УДК 004.2; 004.31 DOI: 10.17587/it.25.682-690

А. Ю. Попов, канд. техн. наук, доц., e-mail: alexpopov@bmstu.ru,
Московский государственный технический университет им. Н. Э. Баумана, г. Москва

Применение гетерогенной вычислительной системы с набором команд дискретной математики для решения задач на графах

В последнее десятилетие существенно вырос интерес к анализу сложных моделей данных, в том числе графов, являющихся наиболее адекватной формой представления данных социальных сетей, компьютерных программ, топологии интегральных схем, биоинформатики и других важных приложений. По мере того, как размер этих наборов данных увеличивается, становится очевидной необходимость поиска более эффективных методов и средств анализа больших графов, в том числе на основе более совершенных аппаратных технических решений. Современные графические процессоры обладают большим параллелизмом и производительностью, однако не решают основных проблем обработки графов: зависимостей по данным, распределения нерегулярных графов по процессорным устройствам, конфликтов при доступе к памяти. В МГТУ им. Н. Э. Баумана разработан специализированный микропроцессор Leonhard ×64 с набором команд дискретной математики (DISC), предназначенный в том числе для обработки графов большой размерности. В статье приведены сведения об операциях дискретной математики и соответствующих им машинных инструкциях микропроцессора Leonhard x64, обосновывается архитектура гетерогенной вычислительной системы на его основе. На примерах алгоритмов Дейкстры и Белмана—Форда демонстрируется различие между реализацией обработки графов, а также эффективность универсальных вычислительных систем, гетерогенных вычислительных систем на основе графических ускорителей GPGPU и системы на основе микропроцессора Leonhard ×64.

Ключевые слова: *набор команд дискретной математики, микропроцессор Leonhard, граф, структура данных, графический ускоритель вычислений*

Обзор современных аппаратных систем

Потоковая обработка сложных моделей данных, таких как графы, становится все более востребованной в аналитических системах. Ускорение их работы может быть достигнуто благодаря более совершенным методам и алгоритмам, а также благодаря повышению эффективности аппаратного обеспечения. Однако традиционные подходы к проектированию параллельных программ и систем, основанные на применении большого числа однотипных универсальных процессоров, уже не обеспечивают требуемого быстродействия, приводят к существенному росту сложности и времени разработки и отладки программного кода. Вместе с этим, мы покажем, что существующие вычислительные системы, использующие массовый параллелизм, еще далеки от совершенства, а заложенные в них аппаратные механизмы не предназначены для обработки зависимых или сильно фрагментированных в оперативной памяти данных. Рассмотрим далее проблемы обработки алгоритмов дискретной оптимизации в универсальных микропроцессорах (подобных семейству ×86 Intel) или же в современных графических ускорителях GPGPU (например, NVidia Tesla), а также в специальных аппаратных системах (на примере отечественного микропроцессора Leonhard ×64).

Современные вычислительные машины используют универсальные микропроцессоры, которые являются не только основными обрабатывающими устройствами с точки зрения архитектуры ЭВМ, но и определяют специфику разработки и функционирования программ. Базовыми принципами построения таких микропроцессоров, позволяющими добиться высокой производительности исполнения программного кода, являются глубокая конвейеризация микроархитектуры,

спекулятивное исполнение команд и многоуровневая подсистема памяти. Как следствие, в микропроцессорах на разных стадиях одновременно находится большое число команд и операндов, продвигаемых по конвейеру в его исполнительную часть. В случае простых итераций, основанных на хорошо предсказуемых итерационных переменных, микропроцессор способен ритмично поставлять на исполнение затребованные операнды, а длинный конвейер оказывается эффективным. Именно на подобных задачах мы можем наблюдать высокую производительность универсальных микропроцессоров. Проблемы зависимых данных, тем не менее, являются характерной чертой подавляющего большинства алгоритмов на графах. Вследствие таких зависимостей нарушается ритмичная работа конвейера, снижается эффективность аппаратной предвыборки, увеличивается число сбросов конвейера из-за неправильно предсказанных переходов [1—3].

Более того, анализ потока команд показывает [4], что команды арифметической и логической обработки составляют лишь порядка 30 % инструкций, и не менее 50 % потока инструкций составляют команды пересылки. Учитывая, что эти данные получены на микропроцессоре с полным набором инструкций (Complex Instruction Set Computing, CISC), при котором доступ к памяти кодируется в одной команде с операциями обработки, можно констатировать большой процент служебных команд в потоке инструкций и, как следствие, малую эффективность универсального подхода в целом.

Одновременно с этим обработка сложных моделей данных, таких как графы, затруднена в связи с особенностями размещения графов в оперативной памяти. Существенной проблемой универсальных микропроцессоров является фрагментация при сегментно-страничной организации виртуальной памяти, которая приводит к потоку двойных обращений к физической оперативной памяти. В этом случае наблюдается замедление обмена данными между процессором и памятью из-за открытия и закрытия большого числа страниц оперативной памяти, а конвейер простаивает в связи с зависимостями по данным. Принятый для современных типов ОЗУ пакетный режим работы также снижает эффективность системы, так как способствует неполному использованию шины и ресурсов процессора. Можно констатировать, что архитектурные решения, заложенные в современные ЭВМ, направлены на ускорение обработки векторных структур и, напротив, замедляют обработку ссылочных структур данных. Также характерно, что в настоящее время отсутствуют средства, ускоряющие обработку зависимых данных.

Одним из перспективных способов повышения производительности универсальных ЭВМ является применение аппаратных ускорителей вычислений (так называемых акселераторов). Ускорение обработки в этом случае достигается благодаря применению более совершенных и параллельных аппаратных механизмов, независимых от работы центрального процессора и системной памяти, а также за счет высвобождения важных ресурсов системы (системных шин, буферов ввода/вывода и пр.). Одновременная обработка многих потоков данных позволяет снять часть вычислительной нагрузки с универсальных микропроцессоров и перенести их на независимое программное и аппаратное обеспечение ускорителей. Таким образом, положительный технический результат достигается благодаря лучшему сочетанию таких параметров, как цена изделия, производительность, простота применения технологии, энергопотребление.

После 2000 г. стало ясно, что мощность графических ускорителей GPU (Graphics Processing Unit) растет быстрее мощности универсальных микропроцессоров. Однако специфика разработки программ для видеоадаптеров ограничивала развитие технологий ускорения вычислений на графических ядрах. Современные процессоры GPU содержат большое число потоковых мультипроцессоров (SM), обладают высокой вычислительной мощностью и относительно низкой стоимостью. Проект NVidia Tesla развивает идею GPGPU (General-purpose computing on graphics processing units) и реализован в целях получения компактных высокопроизводительных систем, обладающих высокой универсальностью и в большей степени использующих универсальные средства разработки программ на языках C/C + + и Fortran. В настоящий момент число параллельных потоков вычислений в чипе GPGPU достигает нескольких десятков тысяч. Однако для эффективной реализации параллельного алгоритма на GPGPU требуется выполнение таких условий как: представление задачи в виде независимых потоков вычислений; отсутствие зависимостей по данным; возможность представления данных в локальной памяти SM.

Очевидно, что при наличии нескольких тысяч потоковых процессоров в GPGPU эффек-

тивность решения задачи зависит от возможности ее представления в виде тысяч параллельных потоков и пропускной способности подсистемы памяти. При этом требуется не только эффективно распределить данные по многим устройствам SM, но и обеспечить получение и анализ результата вычислений. Как следствие, фаза загрузки/подготовки данных и фаза выгрузки результатов занимают достаточно много времени, так как подготовка к запуску вычислений и завершающий этап вычислений сопровождаются перемещением данных между оперативной памятью CPU и глобальной памятью GPGPU по шине PCIe. Этот процесс требует существенных затрат системного времени и ресурсов из-за достаточно высокой латентности шины и необходимости инициализации механизмов копирования (на основе механизмов прямого доступа к памяти, DMA).

Предпочтительным вариантом представления данных для GPGPU являются матрицы и векторы, которые могут быть легко разделены на части, соответствующие объему кэш-памяти SM. Как следствие, при объемах данных, существенно превышающих размеры L1 кэш SM и L2 чипа GPGPU, скорость работы графических ускорителей резко падает. Такой эффект проявляется при обработке разреженных матриц в связи с непредсказуемым шаблоном доступа к глобальной памяти. При существенном увеличении числа вычислительных ядер в GPGPU скорость вычислений возрастает лишь в 1,1...5 раз [5].

В связи с выявленными недостатками обработки зависимых данных на универсальных микропроцессорах и на графических процессорах GPGPU рассмотрим также альтернативный подход к решению задач дискретной оптимизации на основе специализированного микропроцессора Leonhard ×64.

Микропроцессор с набором команд дискретной математики

В данной работе мы предлагаем альтернативный подход к решению задач на графах, основанный на применении специализированного микропроцессора Leonhard ×64 с набором команд дискретной математики (Discrete Mathematics Instructions Set Computer, DISC), разработанного в МГТУ им. Н. Э. Баумана. Для разработки принципов применения такого устройства потребовалось внести изменения в основы функционирования и взаимодействия микропроцессоров в универсальных вычислительных системах. Поясним сказанное на основе модели вычислительной системы, рассмотренной в работах В. Г. Хорошевского. Согласно [6] вычислительная система представляет собой разновидность вычислителя:

$$c = \langle U, g, a(p(D)) \rangle, \quad (1)$$

где U — множество устройств, обеспечивающих ввод, обработку, хранение и вывод информации; g — структура связей между устройствами; a — алгоритм работы вычислителя или алгоритм управления вычислительными процессами при реализации программы p обработки данных D. В работе [6] отмечается, что для любого вычислителя характерны: последовательная обработка информации; фиксированность структуры; неоднородность составляющих устройств и связей между ними.

Дальнейшее совершенствование такой системы невозможно без выделения специальной вычислительной нагрузки и разработки более эффективных средств для ее выполнения. Микропроцессор Leonhard ×64, в частности, предназначен для аппаратного выполнения набора операций над множествами, структурами данных и графами, которые до этого реализовывались на основе универсальных микропроцессоров. Стоит отметить, что обработка структур данных является основой большого числа прикладных алгоритмов. Ускорение их работы всегда являлось приоритетной задачей исследователей, а большинство новых и более эффективных алгоритмов дискретной оптимизации основаны на более совершенных структурах для представления данных в памяти ЭВМ.

Согласно [7] структура данных определяется как совокупность двух множеств:

$$S = (D, R), \quad (2)$$

где D — множество элементов данных; R — множество отношений между элементами данных.

Так как обработка структур данных предполагает взаимодействие устройств из U через каналы связи g и под управлением алгоритма a, то свойства всех указанных компонентов модели влияют на время формирования результата вычислителем. Тогда на основании (2) можно констатировать, что при обработке структур данных должна происходить обработка как отношений R, так и скалярных данных D. В том случае, если такая обработка выполняется раздельно на различных устройствах, можно

определить уточненную модель вычислителя, выполняющего обработку структур данных:

$$c_S = \langle U_D,\ U_R,\ g,\ a_D(p_D(D)),\ a_R(p_R(R))\rangle, \qquad (3)$$

где U_D — множество устройств обработки скалярных данных; U_R — множество устройств обработки отношений данных; g — структура связей между устройствами; a_D и a_R — алгоритмы управления вычислительными процессами в U_D и U_R; p_D — программа обработки данных D; p_R — программа обработки отношений R. Принято следующее допущение: множество устройств вычислителя $U = U_D \cup U_R$; алгоритм работы вычислителя $a = a_D \cup a_R$; программы обработки $p = p_D \cup p_R$. Это означает, что в вычислительной задаче выделяется часть действий по обработке отношений данных, выполнение которых возлагается на специальные вычислительные устройства. В случае полного разделения вычислителя на две независимые подсистемы — части для обработки данных и отношений — будет выполняться более строгое условие: $U_D \cap U_R = \varnothing$; $a_D \cap a_R = \varnothing$, $p_D \cap p_R = \varnothing$.

Уточненная модель вычислителя позволяет сформулировать принцип декомпозиции архитектуры вычислительной системы, согласно которому устройство обработки отношений имеет доступ к независимой памяти, в которой хранятся множества, структуры данных, графы, а также команды их обработки. Результаты выполнения команд направляются в центральный процессор для дальнейшего использования в ходе вычислительного алгоритма. Таким образом, функциями микропроцессора Leonhard ×64 в вычислительной системе являются хранение и независимая обработка множеств, структур данных и графов.

Для определения набора команд микропроцессора Leonhard ×64 были исследованы операции дискретной математики, а также ряд алгоритмов дискретной оптимизации [8]. В результате были определены основные функции устройства обработки отношений, которые могут быть представлены следующей формальной моделью:

$A = \langle A_1, ..., A_n\rangle$ — функция хранения кортежа A из n множеств;

$R(A_i, x, y)$, $x \in A_i$, $y \in A_i$ — функция определения отношения между элементами x и y множества A_i;

$|A_i|$, $i = \overline{1, n}$, — операция определения мощности множества;

$x \in A_i$, $i = \overline{1, n}$, — операция проверки принадлежности элемента x множеству;

$A_i \cup x$, $i = \overline{1, n}$, — операция добавления элемента в множество;

$A_i \setminus x$, $i = \overline{1, n}$, — операция удаления элемента из множества;

$A \setminus A_i$ — операция удаления множества A_i из кортежа A;

$A_i \subset A_j$ — операция отношения включения множества A_i в A_j;

$A_i \equiv A_j$ — операция отношения эквивалентности;

$A_i \cup A_j$ — операция объединения множеств;

$A_i \cap A_j$ — операция пересечения множеств;

$A_i \setminus A_j$ — операция разности множеств;

$A_i \vartriangle A_j$ — операция симметрической разности;

$\overline{A}$ — операция дополнения A_i;

$A_i \times A_j$ — операция декартового произведения;

2^{Ai} — операция определения Булеана.

На основе функциональной модели были разработаны машинные команды процессора обработки структур, которые представляют собой высокоуровневые действия, основанные на операциях дискретной математики [28]. Последняя версия набора команд Leonhard ×64 состоит из 20 высокоуровневых кодов операций, перечисленных ниже:

- **Search (SRCH)** выполняет поиск значения, связанного с ключом.
- **Insert (INS)** вставляет пару ключ—значение в структуру. SPU обновляет значение, если указанный ключ уже находится в структуре.
- Операция **Delete (DEL)** выполняет поиск указанного ключа и удаляет его из структуры данных.
- Последняя версия набора команд Leonhard ×64 была расширена двумя новыми инструкциями (NSM и NGR) для обеспечения требований некоторых алгоритмов. Каждая инструкция набора включает до трех операндов: ключ, значение и номер структуры данных. Команды **NSM/NGR** выполняют поиск соседнего ключа, который меньше (или больше) заданного и возвращает его значение. Операции могут быть использованы для эвристических вычислений, где интерполяция данных используется вместо точных вычислений (например, кластеризация или агрегация).
- **Maximum /minimum (MAX, MIN)** ищут первый или последний ключи в структуре данных.
- Операция **Cardinality (CNT)** определяет число ключей, хранящихся в структуре.
- Команды **AND, OR, NOT** выполняют объединения, пересечения и дополнения в двух структурах данных.

- **Срезы (LS, GR, LSEQ, GREQ)** извлекают подмножество одной структуры данных в другую.
- **Переход к следующему или предыдущему (NEXT, PREV)** находят соседний (следующий или предыдущий) ключ в структуре данных относительно переданного ключа. В связи с тем, что исходный ключ должен обязательно присутствовать в структуре данных, операции NEXT/PREV отличаются от NSM/NGR.
- **Удаление структуры (DELS)** очищает все ресурсы, используемые заданной структурой.
- Команда **Squeeze (SQ)** дефрагментирует блоки памяти DSM, используемые структурой.
- Команда **Jump (JT)** указывает SPU код ветвления, который должен быть синхронизирован с CPU (команда доступна только в режиме MISD).

Далее покажем (табл. 1) соответствие набора команд DISC функциям, операциям и кванторам дискретной математики.

Как следует из табл. 1, в наборе команд DISC остаются не реализованными лишь операция симметрической разности, декартового произведения, а также операция Булеана. Причиной такого ограничения является большая сложность их аппаратной реализации. Также стоит отметить, что перечисленные операции редко требуется применять в алгоритмах дискретной оптимизации, а аппаратные затраты на их реализацию высоки. Кроме того, все указанные операции легко реализуются на основе реализованного подмножества операций:

- симметрическая разность может быть получена с использованием команд NOT и OR ($A \Delta B = (A \setminus B) \cup (B \setminus A)$);
- декартово произведение и Булеан могут быть реализованы на основе последовательного обхода множеств командами NEXT, PREV и INS.

Таблица 1

Соответствие инструкций DISC функциям, кванторам, операциям дискретной математики

Функции, кванторы и операции дискретной математики	Инструкции набора команд DISC
Функция хранения кортежа	INS
Функция отношения элементов множества	NEXT, PREV, NSM, NGR, MIN, MAX
Мощность множества	CNT
Функция принадлежности элемента множеству	SRCH
Добавление элемента в множество	INS
Исключение элемента из множества	DEL, DELS
Исключение подмножества из кортежа	DELS
Включение подмножества в кортеж	LS, GR, LSEQ, GREQ
Отношение эквивалентности множеств	LS, GR, LSEQ, GREQ
Объединение множеств	OR
Пересечение множеств	AND
Разность множеств	NOT
Симметрическая разность множеств	Не реализовано
Декартово произведение множеств	Не реализовано
Булеан множества	Не реализовано

Особенности архитектуры гетерогенной вычислительной системы на основе микропроцессора Leonhard x64

Вычислительная система, разработанная в МГТУ им. Н. Э. Баумана, состоит из параллельно работающих микропроцессоров нескольких типов. Функциями центрального процессора (рис. 1) является общее управление вычислительной системой, т. е. загрузка операционной системы, обмен данными с дисковой подсистемой, запуск вычислительных алгоритмов, обмен информацией с внешними устройствами, визуализация результатов вычислений на экран. В качестве центрального процессора (ЦП) в различных модификациях системы были использованы микропроцессоры фирм Intel, AMD, а также отечественный микропроцессор Байкал T1 производства АО Байкал Электроникс.

ЦП соединен с ускорительной частью через высокоскоростную шину PCIe ×8, по которой происходит загрузка команд и операндов в микропроцессор Leonhard ×64, а также обмен данными с локальным арифметическим процессором (ЛАП, Local arithmetic processor). Последний (т. е. ЛАП) размещен на одном кристалле с Leonhard ×64 версии 3.0, исполняет стандартный набор команд арифметической и логической обработки чисел и призван сократить время простоя микропроцессора Leonhard ×64 в ожидании новых команд и операндов.

Стоит отметить, что предыдущая версия микропроцессора Leonhard (версия 2.0) не предусматривала размещение универсального микропроцессора на одном кристалле [8—10].

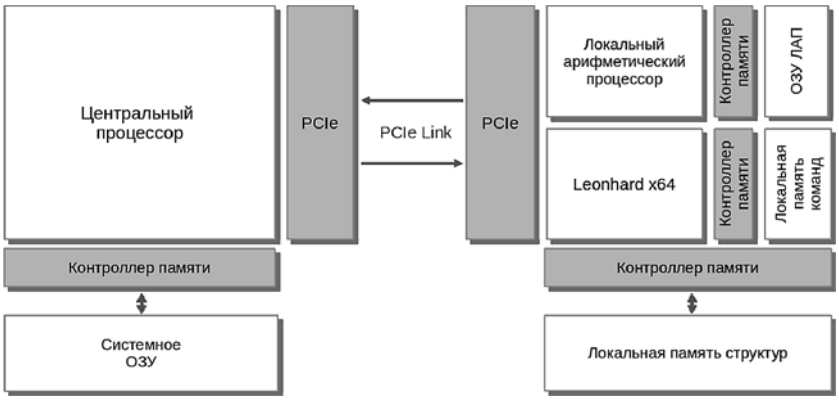

Рис. 1. Архитектура вычислительной системы на основе микропроцессора Leonhard ×64

Как следствие, весь обмен данными между вычислительным алгоритмом и ускорительной частью проводился по шине PCIe. Такой обмен характеризуется не только большой пропускной способностью, но и высокой латентностью в связи с необходимостью использования драйверной модели шины и трехуровневой организацией взаимодействия PCI Express. При этом существенное время тратится на формирование пакетов для трех уровней взаимодействия устройств (PCIe поддерживает обмен на уровне транзакций, уровне представления данных и на физическом уровне), а также получение подтверждения от соответствующего уровня "устройства-визави". Проведенные эксперименты показали, что латентность операции записи и последующего чтения, часто используемая при обращении к ускорителям, снижается приблизительно в 8 раз при использовании локальной шины внутри кристалла ПЛИС (табл. 2).

Также нужно учесть, что важной особенностью обмена данными между устройством выполнения команд дискретной математики и устройством выполнения арифметико-логических команд являются малые объемы передаваемых данных и зависимости по данным. Объяснить это можно тем, что большинство действий над множествами в алгоритмах дискретной математики (например, квантор существования, операция добавления элемента в множество и др.) используют простые числовые операнды, а не большие последовательности чисел.

В результате было выявлено, что использование шины PCIe существенно замедляет работу вычислительных алгоритмов дискретной оптимизации, так как не обеспечивает низкой латентности выполнения транзакций. Выходом из этого положения может быть использование более высокоскоростных аппаратно-управляемых интерфейсов (управление транзакциями не использует программных драйверов). Поэтому было принято решение поместить универсальный микропроцессор на один кристалл с микропроцессором Leonhard ×64 и использовать там один из локальных типов шин, предназначенных для подключения ускорителей. В качестве локального арифметического процессора были использованы синтезируемые микропроцессорные ядра Microblaze, OpenRISC и RISCV. В результате скорость выполнения синхронных операций, таких как поиск (SRCH) и некоторых других, увеличилась до 10,5 раз.

Таблица 2

Сравнение латентности операций обмена данными с ускорителем вычислений (синхронная запись и чтение регистра данных)

Конфигурация системы и вариант реализации взаимодействия	Латентность, нс	Число тактов ЦПУ (1,9 ГГц)	Число тактов ускорителя (100 МГц)
ЦПУ Intel ×86 1.9 ГГц, шина PCIe × 8 версия 1.0 2.5 ГГц, частота интерфейсной части ускорителя 100 МГц	1750	3300	175
Локальный арифметический процессор Microblaze v5 100 МГц, шина FSL 32 бит 100 МГц, частота интерфейсной части ускорителя 100 МГц	220	418	22

Сравнение архитектурной эффективности вычислительных систем при обработке графов

Для экспериментального обоснования принятых архитектурных решений было проведено сравнение числа тактов работы вычислительной системы с использованием и без использования ускорителя операций дискретной математики, а также для различных вариантов взаимодействия ускорителя и центрального процессора. Сравнение аппаратной эффективности приведенных типов систем было рассчитано по результатам тестов на классическом алгоритме Дейкстры и алгоритме Беллмана—Форда поиска кратчайших путей на графах (Single Shortest Path Problem, SSSP). Реализации указанных алгоритмов

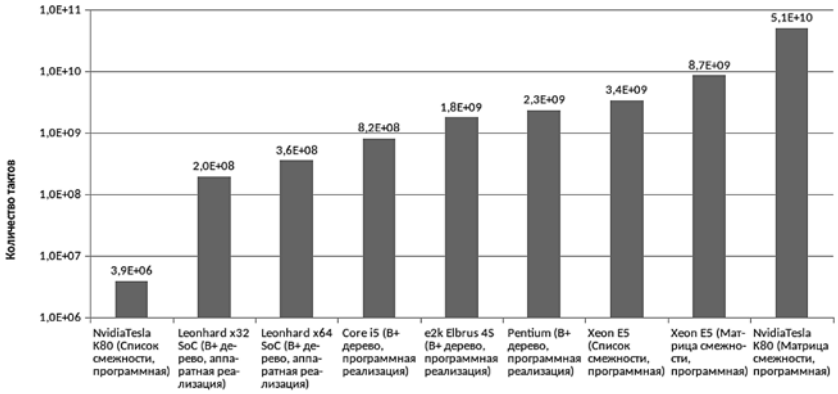

Рис. 2. Результаты сравнения производительности универсальных микропроцессоров (Intel Xeon E5, Intel Core i5, Elbrus e2k), графического ускорителя Nvidia Tesla K80 и микропроцессора Leonhard ×64 на примере задачи SSSP (алгоритм Дейкстры, граф решетки содержит 16 K вершин)

для вычислительной системы с аппаратной поддержкой операций дискретной математики были представлены в работах [12, 13].

При этом было выявлено, что эффективность решения задач существенно зависит как от формата представления графа в локальной памяти микропроцессора, так и используемого алгоритма (рис. 2). Традиционный вариант представления графа в виде матрицы смежности [7] оказывается малопригодным для реализации на GPGPU в связи с сильной разреженностью данных. В связи с этим производительность графических ускорителей при таком представлении графа оказывается существенно ниже в сравнении с универсальными микропроцессорами. В других вариантах тестов было использовано представление графа в виде сжатого списка смежности (так называемый формат CSR, Compressed Sparse Row Format), который обеспечивает высокую параллельность работы алгоритма Беллмана—Форда.

Большинство команд Leonhard ×64 требуют $O(\log_8 n)$ операций доступа к памяти (исключая операции AND/OR/NOT и операции срезов). Однако в связи с аппаратной реализацией механизмов обработки B + деревьев команды микропроцессора Leonhard ×64 выполняются за меньшее число тактов по сравнению с универсальными процессорами. В представленных экспериментах был использован микропроцессор Leonhard ×64, реализованный на основе ПЛИС Virtex 6 со встроенным локальным микропроцессором Microblaze. Мы использовали третью версию Leonhard ×64 со следующими параметрами: 64-битные ключи и значения; максимальное число ключей в памяти структур — 100 млн (100 663 296 записей); максимальное число структур данных — 7; 4 ГБайт объем локальной памяти структур; рабочая частота Leonhard ×64 100 МГц.

Казалось бы, результаты экспериментов однозначно демонстрируют преимущества многоядерных систем на основе графических ускорителей: число тактов для решения задачи оказывается в ~ 10^2 раз меньше в сравнении с микропроцессором Leonhard и в 10^3 меньше Intel Xeon E5. Однако стоит также принимать во внимание аппаратную сложность микропроцессоров, выраженную в количестве ресурсов кристалла (транзисторов или вентилей), необходимых для их реализации. Например, аппаратная сложность GPGPU NVidia Tesla K80 составляет порядка 7,1 млрд транзисторов, в то время как аппаратная сложность микропроцессора Leonhard ×64 составляет порядка 1 млн вентилей вместе с микропроцессорными ядрами PowerPC или Microblaze. Для расчета архитектурной эффективности различных вычислительных платформ будем использовать следующую формулу:

$$E_{HW} = \frac{10^6}{TICK \cdot GATE},$$

где $TICK$ — число тактов работы алгоритма; $GATE$ — число млн вентилей, необходимых для реализации микропроцессора.

В результате сравнения оказалось, что аппаратная эффективность графических ускорителей при решении задач на графах в 10^2 раз ниже специализированного микропроцессора Leonhard ×64 (рис. 3) и Leonhard ×32. Также показано, что более простые по структуре микропроцессоры (отечественный VLIW микропроцессор Elbrus e2k, Intel Pentium IV) обладают большей эффективностью в расчете на один вентиль в сравнении с современными многоядерными микропроцессорами Intel.

Полученные результаты показывают, что высокий уровень производительности современных многоядерных микропроцессоров и ускорителей GPGPU достигается благодаря их высокой параллельности и аппаратной сложности. При этом условный вклад в прирост производительности одного транзистора кристалла оказывается существенно ниже, чем у предшествующих моделей или менее параллельных микропроцессоров. При этом следует учитывать объективные ограничения крем-

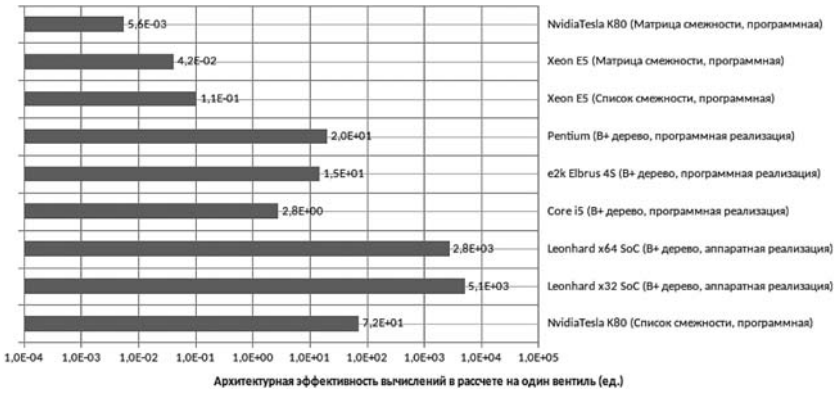

Рис. 3. Сравнение архитектурной эффективности вычислительных систем в расчете на один вентиль

ниевой технологии, влияющие на дальнейшее сохранение указанной тенденции. Размеры кристалла ограничены, а топология критически важных участков заказных СБИС определяется ручной трассировкой, что увеличивает трудозатраты на создание схемной топологии. В результате стоимость изделия зависит от площади кристалла и процента дефектных микросхем [14]. Можно утверждать, что дальнейшее наращивание числа ядер в микропроцессорах приведет к снижению удельной эффективности транзисторов на кристалле.

Выводы и варианты внедрения системы

Проведенные эксперименты и сравнение аппаратной эффективности различных микропроцессоров и ускорителей показывают необходимость разработки и внедрения специальных средств ускорения алгоритмов на графах, несмотря на существенные достижения разработчиков графических ускорителей GPGPU. Вследствие специфики задач обработки графов даже при использовании оптимизированных алгоритмов и структур данных аппаратная эффективность GPGPU и CPU, выраженная в тактах или рассчитанная на один транзистор, оказывается существенно ниже эффективности специальных ускорителей и микропроцессора Leonhard ×64 в частности.

Мы рассматриваем несколько важных областей, в которых применение процессора Leonhard должно улучшить производительность и энергопотребление вычислительных систем.

В функции контроллера программно-определяемых сетей (SDN controller) входит сбор данных по протоколу OpenFlow от многих виртуальных или физических контроллеров сети. Большинство алгоритмов управления сетями строятся на основе графовых моделей и дискретной оптимизации. При этом ключевой характеристикой эффективности контроллера SDN является время его отклика и пропускная способность. Вместе с тем растут требования по аналитической обработке сетевых транзакций. Уже сейчас применяются средства машинного обучения для прогнозирования изменения траффика и предиктивного управления сетевыми ресурсами. Также требуется выполнять анализ траффика на предмет подозрительной активности приложений, сетевых атак и других угроз информационной безопасности. В связи с этим аппаратная поддержка SDN-контроллеров на основе микропроцессора Leonhard является актуальной.

Другим направлением внедрения микропроцессора Leonhard являются робототехнические системы. По мере продвижения по маршруту (или заранее) робот строит граф видимости, в котором ребрами соединены видимые точки пространства. Вес ребра может отражать энергетические затраты, которые необходимы для перехода робота из одного положения в другое. В связи с этим требуется решать задачу поиска кратчайшего пути на графе видимости для эффективного перемещения робота.

Микропроцессор Leonhard был использован для реализации алгоритмов машинного обучения и показал хорошие результаты производительности и точности на задаче классификации телеметрических данных космического аппарата (достигнута точность распознавания 99,7 %). Это позволяет использовать Leonhard в системах управления сложными системами. В настоящее время развивается проект управления беспилотными летательными аппаратами и аппаратный автопилот. Также развивается проект применения Leonhard в системах компьютерного зрения, в которых необходимо не только распознавать видимые предметы, но и управлять роботом на основе анализа сцены.

Список литературы

1. **Patel R., Kumar S.** Visualizing effect of dependency in superscalar pipelining // 2018 4th International Conference on Recent Advances in Information Technology (RAIT). Dhanbad. 2018. P. 1—5.

2. **Patel G. R., Kumar S.** The Effect of Dependency on Scalar Pipeline Architecture // IUP Journal of Computer Sciences. Jan 2017. Vol. 11, N. 1. P. 38—50.

3. **Harish P., Narayanan P. J.** Accelerating Large Graph Algorithms on the GPU Using CUDA // High Performance Com-

puting — HiPC 2007. HiPC 2007. Lecture Notes in Computer Science. Vol. 4873. Springer, Berlin, Heidelberg

4. **Huang I. J., Peng T. C.** Analysis of ×86 instruction set usage for DOS / Windows applications and its implication on superscalar design // IEICE Transactions on Information and Systems. 2002. Vol. E85—D, N. 6. P. 929—939.

5. **Kasmi N., Mahmoudi S. A., Zbakh M., Manneback P.** Performance evaluation of sparse matrix-vector product (SpMV) computation on GPU architecture // 2014 Second World Conference on Complex Systems (WCCS), Agadir. 2014. P. 23—27.

6. **Хорошевский В. Г.** Архитектура вычислительных систем: Учеб. пособ. Москва: Изд-во МГТУ им. Н. Э. Баумана, 2008. С. 520.

7. **Кормен Т., Лейзерсон Ч., Ривест Р.** Алгоритмы: построение и анализ. М.: МЦНМО, 2000. 960 с.

8. **Попов А. Ю.** О реализации алгоритма Форда-Фалкерсона в вычислительной системе с многими потоками команд и одним потоком данных // Наука и образование. МГТУ им. Н. Э. Баумана. Электрон. журн. 2014. № 9. С. 162—180.

9. **Popov A.** An Introduction to the MISD Technology // HICSS50. Proceedings of the 50th Hawaii International Conference on System Sciences. 2017. P. 1003—1012.

10. **Abdymanapov C., Popov A.** Motion Planning Algorithms Using DISC // 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). 2019. P. 1844—1847.

11. **Rasheed B., Popov A. Y.** Network Graph Datastore Using DISC Processor // 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). 2019. N. 1. P. 1582—1587.

12. **Попов А. Ю.** Применение вычислительных систем с многими потоками команд и одним потоком данных для решения задач оптимизации // Инженерный журнал: наука и инновации (электронное издание). 2012. № 1. URL: http://engjournal.ru/catalog/it/hidden/80.html (дата обр. 27.05.2019).

13. **Подольский В. Э.** Об организации параллельной работы некоторых алгоритмов поиска кратчайшего пути на графе в вычислительной системе с многими потоками команд и одним потоком данных // Наука и Образование. Электронный журнал. 2015. № 4.

14. **Sharma P., Rehal M., Bangotra P. K.** Defect Data Classification and Analysis in VLSI Fabrication // 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore. 2018. P. 1447—1453.

A. Yu. Popov, Ph. D., Associate Professor of the Department "Computer Systems and Networks",
e-mail: alexpopov@bmstu.ru,
Bauman Moscow State Technical University, Moscow

Application of a Heterogeneous Computing System with a Discrete Mathematics Instruction Set to Solve Large-Scale Graphs Problems

The interest to the complex data models analysis has grown significantly in the last decade. Graphs known as the most adequate form of the real data representing in many areas from social networks, computer programs structure, topology of integrated circuits, to the bioinformatics and many others. As the size of datasets increases the need to find more efficient methods and tools for analyzing large graphs, including on the basis of more advanced hardware technical solutions, becomes obvious. Modern GPUs have great parallelism and performance, but cannot solve the fundamental problems of the graph processing: the data dependencies problem, the distribution of irregular graphs computing workload on multiple processor devices, as well as conflicts with memory access for many cores. We introduce a specialized microprocessor Leonhard x64 and the Discrete mathematics Instruction Set Computer (DISC) for large dimension graph processing, which was developed in the Bauman University. This article provides information about the discrete mathematics operators and the corresponding Leonhard x64 instructions, and justifies the architecture principles of a heterogeneous computing system based on it. With examples of Dijkstra and Belman-Ford algorithms we show the difference between the implementation of graph processing and the efficiency of universal computing systems, heterogeneous computing systems based on GPGPU, as well as Leonhard x64 microprocessor-based system.

Keywords: *discrete mathematics instruction set computer, Leonhard microprocessor, graph, data structure, GPGPU*

DOI: 10.17587/it.25.682-690

References

1. **Patel R., Kumar S.** *2018 4th International Conference on Information Technology (RAIT)*, Dhanbad, 2018, pp. 1—5.

2. **Patel G. R., Kumar S.** *IUP Journal of Computer Sciences*, Jan 2017, vol. 11, no. 1, pp. 38—50.

3. **Harish P., Narayanan P. J.** (2007) Accelerating Large Graph Algorithms on the GPU Using CUDA, *High Performance Computing — HiPC 2007. Lecture Notes in Computer Science,* vol. 4873, Springer, Berlin, Heidelberg.

4. **Huang I. J., Peng T. C.** *I. J. Trans.*, 2002, vol. E85—D, no. 6, p. 929—939.

5. **Kasmi N., Mahmoudi S. A., Zbakh M., Manneback P.** *2014 Second World Conference on Complex Systems (WCCS),* Agadir, 2014, pp. 23—27.

6. **Khoroshevsky B. G.** Architecture of computing systems: Proc. Allowance, Moscow, Publishing House of Moscow State Technical University H. E. Bauman, 2008, 520 p. (in Russian).

7. **Kormen T., Leyzerson Ch., Rivest R.** Algorithms: construction and analysis, Moscow, MTSNMO, 2000, 960 p. (in Russian).

8. **Popov A. Yu.** *Science and Education. MGTU them. N. E. Bauman. Electron. Journals*, 2014, no. 9, pp. 162-180 (in Russian).

9. **Popov A.** *HICSS50. Proceedings of the 50th Hawaii International Conference on System Sciences*, 2017, pp. 1003—1012.

10. **Abdymanapov C., Popov A.** *2019 IEEE Conference of Young Researchers in Electrical and Electronic Engineering (EIConRus),* 2019, no. 1, pp. 1844—1847.

11. **Rasheed B., Popov A. Y.** *2019 IEEE Conference of Young Researchers in Electrical and Electronic Engineering (EIConRus),* 2019, no. 1, pp. 1582—1587.

12. **Popov A. Yu.** *Engineering Journal: Science and Innovations (electronic edition),* 2012, no. 1, available at: http://engjournal.ru/catalog/it/hidden/80.html (date sample 05/27/2019) (in Russian).

13. **Podolsky V. E.** *Science and Education. Electronic journal*, 2015, no. 4 (in Russian).

14. **Sharma P., Rehal M., Bangotra P. K.** 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, 2018, pp. 1447—1453.