международный журнал
ПРОГРАММНЫЕ ПРОДУКТЫ И СИСТЕМЫ

Параллельная реализация модели map-reduce с использованием шаблонных классов c++

Статья опубликована в выпуске журнала № 2 за 2009 год. [17.06.2009]

Авторы: Московский А.А., Первин А.Ю., Тютляева Е.О.

Ключевые слова: шаблоны параллельного программирования, масштабируемые параллельные приложения, map-reduce

Постоянный адрес статьи:http://swsys.ru/index.php?page=article&id=2247

Задача создания эффективных параллельных программ не только не потеряла своей актуальности с ростом числа вычислительных узлов в высокопроизводительных установках, а, наоборот, приобрела еще большую остроту. Простое увеличение числа вычислителей не гарантирует рост производительности системы. Согласно закону Амдала, максимальный прирост производительности с увеличением количества вычислителей в системе (линейный) достижим только на алгоритме, который вообще не содержит последовательных вычислений (идеальное распараллелива- ние) [1]. Чем больше доля последовательных вычислений в алгоритме, тем меньший выигрыш можно получить от добавления вычислительных узлов. Например, если 5 % кода программы может выполняться только последовательно, то никакое увеличение числа процессоров не позволит добиться прироста производительности более чем в 20 раз. Поэтому большая часть усилий специалистов по параллельному программированию затрачивается на уменьшение доли последовательных вычислений в алгоритме.

Для прикладного программиста, основная сфера интересов которого ограничивается предметной областью, усилия по написанию эффективного параллельного кода являются досадными накладными расходами. К тому же эти усилия не всегда приводят к желаемому результату, поскольку создать действительно эффективную параллельную программу не так просто, часто это требует знания различных аспектов распределенных вычислений и определенных навыков программирования для многопроцессорных установок. Используя стандартные средства, программист вынужден задумываться о синхронизации доступа, балансировке нагрузки на вычислительных узлах, учитывать расходы на передачу данных по сети и т.п. Потребность в сокращении времени на проектирование и реализацию параллельных программ обусловила появление на рынке программных продуктов различных систем, частично автоматизирующих процесс создания таких программ. Однако освоение этих систем, в свою очередь, может занимать существенное время.

Целью нашего исследования было создание программного инструмента, который сделает для программиста прозрачными все аспекты распараллеливания вычислений, не потребует от него дополнительных усилий по освоению новой среды программирования и обеспечит эффективный параллельный код приложения. В качестве такого инструмента авторы предлагают реализацию шаблонов на языке C++ для параллельных вычислений. Шаблоны обеспечивают высокоуровневую конструкцию параллельных программ [2]. В качестве более низкоуровневого средства параллельного программирования используется библиотека шаблонных классов C++ T-Sim [3], реализующая подход автоматического динамического распараллеливания программ. Пользовательский интерфейс предлагаемых шаблонов совместим с Standard Templates Library

(STL) языка C++. Для компиляции используется стандартный компилятор с языка C++. Решение предназначено для кластеров и распределенных сетей (Грид).

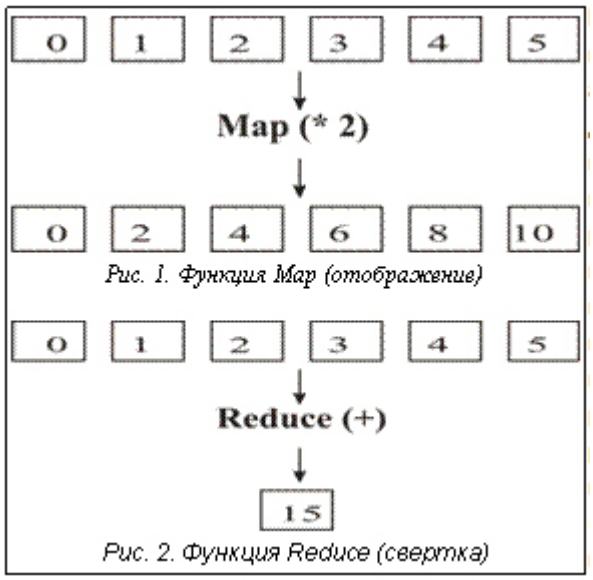

Рис. 1. Функция Map (отображение)

Рис. 2. Функция Reduce (свертка)

Описание модели Map-Reduce

Популярная модель параллельного программирования Map-Reduce хорошо подходит для обработки огромных массивов данных на большом количестве компьютеров. Модель крайне проста и при этом очень эффективна [4]. Она позволяет программисту отвлечься от проблем параллельного выполнения и сконцентрироваться на алгоритмах. Доля последовательного кода в алгоритме стремится к нулю. Алгоритм разбивается на два слоя параллельных операций и в итоге очень хорошо масштабируется. У стандартного алгоритма (имеющего последовательные участки) есть предел, после которого добавление новых процессоров не дает выигрыша в производительности, с вариантом Map-Reduce этот предел почти недостижим. Данная особенность делает модель очень актуальной в условиях постоянного увеличения числа процессоров в современных кластерных установках. Большое количество реальных задач, естественно, выражается в терминах этой модели. Дополнительные плюсы – независимость и возможность при сбое вычислить отдельный участок заново, не перезапуская весь процесс.

В этой модели функция Map (отображение) независимо обрабатывает входные данные, поделенные на блоки. Затем функция Reduce (свертка) формирует окончательный результат из предварительно обработанных данных. Рисунки 1 и 2 иллюстрируют работу функций Map и Reduce. Последовательным аналогом параллельного шаблона Map является шаблон stl::transform из библиотеки STL, который применяет функцию к диапазону элементов и сохраняет результаты. Последовательным аналогом параллельного шаблона Reduce является шаблон stl::accumulate из библиотеки STL, который осуществляет свертку списка или другой структуры в одну величину, применяя бинарный оператор между всеми элементами списка. В нашем примере для наглядности взяты элементарные функции обработки – функция удвоения для шаблона Map и функция сложения для шаблона Reduce.

Реализация: классы стратегий

Шаблоны параллельного программирования (ШПП) Map и Reduce реализованы как шаблонные классы языка C++, по интерфейсу совместимые с библиотекой STL. При реализации использовалась технология классов стратегий (policy classes) [5]. Механизм стратегий основан на комбинации шаблонов C++ с множественным наследованием. Класс, использующий стратегии (главный класс), представляет собой шаблонный класс C++ с многими шаблонными параметрами, каждый из которых является стратегией, определяющей какой-либо один аспект поведения главного класса. Классы стратегий являются базовыми для главного класса (рис. 3). Когда на экземпляре главного класса происходит вызов функции method, вызывается та функция, которая определена в классе стратегии,


Рис. 3. Классы стратегий и главный класс с одной стратегией

переданном в главный класс в качестве параметра при инициации шаблона. В нашем примере это будут функции Strategy1::method для Mainclass и Strategy2::method для Mainclass. У пользователя есть возможность определить свой класс стратегии. Для приведенного примера этот класс обязательно должен содержать определение метода method.

Главный класс делегирует часть функциональных возможностей стратегиям. Можно указывать стратегии по умолчанию. Комбинация различных стратегий дает пользователю набор решений, из которого можно выбрать наиболее удовлетворяющее его задаче. Таким образом, данная технология обеспечивает гибкий механизм настройки поведения главного класса.

В текущей реализации ШПП используются стратегии LoopingTrait и AggregationTrait, которые позволяют настраивать поведение шаблонов для повышения производительности приложения.

Рис. 4. Объединение элементов множества обрабатываемых данных в блоки

Стратегии типа LoopingTrait определяют способ фрагментации входных данных для их последующей обработки. По умолчанию пользователь может воспользоваться стратегией Default Looping, при которой шаблон обрабатывает все переданные ему данные в одном цикле (за один раз). Однако при необходимости обработать очень большое количество данных обработка их за один цикл может привести к потере производительности. Например, если операционной системе не удается загрузить в оперативную память все обрабатываемые данные, начинается использование механизма виртуальной памяти, что сразу приводит к резкой потере производительности шаблонов и приложения в целом. В этом случае лучше разбить исходное множество на несколько фрагментов и обработать их по очереди (за несколько циклов).

Специально для таких случаев разработана стратегия EvenLooping, где N – количество разбиений. Входные данные будут разбиты на N независимых блоков, при этом размерность пользовательских данных необязательно должна быть кратной N. Независимо от размерности данных обработка выполнится корректно. В таких случаях размер последнего обрабатываемого блока данных будет чуть больше остальных за счет остатка.
Кроме того, если ни одна из предлагаемых стратегий не удовлетворяет условиям задачи пользователя, он может написать свою стратегию разбиения. Например, если прикладному программисту понадобится разбить свои данные на неравные сложноструктурированные блоки.

Стратегии типа AggregationTrait определяют способ объединения элементов множества обрабатываемых данных в блоки. Использование различных стратегий типа AggregationTrait позволяет влиять на вес гранулы параллелизма. По умолчанию используется стратегия DefaultAggregation, при которой для каждого элемента из переданного пользователем множества будет порождаться отдельный параллельный поток исполнения. Такое поведение может быть эффективным, если определенная пользователем функция требовательна к ресурсам (процессорному времени). Однако при легковесной операции накладные расходы на распараллеливание исполнения (например, на передачу данных по сети) могут свести на нет выгоды от распараллеливания. Для этого случая следует воспользоваться стратегией Fixed-Aggregation, при которой параллельный поток исполнения будет порождаться не для одного элемента, а для блока из N элементов.

На рисунке 4 продемонстрировано применение класса стратегии FixedAggregation<3>. Использование стратегии разбиения FixedAggregation увеличивает тяжесть одной гранулы в N раз и может обеспечить прирост производительности параллельного приложения в случае легковесной операции обработки данных.

Библиотека T-Sim
В качестве низкоуровневого средства для организации параллельных вычислений ШПП используют библиотеку T-Sim, которая представляет собой набор шаблонов и макроопределений на языке C++. Библиотека реализует сетевой вызов функций, доставку посчитанного результата его потребителю, операции с неготовыми значениями, поддержку механизма неявной синхронизации процессов (приостановка и возобновление вычислений). T-Sim обеспечивает автоматическое динамическое распараллеливание приложения, автоматическую балансировку нагрузки на узлах многопроцессорной вычислительной установки. Механизм реализации планировщика заданий в T-Sim позволяет программисту не только использовать планировщик по умолчанию, но и легко подключать свой алгоритм планирования. Параллельная программа, базирующаяся на использовании библиотеки T-Sim, не содержит явных распараллеливающих конструкций, но все же она должна содержать описание данных и функций с использованием средств T-Sim. Применение ШПП позволяет программисту абстрагироваться от реализации параллельной программы и использования библиотеки T-Sim. Все необходимые для шаблонов Map и Reduce структуры данных и функции, содержащие алгоритм обработки этих данных, описываются на языке C++ обычным способом.

Таким образом, предлагаемое решение использует многоуровневый подход: на верхнем уровне программа оформляется с использованием ШПП, что гарантирует корректный параллельный код приложения; доступ к низкоуровневым средствам библиотеки T-Sim в случае необходимости позволяет программисту оптимизировать параллельное исполнение для специфических условий его приложения, добиваться приемлемых значений производительности и равномерной загрузки вычислительной установки.

Программный интерфейс шаблонов
Вид интерфейса шаблона Map следующий:

template

typename LoopingTrait, typename AggregationTrait>

OutIter Map(InIter first, InIter last, OutIter result, Op& in),

где InIter – тип входного итератора; OutIter – тип возвращаемого итератора; Op – класс, содержащий пользовательскую операцию обработки данных; LoopingTrait – стратегия разбиения на циклы; AggregationTrait – стратегия объединения обрабатываемых данных в блоки; first – итератор начала передаваемых данных; last – итератор конца передаваемых данных; result – итератор на начало результата; in – экземпляр класса, содержащего операцию обработки данных.

В случае использования стратегий по умолчанию можно воспользоваться упрощенным интерфейсом шаблона:

template

OutIter MapS(InIter first, InIter last, OutIter res, Op& op)

{

return Map

(first, end, res, op);

}

Интерфейс шаблона Reduce имеет аналогичный вид.

Для использования ШПП программист должен определить операцию обработки данных на языке C++.

Предлагаемые ШПП были использованы при реализации приложения по классификации мультиспектральных космических снимков с помощью метрики Махалонобиса и при создании экспериментального сервера приложений на основе библиотеки T-Sim.

Предлагаемые шаблоны параллельного программирования реализуют модель Map-Reduce, которая позволяет получать хорошо масштабируемые параллельные приложения.

Шаблоны Map и Reduce обеспечивают высокоуровневую конструкцию параллельных программ. В качестве более низкоуровневого средства параллельного программирования используется библиотека шаблонных классов C++ T-Sim. При программировании приложений с использованием шаблонов Map и Reduce типы данных, специфичные для T-Sim, скрыты от прикладного программиста за интерфейсом шаблонов. Пользовательский интерфейс предлагаемых шаблонов совместим с Standard Templates Library языка C++. Для компиляции шаблонов используется стандартный компилятор с языка C++.

Использование классов стратегий позволяет осуществлять настройку шаблонов в зависимости от специфики алгоритмов обработки данных и объема входных данных, влияя на производительность параллельного приложения.

Шаблоны Map и Reduce позволяют решать широкий класс различных прикладных задач. Например, решение дифференциальных уравнений – это итерационное применение функции Map; любой генетический алгоритм – это итерационное применение шаблона Map к популяции. Алгоритм Reduce может быть использован в задачах оптимизации.

Литература

1. Закон Амдала. URL: http://en.wikipedia.org/wiki/Am-dahl%27s_law (дата обращения: 30.03.2009).
2. Московский А.А., Первин А.Ю., Сергеева Е.О. Первый опыт реализации шаблона параллельного программирования на основе Т-подхода // Программные системы: теория и приложения: тр. Междунар. конф. М.: Наука. Физматлит. 2006. Т. 1.
3. Московский А.А. T-Sim – библиотека для параллельных вычислений на основе подхода Т-системы // Там же.
4. Map Reduce. URL: http://artamonov.ru/2008/09/16/map-reduce/#more-122 (дата обращения: 30.03.2009).
5. Alexandrescu. Modern C++ Design. Addison-Wesley, 2001.