УДК 004.4'242, 004.4'422, 004.432.4

МЕТОД БАЛАНСИРОВКИ ВЫЧИСЛИТЕЛЬНОЙ НАГРУЗКИ ДЛЯ ГИБРИДНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Т. П. Баранова[1] [0000-0002-2003-3737], А. Б. Бугеря[2] [0000-0002-9698-458X], К. Н. Ефимкин[3] [0000-0002-1087-7645]

[1-3]*Институт прикладной математики им. М.В. Келдыша Российской академии наук, г. Москва*

[1]bart1950@yandex.ru, [2]shurabug@yandex.ru, [3]bigcrocodile@yandex.ru

Аннотация

Рассмотрены вопросы распределения вычислений внутри одного узла гибридной вычислительной системы для прикладных программ расчётного характера. Предложены метод статического распределения вычислений, а также метод автоматической балансировки вычислительной нагрузки в процессе выполнения программы. Метод автоматической балансировки основан на периодическом анализе величины загрузки центрального процессора выполняемой программой и принятии решения о перераспределении вычислительной нагрузки в случае необходимости. Приведённые методы реализованы в прикладной программе, решающей задачу из области газодинамики с использованием вычислительных ресурсов многоядерного центрального процессора и графических ускорителей. Получены и проанализированы результаты выполнения программы с различными распределениями данных, как с включённым механизмом автоматической балансировки вычислительной нагрузки, так и без него.

Ключевые слова*: параллельное программирование, автоматизация программирования, балансировка вычислительной нагрузки, гибридные архитектуры, язык НОРМА, автоматическая генерация программ*

ВВЕДЕНИЕ

В современном мире существует огромное количество различных задач, решение которых требует наличия значительных вычислительных мощностей. Такие задачи имеются во всех областях науки и промышленности, в бизнесе и даже в сфере индивидуального применения. Типичными примерами таких ре-

сурсоёмких задач могут служить решение задач математической физики численными методами (например, моделирование процессов, происходящих в ядерном реакторе), моделирование физических, химических и биологических процессов. Постоянно появляются новые задачи подобного рода. Современные вычислительные системы, на которых решаются подобные задачи, предоставляют возможность параллельных вычислений. Поэтому современная эффективная программа обязана быть параллельной.

Существуют различные методы автоматизации разработки параллельных программ. Отметим лишь монографии [1, 2], в которых строго сформулированы математические основы совместного изучения параллельных численных методов и параллельных вычислительных систем и исследована задача отображения программы на архитектуру параллельного компьютера. В частности, показано, что автоматическое отображение уже написанной последовательной программы на параллельную вычислительную систему, в общей постановке, является NP-полной задачей. Это и объясняет отсутствие, до настоящего времени, удовлетворительного с практической точки зрения универсального метода разработки параллельных программ.

Теоретические сложности, возникающие в области разработки параллельных программ, усугубляются постоянным развитием и усложнением архитектур вычислительных систем. Эти возможности, с одной стороны, дают новый потенциал ускорения вычислений, с другой стороны, ставят проблему утилизации этого потенциала, разработки методов и средств программирования в условиях этих новых возможностей.

Помимо центральных процессоров общего назначения (CPU) современные вычислительные системы, как правило, содержат дополнительные вычислители, предназначенные для быстрого и энергоэффективного параллельного выполнения массовых вычислительных операций, одинаковых для большого объёма обрабатываемых данных. Примерами таких вычислителей могут служить графические процессоры (GPU) и ускорители Xeon Phi. Для своего эффективного выполнения параллельная программа должна обеспечить все имеющиеся в её распоряжении вычислители непрерывной загрузкой данными для вычислений. Также она должна обеспечить синхронизацию вычислений, где необходимо, при об-

ращении к общим данным, минимизировать простои вычислителей при синхронизации и доступе к прочим ресурсам, как программным, так и аппаратным. В случае, если некоторые вычислители обрабатывают выделенный им объём данных быстрее других и затем простаивают в ожидании синхронизации, возникает потребность перераспределить обрабатываемые данные между вычислителями в процессе выполнения программы.

Решение задачи распределения данных между вычислителями называется балансировкой вычислительной нагрузки (computational load balancing), а в случае периодического решения её в процессе выполнения программы – динамической балансировкой вычислительной нагрузки. От успешного решения этой задачи в значительной степени зависит и эффективность выполнения всей программы в целом.

Исследования в области создания как методов программирования для новых архитектур, так и реализации этих методов в языковых средствах параллельного программирования, ведутся весьма активно и поддерживаются фирмами-производителями вычислительных систем. Достаточно полная классификация архитектур, методов и средств параллельного программирования представлена на сайте [3], посвященного, в частности, технологиям параллельных вычислений. Из уже реализованных подходов перспективными являются, по нашему мнению, подходы, базирующиеся на вполне разумном симбиозе распараллеливающего компилятора и подсказок со стороны программиста, выполненных в виде специальных программных директив, например, [4, 5].

Интерес также представляет непроцедурный подход к построению параллельных программ [6], в котором точно определяются границы того, что и как можно автоматически распараллелить, и который автоматизирует процесс построения эффективных параллельных программ. Этот подход использует в качестве языка программирования непроцедурный язык НОРМА [7].

В данной статье предложен метод организации автоматической динамической балансировкой вычислительной нагрузки в пределах одного узла гибридной вычислительной системы, в составе которого есть один или более CPU и один или более дополнительных вычислителей. Вопрос распределения вычислительной нагрузки между узлами вычислительной системы остаётся за рам-

ками данной работы. Описываемый метод был разработан для применения в системе программирования НОРМА [6] при трансляции программ для вычислительных систем с графическими ускорителями (GPU). Следует отметить, что метод универсален, не зависит от типа конкретной гибридной вычислительной системы и применяемых средств программирования и, как кажется авторам, с успехом может быть применён при создании параллельных программ вычислительного характера, как при ручном программировании, так и в случае автоматизированного подхода.

Предложенный метод был успешно опробован на программе вычислительного характера из области газовой динамики, на параллельных системах с гибридной архитектурой (графические процессоры).

СТАТИЧЕСКОЕ РАСПРЕДЕЛЕНИЕ ВЫЧИСЛИТЕЛЬНОЙ НАГРУЗКИ

Рассмотрим вопрос распределения вычислительной нагрузки в пределах одного узла гибридной вычислительной системы. В каждом таком узле имеются один или более центральный процессор, CPU. Так как все современные CPU многоядерные и имеют доступ ко всей оперативной памяти узла, то неважно, сколько их в узле – вся их совокупность всегда рассматривается прикладной программой и операционной системой как единый многоядерный процессор. Также в узле гибридной вычислительной системы имеются один или более специальный вычислитель (ускоритель, GPU или Xeon Phi, или, возможно, какие-то другие). Их количество уже имеет важное значение, так как, как правило, каждый такой вычислитель имеет доступ и может обрабатывать данные только из собственной памяти.

Эффективная параллельная программа должна использовать все имеющиеся вычислительные мощности. Соответственно в узле гибридной вычислительной системы весь объём производимых вычислений должен быть как-то распределён между CPU и ускорителями. Вычисления, выполняемые на CPU, должны производиться с использованием технологий для многопоточного программирования, например, OpenMP. Вычисления, выполняемые на ускорителе, должны производиться с использованием доступной для данного типа ускорителей технологии, например, NVIDIA CUDA для GPU фирмы NVIDIA.

Процесс автоматического статического распределения вычислений между CPU и GPU при трансляции программ, написанных на языке НОРМА, подробно описан в работах [8, 9]. Методы и идеи, изложенные в них, могут быть применены к любой параллельной программе и любому другому типу ускорителей. Кратко, эти методы заключаются в следующем.

В языке НОРМА оператор, описывающий какие-то вычисления, может производиться над областью. Область – аналог математического понятия сетки. Таким образом, оператор описывает набор одинаковых вычислений, производимых в точках области (узлах сетки) произвольного типа. Как правило, вычисления в каждой точке области не зависят от значений в других точках, получаемых в этом же операторе на этом же шаге итерации. Тогда вычисления, производимые одним оператором, в каждой точке области не зависимы друг от друга и могут быть выполнены параллельно.

Для распределения таких вычислений между CPU и ускорителями предлагается использовать такой подход, что каждый такой оператор выполняется и на CPU, и на каждом ускорителе, имеющемся в системе. При этом вся область такого оператора (точнее, массив точек, эту область составляющих) распределяется между вычислителями, и каждый вычислитель выполняет оператор только для тех точек области, которые были ему распределены. Для того чтобы вычислитель мог выполнить оператор для своих точек, также необходимо, чтобы переменные-массивы, определённые в этих точках области, как требуемые для вычислений, так и те, которые вычисляются в результате оператора, также были физически распределены между памятью CPU и памятью каждого ускорителя. То есть, фактически, процесс распределения вычислений сводится к процессу распределения данных и затем к синхронному выполнению вычислений каждым вычислителем над предназначенной ему частью общих данных.

Для распределения таких переменных предлагается следующий способ. Вначале весь объём обрабатываемых данных делится на две неравные части: зона, обрабатываемая CPU, и зона, обрабатываемая ускорителем (ускорителями). Размеры зон выбираются в соответствии с предполагаемым соотношением производительности CPU и суммарной производительностью ускорителей. Затем, если ускорителей несколько, их зона делится на соответствующее число подзон и рас-

пределяется между ними поровну. На рис. 1 приведён пример такого распределения.

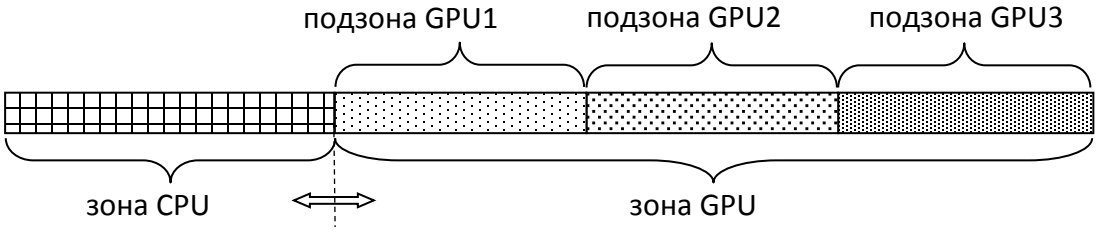

Рисунок 1. Распределение обрабатываемых данных между зонами.

Каждый вычислитель производит вычисления над теми данными, которые попали в его зону (подзону). При этом дополнительно решается задача передачи данных между зонами и подзонами, если в этом есть необходимость.

ДИНАМИЧЕСКОЕ РАСПРЕДЕЛЕНИЕ ВЫЧИСЛИТЕЛЬНОЙ НАГРУЗКИ

Граница распределения вычислений между зоной CPU и зоной ускорителей может быть фиксированной, а может быть изменяемой в процессе выполнения программы. С помощью периодического изменения границы зон в процессе выполнения программы осуществляется динамическая балансировка вычислительной нагрузки, о которой пойдёт речь ниже. При изменении размера зоны GPU соответственно пересчитываются размеры подзон каждого ускорителя.

Чтобы оценить необходимость внести корректировку в положение границы зон, периодически (например, на каждом *n*-ом шаге итерации) запускается процесс определения общей эффективности вычислений при текущем распределении вычислительной нагрузки. В результате этого процесса принимается решение о сохранении текущего положения границы зон или о сдвиге его на определённую величину в ту или иную сторону. В случае сдвига необходимо организовать перераспределение данных, которые в результате сдвига начинают обрабатываться другим вычислителем.

МЕТОД ОПРЕДЕЛЕНИЯ НЕОБХОДИМОСТИ КОРРЕКТИРОВКИ РАСПРЕДЕЛЕНИЯ ВЫЧИСЛИТЕЛЬНОЙ НАГРУЗКИ

Для определения общей эффективности вычислений предлагается использовать следующий метод, основанный на оценке использования программой ресурса CPU. При разработке этого метода мы исходим из предположения, что программа, решающая задачу вычислительного характера, должна постоянно заниматься вычислениями и полностью потреблять ресурс вычислителей. Конечно, в ней могут быть точки синхронизации, когда отдельные процессы и потоки могут ожидать другие процессы и потоки, но время такого ожидания должно быть достаточно мало по сравнению со временем счёта. Если программа периодически ожидает какие-то внешние события и проводит в этом ожидании значительное время, то данный метод к такой программе неприменим.

Таким образом, в идеале, вычислительная программа должна создавать загрузку CPU на 100%. Если же программа гибридная, и наряду с вычислениями на CPU используются вычисления на ускорителе, то в случае, когда ускоритель обрабатывает выделенные ему данные медленнее, чем CPU – выделенные ему, программа будет простаивать в точках синхронизации в ожидании окончания работы ускорителя. Вследствие этого загрузка программой CPU будет меньше 100%. Получение программой данных о том, насколько она потребляет выделенный ей ресурс CPU, делается очень легко и быстро. В OS семейства UNIX это, например, делается с помощью системного вызова **clock_gettime(…)**. Вызов с параметром **CLOCK_REALTIME** даёт общее системное время, а с параметром **CLOCK_PROCESS_CPUTIME_ID** – процессорное время, потреблённое выполняющейся программой. Если засечь эти два параметра за какой-то период, то величина загрузки CPU данной программой за этот период может быть вычислена по следующей формуле:

$$CPU_{load} = t_{CLOCK\_PROCESS\_CPUTIME\_ID} / N_{thread} / t_{CLOCK\_REALTIME} * 100\%,$$

где t – соответствующие времена, а N_{thread} – количество выполняющихся программных потоков на CPU.

Далее рассмотрим, как трактовать полученную величину загрузки CPU. Как говорилось выше, в идеале у вычислительной программы загрузка CPU должна

быть 100%. Но она также будет 100% в случае, если ускоритель обрабатывает свои данные быстрее, чем CPU, и программа не простаивает в ожидании готовности ускорителя. Чтобы иметь возможность диагностировать такую ситуацию, а также чтобы позволить программе проводить некоторое время в точках синхронизации с другими процессами, предлагается считать нормальной загрузкой CPU эмпирическое значение 95%. То есть допускается, чтобы CPU простаивал в ожидании ускорителя, но простаивал совсем немного, не более 5%. Если значение загрузки CPU больше того, что предложено считать нормой, значит, надо уменьшить его зону (и, соответственно, увеличить зону ускорителя). Если наоборот, меньше нормального – то увеличить зону CPU.

Но сдвиг границы зон влечёт за собой запуск процесса перераспределения данных, и выполнение его может занимать значительное время. Поэтому крайне желательно избежать ситуации в виде скачков постоянного то уменьшения, то увеличения размеров зон. Для этого нормальной загрузкой CPU предлагается считать не конкретное значение, а небольшой диапазон, например, от 85% до 95%.

Таким образом, периодически (например, на каждом *n*-ом шаге итерации) оценивая величину загрузки CPU за истёкший с момента предыдущей оценки интервал времени, можно принимать решение о том, оставить ли текущее распределение данных (если загрузка CPU находится в желаемом диапазоне), или увеличить зону CPU (если загрузка CPU ниже диапазона), или уменьшить зону CPU (если загрузка CPU выше диапазона). Также необходимо определить, насколько сильно надо сдвигать границу зон. Очевидно, что чем больше значение загрузки CPU отличается от желаемого, тем больше должно быть изменение границы зон. С другой стороны, двигать границу зон, когда размер одной зоны значительно превышает размер другой, надо с осторожностью, так как даже небольшой сдвиг границы может в процентном отношении существенно изменить объём обрабатываемых данных, что в свою очередь может в корне изменить баланс вычислительной загрузки между CPU и ускорителем. Резкое изменение соотношения размера зон может привести на следующем шаге к необходимости изменения в обратную сторону. И если оно также будет резким, то получатся только постоянные скачки то в одну, то в другую сторону, которых, как объяснялось выше, необходимо избегать. Поэтому вблизи крайних значений относительного размера зон алгоритм изменения грани-

цы зон должен вести себя достаточно аккуратно, сдвигая границу на небольшие значения. В результате алгоритм, определяющий величину изменения границы зон, может быть описан функцией от двух переменных – отклонение от желаемой загрузки CPU и текущий относительный размер зон.

Предложенный метод может быть алгоритмизирован, подходит для широкого класса вычислительных задач и не зависит от характеристик конкретной вычислительной системы. Следовательно, он может быть применён для автоматической балансировки вычислительной нагрузки. Планируется реализовать его в компиляторе программ на языке НОРМА [10] так, чтобы компилятор автоматически генерировал весь необходимый код для определения загрузки CPU, принятия решения о сдвиге границы зон и перераспределения обрабатываемых данных. Пока же изложенный метод реализован «вручную» в одной программе из области газодинамики, и в следующей главе рассказано о результатах его применения.

РЕЗУЛЬТАТЫ ПРИМЕНЕНИЯ ПРЕДЛОЖЕННОГО МЕТОДА

Метод автоматической балансировки вычислительной нагрузки был реализован в гибридной программе, решающей задачу из области газодинамики. Программа построена с использованием технологий MPI для задействования нескольких узлов вычислительного комплекса, OpenMP для вычислений на CPU внутри одного узла и NVIDIA CUDA для вычислений на GPU.

Приведённые ниже результаты были получены на вычислительном кластере K-100 [11] с использованием компиляторов Intel версии 15.0.0, nvcc версии 6.5 и Intel MPI Library 5.0 Update 1. В программе использовались 4 MPI процесса, каждый из которых выполнялся на своём узле вычислительного кластера, имеющего 12 ядер CPU и 3 GPU. Испытания проводились с разным количеством GPU, и метод исправно работал вне зависимости от количества GPU. Но, так как данная программа хорошо подходит для выполнения на GPU, доля вычислений CPU получалась очень маленькой. Поэтому далее приведены данные для запусков с использованием только 1 GPU, чтобы вклад CPU в общую долю вычислений был более заметным, а происходящие процессы были более наглядными. Программа также запускалась на вычислительном кластере K-60 [11], имеющем более мощные GPU. Метод автоматической балансировки вычислительной нагрузки также исправно работал и

там, но доля вычислений на CPU получалась ещё меньше – 4% при использовании всего 1 GPU.

На рис. 2 представлена диаграмма времён выполнения программы при различных значениях размера зоны GPU. Первые 10 столбцов соответствуют запускам с фиксированной границей зон, без использования автоматической балансировки вычислительной нагрузки, и зона GPU задаётся от 100% (когда CPU совсем не используется) до 83%. Последние 4 столбца – запуски с использованием автоматической балансировки вычислительной нагрузки, с различным значением первоначального размера зоны GPU: 100%, 75%, 50% и 0%.

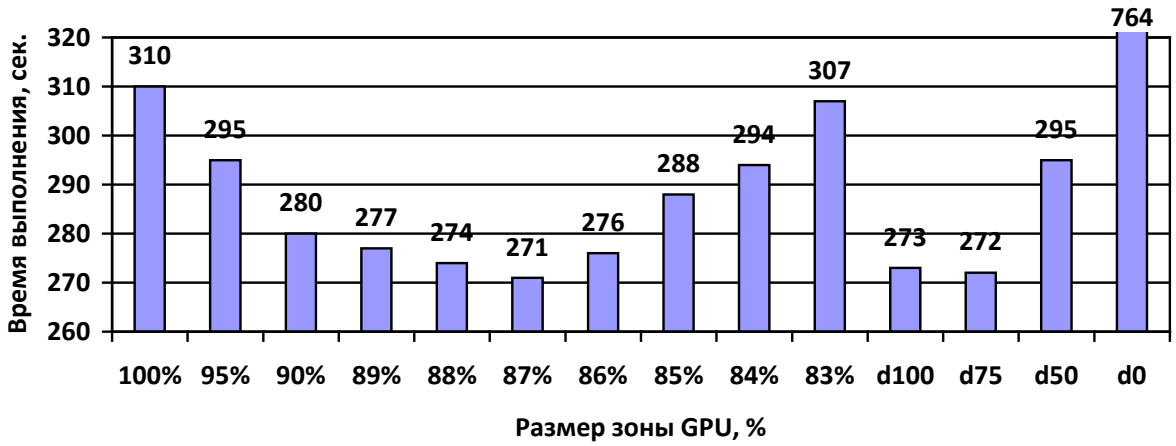

Рисунок 2. Время выполнения программы в зависимости от размера зоны GPU.

Как видно на диаграмме, наименьшее время выполнения программы достигается при размере зоны GPU 87% (и зоны CPU 13% соответственно). Затем, при небольшом увеличении зоны CPU, время выполнения программы начинает быстро расти – фактически на столько же в процентном отношении, на сколько растёт зона CPU, так как именно время работы CPU начинает определять время работы всей программы.

Особый интерес представляют столбцы, соответствующие запускам с использованием автоматической балансировки вычислительной нагрузки. Они показывают, что если первоначальное распределение вычислительной нагрузки было выбрано, хоть и приблизительно, правильно (d100 – начальная зона GPU 100% и d75 – начальная зона GPU 75%), то и время работы всей программы получается близким к идеальному. Но если первоначальное распределение было выбрано неверно

(d50 и, в особенности, d0), то программа тратит значительное время для выхода на своё правильное распределение.

На рис. 3 приведены графики изменения размеров зон в зависимости от шага итерации для запусков с использованием автоматической балансировки вычислительной нагрузки. Значения вычислительной нагрузки анализировались и корректировались на каждом 100-м шаге итерации.

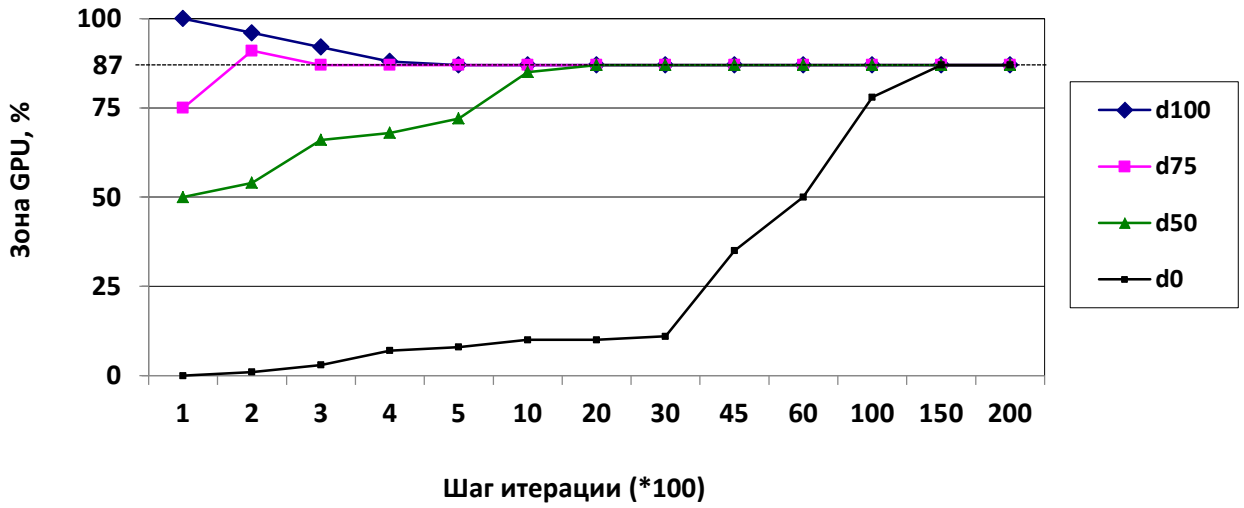

Рисунок 3. Изменение зоны GPU в зависимости от шага итерации.

Как видно на приведённом графике, запуски d100 и d75 уже на 4-й корректировке выходят на распределение, идеальное для данной задачи на данной аппаратуре: 87% для зоны GPU. Запуск d50 тратит на это уже больше шагов корректировки, и, как следствие, общее время его выполнения получается заметно больше. Запуск d0 очень долгое время осторожно отходит от нулевого размера зоны GPU, средние значения распределения зон проходятся уже значительно быстрее, и в итоге он также выходит на то же самое идеальное распределение, 87% для зоны GPU. Но для этого ему потребовалось 150 корректировок, и значительное время было упущено.

ЗАКЛЮЧЕНИЕ

Предложенный метод автоматической балансировки вычислительной нагрузки, несмотря на свою простоту, успешно может быть использован при реше-

нии расчётных задач на гибридных вычислительных системах. Испытания показали, что применение данного метода успешно выводит задачу на её идеальное распределение вычислительной нагрузки, а в случае небольшого изменения нагрузки способно быстро реагировать на это и корректировать такие изменения. Приведённый метод не зависит ни от аппаратных особенностей гибридной вычислительной системы, ни от средств программирования, выбранных для решения прикладной задачи.

СПИСОК ЛИТЕРАТУРЫ

1. *Воеводин В.В*. Математические модели и методы в параллельных процессах. М.: Наука, 1986. 296 с.

2. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. С.-Петербург: БХВ-Петербург, 2002. 608 с.

3. Информационно-аналитический центр по параллельным вычислениям. URL: *http://parallel.ru*

4. OpenACC. URL: http://openacc.org

5. DVM-система. URL: http://www.keldysh.ru/dvm

6. Система НОРМА. URL: http://www.keldysh.ru/pages/norma

7. *Андрианов А.Н., Баранова Т.П., Бугеря А.Б., Гладкова Е.Н., Ефимкин К.Н.* Язык НОРМА // Препринты ИПМ им. М.В.Келдыша. ISSN 2071-2898 (Print). ISSN 2071-2901 (Online). 2019. № 132. 48 с. doi:10.20948/prepr-2019-132.
URL: http://library.keldysh.ru/preprint.asp?id=2019-132.

8. *Андрианов А.Н., Баранова Т.П., Бугеря А.Б., Ефимкин К.Н.* Распределение вычислений в гибридных вычислительных системах при трансляции программ на языке НОРМА // Вычислительные методы и программирование. ISSN 1726-3522. М.: НИВЦ МГУ им. М.В. Ломоносова, 2019. Т. 20, № 3. С. 224–236.
DOI: 10.26089/NumMet.v20r321.
URL: http://num-meth.srcc.msu.ru/zhurnal/tom_2019/pdf/v20r321.pdf

9. *Андрианов А.Н., Баранова Т.П., Бугеря А.Б., Ефимкин К.Н.* Методы распределения вычислений при автоматическом распараллеливании непроцедурных спецификаций // Суперкомпьютерные дни в России: Труды международной конференции. 23–24 сентября 2019 г., г. Москва. Под. ред. Вл.В. Воеводина.

М.: МАКС Пресс, 2019. ISBN 978-5-317-06007-7. e-ISBN 978-5-317-06244-6. С. 59–70. DOI: 10.29003/m680.RussianSCDays.
URL: http://russianscdays.org/files/2019/pdf/59.pdf

10. *Андрианов А.Н., Бугеря А.Б., Ефимкин К.Н., Колударов П.И.* Модульная архитектура компилятора языка Норма+ // М.: Препринт ИПМ им. М.В. Келдыша РАН, 2011. № 64. 16 с.
URL: http://keldysh.ru/papers/2011/prep64/prep2011_64.pdf

11. Центр коллективного пользования ИПМ им. М.В. Келдыша РАН. URL: http://ckp.kiam.ru/?hard

COMPUTATIONAL LOAD BALANCING METHOD FOR HYBRID COMPUTING SYSTEMS

T. P. Baranova[1], A. B. Bugerya[2], K. N. Efimkin[3]

[1-3]*Keldysh Institute of Applied Mathematics, Moscow*

[1]bart1950@yandex.ru, [2]shurabug@yandex.ru, [3]bigcrocodile@yandex.ru

Abstract

The paper considers the issues of the computations distributing within one node of a hybrid computing system for applied programs with computation-intense operations. A method is proposed for static distribution of computations, as well as a method for automatic balancing of the computational load during program execution, which is based on periodic analyzing the CPU load by the executed program and making decision to redistribute computational load if necessary. The proposed methods are implemented in an applied program that solves a gas dynamic problem using the computing resources of the multicore central processor and graphics accelerators. The results of program execution with various data distributions were obtained and analyzed, both with and without the mechanism for automatic balancing of the computational load.

Keywords*: parallel programming, programming automation, computational load balancing, hybrid computing architectures, NORMA language, automatic program generation*

REFERENCES

1. Voevodin V.V. Matematicheskie modeli i metody v parallelnykh protsessakh. M.: Nauka, 1986. 296 s.

2. *Voevodin V.V., Voevodin Vl.V.* Parallelnye vychisleniia. S.-Peterburg: BKhV-Peterburg, 2002. 608 s.

3. Informational Analytical Center. URL: http://parallel.ru/index_eng.html

4. OpenACC. URL: http://openacc.org

5. DVM-system. URL: http://www.keldysh.ru/dvm

6. Sistema NORMA. URL: http://www.keldysh.ru/pages/norma

7. *Andrianov A.N., Baranova T.P., Bugerya A.B., Gladkova E.N., Efimkin K.N.* Iazyk NORMA. // Preprinty IPM im. M.V. Keldysha. ISSN 2071-2898 (Print). ISSN 2071-2901 (Online). 2019. № 132. 48 s. doi:10.20948/prepr-2019-132.
URL: http://library.keldysh.ru/preprint.asp?id=2019-132.

8. *Andrianov A.N., Baranova T.P., Bugerya A.B., Efimkin K.N.* Raspredelenie vychislenii v gibridnykh vychislitelnykh sistemakh pri transliatsii programm na iazyke NORMA // Vychislitelnye metody i programmirovanie. ISSN 1726-3522. M.: NIVTs MGU im. M.V. Lomonosova, 2019. Vol. 20, № 3. S. 224–236.
DOI: 10.26089/NumMet.v20r321.
URL: http://num-meth.srcc.msu.ru/zhurnal/tom_2019/pdf/v20r321.pdf

9. *Andrianov A.N., Baranova T.P., Bugerya A.B., Efimkin K.N.* Metody raspredeleniia vychislenii pri avtomaticheskom rasparallelivanii neprotsedurnykh spetsifikatsii // Superkompiuternye dni v Rossii: Trudy mezhdunarodnoi konferentsii. 23-24 sentiabria 2019 g., g. Moskva. Pod. red. Vl.V. Voevodina. M.: MAKS Press, 2019. P. 59–70.ISBN 978-5-317-06007-7. e-ISBN 978-5-317-06244-6.
DOI: 10.29003/m680.RussianSCDays.
URL: http://russianscdays.org/files/2019/pdf/59.pdf

10. *Andrianov A.N., Bugerya A.B., Efimkin K.N., Koludarov P.I.* Modulnaia arkhitektura kompiliatora iazyka Norma+ // M.: Preprint IPM im. M.V. Keldysha RAN, 2011. № 64. 16 s. URL: http://keldysh.ru/papers/2011/prep64/prep2011_64.pdf

11. Centr kollektivnogo pol'zovaniya IPM im. M.V. Keldysha RAN. URL: http://ckp.kiam.ru/?hard

СВЕДЕНИЯ ОБ АВТОРАХ

БАРАНОВА Татьяна Петровна – научный сотрудник ИПМ им. М.В. Келдыша РАН;

Tat'yana Petrovna BARANOVA – Keldysh Institute of Applied Mathematics, researcher.

email: bart1950@yandex.ru

ORCID: 0000-0002-2003-3737

БУГЕРЯ Александр Борисович – старший научный сотрудник ИПМ им. М.В. Келдыша РАН, к. ф.-м. н.

Alexander Borisovich BUGERYA – Keldysh Institute of Applied Mathematics, senior researcher, Ph.D.

email: shurabug@yandex.ru

ORCID: 0000-0002-9698-458X

ЕФИМКИН Кирилл Николаевич – старший научный сотрудник ИПМ им. М.В. Келдыша РАН, к. ф.-м. н.

Kirill Nikolaevich EFIMKIN – Keldysh Institute of Applied Mathematics, senior researcher, Ph.D.

email: bigcrocodile@yandex.ru

ORCID: 0000-0002-1087-7645

Материал поступил в редакцию 25 ноября 2020 года