УДК 004.434

Мохаммад Мохаммад Ибрахим, А.В.Данилова

О РЕАЛИЗАЦИИ КОНЦЕПЦИИ РЕКУРСИВНО-ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

Воронежский институт высоких технологий
ОАО «Концерн «Созвездие»,

В статье рассматриваются особенности рекурсивно-параллельного программирования. Основным его достоинством является возможность использования потенциального параллелизма алгоритма, зависящего от исходных данных. Другое важнейшее достоинство – возможность обеспечения эффективной динамической балансировки загрузки процессорных модулей (ПМ) во время выполнения программы. При этом прикладной программист не должен ничего знать о количестве и быстродействии ПМ, входящих в состав вычислительной системы. Единственное требование, которому должна удовлетворять программа, заключается в том, что в кратчайшее время работа должна быть разбита на достаточное количество независимых фрагментов по возможности одинакового объема и соответствующим образом оформленных. Для довольно широкого класса задач разбиение работы на два равных (или почти равных) фрагмента не составляет труда.

Ключевые слова: программирование, параллельные процессы, программа, вычислительная система, иерархическая модель.

Данная работа посвящена описанию концепции рекурсивно-параллельного программирования, а также разработанных к настоящему моменту языковых и программных средств поддержки этого подхода к организации параллельных вычислений.

Подход этот заметно отличается от методов параллельного программирования, наиболее широко распространенных сейчас в мире, в частности, методов, основанных на MPI (message passing interface).

Последний базируется на достаточно простых понятиях и предоставляет разработчику удобные средства межпроцессорного взаимодействия. Однако на программиста возлагается вся работа по распределению данных и работы по вычислительной системе, а также ответственность за корректную синхронизацию всех параллельных процессов.

Разработка и отладка таких программ (по крайней мере нетривиальных) весьма сложна и трудоемка, и даже доказательство того, что вычислительный процесс не окажется в состоянии бесконечного ожидания, представляет собой серьезную математическую проблему.

Концепция рекурсивно-параллельного программирования основана на очень простой, даже примитивной, схеме синхронизации параллельных процессов, возможно несколько понижающей гибкость программирования.

Но это позволяет не только существенно упростить процесс разработки и отладки параллельной программы, но и, что еще более важно,

придать программе независимость от состава и быстродействия модулей, образующих вычислительную систему, а также возможность эффективного выявления и использования параллелизма, зависящего от обрабатываемых данных.

Для того, чтобы добиться эффективной параллельной работы программы, следует, во-первых, максимально сократить затраты времени на передачу данных между процессорными модулями (ПМ), во-вторых, распределить работу таким образом, чтобы по возможности более ровно загрузить вычислительные ресурсы системы [1, 2].

Последняя задача усложняется еще и тем обстоятельством, что ход вычислений и длительность выполнения тех или иных параллельных ветвей программы, вообще говоря, зависит от обрабатываемых данных.

Зачастую это обстоятельство делает невозможным при написании программы или в процессе трансляции осуществить эффективное распределение работы.

В ряде случаев данный недостаток приводит к выполнению избыточных вычислений и, соответственно, увеличению времени решения задач.

К примеру, в сеточных задачах (методы конечных разностей или конечных элементов) использование нерегулярных сеток для некоторых применений позволяет повысить точность решения и устранить избыточные вычисления.

Использование нерегулярных сеток в этих случаях диктуется как пространственной, так и временной неоднородностью реальных процессов. В качестве другого примера можно привести задачу численного интегрирования методом адаптивной квадратуры, для которой статическое распараллеливание также не очень эффективно.

Динамическая балансировка, то есть оперативное перераспределение работы между процессорными модулями во время выполнения программы, позволяет добиться более равномерной загрузки вычислительных ресурсов системы и, следовательно, большей эффективности.

Потребность в динамической балансировке возникает в тех случаях, когда невозможно сделать оптимальное статическое отображение процессов на процессорные модули из-за отсутствия достаточной информации об объеме вычислений на параллельных ветвях, либо из-за того, что каждому шагу решения задачи соответствует свое оптимальное отображение.

За примером можно обратиться к тем же сеточным задачам, когда времена вычислений в узлах используемой сетки априори неизвестны и могут изменяться от одного временного шага к другому.

Другим аргументом в пользу динамической балансировки является то, что она делает возможной разработку прикладных программ, инвариантных к конфигурации параллельной вычислительной системы.

Это существенно упрощает процесс программирования и позволяет создавать легко переносимые библиотеки и пакеты прикладных программ. 5 Задача прикладного программиста при этом фактически сводится к декомпозиции задачи на достаточное количество параллельных процессов.

При этом, чем больше их количество, тем более равномерной загрузки мы можем добиться. Опыт показывает, что хорошие результаты достигаются, если их количество параллельных процессов превышает количество процессорных модулей на порядок, даже если трудоемкость параллельных ветвей программы существенно различается.

Однако при таких условиях организация в системе централизованного распределения работы и единой очереди готовых к выполнению процессов (типа "Пул задач" [3]) может привести к появлению "узкого места" и, следовательно, к существенному падению эффективности.

Кроме того, в этом случае оценка времени заполнения всех процессорных модулей работой при условии, что в начальный момент работу имел только один, будет линейной.

В основу данной разработки был положен известный принцип рекурсивного порождения параллельных процессов, идея которого весьма проста и, можно сказать, лежит на поверхности.

Однако в нашем случае этот метод используется не только как средство порождения параллельных ветвей задачи, но и как важный элемент механизма динамической балансировки загрузки, позволяя существенно уменьшить количество передач работы и результатов.

Все это позволяет в значительной степени снизить влияние упомянутых факторов и разрабатывать достаточно эффективные параллельные прикладные программы, обладающие свойством инвариантности к конкретной конфигурации вычислительной системы.

Разработанный алгоритм динамической балансировки носит децентрализованный характер, то есть решение о запросе или передаче работы каждый процессорный модуль принимает самостоятельно на основе имеющейся в его распоряжении информации, может быть и устаревшей.

Однако централизованный характер управления неизбежно накладывает серьезные ограничения на возможность наращивания системы [4, 5].

Оценка скорости заполнения всей системы работой становится логарифмической. Отметим также, что предлагаемый язык программирования и средства поддержки отнюдь не исключают и не

рекурсивный способ порождения параллельных ветвей программы, однако в этом случае временные затраты на распределение работы по системе будут, по-видимому, выше.

Точно так же предусмотрена возможность явного назначения программистом порождаемых параллельных процессов на выполнение конкретным ПМ.

В принципе, зная конфигурацию имеющейся вычислительной системы, в этом случае можно добиться даже некоторого повышения эффективности работы программы, однако в случае любого изменения конфигурации системы производительность ее может существенно снизиться.

Реализация вычислительного модуля на основе рекурсивно-параллельного метода программирования и предлагаемых механизмов порождения и распределения параллельных вычислительных процессов дает возможность создавать модульно-наращиваемые параллельные ВС различной конфигурации вплоть до супер-ЭВМ, обеспечивающие высокую эффективность функционирования на широком классе задач.

Вместе с тем, те же самые методы и механизмы, реализованные программно, вполне пригодны для эффективной организации распределенных вычислений на локальной сети компьютеров.

Такая возможность, в частности, реализована в описанной ниже пользовательской среде рекурсивно- параллельного программирования RPMSHELL.

Образующее ее программное обеспечение позволяет произвести полный цикл создания такой рекурсивно-параллельной программы, включая ее последовательную и параллельную отладку, а также исследование ее поведения путем имитационного моделирования и оптимизацию.

Дальнейшее содержание посвящено описанию применяемой модели параллельных вычислений, основных механизмов порождения и активизации параллельных процессов, краткой характеристики используемого языка рекурсивно-параллельного (РП-) программирования, а также программных инструментальных средств поддержки данного стиля построения алгоритмов и программ [6, 7].

Как следует из сказанного выше, рекурсивно-параллельный стиль программирования является одним из перспективных подходов к организации параллельного вычислительного процесса.

Основным его достоинством является возможность использования потенциального параллелизма алгоритма, зависящего от исходных данных. Другое важнейшее достоинство – возможность обеспечения эффективной динамической балансировки загрузки процессорных модулей (ПМ) во время выполнения программы.

При этом прикладной программист не должен ничего знать о количестве и быстродействии ПМ, входящих в состав вычислительной системы.

Единственное требование, которому должна удовлетворять программа, заключается в том, что в кратчайшее время работа должна быть разбита на достаточное количество независимых фрагментов по возможности одинакового объема и соответствующим образом оформленных.

Для довольно широкого класса задач разбиение работы на два равных (или почти равных) фрагмента не составляет труда.

Например, для нахождения суммы n чисел, являющихся результатом некоторой функции F(i), $1 \le i \le n$, можно независимо вычислить сумму первых n/2 слагаемых (здесь и далее деление целочисленное) и сумму оставшихся n/2 слагаемых. Решением будет сумма двух частичных результатов.

Несложно видеть, что каждая из двух "половинок" работ может быть разбита на две "четвертушки" и т.д. Таким образом, если в системе имеется N процессорных модулей, то теоретически уже за время порядка $\log_2 N$ мы можем их всех обеспечить работой, причем с высокой степенью равномерности.

Как оформить предложенный алгоритм в виде программы? Очень удобным и естественным оказывается оформление его как рекурсивной функции (на языке C):

```
float Sum(int i, int j)
{ if (i==j) return( F(i) );
else return( Sum(i,(i+j)/2) + Sum((i+j)/2+1,j) ); }
```

Здесь параметры i, j – начальный и конечный индексы суммирования. Договоримся называть активацией функции (или процедуры) ее запуск с конкретными значениями параметров. Результатом активации функции Sum(i, j) является, ∑= j k i F(k), т.е. одна из частичных сумм. Чтобы получить сумму всех элементов, достаточно применить оператор вызова Sum(1, n).

Разумеется, для того, чтобы данная функция могла выполниться параллельным образом, необходимо изменить механизм вызова функций таким образом, чтобы в процессе выполнения программы соответствующая команда не приводила к передаче управления в тело функции, а просто порождала бы соответствующий параллельный (потенциально мигрирующий) процесс, а программа продолжала бы выполняться, не дожидаясь его выполнения.

Аппаратура и системное программное обеспечение рекурсивно-параллельного мультипроцессора (RPM) должны обеспечивать передачу

потенциально мигрирующих процессов между ПМ и возвращение результатов.

Кроме того, очевидно, что появляется необходимость принципиально новых операторов языка программирования, обеспечивающих потребности именно параллельной организации вычислительного процесса.

Как минимум, это операторы описания и запуска параллельных функций, а также операторы синхронизации параллельных вычислительных процессов.

Так, в нашем примере мы должны дождаться завершения двух рекурсивных вызовов функции Sum(), прежде чем произведем сложение полученных результатов.

При этом порождается n "листьевых" активаций, после чего начинается процесс "обратного хода" рекурсии, в котором "дочерние" активации процедуры Sum() возвращают свои результаты "родительским" активациям.

Это продолжается, пока не будет получен конечный результат.

В дальнейшем мы увидим, что такое мелкое деление работы неразумно, но пока на этом не будем останавливаться подробнее.

Такой стиль программирования будем называть РП-программированием. РП-программа представляет собой множество процедур, допускающих рекурсивный вызов.

Вызов РП-процедуры с конкретными значениями аргументов мы будем называть активацией. Возврат из "дочерней" активации осуществляется в специальную точку синхронизации "родительской" процедуры, а не в точку, непосредственно следующую за вызовом процедуры, как это имеет место в обычных последовательных программах.

Параллельный алгоритм, построенный таким образом, во-первых, инвариантен к количеству и быстродействию вычислительных модулей в системе, во-вторых, позволяет выявить параллелизм вычислительного процесса, зависящий от обрабатываемых данных, и, следовательно, проявляющийся только в ходе решения задачи.

При этом скорость размножения параллельных процессов и распространения их по системе при грамотной организации может иметь экспоненциальный характер.

Описанные ниже механизмы позволяют произвести действия по распределению и перераспределению работы в системе на фоне основных вычислений, а также минимизировать количество передач, необходимых для равномерного распределения работы в системе.

Последнее свойство заметно снижает накладные расходы на организацию динамического распараллеливания и позволяет добиться практически линейного характера ускорения с ростом числа ПМ.

РП-программа представляет собой иерархическое множество процедур двух типов (параллельные и последовательные), допускающих рекурсивный вызов. Вычислительный процесс, протекающий в вычислительной системе, поддерживающей РП-стиль программирования, является иерархическим параллельным процессом.

Компоненты этого процесса есть активации параллельных процедур, а также некоторые системные функции. Любая компонента, соответствующая активации процедуры, в свою очередь может быть иерархическим параллельным процессом.

В качестве основных типов компонент параллельного процесса можно выделить: - параллельный вызов параллельной процедуры, - последовательный вызов параллельной процедуры, - параллельные операторы доступа в разделяемую общую, а также в статическую память, - параллельные операторы ввода/вывода (в настоящей версии языка не предусмотрены), - оператор синхронизации, - некоторые другие параллельные операторы.

Активация процедуры является параллельной, если ее вызов осуществляется посредством специального оператора порождения параллельного процесса.

В этом случае активация параллельной процедуры может быть выполнена на любом свободном ПМ системы.

После вызова параллельной процедуры вычисления в родительской процедуре продолжаются без приостановки до точки синхронизации.

Возврат из дочерней параллельной процедуры в родительскую процедуру осуществляется в точку синхронизации. Таким образом, вычисления в дочерних процедурах и родительской процедуре могут выполняться параллельно.

В точке синхронизации происходит ожидание завершения всех дочерних параллельных процессов, запущенных до момента попадания в данную точку родительской процедуры.

Иерархическая модель параллельных вычислений регламентирует взаимодействие процессов (активаций процедур) следующим образом:

- каждая активация имеет связь по управлению только с родительской и дочерними активациями, при этом активация может завершить свое выполнение только в том случае, если завершили выполнение все ее дочерние активации;
- родительская активация при порождении дочерней активации может передать ей в качестве параметров исходные данные, а после завершения дочерней активации получить от нее возвращаемые параметры, эта передача осуществляется на уровне локальной памяти процессоров без использования разделяемой памяти;

- любой другой обмен данными между активациями процедур может быть организован через специальные операторы доступа к разделяемой или статической памяти. Подытожим основные достоинства рекурсивной формы представления параллелизма. Рекурсия описывает динамически порождаемые параллельные процессы в виде активаций процедур, что делает возможным:
- разрабатывать параллельные программы, инвариантные к конфигурации аппаратуры, что особенно важно при создании отказоустойчивых систем и написании широко используемых прикладных пакетов и библиотек стандартных параллельных программ;
- представлять параллелизм, зависящий от значений обрабатываемых данных, т.е. тот параллелизм, который можно выявить только в динамике вычислений;
- осуществлять эффективную динамическую балансировку загрузки ПМ системы путем миграции параллельных процессов.

Сам процесс динамической балансировки скрыт от программиста и осуществляется системными средствами [8-11].

Основными отличиями ее в рекурсивно-параллельной вычислительной системе от обычного динамического назначения процессов на обработку являются следующие два.

Во-первых, отличительной особенностью применяемого алгоритма балансировки является то, что он – децентрализованный (то есть не существует общей для всей системы очереди работ). Это позволяет ликвидировать одно из потенциальных "узких мест" в системе.

Во-вторых, ориентация на рекурсивный способ порождения активаций параллельных процедур (а именно активация процедуры является передаваемой единицей работы) в сочетании с описанным ниже механизмом передачи позволяет организовать более эффективное начальное распределение работ.

Передается по коммутационной сети не каждая порождаемая активация параллельной процедуры, а лишь только порожденные первыми достаточно емкие активации. При этом сам процесс порождения работы организован параллельным образом, а передача ее может осуществляться на фоне вычислений. Для заполнения системы с N ПМ требуется N-1 передача, и время порядка $O(\log_2 N)$.

Процесс первоначального распределения завершается, как только у всех ПМ есть работа. Дальнейшее порождение активаций в ходе рекурсивной развертки алгоритма уже не приводит к передаче работы в другой ПМ и, следовательно, к существенному росту накладных расходов.

Последующие миграции параллельных процессов возможны только в случае, когда какой-либо ПМ не имеет готовых к выполнению активаций.

При этом благодаря описанному ниже способу хранения и передачи потенциально мигрирующих процессов, ПМ-работодатель отдает всегда один, самый емкий из имеющихся, что позволяет минимизировать количество необходимых передач работы и возврата результатов, который происходит в обратном порядке.

Как видно из сказанного, рекурсивная организация программы и средства динамической балансировки обеспечивают высокую скорость распространения параллельных процессов среди ПМ системы. Механизм распространения при этом подобен цепной реакции.

Время миграции практически не зависит от объема входных данных, обрабатываемых процессом, т.к. основной их объем обычно размещен в разделяемой либо статической памяти.

Мигрирует только дескриптор процесса и блок из соответствующих параметров.

Благодаря этому свойству вычислительная система может содержать очень большое число ПМ без ущерба динамике миграции параллельных процессов. Кроме того, в "развернутом" виде (с выделенным стеком и блоками локальных данных) одновременно существует сравнительно небольшое количество всех процессов (примерно логарифм двоичный от их общего числа), что существенно снижает требования к необходимой оперативной памяти.

Вывод. В работе проводится анализ возможностей реализации рекурсивно-параллельного программирования. Дан пример реализации такого подхода в программе. Указаны требования к иерархической модели параллельных вычислений.

ЛИТЕРАТУРА

1. Завьялов Д.В. О применении информационных технологий / Д.В.Завьялов // Современные наукоемкие технологии. 2013. № 8-1. С. 71-72.
2. Чопоров О.Н. Методы анализа значимости показателей при классификационном и прогностическом моделировании / О.Н.Чопоров, А.Н.Чупеев, С.Ю.Брегеда // Вестник Воронежского государственного технического университета. 2008. Т. 4. № 9. С. 92-94.
3. Залогова Л. Разработка Паскаль-компилятора / Л. Залогова // Бином. Лаборатория знаний - Москва, 2010. - 184 с.
4. Зяблов Е.Л. Построение объектно-семантической модели системы управления / Е.Л.Зяблов, Ю.П.Преображенский // Вестник Воронежского института высоких технологий. 2008. № 3. С. 029-030.

5. Иванов М.С. Разработка алгоритма отсечения деревьев / М.С.Иванов, Ю.П. Преображенский // Вестник Воронежского института высоких технологий. 2008. № 3. С. 031-032.
6. Паневин Р.Ю. Структурные и функцинальные требования к программному комплексу представления знаний / Р.Ю.Паневин, Ю.П.Преображенский // Вестник Воронежского института высоких технологий. 2008. № 3. С. 061-064.
7. Голицына О. Л. Основы алгоритмизации и программирования / О. Л.Голицына, И. И.Попов // Форум - Москва, 2010. - 432 с.
8. Пахомова А.С. Целенаправленные угрозы компьютерного шпионажа: признаки, принципы и технологии реализации / А.С.Пахомова, О.Н.Чопоров, К.А.Разинкин // Информация и безопасность. 2013. Т. 16. № 2. С. 211-214.
9. Дешина А.Е. Информационные риски в мультисерверных системах: выбор параметров системы защиты / А.Е.Дешина, О.Н.Чопоров, К.А.Разинкин // Информация и безопасность. 2013. Т. 16. № 3. С. 365-370.
10. Чопоров О.Н. Рационализация управления региональными системами на основе использования методов системного анализа, информационных и ГИС-технологий / О.Н.Чопоров, Н.А.Гладских, С.С.Пронин, М.И.Чудинов, С.Н.Семенов, К.Л.Матюшевский // Прикладные информационные аспекты медицины. 2007. Т. 10. № 2. С. 15-19.
11. Кравец О.Я. Особенности программного проектирования коммутационных подсистем в составе распределенных систем оперативного оповещения и мониторинга / О.Я.Кравец, О.Ю.Макаров, С.А.Олейникова, В.М.Питолин, О.Н.Чопоров // Системы управления и информационные технологии. 2013. Т. 52. № 2. С. 50-54.

Mohammad Mohammad Ibrahim, A.V.Danilova

ON THE IMPLEMENTATION OF THE CONCEPT OF RECURSIVE-PARALLEL PROGRAMMING

Voronezh Institute of High Technologies
JSC Sozvezdiye Concern

The paper discusses the features of the recursive-parallel programming. Its main advantage is the ability to use the potential parallelism of the algorithm, independent of the original data. Another important advantage – the ability to ensure an effective dynamic load balancing of processor modules (PM) during program execution. Thus an application programmer need not know anything about the number and performance of PM included in the computing system. The only requirement that must be met by program is that in a short time the work must be divided into a sufficient number of independent fragments of the same volume and appropriately decorated. For a fairly wide class of problems splitting the work into two equal (or almost equal) fragment is easy.

Keywords: programming, parallel processes, program, computer system, hierarchical model.

REFERENCES

1. Zav'yalov D.V. O primenenii informatsionnykh tekhnologiy / D.V.Zav'yalov // Sovremennye naukoemkie tekhnologii. 2013. № 8-1. S. 71-72.
2. Choporov O.N. Metody analiza znachimosti pokazateley pri klassifikatsionnom i prognosticheskom modelirovanii / O.N.Choporov, A.N.Chupeev, S.Yu.Bregeda // Vestnik Voronezhskogo gosudarstvennogo tekhnicheskogo universiteta. 2008. T. 4. № 9. S. 92-94.
3. Zalogova L. Razrabotka Paskal'-kompilyatora / L. Zalogova // Binom. Laboratoriya znaniy - Moskva, 2010. - 184 c.
4. Zyablov E.L. Postroenie ob"ektno-semanticheskoy modeli sistemy upravleniya / E.L.Zyablov, Yu.P.Preobrazhenskiy // Vestnik Voronezhskogo instituta vysokikh tekhnologiy. 2008. № 3. S. 029-030.
5. Ivanov M.S. Razrabotka algoritma otsecheniya derev'ev / M.S.Ivanov, Yu.P. Preobrazhenskiy // Vestnik Voronezhskogo instituta vysokikh tekhnologiy. 2008. № 3. S. 031-032.
6. Panevin R.Yu. Strukturnye i funktsinal'nye trebovaniya k programmnomu kompleksu predstavleniya znaniy / R.Yu.Panevin, Yu.P.Preobrazhenskiy // Vestnik Voronezhskogo instituta vysokikh tekhnologiy. 2008. № 3. S. 061-064.
7. Golitsyna O. L. Osnovy algoritmizatsii i programmirovaniya / O. L.Golitsyna, I. I.Popov // Forum - Moskva, 2010. - 432 c.