УДК 004.021

М. Х. А. Аль-Марди, Ю. А. Шичкина
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

Метод оптимизации параллельного алгоритма за счет уменьшения объема межпроцессорной передачи информации

Предлагается метод построения эффективного алгоритма по числу использованных процессоров, времени выполнения алгоритма и объему межпроцессорных передач. Данный метод может быть применен как к последовательным алгоритмам для получения их параллельного аналога, так и к параллельным алгоритмам с целью повышения их качества. Предлагаемый метод оптимизации параллельного алгоритма позволяет уменьшить объем коммуникаций между процессорами и соответственно сократить время выполнения всего алгоритма. Применение метода оптимизации алгоритма по объему межпроцессорных передач позволяет достичь более высокого уровня производительности, эффективности и высокоскоростной обработки параллельных программ.

Алгоритм, параллельное выполнение, список следования, время выполнения, операция, процесс, процессор, информационная зависимость, эквивалентные преобразования, информационный граф

Благодаря широкому распространению высокопроизводительных кластерных вычислительных систем стало актуальным использование процессоров с распределенной памятью. Вычисления в распределенной памяти отличны от вычислений в общей памяти тем, что в распределенной памяти для взаимодействия процессоров используется интерфейс передачи сообщений. Системы с распределенной памятью являются архитектурно более сложными устройствами, чем системы с общей памятью.

Для подобных систем, прежде чем создавать параллельную программу, необходимо знать общую архитектуру параллельной машины и топологию межпроцессорных связей, которая существенна для программирования. Это связано с отсутствием автоматического распараллеливания, которое позволяло бы превращать любую последовательную программу в параллельную и обеспечивало бы ее высокую производительность [1]. Нужно в явном виде увязывать структуру алгоритма решаемой задачи со структурой вычислительной системы и обеспечивать правильность взаимодействия множества параллельно независимых друг от друга процессов [2].

В последнее время параллельным вычислениям было посвящено много работ. В целом все эти работы можно было бы условно разделить на ряд основных категорий: работы в области исследования параллельных алгоритмов, их структуры и качества [3], [4]; развитие общей теории параллельного программирования [5]–[7]; решение частных прикладных задач; работы, посвященные построению параллельных алгоритмов для задач узкого класса из некоторой области с примерами параллельных алгоритмов вычислительной математики [8]; работы, в которых были предложены формальные модели, позволяющие описывать функционирование последовательных процессов, исполняющихся параллельно [9]–[11]; решение проблем планирования вычислительного процесса и др.

Повысить скорость вычислений, как известно, можно двумя путями. Первый способ – выбрать высокоскоростную модификацию архитектуры ЭВМ, но возможности данного способа ограничены в силу своих физических особенностей. Второй способ – программный. При этом способе разработчику параллельной программы необходимо выбрать модель архитектуры, допускающую параллельную реализацию алгоритма, и решить важный вопрос: как создать параллельную программу.

Сегодня разработчики параллельных программ разделились на 2 основных класса: тех, кто считает, что параллельную программу надо со-

здавать «с нуля», без опоры на последовательные аналоги, и тех, кто полагается на накопленные десятилетиями последовательные программы.

Оба подхода имеют свои достоинства и недостатки, однако едины в необходимости анализа структуры алгоритма на предмет эффективного использования вычислительных ресурсов и поиска возможностей ускорения процессов вычисления. Этот анализ может проводиться предварительно в случае создания параллельного алгоритма на основе последовательного; промежуточно для получения информации об успешности параллельного выполнения; заключительно – для сравнения алгоритмов и их реализаций между собой.

Моделированию последовательных и параллельных алгоритмов уделяется достаточно внимания в последние несколько десятилетий. В целом разработанные на сегодняшний день методы построения параллельных алгоритмов на многопроцессорных вычислительных системах [12]–[14] не позволяют построить достаточно эффективные и быстродействующие программы, так как их отличительной особенностью является адаптация на конкретные задачи с конкретной архитектурой вычислительной системы.

Среди методов, позволяющих получить формальное и наглядное представление о возможных параллельных ветвях в алгоритмах, можно выделить следующие: метод поиска взаимно независимых работ; метод определения ранних и поздних сроков выполнения операций алгоритма [14]; методы составления расписаний, основанные на списках следования [15].

Последний метод наименее трудоемок. Существующий алгоритм разбивается на операции, строится информационный граф [16] и определяются характеристики: высота и ширина алгоритма (время и число процессоров, задействованных в вычислениях). Как и любой другой, метод [15] обладает недостатками (необходимостью разбивать алгоритм на отдельные операции и строить граф информационных зависимостей между операциями) и достоинствами: в результате исследователь может однозначно определить объем вычислительных единиц, необходимых для распараллеливания, эффективность использования вычислительных единиц и, самое главное, возможность параллельной реализации исследуемого метода в виде конкретного алгоритма.

Хочется отметить, что в зависимости от результата анализа структуры алгоритма исследователь в любом случае получит для себя ценную информацию:

– если алгоритм идеально распараллеливается при заданных ограничениях на время и объем вычислительных ресурсов, то исследователь получает модель будущей параллельной программы вместе с расписанием ее выполнения в заданной вычислительной среде;

– если алгоритм возможно распараллелить, но в исходном виде он имеет узкие места, такие, как, например, простои вычислительных единиц или требование вычислительных ресурсов в большем объеме, чем они имеются в наличии, то на основе полученных метаданных исследователь может применить существующие алгоритмы оптимизации параллельных алгоритмов и получить желаемую модель будущей параллельной программы;

– если результаты анализа наихудшие, т. е. алгоритм практически не распараллеливается, то исследователь как минимум экономит средства на разработку неэффективной параллельной программы, а как максимум – получает информацию о существующих проблемах в алгоритме и направлениях их решения.

Одним из показателей качества параллельной программы является плотность загрузки вычислительных узлов. Временные задержки при передаче данных по каналам связи от одного процессора к другому приводят к суммарно длительным простоям процессоров и увеличению в целом времени работы алгоритма.

В данной статье предлагается метод построения эффективного алгоритма по числу использованных процессоров, времени выполнения алгоритма и объему межпроцессорных передач. Данный метод может быть применен как к последовательным алгоритмам для получения их параллельного аналога, так и к параллельным алгоритмам с целью повышения их качества.

Постановка задачи. Любой алгоритм (последовательный или параллельный) представляет собой сложную многосвязную систему с целым набором параметров, влияющих на качество работы этой системы. Оптимизировать работу алгоритма сразу по многим параметрам – нелегкая задача, которая может быть решена поэтапно.

В данной статье приведены результаты первого этапа решения задачи получения расписания выполнения алгоритма, соответствующего заданному

информационному графу. Алгоритм должен быть оптимальным по объему межпроцессорных передач при следующих ограничениях на оптимизируемый алгоритм и вычислительную систему:

– вычислительная система не ограничена по количеству процессоров (ядер);

– каждая операция обладает одинаковым объемом входных данных;

– все операции выполняются за одинаковое время, условно равное 1 о. е.;

– время передачи данных между двумя любыми процессорами постоянно и условно равно 1 о. е.

Очевидно, что на практике алгоритмов и вычислительных систем с такими характеристиками не существует, но эта модель параллельного алгоритма является начальной стартовой моделью для получения метода оптимизации параллельных алгоритмов по времени выполнения с учетом совокупности метаданных самого алгоритма и вычислительной среды.

Рассмотрим пример. Пусть задан информационный граф алгоритма (рис. 1).

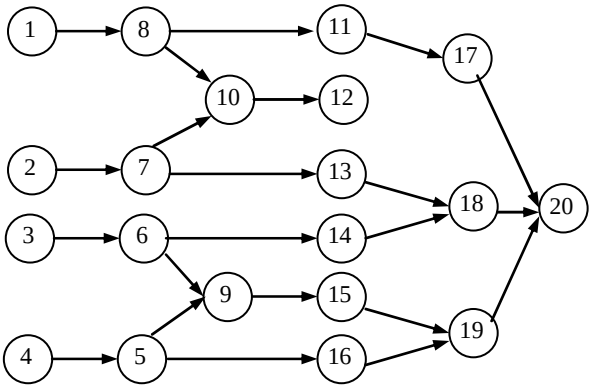

Рис. 1

Распределив вершины графа по ярусам с помощью метода оптимизации информационного графа по ширине на основе матрицы смежности или списков смежности [18], получим соответствующие ярусам начальные группы вершин графа (рис. 2).

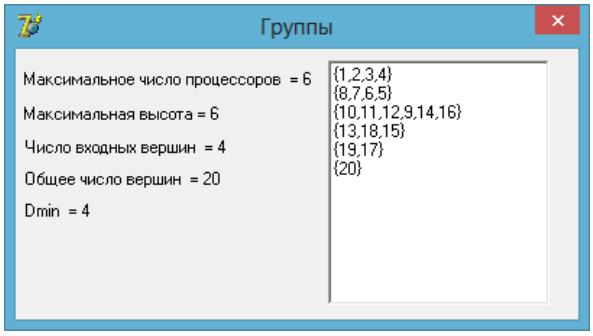

Рис. 2

Если по полученным группам построить расписание выполнения алгоритма на вычислительной системе с учетом межпроцессорных передач данных, то оно будет следующим (рис. 3):

P1: 1, 8, _, 10,13, _, 19, _, _ , 720;
P2: 2, 7, _, 11, _, 18, _, 17;
P3: 3, 6 , _ , 12, _, 15;
P4: 4, 5, _, 9;
P5: _, _, _, 14;
P6: _, _, _, 16,

где Pi – номер процессора $\left(i=\overline{1,6}\right)$; символ подчеркивания '_' – простой процессора («пузырь») в ожидании получения входных данных от других операций.

В соответствии с этим расписанием общее время работы алгоритма t = 10, число процессоров n = 6, суммарный объем простоев p = 16.

С целью определения возможности выровнять плотности вычислений процессоров найдем теоретическую минимальную ширину информационного графа: $D_{\min}$ = 4.

В данном случае ширина информационного графа после начального распределения вершин по ярусам соответствует максимальному числу вершин по группам и равна 6. Следовательно, существует возможность оптимизировать данный граф по ширине.

Применяя алгоритм оптимизации по ширине (по числу процессоров), получим группы, размер которых соответствует оптимальному параметру $D_{\min}$ = 4:

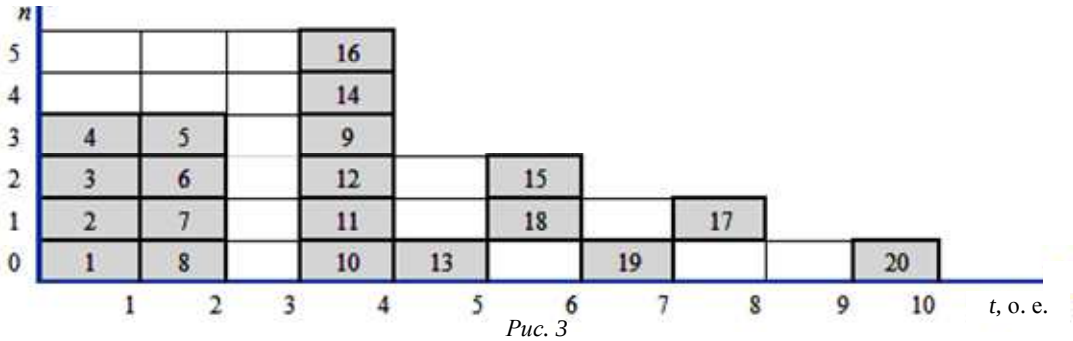

Рис. 3

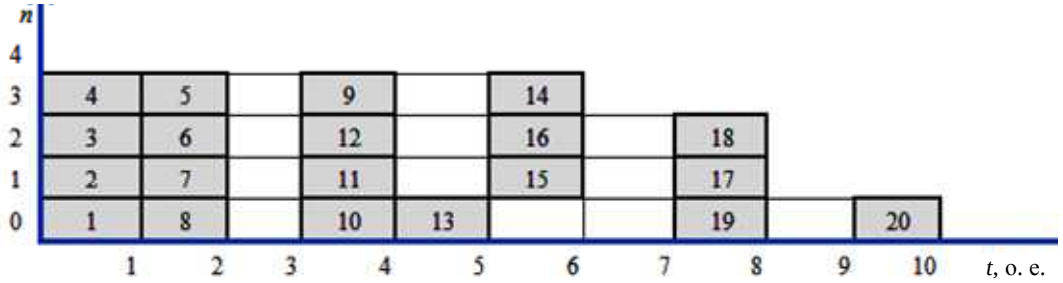

Рис. 4

M1 {1, 2, 3, 4}, M2{8, 7, 6, 5}, M3 {10, 11, 12, 9}, M4 {13, 15, 16, 14}, M5 {19, 17, 18}, M6 {20}

Следует отметить, что это не единственный вариант разбиения множества вершин по группам. Возможны и другие варианты. Все зависит от выбранного метода оптимизации по ширине.

Преобразуем в соответствии с полученными группами временную диаграмму алгоритма (рис. 4).

При этом расписание изменится следующим образом:

P1: 1, 8, _, 10, 13, _, _, 19, _, 20;
P2: 2, 7, _, 11, _, 15, _, 17;
P3: 3, 6, _, 12, _, 16, _, 18;
P4: 4, 5, _, 9, _, 14.

В соответствии с этим расписанием общее время работы алгоритма $t = 10$, число процессоров $n = 4$, суммарный объем простоев $p = 12$.

Полученное расписание лучше начального (рис. 3), так как позволяет для реализации алгоритма использовать вычислительную систему с меньшим числом процессоров, сохранив при этом общее время выполнения алгоритма и сократив простои процессоров.

Метод оптимизации алгоритма посредством уменьшения межпроцессорных коммуникационных связей. Из приведенных диаграмм видно, что время, затрачиваемое на передачу данных, увеличивает время работы процессоров и суммарное время работы алгоритма.

Приведенные примеры согласуются с известным фактом, что функция ускорения вычислений алгоритма на системе из n вычислительных устройств $K = F(n)$ имеет нормальное распределение (рис. 5).

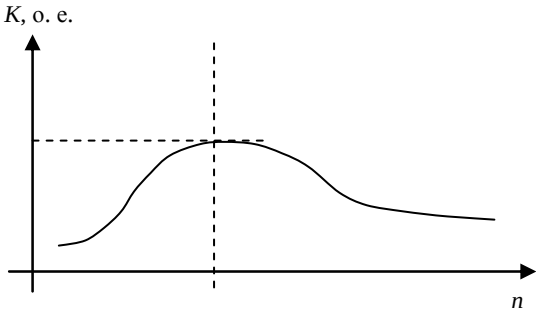

Рис. 5

Начиная с некоторого n ускорение вычислений падает за счет роста объема передачи данных. Для некоторых алгоритмов эта зависимость является линейной убывающей функцией, например, параллельный вариант алгоритма пузырьковой сортировки работает медленнее исходного последовательного метода, так как объем передаваемых данных между процессорами достаточно велик и сопоставим с количеством выполняемых вычислительных операций (и этот дисбаланс объема вычислений и сложности операций передачи дан-

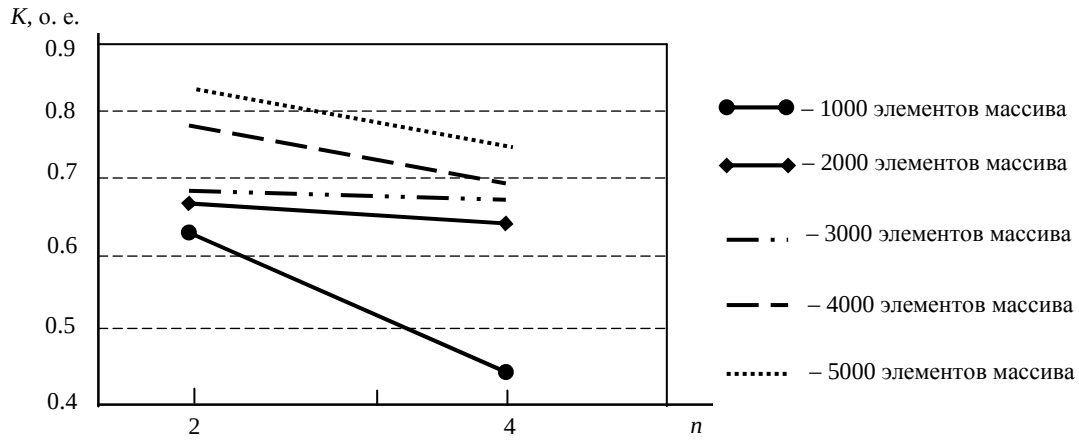

Рис. 6

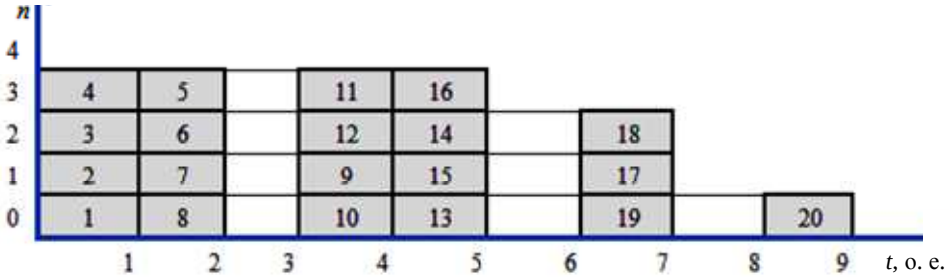

Рис. 7

ных увеличивается с ростом числа процессоров) [16]. На рис. 6 показано как падает ускорение K с ростом числа процессоров n для массивов различного объема (10 000, 20 000, 30 000, 400 00 и 50 000 элементов в массиве).

Следовательно, следующим шагом на пути получения оптимального по времени выполнения алгоритма является уменьшение объема межпроцессорных передач.

Предлагаемый авторами ***метод оптимизации информационного графа путем уменьшения межпроцессорных коммуникационных связей*** заключается в следующем:

1. Взять за основу группы вершин, соответствующих ярусам, полученные произвольным (лучше – формализованным) способом, например методом оптимизации информационного графа по ширине с помощью матрицы или списка смежности.

2. Процесс перестановки вершин начать с последней группы. Допустим, всего групп m, тогда номер очередной группы $k = m$.

В первую очередь необходимо в расписание поставить (с учетом групп) вершины с бинарной связью и только потом – вершины с множественными связями, так как в этом случае существование пузыря неизбежно.

3. В k-й группе выбрать первую вершину. Считать номер позиции этой вершины в группе равным 1: $i = 1$.

4. Сравнить вершину Mk_i (i – номер позиции вершины в группе) с вершинами предыдущей $(k-1)$-й группы. Если в $(k-1)$-й группе существует вершина $(M(k-1)_j$, где j – номер позиции вершины в группе, $j \geq i$), напрямую связанная в информационном графе ребром с данной вершиной, то вершину $M(k-1)_j$ необходимо переместить в своей группе в i-ю позицию. Если в $(k-1)$-й группе нет вершины, связанной с вершиной Mk_i, то переход на шаг 6.

5. Если $k > 2$, то $k = k - 1$ и переход на шаг 4.

6. Если в m-й группе перебраны еще не все вершины, то $k = m$, $i = i + 1$ и переход на шаг 4.

7. Если в последней группе перебраны все вершины и $m > 2$, то $m = m - 1$ и переход на шаг 6.

8. Если $m = 1$, то конец метода.

Пример. Для информационного графа с рис. 1 возьмем за основу группы, полученные после оптимизации графа по ширине:

M1 {1, 2, 3, 4}, M2 {8, 7, 6, 5}, M3 {10, 11, 12, 9}, M4 {13, 15, 16, 14}, M5 {19, 17, 18}, M6 {20}.

Используя метод оптимизации расписания по объему коммуникаций получим следующие группы для каждого яруса:

M1 {1, 2, 3, 4}, M2 {8, 7, 6, 5}, M3 {10, 9, 12, 11}, M4 {13, 15, 14, 16}, M5 {19, 17, 18}, M6 {20}

Временная диаграмма для полученных групп с учетом информационного графа (см. рис. 1) представлена на рис. 7.

При этом расписание изменится следующим образом:

P1: 1, 8, _, 10, 13, _, 19, _, 20;
P2: 2, 7, _, 9, 15, _, 17;
P3: 3, 6, _, 12, 14, _, 18;
P4: 4, 5, _, 11, 16.

Пересчитав объем коммуникаций, получим, что время, затрачиваемое на передачу данных с одного процессора на другой, уменьшилось в 2 раза и стало равным 8 о. е., против начальных 16 о. е.

Суммарное время работы алгоритма уменьшилось с 10 до 9 о. е.

Таким образом, предложенный метод позволяет уменьшить объем коммуникаций между процессорами и соответственно сократить время выполнения всего алгоритма.

Достоинства данного метода:

- Низкая временная трудоемкость $O(md^2)$, где m – число групп, d – ширина графа.
- Возможность работы не с разреженной матрицей смежности, а со списком смежности, что значительно ускоряет процесс расчета расписаний и экономит память.

• Сохранение информационных зависимостей.

• Сохранение начальной ширины информационного графа.

• Возможность комбинирования данного метода с другими методами оптимизации.

Метод оптимизации алгоритма по объему межпроцессорных передач данных эффективно применять в соответствии со следующей методикой:

1. Разбить алгоритм на операции.

2. Построить информационный граф алгоритма.

3. Построить параллельную форму информационного графа и временную диаграмму алгоритма.

4. Провести оптимизацию по ширине (по числу процессоров).

5. Провести оптимизацию по межпроцессорным коммуникациям.

6. Провести укрупнение операций.

Применение метода оптимизации алгоритма по объему межпроцессорных передач позволяет достичь более высокого уровня производительности, эффективности и высокоскоростной обработки параллельных программ.

Следует также отметить, что данный метод является отправной точкой при создании более совершенного метода, учитывающего не только объем входящих в заданную вершину ребер, но и непосредственно объем передаваемых данных, длину пути, время работы каждой операции.

СПИСОК ЛИТЕРАТУРЫ

1. Abramov O. V., Katueva Ya. Multivariant analysis and stochastic optimization using parallel processing techniques // Management problems. 2003. № 4. P. 11–15.

2. Akl S. The Design and Analysis of Parallel Algorithms. NJ, USA: Prentice-Hall, Inc. Upper Saddle River, 1989.

3. Arvindam S., Kumar V., Rao V. Floorplan optimization on multiprocessors // In Proc. Intl Conf. on Computer Design. IEEE Computer Soc. 1989. P. 109–113.

4. Aho A., Hopcroft J., Ullman J. The Design and Analysis of Computer Algorithms // Addison-Wesley Series in Computer Science and Information Proc. Boston: Addison-Wesley Publishing Company, 1974.

5. Andrews. G. R. Concurrent Programming: Principles and Practice. Redwood City, CA: Benjamin / Cummings Publishing Company, 1991.

6. Jordan H. F., Alaghband F. Fundamentals of Parallel Proc. Pearson Education. NJ: Inc. Upper Saddle River, 2003. P. 578.

7. Drake. D. E., Hougardy. S. A linear-time approximation algorithm for weighted matchings in graphs // ACM Transactions on Algorithms. 2005. № 1. P. 107–122.

8. Ahmad I., Kafil M. A Parallel Algorithm for Optimal Task Assignment in Distributed Systems // Proc. of the Advances in Parallel and Distributed Computing Conf., 1997. NJ, USA: Piscataway, 1997. P. 284–290.

9. Hu Chen. MPIPP: An Automatic Profileguided Parallel Process Placement Toolset for SMP Clusters and Multiclusters // Proc. of the 20th annual Intern. Conf. on Super-computing, New York, USA, 2006. P. 353–360.

10. Rauber N., Runger. G. Parallel Programming: for Multicore and Cluster Systems. Chemnitz, Germany: Springer, 2010. 450 p.

11. Boyer L. L., Pawley G. S. Molecular dynamics of clusters of particles interacting with pairwise forces using a massively parallel computer // J. of Computational Physics. 1988. Vol. 78. P. 405–409.

12. Brunet J. P., Edelman A., Mesirov J. P. Hypercube algorithms for direct N-body solver for different granularities, SIAM // J. of Scientific and Statistical Computing. 1993. Vol. 14. P 1143–1149.

13. Hu Y. F., Emerson D. R., Blake R. J. The Communication Performance of the Cray T3D and its Effect on Iterative Solvers // Parallel Computing. 1993. Vol. 22. P. 829–844.

14. Gergel V. P., Fursov V. A. Lectures of Parallel Programming: Proc. Benefit / Samara State Aerospace University Publishing House. Samara, 2009. 163 p.

15. Шичкина Ю. А. Сокращение высоты информационного графа параллельных программ // Науч.-техн. ведомости СПбГПУ. 2009. № 3 (80). С. 148–152.

16. Voevodin V. V., Voevodin Vl. V. Parallel computing. St. Petersburg: BHV-Petersburg, 2002. 608 p.

M. H. A. Al-Mardi, Yu. Shichkina
Saint Petersburg Electrotechnical University «LETI»

METHOD OPTIMIZATION OF THE PARALLEL ALGORITHM BY REDUCING THE AMOUNT OF INTERPROCESSOR COMMUNICATION OF INFORMATION

In this paper, we propose a method for constructing an efficient algorithm by the number of used processors, the execution time of the algorithm and the volume of interprocessor transfers. This method can be applied for sequential algorithms to get their parallel analogue and for parallel algorithm in order to improve their quality. The method of optimization of information graph by reducing interprocessor communication links can reduce the amount of communication between processors and therefore reduce the total execution time of the algorithm. Application of the optimization algorithm can achieve higher levels of performance, efficiency and high-speed processing of parallel programs.

Algorithm, parallel execution, sequence list, execution time, operation, process, processor, information dependence, equivalent conversions, information graph