Vakhlakov Dmitriy Viktorovich – FGUP “NTC “Orion””; e-mail: melnikov@linfotech.ru; Moskow, Russia; researcher.

Peresypkin Vladimir Anatol’evich – e-mail: melnikov@linfotech.ru; research consultant; dr.. of eng. sc.

Germanovich Andrey Valer’evich – Moscow State University, Institute of Asian and African Studies; e-mail: melnikov@linfotech.ru; Moskow, Russia; assistant professor at the arabic philology department; cand. of history. sc.

Melnikov Sergey Yur’evich – OOO “Lingvisticheskie I informatsionye tehnologii” (Limited Liability Company); e-mail: melnikov@linfotech.ru; Moskow, Russia; deputy director; cand. of phys. and math. sc.

Copkalo Nikolaj Nikolaevich – Southern Federal University; e-mail: melnikov@linfotech.ru; Taganrog, Russia; senior researcher; cand. of eng. sc.

УДК 004.75 DOI 10.18522/2311-3103-2021-7-142-153

А.М. Альбертьян, И.И. Курочкин, Э.И. Ватутин

ИСПОЛЬЗОВАНИЕ ГЕТЕРОГЕННЫХ ВЫЧИСЛИТЕЛЬНЫХ УЗЛОВ В ГРИД-СИСТЕМАХ ПРИ РЕШЕНИИ КОМБИНАТОРНЫХ ЗАДАЧ

В настоящее время для решения больших вычислительных задач используются не только многопроцессорные вычислительные системы, но и различные виды распределенных систем. Распределенные вычислительные системы имеют ряд особенностей: возможное наличие отказов узлов и каналов связи, непостоянное время работы узлов, возможные ошибки в расчетах, гетерогенность вычислительных узлов. Под гетерогенностью вычислительных узлов будем понимать не только различную вычислительную способность и различные архитектуры центральных процессоров, но и наличие на узле других компонентов, способных проводить вычисления. К таким компонентам можно отнести видеокарты и математические сопроцессоры. Узел распределенной вычислительной системы будем называть гетерогенным, если помимо одного или нескольких центральных процессоров в его составе есть дополнительные вычислительные устройства. При решении вычислительной задачи на распределенной системе необходимо максимизировать использование всех доступных вычислительных ресурсов. Для этого необходимо не только распределить вычислительные подзадачи на узлы в соответствии с их вычислительной способностью, но и учесть особенности дополнительных вычислительных устройств. Исследованию методов максимизации использования ресурсов на гетерогенных узлах распределенной вычислительной системы посвящена эта работа. Основной целью данной работы является создание переносимого приложения, производящего параллельные вычисления с использованием многопоточной модели выполнения. При разработке приложения акцент делается на наиболее полном использовании доступных аппаратных ресурсов. Одним из основных требований к реализации является оптимизация производительности приложения для различных компьютерных архитектур, а также возможность параллельного выполнения приложения на разнородных вычислительных устройствах, входящих в состав гетерогенного вычислительного комплекса. Была исследована возможность применения ряда методов программно-алгоритмической оптимизации для многопроцессорных архитектур различных поколений. А также была проведена оценка эффективности их использования для высоконагруженных многопоточных приложений. Представлено решение проблемы квазиоптимального динамического распределения вычислительных заданий между всеми доступными на данный момент вычислительными устройствами гетерогенного вычислительного комплекса.

Распределенные вычисления; многопоточное приложение; гетерогенный вычислительный комплекс; повышение производительности; распределение вычислительных ресурсов; грид-система из персональных компьютеров; сопроцессор; Xeon Phi; ортогональные диагональные латинские квадраты; ДЛК.

A.M. Albertian, I.I. Kurochkin, E.I. Vatutin

THE USE OF HETEROGENEOUS COMPUTING NODES IN GRID SYSTEMS IN SOLVING COMBINATORIAL PROBLEMS

The main goal of this work is to create a parallel application that performs computations using a multithreaded execution model, optimized to make the best utilization of all available hardware resources. One of the main implementation requirements is to optimize application performance on different computer architectures, and to enable parallel execution of the application on various computing devices that are part of a heterogeneous computing system. The possibility of applying various methods of software and algorithmic optimization on multiprocessor architectures of different generations was investigated as well as the effectiveness of their use for highly loaded multithreaded applications was estimated. The problem of quasi-optimal dynamic distribution of computational tasks among all currently available computing devices of a heterogeneous computing system was also solved. Currently, not only multiprocessor computing systems are used to solve large computational problems, but also various types of distributed systems. Distributed computing systems have a number of features: possible failures of nodes and communication channels, unstable operating time of nodes, possible errors in calculations, heterogeneity of computing nodes. By heterogeneity of computing nodes, we will understand not only the different computing capacity and different architectures of central processors, but also the presence of other devices on the node capable of performing calculations. Such devices include video cards and mathematical coprocessors. A node of a distributed computing system will be called heterogeneous if, in addition to one or more central processing units, it contains additional computing devices. When solving a computational problem on a distributed system, it is necessary to maximize the utilization of all available computing resources. To do this, it is necessary not only to distribute computing subtasks to nodes in accordance with their computing capacity, but also to take into account the features of additional computing devices. This work is devoted to the study of methods for maximizing the resources utilization of heterogeneous nodes.

Distributed computing; multithreaded application; heterogeneous computing system; performance optimization; computing resources allocation; desktop grid; coprocessor; Xeon Phi; orthogonal diagonal latin squares; DLS.

Введение. Одним из популярных способов проведения крупных вычислительных экспериментов является использование распределенных вычислительных систем. К одному из видов достаточно широко распространенных распределенных вычислительных систем относятся грид-системы из персональных компьютеров (ГСПК) (англ. Desktop grid) [1]. Грид-система представляет собой слабосвязанную среду, состоящую из множества гетерогенных вычислительных устройств. Для грид-систем также характерны ненадежность и вариативность характеристик узлов и каналов передачи информации между ними, следствием чего является значительная автономность расчетов на отдельных узлах [2]. Вследствие этого, необходимым условием эффективного использования ГСПК является возможность разделения исходной вычислительной задачи на множество автономных подзадач [3]. Это является существенным ограничивающим фактором и дополнительно лимитирует типы задач, решаемых с помощью ГСПК [4–6]. Тем не менее, большим преимуществом ГСПК перед прочими системами, является возможность их практически неограниченного масштабирования, при условии вовлечения в процесс вычислений новых заинтересованных участников [7].

В отличие от специализированных вычислительных систем, таких как вычислительные кластеры, в составе ГСПК могут использоваться абсолютно различные по производительности и конфигурации вычислительные узлы [8], в большинстве случаев не рассчитанные на продолжительные вычисления с высокой нагрузкой на центральный процессор (ЦП) и прочие компоненты системы. И, как правило, невозможно централизованное администрирование или модернизация отдельных вычислительных узлов, входящих в ГСПК. Также большое влияние на планирова-

ние вычислительного процесса и распределение заданий между узлами [9] оказывает нестабильность и неоднородность каналов связи, приводящая к необходимости проведения дополнительных вычислений [10].

Все вышеперечисленные особенности ГСПК необходимо было учитывать в процессе разработки вычислительного приложения.

Цели и методы повышения эффективности работы приложения. Одной из основных задач при разработке приложения для ГСПК является оптимизация производительности для различных компьютерных архитектур, а также возможность параллельного выполнения приложения на разнородных вычислительных устройствах, входящих в состав гетерогенного вычислительного узла.

Одной из подзадач для вычислительных экспериментов, связанных с анализом различных характеристик ортогональных диагональных латинских квадратов (ДЛК) порядка 10, является задача нахождения канонических форм (КФ) ДЛК [11]. При этом общее время обработки заданий в наибольшей степени зависит именно от длительности ее выполнения, соответственно оптимизация производительности на данном этапе является приоритетной.

Таким образом, первоочередной задачей для оптимизации была выбрана минимизация времени выполнения данного приложения – канонизатора ДЛК по ЛК [11, 12], для наиболее распространенных компьютерных архитектур, прежде всего для Intel 64 (x86-64), при этом приложение совместимо с множеством прочих вычислительных систем и архитектур. Также была поставлена цель эффективного использования установленных на вычислительном узле сопроцессоров Intel Xeon Phi [13] поколения x100 (кодовое название Knights Corner, KNC) с архитектурой Intel Many Integrated Core (MIC), подключаемых к системе по шине PCIe [14]. Это также обусловило необходимость создания переносимого приложения, так как на сопроцессорах Intel Xeon Phi в качестве операционной системы (ОС) используется Linux при том, что основной ОС на узлах ГСПК во многих случаях являются различные версии Microsoft Windows. Для написания переносимого кода при разработке использовался язык C++ без дополнительных внешних библиотек и расширений, не входящих в стандарт C++11 (ISO/IEC 14882:2011), кроме поддержки ускорителей Intel Xeon Phi и некоторых дополнительных возможностей используемых компиляторов и архитектур, при их наличии.

В результате, приложение использует поддержку потоков, механизмы синхронизации данных, атомарные переменные, некоторые контейнеры и т.д. из стандартной библиотеки C++, а для обеспечения работы и оптимизации производительности для конкретных архитектур были реализованы различные системно-независимые и переносимые компоненты: для обработки ошибок и исключений, подсчета точных временных интервалов, перехвата команд прерывания и управления, работы с различными локализациями, потоками ввода-вывода, консолью оператора и т.д.

При программной оптимизации для работы в различных системах, в каждом конкретном случае была проведена профилировка приложения, с целью исследования производительности различных участков кода и оценки их влияния на итоговое время вычислений, а также оценивалось влияние различных режимов и настроек компилятора на качество получаемого машинного кода и общую скорость работы приложения. Так, включение межпроцедурной оптимизации (англ. Interprocedural Optimization, IPO) совместно с оптимизацией во время компоновки (англ. Link Time Optimization, LTO) для архитектуры Intel 64 позволило уменьшить общее время вычислений на 10-15% при прочих равных условиях. Кроме того, на производительность существенно влияет организация и выравнивание данных в памяти, минимизация использования системных вызовов, функций выделения памяти, некоторых средств языка, таких как исключения C++ и т.д.

Оптимизация многопоточного выполнения. Для повышения производительности параллельного вычислительного приложения с использованием многопоточной модели выполнения, были реализованы специализированные структуры данных и контейнеры с низкоуровневой оптимизацией наиболее существенных операций, гарантирующие необходимое выравнивание элементов и минимизацию количества операций по выделению и перераспределению выделенных блоков памяти, а также позволяющие упростить процесс копирование данных между основной системой и сопроцессорами Intel Xeon Phi. Использование оптимизированных структур данных позволило свести к минимуму накладные расходы при создании потока для обработки следующего блока входных данных и исключить большую часть обращений к функциям выделения памяти в процессе его дальнейшей работы.

Используемый вычислительный алгоритм характеризуется длительным временем обработки для небольшого подмножества входных данных (ЛК), что определяется различной размерностью решаемых подзадач, линейно зависящей от числа трансверсалей в ЛК [15] (рис. 1).

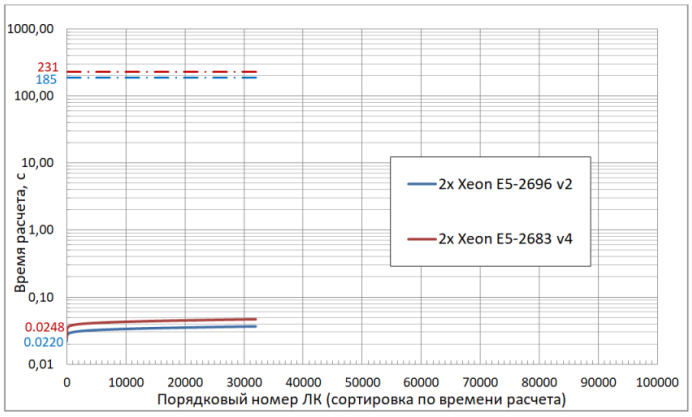

Рис. 1. Зависимость времени обработки от входных данных.

Из приведенного графика видно, что время обработки каждого входного ЛК линейно зависит от производительности одного потока используемого вычислителя – для данной задачи однопоточная производительность Intel Xeon E5-2696 v2 выше, чем Intel Xeon E5-2683 v4, даже несмотря на более старую микроархитектуру, так как у него выше тактовая частота.

В целях уменьшения влияния времени обработки отдельных элементов входных данных на общее время расчета, было использовано динамическое управление размером блока данных, передаваемого для обработки каждому новому вычислительному потоку. Каждый блок представляет собой последовательность исходных ЛК – очередь заданий, последовательно обрабатываемую вычислительным потоком. При небольшом размере блока, ЛК с длительным временем обработки будут более эффективно распределяться между вычислительными потоками, однако значительно возрастают накладные расходы на создание и завершение потока, синхронизацию данных и т.д. При увеличении размера блока, накладные расходы сокращаются, однако ЛК, требующие длительной обработки, неравномерно распределяются по длинным очередям заданий, что приводит в конце процесса вычислений к неравномерному распределению заданий между вычислительными потоками, длительному времени досчитывания для небольшого их числа (при простаивании остальных) и в конечном итоге к значительному увеличению общего времени вычислений.

При динамическом выборе длины очереди заданий для потока, в процессе вычислений производится оценка времени обработки каждого входного ЛК данным конкретным вычислителем и изменение длины очереди в зависимости от его продолжительности, а также среднего времени расчета для исходных ЛК за анализируемый период времени.

В результате применения предложенного метода удалось уменьшить среднее время вычислений каждого отдельного блока входных данных и сделать суммарное время расчета для различных вычислителей более предсказуемым.

Оценка однопоточной производительности различных вычислителей. В качестве основной целевой системы была выбрана серверная платформа Intel, представленная в нескольких различных конфигурациях: с ЦП Intel Xeon различных поколений и установленными сопроцессорами Intel Xeon Phi x100 нескольких типов.

Сопроцессор Intel Xeon Phi x100 поставляется в виде платы расширения PCIe x16 и представляет собой многоядерное вычислительное устройство, совместимое с системой команд Intel 64, а также 8 или более Гбайт локальной оперативной памяти (ОЗУ) стандарта GDDR5. Он содержит не менее 50 объединенных двунаправленной кольцевой шиной вычислительных ядер, обычно – 60 или 61, в зависимости от модели ускорителя [13].

Архитектура сопроцессора Intel Xeon Phi x100 поддерживает систему команд Intel 64, но отличается от большинства современных ЦП отсутствием динамического исполнения инструкций, она не поддерживает спекулятивное выполнение команд, предсказание ветвлений, анализ потока данных и т.д., т.е. используется только последовательное исполнение инструкций (англ. in-order execution) [16].

Тем не менее использование SMT (одновременная многопоточность, англ. Simultaneous Multithreading) с 4 потоками на ядро, дает 240 и более аппаратных потоков на один ускоритель и в определенной степени позволяет устранить задержки, связанные с устаревшей архитектурой с последовательным исполнением инструкций. Для обеспечения возможности дополнительной оптимизации производительности также предоставляется специальный набор SIMD (одиночный поток команд, множественный поток данных, ОКМД, англ. single instruction, multiple data) инструкций для работы с 512 битными векторами (KNC SIMD).

Исходя из заявленных характеристик, можно заметить, что однопоточная производительность ускорителя относительно невысока и практически на порядок уступает производительности современных ЦП, особенно при невозможности использования SIMD (табл. 1).

Таблица 1

Время обработки отобранного ЛК для различных вычислителей

Тип вычислителя	Время обработки, с
Intel Xeon E5-2696 v2	111.6
Intel Xeon E5-2696 v2 (Speculative Store Bypass Disable)	249.3
Intel Xeon E5-2683 v4	170.8
Intel Xeon Phi 5110P	1180.0
Intel Xeon Phi 7120P	1015.7

При этом низкая однопоточная производительность сопроцессора компенсируется большим количеством аппаратных вычислительных потоков, а также полноценным набором инструкций, что выгодно отличает Intel Xeon Phi от различных GPGPU (англ. General-Purpose computing on Graphics Processing Units, неспециализированные вычисления на графических процессорах) [17, 18] и прочих специали-

зированных вычислителей. Это особенно важно и удобно для решения подобных данной комбинаторных задач, содержащих большое количество условных ветвлений [19], вызовы подпрограмм, нерегулярные паттерны обращения к памяти и т.п.

Эффективное использование сопроцессоров Intel Xeon Phi. Разработанное приложение поддерживает режим автоматической выгрузки (англ. Offload) на все установленные в системе ускорители Intel Xeon Phi. При этом можно управлять количеством используемых ускорителей, вычислительных потоков, размером обрабатываемого блока данных – очереди заданий устройства и т.д.

Максимальный размер обрабатываемого блока данных (очереди заданий) для ускорителя вычисляется по формуле:

$$S_i = m \cdot N_i,$$

где i – порядковый номер сопроцессора, S выражается в количестве латинских квадратов (ЛК) во входном блоке данных, m – заданный коэффициент размера очереди выгрузки, по умолчанию равный 40, а N_i – количество используемых вычислительных потоков. По умолчанию, N_i берется равным количеству потоков, возвращаемому операционным окружением данного устройства минус 4, так как операционная среда ускорителя Intel Manycore Platform Software Stack (Intel MPSS) резервирует одно ядро (4 потока) для использования системой. Исходная оптимизированная версия гетерогенного параллельного вычислительного приложения продемонстрировала значительную девиацию общего времени расчета в зависимости от входных данных и используемого коэффициента m (таблица 2, рис. 2).

Таблица 2

Зависимость общего времени расчета (с) от коэффициента m без балансировки

Коэф. m	1x Xeon Phi 7120P	1x Xeon Phi 7120P и 2x Xeon E5-2683 v4	2x Xeon Phi 7120P	2x Xeon Phi 7120P и 2x Xeon E5-2683 v42	2x Xeon E5-2683 v4
5	116.0	32.1	61.0	34.8	40.9
10	96.5	35.3	51.6	24.2	
15	92.8	35.3	49.3	30.4	
20	91.7	35.8	49.8	25.8	
25	90.4	33.6	49.3	22.2	
30	90.2	36.6	52.7	26.3	
35	89.7	29.2	47.3	30.0	
40	89.5	32.9	49.0	33.2	
45	88.7	37.1	52.3	32.8	
50	88.6	41.1	48.4	27.1	
55	90.2	40.7	45.8	23.9	
60	91.4	37.0	48.8	25.2	
65	88.5	34.2	52.7	26.3	
70	88.7	29.0	59.8	29.0	
75	88.2	30.4	58.1	30.3	
80	88.3	32.2	56.0	32.3	
85	88.1	34.3	54.0	34.2	
90	88.0	36.2	52.0	36.2	
95	88.2	38.2	50.1	38.3	
100	88.0	40.1	48.0	40.0	
Лучшее	88.0	29.0	45.8	22.2	40.9

Коэф. *m*	1x Xeon Phi 5110P	1x Xeon Phi 5110P и 2x Xeon E5-2696 v2	2x Xeon Phi 5110P	2x Xeon Phi 5110P и 2x Xeon E5-2696 v2	2x Xeon E5-2696 v2
5	137.0	32.0	71.1	27.3	39.5
10	113.3	31.9	58.5	31.1	
15	110.7	31.3	57.8	23.6	
20	109.7	30.1	59.7	30.0	
25	108.5	37.1	59.5	25.1	
30	108.2	29.5	58.1	30.8	
35	106.9	33.9	56.8	34.0	
40	106.7	38.9	57.7	38.5	
45	105.8	43.4	63.4	30.4	
50	106.2	45.2	58.5	24.0	
55	107.6	36.2	55.4	28.2	
60	106.7	29.8	57.1	29.6	
65	105.6	30.8	61.7	30.9	
70	105.4	33.2	66.4	33.2	
75	105.2	35.5	69.7	35.5	
80	105.2	37.9	67.4	37.8	
85	105.0	40.1	64.9	40.1	
90	105.1	42.4	62.7	42.4	
95	104.9	44.7	60.4	44.8	
100	105.3	47.1	58.1	47.4	
Лучшее	104.9	29.5	55.4	23.6	39.5

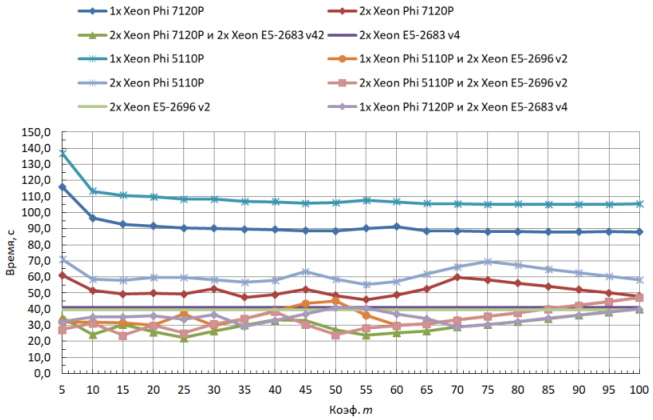

Рис. 2. Время расчета без балансировки.

Итоговое время зависит от общего количества данных в последнем блоке выгрузки, наличия в нем ЛК, требующих длительной обработки, а также от различного размера блоков данных для нескольких ускорителей в составе гетерогенного вычислительно узла. Данная зависимость еще больше усугубляется тем, что время обработки одного ЛК сопроцессором почти на порядок больше времени обработки в случае использования ЦП (табл. 1).

Для минимизации этой зависимости был предложен метод балансировки нагрузки между различными вычислителями – сопроцессорами и ЦП. Суть данного метода заключается в изменении размера очереди заданий для ускорителя и осуществляется следующим образом:

♦ На начальном этапе, при отсутствии статистики, производительность сопроцессора считается пропорциональной максимальному размеру блока данного устройства S_i и зависит от количества поддерживаемых вычислительных потоков.

♦ В процессе вычислений постоянно оценивается общая средняя производительность вычислителя для каждого сопроцессора и ЦП.

♦ Размер очереди заданий для ускорителя изменяется в зависимости от оставшихся необработанных ЛК пропорционально полученным ранее данным о его производительности.

♦ Для предотвращения уменьшения размера блока менее разумного предела, что становится возможным при использовании данного алгоритма и запрете вычислений на центральном процессоре системы (при расчете на двух и более сопроцессорах), размер блока ограничен снизу количеством вычислительных потоков сопроцессора N_i.

♦ Новый размер блока вычисляется для каждой последующей выгрузки, начиная с того момента, когда количество оставшихся для обработки ЛК станет не более, чем прогнозируемая суммарная производительность гетерогенного вычислительного комплекса.

При использования предложенного метода балансировки были получены результаты общего времени расчета, в значительной мере менее зависимые от значения коэффициента m (табл. 3, рис. 3).

Таблица 3

Общее времени расчета (с) с использованием балансировки

Коэф. m	1x Xeon Phi 7120P	1x Xeon Phi 7120P и 2x Xeon E5-2683 v4	2x Xeon Phi 7120P	2x Xeon Phi 7120P и 2x Xeon E5-2683 v42	2x Xeon E5-2683 v4
5	115.6	31.3	61.0	30.1	40.9
10	95.2	29.4	51.3	24.5	
20	91.7	28.8	49.0	22.4	
30	90.9	28.6	48.6	24.3	
40	89.6	28.3	48.1	23.1	
50	88.8	28.3	47.6	21.7	
60	89.4	28.6	48.0	25.1	
70	88.8	28.9	47.8	28.0	
80	88.2	32.3	47.5	28.1	
90	88.1	36.1	48.4	28.0	
100	88.1	40.1	47.5	28.1	
Лучшее	88.1	28.3	47.5	21.7	40.9
Коэф. m	1x Xeon Phi 5110P	1x Xeon Phi 5110P и 2x Xeon E5-2696 v2	2x Xeon Phi 5110P	2x Xeon Phi 5110P и 2x Xeon E5-2696 v2	2x Xeon E5-2696 v2
5	136.5	31.0	72.1	27.0	39.5
10	113.3	29.6	60.0	25.3	
20	109.7	29.2	58.9	23.4	
30	108.2	29.2	57.7	26.8	
40	106.6	29.2	57.1	24.7	
50	106.1	29.2	56.9	23.9	
60	106.5	29.7	56.9	29.7	
70	105.4	33.2	56.7	33.1	
80	105.2	37.8	57.1	33.4	
90	104.9	42.4	56.9	33.4	
100	105.3	47.0	56.9	33.5	
Лучшее	104.9	29.2	56.7	23.4	39.5

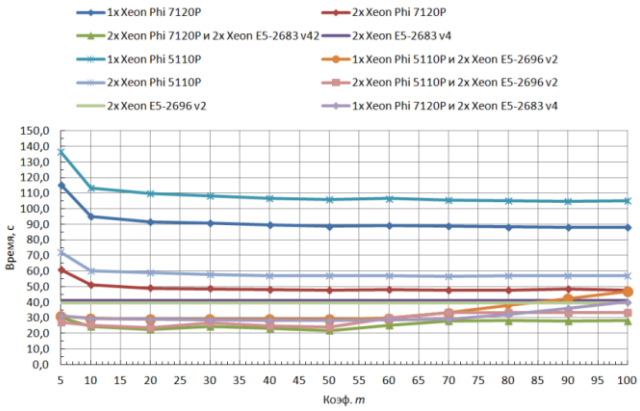

Рис. 3. Время расчета с использованием балансировки

При малых значениях коэффициента *m* становятся особенно заметны накладные расходы на выгрузку данных на сопроцессоры.

Из-за относительно небольшого общего объема входных данных (50000 ЛК), при использовании нескольких вычислителей и больших значений коэффициента *m* алгоритм балансировки не успевает накопить статистику производительности и рассчитывает длину очереди ускорителя исходя из количества вычислительных потоков, что дает отклонение значений времени расчета при сохранении общей линейности.

Заключение. Для данного приложения суммарная производительность гетерогенного вычислительного комплекса практически линейно зависит от производительности входящих в него вычислителей и их количества (табл. 2 и 3). Несмотря на то что однопоточная производительность Intel Xeon Phi практически на порядок уступает производительности современных ЦП (табл. 1), за счет предоставления большого количества вычислительных потоков средняя суммарная производительность ускорителей вполне сопоставима с ними. Нормированные с учетом количества предоставляемых вычислительных потоков средние значения производительности для используемых в работе вычислителей приведены в табл. 4.

Таблица 4

Оценка производительности различных вычислителей

Конфигурация комплекса	Среднее время обработки ЛК, с	Производительность, ЛК/мин
1x Xeon Phi 7120P	0.00160	37419
2x Xeon Phi 7120P	0.00080	74865
2x Xeon E5-2683 v4	0.00079	76215
2x Xeon Phi 7120P и 2x Xeon E5-2683 v4	0.00040	151080
1x Xeon Phi 5110P	0.00192	31191
2x Xeon Phi 5110P	0.00098	61155
2x Xeon E5-2696 v2	0.00077	78171
2x Xeon Phi 5110P и 2x Xeon E5-2696 v2	0.00043	139326

Для достижения максимальной производительности при развертывании гетерогенного вычислительного комплекса отдельное внимание следует уделить оптимальной настройке вычислительной системы для использования в сфере высокопро-

изводительных вычислений. Необходимо провести предварительную настройку системы путем изменения настроек в программе конфигурации UEFI (единый расширяемый интерфейс прошивки, англ. Unified Extensible Firmware Interface) или BIOS (базовая система ввода-вывода, англ. Basic Input/Output System) с целью отключения функций энергосбережения, задания параметров энергопотребления, выбора режима максимальной производительности, а также тюнинга различных настроек, связанных с конфигурацией памяти, процессора, настроек NUMA (архитектура с неравномерной памятью, англ. Non-Uniform Memory Access/Architecture), шины PCIe (особенно при использовании сопроцессоров), виртуализации и т.д.

Особое внимание следует уделить настройке используемой ОС – при необходимости произвести отключение функций энергосбережения системы и отдельных устройств, выбор режима кэширования для накопителей, отключение фоновых служб и приложений, настройку подсистемы управления памятью и планировщика задач и т.д.

Также следует очень внимательно отнестись к вопросу о необходимости применения различных методов борьбы с ошибками утечки информации по побочным каналам, проблема использования которых для большинства современных процессоров начала широко изучаться с начала 2018 года [20] и продолжает преследовать нас в виде все новых и новых открывающихся возможностей для осуществления данного рода атак. Проблема некоторых из используемых для предотвращения подобных атак исправлений для ОС заключается в серьезном, а иногда просто катастрофическом падении производительности для отдельных операций или системы в целом. Например, во второй строке (табл. 1) приведены данные производительности системы при глобальном включении опции SSBD (отключение спекулятивного пропуска сохранения, англ. Speculative Store Bypass Disable) – в итоге падение производительности для данного вычислительного приложения (и множества других) составляет более чем два раза.

Работа выполнена при финансовой поддержке РФФИ гранты №18-29-03264 и 19-07-00802, внутриуниверситетского гранта по программе развития ЮЗГУ "Приоритет – 2030" № ПР2030/2021.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Anderson D.P.* BOINC: A platform for volunteer computing // Journal of Grid Computing. – 2020. – Vol. 18, No. 1. – P. 99-122.
2. *Wang L., Jie W., Chen J.* Grid computing: infrastructure, service, and applications. – CRC Press, 2018.
3. *Braun T.D. et al.* A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems // Journal of Parallel and Distributed computing. – 2001. – Vol. 61, No. 6. – P. 810-837.
4. *Cirne W. et al*. Grid computing for bag of tasks applications // In Proc. of the 3rd IFIP Conference on E-Commerce, E-Business and EGovernment. – 2003.
5. *Posypkin M., Semenov A., Zaikin O.* Using BOINC desktop grid to solve large scale SAT problems // Computer Science. – 2012. – Vol. 13, No. 1. – P. 25.
6. *Yang C.T. et al.* Performance benchmarking of deep learning framework on Intel Xeon Phi // The Journal of Supercomputing. – 2021. – Vol. 77, No. 3. – P. 2486-2510.
7. *Jennett C. et al.* Motivations, learning and creativity in online citizen science // Journal of Science Communication. – 2016. – Vol. 15, No 3.
8. *Foster I., Kesselman C. (ed.).* The Grid 2: Blueprint for a new computing infrastructure. – Elsevier, 2003.
9. *Amalarethinam D.I.G., Josphin A.M.* Dynamic task scheduling methods in heterogeneous systems: a survey // International Journal of Computer Applications. – 2015. – Vol. 110, No. 6.
10. *Choi S.J. et al.* Volunteer availability based fault tolerant scheduling mechanism in desktop grid computing environment // Third IEEE International Symposium on Network Computing and Applications, 2004.(NCA 2004). Proceedings. – IEEE, 2004. – P. 366-371.

11. *Vatutin E., Nikitina N., Belyshev A., Manzyuk M.* On polynomial reduction of problems based on diagonal Latin squares to the exact cover problem // ICCS-DE. – 2020. – Vol. 2638. – P. 289-297. – DOI: 10.47350/ICCS-DE.2020.26.
12. *Brown J.W., Cherry F., Most L., Most M., Parker E.T., Wallis W.D.* Completion of the spectrum of orthogonal diagonal Latin squares // Lecture notes in pure and applied mathematics. – 1992. – Vol. 139. – P. 43-49. – DOI: 10.1201/9780203719916.
13. Intel Xeon Phi Coprocessor System Software Developers Guide. – Intel Corporation, 2014. – P. 164.
14. *Альбертьян А.М., Курочкин И.И.* Использование сопроцессоров Intel Xeon Phi в грид-системах из персональных компьютеров // CEUR-Proceedings: Selected Papers of the II Intern. Sci. Conf." Convergent Cognitive Information Technologies", Moscow, Russia. – 2017. – Vol. 2064. – P. 196-201.
15. *Vatutin E., Belyshev A., Nikitina N., Manzuk M.* Evaluation of Efficiency of Using Simple Transformations When Searching for Orthogonal Diagonal Latin Squares of Order 10 // Communications in Computer and Information Science. – Vol. 1304. – Springer, 2020. – P. 127-146. – DOI: 10.1007/978-3-030-66895-2_9.
16. *James Jeffers, James Reinders* Intel Xeon Phi Processor High Performance Programming. – Morgan Kaufmann, 2013. – 432 p. – ISBN: 978-0-12-410414-3.
17. *De Ravé E.G. et al.* Using general-purpose computing on graphics processing units (GPGPU) to accelerate the ordinary kriging algorithm // Computers & Geosciences. – 2014. – Vol. 64. – P. 1-6.
18. *Nobile M.S. et al.* Graphics processing units in bioinformatics, computational biology and systems biology // Briefings in bioinformatics. – 2017. – Vol. 18, No. 5. – P. 870-885.
19. *Morrison D.R. et al.* Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning // Discrete Optimization. – 2016. – Vol. 19. – P. 79-102.
20. *Hill M.D. et al.* On the Spectre and Meltdown processor security vulnerabilities // IEEE Micro. – 2019. – Vol. 39, No. 2. – P. 9-19.

REFERENCES

1. *Anderson D.P.* BOINC: A platform for volunteer computing, *Journal of Grid Computing*, 2020, Vol. 18, No. 1, pp. 99-122.
2. *Wang L., Jie W., Chen J.* Grid computing: infrastructure, service, and applications. CRC Press, 2018.
3. *Braun T.D. et al.* A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, *Journal of Parallel and Distributed computing*, 2001, Vol. 61, No. 6, pp. 810-837.
4. *Cirne W. et al*. Grid computing for bag of tasks applications, *In Proc. of the 3rd IFIP Conference on E-Commerce, E-Business and EGovernment*, 2003.
5. *Posypkin M., Semenov A., Zaikin O.* Using BOINC desktop grid to solve large scale SAT problems, *Computer Science*, 2012, Vol. 13, No. 1, pp. 25.
6. *Yang C.T. et al.* Performance benchmarking of deep learning framework on Intel Xeon Phi, *The Journal of Supercomputing*, 2021, Vol. 77, No. 3, pp. 2486-2510.
7. *Jennett C. et al.* Motivations, learning and creativity in online citizen science, *Journal of Science Communication*, 2016, Vol. 15, No 3.
8. *Foster I., Kesselman C. (ed.).* The Grid 2: Blueprint for a new computing infrastructure. Elsevier, 2003.
9. *Amalarethinam D.I.G., Josphin A.M.* Dynamic task scheduling methods in heterogeneous systems: a survey, *International Journal of Computer Applications*, 2015, Vol. 110, No. 6.
10. *Choi S.J. et al.* Volunteer availability based fault tolerant scheduling mechanism in desktop grid computing environment, *Third IEEE International Symposium on Network Computing and Applications, 2004.(NCA 2004). Proceedings*. IEEE, 2004, pp. 366-371.
11. *Vatutin E., Nikitina N., Belyshev A., Manzyuk M.* On polynomial reduction of problems based on diagonal Latin squares to the exact cover problem, *ICCS-DE*, 2020, Vol. 2638, pp. 289-297. DOI: 10.47350/ICCS-DE.2020.26.
12. *Brown J.W., Cherry F., Most L., Most M., Parker E.T., Wallis W.D.* Completion of the spectrum of orthogonal diagonal Latin squares, *Lecture notes in pure and applied mathematics*, 1992, Vol. 139, pp. 43-49. DOI: 10.1201/9780203719916.

13. Intel Xeon Phi Coprocessor System Software Developers Guide. Intel Corporation, 2014, 164 p.
14. *Al'bert'yan A.M., Kurochkin I.I.* Ispol'zovanie soprotsessorov Intel Xeon Phi v grid-sistemakh iz personal'nykh komp'yuterov [The use of Intel Xeon Phi coprocessors in grid systems from personal computers], *CEUR-Proceedings: Selected Papers of the II Intern. Sci. Conf." Convergent Cognitive Information Technologies", Moscow, Russia*, 2017, Vol. 2064, pp. 196-201.
15. *Vatutin E., Belyshev A., Nikitina N., Manzuk M.* Evaluation of Efficiency of Using Simple Transformations When Searching for Orthogonal Diagonal Latin Squares of Order 10, *Communications in Computer and Information Science*, Vol. 1304. Springer, 2020, pp. 127-146. DOI: 10.1007/978-3-030-66895-2_9.
16. *James Jeffers, James Reinders* Intel Xeon Phi Processor High Performance Programming. Morgan Kaufmann, 2013, 432 p. ISBN: 978-0-12-410414-3.
17. *De Ravé E.G. et al.* Using general-purpose computing on graphics processing units (GPGPU) to accelerate the ordinary kriging algorithm, *Computers & Geosciences*, 2014, Vol. 64, pp. 1-6.
18. *Nobile M.S. et al.* Graphics processing units in bioinformatics, computational biology and systems biology, *Briefings in bioinformatics*, 2017, Vol. 18, No. 5, pp. 870-885.
19. *Morrison D.R. et al.* Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning, *Discrete Optimization*, 2016, Vol. 19, pp. 79-102.
20. *Hill M.D. et al.* On the Spectre and Meltdown processor security vulnerabilities, *IEEE Micro*, 2019, Vol. 39, No. 2, pp. 9-19.

Статью рекомендовал к опубликованию д.т.н., профессор А.И. Баранчиков.

Альбертьян Александр Михайлович – Федеральный Исследовательский Центр «Информатика и управление» РАН; e-mail: admin@isa.ru; Москва, Россия; УИТС; ведущий инженер.

Курочкин Илья Ильич – Институт проблем передачи информации им. А.А. Харкевича РАН; e-mail: kurochkin@iitp.ru; Москва, Россия; тел.: +74956504225; Центр распределенных вычислений, лаборатория Ц-1; зав. лабораторией; к.т.н.

Ватутин Эдуард Игоревич – Юго-Западный государственный университет; e-mail: evatutin@rambler.ru; г. Курск, Россия; тел.: +74712222670; кафедра вычислительной техники; к.т.н.; доцент.

Albertian Alexander Mikhaylovich – Federal Research Center "Computer Science and Control", Russian Academy of Sciences; e-mail: admin@isa.ru; Moscow, Russia; IT department.

Kurochkin Ilya Il'ich – Institute for Information Transmission Problems, Russian Academy of Sciences; e-mail: kurochkin@iitp.ru; Moscow, Russia; phone: +74956504225; Distributed computing center, Lab-C1, cand. of eng. sc.

Vatutin Eduard Igorevich – Southwest State University; e-mail: evatutin@rambler.ru; Kursk, Russia; phone: 84712222670; the department of computer science; cand. of eng. sc.; associate professor.

УДК 004.056 DOI 10.18522/2311-3103-2021-7-153-167

Л.К. Бабенко, А.С. Кириллов

РАЗРАБОТКА АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ ОБНАРУЖЕНИЯ ВРЕДОНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

При проведении исследований в области обнаружения вредоносного программного обеспечения основной фокус делается именно на методах, игнорируя то, как эти методы практически могли бы быть реализованы. С другой стороны, есть работы, которые раскрывают некоторые технические подробности реализации или оптимизации процесса анализа исследуемого образца и сбора данных о его работе. Однако, необходимо соединять результаты концепций экспериментальных систем и те возможности реализации, кото-