# ПРОБЛЕМА РАСПРЕДЕЛЕНИЯ РЕСУРСОВ МУЛЬТИКОМПЬЮТЕРА В ТЕХНОЛОГИИ ФРАГМЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

**В.А. Перепелкин**

**Введение**

Работа описывает промежуточные результаты проекта отдела МО ВВС ИВМ и МГ СО РАН «Технология фрагментированного программирования» [1-6], а именно – проблеме автоматизации распределения ресурсов мультикомпьютера при реализации прикладных алгоритмов. Проект в целом посвящен созданию системы параллельного программирования, ориентированной на реализацию больших численных моделей для суперкомпьютеров [2, 6]. Первое запланированное приложение – разработка библиотеки параллельных численных подпрограмм [3-4]. Цель системы – упростить реализацию прикладных численных алгоритмов, и вместе с тем повысить качество их реализации за счёт автоматизации обеспечения таких свойств реализации прикладного алгоритма, как равномерная и полная загрузка процессорных элементов, обеспечение динамической балансировки загрузки, настройка на имеющиеся аппаратные ресурсы, переносимость и т.п. [5]

Одна из проблем, которая неизбежно возникает при разработке параллельных программ вне зависимости от используемых средств программирования – это проблема распределения ресурсов, т.е. отображения прикладного алгоритма – операций и переменных – на процессоры и память параллельного вычислителя. Если вычислитель обладает распределенной памятью (а это так при реализации больших численных моделей для суперкомпьютеров), то необходимо также организовать пересылку сообщений между вычислительными узлами, обеспечивающую передачу управления и необходимых значений переменных прикладного алгоритма.

Проблема распределения ресурсов может быть скрыта от прикладного программиста полностью или частично. В общем случае полная автоматизация распределения ресурсов практически невозможна, т.к. при этом приходится решать классическую задачу планирования, которая является NP-полной. Поэтому в системах программирования более широкого профиля к распределению ресурсов приходится привлекать прикладного программиста, особенно это касается области высокопроизводительных вычислений в распределенной памяти.

С другой стороны, целиком взвалить на программиста решение задачи распределения ресурсов затруднительно. Во-первых, для этого от него потребуется понимание не только особенностей реализуемого прикладного алгоритма, но и архитектуры параллельного вычислителя, которая с каждым годом все более усложняется. Во-вторых, для более или менее сложных численных моделей может потребоваться не менее сложная схема управления распределением ресурсов (например, динамическая балансировка загрузки), запрограммировать которую может оказаться большим или непосильным трудом для программиста, не являющегося специалистом в системном параллельном программировании. В-третьих, задача распределения ресурсов непосредственно связана с особенностями вычислителя, поэтому, решив эту задачу для заданной конфигурации вычислителя, программист «привязывает» задачу к этой конфигурации. Смена вычислителя может потребовать значительной переработки прикладной параллельной программы.

Таким образом, имеется два встречных требования к решению задачи распределения ресурсов. С одной стороны, необходимо участие прикладного программиста для преодоления трудноразрешимых составляющих этой задачи. С другой стороны, требуется максимально разгрузить программиста от выполнения тех функций, которые возможно реализовать автоматически.

Целью настоящей работы является разработка подхода полуавтоматического распределения ресурсов, при котором автоматизируется та часть этой задачи, которая может быть эффективно автоматизирована, а роль прикладного программиста ограничивается только той частью, которая (на данный момент или в принципе) не может быть эффективно автоматизирована.

**1. Модель задачи**

Предлагается следующая модель задачи распределения ресурсов. Эту задачу можно разделить на две части – конструирование схемы распределения ресурсов, и её реализация при исполнении прикладной программы. Очевидно, реализация заданной схемы распределения ресурсов вполне может быть автоматизирована. Конструирование схемы, в свою очередь, также можно разделить на две части – частичное определение схемы и её доопределение. Рассмотрим три примера, иллюстрирующие это разделение.

Пример 1. Программист строит предварительное распределение ресурсов, которое затем оптимизируется автоматически. На программисте лежит ответственность за то, чтобы предоставить начальное приближение распределения ресурсов в окрестности некоторого хорошего субоптимума, но при этом не требуется учитывать мелких реализационных деталей.

Пример 2. Программист определяет схему распределения ресурсов частично, задаёт «опорные точки» распределения, на основе которых схема может быть достроена по относительно простым законам. Т.о. схема оказывается определённой.

Пример 3. Программист описывает дополнительную информацию об алгоритме, описывающую структуру или свойства последнего. На основе этой информации автоматически строится план по каким-либо частным алгоритмам.

Во всех трех случаях автоматизируется та часть работы, которая вполне может быть выполнена автоматически, а ответственность за качество результирующего распределения ресурсов остается на прикладном программисте.

Для того, чтобы прикладной программист мог сформулировать свои решения о схеме распределения ресурсов в языке программирования должны быть соответствующие описательные средства. Можно выделить две группы средств – для описания принципиальной схемы распределения ресурсов, и для описания конкретной схемы.

Пример языка параллельного программирования, позволяющего декларативно описывать схему распределения ресурсов – Open MultiProcessing (OpenMP). OpenMP расширяет традиционные языки программирования C и Фортран конструкциями разметки кода на секции. Например, возможно обозначить цикл в прикладной программе как параллельный и выбрать схему распределения итераций цикла по ядрам. При этом собственно распараллеливание будет выполнено автоматически. Тут ответственность за выбор распределения ресурсов (в данном случае ядер) лежит на программисте, а реализация выбранной схемы – на системе OpenMP.

Схема распределение ресурсов не обязательно конструируется статически. Она может быть частично определена при компиляции и доопределяться динамически по мере исполнения программы. Пример – программы, реализующие динамическую балансировку нагрузки процессоров.

В соответствии с предлагаемой моделью можно сформулировать два основных требования к языкам и системам параллельного программирования с точки зрения решения задачи распределения ресурсов. Несоответствие этим требованиям будет означать либо плохое качество автоматического распределения ресурсов, либо узость круга допустимых прикладных алгоритмов, либо необходимость конструировать и/или реализовывать эффективное распределение ресурсов вручную. Вот эти требования:

1. язык программирования должен содержать средства для удобного описания схемы распределения ресурсов. Чем выразительнее эти средства, тем более сложные схемы сможет описывать прикладной программист. Естественно, эти средства должны быть подкреплены соответствующими системными алгоритмами, поддерживающими их.
2. представление алгоритма должно быть пригодным для автоматизации манипуляций над ним как во время компиляции, так и во время исполнения.

Поясним второе требование. Если рассмотреть, например, Message Passing Interface (MPI), то с точки зрения системы программирования параллельная программа представляется как набор процессов, обменивающихся сообщениями. Такая декомпозиция позволяет осуществлять лишь относительно грубую статическую балансировку нагрузки процессоров путем назначения MPI-процессов на вычислительные узлы. Вследствие того, что с процессом связано его адресное пространство затруднительно осуществлять, например, миграцию процессов с узла на узел в процессе вычислений. Ситуация усугубляется также тем, что каждый MPI-процесс – это черный ящик, поведение которого непредсказуемо для системы MPI. Вследствие этого невозможно оценить, например, какой MPI-процесс будет больше загружать процессор или память, а какой – меньше; или когда, кем и кому будут посланы сообщения и какого объема. Таким образом, можно заключить, что представление алгоритма, используемое в MPI обладает описанными недостатками. Отметим, что подобные цели и не преследовались разработчиками MPI.

Другой пример, раскрывающий второе требование – это система программирования Charm++. В этой системе параллельная программа представляется как множество объектов – «чаров». Каждый чар является более мелкой гранулой параллелизма, чем MPI-процесс, и на одном вычислительном узле могут присутствовать сотни чаров. Каждый чар поддержан со стороны прикладного программиста операциями сохранения и восстановления состояния, что позволяет прозрачно для программиста организовывать динамическую балансировку загрузки процессоров (путем миграции чаров). Это пример более удобного, чем в случае MPI, представления алгоритма с точки зрения автоматизации распределения ресурсов.

**2. Технология фрагментированного программирования**

Технология фрагментированного программирования разрабатывалась в соответствии с представленным взглядом на проблему распределения ресурсов. Соответственно, было выбрано представление алгоритма, пригодное для различных манипуляций. Это представление является модификацией вычислительной модели [1] и называется фрагментированным алгоритмом. Фрагментированный алгоритм (ФА) – это двудольный ориентированных граф, состоящий из фрагментов данных (ФД) и фрагментов вычислений (ФВ). ФД – это агрегированные переменные единственного присваивания, например подматрицы некоторой матрицы. ФД представляют переменные прикладного алгоритма. ФВ – это операции без побочных эффектов над ФД. Дуги ФА определяют, какие ФД являются входными, а какие – выходными для данного ФВ. ФВ вычисляет выходные ФД из входных. В целом это известное представление алгоритма как множества переменных и операций над ними с той разницей, что значениями ФД являются не отдельные величины, а агрегаты, например фрагменты

матрицы. В реализации ФД обычно реализуется блоком памяти, а ФВ – вызовом процедуры на традиционном языке программирования.

Множества ФД и ФВ описываются с помощью операторов, аналогичных операторам суперпозиции, примитивной рекурсии и минимизации из теории рекурсивных функций, что обеспечивает тьюринг-полноту такого представления.

Язык фрагментированного программирования LuNA (Language for Numerical Algorithms) [7] располагает средствами во-первых, для описания ФА, а, во-вторых, содержит средства для описания т.н. «рекомендаций» – информации о желаемой схеме исполнения ФА. Рекомендации можно разделить на две группы – низкоуровневые, которые, по сути, представляют собой язык для описания конкретной схемы распределения ресурсов, и высокоуровневые, предназначенные для описания принципиальной схемы распределения ресурсов. На основе высокоуровневых рекомендаций низкоуровневые достраиваются автоматически, хотя программист может определять и низкоуровневые рекомендации, если затрудняется с помощью высокоуровневых сформулировать желаемую схему распределения ресурсов. Совокупность ФА и рекомендаций называется фрагментированной программой (ФП).

ФА является удобным представлением алгоритма с точки зрения автоматизации распределения ресурсов. Во-первых, ФВ – это операции без побочных эффектов, что означает возможность независимого исполнения ФВ, для которых вычислены значения их входных ФД, возможность осуществлять миграцию ФД и ФВ с узла на узел без влияния на значения, вычисляемые программой и прозрачно для программиста. Во-вторых, системе явно виден ход выполнения программы – какие значения ФД уже вычислены, какие ФВ исполняются в настоящий момент. Это что позволяет (частично) прогнозировать и регулировать загрузку процессоров. В-третьих, явно видны информационные связи между ФВ, что позволяет системе оценивать, где, когда и в каком объеме понадобятся коммуникации. В-четвертых, собственно ФА не содержит никакой информации о распределении ресурсов, эта информация содержится только в рекомендациях. Это позволяет решать задачу распределения ресурсов независимо от других подзадач при разработке параллельной программы.

Содержащиеся в ФП рекомендации также отвечают сформулированным ранее требованиям. Именно, они служат для выражения информации о том, как прикладной программист видит исполнение своего ФА. Основой тут являются низкоуровневые рекомендации, которые выражают конкретную схему исполнения. Рекомендации такого вида подкреплены системными алгоритмами, их реализующими. Уже на этом уровне программист может сформулировать требуемое поведение своей программы (в смысле распределения ресурсов), и при этом у него нет необходимости его программировать. Рекомендации высокого уровня используются для более удобного описания схем распределения ресурсов в тех случаях, когда система содержит алгоритм построения низкоуровневых рекомендаций на основе той или иной информации. Три таких примера были рассмотрены в начале раздела 1. По мере развития системы фрагментированного программирования она будет накапливать новые алгоритмы формирования низкоуровневых рекомендаций на основе высокоуровневых, тем самым совершенствуя выразительные средства рекомендаций и повышая уровень программирования для прикладного программиста.

В соответствии с вышесказанным на данном уровне проработки настоящей темы окончательно сформулировать язык рекомендаций затруднительно, однако имеется промежуточная версия этого языка, которая используется для экспериментов. Важный вопрос о языке рекомендаций – это вопрос его полноты, которая означает возможность любое мыслимое поведение программы (в смысле распределения ресурсов). Этот вопрос требует дальнейших исследований.

Для формирования представления о языке рекомендаций ниже приведены два примера – с низкоуровневыми и высокоуровневыми рекомендациями из текущей версии языка и системы LuNA.

### 3. Язык рекомендаций

Определим рекомендации как отображение L множества ФВ на единичный отрезок P. При реализации ФА этот отрезок P разбивается на несколько равных частей по числу процессоров, выделенных на решения задачи, и каждому процессору выделяются для исполнения те ФВ, которые были отображены на его часть отрезка.

Пример. Пусть требуется задать распределение ресурсов для некоторой одномерной явной конечно-разностной трехточечной схемы. Пусть этот алгоритм содержит множество ФВ a[i,j], где i – это номер итерации, а j – номер ФВ внутри итерации. Пусть j принимает значения от 0 до N-1. Тогда зададим отображение L(a[i,j]) как j/N. В результате, ФВ будут отображены на процессоры в соответствии с известной схемой пространственной декомпозиции.

Отметим, что тут от прикладного программиста не требуется программировать никаких коммуникаций или вообще иметь дело с аппаратными ресурсами – он просто задаёт отображение L в виде формулы, а остальную часть работы делает система программирования. Отображение L, таким образом, является средством, с помощью которого программист описывает желаемый способ распределения ресурсов ФА.

На практике вместо отрезка P используется единичный гиперкуб конечной размерности (размерность выбирается в зависимости от ФА). Расчет делается на то, что гиперкуб, как простая структура, может быть

легко автоматически отображен на вычислитель стандартных сетевых топологий (кластер, тор) регулярным образом, а сложности структуры ФА скрыты прикладным программистом в отображении L.

Пример использования высокоуровневых рекомендаций.

Допустим, в предыдущем примере, отображение L задано частично: L(a[i,0]) = 0; L(a[i,N-1]) = 1. Тогда отображение остальных ФВ можно доопределить автоматически на основе анализа информационных зависимостей по какой-либо интерполяционной формуле. Результат получится такой же, как и в предыдущем примере. Это лишь один из примеров того, как рекомендации могут быть заданы на более высоком уровне, а затем быть достроены (или модифицированы) автоматически. Тут в качестве высокоуровневых рекомендаций выступает частичное определенное отображение L, а в качестве низкоуровневых – полностью определенное L.

**Заключение**

Описанный подход к полуавтоматическому распределению ресурсов был реализован в рамках языка и системы фрагментированного программирования LuNA. При тестировании производительности этой системы на ряде тестов, реализующих матрично-векторные операции, была получена производительность, не уступающая ручной реализации тех же алгоритмов в MPI (см. [7]). Эти примеры подтверждают применимость описанного подхода по крайней мере для некоторого круга численных алгоритмов.

В дальнейшем планируется исследовать выразительные средства рекомендаций и, соответственно, расширять базу алгоритмов, поддерживающих эти средства.

ЛИТЕРАТУРА:


1. В.А.Вальковский, В.Э.Малышкин. Синтез параллельных программ и систем на вычислительных моделях. – Наука, Новосибирск, 1988, 128 стр.
2. Kraeva M.A., Malyshkin V.E. Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers. – In the Int. Journal on Future Generation Computer Systems. Vol. 17 (2001), No. 6, pp. 755-765
3. S.Kireev, V.Malyshkin. Fragmentation of numerical algorithms for parallel subroutines library – The Journal of Supercomputing, Springer, Volume 57, Number 2 / August 2011 pp. 161-171
4. S. Kireev, V. Malyshkin, H. Fujita. The LuNA Library of Parallel Numerical Fragmented Subroutines // PaCT-2011 proceedings, Springer, LNCS 6873 (2011), pp. 290-301
5. V. Malyshkin, V. Perepelkin. Optimization methods of parallel execution of numerical programs in the LuNA fragmented programming system // The Journal of Supercomputing, Special issue on Enabling Technologies for Programming Extreme Scale Systems, Volume 61, Number 1 (2012), pp. 235-248, DOI: 10.1007/s11227-011-0649-6
6. V.Malyshkin. Assembling of Parallel Programs for Large Scale Numerical Modeling. – In the Handbook of Research on Scalable Computing Technologies. IGI Global, USA, 2010, 1021 pp, Chapter 13, pp. 295 – 311. ISBN 978-1-60566-661-7
7. V.E. Malyshkin, V.A. Perepelkin. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem - In: Proceedings of the 11th Conference on Parallel Computing Technologis, LNCS 6873 - pp. 53-61, Springer, 2011