Технологии компилятора, которые демонстрируют возможности К компьютера

•Koutarou Taki •Manabu Matsuyama •Hitoshi Murai •Kazuo Minami
FUJITSU Sci. Tech. J., Vol. 48, No. 3, pp. 317–323 (July2012)

Мы разработали SPARC64 VIIIfx- новый процессор для построения огромной вычислительной системы масштаба 10 PFLOPS. Для того, чтобы наилучшим образом использовать особенности этого процессора, мы разработали комплект языков программирования под названием "Parallel navi Technical Computing Language". Данная статья представляет компиляторы Fortran/C/C++ , включенные в комплект языков. В этих компиляторах мы расширили функции оптимизации для последовательной обработки (последовательная оптимизация) и функцию в компиляторах для автоматической генерации кодов потоков параллельной обработки (автоматическое распараллеливание) для выявления всего лучшего в SPARC64 VIIIfx. Кроме того мы предоставили модель гибридного параллельного исполнения, которая сочетает параллельное выполнение потоков с параллельным выполнением процессов для реализации высокопроизводительного исполнения в крупномасштабной системе. Эта модель поддерживает самые последние спецификации языков, соответствующие промышленным стандартам и, следовательно, это позволило нам компилировать более широкий диапазон программ.

1. Введение

Спрос на крупномасштабные, ультра высокоскоростные вычисления в области научных расчетов постоянно растет. В рамках национального проекта в качестве совместной разработки с Институтом физических и химических исследований (RIKEN), была поставлена цель построить систему с производительностью LINPACK в 10PFLOPS (система масштабом в десять квадрилионов [десять тысяч триллионов] операций в секунду), что почти в 100раз превосходит производительность обычных систем. Для построения такой системы мы разработали процессор с проприетарными расширениями архитектуры SPARC.

Чтобы максимизировать производительность CPU важно совершать большинство его функций. С этой целью мы адаптировали технологию компилятора тесно привязавшись к архитектурным расширениям CPU. Производительность CPU была выявлена за счет внедрения новой технологии компилятора в дополнение к использованию существующей технологии оптимизации.

К компьютер[note1)], который был поддержан Министерством образования, культуры, спорта, науки и технологий Японии и разрабатывался совместно RIKEN и Fujitsu, является грандиозной системой с более чем 80 000 соединенными вместе мультиядерными процессорами. Для достижения высокой эффективности выполнения в такой большой системе мы предоставили поддержку гибридной модели исполнения объединением одновременно исполняемых потоков и процессов.

Данная статья представляет технологии, приспособленные для компиляторов используемых К компьютером и наши подходы к улучшению их работы. В следующих разделах приводится описание, сосредоточенное на 1) описании языка (поддержка новых стандартов и промышленных нормативов), 2) аппаратных свойств, 3) оптимизации последовательной обработки, которая применяет SPARC64 VIIIfx, 4) технология автоматического распараллеливания и модель гибридного параллельного исполнения, 5) функция отладки.

2. Особенности, связанные со спецификацией языка программирования

Для К компьютера мы разработали три компилятора: Fortran, C и C++. При разработке этих компиляторов мы обратили внимание на следующие моменты, связанные со спецификацией языка программирования.

1) Применение новых стандартов

Стандарты языков программирования не являются неизменными. Они регулярно пересматриваются каждые несколько лет для того, чтобы улучшить характеристику эффективности на основе прошлого опыта технологии программирования. Важно поддерживать новые стандарты, чтобы уменьшить все время исследований, включая период разработки прикладной программы.

note1) "K computer" - английское название, которое RIKEN использовал для суперкомпьютера в данном проекте начиная с июля 2010. "K" пришло из японского слова "Kei," которое обозначает 10пета или 10 в 16й степени.

2) Поддержка промышленных нормативов

При современной разработке программ обычно используется хорошо разработанное программное обеспечение с открытым исходным кодом (OSS, open-source software), тем самым снижая человеко-часы разработки для того чтобы сосредоточиться на сути проблемы, которую необходимо решить. Соответственно, важное значение имеет поддержка стандартных промышленных спецификаций используемых в OSS.

3) Оптимизация аппаратных средств

Важно иметь библиотеки и механизмы оптимизации, которые позволяют максимизировать производительность целевой платформы. Мы должны предложить соответствующую поддержку новых функций, внедренных в SPARC64 VIIIfx.

Для удовлетворения этих требований мы установили следующие спецификации языка.

- Компилятор Fortran

Приняты основные стандарты Fortran 2003. Предусмотрены соответствия OpenMP 3.0.

- Компилятор C

Предусмотрены соответствия стандарту C99. Принята поддержка некоторых спецификаций расширения языка C, реализованных в компиляторе GNU C версии 4.1.2. Готовые встроенные функции, способные непосредственно обрабатывать инструкции SIMD (single instruction multiple data, одиночная инструкция над множеством данных), добавленные в HPC-ACE (High Performance Computing - Arithmetic Computational Extensions, арифметические расширения высокопроизводительных вычислений). Предусмотрены соответствия OpenMP 3.0.

- Компилятор C++

Предусмотрены соответствия стандарту C++03. Готовое расширение спецификации GNU и встроенных функций, способные непосредственно обрабатывать инструкции SIMD, эквивалентные описанным выше для компилятора C.Предусмотрены соответствия OpenMP 3.0.

Для трех описанных выше компиляторов, мы подготовили BLAS, LAPACK и ScaLAPACK, математические библиотеки, являющиеся промышленным стандартом, оптимизированные для K компьютера и собственные библиотеки SSL II, C-SSL II и SSL II/MPI.

В качестве средства межузловой параллелизации мы предоставили XPFortran, собственный язык параллельного программирования Fujitsu, дополнительно к библиотеке MPI (Message Passing Interface). XPFortran имеет грамматику Fortran, расширенную директивами и позволяющий гибридное параллельное исполнение (автоматические параллельные потоки и MPI параллельные процессы) объединяя последовательную оптимизацию и автоматическое распараллеливание, которые описываются ниже.

3. Аппаратные свойства

Данный раздел описывает HPC-ACE, набор расширений SPARC64 VIIIfx, и VISIMPACT (Virtual Single Processor by Integrated Multicore Parallel Architecture, виртуальный одиночный процессор с интегрированной многоядерной параллельной архитектурой), который уже использовался для архитектуры Fujitsu SPARC.

3.1 HPC-ACE

HPC-ACE является архитектурой, основанной на обычной SPARC архитектуре, расширенной для суперкомпьютеров. HPC-ACE включает следующие четыре расширения.

1) Регистр расширения

По отношению к обычной SPARC архитектуре, число целочисленных регистров было увеличено до 188 со 156, а число регистров для чисел с плавающей запятой до 256 с 32. Это обеспечивает возможность для улучшения параллелизма на уровне команд и уменьшает число сохранений данных и их восстановлений из памяти, являющихся результатом недостаточного числа регистров, что в итоге влечет к более быстрым приложениям.

2) Добавление инструкций SIMD

SIMD обеспечивает механизм для выполнения одной операции на нескольких фрагментах данных одновременно одной инструкцией. HPC-ACE позволяет одновременное выполнение двух операций одной инструкцией.

3) Добавление новых инструкций.

Были представлены новые инструкции, часто используемые в научных расчетах, включая инструкции обратных аппроксимаций, вспомогательных тригонометрические функций, замены по маске и значения максимума/минимума. Инструкции обратных аппроксимаций не прерывают конвейерное выполнение, в результате высокая вычислительная производительность может быть сохранена. В простой программе деления использующей инструкцию обратной аппроксимации время исполнения сокращается до одной трети или менее. Инструкции замены по маске позволяют уменьшить условные переходы (операторы IF) при выполнении цикла, что ускоряет оптимизацию выполнения цикла.

4) Секторизованный кэш.

Секторизованный кэш является функцией, в которой кэш разделен на два поля чтобы отделить повторно используемые данные от временных данных, что позволяет в максимальной степени кэшировать повторно используемые данные. Это увеличивает соотношение попадания в кэш и позволяет повысить скорость выполнения приложения.

3.2 VISIMPACT

VISIMPACT является архитектурой, которая позволяет пользователю работать с множеством ядер как с одним высокоскоростным процессором. Этот механизм может использоваться для ускорения выполнения параллельных потоков и выполнения гибридного распараллеливания в многоядерных процессорах.

Две важные технологии были реализованы с использованием VISIMPACT.

1) Совместно используемый L2 кэш

Структура разделения L2 кэш-памяти между всеми ядрами процессора уменьшает воздействие неверного совместного использования[note2)] и подавляет деградацию производительности приложений.

2) Межъядерный аппаратный барьер

Для обеспечения параллельного выполнения межъядерных потоков может быть необходима синхронизация между потоками. Выполнение межъядерного аппаратного барьера приблизительно в десять раз быстрее, чем выполнение барьера программными средствами; даже когда мал масштаб задачи (мелкозернистый параллелизм), в результате может быть достигнута высокая эффективность выполнения параллельных межъядерных потоков.

4. Технология последовательной оптимизации

Данная статья описывает технологию последовательной оптимизации, одно из усовершенствований компиляторов для аппаратных возможностей, упомянутых в предыдущем разделе.

4.1 Регистр расширения и SIMD

Регистры, добавленные HPC-ACE, могут использоваться совместно с регистрами, определенными для SPARC-V9. Это означает, что число регистров, выделяемых компиляторами, просто увеличивается.

В то время, как не-SIMD инструкции могут получить доступ ко всем произвольно выбранным регистрам, инструкции SIMD выполняют одновременную операцию на паре из основного и дополнительного регистров. Существует ограничение, что номер дополнительного регистра фиксируется ограничением в пределах от соответствующего основного регистра плюс 256. Например, инструкция SIMD операции одновременно выполняется на f[0] и f[256]. Из-за этого ограничения анализ и оптимизация зависимостей усложняются в случае, когда перемешаны вместе SIMD и не-SIMD инструкции. Тем не менее эффект выделения регистров с минимальными потерями остается важным. Например, если не-SIMD инструкция выделяет f[0] и f[2],число регистров, которое можно использовать SIMD инструкцией уменьшается на две, однако, если выделяются f[0] и f[256] уменьшение в числе регистров, которое может использовать инструкция SIMD составит только единицу.

Когда существуют некие ограничения на выделяемые регистры, данные обычно пересылаются в регистры, удовлетворяющие ограничению, однако число пересылок сводится к минимуму для повышения производительности исполнения. Однако, если выделение регистров выполняется одновременно с оптимизацией через сведение к минимуму количества инструкций пересылки, то сложность обработки возрастает, а это значит, что для компиляции требуется много времени и памяти.

Мы выбрали подход, при котором инструкции пересылки генерируются до выделения регистров, для того, чтобы выделять одни и те же регистры до и после передачи данных как можно чаще. Инструкции передачи данных в одни и те же регистры до и после передачи могут быть удалены после распределения регистров, тем самым избегается генерация и исполнение большего, чем требуется, числа инструкций. Таким образом было достигнуто оптимальное распределение регистров в практичных пределах значений времени и памяти.

4.2 Выявление параллелизма уровня инструкций

Параллелизм на уровне инструкций является индикатором того, как много работы может быть выполнено параллельно в последовательности инструкций. Больший параллелизм означает, что вычислительные блоки простаивают в течение более короткого времени и сильно заняты, что приводит к ускорению выполнения приложений. Компилятор использует оптимизацию, такую как планирование инструкций и программную конвейеризацию для изменения порядка и генерации последовательности инструкций с целью повышения более высокого параллелизма. При улучшении параллелизма уровня инструкций, однако, одновременно возрастает потребность во временном удержании данных в регистрах, что в результате приводит к большему числу используемых регистров. По этой причине максимальный параллелизм на уровне инструкций может не всегда достигаться на обычной архитектуре.

При использовании HPC-ACE, большее число регистров позволяет решительно применять планирование инструкций и программную конвейеризацию. Сочетание такой оптимизации с инструкциями обратных приближений и маскируемых подстановок привело к достижению еще более высокого чем ранее параллелизма на уровне инструкций.

[note2)] В системе, состоящей из множества ядер с соответствующими кэшами, когда не разделяемые данные индивидуальные для ядер находятся в одной строке кэша, некорректное совместное использование представляет собой явление, в котором запись в соответствующие данные приводит к тому, что строка кэша признается недействительной и происходит обмен данными между соответствующими кэшами.

4.3 Эффект HPC-ACE

Мы оценили производительность последовательной оптимизации компилятора, который использует регистр расширения и инструкции SIMD, являющиеся характерными особенностями HPC-ACE. Для выполнения этого мы использовали 146 программ оценки производительности на Fortran-e, разработанных нашим подразделением (Рисунок 1). Вертикальная ось на рисунке показывает соотношение производительности (скорость улучшения) по отношению к производительности при выделении только основных регистров и игнорировании SIMD инструкций, используемое как единичное значение производительности. Результат показывает, что подтверждается улучшение производительности в 1.26 раз в среднем и в 4.03 раза в максимуме только за счет использования расширенного набора регистров (без использования инструкций SIMD). При одновременном использовании и расширенного набора регистров и инструкций SIMD подтверждается и улучшение в среднем в 1.66 раз и в 5.62 раз в максимуме.

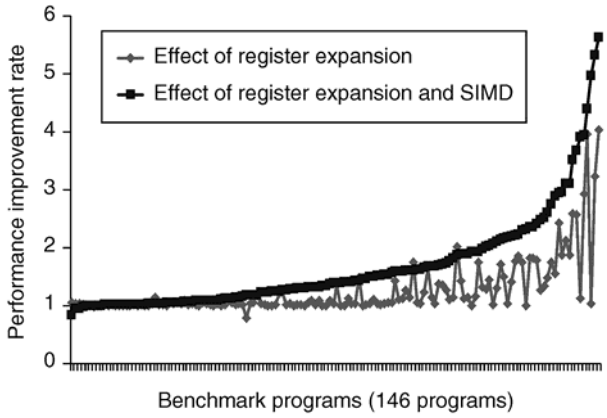

Рисунок 1
Результаты регистра расширения и SIMD

5. Технология автоматического распараллеливания и модель выполнения

Из всех реализованных для улучшения автоматического распараллеливания технологий, в данной статье описываются приватизация переменных, использование функции аппаратного барьера и модель исполнения, которая сочетает параллельные потоки и параллельное выполнение процессов (модель гибридного распараллеливания).

5.1 Приватизация переменных

Приватизация переменных внутри цикла позволяет автоматическое распараллеливание циклов, которое не было доступно в прошлом, и расширение масштабов распараллеливания, которое приводит к улучшению скорости распараллеливания. С помощью этой функции становится возможным распараллеливание для всех циклов с инструкциями OpenMP в эталоне NAS Parallel (Таблица 1). То есть с программами типа эталона NAS Parallel не требуется ручной анализ программ и директив OpenMP при том, что эквивалентная распараллеливание может быть выполнена с помощью автоматического распараллеливания.

Таблица 1
Производительность автоматического распараллеливания с NPB 3.3.

	Число циклов с OpenMP	**Число циклов, подлежащее автоматическому распараллеливанию**
CG	25	25
FT	13	13
MG	22	22
BT	38	38
SP	74	74
LU	64	64

5.2 Использование функции аппаратного барьера

Для уменьшения накладных расходов до 350нс с 750нс в библиотеке времени исполнения был использован расширенный в SPARC64 VIIIfx функция аппаратного барьера при потоковом распараллеливании. Это улучшение сделало возможным распараллеливание малых циклов, которые не распараллеливались ранее, т.к. традиционные технологии не достигали эффекта распараллеливания, при этом одновременно улучшая производительность выполнения.

5.3 Модель гибридного параллельного выполнения

Для параллельного приложения, которое выполняется на PC кластерах и тому подобном оборудовании, часто используется одноуровневый MPI (модель параллельного выполнения, которая также использует MPI для параллельного выполнения между ядрами), который также применяется для многоядерных конфигураций. При использовании одноуровневого MPI, число одновременно выполняемых процессов возрастает по мере того, как увеличивается число ядер процессоров, используемых параллельным приложением. Использование буферной памяти для обмена данными между параллельными процессами и число требуемых связей возрастает в соответствии с числом элементов на другой стороне обмена или параллельных процессов. Это ограничивает объем памяти, который можно использовать в приложении. Кроме того ограничение на количество буферов обмена данными может вызывать снижение производительности обмена данными. По этой причине может быть уменьшена эффективность выполнения.

Для К компьютера, который имеет целью достижения высокой производительности посредством сверхвысокого распараллеливания, обеспечивается гибридная модель распараллеливания: параллельное выполнение процессов используется для межузлового распараллеливания, в то время как потоковое параллельное выполнение которое использует VISIMPACT для внутриузлового распараллеливания. Таким образом, увеличение числа параллельных процессов сводится к минимуму, чтобы избежать уменьшения объема используемой памяти в которой могут быть выполнены параллельные процессы (Рисунок 2).

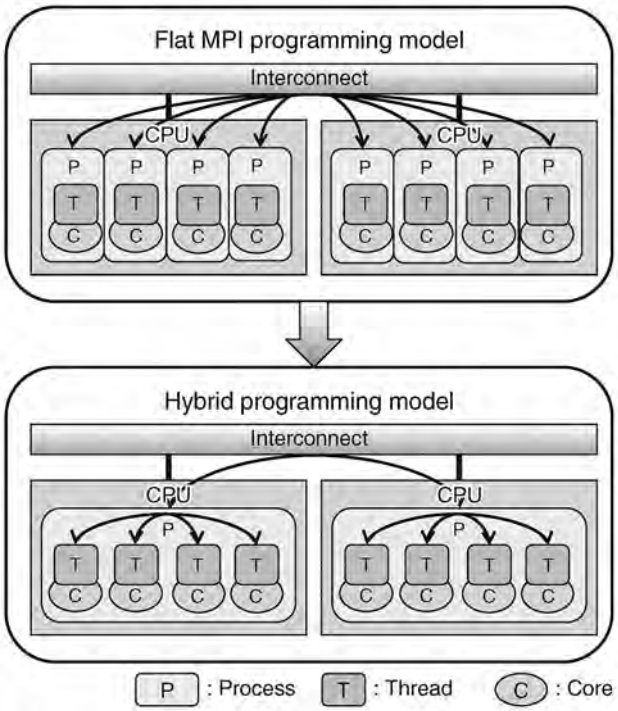

Рисунок 2
Модель гибридного распараллеливания

6. Функция отладки

В дополнение к отладке с помощью обычных инструментов, компиляторы Fujitsu обеспечивают функцию проверки времени исполнения, которая проверяет программу во время компиляции, а также работает и выдает предупреждения, если существуют отклонения от спецификации языка. Данная функция позволяет разработчикам приложений определять до некоторой степени ошибки при трансляции и выполнении программ. Для существующих трансляторов, однако, разрешение такой функции проверки приводит к увеличению времени исполнения в диапазоне от десятков до нескольких сотен раз.

Для компиляторов К компьютера эта функция была пересмотрена и реализована в виде "встроенной функции отладки" которая работает на высокой скорости. Эта встроенная функция отладки проводит минимум проверок таких, как проверки параметров процедур и диапазонов индексов. Для обнаружения неопределенных переменных была разработана функция, в котором в качестве начального значения устанавливается значение NaN (*not a number*) и любая недопустимая операция определяется как исключение. Была сделана выбираемой по желанию упрощенная функция, которая в качестве сообщений об ошибках выдает только номера строк и типов ошибок или детализированная версия, которая дополнительно выводит имена переменных и процедур.

Разработанная встроенная функция отладки доступна в компиляторах C/C++ в дополнение к Fortran и может быть объединена с потокового распараллеливания (OpenMP и автоматическое распараллеливание).

С помощью программы на Fortran-e, разработанной в нашем подразделении, была сделана оценка влияния функции отладки на производительность выполнения (Рисунок 3). В качестве опции оптимизации была определена -Kfast. Было установлено, что использование новой встроенной функции отладки требует [1)] в 8.4 раз больше времени для проверки ошибок + детализированный вывод сообщений, 2) в 5.6 раз больше времени для проверки на ошибки с NaN + детализированный вывод сообщений и 3) в 2.7 раз больше времени для проверки на ошибки с NaN + упрощенный вывод сообщений. По сравнению с существующими функциями отладки, которые требовали превышения времени в диапазоне от нескольких десятков до нескольких сотен раз, производительность существенно улучшилась и отладка требует только в 2.7 раз больше времени для кратчайшей отладки, что, как мы полагаем, достигло практического уровня.

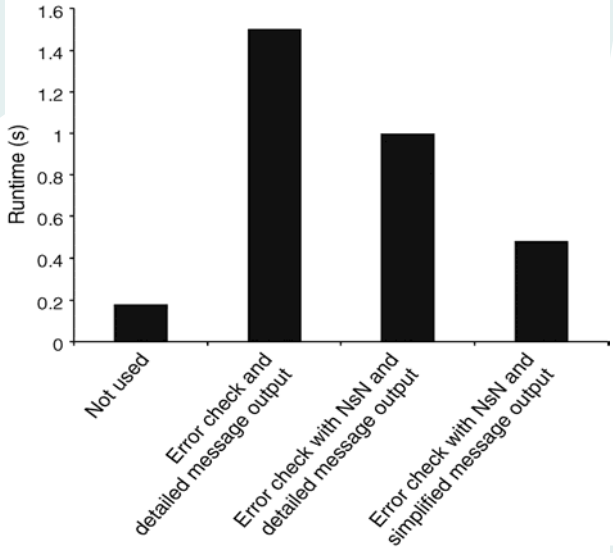

Рисунок 3
Время выполнения новых встроенных функций отладки

7. Заключение

Для разрешения выполнения потокового распараллеливания компиляторы Fujitsu имеют давно поддерживаемое автоматическое распараллеливание, к которому применяется функция векторизации компиляторов для VPP машин. Благодаря накоплению на протяжении лет технологий, изменения вызванные увеличением количества ядер до восьми в SPARC64 VIIIfx незначительно, и это позволило нам сосредоточиться на повышении автоматического распараллеливания самого по себе и успешно достигнуть высокой эффективности распараллеливания. Fujitsu заявила 16-ядерную архитектуру SPARC64 IXfx (уже реализована- прим.перев.), преемника SPARC64 VIIIfx, и уже начали предлагать PRIMEHPC FX10, систему, содержащую данный CPU (см., например систему Университета Токио OakLeaf прим.перев.). Наше подразделение реализовало технологию компиляции, которая была коммерциализована пакетом языков для К компьютера, а также в продукте для PRIMEHPC FX10, называемом Technical Computing Suite.

Спрос на более быстрые суперкомпьютеры постоянно растет, а расширение и распространение спецификаций оборудования будет неизбежно продолжаться и в будущем. Мы можем ожидать, что будущие высокопроизводительные суперкомпьютеры должны иметь даже больше ядер. В конечном итоге станут необходимыми фундаментальные прорывы. Основываясь на успешном опыте по занятию К компьютером первой позиции в мировом рейтинге, мы намерены преодолеть эти проблемы и продолжать предлагать системы и продукты, отвечающие потребностям пользователей.