ИНФОРМАТИКА

2004 январь-март №1

УДК 681.5.004.4, 681.5.004.7

А.В. Отвагин, Р.Х. Садыхов

АРХИТЕКТУРА СИСТЕМЫ АНАЛИЗА И ПРОЕКТИРОВАНИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

Предложена общая структура системы анализа и проектирования параллельных программ на основе анализа методологии разработки параллельных приложений. Рассмотрены методы реализации основных функций, входящих в структуру подсистем с использованием оригинальных алгоритмов оптимизации.

Введение

Развитие вычислительной техники, методов и алгоритмов обработки информации, появление новых задач привели к необходимости значительного увеличения вычислительной производительности современных систем для решения сложных задач. Несмотря на успехи в создании новых высокопроизводительных процессоров и систем на их основе, в настоящее время наиболее реальным способом достижения лучшего соотношения «цена – производительность» является использование параллельных вычислительных систем (ПВС).

Реализация ПВС для решения конкретной задачи может осуществляться двумя способами:

1. Проектируется аппаратная часть и создается программное обеспечение (ПО). Аппаратная часть обычно представляет собой специализированные архитектурные решения, отличающиеся как высокой производительностью, так и высокой стоимостью. Программы, разрабатываемые для подобных систем, тесно привязаны к архитектуре ПВС, поэтому при дальнейшем развитии и замене аппаратуры их приходится дорабатывать, а иногда и создавать заново.

2. Разрабатываются обобщенные библиотеки поддержки виртуальной программной модели, обеспечивающей промежуточный слой между библиотеками операционной системы и среды программирования ПВС, с одной стороны, и физической архитектурой ПВС – с другой. При этом перенос виртуальной программной модели на другую физическую архитектуру позволяет обойтись лишь небольшими по объему и времени изменениями уже имеющегося ПО. Такой подход не только экономит средства, но и ведет к эволюционному развитию уже разработанного сложного ПО. Сам вычислительный комплекс при таком подходе может быть скомпонован из относительно дешевых компонентов, что существенно уменьшает стоимость готовой системы, обеспечивая при этом сопоставимую производительность.

Общая методология проектирования параллельных алгоритмов [1], в значительной степени способная оградить проектировщика от потенциальных ошибок, предусматривает распараллеливание, которое рассматривает машинно-независимые аспекты реализации алгоритма, такие как параллелизм, на первой стадии, а особенности проектирования, связанные с конкретной ПВС, – на второй. Данный подход разделяет процесс проектирования на четыре отдельных этапа: декомпозиция (partitioning), коммуникации (communications), кластеризация (agglomeration) и распределение (mapping). Все эти этапы могут быть частично или полностью автоматизированы. Необходимо создать средства автоматизации, которые позволили бы разработчикам параллельных программ уделять основное внимание алгоритмам обработки информации, а не решению вопросов оптимального распределения задач в вычислительной системе. Кроме того, автоматизация может существенно облегчить процедуру необходимого оформления параллельной программы с учетом требований используемой библиотеки поддержки виртуальной программной модели.

1. Структура системы проектирования и анализа параллельных программ

Большинство существующих параллельных приложений, разработанных для многоцелевых параллельных систем, содержат два типа функций:

– функции, непосредственно реализующие определенный алгоритм обработки информации;

– функции адаптации алгоритма к архитектуре ПВС.

Данное разделение приложения определяет некоторые важные особенности его проектирования в рамках существующих языков и средств параллельного программирования:

– параллельное приложение сложнее разработать при использовании особенностей архитектуры ПВС;

– полученный алгоритм не будет переносимым даже для незначительно отличающихся архитектур;

– платформо-зависимый исходный код, который автоматически оптимизируется, чтобы гарантировать переносимость и производительность, достаточно трудно выделить из общего алгоритма.

Средство проектирования параллельных программ должно содержать два взаимосвязанных компонента, а именно [2]:

– высокоуровневый язык параллельного программирования с явным параллелизмом, содержащим четко определенные основные примитивы параллельных программ;

– набор средств компиляции и оптимизации, учитывающий все машинно-зависимые аспекты построения программы и позволяющий достичь максимальной степени производительности, переносимости и сопровождения параллельного ПО.

Существует достаточно много средств автоматизации параллельного программирования, построенных в виде интегрированных сред поддержки проектирования. Примером могут служить программные пакеты CODE [3], HeNCE, GRADE [4], EDPEPPS[5] и др. Обычно подобные средства используют стандартизированные библиотеки поддержки параллельных вычислений в качестве основы для построения параллельных программ. В настоящее время наиболее широко используются MPI (Message Passing Interface) [6] (варианты реализации MPICH [7], LAM/MPI) или PVM (Parallel Virtual Machines) [8].

Средство автоматизации разработки параллельных программ содержит компоненты для решения следующих задач:

– соблюдение методологии проектирования программ;

– обеспечение платформо-независимой разработки;

– поддержка различных библиотек организации параллельных вычислений.

На основании вышесказанного предлагается следующая структура системы проектирования и анализа параллельных программ (рис. 1).

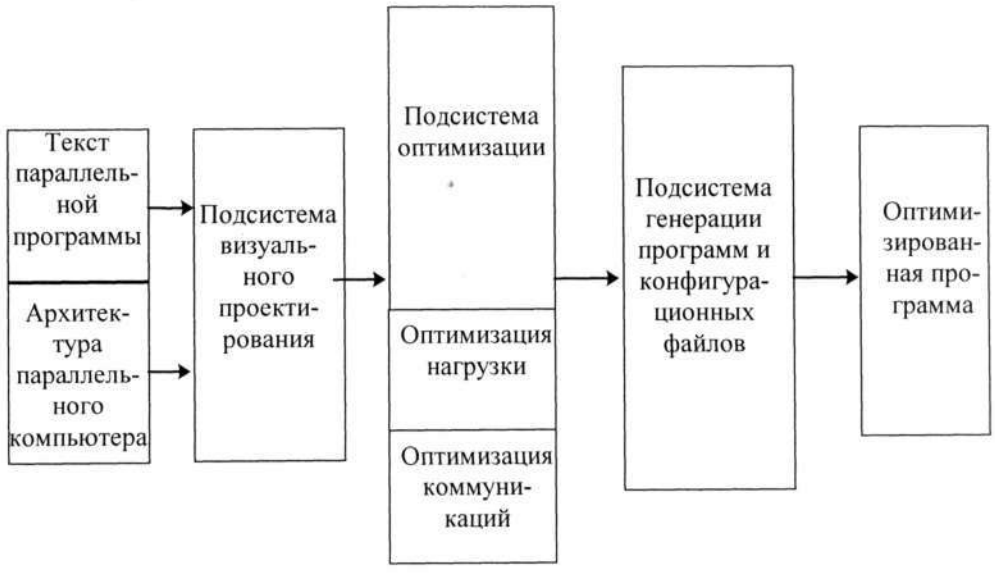

Рис. 1. Архитектура системы проектирования и анализа параллельных программ

2. Назначение и реализация основных подсистем

В состав системы анализа и проектирования параллельных программ входят следующие подсистемы:

- визуального проектирования схем программ;
- оптимизации программ с учетом архитектуры ПВС;
- генерации скелета параллельного приложения.

Подсистема визуального проектирования предназначена для графического ввода и редактирования схем параллельных программ и топологий параллельного компьютера. Как правило, параллельные алгоритмы и архитектуры ПВС представляются в виде графов. В процессе редактирования определяются характеристики отдельных элементов. Для модулей параллельной программы определяются их вычислительная сложность, требования к специализированным ресурсам и, возможно, другие характеристики, например директивный срок завершения (для систем реального времени). Для вычислительных устройств, входящих в состав физической архитектуры параллельной системы, определяются производительность и наличие специализированных ресурсов. Созданные информационные модели могут экспортироваться в файлы XML. В процессе создания моделей происходит их преобразование в объектно-ориентированное представление для последующего анализа.

Средства анализа направлены в первую очередь на оптимизацию производительности параллельной программы. В понятие производительности входит время выполнения программы, эффективность использования оборудования, масштабируемость и стоимость разработки.

Подсистема оптимизации параллельной программы решает задачи распределения множества ветвей параллельной программы по процессорам вычислительной системы, а также задачи оптимизации объема внутрипрограммных коммуникаций. Для решения задачи распределения используется алгоритм декомпозиции параллельной программы, основанный на модели виртуальной сети [9, 10]. Данный алгоритм позволяет добиться сбалансированной нагрузки процессоров (load balancing). Неравномерное распределение вычислительной работы между процессорами, когда отдельные процессоры могут простаивать (idle processors), не обеспечивает высокой степени параллелизма, что, в свою очередь, снижает эффективность. В процессе оптимизации алгоритм формирует кластеры процессов, назначенных на один процессор, минимизируя общее время выполнения программы и объем информационных обменов. Алгоритм может использоваться как для статических, так и для динамических моделей и легко масштабируется на произвольное число задач и процессоров. Кроме того, тестирование алгоритма для различных моделей коммуникационных сетей (с фиксированным и варьируемым временем передачи сообщения) также свидетельствует об эффективности его применения.

Использование в модели виртуальной сети методов теории генетических алгоритмов [11] в сочетании с технологиями обучения нейронных сетей позволяет осуществлять непрерывную оптимизацию в процессе поиска решений. По сравнению с другими алгоритмами оптимизации, например с классическим генетическим алгоритмом, алгоритм виртуальной сети демонстрирует лучшую производительность при увеличении сложности параллельной программы. Кроме того, алгоритм виртуальной сети, наряду с оптимизацией времени выполнения параллельной программы, увеличивает равномерность загрузки процессоров и способствует более эффективному использованию вычислительной системы.

Задачи эффективной реализации информационных обменов внутри параллельной программы тесно связаны с задачей распределения модулей программы по процессорам. При обмене сообщениями между процессорами могут возникать задержки как вследствие особенностей программной реализации алгоритма (например, на момент принятия сообщения процессором оно еще не отправлено другим процессором), так и вследствие аппаратных причин (например, загруженности сети). При проектировании алгоритма необходимо свести к минимуму число коммуникаций, чтобы сделать отношение времени вычислений T_{op} ко времени коммуникаций T_{comm} как можно большим. Для обеспечения своевременного и надежного взаимодействия в рамках параллельной системы требуется обеспечить эффективную маршрутизацию сообщений в коммуникационной подсистеме. Полученная маршрутизация должна обеспечить максимальную скорость передачи без перегрузок линий связи. Перегрузки выражаются в чрезмерных

объемах информации, передаваемых по отдельным маршрутам, вследствие чего возникают задержки в выполнении ветвей параллельного алгоритма. Многие алгоритмы маршрутизации не учитывают этот факт.

В настоящее время наиболее распространенной технологией передачи сообщений в параллельных системах является маршрутизация с коммутацией каналов (wormhole routing) [12]. Такая маршрутизация достаточно просто реализуется и обеспечивает высокую скорость передачи сообщений при отсутствии конфликтов. Скорость обеспечивается за счет уменьшения влияния задержки в транзитных узлах на общее время передачи сообщения. Однако маршрутизация с коммутацией каналов обладает недостатками коммутируемых сетей, в частности, при увеличении трафика в сети ее производительность нелинейно уменьшается из-за роста числа конфликтов между сообщениями. С другой стороны, неправильная организация очередности передачи сообщений может привести к взаимной блокировке процессов.

Для обеспечения равномерности нагрузки на линии связи с одновременным стремлением к максимальной скорости передачи используется статическая маршрутизация, основанная на технологии генетических алгоритмов [13]. Данный подход позволяет добиться оптимизации таблицы маршрутов с учетом различных критериев за короткое время, поскольку предполагает параллельный поиск в пространстве возможных решений. Наряду с оптимизацией маршрутов можно определить потенциальные тупики, возникающие из-за взаимной зависимости в порядке передач сообщений, и своевременно устранить эту ситуацию при проектировании совокупности маршрутов.

После выполнения оптимизационных процедур полученная модель распределения задач может быть протестирована в системе имитационного моделирования. Имитационное моделирование позволяет получить информацию о выполнении отдельных модулей, моментах информационного взаимодействия, оценить время выполнения параллельной программы, а также другие характеристики. На основании этих данных разработчик может принять решение об изменении программы, например объединении некоторых модулей в более крупный или, наоборот, дроблении модуля на несколько мелких. После внесения изменений в информационную модель процесс анализа можно повторить.

Результаты анализа используются при генерации текста параллельной программы. Автоматизация генерации текста позволяет программисту сосредоточиться на алгоритме обработки информации. При этом особенности виртуальной программной модели, например преобразования типов, обязательные вызовы инициализирующих функций, общая структура приложения, могут быть скрыты от пользователя. В процессе генерации создается скелет приложения, содержащий вызовы процедур обработки информации и коммуникационные обмены, настроенные с учетом оптимизированной модели программы. Скелет приложения может перетранслироваться в версию программы для определенной библиотеки поддержки параллельных вычислений.

Заключение

Представленная архитектура система поддержки проектирования параллельных программ позволяет создать средства автоматизации, обеспечивающие существенное ускорение процесса проектирования. Указанные средства могут осуществлять автоматизацию контроля правильности оформления параллельной программы, скрывать от разработчика платформо-зависимые аспекты организации приложения, использовать различные библиотеки поддержки параллельных вычислений. Модульность архитектуры системы предполагает простые возможности ее модификации и расширения.

В настоящее время производится разработка макета системы проектирования параллельных программ, построенной по приведенной архитектуре.