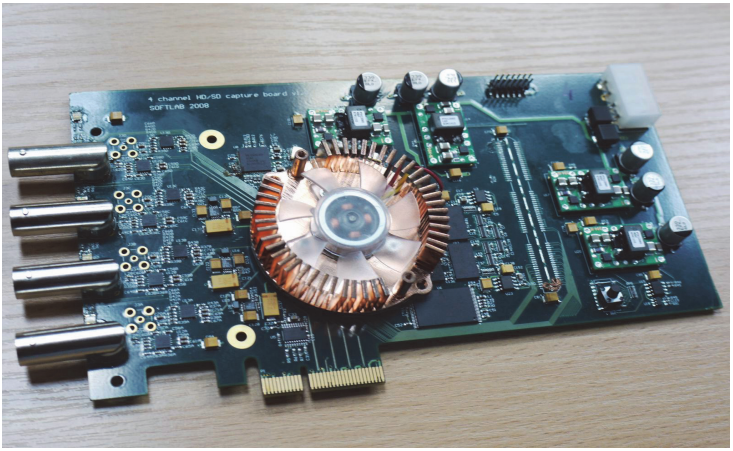


Рисунок 1. Общий вид аппаратного решения HDG.

Аппаратная часть HDG выполнена в виде платы для ее монтажа в корпус формата 2U. В ее состав входят:

- FPGA Xilinx семейства Virtex5 XC5VLX50T –3FF656;
- Два банка памяти стандарта DDRII объемом 128 МБ каждый;
- 4 PCM 1802 (24-битный АЦП в диапазоне 16 Гц - 96 КГц)
- 8 разъемов SDI / HD-SDI (4 входа и 4 выхода);
- Интерфейс PCI-E x4.

Сегодня наиболее распространены форматы передачи некомпрессированного видео SD-SDI (Standart Definition) и HD-SDI (Hight Definition). Поддерживаются форматы входных цифровых аудио-видео потоков: SDI / DVB-ASI (143, 176, 270 и 360МГц) и HD-SDI (1.4835 и 1.485 МГц).

Ниже в таблице приведены пропускные способности спецвычислителя при передаче данных между внутренней памятью и памятью ПК для различных конфигураций (табл. 1).

Таблица 1

Потоки данных HDG

Конфигурация ПК	Запись данных в ПК (МБайт/с)	Чтение данных из ПК (МБайт/с)
Мат. плата: Asus P5Q-E Чипсеты: P45 + ICH10R Процессор: Intel Core2Duo T6600 Память: DDR2-667	**675**	**445**
Мат. плата: Asus P5E64 WS Evolution Чипсеты: X48 + ICH9R Процессор: Intel Core2Duo E8500 Память: DDR3-1333	**600**	**476**

Установленная память стандарта DDRII в совокупности с контроллером собственной разработки позволяет получить симметричный поток данных (чтения и записи) не менее 2 ГБайт/с. Этого достаточно для организации буфера глубиной 2 кадра для четырех входные потоков совместно со звуком в максимальном разрешении (HD-SDI), либо на 11 кадров в разрешении SD (270 Мбит/с).

Структурное решение программных модулей FPGA

Зачастую, в проекте присутствуют три программных модуля: контроллер взаимодействия с ПК, контроллер взаимодействия с памятью и модуль обработки данных.

Такой набор является минимально необходимым, но в реальных задачах добавляется еще ряд модулей для обеспечения полной необходимой функциональности. Представленная программная модель позволяет эффективно решать различные задачи без изменения общей архитектуры системы, а также при сохранении всей архитектуры переходить на другое аппаратное решение, созданное согласно сформулированным принципам, что достигается за счет выделения следующих программных модулей (см. рис. 2).

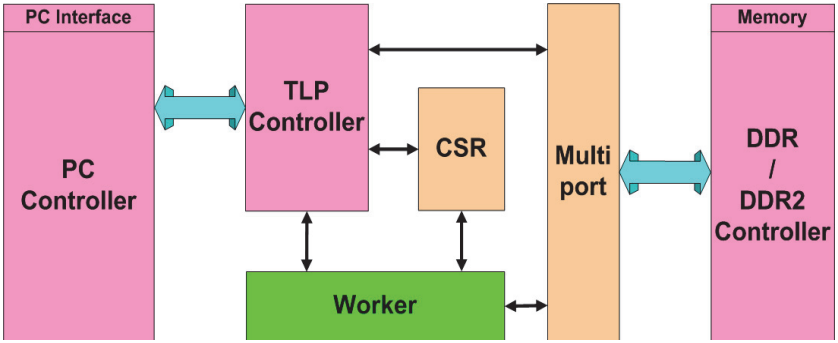


Рисунок 2. Структурное решение программных модулей HDG.

Worker – компонент реализующий обработку данных и имеющий унифицированный интерфейс для управления его работой. При этом можно сказать, что данный компонент представляет собой именно уровень реализации алгоритмов обработки, а все описанные далее компоненты обеспечивают уровень интерфейсного взаимодействия.

1) PC Controller – компонент для реализации физического уровня интерфейса общения с ПК. В случае PCI-X включает в себя буфера и управление базисами тактовых сигналов, а в случае PCI-E реализует физический уровень протокола.
2) TLP Controller – модуль обработки запросов шины ПК, также реализующий метод Scatter/Gather Lists, позволяющий непрерывно передавать блоки данных до 4 МБ между оперативной памятью ПК и устройством на аппаратном уровне.
3) CSR – Control Status Registers. Модуль выполняет функцию управления работой всего устройства, что позволяет даже в случае замены интерфейсной шины или памяти оставить без изменения драйвер устройства и пользовательское приложение.
4) Multiport – модуль, предназначенный для выполнения функции арбитра для нескольких клиентов, обращающихся к единому ресурсу – памяти в составе программно-аппаратного комплекса.

5) DDR/DDR2 Controller – компонент контроллера динамической памяти.

Описанные принципы построения спецвычислителей на базе FPGA позволяют использовать преимущества архитектуры FPGA и получать увеличенную производительность по сравнению со стандартными решениями. При таком разделении на функциональные программные модули сохраняется гибкость применения всего программно-аппаратного решения, что позволяет использовать его для реализации различных алгоритмов обработки данных.

ИСПОЛЬЗОВАНИЕ ПЛИС АРХИТЕКТУРЫ FPGA ДЛЯ ПРОЕКТИРОВАНИЯ «СИСТЕМЫ НА КРИСТАЛЛЕ» В СОСТАВЕ ВЫСОКОСКОРОСТНОГО ВИДЕОРЕГИСТРАТОРА ПОТОКА ИЗОБРАЖЕНИЙ

А.И. Постоев, А.А. Соловьев, В.И. Иордан, С.А. Скобкарев

ФГБОУ ВПО «Алтайский государственный университет»

Для исследования быстропротекающих физических процессов достаточно эффективны методы цифровой обработки потока видеоизображений, регистрируемых высокоскоростными цифровыми фотокамерами (порядка тысячи кадров/с и более). Основными элементами таких фотокамер являются современные матричные фотодиодные (ФД) приемники или приемники на основе приборов с зарядовой связью (ПЗС), представляющие собой большие и сверхбольшие интегральные схемы (БИС и СБИС) с интегрированным процессором [1]. Например, в области диагностики структуры и волновой динамики процессов плазменного и детонационно-газового напыления (ДГН) порошковых покрытий, а также процесса «самораспространяющегося высокотемпературного синтеза (СВС)» материалов из порошков металлов, до сих пор актуальна задача создания комплекса регистрирующей электронной аппаратуры, способной производить в режиме реального времени анализ и интеллектуальную обработку информации, заключенной в потоке изображений. Скоростная съемка быстродействующими цифровыми камерами на основе матричных приемников класса КМОП-ФД СБИС позволяет регистрировать характерную скоротечность процессов теплового взрыва и горения в процессах ДГН. В отношении реакции СВ-синтеза появляется возможность исследовать форму и размеры очагов горения, их фазовых состояний и температур (в т.ч. и сверхадиабатических температур).

Современные фотоприемные КМОП-ФД СБИС содержат более 10^6 фоточувствительных элементов с цифровым 8-10 битным выходом и скоростью

вывода более 500 кадров/с. В настоящее время разработаны многокамерные системы, позволяющие получать круговой обзор и синтезировать изображение интересующих объектов с высоким разрешением, и многоспектральные системы видимого и ИК-диапазонов, построенные на приемниках с узкополосными оптическими фильтрами [1].

В настоящее время наиболее перспективными являются «интеллектуальные» камеры на основе КМОП-ФД матриц, которые предоставляют возможность создания однокристальных и многокристальных цифровых камер с устройствами аналоговой, цифровой и нейроподобной обработки потока изображений, а также в ближайшей перспективе позволят реализовать системы технического и искусственного зрения, сопоставимого по характеристикам с биологическим зрением.

Важным преимуществом КМОП-матрицы является возможность объединения на одном кристалле аналоговой, цифровой и обрабатывающей частей (КМОП-технология, являясь в первую очередь процессорной технологией, подразумевает не только «захват» света, но и процесс преобразования, обработки, очистки сигналов не только собственно-захваченных, но и сторонних компонентов РЭА), что послужило основой для миниатюризации фотокамер и снижения их стоимости ввиду отказа от дополнительных процессорных микросхем.

С помощью механизма произвольного доступа можно выполнять «кадрированное» считывание (англ. windowing readout – считывание выбранных групп пикселов), которое позволяет уменьшить размер захваченного изображения и потенциально увеличить скорость считывания по сравнению с ПЗС-сенсорами, для которых необходимо выгружать всю информацию. Можно отсканировать часть кадра и отобразить ее на весь экран (имеется возможность качественной ручной фокусировки и вести скоростную съёмку с меньшим размером кадра). В дополнение к усилителю внутри пикселя, усилительные схемы могут быть размещены в любом месте по цепи прохождения сигнала (можно повышать чувствительность в условиях плохого освещения).

В настоящее время в мире представлено большое количество компаний, предлагающих такого рода матрицы: Cypress Semiconductor Corporation (LUPA-1300), Aptina Imaging (MT9M413), Photonfocus (A1312-160), CMOSIS (CMV2000), Dalsa (CR-GEN0-H640) и др. Для регистрации быстропротекающих процессов по основным характеристикам наиболее подходящим является высокоскоростной цифровой видеосенсор LUPA-1300 производства компании Cypress Semiconductor Corporation (см. рис. 1), выполненный на основе КМОП-технологии и обеспечивающий частоту сканирования до 450 кадров в секунду при разрешении 1280 x 1024 пикселей. Размер пикселя составляет 14x14 мкм.

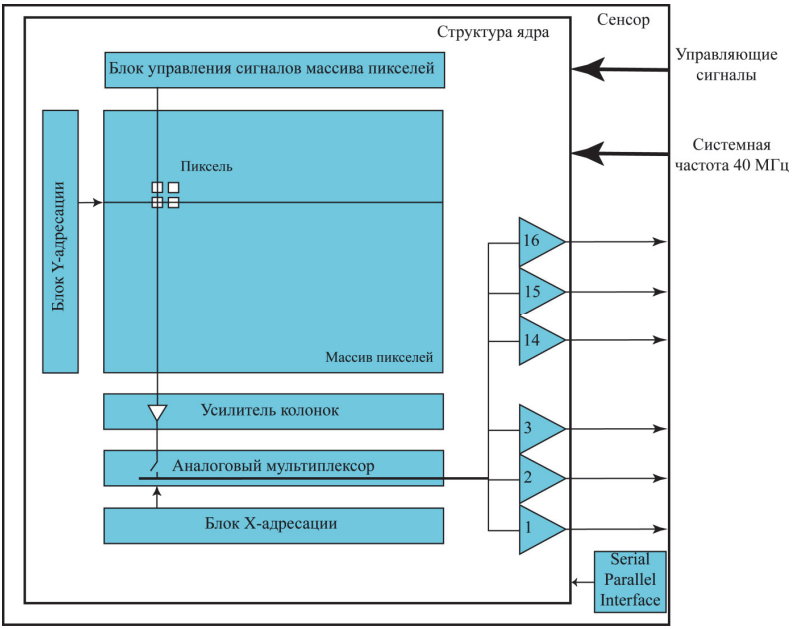


Рисунок 1. Структурная схема видеосенсора LUPA-1300.

Такая высокая частота кадров достигнута с помощью использования 16 параллельных выходных усилителей, каждый из которых работает на частоте 40 МГц. Скорость считывания может быть повышена за счет режима выборки с уменьшенным разрешением (сканирование через строку или через столбец), что

позволяет производить видеосъемку изображения с частотой 900 кадров в секунду. Значительно увеличить максимальную частоту кадров можно при помощи уменьшения размера области сканирования, то есть при уменьшении количества строк и их длины происходит пропорциональное увеличение частоты сканирования. Размер сенсора – 17,92 мм х 14,34 мм; диагональ – 23,3 мм (оптический формат 1,43 дюйма), коэффициент заполнения составляет 40%. Разрядность аналого-цифрового преобразователя равна 10 бит. При работе с сенсором LUPA-1300 используется 3-проводной последовательно-параллельный интерфейс (Serial Parallel Interface).

Управление видеосенсором производится с помощью управляющих сигналов. Выбор считываемого изображения, производится посредством блока управления сигналов массива пикселей (блоками X-адресации и Y-адресации). При считывании изображения сначала происходит выбор нужной строки блоком Y-адресации, затем сигнал поступает на усилитель колонок, а далее на аналоговый мультиплексор, управляемый блоком X-адресации, и затем сигнал поступает на один из 16 выходных усилителей.

В качестве внешней памяти в видеорегистраторе (рис. 2) предполагается использовать твердотельный накопитель (SSD, solid-state drive) без движущихся механических частей. SSD состоят из микросхем памяти и управляющего контроллера. Твердотельный накопитель имеет странично-блочную организацию (запись и чтение производится по странице целиком, стирание – поблочно). В новом поколении твердотельных накопителей используется интерфейс SATA третьего поколения, его применение весьма актуально с учетом его более высокой производительности. Скорости чтения и записи последних моделей дисков находятся, соответственно, на уровне 500 и 315 МБ/с. С целью сокращения количества микросхем в процессе проектирования и реализации видеорегистратора среди двух одних из наиболее перспективных архитектур ПЛИС, таких как CPLD и FPGA, была применена ПЛИС архитектуры FPGA компании Xilinx.

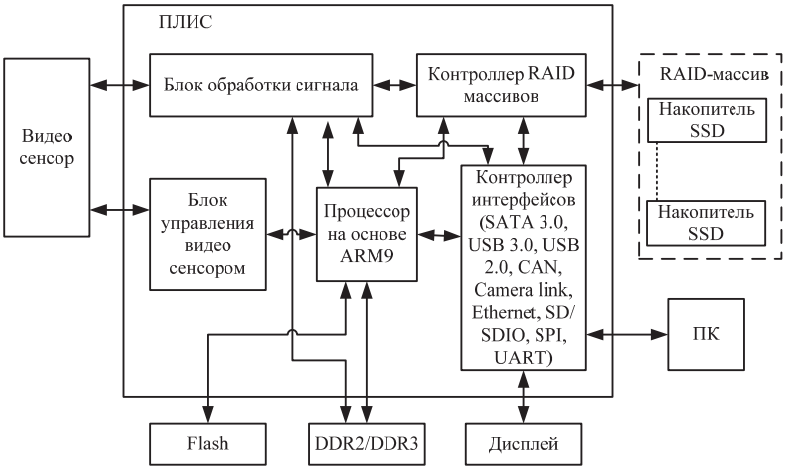


Рисунок 2. Структурная схема видеорегистратора.

Это позволило повысить надёжность системы и снизить сложность разработки печатной платы, обеспечить преимущества перед другими вариантами реализации устройства, сохраняя при этом гибкость системы и возможность её быстрой реконфигурации. Функции управления и сопряжения с внешними устройствами в видеорегистраторе могут быть обеспечены двумя способами. Первый способ заключается в самостоятельной разработке специализированного процессорного ядра, выполняющего необходимые функции в соответствии с заданным алгоритмом управления. Второй способ заключается в формировании на кристалле процессорного ядра с архитектурой одного из перспективных современных микроконтроллеров и необходимого контроллера интерфейсов. Более эффективным для видеорегистратора является второй вариант, который позволяет использовать стандартные средства разработки программного обеспечения для проектируемой «системы на кристалле». Было принято решение использовать в качестве управляющего процессор на основе ядра ARM9, реализующего архитектуру высокопроизводительного 32-разрядного процессора. Основными блоками видеорегистратора являются видеосенсор и ПЛИС (рис. 2). В свою очередь, к ПЛИС подключаются дополнительные элементы, такие как дисковый массив,

флеш-память, DDR-память и дисплей. На основе ПЛИС реализованы: процессор на ядре ARM9, контроллер интерфейсов, контроллер RAID-массивов, блок управления видеосенсором и блок обработки видеосигнала.

Данные, хранимые во флеш-памяти, либо загруженные в блок управления видеосенсором из персонального компьютера производят конфигурирование режима работы видеосенсора. Блок управления определяет размер области сканирования, частоту кадров, смещение для усилителя с программируемым коэффициентом передачи. При записи изображения происходит передача зарегистрированных данных от видеосенсора в блок обработки сигнала для комплексной обработки видеоизображения с последующей его передачей либо в контроллер RAID-массивов (для записи на внешние накопители), либо в процессор (для дальнейшей математической обработки), либо в контроллер интерфейсов (для передачи на внешнее устройство через специальный интерфейс).

Заключение

Архитектурные решения на основе современных ПЛИС и дальнейшее развитие технологии и схемотехники СБИС расширяют возможности по исследованию различных быстропротекающих процессов.

Литература

1) Стемпковский А.Л. СБИС – перспективная элементная база однокристальных систем приёма и обработки изображений / А.Л. Стемпковский, В.А. Шилин // Электроника: наука, технология, бизнес. – 2003. – № 2. – С. 14–20.

РЕАЛИЗАЦИЯ ПРОТОКОЛА I²C СРЕДСТВАМИ ПЛИС

Р.С. Викулов, В.А. Мали

ФГБОУ ВПО «Пензенский государственный университет», факультет приборостроения, информационных технологий и систем

В ПЛИС заложены возможности, которые позволяют превратить ее в ИС с любой функцией цифровой логики. Проектирование сводится к выявлению программируемых элементов (перемычек или запоминающих ячеек), после удаления которых в структуре схемы остаются только те связи, которые необходимы для выполнения требуемых функций. На практике эта задача весьма непростая, так как современные ПЛИС содержат в среднем несколько десятков тысяч перемычек. Поэтому для проектирования обязательно применяют системы автоматизированного проектирования (САПР ПЛИС).

Благодаря наличию различных систем автоматизированного проектирования, а также структурным и технологическим особенностям, ПЛИС представляют технологию рекордно-короткого цикла разработки радиоэлектронной аппаратуры. Причем весь цикл проектирования и изготовления готового устройства осуществляется самим разработчиком, что значительно снижает стоимость РЭА по сравнению с использованием БМК.

Наиболее эффективно использование ПЛИС в изделиях, требующих нестандартных схемотехнических решений. В этих случаях ПЛИС даже средней степени интеграции (24 вывода) заменяет, как правило, до 10-15 обычных интегральных микросхем.

Целью данной работы является создание контроллера шины I²C средствами ПЛИС. I²C — последовательная шина данных для связи интегральных схем, разработанная фирмой Philips, как простая шина внутренней связи для создания управляющей электроники. Используется для соединения низкоскоростных периферийных компонентов с материнской платой, встраиваемыми системами и мобильными телефонами. I²C использует две двунаправленных линии, подтянутые к напряжению питания и управляемые через открытый коллектор

или открытый сток — последовательная линия данных SDA и последовательная линия тактирования SCL. Стандартные напряжения +5 В или +3,3 В, однако допускаются и другие.

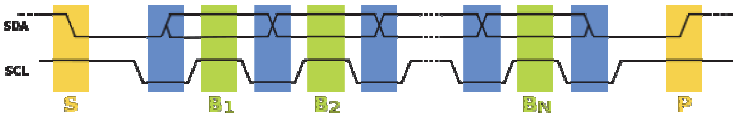


Рисунок 1. Тактирование последовательности передачи данных.

Процесс проектирования был произведен в несколько этапов:

1. Выбор микросхемы

Для реализации устройства была выбрана ПЛИС фирмы Actel кристалл AGLN250V2. Данная организация производит цифровые микросхемы с малым энергопотреблением (до 5 мкВт), что способствует применению их в портативных устройствах, где ограничено энергопотребление.

2. Разработка алгоритма

Вторым этапом была разработка алгоритма работы ведущего и ведомого устройства. Алгоритм работы ведущего и ведомого устройства был взят из стандарта «THE I²C-BUS SPECIFICATION» версии 2.1. Для передачи мастер устройство вырабатывает СТАРТ, это изменение уровня на линии SDA с 1 на 0, в то время когда SCL в 1, дальше идёт адресный байт и направление, адрес состоит из восьми бит, после адреса следом за ним, бит записи(0) или бит чтения (1), в это время на SCL, всё это время на SCL генерируются импульсы с длиной логического нуля 4.7 us и логической единицы 4.0 us. Фактически все изменения на шине SDA происходят когда SCL в нуле, чтение с шины когда SCL в единице. В спокойном состоянии линия SDA и линия SCL находятся в единице, они подтянуты к ней с помощью резистора. Следовательно все общение по этим линиям происходит нулём. Надо выставить бит, кидаем SCL в ноль и выставляем бит, отпускаем и так до восьмого бита, после него подаём импульс SCL для подтверждения от ведомого с помощью бита ACK, в случае передачи, ведомый будет держать шину

в нуле до тех пор пока не будет готов к приёму следующего байта информации, а в это время мы выставим на линию SDA первый бит следующего байта, как только ведомый отпускает SCL мы выставим следующий бит. Если ack принимаемое от ведомого будет равно единице, то мастер имеет право уйти на СТОП. Изменение шины SDA с нуля на единицу, в то время когда SCL в единице. За одну транзакцию может быть передано сколько угодно байт. В момент приема ведомый будет выставлять данные на шину в то время когда SCL находится нуле, частоту на SCL всегда вырабатывает мастер, ведомый может её лишь задерживать в нуле. Мастер будет читать шину в то время когда SCL находится в единице после каждого успешно принятого байта также выдаёт бит ACK. В транзакции должен быть передан байт длины сообщения. С помощью него мастер выдаёт стоп, после последнего успешно принятого байта.

3. Организация архитектуры будущей программы

Третьей стадией стала организация “скелета” для будущего контроллера. На этом этапе был произведен анализ и сделаны выводы о том, что реализация I²C контроллера на операторе ветвления “case” в ПЛИС занимала меньше ресурсов, чем идентичные ей конструкции, в том числе и “If ”.

4. Разработка программного обеспечения

Четвёртым этапом была осуществлена разработка ПО. Написание программы для ведомого устройства велось параллельно с написанием программы для ведущего и записывалось в тестовые воздействия. В процессе моделирования были выявлены и устранены некоторые недочеты связанные с двунаправленностью шины.

5. Синтез и моделирование

На заключительном этапе был выполнен синтез кристалла и моделирование воздействий на устройство.

РЕАЛИЗАЦИЯ ФУНКЦИИ САМОКОНТРОЛЯ ДЛЯ ПРОГРАММНЫХ ПРОЦЕССОРОВ

Е.Д. Кашаев, С.П. Хворостухин

ФГБОУ ВПО «Пензенский государственный университет», факультет приборостроения, информационных технологий и систем, кафедра «Информационная безопасность систем и технологий»

В последнее время часто цифровая аппаратура разрабатывается как «система на кристалле», в которой наряду с цифровой аппаратной логикой используются программные процессоры, что экономически выгодно и процесс проектирования становится проще.

Программный процессор – это микропроцессорное ядро, которое может быть полностью создано с использованием только логического синтеза. Он реализуется с использованием различных полупроводниковых устройств, содержащих программируемую логику – ПЛИС [1]. Однако, применение программных процессоров реализованных на базе ПЛИС в системах реального времени, критичных к возможным ошибкам, требует применение контроля за вычислительным процессом [2]. Крайне важно применение подобных механизмов в аппаратуре, работающей в условиях неблагоприятного воздействия окружающей среды (электромагнитные возмущения, установка на подвижные объекты и т.д.).

При разработке функциональных блоков программного процессора для специализированных систем должны быть заложены функции контроля за достоверностью вычисления. Следует рассмотреть реализацию данных функций в программных процессорах на ПЛИС.

Рассматривая необходимость контроля, можно разделить данную задачу на две:

- контроль инструкций и данных, поступающих в процессор;
- контроль за данными, размещенными в регистрах процессора и поступающими на выход.

Традиционные методы обеспечения целостности (контрольные суммы, хэш-функции) являются не эффективными, по причине снижения быстродействия и значительного потребления ресурсов микросхемы ПЛИС.

Для обеспечения функции самоконтроля и самоблокировки в программных процессорах может быть осуществлен перевод с двоичной арифметики на арифметику, построенную на числах Фибоначчи. В отличие от классической двоичной системы код Фибоначчи является избыточным кодом. При этом его избыточность кода проявляет себя в свойстве многозначности представления натуральных чисел. Это свойство лежит в основе контроля всех арифметических операций и обеспечивает другие важные технические преимущества данного кода. Арифметические и схемотехнические основы [3÷5] показывают, что в кодах Фибоначчи можно выполнять все логические и арифметические операции. При этом эти коды позволяют создавать компьютерные устройства и структуры, обладающие свойством самоконтроля всех компьютерных структур (счетчиков, регистров, таймеров и др.). При этом важным свойством данных является сохранение двоичного представления данных, что позволяет реализацию устройств, работающих в кодах Фибоначчи, на базе ПЛИС.

На рисунке 1 представлена схема регистра свертки для процессора, работающего с кодами Фибоначчи.

Операция свертки является одной из базовых микроопераций. Все возможные логические и арифметические операции могут быть сведены к базовым микрооперациям [5].

Операция свертки является одной из базовых для арифметики Фибоначчи и обеспечивает возможность самоконтроля без изменения числовой информации, записанной в регистре.

Из принципа функционирования регистра свертки вытекает, что логическая 1 возникает на контрольном выходе только для двух ситуаций:

- двоичная комбинация, записанная в регистр свертки не представлена в минимальной форме. Это означает, что условие свертки выполняется хотя бы для одной тройки соседних триггеров регистра. Это вызывает появление ло-

гический 1 на выходе соответствующего элемента AND. Следовательно, в этом случае появление логический 1 на контрольном выходе регистра свертки указывает на то, что процесс свертки не завершен;

- возникновение постоянной логической 1 на контрольном выходе регистра свертки является свидетельством отказа в регистре свертки.

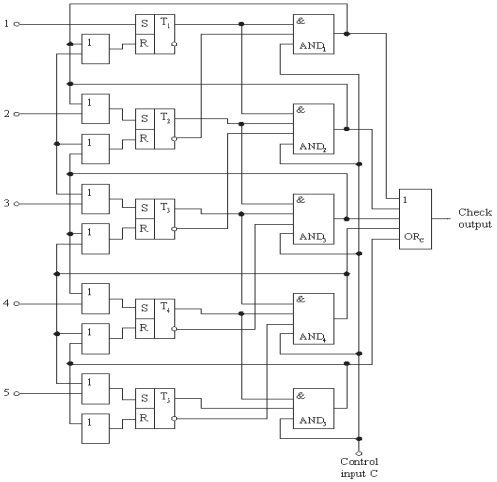


Рисунок 1. Регистр свертки.

Регистр свертки состоит из пяти RS-триггеров и логических элементов AND, OR, которые предназначены для реализации свертки, что позволяет реализовать данный регистр с использованием стандартных примитивов синтеза [6].

Реализация данной схемы для ПЛИС фирмы Xilinx Spartan 2E, требует всего 3 ячеек (slices). Увеличение разрядности регистра существенно не увеличивает потребляемых ресурсов. В общем случае, количество использованных ячеек определяется количеством триггеров в схеме, то есть разрядностью регистра. Для реализации были использованы примитивы библиотеки Unisim: FDRS, OR2, AND3, AND4, OR5.

На рисунке 2 представлена блок-схема модуля самоконтроля для операции свертки.

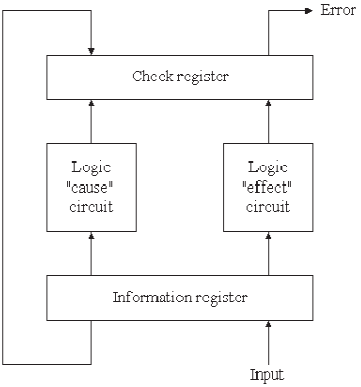


Рисунок 2. Блок-схема модуля самоконтроля на базе регистра свертки.

Модуль состоит из информационного и контрольного регистров, которые связаны с помощью логических схем "cause" ("причина") и "effect" ("следствие"). Кодовая информация, поступающая в информационный регистр, анализируется логической схемой "cause" ("причина"). Наличие нулевой кодовой комбинации в контрольном регистре после завершения всех микроопераций, означает что все микрооперации выполнены правильно. Если контрольный регистр содержит хотя бы одну двоичную 1 в некотором триггере, это означает что, по крайней мере, хотя бы одна базовая микрооперация выполнена неправильно. Двоичные единицы в триггерах контрольного регистра вызывают сигнал ошибки на выходе модуля [5].

Если на контрольном выходе модуля формируется двоичный сигнал 1 (наличие ошибки), то все информационные выходы модуля блокируются. Чтобы исправить ошибку, необходимо повторить предшествующую микрооперацию. Если при повторении на контрольном выходе возникает двоичный сигнал 0 (отсутствие ошибки), то это означает, что ошибка возникла в результате случайного сбоя,

и блокирование информационных выходов снимается. Если при повторении микрооперации двоичный сигнал 1 снова появляется на выходе ошибка, это является следствием отказа в модуле [5].

Возможность самоконтроля арифметических операций в процессорах является необходимым условием для применения в специализированных системах. Применение кодов Фибоначчи в программных процессорах позволяет обеспечить данное требование, сохраняя простоту реализации вычислительных устройств на ПЛИС с использованием стандартных примитивов синтеза.

Литература

1. Вавилин А.И., Бурмистров А.В. Обзор программных процессоров для ПЛИС фирмы Xilinx//Успехи современного естествознания, № 7, 2011.
2. Зыль С. Безопасность систем жесткого реального времени//Открытые системы, №7, 2008.
3. Стахов А.П. Введение в алгоритмическую теорию измерения. – М: Советское Радио, 1977.
4. Стахов А.П. Коды золотой пропорции. – М: Радио и связь, 1984.
5. Стахов А.П. Помехоустойчивые коды: Компьютер Фибоначчи. Серия «Радиоэлектроника и связь», вып.6 – М.: Знание, 1989.
6. Зотов В. Разработка VHDL-описаний цифровых устройств, проектируемых на основе ПЛИС фирмы Xilinx, с использованием шаблонов САПР ISE Design Suite. Часть 8// Компоненты и технологии, №9, 2010.

ОСОБЕННОСТИ ИСПОЛЬЗОВАНИЯ ПЛИС В СРЕДСТВАХ ЗАЩИТЫ ИНФОРМАЦИИ

Е.Д. Кашаев, С.П. Хворостухин, М.А. Дмитриев, А.И. Ильин

ФГБОУ ВПО «Пензенский государственный университет», факультет приборостроения, информационных технологий и систем, кафедра «Информационная безопасность систем и технологий»

При проектировании средств защиты значительное внимание уделяется выбору элементной базы. На данном этапе следует принять решение по типу используемой логики: использование стандартных серийных микросхем, заказные микросхемы либо элементы программируемой логики. Каждый из данных вариантов имеет как достоинства, так и недостатки.

В связи с невозможностью использования иностранной элементной базы в средствах защиты критичных устройств и систем и отсутствием аналогов российского производства, необходимо рассмотреть преимущества и недостатки их замены компонентами на базе ПЛИС отечественного производства.

Одним из вариантов подобной замены является возможность использования программных процессоров и многопроцессорных систем, построенных на базе кластеров подобных процессоров.

В качестве достоинств использования программных процессоров и ПЛИС при построении средств защиты информации следует отметить следующие:

- гибкость процесса разработки аппаратных компонентов средств защиты – обеспечивается простотой модификации проекта при внесении новых функций, возможностью настройки функциональных блоков под требования целевой системы. Использование системы на кристалле, адаптированной под требуемые задачи, позволяет получить большую производительность в сравнении с системами, построенными на универсальных микросхемах, и меньшую стоимость разработки, в сравнении с заказными микросхемами;

- возможность верификации проекта – в соответствие с требованиями руководящих документов аппаратное и программное обеспечение, используемое в средствах защиты, должно проходить специальные проверки на наличие не декларированных возможностей [1÷3]. Возможность предоставления VHDL-модулей функциональных блоков ПЛИС значительно упрощает процедуру верификации [4] проекта на требования руководящих документов и соответствие техническому заданию;
- компактность системы – обеспечивается возможностью построения всей системы на одной или нескольких микросхемах ПЛИС вместо набора специализированных микросхем, что позволяет значительно снизить уровни электромагнитного излучения и выделяемого тепла, а также позволяет уменьшить габариты проектируемых устройств. Сокращение числа проводников и уменьшение их длины позволяет уменьшить утечку информации по ПЭМИН при обработке. Компактность системы также обеспечивает возможность интеграции разработанных аппаратных модулей защиты в универсальные средства вычислительной техники (применение мезонинных плат) [5];
- возможность интеграции в микропроцессорную систему специализированных сопроцессоров для повышения производительности при выполнении типовых операций целевой системы, например сопроцессор модулярной арифметики для повышения скорости выполнения криптографических операций (например, для систем шифрования RSA).

В качестве недостатков данного решения следует отметить:

- необходимость защиты программных компонентов - проекты, реализованные на ПЛИС класса FPGA, уязвимы для копирования, поскольку конфигурационный поток данных может быть перехвачен в момент загрузки при старте системы [6]. Защита проекта от несанкционированного доступа требует дополнительных ресурсов, например использование микросхем специальной памяти [6];
- возможность утечки информации на уровне микросхемы ПЛИС, за счет электромагнитных эффектов внутри кристалла и печатной платы – при проекти-

ровании ПЛИС значительное внимание уделяется задаче обеспечения целостности сигналов и устранения задержек распространения сигнала в микросхеме ПЛИС [7]. Однако, эффекты перекрестных помех и задержки сигналов, возникающие в микросхеме ПЛИС, служат источником утечки информации по техническим каналам. С целью снижения возможности утечки и уровня возникающих паразитных сигналов необходимо производить разводку функциональных блоков с учетом взаимного влияния соседних информационных линий. При изменении логических уровней сигналов, протекающих в расположенных рядом проводниках, емкостная связь между дорожками внутри микросхемы приводит к тому, что часть зарядов перетекает из одного проводника в другой [7]. С повышением рабочей частоты взаимное влияние, оказываемое сигналами друг на друга (перекрестные помехи) и их взаимодействие с окружающей средой (электромагнитные излучения) начинает возрастать [8];

- высокая стоимость разработки программных продуктов.

Отмеченные преимущества можно рассмотреть на примере замены микропроцессорной системы, построенной на стандартных микросхемах отечественного производства, на систему на кристалле (SoC) реализованную на ПЛИС. Основу данной системы составляет 32-разрядное RISC-ядро, дополненное специализированными модулями для повышения скорости фильтрации. Данная замена позволила повысить частоту работы микропроцессорной системы в 1,5 раза (со 100 МГц до 150 МГц) и уменьшить вдвое габариты устройства.

Из рассмотренных достоинств и недостатков следует, что решение использовать ПЛИС в качестве аппаратной основы средств защиты обеспечивает ряд существенных преимуществ. Однако, отмеченные недостатки требует особого подхода к разработке компонентов, и в частности программных процессоров, на базе ПЛИС.

Литература

1. ГОСТ Р 51275-2006. «Защита информации. Объект информатизации. Факторы, воздействующие на информацию. Общие положения».

2. Руководящий документ. Защита от несанкционированного доступа к информации. Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия недекларированных возможностей.
3. Доктрина информационной безопасности Российской Федерации.
4. Руководящий документ. Защита от несанкционированного доступа к информации Термины и определения.
5. Лысенко В.В., Счастный Д.Ю. Построение систем защиты информации на базе контроллера "Аккорд СБ/2". Электронный ресурс. Режим доступа: http://www.okbsapr.ru/lisenko_schastnii.html, свободный.
6. Комолов Д., Золотухо Р. Использование микросхем специальной памяти для обеспечения защиты FPGA от копирования//Компоненты и технологии, №12, 2008.
7. Максфилд К. Проектирование на ПЛИС. Курс молодого бойца – М.: «Додэка-XXI», 2007 – 408 с.
8. Говард Джонсон, Мартин Грэхем. Конструирование высокоскоростных цифровых устройств: начальный курс черной магии: Пер. с англ. – М.: «Вильямс», 2006 – 624 с.

РЕАЛИЗАЦИЯ ДАТЧИКА ПСЕВДОСЛУЧАЙНОЙ ПОСЛЕДОВАТЕЛЬНОСТИ НА ПЛИС СЕМЕЙСТВА XILINX

А.А. Кузьминов, А.П. Иванов

ФГБОУ ВПО «Пензенский государственный университет», факультет приборостроения, информационных технологий и систем

Современные системы защиты информации имеют многоэлементную структуру. В процессе разработки аппаратуры данного профиля встает естественная задача проверки и настройки, как отдельных узлов, так и системы защиты информации в целом. Поскольку аппаратура защиты информации в основном разрабатывается на цифровой элементной базе, то решение задачи сводится к подаче на вход отдельного узла тестового сигнала и сравнении сигнала на выходе узла с эталонным сигналом.

Наиболее подходящим тестовым сигналом является псевдослучайная последовательность (ПСП). Она позволяет проверить аппаратуру защиты информации в условиях действия трудно проверяемых неисправностей за счёт чёткой идентификации и хороших корреляционных свойств. Обычно для получения такой последовательности используются датчики ПСП, которые могут быть реализованы на базе рекуррентных линий задержек (РЛЗ) и различных логических элементах.

РЛЗ представляют собой регистр сдвига (набор специальным образом последовательно соединённых триггеров), в который определенным образом вводится обратная связь на основе элемента исключающего ИЛИ. В связи с этим сами РЛЗ и датчики ПСП можно реализовывать с помощью ПЛИС.

Проектирование цифровых устройств на основе ПЛИС заключается в выполнении следующих этапов в системе проектирования [1]:

1. Создание принципиальной схемы проектируемого устройства в схемотехническом редакторе или описании данного устройства на языке VHDL или Verilog;

2. Предварительное функциональное или временное моделирование для выявления ошибок и проверки работоспособности проектируемого проекта и отдельных его частей;

3. Привязка выводов проекта к входам-выводам кристалла, выбор выходных уровней, критичных цепей и т. д.

4. Запуск автоматизированного размещения проекта в кристалле и анализ генерируемых отчетов для выявления предупреждений и ошибок, а при отсутствии таковых переход к следующему этапу.

5. Верификация проекта, т. е. окончательное временное моделирование после размещения проекта в кристалле при всех реальных задержках распространения сигналов внутри микросхемы ПЛИС.

6. Конфигурирование кристалла ПЛИС с помощью битового потока.

В данной работе рассмотрен первый этап проектирования цифрового устройства на основе ПЛИС применительно к реализации датчика ПСП, а именно описание данного устройства на языке VHDL.

В результате выполнения данного этапа проектирования должен быть реализован модуль датчика ПСП, приведённый на рисунке 1, где слева обозначены входные сигналы: areset – асинхронный сигнал сброса всех РЛЗ в начальное состояние, clk – сигнал тактовой синхронизации, en_i – сигнал разрешения сдвига i-ой РЛЗ, а справа выходной сигнал do – выходной бит датчика ПСП. Все входные сигналы активны по переднему фронту.

Структурная схема датчика ПСП представлена на рисунке 2, где датчик ПСП состоит из четырёх рекуррентных линий задержек, трёх отдельно вынесенных сумматоров по модулю два, двухразрядного мультиплексора и элемента задержки (D-триггер). РЛЗ строится на основе порождающего многочлена с помощью последовательно соединённых D-триггеров и сумматора по модулю два, соединяющего выходы двух триггеров. Пример РЛЗ с порождающим многочленом x^4+x+1 приведён на рисунке 3, где сдвиг РЛЗ происходит от младших разрядов к старшим.

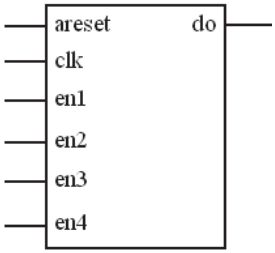


Рисунок 1. Модуль датчика ПСП.

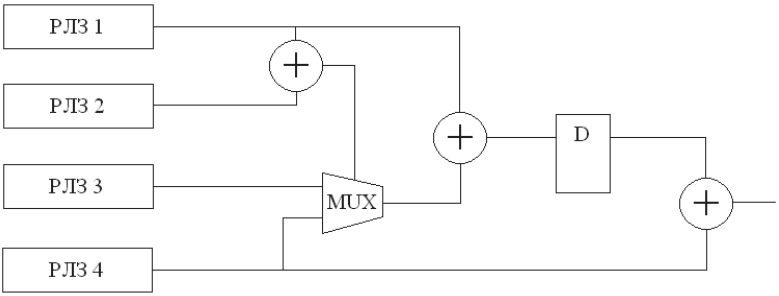


Рисунок 2. Структурная схема датчика ПСП.

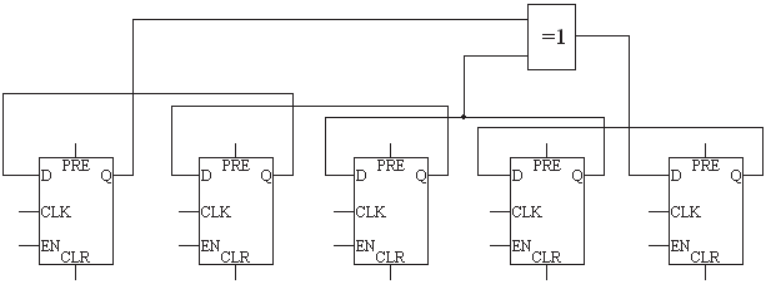


Рисунок 3. РЛЗ с порождающим многочленом x^4+x+1.

Алгоритм работы датчика ПСП следует разделить на два этапа. На первом этапе разрабатывается алгоритм работы РЛЗ, а на втором – алгоритм работы датчика ПСП в целом.

Алгоритм работы РЛЗ следующий:

1) Если сигнал areset равен единице, то триггеры РЛЗ (rlz_1) принимают значения в соответствии с начальным значением (tmp1), определённым ранее.
2) Если сигнал areset равен нулю, а так же происходит изменение сигнала тактовой синхронизации с нуля на единицу и сигнал разрешения сдвига равен единице, вычисляется бит обратной связи (rlz1) и сдвигается содержимое РЛЗ на один разряд. Бит обратной связи, который получается путём суммирования по модулю два сигналов со старшего триггера (N1) и точки съёма (TS1), записывают в триггер, соответствующий младшему разряду порождающего многочлена.

В соответствии с данным алгоритмом разработан программный код для языка VHDL. Ниже приведен программный код реализации первой РЛЗ (см. рисунок 3):

```
process(clk, areset)
        begin
        if (areset = '1') then
                rlz_1 <= tmp1;
        elsif (clk'event and clk='1') then
                if (en1 = '1') then
                        rlz1 <= rlz_1(N1) xor rlz_1(TS1);
                        rlz_1 <= rlz_1(N1-1 downto 0) & rlz1;
                end if;
        end if;
end process;
```

Алгоритм работы датчика ПСП в целом следующий:

1. Биты обратных связей первой (rlz1) и второй (rlz2) РЛЗ суммируются между собой по модулю два, и этот результат (rlz_1_2) является битом управления на мультиплексоре.

2. На входы мультиплексора поступают биты обратных связей третей (rlz3) и четвёртой (rlz4) РЛЗ.

3. Если бит управления (rlz_1_2) равен нулю, то на выход мультиплексора (mux_1) поступает бит (rlz3). В противном случае - бит (rlz4).

4. Бит с выхода мультиплексора (mux_1) суммируется с битом с выхода первой РЛЗ по модулю два, и результирующий бит (rlz_1_3_4) поступает на вход D-триггера (элемент задержки сигнала на такт).

5. Выходной сигнал датчика ПСП получается путём суммирования по модулю два бита с выхода элемента задержки (rlz_1_3_4_z) и бита с выхода четвёртой РЛЗ.

В соответствии с данным алгоритмом разработан программный код для языка VHDL:

```
process(clk)
        begin
                if (clk'event and clk='1') then
                        rlz_1_3_4_z <= rlz_1_3_4;
                end if;
        end process;
rlz_1_2 <= rlz1 xor rlz2;
mux_1 <= rlz3 when rlz_1_2 = '0' else rlz4;
rlz_1_3_4 <= rlz1 xor mux_1;
do <= rlz_1_3_4_z xor rlz4;
```

Таким образом, рассмотрен первый этап проектирования цифрового устройства на основе ПЛИС семейства Xilinx применительно к реализации датчика ПСП, а именно описание данного устройства на языке VHDL. Работа выполнена в рамках целевой подготовки на базовом предприятии кафедры «Информационная безопасность систем и технологий» Пензенского государственного университета. В дальнейшем планируется реализация разработанного модуля для использования в отдельных узлах аппаратуры защиты информации.

Литература

1. Зотов В.Ю. Проектирование цифровых устройств на основе ПЛИС фирмы Xilinx в САПР Web PACK ISE. – М.: Горячая линия-Телеком, 2003. – 624с.

ПРОГРАММНЫЙ КОМПЛЕКС, РЕАЛИЗУЮЩИЙ КОНТРОЛЬ ЦЕЛОСТНОСТИ ЗАГРУЖАЕМЫХ ПОЛЬЗОВАТЕЛЕМ ДАННЫХ ВО ВНУТРЕННЮЮ ПАМЯТЬ И ВНЕШНЮЮ ФЛЕШ-ПАМЯТЬ ПЛИС

Д.А. Молянов, В.А. Мали

ФГБОУ ВПО «Пензенский государственный университет», Факультет приборостроения, информационных технологий и систем

Современный этап развития цифровой техники характеризуется широким применением программно-аппаратных комплексов, которые строятся по принципу «система на кристалле» [1]. При этом в качестве элементной базы для реализации таких систем все чаще используются программируемые логические интегральные схемы (ПЛИС) [1].

В настоящее время наиболее распространенные серии ПЛИС имеют следующую архитектуру:

– CPLB (Complex Programmable Logic Device), устройства, использующие для хранения конфигурации энергонезависимую память;

– FPGA (Field Programmable Gate Array), устройства, использующие для хранения конфигурации энергозависимую память, которая требует инициализации после включения питания [2].

ПЛИС CPLD относятся к устройствам начального уровня и предназначены для реализации устройств небольшого логического объема [2].

По сравнению с архитектурой CPLD, ПЛИС FPGA содержат много большее число меньших по размерам отдельных логических блоков и имеют развитую распределенную структуру внутренних соединений, которая занимает почти весь кристалл [3]. Помимо большего объема логических ресурсов, современные ПЛИС FPGA отличаются наличием аппаратно выделенных ресурсов для решения типичных задач цифровой обработки сигналов и используются для построения более сложных устройств.

ПЛИС серий FPGA выполнены на основе статического ОЗУ. Так как информация о конфигурации кристалла записывается во внутреннее «теневое» ОЗУ, то при выключении источника питания эти данные не сохраняются [4]. Конфигурация (прошивка) обычно хранится в ПЗУ, расположенном на одной плате рядом с ПЛИС [5]. После включения питания или по сигналу сброса она автоматически переписывается в программирующий сдвиговый регистр ПЛИС [5].

При использовании ячеек статического ОЗУ может оказаться довольно сложно защитить интеллектуальную собственность, реализованную в данном устройстве [6]. Это обусловлено тем, что конфигурационный файл, используемый для программирования устройства, в том или ином виде хранится во внешней памяти [6].

Для противодействия несанкционированному исследованию и копированию конфигурационных данных производители ПЛИС предлагают различные методы, такие как запрет на чтение внешней конфигурационной памяти путем использования специального бита секретности, криптографические преобразования над хранящейся во внешнем ПЗУ конфигурацией ПЛИС и другие [7]. Могут использоваться и более сложные программно-аппаратные способы, «привязывающие» конфигурацию ПЛИС к конкретной плате [8].

Однако использование методов защиты, предлагаемых производителями ПЛИС, может быть недостаточно или неприемлемо. В таких случаях речь идет о недостаточной степени доверия к реализуемым стандартными защитными мерами алгоритмам и (или) о невозможности их использования. Например, программирование ПЛИС при помощи стандартных средств разработки позволяет осуществить как контроль целостности, так и шифрование, но только для определенных семейств ПЗУ и только в случае загрузки конфигурационных файлов.

Целью данной статьи является разрешение одного из возможных противоречий между доступностью и возможностями/надежностью стандартных методов защиты ПЛИС. Задача статьи – разработка программного комплекса для аппаратуры передачи данных (АПД) на базе ПЛИС, реализующего контроль целостности загружаемых пользователем данных во внутреннюю память и внешнюю

флеш-память ПЛИС. Назначение программного комплекса состоит в использовании его при разработке аппаратуры на базе ПЛИС, требующей высокого уровня доверия к процессу.

Для АПД (например, маршрутизаторов) наиболее важным параметром, определяющим их эффективность, является быстродействие, которое при этом может быть дополнено другими условиями, например, интервалом рабочих температур. Во многих случаях выбор ПЛИС в качестве основной элементной базы для реализации АПД является оптимальным решением, учитывая те специальные задачи, которые зачастую ставятся перед разработчиками. Примером такой задачи является реализация технологии виртуальных частных сетей (VPN).

В данной статье подходящая модель ПЛИС выбирается, исходя из следующих требований:

- возможность передачи данных со скоростью до 1 Гбит/с;
- возможность работы при температуре от минус 50 до плюс 50 °С;
- направленность на цифровую обработку сигналов;
- доступность элементной базы и средств проектирования (САПР);
- низкая стоимость.

С учетом изложенных требований выбор по факту осуществлялся между ПЛИС фирм Xilinx и Altera, являющихся ведущими мировыми производителями ПЛИС [2]. Каждая из них выпускает целый спектр продукции, включая ПЛИС с различной архитектурой, флеш-ПЗУ для хранения конфигурации, САПР, средства программирования и отладки [2].

По результатам анализа в качестве элементной базы для реализации АПД были выбраны ПЛИС семейства Virtex-4 FX, производимых фирмой Xilinx. ПЛИС данного семейства удовлетворяют всем предъявленным требованиями по быстродействию и условиям функционирования и условиям функционирования, кроме того, отечественными разработчиками накоплен большой опыт работы с ПЛИС Virtex, начиная с семейства Virtex-2.

Также в составе поставляемой САПР Xilinx Platform Studio присутствует отладчик прикладных программ GNU Debugger (GDB), а также средство отладки

Xilinx Microprocessor Debugger (XMD), которые позволяют осуществить как чтение, так и запись в память ПЛИС. GDB при этом используется в паре с XMD, что позволяет осуществить отладку на аппаратном уровне на конкретной ПЛИС.

Для достижения поставленной задачи предлагается следующий алгоритм контроля целостности. Вначале осуществляется предварительный расчет контрольной суммы для данных, загружаемых в память ПЛИС, и рассчитанная сумма передается вместе с данными. После этого производится расчет контрольной суммы для загруженных в память ПЛИС данных, и полученные значения переданной и вычисленной контрольных сумм сверяются. При этом используется один и тот же заранее выбранный способа контроля целостности для вычисления контрольной суммы.

Для выбранного алгоритма целесообразным является использование двух взаимодействующих программных модулей для реализации такого метода. Первый программный модуль находится в памяти локального компьютера, то есть компьютера, к которому через соответствующий интерфейс (например, USB) подключен загрузочный кабель ПЛИС. Этот модуль осуществляет следующие функции: расчет контрольной суммы для загружаемых данных, запись данных в память ПЛИС, передача вычисленной контрольной суммы и иных параметров второму программному модулю, получение результата контроля целостности от второго программного модуля.

Второй программный модуль находится в памяти ПЛИС и выполняется под ее управлением. Он осуществляет следующие функции: получение от первого программного модуля переданной контрольной суммы и иных параметров, расчет контрольной суммы для загруженных в память ПЛИС данных, сравнение вычисленного и переданного значений контрольных сумм и передача результата сравнения первому модулю.

В предложенном алгоритме и для выбранного семейства ПЛИС взаимодействие первого и второго программных модулей напрямую невозможно, но возможно косвенное взаимодействие при помощи операций чтения и записи областей памяти ПЛИС с прямым доступом. В эти области памяти записываются дан-

ные, необходимые для взаимодействия двух модулей, например: статус (флаг) успешной загрузки данных, начальный адрес загруженных данных и их размер, переданное и вычисленное значение контрольной суммы, флаг ошибки и т.д.

Взаимодействие модулей осуществляется путем передачи соответствующих команд согласно протоколу RSP на сервер GDB, который интерпретирует и передает принятые команды в вид, «понятный» соответствующему контроллеру ПЛИС. В качестве сервера GDB используется фирменный отладчик XMD, запущенный на локальной машине. Взаимодействие ПЛИС с локальной машиной осуществляется по интерфейсу JTAG с использованием загрузочного кабеля Platform cable USB. Схема взаимодействия двух программных модулей приведена на рисунке 1.

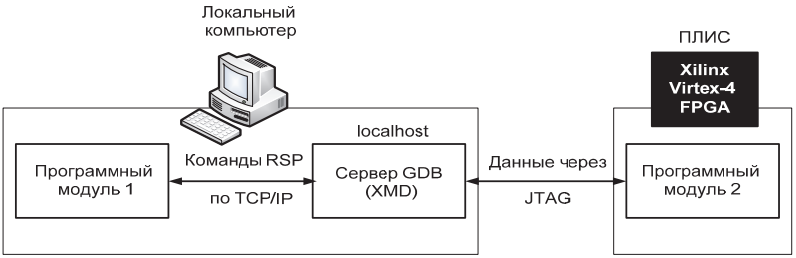


Рисунок 1. Схема взаимодействия программных модулей в процессе работы.

Тестирование разработанного программного комплекса показало, что он обладает следующими возможностями:

- скорость загрузки данных в блочную память ПЛИС – до 1 Мбит/с;
- скорость загрузки данных во внешнюю флеш-память ПЛИС – до 272 Кбит/с;
- время проверки целостности по алгоритму CRC32 загружаемого файла размером 1,8 Мб – до 10 с;
- объем памяти, занимаемой внутренним модулем – 15 Кб.

Результатом проведенных исследований является программный комплекс, позволяющий осуществить контроль целостности загружаемых пользователем данных во внутреннюю память и внешнюю флеш-память ПЛИС. Данный про-

граммный комплекс может быть использован в процессе разработки аппаратуры на базе ПЛИС, требующем высокого уровня доверия. Таким образом, путем разработки нового средства разрешается одно из возможных противоречий между доступностью и возможностями/надежностью стандартных методов защиты ПЛИС.

Литература

1. Зотов В. Ю. Проектирование встраиваемых микропроцессорных систем на основе ПЛИС фирмы Xilinx – М.: Горячая Линия, 2006. – 520с.
2. Тарасов И.Е. Разработка цифровых устройств на основе ПЛИС Xilinx с применением языка VHDL – М.: Горячая Линия, 2006. – 252с.
3. Дж. Ф. Уэйкерли. Проектирование цифровых устройств. Том 1 М.: Постмаркет 2002. – 543с.
4. Зотов В. Ю. Проектирование встраиваемых микропроцессорных систем на основе ПЛИС фирмы Xilinx в САПР Webpack ISE – М.: Горячая Линия-, 2003. – 624с.
5. Сергиенко А. М. VHDL для проектирования вычислительных устройств – К ЧП «Корнейчук», ООО «ТИД «ДС» 2003. – 208с.
6. Максфилд К. Проектирование на ПЛИС – М.: Издательский дом «Додэка-XXI» 2007. – 408с.
7. Virtex-4 Configuration guide [Электронный ресурс].
8. FPGA IFF Copy Protection Using Dallas Semiconductor/Maxim DS2432 Secure EEPROMs [Электронный ресурс].

Часть 3. Обработка сигналов

СКРЕМБЛИРУЮЩИЕ ПОСЛЕДОВАТЕЛЬНОСТИ ДЛЯ СИСТЕМ ПЕРЕДАЧИ ДАННЫХ

С.С.Савельев

ОАО «БСКБ «Восток»

Во многих военных стандартах США для аппаратуры КВ связи в процедурах установления связи и передачи заголовочной информация используется метод прямого расширения спектра. Использование этого метода позволяет повысить достоверность передачи информации за счет уменьшения информационной скорости передачи при полном использовании полосы частот канала связи.

Для расширения спектра в указанных стандартах используют последовательность из 32 символов, каждый из которых является одним из символов сигнально-кодовой конструкции представленной на рисунке 1.

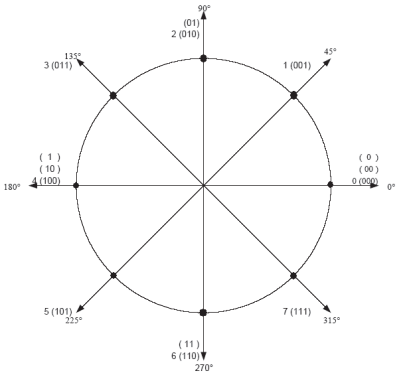


Рисунок 1. Сигнально-кодовая конструкция.

Последовательность называется скремблирующей и имеет следующий вид

7 4 3 0 5 1 5 0 2 2 1 1 5 7 4 3 5 0 2 6 2 1 6 2 0 0 5 0 5 2 6 6

Одним из требований предъявляемым к скремблирующей последовательности, используемой для целей указанных выше, является минимизация уровня боковых лепестков ее корреляционной функции. При выполнении этого условия достигается максимальная достоверность обнаружения скремблирующей последовательности на фоне помех.

Для максимизации достоверности обнаружения скремблирующей последовательности на фоне помех была поставлена задача поиска 32-символьной скремблирующей последовательности с корреляционной функцией лучшей, чем у исходной последовательности.

Главный лепесток скремблирующей последовательности равен 32. Максимальный уровень боковых лепестков исходной скремблирующей последовательности равен 11/32. С помощью программы, осуществляющей случайный перебор скремблирующих последовательностей и вычисление уровня их боковых лепестков, было найдено более 100 скремблирующих последовательностей с уровнем боковых лепестков менее 6/32. Часть из найденных скремблирующих последовательностей приведена в Приложении. Рядом с каждой скремблирующей последовательностью приведен максимальный уровень ее боковых лепестков. Первыми в приложении приведены скремблирующие последовательности с уровнем боковых лепестков менее 4,7/32. Их корреляционные функции изображены на рисунках 1-3 сплошной линией. Для сравнения на тех же рисунках пунктирной линией изображена корреляционная функция исходной скремблирующей последовательности.

Скремблирующая последовательность, использованная в стандарте MIL-STD-188-110B, кроме хорошей корреляционной функции удовлетворяет еще ряду требований. Найденные скремблирующие последовательности не исследовались на соответствие всем требованиям, предъявляемым к подобным последовательностям. В связи со сказанным не следует делать однозначных выводов о большей или меньшей пригодности найденных скремблирующих последовательностей для расширения спектра в системах передачи данных, однако, использование их для синхронизации, безусловно, оправдано.

Кроме того, найденные последовательности составляют лишь небольшую часть из всех возможных последовательностей, так как перебор был случайным и охватил лишь незначительную часть и 8**32 всех возможных последовательностей.

Актуальной задачей, на мой взгляд, является задача поиска алгоритма генерации скремблирующих последовательностей с заданными свойствами, а также определения максимально достижимых характеристик по каждому из критериев.

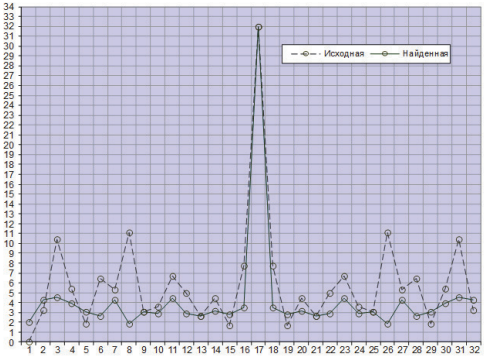


Рисунок 1

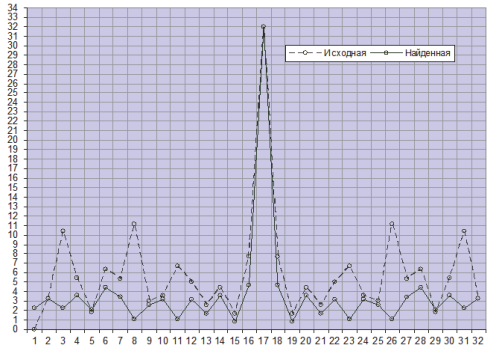


Рисунок 2

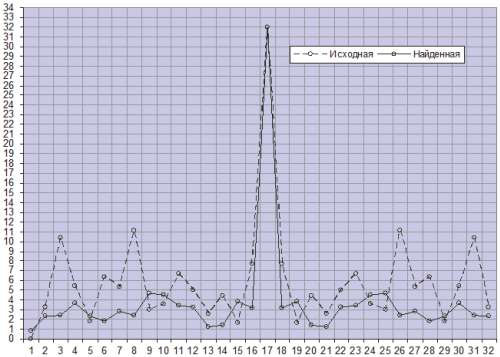


Рисунок 3

Таблица

Макс. уровень бокового лепестка КФ	Скремблирующая последовательность
4.525175,	2, 4, 1, 6, 3, 1, 4, 7, 5, 5, 1, 2, 1, 2, 7, 0, 3, 6, 6, 0, 7, 0, 5, 1, 7, 2, 1, 7, 3, 2, 5, 6
4.660048,	7, 5, 7, 3, 2, 7, 6, 5, 7, 7, 7, 2, 1, 5, 1, 2, 1, 0, 7, 4, 5, 4, 5, 0, 7, 3, 5, 7, 3, 4, 4, 4
4.689305,	6, 4, 0, 3, 7, 7, 6, 4, 0, 1, 6, 7, 2, 5, 3, 7, 0, 0, 6, 5, 7, 6, 2, 2, 2, 3, 3, 0, 0, 4, 1, 1
5.030138,	6, 6, 4, 1, 5, 1, 1, 5, 0, 1, 0, 5, 0, 3, 2, 7, 7, 6, 3, 4, 1, 0, 2, 5, 2, 5, 6, 6, 0, 6, 0, 4
5.098752	2, 6, 0, 5, 0, 5, 1, 0, 3, 0, 5, 6, 3, 3, 0, 3, 0, 6, 5, 0, 1, 4, 4, 3, 3, 6, 4, 3, 5, 6, 3, 5
5.151361,	1, 1, 7, 4, 0, 2, 1, 5, 3, 2, 4, 4, 6, 1, 1, 4, 5, 5, 6, 3, 0, 0, 3, 2, 1, 7, 2, 4, 2, 4, 1, 1
5.244998,	0, 4, 6, 6, 6, 5, 0, 3, 5, 0, 7, 2, 3, 2, 7, 1, 1, 7, 6, 5, 7, 4, 5, 7, 3, 7, 4, 7, 5, 6, 6, 3
5.245502,	3, 4, 1, 6, 5, 0, 2, 5, 2, 6, 2, 5, 6, 5, 1, 0, 1, 5, 5, 1, 3, 0, 4, 4, 6, 4, 6, 7, 2, 4, 3, 3
5.245673,	0, 2, 4, 3, 3, 6, 7, 5, 3, 6, 2, 0, 2, 7, 2, 7, 7, 4, 1, 5, 4, 1, 3, 5, 6, 0, 1, 1, 4, 6, 7, 3
5.258323,	0, 3, 5, 0, 2, 7, 6, 3, 4, 7, 3, 2, 5, 6, 7, 5, 7, 3, 0, 2, 2, 0, 6, 6, 1, 7, 0, 7, 3, 5, 4, 3
5.323114,	5, 1, 4, 0, 1, 6, 2, 1, 3, 4, 1, 6, 4, 5, 6, 1, 0, 5, 5, 5, 4, 0, 1, 1, 2, 1, 3, 6, 3, 2, 3, 2
5.323502,	2, 5, 4, 0, 0, 2, 2, 5, 3, 3, 2, 7, 4, 7, 1, 0, 7, 0, 3, 5, 0, 1, 1, 4, 7, 3, 0, 7, 7, 6, 4, 3
5.335824,	0, 0, 7, 0, 6, 5, 4, 6, 1, 3, 7, 5, 0, 0, 2, 1, 0, 3, 0, 1, 2, 2, 1, 4, 4, 0, 7, 2, 7, 4, 1, 3
5.335938,	1, 0, 3, 7, 2, 2, 1, 2, 5, 1, 3, 1, 7, 2, 1, 6, 5, 5, 4, 6, 2, 1, 4, 3, 0, 7, 0, 6, 4, 2, 2, 7
5.337064,	7, 0, 2, 4, 0, 6, 4, 1, 4, 6, 7, 7, 0, 4, 3, 1, 2, 7, 2, 5, 4, 3, 3, 3, 4, 3, 1, 2, 6, 0, 3, 6
5.387111,	7, 5, 6, 3, 0, 7, 0, 1, 1, 1, 3, 4, 2, 5, 1, 1, 5, 3, 7, 0, 6, 0, 7, 0, 1, 5, 5, 6, 0, 2, 7, 1
5.413182,	2, 2, 0, 4, 5, 3, 6, 5, 1, 7, 6, 4, 4, 2, 1, 4, 1, 0, 2, 6, 0, 2, 3, 0, 6, 0, 2, 4, 0, 6, 1, 2
5.413182,	3, 6, 5, 6, 7, 0, 4, 5, 1, 7, 7, 4, 7, 6, 6, 6, 5, 4, 1, 1, 1, 5, 1, 3, 4, 7, 7, 2, 6, 5, 0, 1
5.443904,	3, 7, 4, 4, 3, 4, 6, 5, 1, 6, 3, 7, 1, 7, 5, 4, 4, 3, 2, 1, 7, 3, 2, 6, 1, 2, 3, 3, 7, 6, 0, 2
5.444688,	7, 4, 1, 1, 1, 5, 5, 5, 3, 5, 5, 6, 4, 1, 3, 1, 7, 4, 5, 7, 6, 5, 3, 3, 6, 2, 5, 0, 2, 0, 3, 5
5.445086,	6, 0, 2, 3, 4, 6, 1, 5, 1, 0, 0, 1, 2, 2, 7, 1, 3, 0, 0, 1, 4, 7, 4, 3, 6, 3, 0, 0, 4, 0, 0, 6
5.445289,	4, 4, 0, 1, 1, 5, 0, 1, 4, 0, 6, 7, 6, 5, 7, 1, 4, 2, 1, 7, 0, 5, 3, 4, 6, 5, 3, 6, 3, 1, 4, 4
5.445413,	1, 0, 1, 5, 1, 0, 1, 2, 4, 4, 1, 3, 3, 1, 6, 3, 1, 1, 1, 3, 5, 2, 4, 2, 7, 0, 4, 5, 1, 6, 0, 3
5.445602,	4, 6, 3, 4, 4, 6, 4, 3, 1, 0, 0, 2, 0, 5, 3, 1, 1, 7, 5, 1, 6, 6, 1, 4, 3, 7, 2, 6, 0, 2, 2, 7
5.475250,	6, 5, 7, 1, 1, 1, 3, 0, 3, 6, 7, 3, 5, 1, 5, 1, 0, 1, 6, 0, 3, 4, 4, 6, 5, 5, 4, 5, 1, 5, 3, 2
5.476298,	0, 1, 4, 7, 4, 5, 6, 0, 6, 6, 2, 2, 6, 4, 1, 2, 4, 1, 0, 6, 3, 5, 0, 3, 1, 3, 2, 4, 4, 1, 2, 2
5.476307,	3, 6, 1, 2, 1, 1, 7, 2, 0, 1, 5, 2, 6, 5, 4, 3, 0, 5, 6, 2, 2, 7, 1, 0, 4, 6, 6, 3, 2, 2, 2, 7
5.476499,	4, 6, 7, 6, 7, 6, 2, 6, 5, 5, 4, 3, 4, 4, 7, 2, 2, 3, 7, 6, 1, 3, 0, 3, 6, 6, 1, 0, 5, 4, 3, 5
5.538893,	6, 0, 2, 6, 6, 6, 2, 2, 2, 3, 3, 4, 6, 5, 3, 5, 1, 2, 7, 5, 1, 7, 4, 1, 4, 5, 5, 0, 7, 2, 6, 1
5.538983,	7, 4, 7, 5, 3, 1, 6, 3, 2, 2, 0, 1, 5, 2, 1, 7, 1, 2, 4, 2, 5, 1, 4, 1, 7, 2, 5, 7, 2, 3, 6, 5
5.539402,	1, 6, 7, 0, 6, 5, 3, 4, 0, 3, 1, 3, 0, 5, 2, 0, 4, 2, 0, 4, 3, 2, 4, 0, 1, 3, 3, 3, 7, 3, 4, 2
5.544386,	6, 4, 0, 4, 5, 4, 1, 2, 7, 0, 4, 0, 1, 2, 3, 1, 5, 7, 6, 2, 4, 4, 7, 5, 6, 6, 4, 0, 0, 7, 0, 4
5.551119,	0, 6, 5, 3, 7, 1, 3, 0, 4, 7, 6, 7, 4, 2, 5, 7, 6, 7, 6, 5, 0, 7, 7, 2, 4, 7, 3, 3, 6, 5, 0, 4

5.551286,	3, 2, 0, 7, 7, 4, 4, 6, 4, 3, 6, 5, 1, 4, 0, 1, 7, 4, 5, 1, 6, 3, 1, 4, 6, 0, 7, 0, 3, 2, 7, 4

Литература

1. MIL-STD-188-110B, Interoperability and performance standards for data modems, U.S. Army Information System Engineering Command, 2000

ОБРАБОТКА РЕЧЕВЫХ ДАННЫХ НА ОСНОВЕ ИСПОЛЬЗОВАНИЯ ПЕРЕМЕННОЙ ДЛИНЫ СЕГМЕНТА АНАЛИЗА

А.А. Афанасьев

Академия ФСО России

Основной проблемой цифровой обработки речевого сигнала является задача качественного и компактного представления данных. Решение этой проблемы позволит в условиях заданного критерия качества улучшить качество обработки. Широкое распространение в инфо–коммуникациях в настоящее время получили методы обработки речевых сигналов для их передачи с переменной скоростью передачи и асинхронным вводом в канал связи.

Среди многообразия методов обработки речи одним из наиболее эффективных является метод линейного предсказания.[1] Метод линейного предсказания речи принадлежит к классу методов, использующих модель речевого сигнала в виде отклика линейной системы с переменными параметрами (голосового тракта) на соответствующий сигнал возбуждения (порождающий сигнал). Анализатор речепреобразующего устройства выделяет из короткого сегмента речевого сигнала параметры состояния линейной системы и сигнала возбуждения, позволяющие синтезатору восстановить исходный сигнал с требуемой степенью верности.

Сущность метода линейного предсказания заключается в том, что выборка речевого сигнала S(n) может быть приблизительно предсказана линейной комбинацией предшествующих отсчетов этого сигнала (1):

$$S'(n) = \sum_{i=1}^{M} a_i S_{n-i} + e(n), \qquad (1)$$

где $S'(n)$ – предсказанное значение речевого сигнала; a_i – коэффициент линейного предсказания; M – число коэффициентов линейного предсказания или порядок предсказания.

Параметры речевого сигнала изменяется с течением времени медленно, что позволяет рассматривать речевой сигнал как стационарный во временные интер-

валы порядка 2,5-30 мс, называемые окнами. Возможность линейного прогнозирования текущего отсчета объясняется наличием значительных корреляционных связей между отсчетами речевого сигнала при его равномерной дискретизации. В вокодерах, основанных на методе линейного предсказания (липредерах), порядок предсказания, как правило, является постоянным, а значения коэффициентов a_i фиксируются на коротких, примыкающих друг к другу и равных по длительности временных интервалах. При этом исходный речевой сигнал рассматривается как совокупность отрезков стационарных случайных последовательностей, каждый из которых порождается системой с постоянными параметрами. Специальные вычислительные приемы позволяют снизить нежелательное влияние переходных процессов, возникающих при переходе от одного отрезка к другому. Такой подход основан на гипотезе о локально-стационарном характере процесса речеобразования, которая позволяет оценивать изменения параметров или вероятностных характеристик нестационарного речевого сигнала на основе моделей, инвариантных к временному сдвигу.

В настоящее время известны и описаны различные способы ЛП, отличающиеся видом сигнала возбуждения и параметрами, описывающими состояния линейной формирующей системы [2].

Недостатками известных способов при этом является относительно большой объем данных при заданном качестве синтеза речи, а также значительное расходование информационного ресурса на представление параметров, описывающих передаточную функцию голосового тракта, что объясняется тем, что выделение и кодирование этих параметров осуществляют на каждом фиксированном интервале квазистационарности.

В устройствах, реализующих данные методы, осуществляется анализ речи на участке квазистационарности, который по разным оценкам составляет 2,5-30 мс., при этом, при этом выделяется информация о параметрах формирующей модели сигнала возбуждения. На данном участке сигнал принято считать близким к стационарному, вследствие чего он получил название квазистационарный. Выбор

длины данного сегмента является весьма важной задачей. Его увеличение приводит к уменьшению скорости передачи в канале связи, а сокращение к повышению качественных характеристик синтезируемого сигнала, так как сигнал становится близким к стационарному, и, уменьшению времени задержки сигнала на обработку. Анализ речевого сигнала на фиксировано выбранном сегменте квазистационарности является достаточно грубым допущением, так как за пределами сегмента сигнал представляется равным нулю, что не соответствует действительности и приводит к появлению искажений на стыках сегментов при их анализе и кодировании, а также искажений в восприятии синтезированной речи, при этом на вокализованных участках речи длина сегмента стационарности может быть увеличена, что связано с линейным характером образования на этом участке, а на шумоподобных участках желательно ее уменьшать, так как речевой сигнал в данном случае имеет нестационарные свойства.

Сущность предлагаемого подхода заключается в следующем. Выполняется анализ поступающего речевого сигнала на основе линейного предсказания, деля его на сегменты по 20 миллисекунд, если принимается решение о том, что сигнал является активной речью, то выделяется переход огибающей сигнала через нулевое значение и от положения отсчета со значением наиболее близким нулю выбирается длина сегмента соответствующая 20 миллисекундам и рассчитывается значение частоты основного тона и сигнала тон-шум, если принимается решение о вокализованности анализируемого сигнала, то увеличивается длительность сегмента квазистационарности на количество отсчетов кратное периоду основного тона, но не более чем на 60 миллисекунд с обязательной проверкой на вокализованность следующих сегментов по 20 миллисекунд. Если принимается решение о шумоподобности следующего сегмента, то граница сегмента анализа выбирается кратной количеству отсчетов на периоде основного тона, но не более половины следующего сегмента длительностью 20 миллисекунд. Если принимается решение о шумоподобности анализируемого сегмента, то длину сегмента анализа уменьшается, при этом границу сегмента формируется на значении близком нулю и кратном вычисленному периоду основного тона.

При таком подходе с высокой вероятностью можно утверждать, что начальный и конечный отсчеты во вновь сформированном сегменте будут иметь значения близкие нулю, что значительно уменьшит возможные искажения на стыках сегментов. Использование данного способа для выделения сегментов речи при обработке будет рационально для класса систем кодирования речевых сигналов, так как позволит уменьшить среднюю скорость. Такой подход позволит значительно снизить объем обрабатываемых речевых данных, при этом качественные показатели синтезированного речевого сигнала значительно повышаются.

К достоинствам использования предлагаемого способа следует отнести тот факт, что изменение длительности сегментов анализа при обработке дает возможность уменьшить объем обрабатываемых данных.

Литература

1. Маркел, Дж. Д. Линейное предсказание речи: пер. с англ. / А. Х. Грей. под ред. Ю. Н. Прохорова и В. С. Звездина. М.: Связь, 1980. 308 с.ил.
2. Шелухин, О. И. Цифровая обработка и передача речи / Под ред. Шелухина О.И. – М.: Радио и связь, 2000. – 456 с.: ил.

ВЫБОР ФУНКЦИИ ПОТЕРЬ ПРИ СИНТЕЗЕ СИСТЕМ МОДУЛЯЦИИ – ДЕМОДУЛЯЦИИ СИГНАЛОВ

К.А. Батенков, Д.А. Рыболовлев

Академия ФСО России

Количественной мерой эффекта от принятия того или иного решения, в том числе и при синтезе систем модуляции-демодуляции сигналов, является функция потерь $g(\mathbf{X}', \mathbf{X})$, причем совершенно естественно, что ее минимальное значение приводит к наиболее благоприятным последствиям. Данная характеристика по сути представляет собой априорную оценку последствий принятия тех или иных решений $\mathbf{X}'$ при наблюдении данных $\mathbf{Y}$ при условии, что непосредственный доступ к слежению за передаваемыми сообщениями $\mathbf{X}$ отсутствует. В тоже время необходимо подчеркнуть, что в общем случае функция потерь $g(\mathbf{X}', \mathbf{X})$ зависима и от наблюдаемых данных $\mathbf{Y}$, поскольку существуют задачи, для которых принципиально важным моментом оказывается то, какая именно из возможных реализаций была идентифицирована, а следовательно для различных исходов эффекты от принятия решений являются различными.

Условно используемые функции потерь подразделяются на две группы: зависящие от конкретных значений передаваемых сообщений $\mathbf{X}$ и решений $\mathbf{X}'$ – $g(\mathbf{X}', \mathbf{X})$ и зависящие только от их разности – $g(\mathbf{X}'–\mathbf{X})$. Во втором случае (простая функция потерь) некоторая ограниченность общей постановки задачи приводит к возможности получения аналитических результатов для правил оценки, однако первый позволяет более точно задать исходные данные, а следовательно синтезировать систему, более полно соответствующую условиям своего функционирования.

В качестве наиболее употребимых простых функций потерь можно отметить: квадратическую, абсолютную и равномерную. Так, в случае использования пространств со счетным числом измерений они могут быть представлены в соответствующем виде:

$$g_q(\mathbf{X'}-\mathbf{X}) = (\mathbf{X'}-\mathbf{X})^T(\mathbf{X'}-\mathbf{X}), \quad (1)$$

$$g_a(\mathbf{X'}-\mathbf{X}) = \sum|\mathbf{X'}-\mathbf{X}|, \quad (2)$$

$$g_u(\mathbf{X'}-\mathbf{X}) = \begin{cases} 0, & \sum|\mathbf{X'}-\mathbf{X}| \le \Delta, \\ 1, & \sum|\mathbf{X'}-\mathbf{X}| > \Delta. \end{cases} \quad (3)$$

где ^{T} – оператор транспонирования;

Δ – некоторый порог.

В ситуации, когда природа исследуемых пространств характеризуется несчетным числом измерений формулы для простых функций потерь должны быть изменены в соответствии с требованиями, накладываемыми свойствами существа используемых данных. Однако данное предположение не меняет коренным образом общую постановку задачи и при необходимости может быть уточнено.

В то же время необходимо отметить, что использование простой функции потерь в качестве меры эффекта принятия решений для систем передачи информации оказывается не совсем точным. Причиной этого факта выступает предназначение подобных систем, сводящееся к необходимости доставки к получателю не просто достоверной, но и своевременной информации. Т.е. при принятии решения важна не просто раздельная оптимальность степени точности воспроизводимых сообщений и их количества, а некоторый интегральный показатель, характеризующий обе грани процесса передачи. Так, если принятие решения осуществляется в соответствии с правилом, гарантирующим минимальную в некотором смысле погрешность воспроизведения переданных данных, то, поскольку решения оказываются все же неточны вследствие стохастической природы передаваемых сигналов и помех, для достоверности после принятия решения необходима дополнительная информация, позволяющая точно указать на сколько принятая информация искажена. То есть по сути необходимо затратить дополнительный ресурс, например временной, для передачи этой информации, причем гарантировать, что требуемый ресурс окажется минимальным не возможно, поскольку вынесение решения базируется на принципе минимальных искажений. Следовательно может оказаться, что требование к своевременности передачи не выпол-

няется. С другой стороны уменьшение объема передаваемых данных с целью повышения достоверности является не всегда возможной альтернативой, поскольку при этом также может не выполняться требование к своевременности доставки сообщений. Таким образом, принятие решение должно приводить не просто к похожести в некотором смысле переданной информации и воспроизведенной, а к сведению к минимуму неопределенности (недостоверности) о сообщении источника $\mathbf{X}$ при наблюдении $\mathbf{Y}$. Именно подобная мера эффекта принятия решений является наиболее целесообразной, поскольку позволяет учесть как своевременность, так и достоверность передаваемого сообщения в целом.

Согласно [**Ошибка! Источник ссылки не найден.**] неопределенность о некоторых параметрах источника $\mathbf{X}$ при вынесении решения $\mathbf{X'}$ при заданном наблюдении $\mathbf{Y}$ трактуется как условная собственная информация, содержащаяся в конкретном событии $\mathbf{X}$ при условии принятия некоторого $\mathbf{X'}$. Следовательно функция потерь имеет вид:

$$g_i(\mathbf{X'},\mathbf{X}) = -\log_2 \omega_{\mathbf{X}/\mathbf{X'}}(\mathbf{X},\mathbf{X'}), \qquad (4)$$

где $\omega_{\mathbf{X}/\mathbf{X'}}(\mathbf{X},\mathbf{X'})$ – апостериорная плотность вероятности параметров источника $\mathbf{X}$ после принятия решения $\mathbf{X'}$, вычисляемая согласно формуле Байеса:

$$\omega_{\mathbf{X}/\mathbf{X'}} = \frac{\omega_{\mathbf{X'}/\mathbf{X}}\omega_{\mathbf{X}}}{\omega_{\mathbf{X'}}}. \qquad (5)$$

Таким образом, она определяется не только передаваемым параметром источника $\mathbf{X}$ и вынесенным решением $\mathbf{X'}$, но и правилами их принятия $\omega_{\mathbf{X'}/\mathbf{Y}}$, а следовательно зависит от наблюдаемых данных $\mathbf{Y}$ [Маркел, Дж. Д. Линейное предсказание речи: пер. с англ. / А. Х. Грей. под ред. Ю. Н. Прохорова и В. С. Звездина. М.: Связь, 1980. 308 с.ил.

Шелухин, О. И. Цифровая обработка и передача речи / Под ред. Шелухина О.И. – М.: Радио и связь, 2000. – 456 с.: ил.].

Литература

1. Галлагер Р. Теория информации и надежная связь. Пер. с англ., под ред. М.С. Пинскера и Б.С. Цыбакова. – М.: Советское радио, 1974. – 720 с.
2. Хворостенко Н. П. Статистическая теория демодуляции дискретных сигналов.– М.: Издательство Связь, 1968. – 336 с.

АДАПТАЦИЯ ПАРАЛЛЕЛЬНОГО АЛГОРИТМА ФРАКТАЛЬНОГО СЖАТИЯ ЦИФРОВЫХ ИЗОБРАЖЕНИЙ ДЛЯ МНОГОЯДЕРНЫХ ПРОЦЕССОРОВ

И.В. Бойченко, С.С. Кулбаев

Томский государственный университет систем управления и радиоэлектроники, Факультет систем управления

В данной работе предлагается способ организации параллельного вычислительного процесса сжатия цифровых изображений на основе фракталов. Предложено использовать диспетчер вычислительных потоков, распределяющий нагрузку между процессорными ядрами высокопроизводительной вычислительной системы (ВПВС).

Введение

Для эффективного использования ресурсов современных вычислительных систем с SMP-архитектурой (Symmetric Multiprocessing) актуальной является задача разработки приложений, поддерживающих параллелизм, присущий решаемой задаче. К такого рода приложениям можно отнести сжатие (компрессию) изображений на основе фракталов [1].

Компрессия изображений является ключевой технологией при хранении и передаче изображений. Существуют методы сжатия изображений, обеспечивающие высокий коэффициент сжатия, но при этом требующие больших вычислительных затрат. Адаптация параллельного алгоритма обработки изображений на высокопроизводительных вычислительных системах (кластерах) или на специализированных вычислительных акселераторах позволяет реализовать методы с большим коэффициентом сжатия цифровых изображений. При этом процесс декомпрессии является существенно менее трудоёмким.

Фрактальные алгоритмы обеспечивают удачное соотношение между коэффициентом сжатия и качеством, и, обладают уникальным свойством

детализации при произвольном масштабировании [1]. Однако большинство известных алгоритмов фрактального сжатия имеют высокую трудоемкость и требуют много времени для обработки данных [2]. Фрактальное сжатие поддается распараллеливанию, что позволяет создавать программное обеспечение для высокопроизводительных вычислительных систем (ВПВС) с параллельной архитектурой.

Многопоточная обработка (multithread processing)

Так как область входных данных фрактального алгоритма представляет собой не пересекающихся ранговых блоков, то на каждом из вычислительных блоков (ядер) рабочей станции можно обрабатывать соответствующий ранговый блок независимо от остальных [3]. Таким образом, происходит разделение области входных данных на части, и для каждой части вычисления можно производить независимо и одновременно.

Например, необходимо сжать некоторое изображение, и имеется n процессоров (ядер), тогда можно разбить все изображение на n частей (рангов) и на каждом из вычислительных блоков одновременно обрабатывать соответствующую часть, и результаты обработки заносятся в выходной буфер. По окончании обработки буфер сжимается по алгоритму Хаффмана [4] и записывается в файл.

Одним из преимуществ многопоточной обработки является возможность в многопроцессорных системах одновременно выполнять несколько потоков на разных процессорах, увеличивая производительность [5].

Диспетчер исполняемых потоков для высокопроизводительных вычислений

Адаптация алгоритма на многоядерных процессорах происходит за счет диспетчера исполняемых потов. Диспетчер исполняемых потоков в многопоточной обработке изображений с помощью фрактального алгоритма работает следующим образом:

1) Исходное изображение разбивается на ранговые блоки и на группы доменных блоков.

2) Диспетчер исполняемых потоков определяет, сколько вычислительных ядер имеет данная вычислительная система и создает потоки, равные числу ядер в данной системе.

3) Диспетчер потоков создает глобальные и локальные очереди. В глобальные очереди помещается исходные ранговые блоки. И, каждый поток обрабатывает по ранговому блоку. Потоки работают независимо друг от друга, сравнивая выбранный ранговый блок с каждым доменным блоком. В случае несоответствия рангового блока с доменными блоками, данный поток разбивает выбранный ранговый блок на меньшие блоки и помещает их в локальные очереди.

4) Когда поток готов/свободен для работы, в первую очередь он обращается к локальной очереди. Доступ к локальным очередям осуществляется в порядке «последним поступил — первым обслужен» (LIFO). Если локальная очередь пуста, то поток обращается к глобальным очередям. И цикл повторяется до тех пор, пока не будет достигнут минимальный размер блока (рис. 1).

Чем меньше размер блока, тем меньше потери качества изображения. При его достижении, поток записывает в локальный буфер не сами данные рангового блока, а структуру данного рангового блока. В состав структуры рангового блока входят: размер рангового блока; координаты по оси x, y; форма аффинного преобразования [2]; три цветовые составляющие (YUV).

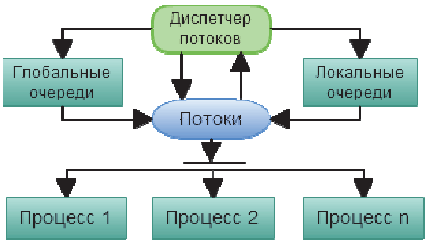


Рисунок 1. Схема работы диспетчера исполняемых потоков параллельного алгоритма фрактального сжатия изображения.

Оценка сложности вычислений фрактального алгоритма

Предположим, что *n*x*m* - размер исходного изображения в пикселях, *rsz*– сторона рангового блока. Тогда количество ранговых блоков в изображении будет

$$r_{size} = \frac{n}{rsz} \cdot \frac{m}{rsz} \quad (1)$$

а доменных (учитывая, что сторона доменного блока всегда вдвое больше стороны рангового)

$$d_{size} = \frac{(n - 2 \cdot rsz)}{2} \cdot \frac{(m - 2 \cdot rsz)}{2} \quad (2)$$

Формулы (1) и (2) подробно описаны в [2]. Таким образом, необходимо произвести $r_{size} \times d_{size}$ поблочных сопоставлений. В [2] также приводится оценка затрачиваемых операций умножения, сложения и вычитания. Эта оценка составляет три операции умножения, три операции сложения и одну операцию вычитания на один пиксель изображения. Например, для изображения размером 512x512 и стороной рангового блока 8 пикселей составляет 251 920 384 сравнений, то есть порядок сложности 10^9. Так же в данном алгоритме в каждом таком сопоставлении необходимо организовать цикл по всем пикселям блока (в данном примере в блоке пикселей 64) для расчета математического ожидания и дисперсии. Время, затрачиваемое на сжатие изображения в градациях серого в простейшем случае, утраивается, если речь идет о цветном изображений (так как для представления информации о яркости и цвете пикселя требуется как минимум три составляющих). Таким образом, общая сложность фрактального сжатия изображения размером 512x512 составляет приблизительно $10^{10} – 10^{11}$ операций.

Результаты эксперимента

Результаты работы приведены в таблице 1. Для тестов использованы цветные (TrueColor) изображения 24 бит на пиксель группы фрактального кодирования и анализа (fractal coding and analysis group) [6]. Используемые характеристики включают в себя время работы алгоритма сжатия, алгоритма декодирования, размер сжатого файла, качество по метрикам SSIM и PSNR. В

состав параметров фрактального алгоритма входят (в скобках указано конкретное значение):

- размер рангового блока (8);
- шаг поиска домена (4);
- погрешность (среднеквадратичное отклонение 0.01).

Для вычислений был использован четырехъядерный процессор Intel Core i7, поддерживающий до 8-ми вычислительных потоков на четырех физических ядрах с тактовой частотой 2.8 ГГц.

Таблица 1

Результаты работы фрактального алгоритма и алгоритма JPEG на четырехядерном процессоре Intel Core i7

Изображение	Фрактальный алгоритм					Алгоритм JPEG				
	Время сжатия, с	Время Декодера, с	Размер файла, Кбайт	SSIM	PSNR, дБ	Время сжатия, с (Image Magick)	Время Декодера, с (Image Magick)	Размер файла, Кбайт	SSIM	PSNR, дБ
lena3	0.32	0.018	52.5	1.00	46.6	0.042	0.032	74.6	0.93	47.7
monarch	0.48	0.015	81.0	0.98	43.5	0.056	0.042	103.3	0.99	44.2
peppers3	0.32	0.017	52.5	0.99	39.1	0.055	0.034	72.7	0.94	40.4

Так же была проведена серия экспериментов на увеличение производительности вычислительных систем в зависимости от количества используемых ядер процессора (рис. 2). Частота одного ядра 2,8 ГГц.

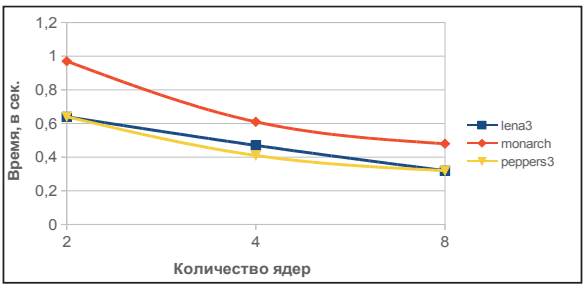


Рисунок 2. Время выполнения на много-ядерных вычислитель-ных системах.

Можно считать, что время вычислений зависит линейно от количества вычислительных ядер.

Заключение

В работе реализован параллельный алгоритм сжатия цифровых изображений на основе фракталов с диспетчером исполняемых потоков для высокопроизводительных вычислений. Использование диспетчера потоков улучшает механизмы эффективного поиска доменных блоков для каждого рангового блока и адаптирует параллельный алгоритм на многоядерных процессора. Причем при наличии незанятого потока очередная порождаемая подзадача поиска и сравнения доменных блоков будет исполняться на свободном потоке. В целом, такая организация вычислительного процесса обеспечивает более равномерную загрузку процессоров (ядер) вычислительной системы при сжатии изображений на основе фракталов.

Литература

1. Уэлстид С.Т. Фракталы и вейвлеты для сжатия изображений в действии: учеб. пособие / С.Т. Уэлстид. – М.: Триумф, 2003. – 320 с.
2. Fisher Y. Fractal Image Compression – Theory and Application. – N.Y.: Springer Verlag, 1994. – 341 p.
3. Кулбаев С.С., Бойченко И.В. Реализация алгоритмов фрактального сжатия изображений на многоядерных векторных процессорах. Сборник статей региональной научно-практической конференции «Многоядерные процессоры и параллельное программирование» 25 февраля 2011г. г. Барнаул - С. 108-112.
4. Алгоритм Хаффмана [Электронный ресурс] /Алгоритмы и методы. 2012 Режим доступа: http://algolist.manual.ru/compress/standard/huffman.php, свободный.
5. Гергель В.П. Основы параллельных вычислений для многопроцессорных вычислительных систем / В.П. Гергель, Р.Г. Стронгин // Н.Новгород, ННГУ — 2003. - 2 изд.

6. Test image repository [Электронный ресурс] / Fractal coding and analisys group. 2011. Режим доступа:: http://links.uwaterloo.ca/Repository.html, свояодный (дата обращения 17.02.2012).

ОЦЕНКА ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ АЛГОРИТМОВ ОБРАБОТКИ РЕЧЕВОГО СИГНАЛА, ОСНОВАННЫХ НА МОДЕЛИ СЛУХА

Р.В. Мещеряков, С.Д. Тиунов

Томский государственный университет систем управления и радиоэлектроники, Кафедра комплексной информационной безопасности электронно-вычислительных систем

Введение

В приложениях автоматической обработки речевого сигнала могут успешно применяться методы, основанные на моделировании слуховой системы человека [1-4]. В свою очередь в приложениях речевых технологий к обучению иностранным языкам существует тенденция к развитию игровых систем, способных динамично взаимодействовать с пользователем [5]. Для того, чтобы применить указанные методы в таких приложениях необходимы реализации этих методов в реальном времени. В настоящей работе будет произведена оценка вычислительной сложности нескольких вариантов алгоритмов из модели слуха и сделаны выводы об аппаратном обеспечении, необходимом для работы обучающей системы в реальном времени.

Описание оцениваемых алгоритмов

Наиболее трудоемкие для вычисления алгоритмы из модели слуха представлены ниже:

- выделение спектрального представления речевого сигнала (с помощью вейвлет-преобразования);
- выделение матрицы последовательной маскировки;
- выделение матрицы одновременной маскировки.

Первые два алгоритма могут быть реализованы в виде K цифровых фильтров, а последний алгоритм при определенных условиях – в виде одного цифрового фильтра [6]. Для реализации цифровой фильтрации потока данных использу-

ются методы пересечения с накоплением и пересечения со сложением [7]. В первом случае используемая функция циклической свертки вычисляет готовый блок данных, который помещается непосредственно в выходной поток, а во втором случае блок данных, вычисляемый с помощью линейной свертки, прибавляется к выходному потоку с некоторым смещением. В этой связи предпочтение было отдано методу пересечения с накоплением.

Таким образом, единицей оптимизации является операция циклической свертки. Данная операция является бинарной и проводится над двумя векторами. Вектора интерпретируются как полиномы, и операция циклической свертки является их умножением по модулю третьего полинома. При этом линейная свертка является их умножением и может быть сведена к циклической путем приведения по модулю третьего полинома.

Поскольку в сигнале используются вычисления с вещественными числами, то в качестве примерной меры вычислительной сложности используем количество вещественных операций в секунду (FLOPS).

В методе пересечения с накоплением результат циклической свертки отправляется в выходной поток не полностью, а без части размером в окно фильтрации (поэтому размер векторов для цифровой фильтрации должен быть значительно больше, чем окно фильтрации). Таким образом, вычислительная сложность алгоритма цифровой фильтрации будет рассчитана по следующей формуле:

$$N_{a\lg} = K \cdot N_{conv} \cdot F_q \big/ \left(M_{conv} - W\right) \qquad (1)$$

где N_{alg} – вычислительная сложность алгоритма цифровой фильтрации (в FLOPS);

K – количество цифровых фильтров;

N_{conv} – вычислительная сложность алгоритма циклической свертки (количество вещественных операций – FLOP);

F_q – частота дискретизации входного сигнала;

M_{conv} – размер циклической свертки;

W – размер окна цифровой фильтрации.

В формуле (1) множитель $F_q/(M_{conv}-W)$ представляет собой среднее количество циклических сверток размера M_{conv}, которые необходимо вычислить для цифровой фильтрации 1 секунды входного сигнала.

Оценка вычислительной сложности циклической свертки

Циклическая свертка большого размера может быть вычислена различными способами:

«наивный» способ – прямое вычисление суммы произведений, как записано в определении свертки;

алгоритм, основанный на теореме о свертке с применением дискретного преобразования Фурье;

итерационный алгоритм, основанный на использовании алгоритмов свертки малого размера.

Во всех случаях основными элементарными операциями являются вещественное сложение и вещественное умножение. Поскольку в различных ЭВМ длительность данных операций может различаться сильно или практически не различаться, то расчет количества операций сложений и умножений будет вестись отдельно.

Для вычисления циклической свертки размера M наивным способом необходимо M^2 умножений и $(M-1)$ сложение.

Для вычисления циклической свертки размера M по теореме о свертке необходимо сделать два прямых ДПФ размера M, M комплексных умножений и одно обратное ДПФ размера M. Для вычисления ДПФ удобно принять $M = 2^m$, где m – натуральное число. В этом случае для вычисления ДПФ можно использовать алгоритм Кули-Тьюки по основанию 2, который требует $2Mm$ вещественных умножений и $3Mm$ вещественных сложений. Каждое комплексное умножение можно представить в виде четырех вещественных умножений и двух вещественных сложений. Таким образом, для вычисления циклической свертки размера M по теореме о свертке и ДПФ необходимо $2Mm+4M$ вещественных умножений и $3Mm+2M$ вещественных сложений.

Итерационный метод позволяет вычислять линейную свертку большого размера, используя алгоритмы для вычисления малых линейных сверток. (После этого линейную свертку можно свести к циклической за M сложений). В данном случае можно использовать алгоритм Тоома-Кука для вычисления 4-точечной линейной свертки, который в реализации [8] включает в себя 8 умножений и 26 сложений на этапе предсложений, 7 умножений на этапе умножений и 10 умножений и 18 сложений на этапе постсложений. При итерационной реализации данного алгоритма по основанию $M=4^m$ требуется N_m умножений и N_a сложений, где:

$$N_m = 18\,(7^m - 4^m) / (7 - 4) + 7^m = 6\,(7^m - 4^m) + 7^m \quad (2)$$

$$N_a = (26+18) * (7^m - 4^m) / (7 - 4) = 44 * (7^m - 4^m) / 3 \quad (3)$$

Вычислительная сложность указанных выше алгоритмов циклической свертки для векторов различного размера M приведена в таблице 1 (в первых столбцах – количество умножений, во вторых – количество сложений, в третьих – общее количество вещественных операций).

Таблица 1

M	Наивный метод			Итерационный			БПФ		
16	256	15	271	247	484	731	448	608	1056
64	4096	63	4159	2017	4092	6109	2560	3584	6144
256	65536	255	65791	15271	31460	46731	13312	18944	32256
1024	1048576	1023	1049599	111505	231484	342989	65536	94208	159744
2048	4194304	2047	4196351	298982	622147	921129	143360	206848	350208
4096	1.7×10^7	4095	1.7×10^7	798967	1665444	2464411	311296	450560	761856
8192	6.7×10^7	8191	6.7×10^7	2129738	4445144	6574882	671744	974848	1646592
16384	2.7×10^8	16383	2.7×10^8	5666497	1.2×10^7	1.8×10^7	1441792	2097152	3538944
32768	1.1×10^9	32767	1.1×10^9	1.5×10^7	3.1×10^7	4.7×10^7	3080192	4489216	7569408

Как видно из таблицы 1, для малых длин векторов лучшим является наивный метод, при увеличении длин векторов увеличивается превосходство двух других методов (их взаимное расположение зависит от соотношения длительности умножения и сложения). Наконец, для больших длин векторов наилучшим из рассмотренных методов является основанный на БПФ.

Оценка вычислительной сложности цифровой фильтрации

Расчет вычислительной сложности цифровой фильтрации будет осуществляться для алгоритма вычисления спектра сигнала при следующих исходных данных: $K = 256$; $F_q = 8000, 12000, 16000$; $W = 424, 634, 846$ (W зависит от F_q).

Как было указано ранее, размер циклической свертки должен быть не менее размера окна фильтрации (W). В данном случае наименьший подходящий размер свертки равен 1024.

В таблице 2 приведен расчет вычислительной сложности цифровой фильтрации для различных размеров свертки и различных исходных данных.

Таблица 2

M	БПФ FLOP	FLOPS		
1024	159744	545259520	1258291200	3675906876
2048	350208	441641616	760847932	1193387661
4096	761856	424913150	676031667	960172977
8192	1646592	434116943	669268408	918110650
16384	3538944	454120132	690262601	932907364
32768	7569408	479289747	723632955	971251650
		$F_q = 8000$ Гц	$F_q = 12000$ Гц	$F_q = 16000$ Гц

Из таблицы 2 видно, что для частоты дискретизации сигнала 8 кГц оптимальной длиной циклической свертки является 4096, поскольку требуемое количество операций в секунду наименьшее. Аналогично для частот дискретизации 12 кГц и 16 кГц оптимальная длина циклической свертки равна 8192. Наличие таких минимумов, по-видимому, связано с различным характером роста размера свертки M и вычислительной сложности циклической свертки размера M, поскольку остальные составляющие в формуле (1) были зафиксированы.

Оценка требуемого аппаратного обеспечения

Поскольку целевое приложение предназначено для использования в обучающих классах или для личного обучения, то в качестве возможного аппаратного обеспечения следует рассматривать центральные процессоры (CPU) и графические процессоры (GPU).

Как видно из таблицы 2, в зависимости от частоты дискретизации, для вычисления спектра сигнала необходимо приблизительно от 0,5 до 1 GFLOPS. Значения вычислительной мощности, указанные для большинства современных CPU, несколько превосходят данный порог. В частности наиболее слабый из поддерживаемых Intel процессоров – Intel Pentium M 723, предназначенный для использования в ноутбуках – обеспечивает 1,5 GFLOPS [9]. Более современный мобильный процессор Intel Core 2 Duo L7400 обеспечивает 12 GFLOPS.

При этом следует учитывать, что данные значения указаны для пиковой производительности процессоров, что достижимо при использовании новейших технологий, интегрированных в них, для простейшего алгоритма и при минимальной фоновой нагрузке, например, в однозадачном режиме. Разумеется, что идеальные условия, как правило, не будут выполнены. Использование современных технологий процессоров, однако, можно обеспечить за счет использования специализированных библиотек для вычисления БПФ, содержащих множество реализаций для различных аппаратных платформ, например, библиотеки FFTW [10].

Также следует учитывать, что помимо вычисления спектра сигнала, необходимо произвести и другие операции. В частности, указанные выше моделирование одновременной и последовательной маскировки сравнимы по вычислительной сложности.

Таким образом, для работы такой системы в реальном времени необходимо значительное превосходство вычислительной мощности процессора над минимально необходимой. Такие условия выполняются современными процессорами для рабочих станций (30-150 GFLOPS) и графическими процессорами (500-1500 GFLOPS) [11].

Заключение

В статье оценивается возможности реализации потоковой обработки речи с помощью алгоритмов из модели слуха в реальном времени. Оценена вычислительная сложность циклической свертки для различных размеров свертки. Пока-

зано, что существует оптимальный размер свертки, который зависит от частоты дискретизации входного сигнала. Показано, что выполнение рассмотренных алгоритмов в реальном времени может быть реализовано на современных центральных и графических процессорных устройствах.

Литература

1. Bondarenko V.P., Moor V.N., Chabanets A.N. The analysis of speech perception mechanisms on the models of auditory system // Proceedings Xith ICPhS The eleventh international congress of phonetic sciences. Volume 2. 1987.
2. Bondarenko V., Kotsubinski V., Ponomarev A., Velikotski D. Speech signal analysis wavelet-transformation and signal processing at the periphery of acoustical system // SPECOM-2004, p.p. 181-185.
3. Бондаренко В.П., Пономарев А.А., Рогозинская Е.А. Модель одновременной маскировки // Интеллектуальные системы в управлении, конструировании и образовании – Томск: STT, 2004. - 216 с. – с. 167-174.
4. Конев А.А. Программный комплекс для исследования речи / А.А. Конев, Е.Ю. Костюченко, А.А. Пономарев // Сборник трудов XVII сессии Российского акустического общества. – Т.3. – М. : ГЕОС, 2006. – С. 23–27.
5. Eskenazi, M. An overview of spoken language technology for education / M. Eskenazi // Speech Communication. – 2009. – Vol. 51, – 10. – P. 832–844.
6. Мещеряков, Р. В. Оптимизация времени цифровой фильтрации в исследованиях, связанных с обработкой речи / Р. В. Мещеряков, С. Д. Тиунов // Ежегодная международная открытая научная конференция «Современные проблемы информатизации». – 2010.
7. Блейхут Р. Быстрые алгоритмы цифровой обработки сигналов: Пер. с англ. – М.: Мир, 1989. – 448 с.
8. Bodrato, M. What about toom-cook matrices optimality? – 2006. http://bodrato.it/papers/WhatAboutToomCookMatricesOptimality.pdf
9. Intel® Processors : Intel® microprocessor export compliance metrics // Intel Corporation. – 2011. http://www.intel.com/support/processors/sb/CS-017346.htm

10. Frigo, M. The design and implementation of FFTW3 / M. Frigo, S. G. Johnson // Proceedings of the IEEE. – 2005. – Vol. 93, – 2. – P. 216–231.
11. Grafikkarten Prozessoren // Hardware-INFOS. – 2012. http://www.hardware-infos.com/grafikkarten_nvidia.php

ИССЛЕДОВАНИЕ СТОХАСТИЧЕСКОГО ГРАДИЕНТНОГО МЕТОДА В АЛГОРИТМАХ БЫСТРОГО ПОИСКА В СИСТЕМАХ С ПСЕВДОСЛУЧАЙНОЙ ПЕРЕСТРОЙКОЙ РАБОЧЕЙ ЧАСТОТЫ

А.Ю. Колотков, Б.В.Султанов

ФГБОУ ВПО «Пензенский государственный университет»

Изначально методы расширенного спектра (spread spectrum - SS) применялись при разработке военных систем управления и связи. К концу второй мировой войны в радиолокации расширение спектра уже применялось для борьбы с преднамеренными помехами [1], а в последующие годы развитие данной технологии объяснялось желанием создать помехоустойчивые системы связи. Методы расширенного спектра получили своё название благодаря тому, что полоса, используемая для передачи сигнала, намного шире минимальной, необходимой для передачи данных. Система связи называется системой с расширенным спектром в следующих случаях:

- используемая полоса значительно шире минимальной, необходимой для передачи данных;
- расширение спектра производится с помощью так называемого расширяющего (или кодового) сигнала, который не зависит от передаваемой информации;
- восстановление исходных данных приёмником («сужение спектра») осуществляется путём сопоставления полученного сигнала и синхронизированной копии расширяющего сигнала.

Выделяют следующие основные преимущества систем связи расширенного спектра [1]: хорошая временная разрешающая способность, возможность построения системы множественного доступа, подавление помех, снижение плотности энергии.

Основными методами расширения спектра являются – метод прямой последовательности (direct sequencing – DS) и метод псевдослучайной перестройки рабочей частоты ППРЧ (frequency hopping –FH).

В данной работе объектом исследования является метод скачкообразной перестройки частоты, точнее его схема с использованием М-арной частотной манипуляции (M-ary frequency shift keying – MFSK). Этот метод позволяет для перестройки частоты использовать полосы шириной порядка несколько гигагерц, что намного превышает аналогичные показатели систем DS [2]. При этой модуляции $k = \log_2 M$ информационных бит используются для определения одной из M передаваемых частот. Положение М-арного множества сигналов скачкообразно изменяется синтезатором частот (управляется псевдослучайной цифровой последовательностью PN) на псевдослучайную величину, принадлежащую полосе W_{ss}. Т.е. систему FH/MFSK можно рассматривать как двухэтапный процесс модуляции – модуляции информации и модуляции с перестройкой частоты, в результате чего передаётся один тон. Следует отметить, что для повышения устойчивости системы и её свойства "необнаруживаемости передачи" можно и нужно использовать дополнительное разнесение сигнала по частотам в пределах передачи одной информационной посылки (fast frequency hopping – FFH).

Таким образом, именно FFH/MFSK модификация метода ППРЧ и исследуется в данной работе. Одной из основных задач при построении подобных систем является задача синхронизации PN управляющей последовательности между «передатчиком» и «приёмником». Факторы, обуславливающие необходимость осуществления синхронизации [1]:

- «приёмнику» не известен момент выхода «передатчика» на связь и соответственно не известно состояние PN управляющей последовательности «передатчика» в этот момент;
- из-за различных эффектов в канале искажается фаза передаваемого сигнала, что мешает «приёмнику» точно определять момент перестройки частоты, а также появляются временные задержки.

В соответствии с вышесказанным процесс синхронизации делится на два этапа: этап первоначальной синхронизации (синхронизация PN последовательностей «передатчика» и «приёмника»); этап точной синхронизации (этап сопровож-

дения – окончательная частотная и временная синхронизация, выбирается сигнал наиболее точно соответствующий полученному).

Основным требованием к подобным системам синхронизации является быстрота вхождения «передатчика» и «приёмника» в синхронизм. Наиболее требовательным по временным затратам, сложности реализации, а также наиболее важным является первый этап синхронизации – этап первоначальной синхронизации PN управляющих последовательностей «передатчика» и «приёмника».

Наименьшее время вхождения в синхронизм при реализации первоначальной синхронизации обеспечивает подход, основанный на извлечении в течение n_1 последовательных чипов входного сигнала по одному старшему биту двоичного кода частоты чипа и последовательной загрузке этих бит в линейный регистр сдвига с обратной связью длиной n_1, на основе которого реализован PN генератор «приёмника» [3]. Эта идея основывается на то том факте, что одно состояние PN генератора однозначно определят частоту текущего и всех последующих чипов. Основная задача при таком подходе – это измерение частоты на каждом из n_1 последовательных чипов. Измерение частоты непростая задача и точно определить частоту невозможно. Если бы это было реализуемо, то можно было бы загрузить весь линейный регистр сдвига PN генератора «приёмника» всего за один чип. Практически же для загрузки в линейный регистр сдвига на каждом чипе используется лишь старший бит двоичного кода частоты (MSB – most significant bit), точность определения которого можно обеспечить. Один из наиболее подробно описанных подходов к измерению частоты – это метод авторегрессионной спектральной оценки АСО (autoregressive spectral estimation acquisition technique – ASEAT).

В данной работе исследуется следующий подход к реализации ASEAT [4]. В течение одного скачка FFH/MFSK отсчёты входного сигнала поступают на вход линии задержки адаптивного трансверсального фильтра, коэффициенты которого подстраиваются стохастическим градиентным методом. Затем на основе полученных коэффициентов строится спектральная оценка сигнала и принимается решение о значении мгновенной частоты полученного сигнала.

Структурная схема фильтра представлена на рисунке 1.

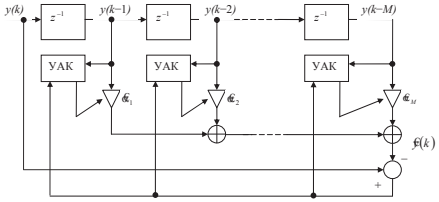


Рисунок 1. Структурная схема адаптивного трансверсального фильтра.

В узлах адаптивной коррекции УАК на каждом k-м шаге дискретизации формируются приращения $\Delta_i(k)$ коэффициентов $\alpha_m(k), m=\overline{1\ldots M}$ в соответствии с выражением

$$\Delta_m(k)=\gamma\, e^f(k)\, y(k-m), \tag{1}$$

где

$e^f(k)=y(k)-\hat{y}(k)$ − ошибка предсказания вперёд,

γ − постоянная, определяющая характеристики сходимости алгоритма. При этом новые значения коэффициентов фильтра вычисляются как

$$\alpha_m(k+1)=\alpha_m(k)+\Delta_m(k), \tag{2}$$

что позволяет в конечном итоге минимизировать среднеквадратическую ошибку $\overline{e(k)^2}$.

Для того чтобы повысить разрешающую способность АСО, в алгоритме подстройки коэффициентов фильтра наряду с величиной $e^f(k)$ используется ошибка предсказания назад:

$$e^b(k)=y(k-M)-\sum_{m=1}^{M}\alpha_m(k)\, y(k-M+m). \tag{3}$$

При этом выражение для подстройки коэффициентов принимает вид

$$\alpha_m(k+1)=\alpha_m(k)+\gamma\,[e^f(k)\, y(k-m)+e^b(k)\, y(k-M+m)]. \tag{4}$$

Как видно из описания алгоритма стохастического градиентного метода подстройки коэффициентов фильтра, одной из задач при его реализации является

выбор значения γ. Для обоснованного выбора значения этого параметра в рамках данной работы была разработана программная модель процесса быстрого поиска системы FFH/MFSK, использующего стохастический градиентный и были проведены машинные эксперименты с целью оценки влияния значения γ на вероятность ошибки определения MSB.

На рисунках 2, 3, 4 приведены результаты экспериментов при порядках фильтра 7, 14 и 21 и при следующих основных параметрах модели:

- длина регистра сдвига PN последовательности – 9 бит;
- длина информационной посылки – 1 бит;
- число скачков частоты на одной информационной посылке – 10;
- частота дискретизации – 24000000 Гц;
- число тактов дискретизации, приходящихся на один скачок частоты FFH/MFSK – 2400;
- помеха в канале – аддитивный белый гауссовский шум, -12дБ.

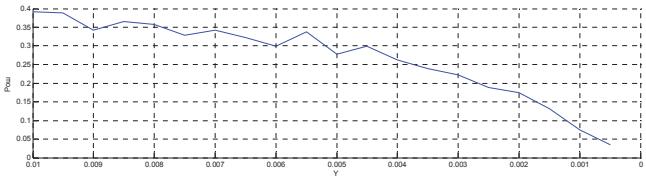


Рисунок 2. Зависимость $P_{ош}$ определения MSB от γ, порядок фильтра равен 7.

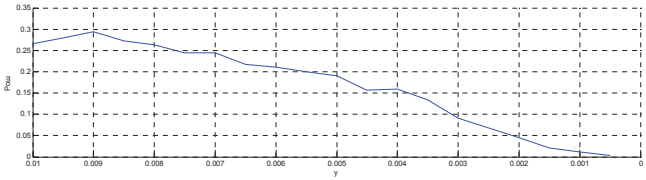


Рисунок 3. Зависимость $P_{ош}$ определения MSB от γ, порядок фильтра равен 14.

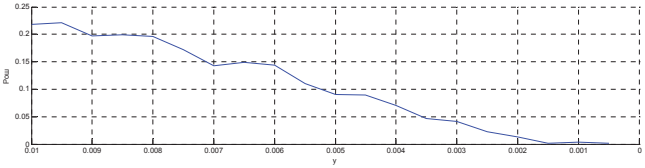


Рисунок 4. Зависимость $P_{ош}$ определения MSB от γ, порядок фильтра равен 21.

По полученным результатам можно сделать следующие выводы:

- при уменьшении порядка фильтра, уменьшая также значение γ, можно добиться приемлемого значения $P_{ош}$ определения MSB;
- при заданных основных параметрах системы FFH/MFSK и порядке фильтра в пределах от 14 до 21 при значениях γ меньше 0.001 величина $P_{ош}$ остаётся практически неизменной.

Литература

1. Скляр Б. Цифровая связь, Теоретические основы и практическое применение. 2-е издание.: Пер. с англ. − М.: «Вильямс», 2003.
9. Schilling D. L., et. al. Broadband CDMA for Personal Communications Systems. IEEE Communications Magazine, vol. 29, n. 11, November 1991, pp. 86-93.
10. Ponnusamy J.A., Srinath M.D. Asquisition of pseudonoise codes in FH systems, IEEE transactions on aerospace and electronic systems vol. aes-17, no. 3 may 1981.

11. Elhakeem A.K., Takhar G.S. and Gupta S.C. (1980) New code acquisition techniques in spread spectrum communication, IEEE Transactions on Communications, Feb. 1980, COM-28, 249-257.

ПРИМЕНИЕ СТАНДАРТА OPENMP ДЛЯ ТЕСТИРОВАНИЯ ПСЕВДОСЛУЧАЙНЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ

Р.Р. Вильданов, В.В. Маркин, Р.В. Мещеряков

ФГБОУ ВПО Томский государственный университет систем управления и радиоэлектроники, Кафедра комплексной информационной безопасности электронно-вычислительных систем

Введение

Тестирование псевдослучайных последовательностей — совокупность методов определения меры близости заданной псевдослучайной последовательности к случайной. В качестве такой меры обычно выступает наличие равномерного распределения, большого периода, равной частоты появления одинаковых подстрок и т. п.

Зная вероятностные свойства истинно случайной последовательности, можно на их основе проверять гипотезу о том, на сколько сгенерированная последовательность похожа на случайную. Для этого для каждого теста подбирается подходящая статистика, вычисляется её значения для идеальной и сгенерированной последовательности. Если разность этих значений превышает некоторое критическое значение, установленное заранее, то последовательность считается неслучайной[1].

Разработчиками тестов применяются несколько различных подходов в интерпретации результатов тестирования:

- пороговое значение – подсчет какой-либо статистической величины двоичной последовательности $c(s)$ и его сравнении с некоторым пороговым значением. Если полученное значение $c(s)$ меньше порогового значения, то последовательность s не проходит данный тест;
- фиксированный диапазон значений – подсчет некоторой статистической величины двоичной последовательности $c(s)$. Если $c(s)$ выходит за пределы некоторого диапазона значений, то последовательность s не проходит данный тест;

- значения вероятности – определение того факта, что двоичная последовательность s проходит статистический тест, включает подсчет не только статистической величины $c(s)$, но и соответствующее ей значение вероятности p. Обычно подсчет конкретной статистической величины производится таким образом, чтобы её большие значения предполагали неслучайный характер последовательности s. Тогда p есть вероятность получения значения $c(s)$ большего либо равного значению $c(s')$, высчитанного для истинно случайной последовательности s'. Следовательно, малые значения вероятности p (обычно $p< 0,05$ или $p< 0,01$) могут быть интерпретированы как доказательство того, что s не является случайной. Таким образом, если для некоторого фиксированного значения α значение вероятности $p<\alpha$, то двоичная последовательность s не проходит данный тест. Как правило, α принимает значения из интервала [0,001; 0,01].

Статистические тесты NIST – пакет статистических тестов, разработанный Лабораторией информационных технологий, являющейся главной исследовательской организацией Национального института стандартов и технологий (NIST). В его состав входят 15 статистических тестов, целью которых является определение меры случайности двоичных последовательностей, порождённых либо аппаратными, либо программными генераторами случайных чисел. Эти тесты основаны на различных статистических свойствах, присущих только случайным последовательностям.

Стандарт программирования OpenMP позволяет с минимальной корректировкой существующего кода адаптировать программы для работы на многопроцессорных компьютерах, а такие мощные и в то же время легкие для использования программистами средства как автоматическое распараллеливание циклов с использованием функций редукции и параллельные секции упрощают и ускоряют реализацию задач параллельных вычислений[2]. На диаграммах 1 и 2 показаны результаты сравнения времени выполнения тестов. В таблицах 1 и 2 приведены численные значения результатов выполнения тестов в секундах.

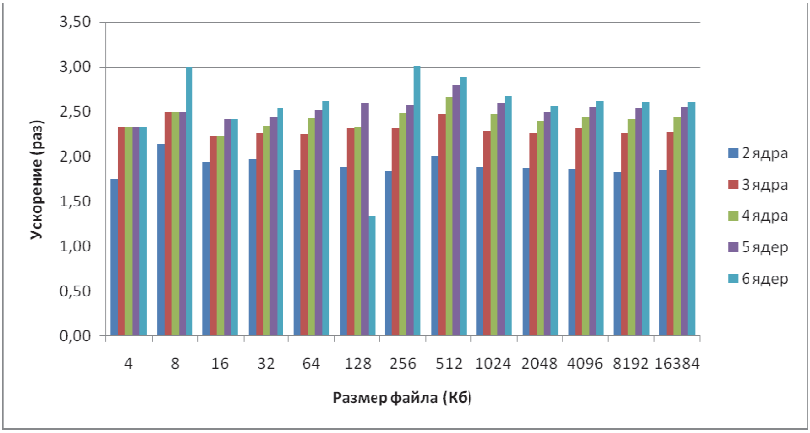


Рисунок 1. Ускорение времени тестирования при увеличении количества ядер (все тесты распараллелены и выполняются последовательно).

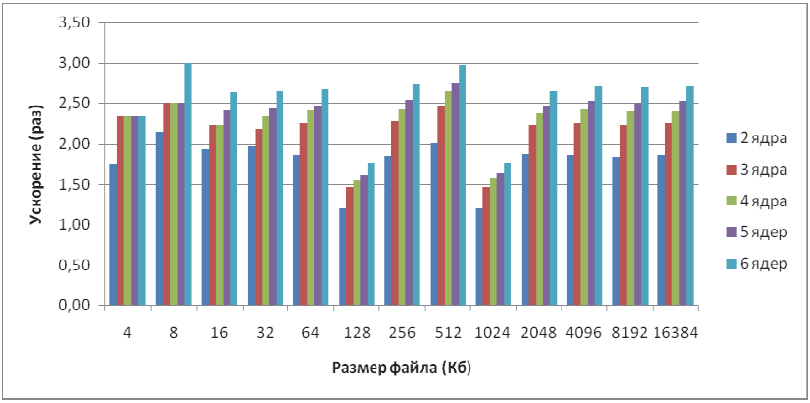


Рисунок 2. Ускорение времени тестирования при увеличении количества ядер (все тесты распараллелены и выполняются параллельно).

Таблица 1

Время тестирование последовательностей различной длины с использованием пакета статистических тестов NIST(все тесты распараллелены и выполняются последовательно), с

Размер файла (Кб)	Оригинал NIST	Оптимизация NIST (1 ядро)	Оптимизация NIST (2 ядра)	Оптимизация NIST (3 ядра)	Оптимизация NIST (4 ядра)	Оптимизация NIST (5 ядер)	Оптимизация NIST (6 ядер)
4	0,22	0,07	0,04	0,03	0,03	0,03	0,03
8	0,43	0,15	0,07	0,06	0,06	0,06	0,05
16	0,87	0,29	0,15	0,13	0,13	0,12	0,12
32	1,78	0,61	0,31	0,27	0,26	0,25	0,24
64	3,49	1,26	0,68	0,56	0,52	0,5	0,48
128	6,98	2,59	1,37	1,12	1,11	1	1,94
256	14,39	5,18	2,81	2,24	2,09	2,01	1,72
512	28,82	11,27	5,62	4,56	4,23	4,03	3,9
1024	56,35	20,95	11,1	9,15	8,49	8,08	7,82
2048	113,51	39,52	21,13	17,5	16,45	15,82	15,43
4096	227,63	82,9	44,64	35,79	33,89	32,46	31,61
8192	452,06	165,16	90,03	73,03	68,41	65,07	63,38
16384	921,68	332,6	179,79	146,05	136,3	130,47	127,27
Время (с)	1828,21	662,55	357,74	290,49	271,97	259,84	253,99

Таблица 2

Время тестирование последовательностей различной длины с использованием пакета статистических тестов NIST(все тесты распараллелены и выполняются параллельно), с

Размер файла (Кб)	Оптимизация NIST (1 ядро)	Оптимизация NIST (2 ядра)	Оптимизация NIST (3 ядра)	Оптимизация NIST (4 ядра)	Оптимизация NIST (5 ядер)	Оптимизация NIST (6 ядер)
4	0,07	0,04	0,03	0,03	0,03	0,03
8	0,15	0,07	0,06	0,06	0,06	0,05
16	0,29	0,15	0,13	0,13	0,12	0,11
32	0,61	0,31	0,28	0,26	0,25	0,23
64	1,26	0,68	0,56	0,52	0,51	0,47
128	1,65	1,37	1,13	1,06	1,02	0,94
256	5,18	2,81	2,28	2,13	2,04	1,89
512	11,27	5,62	4,57	4,24	4,1	3,79
1024	13,36	11,1	9,16	8,5	8,18	7,61
2048	39,52	21,13	17,73	16,61	16,09	14,95
4096	82,9	44,64	36,74	34,16	32,85	30,6
8192	165,16	90,03	74,04	68,55	66,07	61,32
16384	332,6	179,79	147,29	138,55	131,99	122,88
Время (с)	654,02	357,74	294,00	274,80	263,31	244,87

Заключение

Таким образом, показана возможность применения стандарта программирования OpenMP при тестировании псевдослучайных последовательностей. Необходимо отметить, что распараллеливание тестов позволяет сократить время проведения тестов.

Литература

1. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications [Электронный ресурс] / Rukhin A., Soto J., Nechvatal J., Smid M., Barker E., Leigh S., Levenson M., Vangel M., Banks D., Heckert A.,

Dray J., Vo S. – Режим доступа: http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf . Дата обращения: 10.02.2012

2. The OpenMP® API specification for parallel programming [Электронный ресурс]. – Режим доступа: http://openmp.org . Дата обращения: 10.02.2012
3. Иванов М.А. Теория, применение и оценка качества генераторов псевдослучайных последовательностей / Чугунков И.В. – М: КУДИЦ-ОБРАЗ, 2003. – 240 с. – (СКБ – специалисту по компьютерной безопасности)

УНИВЕРСАЛЬНАЯ ОПЕРАЦИЯ НАД ПРЯМОУГОЛЬНЫМИ МАТРИЦАМИ С МНОГОМЕРНОЙ ИНДЕКСАЦИЕЙ

Г.Г. Исламов

Удмуртский государственный университет

В работах [1-7] раскрыта фундаментальная роль универсальной операции над прямоугольными матрицами с одномерной индексацией [1]. Эта операция естественным образом распространяется на прямоугольные матрицы с многомерной индексацией. В докладе будут рассмотрены различные применения этого обобщения на прямоугольные матрицы с трёхмерной индексацией и его реализация средствами многопоточного программирования технологии CUDA [8].

Пусть N есть множество целых чисел. Переменная i (j) со значениями из декартового произведения N^m (N^n) называется $m-$ мерным (n-мерным) индексом. Сравнение наборов из одного и того же декартова произведения покоординатное. Оно порождает лексикографическое упорядочение, относительно которого отрезки $I=[I_1,I_2]\subset N^m$ ($J=[J_1,J_2]\subset N^n$) определяются стандартным образом. Наборы вещественных чисел $A=(a_{ij})_{i\in[I_1,I_2]}^{j\in[J_1,J_2]}$ образуют прямоугольную матрицу порядка $I\times J$ с m–мерной индексацией строк и n–мерной индексацией столбцов.

Стандартные бинарные операции в алгебре прямоугольных матриц порядка $I\times J$ над полем вещественных чисел дополним унарной операцией. Рассмотрим две группы переменных $(x_j)_{j\in[J_1,J_2]}$ и $(y_i)_{i\in[I_1,I_2]}$. Линейное отображение $y_i=\sum_{j\in[J_1,J_2]}a_{ij}x_j$, $i\in[I_1,I_2]$ и ненулевой элемент a_{kl} порождают новое линейное отображение группы переменных $(x_j)_{j\in[J_1,J_2]}\setminus\{x_l\}\cup\{y_k\}$ в группу переменных $(y_i)_{i\in[I_1,I_2]}\setminus\{y_k\}\cup\{x_l\}$ по формулам

$$\begin{cases} x_l = \dfrac{1}{a_{kl}} y_k + \sum\limits_{j \in [J_1, J_2] \setminus \{l\}} \dfrac{-a_{kj}}{a_{kl}} x_j, \\ y_i = \dfrac{a_{il}}{a_{kl}} y_k + \sum\limits_{j \in [J_1, J_2] \setminus \{l\}} \left(a_{ij} - \dfrac{a_{il} \cdot a_{kj}}{a_{kl}} \right) x_j, \; i \neq k. \end{cases} \tag{1}$$

Обозначая через b_{ij} элементы матрицы нового линейного отображения, получим формулы универсальной операции над матричными структурами

$$b_{ij} = \frac{1}{a_{kl}} \begin{cases} 1, \; \textit{если } i=k \; \textit{и} \; j=l, \\ a_{il}, \; \textit{если } j=l, \\ -a_{kj}, \; \textit{если } i=k, \\ a_{ij} \cdot a_{kl} - a_{il} \cdot a_{kj}, \; \textit{если } i \neq k \; \textit{и} \; j \neq l. \end{cases} \tag{2}$$

Ряд свойств этой операции для прямоугольных матриц с одномерной индексацией рассмотрен в работе [5]. В докладе будут представлены многоцелевые алгоритмы [7], в основе которых лежит универсальная операция для прямоугольных матриц с трёхмерной индексацией строк и столбцов.

Вид формул универсальной операции над матричными структурами вызывает определённые ассоциации при её программной реализации. Как известно, операция деления является наиболее затратной при выполнении машинной арифметики [9]. Значит, необходима такая форма представления числа, которая исключала бы деление при получении промежуточных результатов. Этого можно достичь за счёт введения упорядоченной пары (α, β) ([10], с. 119), которая представляет отношение вещественных чисел $\frac{\alpha}{\beta}$. С другой стороны, операция вычитания близких чисел приводит к резкому повышению относительной погрешности вычислений. Поэтому вычитание также следует исключить при получении промежуточных результатов за счёт введения дополнительной упорядоченной пары $<\alpha, \beta>$ ([10], с. 120), обозначающей разность $\alpha - \beta$. В любой реализации множество компьютерных чисел конечно, арифметические операции коммутативны, но не ассоциативны и не

дистрибутивны. Этот недостаток устраняется за счёт введения класса *приближённых чисел* как совокупности пар вида $\{a, \delta_a\}$, где a есть компьютерное число, а δ_a - переменная с известной границей изменения. Каждая такая пара представляет вещественное число вида $A = a(1+\delta_a)$. Следует помнить, что результатом вычислений на компьютере является не компьютерное число, а приближённое число с указанной границей изменения переменной δ_a относительной погрешности. Для положительных чисел $A = a(1+\delta_a)$ и $B = b(1+\delta_b)$, их суммы $C = A + B$ и произведения $D = A \cdot B$ имеем $C = a \oplus b(1+\delta_c)$ и $D = a \otimes b(1+\delta_d)$, где $a \oplus b$ и $a \otimes b$ есть компьютерные числа и можно указать вполне определённые границы для переменных $\delta_c = \delta_a \frac{a}{a+b} + \delta_b \frac{b}{a+b} + \delta_+$ и $\delta_d = \delta_a + \delta_b + \delta_*$. Всё это указывает на то, что при программной реализации универсальной операции следует ограничиться машинным представлением только положительных чисел.

Структура $(<\alpha_1, \beta_1>, <\alpha_2, \beta_2>)$, обозначающая число вида $\frac{\alpha_1 - \beta_1}{\alpha_2 - \beta_2}$ позволяет при работе с компьютерными числами применять только две операции: сложение и умножение положительных чисел. Воспользуемся встроенным векторным типом float4 CUDA-расширения языка C [8] для программной реализации введённой тетрады представления чисел и арифметических операций над этими тетрадами. Ниже приводится функция операции вычитания. Аналогичные функции могут быть выписаны для операций сложения, умножения и деления тетрад.

```
__device__ float4 operator – (const float4 & a, const float4 & b);
{
    float 4 c;
        c.x = a.x·b.z + a.y·b.w + a.z·b.y + a.w·b.x;
```

$c.y = a.y \cdot b.z + a.x \cdot b.w + a.z \cdot b.x + a.w \cdot b.y;$

$c.z = a.z \cdot b.z + a.w \cdot b.w;$

$c.w = a.w \cdot b.z + a.z \cdot b.w;$

return c;

}

Приведём соответствие встроенных переменных CUDA-расширения языка программирования С введённым выше понятиям, относящимся к прямоугольным матрицам с многомерной индексацией.

Переменная *gridDim* типа *Dim*3 содержит размерность сетки (*grid*) блоков иерархической структуры потоков (*threads*) CUDA, выполняющих процедуру *Pivot*, реализующую универсальную операцию *Pivot*. Эта переменная отвечает размерности I строк исходной матрицы A. Переменная *blockIdx* типа $u$int3 содержит индекс блока внутри сетки и отвечает трёхмерному индексу i строки исходной матрицы A.

Переменная *blockDim* типа *Dim*3 содержит размерность блока иерархической структуры потоков (*threads*) CUDA, выполняющих процедуру *Pivot*. Эта переменная отвечает размерности J столбцов исходной матрицы A. Переменная *threadIdx* типа *u* int *3* содержит индекс потока внутри блока и отвечает трёхмерному индексу j столбца исходной матрицы A.

Приводимый ниже фрагмент программы на языке С для CUDA описывает универсальную операцию над матричными структурами для случая одномерной индексации.

```
__global__ void Pivot(float * Ad, float * Bd, int p, int q, int M)
{
    float a,b,c,d;
    int i = threadIdx.x;
    int j = threadIdx.y;
     a = Ad[i+M*j]; b = Ad[i+M*q];
```

```
    c = Ad[p+M*j]; d = Ad[p+M*q];
     if ((i == p) && (j == q)) Bd[i+M*j] = 1 / d;
    else if (i == p) Bd[i+M*j] = -a / d;
      else if (j == q) Bd[i+M*j] = a / d;
           else Bd[i+M*j] = a - b * c /d;
  __syncthreads();
}
```

Последний параметр этой процедуры задаёт размеры преобразуемой матрицы, которая определяется первым параметром. Выбор графического устройства сильно влияет на точность и быстродействие вычислений, производимых с помощью процедуры Pivot. Наиболее перспективными для наших целей оказываются графические устройства на базе процессоров Fermi компании NVIDIA [8].

Литература

1. Исламов Г.Г. Универсальная операция над матричными структурами // Современные проблемы вычислительной математики и математической физики: Международная конференция, Москва, МГУ имени М.В. Ломоносова, 16-18 июня 2009 г. Тезисы докладов. – М.: Издательский отдел факультета ВМК МГУ имени М.В. Ломоносова; Макс Пресс, 2009. – 396 с.
2. Исламов Г.Г. Реализация на процессорах с технологией CUDA универсальной операции над матричными структурами // Труды Второй международная конференция «Трехмерная визуализация научной, технической и социальной реальности. Технологии высокополигонального моделирования», Т. 1, Ижевск – УдГУ, 2010. - С. 97-100.
3. Исламов Г.Г. Факторы процесса внедрения высокопроизводительных вычислений в учебно-научный и производственный потенциал Удмуртской Республики // Труды Первой международной конференции «Трёхмерная визуализация научной, технической и социальной реальности. Кластерные технологии моделирования, Т. 1. Ижевск, УдГУ (4-6 февраля 2009 г.), С. 139-143.

4. Исламов Г.Г., Коган Ю.В. Алгоритм нахождения начального состояния марковского процесса с полиэдральными ограничениями // Вестник Удмуртского университета. – 2007. – № 1: Математика. – С. 67–74.
5. Исламов Г.Г., Коган Ю.В. Об одном алгоритме поиска базисного минора матрицы // Вестник удмуртского университета. – 2006. - № 1 : Математика. – С. 63 – 70.
6. Исламов Г.Г. Принципы оптимальности. Учебное пособие. Ижевск, Изд-во УдГУ, 1998. – 124 с.
7. Исламов Г.Г., Исламов А.Г. Многоцелевые алгоритмы: структура, программная реализация и приложения // Технологии информатизации профессиональной деятельности (в науке, образовании и промышленности) ТИПД-2011: Труды 3 Всерос. науч. конф. с междунар. участием, Ижевск, УдГУ, 2011. - С. 32-22.
8. NVIDIA CUDA Programming Guide Version 4.1 // www.nvidia.com.
9. Шауман А. М. Основы машинной арифметики. – Л.: Изд-во ЛГУ, 1979. – 312 с.
10. Арнольд И В. Теоретическая арифметика. – М.: Учпедгиз, 1939. – 400 с.

ИЗМЕРИТЕЛЬ СРЕДНЕКВАДРАТИЧЕСКОГО ОТКЛОНЕНИЯ ФЛУКТУАЦИИ РАЗНОСТИ ФАЗ СИНХРОНИЗИРУЮЩИХ СИГНАЛОВ ДЛЯ НАВИГАЦИОННОЙ АППАРАТУРЫ

А.Ю. Тараненко, П.В. Штро, А.Г. Андреев

Институт инженерной физики и радиоэлектроники СФУ

660074, г. Красноярск, ул. Киренского, 26

В радионавигационных приемниках, как правило, применяют блок опорных частот (БОЧ) для обеспечения работы приемника в режиме внутренней и внешней синхронизации. Погрешность привязки внутреннего генератора к сигналу внешней синхронизации приводит к погрешности измерения навигационных параметров в приемнике и в наибольшей степени обусловлена величиной флуктуации разности фаз между опорной и внешней частотами. Для навигационных задач абсолютное значение разности фаз внутреннего и внешнего сигналов является устраняемой систематической погрешностью. В свою очередь, флуктуация разности фаз является случайной погрешностью и служит источником ошибки измерения навигационных параметров.

Актуальность работы заключается в необходимости высокоточного измерения СКО флуктуации разности фаз внутренней и внешней опорных частот для оценки качества работы режима внешней синхронизации навигационного приемника.

В простейшем случае, измеритель СКО разности фаз должен состоять из фазометра и устройства, которое вычисляет по заданному количеству измерений СКО флуктуации разности фаз. Задача измерения разности фаз может быть решена многими способами: методом компенсации фазы, методом преобразования интервала времени в напряжение [1, 2, 3], цифровым методом подсчета количества импульсов [2], методом измерения фазы с преобразованием частоты [3], синхронным детектированием [1]. Все перечисленные способы имеют свои преиму-

щества и недостатки, в данном измерителе применяется квадратурный метод измерения фазового сдвига, так как он обладает следующими достоинствами:

- относительная простота и легкость реализации в ПЛИС;
- возможность проведения измерений для зашумленных сигналов.

Как будет видно из дальнейших выкладок, важным достоинством данного метода, является то, что он не чувствителен к амплитудам входных сигналов, если они постоянны, или меняются достаточно медленно за время одного периода входного сигнала.

Устройство, вычисляющее СКО флуктуации по заданному количеству измерений разности фаз, удобно реализовать на той же ПЛИС в виде встраиваемого микропроцессорного ядра, которое обрабатывает данные и отправляет результаты в ПК.

Рассмотрим два сигнала, одинаковой частоты, между которыми необходимо измерить СКО флуктуации разности фаз:

$$S_1(t) = A_1 \sin(\omega t + \varphi_1); \tag{1}$$

$$S_2(t) = A_2 \sin(\omega t + \varphi_2); \tag{2}$$

Квадратурный метод измерения разности фаз предполагает формирование синфазной и квадратурной составляющих. Перемножив сигнал $S_1(t)$ на сигнал $S_2(t)$ получим синфазную составляющу, а перемножив $S_1(t)$ на $S_2(t)$, преобразованный по Гильберту, получим квадратурную составляющую:

$$\begin{aligned} S_I &= S_1(t)S_2(t) = A_1 \sin(\omega t + \varphi_1) \cdot A_2 \sin(\omega t + \varphi_2) = \\ &= \frac{A_1 A_2}{2}\left[(\cos(\varphi_1 - \varphi_2) - \cos(2\omega t + \varphi_1 - \varphi_2)\right]; \end{aligned} \tag{3}$$

$$\begin{aligned} S_Q &= S_1(t) \cdot H(S_2(t)) = A_1 \sin(\omega t + \varphi_1) \cdot A_2 \cos(\omega t + \varphi_2) = \\ &= \frac{A_1 A_2}{2}\left[(\sin(\varphi_1 - \varphi_2) + \sin(2\omega t + \varphi_1 + \varphi_2)\right]; \end{aligned} \tag{4}$$

Для того чтобы уменьшить шум при цифровой обработке удобно применить накопление, это также позволяет избавиться от колебания на удвоенной частоте. Тогда при времени накопления Δt много большем, чем период измеряемого сигнала T, можно записать:

$$\int_0^{\Delta t} \cos(\varphi_1 - \varphi_2)dt >> \int_0^{\Delta t} \cos(2\omega t + \varphi_1 + \varphi_2)dt; \quad (5)$$

Рассматривая аналогичное выражение для удвоенной частоты в формуле (4), в итоге для средних значений синфазной и квадратурной составляющих получим:

$$\overline{S}_I = \frac{A_1 A_2}{2}\cos(\varphi_1 - \varphi_2); \overline{S}_Q = \frac{A_1 A_2}{2}\sin(\varphi_1 - \varphi_2); \quad (6)$$

Таким образом, разность фаз может быть определена как арктангенс отношения средних значений квадратурной и синфазной составляющих:

$$\varphi_1 - \varphi_2 = \arctan\left(\frac{\overline{S}_Q}{\overline{S}_I}\right) = \arctan\left[\frac{A_1 A_2 \sin(\varphi_1 - \varphi_2)}{A_1 A_2 \cos(\varphi_1 - \varphi_2)}\right]; \quad (7)$$

Подставляя выражения (6) в формулу (7) можно заметить, что разность фаз не зависит от амплитуд входных сигналов, так как они сокращаются в формуле (7).

Структурная схема измерителя приведена на рис. 1. Цифровая часть устройства синтезирована в ПЛИС Spartan 6 фирмы Xilinx. В этой же ПЛИС расположен микропроцессорный блок MicroBlaze, который накапливает заданное число измерений и вычисляет СКО разности фаз. Результаты измерений передаются в ПК через сеть Ethernet по TCP/IP протоколу, через который также осуществляется управление измерителем.

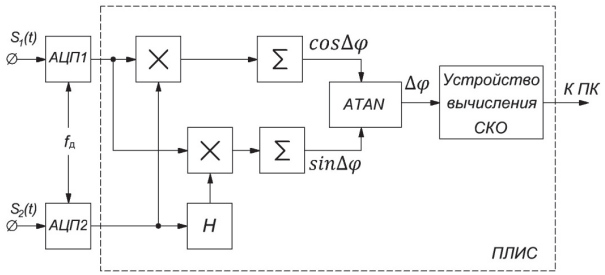


Рисунок 1. Структурная схема измерителя СКО разности фаз.

На рисунке 1 обозначено:

- $S_1(t), S_2(t)$ – входные сигналы;

- АЦП1, АЦП2 – аналогово-цифровые преобразователи, тактируемые частотой $f_д$;
- X – устройство перемножения отчетов;
- H – преобразователь Гильберта;
- ∑ – аккумуляторы, осуществляющие усреднение синфазной и квадратурной составляющих;
- ATAN – устройство вычисления функции арктангенса от отношения двух входных аргументов.

Устройство вычисления СКО разности фаз накапливает заданное число измерений N и вычисляет среднеквадратическое отклонение по стандартной формуле:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\Delta\varphi_i - \Delta\bar{\varphi})^2}; \qquad (8)$$

В программе LabView было проведено моделирование работы измерителя, в модели присутствовал источник фазового шума, СКО которого требовалось измерить, а также источник аддитивного шума. Результаты работы приведены на рис.2-4.

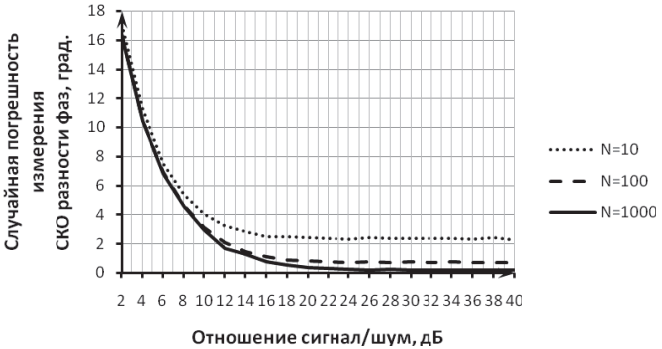


Рисунок 2. Случайная погрешность измерения СКО разности фаз в зависимости от отношения сигнал/шум при различном количестве накоплений N (8 разрядов АЦП, СКО разности фаз 10°).

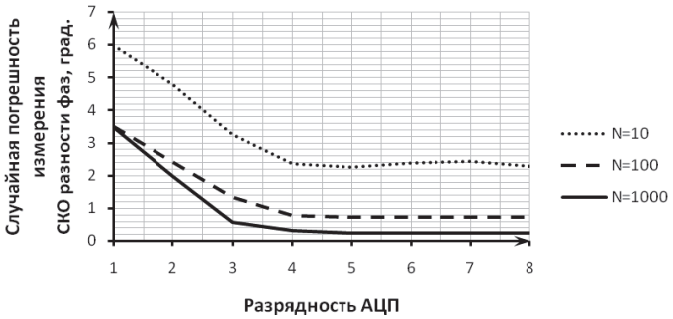


Рисунок 3. Случайная погрешность измерения СКО разности фаз в зависимости от разрядности АЦП при различном количестве накоплений N (СКО разности фаз 10°, с/ш 30 дБ).

Как видно из рис. 2, измерения достаточной точности (порядка десятых градуса) получаются при отношении сигнал/шум более 20 дБ (по мощности) и разрядности АЦП более 5, рис. 3. Из графиков на рис. 4 также видно, что аддитивный шум достаточно хорошо усредняется при значительном накоплении выборок.

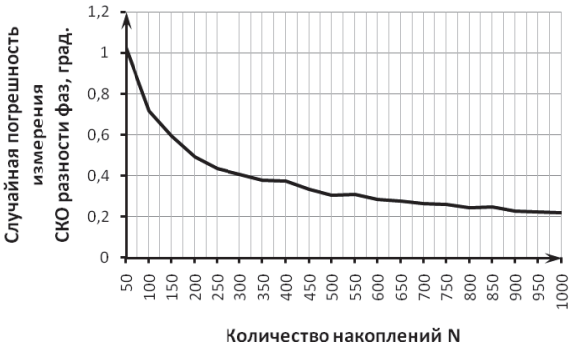


Рисунок 4. Случайная погрешность измерения СКО разности фаз в зависимости от количества накопленных выборок N (СКО разности фаз 10°, с/ш 30 дБ).

Таким образом, рассмотрен вариант цифрового измерителя среднеквадратического отклонения флуктуации разности фаз, основанный на квадратурном методе. Результаты моделирования работы измерителя в LabView показали, что целесообразно использовать прибор для измерения СКО разности фаз сигналов со значительным отношением сигнал/шум и относительно малыми флуктуациями фазы, так при использовании 8-ми разрядов АЦП, отношении сигнал/шум 30 дБ и накоплении 1000 измерений случайаная ошибка измерения составляет около 0,2°.

Литература

1. Клаассен, К.Б. Основы измерений. Электронные методы и приборы в измерительной технике / К. Б. Классен. – Москва: Постмаркет, 2000. – 352 с.
2. Гаврилов, Ю.С. Справочник по радио- измерительным приборам / Ю. С. Гаврилов, А.С. Еременко. – Москва: «Энергия», 1976. – 624 с. с ил.
3. Терешин, Г.М. Радио-измерения / Г.М Терешин.– М: «Энергия», 1968. – 400 с. с. ил.

ИМИТАТОР КОРОТКОВОЛНОВОГО КАНАЛА НА БАЗЕ DSP TMS320VC5410

Д.С. Зузлов, П.О. Иванов, А.П. Иванов

ФГБОУ ВПО «Пензенский государственный университет», факультет приборостроения, информационных технологий и систем

Сокращение сроков и повышение эффективности разработки и проектирования аппаратуры передачи данных (АПД), в частности, модемов, представляет собой актуальную задачу в развитии современной техники телекоммуникаций. Одним из путей её решения является внедрение и использование модельных экспериментов на этапах научно-исследовательских и опытно-конструкторских работ и испытания АПД.

Среди возможных объектов моделирования особенно существенную роль играет канал связи. Моделирование канала связи делает возможными испытания различных реализаций АПД в условиях, максимально приближенных к реальным, обеспечивает разнообразные желаемые условия экспериментов и высокую воспроизводимость их результатов, а также позволяет сократить объем натурных испытаний опытных и серийных образцов АПД за счет замены их лабораторными испытаниями. Наиболее важно для практики моделирование многолучевых каналов с замираниями. Необходимость моделирования таких каналов обусловлена также и тем, что при сравнительных испытаниях различных вариантов АПД на реальных трассах невозможно получить одинаковые условия распространения и повторить сеанс связи с задаваемыми разработчиком условиями распространения.

Среди известных методов моделирования каналов связи [1] особый интерес представляет функциональное моделирование с использованием сигнальных процессоров (DSP), заключающееся в создании специализированных имитаторов каналов связи. Такие имитаторы легко сопрягаются с реальной аппаратурой, входящей в состав систем передачи данных, позволяют с высокой степенью точно-

сти моделировать сигнал на выходе канала связи, представляют собой ценное дополнение для решения комплекса задач, связанных с разработкой и отладкой алгоритмов работы модемов, а также протоколов передачи данных и систем маршрутизации.

На кафедре «Информационная безопасность систем и технологий» Пензенского государственного университета разработан имитатор коротковолнового (КВ) канала, который представляет собой техническое средство для решения комплекса задач связанных с разработкой, отладкой, исследованием и испытанием средств связи и защиты информации.

Если учитывать реальный механизм многолучевого распространения сигналов в средах со случайно изменяющимися физическими свойствами, то модель многолучевого канала связи можно записать в следующем виде:

$$y(t)=K\{s(t)\}+\xi(t)=\sum_{i=1}^{N} m_i(t)s(t-\tau_i)+\xi(t), \qquad (1)$$

где $s(t)$ – сигнал на входе многолучевого канала связи;

K – это оператор, который отображает воздействие среды распространения вместе с антенными системами (собственно канала) на передаваемый сигнал $s(t)$;

$\xi(t)$ – аддитивная помеха;

i – номер луча распространения;

N – число лучей, формирующих выходной сигнал канала;

$m_i(t)$ – коэффициент передачи, характеризующий *i-ый* путь распространения;

τ_i – время запаздывания сигнала при распространении по i-му лучу.

В имитаторе КВ-канала выбран метод генерирования сигналов отдельных лучей, при котором весь высокочастотный тракт (радиопередатчик - среда - радиоприемник) заменен его низкочастотным эквивалентом и обработка ведется в диапазоне тональных частот выходного сигнала АПД. Каждая ветвь имитатора соответствует определенному пути распространения радиоволн. Замирания, моделируемые в каждой ветви, носят частотно-гладкий характер, а после объедине-

ния сигналов различных ветвей сумматором - частотно-селективный характер.

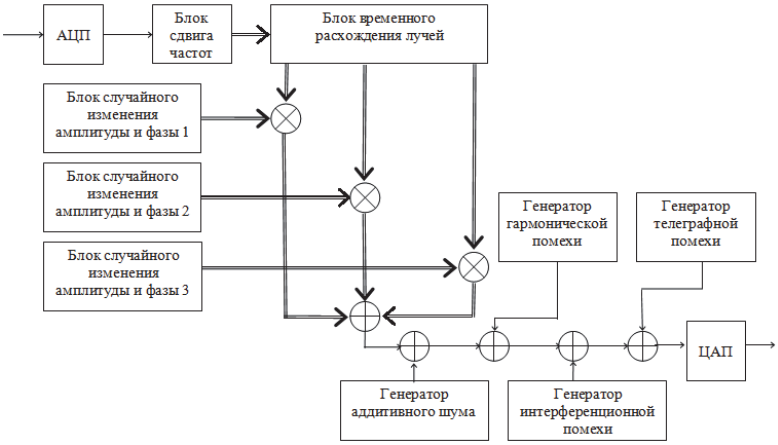


Рисунок 1. Структурная схема имитатора КВ-канала.

Имитатор КВ-канала разработанный на базе сигнального процессора TMS320VC5410 включен в состав прибора КП-ИАТС-М изготовленного на ФГУП «Пензенский научно-исследовательский электротехнический институт». Пользовательский интерфейс разработанного имитатора КВ-канала в составе прибора КП-ИАТС-М приведен на рисунке 2.

Имитатор КВ-канала имеет следующие технические характеристики:

- диапазон частот входного сигнала: от 300 Гц до 3400 Гц;
- максимальное количество лучей: 3;
- максимальная временная задержка лучей: 10мс;
- величина доплеровского рассеяния: от 0 до 5Гц;
- частота рассинхронизации: ±100 Гц;
- имитатор обеспечивает моделирование следующих непреднамеренных и преднамеренных помех: аддитивный шум; гармоническая помеха, сканирующая по частоте (интерференционная) и телеграфная помехи;
- возможность задания протяженности трассы.

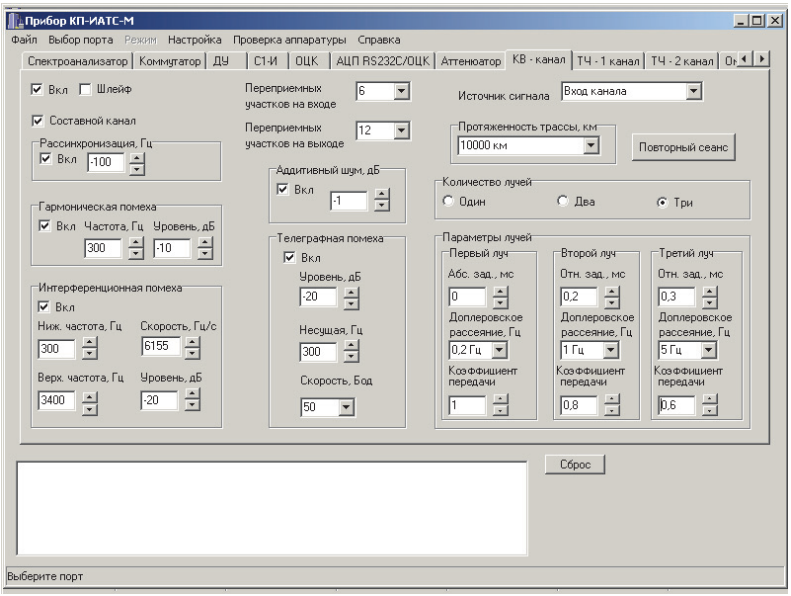


Рисунок 2. Пользовательский интерфейс имитатора КВ-канала.

Имитатор КВ-канала обладает следующими преимуществами:

- обеспечивает скрытность испытаний путем замены натурных испытаний их лабораторными эквивалентами;
- экономически выгоден за счет отсутствия натурных испытаний и низкой собственной себестоимости;
- имеет высокую воспроизводимость результатов испытаний;
- обладает гибкостью и расширяемостью за счет заранее заложенного запаса вычислительных ресурсов и применению современных технологий разработки.

Имитатор КВ–канала может быть использован для автоматизированного контроля параметров аппаратуры телекоммуникационных систем при настройке и приемо-сдаточных испытаниях в процессе производства на заводах-изготовителях, а также разработчиками и (или) заказчиками телекоммуникационных систем для их испытания, как по отдельности, так и в комплексе в режимах,

которые они посчитают нужными в процессе испытания или тренажа.

Литература

1. Галкин А.П., Лапин А.Н., Самойлов А.Г. Моделирование каналов систем связи. – М.: Связь, 1979. – 96 с.

Часть 4. Распределенные системы

ПРИМЕНЕНИЕ GNU MAKE К ПАРАЛЛЕЛЬНОЙ ОБРАБОТКЕ ДАННЫХ СПУТНИКОВОГО ПРИБОРА MODIS

И.А. Шмаков

ККУ «УГО ЧС и ПБ в Алтайском крае»

Аннотация

Описывается опыт применения GNU Make к параллельной обработке данных спутниковых приборов MODIS (от уровня 2 к уровня 3.) Обозначены перспективы применения созданного инструментария к обработке данных в рамках распределенной вычислительной системы.

Введение

Задачи мониторинга состояния окружающей среды на уровнях от регионального и выше немыслимы без применения спутниковых приборов. Среди таких приборов особое место занимают широкоугольные многоканальные радиометры, в т. ч. размещенные на космических платформах Terra и Aqua приборы MODIS [1], и прибор VIIRS в составе платформы Suomi NPP [2].

Данные этих приборов доступны через Internet, однако более примечательна поддержка ими режима «прямого вещания», позволяющего любому желающему, при наличии соответствующего оборудования, получать поток данных в реальном режиме времени.

Доступность исходных кодов программного обеспечения, как оригинального, используемого NASA, так и адаптированных для «конечного пользователя» версий (в частности, [3]), открывает перед исследователем определенный простор для творчества, — совершенствования алгоритмов и их адаптации к конкретным условиям.

Кроме того, вместе с режимом прямого вещания, данные этих приборов можно применить к задачам оперативного мониторинга состояния окружающей

среды. Среди прочего, обработка данных упомянутых приборов позволяет получать оперативную информацию по термальным аномалиям (включая пожары), состоянию снежного покрова, а также профили температуры и влажности (что можно применить к прогнозированию нестабильности атмосферы, гроз, пожароопасной обстановки, и т. д.)

Проблема

Применение оригинальных алгоритмов и кодов, однако, осложняется наличием нетривиальных зависимостей, как между отдельными процессами в составе системы обработки, так и этих процессов от получаемых через Internet вспомогательных данных (состояние атмосферы — температура, влажность, содержание отдельных атмосферных газов; состояние подстилающей поверхности — данные о снежном покрове, и т. д.)

Кроме того, в исследовательских целях может оказаться полезным выполнение фрагмента цепи обработки, с использованием данных NASA в качестве входных.

В последнем случае, однако, нужных входных данных может не оказаться в архиве. Так, ряд кодов т. н. «уровня 3» (L3) опирается на данные по коэффициентам спектральной яркости подстилающей поверхности (КСЯ ПП), перенесенных на регулярную (т. н. L2G) сетку. Полный набор таких данных представлен следующими продуктами:

- M*x*D09GST, M*x*D09GHK, M*x*D09GQK — КСЯ ПП; флаги состояния, данные с разрешением 500 м и 250 м;
- M*x*DMGGAD — геометрия наблюдения и освещения (только дневные данные, разрешение 1 км);
- M*x*DPT1KD, M*x*DPTHKM, M*x*DPTQKM — соответствие L2G-сеток различных разрешений, а также сеток L2G и сетки данных прибора (только дневные данные.)

Однако, в архиве [4] представлены лишь производные от них данные «L2G-lite» (M*x*D09GA, M*x*D09GQ), для применения которых необходима по меньшей мере адаптация вычислительных кодов.

Альтернативой является выполнение обработки начиная с более раннего этапа — получения вышеперечисленных данных L2G из публикуемых NASA данных по КСЯ ПП (M*x*D09) и геометрии наблюдения и освещения (M*x*D03) на сетке данных прибора (L2 «swath.»)

Кроме того, получение конкретного продукта из архива может потребовать предварительного оформления (через соответствующие HTTP-запросы) «заказа» на обработку и последующего слежения за состоянием последнего. В частности, архив NASA не содержит в полном объеме требуемые для построения L2G-продуктов КСЯ ПП данные M*x*D09.

Сложность результирующей системы обработки предполагает применение инструментов, выполняющих планирование обработки на основании информации об имеющихся и целевых данных, вычислительных кодах, и зависимостях между ними.

Предлагаемое решение

Одним из наиболее развитых инструментов «Make-семейства» является GNU Make [5].

Хотя Make и воспринимается, в большей степени, как инструмент для сборки программ из исходного кода, опыт показывает, что его также можно применить для организации многоэтапной обработки данных физического эксперимента.

Ясно, что «знание» этим инструментом *зависимостей* между различными этапами обработки (что необходимо для планирования обработки) позволяет, при соблюдении некоторых правил, выполнять распределение вычислений на уровне «плана обработки», причем как в пределах одного (многопроцессорного или многоядерного) вычислительного узла, так и обработку на кластере (см., например, [6].)

Очевидное преимущество этого подхода — отсутствие необходимости внесения изменений в, собственно, целевые вычислительные коды.

Основным неудобством Make, отмеченным нами ранее [7], является

использование имен файлов в качестве идентификаторов источников и целей. В простых случаях, в т. ч. при сборке ПО, соответствующие правила могут быть столь просты, как (зависимость объектного .o-файла от соответствующего исходного кода на языке C и соответствующая реализация — запуск компилятора $(CC)):

```
%.o : %.c
    $(CC) $(CFLAGS) -o $@ $<
```

В случае обработки результатов физического эксперимента, в качестве идентификатора удобнее было бы применить *кортеж-описание* — совокупность свойств набора данных (тип данных, время получения, географическое покрытие, инструмент и платформа, и т. д.)

В рамках разработываемой системы управления обработкой, вычисление зависимостей явно вынесено из контекста Make. Пример создаваемого описания зависимостей:

```
MYDPT1KD.A2011153.h22v03.hdf \
    MYDPTHKM.A2011153.h22v03.hdf \
    MYDPTQKM.A2011153.h22v03.hdf : \
  MYD03.A2011153.0440.hdf MYD03.A2011153.0445.hdf \
  MYD03.A2011153.0620.hdf MYD03.A2011153.0755.hdf \
  MYD03.A2011153.0800.hdf MYD03.A2011153.0935.hdf \
  MYD03.A2011153.1945.hdf MYD03.A2011153.1950.hdf \
  MYD03.A2011153.2125.hdf MYD03.A2011153.2130.hdf \
  MYD03.A2011153.2305.hdf

MYD09GST.A2011153.h22v03.hdf : \
  MYDPT1KD.A2011153.h22v03.hdf \
  MYDPTHKM.A2011153.h22v03.hdf \
  MYD09.A2011153.0440.hdf MYD09.A2011153.0445.hdf \
  MYD09.A2011153.0620.hdf MYD09.A2011153.0755.hdf \
  MYD09.A2011153.0800.hdf MYD09.A2011153.0935.hdf \
```

*MYD09.A2011153.1945.hdf MYD09.A2011153.2125.hdf *

MYD09.A2011153.2305.hdf

Отметим, что в обработке данных КСЯ ПП не участвуют гранулы 19:50Z и 21:30Z, поскольку они не содержат результатов «дневных» наблюдений.

Кроме блока генерирования зависимостей, в систему входят следующие компоненты:

- «главный» Makefile и Makefile-шаблон (subdir.mk.in) для рабочих директорий вычислительных процессов;
- программа приведения имен к «каноническому» и «короткому» (используемому в описаниях зависимостей) видам (согласно полю описания LOCALGRANULEID; hdfeos-symlink.sh);
- программа извлечения имен результирующих и временных файлов из управляющих (.pcf) файлов (pcf-files.sh);
- генераторы управляющих файлов (prep-*.sh) и обертки запуска (run-*.sh.)

Заключение

В настоящее время, система выполняет обработку данных MODIS L2 до продуктов L2G (КСЯ ПП, геометрия, соответствие сеток.)

Включение кодов обработки данных L2G до продуктов L3 требует разработки отдельного блока генерирования зависимостей с учетом временного окна.

В систему заложены средства защиты от одновременного выполнения одного и того же этапа обработки, в т. ч. при использовании нескольких Make-процессов. Предполагается, что при использовании общей (сетевой) файловой системы, это позволит выполнять параллельную обработку данных на вычислительном кластере, а при изменении реализации механизма защиты — и в составе слабосвязанной распределенной вычислительной системы.

Использованный способ вычисления зависимостей не бесспорен. В настоящее время, вычисление зависимостей полностью вынесено из контекста Make, что снижает удобство использования системы.

Отдельных усилий потребует автоматизация получения подлежащих обработке данных из архивов NASA, включая поиск гранулы и (при необходимости) оформление «заказа» на обработку.

Предполагается, что код системы управления обработкой будет выпущен как свободное программное обеспечение.

Литература

1. MODIS Website [Электронный ресурс] — 2012. — Режим доступа: http://modis.gsfc.nasa.gov/, свободный. — Загл. с экрана — Яз. Англ.
2. Suomi NPP [Электронный ресурс] — 2012. — Режим доступа: http://npp.gsfc.nasa.gov/, свободный. — Загл. с экрана — Яз. Англ.
3. DRL Document [Электронный ресурс] — 2012. — Режим доступа: http://directreadout.sci.gsfc.nasa.gov/, свободный. — Загл. с экрана — Яз. Англ.
4. LAADS Web [Электронный ресурс] — 2012. — Режим доступа: http://ladsweb.nascom.nasa.gov/, свободный. — Загл. с экрана — Яз. Англ.
5. Stallman, R. M. GNU Make Version 3.81. / Stallman, R. M., McGrath, R., Smith, P. D., — GNU Press.
6. distcc — Distributed compilation for faster C/C++ builds [Электронный ресурс] — 2012. — Режим доступа: http://distcc.googlecode.com/, свободный. — Загл. с экрана — Яз. Англ.
7. Shmakov I., A model for dependency-driven digital data processing // Neuroinformatics and its Applications: Proc. of the XIX Pan-Russian Workshop, Oct., 1–3, 2011; — Krasnoyarsk, Siberian Federal University, 2011.

РАСПРЕДЕЛЕННАЯ СИСТЕМА СБОРА И ОБРАБОТКИ ДАННЫХ ТЕХНОЛОГИЧЕСКИХ ПРОЦЕССОВ.

С.С. Черепанов

ФГБОУ ВПО «Алтайский государственный университет»

Введение

С точки зрения информационной системы, современный технологический процесс является совокупностью данных полученных с устройств регистрации характеристик данного процесса, во время его существования. Эти данные представляют собой весьма разнородную телеметрическую информацию, такую как: показания датчиков, аудио, фото, видеоинформацию.

Таким образом, этот процесс может быть представлен как распределенная система обработки данных, состоящая из многофункциональных узлов и накопителей, т. е это программно и аппаратно разнородная вычислительная сеть.

В силу разнородности этой системы, набор интерфейсов передачи данных в ней, может быть весьма широк. Каждый из модулей поддерживает, по крайней мере, один из стандартных интерфейсов, но этого бывает недостаточно для объединения всех функциональных узлов.

Исходя из вышеизложенного, представляет интерес создание системы обмена данными в таких сетях.

В данной работе рассмотрена система коммутации для разнородной сети технологических процессов.

Коммуникатор разнообразных источников данных

Существующие реализации подобных устройств коммутации, построены на основе аппаратных решений. Они весьма специфичны для каждой конкретной задачи, и соответственно, имеют узкую специализацию. Поэтому стоимость подобных решений, является весьма высокой. Кроме того данные комплексы нуждаются в весьма сложном и дорогостоящем обслуживании.

В связи с этим является целесообразным разработка программного, либо программно-аппаратного комплекса, поддерживающего множество интерфейсов, и способного выполнять указанные выше задачи.

Рассмотрим сетевую архитектуру разнородной вычислительной среды, с точки зрения обмена данными, она состоит из следующих частей:

- источники данных,
- интерфейсы источников,
- внутреннее представление,
- коммутатор,
- интерфейсы приемников,
- приемника данных.

Примерная функциональная структура системы представлена на рисунке 1.

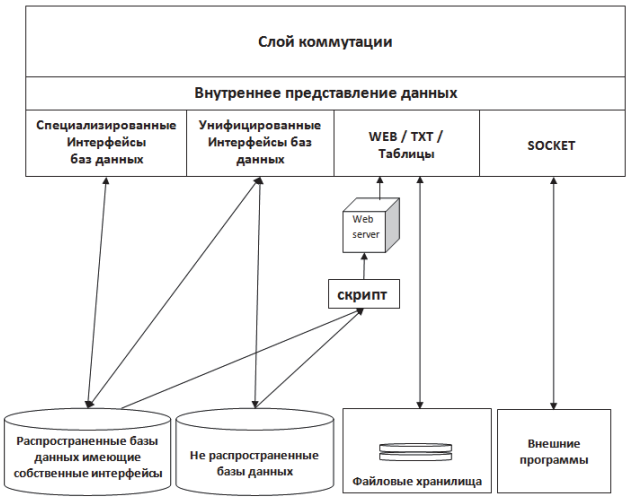


Рисунок 1. Функциональная структура системы.

Источники и приемники данных

Программы производящие распределенные вычисления, сохраняют результаты своих вычислений в некие временные хранилища, которые с точки зрения обмена данными между узлами, являются источниками. Такими хранилищами могут служить: базы данных, текстовые файлы, электронные таблицы и т.п. Про-

граммы, производящие вычисления могут напрямую передавать данные, без временного хранения.

Потенциальные приемники данных ничем не отличаются от источников, это некие хранилища информации с которыми работают программы выполняющие логику распределенных вычислений. Сами эти программы могут являться приемниками, без использования временных хранилищ.

Каждый источник данных, имеет свой набор интерфейсов доступа к информации, хранящейся в нем. Например это специализированные или унифицированные интерфейсы баз данных, WEB-интерфейс, или адреса файлов, хранящихся в дисковом пространстве серверов.

Некоторые интерфейсы являются односторонними, ввиду этого приемники не всегда имеют возможность воспользоваться теми же каналами передачи данных, через которые они поступили от источников. К примеру, среди односторонних можно выделить WEB – интерфейс, файловые хранилища на накопителях (при установленной защите от записи).

Конвертация данных при передаче между источниками и приемниками

В реальных системах обработки данных обмен чаще всего происходит посредством распространения интерфейсов и поддерживается набором более или менее стандартных протоколов. Однако интерфейсы и протоколы источников и приемников не всегда совпадают. Другими словами часто наблюдается следующая ситуация – данные получаются в рамках одного протокола и должны быть переданы по правилам другого протокола.

После получения данных по одному из интерфейсов, их необходимо преобразовать в формат способный хранить данные любого типа. Это необходимо для создания модульной системы. Благодаря этому для расширения количества интерфейсов, отпадает необходимость в написании «конверторов» из нового типа пришедших данных, во все ранее заложенные. Необходимо лишь уметь преобразовывать данные в общий тип данных и обратно. Это позволит создать систему, которая может расширяться за счет плагинов сторонних разработчиков, написанных по общему принципу.

Каждая порция данных на пути от источника к приемнику, проходит через коммутатор, отвечающий за соединение необходимых каналов. Для каждой пары источник-приемник порождается свой вычислительный поток, который отвечает за выполнение передачи данных между этими двумя хранилищами.

Пользовательская архитектура системы

Архитектура программного комплекса может быть представлена интерфейсом пользователя, который должен позволять настраивать систему, в соответствии с нуждами распределенной вычислительной среды. Работа системы определяется проходящими через неё потоками данных. Каждый поток данных определяется источником данных, приемником данных, их интерфейсами, а также периодичностью передачи. В качестве периода можно задавать нулевое значение, что позволит организовать передачу данных в режиме реального времени. Также при необходимости надо учесть механизмы авторизации в хранилищах данных.

Система сбора информации со стритсерверов для нужд ГИБДД

В качестве примера рассмотрим работу системы автоматической фотофиксации автомобольных правонарушений города Барнаула. ГИБДД г.Барнаула закупила у компании ООО «ВОКОРД» системы фотофиксации правонарушений. С технической стороны - это автономные устройства, состоящие из сервера на базе операционной системы Windows Embeded, оснащенного камерами для фотосъемки, и радарами для измерения скорости проезжающих автомобилей. На каждом сервере установлено специальное программное обеспечение, которое отвечает за управление камерами и радарами, а также распознавание государственных регистрационных знаков автомобилей. Информация о каждом автомобиле, проехавшем под таким устройством, записывается в базу данных MS SQL соответствующего сервера.

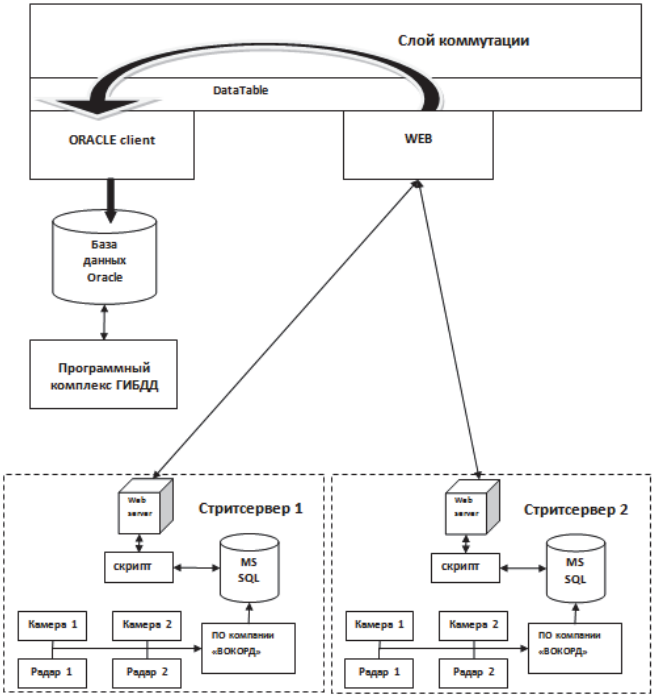


Рисунок 2. Абстрактная модель.

Для обработки и долгосрочного хранения этих данных появилась необходимость в передаче этих записей на внутренние серверы ГИБДД. По причине замкнутости системы Embeded и невозможности установки дополнительного системного программного обеспечения, на каждом стритсервере был установлен портативный WEB – сервер. При помощи скрипта написанного на языке php, WEB – сервер читает данные из базы MS SQL, и, формируя их, передает по HTTP протоколу.

Основная программа, находящаяся на внутреннем сервере ГИБДД, отправляя запросы к Web – серверам, по сути, управляет очередностью загрузки данных с разных стритсерверов. Получая очередную порцию записей, конвертирует её в

тип DataTable. После чего каждая запись переводится в тип OracleCommand и записывается в базу данных Oracle, используя интерфейс Oracle Client.

На данный момент описанная система введена в эксплуатацию и работает без перебоев в течение трех месяцев.

Заключение

Предложена и описана система обмена данными в распределенных системах технологических процессов. Описана структура и архитектура данной системы. Рассмотренные решения, апробированы на примере работы системы сбора данных с устройств фотофиксации ГИБДД.

АППАРАТНО-ПРОГРАММНЫЙ КОМПЛЕКС «СОКОЛ» ДЛЯ УДАЛЕННОГО МОНИТОРИНГА СЕТЕВЫХ ШКАФОВ

С.А. Бабичев, П.М. Зацепин

ФГБОУ ВПО Алтайский государственный университет

В сферах деятельности имеющих распределенные узлы, требующие постоянного контроля, часто встает вопрос о своевременном, а зачастую и в режиме реального времени сборе информации о состоянии узлов. В первую очередь это связанно с безопасностью содержимого узлов.

Проблемы защиты антивандальных шкафов интернет провайдеров.

При функционировании такой организации, как интернет провайдер одной из важных проблем является защита оборудования от случайного или намеренного повреждения. Эта проблема может быть решена посредством применения так называемых антивандальных шкафов. Данные шкафы, как правило, размещаются в местах общей доступности, что предусматривает человеческий фактор влияния на функционирование их внутреннего содержимого. Из общей практики эксплуатации оборудования выявлены основные факторы воздействия на антивандальный шкаф: температура, влажность, вибрации. Данные факторы, при превышении допустимых норм приводят к неисправностям и поломке дорогостоящего оборудования внутри шкафа.

Для решения указанной проблемы необходимо постоянно контролировать вышеперечисленные факторы, производить мониторинг всех узлов и своевременно сообщать оператору об внештатных ситуациях. Для обеспечения безопасности и улучшения качества обслуживания необходимо дополнительно производить регистрацию случаев открытия дверцы шкафа, это предотвратит несанкционированный доступ в ящик, и отключения электроэнергии, это позволит более точно диагностировать отсутствие отклика оборудования на узле связи.

Чтобы обеспечить полный набор функций по сбору и обработке информации, необходим аппаратно-программный комплекс мониторинга сетевых шкафов. В данной работе предлагается такая система под рабочим названием «Сокол».

Программная часть предлагаемого комплекса

На рынке имеется множество решений в различных исполнениях для конкретной задачи, но в каждом решении есть различные недостатки, такие как: стоимость, функциональность, избыточность и др.

Программная часть комплекса состоит из сервера Zabbix.

Zabbix является открытым программным обеспечением, созданным для мониторинга и отслеживания статусов разнообразных сервисов компьютерной сети, серверов и сетевого оборудования.

Zabbix поддерживает несколько видов мониторинга:

- Simple checks – может проверять доступность и реакцию стандартных сервисов, таких как SMTP или HTTP без установки какого-либо программного обеспечения на наблюдаемом хосте.
- Zabbix agent – может быть установлен на UNIX – подобных или Windows хостах для получения данных о нагрузке процессора, использования сети, дисковом пространстве и т. д.
- External check – выполнение внешних программ. ZABBIX также поддерживает мониторинг через SNMP.

Основными частями Zabbix являются: Zabbix сервер – ядро программного обеспечения, несет основную часть работы комплекса в целом; Zabbix прокси - собирает данные о производительности и доступности от имени Zabbix сервера; Zabbix агент - контроль локальных ресурсов и приложений (таких как жесткие диски, память, статистика процессора и т.д.) на сетевых системах; Веб-интерфейс - интерфейс является частью Zabbix сервера, работает на PHP.

К важным особенностям программы можно отнести следующие возможности:

- распределенный мониторинг вплоть до 1000 узлов;

- сценарии на основе мониторинга;
- автоматическое обнаружение;
- централизованный мониторинг log-файлов;
- web-интерфейс для администрирования и настройки;
- отчетность и тенденции;
- SLA мониторинг;
- комплексная реакция на события;
- поддержка SNMP v1, 2, 3;
- расширение за счет выполнения внешних программ;
- гибкая система шаблонов и групп;
- возможность создавать карты сетей;
- автоматическое обнаружение по диапазону IP-адресов, доступным сервисам и SNMP проверка;
- автоматический мониторинг обнаруженных устройств;
- автоматическое удаление отсутствующих хостов;
- распределение по группам и шаблонам в зависимости от возвращаемого результата.

Оповещение администратора о внештатном состоянии узлов происходит по CMC, ICQ, E-mail.

Важным моментом во всем этом является то, что система является свободно распространяемой, что существенно уменьшает затраты на весь комплекс.

Структура комплекса «Сокол»

Ввиду выше описанных проблем предлагается аппаратно-программный комплекс «Сокол». Данный комплекс разрабатывается с учетом всех недостатков каждой подобной системы, что позволяет ему стать лидирующим продуктом для решения подобного круга задач. Структурная схема комплекса представлена на рис. 1.

Аппаратная часть имеет блочную структуру.

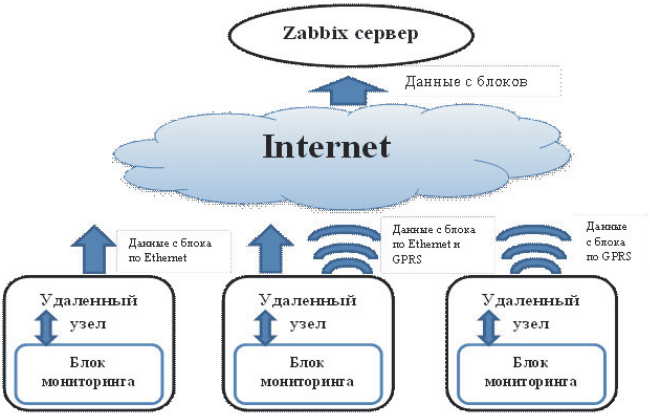


Рисунок 1. Структурная схема комплекса «Сокол».

Основной блок содержит интерфейс для подключения контактов датчиков типа «сухой контакт», интерфейс для подключения дополнительного блока, разъем для внешнего питания, тумблер включения/отключения питания. Внутри данного блока имеется резервное питание, управляющий контроллер. Он собираем всю информацию с внешних датчиков.

Дополнительный блок разрабатывается в трех исполнениях: с Ethernet контроллером и Ethernet портом соответственно, контроллером, передающим данные от управляющего контролера во внешний канал, интерфейсом для подключения к основному модулю; с GPRS модулем, контроллером передающим данные от управляющего контролера во внешний канал, интерфейсом для подключения к основному модулю; с Ethernet контроллером и Ethernet портом соответственно, с GPRS модулем, контроллером, передающим данные от управляющего контролера во внешний канал, интерфейсом для подключения к основному модулю.

При отключении внешнего питания модуль должен переходить на резервное питание. Основным каналом передачи данных является канал Ethernet, при отсутствии связи данные передаются по GPRS каналу.

Программная часть комплекса должна собирать данные с распределенных узлов, строить графики, систематизировать полученную информацию, оповещать администраторов о событиях на узлах.

Заключение

В итоге предлагается аппаратно-программный комплекс «Сокол». Комплекс предназначен для удаленного мониторинга распределенных узлов связи и антивандальных шкафов. Преимуществами комплекса являются дешевизна, всесторонний мониторинг, оповещение оператора в режиме реального времени, наличие нескольких каналов передачи данных.

ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ ПРОЕКТИРОВАНИЯ СКС ЗДАНИЯ

А.Е. Фролов, А.А. Соловьев

ФГБОУ ВПО Алтайский государственный университет

В настоящее время в нашей жизни вычислительной технике отводится большая роль, зачастую, в той или иной организации её становится настолько много, что она располагается в нескольких кабинетах, а в отдельных случаях занимает несколько этажей или даже зданий. Для облегчения работы необходимо создать структурированную кабельную систему (СКС), для объединения необходимой вычислительной техники в единую систему. Но, значительное число малых и средних компаний все еще приценяются к таким системам, сравнивая прогнозируемые выгоды от ее внедрения с финансовыми затратами на ее проектирование и установку. Для преодоления этого барьера необходимо создать программный продукт, который позволил бы лицам, не имеющим большого опыта в данной области, выполнять проектирование СКС предприятия, тем самым, сокращая расходы на заказ проектов у сторонних организаций.

Для решения поставленной задачи, создаваемый программный продукт должен строить модель здания: рисовать сети, размещать рабочие места, сетевое оборудование, розетки и т.д. Основной особенностью программного продукта будет возможность проектирования СКС в полуавтоматическом или автоматическом режиме с оптимизацией трасс, а также будет присутствовать меню для полного расчета затрат, себестоимости и количества необходимого оборудования. Так же предусмотрена обработка ошибок, чтобы соответствовать регламенту ГОСТ Р 53246-2008. Структурная схема предлагаемого программного продукта представлена на рис. 1.

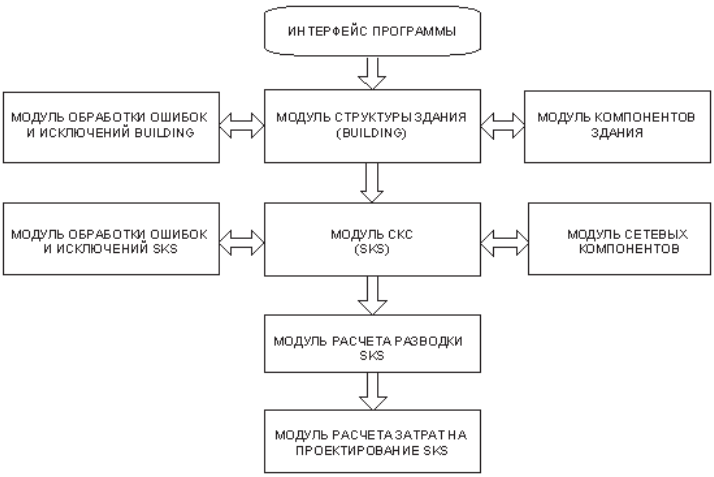


Рисунок 1. Структурная схема программного продукта СКС.

Как видно из рисунка программный продукт будет состоять из следующих модулей:

- интерфейс программы – отображение результатов работы всех остальных модулей;
- модуль структуры здания – построение модели здания, цепей питания и других объектов, которые будут присутствовать в здании, влияющие на проектирование СКС;
- модуль СКС – в этот модуль будет передаваться готовая модель здания, а после можно будет расставлять необходимые рабочие места и разводиться сети;
- модуль расчета разводки СКС – необходимые функции для автоматической разводки СКС;
- модуль расчета затрат на проектирование СКС - расчет количества кабеля, оборудования и т.д.

Модуль здания разделен на 2 подмодуля:

- модуль компонентов здания – основные компоненты здания: стены, лестницы, окна и т.д.;

- модуль обработки ошибок и исключений – основные требования по проектированию здания.

Исходные данные модуля здания будут передаваться в модуль СКС, который в свою очередь тоже разделен на 2 подмодуля:

- модуль сетевых компонентов – основные сетевые компоненты: розетки, кабели, сетевое оборудование и т.д.;
- модуль обработки ошибок и исключений – общие требования ГОСТ Р53246 - 2008.

Далее идет модуль расчета разводки СКС, в котором будет рассчитываться вся длина кабеля, количество необходимого оборудования и т.д. После идет модуль расчета стоимости затрат на проектирование СКС, в котором можно будет указать необходимые параметры для выяснения конечной стоимости затрат на реализацию конкретной СКС.

Предлагаемая модель программы позволит создавать 3D-модель помещений, давать возможность ручной и автоматической прокладки кабельной структуры, высчитывая оптимальный путь прокладки сети. Так же она будет иметь возможность расчета необходимого оборудования и кабеля. Будет ориентирована на малые, средние и большие организации и позволит сократить расходы на проектирование СКС.

Литература

1. http://www.intuit.ru/department/network/baslocnet/15/.
2. ГОСТ Р-53246-2008.
3. В. Олифер. Н Олифер. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 4-е изд. – СПб.: ПИТЕР, 2010.
4. Семенов А.Б. Проектирование и расчет структурных кабельных систем и их компонентов. – М.: ДМК Пресс; М.: АйТи, 2003.

РАСЧЕТ ПАРАМЕТРОВ ЭЛЕКТРОСТАТИЧЕСКОГО ПОЛЯ С ИСПОЛЬЗОВАНИЕМ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ

В.С. Афонин, Л.Э. Шейда

Алтайский государственный технический университет им. И.И. Ползунова

При расчете и описании режимов работы емкостных измерительных преобразователей часто возникает необходимость определении распределения электростатического поля в системе электродов с неоднородным диэлектриком. Типичной задачей является нахождение поля при неизвестном местоположении исходных зарядов, но заданном электрическом потенциале на границах области. В инженерной практике потенциалы электродов обычно задаются источниками питания и могут быть измерены или вычислены. Целью расчёта является нахождение потенциала φ и напряженности поля E по заданному расположению и форме заряженных тел – электродов – и граничным условиям. Такая задача называется прямой задачей.

Для описания электромагнитных полей в общем случае используется полная система уравнения Максвелла в дифференциальной форме [1]. Для электростатических полей, обусловленных действием неподвижных электрических зарядов, справедливы уравнения:

$$rot\,\overline{E} = 0;\ div\overline{D} = \rho;\ \overline{D} = \varepsilon\varepsilon_0\overline{E}.$$

Поля подобного типа являются безвихревыми, что позволяет исследовать их путём введения потенциальной функции φ, которая связанным с напряженностью $\overline{E}$. Потенциал позволяет свести сложные векторные уравнения Максвелла в дифференциальные уравнения в частных производных – уравнения Лапласа и Пуассона. В однородной среде для потенциала справедливо уравнение Пуассона[1-3]:

$$\Delta\varphi = \frac{\partial^2}{\partial x^2}\varphi + \frac{\partial^2}{\partial y^2}\varphi + \frac{\partial^2}{\partial z^2}\varphi = -\frac{\rho}{\varepsilon\varepsilon_0};\ \nabla^2\varphi = -\frac{\rho}{\varepsilon\varepsilon_0}$$

и, в частности, где отсутствуют свободные заряды, уравнение Лапласа:

$$\Delta\varphi = \frac{\partial^2}{\partial x^2}\varphi + \frac{\partial^2}{\partial y^2}\varphi + \frac{\partial^2}{\partial z^2}\varphi = 0\,;\ \nabla^2\varphi = 0\,.$$

Уравнение поля решается, как правило, аналитическими или численными методами. Во многих случаях точные аналитические решения получить крайне сложно или невозможно, численные методы позволяют рассчитать поле с любой конфигурацией электродов.

Метод конечной разности позволяет произвести замену дифференциальных уравнений конечно-разностными уравнениями. Для этого исследуемая область разбивается координатными линиями на некоторое количество ячеек. Каждая точка пересечения двух линий образует узел (рисунок 1).

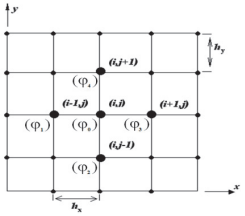


Рисунок 1 - Пятиточечная прямоугольная сетка

Потенциал в узле (i, j) сетки находится из выражения:

$$\phi_{i,j} = \frac{1}{4}(\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1})\,.$$

В начале расчета произвольно задаются начальные значения потенциала во всех внутренних узлах сетки. Затем по уравнению в каждом из узлов эти значения потенциала пересчитываются в новые. Найденные таким образом значения еще раз подставляются в уравнение, и так до тех пор, пока изменения потенциала от итерации к итерации в каждом из узлов не будут меньше некоторого заданного уровня. При этом во всех узлах, расположенных на границах, потенциал на каждой итерации задается постоянным, что гарантирует выполнение граничных ус-

ловий. Такой метод достаточно часто применяется для решения систем линейных алгебраических уравнений и носит название метода Якоби.

Применяя локальные обозначения узлов пятиточечной сетки итерационный процесс Якоби можно представить в виде

$$\phi_0^{(p+1)} = \frac{1}{4}(\phi_1^{(p+1)} + \phi_2^{(p+1)} + \phi_3^{(p)} + \phi_4^{(p)}).$$

Среди итерационных процедур метод последовательной верхней релаксации сходится наиболее быстро. Метод использует определенное количество итераций для определения потенциала во всех точках заданной области. Каждая последующая итерация использует данные, полученные предыдущего расчета. Поэтому все перерасчеты потенциалов производятся последовательно, однако расчет потенциалов в узлах соседних строк можно выполнять параллельно. Потенциал каждого узла рассчитывается относительно предыдущего значения и четырех потенциалов соседних точек $\varphi_1,\varphi_2,\varphi_3,\varphi_4$. Причем потенциалы двух точек известны с предыдущего расчета, а потенциалы точек 1 и 2 должны быть вычислены заранее. Таким образом, вычислительный процесс строки i+1 должен отставать от процесса вычисления строки i на один столбец j. Тогда количество вычислительных процессов равняется количеству строк в матрицы узловых потенциалов, что значительно сократит время вычисление поля. Для ускорения вычислений результирующей матрицы потенциалов необходимо производить параллельные вычисления на независимых процессорах.

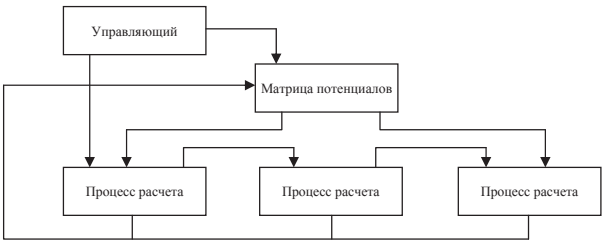


Рисунок 2 - Схема реализации параллельных вычислений

Схема вычисления представлена на рисунке 2. Управляющий процесс формирует матрицу потенциалов на основе заданных граничных условий и запускает процесс вычисления потенциалов первой строки. После вычисления первого значения запускается процесс вычисления 2-ой строки, и т.д. Каждый процесс выполняется заданное количество раз. Такой подход значительно сокращает время вычисления распределения потенциала электростатического поля в исследуемой системе.

Литература

1. Исаев Ю. Н. Современные пакеты программ для расчетов и моделирования электромагнитных полей [Электронный ресурс] / Ю. Н. Исаев // Корпоративный портал Томский политехнический университет. - Режим доступа: http://portal.tpu.ru/SHARED/i/ISAEV/Job/Tab/Pole.pdf. - Загл. с экрана.;

2. Зайцев, В.В. Электростатическое моделирование полосковых линий: Учебное пособие [Электронный ресурс] / В.В.Зайцев, В.И. Занин, В.М. Трещев // Единое окно доступа к образовательным ресурсам. – Самара: Изд-во «Универс-групп», 2005. - Режим доступа: http://window.edu.ru/window/catalog?p_mode=1&p_rid=74923. - Загл. с экрана.;

3. Борисов П.А. Потенциальные электрические поля. Учебное пособие по курсам ТОЭ (часть вторая). - Теория электромагнитного поля. Электромагнитные поля и волны [Электронный ресурс] / П.А. Борисов, Ю.М. Осипов // Единое окно доступа к образовательным ресурсам. – СПб: СПб ГУИТМО, 2006. - Режим доступа: http://window.edu.ru/window/library?p_rid=27829. - Загл. с экрана.

ПРИМЕНЕНИЕ МИКРОКОНТРОЛЛЕРА LPC2103 В УСТРОЙСТВАХ ЗАЩИТЫ ИНФОРМАЦИИ

А.П. Иванов, М.С. Тикин

ФГБОУ ВПО «Пензенский государственный университет»

Факультет приборостроения, информационных технологий и систем

В настоящее время трудно переоценить значение средств связи всех видов. Разработка аппаратуры связи сопряжена с необходимостью обеспечения надежности функционирования её аппаратных и программных компонентов в различных режимах и условиях. Одной из задач, стоящих перед разработчиками, является создание методов контроля работоспособности отдельных узлов аппаратуры и их реализация. Примером решения данной задачи является прибор КП-ПА671, разработанный для проверки работоспособности плат ПА-671, входящих в состав коммутаторов защищенной связи. Прибор контролирует функционирование цепей питания и запоминающих устройств плат ПА-671 путём сравнения отправленных на них информационных кодов со считанными впоследствии.

Основой прибора является микроконтроллер LPC2103 фирмы NXP. Микроконтроллер имеет процессорное ядро ARM7TDMI-S, позволяющее сочетать в приборах на базе LPC2103 небольшие размеры и сравнительно низкое энергопотребление. Эти достоинства имеют неоспоримо высокое значение для применения в современных компактных устройствах. Микроконтроллеры LCP2103 успешно используются в системах управления доступом. Наличие нескольких последовательных коммуникационных интерфейсов и оперативного запоминающего устройства объёмом до 64 кбайт позволяют использовать микроконтроллеры данного семейства в сетевых шлюзах и преобразователях протоколов, программных модемах, устройствах распознавания речи и устройствах обработки графической информации. Благодаря различным 32-разрядным таймерам, ШИМ-каналам и 32 линиям ввода/вывода общего назначения эти микроконтроллеры подходят

для применения в оборудовании управления производственными процессами и медицинском оборудовании [1].

В приборе КП-ПА671 применена макетная плата LPC-H2103, выполненная в форм-факторе DIL40 фирмой OLIMEX. Установленный в центре неё микроконтроллер LPC2103FBD48 имеет:

- 32 кб Flash-памяти программ;
- 8 кб ОЗУ;
- часы реального времени;
- два порта UART;
- два порта I2C;
- пять 32-битных таймеров;
- ШИМ;
- сторожевой таймер;
- 5В-совместимые входы/выходы;
- восемь АЦП по 10 бит.

Помимо микроконтроллера макетная плата содержит:

- 14,7456 МГц кварцевый резонатор;
- пользовательскую кнопку;
- светодиод состояния.

Компактность платы (её размеры составляют 51×20мм) является основным фактором выбора её для использования в приборе КП-ПА671. Данное испытательное оборудование используется для проверки и приемки плат ПА671 по разработанным производителем методикам и инструкциям. Помимо того, прибор КП-ПА671 имеет технологический режим для проверки собственной исправности. При включении технологического режима производится вывод тестовых сигналов на внешние разъемы прибора.

Прибор имеет несколько тумблеров для установки режимов работы и проверяемых адресов платы ПА671, гнезда для подключения средств измерений при

проверке сопротивления сигнальных цепей ПА671 и соединители для подключения прибора к плате.

Таким образом, разработка программного обеспечения прибора КП-ПА671 сводится к задаче алгоритмизации и реализации на языке Си считывания и установки состояний портов микроконтроллера LPC2103 в дискретные моменты времени.

Язык Си известен как универсальный язык программирования с компактным способом записи выражений, современными механизмами управления структурами данных и богатым набором операторов. Благодаря широким возможностям и универсальности для решения многих задач он удобнее и эффективнее, чем предположительно более мощные языки [2]. Так как относительно прибора КП-ПА671 быстродействие не является критичной характеристикой, язык Си имеет преимущество и перед языками ассемблера в виде простоты использования.

К примеру, для работы с портами ввода/вывода следует объявить их командой IODIR:

IODIR1 = 0x04100000,

где 0x04100000 = 00000100000100000000000000000000b означает, что двадцатая и двадцать шестая линии используются для вывода, а все остальные – для ввода.

Для установки любой линии вывода первого порта в состояние единицы используется команда IOSET1, а для сброса в состояние нуля – IOCLR1 [3]. К примеру, сброс в ноль двадцатой шестой линии ввода/вывода (включение сигнальной лампы на лицевой панели прибора) выполняется командой:

IOCLR1 = 1 << 26.

Для выдачи сигнала на внешний разъём с частотой около 1 кГц во время активного технологического режима разработана функция задержки, представляющая собой цикл на 4802 итерации, который выполняется 522 микросекунды. Таким образом, эта функция выполняется один раз, после чего производится ин-

вертирование уровня сигнала на необходимом выводе и функция выполняется вновь.

Мигание лампы с частотой 1 Гц в технологическом режиме осуществляется отсчитыванием количества импульсов на внешний разъём. После прохождения 478 импульсов состояние лампы меняется на противоположное, так достигается необходимая частота с допустимой погрешностью.

Важным фактором при считывании состояния тумблеров остается дребезг контактов. Для защиты от него введено дублирование функции: состояние тумблера считывается дважды, с разницей в 300 миллисекунд. При этом, во избежание лишних задержек, повторное считывание производится только в случае, когда первая проверка состояния тумблера показала изменение. Если же состояние тумблера не изменялось, то дребезга не возникает и дублирующее чтение не применяется.

В режиме проверки плат ПА671 работа прибора заключается в аппаратной записи адресной информации, введенной посредством тумблеров, в запоминающее устройство платы, и программное считывание этой информации из платы с последующим сравнением с оригиналом. Данные действия, выполняемые циклически посредством периодической выдачи разрешающих сигналов чтения и сигнала синхронизации, осуществляются описанными выше командами IOSET1 и IOCLR1. После получения состояния тумблеров и адресной информации из платы производится их сравнение, в случае совпадения проверяемый канал считается исправным и контрольная лампа на лицевой панели непрерывно горит. В случае несовпадения канал считается неисправным и контрольная лампа гасится.

Описанные приемы программирования микроконтроллера LPC2103 показывают простоту и удобство его использования в широком круге задач по защите информации, в которые входит первичная аттестация и постоянный контроль работоспособности аппаратуры защищенной связи. Существовавшая проблема разработки и реализации метода контроля работоспособности успешно решена.

Литература

1. Пантелейчук А. Обзор популярного семейства LPC210x недорогих микроконтроллеров от NXP. // Журнал «Компоненты и технологии» №12 – 2007.
2. Керниган Б.У., Ритчи Д.М. Язык программирования С, 2-е издание. : Пер. с англ. – М. : Издательский дом «Вильямс», 2011.
3. Мартин Т. Микроконтроллеры ARM7 семейств LPC2300/2400. Вводный курс разработчика. : Пер. с англ. Евстифеева А.В. – М. : Додэка-XXI, 2010.

ОСОБЕННОСТИ СЕТЕЙ ETHERNET ДЛЯ РАСПРЕДЕЛЁННЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

С.Д. Цветков, П.М.Зацепин, Ю.Я. Матющенко

ФГБОУ ВПО Алтайский государственный университет

Технология Ethernet изначально предназначенная для внутриофисных коммуникаций, во многом, благодаря дешевизне, простоте и распространённости поступенно заняла доминирующее положения почти во всех нишах рынка сетевых интерфейсов, включая, глобальные сети и высокопроизводительные вычисления. Важность сетевой среды для распределённых вычислений сложно переоценить, по информации некоторых источников некорректная работа сети повлияла на результат недавних измерений скорости нейтрино, произведя сомнительную сенсацию о «сверхсветовой скорости» этих частиц.

Распределённые вычисления, по Флинну могут принадлежать к одному из классов:

- MIMD – параллельные вычисления, высокопроизводительные вычисления, информационные системы;
- SIMD – параллельные вычисления (с однородными вычислителями), информационные системы;
- MISD – конвейерные вычисления, телекоммуникационные системы.

Современный этап развития технологии Ethernet позволяет эффективно применять её в любых типах информационных систем, включая системы параллельного и распределённого вычислений. Являясь, связующим звеном между вычислительными модулями, любая сеть имеет свои особенности, характерных для сетей с пакетной коммутацией:

- статистическое разделение полосы пропускания;
- время коммутации пакета сравнимо со временем его передачи;

- невозможность гарантировать определённую полосу передачи между узлами в каждый момент времени;
- негарантированное время задержки при передаче данных;
- большие накладные расходы при передаче пакетов малого размера;
- приоритет при передаче данных указывается относительно, нет средств выделения гарантированной полосы пропускания;
- отсутствуют кольцевые структуры (на физическом уровне сеть может выглядеть как кольцо, но на канальном уровне кольцо, в любом случае, должно быть разорвано);
- поддержка многоадресных (точка-многоточие) рассылок;
- Ethernet предоставляет только коммуникационную среду, все пользовательские функции вынесены на вышестоящие уровни.

Отдельно стоит выделить проблему падения производительности при уменьшении размера передаваемого сообщения:

- во-первых, каждый пакет, кроме данных несёт ещё и заголовки, накладные расходы, связанные с передачей могут достигать 50%;
- во-вторых, коммутационное оборудование так же имеет характеристику по производительности именно в пакетах, обрабатываемых за секунду времени, что может быть весьма существенно;
- в третьих, время коммутации пакета на скорости для технологии GE уже сопоставимо со временем его передачи, на скорости 10GE передача занимает существенно меньшее время.

На рисунке приведена зависимость эффективности использования полосы пропускания от размера пакета в наилучшем случае (Ethernet-кадр (802.3) + IPv4 + UDP)

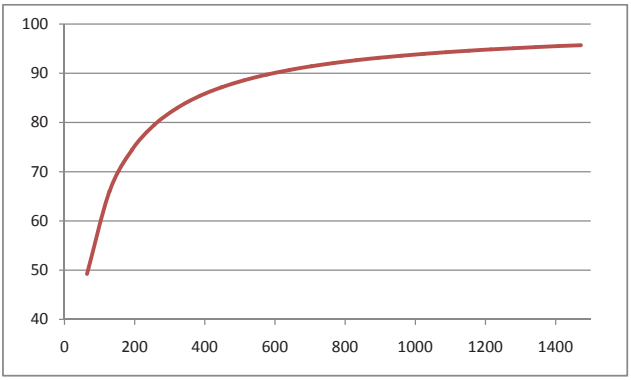


Рисунок 1. Зависимость эффективности использования полосы пропускания (по оси Y в %) от размера полезной нагрузки (по оси X).

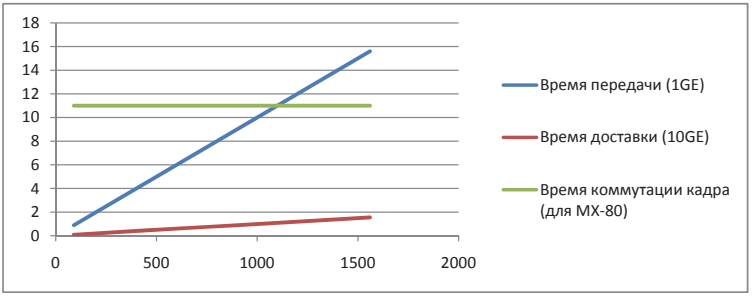


Рисунок 2. Сравнение времени коммутации кадра (из документации к Juniper MX-80) и времени передачи кадра различной длины для 1GE и 10 GE. Цифры указаны в мкс.

Системы распределённых вычислений, в свою очередь, так же имеют ряд особенностей:

- большие задержки при передаче данных между вычислителями;
- большая и неравномерная во времени нагрузка на сеть;
- вычислительная сеть может быть разного размера (от единиц до нескольких сотен устройств);

- топология распределённой системы может быть абсолютно различной – вычислители РС могут находиться в одной стойке, а могут быть разнесены по разным континентам;
- большинство средств распределённых вычислений, в случае с Ethernet-сетью, работает «поверх» вышестоящих сетевых протоколов (стек TCP/IP, прикладная часть протокола).

Тем не менее, у всех подобные ИС есть общие черты, которые можно и нужно учитывать, при построении сетей коммутации:

- причина, по которой требуется распределение вычислений (хорошая распараллеливаемость задачи, требование обрабатывать данные как можно ближе точке их получения, необходимость территориального разнесения узлов ВС для устойчивости к катастрофам);
- преобладающий тип обмена информацией между узлами системы;
- определить критичность ко времени передачи, задержкам, вариации задержки, полосе пропускания и т.п;

Системы распределённых вычислений характеризуются большим объёмом данных между вычислительными узлами, высокими требованиями к задержкам и полосе пропускания. Располагаются, как правило, компактно (в пределах здания). Характерно большое количество «мелких» - менее 1 КБ пакетов с данными, производительность системы сильно зависит от времени ожидания передачи. Большой объём передаваемых данных может оказывать негативное влияние на вспомогательное компоненты ВС, такие как управление узлами, средства ввода-вывода на накопители и т.п.

Для подобных систем особенностью сетей можно считать сопоставимое время коммутации и передачи кадра (при использовании Gigabit Ethernet в качестве сети) и несопоставимо большое, по сравнению со временем передачи время коммутации для 10GE.

Особенности архитектуры систем класса x86 (x86_64), отражаются на производительности вычислительной системы: рост количества пакетов и большая

интенсивность обмена данными (порядка нескольких Гбит/с на современных вычислительных машинах с применением специализированных адаптеров) вызывает рост количества прерываний, в результате, время непосредственно вычислений нелинейно снижается с ростом нагрузки, так как большое количество процессорного времени может тратиться на обработку прерываний. Для решения указанной проблемы разработаны версии драйверов сетевых адаптеров, поддерживающий режим «опроса», однако использование этого режима, хотя и значительно уменьшает количество прерываний, так же значительно повышает задержку передачи сообщения (до нескольких сотен микросекунд, в зависимости от настроек). Накладные расходы по вычислительному времени на расчёт контрольных сумм, проверку заголовков и т.п. так же становятся достаточно высокими при увеличении нагрузки.

Высокие требования к сетям параллельных вычислений обуславливают целый ряд особенностей настройки коммутационного оборудования. Современные коммутаторы – это сложный программно-аппаратный комплекс, где часть функций реализована на специализированной логике (ASIC), а часть – на специализированных процессорах. Ядром современного коммутатора является одна или несколько микросхем тернарной памяти, причём за одну операцию может быть проверено одно или несколько совпадений (правила коммутации и фильтрации). Типичное время обращения к подобной логике составляет несколько десятков микросекунд, а лишнее правило обработки пакета может привести к появлению лишних запросов и значительно увеличить время коммутации.

Традиционно, сети Ethernet строятся по топологии «звезда» или «дерево», причём все, присутствующие на рынке решения по построению других топологий (кольца, решётки, так, или иначе, предполагают разрыв кольца, удаление лишних граней из решётки и т.п. Следовательно, при построении Ethernet-сети для параллельных вычислений нужно грамотно планировать сетевую топологию. В сетях параллельных вычислений, традиционно сильное положение занимает топология типа «N-мерный тор», связанно это с тем, что именно такая топология

обеспечивает наибольшую эффективность при решении основной массы задач параллельных вычислений (моделирование, матричные операции и т.п.).

Частым требованием к сетям для параллельных вычислений – обеспечение групповых операций – широковещательных рассылок одного набора данных группе узлов, распределения данных по множеству узлов, сбор данных от множества узлов. Среда Ethernet, по своей природе являющаяся широковещательной, позволяет производить подобные операции достаточно эффективно, однако, распространение широковещательного трафика нужно планировать. Средства типа IGMP Snooping подобное планирование позволяют, однако для их полноценной реализации могут понадобиться средства вышестоящих сетевых уровней, особенно если речь идёт о построении сетей на основе не специализированных компонентов.

Ethernet, в отличие от специализированных сетевых сред, типа InfiniBand не имеет встроенных средств для организации типичных задач сетевой среды параллельных вычислений:

- передача сообщений (в кадр Ethernet должен быть инкапсулирован пакет вышестоящего протокола);
- доступ к устройствам хранения (через средства вышестоящих протоколов, типа ATAoE, iSCSI).

Средства вышестоящих сетевых протоколов могут создать дополнительную нагрузку на вычислительные узлы.

Заключение

Как и любое другое сложное устройство, сеть для распределённых вычислений является компромиссом, между пожеланиями и финансовыми возможностями конечного пользователя, относительно к уровню научно-технического прогресса. Являясь универсальной технологией Ethernet, безусловно, проигрывает по своим техническим характеристикам специализированным интерфейсам, таким как InfiniBand.

Изначальная простота и низкая специализированность Ethernet-сетей, в свою очередь позволяют разрабатывать решения, предназначенные для решения практически любых задач. Целый стек смежных технологий, во многом, позволяет строить простые и экономически эффективные сети для высокопроизводительных и распределённых вычислений, с оптимальным балансом стоимости и производительности, а учёт особенностей среды позволяет избежать накладных расходов по вычислительным ресурсам, программному и аппаратному обеспечению.

Литература

1. Эндрю Таненбаум, Мартин ван Стеен Распределенные системы. Принципы и парадигмы = Andrew S. Tanenbaum, Maarten van Steen. "Destributed systems. Principles and paradigms". — Санкт-Петербург: Питер, 2003. — 877 с. — (Классика computer science). — ISBN 5-272-00053-6
2. Пятибратов, А.П.; Гудыно, Л.П.; Кириченко, А.А. Вычислительные системы, сети и телекоммуникации; М.: Финансы и статистика; Издание 2-е, перераб. и доп., 2004. - 512 с.
3. Juniper networks technical documentation [Электронный ресурс] / Режим доступа:http://www.juniper.net/techpubs/

Подписано в печать 27.02.2012 г.
Формат 60х84 $^{1}/_{16}$. Бумага офсетная.
Печать цифровая. Гарнитура «Times».
Усл. печ. л. 12,21
Заказ №.... Тираж 60 экз.

Отпечатано в типографии
ООО «Издательский дом «Барнаул»,
г. Барнаул, ул. Чеглецова 3-а.
Тел. (3852) 35-77-55, 22-93-39.
E-mail: idbarnaul@mail.ru