Автоматизация создания вычислительных программ для современных кластеров*

В.А. Крюков

ИПМ им. М.В. Келдыша РАН

В докладе будут рассмотрены три направления автоматизации параллельного программирования:
1. Использование гибридных языков, объединяющих разные модели параллельного программирования;
2. Использование языков с неявным параллелизмом;
3. Автоматизация преобразования имеющихся последовательных программ или параллельных MPI-программ в эффективные параллельные программы на гибридных языках или языках с неявным параллелизмом.

1. Введение

При разработке программ для многоядерных кластеров программисту приходится использовать две сильно различающиеся модели программирования (MPI, OpenMP) и соответствующие инструментальные средства. Появление в узлах кластера графических процессоров серьезно усложнило разработку программ, поскольку потребовало дополнительно использовать еще и низкоуровневую технологию CUDA [1]. На подходе новые процессоры с большим количеством ядер (например, 48-ядерный Intel SCC процессор, который стал недавно доступен для широкого круга исследователей [2]), для эффективного использования которых потребуются новые модели программирования.

Разработчики новых языков параллельного программирования явно не успевают отслеживать архитектурное разнообразие многоядерных процессоров. Очень маловероятно, что в течение 5-10 лет появится и широко распространится новый высокоуровневый язык. Поэтому в ближайшие годы программистам придется использовать гибридные языки, объединяющие разные модели параллельного программирования.

Однако такой подход может рассматриваться только в качестве временной меры - нельзя и дальше усложнять модели параллельного программирования. Если в долгосрочном плане можно надеяться на появление и стандартизацию новых языков программирования высокого уровня, то в среднесрочном плане наиболее привлекательным подходом представляется использование для создания параллельных программ языков с неявным параллелизмом (Фортран, Си, Си++). Вполне вероятно, что такие языки надолго останутся наиболее удобными языками для создания вычислительных программ, алгоритмы которых, как правило, не требуют для своего выражения привлечения механизмов параллельных процессов и средств их взаимодействия.

Независимо от того, какой из упомянутых подходов будет использоваться программистами, разработка параллельных программ намного бы упростилась и ускорилась, если бы им были предоставлены средства автоматизации преобразования имеющихся у них последовательных программ (и уже широко распространенных параллельных MPI-программ) в эффективные параллельные программы для современных кластеров.

2. Использование гибридных моделей и языков

В настоящее время при разработке программ для высокопроизводительных вычислений на современных кластерах широко используются три модели программирования – MPI, OpenMP и CUDA. При этом пока вполне можно обходиться комбинацией MPI/OpenMP или MPI/CUDA, поскольку GPU используют для тех программ, для которых они на порядок эффективнее, чем

* Работа поддержана программой Союзного государства СКИФ-ГРИД, программами Президиума РАН №14 и №15, грантом РФФИ № 10-07-00211, грантом Президента РФ МК-3324.2010.9.

многоядерные процессоры, и, следовательно, неполной загрузкой многоядерных процессоров можно пренебречь. Если будут появляться программы, в которых только часть вычислений выгодно производить на GPU, а оставшиеся вычисления лучше оставить на универсальном многоядерном процессоре, то придется использовать и комбинацию из трех перечисленных моделей. Технически объединять низкоуровневые модели программирования, реализованные через библиотеки, проще, чем высокоуровневые модели, реализуемые посредством языков и соответствующих компиляторов. Но программировать, отлаживать, сопровождать, переносить на другие ЭВМ такие программы гораздо сложнее. Например, при переходе от GPU фирмы NVIDIA к GPU фирмы AMD придется заменить CUDA на OpenCL [3]. Поэтому важно создавать высокоуровневые гибридные модели и языки программирования.

Примером такой модели является гибридная модель DVM/OpenMP [4] разработанная в ИПМ им. М.В.Келдыша РАН. Она объединяет две модели (DVM [5] и OpenMP), реализуемые через расширения стандартных языков директивами компилятору. Недавно аналогичная модель (PGI_APM [6]) появилась и для GPU, ориентированная на ускорители разной архитектуры (в частности, для GPU NVIDIA и AMD). Все эти три модели (DVM, OpenMP, PGI_APM) объединяет и то, что программист может полностью контролировать отображение данных и вычислений на аппаратуру. Поэтому вполне естественно создать гибридные модели DVM/PGI_APM и DVM/OpenMP/PGI_APM и соответствующие языки и компиляторы. Это не только упростит дальнейшую разработку автоматически распараллеливающих компиляторов, но и предоставит возможность использовать эти гибридные языки для некоторого промежуточного представления распараллеленной программы, на котором при необходимости программист сможет проводить дополнительную ручную оптимизацию программы.

3. Использование языков с неявным параллелизмом

В настоящее время многие специалисты предлагают использовать для разработки параллельных программ языки с неявным параллелизмом, при программировании на которых не требуется знать архитектуру параллельной ЭВМ, и автоматически отображать такие программы на параллельные машины. Ярким примером такого языка является разработанный под руководством И.Б. Задыхайло в 1985 году декларативный язык НОРМА [7]. В качестве таких языков особенно привлекательно использовать Фортран и Си/Си++, поскольку их в основном и используют программисты при решении задач, наиболее остро требующих распараллеливания.

Таким образом, снова ставится вопрос об автоматическом распараллеливании последовательных программ. Что же изменилось за последние 20 лет, чтобы снова возвращаться к проблеме, уже давно (с появлением многопроцессорных ЭВМ с распределенной памятью) считающейся непреодолимой?

Во-первых, для обеспечения возможности использования языков с неявным параллелизмом задача автоматического распараллеливания теперь имеет другую постановку – автоматически распараллеливать надо не уже имеющиеся программы, а новые программы, предназначенные для параллельного выполнения. Такие препятствия для распараллеливания, как применение сугубо последовательного алгоритма или неудачные технические решения (например, вместо двумерного массива в программе с целью экономии памяти используется список векторов разной длины) должны быть устранены программистом.

Во-вторых, проблемы статического анализа, неспособного справиться в случаях использования в программе косвенной индексации или зависимости индексов и параметров циклов от динамически вводимых данных, могут быть решены с помощью динамического анализа или дополнительных аннотаций, вставляемых программистом в программу. Да и техника статического анализа за прошедшие два десятилетия существенно продвинулась – межпроцедурный анализ хорошо изложен в учебной литературе [8], появились доступные программные средства [9], существенно упрощающие его реализацию. Да и многократно возросшие возможности ЭВМ позволяют справляться в приемлемое время с анализом даже очень больших программ.

В-третьих, в работах [10, 11] показано, что для достаточно широкого класса программ удается их эффективно распараллелить для многопроцессорных ЭВМ с распределенной памятью. Распараллеливание для таких ЭВМ принципиально отличается от распараллеливания для мультипроцессоров и GPU тем, что невозможно инкрементальное (постепенное) распараллелива-

ние – надо находить глобальные решения по распределению данных и вычислений, что влечет за собой необходимость прогнозирования времени выполнения программы для разных возможных вариантов ее распараллеливания. Несомненно, что построение упомянутого выше автоматического распараллеливателя для многопроцессорных ЭВМ с распределенной памятью было бы проблематичным, если бы не был использован многолетний опыт ручного распараллеливания последовательных программ, существенный задел по реализации высокоуровневых языков параллельного программирования, развитые средства функциональной отладки и отладки эффективности параллельных программ [12].

Полученный нами опыт автоматического распараллеливания для многоядерных кластеров [13] и публикации об успешных экспериментах с автоматическим распараллеливанием для GPU [14, 15], позволяет нам надеяться, что языки с неявным параллелизмом очень скоро станут широко использоваться для создания программ для кластеров с многоядерными и графическими процессорами.

Необходимо отметить, что методы автоматического распараллеливания могут быть применены не только к последовательным программам, но и к имеющимся параллельным MPI-программам для адаптации их к многоядерным и графическим процессорам, используемым в узлах современных кластеров.

Тем не менее, надо учитывать, что автоматическое распараллеливание для многопроцессорных ЭВМ с распределенной памятью принципиально отличается от привычной оптимизации, широко применяемой в компиляторах. Вполне вероятно, что такое автоматическое распараллеливание не сможет рассматриваться как отдельная фаза компиляции, а будет представлять собой отдельный шаг преобразования последовательной программы в параллельную программу и потребует создания специальных методов сокращения требуемых ресурсов. Возможно, эти методы будут очень схожи с когда-то популярным методом инкрементальной компиляции, в котором отслеживались модификации программы, сделанные после ее предыдущей компиляции.

4. Автоматизация преобразования имеющихся программ

Изменение постановки задачи автоматического распараллеливания вызывает изменение подхода к автоматизации распараллеливания программ: использовать диалог с программистом прежде всего для уточнения свойств последовательной программы, а выбор наилучших решений по ее распараллеливанию осуществлять полностью автоматически.

Преобразование последовательной программы в параллельную программу можно представить состоящим из двух основных этапов.

На первом этапе проводится автоматизированное исследование и преобразование программистом последовательной программы с целью получить такую последовательную программу, которую автоматически распараллеливающий компилятор может преобразовать в эффективную параллельную программу. При этом от программиста может потребоваться описание свойств программы (через диалог или в виде специальных аннотаций в тексте программы), необходимых или очень существенных для ее автоматического распараллеливания. Например, в случае косвенной индексации элементов массива статический анализатор может сообщить о возможной зависимости между витками цикла, но программист может указать, что этой зависимости нет.

На втором этапе автоматически распараллеливающий компилятор преобразует потенциально параллельную программу в параллельную программу для заданной ЭВМ.

Достижение приемлемой эффективности выполнения параллельной программы может потребовать многократного повторения этих двух этапов, но система автоматизации распараллеливания должна быть ориентирована на сокращение таких итераций. Прежде всего, она должна позволить на инструментальной машине и еще до появления текста параллельной программы оценить эффективность ее работы на разных ЭВМ.

Эта способность быстрой оценки влияния разных решений по распараллеливанию на эффективность программы, которая сопровождается детальной разъясняющей информацией, очень важна для ускорения разработки параллельных программ, а также при обучении параллельному программированию.

Тут опять следует отметить принципиальное отличие распараллеливания для кластеров от распараллеливания для мультипроцессоров, вызванное невозможностью инкрементального распараллеливания. При распараллеливании на мультипроцессор программисту достаточно показать причины, по которым не удается распараллелить те циклы, выполнение которые требует больше всего времени. При распараллеливании же на кластер возможна ситуация, когда абсолютно все циклы могут быть распараллелены, но не удается найти такой вариант распределения данных, чтобы время выполнения параллельной программы было бы меньше, чем время выполнения исходной последовательной программы. Как убедить программиста, что такого варианта распределения данных не существует? Наверное, можно ему предоставить возможность задать самому распределение всех массивов, а система должна показать, что заданный им вариант распределения не лучше тех, которые были найдены ею.

5. Заключение

При разработке программ для многоядерных кластеров программисту сейчас приходится использовать две сильно различающиеся модели программирования (MPI, OpenMP) и соответствующие инструментальные средства. Появление в узлах кластера графических процессоров серьезно усложнило разработку программ, поскольку потребовало дополнительно использовать еще и низкоуровневую технологию CUDA. На подходе новые процессоры с большим количеством ядер, для эффективного использования которых потребуются новые модели программирования.

Разработчики новых языков параллельного программирования явно не успевают отслеживать архитектурное разнообразие многоядерных процессоров. Новые языки (Chapel [16], X10 [17], Fortress [18]) еще не доказали свою эффективность для современных кластеров, а уже выглядят слишком сложными и непривычными для программистов.

Использование гибридных языков, объединяющих разные модели параллельного программирования может рассматриваться только в качестве временной меры – нельзя и дальше усложнять модели параллельного программирования.

Поэтому наиболее привлекательным подходом представляется использование для создания параллельных программ вычислительного характера языков с неявным параллелизмом (Фортран, Си, Си++) и автоматически распараллеливающих компиляторов. За последние годы получены убедительные доказательства, что такие компиляторы для современных кластеров могут быть созданы.

Появление автоматически распараллеливающих компиляторов позволит создавать качественно новые системы автоматизации преобразования имеющихся последовательных программ (и уже широко распространенных параллельных MPI-программ) в эффективные параллельные программы для современных кластеров. Их способность быстрой оценки влияния разных решений по распараллеливанию на эффективность программы, которая сопровождается детальной разъясняющей информацией, очень важна для ускорения разработки параллельных программ, а также при обучении параллельному программированию.

Литература

1. NVIDIA CUDA, http://developer.nvidia.com/object/cuda.html
2. Timothy G. Mattson, Rob F. Van der Wijngaart, and other. ”The 48-core SCC processor: the programmer’s View”/ Proceedings of the 2010 ACM/IEEE conference on Supercomputing, SC10, New Orleans, Louisiana, Nov. 2010
3. Nunshi, A., Ed. The OpenCL Specification. The Kronos Group, Oct. 2009
4. В.А. Бахтин, Н.А.Коновалов, В.А. Крюков. Расширение языка OpenMP Fortran для распределенных систем. Вопросы атомной науки и техники. сер. Математическое моделирование физических процессов. 2002 г. Вып.4. стр.65-70

5. Konovalov N.A., Krukov V.A., Mihailov S.N. and Pogrebtsov A.A. Fortran DVM - a Language for Portable Parallel Program Development. Proceedings of Software For Multiprocessors & Supercomputers: Theory, Practice, Experience. Institute for System Programming, RAS, Moscow, 1994
6. The Portland Group, Inc. The PGI Fortran and C Accelerator Programming Model, Nov. 2009. available at www.pgroup.com/accelerate
7. А.Н.Андрианов, К.Н.Ефимкин, И.Б.Задыхайло, Н.В.Поддерюгина. Язык Норма. Препринт ИПМ им.М.В.Келдыша АН СССР, N165, 1985, 34 с
8. Альфред *В.* Ахо*,* Моника *С.* Лам*,* Рави Сети*,* Джеффри Д. Ульман. Компиляторы: Принципы, технологии и инструментарий, 2-е издание: Пер. с англ.- М.ООО "И.Д.Вильямс", 2008.-1184 с
9. Дроздов А.Ю. Компонентный подход к построению оптимизирующих компиляторов. // Программирование. 2009. № 5, с. 70-80
10. Клинов М.С., Крюков В.А. Автоматическое распараллеливание Фортран-программ. Отображение на кластер // Вестник Нижегородского университета им. Н.И. Лобачевского. 2009. № 2. С. 128–134
11. М.С. Клинов. Алгоритмы автоматического отображения последовательных Фортран-программ на кластер. // Труды Всероссийской научной конференции “Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность”, сентябрь 2009 г., г. Новороссийск. – М.: Изд-во МГУ, 2009, с. 298-299
12. DVM система: http://www.keldysh.ru/dvm
13. В.А. Бахтин, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина. Автоматическое распараллеливание последовательных программ для многоядерных кластеров. //Труды Международной научной конференции “Научный сервис в сети Интернет: суперкомпьютерные центры и задачи”, сентябрь 2010 г., г. Новороссийск. – М.: Изд-во МГУ, 2010, с. 12-15
14. Muthu Manikandan Baskaran, J. Ramanujam, and P. Sadayappan/ Automatic C-to-CUDA Code Generation for Affine Programs, R. Gupta (Ed.): CC 2010, LNCS 6011, pp. 244–263, 2010
15. Lee, S., Min, S.-J., and Eigenmann, R. OpenMP to GPGPU: A compiler framework for automatic translation and optimization. In Proceedings of the 2009 ACM SIGPLAN Symposium in Principles and Practice of Parallel Programming (Raleigh, N.C., Feb. 2009), pp. 101–110
16. http://chapel.cray.com/
17. http://x10-lang.org/
18. E. Allen, D. Chase, J. Hallett, V. Luchangco, J.-W. Maessen, S. Ryu, G. Steele, and S. Tobin-Hochstadt. The Fortress language specification. Available from http://research.sun.com/projects/plrg/

Отформатировано: русский
Отформатировано: русский
Отформатировано: русский
Отформатировано: русский
Отформатировано: русский