ISSN 0130- 5395

Национальная академия наук Украины | Международный научно-учебный центр информационных технологий и систем | Институт кибернетики им. В.М. Глушкова | Фонд Глушкова

УСИМ

УПРАВЛЯЮЩИЕ СИСТЕМЫ И МАШИНЫ

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

Международный научный журнал

№ 4 июль – август **2006**

Основные темы выпуска:

Алгоритмы быстрого умножения больших чисел

●

Об оптимизации числа *LUT*-элементов

●

Оценка качества фонограмм и идентификация голоса

ISSN 0130-5395

9 770130 539008

УПРАВЛЯЮЩИЕ СИСТЕМЫ И МАШИНЫ

УСиМ

- Национальная академия наук Украины
- Международный научно-учебный центр информационных технологий и систем
- Институт кибернетики имени В.М. Глушкова
- Фонд Глушкова

Международный научный журнал
(статьи публикуются на русском, украинском, английском языках)

4 (204) **2006** июль – август

Основан в сентябре 1972 г.
Выходит раз в два месяца

Содержание

УДК 681.3.06:519

Н.С. Коваленко, П.А. Павлов

Приемы ускорения вычислений при распределенной обработке

Введено пять групп приемов ускорения вычислений при распределенной обработке; проведена их классификация с учетом числа операций программной реализации. Сформулировано утверждение эффективности различных приемов ускорения вычислений.

The indicators of estimation of the efficiency of parallel algorithms for the system of distributed data calculation are considered. The classification of the techniques to make the calculations faster is carried out and the problems of its efficiency are discussed.

Введено п'ять груп заходів прискорення обчислень за умов розподіленої обробки; виконано їх класифікацію з урахуванням числа операцій програмної реалізації. Сформульовано твердження ефективності різних заходів прискорення обчислень.

Введение. Распределенные вычисления – перспективная и динамично развивающаяся область организации *параллелизма* [1, 2]. Этот термин обычно используется при параллельной обработке данных нескольких функциональных устройств, достаточно удаленных одно от другого, в которых передача данных по линиям связи приводит к существенным временным задержкам. Эффективная обработка данных при таком способе организации вычислений возможна только для алгоритмов с низкой интенсивностью потоков межпроцессорных передач данных. Перечисленные условия характерны, например, при организации вычислений в многомашинных вычислительных комплексах, высокопроизводительных вычислительных кластерах, локальных или глобальных информационных сетях. В связи с этим производительность при распределенной обработке параллельных процессов может существенно колебаться в зависимости от целого ряда факторов в достаточно широком диапазоне, а ускорение вычислений может быть достигнуто более богатым набором приемов.

1. Основные понятия и определения

Аппарат теории графов, будучи мощным математическим средством, получил широкое распространение при разработке программ, их тестировании, при анализе и оценке сложности программ, распараллеливании последовательных программ [3, 4]. Он позволяет наглядно представить совокупность операций алгоритма, связь между отдельными операциями и порядок их выполнения.

Представим множество операций, выполняемых в исследуемом алгоритме решения задачи, и существующие между операциями информационные зависимости в виде ациклического ориентированного мультиграфа $G=(V,R)$, называемого *графом алгоритма* (ГА) [1], где $V=\{1, 2, ..., |V|\}$ есть множество вершин графа, представляющее выполняемые операции алгоритма, а R – множество дуг графа, причем дуга $r=(i,j)$ принадлежит графу G, если операция j использует результат выполнения операции i. Предположим, что множество вершин V разбито на такие непересекающиеся подмножества $V_1, V_2, ..., V_k$, что если $u \in V_i$, $v \in V_j$ и существует дуга (u, v), то $i < j$ В этом случае разбиение $V=\bigcup_{i=1}^{k} V_i$ называется *строгой параллельной формой алгоритма*, а подмножества $V_1, V_2, ..., V_k$ – ее *ярусами* или *уровнями*. Существует строгая параллельная форма, при которой максимальная из длин путей, оканчивающихся в вершине с индексом i, равна $i-1$. Для этой параллельной формы число используемых индексов на единицу больше критического пути графа. Строгая параллельная форма называется *канонической*, если все входные вершины находятся в группе с одним индексом, равным единице. Для заданного графа алгоритма его каноническая форма *единственна*. Число k называют *высотой* параллельной формы, число вершин в ярусе $|V_i|$, $i=\overline{1,k}$ – *шириной* i–го яруса, а максимальная ширина ярусов $\max|V_i|$, $i=\overline{1,k}$ – *шириной* параллельной формы [1].

Необходимо отметить, что множество V может быть разбито на два непересекающихся подмножества V_C и V_B скалярных и векторных операций соответственно. Тогда *трудоемкостью* или *числом операций* алгоритма будет называться число

$$N = |V_C| + \sum_{i \in V_B} L_i ,$$

где $|V_C|$ – число элементов множества V_C, а L_i – длина вектора, используемого в i–й векторной операции [5]. Заметим, что в общем случае в оценку числа операций алгоритма необходимо включать также и число условных и безусловных переходов.

Пусть задан произвольный ГА и две его программные реализации P_1 и P_p соответственно последовательного и распределенного на p процессорах алгоритмов решения задачи, для которых определены величины N_i, T_i – число операций и время выполнения соответствующих программных реализаций, $i = 1, p$.

Известно [5], что *производительность* λ_i программной реализации P_i вычисляется по формуле:

$$\lambda_i = \frac{N_i}{T_i}, \; i = 1, p .$$

Ускорением вычислений будем называть отношение времени выполнения программной реализации P_1 к времени выполнения программной реализации P_p и обозначать через $\alpha = T_1/T_p$. Все преобразования графа алгоритма или соответствующей ему программной реализации, приводящие к ускорению вычислений без нарушения их функциональных возможностей при получении конечного результата, будем называть *приемами ускорения* вычислений. Приемы, для которых $\alpha > 1$, будем называть *эффективными*.

Эффективность использования p процессоров при решении задачи распределенным алгоритмом [2] определяет среднюю долю времени выполнения алгоритма, в течение которой процессоры реально используются для решения задачи и определяется соотношением $\beta = \alpha/p$. Как следует из приведенных соотношений, в наилучшем случае $\alpha = p$ и $\beta = 1$.

Величину $\eta = N_p/N_1$ будем называть *относительной трудоемкостью*, а величину $\mu = \lambda_p/\lambda_1 = \alpha\eta$ – *относительной производительностью* программной реализации P_p по отношению к P_1.

Если $\eta < 1$, то величину $\nu_\eta = 1/\eta$ будем называть *выигрышем от уменьшения трудоемкости*, а при $\eta > 1$ величину $p_\eta = \eta$ будем называть *проигрышем в трудоемкости*. Аналогично, если $\mu > 1$, то величину $\nu_\mu = \mu$ будем называть *выигрышем в производительности*, а при $\mu < 1$ величину $p_\mu = 1/\mu$ будем называть *проигрышем в производительности* программной реализации P_p по отношению к P_1.

С учетом введенных обозначений, ускорение вычислений α можно представить в следующем виде:

$$\alpha = \frac{\mu}{\eta}. \qquad (1)$$

2. Приемы ускорения вычислений и их классификация

Для ЭВМ последовательного типа одним из наиболее эффективных приемов ускорения вычислений является снижение числа операций в программных реализациях и самих алгоритмах. При распределенной обработке параллельных процессов производительность многопроцессорной системы может существенно колебаться в зависимости от целого ряда факторов в достаточно широком диапазоне, следовательно, ускорение вычислений можно достичь более богатым набором приемов, по сравнению с ЭВМ последовательного типа.

Утверждение 1. Согласно формуле (1), приемами ускорения вычислений при распределенной обработке являются:

1. $\mu < 1$, $\eta < 1$ и $\eta < \mu$, т.е. уменьшается относительная производительность системы распределенной обработки и число операций в программной реализации, причем число операций должно сократиться в большее число раз, чем уменьшится производительность;

2. $\mu = 1$, $\eta < 1$, т.е. относительная производительность системы распределенной обработки не изменяется, а сокращается число операций в программной реализации;

3. $\mu > 1$, $\eta < 1$, т.е. относительная производительность распределенной обработки повышается, и сокращается число операций в программной реализации;

4. $\mu > 1$, $\eta = 1$, т.е. повышается относительная производительность системы распределенной обработки, а число операций не изменяется;

5. $\mu > 1$, $\eta > 1$ и $\eta < \mu$, т.е. повышается относительная производительность системы распределенной обработки и увеличивается число операций в программной реализации, причем относительная производительность должна повыситься быстрее, чем увеличиться число операций.

Все пять перечисленных групп приемов ускорения вычислений при распределенной обработке схематически изображены на рис. 1, где первой группе соответствует нижняя треугольная область, второй – вертикальная полоса, третьей – прямоугольная область, четвертой – горизонтальная полоса, а пятой – верхняя треугольная область.

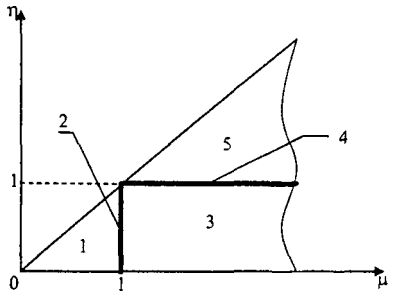

Рис. 1

Из утверждения 1 следует, что для систем распределенной обработки приемы ускорения вычислений применительно к числу операций программной реализации или алгоритма, могут быть отнесены к одному из трех следующих классов.

1. *Приемы ускорения вычислений, основанные на уменьшении числа операций программной реализации или самого алгоритма.* Производительность систем распределенной обработки при этом может уменьшаться, оставаться постоянной или увеличиваться. К данному классу относятся приемы групп 1–3.

2. *Приемы ускорения вычислений, для которых число операций программной реализации и алгоритма не изменяется.* При этом повышается производительность системы распределенной обработки. К данному классу относятся приемы группы 4.

3. *Приемы ускорения вычислений, основанные на избыточности операций или данных.* При этом повышается производительность системы распределенной обработки наряду с увеличением числа операций или объема обрабатываемой информации. К данному классу относятся приемы группы 5.

3. Эффективность приемов ускорения вычислений

Вполне естественным является вопрос об эффективности той или иной группы приемов ускорения вычислений. Но однозначного вывода сделать нельзя, поскольку следует учитывать особенности конкретного алгоритма и решаемой задачи. Несмотря на это, можно оценить потенциальные возможности рассмотренных групп приемов ускорения вычислений, характеризующих следующее утверждение.

Утверждение 2. Группа приемов ускорения вычислений при распределенной обработке будет эффективной, т.е. $\alpha > 1$, если α удовлетворяет условиям:

1) $\alpha = \nu_\eta / p_\mu$, т.е. равно отношению выигрыша от уменьшения трудоемкости программной реализации к проигрышу производительности системы распределенной обработки;

2) $\alpha = \nu_\eta$, т.е. равно выигрышу от уменьшения трудоемкости программной реализации;

3) $\alpha = \nu_\eta \nu_\mu$, т.е. равно произведению выигрыша от уменьшения трудоемкости программной реализации и выигрыша производительности распределенной системы;

4) $\alpha = \nu_\mu$, т.е. равно выигрышу производительности распределенной системы;

5) $\alpha = \nu_\mu / p_\eta$, т.е. равно отношению выигры-

ша производительности распределенной системы к проигрышу в трудоемкости программной реализации.

Как следует из утверждения 2, наибольшими потенциальными возможностями ускорения вычислений обладают приемы третьей группы. Однако очевидно, что приемы ускорения вычислений должны применяться на различных уровнях оптимизации, а наибольшее ускорение вычислений может быть достигнуто лишь на основе анализа особенностей конкретного алгоритма, объема входной информации и выбора соответствующего приема ускорения вычислений или их совокупности.

Уровни оптимизации схематически изображены на рис. 2. Поэтому приемы ускорения вычислений различных классов могут быть отнесены к одному из этих уровней или к их совокупности [5].

Пример. Рассмотрим организацию распределенных вычислений на примере задачи умножения матрицы на вектор [6], которая определяется соотношением $y_i = \sum_{j=1}^{n} a_{ij} x_j$, $i = \overline{1,n}$. Получение результирующего вектора y предполагает повторение n однотипных операций по умножению строк матрицы A и вектора x. Общее количество необходимых скалярных операций оценивается величиной $N_1 = 2n^2$.

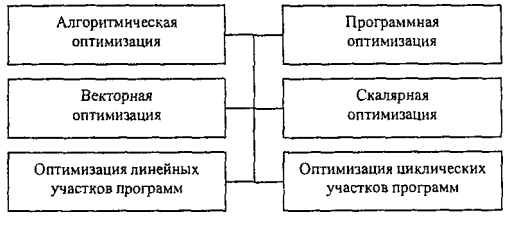

Рис. 2

1. *Достижение максимального быстродействия $p = n^2$*

Выполним анализ информационных зависимостей в алгоритме умножения матрицы на вектор:

- операции умножения отдельных строк матрицы на вектор – независимы и могут быть выполнены параллельно;
- умножение каждой строки на вектор включает в себя независимые операции поэлементного умножения и тоже может быть выполнено параллельно;
- суммирование получаемых произведений в каждой операции умножения строки матрицы на вектор может быть выполнено по одному из вариантов суммирования (последовательный алгоритм, обычная и модифицированная каскадная схемы) [6].

Таким образом, максимально необходимое количество процессоров определяется величиной $p = n^2$. Использование такого количества процессоров может быть представлено следующим образом. Множество процессоров P разбивается на n групп $P = (P_1, P_2, \ldots, P_n)$, каждая из которых представляет собой набор процессоров для выполнения умножения отдельной строки матрицы на вектор. В начале вычислений на каждый процессор группы пересылаются элемент строки матрицы A и соответствующий элемент x. Далее каждый процессор выполняет операцию умножения. Последующие вычисления выполняются по каскадной схеме суммирования. На рис. 3 приведена вычислительная схема для процессоров группы P_i при размерности матрицы $n = 4$.

Время выполнения параллельного алгоритма при использовании $p = n^2$ процессоров определяется временем выполнения параллельной операции умножения и временем выполнения каскадной схемы $T_p = 1 + \log_2 n$.

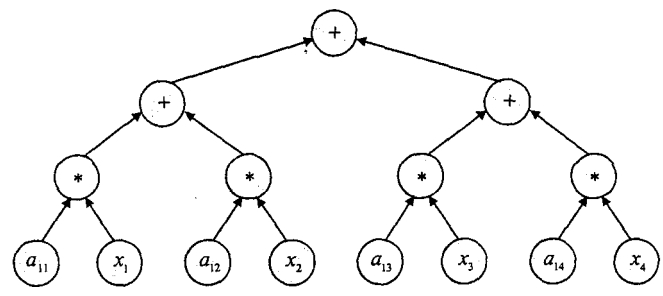

Рис. 3

Ускорение вычислений и показатель эффективности использования $p = n^2$ процессоров определяются следующими соотношениями:

$$\alpha = 2n^2 / (1 + \log_2 n),$$

$$\beta = 2n^2 / p(1+\log_2 n) = 2/(1+\log_2 n).$$

2. *Параллелизм среднего уровня $n < p < n^2$*

При уменьшении количества используемых процессоров $p < n^2$, обычная каскадная схема суммирования при выполнении операций умножения строк матрицы на вектор становится неприемлемой. Для простоты положим $p = nk$ и воспользуемся модифицированной каскадной схемой [6]. Начальная нагрузка каждого процессора в этом случае увеличивается, и процессор загружается n/k частями строк матрицы A и вектора x. Время выполнения операции умножения матрицы на вектор может быть оценено величиной

$$T_p = 2(n/k) + \log_2(k) = 2(n/(p/n)) +$$
$$+ \log_2(p/n) = 2(n^2/p) + \log_2(p/n).$$

При использовании количества процессоров, необходимого для реализации модифицированной каскадной схемы, т.е. при $p = 2n(n/\log_2 n)$, данное выражение при $n \geq 4$ дает оценку времени исполнения $T_p \leq 2\log_2 n$.

При количестве процессоров $p = 2n$, когда время выполнения алгоритма оценивается величиной $T_p = n + 1$, может быть предложена новая каскадная схема параллельного выполнения вычислений, при которой для каждой итерации каскадного суммирования используются неперекрывающиеся наборы процессоров. При таком подходе имеющегося количества процессоров достаточно для реализации только одной операции умножения строки матрицы A и вектора x. Кроме того, при выполнении очередной итерации каскадного суммирования процессоры, ответственные за исполнение всех предшествующих итераций, свободны. Однако этот недостаток предлагаемого подхода можно обратить в достоинство, используя простаивающие процессоры для обработки следующих строк матрицы A. В результате может быть сформирована следующая схема конвейерного выполнения умножения матрицы и вектора:

- множество процессоров P разбивается на непересекающиеся процессорные группы $P = (P_0, \ldots, P_k)$, $k = \log_2 n$, при этом группа P_i, $1 \leq i \leq k$, состоит из $n/2^i$ процессоров и используется для выполнения i–й итерации каскадного алгоритма; группа P_0 применяется для реализации поэлементного умножения; общее количество процессоров $p = 2n - 1$;
- инициализация вычислений состоит в поэлементной загрузке процессоров группы P_0 значениями первой строки матрицы и вектора x; после начальной загрузки выполняется параллельная операция поэлементного умножения и последующей реализации обычной каскадной схемы суммирования;
- при выполнении вычислений каждый раз после завершения операции поэлементного умножения осуществляется загрузка процессоров группы P_0 элементами очередной строки матрицы и инициируется процесс вычислений для вновь загруженных данных.

В результате применения описанного алгоритма множество процессоров P реализует конвейер для выполнения операции умножения строки матрицы на вектор. На подобном конвейере одновременно могут находиться несколько отдельных строк матрицы на разных стадиях обработки. Так, например, после поэлементного умножения элементов первой строки и вектора x процессоры группы P_1 будут выполнять первую итерацию каскадного алгоритма для первой строки матрицы, а процессоры группы P_0 будут исполнять поэлементное умножение значений второй строки матрицы и т.д. На рис. 4 приведена ситуация процесса вычисления после второй итерации конвейера при $n \geq 4$.

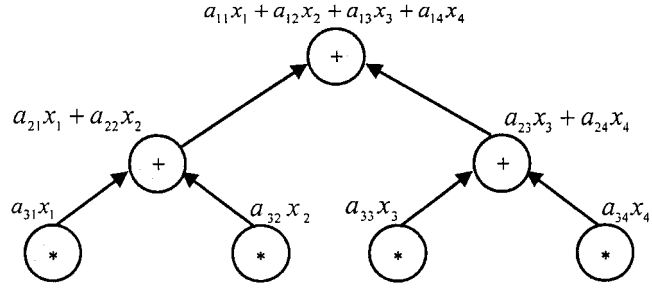

Рис. 4

Умножение первой строки на вектор в соответствии с каскадной схемой будет завершено, как и обычно, после выполнения $\log_2 n+1$ параллельных операций. Для других строк, в соответствии с конвейерной схемой организации вычислений умножение будет происходить после завершения каждой последующей итерации конвейера. Общее время выполнения операции умножения матрицы на вектор может быть выражено величиной

$$T_p = \log_2 n+1+n-1 = \log_2 n+n .$$

Данная оценка несколько больше, чем время выполнения параллельного алгоритма, описанного в предыдущем пункте $T_p = n+1$, однако вновь предлагаемый способ требует меньшего количества передаваемых данных, так как вектор x пересылается только однажды. Кроме того, использование конвейерной схемы приводит к более раннему появлению части результатов вычислений, что может быть полезным в ряде ситуаций обработки данных.

Показатели эффективности алгоритма определяются соотношениями следующего вида:

$$p = 2n,\ \alpha = 2n^2/(n+\log_2 n),$$

$$\beta = 2n^2/p(n+\log_2 n) = n/(n+\log_2 n).$$

3. *Организация параллельных вычислений при p = n*

При использовании n процессоров для умножения матрицы A на вектор x может быть использован ранее рассмотренный параллельный алгоритм построчного умножения, при котором строки матрицы распределяются по процессорам построчно и каждый процессор реализует операцию умножения какой-либо отдельной строки матрицы A на вектор x. Другой возможный способ организации параллельных вычислений может состоять в построении конвейерной схемы для операции умножения строки матрицы на вектор (скалярного произведения векторов) путем расположения всех имеющихся процессоров в виде линейной последовательности.

Подобная схема вычислений может быть определена следующим образом. Представим множество процессоров в виде линейной последовательности $P = \{p_1, p_2, \ldots, p_n\}$, где каждый процессор p_j, $1 \le j \le n$, используется для умножения элементов j–го столбца матрицы и j–го элемента вектора x (рис. 5). Выполнение вычислений на каждом процессоре p_j, $1 \le j \le n$ состоит в следующем:

- запрашивается очередной элемент j–го столбца матрицы;
- выполняется умножение элементов a_{ij} и x_j;
- запрашивается результат вычислений S предшествующего процессора;
- выполняется сложение значений $S \leftarrow S+a_{ij}x_j$;
- полученный результат S пересылается следующему процессору.

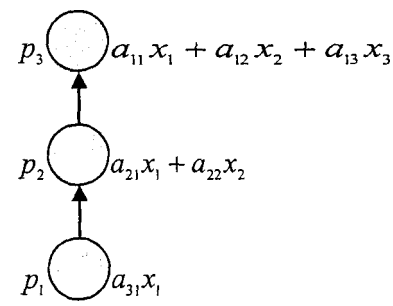

Рис. 5

При инициализации описанной схемы необходимо выполнить ряд дополнительных действий:

- при выполнении первой итерации каждый процессор дополнительно запрашивает элемент вектора x_j;
- для синхронизации вычислений на этапе инициализации процессор p_j, $1 \le j \le n$ выполняет $(j-1)$-й цикл ожидания.

Кроме того, для однородности описанной схемы первого процессора p_1, у которого нет предшествующего процессора, целесообразно ввести пустую операцию сложения $S = 0$, $S \leftarrow S + a_{ij}x_j$.

На рис. 5 показано состояние процесса вычислений после второй итерации конвейера при $n = 3$.

Умножение первой строки на вектор в соответствии с описанной конвейерной схемой будет завершено после выполнения $n+1$ па-

раллельных операций. Умножение следующих строк будет происходить после завершения каждой очередной итерации конвейера. Как результат, общее время выполнения операции умножения матрицы на вектор может быть выражено соотношением

$$T_p = n + 1 + 2(n-1) = 3n - 1.$$

Данная оценка также является большей, чем минимально возможное время $T_p = 2n$ выполнения параллельного алгоритма при $p = n$. Целесообразность использования конвейерной вычислительной схемы состоит в уменьшении количества передаваемых данных и в более раннем получении части результатов вычислений. Показатели эффективности данной вычислительной схемы при $p = n$ определяются соотношениями:

$$\alpha = 2n^2/(3n-1),$$
$$\beta = 2n^2/p(3n-1) = 2n/(3n-1).$$

4. *Ограниченное число процессоров $p \le n$*

При уменьшении количества процессоров до величины $p \le n$ параллельная вычислительная схема умножения матрицы на вектор может быть получена в результате адаптации алгоритма построчного умножения. В этом случае каскадная схема суммирования результатов поэлементного умножения вырождается, и операция умножения строки матрицы на вектор полностью выполняется на единственном процессоре. Получаемая при таком подходе вычислительная схема может быть конкретизирована следующим образом:

- на каждый из процессоров пересылается вектор x и $k = n/p$ строк матрицы;
- выполнение операции умножения строк матрицы на вектор выполняется при помощи обычного последовательного алгоритма.

Следует отметить, что размер матрицы может оказаться некратным количеству процессоров, и тогда строки матрицы не могут быть разделены поровну между процессорами. В этих ситуациях можно отступить от требования равномерности загрузки процессоров и для получения более простой вычислительной схемы принять правило, что размещение данных на процессорах осуществляется только построчно, т.е. элементы одной строки матрицы не могут быть разделены между несколькими процессорами. Неодинаковое количество строк приводит к разной вычислительной нагрузке процессоров, а значит общая длительность задачи будет определяться временем работы наиболее загруженного процессора, при этом часть от общего времени отдельные процессоры могут простаивать вследствие исчерпания своей доли вычислений. Неравномерность загрузки процессоров снижает эффективность использования многопроцессорных вычислительных систем и, как результат рассмотрения данного примера можно заключить, что проблема балансировки относится к числу важнейших задач параллельного программирования.

1. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. – СПб.: БХВ-Петербург, 2002. – 608 с.
2. *Эндрюс Г.Р.* Основы многопоточного, параллельного и распределенного программирования. – М.: Изд. дом «Вильямс», 2003. – 512 с.
3. *Основи* дискретної математики / Ю.В. Капітонова, С.Л. Кривий, О.А. Летичевський та ін. – К.: Наук. думка, 2002. – 580 с.
4. *Евстегнеев В.Н., Евстегнеев В.А.* Графы в программировании: обработка, визуализация и применение. – СПб.: БХВ-Петербург, 2003. – 1104 с.
5. *Приемы* ускорения вычислений при векторно–конвейерной обработке / А.М. Вабищевич, Н.С. Коваленко. – Минск, 1991. – 20 с. – (Препр. / АН БССР. Ин–т математики; № 10(460)).
6. *Гергель В.П., Стронгин Р.Г.* Основы параллельных вычислений для многопроцессорных вычислительных систем. – Н. Новгород: ННГУ, 2001. – 122 с.

Поступила 22.08.2005
Тел. для справок: (0172) 296-0781 (Минск)
E-mail: pin2535@tut.by