Новые технологии параллельного и традиционного программирования. Процедуры с планированием повторного входа

В.В. Пекунов

Ивановский государственный энергетический университет,
pekunov@mail.ru

Одним из традиционных приемов программирования является сведение задачи к схожим подзадачам меньшей размерности. Существует серия алгоритмов, основанных на таком подходе, предполагающих планирование последовательности решения подзадач на базе формализмов стека, дека, очереди. Назовем алгоритмы: а) поиска в ширину в дереве [1]; б) последовательной (по уровням) нумерации узлов дерева; в) поиска кратчайшего пути между электронными деталями на плате с применением волнового алгоритма Ли; г) нахождения в графе цепочки сетевых подграфов.

Классические решения с применением типовых шаблонных структур данных (реализованных, например, с помощью STL) требуют неявных ввода и контроля целого комплекса новых классов и явного ввода переменных, что усложняет код, повышая вероятность появления ошибок программирования, и негативно отражается на его плотности и скорости исполнения.

Традиционные подходы к распараллеливанию получаемых решений в системах с общей памятью подразумевают ввод в код дополнительной разметки (в соответствии с OpenMP, HPC или иной технологией), маркирующей распараллеливаемые фрагменты. Соответственно, возможности указания параллелизма в существенной степени зависят от того, насколько явно он вынесен на поверхность в исходном коде. В классических решениях некоторые аспекты параллелизма иногда неочевидны, соответственно есть стимул для разработки новых технологий, позволяющих выявить их более четко.

Для рассматриваемого класса алгоритмов предлагаются новые технологии традиционного и пареллельного программирования, упрощающие составление кода, позволяющие уменьшить его объем в 1,5-2 раза, выносящие на поверхность скрытый параллелизм задачи с учетом наличия зависимостей в данных, в ряде случаев дающие более эффективное параллельное решение с минимумом порождаемых потоков и вносимых в код директив разметки.

1. Планирование повторного входа в процедуру

Предлагается ввести в алгоритмические языки высокого уровня формализм процедур с планированием повторного входа, позволяющий более компактно и естественно представлять вышеуказанные алгоритмы. Такие процедуры будут отличаться от обычных наличием плана исполнения, элементами которого (этапами) являются векторы значений параметров для очередного вызова процедуры. Спецификация синтаксиса процедур с повторным входом (void-функций C/C++) и связанных с ними конструкций отчасти изложена в работе [2] и в полном объеме доступна по ссылке: http://www.pekunov.chat.ru/Progs.htm#Reenterable

1.1. Динамический план

Любой явный (в том числе рекурсивный) вызов процедуры с повторным входом создает *новый план*, который сначала содержит один элемент (начальный этап) — вектор значений параметров, указанных при вызове процедуры. В ходе исполнения процедура может включать в начало или конец плана дополнительные этапы с указанием соответствующих значений параметров процедуры. При выходе из процедуры производится проверка плана: если план пуст, то осуществляется возврат в вызывающую программу, в противном случае из начала плана извлекается очередной этап, соответствующие ему значения параметров

помещаются в параметры процедуры и производится повторный вход (переход в начало процедуры).

1.2. Статический план

Если процедура имеет *статический (постоянный) план*, то его исполнение может быть остановлено (с выходом из процедуры) и возобновлено при следующем входе в процедуру. Это позволяет разрабатывать *простые или рекурсивные генераторы* (разновидности сопрограмм), компактно реализующие, например, стек и очередь для сложных типов данных без явного ввода дополнительных структур.

1.3. Передача параметров. Редукция

Передача параметров по значению осуществляется обычным образом. Параметры, передаваемые по ссылке, по умолчанию «туннелируют» между вызовами: их последние значения с предыдущего этапа переходят на текущий этап (за исключением случая *отсечения*, где новое значение параметра для этапа указывается явно). Для такого параметра возможна *редукция*, когда к множеству его значений применяется некоторая бинарная коммутативная операция/функция, результат помещается в тот же параметр. Редукция возможна как в традиционном, так и в параллельном вариантах программы.

2. Применение в параллельном программировании

Процедуры с повторным входом предполагают исполнение линейного плана этапов обработки. Предлагается представить план в виде последовательности непересекающихся групп этапов, разделенных специальными пометками (маркерами). Этапы группы могут выполняться либо последовательно, либо параллельно. *В параллельном режиме* группа рассматривается как динамически пополняемый «портфель задач» [4]. Можно указать количество процессоров, на котором исполняется процедура.

Перспективным представляется применение процедур с повторным входом для распараллеливания изначально рекурсивных алгоритмов, а также алгоритмов групповой обработки элементов рекурсивных структур данных (сетей, деревьев). Полная или частичная *замена рекурсии планированием* (итерацией) иногда дает столь же простое решение (но, возможно, с генерацией меньшего количества потоков) как и при использовании Т-параллелизма [3]. Для рекурсивных структур данных легко *указать наличие зависимостей между элементами* по порядку обработки. Если каждому элементу соответствует один этап обработки, то можно автоматизировать поиск в «портфеле задач» этапов, готовых к исполнению, то есть сопоставленных элементам, зависящим только от уже обработанных элементов.

Реализован ряд стандартных алгоритмов: поиска максимального элемента в дереве; расчета оптимального хода в логических играх с применением минимаксного дерева; сортировок слиянием и Шелла. Показана возможность достаточно эффективного *распараллеливания циклических алгоритмов с заранее неизвестным количеством независимых друг от друга итераций*.

3. Цепи процедур с планированием повторного входа

Рассмотрим алгоритмы, имеющие последовательно зависимые (стадийные) или полностью независимые (параллельные) сегменты решения, каждый из которых предполагает генерацию и исполнение серии подзадач. Особый интерес представляют стадийные алгоритмы, в которых набор подзадач сегмента планирует множество подзадач следующего сегмента, предполагая иную последовательность их исполнения. Например, в волновом алгоритме поиска кратчайшего пути на первой стадии моделируется распространение волны и генерируется (в обратном порядке) план второй стадии, на которой найденный обратный путь неявно трансформируется в прямой. Сегментированные алгоритмы достаточно естественно могут быть реализованы *цепью процедур* с

планированием повторного входа, в которой *каждая процедура реализует один из сегментов решения и может являться генератором плана для следующей процедуры в цепи*.

4. Векторный и конвейерный параллелизм в цепи процедур

Для реализации *векторного параллелизма* логичным решением является ввод механизма параллельного запуска процедур цепи. Аналогично реализуется *конвейерный параллелизм*: причем конвейерная передача данных реализуется путем вставки этапов в начало или конец плана следующей в цепи процедуры. Любая из процедур включается в работу, как только в ее плане появится по меньшей мере один этап работ. Процедура цепи может исполняться на нескольких процессорах («портфель задач»), а также запускать иные цепи. Таким образом реализуется *вложенный параллелизм*.

Реализованы векторный алгоритм умножения матриц и двустадийный конвеерный алгоритм поиска минимума и максимума среди элементов двоичного дерева (с внутренним распараллеливанием стадии).

5. Каналы

Для реализации более сложных алгоритмов параллельной обработки необходимы дополнительные, абстрагированные от архитектуры многопроцессорной системы средства обмена данными между процедурами, исполняемыми на различных процессорах (в разных потоках). Такими средствами могут быть *блокирующие типизированные каналы передачи данных*, аналогичные по функциональности введенным в языке MC# [5].

6. Заключение

Предложен новый формализм процедур с планированием повторного входа, дающий более компактную и очевидную реализацию для ряда задач, традиционно решаемых с помощью вспомогательных типовых структур данных: очереди, стека и дека. Поддерживаются редукционные операции над параметрами процедур. Возможна разработка аналогов простых и рекурсивных генераторов. Предложен механизм цепей процедур с повторным входом, позволяющий достаточно просто записывать некоторые алгоритмы, сводимые к решению серии подзадач с последовательной (стадийной) или параллельной (векторной) обработкой данных.

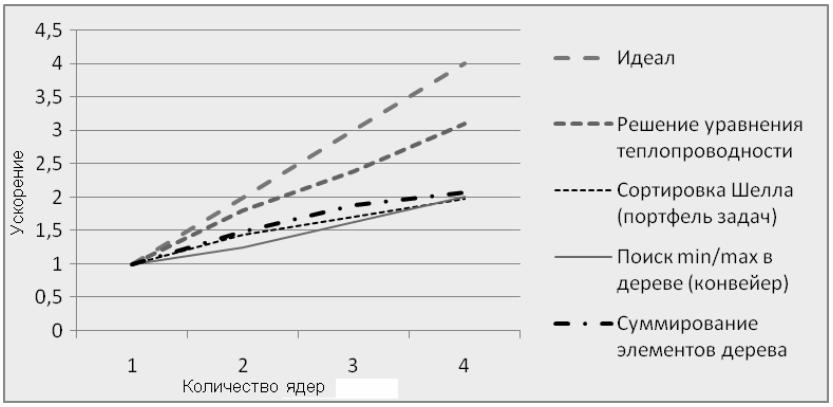

Рисунок. Результаты замеров ускорения расчета для различных алгоритмов при работе с четырехъядерным компьютером (2×Opteron 270, 2 ГГц).

Получаемый в результате применения предложенных конструкций код может иметь более явно выраженный параллелизм (векторный, конвейерный, «портфель задач»), легко выделяемый с помощью нескольких директив распараллеливания. При параллельной обработке рекурсивных структур данных существенно упрощаются (сводятся к разметке декларации соответствующих типов) проверки на наличие зависимо-

стей по порядку обработки элементов. Таким образом, разработана новая компромиссная технология быстрого распараллеливания ряда алгоритмов, в ряде случаев менее громоздкая по сравнению с традиционными средствами (OpenMP, DVM) и порождающая меньшее количество потоков, хотя, возможно, и не столь эффективная как программирование с применением семафоров и блокировок.

Разработан прототипный вариант препроцессора для языка C++ (http://www.pekunov.chat.ru/Progs.htm#Reenterable). Планируется развитие подхода с целью дальнейшего повышения эффективности обработки рекурсивных структур данных, характерных, например, для нейронных сетей. Автор выражает благодарность доценту Е.И.Ляпунову за идею применения процедур с повторным входом для конвейеризации расчета.

Литература

1. Рассел, С., Норвиг, П. Искусственный интеллект: современный подход.— М.: «Вильямс», 2007. — 1408 с.
2. Пекунов, В.В. Процедуры с планированием повторного входа в языках высокого уровня при традиционном и параллельном программировании // Информационные технологии.— 2009.— №8.— С.63-67.
3. Воеводин, В.В., Воеводин, Вл.В. Параллельные вычисления. — СПб.: БХВ-Петербург, 2002. — 608 с.
4. Эндрюс, Г.Р. Основы многопоточного, параллельного и распределенного программирования. М.: «Вильямс», 2003.
5. Петров, А.В. MC# — универсальный язык параллельного программирования / А.В. Петров, В.Б. Гузев // Информационные технологии. — 2008. — №4. — С.29-32.