ИПМ им.М.В.Келдыша РАН

Абрау-2020 • Труды конференции

А.Н. Андрианов, Т.П. Баранова,
А.Б. Бугеря, Е.Н. Гладкова,
К.Н. Ефимкин, П.И. Колударов

Автоматическая балансировка вычислительной нагрузки между центральным процессором и ускорителем

Рекомендуемая форма библиографической ссылки

Андрианов А.Н., Баранова Т.П., Бугеря А.Б., Гладкова Е.Н., Ефимкин К.Н., Колударов П.И. Автоматическая балансировка вычислительной нагрузки между центральным процессором и ускорителем // Научный сервис в сети Интернет: труды XXII Всероссийской научной конференции (21-25 сентября 2020 г., онлайн). — М.: ИПМ им. М.В.Келдыша, 2020. — С. 19-28.
https://doi.org/10.20948/abrau-2020-25
https://keldysh.ru/abrau/2020/theses/25.pdf

Видеозапись выступления

Размещена также презентация к докладу

Автоматическая балансировка вычислительной нагрузки между центральным процессором и ускорителем

А.Н. Андрианов[1], Т.П. Баранова[1], А.Б. Бугеря[1], Е.Н. Гладкова[1], К.Н. Ефимкин[1], П.И. Колударов[2]

1 Институт прикладной математики им. М.В. Келдыша РАН
2 ООО «ЕМЕ»

Аннотация. В работе рассмотрены вопросы балансировки вычислительной нагрузки, возникающие при распределении вычислений внутри одного узла гибридной (содержащей, помимо центральных процессоров, ещё и специализированные ускорители) вычислительной системы. Предложен метод для статического распределения вычислений, а также метод для автоматической балансировки вычислительной нагрузки в процессе выполнения программы, который основывается на периодическом анализе величины загрузки центрального процессора выполняемой программой и принятии решения о перераспределении вычислительной нагрузки в случае необходимости. Предложенные методы реализованы в программе, решающей задачу из области газодинамики с использованием вычислительных ресурсов 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, балансировка вычислительной нагрузки

Automatic computational load balancing between CPU and accelerator

A.N. Andrianov[1], T.P. Baranova[1], A.B. Bugerya[1], E.N. Gladkova[1], K.N. Efimkin[1], P.I. Koludarov[2]

[1] Keldysh Institute of Applied Mathematics of the Russian Academy of Sciences
[2] ООО «ЕМЕ»

Abstract. The paper considers the issues of the computational load balancing arisen during the distribution of computing within one node of the

hybrid (containing, in addition to the central processors, specialized accelerators) computing system. There proposed a method for the static distribution of computations, and another one for automatic balancing of the computational load during program execution. This method is based on a periodic analysis of the CPU load by the program and deciding on the redistribution of computational load if necessary. The proposed methods are implemented in a program that solves the problem of gas dynamics using the computing resources of the 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; computational load balancing

1. 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 ресурсоёмких 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.

Помимо центральных процессоров общего назначения (CPU) такие 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 обращении к общим данным, минимизировать простои вычислителей при синхронизации и при 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, и от успешного решения этой задачи в значительной степени зависит и эффективность выполнения всей программы в целом.

В данной статье будет рассмотрен подход к организации автоматической динамической балансировкой вычислительной нагрузки в пределах одного узла вычислительной системы, в составе которого есть один или более CPU и один или более GPU. Вопрос распределения вычислительной нагрузки между узлами вычислительной системы остаётся за рамками данной работы. Описываемый подход был разработан для применения в системе программирования HOPMA [1] при трансляции программ для вычислительных систем с гибридной архитектурой.

Подход был успешно опробован на программе вычислительного характера из области газовой динамики, на параллельных системах с гибридной архитектурой.

2. Метод распределения вычислительной нагрузки

Рассмотрим вопрос распределения вычислительной нагрузки в пределах одного узла гибридной вычислительной системы. В каждом таком узле имеется 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 как единый многоядерный процессор. Также в узле гибридной вычислительной системы имеется 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 обрабатываемых данных должен быть распределён между CPU и ускорителями. Данные, обрабатываемые на CPU, должны обрабатываться с использованием технологий для многопоточного программирования, например, OpenMP. Данные, обрабатываемые на ускорителе, должны обрабатываться 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 HOPMA, подробно описан в работах [2, 3]. Методы и идеи, изложенные в них, могут быть применены к любой параллельной программе и любому

другому типу ускорителей. Кратко, эти методы заключаются в следующем. Весь объём обрабатываемых данных делится на 2 неравные части: зона, обрабатываемая CPU и зона, обрабатываемая ускорителем (ускорителями). Затем, если ускорителей несколько, то их зона делится на соответствующее число подзон и распределяется между ними поровну. На рис. 1 приведён пример такого распределения. При этом дополнительно решается задача передачи данных между зонами и подзонами, если в этом есть необходимость.

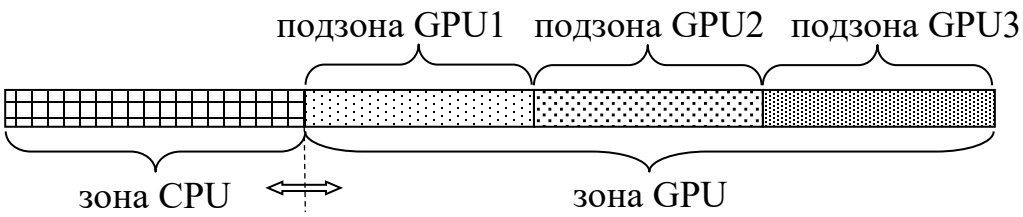

Рис. 1. Распределение обрабатываемых данных между зонами

Граница между зоной CPU и зоной ускорителей может быть фиксированной, а может, если применяется динамическая балансировка вычислительной нагрузки, о которой пойдёт речь ниже, быть изменяемой в процессе выполнения программы. Для того, 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.

3. 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 и на ускорителе, то, в случае, когда ускоритель обрабатывает выделенные ему данные медленнее, чем CPU – выделенные ему, то программа будет простаивать в точках синхронизации в ожидании окончания работы ускорителя. И, 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** даёт 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

Теперь разберёмся, 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 нормального, 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, что может быть дорого по времени. Поэтому скачков в виде постоянного то уменьшения, то увеличения размеров зон требуется категорически избегать. 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 прыжки то в одну, то в другую сторону. Поэтому вблизи крайних значений относительного размера зон алгоритм изменения границы 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 [4] так, чтобы компилятор автоматически генерировал бы весь необходимый код для определения загрузки CPU, принятия решения о сдвиге границы зон и для перераспределения обрабатываемых данных. Пока же изложенный метод реализован «вручную» в одной программе из области газодинамики и в следующей главе рассказывается о результатах его применения.

4. 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. Испытания проводились на вычислительном кластере K-100 [5] с использованием компиляторов Intel версии 15.0.0, nvcc версии

6.5 и Intel MPI Library 5.0 Update 1. В программе использовалось 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, и происходящие процессы были более наглядными.

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%.

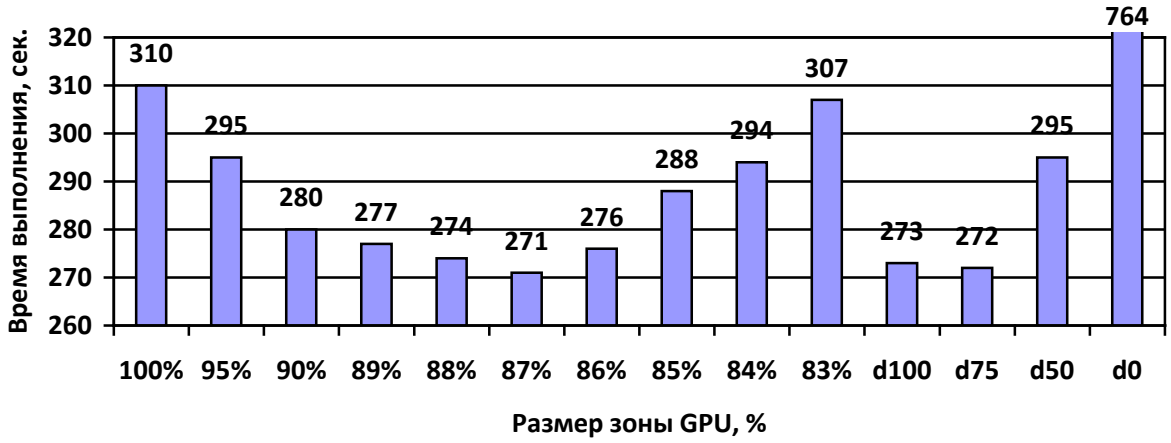

Рис. 2. Время выполнения программы в зависимости от размера зоны GPU

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.

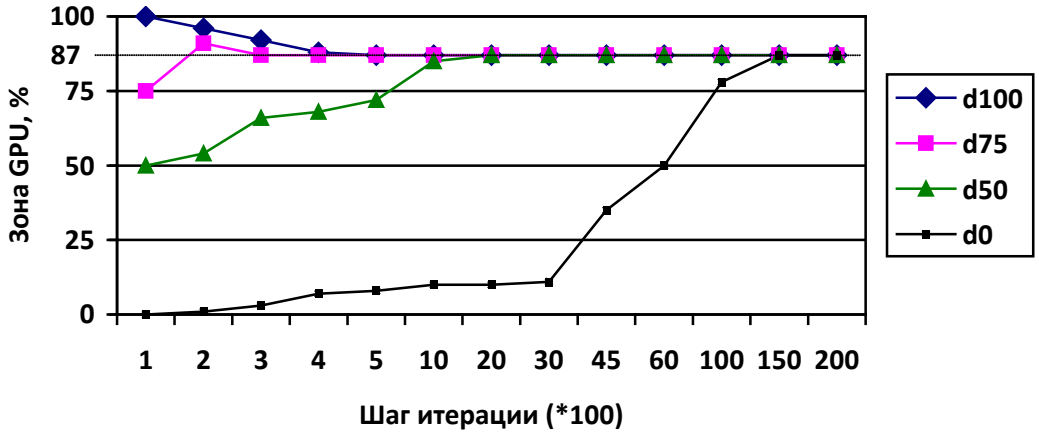

Рис. 3. Изменение зоны GPU в зависимости от шага итерации

Как видно на приведённом графике, запуски d100 и d75 уже на 4-й корректировке выходят на идеальное для данной задачи на данной аппаратуре распределение: 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.

5. Заключение

Предложенный метод автоматической балансировки вычислительной нагрузки, несмотря на свою простоту, успешно может быть использован для решения 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 корректировать такие изменения.

Работа выполнена при финансовой поддержке гранта РФФИ № 18-01-00131-а.

Литература

1. Система НОРМА. — URL: http://www.keldysh.ru/pages/norma
2. 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
3. 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
4. А.Н. Андрианов, А.Б. Бугеря, К.Н. Ефимкин, П.И. Колударов. Модульная архитектура компилятора языка Норма+ // Препринты ИПМ им. М.В. Келдыша, 2011. — № 64. — 16 стр. — URL: http://keldysh.ru/papers/2011/prep64/prep2011_64.pdf
5. Гибридный вычислительный кластер К–100. — URL: http://ckp.kiam.ru/?hard

References

1. Sistema NORMA. — URL: http://www.keldysh.ru/pages/norma
2. Andrianov A.N., Baranova T.P., Bugeria 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. — P. 224–236. — DOI: 10.26089/NumMet.v20r321. — URL: http://num-meth.srcc.msu.ru/zhurnal/tom_2019/pdf/v20r321.pdf
3. Andrianov A.N., Baranova T.P., Bugeria 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. — ISBN 978-5-317-

06007-7. — e-ISBN 978-5-317-06244-6. — P. 59–70. — DOI: 10.29003/m680.RussianSCDays. — URL: http://russianscdays.org/files/2019/pdf/59.pdf
4. A.N. Andrianov, A.B. Bugeria, K.N. Efimkin, P.I. Koludarov. Modulnaia arkhitektura kompiliatora iazyka Norma+ // M. : Preprint IPM im. M.V. Keldysha RAN, 2011. — № 64. — 16 p. — URL: http://keldysh.ru/papers/2011/prep64/prep2011_64.pdf
5. Gibridnyi vychislitelnyi klaster K–100. — URL: http://ckp.kiam.ru/?hard