МАТЕМАТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ

DOI 10.17223/20710410/20/6

УДК 004.032.26(06)

ОТОБРАЖЕНИЕ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ НА МНОГОЯДЕРНЫЕ КОМПЬЮТЕРЫ РЕКУРРЕНТНЫМИ НЕЙРОННЫМИ СЕТЯМИ

М. С. Тарков

Институт физики полупроводников им. А. В. Ржанова СО РАН, г. Новосибирск, Россия

E-mail: tarkov@isp.nsc.ru

Рассмотрена задача отображения графа параллельной программы со взвешенными вершинами (процессами) и рёбрами (межпроцессными обменами) на произвольный взвешенный граф распределённой вычислительной системы. Предложен алгоритм решения этой задачи, основанный на использовании сетей Хопфилда. Алгоритм протестирован на отображении ряда графов параллельных программ на многоядерный компьютер. Эксперименты показали, что алгоритм позволяет получить хорошо сбалансированные субоптимальные отображения.

Ключевые слова: *графы параллельных программ, многоядерные системы, выравнивание нагрузки процессоров, нейрон, сети Хопфилда.*

Введение

В связи с постоянным увеличением мощности компьютеров появилась возможность создавать на их основе высокопроизводительные мультикомпьютерные вычислительные системы (ВС) [1 – 3]. Такая система в общем случае представляет собой объединённое линиями связи множество вычислительных узлов, которые обладают высокой степенью обособленности, т. е. каждый узел имеет свой процессор, зачастую многоядерный, свою память, возможно, свой собственный жёсткий диск и свой способ взаимодействия с другими узлами (сетевая карта, модем и т. п.). Узлы могут иметь разную производительность и разные коммуникационные возможности. В общем случае структура вычислительной сети произвольна и зависит от нескольких факторов, таких, как, например, аппаратные возможности, цель, для которой создаётся вычислительная сеть, финансовые ограничения.

В отличие от систем с общей памятью, в мультикомпьютерных ВС более остро стоит вопрос об уменьшении объёма межмашинных взаимодействий, так как пропускная способность сети связи низка. Необходимо таким образом распределить процессы по доступным вычислительным узлам, чтобы минимизировать время выполнения параллельной программы. Для этого нужно загрузить эти процессоры с учетом их производительности (чем выше производительность, тем выше нагрузка) и минимизировать взаимодействия между процессами, находящимися на разных узлах, что позволяет уменьшить простои процессоров, снижающие эффективность использования ВС.

В общем случае эта задача сводится к вложению графа параллельной программы в граф вычислительной системы [2, 3]. Целью вложения является минимизация

времени выполнения программы. Ввиду сложности задачи (она NP-полна) широко используются разнообразные эвристики для поиска оптимального вложения. В настоящее время популярностью пользуются методы, основанные на аналогиях с физикой и биологией, например такие, как метод имитации отжига, генетические алгоритмы и нейронные сети [4]. К последним относятся сети Хопфилда [5, 6].

1. Задача отображения

Цель оптимального распределения процессов (ветвей) параллельной программы по процессорам ВС заключается в минимизации времени выполнения программы, что эквивалентно минимизации времени простоя каждого процессора системы, участвующего в решении задачи, и одновременно минимизации затрат на взаимодействия между процессами, размещёнными в разных процессорах.

Пусть:

$G_p(V_p, E_p)$ — граф параллельной программы;

V_p — множество ветвей программы, $|V_p| = n$;

E_p — множество логических (виртуальных) каналов, реализующих взаимодействия между ветвями;

$G_s(V_s, E_s)$ — граф вычислительной системы;

V_s — множество процессоров (ядер), $|V_s| = m \leqslant n$;

E_s — множество связей между элементарными машинами (ЭМ);

w_x — вес (вычислительная сложность) ветви $x \in V_p$;

ϑ_i — производительность процессора $i \in V_s$;

$\tau_{xi} = w_x/\vartheta_i$ — время выполнения ветви $x \in V_p$ на процессоре $i \in V_s$;

c_{xy} — вес ребра $(x, y) \in E_p$, равный числу единиц информации, которыми обмениваются ветви x и y;

d_{ij} — время передачи единицы информации между процессорами (ядрами) i и j.

Пусть $f_m : G_p \to G_s$ — вложение графа программы G_p в граф вычислительной системы G_s. Качество вложения будем оценивать целевой функцией

$$H_g(f_m) = H_{gc}(f_m) + H_{gt}(f_m),$$

где $H_{gc}(f_m)$ — оценка небаланса вычислительной нагрузки; $H_{gt}(f_m)$ — оценка полного времени межпроцессорных взаимодействий.

Для вложения f_m полное время вычислений i-й ЭМ равно

$$t_i = \sum_{f_m(x)=i} \tau_{xi}.$$

Отсюда

$$H_{gc}(f_m) = \sum_{i=1}^{m} (t_i - t_{\min})^2,$$

где $t_{\min} = \sum\limits_x w_x / \sum\limits_i \vartheta_i$ — минимально возможное (идеальное) время выполнения параллельной программы (время выполнения программы на одном процессоре с производительностью, равной суммарной производительности всех ЭМ системы, при нулевых затратах на взаимодействия между ветвями программы).

Затраты на взаимодействия оцениваются функцией

$$H_{gt}(f_m) = \sum_{\substack{x \neq y, \\ i=f_m(x), j=f_m(y)}} c_{xy} d_{ij},$$

где суммирование производится по всем парам (x, y) взаимодействующих ветвей параллельной программы.

2. Сеть Хопфилда для задачи отображения

Рассмотрим матрицу нейронов v размером $n \times m$, каждая строка которой соответствует ветви параллельной программы, каждый столбец — процессору (ядру). Каждая строка матрицы v должна содержать один и только один ненулевой элемент, равный единице, остальные элементы должны быть равны нулю (ветвь параллельной программы не может быть отображена одновременно на несколько процессоров). Каждый столбец матрицы v может содержать произвольное (в том числе и нулевое) число элементов, равных единице, но суммарное количество единичных элементов должно быть равно числу ветвей n параллельной программы.

Энергия соответствующей нейронной сети Хопфилда описывается функцией Ляпунова

$$H = \frac{A}{2} H_c + \frac{B}{2} H_g. \tag{1}$$

Здесь A и B — параметры функции Ляпунова. Минимум H_c обеспечивает выполнение вышеуказанных ограничений на элементы матрицы v; H_g — целевая функция;

$$H_c = H_{c_1} + H_{c_2}, \quad H_{c_1} = \left(\sum_x \sum_i v_{xi} - n \right)^2, \quad H_{c_2} = \sum_x \left(\sum_i v_{xi} - 1 \right)^2; \tag{2}$$

минимум H_{c_1} обеспечивает наличие в матрице v ровно n единиц; минимум H_{c_2} обеспечивает наличие в каждой строке матрицы v ровно одной единицы;

$$H_g = H_{gc} + H_{gt}, \quad H_{gc} = \sum_i \left(\sum_x v_{xi} \tau_{xi} - t_{\min} \right)^2, \quad H_{gt} = \sum_x \sum_i v_{xi} \sum_y \sum_j v_{yj} c_{xy} d_{ij}. \tag{3}$$

Здесь v_{xi} — состояние нейрона строки x и столбца i матрицы v.

Динамика сети Хопфилда, минимизирующей функцию (1), описывается системой уравнений

$$\frac{\partial u_{xi}}{\partial t} = -\frac{\partial E}{\partial v_{xi}}, \tag{4}$$

где u_{xi} — активация нейрона с индексами $x = 1, \ldots, n$, $i = 1, \ldots, m$; $v_{xi} = (1 + {} + \exp(-\beta u_{xi}))^{-1}$ — состояние (выходной сигнал) нейрона, β — параметр функции активации. Из (1)–(4) получаем

$$\begin{aligned} \frac{\partial u_{xi}}{\partial t} = &-A \left(\sum_y \sum_j v_{yj} + \sum_j v_{xj} + \sum_y v_{yi} - n - 1 \right) - \\ &-B \left(\left(\sum_y v_{yi} \tau_{yi} - t_{\min} \right) \tau_{xi} + \sum_y \sum_j v_{yj} c_{xy} d_{ij} \right). \end{aligned} \tag{5}$$

Соответствующее (5) разностное уравнение имеет вид (t — номер текущей итерации, Δt — величина временного шага)

$$\begin{aligned} u_{xi}^{t+1} = &u_{xi}^t - \Delta t \left(A \left(\sum_y \sum_j v_{yj} + \sum\nolimits_j v_{xj} + \sum_y v_{yi} - n - 1 \right) + \right. \\ &\left. + B \left(\left(\sum_y v_{yi} \tau_{yi} - t_{\min} \right) \tau_{xi} + \sum_y \sum_j v_{yj} c_{xy} d_{ij} \right) \right). \end{aligned} \tag{6}$$

С целью ускорения сходимости сеть Хопфилда (6) преобразуется в сеть Вана [7] умножением целевой функции оптимизационной задачи на $\exp(-t/\tau)$, где τ — параметр:

$$u_{xi}^{t+1} = u_{xi}^{t} - \Delta t \left(A \left(\sum_y \sum_j v_{yj} + \sum_j v_{xj} + \sum_y v_{yi} - n - 1 \right) + \right.$$
$$\left. + B \left(\left(\sum_y v_{yi}\tau_{yi} - t_{\min})\tau_{xi} + \sum_y \sum_j v_{yj} c_{xy} d_{ij} \right) \exp(-t/\tau \right) \right). \tag{7}$$

Новое значение v_{xi}^{t+1} вычисляется сразу же после вычисления соответствующего значения u_{xi}^{t+1} (метод Гаусса — Зейделя).

3. Эксперименты

В экспериментах исследовано отображение на компьютер с $m = 2, 4, 8$ ядрами следующих вариантов параллельных программ с одинаковыми весами w_x, $x = 1, \ldots, n$, вершин (ветвей) и весами $c_{xy} = 0$ или $c_{xy} = 1$, $x, y = 1, \ldots, n$, $x \neq y$, рёбер графа программы:

1) множество независимых ветвей (отсутствуют обмены между ветвями, $c_{xy} = 0$);
2) типовые графы параллельных программ (линейка, кольцо и решётка) (рис. 1), $c_{xy} = 1$;
3) нерегулярные сетки, $c_{xy} = 1$.

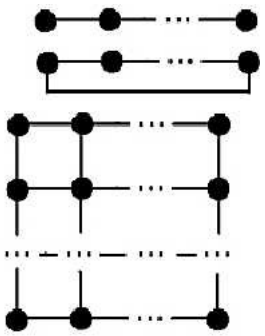


Рис. 1. Типовые графы параллельных программ (линейка, кольцо, решётка)

Изменяя параметры d_{ij} при фиксированной производительности ядер ϑ_i, мы изменяем связность подмножеств вершин, отображённых на одно и то же ядро, т. е. оптимальность затрат на межъядерные обмены информацией.

Параметры вычислительной системы: производительность ядер $\vartheta_i = 1$, $i = 1, \ldots, m$;

$$d_{ij} = \begin{cases} 0, & i = j, \\ 1, & i \neq j. \end{cases} \tag{8}$$

Согласно (8), затраты на обмены данными между ветвями программы внутри ядра считаются пренебрежимо малыми по отношению к межъядерным обменам. Иначе говоря, данные в ядре рассматриваются как массивы, обрабатываемые одной ветвью.

При $w_x = 1$ и $\vartheta_i = 1$ имеем $\tau_{xi} = 1$, $x = 1, \ldots, n$, $i = 1, \ldots, m$. Параметры нейронной сети: $A = 1000$; $B = 100$; $\Delta t = 1$; $\beta = 1$; $\tau = 100$.

Процедура отображения параллельной программы на вычислительную систему имеет следующий вид:

```
do
{initialize();
  do
  {iterate(); iter = iter+1;
   δ = nobalance();
  } while (δ > 0 && iter < maxiter);
} while (δ > maxnb || noncorrect());
```

Здесь:

— `initialize` задает начальные значения элементов матриц u^0 (и соответственно v^0), используя генератор псевдослучайных чисел;

— `iterate` выполняет шаг итерационного процесса (7), вычисляя новые значения элементов матриц u^{t+1} и v^{t+1};

— `nobalance` вычисляет небаланс нагрузки по формуле

$$\delta = \frac{\sqrt{\sum_{i=1}^{m}(t_i - t_{\min})^2}}{m \cdot t_{\min}};$$

— `noncorrect` проверяет выполнение условий (2) корректности решения v^{t+1}.

Итерационный процесс (7) продолжается до тех пор, пока не будет достигнут баланс нагрузки ($\delta = 0$) или не будет выполнено максимально допустимое количество итераций `maxiter`. Если по указанным условиям итерационный процесс завершён, проверяется корректность полученного решения v^{t+1}. Если решение некорректно или небаланс нагрузки превышает допустимый максимум `maxnb`, то задаются новые начальные условия и итерационный процесс повторяется (производится рестарт алгоритма).

В табл. 1–4 приведены следующие результаты серий из 100 испытаний алгоритма при отображении графов программ на компьютер с четырьмя ядрами ($m = 4$), полученные на процессоре Pentium (R) Dual-Core CPU E 52000, 2,5 ГГц:

I_a — среднее число итераций по 100 испытаниям;

I_m — максимальное число итераций;

t_a — среднее время выполнения алгоритма отображения (в секундах);

t_m — максимальное время выполнения алгоритма (в секундах);

N_a — среднее количество рестартов алгоритма при невыполнении условия `(δ > maxnb || noncorrect())`;

N_m — максимальное количество рестартов алгоритма при невыполнении условия `(δ > maxnb || noncorrect())`;

C_a — средний суммарный объём данных, передаваемых между ядрами компьютера для вычисленного отображения;

C_m — максимальный суммарный объём данных, передаваемых между ядрами компьютера для вычисленного отображения.

Отметим, что при указанных выше весах c_{xy} величины C_a и C_m задают количество рёбер графа программы $G_p(V_p, E_p)$, соединяющих вершины этого графа, отображённые на разные ядра компьютера.

Таблица 1

Независимые задания

n	I_a	I_m	t_a	t_m	N_a	N_m
4	61	374	0,0084	0,109	0,51	6
8	50	265	0,026	0,235	0,59	5
16	50	275	0,05	0,579	0,41	7
32	57	356	0,066	0,781	0,28	3
64	70	222	0,441	2,92	0,82	5

Таблица 2

Граф программы — линейка

n	I_a	I_m	t_a	t_m	N_a	N_m	C_a	C_m
4	75	312	0,0027	0,032	0,19	2	3	3
8	68	423	0,003	0,062	0,05	2	4,23	7
16	70	515	0,014	0,156	0,17	3	5,13	10
32	83	285	0,046	0,454	0,21	2	7,24	13
64	112	529	0,334	2,985	0,47	4	11,17	17

Таблица 3

Граф программы — кольцо

n	I_a	I_m	t_a	t_m	N_a	N_m	C_a	C_m
4	110	348	0,0061	0,047	0,37	3	4	4
8	71	243	0,0044	0,047	0,12	2	4,86	7
16	59	213	0,0078	0,141	0,07	2	5,53	10
32	74	273	0,055	0,796	0,26	2	7,82	15
64	97	216	0,336	5,93	0,43	5	11,68	19

Таблица 4

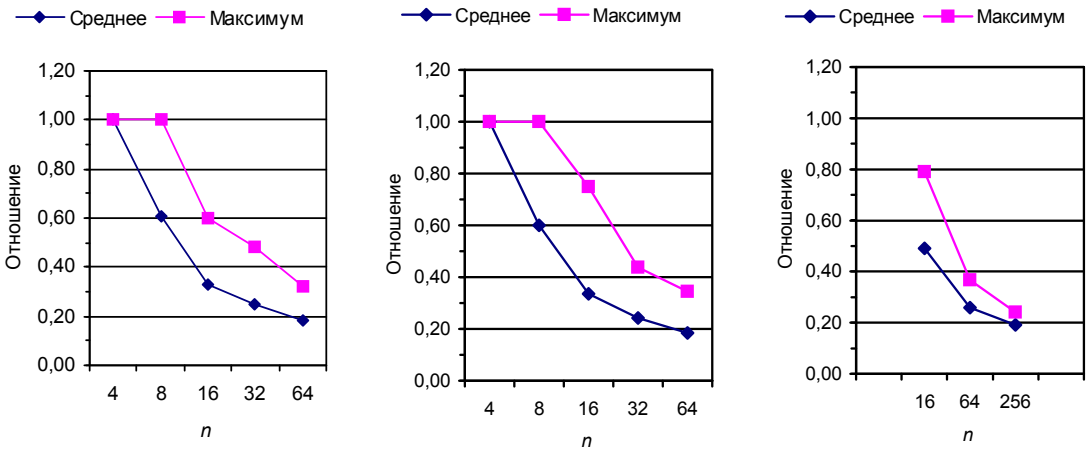
Граф программы — решётка

n	I_a	I_m	t_a	t_m	N_a	N_m	C_a	C_m
4	104	353	0,0064	0,047	0,54	3	4	4
16	60	166	0,0097	0,032	0,24	2	11,41	17
64	89	306	0,283	1,77	0,62	4	29,82	40
256	571	1000	15,8	99,7	1,28	15	92,32	123

Для всех случаев, указанных в табл. 1–4, получены полностью сбалансированные отображения ($\delta = 0$), за исключением отображения решётки с числом вершин $256 = 16 \times 16$, где $0 < \delta < 0{,}01$. Рассматривались решетки с $n = k \times k$ вершинами, $k = 4, 8, 16$ (при $k = 2$ решётка вырождается в кольцо).

На рис. 2 приведены графики отношений $r_a = C_a/C_{\max}$ и $r_m = C_m/C_{\max}$, где $C_{\max}$ — число рёбер графа программы, для линейки ($C_{\max} = n - 1$), кольца ($C_{\max} = n$) и квадратной решётки ($C_{\max} = \sqrt{n}(\sqrt{n} - 1)$) соответственно. Графики показывают, что несмотря на рост объёма информации, передаваемой между ядрами системы, с увеличением числа вершин графа программы относительная доля рёбер графа, приходящихся на межъядерные взаимодействия, существенно уменьшается.

На рис. 3–5 приведены примеры отображений решётки (рис. 3) и нерегулярных сеток (рис. 4 и 5). Одинаковыми знаками помечены вершины графа программы, отображённые на одно и то же ядро. Рисунки свидетельствуют о субоптимальности полученных отображений.



а *б* *в*

Рис. 2. Отношения r_a (среднее) и r_m (максимум): *а* — линейка; *б* — кольцо; *в* — рёшетка

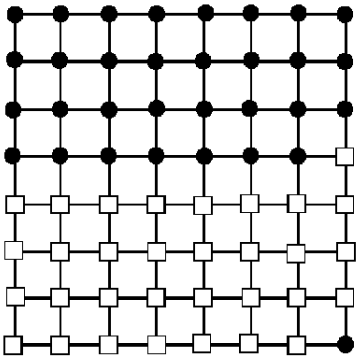


Рис. 3. Пример отображения решётки 8×8 вершин на систему из двух ядер

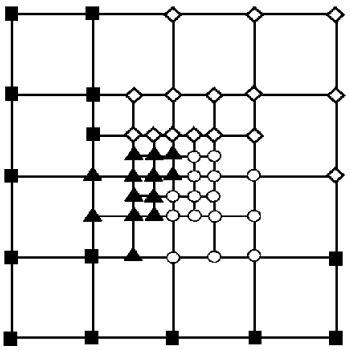


Рис. 4. Пример отображения прямоугольной нерегулярной сетки на систему из четырёх ядер ($n = 57$)

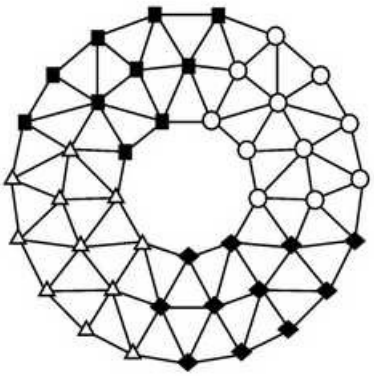


Рис. 5. Пример отображения триангуляционной нерегулярной сетки на систему из четырёх ядер ($n = 43$)

Эксперименты показали, что предложенный алгоритм отображения, основанный на использовании сети Вана [7, 8], позволяет получить для компьютера с числом ядер $m = 2, 4, 8$:

1) полный баланс вычислительной нагрузки (нагрузка распределяется равномерно) для типовых графов параллельных программ (пустой граф, линейка, решётка, кольцо) с числом вершин $n \in \{2, 4, 8, 16, 32, 64\}$;
2) субоптимальные отображения для нерегулярных сеток с несколькими десятками вершин (небаланс нагрузки не превышает 5 %);
3) существенное снижение доли рёбер графа программы, приходящихся на межъядерные обмены, при увеличении числа вершин графа программы.

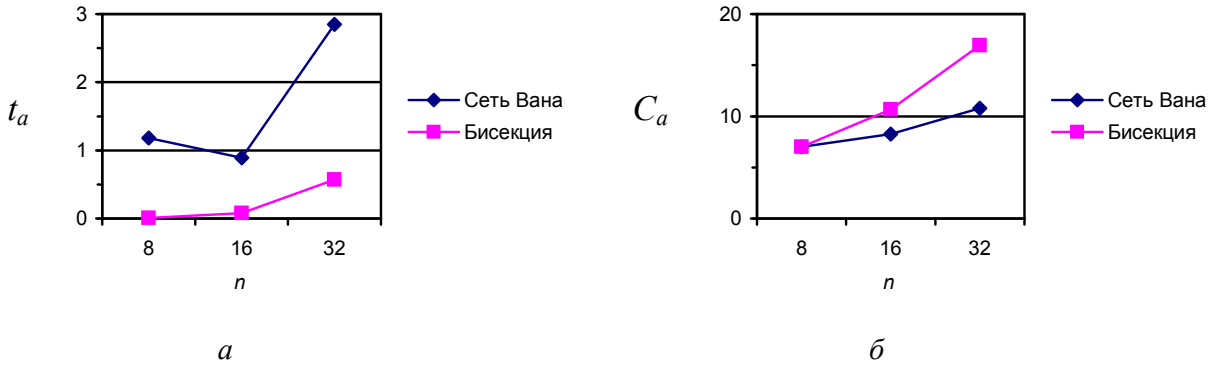
Табл. 5 показывает ($n = 32$), что при числе ядер $m = 2, 4$ времена работы алгоритма отображения t_a и t_m и количества рестартов алгоритма малы, но при увеличении числа ядер до $m = 8$ эти величины резко возрастают.

Таблица 5

Граф программы — линейка ($n = 32$)

m	I_a	I_m	t_a	t_m	N_a	N_m	C_a	C_m
2	14	81	0,0026	0,031	0,58	6	5,69	13
4	83	285	0,046	0,454	0,21	2	7,24	13
8	177	375	2,85	27,39	0,58	23	10,79	18

Проведены эксперименты по отображению графов программ для числа ядер $m = 8$ рекурсивной бисекцией, когда граф программы разбивается на два подграфа, которые, в свою очередь, рекурсивно разбиваются на подграфы вплоть до заданного числа ядер. Каждый шаг рекурсивной бисекции реализуется сетью Вана. Установлено, что рекурсивная бисекция существенно сокращает время отображения (рис. 6, *а*) при равномерном распределении вычислительной нагрузки, но затраты C_a и C_m на межъядерные обмены информацией в полученном отображении при этом возрастают (рис. 6, *б*).



а *б*

Рис. 6. Сравнение отображений линейки на компьютер с числом ядер сетью Вана и рекурсивной бисекцией: *а* — по среднему времени отображения; *б* — по средним затратам на межъядерные обмены в полученном отображении

Случай большого числа вершин (сотни и тысячи) можно свести к рассмотренным выше, используя методику Карписа — Кумара [8] огрубления графа программы.

Заключение

Рассмотрена задача отображения графа параллельной программы со взвешенными вершинами (процессами) и рёбрами (межпроцессорными обменами) на произвольный взвешенный граф распределённой вычислительной системы. Предложен алгоритм решения этой задачи, основанный на использовании сетей Хопфилда. Алгоритм

протестирован на частном случае рассмотренной задачи — отображении ряда графов параллельных программ (однородное множество независимых заданий — пустой граф, линейка, кольцо, решётка, неоднородная решетка) на многоядерный компьютер. Эксперименты показали, что для графов параллельных программ с несколькими десятками вершин предложенный алгоритм позволяет получить хорошо сбалансированные субоптимальные отображения: величина относительного небаланса нагрузки на ядро не превышает 5 %; подграф графа программы, отображённый на отдельное ядро, имеет близкое к минимуму количество внешних рёбер, соответствующих межъядерным обменам.

ЛИТЕРАТУРА

1. *Корнеев В. В.* Параллельные вычислительные системы. М.: Нолидж, 1999. 320 с.
2. *Bokhari S. H.* On the mapping problem // IEEE Trans. Comp. 1981. V. C-30. No. 3. P. 207–214.
3. *Тарков М. С.* Вложение структур параллельных программ в структуры живучих распределенных вычислительных систем // Автометрия. 2003. Т. 39. № 3. С. 84–96.
4. *Осовский С.* Нейронные сети для обработки информации. М.: Финансы и статистика, 2002. 344 с.
5. *Меламед И. И.* Нейронные сети и комбинаторная оптимизация // Автоматика и телемеханика. 1994. № 4. С. 3–40.
6. *Smith K. A.* Neural networks for combinatorial optimization: a review of more than a decade of research// INFORMS J. Computing. 1999. V. 11. No. 1. P. 15–34.
7. *Hung D. L. and Wang J.* Digital hardware realization of a recurrent neural network for solving the assignment problem // Neurocomputing. 2003. V. 51. P. 447–461.
8. *Karypis G. and Kumar V.* Multilevel k-way partitioning scheme for irregular graphs // J. Parallel and Distributed Computing. 1998. V. 48. P. 96–129.