ИНФОРМАТИКА И ПРОГРАММИРОВАНИЕ

УДК 681.323
DOI: 10.17223/19988605/40/7

Н.А. Лукин, А.Ю. Филимонов

ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ АЛГОРИТМОВ НА АРХИТЕКТУРАХ ОДНОРОДНЫХ ВЫЧИСЛИТЕЛЬНЫХ СРЕД

Рассматриваются проблемы программирования систем с массированным параллелизмом вычислений и подходы к их решению. Приведены примеры реализации нетрадиционных подходов к построению платформ программирования массивно-параллельных систем.
Ключевые слова: однородные вычислительные среды; императивные и декларативные языки программирования параллельных вычислений; системы потоковой обработки.

Однородными вычислительными средами (ОВС) принято называть специализированные вычислительные системы, которые образованы из одинаково соединенных друг с другом универсальных вычислительных элементов, каждый из которых программно настраивается на выполнение арифметической или логической функции, а также функции соединения с соседними элементами [1, 2]. При этом под универсальным процессорным элементом (ПЭ), как правило, подразумевается однобитный процессор, выполняющий последовательную обработку данных, которые поступают на его входы от соседних процессоров.

Основным достоинством однородных вычислительных сред является изначально присущий им массовый параллелизм вычислений. Кроме того, регулярность ОВС позволяет наращивать ее вычислительные возможности путем простого увеличения размеров матрицы, позволяя при этом организовать достаточно простые методы контроля и диагностики ОВС в процессе производства и эксплуатации. Избыточность, изначально присущая однородной среде, позволяет, благодаря наличию резервных элементов в схеме, обеспечить при необходимости высокую живучесть вычислительной системы.

Указанные выше достоинства ОВС позволяют эффективно использовать их при построении функционально-ориентированных процессоров (ФОП) для систем реального времени. При этом номенклатура реализуемых алгоритмов может быть весьма велика:

– цифровая обработка сигналов и изображений;
– распознавание образов;
– решение систем уравнений (алгебраических и дифференциальных);
– логико-комбинаторные алгоритмы, прежде всего, сортировка и поиск.

Программирование ОВС представляет собой технологический процесс настройки массива ПЭ на выполнение операций обработки и передачи данных, который завершается «укладкой» графа алгоритма в вычислительную решетку. Настоящая работа посвящена рассмотрению некоторых компонентов этого технологического процесса, а также проблемам, с которыми сталкиваются создатели подобных платформ.

В первом разделе рассматриваются вопросы применимости универсальных языков для программирования параллельных вычислительных систем и сравниваются основные подходы, которые применяются при создании соответствующих программных платформ.

Второй раздел посвящен анализу опыта применения традиционного подхода к созданию специальных языков программирования параллельных вычислительных систем.

Специфические особенности программирования вычислительных систем, управляемых потоками данных, рассматриваются в третьем разделе.

В четвертом разделе рассмотрен опыт реализации комплексного технологического процесса программирования потоковых вычислительных систем.

В заключении дан краткий анализ перспектив применения современных параллельных программных средств для программирования ОВС.

1. Применение традиционных подходов для программирования однородных вычислительных систем

1.1. Адаптация традиционных языков программирования

Достоинством данного подхода является возможность плавной миграции всего комплекса современных программных продуктов на новые высокопроизводительные параллельные вычислительные системы. Однако на сегодняшний день многочисленные проекты по созданию универсальных векторизующих компиляторов показывают обнадеживающие результаты только при решении весьма ограниченного класса задач [3], ни один из них не увенчался сколько-нибудь заметными успехами в распараллеливании последовательной программы произвольного вида [4–6].

«Параллельная невыразительность» традиционных языков программирования может быть устранена путем ввода специальных лексических конструкций, которые позволяют программисту описывать и организовывать параллельные вычислительные процессы. Как показывает опыт, при практической реализации данного подхода могут возникать очень серьезные трудности, которые применительно к параллельному языку HPF (High Perfomance Fortran) описаны в отчетной работе [7] группой основных разработчиков. Примечательно, что программисты в качестве основных недостатков указывали отсутствие возможности оценки (прогнозирования) времени выполнения программы HPF на конкретной вычислительной архитектуре.

Сегодня наибольшее распространение получило применение параллельных расширений (диалектов) существующих последовательных языков для программирования вычислительных структур, которые имеют заранее определенную архитектуру, как правило, это кластеры универсальных процессоров или гибридные вычислители, состоящие из универсальных вычислителей и векторных сопроцессоров (GPU/CPU). Поскольку в этом случае вычислительные комплексы содержат компоненты с классической архитектурой, дополнительные преимущества при их программировании получают именно те платформы, которые максимально используют возможности последовательных языков программирования (Open CL, Open MP, UPC и т.д.).

1.2. Языки параллельного программирования

Работы по созданию универсальных параллельных языков проводились компаниями Cray, SUN и IBM в период с 2002 по 2010 г. по заказу агентства перспективных проектов (Defense Advanced Research Projects Agency – DARPA) Министерства обороны США в рамках программы создания высокопроизводительных вычислительных систем (High Productivity Computing Systems – HPCS) и завершились представлением языков программирования Chapel (2009 г.), Fortress (2008 г.) и X10 (2004 г.) [8]. Эти языки основаны на модели разделяемого адресного пространства Partitioned Global Address Space (PGAS), причем X10 в большей степени, чем Chapel и Fortress, опирается на традиционный синтаксис и базовые понятия последовательных языков. Тот факт, что даже самый близкий к традиционным из указанных языков (X10) не получил заметного распространения, разработчики объясняют неготовностью сообщества программистов жертвовать удобством процесса программирования ради повышения его эффективности [9]. В то же время приведенные в [10, 11] выводы о большей выразительности и универсальности Chapel по сравнению с X10 указывают на ограниченные возможности применения традиционных методов для программирования систем с массовым параллелизмом вычислений.

В условиях отсутствия универсальных инструментов параллельные проблемно-ориентированные языки (ПОЯ) (Domain Specific Language – DSL) позволяют достичь требуемой эффективности и выразительности программирования за счет существенного сокращения области применения языков [12].

Применение ПОЯ, как правило, предполагает полную осведомленность программиста об особенностях организации вычислительного процесса в программируемом устройстве, что позволяет наиболее полно использовать специфические возможности данного вычислителя [14]. Поэтому ПОЯ в первую очередь оказываются полезными в качестве временного решения на этапе макетирования вычислительных систем [13].

1.3. Перспективы применения традиционных программных средств для программирования ОВС

Практически все рассмотренные выше языки, как и подавляющее большинство современных языков программирования, являются императивными и основаны на принципе обработки данных потоками команд (Control Flow). Повсеместное распространение императивных языков программирования определяется их точным соответствием требованиям и особенностям организации вычислительного процесса на вычислителях с классической архитектурой от фон Неймана. Однако в тех случаях, когда архитектура программируемого вычислителя отличается от классической, это преимущество императивных языков программирования превращается в недостаток, который будет проявляться тем сильнее, чем сильнее различаются архитектуры вычислителей. Например, применение императивных языков для программирования систем с архитектурой Data Flow, которую образует массив исполнительных устройств и устройств сопоставления соединенных с контекстно-адресуемым запоминающим устройством, вызывает серьезные проблемы, связанные с побочными эффектами и ограниченной доступностью данных [15]. Примечательно, что в последнее время создаются серьезные проекты, основанные на модели вычислений Data Flow. Так, в частности, разработчики крупного европейского проекта TERAFLUX по созданию высокопроизводительных вычислительных структур отмечают, что применение модели организации вычислительного процесса Data Flow позволяет максимально выявлять и использовать параллелизм вычислений и преодолеть ряд существенных ограничений, присущих традиционной модели Control Flow [16]. Эти результаты, безусловно, следует принимать во внимание при выборе системы программирования ОВС, поскольку модель организации вычислительного процесса Data Flow гораздо лучше, чем Control Flow, подходит для описания вычислений в ОВС, а форма графа потоков данных, используемая для указания информационных связей между операндами выражения, вычисляемого в системе DataFlow, по сути, представляет собой архитектуру ОВС, которая обеспечивает вычисление данного выражения.

2. Применение альтернативных подходов для программирования однородных вычислительных систем

2.1. Программирование вычислений, управляемых потоками данных (Data Flow)

Обобщенные в отчете [15] результаты многочисленных исследований показывают, что ключевым фактором, который способен обеспечить требуемую эффективность программирования вычислителей, управляемых потоком данных, являются радикальные изменения понятия переменной и процедуры присваивания. В обзоре [17] приведены результаты использования адаптированного языка С для программирования алгоритмов обработки изображений на реконфигурируемом вычислителе, построенном на FPGA семейства Virtex (Xilinx, США). Язык программирования SA-C представляет собой диалект, наиболее характерными отличиями от базового языка С являются:

– принцип однократного присваивания значений переменным (single assignment);

– исключение операций с компонентами общей памяти;

– ориентированность на выражения (expression oriented – любая языковая конструкция обязательно возвращает как минимум одно значение).

В ряде случаев для обеспечения выразительности и адекватности описания вычислительного процесса в потоковых вычислительных системах с массивным параллелизмом платформы программирования, основанные на императивных языках, вынуждены перенимать базовые принципы декларативной концепции программирования [18, 19]. К преимуществам использования декларативных языков для

программирования однородных вычислительных систем следует отнести отсутствие конструкций и понятий, которые не имеют адекватного отображения в ОВС, – такие как общая память, переменные, указатели, индексы, циклы. Главными факторами, которые сдерживают применение и распространение декларативной парадигмы программирования, долгое время оставались отсутствие зрелых платформ программирования и психологическая неготовность сообщества программистов к радикальному изменению своего инструментария [20, 21]. Однако за последние несколько лет отношение разработчиков программного обеспечения к декларативным языкам стало меняться, что привело к появлению гибридных языков и новых декларативных платформ для программирования параллельных вычислителей.

2.2. Не-императивные языки программирования

Анализ динамики (по данным Wikipedia) создания новых языков программирования за последние 15 лет показывает устойчивое уменьшение среди новых языков программирования доли чистых императивных языков (–24%) и такой же устойчивый рост доли гибридных языков, которые поддерживают декларативный стиль программирования (+34%). Объяснить это можно двумя причинами:

1) отсутствием заметного прогресса в применении императивных программных систем для программирования вычислителей с массивным параллелизмом вычислений;

2) появлением и все более широким внедрением распределенных облачных и туманных вычислений (cloud, fog computing); основной тенденцией в организации вычислительных процессов становится уход от классической схемы фон Неймана, что уменьшает область эффективного применения чистых императивных языков программирования.

Гибридные языки, по замыслу разработчиков, должны сочетать в себе основные достоинства указанных выше парадигм программирования:

– наглядность (простота восприятия) программного кода, наличие мощных инструментов для его создания и отработки, которые свойственны современным императивным языкам программирования;

– автоматическое выявление ветвей параллельных вычислений, отсутствие побочных эффектов при их выполнении, которые свойственны чистым декларативным языкам программирования.

В качестве примеров реализации подобных решений в последнем разделе будут рассмотрены платформа программирования FPGA-сопроцессоров, основанная на гибридном языке Mitrion-C, и платформа программирования параллельных вычислений, основанная на декларативном языке SquenceL. Поскольку данные платформы способны обеспечить массовый параллелизм вычислительного процесса, основные принципы их построения и опыт реализации могут быть использованы при создании систем программирования однородных вычислительных сред.

3. Платформы программирования систем с массовым параллелизмом вычислений

3.1. Платформа, основанная на гибридном языке программирования

Платформа компании Mitrion включает в себя все компоненты, которые необходимы для реализации полного цикла программирования сопроцессоров, построенных на FPGA:

– транслятор Mition-C, который предназначен для лингвистической верификации исходного кода, написанного на языке программирования высокого уровня, и формирования объектного кода программы;

– виртуальный процессор Mitrion, который обеспечивает отладку объектного кода программы на интерпретаторе с использованием FPGA – независимого эмулятора;

– блоки конфигурирования (Processor Configuration Unit, SPR Tool), которые выполняют размещение вычислительных компонентов на FPGA, создание соединений между ними (Place & Route) и обеспечивают формирование загрузочного кода (FPGA bitstream – код укладки программы) с учетом специфики выбранной аппаратной платформы FPGA.

По замыслу разработчиков платформы Mitrion, язык программирования должен обеспечивать интеллектуальную поддержку программиста в обеспечении параллельного выполнения программ и в то же время быть достаточно простым для изучения и использования [22].

Поскольку лексика языка Mition-C не опирается явно на принятые в декларативных языках механизмы описания шаблонов функций, она, на первый взгляд, гораздо ближе к лексике императивных языков программирования. Декларативный характер Mition-C подчеркивается также принятыми в данном языке программирования способами определения и использования переменных; присвоение значения переменной в языке программирования Mition-C представляет собой не выражение, как это принято в императивных языках программирования, а утверждение. Отличие заключается в том, что единожды определенное при помощи процедуры присваивания значение переменной Mition-C может затем многократно использоваться (например, для определения значений других переменных), но, в то же время, не может изменяться (переопределяться) [22].

Особый интерес при этом, безусловно, представляют оценки выразительности нетрадиционных лингвистических конструкций языка Mition-C. Действительно, многие испытатели отмечают, что им требовалось определенное время для того, чтобы привыкнуть к специфическим особенностям программирования на Mition-C [23, 24]. Большинство из них, однако, при этом указывают, что после завершения адаптации процесс программирования уже не у них вызывал значительных затруднений. При этом программисты по достоинству оценили оригинальные решения разработчиков языка, которые обеспечивают возможность управления точностью представления данных [23] и функциональную полноту платформы, которая обеспечивает выполнение предварительной отладки программных решений на интерпретаторе [25].

Предварительные сравнительные оценки средств программирования FPGA-сопроцессоров [24, 26–28] показывают, что платформа Mitrion (язык Mitrion–C) имеет преимущество в эффективности по отношению к традиционным программным средствам (Impulse-C и Dime-C), но уступает как средствам аппаратного (VHDL и Verilog), так и графического (DLPlogic) программирования. Рассматривая эти результаты, не стоит забывать о том, что только программные платформы способны обеспечить мобильность программного кода и востребованную сегодня возможность его независимой отладки [25, 29].

Не вызывает сомнения, что перспективы применения для программирования ОВС платформ, основанных на гибридных языках, во многом будут определяться совокупностью взаимно исключающих требований удобства программирования и степени абстрагирования от архитектуры вычислительного устройства. Основываясь на изложенном выше (здесь уместно будет вспомнить опасения [5] разработчиков Chapel), можно предположить, что чем больше императивных возможностей разработчик гибридного языка оставит программисту, тем менее приспособленный для параллельного выполнения код получит транслятор. Поэтому значительно более перспективным представляется применение для программирования ОВС решений, которые основаны на чистых декларативных языках программирования. Как будет показано далее, лаконичность программных конструкций в совокупности с развитыми процедурами выявления параллелизма позволяет успешно решать многие проблемы, с которыми могут столкнуться разработчики систем программирования однородных вычислительных сред.

3.2. Платформа, основанная на декларативном языке программирования

Очевидно, что максимально эффективной (по определению) будет программа, которая наиболее точно адаптирована к архитектуре вычислительной системы. Важно при этом определить, на какой именно стадии создания программы и каким образом должна происходить эта адаптация. Наиболее неудачным представляется вариант, когда программа изначально строится на основе типового алгоритма, который не учитывает ни структуру обрабатываемых данных, ни архитектуру вычислительной системы, а потом под них адаптируется программистом или компилятором. Проблема заключается в том, что алгоритм сам по себе уже является адаптацией выполняемой задачи, как правило, для классической архитектуры фон Неймана, поэтому его автоматическая адаптация компилятором сводится к решению описанной ранее проблемы построения «универсального векторизующего компилятора» [3], а процесс ручной адаптации рано или поздно возвращает программиста к условиям поставленной задачи. Значительно более продуктивным поэтому представляется процесс построения программы непосредственно из условий решаемой задачи с одновременным учетом структур обрабатываемых данных и доступных функций.

Во многом подобный процесс реализован в системе программирования, основанной на языке SequenceL, представляющем собой чистый декларативный язык высокого уровня, который предназначен для программирования распределенных и потоковых вычислений [29]. Отличительными особенностями этого языка программирования является очень лаконичный синтаксис и развитые механизмы неявного распараллеливания вычислений.

По замыслу создателей, эти особенности языка SequenceL позволяют избавить программиста от алгоритмического планирования вычислительного процесса, что особенно важно для программирования реконфигурируемых систем с массированным параллелизмом вычислений, когда структура (архитектура) вычислительного устройства не определена на момент создания программы. В то же время лаконичная семантика SequenceL поощряет программиста формулировать решаемые задачи в форме, которая способствует выявлению и использованию скрытого параллелизма вычислений [28].

Как следует из названия языка, базовым элементом данных в SequenceL являются последовательности, элементами которых также могут быть последовательности. Скаляр представляется в SequenceL в виде 0-мерной последовательности, которая состоит из одного элемента, вектор представляется одномерной последовательностью, матрица – двумерной (последовательность векторов) и т.д. Размерность (Type – тип) аргументов указывается в определениях функций и операторов SequenceL и играет ключевую роль для распараллеливания их вычислений [30]. В случае, если тип аргумента при обращении к некоторой функции превышает значение типа, которое было задано при ее определении, для этого аргумента фиксируется состояние «превышение типа» (overtyping), которое влечет за собой выполнение для него и для данной функции комплекса согласующих операций – NTD (Normalize-Transpose-Distribute). Назначение операций NTD поясняется приводимым ниже примером умножения (скалярная операция) на вектор, который взят из [30]:

10 * [1,2,3]
(normalize) → [[10,10,10],[1,2,3]]
(transpose) → [[10, 1],[10,2],[10,3]]
(distribute) → [[10*1],[10*2],[10*3]]
→ [10,20,30]

Применение комплекса NTD в языке SequenceL позволяет на стадии трансляции выражений автоматически выявлять скрытый параллелизм их вычисления и одновременно избавляет программиста от необходимости использования дополнительных программных конструкций для управления вычислительным процессом [30].

Интересно отметить, что хотя описанные выше основные принципы построения языка программирования SequenceL были представлены разработчиками еще в прошлом веке [28], заметное распространение этот язык начал получать только в последние несколько лет. В апреле 2012 г. основанная разработчиками языка компания Texas Multicore Technologies (TMT) представила бета-версию коммерческого компилятора SequenceL, который использовался в качестве пре-процессора к языку C++ для автоматического распараллеливания вычислений. В декабре 2014 г. TMT, уже будучи одним из партнеров AMD в области разработки программного обеспечения, представила новую версию своей платформы, которая обеспечивала совместное использование SequenceL с такими языками программирования, как Java, C#, Python и Fortran, что обеспечило успешное применение для программирования распределенных вычислений. Как показывают испытания, сегодня платформа программирования TMT, которая включает в себя интерпретатор, отладчик, компилятор SequenceL и системные библиотеки для поддерживаемых языков программирования, позволяет автоматически создавать программы, которые превосходят написанные вручную по эффективности использования вычислительных ресурсов [31].

Опыт разработки и внедрения платформы программирования TMT подтверждает представленное выше предположение о растущем интересе к системам, которые основаны на не-императивных языках программирования в связи с наблюдаемыми системными изменениями в организации вычислительного процесса (тенденция ухода от классической схемы фон Неймана).

3.3. Общие требования к платформе программирования ОВС

Представленный выше анализ текущего состояния и тенденций развития технологий программирования систем с массовым параллелизмом вычислений позволяет определить основные требования к составу и функциям компонентов платформы программирования однородных вычислительных сред. Основными компонентами такой платформы должны быть собственно система программирования и система построения загрузочного кода (компоновки).

Система программирования, по нашему мнению, должна быть построена на декларативном языке и обеспечивать синтаксическую и семантическую отладку управления точностью и разрядностью вычислений.

Система трансляции предназначена для формирования объектного кода разработанной программы с учетом особенности организации вычислительного процесса в ОВС и специфики их компоновки, а при необходимости – автоматического эквивалентного преобразования объектного кода.

Система компоновки должна быть построена по блочно-модульному принципу с тем, чтобы обеспечить итеративную форму адаптации объектного кода для оптимизации процесса построения загрузочного кода и возможность дальнейшего развития системы.

Наличие, кроме естественной временной метрики обработки данных (потактные вычисления потоков данных на массивах ПЭ), еще и явно выраженной пространственной метрики (прямоугольная матрица локально связанных ПЭ) обусловливает «геометрический» характер функционирования компилятора. Он должен учитывать возможные траектории распространения потоков данных в двумерном пространстве ПЭ и оптимизировать длины этих траекторий. Это принципиально отличает принципы построения и функционирования компиляторов ОВС от компиляторов современных систем параллельной обработки данных.

Отмеченные особенности ложатся в основу создания технологии программирования ОВС, при этом первым этапом становится создание языка и принципов построения ОВС-компилятора.

Заключение

ОВС представляют собой класс архитектур с массовым параллелизмом, реализуемым на двумерных массивах локально связанных процессорных элементов. По принципу обработки информации это реконфигурируемые машины потоков данных с однородной структурой. Поэтому основными чертами технологии программирования ОВС являются декларативный принцип языка программирования, структурно-функциональный принцип работы компилятора, учитывающий топологию ОВС и возможность программного управления архитектурой ОВС (реконфигурации) при возникновении соответствующих условий и проблем.

ЛИТЕРАТУРА

1. Дмитриенко Н.Н., Каляев И.А., Левин И.И., Семерников Е.А. Семейство многопроцессорных вычислительных систем с динамически перестраиваемой архитектурой // Вестник компьютерных и информационных технологий. 2009. № 6, 7.
2. Лукин Н.А. Реконфигурируемые процессорные массивы для систем реального времени: архитектуры, эффективность, области применения // Известия ТРТУ. 2004. № 9. С. 36–45.
3. Maleki S., Yaoqing Gao, Garzaran M.J., Wong T., Padua D.A. An Evaluation of Vectorizing Compilers Parallel Architectures and Compilation Techniques (PACT) // 2011 International Conference. 2011. 10–14 Oct. P. 372–382.
4. The Landscape of Parallel Computing Research: A View from Berkeley – Technical Report No. UCB/EECS-2006-183.
5. Chamberlain B.L., Callahan D., Zima H.P. Parallel Programmability and the Chapel Language // Journal of High Performance Computing Applications. 2007. August. V. 21, No. 3. P. 291–312.
6. Bal H.E. Matthew Haines Approaches for Integrating Task and Data Parallelism // Journal IEEE Concurrency – July. 1998. V. 6, Issue 3. P. 74–84.
7. Kennedy K., Koelbel C., Zima H. The Rise and Fall of High Performance Fortran: An Historical Object Lesson // Communications of the ACM. November. 2011. V. 54, No. 11. P. 74–82.
8. Dongarra J., Graybill R., Harrod W., Lucas R., Lusk E., Luszczek P., Mcmahon J., Snavely A., Vetter J., Yelick K., Alam S., Campbell R., Carrington L., Tzu-Yi Chen, Khalili O., Meredith J., Tikir M. DARPA's HPCS Program: History, Models, Tools // Languages Advances in Computers. 2008. V. 72. P. 1–100.

9. Richards J., Brezin J. A decade of progress in parallel programming productivity // Communications of the ACM. November. 2014. V. 57, Issue 11. P. 60–66.
10. Weiland M. Chapel, Fortress and X10: Novel languages for HPC. Technical Report from the HPCx Consortium. 2007.
11. Feldman M. Closing the Parallelism Gap with the Chapel Language // HPCwire. 2008. November 19.
12. Voemel C. How to talk new computers into working harder. Technical Reports № 603. ETH Zurich. ICOS–08. 2008.
13. Титов В.Г., Лукин Н.А. Язык макросов для программирования однородной вычислительной среды MiniTera II // Известия Томского политехнического университета. 2008. Т. 313, № 5. С. 93–96.
14. Васильев С.С., Новосельцев В.Б. Об использовании в программировании проблемно-ориентированных языков // Известия Томского политехнического университета. 2008. Т. 313, № 5. С. 68–72.
15. Johnston W.M., Hanna J.R.P. and Millar R.J. Advances in dataflow programming languages // ACM Comput. Surv. 2004. V. 36, No. 1.
16. Solinas M., Badia R.M., Bodin F., Cohen A. The TERAFLUX Project: Exploiting the DataFlow Paradigm in Next Generation Teradevices // 16th Euromicro Conference on Digital System Design. 2013.
17. Najjar W.A., Bohm W., Draper B.A., Hammes J., Rinker R., Beveridge J.R., Chawathe M., Ross C. High-Level Language Abstraction for Reconfigurable Computing // IEEE Computer. 2003. Aug. P. 63–69.
18. Böhm W., Hammes J., Draper B., Chawathe M., Ross C., Rinker R., Najjar W. Mapping a Single Assignment Programming Language to Reconfigurable Systems // Supercomputing 21. 2002. P. 117–130.
19. Hammarlund P., Lisper B. Data Parallel Programming: A Survey and a Proposal for a New Model // Royal Institute of Technology, Department of Teleinformatics. Tech. Rep.TRITA-IT-9308. September. 1993.
20. Wadler P. Why no one uses functional languages // ACM SIGPLAN Notices. 1998. V. 33, No. 8. P. 23–27.
21. Budiu M., Goldstein S.C. Compiling application-specific hardware // In International Conference on Field Programmable Logic and Applications (FPL). Montpellier (La Grande-Motte). France. 2002. September. P. 853–863.
22. Mitrion Users' Guide 2.0.3-001. 2009.
23. Kryjak T., Gorgoń M. Parallel implementation of local thresholding in Mitrion-C // AGH University of Science and Technology, Al. Mickiewicza. Department of Automatics. Laboratory of Biocybernetics Poland. Cracow. 2010. September. V. 20, Issue 3. P. 571–580.
24. El-Araby E., Taher M., Abouellail M., ElGhazawi T. and Newby G.B. Comparative Analysis of High Level Programming for Reconfigurable Computers: Methodology and Empirical Study // III Southern Conference on Programmable Logic (SPL 2007). Argentina. Mar del Plata, 2007.
25. Koo J., Fernandez D., Haddad A. Gross W.J. Evaluation of a High-Level-Language Methodology for High-Performance Reconfigurable Computers // Application-specific Systems. Architectures and Processors ASAP – IEEE International Conf. 2007.
26. Park S.J., Shires D., Henz B. Reconfigurable Computing: Experiences and Methodologies // Army Research Laboratory Aberdeen Proving Ground. MD 21005-5067 ARL-TR-4358. 2008. January.
27. Kindratenko V.V., Brunner R.J., Myers A.D. Mitrion-C Application Development on SGI Altix 350/RC100 // IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'07). 2006. April 23–25.
28. Cooke D., Gelfond M., Urban J. Computer Language Advances // Handbook of Software Engineering and Knowledge Representation. World Scientic. 2001. V. 1. P. 121.
29. Cooke D., Rushton N. SequenceL – An Overview of a Simple Language // Proceedings of The 2005 International Conference on Programming Languages and Compilers PLC. 2005. USA. Nevada. Las Vegas. 2005. June 27–30.
31. Daniel E., Cooke J., Rushton N., Nemanich B., Watson R., Andersen P. Normalize, transpose, and distribute: An automatic approach for handling nonscalars // ACM Trans. Program. Lang. Syst. 2008. V. 30, No. 2.
32. Rushton N., Norton D. SequenceL for C++ programmers saves time and angst // Embedded Computing Design August. 2014.

***Лукин Николай Алексеевич*,** канд. техн. наук. E-mail: n.a.lookin@urfu.ru
Институт машиноведения УрО РАН, Уральский федеральный университет (г. Екатеринбург)
Филимонов Александр Юрьевич. E-mail: af.1015@yahoo.com
Уральский федеральный университет (г. Екатеринбург)

Поступила в редакцию 25 июня 2017 г.

Lookin Nick A. (Institute of Engineering Science, Ural Branch of Russian Academy of Science; Ural Federal University, Yekaterinburg, Russian Federation).
Filimonov Alexander Yu. (Ural Federal University, Yekaterinburg, Russian Federation).

Software technologies for homogenous computing environment.

Keywords: homogenous computing environment; imperative and declarative programming languages for parallel computations; data flow systems.

DOI: 10.17223/19988605/40/7

Homogeneous computing environment (HCE) it's special-purpose computer systems, which are represents as the mesh-connected identical processing elements (PE), each of which is configured to perform arithmetic or logic functions and certain kind connections with the "neighboring" PE. HCE programming is a PE array set-up process for the implementation of data processing and transmission algorithm. The main result of programming is "laying" algorithm graphs on lattice of PE.

To date, numerous projects dedicated to applying of general-purpose vectorizing compilers for adaptation the common programming languages to systems with massively parallel data processing have not been any success with respect to the parallel programs of any kind. At the same time, some of new languages for parallel programming are not getting noticeable proliferation, indicating that the limited capacity of traditional methods for systems programming of massively parallel computing.

The results of numerous researches show that the one of key factor for the efficiency of parallel programming is a radical change of the concept of variable and assignment procedures. That is why the platform of software for parallel data processing based on imperative programming languages, are forced to adopt the basic principles of declarative programming concepts.

As examples of such language implementation the FPGA-coprocessor software based on a hybrid language Mitrion-C programming and parallel computing platform based on a declarative language SquenceL are described in article.

Analysis of the current status and trends of software techniques for massively parallel computing allows define the basic requirements for the structure and functions of the components of the software platform of HCE. The main components of such a platform are programming system and the subsystem of code layout.

The programming system should be based on the declarative language and provide debugging on syntactic and semantic levels and accuracy control. The layout system should be based on the modular principle in order to ensure the iterative adaptation of the object code for optimizes the building boot code and the possibility of further development of the system.

In case of HCE both time (data streams computation on PE arrays) and space (rectangular matrix locally connected PE) metric of computations are taking place. It is causes the "geometric" nature of the operation of the compiler. It must take into account the possible propagation path data streams in two-dimensional space of PE and optimize the lengths of these "trajectories". This fact fundamentally distinguishes the principles of construction and functioning of compilers for HCE and usual parallel computer systems.

REFERENCES

1. Dmitrienko, N.N., Kalyaev, I.A., Levin, I.I. & Semernikov, E.A. (2009) Multiprocessor computer systems with dynamically reconfigurable architecture. *Vestnik komp'yuternykh i informatsionnykh tekhnologiy – Herald of Computer and Information Technologies.* no. 6,7. (In Russian).
2. Lukin, N.A. (2004) Rekonfiguriruemye protsessornye massivy dlya sistem real'nogo vremeni: arkhitektury, effektivnost', oblasti primeneniya [Reconfigurable Processor Arrays for Real-Time Systems: Architectures, Efficiency, Application]. *Izvestiya TRTU.* 9. pp. 36-45. (In Russian).
3. Maleki, S., Yaoqing Gao, Garzaran, M.J., Wong, T. & Padua, D.A. (2011) An Evaluation of Vectorizing Compilers Parallel Architectures and Compilation Techniques (PACT). *PACT '11 Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques*. pp. 372–382. DOI: 10.1109/PACT.2011.68
4. Asanović, K., Bodik, R., Catanzaro, B.C., Gebis, J.J., Husbands, P., Keutzer, K., Patterson, D.A., Plishker, W.L., Shalf, J., Williams, S.W. & Yelick, K.A. (2006) *The Landscape of Parallel Computing Research: A View from Berkeley*. Technical Report No. UCB/EECS-2006-183.
5. Chamberlain, B.L., Callahan, D. & Zima, H.P. (2007) Parallel Programmability and the Chapel Language. *Journal of High Performance Computing Applications*. 21(3). pp. 291–312. DOI: 10.1177/1094342007078442
6. Bal, H.E. & Haines, M. (1998) Approaches for Integrating Task and Data Parallelism. *Journal IEEE Concurrency.* 6(3). pp. 74–84. DOI: 10.1109/4434.708258
7. Kennedy, K., Koelbel, C. & Zima, H. (2011) The Rise and Fall of High Performance Fortran: An Historical Object Lesson. *Communications of the ACM.* 54(11). pp. 74–82. DOI: 10.1145/29873.29875
8. Dongarra, J., Graybill, R., Harrod, W., Lucas, R., Lusk, E., Luszczek, P., Mcmahon, J., Snavely, A., Vetter, J., Yelick, K., Alam, S., Campbell, R., Carrington, L., Tzu-Yi Chen, Khalili, O., Meredith, J. & Tikir, M. (2008) DARPA's HPCS Program: History, Models, Tools. *Languages Advances in Computers*. 72. pp. 1–100. DOI: 10.1016/S0065-2458(08)00001-6
9. Richards, J. & Brezin, J. (2014) A decade of progress in parallel programming productivity. *Communications of the ACM.* 57(11). pp. 60–66. DOI: 10.1145/2669484
10. Weiland, M. (2007) *Chapel, Fortress and X10: Novel languages for HPC*. Technical Report from the HPCx Consortium.
11. Feldman, M. (2008) Closing the Parallelism Gap with the Chapel Language. *HPCwire*. November 19.
12. Voemel, C. (2008) *How to talk new computers into working harder*. Technical Reports №603. ETH Zurich. ICOS□08.
13. Titov, V.G. & Lukin, N.A. (2008) The Macro's Language for Programming of HCE MiniTera II. *Izvestiya Tomskogo politekhnicheskogo universiteta – Bulletin of Tomsk Polytechnic University.* 313(5). pp. 93–96. (In Russian).
14. Vasiliev, S.S. & Novoseltsev, V.B. (2008) On the use of domain-specific languages for programming. *Izvestiya Tomskogo politekhnicheskogo universiteta – Bulletin of Tomsk Polytechnic University.* 313(5). pp. 68–72. (In Russian).
15. Johnston, W.M., Hanna, J.R.P. & Millar, R.J. (2004) Advances in dataflow programming languages. *ACM Comput. Surv.* 369(1). DOI: 10.1145/1013208.1013209
16. Solinas, M., Badia, R.M., Bodin, F. & Cohen, A. (2013) The TERAFLUX Project: Exploiting the DataFlow Paradigm in Next Generation Teradevices. *16th Euromicro Conference on Digital System Design.* pp. 1–6. DOI: 10.1109/DSD.2013.39. 272
17. Najjar, W.A., Bohm, W., Draper, B.A., Hammes, J., Rinker, R., Beveridge, J.R., Chawathe, M. & Ross, C. (2003) High-Level Language Abstraction for Reconfigurable Computing. *IEEE Computer.* Aug. pp. 63–69. DOI: 10.1109/MC.2003.1220583
18. Böhm, W., Hammes, J., Draper, B., Chawathe, M., Ross, C., Rinker, R. & Najjar, W. (2002) Mapping a Single Assignment Programming Language to Reconfigurable Systems. *Supercomputing 21*. pp. 117–130. DOI: 10.1023/A:1013623303037
19. Hammarlund, P. & Lisper, B. (1993) Data Parallel Programming: A Survey and a Proposal for a New Model. *Royal Institute of Technology, Department of Teleinformatics. Tech. Rep.*TRITA-IT-9308. September.

20. Wadler, P. (1998) Why no one uses functional languages. *ACM SIGPLAN Notices*. 339(8). pp. 23–27. DOI: 10.1145/286385.286387
21. Budiu, M. & Goldstein, S.C. (2002) Compiling application-specific hardware. *International Conference on Field Programmable Logic and Applications (FPL)*. Montpellier (La Grande-Motte). France. September. pp. 853–863.
22. *Mitrion Users' Guide* 2.0.3-001. (2009).
23. Kryjak, T. & Gorgoń, M. (2010) Parallel implementation of local thresholding in Mitrion-C. *AGH University of Science and Technology, Al. Mickiewicza*. 20(3). pp. 571–580. DOI: 10.2478/v10006-010-0042-2
24. El-Araby, E., Taher, M., Abouellail, M., ElGhazawi, T. & Newby, G.B. (2007) Comparative Analysis of High Level Programming for Reconfigurable Computers: Methodology and Empirical Study. *III Southern Conference on Programmable Logic (SPL 2007)*. Argentina. Mar del Plata.
25. Koo, J., Fernandez, D., Haddad, A. & Gross, W.J. (2007) Evaluation of a High-Level-Language Methodology for High-Performance Reconfigurable Computers. *Application – specific Systems. Architectures and Processors ASAP – IEEE International Conf.* DOI: 10.1109/ASAP.2007.4429954
26. Park, S.J., Shires, D., Henz, B. (2008) Reconfigurable Computing: Experiences and Methodologies. *Army Research Laboratory Aberdeen Proving Ground*. MD 21005-5067 ARL-TR-4358. January.
27. Kindratenko, V.V., Brunner, R.J. & Myers, A.D. (2006) Mitrion-C Application Development on SGI Altix 350/RC100. *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'07)*. April 23-25. pp. 1–6. DOI: 10.1109/FCCM.2007.17
28. Cooke, D., Gelfond, M. & Urban, J. (2001) Computer Language Advances. *Handbook of Software Engineering and Knowledge Representation*. 1. pp. 121.
29. Cooke, D. & Rushton, N. (2005) Sequence L – An Overview of a Simple Language. *Proceedings of The 2005 International Conference on Programming Languages and Compilers PLC*. USA. Nevada. Las Vegas. June 27–30, 2005.
31. Daniel, E., Cooke, J., Rushton, N., Nemanich, B., Watson, R. & Andersen, P. (2008) Normalize, transpose, and distribute: An automatic approach for handling nonscalars. *ACM Trans. Program. Lang. Syst.* 30(2). DOI: 10.1145/1330017.1330020
32. Rushton, N. & Norton, D. (2014) SequenceL for C++ programmers saves time and angst. *Embedded Computing Design*. August.