УДК 519.685.1

СИСТЕМА АСИНХРОННОГО ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ "АСПЕКТ"

С. Б. Арыков[1], **В. Э. Малышкин**[1]

Описываются основные идеи системы асинхронного параллельного программирования высокого уровня, предоставляющей пользователю возможности автоматического конструирования параллельных программ по непроцедурному представлению алгоритма. Предложена специализированная асинхронная модель с группировкой вычислений, ориентированная на алгоритмы с регулярными структурами данных, и рассмотрены особенности ее практической реализации. Кратко обсуждается язык программирования Аспект, позволяющий представлять алгоритмы в непроцедурной форме. Статья подготовлена по материалам доклада авторов на международной научной конференции "Параллельные вычислительные технологии" (ПаВТ-2008; http://agora.guru.ru/pavt).

1. Введение. В настоящее время все большее число прикладных специалистов используют численное моделирование для решения практических задач. В этой связи повышается потребность в системах программирования высокого уровня, которые, по возможности, скрывают от программиста вопросы распределения ресурсов и автоматически обеспечивают динамические свойства прикладных параллельных программ, позволяя сосредоточиться на алгоритме решаемой задачи. С другой стороны, широко доступными становятся большие вычислительные мощности, что также требует разработки инструментов, позволяющих эффективно использовать эти мощности.

Существующие системы параллельного программирования имеют недостаточно высокий уровень. Так, наиболее часто используемый стандарт MPI заставляет пользователя оперировать примитивами типа принять/получить сообщение. Системы программирования DVM [1], mpC [2] и ряд подобных позволяют скрыть от пользователя организацию коммуникаций. Тем не менее задача выделения параллельных процессов и их синхронизации все равно ложится на программиста. Предпринимаются также попытки создания систем с автоматизированным конструированием параллельных программ (например, Т-система [3, 4] и НОРМА [5]), однако такие системы имеют недостаточную производительность на реальных вычислительных задачах.

В Институте вычислительной математики и математической геофизики СО РАН ведется разработка системы асинхронного параллельного программирования Аспект, в которой за счет специализации (ориентация на задачи с регулярными структурами данных) предполагается существенно повысить уровень программирования, сохранив при этом накладные расходы на приемлемом уровне. Следует подчеркнуть, что высокоуровневость здесь понимается в технологическом, а не в функциональном смысле: программирование в системе Аспект не является декларативным, программист не может оперировать понятиями предметной области (такими как сетка или пространство моделирования), однако все технические детали конструирования и исполнения параллельных программ система берет на себя.

Основная идея системы Аспект заключается в следующем: будем представлять алгоритмы в непроцедурной форме (которая сохраняет естественный параллелизм) и по этому непроцедурному представлению автоматически генерировать параллельную программу такой степени непроцедурности, которая была бы в некотором смысле оптимальна для исполнения на целевой вычислительной системе (вычислителе).

2. Модель вычислений. Асинхронная модель вычислений [6] является наиболее подходящей для представления параллельных алгоритмов, поскольку она позволяет полностью сохранять естественный параллелизм. Однако при исполнении асинхронные программы часто показывают низкую эффективность вследствие больших накладных расходов на организацию управления. Другой существенный момент состоит в том, что для улучшения реализации асинхронной программы необходимо иметь возможность варьировать размер (объем вычислений) асинхронного блока, подстраивая его под архитектуру конкретного вычислителя (например, под размер кэш-памяти процессора).

[1] Институт вычислительной математики и математической геофизики СО РАН, пр. Лаврентьева, 6, 630090, Новосибирск; e-mail: arykov@mail.ru; malysh@ssd.sscc.ru

Для решения указанных проблем была разработана специализированная модель вычислений, которая позволяет плавно изменять степень непроцедурности [7, 8] программ, тем самым добиваясь компромисса между затратами на реализацию управления в асинхронной программе и размером блока.

2.1. Базовая асинхронная модель вычислений. Асинхронная программа (или А-программа) — это конечное множество А-блоков, определенных над информационной IM и управляющей CM памятями [8].

Память состоит из переменных с неразрушающим чтением и записью, стирающей предыдущее содержимое переменной. Информационная память используется для хранения данных решаемой задачи, а управляющая память — для организации управления в программе.

Каждый А-блок представляется тройкой (T, O, C), где T — спусковая функция (или триггер-функция), представляющая собой некоторый предикат над CM; O — операция над IM, вычисляющая по входным параметрам значения выходных параметров; C — управляющий оператор, изменяющий значение управляющей памяти и организующий управление в программе. Таким образом, вычисления в асинхронной модели полностью отделены от управления, что является одним из достоинств модели.

Вычисления по асинхронной программе организуются следующим образом.

1. Для всех А-блоков вычисляются их триггер-функции. А-блок с истинной триггер-функцией объявляется готовым к исполнению. Формируется множество готовых к исполнению А-блоков.

2. Выполняется некоторое подмножество готовых к исполнению А-блоков. Выполнение А-блока состоит в вычислении его операции и управляющего оператора. После завершения выполнения А-блоков переходим на шаг 1.

3. Выполнение А-программы завершается, когда ни один А-блок не выполняется и триггер-функции всех А-блоков ложны.

Для представления массовых вычислений в работе [9] в асинхронную модель было введено понятие массового А-блока, при этом появляется возможность определять А-блоки над массивами. В системе Аспект это понятие также широко используется.

2.2. Асинхронная модель с группировкой вычислений. Сгруппированная А-программа представляет собой конечное множество А-групп, определенных над информационной и управляющей памятями, и конструируется из асинхронной программы по следующему алгоритму.

Перебираем все А-блоки асинхронной программы и для каждого А-блока выполняем следующие действия. Если А-блок простой, то он является А-группой по определению (остается без изменений). Если А-блок массовый, тогда экземпляры А-блока могут объединяться в А-группы по n экземпляров в каждой. Каждая А-группа формируется следующим образом:

— триггер-функция А-группы есть конъюнкция триггер-функций всех экземпляров группы;

— операция А-группы есть новая операция, представляющая собой последовательность операций всех экземпляров группы;

— управляющий оператор А-группы есть новый управляющий оператор, представляющий собой последовательность управляющих операторов экземпляров группы.

Определенная в столь общем виде модель мало пригодна для реализации, так как в рамках А-группы по-прежнему выполняется вычисление управления для каждого экземпляра А-блока, поэтому желаемого сокращения управления вследствие уменьшения асинхронности не происходит.

Чтобы вычислять управление для всей А-группы целиком, необходима дополнительная информация о характере вычислений. Ее можно получить, воспользовавшись специализированностью системы Аспект (ориентация на регулярные структуры данных) и введя следующие ограничения, приемлемые для выбранной специализации:

— линейность областей применимости [9] массовых А-блоков; за счет линейности проверку всего диапазона удается заменить проверкой на границах;

— размер области применимости известен до начала вычислений; для массовых А-блоков, область применимости которых до начала вычислений не известна, А-группы не строятся (точнее, каждый экземпляр такого А-блока является А-группой);

— для массовых А-блоков триггер-функция не может зависеть от массовых переменных, что позволяет исключить ситуации, когда из-за ложности триггер-функции одного экземпляра отменяется выполнение всей А-группы целиком; если же возможность запуска каждого экземпляра необходимо подчинить некоторому условию, зависящему от массовых переменных, то проверку этого условия программист может поместить непосредственно в код операции экземпляра.

С учетом введенных ограничений управляющий оператор для А-группы можно построить по следующему алгоритму.

Перебираем все переменные, вычисляемые A-группой A, и для каждой переменной x: формируем множество $\mathbf{M}$ A-групп, которые зависят от x, и вычисляем число зависимости μ переменной x. Это число показывает, насколько уменьшается зависимость A-групп из $\mathbf{M}$ от A, если вычислена переменная x (для простых переменных) либо одна компонента x (для массивов). Для каждой A-группы F из множества $\mathbf{M}$ выполняем следующие действия:

— если x простая переменная, то уменьшаем зависимость F на величину μ;

— если x массив, то определяем пересечение $\mathbf{C}$ диапазона значений переменной x, потребляемого A-группой F, с диапазоном значений переменной x, вычисляемым A-группой A, и уменьшаем зависимость F на величину $\mu\,|\mathbf{C}|$.

Таким образом, программа в сгруппированном виде имеет меньшую асинхронность, но и меньший объем управления, т.е. может исполняться более эффективно. Степень асинхронности получаемой программы зависит от размера A-группы и может варьироваться в широких пределах.

3. Реализация системы параллельного программирования Аспект.

3.1. Общая архитектура. Система программирования Аспект состоит из двух основных компонентов: транслятора и исполнительной подсистемы (рис. 1).

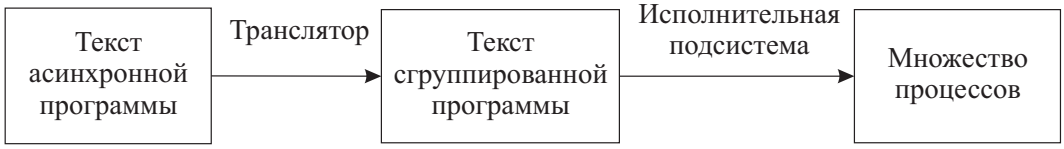

Рис. 1. Процесс разработки в системе программирования Аспект

На вход транслятору подается текст программы на языке Аспект (текст асинхронной программы), а на выходе генерируется программа на C++ (текст сгруппированной программы), которая затем отображается исполнительной подсистемой в множество вычислительных процессов. В отличие от базовой асинхронной модели, управление в тексте асинхронной программы задается не с помощью управляющих операторов, а с помощью указания частичного порядка на множестве A-блоков.

Рассмотрим каждый компонент подробнее.

3.2. Язык программирования Аспект. Основные особенности языка рассмотрим на простом примере программирования фрагментированного алгоритма [9, 10] умножения матриц. Текст программы, решающей эту задачу, приведен на рис. 2.

```
program MultMatrix {
variables:
    local {
        int A[m][n], B[n][l], C[n][l][m], D[m][l]
    }
operations:
    F1(in A[k][i], B[i][j]; out C[i][j][k])
        where i: 0..n-1, j: 0..l-1, k: 0..m-1
    {
        C[i][j][k] = A[k][i]*B[i][j];
    };
    F2(in C[1..n][j][i]; out D[i][j])
        where i: 0..m-1, j: 0..l-1
    {
        for(k=0; k < n; k++)
            D[i][j] += C[k][j][i];
    }
control:
    F1 < F2
}
```

Рис. 2. Непроцедурное умножение матриц

Раздел “variables” служит для объявления информационных переменных. В данном случае объявлены четыре массива типа int различной размерности.

Разделы “operations” и “control” позволяют задавать A-блоки. Два раздела используется для того, чтобы полностью отделить вычисления от управления. Всего задано два массовых A-блока: F1 и F2. Область применимости каждого из них описана после ключевого слова “where”.

На множестве A-блоков задан частичный порядок F1 < F2. Для массовых операций такой порядок означает, что как только будут готовы данные для одного из экземпляров зависимого A-блока, он немедленно попадет в очередь на выполнение.

Из примера на рис. 2 следует, что как только экземплярами F1 будут вычислены очередные n элементов матрицы C (имеются в виду элементы из первой размерности), очередной экземпляр A-блока F2 будет добавлен в очередь, не дожидаясь завершения остальных экземпляров A-блока F1.

3.3.Транслятор. Транслятор с языка программирования Аспект имеет классическую архитектуру

и состоит из лексического анализа, синтаксического анализа, контекстного анализа, генератора внутреннего представления и генератора кода. Реализация первых четырех этапов трансляции программы стандартна и далее не рассматривается.

На вход транслятору подается один или несколько файлов, в каждом из которых может быть одна или несколько А-программ, а также специальный конфигурационный файл, в котором для каждого массового А-блока указан требуемый размер А-группы по каждой размерности. На выходе транслятор генерирует сгруппированную программу на языке C++, состоящую из одного главного файла (main.cpp) и нескольких заголовочных файлов — по одному на каждую найденную А-программу.

Каждая А-программа представляется в виде класса, наследуемого от виртуального класса AProgram. А-программы могут быть вложенными (операция А-блока может реализовываться другой А-программой), и наличие единого предка позволяет унифицировать управление деревом А-программ.

Локальные информационные переменные представляются закрытыми (private) переменными класса, а входные/выходные информационные переменные — закрытыми ссылками. Все управляющие переменные также являются закрытыми. Для доступа к информационным переменным автоматически генерируются inline-функции доступа, которые помимо возврата данных могут производить проверку корректности обращения (например, проверку на выход за границы массива).

Для каждой триггер-функции генерируется открытая (public) функция типа bool, а для каждой операции и управляющего оператора — соответствующие открытые процедуры.

3.4. Исполнительная подсистема. Исполнительная подсистема состоит из трех слоев: слоя абстрагирования от операционной системы (ОС), слоя функциональных модулей и слоя А-программы, состоящей из А-групп (рис. 3).

Слой абстрагирования от ОС включает в себя все подпрограммы, в которых используются API (Application Programming Interfaces) операционной системы (создание/уничтожение/синхронизация потоков, функции определения доступных ресурсов, функции для работы со временем и т.д.). Это позволяет переносить исполнительную подсистему из одной ОС в другую заменой лишь одного, выделенного слоя.

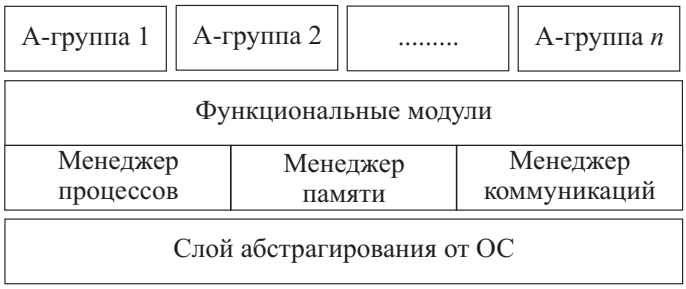

Рис. 3. Архитектура исполнительной подсистемы

Менеджер процессоров управляет распределением работы между процессорами (ядрами). При старте программы создается по одному потоку на каждый процессор (ядро). Каждый поток обращается за очередной порцией работы в общую очередь, которая реализована в виде монитора Хоара. Программа завершается, когда очередь становится пустой.

Менеджер памяти управляет распределением памяти в системе. С точки зрения асинхронной модели с группировкой вычислений его основной задачей является представление всех данных в сгруппированном виде, т.е. если массовый А-блок разбивается на n А-групп по k экземпляров в каждой, то данные, обрабатываемые этим А-блоком, также хранятся в сгруппированном виде (n групп по k элементов в каждой). Это позволяет при необходимости легко переносить одну или несколько А-групп с одного узла на другой (например, для динамической балансировки загрузки).

Менеджер коммуникаций отвечает за прием/передачу сообщений между узлами ЭВМ с распределенной памятью, однако в настоящее время этот модуль не реализован.

4. Результаты экспериментов. Рассмотрим результаты тестирования ядра системы Аспект на примере модельной задачи непроцедурного умножения матриц (текст программы на языке Аспект приведен на рис. 2).

Для тестирования использовалась следующая конфигурация: Athlon 64 X2 3600+ (2*256 L2), 1024 DDR2 RAM, Windows Vista Ultimate. Результаты тестирования для матриц размером 400 на 400 элементов приведены на рис. 4 (отметка Б/А показывает скорость аналогичной программы, разработанной вручную).

Из диаграммы видно, что ускорение достигает значения 1.93 (при размере А-группы в 50 экземпляров). Это объясняется тем, что, хотя каждое ядро имеет отдельный кэш второго уровня, доступ к оперативной памяти производится через общий контроллер. Таким образом, понятно, что эффективное использование кэш-памяти для многоядерных процессоров имеет первостепенное значение.

Скачок при размере А-группы в 400 экземпляров происходит потому, что ее размер совпадает с

исходным размером матрицы, т.е. вся матрица представляет собой одну А-группу. В этой ситуации для второго ядра просто нет работы.

Следует обратить внимание, что ось А-групп не линейна. Например, при размере А-группы в 100 экземпляров всего получается 80 А-групп ($4 \times 4 \times 4 + 4 \times 4$), а при размере в 50 экземпляров — уже 576 А-групп. Таким образом, при существенном увеличении числа групп скорость счета не возрастает (она даже несколько снижается под влиянием кэш-памяти), что свидетельствует об эффективности работы ядра системы.

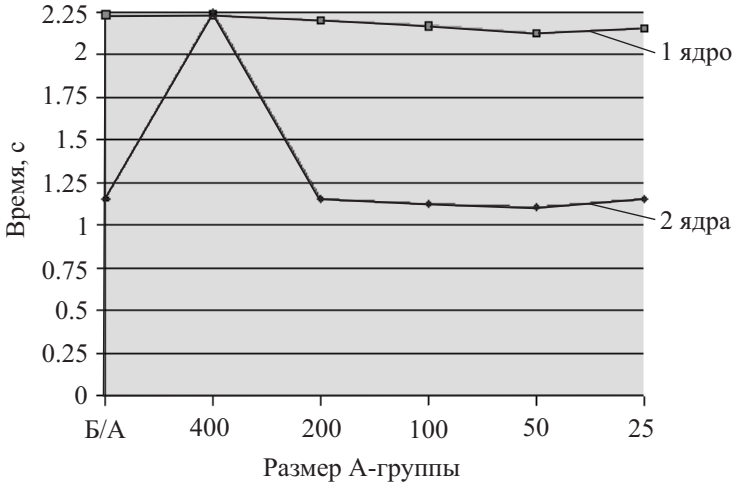

Рис. 4. Результаты непроцедурного умножения матриц с различными размерами А-группы

5. Заключение. В настоящей статье рассмотрена система асинхронного параллельного программирования Аспект, позволяющая программисту по непроцедурному представлению алгоритма автоматически конструировать параллельную программу заданной степени непроцедурности (определяется размерами А-групп). Варьируя степень непроцедурности получаемой программы, программист может в автоматическом/автоматизированном режиме подобрать ее наиболее оптимальное значение для конкретной ЭВМ, на которой необходимо исполнять программу.

Систему программирования Аспект предполагается использовать для решения задач численного моделирования с помощью конечно-разностных методов, а также методов типа “частицы-в-ячейках” [11].

Дальнейшие планы по разработке системы связаны, прежде всего, с доработкой группировки данных и реализацией менеджера коммуникаций, что позволит исполнять программы не только на ЭВМ с общей памятью, но также на кластерах и массивно-параллельных системах. Другим перспективным направлением развития является разработка алгоритмов, оптимизирующих выборку готовых А-групп из очереди согласно некоторым критериям.

СПИСОК ЛИТЕРАТУРЫ

1. *Коновалов Н.А., Крюков В.А., Сазанов Ю.Л.* C-DVM — язык разработки мобильных параллельных программ // Программирование. 1999. № 1. 46–55.
2. *Lastovetsky A.L.* Parallel computing on heterogeneous networks. Hoboken: John Wiley & Sons, 2003.
3. *Васенин В.А., Водомеров А.Н.* Формальная модель системы автоматизированного распараллеливания программ // Программирование. 2007. № 4. 3–19.
4. *Moskovsky A., Roganov V., Abramov S.* Parallelism granules aggregation with the T-system // Proc. of the 9th Int. Conf. on Parallel Computing Technologies (PaCT-2007). Lecture Notes in Computer Science. Vol. 4671. Berlin: Springer, 293–302.
5. *Андрианов А.Н.* Система Норма: разработка, реализация и использование для решения задач математической физики на параллельных ЭВМ. Дисс. ... д-ра техн. наук. Москва, 2001.
6. *Котов В.Е.* О практической реализации асинхронных параллельных вычислений // Системное и теоретическое программирование. Новосибирск: ВЦ СО АН СССР, 1972. 110–125.
7. *Вальковский В.А., Малышкин В.Э.* К уточнению понятия непроцедурности языков программирования // Кибернетика. 1981. № 3. 55.
8. *Малышкин В.Э., Корнеев В.Д.* Параллельное программирование мультикомпьютеров. Новосибирск: Изд-во НГТУ, 2006.
9. *Вальковский В.А., Малышкин В.Э.* Синтез параллельных программ и систем на вычислительных моделях. Новосибирск: Наука, 1988.
10. *Kalgin K.V., Malyshkin V.E., Nechaev S.P., Tschukin G.A.* Runtime system for parallel execution of fragmented subroutines // Proc. of the 9th Int. Conf. on Parallel Computing Technologies (PaCT-2007). Lecture Notes in Computer Science. Vol. 4671. Berlin: Springer, 544–552.
11. *Kraeva M.A., Malyshkin V.E.* Assembly technology for parallel realization of numerical models on MIMD-multicomputers // Future Generation Computer Systems. 2001. **17**, N 6. 755–765.

Поступила в редакцию
30.03.2008