Разработка инструментальной среды проектирования программного обеспечения для рекуррентно-потоковой модели вычислений[*]

Д.В. Хилько[1]*, Ю.И. Шикунов*[2]

Аннотация. Статья посвящена новой рекуррентно-потоковой модели вычислений и основным проблемам разработки ПО для этой модели. Рассмотрен вопрос сходимости рекуррентной организации вычислительного процесса, а также ключевые аспекты методологии программирования для новой модели. Предлагается возможная архитектура инструментальной среды проектирования ПО, предназначенного для рекуррентно-потоковой модели. Рассматриваются перспективы развития предложенной архитектуры.

Ключевые слова: модель вычислений, рекуррентность, рекурсивность, методология программирования, разработка ПО.

Введение

Начиная со второй половины XX века ведутся исследования и разработки систем потоковой архитектуры [1], но, несмотря на видимые преимущества – такие, как отсутствие «узких мест», характерных для фон-неймановской архитектуры [2], исключение вероятности обработки неподготовленных данных, – ряд проблем как технического, так и алгоритмического характера препятствует массовому применению потоковых архитектур.

В поисках путей усовершенствования потоковой модели вычислений, коллективом Института кибернетики имени В. М. Глушкова НАН Украины (Палагин А.В., Яковлев Ю.С. Махиборода А.В., и др.) была предложена идея новой потоковой модели вычислений [3], которая, впоследствии, была развита и доработана сотрудниками Института проблем информатики РАН (ИПИ РАН) – Степченковым Ю.А., Петрухиным В.С. и др. Полученная модель вычислений

* Работа выполнена при частичной финансовой поддержке по программам фундаментальных исследований ОНИТ РАН за 2013 г. (проект 1.5) и Президиума РАН (проект 16П-1)
[1] Институт проблем информатики Российской академии наук, dhilko@yandex.ru
[2] Институт проблем информатики Российской академии наук, YIShikunov@gmail.com

была названа рекуррентно-потоковой [4]. На ее основе была разработана многоядерная потоковая рекуррентная архитектура (МПРА).

В основе рекуррентно-потоковой модели лежит понятие самодостаточных данных, т.е. единого потока данных и инструкций. В системе самодостаточных данных каждый следующий шаг обработки порождается в ходе развития процесса как функция от предыдущего. Таким образом, исходный поток инструкций рекуррентно сворачивается, тем самым позволяя резко сократить накладные расходы, связанные с опережающим хранением трассы вычислительного процесса. Кроме того, данная особенность позволяет сократить практически в два раза время, требуемое для обработки каждой отдельной инструкции и связанных с ней данных.

В работе [5] было показано, что способ организации вычислительного процесса, в рамках рекуррентно-потоковой модели вычислений, принципиально отличается от уже существующих способов в системах как традиционной, так и нетрадиционной архитектуры. Данный факт обуславливает необходимость и разработки новых методов и алгоритмов подготовки, управления и обработки данных для решения задач в среде рекуррентно-потоковой модели вычислений, а также специализированных программных средств.

1. Рекуррентно-потоковая модель вычислений. Сходимость рекуррентного вычислительного процесса

В работе [4] приводится анализ различных архитектур вычислительных систем, приведена их классификация и определено место возможных реализаций рекуррентно-потоковых вычислительных архитектур, а в работе [6] выявлены и проанализированы принципиальные отличия трех моделей вычислений.

К первой модели вычислений в [6] отнесены архитектуры, основанные на принципах, введенных фон Нейманом. Авторы [6] пишут:

“Для выполнения программы в традиционной классической фон-неймановской архитектуре требуется некоторый объем запоминающей среды

(памяти), которая функционально (а для гарвардской архитектуры и физически) разделена на две области - программ и данных. Инициатором выполнения вычислительной процедуры является поток инструкций, извлекаемый из памяти программ ЦПУ (первичный поток инструкций). Программа-инициатор процесса хранится в памяти инструкций в полном объеме и в статическом виде. При этом существует проблема определения момента готовности данных для их обработки. Соответствующая модель вычислений была названа CF/S - Control Flow/Static.”

Ко второй модели вычислений отнесены архитектуры, основанные на принципах потока данных (или потоковые). Авторы [6] пишут:

“Для потоковых архитектур (DF - Data Flow) сохраняется разделение ресурсов памяти на две области. Однако статус памяти данных меняется - из пассивной (вторичной) она превращается в активную (первичную), в ячейках которой хранятся данные (операнды) с дополнительными функциональными полями (полями тегов). Суммарный объем требующихся ресурсов памяти ориентировочно не изменяется. Как правило, функциональные поля содержат в себе информацию об исполнительном адресе инструкции (микрокоманды), которая должна быть извлечена из памяти программ для выполнения требуемых действий. Полный объем привлекаемых инструкций также хранится в статическом виде.”

Поэтому, данная модель была названа DF/S - Data Flow/ Static.

Для рекуррентно-потоковой модели вычислений характерны следующие отличия от DF/S [6]:

1) Тегируемые данные являются самодостаточными, т.е. содержат помимо самих данных также и управляющую информацию. Такие тегируемые данные также могут называться рекуррентно разворачивающимися.

2) 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 рамках рекуррентной модели в состав ЦПУ включено

устройство преобразования тегов (ПТ), которое обладает функциональностью саморазвертки рекуррентного вычислительного процесса. Устройство ПТ инициируется операндами, пришедшими на обработку в ЦПУ, работает параллельно с ЦПУ и определяет действие на следующем шаге вычислительного процесса (модифицирует тег).

3) В памяти нет исполняемой программы в традиционном смысле. Есть только начальные значения функциональных полей операндов, которые динамически подвергаются рекуррентной саморазвёртке устройством ПТ (отсюда название архитектуры DF/D - Data Flow/Dynamic). Чтобы выполнить алгоритм, необходимо задать начальные значения функциональных полей.

Рис. 1 (рис. П1 из [6]) и 2 (рис. П2 из [6]) иллюстрируют сравнение описанных принципиальных классов по организации и механизму работы соответственно.

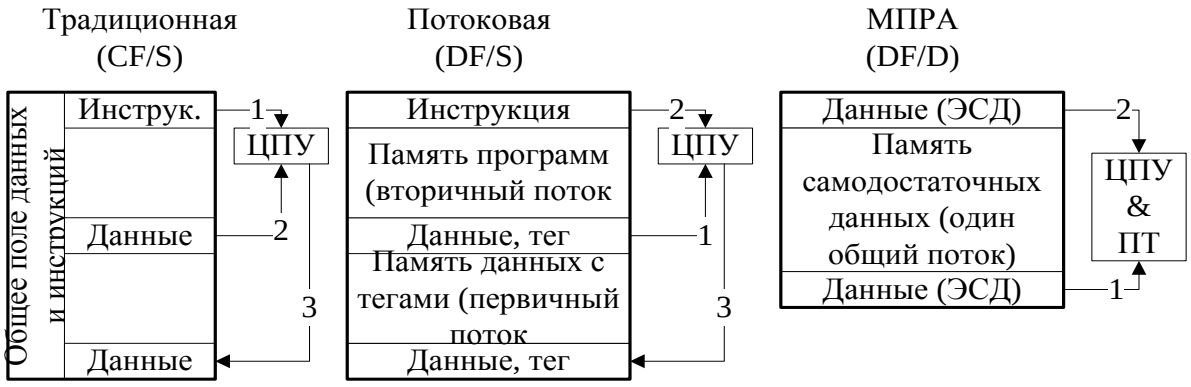

Рис. 1 Принципиальные отличия сравниваемых моделей вычислений

(рис. П1 из [6])

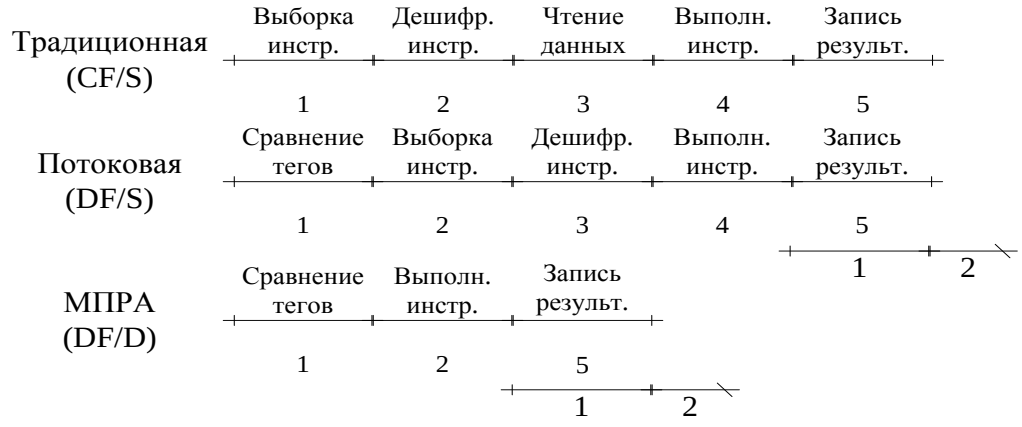

Рис. 2 Выполнение инструкций в сравниваемых архитектурах

(рис. П2 из [6])

Для дальнейших исследований новой модели вычислений необходимо было доказать сходимость рекуррентного вычислительного процесса. В работе [7] в ходе доказательства автор пишет: “использовались следующие термины и теоремы теории рекурсивных функций: теорема о нумерации, определение примитивно рекурсивной функции, определение частично рекурсивной функции, s-n-m-теорема, тезис Черча. Было показано, что существует способ построения частично рекурсивной функции одной переменной, описывающей рекуррентные преобразования над функциональным полем. Полученная функция имеет вид, приведенный в формуле (1).

$$f(0) = f_0; f(1) = f_1; \dots, f(t) = f_t, \dots, f(n-1) = f_{n-1}$$
$$f(t) = f(t-1) + g(k) \quad (1)$$
$$g(k) = \begin{cases} f_t - k, \text{если } k = f_{t-1} \\ g(k-1), \text{если } k \neq f_{t-1} \end{cases}, k = \overline{1,l}$$

Здесь n – количество шагов преобразований, t – номер текущего шага преобразований, l – мощность упорядоченного и пронумерованного множества всех возможных значений функционального поля, f_t – номер значения, которое должно принять функциональное поле на *t*-ом шаге преобразований, *g(k)* – функция вычисления приращения (иначе говоря, функция выбора подходящего значения функционального поля из всего множества возможных), *f(t)* – функция вычисления рекуррентной последовательности номеров значений.”

По теореме о нумерации существует частичная функция нескольких переменных, а по s-n-m-теореме существует некоторый соответствующий этой функции Геделев номер m_0. Это означает, что существует множество функций видов функций, вычисляющих требуемую цепочку рекуррентных преобразований.

В текущей реализации рекуррентно-потоковой модели вычислений, реализована одна из возможных функций с номером m_0 – универсальная функция преобразований, обеспечивающая любую требуемую глубину

развертки. При этом, начиная с третьего шага развертки, преобразования зацикливаются, возвращая тем самым один и тот же результат (эта ситуация не является тупиком). Данная функция имеет вид, указанный в формуле (2).

$$f(t,k) = \begin{cases} f(t-1,k) - 2^m, \text{если } f(t,k) > 0 \\ \text{останавливается, если } f(t,k) \le 0 \end{cases}$$
$$g(t,k) = \begin{cases} g(t-1,k) + 1, \text{если } f(t,k) > 0 \\ \text{останавливается, если } f(t,k) \le 0 \end{cases} \quad (2)$$
$$g(0,k) = 0, k = \overline{0, n-1}$$
$$f(0,0) = f_0,$$
$$f(0,k) = g(t_0, k-1), k = \overline{1, n-1}$$

Здесь k – номер шага рекуррентной развертки, t – номер шага вычислений функций f и g, t_0 – номер шага, на котором остановились функции f и g, m – размер функционального поля в разрядах, $f(t,k)$ – функция преобразования значения функционального поля на шаге преобразований k, $g(t,k)$ – функция вычисления нового значения функционального поля на шаге преобразований k.

2. Результаты применения рекуррентно-потоковой методологии программирования

Подробное описание разработанной методологии, а также пример ее применения для реализации программы распознавателя изолированных слов приводится в работе [7]. Методология включает в себя *четыре* основных этапа, ключевым из которых является этап III, названный «методикой капсульного программирования». В рамках настоящей статьи рассмотрены результаты применения новой методологии для реализации программы распознавания слов.

Новая методология была использована для решения задачи распознавания слов в среде рекуррентно-потоковой модели вычислений. Были осуществлены оценки коэффициентов ускорения реализованных в капсульном виде алгоритмов, относительно одноядерного микроконтроллера dsPic30F. Результаты приведены в таблице 1.

Таблица 1 — Результаты реализации алгоритмов распознавания

Название алгоритма	Кол-во шагов для dsPic30F	Кол-во шагов для МПРА		Коэф. ускорения	
		Вар. 1	Вар. 2	Вар. 1	Вар. 2
Баттеруорт (одна секция)	679	288	-	~2,36	-
Полосовой фильтр (одна полоса)	1428	420	-	3,4	-
Натуральный логарифм	36	12	26	3	~1,38*4
RASTA фильтр	153	28	-	~5,46	-
Экспоненцирование	36	12	26	3	~1,38*4
Косинусное ИДПФ	36	12	26	3	~1,38*4
Рекурсия Дурбина-Скурра	~640	~110	-	~5,8	-
PLP параметры	144	32	-	4,5	-
Витерби (расчет решетки для текущего N *)	91*N-143	99*N	(1-(8*N+143)/(99*N))*4		

* - N – кол-во наблюдений в векторе наблюдений (N>5)

Некоторые из множества алгоритмов были реализованы в двух вариантах. При этом вариант 1 означает, что алгоритм был реализован с использованием всех 4 вычислительных устройств (ВУ), а вариант 2 означает, что алгоритм был реализован для 4-х комплектов входных данных с использованием одного ВУ для каждого комплекта.

3. Инструментальная среда проектирования ПО для новой модели вычислений

Для того чтобы автоматизировать процесс разработки ПО, предназначенного для новой модели вычислений, необходимо реализовать предложенную методологию программирования в виде готовой технологии. Одной из первоочередных задач в данном направлении является разработка интегрированной среды проектирования ПО.

На текущем этапе разработки представляется возможным определить базовый набор библиотек и компонентов, которые должны войти в состав

инструментальной среды разработки. На рисунке 3 представлена возможная обобщенная архитектура среды разработки ПО для МПРА (GARIS IDE).

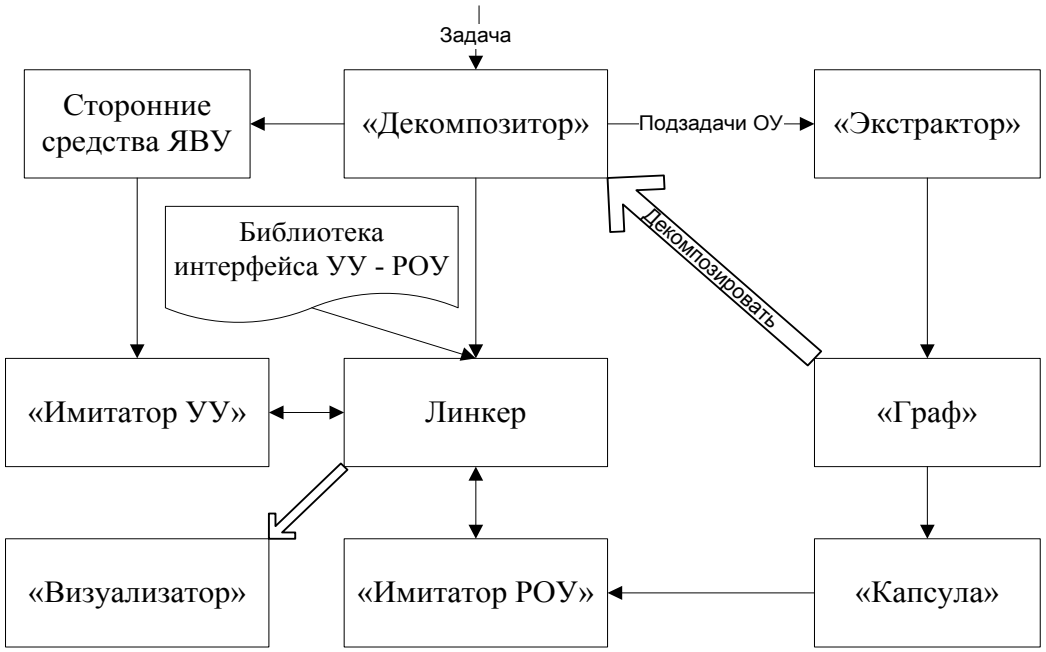

Рис. 3 Обобщенная архитектура GARIS IDE

Компонент «Декомпозитор» - предназначен для декомпозиции решаемой задачи и предоставляет следующие функциональные возможности:

- разбиение решаемой задачи (ручное или автоматизированное) на два множества (подзадачи УУ и подзадачи ОУ);
- создание и хранение спецификации форматов входных и выходных данных для каждой подзадачи;
- создание карт памяти и данных на основе спецификаций форматов данных.

Сторонние средства ЯВУ (языков высокого уровня) – набор сторонних программных средств для разработки традиционного программного обеспечения. Предназначены для реализации программы УУ.

Компонент «Экстрактор» - предназначен для извлечения графа вычислительного процесса (ВП) из описания подзадачи на языке высокого уровня. Состоит из следующих модулей:

- текстовый редактор ЯВУ;

- транслятор из ЯВУ в текстовую форму графа ВП;
- транслятор из текстовой формы графа в визуальный граф ВП.

Компонент «Граф» - многофункциональный компонент, предоставляющий широкий спектр возможностей для работы с визуальными графами. Может базироваться на свободно распространяемой библиотеке. В случае необходимости (в соответствии с рекуррентно-потоковой методологией) подзадача может быть передана в «декомпозитор» для повторной декомпозиции. Состоит из следующих модулей:

- визуальный редактор графов;
- модуль преобразования визуального графа ВП в потоковой граф;
- модуль извлечения рекуррентных последовательностей;
- транслятор графа из потокового вида в динамический;
- транслятор графа из динамического вида в граф-капсулу.

Компонент «Капсула» - предназначен для работы с капсулами. Состоит из модулей:

- аналог визуального конструктора капсул программы СКАТ;
- текстовый конструктор капсулы и синтаксический анализатор;
- модуль рекуррентной свертки граф-капсулы в капсулу.

Компонент «Имитатор УУ» - предназначен для интерпретирования программы управляющего уровня. Может работать в двух режимах: полной интерпретации программы и интерпретации при помощи программы ПРАПОР.

Компонент «Имитатор РОУ» - предназначен для интерпретирования программы операционного уровня. Может работать в четырех различных режимах: интерпретация в режиме ПРАПОР, имитационное моделирование при помощи СИМПРА, моделирование при помощи VHDL-модели, интерпретация на опытном образце.

Компонент «Линкер» - предназначен для организации взаимодействия между имитаторами УУ и РОУ с учетом информации, хранящейся в картах памяти и данных. Осуществляет сборку и интерпретацию всей задачи в целом.

Компонент «Визуализатор» - предназначен для отображения результатов моделирования и отладки ПО.

Представленный набор компонентов является минимально необходимым для разработки GARIS IDE. Реализация предлагаемых компонентов в полном объеме позволит в максимальной степени автоматизировать процесс разработки ПО, что удовлетворяет требованиям, предъявляемым к GARIS IDE. Следует также отметить, что для каждого из указанных трансляторов необходимо разработать модели языков и соответствующие библиотеки. На текущем этапе разработки только программы ПРАПОР и СИМПРА имеют высокую степень завершенности.

Для описания капсулы разработана модель языка капсульного программирования, представленная в нотации расширенной нормальной формы Бекуса-Нуара.

$$S_i = tA_i \mid tC_i \mid tA_iF_i \mid tD_i \mid t \mid tC_iF_i \mid tF_i \mid tF_iC_i \mid tF_iA_i \mid tD_iF_i;$$

$$A_i = \lambda \mid C_a \mid S_a;$$

$$\lambda - \text{пустое поле, индекс } a - \text{ вспомогательное поле}$$

$$\mathrm{C}_i = I0_c \mid I1_c \mid T_c \mid C_c \mid A_c \mid B_c \mid B1_cB2_c;$$

$$\text{индекс } c - \text{ управляющее поле}$$

$$F_i = D_f \mid S_f \mid O_f ;$$

$$\text{индекс } f - \text{функциональное поле}$$

$$D_i = V_d \mid V0_dV1_dV2_d \mid V0_dV1_dV2_dV3_dV4_dV5_d;$$

$$\text{индекс } d - \text{содержательное поле}$$

$$\mathrm{C}_a = xprsec;$$

$$S_a = t\lambda \mid tn\lambda;$$

$$I0_c(I1_c) = nis;$$

$$T_c = rhmuctse;$$

$$C_c = am \mid jldbsm;$$

$$A_c = t \mid diu;$$

$$B_c = i \mid uchmse;$$

$$B1_c = hmuctse;$$

$$B2_c = b \mid \lambda;$$

$$D_f = rse \mid r;$$

$$S_f = hm;$$

$$O_f = cut \mid u;$$

Здесь: D_i – содержательная часть; F_i – функциональная (functional) часть; C_i – управляющая (control) часть; A_i – вспомогательная (additional) часть; t – тип операнда; S_i – i-ый операнд; строчной латинской буквой – имена подполей (на каждое имя отводится *ровно одна* буква, таким образом, комбинация вида *mnk* обозначает следующие друг за другом подполя *m, n и k*).

Следует вновь подчеркнуть, что предложенный набор средств и компонентов является минимально необходимым. Для обеспечения комфортных условий программирования в состав среды требуется также ввести ряд сервисных утилит, аналогичных многим существующим современным средам высокоуровневого программирования. Кроме того, в состав среды необходимо интегрировать разработанную программу СКАТ [7].

Заключение

В ходе работ по решению основных проблем, связанных с разработкой программного обеспечения для рекуррентно-потоковой модели вычислений были получены следующие значимые результаты:

- произведено доказательство сходимости рекуррентной организации вычислительного процесса, позволяющее гарантировать получение требуемых результатов;
- получено математическое описание процесса рекуррентных преобразований, применяемых в текущей реализации модели;
- разработана специализированная методология программирования в среде новой модели вычислений, а также показана ее эффективность, на примере реализации задачи распознавания слов;
- предложена архитектура и функциональность программных средств,

которые должны войти в состав инструментальной среды проектирования ПО для рекуррентно-потоковой модели вычислений;

- определены перспективы дальнейшего развития средств проектирования.

Таким образом, накоплен необходимый задел для внедрения рекуррентно-потоковой методологии в виде новой технологии программирования в процесс реализации новой модели вычислений. Представленные в статье результаты опубликованы в отчете по проекту 16П-1 «Капсула 2» [9].

Список литературы

1. T. Agerwala and Arvind, Data flow systems, IEEE Computer, 15 (Feb. 1982): PP. 10-13.
2. Arvind and R.A. Iannucci, A critique of multiprocessing von Neumann style, in Proc. 10th ISCA, June 1983: PP. 426-436.
3. Палагин А.В., Яковлев Ю.С., Махиборода А.В., Карпович В.А., Макаров Г.П. и Сергеев В.К. Система потоковой обработки информации с интерпретацией функциональных языков // Патент SU 1697084 . 1991. Бюл. №45.
4. Степченков Ю.А., Петрухин В.С. Перспективы развития цифровых, сигнальных процессоров и возможная реализация рекуррентного обработчика сигналов / Специальный выпуск «Методы и средства разработки информационно-вычислительных систем и сетей». – М.: ИПИ РАН, 2004. – С. 92-140.
5. 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.

6. Степченков Ю.А., Петрухин В.С., Филин А.В. Рекуррентное операционное устройство для процессора обработки сигналов / Системы и средства информатики: Вып. 11 / Под ред. И.А. Соколова. – М.: Наука, 2001. – С. 283-315.
7. Хилько Д.В., Степченков Ю. А. Теоретические аспекты разработки методологии программирования рекуррентной архитектуры / «Системы и средства информатики» – М.: ТОРУС ПРЕСС, Т. 23, № 2, 2013 – С. 136-156.
8. 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.
9. Обработка системы программирования многоядерных потоковых рекуррентных компьютерных систем предметной области: Отчет о НИР (заключительный), Шифр «КАПСУЛА2», № г.р. 01201368527. М.: ИПИ РАН, 2013, 34 С.

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

Сведения об авторе: Шикунов Юрий Игоревич, год рождения: 1995. Место обучения: МГТУ им. Н.Э. Баумана. Место работы: ИПИ РАН, 22 отд., стажер-исследователь. Направления научных интересов: нетрадиционные архитектуры микропроцессоров; параллельное и потоковое программирование,

имитационное моделирование, интеллектуальные системы, алгоритмы цифровой обработки.