УДК 004.42

ИССЛЕДОВАНИЕ АВТОМАТИЧЕСКОЙ ОПТИМИЗАЦИИ ПРИ КОМПИЛЯЦИИ

Нещадым Валентин Андреевич
ГВУЗ «Национальный горный университет», http://www.nmu.org.ua

Для повышения производительности сложных программных комплексов требуются нетривиальные улучшения исходного кода, и один из популярных методов — использование оптимизирующих компиляторов. Рассматривается автоматическая оптимизация при компиляции на базе компилятора Intel.

Ключевые слова – компиляция; оптимизация; производительность приложения; программы.

ВСТУПЛЕНИЕ

При создании современного программного обеспечения применяются два доминирующих подхода:

1. Написание программ на языке высокого уровня, таких как C, C++, Fortran и других, а после компиляции они распространяются в виде двоичных кодов конкретных аппаратных платформ.
2. Написание программ на переносимых языках высокого уровня, таких как Java или C#, и они распространяются после компиляции в виде кодов виртуальных платформ.

Первый подход ставит перед разработчиками программного обеспечения трудную проблему адаптации программ к различным аппаратным платформам. Прямым следствием этих трудностей становится сокращение числа поддерживаемых платформ, что в конечном итоге приводит к доминированию одной аппаратной платформы. Второй подход оказывается гораздо более привлекательным для разработчиков программного обеспечения, поскольку в этом случае задача адаптации передается тем, кто реализует виртуальные платформы на конкретных аппаратных платформах. Однако выбор языка реализации и платформы определяется не только удобством распространения программ, но и требованиями к их производительности.

Значительная часть создаваемого программного обеспечения требует высокой производительности, поэтому вопросам оптимизации исполнения программ уделяется повышенное внимание. Поэтому активно ведется поиск аппаратных решений, при которых функции оптимизации и обеспечения совместимости были бы целиком переданы оптимизирующим компиляторам [1].

АВТОМАТИЧЕСКАЯ ОПТИМИЗАЦИЯ ПРИ КОМПИЛЯЦИИ

Все современные процессоры, как правило, многопроцессорные и многоядерные, поэтому умение задействовать для вычислений все доступные ресурсы – одна из важных задач для оптимизирующего компилятора. Компилятор Intel предлагает разработчикам использовать автопараллелизацию – механизм автоматического распараллеливания циклов. Это, по сути, самый дешевый в плане затрат труда способ создать многопоточное приложение – запустить построение приложения со специальной опцией. Но с точки зрения оптимизируещего компилятора это непростая задача. Необходимо доказать допустимость оптимизации, необходимо правильно оценить объем работы, выполняемой внутри цикла, чтобы решение о параллелизации улучшило производительность. Автоматическая параллелизация активно взаимодействует с другими цикловыми оптимизациями, изменяя порядок и эвристические механизмы принятия решений по оптимизации. В этой области наилучшие оптимизационные алгоритмы еще не написаны, и простор для экспериментов и новаторских идей по прежнему велик. Для развития более наглядных для программистов методов параллелизации создаются новые языковые расширения и различные библиотеки поддержки, такие как CILK+ или Threading Building Blocks. На рис. 1 приведена схема автоматической параллелизации.

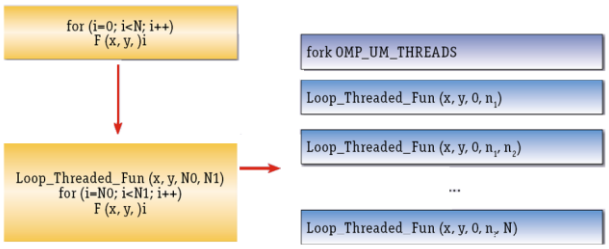

Рисунок 1. Схема автоматической параллелизации

Таким образом, в области создания компиляторов постоянно идет развитие различных идей и алгоритмов, призванных улучшить производительность.

Очень важным компонентом компилятора являются межпроцедурные оптимизации, такие как подстановка функций, дублирование функций, протяжка внутрь функции общих свойств аргументов и т. д. Помимо производительности приложения, для пользователя важны такие характеристики, как размер программы и время компиляции проекта, поэтому нужно ограничивать уровень подстановки так, чтобы эти характеристики оставались в разумных пределах. Компилятор имеет сложный механизм для определения выгодности подстановки той или иной функции. Результаты межпроцедурного анализа используются для лучшей автопараллелизации и векторизации. К сожалению, включение

межпроцедурного анализа увеличивает ресурсы, необходимые для компиляции, – время и размер кода. На рис. 2 представлена двухпроходная и однопроходная схема компиляции.

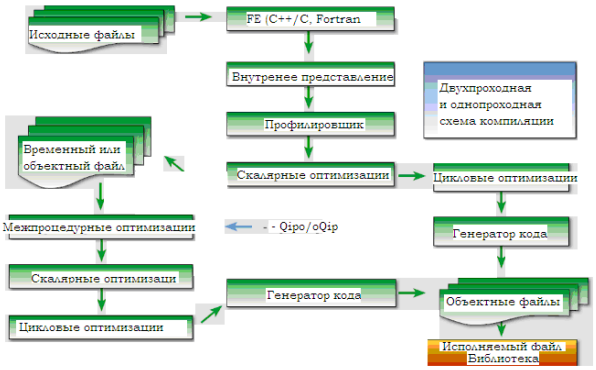

Рисунок 2. Порядок выполнения оптимизаций при однопроходной и двухпроходной схеме компиляции

ВЫВОД

Следствием постоянного увеличения сложности пользовательских приложений стала необходимость экономичного расходования при компиляции имеющихся ресурсов. Важной задачей является улучшение производительности самого компилятора — времени компиляции. Поскольку компилятор Intel в первую очередь ориентирован на получение высокой производительности, то он включает в себя множество различных оптимизационных алгоритмов. Это ведет к тому, что по времени компиляции он иногда существенно проигрывает конкурентам, поэтому нужно уделить серьезное внимание проведению анализа различных алгоритмов и попыткам улучшить их производительность. Развиваются также методы параллельной компиляции.

Активное продвижение на рынок различных мобильных устройств порождает необходимость снижения энергопотребления программ — в задачу компилятора входит определение баланса между быстродействием приложения и энергопотреблением. Например, создание новых потоков требует дополнительных ресурсов мобильного устройства, поэтому нужно тщательно взвешивать все плюсы и минусы параллелизации [2].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. В.Ю. Виконский «Современные оптимизирующие компиляторы».

2. Журнал «Открытые системы» №10, 2011г/ Способ доступа: URL: http://www.osp.ru/os/2011/10/13012227.