УДК 004.272.44 DOI 10.31114/2078-7707-2018-3-2-9

Развитие средств капсульного программирования потоковой рекуррентной архитектуры

Д.В. Хилько, Ю.А. Степченков, Ю.И. Шикунов, Г.А. Орлов

Институт проблем информатики Федерального государственного учреждения «Федеральный исследовательский центр «Информатика и управление» Российской академии наук», г. Москва

dhilko@yandex.ru, ystepchenkov@ipiran.ru, yishikunov@yandex.ru, orlov.jaja@gmail.com

***Аннотация* — В статье рассматриваются новые результаты, полученные в ходе работ по направлению разработки методов и средств программирования многоядерной потоковой рекуррентной архитектуры. На текущем этапе разработки основной целью является автоматизация построения специального инструмента программиста – граф-капсулы, который позволяет наглядно отображать распределение ресурсов архитектуры в процессе выполнения программы. Для этого был разработан компонент построения граф-капсул в числовом виде, использующий результаты моделирования. Следующим шагом в развитии средств программирования является разработка инструментария для построения потоковых графов и граф-капсул на их основе в символьном виде, что позволит заложить основу для создания средств компиляции в будущем. Обсуждению результатов решения данной задачи посвящена настоящая статья.**

***Ключевые слова* — потоковая архитектура, потоковый граф, граф-капсула, капсульное программирование.**

I. Введение

Большинство современных вычислительных систем поддерживает параллельные вычисления на различных уровнях. Перспективное, но не получившее широкого распространения, направление развития параллельных вычислительных систем – разработка вычислительных устройств на основе архитектуры, управляемой потоком данных, которая также называется потоковой. Одним из главных преимуществ потоковых архитектур является «естественная» поддержка параллелизма, что достигается благодаря принципу вычислений по готовности данных. Тем не менее, действующего прототипа на основе потоковой архитектуры до сих пор не было создано, что обусловлено целым рядом проблем [1] – [3].

В России коллективом отдела «Архитектурные и схемотехнические основы инновационных вычислительных систем» Института проблем информатики РАН ведётся разработка многоядерной потоковой рекуррентной архитектуры (МПРА), апробация которой осуществляется на предметной области ЦОС. Основные аспекты архитектуры МПРА описаны в работах [4] – [6]. Как показано в работе [7] заложенные в архитектуру принципы позволяют частично или полностью решить проблемы, присущие потоковым архитектурам. В настоящее время разработка и развитие МПРА идет по трем основным направлениям: повышение ее производительности; создание инструментов разработки и отладки ПО и уменьшение избыточности данных.

Теоретические и технические аспекты архитектуры изложены в рамках трудов конференции МЭС [7], [8], где описаны механизмы и решения, позволившие достичь высокой производительности прототипа МПРА на демонстрационной задаче распознавания изолированных слов. Проблемы уменьшения избыточности данных в потоковых архитектурах и механизмы их решения подробно рассмотрены в работах [9], [10]. Как показано в работах [11] – [13] ключевые особенности архитектуры не позволяют использовать существующие средства программирования. Поэтому для МПРА создаются специализированные средства разработки и отладки ПО.

Для экспериментальной апробации рассматриваемой архитектуры разработан ее прототип: рекуррентный обработчик сигналов (РОС), исполняемый в гибридном двухуровневом варианте с ведущим фоннеймановским процессором на управляющем (верхнем) уровне (УУ) и рядом потоковых процессоров на нижнем уровне – рекуррентном операционном устройстве (РОУ) [14]. Данный прототип был назван «Гибридная архитектура рекуррентного обработчика сигналов» (ГАРОС).

На сегодняшний день разработаны и внедрены рекуррентно-потоковая методология программирования, средства аппаратно-программного моделирования СИМПРА и СПРУТ [15], [16], интегрированная среда разработки GAROS IDE [13], [17]. Программа в МПРА представляет собой последовательность самодостаточных данных, в которых закодированы инструкции по их обработке. Подобное представление было названо капсулой, а сам стиль программирования – капсульной парадигмой программирования.

Дальнейшее развитие средств программирования привело к необходимости пересмотра и развития как методологии программирования, так и архитектуры средств программирования. В работе [18] обсуждаются результаты первых шагов развития методологии программирования, а также приводится описание новых средств разработки, позволяющих строить специаль-

ный инструмент программиста – граф-капсулу в числовом виде. Опыт использования данного инструмента показал необходимость дальнейшего развития методологии программирования, а также создания инструментария построения граф-капсул в символьном виде. Результатам решения данных задач и посвящается настоящая статья.

II. Рекуррентно-потоковая методология программирования

A. *Описание методологии программирования*

В статье [12] приводятся теоретические аспекты разработки элементов методического обеспечения процесса программирования МПРА. В том числе, в статье определено понятие рекуррентно-потоковой методологии программирования, которая описывает процесс разработки ПО, начиная от математического описания задачи до создания капсул и сопутствующих им структур данных. Ключевым этапом рассматриваемой методологии является этап «Капсульное программирование», в рамках которого описывается процесс разработки капсулы, предназначенной для решения конкретной задачи. На рис. 1 представлена общая структура методологии.

Рис. 1. Структура методологии программирования

Капсула, как и любая другая программа, имеет соответствующее текстовое представление, которое было названо символьной капсулой. Для отладки символьной капсулы используются как тестовые, так и реальные пакеты данных, на основе которых формируется и моделируется числовая капсула. При этом составляется легенда имен входных данных.

Фундаментальное свойство МПРА («рекуррентность») позволяет создавать программы с динамически порождаемой схемой вычислительного процесса (ВП). С одной стороны, это позволяет сжать исходную программу, но с другой – существенно затрудняет разработку и отладку таких программ (капсул). Поэтому в рамках методологии был введен новый элемент разработчика – «граф-капсула», который является графической интерпретацией процесса разворачивания схемы ВП, закодированного в капсуле.

Опыт программирования и решения задач позволил организовать итеративный процесс разработки капсул на основе потокового графа, граф-капсулы и результатов моделирования. При этом граф-капсула используется в качестве средства верификации.

Использование граф-капсул в качестве основного инструмента отладки доказало свою практическую эффективность. Однако ручное построение данного инструмента разработчика связано со значительными трудностями. Естественным образом возникла проблема автоматизации построения граф-капсул. Решение этой научно-технической проблемы позволило не только на порядок сократить время разработки и отладки капсул, но и повысить наглядность результатов моделирования капсул, детально оценивать степень загруженности вычислительных ресурсов архитектуры.

На текущем этапе развития средств программирования удалось частично решить задачу автоматизации построения граф-капсул, но только в числовом виде на основе результатов моделирования числовой капсулы. Первый опыт эксплуатации числовых граф-капсул представлен в работе [18]. Этот опыт показал, что отладка и верификация с помощью числовой граф-капсулы сопряжена с рядом трудностей, которые существенно снижали эффективность ее использования как разработчиками поведенческих моделей, так и разработчиками макетного образца МПРА.

В качестве примера на рис. 2 приводится фрагмент потокового графа (рис. 1 из [18]) реализации алгоритма Витерби, а на рис. 3 и рис. 4 (рис. 4 и рис. 5 из [18]) соответствующая ему числовая граф-капсула.

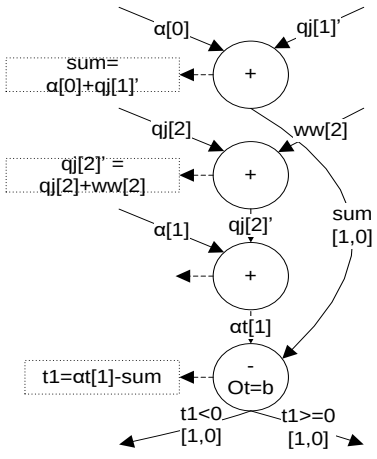

Рис. 2. Фрагмент потокового графа (рис. 1 из [18])

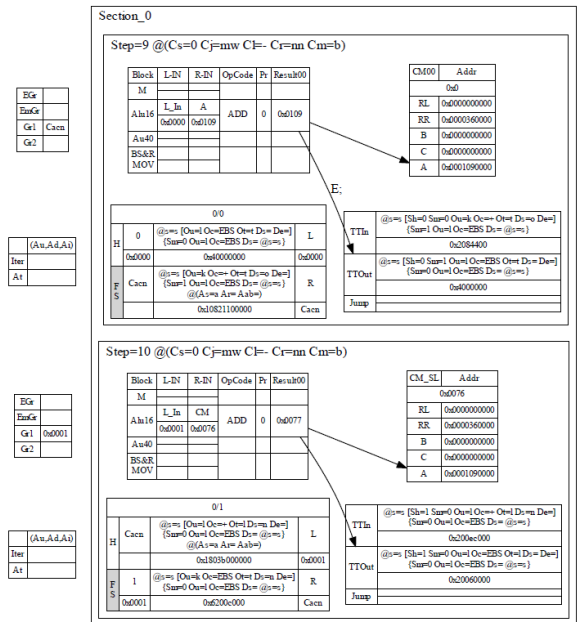

Рис. 3. Числовая граф-капсула, часть 1 (рис. 4 из [18])

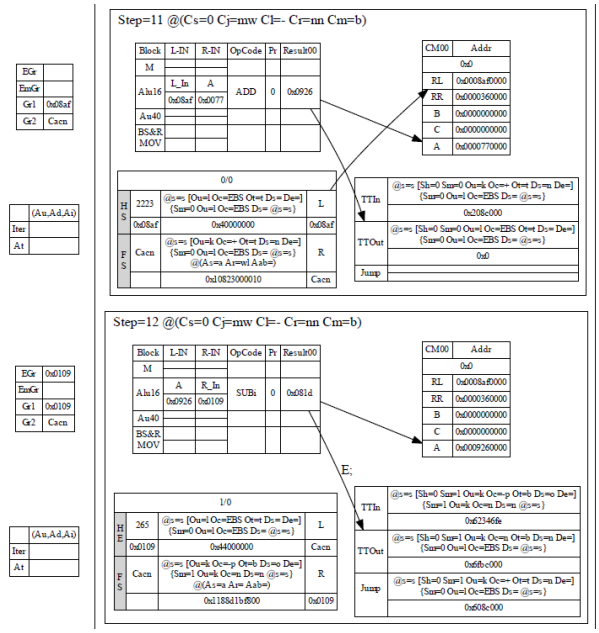

Рис. 4. Числовая граф-капсула, часть 2 (рис. 5 из [18])

B. Развитие методологии

Одним из ключевых подэтапов (рис. 1) «капсульного программирования» является Подэтап III-2, который в статье [12] назван «Методика капсульного программирования». В работе [18] была предложена доработанная методика, согласно которой на данном подэтапе осуществляется построение:

- развернутого потокового графа в символьном виде;
- свернутого символьного потокового графа;
- символьной капсулы;
- числовой капсулы и числовой граф-капсулы.

Далее осуществляется установление идентичности между символьным потоковым графом и числовой граф-капсулой. Одной из главных проблем на данном этапе является трудоемкость сопоставления символьных данных потокового графа и числовых данных граф-капсулы. Фактически, произвести подобное сопоставление является возможным при условии хранения легенды имен входных данных, а также при постоянном глубоком анализе выполняемых на каждом шаге операций. Кроме того, из рис. 2 и 3 видна высокая степень информационной нагрузки граф-капсулы, что сильно затрудняет ее анализ.

Необходимость постоянного сопоставления наблюдаемых числовых значений с их семантическим смыслом значительно снижает эффективность использования числовых граф-капсул. В тоже время использование символьных обозначений является более естественным способом верификации, т.к. получаемые результаты легко сопоставимы с исходным математическим описанием задачи по п.п. 1. Очевидной стала необходимость внедрения символьных граф-капсул для верификации.

Возможность автоматизированного построения как числовой, так и символьной граф-капсул позволило логически разделить области их применения. Очевидно, что от разработчика аппаратного обеспечения нельзя требовать детального знания математической постановки решаемой задачи и специфики ее капсульного программирования. Зато в процессе отладки аппаратного обеспечения числовая граф-капсула оказывается крайне полезным инструментом за счет наглядной интерпретации ВП в аппаратной среде.

В тоже время на первых этапах разработки программного обеспечения программисту необходимо иметь возможность оценивать логическую корректность формируемой капсулы, например: правильность связывания пар операндов, правильность рассылки пар операндов, правильность инициализации внутренних ресурсов МПРА и т.д. Также на данном этапе разработки капсулы нет необходимости решать вопросы точности и отслеживать корректность числовых значений переменных. С учетом всего вышесказанного, символьная граф-капсула становится наиболее полезным инструментом разработки символьной капсулы.

Таким образом, переработанная методика капсульного программирования будет включать следующие этапы:

1) анализ математического описания задачи и построение потокового графа параллельного алгоритма;

2) доработка потокового графа в соответствии с ограничениями, накладываемыми спецификацией прототипа МПРА;

3) построение динамического потокового графа путем осуществления рекуррентной свертки подграфов потокового графа;

4) разработка фрагмента символьной капсулы, реализующей часть динамического графа;

5) формирование и моделирование фрагмента числовой капсулы средствами GAROS IDE;

6) построение символьной и числовой граф-капсул на основе результатов моделирования;

7) верификация программы с помощью символьной граф-капсулы;

8) верификация как моделей и макетного образца, так и программы с помощью числовой граф-капсулы;

9) капсула считается завершенной в случае полного соответствия развернутого потокового графа и соответствующих ему граф-капсул, в противном случае перейти к п.п. 4.

Для реализации обновленной методики в состав средств программирования необходимо ввести средства автоматизации построения символьных граф-капсул, а в перспективе и средств автоматизации верификации.

III. Автоматизация построения граф-капсул в GAROS IDE

A. *Описание GAROS IDE*

Разработка капсул для МПРА сопряжена с многими трудностями в силу того, что язык капсульного программирования – это язык «ассемблерного» типа. Кроме того, данные и инструкции в капсуле хранятся в рекуррентно-сжатом виде. Поэтому ведется разработка специализированной интегрированной среды разработки GAROS IDE, функциональные возможности которой в значительной степени реализуют основные этапы рекуррентно-потоковой методологии программирования. На рис. 5 представлена архитектура GAROS IDE.

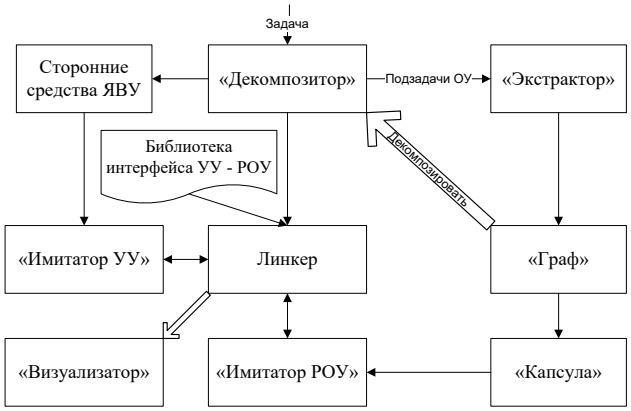

Рис. 5. Архитектура GAROS IDE

Компонент «Декомпозитор» предназначен для декомпозиции решаемой задачи (Этап II методологии на рис. 1), пока не реализован. *Сторонние средства ЯВУ* (языков высокого уровня) – набор программных средств для разработки ПО, исполняемого на управляющем уровне (УУ). *Компонент «Экстрактор»* предназначен для извлечения параллелизма на основе описания задачи на языке высокого уровня (подэтап III-1 на рис. 1), пока не реализован. *Компонент «Граф»* предназначен для ручного или автоматизированного построения потоковых графов и граф-капсул, частично реализован. Позволяет строить как символьные, так и числовые граф-капсулы (подэтап III-2 на рис. 1). *Компонент «Капсула»* предназначен для построения капсул.

Компонент «Имитатор УУ» предназначен для интерпретирования программы управляющего уровня. *Компонент «Имитатор РОУ»* предназначен для интерпретирования программы операционного уровня. Включает в себя средства имитационного моделирования СИМПРА. *Компонент «Линкер»* предназначен для организации взаимодействия между имитаторами УУ и РОУ с учетом информации, хранящейся в вспомогательных структурах данных капсул. Осуществляет сборку и интерпретацию всей задачи в целом. *Компонент «Визуализатор»* предназначен для отображения результатов моделирования, а также потоковых графов и граф-капсул.

Таким образом, более половины компонент среды разработки уже реализованы. Также осуществляется постоянная модификация уже разработанных компонент, в частности, компонент «Визуализатор» был интегрирован в компонент «Граф». В рамках данной статьи рассматриваются результаты расширения функциональных возможностей GAROS IDE для построения символьных граф-капсул.

B. *Методика построения символьных граф-капсул*

Входящие в состав GAROS IDE средства имитационного моделирования в настоящее время не поддерживают символьного режима моделирования. Поэтому напрямую преобразовать символьный потоковый граф в символьную граф-капсулу не представляется возможным. В то же время, разработанный и успешно апробированный функционал построения числовых граф-капсул может быть эффективно использован для построения символьных граф-капсул. Таким образом, задача может быть сформулирована следующим образом: необходимо разработать метод преобразования числовой граф-капсулы в символьную на основе развернутого символьного потокового графа.

Указанный потоковый граф строится в соответствии с рассмотренной методикой капсульного программирования, при этом к нему предъявляются следующие требования:

• для каждого узла вводится понятие контекста номера вычислительного шага и имени параллельного вычислительного потока;

• для каждого узла вводится типизация: простой, составной (для суперскалярных режимов);

• каждый узел описывает операцию из системы команд РОУ;

• каждой дуге ставится в соответствие данное и его символьное имя;

• для каждого данного вводится типизация: входное, внутреннее, промежуточное, выходное;

• допускается множественная типизация дуг (данных);

• для каждого данного типа «входное» устанавливается соответствие L- или R- компоненту пары.

С учетом данных требований методика преобразования числовой граф-капсулы в символьную включает следующие этапы:

1) преобразование развернутого потокового графа в соответствии с приведенными требованиями;

2) построение соответствующей символьной капсулы;

3) построение и моделирование числовой капсулы;

4) построение символьной граф-капсулы на основе результатов моделирования и преобразованного потокового графа.

C. Реализация средств построения граф-капсул

Для решения задачи автоматизации построения символьных граф-капсул в состав GAROS IDE были доработаны компоненты «Имитатор РОУ» и «Граф»:

• в «Граф» была добавлена утилита построения развернутого потокового графа и символьной граф-капсулы;

• в «Имитаторе РОУ» был расширен сбор информации об истории моделирования.

В статье [18] описана первая версия средства построения числовых граф-капсул, интегрированное в компонент «Граф», а также приведены результаты исследования различных библиотек построения и визуализации графов. Опыт ее использования позволил выявить ряд недоработок в оформлении и информативности числовой граф-капсулы: отсутствие информации о внутреннем типе операндов пары; низкая степень наглядности использования внутренних ресурсов вычислительных блоков и т.п. В обновленной версии компонента «Граф» данные проблемы были решены, причем для обоих вариантов построения граф-капсул.

В качестве средства построения и обработки потокового графа была выбрана библиотека QuickGraph. Для визуализации потокового графа использовалась библиотека GraphX, а для символьной и числовой граф-капсулы – утилита GraphViz. Средствами библиотеки QuickGraph осуществляется построение объектной структуры развернутого потокового графа в памяти приложения, которая:

• может быть передана в утилиту построения символьных граф-капсул;

• может быть сохранена в файл;

• может быть визуализирована средствами GraphX.

Результирующий потоковый граф вместе с результатами моделирования используется для построения символьной граф-капсулы. Благодаря введению понятия контекста для каждого узла потокового графа, процедура его сопоставления с результатами числового моделирования значительно упрощается. Для каждого очередного контекста (шага) в ходе построения символьной граф-капсулы осуществляются следующие действия:

• Загрузка символьных имен, назначенных внутренним ресурсам на предыдущем шаге.

• Осуществляется проверка на наличие выходных данных; если они имеются, то осуществляется подстановка символьного имени дуги с типом «выходное».

• Для дуг типа «входные» с маркерами L и R их символьные имена присваиваются соответствующим входным шинам данных вычислительного блока РОУ.

• Для дуг типа «внутренние» их символьные имена присваиваются либо непосредственно внутренним ресурсам (если дуга выходная для текущего узла), либо подставляется имя соответствующего ресурса-источника (если дуга входная для текущего узла), ресурсы определяются на основе анализа истории моделирования.

• Для дуг типа «Промежуточные» - имя присваивается результату промежуточных вычислений в суперскалярном режиме.

• По мере обработки истории моделирования сохраняются ранее присвоенные символьные имена, до тех пор, пока данные не будут перезаписаны.

За счет информации, представленной в узлах потокового графа и типизированных дуг, можно однозначно установить соответствие данного, представленного в истории моделирования и имени, не требуя при этом явной привязки к физическим ресурсам РОУ.

В качестве демонстрации на рис. 6 и 7 приводится результат построения символьной граф-капсулы, соответствующей фрагменту потокового графа и числовой граф капсуле на рис. 2 – 4.

Для большей ясности введем последовательную нумерацию узлов фрагмента графа на рис. 2 от 0 до 3. В узле №0 выполняется операция *sum=α[0]+qj[1]'*. На символьной граф-капсуле (рис. 6, Step=9) мы можем легко идентифицировать аналогичную операцию *sum=alpha[0]+qj[1]'*, в то время как на числовой граф-капсуле (рис 3, Step=9) выполняется та же самая операция, однако семантика ее не очевидна без хранения и анализа дополнительной информации о ВП.

Аналогичным образом можно рассмотреть узлы №№ 1-3 на рис. 2 и убедиться, что символьная граф-капсула позволяет легко осуществлять верификацию и отладку логики исполнения капсулы. В свою очередь, числовая граф-капсула позволяет достигать требуемую точность вычислений. С учетом того, что оба варианта граф-капсул строятся на основе одних и тех же результатов моделирования, мы можем гарантировать их взаимно-однозначное соответствие.

IV. Заключение

Разработанные средства построения потоковых графов и символьных граф-капсул предоставляют мощный и гибкий инструментарий верификации капсул. Практика их использования позволила сократить среднее время разработки и отладки капсул в 2-3 раза по сравнению с использованием числовых граф-капсул.

Основным научно-практическим результатом данной работы является развитие методических и программных средств разработки и отладки ПО для МПРА:

• введение инструмента символьная граф-капсула, как основного средства верификации исполнения программ;

• развитие инструмента числовая граф-капсула, как основного средства отладки макетного образца МПРА.

Кроме того, одновременное использование развернутого потокового графа и символьной граф-капсулы позволило автоматизировать процесс поиска ошибок при проектировании капсулы. Это достигается за счет автоматизации анализа на возможность использования тех или иных внутренних ресурсов РОУ в утилите построения символьных граф-капсул.

Дальнейшее развитие средств программирования МПРА видится в решении обратной задачи, а именно, в построении потокового графа на основе символьной граф-капсулы. Создание подобного инструментария позволит автоматизировать процесс верификации капсулы на всех этапах «методики капсульного программирования». Следующим шагом уже станет разработка первых версий средств компиляции и трансляции потоковых графов напрямую в капсулы.

Благодарности

В заключение хотим выразить свои благодарности Морозову Н.В., Дьяченко Ю.Г. и Рождественскому Ю.В. за неоценимый вклад в разработку и отладку капсул на аппаратной модели МПРА.

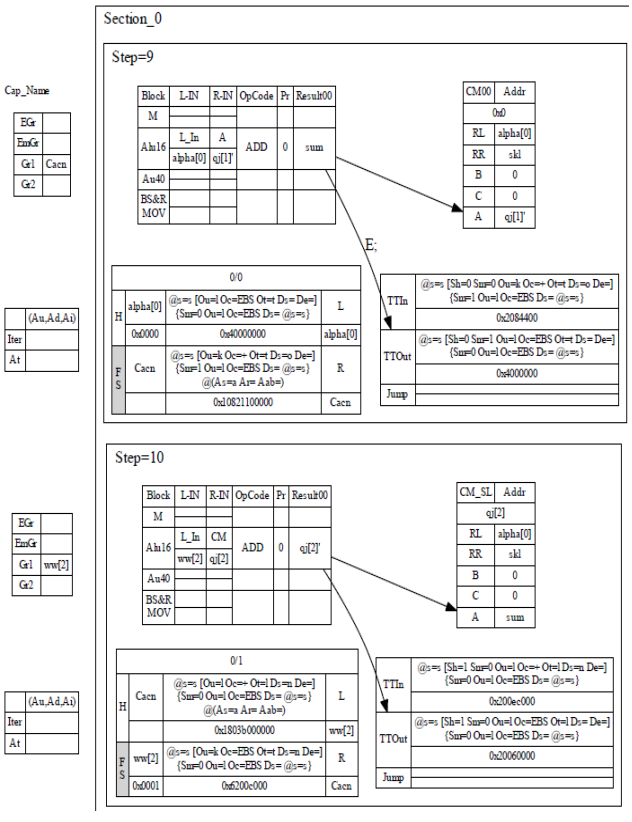

Рис. 6. Символьная граф-капсула, часть 1

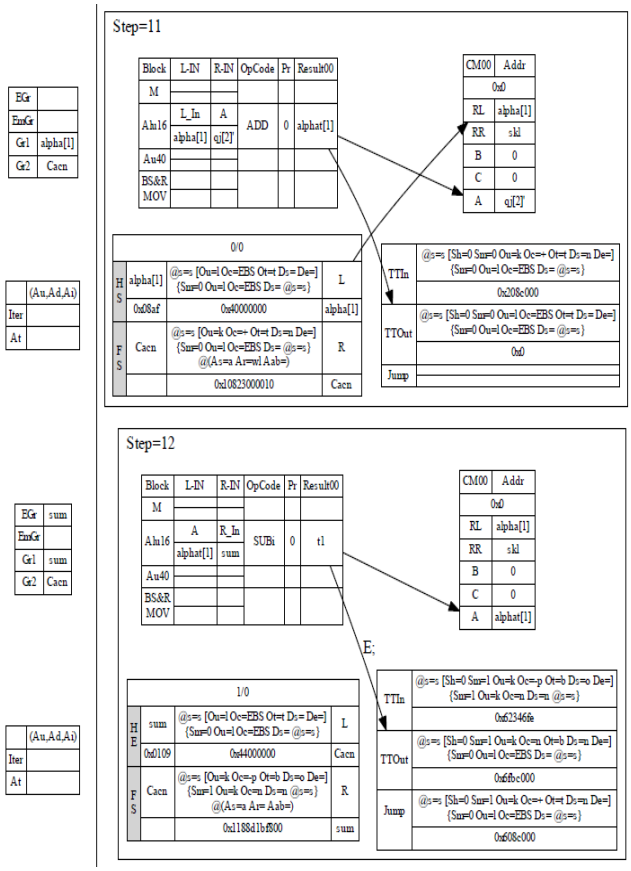

Рис. 7. Символьная граф-капсула, часть 2

Поддержка

Исследование выполнено в рамках государственного задания № 0063-2017-0011.

Литература

[1] E. A. Lee and J. C. Bier. Architectures for Statically Scheduled Dataflow // Parallel Algorithms and Architectures for DSP Applications / edited by Magdy A. Bayoumi. Dordrecht. Kluwer Academic Publishers, 1991. P. 159-190.

[2] V.P. Srini. DFS-SuperMPx: Low-cost Parallel Processing System for Machine Vision and Image Processing // Proc. Third International Conference “Parallel Computing Technologies”, PaCT-95. St. Petersburg, 1995. Vol. 3. P. 356-369.

[3] S. Voigt, M. Baesler, T. Teufel. Dynamically reconfigurable dataflow architecture for high-performance digital signal processing // Journal of Systems Architecture, 2010, №. 56. P. 561-576.

[4] Степченков Ю.А., Волчек В.Н., Петрухин В.С., Прокофьев А.А. Механизмы обеспечения поддержки алгоритмов цифровой обработки речевых сигналов в рекуррентном обработчике сигналов // Системы и средства информатики – М.: ТОРУС ПРЕСС, Вып.20, № 1, 2010. С. 31-47.

[5] Yu. Shikunov, D. Khilko, Yu. Stepchenkov. Hardware and Software Modelling and Testing of Non-Conventional Data-Flow Architecture // Proceedings of the 2016 IEEE North West Russia Section Young Researchers in Electrical and Electronic Engineering Conference (ElConRusNW), 2016. P. 360-364.

[6] Hu. Stepchenkov, D. Khilko, Yu. Diachenko, Yu. Shikunov and D. Shikunov. Software and hardware testing of dataflow recurrent digital signal processor // Proceedings of IEEE East-West Design & Test Symposium (EWDTS'2016), Yerevan, October, 14 - 17, 2016. P. 168-171.
[7] D. Khilko, Yu. Stepchenkov, D. Shikunov, Yu. Shikunov. Recurrent data-flow architecture: technical aspects of implementa-tion and modeling results // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2016. Proceedings / edited by A. Stempkovsky, Moscow, IPPM RAS, 2017. Part II. P. 59-64.
[8] Yu. A. Stepchenkov, Yu. G. Diachenko, D. V. Khilko, V.S. Petrukhin. Recurrent data-flow architecture: features and realization problems // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2016. Proceedings / edited by A. Stempkovsky, Moscow, IPPM RAS, 2017. Part II. P. 52-58.
[9] Yu. Shikunov, Yu. Stepchenkov, D. Khilko, D. Shikunov. Data redundancy problems in data-flow computing and solutions implemented on the recurrent architecture // Proceedings of the 2017 IEEE Russia Section Young Researchers in Electrical and Electronic Engineering Conference (EIConRus), 2017 IEEE. pp. 335 – 338.
[10] Yu. Shikunov, Yu. Stepchenkov, D. Khilko. Recurrent mechanism developments in the data-flow computer architecture // Proceedings of the 2018 IEEE Russia Section Young Researchers in Electrical and Electronic Engineering Conference (EIConRus), 2018 IEEE. pp. 1413 – 1418.
[11] Хилько Д.В. Средства программирования нетрадиционной многоядерной архитектуры и перспективы их развития // Сборник статей II региональной научно-практической конференции «Многоядерные процессоры и параллельное программирование». Барнаул 2012. С.62-70.
[12] Хилько Д.В., Степченков Ю.А. Теоретические аспекты разработки методологии программирования рекуррентной архитектуры // Системы и средства информатики, 2013. Т. 23 №2. С. 133-153.
[13] Хилько Д.В., Шикунов Ю.И. Разработка инструментальной среды проектирования программного обеспечения для рекуррентно-потоковой модели вычислений // Четвертая школа молодых ученых ИПИ РАН, 2013. Сборник докладов. С. 65-77.
[14] Волчек В.Н., Степченков Ю.А., Петрухин В.С., Прокофьев А.А., Зеленов Р.А. Цифровой сигнальный процессор с нетрадиционной рекуррентной потоковой архитектурой // Проблемы разработки перспективных микро- и наноэлектронных систем – 2010. Сборник трудов / под общ. ред. академика А.Л. Стемпковского. М.: ИППМ РАН, 2010. С. 412-417.
[15] Хилько Д. В., Степченков Ю. А., Шикунов Ю.И., Дьяченко Ю.Г. Средства имитационного моделирования потоковой рекуррентной архитектуры (СИМПРА). Версия 2. Свидетельство о государственной регистрации программы для ЭВМ № 2014610123 от 09.01.14.
[16] Хилько Д.В., Степченков Ю.А., Шикунов Ю.И., Шикунов Д.И. Программный комплекс проектирования и моделирования гибридных потоковых рекуррентных систем (СПРУТ). Свидетельство о государственной регистрации программы для ЭВМ № 2017610828 от 18.01.17.
[17] Хилько Д.В., Степченков Ю.А., Шикунов Ю.И. Инструментальная среда проектирования ПО для гибридной архитектуры рекуррентного обработчика сигналов (GAROS IDE). Свидетельство о государственной регистрации программы для ЭВМ № 2015614004 от 01.04.15.
[18] Yu. Shikunov, Yu. Stepchenkov, D. Khilko, G. Orlov. Graph-capsule construction toolset for data-flow computer architecture // Proceedings of the 2018 IEEE Russia Section Young Researchers in Electrical and Electronic Engineering Conference (EIConRus), 2018 IEEE. P. 1419 – 1423.

Development of Capsule Programming Tools for Recurrent Data-Flow Architecture

D.V. Khilko, Yu. A. Stepchenkov, Yu.I. Shikunov, G.A. Orlov

The Institute of Informatics Problems, Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences

dhilko@yandex.ru, ystepchenkov@ipiran.ru, yishikunov@yandex.ru, orlov.jaja@gmail.com

Abstract - This paper covers one of the directions of work on the development of the multicore recurrent data-flow architecture (MRDA) - development of methods and tools for software programming and debugging. Each program in the MRDA environment is a single, recurrently compressed flow of self-sustained data, called a capsule. Furthermore, representing programs as a set of interacting capsules is called the capsular programming paradigm.

Description of already accumulated results in this field is given. The main theoretical result is the creation of methodical support elements for the process of software development and debugging, including a specialized recurrent data-flow programming methodology. The main practical result is the extension of the functionality of an integrated software development environment called GAROS IDE.

Within the framework of the methodology and the GAROS IDE environment, the key tool is the graph-capsule. This element of the methodological support visualizes the distribution of computing unit's resources of the MRDA. To automate its creation, a component was developed to construct graph-capsules in numerical form using the modeling results. The integration of numerical graph capsules into the pro-

gramming methodology made it possible to significantly accelerate the software development process. Nevertheless, debugging and verification with their help involves a number of difficulties.

The paper covers new results obtained during the work on the development of tools for constructing data-flow graphs, and symbol graph-capsules based on them. To solve this problem, a symbol graph-capsule has been integrated into methodology. Then, using existing means of constructing numerical graph capsules, it became possible to automate the processes of constructing symbol graph-capsules and capsule verification.

Utilization of the developed tools allowed us to reduce the average development and debugging time by 2-3 times. Further development of the MRDA programming tools is seen in the construction automation of a data-flow graph from a symbol graph-capsule.

***Keywords* — data-flow architecture, data-flow graph, graph-capsule, capsule programming.**

SUPPORT

The research was carried out within the framework of state task No. 0063-2017-0011.

REFERENCES

[1] E. A. Lee and J. C. Bier. Architectures for Statically Scheduled Dataflow // Parallel Algorithms and Architectures for DSP Applications / edited by Magdy A. Bayoumi. Dordrecht. Kluwer Academic Publishers, 1991. P. 159-190.

[2] V.P. Srini. DFS-SuperMPx: Low-cost Parallel Processing System for Machine Vision and Image Processing // Proc. Third International Conference “Parallel Computing Technologies”, PaCT-95. St. Petersburg, 1995. Vol. 3. P. 356-369.

[3] S. Voigt, M. Baesler, T. Teufel. Dynamically reconfigurable dataflow architecture for high-performance digital signal processing // Journal of Systems Architecture, 2010, no. 56. P. 561-576.

[4] Yu. Stepchenkov, V. Volchek, V. Petrukhin, A. Prokofyev. Hardware maintenance for digital processing of speech signals in the recurrent dataflow processor // Systems and means of informatics – TORUS PRESS, Moscow, 2010. P. 31-47 (in Russian).

[5] Yu. Shikunov, D. Khilko, Yu. Stepchenkov. Hardware and Software Modelling and Testing of Non-Conventional Data-Flow Architecture // Proceedings of the 2016 IEEE North West Russia Section Young Researchers in Electrical and Electronic Engineering Conference (ElConRusNW), 2016. P. 360-364.

[6] Hu. Stepchenkov, D. Khilko, Yu. Diachenko, Yu. Shikunov and D. Shikunov. Software and hardware testing of dataflow recurrent digital signal processor // Proceedings of IEEE East-West Design & Test Symposium (EWDTS’2016), Yerevan, October, 14 - 17, 2016. P. 168-171.

[7] D. Khilko, Yu. Stepchenkov, D. Shikunov, Yu. Shikunov. Recurrent data-flow architecture: technical aspects of implementa-tion and modeling results // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2016. Proceedings / edited by A. Stempkovsky, Moscow, IPPM RAS, 2017. Part II. P. 59-64.

[8] Yu. A. Stepchenkov, Yu. G. Diachenko, D. V. Khilko, V.S. Petrukhin. Recurrent data-flow architecture: features and realization problems // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2016. Proceedings / edited by A. Stempkovsky, Moscow, IPPM RAS, 2017. Part II. P. 52-58.

[9] Yu. Shikunov, Yu. Stepchenkov, D. Khilko, D. Shikunov. Data redundancy problems in data-flow computing and solutions implemented on the recurrent architecture // Proceedings of the 2017 IEEE Russia Section Young Researchers in Electrical and Electronic Engineering Conference (EIConRus), 2017 IEEE. P. 335 – 338.

[10] Yu. Shikunov, Yu. Stepchenkov, D. Khilko. Recurrent mechanism developments in the data-flow computer architecture // Proceedings of the 2018 IEEE Russia Section Young Researchers in Electrical and Electronic Engineering Conference (EIConRus), 2018 IEEE. P. 1413 – 1418.

[11] Khilko D.V. Programming tools of non-conventional multi-core architecture and prospects for their development // Sbornik statej II regional'noj nauchno-prakticheskoj konferencii «Mnogoyadernye processory i parallel'noe programmirovanie» – Proceedings of II regional scientific-practical conference "Multi-core processors and parallel programming". Barnaul 2012. pp.62-70 (in Russian).

[12] Khilko D.V., Stepchenkov Yu.A. Theoretical Aspects of Recurrent Architecture Programming Methodology Development // Sistemy i sredstva informatiki, 2013, Vol. 23 no. 2, P. 133-156 (in Russian).

[13] Khilko D.V., Shikunov Yu. I. Software devolpment environment creation for the recurrent data-flow computational model // Chetvertaya shkola molodyh uchenyh IPI RAN, 2013. Sbornik dokladov – Proceedings of fourth young scientists school IPI RAS. P. 65-77 (in Russian).

[14] Volchek V.N., Stepchenkov Yu.A., Petrukhin V.S., Prokofyev A.A., Zelenov R.A. Digital Signal Processor With Non-Conventional Recurrent Data-Flow Architecture // Problemi razrabotki perspektivnih mikro- i nanoelektronnih system (MES), Moscow, IPPM RAS, 2010. P. 412-417 (in Russian).

[15] Khilko D. V., Stepchenkov Yu. A. N, Shikunov Yu.I., Dyachenko Yu.G. The imitational modeling utilities of recurrent dataflow multicore architecture (SIMPRA). Version two. Certificate of state registration of computer programs № 2014610123 from 09.01.14.

[16] Khilko D. V., Stepchenkov Yu. A., Shikunov Yu. I., Shikunov D. I. The program complex for design and simulation of hybrid data-flow recurrent systems (SPRUT). Certificate of state registration of computer programs № 2017610828 from 18.01.17.

[17] Khilko D.V., Stepchenkov Yu.A., Shikunov Yu.I. The instrumental software development environment for hybrid ar-chitecture of recurrent signal processor (GAROS IDE). Certificate of state registration of computer program No. 2015614004 from 01.04.15.

[18] Yu. Shikunov, Yu. Stepchenkov, D. Khilko, G. Orlov. Graph-capsule construction toolset for data-flow computer architecture // Proceedings of the 2018 IEEE Russia Section Young Researchers in Electrical and Electronic Engineering Conference (EIConRus), 2018 IEEE. P. 1419 – 1423.