В. А. Евстигнеев

NUMA-АРХИТЕКТУРА: НЕКОТОРЫЕ ОСОБЕННОСТИ КОМПИЛЯЦИИ И ГЕНЕРАЦИИ КОДА[1]

ВВЕДЕНИЕ

Известно, что мультипроцессорные системы характеризуются наличием множества идентичных процесоров, сообща решающих одну и ту же задачу и использующих общие ресурсы системы (например, память). Характерной особенностью мультипроцессоров является единое адресное пространство, поддерживаемое аппаратно. Конструктивно общая память может быть выполнена различно как в виде отдельной центральной памяти (как, например, в машинах Cray, NEC, Encore, Sequent и др.), так и в виде пространственно разделенной памяти (как в машинах Alliant, KSR, BBN, Cedar). В последнем случае машины приобретают свойства наращиваемой (scalable) вычислительной системы.

В машинах с общей памятью процессоры обмениваются информацией с помощью доступа к одной и той же ячейке памяти. В машинах с распределенной памятью процессоры с их локальной памятью обмениваются информацией путем явной посылки и получения сообщений. Накладные расходы поддержки обмена сообщениями — главная забота этих машин.

Можно утверждать, что различие между этими двумя семействами MIMD-машин исчезает. В машинах с общей памятью для сокрытия задержек при обращении к памяти вводятся такие механизмы, как хорошо организованная иерархическая система памяти. Для машин с распределенной памятью ожидается применение лучших схем выбора маршрутов для посылки сообщений для уменьшения накладных расходов, связанных с обменом сообщениями. Эта тенденция может привести к появлению машин с распределенной памятью, поддерживающих вычисления мелкозернистого уровня.

Единое глобальное пространство общей памяти на машинах с общей памятью обеспечивает более прозрачную модель программирования для MIMD-машин. Однако использование иерархической системы

[1]Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

памяти не позволит машинам с общей памятью быть наращиваемыми в достаточной степени. В противоположность этому машины с распределенной памятью могут легко наращиваться, но становятся более трудными для прикладного программирования. На конференции Supercomputing'91 фирма Alliant Computer Systems анонсировала систему Campus/800 с 800 процессорами. Она объявила, что Campus/800 есть первая система, поддерживающая и распределенную, и общую память. Ожидается, что система может облегчить программирование и повысить эффективность в реальных применениях.

Под наращиваемостью системы понимается возможность собирать ее из большого числа базисных компонент без принципиальной перестройки системы и ее программного обеспечения. Такими компонентами могут быть процессоры, блоки памяти, блоки сети связи и т.д. Ключ к наращиваемости системы — наличие пространственной и временной локальности вычислений. Естественно, что для каждой системы имеются пределы наращиваемости. Так, Cray Y-MP допускает конфигурации от одного до восьми процессоров, CM-2 — от 8K до 64K процессорных элементов, Cray C-90 имеет коэффициент наращиваемости 16, KSR-1 — 128 (от 8 до 1088 процессоров) и т.д.

1. ПОНЯТИЕ NUMA-АРХИТЕКТУРЫ

Масштабируемые (scalable) параллельные машины часто организуются как сети пар процессор-память; примерами таких машин являются машины фирмы BBN (Butterfly, TC 2000, GP 2000), KSR-1 и KSR 2 фирмы Kendall Square Research, NCUBE 2, а также мультикомпьютеры, подобные Intel iPSC/i860. Такие машины называются *машинами с неоднородным доступом к памяти*, или NUMA-машинами (*non-uniform memory access*), поскольку процессор получает доступ к своей локальной памяти в сотни и тысячи раз быстрее, нежели доступ к нелокальным данным. Например, в KSR-1 доступ к локальной памяти составляет 18 тактов, в то время как к нелокальной памяти — 175 тактов. Машины с распределенной памятью, подобные Intel iPSC/i860, имеют значительно большую неоднородность относительно времени доступа, так как доступ к нелокальным данным должен сопровождаться обменом сообщений. Если нелокальные доступы располагаются на критическом пути в программе, то превращение их в локальные путем управления данными будет ускорять исполнение программы.

Следующая особенность большинства NUMA-архитектур состоит в

том, что пересылка данных между процессорами блоками более эффективна, нежели посылка этих данных большим количеством маленьких сообщений. Пересылка данных между процессорами может рассматриваться как конвейер с большим временем разгона, сравнимым с временем работы одной ступени конвейера. Например, для Intel iPSC/i860 требуется 70 микросекунд для начала коммуникации, и лишь одна микросекунда для пересылки числа с плавающей точкой двойной точности между ближайшими соседями, если коммуникация между ними была уже налажена. Поэтому, когда некоторое число элементов данных должно быть отправлено между двумя процессорами, желательно использовать одно длинное сообщение для того, чтобы погасить время разгона.

1.1. Мультикомпьютеры

Итак, кроме машин, представляющих собой мультипроцессоры с общей разделенной памятью, к машинам с NUMA-архитектурой относятся практически все мультикомпьютеры. Различие между мультипроцессорными и мультикомпьютерными системами заключется в следующем:

- В мультикомпьютерах отсутствует аппаратная поддержка единого адресного пространства. Каждый компьютер в мультикомпьютере имеет свое адресное пространство. Общее глобальное адресное пространство формируется программными средствами как конкатенация номера компьютера представляющего собой узел мультикомпьютерной системы, и адреса внутри узлового адресного пространства для поддержки SPMD-модели вычислений.
- Мультикомпьютеры представляют собой семейство независимых, связанных между собой ЭВМ под управлением распределенной ОС сетевого типа. Каждый компьютер имеет копию ядра ОС.
- Распределение работ (программ и относящихся к ним данных) в мультикомпьютере происходит в основном статически, при изменении нагрузки компьютеров закрепленная задача может быть передвинута в другой узел.

  Размер задачи ограничен размером памяти узла. Мультикомпьютер не способен (или не очень эффективен) исполнять семейство больших скалярных программ, которые типичны для таких областей применения, как цифровое моделирование.
- Обмен данными в мультикомпьютерах осуществляется путем явной посылки сообщений, которые могут быть скрыты от пользо-

вателя аппаратно или программно. При этом возникает проблема такого распределения задач и данных по узлам, при котором расходы на организацию посылки сообщений были бы минимальны. Для минимизации расходов на обмен сообщениями данные могут перемещаться или переименовываться так же, как в системе виртуальной памяти.

- В отличие от мультипроцессоров, где данные хранятся в общей памяти в единственном экземпляре, в мультикомпьютерах из-за использования механизма посылки сообщений в одно и то же время в разных компьютерах могут храниться копии одних и тех же переменных.
- Каждое нелокальное (т.е. не к своей локальной памяти) обращение к данным в мультикомпьютере требует специального программного механизма для вычисления адреса, организации посылки сообщения и управления памятью, чтобы иметь возможность обрабатывать копии данных.
- В отличие от мультипроцессоров, где механизм посылки сообщений представляет собой работу с указателями, мультикомпьютеры требуют обязательного перемещения данных.
- Мультикомпьютеры хорошо работают на очень больших параллельных программах, исполняемых до полного завершения. Если любой из узлов испытывает недостаток некоторых средств, которые должны быть получены от других узлов, то возникают трудно устранимые узкие места.
- Мультикомпьютеры чаще всего не являются системами общего назначения в терминах области применения или размера задачи.

Примечание. Для уменьшения внутрисистемных задержек процессоры и устройства локальной памяти объединяются для создания эффективных вычислительных узлов мультикомпьютера. Используется также размещение одной и той же программы во всех узлах и рассылка данных по всем компьютерам для уменьшения расходов на перемещение данных. Последнее основано на принципе широковещания.

1.2. Прочие машины с NUMA-архитектурой

Среди новейших машин к классу NUMA-машин следут отнести суперкомпьютеры NCUBE 2, IBM SP-2, Convex SPP 1200/XA, Intel Paragon, Thinking Machine CM5, IBM RP3, проект DASH (Стенфорд), проект ALEWIFE (МТИ), Horizon/Tera.

2. ОСОБЕННОСТИ ПО NUMA-МАШИН

Для разработчика ПО следствием указанных выше особенностей NUMA-машин является то, что программы должны не только эксплуатировать параллелизм, но и управлять данными там, где это возможно, для исключения нелокальных ссылок; там, где нелокальные ссылки необходимы, они должны быть сгруппированы для блочной пересылки. Существующая компиляторная технология ориентирована большей частью на машины с *однородным* доступом к памяти, в которых одна забота — эксплуатация параллелизма. Параллельный код генерируется путем распределения среди процессоров итераций самого внешнего цикла в гнезде циклов вместе с вставкой команд синхронизации, чтобы позаботиться о зависимостях, порожденных этим циклом. Для уменьшения синхронизации применяются преобразования типа перестановки циклов для перемещения параллельных циклов в сторону самого внешнего, где это возможно. Этот подход не исполняет любое управление данными; он не пригоден для генерации хорошего кода для NUMA-архитектур.

Альтернативный подход, реализованный в языке Фортран-D, состоит в том, чтобы дать программисту управление тем, как структуры данных распределяются по процессорам. Компилятор использует информацию о *декомпозиции данных* для определения того, как распределять работу по процессорам. Один простой способ сделать это — использовать так называемое правило *собственности* — процессор исполняет оператор присваивания, если левосторонняя переменная оператора отображается в локальную память этого процессора. Процессор исполняет итерацию цикла, если он способен выполнить любую работу в теле цикла на этой итерации. Хотя эта стратегия принимает в расчет отображения данных, компилятор может генерировать неэффективный код, в котором все процессоры исполняют все итерации “разыскивая работу для выполнения”, если структура гнезда циклов не соответствует распределению данных. Во многих таких случаях реструктуризация цикла может улучшить качество кода, но никакой общий подход к преобразованиям цикла недопустим в этом контексте.

2.1. Реструктуризация циклов

В работе [1] представлен систематический подход к реструктуризации циклов для параллельных машин с иерархией памяти. Как и в под-

ходе, основанном на собственности, наша стартовая точка есть язык типа Фортран-D со специфицированной пользователем декомпозицией данных. Однако прежде чем использовать эту информацию непосредственно для генерации кода, мы используем информацию о распределении данных для управления *нормализацией доступа*, которая представляет собой метод реструктуризации цикла, включающий в себя перестановку циклов, скашивание цикла (loop skewing), обращение цикла (loop reversal) и масштабирование цикла (loop scaling). Цель рестуктуризации — преобразовать гнезда циклов так, чтобы код мог бы быть сгенерирован распределением итераций самого внешнего цикла по процессорам без компроментации локальности. Структура внутренного цикла выбирается так, что данные могут быть перенесены с использованием, где это возможно, блока передачи (block transfers).

Указанная статья содержит следующие результаты.

- Описан новый метод, называемый *нормализацией доступа*, обеспечивающий компиляцию программ для параллельных машин с неоднородным доступом к памяти.
- Наши преобразования циклов выражаются в рамках *обратимых* матриц и теории целочисленных решеток, что является важным обобщением шаблона Банержи для работы с унимодулярных матриц. Это обобщение интересно само по себе и имеет приложения в других областях.

2.2. Пороговое планирование графа заданий

Оптимальное планирование в его наиболее общей форме — NP-трудная проблема. Известно несколько схем планирования. В работе [2] рассматривается проблема статического планирования бесконтурного графа заданий с ненулевыми вершинной и реберной стоимостями для машин с распределенной памятью.

Исходя из теоретической сложности проблемы планирования, предлагаются различные эвристики, которые обеспечивают получение ответа за полиномиальное время, но не гарантируют оптимальность. Эвристики могут сравниваться по их близости к оптимальному решению и вычислительной сложности.

Известные подходы не касаются сущности отображения заданий на машины с распределенной памятью как компромисса между длиной расписания и числом требуемых процессоров. Эта сущность приобрела важность из-за доступности систем с распределенной памятью, по-

добных Intel Paragon, который имеет низкую коммуникационную стоимость и большое число процессоров. В работе [3] разработана схема планирования времени компиляции, которая порождала масштабируемый (scalable) код для различного числа процессоров, для параллельного функционального языка Sisal [4,5]. Планировщик времени компиляции стремится значительно уменьшить длину расписания для систем с большим числом доступных процессоров. Для систем с малым числом доступных процессоров схема пытается сократить, насколько это возможно, число требуемых процессоров. Для обеих целей вначале предполагается бесконечное число процессоров при реализации планирования во время компиляции. Однако если во время прогона число доступных процессоров меньше, чем число требуемых процессоров, то предпрогоночный планировщик перестраивает расписание, подгоняя его к числу доступных процессоров. Наша стратегия планирования применима в общем случае для отображения функционального параллелизма в любом языке для машин с распределенной памятью.

Другой важной стороной дела является стоимость разбиения программ и планирования для семейства систем с распределенной памятью, которые имеют одну и ту же стоимость вычислений, но различную стоимость коммуникаций. Разница в стоимости коммуникаций может быть атрибутирована как разница соответствующих архитектур, аппаратуры маршрутизации и алгоритмов. Примером таких популярных систем служит семейство систем Intel iPSC на базе микропроцессоров i860. Эти системы имеют одну из следующих трех конфигураций:

- Intel Gamma — гиперкуб;
- Intel Delta — сеть (mesh);
- Intel Paragon — сеть с микроядрами OSF/1.

Таким образом, можно увидеть, что стоимость вычислений на этих системах одна и та же, так как все они базируются на микропроцессорах i860, но коммуникационная стоимость убывает в указанном выше порядке. Коммуникационная стоимость на системе Delta меньше, чем на системе Gamma главным образом из-за лучшей архитектуры, тогда как лучшая аппаратура маршрутизации служит причиной того, что коммуникационная стоимость в Paragon меньше, чем в Delta.

В работе [2] показано, что для крупнозернистых заданий, каковыми являются, например, вершины графа IF-2 для языка SISAL, можно в определеной степени избежать разбиения программ и регенерации расписания при компиляции для семейства систем, имеющих равные

вычислительные, но различные коммуникационные стоимости.

Имеет смысл сделать некоторые предположения относительно исполнения графов заданий Sisal IF-2 на целевой машине Intel iPSC/860.

1. Задания строгие (или, другими словами, задание не может начать исполнение, пока все входы не станут доступными), невытесняемые (non-preemptive) и имеют небольшие накладные расходы. Эти ограничения определяются семантикой Sisal.
2. Значения обмениваются между двумя процессорами в виде сообщений, используя блокирующие вызовы send() и receive(). Это предположение сделано специально для iPSC/860.
3. При реализации планирования во время компиляции предполагается бесконечное число процессоров.

2.3. Унифицирующие преобразования данных и управления

Большинство машин с общей памятью, базирующейся как на аппаратуре, так и на ПО, полагаются на кэширование данных для эксплуатации их локальности и на уменьшение коммуникаций. Когерентность кэша должна быть поддержана для многих копий одних и тех же данных на многих процессорах. Сейчас существует большое число работ в области когерентных протоколов в крупномасштабных мультипроцессорах. Эти протоколы варьируются от чисто аппаратной до программной реализации эмуляции общей памяти для машин с посылкой собщений. Существуют гибридные методы, которые содержат в себе и миграцию, и удаленные ссылки на NUMA-машинах. Для уменьшения стоимости когерентности данных и коммуникаций должна использоваться локальность данных.

Когда локальность эксплуатируется только с помощью времени прогона, ядер и политики уровня аппаратуры, которые наблюдают поведение программы снизу, то ложное общее использование (false sharing) становится важной проблемой. Неформально, ложное общее использование может быть описано следующим образом: два или более процесса осуществляют доступ к неперекрывающимся областям одного и того же когерентного блока (по крайней мере один из них пишет), вызывая ненеобходимый когерентный трафик и перемещение данных. Ложное общепользование является серьёзной помехой для высокой производительности машин с распределенной общей памятью.

В работе [6] предлагается алгебраическое представление отображений данных, модели локальности данных, новый алгоритм преобразо-

вания данных для локальности, а также унифицированный подход к улучшению локальности с преобразованиями данных и управлению для машин с общей распределенной памятью. Экспериментальные результаты, использующие множество приложений на параллельных машинах, показывают, что новые оптимизации значительно улучшают производительность.

3. БЛИЗКИЕ РАБОТЫ

Из многочисленных работ, касающихся машин с распределенной общей памятью, выделим следующие, имеющие непосредственное отношение к реструктуризации циклов и преобразованиям данных и управления: [7–13].

СПИСОК ЛИТЕРАТУРЫ

1. **Li Wei, Pingali K.** Access normalization: loop restructuring for NUMA compilers: Techn. Rep. / Cornell Univ., Comp. Sci. — N TR 92-1278. — Ithaca, 1992.
2. **Pande S., Psarris K.** A compilation technique for varying communication cost NUMA architectures // Proc. 6th Intern. PARLE Conf. Athens, Greece, 1994. — Berlin a.o.: Springer-Verlag, 1994. — P. 49–60. — (Lect. Notes Comput. Sci.; Vol. 817).
3. **Pande S., Agrawal D.P., Mauney J.** A fully automatic compiler for distributed memory machines // Proc. of the 26th Hawaii Intern. Conf. on System Sciences, January 1993. — P. 536–545.
4. **Евстигнеев В.А., Городняя Л.В., Густокашина Ю.В.** Язык функционального программирования SISAL // Интеллектуализация и качество программного обеспечения. — Новосибирск: ИСИ СО РАН, 1994. — С. 21–42.
5. **Густокашина Ю.В., Евстигнеев В.А.** IF1 — промежуточное представление SISAL-программ // Проблемы конструирования эффективных и надежных программ. — Новосибирск: ИСИ СО РАН, 1995. — С. 70–78.
6. **Cierniak M., Li Wei.** Unifying data and control transformations for distributed shared-memory machines: Techn. Rep. / Univ. of Rochester, Comp. Sci. — N 542. — Rochester, 1994.
7. **Balasundaram V., Fox G., Kennedy K., Kremer U.** An interactive environment for data partitioning and distribution // Proc. 5th Distributed Memory Comput. Conf., April 1990.
8. **Hudak D., Abraham S.** Compiler techniques for data partitioning of sequentially iterated parallel loops // Proc. ACM Intern. Conf. Supercomputing, June 1990.
9. **Knobe K., Lucas J., Steele G.** Data optimization: allocation of arrays to reduce communication on SIMD machines // J. of Parallel and Distrib. Computing. — Vol. 8, Feb. 1990. — P. 102–118.
10. **Li J., Chen M.** Index domain alignment: minimizing cost of cross-referencing between distributed arrays: Techn. Rep. / Yale Univ., 1989.

11. **Ramanujam J., Sadayappan P.** Compile-time techniques for data distribution in distributed memory machines // IEEE Trans. on Parallel and Distrib. Systems. — 1991. — Vol. 2, Oct. — P. 472–482.
12. **Wolf M., Lam M.** A data locality optimizing algorithm // Proc. ACM SIGPLAN 91 Conf. on Progr. Lang. Design and Impl., June 1991. — P. 30–44.
13. **Gannon D., Jalby W., Gallivan K.** Strategies for cach and local memory management by global program transformations // J. of Parallel and Distrib. Comp. — 1988. — Vol. 5. — P. 587–616.