Постановка задачи программирования многоядерной потоковой рекуррентной архитектуры

Д.В. Хилько
Учреждение российской академии наук Институт Проблем Информатики Российской Академии Наук (ИПИ РАН)

Аннотация: В статье рассматриваются некоторые особенности многоядерной потоковой рекуррентной архитектуры (МПРА). Приведены основные методологии программирования систем нетрадиционной архитектуры. Поставлены проблемы программирования МПРА и обозначены пути их решения.

Введение

В Институте Проблем Информатики Российской академии наук (ИПИ РАН) ведутся работы по созданию нетрадиционной рекуррентной архитектуры, предназначенной для реализации параллельных вычислений ограниченной размерности в области сигнальной обработки. Для экспериментальной апробации предлагаемой архитектуры разрабатывается рекуррентный обработчик сигналов (РОС), исполняемый 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].

Основная особенность разрабатываемой архитектуры - это способ представления потоков данных и команд, объединенные в единый поток, который рекуррентно сворачивается при подготовке и разворачивается в ходе исполнения. Данное представление, по мнению разработчиков, позволяет эффективно реализовать рекуррентные и рекурсивные вычисления, а также решить другие проблемы, которые свойственны потоковой и параллельной архитектурам.

Для достижения максимальной производительности разрабатываемой архитектуры необходимо разработать соответствующее программное обеспечение, оптимизированное для выполнения в ее среде. Специфические особенности архитектуры, не свойственные другим классам, делают не возможным применение в полном объеме известных методов и технологий программирования. Желаемого результата можно достичь, объединив наиболее сильные стороны существующих методологий и подходов программирования, дополнив их новыми методами, которые будут эффективны в среде разрабатываемой архитектуры.

1. Существующие методологии программирования систем нетрадиционной архитектуры

На сегодняшний день существует множество методологий программирования, ориентированных на системы с нетрадиционной архитектурой. Большинство из них использует императивную, декларативную и потока данных парадигмы [2]. Наиболее известными технологиями, основанными на императивной парадигме программирования, являются:

1) Технология OpenMP.

Данная технология ориентирована на параллельные вычислительные системы с общей памятью. На сегодняшний день она является одним из наиболее популярных средств программирования систем, построенных на подобных принципах. В качестве отправной точки берется последовательная программа. Для создания ее параллельной версии пользователю предоставляется набор директив, переменных окружения и процедур. Стандарт OpenMP разработан для языков Fortran (77, 90, 95), C и C++ и поддерживается практически всеми производителями больших вычислительных систем.

На рис. 1. можно увидеть процесс исполнения программы согласно стандарту OpenMP. Весь текст программы разбит на последовательные и параллельные области.

Рис. 1. OpenMP: процесс исполнения программы

Рассмотренные понятия областей программы и классов переменных определяют общую идею написания параллельной программы в рамках технологии OpenMP. Можно отметить два основных преимущества технологии OpenMP. Во-первых, технология изначально спроектирована таким образом, чтобы пользователь мог работать с единым текстом параллельной и последовательной программ. Другим достоинством OpenMP является возможность постепенного «инкрементного» распараллеливания программы.

2) Система программирования DVM.

DVM-система состоит из пяти основных компонентов: компиляторы с языков Fortran-DVM, C-DVM, система поддержки выполнения параллельных программ, отладчик параллельных программ, анализатор производительности,

предсказатель производительности. При проектировании данной системы авторы опирались на следующие принципы:

- Система должна базироваться на высокоуровневой модели выполнения параллельной программы, удобной и понятной для программиста, привыкшего программировать на последовательных языках.
- Спецификации параллелизма (правила параллельного выполнения алгоритма) должны быть прозрачными для обычных компиляторов.
- Языки параллельного программирования должны представлять собой традиционные языки последовательного программирования, расширенные спецификациями параллелизма. Эти языки должны предлагать программисту модель программирования, близкую к модели выполнения.
- Основная работа по реализации модели выполнения параллельной программы (например, распределение данных и вычислений) должна осуществляться динамически системой поддержки выполнения DVM-программ.

В любой DVM-программе могут быть использованы два уровня параллелизма. На верхнем уровне в программе описывается какое-то число независимых ветвей (задач), которые могут выполняться параллельно. Задачи DVM - это независимые по данным крупные блоки программы. В конце ветвей допускается выполнение глобальной редукционной операции. В рамках каждой ветви могут дополнительно выделяться параллельные циклы. Никакой другой иерархии параллелизма DVM не допускает, и описать в теле параллельного цикла еще несколько независимых ветвей нельзя.

1) Система программирования mpC.

Язык mpC является одним из первых языков высокого уровня, разработанный специально для программирования неоднородных сетей. Язык позволяет программисту определить все основные свойства и параметры параллельного алгоритма, влияющие на скорость его выполнения в неоднородной вычислительной среде. Также трС позволяет изменять эти характеристики динамически во время исполнения программы. Информация, извлеченная из описания параллельного алгоритма, вместе с данными о реальной производительности процессоров и коммуникационных каналов, помогает системе mpC найти эффективный способ отображения процессов программы на компьютеры сети.

2) Коммуникационные средства (MPI, PVM).

Наиболее распространенной технологией программирования параллельных компьютеров с распределенной памятью в настоящее время является Message passing Interface (MPI). Основным способом взаимодействия в таких системах

является передача сообщений (отсюда и название - коммуникационные средства).

MPI-программа - это множество параллельных взаимодействующих процессов. Все процессы порождаются один раз, образуя параллельную часть программы. В ходе выполнения программы порождение новых или уничтожение старых процессов не допускается. Каждый процесс работает в своем адресном пространстве, никаких общих данных или переменных в MPI нет. Допускается создание групп процессов произвольного состава для локализации их взаимодействия, предоставляя им среду для общения - коммуникатор. При старте программы считается, что все процессы работают в рамках всеобъемлющего коммутатора.

Общение процессов осуществляется посредством передачи сообщений. Сообщение - это набор данных некоторого типа. Каждое сообщение имеет несколько атрибутов, позволяющих идентифицировать источник, приемник, а также отличать сообщения между собой. Существуют некоторые ограничения на типы передаваемых в сообщениях данных. Чтобы снять эти ограничения, в MPI предусмотрен механизм для введения производных типов данных. Описав состав и схему размещения в памяти посылаемых данных, пользователь в дальнейшем с такими типами, как со стандартными.

Развитие идей параллельного программирования в среде функциональных языков программирования нашло свое отражение в таких системах как: Sisal, Haskell, Cilk, Т-система, НОРМА и др.

1) Т-система

Т-система представляет собой технологию автоматического динамического распараллеливания программ. Разработка этой системы была начата в Институте программных систем РАН в конце 80-х годов XX века.

Т-система использует парадигму функционального программирования для обеспечения динамического распараллеливания программ. На этой основе удалось найти и реализовать в Т-системе нестандартные формы организации параллельных вычислений, в частности, для синхронизации или распределения нагрузки. Кроме того, функциональный стиль Т-системы был совмещен с традиционными языками программирования с помощью расширений языков С и С++. Явные параллельные конструкции в языке отсутствуют. Они определяются из базового свойства «чистоты» функций.

2) Sisal

Заметное место среди языков функционального программирования занимают языки параллельного программирования. Довольно известен язык функционального программирования Sisal. Название языка расшифровывается как “Streams and Iterations in a Single Assignment Language”, он представляет собой дальнейшее развития языка VAL, получившего распространение в середине 70-х годов XX века. Среди целей разработки языка Sisal следует

отметить наиболее характерные, связанные с функциональным стилем программирования:

- создание универсального функционального языка.
- разработка техники оптимизации для высокоэффективных параллельных программ.
- достижение эффективности исполнения, сравнимой с императивными языками типа Fortran и C.
- внедрение функционального стиля программирования для больших научных программ.
- Пользователь языка Sisal получает возможность сконцентрироваться на конструировании алгоритмов и разработке программ в терминах крупноблочных и регулярно организованных построений, опираясь на естественный параллелизм уровня постановки задачи.

Несомненным преимуществом языка Sisal над уже рассмотренными является то, что он разрабатывался как язык параллельного программирования и использует скрытый параллелизм непосредственно. Кроме того, компилятор языка OSC позволяет не только компилировать и исполнять программы с различной степенью параллелизма, но и строить текстовые графы промежуточных представлений, отражающие как чистую, математическую параллельность, так и указанную вручную.

2. Основные особенности МПРА

Для существующих традиционных и нетрадиционных компьютерных архитектур характерно наличие двух потоков: активного потока инструкций и пассивного потока данных. В РОУ оба потока сливаются в один общий поток самодостаточных данных, в котором, помимо собственно обрабатываемых данных, содержится и необходимая для их обработки управляющая и служебная информация (в существующих архитектурах кодируемая в форме инструкций), которая представлена в виде специальных теговых полей. Теги содержат некоторую начальную сжатую информацию, обеспечивающую выполнение требуемой процедуры. Каждый следующий шаг процедуры рекуррентно самоопределяется, в том числе с учетом результата предыдущего шага.

В текущем исполнении модель РОС имеет четыре процессорных ядра (ПЯ), способных функционировать параллельно. Таким образом, можно говорить о параллелизме на уровне ПЯ. Однако, это не единственный уровень параллелизма, задействованный в РОС, Специфика исполнения архитектуры позволяет выделить достаточно глубокий конвейер, каждая из ступеней которого может функционировать независимо. Также можно выделить параллелизм на самом низком уровне - вычислительных блоках и регистрах. В настоящем исполнении модель РОС поддерживает несколько разновидностей суперскалярных режимов вычислений [3].

Специфические особенности МПРА выражаются также и в способе представления программного обеспечения. Гибридность архитектуры РОС определяет гибридность программ, функционирующих в его среде. Часть решаемой задачи описывается на языке высокого уровня и исполняется на УУ, другая часть - на РОУ. Самодостаточность данных определяет структуру программ, функционирующих в среде РОУ. Этот особый вид программы называется *капсула.*

3. Проблемы программирования МПРА

Разрабатываемая архитектура базируется на новой вычислительной парадигме, названной рекуррентной [4].В основе рекуррентных вычислений лежит теория рекурсивных функций. В общем случае рекуррентная формула имеет вид

$$x\{t+1\} = F[x(t)] \qquad (1)$$

Другими словами значение функции на t+1-ом шаге получается воздействием оператора F на значение, полученное на шаге t. Использование рекурсии можно наблюдать повсеместно, особенно это стало актуально при использовании хвостовой рекурсии и позволило существенно сократить затраты памяти.

Однако общие для всех архитектур проблемы организации рекуррентных вычислений остаются. К ним можно отнести:

- Необходимость хранить историю вычислений предыдущих шагов алгоритма;
- Явная зависимость по данным, не позволяющая в полной мере использовать потенциал параллельных и потоковых вычислений;
- Часто рекуррентные вычисления используют в разделах численных методов. Это означает, что заранее предсказать количество шагов, которое необходимо сделать для получения достаточного приближения не известно;
- Балансировка вычислительной нагрузки на параллельные процессы/потоки.

В различных архитектурах были сделаны определенные попытки решения данных проблем с переменным успехом. Однако в целом проблемы остаются нерешенными. Одним из приоритетных направлений развития МПРА является эффективная организация рекуррентных вычислений.

Применительно к МПРА также важными проблемами программирования являются:

- Гибридное представление архитектуры, определяющее гибридную структуру программы.
- Наличие большого количества сложных внутренних механизмов, оказывающих прямое воздействие на структуру программ.

- Отсутствие методологической и технологической базы разработки программного обеспечения МПРА.

3. Возможные пути решения проблем программирования

Существующие подходы и языки программирования не могут быть в полной мере применены для эффективного программирования РОС. Это наталкивает на мысль о заимствовании сильных сторон того или иного подхода в программировании РОС.

В результате анализа существующих подходов программирования [5], в качестве отправной точки, был выбран функциональный язык параллельного и потокового программирования Sisal, который является развитием языка VAL. Результатом компиляции Sisal-программы является не только оптимизированный под параметры параллельной системы объектный код, но и мультиграф вычислительного процесса.

Развитие точных методов в программировании привело к возникновению различных формальных моделей программ, в том числе и моделей параллельных программ. Среди них можно выделить: операторные схемы программ, сети Петри, UCLA-графы и др.[6]. Эти модели позволяют представить алгоритм в виде графовых структур, демонстрирующих последовательные и параллельные участки как вычислительного, так и управляющего процессов.

Исполняемая на РОУ программа представляется в виде капсулы. Выявление параллелизма и построение графа вычислительного процесса являются необходимыми, но не достаточными условиями написания капсулы. Необходимо также корректно отобразить управляющие процессы, с учетом специфичности архитектуры РОС. Развитием идеи моделей параллельных программ в рамках данной работы - стала разработка внутреннего, графодинамического представления алгоритмов.

Одной из важнейших задач является создание методологической базы разработки программного обеспечения МПРА. На текущем этапе эта база находится в стадии разработки, однако, ее обобщенное представление может быть следующее.

Обобщенная методология разработки ПО для РОУ

Реализуемая средствами МПРА задача будет подразделяться на два основных блока: блок сложных вычислений (параллельная и потоковая обработка) и управляющий блок.

В рамках предлагаемой методологии необходимо выполнить следующее:

1) Провести анализ задачи и определить, имеется ли необходимость производить сложные и ресурсоемкие вычисления, и какую долю (приблизительно) в решении задачи они составляют

2) Если имеют место сложные вычислительные задачи и их доля достаточно велика, то декомпозировать исходную задачу на рад подзадач по принципу:

- Множество подзадач управления вычислительным процессом
- Множество подзадач низкой вычислительной сложности
- Множество подзадач высокой вычислительной сложности

3) Первые два класса задач необходимо решать при помощи верхнего (управляющего) уровня МПРА, третий класс - необходимо решать при помощи РОУ.

Критерии принятия решения о реализации той или иной задачи средствами МПРА, оценки сложности подзадач необходимо четко определить. На текущем этапе разработки МПРА, в качестве основных критериев оценки сложности задачи (подзадачи), были приняты следующие:

- Большой объем входного набора (потока) данных (например, цифровые фильтры, свертки и др);
- Скрытый или явный параллелизм вычислительного графа задачи степени 3 или более;
- Большой объем потока промежуточных данных;
- Большой объем выходного набора (потока) данных.

Данный список критериев не претендует на полноту. Он требует конкретизации, дополнения и коррекции.

Сам же процесс разработки капсул в общем виде можно представить в виде цепочки преобразования математической постановки задачи к виду вычислительного параллельного графа. Над этим графом необходимо провести соответствующие преобразования, которые позволят превратить его в рекуррентно-сворачиваемый динамический граф. Конечное формальное представление этого свернутого графа и будет прототипом капсулы.

С целью автоматизации разработки ПО для МПРА ведутся работы по созданию имитационных моделей исследуемой архитектуры, а также необходимый набор лингвистических средств, включающий в себя языки программирования (промежуточные и высокого уровня) и компилятор.

4. Заключение

В ходе исследования и решения проблем программирования были достигнуты следующие результаты:

- подход к программированию МПРА является комбинацией сильных сторон существующих парадигм;
- среди императивных методологий наиболее подходящей является - MPI;
- среди декларативных языков наиболее подходящим является Sisal;

- графодинамическое представление алгоритмов должно быть неотъемлемой частью процесса разработки программ в среде МПРА;
- создана обобщенная методология программирования МПРА;
- создаются программные средства автоматизации процесса разработки ПО.

Накопленные результаты позволят приблизиться к решению поставленных задач и весь разрабатываемый инструментарий войдет в состав системы капсульного программирования и отладки СКАТ [7].

Литература

1. *Степченков Ю.А., Петрухин В.С.* Особенности гибридного варианта реализации на ПЛИС рекуррентного обработчика сигналов // Системы и средства информатики: Доп. Вып. - М.: ИЛИ РАН, 2008. - С. 118-129.
2. *Воеводин В.В., Воеводин Вл. В.* Параллельные вычисления. - СПБ.: БХВ-Петербург, 2002. - 608 с.: ил.
3. *Степченков Ю.А., Волчек В.Н., Петрухин В.С., Прокофьев А.А., Зеленое 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. *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.
5. *Степченков Ю.А., Петрухин В.С., Хилько Д.В.* Выбор языковых средств представления параллельных алгоритмов для рекуррентного обработчика сигналов // Системы и средства информатики: Доп. Вып. - М.: ИЛИ РАН, 2008.-С. 149-158
6. *Питерсон Дж.* Теория сетей Петри и моделирование систем: Пер. с англ. - М.: Мир, 1984. - 264 с., ил.
7. *Зеленое 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

Сведения об авторе:

Хилько Дмитрий Владимирович, год рождения: 1987. Место обучения: аспирантура ИПИ РАН. Место работы: ИПИ РАН, 22 отд., инженер-исследователь. Направления научных интересов: нетрадиционные архитектуры микропроцессоров; параллельное и потоковое программирование, имитационное и математическое моделирование, интеллектуальные системы.