Обзор аппаратных и программных средств реализации параллельной обработки

Проблема организации параллельных вычислений возникла уже очень давно, но до сих пор не существует универсальных методов ее решения. Обзор средств реализации параллельной обработки позволяет сделать некоторые интересные выводы.

Михаил Долинский,
Алексей Толкачев

dolinsky@gsu.unibel.by

Существующие подходы к аппаратной реализации параллельной обработки

Основными широко применяемыми архитектурами вычислительных систем с распределенными ресурсами являются многопроцессорные системы, вычислительные кластеры, векторные процессоры, VLIW-процессоры, суперскалярные процессоры и процессоры, ориентированные на конкретную задачу.

1. Многопроцессорные системы

Многопроцессорные вычислительные системы представлены двумя типами. Первый, наиболее распространенный — это многопроцессорные серверы с общей памятью. Ведущие производители выпускают многопроцессорные серверы, стремясь предоставить пользователям программное окружение, доступное в среде традиционных однопроцессорных компьютеров. Типичными представителями данного класса являются системы SMP (symmetric multiprocessors) — симметричные мультипроцессоры, в которых все процессоры обладают равной производительностью, являются равноправными при доступе к общей памяти и время доступа к памяти является одинаковым.

Второй тип многопроцессорных систем — параллельные суперкомпьютеры MPP (massive parallel processing) с большим количеством процессоров и разделяемой памятью. Использующие подобную модель суперкомпьютеры построены как массив отдельных машин (узлов), взаимодействующих через высокоскоростные каналы связи. Каждый узел получает доступ только к локальной памяти, выполняемое параллельное приложение оказывается разделенным на ряд параллельно выполняемых слабо взаимодействующих процессов, обменивающихся информацией путем передачи и приема сообщений.

Системы обоих классов часто строятся из тех же микропроцессоров, на базе которых выпускаются персональные компьютеры и рабочие станции.

2. Вычислительные кластеры и вычислительные сети

Вычислительные кластеры представляют собой системы, состоящие из множества узлов, связанных коммуникационной средой. В качестве узлов могут использоваться компьютеры в составе локальной или глобальной сети. Каждый узел имеет локальную память, общей оперативной памяти нет. В составе кластера могут быть узлы с различной архитектурой и производительностью. В случае, когда все узлы кластера имеют одинаковую архитектуру и производительность, кластер называют однородным, иначе — неоднородным.

Архитектура вычислительного кластера подобна суперкомпьютерам MPP, кластеры часто используются в качестве их дешевой альтернативы, поскольку могут быть построены на базе уже имеющихся в организации персональных компьютеров. Любой кластер можно рассматривать как единую аппаратно-программную систему, имеющую единую коммуникационную систему, единый центр управления и планирования загрузки. Часто в целях уменьшения стоимости кластера в качестве узлов используются доступные в данный момент компьютеры, имеющие разные характеристики, и, возможно, частично загруженные решением других задач.

Вычислительные сети (GRID) объединяют ресурсы множества кластеров, многопроцессорных и однопроцессорных компьютеров, имеющих различное географическое расположение, принадлежащих разным организациям и подчиняющихся разным дисциплинам использования. Особая разновидность систем данного класса — глобальные вычислительные сети, использующие Интернет в качестве коммуникационной среды.

Программное обеспечение неоднородных кластеров и вычислительных сетей должно обеспечивать адаптацию к динамическому изменению конфигурации, учитывать неоднородность архитектуры и возможность сбоев отдельных узлов.

Особенностью реализации параллельной обработки является то, что обрабатываемая информация должна распределяться между узлами относительно большими порциями, поскольку интервалы времени, необходимые для передачи информации между узлами, обычно на несколько порядков превосходят время передачи в пределах одного узла.

3. Векторные процессоры

Альтернативное название — процессоры SIMD-архитектуры. Имеется один поток команд, содержащий векторные инструкции. Одна векторная инструкция выполняет определенную арифметическую операцию одновременно для всех элементов вектора данных.

Процессоры различных архитектур могут иметь векторные инструкции для ускорения выполнения некоторых операций. Векторные процессоры применяются в некоторых суперкомпьютерах. Недостатком векторных процессоров является их высокая стоимость.

4. Использование VLIW-технологии

Процессоры с длинным словом инструкции (Very Long Instruction Word, VLIW) реализуют простейший способ параллельной обработки на уровне отдельных инструкций. Каждая команда VLIW-процессора может состоять из нескольких параллельно исполняемых инструкций, работающих с разными данными.

При использовании VLIW-процессоров распараллеливание алгоритма осуществляется либо компилятором на этапе компиляции программы, либо программистом при программировании на ассемблере. Архитектура процессоров данного типа накладывает существенные ограничения на возможность параллельного исполнения, поскольку в них невозможно создание нескольких параллельно исполняемых потоков инструкций. Фактически существует только один поток инструкций, но каждая из них может содержать несколько команд обработки, выполняемых одновременно.

5. Суперскалярные процессоры

Исполняемый код суперскалярных процессоров обычно не содержит информации о параллельной обработке. Распараллеливание инструкций происходит на этапе исполнения программы. Для этого в процессоре присутствует специальный блок анализа. Архитектура таких процессоров значительно сложнее VLIW-архитектуры, поэтому они редко применяются при проектировании встроенных систем.

Наиболее часто такие процессоры используются в качестве более производительных моделей, совместимых с предыдущими, не имевшими возможностей параллельной обработки. Например, процессоры компании Intel, начиная с Pentium, относятся к этому классу. На базе суперскалярных процессоров построены практически все современные персональные компьютеры и многопроцессорные системы.

6. Процессоры, ориентированные на конкретную задачу

Разрабатываются специально для решения некоторой задачи. Параллельная обработка может осуществляться на уровне аппаратной реализации некоторых алгоритмов.

Разработка таких процессоров осуществляется с помощью языков описания аппаратуры, например, VHDL или Verilog.

Недостатком такого подхода является сложность проектирования и отладки, а также отсутствие высокоуровневых средств разработки программного обеспечения для полученного процессора.

7. Сравнение эффективности и возможности применения различных методов построения параллельных систем

Вычислительные системы с распределенными ресурсами применяются для решения самых различных задач.

Многопроцессорные суперкомпьютеры различных архитектур, вычислительные кластеры и вычислительные сети применяются для решения задач, требующих огромных объемов вычислений.

Многопроцессорные SMP-серверы также широко используются для обработки данных в реальном времени, например, в качестве веб-серверов.

Другое направление использования параллельной обработки — увеличение производительности отдельных микропроцессоров. Универсальные микропроцессоры, используемые в персональных компьютерах и многопроцессорных системах, имеют суперскалярную архитектуру. В некоторых суперкомпьютерах применяются и векторные процессоры.

Отдельно следует отметить процессоры, используемые во встроенных системах. При выборе процессора для встроенной системы особое внимание уделяется производительности, стоимости и потребляемой энергии. Параллельная обработка часто позволяет наиболее эффективно использовать аппаратные ресурсы. Поэтому во встроенных системах часто применяются VLIW-процессоры, а также специализированные решения, аппаратно реализованные с использованием ПЛИС или СБИС.

Универсальные средства разработки программного обеспечения вычислительных систем с распределенными ресурсами

1. Система программирования на основе передачи сообщений MPI

Библиотека MPI (Message Passing Interface, интерфейс передачи сообщений) является самым распространенным средством программирования для параллельных вычислительных систем с разделяемой памятью, в частности, вычислительных кластеров и многопроцессорных суперкомпьютеров с разделяемой памятью.

В вычислительной модели MPI программа является множеством процессов, каждый из которых выполняется в собственном адресном пространстве. Процессы взаимодействуют посредством передачи сообщений.

Существуют реализации MPI для языков C/C++ и Fortran 77/90 для большинства многопроцессорных суперкомпьютеров и сетей рабочих станций UNIX и Windows NT.

Библиотека MPI предоставляет разработчику множество функций для межпроцессного взаимодействия, предоставляющих возможности обмена сообщениями между процессами, выбор режимов передачи сообщений, коллективные операции, множество редукционных операций, эффективно работающих с данными на разных процессорах, возможность указания типов передаваемых данных.

Существует две версии MPI — первый стандарт был принят в 1993 году, а в 1997 был разработан проект стандарта MPI-2, расширяющий MPI и предусматривающий динамическое создание и уничтожение процессов, использование общей памяти и операции параллельного ввода-вывода.

Стандарты MPI и MPI-2 достаточно сложны и громоздки, поэтому на сегодняшний день не существует реализации MPI, полностью соответствующей стандартам.

2. Технология OpenMP для систем с общей памятью

Наиболее распространенным средством программирования многопроцессорных систем с общей памятью является технология OpenMP.

Данная технология позволяет легко добавлять использование параллельной обработки в существующие последовательные программы.

При использовании OpenMP исходный операторный код программы остается таким же, как и для последовательной архитектуры. С помощью специальных директив препроцессора разработчик указывает в исходном коде программы параллельные области, в которых один и тот же исполняемый код может быть выполнен сразу на нескольких процессорах. В параллельных блоках могут использоваться локальные и общие для всех потоков данные. Препроцессор заменяет директивы параллелизма обращениями к какой-либо библиотеке, реализующей функции параллельной обработки для конкретной архитектуры.

Таким образом, с помощью директив препроцессора программа разбивается на последовательные и параллельные участки. Данный способ реализации параллелизма получил название «вилочного» или «пульсирующего» параллелизма.

Технология OpenMP реализована в последних версиях компиляторов компании Intel для эффективного использования возможностей многопроцессорных систем, а также процессоров, поддерживающих технологию Hyper Threading.

При программировании кластеров, состоящих из многопроцессорных узлов, часто используется гибридный подход MPI + OpenMP, когда взаимодействие между узлами программируется с использованием MPI, а реализация задач на отдельных узлах осуществляется с помощью OpenMP.

3. Использование директив параллельной обработки в системе DVM

Система разработки параллельных программ DVM была разработана в Институте прикладной математики им. М. В. Келдыша РАН. Существуют две реализации — C-DVM и Fortran-DVM.

Основная особенность системы DVM — использование модели параллелизма по данным и вычислениям. Программисту предоставляются следующие возможности реализации параллельной обработки:

- Объявление «распределенных» массивов, элементы которых распределяются между параллельными процессами (все остальные переменные «размножаются» по всем процессорам). Имеются директивы выравнивания массивов, то есть явного указания того, какие элементы различных массивов следует распределять на один и тот же процессор; описания различных типов удаленных данных (хранящихся на других процессорах) и др.
- Распределение витков циклов между параллельными процессорами, выравнивание витков циклов с элементами массивов (указание того, что определенные витки циклов следует выполнять на тех процес-

сорах, на которые распределены указанные элементы массивов).
- Создание параллельных задач — параллельно выполняющихся блоков программы.
- Множество редукционных операций.

Используемая модель параллелизма удобна для реализации алгоритмов численных методов решения математических задач.

Система DVM ориентирована на использование многопроцессорных систем с разделяемой памятью. Распределение данных и вычислений между параллельными процессорами осуществляется динамически системой поддержки выполнения DVM-программ. Многопроцессорной системой может служить группа MPI-процессов.

Программа DVM представляет собой последовательную программу, в которую с помощью специальных директив препроцессора добавляются спецификации параллелизма. Программа может быть скомпилирована обычным компилятором, не поддерживающим директивы DVM. Это дает возможность использовать обычные средства отладки последовательных программ. Затем параллельная DVM-программа может быть выполнена в режиме отладки, когда промежуточные результаты вычислений сравниваются с результатами, полученными при последовательном выполнении.

4. Язык mpC для программирования неоднородных вычислительных систем

Язык mpC разработан для программирования параллельных вычислений на неоднородных вычислительных кластерах.

Параллельная программа на mpC — множество параллельных процессов, взаимодействующих (синхронизирующих свою работу и обменивающихся данными) посредством передачи сообщений. Программист не может управлять тем, сколько процессов составляет программу и на каких узлах эти процессы выполняются. Это делается внешними по отношению к языку средствами. Исходный код на mpC управляет лишь тем, какие именно вычисления выполняются каждым из процессов, составляющих программу.

Параллелизм описывается в терминах процессов и сетей (групп из множества процессов). Сети могут создаваться и уничтожаться в процессе работы программы. Процессы распределяются между физическими процессорами динамически во время исполнения.

Особое внимание в mpC уделяется неоднородности вычислительной системы. Среда выполнения оценивает производительность каждого узла вычислительного кластера, на котором выполняется программа, и распределяет процессы таким образом, чтобы минимизировать общее время выполнения программы.

По умолчанию система считает, что все процессы программы выполняют примерно одинаковые объемы вычислений. Исходя из этого, процессы распределяются на узлы кластера таким образом, чтобы их количество было пропорционально производительности данного узла.

В случае, когда объемы вычислений, выполняемых различными процессами, значительно отличаются, программист имеет возможность указывать относительные объемы вычислений процессов сети.

Язык mpC имеет средства описания функций оценки производительности узлов, что позволяет замерять производительность именно на нужном типе задач. Замеры могут осуществляться динамически непосредственно перед созданием процесса. Это позволяет создавать программы, в которых вычисления распределяются в соответствии с реальной производительностью узлов на момент выполнения вычислений.

Благодаря перечисленным возможностям, mpC позволяет эффективно использовать вычислительные кластеры, построенные из компьютеров с различными характеристиками и, возможно, одновременно используемых для решения других задач.

5. Динамическое распараллеливание программ в T-системе

T-система реализует автоматическое динамическое распараллеливание программ за счет использования функциональной парадигмы: участки программы, которые могут быть распараллелены, должны быть описаны в виде «чистых» функций (T-функций). Все данные для вычислений в теле T-функции должны быть явно получены через ее аргументы, все результаты вычислений — переданы через результаты.

T-система ориентирована на решение задач, имеющих следующие характеристики:
- большой объем вычислений;
- сложная логика вычислительного процесса;
- обработка нечисловых данных или данных, имеющих сложное представление — динамически порождаемые списки, деревья, графы и др.
- наличие параллелизма, скрытого до момента исполнения программы.

Для программирования в T-системе используется язык t2cp — расширение языка C конструкциями, предназначенными для описания T-функций, передачи T-значений, вызовов T-функций и операций над T-структурами.

Основные разработчики T-системы — Институт программных систем РАН и Исследовательский центр мультипроцессорных систем. В настоящий момент T-система реализована для платформ класса «IP-сеть рабочих станций с ОС Linux» (Intel, PC, mono или SMP). T-система применяется для разработки программного обеспечения суперкомпьютера СКИФ.

Средства разработки компиляторов для систем с распределенными ресурсами

1. Программный комплекс для проектирования трансляторов GCC

Настраиваемый компилятор языков высокого уровня GCC (GNU Compiler Collection) является свободно распространяемым в соответствии с GPL (GNU Public License) с открытым исходным кодом. Изначально GCC был разработан для операционной системы UNIX и затем перенесен на множество других платформ. В настоящее время существуют front-end-компиляторы с языков C, C++, Java, Fortran, Objective C, Ada 9X, Modula-3, Pascal, Cobol и др. Поддерживается большое количество целевых платформ — Unix, Microsoft Windows всех версий, HP-UX для различных процессоров, существуют кросс-компиляторы для ARM, Atmel AVR, некоторых DSP-процессоров семейства TMS от Texas Instruments. GCC является системным компилятором большинства Unix-систем. Может использоваться как кросс-компилятор.

Процесс компиляции в GCC состоит из следующих шагов:
- Синтаксический анализ: front-end для исходного текста строит бинарное представление в виде синтаксических деревьев для каждой функции в исходном коде.
- Генерация представления функций в виде древовидных GIMPLE-структур. На этом этапе осуществляется окончательная семантическая проверка компилируемого кода. Полученное представление структур GIMPLE является независимым от исходного языка и используется компилятором на следующих шагах.
- Менеджер проходов выполняет запуск проходов компилятора в правильном порядке. Для каждого прохода имеется определенный набор структур, необходимых для его запуска. Менеджер следит за тем, чтобы все структуры были подготовлены перед запуском очередного прохода.
- Выполнение проходов оптимизации для GIMPLE-представления. Выполняется большое количество шагов, порядок выполнения которых определяется менеджером проходов (удаление неиспользуемых выражений, построение графа потока управления, оптимизация циклов, предсказание частот выполнения операторов, удаление хвостовой рекурсии, выполнение других платформо-независимых оптимизаций).
- Выполнение RTL-проходов — оптимизация на низком уровне, генерация объектного кода. На этом этапе используется описание инструкций целевой платформы.

Настройка GCC на целевую платформу состоит из двух частей: описания шаблонов инструкций, используемых в ходе выполнения RTL-проходов компиляции, и описания макросов для управления процессом компиляции.

Возможности описания шаблонов инструкций первоначально не были ориентированы на использование параллельных целевых архитектур. Несмотря на это, имеются некоторые возможности оптимизации для суперскалярных и VLIW-процессоров на уровне отдельных инструкций. Использование peephole-оптимизации позволяет дополнительно оптимизировать сгенерированный объектный код за счет изменения порядка рядом стоящих команд, замены одних последовательностей команд другими и т. д. Однако на сегодняшний день GCC нельзя назвать подходящим средством разработки компиляторов для целевых архитектур, использующих мелкозернистый параллелизм.

Основная область применения GCC для программирования вычислительных систем с распределенными ресурсами — это реализация компиляторов для многопроцессорных систем, узлов MPP-систем и вычислительных кластеров. Многие библиотеки для параллельного

программирования (MPI, OpenMP) позволяют использовать компилятор GCC, например, существует реализация GCC для SMP-компьютеров на базе процессоров Intel Itanium.

2. Настраиваемый компилятор LCC

Компилятор языка ANSI C, настраиваемый на целевую платформу. Существуют реализации для процессоров Alpha, Sparc, MIPS R3000 и семейства Intel x86, а также LCC.NET для виртуальной машины MSIL, входящей в состав Microsoft .NET.

Исходный код LCC открыт, компилятор может свободно использоваться в некоммерческих целях.

Промежуточное представление программы обеспечивает взаимодействие платформо-независимого front-end и back-end кодогенератора для целевой платформы. Интерфейс состоит из нескольких структур данных, операторного языка из 33 операторов, с помощью которого исходная программа кодируется в виде ориентированных ациклических графов (ОАГ), и 18 функций для операций с ОАГ и структурами данных. Реализация этих функций для большинства платформ очень проста.

Front-end и back-end реализованы в составе одной программы и вызывают функции друг друга. Front-end вызывает back-end для генерации и сохранения объектного кода; back-end вызывает front-end для выполнения вывода, резервирования памяти, работы с типами, узлами ОАГ, символами и строками.

Внутренняя структура LCC весьма проста и хорошо документирована, исходный код написан в одном стиле со множеством комментариев. Все это в сочетании с простотой интерфейса делает настройку LCC нетрудоемкой задачей. Процесс настройки LCC на целевую платформу и описания правил оптимизации значительно проще, чем GCC, при этом производительность генерируемого LCC кода для платформы x86 выше.

В LCC отсутствуют встроенные средства поддержки целевых архитектур с возможностью параллельного исполнения на уровне отдельных инструкций. Однако благодаря простой и открытой структуре компилятора могут быть реализованы при необходимости дополнительные блоки для поддержки архитектур с мелкозернистым параллелизмом, например, VLIW-процессоров.

3. Программный комплекс разработки компиляторов CoSy

Система для проектирования оптимизирующих компиляторов CoSy разработана организацией ACE (Associated Compiler Experts). CoSy позволяет проектировать компиляторы с языков C, C++ и DSP-C для широкого класса DSP, NPU, RISC, VLIW, а также 8/16/32-разрядных микропроцессоров.

Компилятор имеет модульную структуру. В основе лежит универсальное промежуточное представление, которое может расширяться при необходимости. Разработчик может подключать модули оптимизации, генерации кода и др., необходимые для целевой архитектуры. Имеется более 50 модулей, которые могут быть использованы при проектировании компилятора.

Для DSP-процессоров используется язык DSP-C — расширение языка C, позволяющее разработчикам использовать специфические характеристики целевого процессора.

Реализовано множество алгоритмов оптимизации для эффективного использования данных архитектур, в том числе модуль анализа для распараллеливания на уровне инструкций для параллельных VLIW- и DSP-процессоров.

Программный комплекс CoSy используется производителями микропроцессоров для разработки компиляторов для новых архитектур, производителями программного обеспечения, а также в исследованиях микропроцессорных архитектур и технологий проектирования компиляторов.

Перспективные направления автоматизации разработки параллельных систем

На основе анализа существующих аппаратных и программных средств разработки вычислительных систем с распределенными ресурсами можно сделать следующие выводы:

- практически все распространенные средства разработки ПО для вычислительных систем с распределенными ресурсами ориентированы на многопроцессорные архитектуры и вычислительные кластеры;
- возможности разработки ПО для процессоров с поддержкой параллельного исполнения на уровне инструкций ограничиваются использованием алгоритмов peephole-оптимизации после первичной генерации исполняемого кода.

Существующие средства проектирования компиляторов для вычислительных систем с распределенными ресурсами можно разделить на два класса:

- настраиваемые компиляторы, не имеющие встроенных средств поддержки параллельных архитектур (например, GCC и LCC). Они используются в качестве компиляторов для узлов многопроцессорных систем или вычислительных кластеров, построенных на основе последовательных или суперскалярных процессоров. Для поддержки параллельной обработки применяются высокоуровневые библиотеки (например, MPI, OpenMP, DVM и др.), не входящие в состав компилятора;
- настраиваемые компиляторы, имеющие специальные возможности для генерации кода для DSP- или VLIW-процессоров с возможностью параллельного исполнения на уровне инструкций (примеры — CoSy, настройки GCC для некоторых процессоров). Оптимизация и распараллеливание инструкций выполняется после генерации кода с помощью peephole-оптимизации.

Существует две причины, приводящие к увеличению требований к средствам разработки компиляторов:

- Расширение возможностей параллельной обработки на уровне инструкций в существующих процессорах. Peephole-оптимизация позволяет внести лишь локальные улучшения в имеющийся исполняемый код, наиболее же эффективное использование ресурсов процессора может дать анализ потока данных и управления в программе до генерации исполняемого кода.
- Распространение конфигурируемых архитектур позволяет разработчику самостоятельно спроектировать процессор либо аппаратную схему, эффективно выполняющую определенные алгоритмы обработки. В настоящее время такой подход весьма трудоемок, поскольку отсутствуют средства автоматизации проектирования эффективных аппаратных решений, выполняющих сложные алгоритмы обработки.

Актуальной задачей является разработка новых методов и программных средств проектирования компиляторов для вычислительных систем с распределенными ресурсами, имеющих следующие характеристики:

- Наличие алгоритмов автоматического выявления параллелизма в последовательных программах. Алгоритм выявления параллелизма должен осуществлять анализ потока данных и управления и находить отдельные операторы и целые блоки программы, которые могут выполняться параллельно. Результатом работы алгоритма является параллельный граф передачи управления.
- Низкая трудоемкость описания входного языка высокого уровня. Описание синтаксиса входного языка должно осуществляться с использованием РБНФ с поддержкой достаточно широкого класса грамматик.
- Универсальность по отношению к входному языку и целевой архитектуре.
- Интеграция с универсальной отладочной средой.