Технология фрагментированного программирования*

В.Э. Малышкин

Институт вычислительной математики и математической геофизики СО РАН

Новосибирский государственный университет

Кратко представлена технология фрагментированного программирования и реализующие её язык и система фрагментированного программирования LuNA, разрабатываемые в ИВМиМГ СО РАН. Технология ориентирована на поддержку разработки параллельных программ, реализующих большие численные модели, и их исполнения на суперкомпьютерах. Система LuNA автоматически обеспечивает такие динамические свойства параллельных программ как динамическая настройка на все доступные ресурсы, динамическая балансировка нагрузки, учет динамики поведения моделируемого явления и т.п.

1. Введение

В течении последних 15 лет в ИВМиМГ СО РАН велись работы по созданию методов и средств параллельной реализации больших численных моделей на суперкомпьютерах, а также параллельной реализации таких моделей. Накопленный опыт позволил сформировать идеи технологии фрагментированного программирования. Основная проблема параллельной реализации больших численных моделей состоит в том, что необходимая сложность программирования заметно превосходит квалификацию программистов, работающих обычно в численном моделировании, так как в программе моделирования понадобилось реализовывать динамические системные функции.

Чтобы преодолеть эту проблему, нужна поддержка процесса параллельного программирования таких моделей. Необходимая технология – технология фрагментированного программирования и реализующие её язык и система программирования LuNA - разработаны в ИВМиМГ СО РАН.

Поддерживая технику разработки программ численного моделирования также хотелось бы обеспечивать:

- должную степень непроцедурности [1] представления алгоритма в параллельной программе (не знать MPI, свойства вычислителя и его коммуникационной сети, методы и средства параллельного программирования и т.д.),
- неизменность алгоритма, его независимость от оборудования (раздельное описание алгоритма и реализующей его программы), автоматическая генерация фрагментированной программы, что требуется для обеспечения её переносимости,
- автоматическое включение реализации динамических свойств в прикладные параллельные программы, такие как динамическая балансировка нагрузки, настраиваемость на все доступные ресурсы вычислителя, динамическое распределение ресурсов, выполнение коммуникаций на фоне счета, учет поведения моделируемого явления.

Теоретическую базу проекта LuNA составляет теория синтеза параллельных программ на вычислительных моделях [2]. В проекте системы LuNA учтен опыт разработки как больших численных моделей [3,4, 13, 14], так и различных систем сборочного программирования в мире [5-8]. Текущие технологические результаты представлены в настоящей статье. Теоретические аспекты проекта не рассматриваются. Более ранние результаты опубликованы в [9-11].

* Работа выполнена при частичной поддержке РФФИ, грант 10-07-00454-а

2. Идеология системы фрагментированного программирования LuNA

Перечисленные мотивы, и ряд других, после общего анализа трансформировались в следующие исходные проектные решения в системе фрагментированного программирования LuNA.

2.1 Основные проектные решения

1). Технология фрагментированного программирования поддерживает процесс сборки целой программы из фрагментов вычислений (модулей, процедур, их входных/выходных фрагментов данных и т.п.) и ее исполнение.
2). Каждый фрагмент вычислений - независимая единица программы (рис. 1), содержит описание входных/выходных переменных и кода (модуля, процедуры) фрагмента.
3). Фрагментированная программа – это рекурсивно перечислимое множество фрагментов вычислений и их входных/выходных переменных.

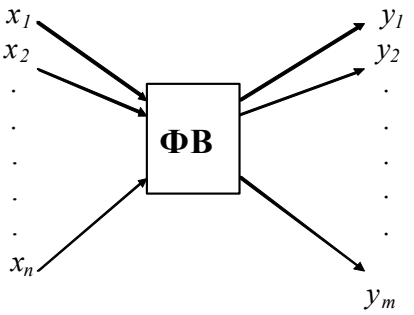


Рис. 1. Фрагмент вычислений

Таким образом, фрагментированная программа определяется как множества переменных (фрагментов данных) и фрагментов вычислений. К фрагменту вычислений можно обращаться по-разному, например, как к обычной процедуре в последовательном языке программирования

4). В отличие от технологии модульного программирования, в ходе исполнения фрагментированная структура программы сохраняется. Каждый фрагмент вычислений определит в ходе исполнения независимо исполняющийся процесс программы, взаимодействующий с другими процессами.
5). Следуя С.Клини [12], в качестве базового представления алгоритма взято рекурсивно перечислимое множество функциональных термов. Фрагментированный алгоритм при необходимости извлекается алгоритмом вывода из множества фрагментов вычислений.

2.2 Исполнение фрагментированной программы

Базовый алгоритм:

- Фрагмент вычислений исполняется, если все его входные переменные получили значения
- После выполнения фрагмента вычислений получают значения его выходные переменные.
- Алгоритм может реализоваться либо управлением в сгенерированной программе, либо run-time системой. В системе LuNA для обеспечения необходимой степени асинхронности выбрано исполнение фрагментированной программы run-time системой.

Эти правила определяют асинхронное исполнение фрагментированной программы, в которой порядок исполнения фрагментов вычислений определяется лишь информационными зависимостями между ними. При исполнении фрагментированной программы run-time система ищет лучшие способы исполнения фрагментированного алгоритма.

3. Шаги разработки ФП

Фрагментированная программа разрабатывается в несколько последовательных этапов.

- Разработка исходного алгоритма решения задачи.
- Фрагментация исходного алгоритма. Фрагментация рассматривается здесь как универсальный способ распараллеливания алгоритмов, что позволяет поддержать его реализацию системой программирования (примеры фрагментации приведены в разделе 4). Фрагментация нередко является очень сложной работой. Например, фрагментация алгоритма прогонки [15] заняла 2 года и завершилась защитой кандидатской диссертации. На тестах получено ускорение исполнения алгоритма прогонки в 6000 раз на 8000 процессоров [15].
- Описание фрагментированной программы на языке LuNA. Входной язык системы LuNA устроен просто, в него в основном включены средства для определения множеств фрагментов данных и вычислений, синтаксис языка не очень интересен. Пример фрагментированной программы на языке LuNA можно видеть в разделе 4.1.
- Компиляция и анализ фрагментированного алгоритма. Генерация платформо-ориентированной фрагментированной программы.
- Исполнение фрагментированной программы.

4. Примеры фрагментированных алгоритмов

Несколько примеров поясняют технологию фрагментированного программирования и проблемы разработки системы LuNA.

4.1 Исходный алгоритм умножения квадратных матриц

На рис.2 представлен исходный алгоритм умножения квадратных матриц. Вычисления проводятся по формулам:

$$c_{i,j} = \sum_{k=1}^{N} a_{i,k} \times b_{k,j}, \quad i, j = 1, 2, \ldots, N.$$

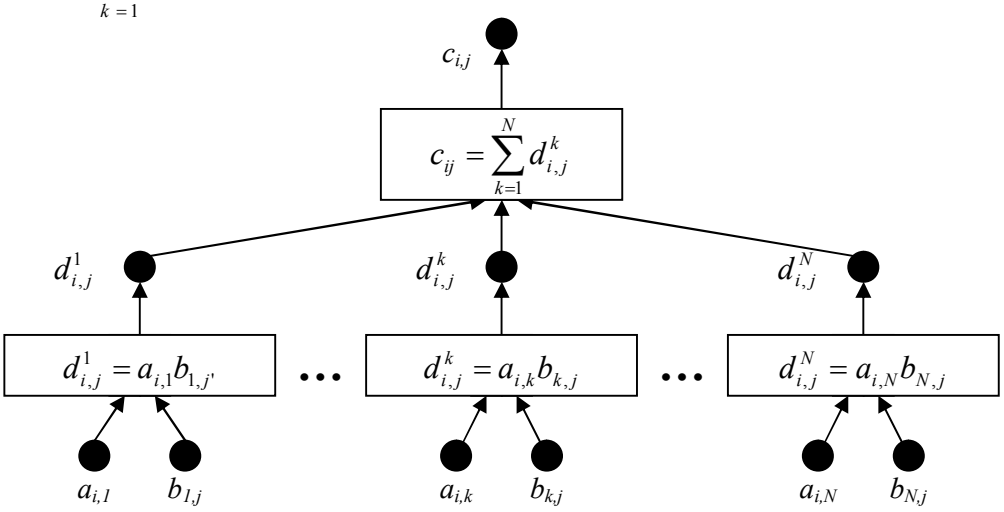


Рис. 2. Множество функциональных термов, представляющих исходный алгоритм

Так как множество функциональных термов хранить нехорошо, то в языке LuNA вместо множества функциональных термов определяются множества фрагментов вычислений и фрагментов данных, а нужные термы конструируются из них по мере необходимости алгоритмом вывода.

4.2 Фрагментированный алгоритм умножения квадратных матриц

В системе LuNA исходный алгоритм может быть запрограммирован как фрагментированный, т.е. каждая операция $c_{ij} = \sum_{k=1}^{N} d_{i,j}^{k}$ объявлена фрагментов вычислений (и будет реализо-

ваться как независимый процесс программы), переменные a_{il}, b_{lj}, c_{ij}, d_{ij} объявлены фрагментами данных. Но такая фрагментированная программа будет исполняться с большим замедлением, примерно с 1000 кратным, по сравнению с программой с использованием MPI из-за больших расходов на реализацию управления. Поэтому для численных алгоритмов, отличающихся высоко регулярностью, в процессе фрагментации проводится агрегация и переменных и операций. На рис. 3 представлена схема фрагментации алгоритма умножения матриц, которая показывает способ агрегации данных и вычислений, а на рис. 4 – фрагментированный алгоритм. Здесь:

$$C_{I,J} = \sum_{L=1}^{K} A_{I,L} \times B_{L,J}, \qquad C_{I,J}, A_{I,L}, B_{L,J}, \quad I,J,L = 1,2,...,K$$

Даже в таком простом примере, как алгоритм умножения матриц, информационные зависимости не определяют хорошего исполнения фрагментированной программы. Например, если для всех I и J выполнить сначала все, кроме K-го, фрагменты вычислений $D^L_{I,J} = A_{I,L}B_{L,J}$, а потом исполнить все K-ые фрагменты, то память вычислителя должна будет хранить все выработанные, но своевременно не потребленные, промежуточные данные. Объём хранимых промежуточных данных в K раз большие чем может понадобиться при хорошей организации вычислений. В результате будет ограничен размер решаемой задачи и замедлится исполнение программы. Поэтому основными задачами, решаемыми run-time системой, являются распределение ресурсов и выбор наиболее подходящего очередного фрагмента вычислений на исполнение.

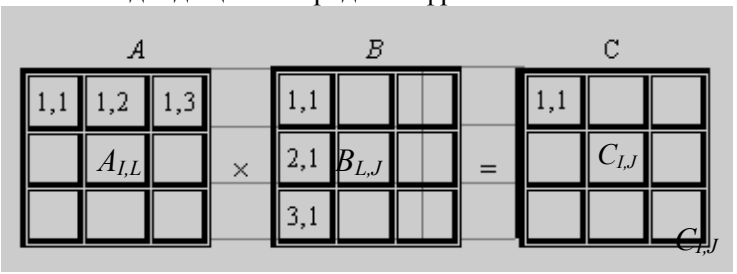


Рис. 3. Агрегация данных и фрагментов вычислений исходного алгоритма

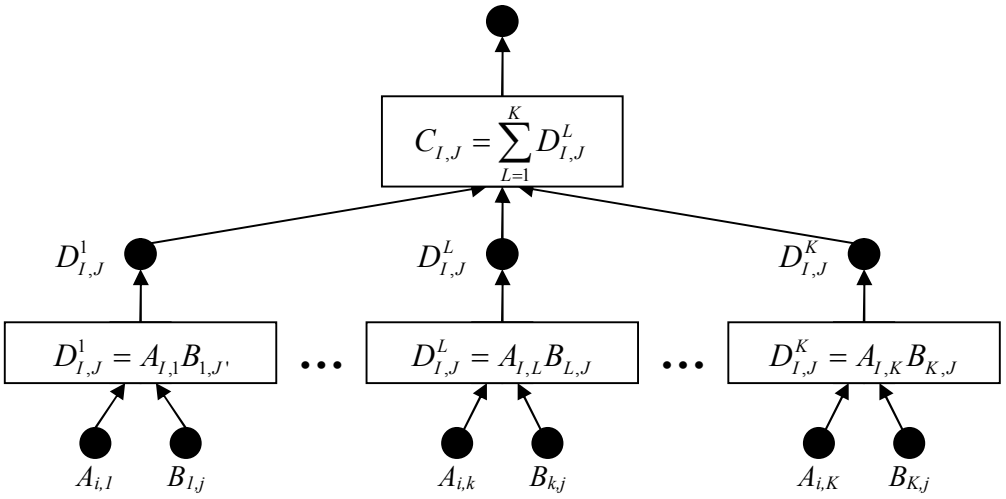


Рис. 4. Фрагментированный алгоритм

В качестве переменных фрагментированной программы используются агрегаты переменных исходного алгоритма, составляющих подматрицы исходной матрицы. Аналогично, код фрагмента вычислений является агрегатом кодов операций исходного алгоритма. В системе LuNA конструируются программы, в которых размер фрагментов данных является входным параметром. Ниже в качестве примера приведена часть фрагментированной программы умножения матриц, записанная в языке LuNA:

Множество фрагментов данных:

```
df a[i,k] := block(4*M*M) | i=0..K-1, k=0..K-1;
df b[k,j] := block(4*M*M) | k=0..K-1, j=0..K-1;
```

```
df c[i,j] := block(4*M*M) | i=0..K-1, j=0..K-1;
df d[i,j,k] := block(4*M*M) | i=0..K-1, j=0..K-1, k=0..K-1;
```

Множество фрагментов вычислений

```
cf initc[i,j] := proc_zero<M,M> (out: c[i,j])
| i=0..K-1, j=0..K-1;
cf mul[i,j,k] := proc_mmul<M,M,M> (in: a[i,k],b[k,j];
out: d[i,j,k]) | i=0..K-1, j=0..K-1, k=0..K-1;
cf sum[i,j,k] := proc_add<M,M> (in: d[i,j,k],c[i,j]; out: c[i,j])
| i=0..K-1, j=0..K-1, k=0..K-1;
```

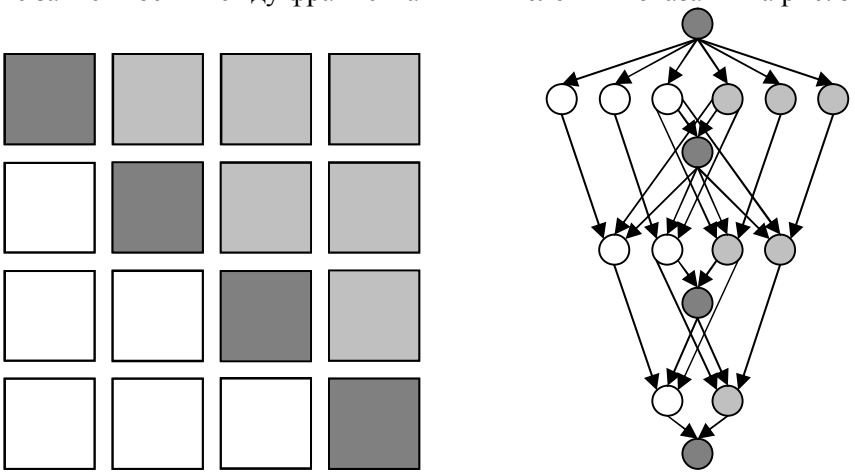
4.3. LU-разложение

LU-разложение преобразует квадратную матрицу A по формулам $l_{i,j} = a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} u_{k,j}$,

$u_{i,j} = \frac{1}{l_{i,i}} \left(a_{i,j} - \sum_{k=1}^{i-1} l_{i,k} u_{k,j} \right)$ к виду $A=LU$, где:

$$L = \begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \ldots \\ l_{31} & l_{32} & l_{33} \\ \ldots & & \end{pmatrix} \qquad U = \begin{pmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \ldots \\ 0 & 0 & 1 \\ \ldots & & \end{pmatrix}$$

Матрица A делится на фрагменты данных (рис. 5*a*), каждый фрагмент данных – подматрица матрицы A, для обработки каждого фрагмента данных формируется фрагмент вычислений. Фрагменты вычислений должны выполняться в следующем порядке (таковы информационные зависимости): вначале исполняется фрагмент (1,1), затем могут выполняться все фрагменты первого столбца и первой строки, затем может исполняться фрагмент (2,2) и т.д. Информационные зависимости между фрагментами вычислений показаны на рис. 5.*b*.



a) *b)*

Рис. 5. Фрагментация вычислений *a)* алгоритма LU-разложения и информационные зависимости между фрагментами вычислений *b)*

Этот порядок нехорош тем, что создает неравномерную нагрузку на вычислитель: есть моменты времени, когда только один фрагмент вычислений готов к исполнению, и есть моменты времени, когда много более одного фрагмента готовы исполняться.

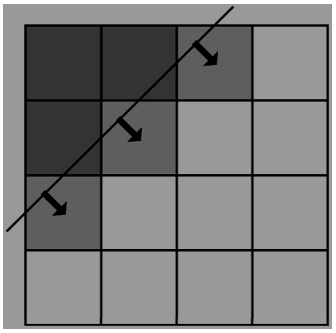


Рис. 6. Исполнение фрагментов вычислений гиперплоскостями

Лучше было бы организовать исполнение фрагментов вычислений по гиперплоскостями (рис. 6), при котором формируется более равномерная нагрузка процессоров вычислителя.

4.3. Неравномерность загрузки процессоров в модели эволюции облака пыли

Параллельная реализация модели эволюции протопланетного диска описана в [13, 14]. На рис. 7 показана неравномерность распределения нагрузки на узлы мультикомпьютера в процессе моделирования. Каждый прямоугольник изображает фрагмент вычислений и потребляемые им ресурсы. Динамическая балансировка нагрузки узлов вычислителя планируется run-time системой и реализуется миграцией фрагментов вычислений с перегруженного узла на соседние, менее загруженные. На этом основана автоматическая реализация динамических свойств фрагментированной программы.

Планирование, точный расчет балансировки нагрузки не делается, а моделируется физический процесс диффузии в жидких и/или газообразных средах. Средой здесь является множество фрагментов вычислений. Точный расчет балансировки нагрузки в условиях динамически меняющегося состояния системы взаимодействующих процессов фрагментированной программы не имеет смысла, и, как минимум, не технологичен.

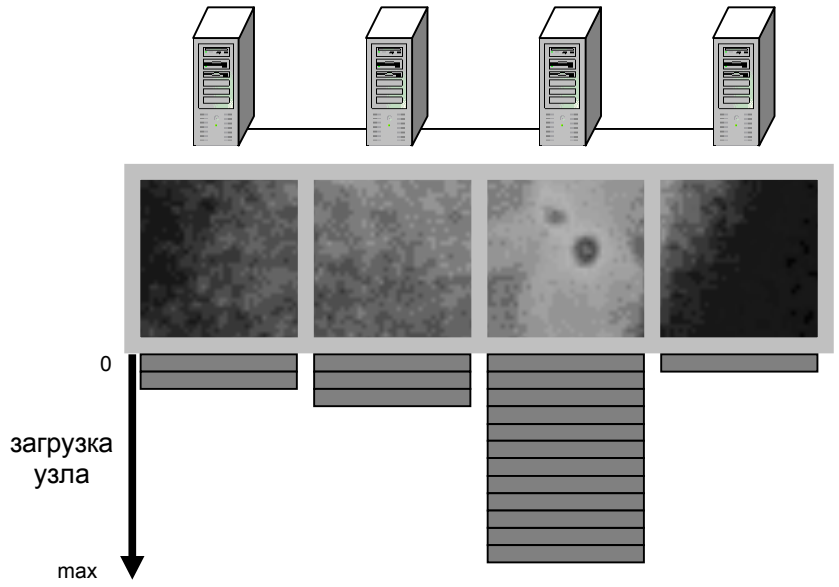


Рис. 7. Неравномерная нагрузка узлов вычислителя

Завершая обсуждение фрагментации численных алгоритмов, необходимо указать на одну важную качественную характеристику исполнения фрагментированной программы (рис. 8).

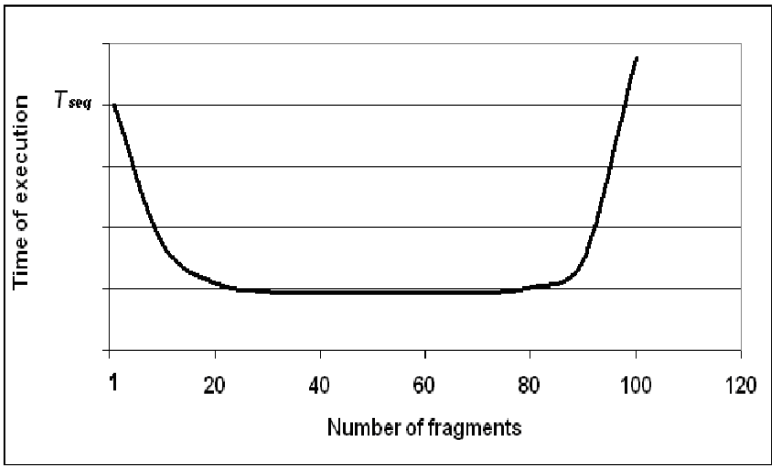


Рис. 8. Качественный график времени исполнения фрагментированной программы

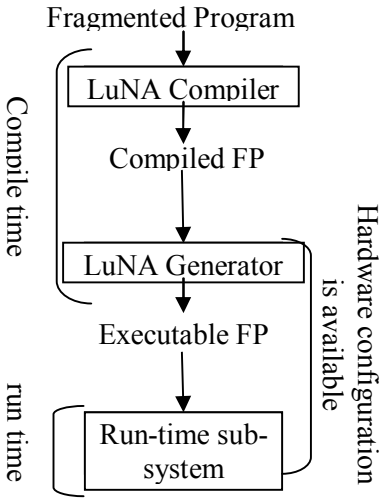


Рис. 9. Компоненты системы LuNA

Фрагментированная программа исполнялась в одном узле, сравнение производилось с последовательной программой. На рисунке T_{seq} – это время исполнения фрагментированной программы, состоящей из одного фрагмента, т.е. это время исполнения последовательной программы. Затем число фрагментов данных увеличивалось (соответственно, размер фрагментов данных уменьшался), при этом наблюдалось уменьшение времени исполнения программы. Минимальное время исполнения программы получалось, когда фрагмент вычислений со всеми обрабатываемыми им фрагментами данных попадал целиком в кэш-память. Увеличение времени исполнения программы начиналось с ростом числа фрагментов данных и вычислений, что приводило к увеличению накладных расходов на реализацию управления и динамического распределения ресурсов.

5. Язык и система фрагментированного программирования LuNA

Входной язык LuNA - теоретико-множественный, единственного присваивания и единственного исполнения фрагментов вычислений. Фрагменты данных и вычислений задаются рекурсивно перечислимыми множествами с использованием индексных выражений. Управление в LuNA-программе задается отношением частичного порядка на множестве фрагментов вычислений.

Отношение соседства на множестве фрагментов данных определяет, какие фрагменты данных должны размещаться в одном либо в соседних узлах. Дополнительно имеются операторы-рекомендации по распределению ресурсов вычислителя, по определению требуемого порядка исполнения фрагментов вычислений. Средства задания приоритетов исполнения фрагментов вычислений используются run-time системой для выбора наиболее подходящего фрагмента на исполнение.

Функциональная структура системы фрагментированного программирования LuNA представлена на рис. 9. Язык, некоторые теоретические и технологические аспекты системы программирования LuNA описаны в ряде публикаций [10, 11] .

LuNA имеет три уровня преобразования фрагментированного алгоритма в программу: Компиляция, Генерация и Исполнение:

- Компилятор принимает решения (и вносит их в формируемую программу), которые можно принять, используя только информацию о свойствах алгоритма.

- Генератор принимает решения, которые зависят от свойств конкретного вычислителя (количество и типы доступных ресурсов, производительность ресурсов, нумерация узлов и т.п.), на котором программа должна исполняться.

- Run-time система принимает те решения, которые могут быть сделаны только динамически, в ходе вычислений. В их числе выбор фрагмента вычислений на исполнение, динамическая балансировка нагрузки узлов вычислителя, динамическое распределение ресурсов, включая назначение процессора для исполнение фрагмента вычислений и многое другое.

В системе LuNA есть еще один компонент – профилировщик, который собирает реальную информацию о ходе исполнения фрагментированной программы. Эта информация используется затем для улучшения последующих исполнений программы.

6. Родственные работы

Общее представление о работах, посвященных сборке программ из готовых фрагментов, дают проекты [5-8]. Проект *Charm* [5] за долгое время развития стал хорошо известной системой программирования. Её основным недостатком является отсутствие глобальной оптимизации исполнения и не соответствующие входной язык и технология программирования, не позволяющие использовать в полной мере достоинства системы программирования, что свойственно и другим проектам

Литература

1. Вальковский В.А., Малышкин В.Э. К уточнению понятия непроцедурности языков программирования. // Кибернетика, № 3, 1981г. стр. 55
2. Вальковский В.А., Малышкин В.Э. Синтез параллельных программ и систем на вычислительных моделях. Издат. "Наука", Сибирское отделение, Новосибирск, 1988г. 129 стр.
3. Kraeva M.A., Malyshkin V.E. Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers. // International Journal on Future Generation Computer Systems. Vol. 17 (2001), No. 6, pp. 755-765.
4. V.Malyshkin. Assembling of Parallel Programs for Large Scale Numerical Modeling. IGI Global, USA, 2010, 1021 pp. The Handbook of Research on Scalable Computing Technologies. Chapter 13, pp. 295 – 311. ISBN 978-1-60566-661-7
5. Charm++**.** URL: http://charm.cs.uiuc.edu/papers
6. ProActive Parallel Suite. URL: http://proactive.inria.fr/
7. S-Net home page. — URL: http://www.snet-home.org/
8. Berzins M., Meng O., Schmidt J., Sutherland J.C. DAG-Based Software Frameworks for PDEs. URL: http://www.csafe.utah.edu/pdf/papers/2011_Berzins_Meng_Schmidt_Sutherland_(DAG-Based_Software_Frameworks_for_PDEs).pdf
9. Kireev S., Malyshkin V. Fragmentation of numerical algorithms for parallel subroutines library // The Journal of Supercomputing, Springer, Volume 57, Number 2, August 2011, pp. 161-171
10. Kireev S., Malyshkin V., Fujita H.. The LuNA Library of Parallel Numerical Fragmented Subroutines // PaCT-2011 proceedings, Springer, LNCS 6873 (2011), pp. 290-301
11. Malyshkin V., Perepelkin V. Optimization Methods of Parallel Execution of Numerical Programs in the LuNA Fragmented Programming System. // The Journal of Supercomputing. pp. 1-14. DOI: 10.1007/s11227-011-0649-6
12. Клини С. Введение в метаматематику. Иностранная литература, М.: 1957
13. Киреев С.Е. Параллельная реализация метода частиц в ячейках для моделирования задач гравитационной космодинамики // Автометрия. 2006. №3. с. 32-39.
14. Kireev S.E. A Parallel 3D Code for Simulation of Self-gravitating Gas-Dust Systems // PaCT-2009 Proceedings, Springer, LNCS 5698 (2009), pp. 406–413.
15. Terekhov A.V. Parallel Dichotomy Algorithm for solving tridiagonal system of linear equations with multiple right-hand sides // Parallel Computing, Volume 36, Issue 8, 2010. P. 423-438.