Ресурсонезависимое программирование гибридных реконфигурируемых вычислительных систем

А.И. Дордопуло[1], И.И. Левин[1]

ООО «НИЦ супер-ЭВМ и нейрокомпьютеров»[1]

Анализ тенденций развития современных процессоров показывает, что дальнейшее развитие процессорной архитектуры возможно только по пути гибридизации - объединения на одном кристалле процессора и специализированного вычислителя. ПЛИС является наиболее перспективным направлением гибридизации процессоров как по технологическим причинам, по тепловыделению и возможным интерфейсам доступа к памяти и синхронизации вычислений, так и по возможности адаптации архитектуры под структуру решаемой задачи. Для эффективного программирования таких гибридных реконфигурируемых вычислительных устройств необходимы новые методы и средства программирования, объединенные в технологию ресурсонезависимого программирования, рассмотрению которой посвящена настоящая статья.

Ключевые слова: язык программирования высокого уровня, программирование вычислительных систем гибридного типа, технология ресурсонезависимого программирования, редукция производительности.

1. Введение

Динамика роста степени интеграции современных процессоров и вычислительных систем на их основе замедлилась как минимум вдвое по сравнению с действовавшим более 30 лет законом Мура [1]. Резерв повышения производительности процессоров за счет только технологических решений практически исчерпан: планируемый предел технологического процесса для выпуска современных процессоров в 10 и 7 нм будет достигнут к 2019-2020 года [2], после чего физические ограничения на размер транзисторов и проводников не позволят наращивать объем кристалла за счет техпроцесса. Возможности архитектурных улучшений процессоров также практически исчерпаны, большинство технологий последних 10 лет являются, по своей сути, маркетинговыми ходами, сопровождающими выход любого нового продукта на рынок и не вносящими существенный вклад в вычислительную производительность процессора. Многоядерная архитектура процессоров, которую считали «точкой перегиба всей индустрии» [3], не в полной мере оправдала возложенные на нее надежды и ожидания, поскольку вскрыла еще более серьезные проблемы, для решения которых необходимо кардинальное изменение архитектуры процессора. Поэтому можно констатировать завершение длящейся более 40 лет эпохи процессоров как основного направления вычислительной техники и ожидать появления новых, гибридных по своей природе, вычислительных архитектур, которые потребуют смену базовых парадигм организации вычислительного процесса и программирования вычислительных систем.

2. Тенденции развития современной элементной базы высокопроизводительных вычислительных систем

Самым серьезным сдерживающим фактором сбалансированного совершенствования аппаратных и программных средств современной высокопроизводительной вычислительной техники является фон-неймановская архитектура процессоров, которая не может обеспечить заданный научно-техническим прогрессом рост производительности вычислительных систем (ВС). Механическое соединение процессоров не позволяет обеспечить пропорциональное ускорение производительности ВС вследствие того, что исходно последовательные вычислительные процессы, реализуемые в фон-неймановском процессоре, не удовлетворяют современным требованиям, а большинство технологий распараллеливания порождают довольно сложные и искусственные решения.

Поэтому вся отрасль вычислительной техники, связанная с высокоскоростной обработкой информации, вплотную приблизилась к необходимости смены существующей массовой вычислительной архитектуры фон Неймана, одной из наиболее успешных попыток которой стало появление графических ускорителей как самостоятельных вычислительных устройств и технологии GPGPU (англ. **General-purpose** computing for **graphics processing units**, неспециализированные вычисления на графических вычислительных устройствах) [4].

Графические ускорители, технологии и архитектуры которых интенсивно развиваются в течение последних 7 лет, являются первой попыткой гибридизации архитектуры процессоров. Успех архитектуры графических ускорителей во многом обеспечивался подготовленной технологической базой их производства: фактически технологический процесс, использовавшийся для производства предыдущих поколений процессоров, мог без существенных изменений использоваться для производства графических ускорителей. Графические ускорители, несмотря на наличие большого числа одновременно работающих устройств, обладающих фиксированной структурой связей, по своей сути во многом повторяют структуру процессора, поэтому обладают свойственными процессорам недостатками: загрузкой большого числа одновременно работающих элементарных процессоров, синхронизацией вычислений, низкой скоростью доступа к памяти («стена памяти») [4, 5], падением скорости доступа к процессору («узкое горло») [4] и существенным тепловыделением («энергетическая стена») [4].

Анализируя последние разработки основных производителей процессоров и графических ускорителей (Intel, AMD, nVidia, IBM), можно сделать вывод о том, что дальнейшее развитие вычислительных архитектур ведущих мировых производителей планируется по пути их гибридизации, т.е. объединения на одном кристалле нескольких ядер процессора со специализированными вычислительными устройствами, в роли которых могут выступать графические процессоры (AMD Radeon или NVIDIA GPU), специализированные сопроцессоры (Intel Xeon Phi) или ПЛИС (Altera или Xilinx). На сегодняшний день в области высокопроизводительных вычислений только гибридные вычислительные системы имеют перспективы дальнейшего развития в соответствии со сложившейся технологией производства процессоров и парадигмой вычислений. Основной вопрос в направлении дальнейшей гибридизации: компания AMD, по-видимому, сделала свой выбор в пользу технологии гибридизации по принципу «процессор и графический ускоритель» [6], а компания Intel делает ставку [7] на связку «процессор и ПЛИС» для новых устройств.

Объединение графических ускорителей с процессором в рамках одного кристалла, скорее всего, усилит свойственные как процессорам, так и графическим ускорителям проблемы, а эффективное решение для них в настоящее время не просматривается, поэтому гибридизация процессоров и ПЛИС выглядит более перспективным направлением. ПЛИС по сравнению с графическими ускорителями и сопроцессорами при одинаковом или даже большем количестве вычислительных устройств обладают архитектурными преимуществами (возможность адаптации вычислительной структуры под структуру решаемой задачи), запасом по технологическим нормам производства (20 нм на сегодняшний день, 16 нм - в двухлетней перспективе), по тепловыделению (по сравнению с процессорами ПЛИС гораздо «холоднее») и по возможным интерфейсам доступа к памяти и синхронизации вычислений.

Возможность программирования архитектуры ПЛИС обеспечивает существенно большую гибкость и позволяет достигать высоких значений реальной и удельной производительности при решении широкого класса задач, кроме ряда задач управления, содержащих множество вложенных условных операторов.

Единственным недостатком ПЛИС по сравнению с процессорами и графическими процессорами является практическое отсутствие эффективных высокоуровневых средств программирования многокристальных решений для ПЛИС и гибридных вычислительных систем, содержащих в своем составе процессоры и ПЛИС, позволяющие обеспечить программиста привычным для процессорных систем уровнем удобства разработки и отладки прикладных программ.

Большинство реальных практических задач, решаемых на современных высокопроизводительных вычислительных системах, требует совмещения в едином вычислительном контуре как последовательных, так и параллельных вычислительных фрагментов [8] для эффективной реализации структурных и процедурных фрагментов вычислений. Решение этой проблемы многие разработчики видят в создании вычислительных систем с гибридной организацией вычислений, содержащих различные по архитектуре вычислительные узлы, объединенные каналами передачи

данных и позволяющие реализовать структурные и процедурные вычисления в едином вычислительном контуре. Симбиоз узлов различной архитектуры [9–10] в одной вычислительной системе теоретически позволяет повысить реальную производительность вычислительной системы за счет эффективной реализации как структурных, так и процедурных фрагментов вычислений на узлах различной архитектуры.

3. Гибридная реконфигурируемая вычислительная система

Будем рассматривать гибридные реконфигурируемые вычислительные системы (ГРВС), содержащие множество реконфигурируемых вычислительных узлов с ПЛИС и узлы универсальных процессоров, например, РВС-7 [11].

В настоящее время для программирования таких вычислительных систем зачастую используются технологии программирования гетерогенных вычислительных систем: CUDA, OpenACC, OpenCL и т.д., в основе которых лежат расширения языков программирования C, C++, FORTRAN, учитывающие архитектуру специализированного микропроцессорного узла. К существенным недостаткам этих технологий программирования относятся плохая переносимость готовых решений между вычислительными системами различной архитектуры и конфигурации и плохая масштабируемость программ. Основной причиной указанных недостатков, по нашему мнению, является подход к программированию ВС, при котором осуществляется разбиение задачи на отдельные фрагменты, каждый из которых реализуется на отдельном узле (или отдельном устройстве) ГРВС. Таким образом, каждый задействованный узел ВС программируется независимо, поэтому любое изменение конфигурации ВС или изменение исходного кода прикладной программы приводит к необходимости повторного переразбиения задачи на фрагменты и разработке новых локальных программ для каждого узла ВС.

Специализированные языки высокого уровня для программирования реконфигурируемых вычислительных систем (РВС) обладают привычным для большинства программистов персональных ЭВМ синтаксисом языка С и отличаются между собой семантическими особенностями вызова и использования операторов. Для описания параллельных процессов в РВС в этих языках используется изначально последовательная парадигма языка С, семантика которого ориентирована на взаимодействие последовательных процессов, что и не позволяет в полной мере использовать все возможности РВС при разработке параллельных программ на этих языках. Это приводит к семантическому разрыву между исходным информационным графом задачи, его описанием на языке высокого уровня и созданной транслятором схемотехнической реализацией. Результатом этого разрыва является существенное снижение эффективности параллельной программы, как правило, в 3-5 раз более низкая производительность по сравнению с приложениями, разработанными на языках HDL-группы [12,13].

Перспективным направлением в области программирования РВС является язык высокого уровня COLAMO [8], предназначенный для описания реализации параллельного алгоритма и создания на основе принципов структурно-процедурной организации вычислений специализированной вычислительной структуры в архитектуре РВС. Программа на языке программирования COLAMO выполняет последовательную смену структурно (аппаратно) реализованных фрагментов информационного графа задачи, каждый из которых является вычислительным конвейером потока операндов. Высокая эффективность языка программирования высокого уровня COLAMO при решении задач различных классов обуславливает его выбор в качестве основы для программирования ГРВС.

Для эффективного программирования ГРВС языковые средства должны иметь возможность описания фрагментов вычислений, работающих с различными частотой, скважностью и разрядностью обрабатываемых данных для обеспечения масштабирования как фрагментов, так и отдельных устройств не только при увеличении аппаратного ресурса, но и при его сокращении, а также возможность работы с данными переменной разрядности для эффективного использования аппаратного ресурса ГРВС. Для обеспечения ресурсонезависимого программирования ГРВС средства программирования должны в автоматизированном режиме определять эффективные параметры масштабирования и редуцирования производительности, это должно выполняться без участия пользователя и автоматически адаптировать прикладную программу под текущую конфигурацию ГРВС.

4. Единая параллельно-конвейерная форма программы на языке COLAMO для ГРВС

Преобразование программы в единую параллельно-конвейерную форму, обеспечивающую возможность как увеличения параллелизма задачи (индукция) при увеличении аппаратного ресурса, так и сокращения (редукция) при сокращении вычислительного ресурса, является основой для применения автоматизированных средств. Под параллельно-конвейерной формой [8] представления переменных и массивов здесь и далее понимается описание данных и программных конструкций на языке программирования высокого уровня COLAMO, обладающее одновременно как параллельным, так и последовательным типами доступа. На языке высокого уровня COLAMO параллельный и последовательный типы доступа на уровне данных задаются с помощью ключевых слов Vector и Stream, а на уровне разрядов - ключевыми словами BitVector и BitStream [8].

Каноническая параллельно-конвейерная форма – сбалансированное описание всех переменных, массивов и операторов программы на языке программирования COLAMO, позволяющее автоматизировано менять основные параметры параллельной программы (число одновременно реализуемых подграфов вычислений, разрядность обрабатываемых данных, число операций и др.) с помощью программы-препроцессора без участия пользователя, не нарушая семантику программы. Приведение исходной программы к канонической форме представления выполняется в два этапа. На первом этапе выполняется преобразование переменных и конструкций программы к форме параллельно-конвейерной обработки на уровне данных, а на втором этапе – на уровне разрядов. В общем виде метод преобразования программы к канонической форме представления можно представить следующим образом.

1. Все массивы в исходной программе на языке программирования высокого уровня COLAMO преобразуются в ПКФ (при необходимости добавляются параллельный (Vector) или последовательный (Stream) типы доступа).

2. Все переменные параллельной программы на языке программирования высокого уровня COLAMO (кроме счетчиков цикла) преобразуются в формат конструкции union, содержащей механизм как непосредственного обращения к переменной по её типу, так и параллельный (bitvector) и последовательный (bitstream) типы доступа.

3. Все подкадры из параллельной программы на языке COLAMO преобразуются в конструкции Implicit.

4. Все конструкции и операторы программы преобразуются согласно модифицированным параметрам переменных.

5. В сформированной в единой ПКФ программе на языке программирования высокого уровня COLAMO задаётся глобальная директива препроцессора редукции производительности по функциональным устройствам, равная 1.

Преобразование переменных к параллельному и последовательному типам доступа (смешанному типу доступа) выполняется согласно следующим правилам:

- если для доступа к элементам массива использовался только последовательный тип доступа, то в объявлении массива добавляется параметр Vector единичной размерности, описывающий параллельный тип доступа;

- если для доступа к элементам массива использовался только параллельный тип доступа, то в объявлении массива добавляется параметр Stream единичной размерности, описывающий последовательный тип доступа;

- если для доступа к элементам массива использовался смешанный тип доступа, то преобразования для данного массива не выполняются.

Как правило, расширение размерности выполняется для одномерных массивов. Так, при наличии в тексте программы на языке COLAMO массива A следующего вида:

Var A : Array Integer [N : Stream] Mem

в результате преобразования к канонической параллельно-конвейерной форме должно быть выполнено преобразование к объявлению вида

Var A : Array Integer [M : Vector, K : Stream] Mem,

где $M*K=N.$, а начальное значение M=1 для обеспечения эквивалентности информационных графов исходной и модифицированной программ.

На рис. 1 показан пример преобразования исходной программы.

```
Const N = 10;
Var a, b, c, d : Array Integer [N: Vector] Mem;
Var i : Number;
Cadr ExpParallel;
    For i := 0 to N - 1 do
      Begin
          If (A[i] > 5)
              C[i] := A[i] - B[i];
          Else
              C[i] := A[i] + D[i];
      end;
EndCadr;
```

а)

```
Const N = 10; Const M = 10; Const K = N/M;
Var a, b, c, d : Array Integer [M : Vector, K : Stream] Mem;
Var i, vc_1 : Number;
Cadr ExpParallelConvData;
    For vc_1 := 0 to K - 1 do
        For i := 0 to M - 1 do
          Begin
              If (A[i, vc_1] > 5)
                  C[i, vc_1] := A[i, vc_1] - B[i, vc_1];
              Else
                  C[i, vc_1] := A[i, vc_1] + D[i, vc_1];
          end;
EndCadr;
```

б)

Рис. 1. Преобразование к канонической форме на уровне данных: а) исходная программа; б) программа, преобразованная к канонической форме

Как видно из текста параллельно-конвейерной программы в ПКФ (рис. 1,б), все одномерные массивы с параллельным типом доступа (N : Vector) были преобразованы к двумерным массивам, имеющим смешанный тип доступа (M : Vector, K : Stream), а также выполнена модификации всех обращений к данным массивам с использованием нового оператора цикла с индексной переменной VC_1. Для эффективной адаптации программы для ГРВС такие же преобразования должны быть выполнены и для разрядов путем перехода от статических типов данных (Integer, Real, Int64 и т.д.) к пользовательским типам данных с указанием параллельного (BitVector) и последовательного (BitStream) типов доступа к разрядам.

5. Ресурсонезависимое программирование ГРВС

Основой для простого масштабирования и адаптации прикладной программы как для случая увеличения, так и для случая сокращения доступного аппаратного ресурса, является редукция производительности [14, 15] прикладной программы, под которой понимается пропорциональное сокращение производительности во всех без исключения фрагментах информационного графа задачи с возможным сокращением аппаратных затрат на реализацию вычислительной структуры. Редуцирование производительности параллельных программ позволяет изменять ключевые параметры параллельных программ (число задействованных вычислительных устройств, число каналов памяти, разрядность операндов, частота и др.), поскольку структурная реализация задачи может привести к нехватке доступного аппаратного ресурса, что особенно актуально при переносе задачи на ГРВС различных архитектур и конфигураций. В отличии от традиционных технологий и методов программирования многопроцессорных вычислительных

систем (MPI, CUDA, OpenAcc и др.), которые предполагают распараллеливание базового информационного подграфа прикладной задачи в зависимости от конфигурации доступного вычислительного ресурса, для применения редукции производительности информационный граф задачи описывается в канонической параллельно-конвейерной форме с максимальным присущим задаче параллелизмом. В зависимости от числа доступных вычислительных узлов ГРВС исходный информационный граф сокращается (редуцируется) с помощью специальных *редукционных* преобразований, которые сбалансированно сокращают производительность всех фрагментов информационного графа и в ряде случаев сокращают занимаемый задачей аппаратный ресурс ГРВС.

Схема функционирования технологии ресурсонезависимого программирования гибридных реконфигурируемых вычислительных систем, содержащих реконфигурируемые и микропроцессорные вычислительные узлы, представлена на рис. 2.

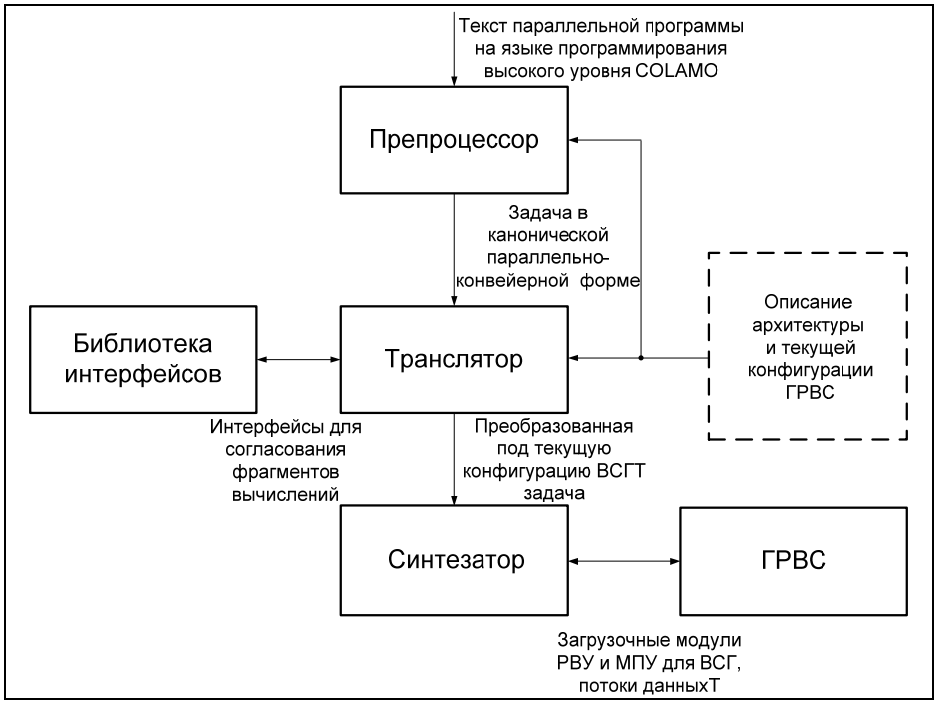

Рис. 2. Структурная схема функционирования технологии ресурсонезависимого программирования для программирования ГРВС

Модуль анализа исходной параллельной программы препроцессора языка COLAMO преобразует параллельную программу в каноническую форму, после чего подсчитывает необходимый для ее реализации аппаратный ресурс и сопоставляет его с текущей конфигурацией ГРВС для определения максимальной степени и типов необходимых преобразований. Если текущего аппаратного ресурса ГРВС достаточно для выполнения программы в исходной максимально параллельной форме, то синтезируются загрузочные модули для задействованных узлов ГРВС, в противном случае выполняются специальные редукционные преобразования, сбалансированно сокращающие производительность программы и задействованный аппаратный ресурс ГРВС. Редукция производительности выполняется в следующем порядке: редукция по одновременно выполняемым подграфам программы, редукция по разрядности, редукция по командам (устройствам), редукция по скважности (частоте). Сокращение занимаемого аппаратного ресурса для каждого вида редукции с учетом ее теоретически допустимой степени выполняет модуль анализа препроцессора, который определяет наиболее рациональный вариант использования редукции для обеспечения максимально возможной производительности программы для текущей конфигурации ГРВС. Полученный в автоматизированном режиме после препроцессора текст параллельной программы в канонической параллельно-конвейерной форме передается транслятору языка программирования COLAMO, который создает развернутый информационный граф прикладной задачи. Информационный граф прикладной задачи, содержащий фрагменты, реализуемые структурно на реконфигурируемых вычислительных узлах и процедурно на микропроцес-

сорных вычислительных узлах, передается программе-синтезатору для автоматического распределения фрагментов задачи по доступным в текущей конфигурации ГРВС реконфигурируемым и микропроцессорным вычислительным узлам. Согласование потоков данных между различными по типам организации вычислений узлам ГРВС осуществляется на основе библиотеки интерфейсов и файла описания конфигурации ГРВС, содержащего данные о типах синхронизируемых вычислительных узлов, разрядности шины данных, частоте их работы, скважности подачи данных и др. Согласование потоков данных для промежуточных вариантов реализации задачи, сочетающих схемотехническую реализацию фрагмента задачи на реконфигурируемом вычислительном узле ГРВС и многопоточные вычисления в микропроцессорах общего назначения ГРВС, осуществляется с помощью интерфейсов и элементов синхронизации, которые автоматически устанавливает либо транслятор, либо синтезатор (в зависимости от текущего шага преобразования) из соответствующей библиотеки. После установки необходимых интерфейсов и элементов синхронизации программа-синтезатор создает загрузочные конфигурационные файлы *.bit для реконфигурируемых вычислительных узлов и загрузочные файлы *.exe для микропроцессорных узлов ГРВС и единую для всех вычислительных модулей ГРВС управляющую вычислительным процессом программу.

Разработанная технология программирования ГРВС была апробирована на тестовых задачах из трёх разных предметных областей: цифровой обработки сигналов (прямое быстрое преобразование Фурье с прореживанием по частоте), символьной обработки (алгоритм симметричного блочного шифрования ГОСТ 28147-89) и мониторинга компьютерных сетей (поиск и определение частоты встречаемости заданных битовых последовательностей (шаблонов) в потоке данных).

Для всех трёх задач были разработаны программы на языке COLAMO, каждая из которых была оттранслирована для различных вариантов архитектур ГРВС с разным числом вычислительных узлов. Исследования, проведённые на реальном образце ГРВС, в роли которого использовалась созданная в 2013 году РВС-7 [11], показали эффективность разработанной технологии для адаптации текста параллельной программы к имеющемуся аппаратному ресурсу как при его увеличении, так и в случае переноса на архитектуру ГРВС с меньшим аппаратным ресурсом.

6. Заключение

Для эффективного программирования гибридных реконфигурируемых вычислительных систем предложен язык программирования высокого уровня COLAMO, позволяющий описывать в едином вычислительном контуре различные формы организации параллельных вычислений в канонической параллельно-конвейерной форме. Единая параллельно-конвейерная форма прикладной программы в совокупности с разработанными методами редукции производительности позволяет автоматизированно адаптировать прикладную программу под изменившуюся архитектуру или конфигурацию ГРВС. Предложенная технология позволяет рационально использовать ресурсы узлов с разной архитектурой при программировании ГРВС, обеспечивает пользователя набором необходимых средств для быстрой разработки ресурсонезависимых масштабируемых параллельных программ для ГРВС в едином языковом пространстве, что снижает сложность программирования ГРВС и повышает скорость разработки параллельных прикладных программ

Литература

1. Хорошевский В.Г. Архитектура вычислительных систем. Москва: Изд-во МГТУ имени Н.Э. Баумана, 2008. 520 с.
2. URL: http://www.intel.ru/content/www/ru/ru/products/processors.html (дата обращения: 12.04.2017).
3. Черняк Л. Архитектура фон Неймана, реконфигурируемые компьютерные системы и антимашина // Открытые системы. СУБД, 2008. № 06. С. 2–10.

4. Андреев С.С., Дбар С.А., Давыдов А.А., Лацис А.О., Савельев Г.П., Орлов В.Л., Плоткина Е.А., Простов И.В. Гибридный суперкомпьютер К-100: что дальше? // Информационные технологии и вычислительные системы, 2012. №2. С. 29–35.
5. Wulf W.A., McKee S.A. Hitting the Memory Wall: Implications of the Obvious // Computer Architecture News, Mar. 1995. Vol. 23, No. 1. P. 20–24.
6. URL: http://www.amd.com/ru/technologies/zen-core (дата обращения: 12.04.2017).
7. Intel представит первые процессоры Xeon со встроенными FPGA в первом квартале 2016 [электронный ресурс]. URL: http://www.3dnews.ru/923925 (дата обращения: 12.04.2017).
8. Kalyaev I.A., Levin I.I., Semernikov E.A., Shmoilov V.I. Reconfigurable multipipeline computing structures. New York: Nova Science Publishers, 2012. 330 p.
9. Liang T.-Y., Li H.-F., Lin Y.-J., Chen B.-S. A Distributed PTX Virtual Machine on Hybrid CPU/GPU Clusters // Journal of Systems Architecture, 1 January 2016. Vol. 62. P. 63–77.
10. Li H.-F., Liang T.-Y., Lin Y.-J. An OpenMP programming toolkit for hybrid CPU/GPU clusters based on software unified memory // Journal of Information Science and Engineering, May 2016. Vol. 32, Issue 3. P. 517–539.
11. Levin I.I., Kovalenko V.B., Gudkov V. A., Gulenok A.B. High-Performance Reconfigurable Computer Systems Based on Virtex FPGAs // 13th International Conference on Parallel Computing Technologies (PaCT-2015), Petrozavodsk, Russia, August 31-September 4, 2015. P. 349–362.
12. El-Araby E., Taher M., Abouellail M., El-Ghazawi T., Newby G.B. Comparative analysis of high level programming for reconfigurable computers: Methodology and empirical study // 3rd Southern Conference on Programmable Logic, SPL'07; Mar del Plata; Argentina; 26 February 2007 through 28 February 2007; Category number07EX1511; Code 70259. 2007, Article number 4234328. P. 99–106.
13. Xu J, Subramanian N, Alessio A, Hauck S. Impulse C vs. VHDL for accelerating tomographic reconstruction // 18th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2010; Charlotte, NC; United States; 2 May 2010 through 4 May 2010; Category numberP4056; Code 80904. 2010, Article number 5474054. P. 171–174.
14. Gulenok A.A., Dordopulo A.I., Levin I.I., Gudkov V.A. Hybrid computer system programming technology with adaptation and scaling of calculations // Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2017. Vol. 6. No. 1. P. 73–86. DOI: 10.14529/cmse170105.
15. Дордопуло А.И., Левин И.И., Каляев И.А., Гудков В.А., Гуленок А.А. Программирование вычислительных систем гибридного типа на языке программирования COLAMO // Известия ЮФУ. Технические науки. Ростов-на-Дону: Изд-во ЮФУ, 2016. №11(184). С. 39–54. ISSN 1999-9429.

Resource-independent programming of hybrid reconfigurable computer systems

A.I. Dordopulo[1], I.I. Levin[1]

Scientific Research Centre of Supercomputers and Neurocomputers[1]

Analysis of progress trends of modern processors shows that further development of processor architectures is possible owing to hybridization - combination of a processor and a specialized calculator on a single chip. Use of FPGAs is the most promising direction of processor hybridization because of its technology, heat dissipation, and possible use of interfaces for memory access and synchronization of calculations. Besides, there is possibility of adaptation FPGA architectures to the structure of any solving task. New methods and programming tools, united into the resource-independent programming technology, which is considered in this paper, are necessary for effective programming of such hybrid reconfigurable computer systems.

Keywords: high-level programming language, programming of hybrid computer systems, resource-independent programing technology, performance reduction.

References

1. Khoroshevskiy V.G. Arkhitektura vychislitelnykh sistem [Architecture of computing systems]. Moscow: Publishing house of MGTU named H. E. Bauman, 2008. 520 p.
2. http://www.intel.ru/content/www/ru/ru/products/processors.html (accessed: 12.04.2017).
3. Chernyak L. Arkhitektura fon Neymana. rekonfiguriruyemyye kompyuternyye sistemy i antimashina [Von Neumann Architecture, reconfigurable computer systems and antimachine] // Open systems. DBMS 2008. No. 06. P. 2–10.
4. Andreyev S.S., Dbar S.A., Davydov A.A., Latsis A.O., Savelyev G.P., Orlov V.L., Plotkina E.A., Prostov I.V. Gibridnyy superkompyuter K-100: chto dalshe? [Hybrid supercomputer K-100: what next?] // Information technologies and computing systems, 2012. No. 2. P. 29–35.
5. Wulf W.A., McKee S.A., Hitting the Memory Wall: Implications of the Obvious // Computer Architecture News, Mar. 1995. Vol. 23. No. 1. P. 20–24.
6. https://www.amd.com/ru/technologies/zen-core (accessed: 12.04.2017).
7. Intel predstavit pervyye protsessory Xeon so vstroyennymi FPGA v pervom kvartale 2016 [Intel introduced the first Xeon processors with integrated FPGAs in the first quarter of 2016]. URL: http://www.3dnews.ru/923925 (accessed: 12.04.2017).
8. Kalyaev I.A., Levin I.I., Semernikov E.A., Shmoilov V.I. Reconfigurable multipipeline computing structures. New York: Nova Science Publishers, 2012. 330 p.
9. Liang T.-Y., Li H.-F., Lin Y.-J., Chen B.-S. A Distributed PTX Virtual Machine on Hybrid CPU/GPU Clusters // Journal of Systems Architecture, 1 January 2016. Vol. 62, P. 63–77.
10. Li H.-F., Liang T.-Y., Lin Y.-J. An OpenMP programming toolkit for hybrid CPU/GPU clusters based on software unified memory // Journal of Information Science and Engineering, May 2016. Vol. 32, Issue 3. P. 517–539.
11. Levin I.I., Kovalenko V.B., Gudkov V. A., Gulenok A.B. High-Performance Reconfigurable Computer Systems Based on Virtex FPGAs // 13th International Conference on Parallel Computing Technologies (PaCT-2015), Petrozavodsk, Russia, August 31-September 4, 2015. P. 349–362.

12. El-Araby E., Taher M., Abouellail M., El-Ghazawi T., Newby G.B. Comparative analysis of high level programming for reconfigurable computers: Methodology and empirical study // 3rd Southern Conference on Programmable Logic, SPL'07; Mar del Plata; Argentina; 26 February 2007 through 28 February 2007; Category number07EX1511; Code 70259. 2007, Article number 4234328. P. 99–106.
13. Xu J, Subramanian N, Alessio A, Hauck S. Impulse C vs. VHDL for accelerating tomographic reconstruction // 18th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2010; Charlotte, NC; United States; 2 May 2010 through 4 May 2010; Category numberP4056; Code 80904. 2010, Article number 5474054. P. 171–174.
14. Gulenok A.A., Dordopulo A.I., Levin I.I., Gudkov V.A. Hybrid computer system programming technology with adaptation and scaling of calculations // Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2017. Vol. 6. No. 1. P. 73–86. DOI: 10.14529/cmse170105.
15. Dordopulo A.I., Levin I.I., Kalyayev I.A., Gudkov V.A., Gulenok A.A. Programmirovaniye vychislitelnykh sistem gibridnogo tipa na yazyke programmirovaniya COLAMO [Programming computing systems hybrid programming language COLAMO] // Izvestiya yufu. Technical Sciences. Rostov-na-Donu: Izd-vo yufu, 2016. No. 11(184). P. 39-54. ISSN 1999-9429.