УДК 004.021

ИССЛЕДОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ ЗАДАЧИ ПОИСКА ВШИРЬ В ГРАФЕ НА СОПРОЦЕССОРАХ СЕМЕЙСТВА INTEL XEON PHI

Е. А. Головина[1], **А. С. Семенов**[2], **А. С. Фролов**[3]

Одним из важнейших алгоритмов обработки графов является поиск вширь, лежащий в основе рейтинга суперкомпьютеров Graph500. Графовые задачи характеризуются интенсивным нерегулярным доступом к памяти и обычно решаются на современных процессорах с низкой эффективностью. В статье представлены результаты исследования выполнения поиска вширь в графе на новых сопроцессорах семейства Intel Xeon Phi. Для получения высокой производительности применен потоковый подход с эффективным использованием пропускной способности памяти при последовательном доступе и с сохранением нерегулярного доступа к памяти, при этом необходимо выполнение ручной развертки цикла и преднакачки данных в кэш. В сравнении с Intel Xeon E5-2660 для разных графов Intel Xeon Phi 7120P оказался в среднем быстрее на 37%, в лучшем случае — на 78%; Intel Xeon Phi 5110P быстрее Intel Xeon E5-2660 в лучшем случае на 34%, медленнее в худшем случае на 29%, в среднем производительность приблизительно одинаковая. Полученный на Intel Xeon Phi 7120P результат в 4366 миллионов пройденных дуг в секунду вошел в ноябрьскую редакцию рейтинга Graph500 (2013 г.) и занял 89-е место среди всех систем и 4-е место среди исследовательских групп в классе одноузловых систем на базе платформы x86. Авторы благодарят компанию “Свет Компьютерс” за предоставленную расчетную систему IntellectDigital SciPhi 470 с сопроцессором Intel Xeon Phi 7120P.

Ключевые слова: поиск вширь в графе, BFS, Intel Xeon Phi.

1. Введение. Обработка больших графов — относительно новая, интенсивно развивающаяся область приложений для суперкомпьютерных и кластерных систем, характеризуемая интенсивной работой с большими объемами данных, размещенных в оперативной памяти вычислительных узлов, низкой пространственно-временно́й локализацией обращений к памяти и, как следствие, неэффективным использованием иерархической кэш-памяти процессоров, большим количеством промахов в кэш дескрипторов страниц TLB (Translation Lookaside Buffer) и резким снижением эффективности работы встроенного в процессор контроллера памяти.

Использование специализированных ускорителей и сопроцессоров, таких как NVIDIA Kepler, AMD Fire-Stream и Intel Xeon Phi, находит все большее применение в составе крупных высокопроизводительных вычислительных систем, занимающих первые строки в списке Top500. Так, например, первое место в новой редакции списка Top500 (ноябрь 2013 г.) занимает китайский суперкомпьютер, построенный с использованием 48 тысяч ускорителей Intel Xeon Phi с общей пиковой производительностью около 50 PFLOP/s.

Графические ускорители NVIDIA и AMD изначально разрабатывались для решения задач, хорошо отображающихся на потоковую архитектуру, например для задач, работающих с плотно заполненными матрицами и регулярными потоками данных. Однако даже в случае графических ускорителей приложения с интенсивным нерегулярным доступом к памяти могут быть определенным образом оптимизированы для достижения высокой производительности. В отличие от графических ускорителей сопроцессор Intel Xeon Phi имеет архитектуру мультитредового многоядерного процессора и в большей степени универсален, что потенциально позволяет эффективно применить его для решения задач с интенсивной работой с памятью, таких как графовые приложения и нерегулярные сеточные методы с адаптивными сетками. Однако, на первый взгляд, достижение высокой производительности на Intel Xeon Phi связано с наличием 512-битных векторных устройств, а векторизация графовых приложений представляется довольно затруднительной.

[1] Научно-исследовательский центр электронной вычислительной техники (ОАО “НИЦЭВТ”), Варшавское шоссе, 125, 117587, Москва; инженер-программист, e-mail: golovina@nicevt.ru

[2] Научно-исследовательский центр электронной вычислительной техники (ОАО “НИЦЭВТ”), Варшавское шоссе, 125, 117587, Москва; начальник сектора, e-mail: semenov@nicevt.ru

[3] Научно-исследовательский центр электронной вычислительной техники (ОАО “НИЦЭВТ”), Варшавское шоссе, 125, 117587, Москва; начальник отдела, e-mail: frolov@nicevt.ru

В настоящей статье обсуждаются результаты исследования производительности ускорителей Intel Xeon Phi на задаче поиска достижимых вершин в графе методом поиска вширь и сравнения с производительностью, полученной на 8-ядерном процессоре Intel Sandy Bridge-EP. Задача поиска вширь лежит в основе рейтинга Graph500 [1], включающего в себя как большие суперкомпьютеры, так и многопроцессорные вычислительные узлы. В данной работе, являющейся расширением статьи [2], рассмотрены ускорители Intel Xeon Phi 5110P и 7120P, а исследование проведено на четырех видах графов с различными параметрами.

2. Алгоритмы реализации задачи поиска вширь в графе. Одним из наиболее популярных алгоритмов обработки графов является поиск вширь (Breadth-First Search, BFS). Для заданного графа и заданной вершины r этот алгоритм находит все вершины, достижимые через ребра графа от вершины r. Особенностью алгоритма является то, что алгоритм BFS не должен анализировать вершины, отстоящие от вершины r на расстояние $s+1$ ребер, до тех пор, пока не проанализирует все вершины, отстоящие от вершины r на расстояние s ребер. Вершины, отстоящие от исходной вершины r на заданное расстояние, называются уровнем.

В статье исследуются два подхода к параллельной реализации алгоритма BFS: Queue-based [3, 4] и Read-based [5]. В каждом подходе рассматриваются несколько алгоритмов: подход Queue-based — алгоритм naive, алгоритм block; подход Read-based — нисходящий алгоритм read, восходящий гибридный алгоритм hybrid.

Подход Queue-based основан на непосредственном распараллеливании обработки очереди вершин, расположенных на каждом уровне. Подход Read-based изначально предложен для GPU, является потоковым и заключается в итерационной параллельной обработке массива, содержащего номер уровня для каждой вершины. В разделе 3.2 будут рассмотрены несколько оптимизаций алгоритмов подхода Read-based.

Во всех рассматриваемых алгоритмах граф задается матрицей смежности, которая является разреженной и хранится в формате CRS (Compressed Row Storage) с плотным хранением строк ненулевых элементов. Распараллеливание осуществлено при помощи технологии OpenMP.

2.1. Подход Queue-based. В первом подходе к реализации поиска вширь в графе, который носит название Queue-based и схема которого изображена ниже, каждому уровню соответствует массив вершин Q, в результате обхода которых формируется массив Q_{next} вершин следующего уровня. При параллельной работе тредов необходимо, чтобы каждый тред добавлял значение в Q_{next} с использованием атомарной операции __sync_fetch_and_add. Для анализа вершин нужно хранить в массиве *marked* информацию о том, была ли вершина пройдена ранее.

В рамках подхода Queue-based рассмотрено два алгоритма. Первый алгоритм naive просто реализует описанный подход:

```
1  Q_counter = 1 // инициализация счетчика вершин Q
2  Q[0] = r // инициализация текущего уровня Q
3  marked[r] = 1 // отметка начальной вершины r
4  while Q_counter > 0
5    Q_next_counter = 0 // обнуление счетчика следующего уровня
6    #pragma omp parallel // параллельная обработка уровня
7    for all vertex in Q do
8      for all w : (vertex, w) in E do
9        if marked[w] == 0 then // если вершина не отмечена
10         Q_next[__sync_fetch_and_add(Q_next_counter, 1)] = w // добавление w в Q_next
11         marked[w] = 1 // отметка вершины w
12       end if
13     end for
14   end for
15   swap(Q, Q_next) // переход на следующий уровень: обмен Q и Q_next, Q_counter = Q_next_counter
16 end while
```

Во втором, "блочном" (block) алгоритме для уменьшения числа использований атомарной операции каждый тред получает порцию элементов для заполнения массива Q_{next} размера *size* [4]. Как только эта порция заполнена, тред с помощью атомарной операции получает следующую порцию для заполнения. В результате атомарная операция вызывается не с каждой отмеченной вершиной, а в *size* раз реже.

2.2. Подход Read-based. Подход Read-based (или потоковый подход) значительно отличается от подхода Queue-based. Основным рабочим массивом в подходе Read-based является массив *levels*, длина

которого равна числу вершин в графе, а в каждой ячейке хранится номер уровня, на котором расположена вершина, или -1, если вершина еще не обрабатывалась. На каждом уровне просматривается весь массив $levels$ и тем или иным способом анализируются вершины. В рамках подхода Read-based мы рассматриваем нисходящий алгоритм read и гибридный алгоритм hybrid.

2.2.1. Нисходящий алгоритм. В нисходящем алгоритме read на каждом уровне просматривается весь массив $levels$ с начала до конца и анализируются те вершины, которые находятся на текущем уровне. Смежным им вершинам в массиве $levels$ приписывается следующий уровень и, тем самым, они помечаются. Схема алгоритма read (подход Read-based) имеет следующий вид:

```
1  numLevel = 0 // инициализация номера уровня
2  levels[r] = numLevel // начальная вершина r будет обрабатываться на первом уровне
3  numLevelVerts = 1 // количество вершин на уровне
4  while numLevelVerts > 0 // есть вершины на текущем уровне
5    numLevelVerts = 0
6    #pragma omp parallel for reduction (+:numLevelVerts) // параллельно, с редукцией
7    for all vertex in V do // для всех вершин графа
8      if levels[vertex] == numLevel then // только для вершин текущего уровня
9        for all w : (vertex, w) in E do // для всех вершин w, смежных с vertex
10         if levels[w] == −1 then // если уровень вершины не отмечен
11           levels[w] = numLevel + 1 // отметка ее для следующего уровня
12           numLevelVerts = numLevelVerts + 1
13         end if
14       end for
15     end if
16   end for
17   numLevel = numLevel + 1
18 end while
```

2.2.2. Гибридный алгоритм. В статье [6] предложена остроумная идея для реализации поиска вширь в графе. Для большого количества типов графов количество вершин на каждом следующем уровне резко возрастает, достигая максимальных значений на нескольких уровнях. Затем количество вершин на уровне падает. Когда вершин на уровне становится много, а значительная часть поиска уже позади, среди соседей вершин текущего уровня очень мало неотмеченных вершин; таким образом, бóльшая часть работы производится впустую. Для таких уровней становится выгодным использовать восходящий алгоритм обработки вершин, при котором просматриваются соседи всех неотмеченных вершин, а если какой-то сосед неотмеченной вершины лежит на текущем уровне, то данная вершина является потомком этого соседа в дереве поиска, при этом других соседей данной вершины просматривать уже не нужно. Восходящий алгоритм также может быть реализован в рамках подхода Read-based. Схема обработки одного уровня восходящим алгоритмом (подход Read-based) имеет следующий вид:

```
1  #pragma omp parallel for reduction (+:numLevelVerts) // параллельно, с редукцией
2  for all vertex in V do // для всех вершин графа
3    if levels[vertex] == −1 then // если вершина vertex не отмечена
4      for all w : (vertex, w) in E do // для всех вершин w, смежных с vertex
5        if levels[w] == numLevel then // если w принадлежит текущему уровню
6          levels[vertex] = numLevel + 1 // отметка vertex для следующего уровня
7          numLevelVerts = numLevelVerts + 1
8          break // выход из цикла for просмотра вершин w
9        end if
10     end for
11   end if
12 end for
```

В дальнейшем ниже исследуется гибридный алгоритм hybrid, в котором для первых и последних уровней используется нисходящий алгоритм read подхода Read-based (в отличие от статьи [6], в которой используется алгоритм подхода Queue-based), а для некоторых уровней обработка производится при помощи восходящего алгоритма.

3. Исследование производительности. Производительность алгоритмов реализации поиска вширь в графе исследовалась на процессоре Intel Xeon Sandy Bridge-EP и сопроцессорах Intel Xeon Phi 5110P и Intel Xeon Phi 7120P, которые условно будем называть Sandy Bridge-EP, Phi-5110P и Phi-7120P соответ-

Таблица 1

Характеристики используемых систем

	Sandy Bridge-EP	Phi-5110P	Phi-7120P
Название модели	Xeon E5-2660	Xeon Phi 5110P	Xeon Phi 7120P
Частота, ГГц	2.2	1.05	1.238
Количество сокетов	1	1	1
Количество ядер	8	60	61
Количество аппаратных тредов в ядре	2	4	4
Объем памяти кэшей данных, КБ	64* / 2 МБ* / 20 МБ	32* / 512*	32* / 512*
Объем установленной памяти, ГБ	32	8	16
Тип используемой памяти	DDR3	GDDR5	GDDR5
Пропускная способность памяти, ГБ/с	51	352	352
Задержка обращения в память, тактов	200	300	350

* — в расчете на ядро

ственно, их основные характеристики приведены в табл. 1. Сопроцессоры Phi-5110P и Phi-7120P — это реализации одного и того же ядра сопроцессора, причем Phi-7120P представлен во втором квартале 2013 г., на полгода позже сопроцессора Phi-5110P. Сопроцессор Phi-7120P отличается повышенной частотой по технологии Intel®Turbo Boost Technology и вдвое бóльшим объемом памяти.

Характеристикой производительности алгоритмов BFS принято считать количество миллионов пройденных ребер графа в секунду (MTEPS, Traversed Edges Per Second). Для получения результатов на Intel Xeon Phi все алгоритмы запускались непосредственно на сопроцессоре в режиме native.

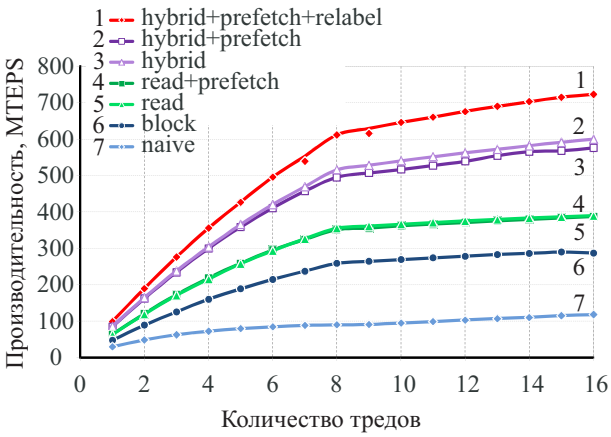

Рис. 1. Производительность различных алгоритмов поиска вширь на случайном графе с числом вершин 134 миллиона и средней связностью вершины $k = 8$ в зависимости от числа используемых тредов на процессоре Sandy Bridge-EP

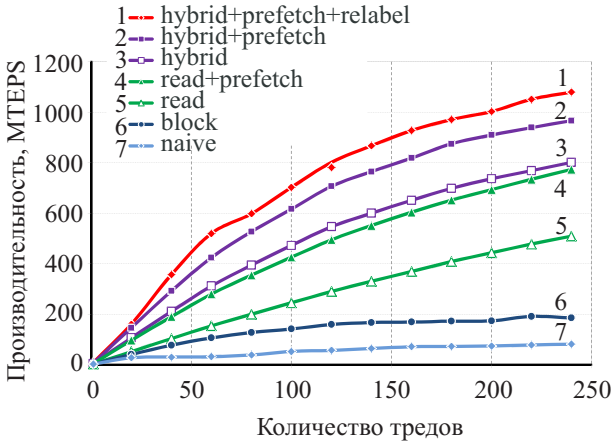

Рис. 2. Производительность различных алгоритмов поиска вширь на случайном графе с числом вершин 134 миллиона и средней связностью вершины $k = 8$ в зависимости от числа используемых тредов на сопроцессоре Phi-5110P

3.1. Исследование производительности разработанных алгоритмов. На рис. 1 и 2 приведены графики производительности различных алгоритмов поиска вширь в графе для неориентированного случайного графа (графа Пуассона) с числом вершин 134 миллиона и средней связностью вершины $k = 8$ в зависимости от числа используемых тредов для Sandy Bridge-EP и Phi-5110P. Следует отметить, что для одного треда для алгоритма naive скорость работы Phi-5110P в 20 раз ниже по сравнению с Sandy Bridge-EP. Поэтому достижение высокой производительности на сопроцессорах Intel Xeon Phi связано в первую очередь с масштабируемостью алгоритмов при использовании большого числа тредов.

Алгоритмы naive и block подхода Queue-based показывают очень низкую масштабируемость, несмотря на то что в алгоритме block использование атомарной операции сокращено.

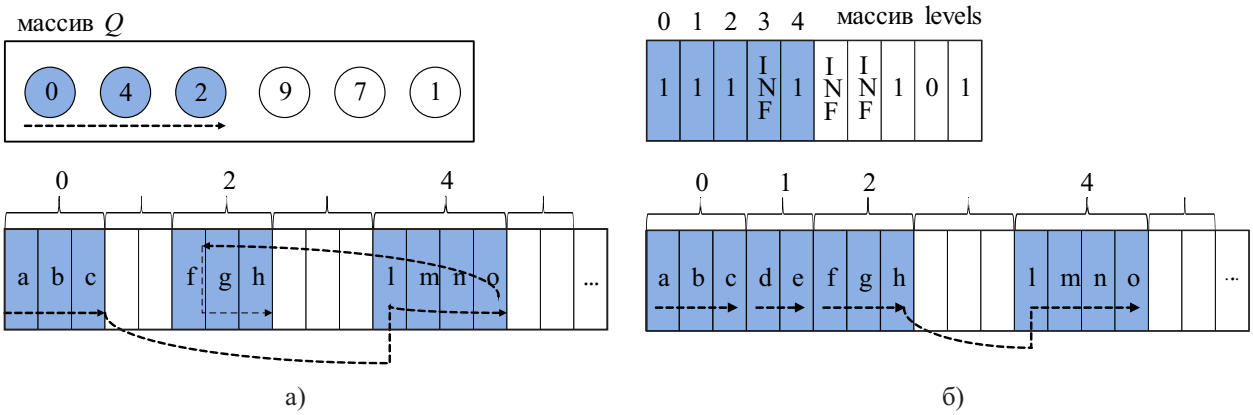

Рис. 3. Шаблоны доступа к памяти в подходах (а) Queue-based и (б) Read-based

Алгоритм read дает качественно другой результат, что связано с преимуществами подхода Read-based. Во-первых, в нем отсутствуют атомарные операции, которые очень сильно ограничивают масштабирование задачи с ростом числа используемых ядер и тредов. Во-вторых, совершенно исчезают массивы Q и Q_{next}, что приводит к экономии объема и пропускной способности памяти. Минимизация объема памяти означает более эффективное использование кэша данных. И наконец, в-третьих, пространственная локализация обращений к памяти для подхода Read-based значительно выше по сравнению с подходом Queue-based. На рис. 3а изображена очередь Q с вершинами $\{0, 4, 2, 9, 7, 1\}$ и показан доступ к массиву номеров соседей формата CRS для первых трех вершин очереди Q. На рис. 3б изображен доступ к данным для того же уровня с номером 1 для подхода Read-based. Доступ к массиву *levels* является полностью последовательным, а к массиву номеров соседей — последовательным с возможными пропусками. Последовательный доступ позволяет использовать высокую пропускную способность памяти, увеличивает эффективность использования кэша, позволяет эффективно работать программной и аппаратной преднакачке, уменьшает количество промахов в кэш дескрипторов страниц TLB. Кроме того, при распараллеливании треды лучше разделяют между собой массив номеров соседей, не закачивая его в кэши ядер по нескольку раз.

Потоковый подход Read-based был изначально предложен для графических процессоров, и именно эффективное использование в этом подходе пропускной способности подсистемы памяти натолкнуло авторов настоящей статьи на мысль о том, что он может быть с успехом применен для сопроцессоров Intel Xeon Phi.

Таблица 2

	Последовательный доступ		Случайный доступ	
	Чтение	Запись	Чтение	Запись
Sandy Bridge-EP	42	19	3.3	2.2
Phi-5110P	183	160	3.8	3.4

Можно предположить, что недостаток подхода Read-based заключается в лишней работе, связанной с просмотром на каждой итерации всего массива *levels*. Однако скорость последовательного доступа к массиву очень велика; кроме того, время обработки вершин массива *levels*, которые не принадлежат текущему уровню, очень мало. Поэтому накладные расходы этого алгоритма незначительны.

Тем не менее, случайный доступ остается в подходе Read-based. В строке 10 схемы алгоритма read (см. раздел 2.2.1) обращение к массиву *levels* происходит по вершине w из списка соседних вершин, номер которой часто случаен. Характеристики пропускной способности памяти в ГБ/с при чтении и записи для Sandy Bridge-EP и Phi-5110P при последовательном и случайном шаблонах доступа к памяти приведены в табл. 2. Результаты для векторизованных вариантов программ взяты из [7] для сопроцессора Intel Xeon Phi SE10P. Все остальные результаты получены при помощи пакета DISBench [8]. Сравнение показывает, насколько дорого обходится случайный доступ к памяти. Это связано с тем, что во время работы с потоком случайных адресов при выборке одного слова данных из подгружаемой кэш-строки резко снижается эффективность работы встроенного в процессор контроллера памяти как за счет уменьшения количества полезных данных, передаваемых по шине памяти, так и за счет выполнения дополнительных команд непосредственно в микросхемах памяти, необходимых для работы по случайным адресам.

На рис. 1 и 2 видно, что алгоритм read на Sandy Bridge-EP и Phi-5110P показывает очень хорошую масштабируемость. По общей производительности Phi-5110P уже обгоняет Sandy Bridge-EP: 509 MTEPS против 390 MTEPS.

Алгоритм hybrid позволяет значительно ускорить обработку некоторых уровней, что сказывается на общей производительности. Однако в этом алгоритме по-прежнему присутствует случайный доступ к массиву $levels$.

3.2. Оптимизация. Чтобы понять, что является узким местом реализации на Intel Xeon Phi — пропускная способность или задержка случайных обращений, в алгоритм read вводится ручная развертка цикла на строке 9 (см. схему алгоритма read в разделе 2.2.1) и ручная преднакачка значения $levels[w]$ в кэш. Для сопроцессора Phi-5110P на одном треде производительность повысилась в 2.1 раза. Для 240 тредов производительность новой реализации (алгоритм read+prefetch) по сравнению с базовым алгоритмом read выросла на Phi-5110P на 52% до 773 MTEPS (рис. 2). Такое улучшение результатов показывает, что узким местом алгоритма read на архитектуре Intel Xeon Phi была задержка случайных обращений в память. В новом алгоритме read+prefetch ограничением, по-видимому, является уже пропускная способность памяти при случайном доступе.

Таблица 3

Используемые графы

Обозначение	Тип графа	# вершин	# дуг, M	AvgDegree	Directed	Генератор
random-k-n	Случайный	2^n	$k * 2^n$	k	Нет	собств.
RMAT-k-n	RMAT	2^n	$k * 2^n$	k	Да	[12]
SSCA2-25	SSCA2	2^{25}	267.8	8.0	Да	[13]
SSCA2-26	SSCA2	2^{26}	720.1	10.7	Да	[13]
graph500-25	Кронекер	2^{25}	1047.2	31.2	Нет	[1]

В то же время на процессоре Sandy Bridge-EP производительность алгоритма read+prefetch совпадает с производительностью алгоритма read, что свидетельствует о том, что для алгоритма read ограничением на Sandy Bridge-EP уже являлась, по-видимому, пропускная способность памяти.

Для алгоритма hybrid аналогично введена развертка цикла и ручная преднакачка данных в кэш, этот алгоритм на рис. 1 и 2 называется hybrid+prefetch.

Другим возможным способом повышения производительности является локализация данных даже для случайного обращения к массиву $levels$. Для повышения пространственной и временно́й локализации обращений к массиву $levels$ (строка 10 алгоритма read в разделе 2.2.1) матрица графа подвергается предобработке. Применяется обратный алгоритм Cuthill-McKee [9] для приведения матрицы графа к ленточной структуре и возможному уменьшению ширины этой ленты. В результате, так как строки матрицы обрабатываются в алгоритмах read и hybrid последовательно в соответствии с последовательным движением по массиву $levels$, возрастает количество кэш-попаданий при обращении к массиву $levels[w]$, где w — вершина-сосед для данной обрабатываемой вершины. Кроме того, списки вершин-соседей сортируются для уменьшения количества TLB-промахов. Алгоритм hybrid+prefetch с предобработкой будем называть hybrid+prefetch+relabel. Производительность последнего алгоритма по сравнению с hybrid+prefetch для максимального количества тредов на Phi-5110P увеличилась на 12%.

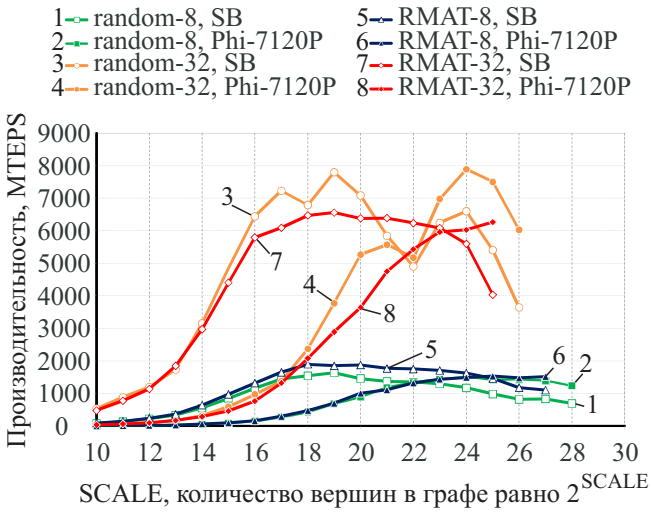

Рис. 4. Производительность лучшего алгоритма для Sandy Bridge-EP (SB) и Phi-7120P в зависимости от количества вершин в графе для различных графов

3.3. Сравнение производительности. Сравнивать производительность сопроцессоров Intel Xeon

Phi и Intel Xeon Sandy Bridge-EP мы будем на четырех видах графов: случайных графах Пуассона, графах RMAT [10], графах из теста SSCA2 [11] и графах Кронекера из Graph500 [1]. Рассматриваемые графы приведены в табл. 3, в которой приняты следующие обозначения: # — количество, M — миллионы, AvgDegree — средняя связность, Directed — ориентированность, собств. — генератор, разработанный авторами настоящей статьи. В дальнейшем в тексте обозначения random-k и RMAT-k означают случайный и RMAT-графы со средней связностью k.

Производительность лучшего из алгоритмов hybrid+prefetch и hybrid+prefetch+relabel (с предобработкой) для сопроцессора Phi-7120P (244 треда) и процессора Sandy Bridge-EP (16 тредов) для четырех типов графов в зависимости от размера графа представлена на рис. 4. На каждом типе графа для Phi-7120P была подобрана оптимальная развертка циклов с преднакачкой, исследовались развертки на 2, 4, 8, 16. Исследуя приведенные графики, можно отметить следующие закономерности.

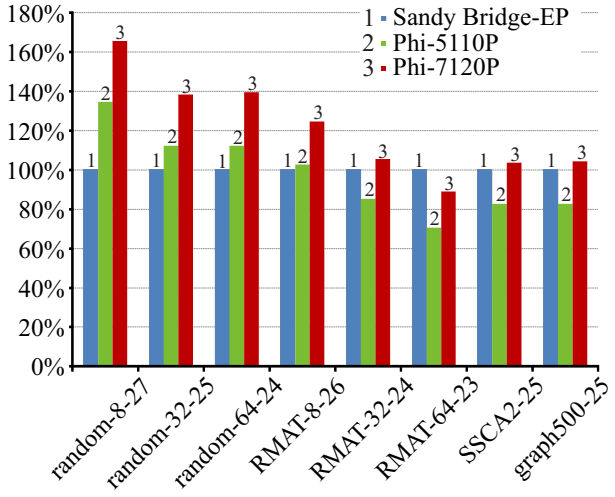

Рис. 5. Сравнение производительности лучших алгоритмов на Sandy Bridge-EP, Phi-5110P и Phi-7120P для самых больших графов, помещающихся в 8 ГБ памяти сопроцессора Phi-5110P

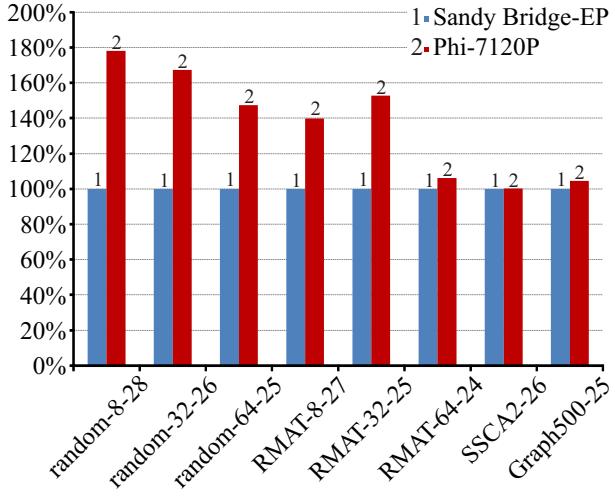

Рис. 6. Сравнение производительности лучших алгоритмов на Sandy Bridge-EP и Phi-7120P для самых больших графов, помещающихся в 16 ГБ памяти сопроцессора Phi-7120P

Таблица 4

Организация	Система	Graph500 Scale	GTEPS
Chuo University	4x Intel(R) Xeon(R) CPU E5-4650	27	31.6
University of Tsukuba	Xeon E5-2650 v2, GeForce GTX TITAN	25	17.2
National University of Defense Technology	SMP, x86-based	24	9.7
DISLab, NICEVT / svetcorp.net	Intel Xeon Phi 7120P	25	4.4

Для Sandy Bridge-EP производительность сначала быстро растет, а затем, когда данные перестают помещаться в кэше, постепенно падает. Можно сделать вывод, что процессор Sandy Bridge-EP плохо приспособлен к решению задач с интенсивным нерегулярным доступом к большому объему данных. Intel Xeon Phi ведет себя наоборот: производительность очень медленно растет и в какой-то точке опережает Sandy Bridge-EP. Причина этого в том, что Intel Xeon Phi изначально рассчитан на массовый параллелизм и эффективное использование пропускной способности памяти.

Общее сравнение производительности Sandy Bridge-EP, Phi-5110P и Phi-7120P на восьми самых больших графах, помещающихся в 8 ГБ памяти Phi-5110P, приведено на рис. 5. Максимальная, средняя и минимальная производительность на Phi-5110P и Phi-7120P составляет 134%, 98%, 71% и 165%, 121%, 89% от производительности на Sandy Bridge-EP. Общее сравнение производительности Sandy Bridge-EP и Phi-7120P на восьми самых больших графах, помещающихся в 16 ГБ, представлено на рис. 6. Максимальная, средняя и минимальная производительность Phi-7120P составляет 178%, 137% и 100% от

производительности Sandy Bridge-EP. Таким образом, увеличение частоты ядра Phi-7120P по сравнению с Phi-5110P, а также вдвое больший объем памяти имеют решающее значение для получения хороших результатов на Intel Xeon Phi.

Лучший из реализованных в работе алгоритмов использовался для получения на сопроцессорах Intel Xeon Phi результатов для рейтинга Graph500. На графе с 8 миллионами вершин на сопроцессоре Intel Xeon Phi 5110P получена производительность 1801 MTEPS, этот результат вошел в Graph500 в ноябре 2013 г. и занял 105-е место. Сопроцессор Intel Xeon Phi 7120P вошел в Graph500 и занял 89-е место с производительностью 4366 MTEPS на графе с 32 миллионами вершин. Других результатов на Intel Xeon Phi в этом списке нет, а в классе одноузловых x86-систем этот результат является четвертым среди исследовательских групп. В табл. 4 показана производительность одноузловых систем на базе платформы x86 в списке Graph500 для различных исследовательских групп.

4. Заключение. В статье рассмотрены разные подходы и алгоритмы реализации задачи поиска вширь в графе (Breadth-First Search, BFS). Наиболее эффективным для процессора Intel Xeon Sandy Bridge-EP и сопроцессора Intel Xeon Phi оказался потоковый подход с эффективным использованием пропускной способности памяти при последовательном доступе и с сохранением при этом нерегулярного доступа к памяти. Алгоритм реализации в рамках данного подхода характеризуется отсутствием атомарных операций. Однако для сопроцессора Intel Xeon Phi для получения высокой производительности пришлось осуществлять ручные развертку цикла и преднакачку данных в кэш.

В отличие от Intel Xeon Sandy Bridge-EP, на Intel Xeon Phi производительность, в основном, растет с ростом размера графа. В настоящей статье рассматривались две версии сопроцессора Intel Xeon Phi — Intel Xeon Phi 5110P и Intel Xeon Phi 7120P, последний отличается большей частотой и вдвое бóльшим объемом памяти — 16 ГБ. В сравнении с Intel Xeon E5-2660 на восьми самых больших графах, помещающихся в 16 ГБ, Intel Xeon Phi 7120P оказался в среднем быстрее на 37%, в лучшем случае — на 78%. На восьми самых больших графах, помещающихся в 8 ГБ, Intel Xeon Phi 5110P быстрее Intel Xeon E5-2660 в лучшем случае на 34%, медленнее в худшем случае на 29%, в среднем производительность приблизительно одинаковая. Таким образом, Intel Xeon Phi 7120P значительно предпочтительнее младшей модели — увеличение частоты ядра этого сопроцессора, а также вдвое бóльшая память оказывают решающее значение для получении хороших результатов на задаче поиска вширь в графе.

Лучший из реализованных в работе алгоритмов использовался для получения на сопроцессорах Intel Xeon Phi результатов для рейтинга Graph500. Сопроцессор Intel Xeon Phi 7120P вошел в Graph500 на 89-м месте с производительностью 4366 MTEPS на графе с 32 миллионами вершин. Других результатов на Intel Xeon Phi в этом списке нет, а в классе одноузловых x86-систем этот результат является четвертым среди исследовательских групп в классе одноузловых систем на базе платформы x86.

Авторы выражают благодарность компании “Свет Компьютерс” [14] за предоставленный для тестирования персональный суперкомпьютер IntellectDigital SciPhi 470 с сопроцессором Intel Xeon Phi 7120P, а также за доброжелательное отношение и оперативное решение всех вопросов.

Статья рекомендована к публикации Программным комитетом Международной конференции “Научный сервис в сети Интернет: все грани параллелизма” (http://agora.guru.ru/abrau2013).

СПИСОК ЛИТЕРАТУРЫ

1. Graph500 benchmark (http://www.graph500.org/).
2. *Головина Е.А., Семенов А.С., Фролов А.С.* Исследование производительности задачи поиска вширь в графе на сопроцессоре Intel Xeon Phi // Тр. Межд. суперкомпьютерной конференции “Научный сервис в сети Интернет: все грани параллелизма”, 23–28 сентября 2013 г., г. Новороссийск. М.: Изд-во Моск. гос. ун-та, 2013. 135–141.
3. *Agarwal V., Petrini F., Pasetto D., Bader D.* Scalable graph exploration on multicore processors // Proc. 2010 ACM/IEEE International Conference on High Performance Computing, Networking, Storage, and Analysis (SC'10). New York: IEEE Press, 2010. 1–11.
4. *Saule E., Catalyeurek U.* An early evaluation of the scalability of graph algorithms on the Intel Phi architecture // IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW'12). New York: IEEE Press, 2012. 1629–1639.
5. *Hong S., Oguntebi T., Olukotun K.* Efficient parallel graph exploration on multi-core CPU and GPU // Proc. Int. Conf. on Parallel Architectures and Compilation Techniques (PACT'11). New York: IEEE Press, 2011. 78–88.
6. *Beamer S., Asanović K., Patterson D.* Direction-optimizing breadth-first search // Proc. Int. Conf. on High Performance Computing, Networking, Storage and Analysis (SC'12). New York: IEEE Press, 2012. 1–10.
7. *Saule E., Kaya K., Catalyurek U.* Performance evaluation of sparse matrix multiplication kernels on Intel Xeon Phi // arXiv:1302.1078, 5 Feb 2013 (http://arxiv.org/pdf/1302.1078.pdf).

8. *Frolov A., Gilmendinov M.* DISBench: benchmark for memory performance evaluation of multicore multiprocessor // Lecture Notes in Computer Science. Vol. 7979. Heidelberg: Springer, 2013. 197–207.
9. *Cuthill E., McKee J.* Reducing the bandwidth of sparse symmetric matrices // Proc. 24th National Conference (ACM'69). New York: ACM Press, 1969. 157–172.
10. *Chakrabarti D., Zhan Y., Faloutsos C.* R-MAT: a recursive model for graph mining // SIAM International Conference on Data Mining. Vol. 6. Philadelphia: SIAM, 2004. 442–446.
11. *Bader D., Madduri K.* Design and implementation of the HPCS graph analysis benchmark on symmetric multiprocessor // Lecture Notes in Computer Science. Vol. 3769. Heidelberg: Springer, 2005. 465–476.
12. *Bader D., Madduri K.* SNAP: Small-world network analysis and partitioning: an open-source parallel graph framework for the exploration of large-scale networks // Proc. Int. Parallel and Distributed Processing Symposium (IPDPS). New York: IEEE Press, 2008. 1–12.
13. *Bader D., Madduri K.* GTGraph: a synthetic graph generator suite. 2006 (http://www.cse.psu.edu/m̃adduri/software/GTgraph).
14. Компания "Свет Компьютерс" (http://svetcorp.net).

Поступила в редакцию
12.12.2013

Performance Evaluation of Breadth-First Search on Intel Xeon Phi

E. A. Golovina[1], A. S. Semenov[2], and A. S. Frolov[3]

[1] *Scientific Research Centre for Electronic Computer Technology; Varshavaskoe shosse 125, Moscow, 117587, Russia; Software Engineer, e-mail: golovina@nicevt.ru*

[2] *Scientific Research Centre for Electronic Computer Technology; Varshavaskoe shosse 125, Moscow, 117587, Russia; Ph.D., Head of Sector, e-mail: semenov@nicevt.ru*

[3] *Scientific Research Centre for Electronic Computer Technology; Varshavaskoe shosse 125, Moscow, 117587, Russia; Head of Department, e-mail: frolov@nicevt.ru*

Received December 12, 2013

Abstract: Breadth-First Search (BFS) is one of the most important kernels in graph computing. It is the main kernel of the Graph500 rating that evaluates performance of large supercomputers and multiprocessor nodes in terms of traversed edges per second (TEPS). In this paper we present the results of BFS performance evaluation on a recently released high-performance Intel Xeon Phi coprocessor. We examine the previously proposed Queue-based and Read-based approaches to BFS implementation. We also apply several optimization techniques, such as manual loop unrolling and prefetching, that significantly improve performance on Intel Xeon Phi. On a representative graph set, Intel Xeon Phi 7120P demonstrates 78% maximum and 37% average speedup as compared to the Intel Xeon E5-2660 processor. We achieve 4366 MTEPS on Intel Xeon Phi 7120P for a graph with scale 25, and have 89th place on the November 2013 Graph500 list. This is the fourth place among research teams in the class of single node x86-based systems. The authors would like to thank the Svet Computers company for the provided IntellectDigital SciPhi 470 desktop supercomputer with Intel Xeon Phi 7120P coprocessor.

Keywords: Breadth-First Search, graph algorithms, Intel Xeon Phi.

References

1. Graph500 benchmark, available at http://www.graph500.org/.

2. E. A. Golovina, A. S. Semenov, and A. S. Frolov, "Performance Evaluation of Breadth-First Search on a Graph Using Intel Xeon Phi," in *Proc. Int. Supercomputer Conf. on Scientific Service in Internet* (Mosk. Gos. Univ., Moscow, 2013), pp. 135–141.

3. V. Agarwal, F. Petrini, D. Pasetto, and D. Bader, "Scalable Graph Exploration on Multicore Processors," in *Proc. Int. Conf. on High Performance Computing, Networking, Storage, and Analysis* (IEEE Press, New York, 2010), pp. 1–11.

4. E. Saule and U. Catalyeurek, "An Early Evaluation of the Scalability of Graph Algorithms on the Intel Phi Architecture," in *IEEE 26th Int. Parallel and Distributed Processing Symposium Workshops* (IEEE Press, New York, 2012), pp. 1629–1639.

5. S. Hong, T. Oguntebi, and K. Olukotun, "Efficient Parallel Graph Exploration on Multi-Core CPU and GPU," in *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques* (IEEE Press, New York, 2011), pp. 78–88.

6. S. Beamer, K. Asanović, and D. Patterson, "Direction-Optimizing Breadth-First Search," in *Proc. Int. Conf. on High Performance Computing, Networking, Storage and Analysis* (IEEE Press, New York, 2012), pp. 1–10.

7. E. Saule, K. Kaya, and U. Catalyurek, *Performance Evaluation of Sparse Matrix Multiplication Kernels on Intel Xeon Phi*, arXiv preprint: 1302.1078 [cs.DC] (Cornell Univ. Library, Ithaca, 2013), available at http://arxiv.org/pdf/1302.1078.pdf.

8. A. Frolov and M. Gilmendinov, "DISBench: Benchmark for Memory Performance Evaluation of Multicore Multiprocessor," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2013), Vol. 7979, pp. 197–207.

9. E. Cuthill and J. McKee, "Reducing the Bandwidth of Sparse Symmetric Matrices," in *Proc. 24th ACM Natl. Conf.* (ACM Press, New York, 1969), pp. 157–172.

10. D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-MAT: A Recursive Model for Graph Mining," in *SIAM Int. Conf. on Data Mining* (SIAM, Philadelphia, 2004), Vol. 6, pp. 442–446.

11. D. Bader and K. Madduri, "Design and Implementation of the HPCS Graph Analysis Benchmark on Symmetric Multiprocessors," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2005), Vol. 3769, pp. 465–476.

12. D. Bader and K. Madduri, "SNAP: Small-World Network Analysis and Partitioning: An Open-Source Parallel Graph Framework for the Exploration of Large-Scale Networks," in *Proc. Int. Parallel and Distributed Processing Symposium* (IEEE Press, New York, 2008), pp. 1–12.

13. D. Bader and K. Madduri, *GTGraph: A Synthetic Graph Generator Suite*, available at http://www.cse.psu.edu/m̃adduri/software/GTgraph/ (2006).

14. Svet Computers Company, available at http://svetcorp.net/.