Система асинхронного параллельного программирования «Аспект»

С.Б. Арыков, В.Э. Малышкин

В ИВМиМГ СО РАН ведется разработка системы асинхронного параллельного программирования высокого уровня, предоставляющей пользователю возможность автоматического конструирования параллельных программ по непроцедурному представлению алгоритма. Статья посвящена описанию основных идей, заложенных в данную систему, ее модели вычислений и особенностям практической реализации. Кратко рассматривается язык программирования Аспект, позволяющий представлять алгоритмы в непроцедурной форме.

1. Введение

В настоящее время все больше прикладных специалистов использует численное моделирование для решения практических задач. Поскольку такие специалисты, как правило, недостаточно разбираются в тонкостях параллельного программирования, растет и потребность в системах программирования высокого уровня, которые, по возможности, скрывают от программиста вопросы распределения ресурсов, позволяя сосредоточиться на алгоритме решаемой задачи. С другой стороны, широко доступными становятся большие вычислительные мощности, что также требует разработки инструментов, позволяющих эти мощности эффективно использовать.

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 остается на программисте. Предпринимаются и попытки создания систем с автоматизированным распараллеливанием (например, Т-система[3], НОРМА[4]), однако они имеют недостаточную производительность на реальных вычислительных задачах.

В ИВМиМГ СО РАН ведется разработка системы асинхронного параллельного программирования Аспект, в которой за счет специализации (численное моделирование) 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 непроцедурному представлению автоматически генерировать представление такой степени непроцедурности, которое было бы оптимально в некотором смысле для вычислителя, на котором предполагается исполнять программу.

2. Фрагментированная асинхронная модель вычислений

Асинхронная модель вычислений[5] 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. Для преодоления этого недостатка была разработана специализированная фрагментированная модель вычисле-

ний, которая позволяет плавно изменять уровень непроцедурности[6] программ, тем самым варьируя объем накладных расходов на управление.

2.1 Асинхронная модель вычислений

Асинхронная программа (или *A*-программа) – это конечное множество *A*-блоков, определенных над информационной *IM* и управляющей *CM* памятями.

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 *A*-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 произвольного подмножества *A*-блоков вычисляются их триггер-функции.
2. Выполняется некоторое подмножество *A*-блоков из числа тех, у которых триггер-функция истина. Выполнение *A*-блока состоит в вычислении его операции и управляющего оператора. После завершения выполнения *A*-блоков переходим на шаг 1.
3. Выполнение *A*-программы завершается, когда ни один *A*-блок не выполняется и все триггер-функции всех *A*-блоков ложны.

Для представления массовых вычислений в работе [7] в асинхронную модель было введено понятие массового *A*-блока. В системе Аспект оно также широко используется.

2.2 Фрагментированная модель вычислений

Фрагментированная программа представляет собой конечное множество *A*-фрагментов, определенных над информационной и управляющей памятями, и конструируется из асинхронной программы по следующему алгоритму:

Перебираем все *A*-блоки асинхронной программы и для каждого *A*-блока выполняем следующее:

- Если *A*-блок простой, то он является *A*-фрагментом.
- Если *A*-блок массовый, тогда объединяем экземпляры *A*-блока в группы по *n* экземпляров в каждой. Для каждой группы формируем *A*-фрагмент:
  - Триггер-функция *A*-фрагмента есть конъюнкция триггер-функций всех экземпляров группы;
  - Операция *A*-фрагмента есть новая операция, представляющая собой последовательность операций всех экземпляров группы;
  - Управляющий оператор *A*-фрагмента есть новый управляющий оператор, представляющий собой последовательность управляющих операторов экземпляров группы.

Определенная в столь общем виде, фрагментированная программа мало пригодна для реализации, т.к. в рамках *A*-фрагмента по-прежнему выполняется вычисление управления для каждого экземпляра *A*-блока. Поэтому желаемого сокращения управления вследствие уменьшения асинхронности не происходит.

Чтобы вычислять управление для всего *A*-фрагмента целиком, необходима дополнительная информация о характере вычислений. Ее можно получить, воспользовавшись специализированностью системы Аспект и введя ряд ограничений, приемлемых для предметной области (численного моделирования):

1. Линейность областей применимости массовых *A*-блоков. За счет линейности проверку всего диапазона удается заменить проверкой на границах.
2. Размер области применимости известен до начала вычислений. Для массовых *A*-блоков, область применимости которых до начала вычислений не известна,

A-фрагменты не строятся (точнее, каждый экземпляр такого *A*-блока является *A*-фрагментом).

3. Для массовых *A*-блоков триггер-функция не может зависеть от массовых переменных. Это позволяет исключить ситуации, когда из-за ложности триггер-функции одного экземпляра отменяется выполнение всего *A*-фрагмента целиком.

С учетом введенных ограничений, управляющий оператор для *A*-фрагмента можно построить по следующему алгоритму:

Перебираем все переменные, вычисляемые *A*-фрагментом *A* и для каждой переменной *x*:

1. Формируем множество **M** *A*-фрагментов, которые зависят от *x*;
2. Вычисляем число зависимости μ переменной *x*. Это число показывает, насколько уменьшается зависимость *A*-фрагментов из **M** от *A*, если вычислена *x* (для простых переменных), либо один компонент *x* (для массивов).
3. Для каждого *A*-фрагмента *F* из множества **M**:
   - если *x* – простая переменная, то уменьшаем зависимость *F* на величину μ;
   - если *x* – массив, то определяем пересечение **C** диапазона значений переменной *x*, потребляемого *A*-фрагментом *F* с диапазоном значений переменной *x*, вычисляемого *A*-фрагментом *A* и уменьшаем зависимость *F* на величину $\mu \cdot |\mathbf{C}|$.

Таким образом, программа во фрагментированном виде имеет меньшую асинхронность, но и меньший объем управления, т.е. может исполняться более эффективно. Уровень асинхронности получаемой программы зависит от размера *A*-фрагмента 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 (см. рис. 1).

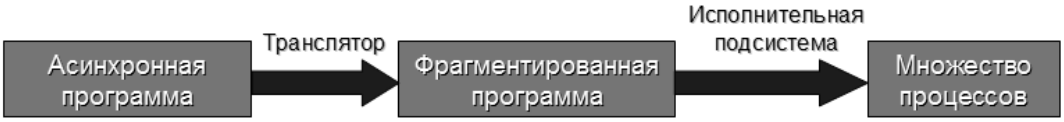

Рис. 1. Процесс разработки в системе программирования Аспект

На вход транслятору подается программа на языке Аспект (асинхронная программа), на выходе генерируется программа на C++ (фрагментированная программа), которая затем отображается исполнительной подсистемой в множество вычислительных процессов.

Рассмотрим каждый компонент подробнее.

3.2 Язык программирования Аспект

Детальное описание языка программирования Аспект выходит за рамки данной статьи, поэтому рассмотрим его основные особенности на простом примере – непроцедурном умножении матриц. Текст программы, решающей эту задачу, приведен на рис. 2.

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 он означает, что как только будут готовы данные для одного из экземпляров зависимого *A*-блока, он немедленно попадет в очередь на выполнение.

```
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

В примере на рис. 2, как только экземплярами *F1* будут вычислены очередные *n* компонент матрицы *C* (имеются в виду компоненты 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 модулей стандартна и далее не рассматривается.

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

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

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 ОС, слоя функциональных модулей и слоя *A*-фрагментов.

Слой абстрагирования от ОС включается в себя все подпрограммы, в которых используются API операционной системы (создание/уничтожение/синхронизация потоков, функции опре-

деления 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.

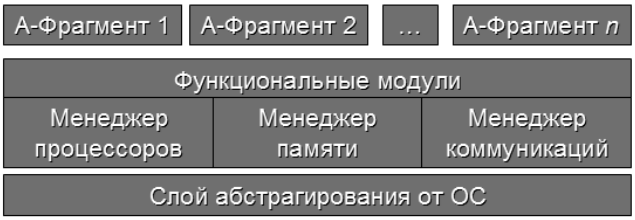

Рис. 3. Архитектура исполнительной подсистемы

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 фрагментированной модели вычислений его основной задаче является представление всех данных во фрагментированном виде, т.е. если массовый *A*-блок разбивается на *n A*-фрагментов по *k* экземпляров в каждом, то данные, обрабатываемые этим *A*-блоком, также хранятся во фрагментированном виде (*n* фрагментов по *k* элементов в каждом). Это позволяется при необходимости легко переносить один или несколько *A*-фрагментов 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

К настоящему моменту разработка системы программирования Аспект завершена не полностью и эксперименты по решению практических задач не проводились. Поэтому рассмотрим результаты тестирования ядра самой системы Аспект на примере модельной задачи непроцедурного умножения матриц (текст программы на языке Аспект приведен в разделе 3.2).

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 *A*-фрагмента 50 экземпляров). Это объясняется тем, что хотя каждое ядро имеет отдельный кэш второго уровня, доступ к оперативной памяти производится через общий контроллер. Таким образом, эффективное использование кэш-памяти для многоядерных процессоров имеет первостепенное значение.

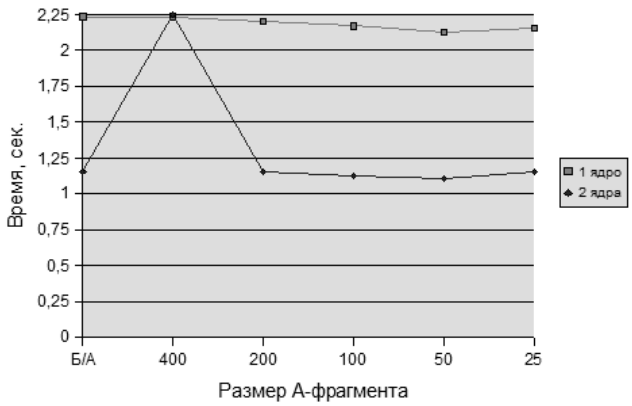

Рис. 4. Результаты непроцедурного умножения матриц с различными размерами *A*-фрагментов

Скачок при размере *A*-фрагментов в 400 экземпляров происходит потому, что его размер совпадает с исходным размером матриц, т.е. вся матрица представляет собой один *A*-фрагмент. В этой ситуации для второго ядра просто нет работы.

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

5. Заключение

В статье рассмотрена система асинхронного параллельного программирования Аспект, позволяющая программисту по непроцедурному представлению алгоритма автоматически конструировать параллельную программу заданного уровня непроцедурности (определяется размером *A*-фрагмента). Варьируя уровень непроцедурности получаемой программы, программист может в автоматическом/автоматизированном режиме подобрать его 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»[8].

Дальнейшие планы по разработке системы связаны, прежде всего, с доработкой фрагментированности по данным 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 *A*-фрагментов из очереди согласно некоторым критериям.

Литература

1. Коновалов Н.А., Крюков В.А., Сазанов Ю.Л. C-DVM – язык разработки мобильных параллельных программ // Программирование. – 1999. – № 1. – С.46-55.
2. Lastovetsky A.L. Parallel Computing on Heterogeneous Networks. John Wiley & Sons, 423 pages, 2003.
3. Васенин В.А., Водомеров А.Н. Формальная модель системы автоматизированного распараллеливания программ // Программирование. – 2007. – № 4. – С. 3-19.
4. Андрианов А.Н. Система Норма: разработка, реализация и использование для решения задач математической физики на параллельных ЭВМ. Дис. д-ра техн. наук. 05.13.11. – Москва, 2001. – 158 с.
5. Котов В.Е. О практической реализации асинхронных параллельных вычислений // Системное и теоретическое программирование. – Новосибирск: ВЦ СО АН СССР, 1972. – С. 110-125.
6. Вальковский В.А., Малышкин В.Э. К уточнению понятия непроцедурности языков программирования // Кибернетика. – 1981. – № 3. – С. 55.
7. Вальковский В.А., Малышкин В.Э. Синтез параллельных программ и систем на вычислительных моделях. – Новосибирск: Наука, 1988. – 129 с.
8. Kraeva M.A., Malyshkin V.E. Assembly technology for parallel realization of numerical models on MIMD-multicomputers // Future generation computer systems, Vol. 17 (2001), No. 6, pp. 755-765.