УДК 681.324

Алгоритмы вложения параллельных программ в иерархические распределённые вычислительные системы

М. Г. Курносов[1]

Рассматривается задача оптимального вложения в иерархические распределённые вычислительные системы (ВС) параллельных программ с целью минимизации времени их выполнения. Предложены эвристические алгоритмы решения задачи. Эффективность алгоритмов подтверждена результатами численных и натурных экспериментов по вложению тестовых MPI-программ из пакетов NAS Parallel Benchmarks и High-Performance Linpack в действующие вычислительные кластеры. Приводится описание созданного программного средства оптимизации вложения MPI-программ в вычислительные SMP-кластеры.

Ключевые слова: вложение параллельных программ, MPI, вычислительные кластеры, распределённые вычислительные системы.

1. Введение

Возрастающая потребность в решении сложных задач науки и техники привела к созданию распределённых вычислительных систем (ВС). В архитектурном плане распределённая ВС представляется множеством взаимодействующих элементарных машин, оснащённых средствами коммуникаций и внешними устройствами [1, 2]. Элементарная машина (ЭМ) – это основной функциональный и структурный элемент ВС; конфигурация ЭМ допускает варьирование в широких пределах – от процессорного ядра до ЭВМ. Все основные ресурсы распределённых ВС (арифметико-логические устройства, память, средства управления и коммуникаций) являются логически и технически рассредоточенными. Число ЭМ в распределённых ВС допускает варьирование от нескольких единиц до сотен тысяч (например, в МВС-15000ВМ число процессоров равно 1148, в IBM Roadrunner – 122400, а в системах IBM BlueGene второго поколения до 884736).

Время выполнения параллельных программ на распределённых ВС существенно зависит от того, насколько они эффективно вложены в систему. Под эффективным вложением понимается такое распределёние ветвей параллельной программы между ЭМ системы, при котором достигаются минимумы накладных расходов на межмашинные обмены информацией и дисбаланса загрузки ЭМ.

Распределённые вычислительные системы 70–90 годов XX века характеризовались однородными структурами сетей межмашинных связей. Для таких систем разработаны эффективные методы и алгоритмы оптимизации вложения параллельных программ. Первые работы были выполнены в Институте математики Сибирского отделения АН СССР [2, 3].

Современные распределённые ВС являются мультиархитектурными. В зависимости от уровня рассмотрения их функциональных структур, они могут выглядеть и как MISD, и как

[1] Работа выполнена при поддержке РФФИ (гранты № 08-07-00018, 08-07-00022, 07-07-00142, 08-08-00300) и Совета по грантам Президента РФ (грант № НШ-2121.2008.9).

SIMD, и как MIMD системы. Для таких систем характерны иерархическая организация и различные пропускные способности каналов связи между их ресурсами (вычислительными узлами, ЭМ, процессорами и их ядрами).

При разработке параллельных программ для распределённых ВС наибольшее распространение получила модель передачи сообщений (в частности, библиотеки MPI и PVM). В этой модели параллельная программа характеризуется информационным графом, вершинам которого соответствуют параллельные ветви, а ребрам – информационные обмены между ними.

На рис. 1 приведён пример вложения параллельной MPI-программы – теста High-Performance Linpack в двухузловой вычислительный кластер. Коммуникационная среда кластера имеет иерархическую организацию. Процессорные ядра одного узла могут обмениваться информацией через общую память, а разных узлов – через сеть связи Gigabit Ethernet. Время выполнения теста с вложением стандартными средствами (первый случай) составляет 118 с., а с учётом структуры распределённой ВС и информационного графа задачи (второй случай) – 100 с. Видно, что даже на небольших системах оптимизация вложения позволяет сократить время выполнения параллельных программ.

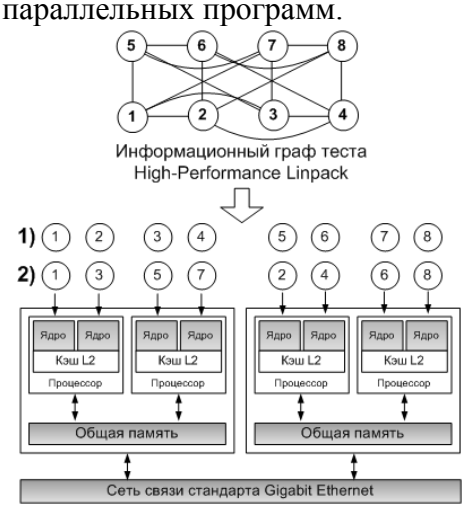


Рис. 1. Пример вложения теста High-Performance Linpack
(8 параллельных ветвей) в вычислительный кластер:
1 – вложение без учёта структуры графа и системы, *2* – вложение с учётом структуры

В данной работе рассматривается задача оптимального вложения параллельных программ в иерархические распределённые ВС. Для приближённого решения задачи предлагаются эвристические алгоритмы.

2. Предшествующие работы

Существующие методы и алгоритмы вложения параллельных программ в ВС различаются по типу систем, на которые они ориентированы, по используемым моделям параллельных программ, показателям оптимальности вложения, точности, централизованности или децентрализованности алгоритмов, по статической или динамической схеме формирования вложений.

В работах [4–10] предложены алгоритмы вложения параллельных программ, в которых ВС представлена в виде графа межмашинных связей. Предполагается, что информация между ЭМ передаётся по кратчайшему пути и все каналы связи однородны. В качестве показателя производительности канала связи между ЭМ используется длина пути (в смысле теории графов). В работах [11–16] описаны алгоритмы вложения в системы фиксированной структуры (например, 3*D*-торы, гиперкубы и т. д.). Применение подобных алгоритмов осложнено

тем, что современные ВС мультиархитектурны, их коммуникационные среды имеют иерархическую организацию и, как следствие, неоднородные каналы связи между ЭМ. Кроме того, большинство систем строится на базе коммутаторов, в которых применяются алгоритмы динамической маршрутизации. Поэтому целесообразно использование моделей вычислительных систем, в которых производительность каналов связи характеризуется такими показателями, как пропускная способности и латентность.

Эффективное вложение в ВС параллельной программы требует учёта и структуры информационных обменов между параллельными ветвями. В работах [4, 6, 7] предложены алгоритмы, опирающиеся лишь на структуру информационного графа программы и игнорирующие объёмы и интенсивность межмашинных обменов. Однако учёт таких параметров практически необходим. Анализ популярных свободно распространяемых параллельных программ, библиотек и тестов производительности High-Performance Linpack, NAS Parallel Benchmarks и SPEC MPI2007 показывает, что большинство из них характеризуется неоднородными информационными графами. Неоднородность выражается в объёмах данных, передаваемых между ветвями, размерах используемых сообщений, а также в схемах реализуемых обменов.

В качестве показателей оптимальности вложения используют время выполнения параллельной программы или функцию, минимизация которой доставляет минимум времени выполнения программы.

Алгоритмы вложения, предложенные в [5, 17–22], опираются на предположение о том, что время выполнения параллельной программы – это сумма времён выполнения её ветвей. Однако практика решения промышленных задач и анализа их производительности показывает, что время выполнения программы определяется максимальным из времён выполнения её ветвей. Кроме того, в ряде работ [11, 18, 23, 24] используются составные показатели в виде линейной комбинации двух функций, например, средней загруженности элементарных машин в системе и времени передачи данных по каналам связи. Использование подобных показателей осложнено выбором коэффициентов пропорциональности, которыми связанны функции и зачастую их несовместимыми размерностями. Например, размерность первой функции – количество операций, а второй – единицы времени.

В данной работе предлагаются алгоритмы оптимизации вложения параллельных программ в современные мультиархитектурные распределённые ВС, характеризующиеся иерархической организацией коммуникационных сред.

3. Вложение параллельных программ в иерархические распределённые ВС

3.1. Модель иерархической распределённой ВС

Пусть иерархическая распределённая ВС укомплектована N однородными ЭМ. Коммуникационная среда системы может быть представлена в виде дерева, содержащего L уровней. Каждый уровень системы образован отдельным видом функциональных модулей (например, телекоммуникационные шкафы, вычислительные узлы и т. п.), которые объединены каналами связи своего уровня. Для описания системы приняты следующие обозначения: n_l – количество элементов на уровне $l \in \{1, 2, \ldots, \mathrm{L}\}$; n_{lk} – количество прямых дочерних узлов элемента $k \in \{1, 2, \ldots, n_l\}$, находящегося на уровне l; $g(l, k_1, k_2)$ – номер уровня, на котором находится элемент, являющийся ближайшим общим предком для элементов $k_1, k_2 \in \{1, 2, \ldots, n_l\}$ с уровня l; b_l – значение пропускной способности каналов связи на уровне l ($[b_l]$ = байт/с); C_{lk} – множество элементарных машин, принадлежащих потомкам элемента k с уровня l; $c_{lk} = |C_{lk}|$; $C_{11} = C$; $C = \{1, 2, \ldots, N\}$.

На структуру дерева наложены следующие ограничения $\forall l \in \{1,2,\ldots,L\}$, $\forall k_1, k_2 \in \{1,2,\ldots,n_l\}$:

$$n_{lk_1} = n_{lk_2}, \tag{1}$$

$$c_{lk_1} = c_{lk_2}. \tag{2}$$

Первое ограничение (1) гарантирует, что элементы одного уровня имеют одинаковое количество дочерних узлов. Второе (2) обеспечивает равенство числа ЭМ, принадлежащих потомкам элементов одного уровня.

На рис. 2 приведён пример вычислительного кластера из 3 двухпроцессорных узлов на базе двухъядерных процессоров AMD Opteron 275. В качестве элементарных машин выступают процессорные ядра. Первый уровень коммуникационной среды образован сетью связи стандарта InfiniBand, через которую взаимодействуют узлы, второй уровень представлен шиной HyperTransport, через которую взаимодействуют процессоры узла, третий уровень – средства доступа процессорных ядер к их общей памяти.

На рис. 2 ближайшим общим предком для элементов 3, 5 уровня 3 является элемент 1 уровня 1, т.е. $g(3, 3, 5) = 1$.

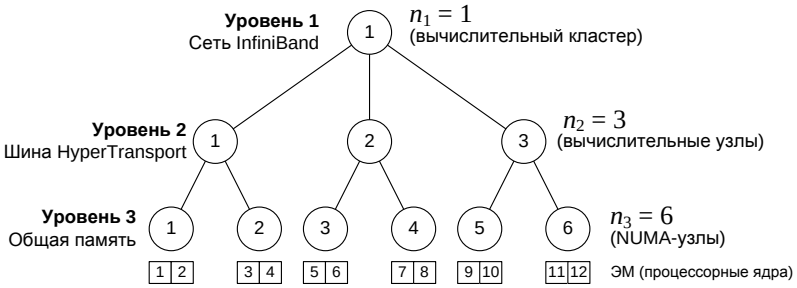


Рис. 2. Пример иерархической организации коммуникационной среды ВС:
$N = 12$; $L = 3$; $n_{23} = 2$; $C_{23} = \{9, 10, 11, 12\}$; $c_{23} = 4$; $g(3, 3, 4) = 2$; $z(1, 7) = 1$

3.2. Оценка ожидаемого времени выполнения параллельных программ на вычислительных системах

Выразим ожидаемое время выполнения параллельной программы на иерархической распределённой ВС. Считаем, что параллельная программа представлена информационным графом $G = (V, E)$, где $V = \{1, 2, \ldots, M\}$ – множество параллельных ветвей программы; $E \subseteq V \times V$ – множество информационно-логических связей между ветвями (обмены информацией). Через d_{ij} обозначим вес ребра $(i, j) \in E$, отражающий объём данных, передаваемый по нему за время выполнения программы ($[\, d_{ij} \,]$ = байт).

Вложение параллельной программы в ВС характеризуется инъективной функцией $f : V \to C$, ставящей в соответствие ветвям параллельной программы элементарные машины системы. Функция f задается значениями x_{ij}:

$$X = \{ x_{ij} : i \in V, j \in C\}, \quad x_{ij} = \begin{cases} 1, & \text{если } f(i) = j; \\ 0 & \text{иначе.} \end{cases}$$

Время T выполнения параллельной программы с заданным вложением X в систему определяется максимальным из времён выполнения её ветвей. По условию, ЭМ системы имеют одинаковую производительность, поэтому в функции $T(X)$ будем учитывать только накладные расходы на межмашинные обмены. Тогда

$$T(X) = \max_{i \in V}\{t_i\} = \max_{i \in V}\left\{ \sum_{j=1}^{M} \sum_{p=1}^{N} \sum_{q=1}^{N} x_{ip}\ x_{jq}\ t(i, j, p, q) \right\}. \quad (3)$$

Для оценки времени передачи сообщений между параллельными ветвями будем использовать модель Хокни (R. Hockney) [25]. Тогда функция $t(i, j, p, q)$ может быть записана в виде $t(i, j, p, q) = d_{ij} / b_{z(p,q)}$. Функция $z(p, q)$ ставит в соответствие ЭМ p и q номер уровня элемента, являющегося ближайшим общим предком для них (говоря иначе, уровень, через который они взаимодействуют). На рис. 2 ЭМ 1 и 7 принадлежат различным узлам, взаимодействие между которыми осуществляется через сеть InfiniBand, следовательно, $z(1, 7) = 1$.

3.3. Задача оптимального вложения параллельных программ в ВС

Учитывая инъективность функции *f*, сформулируем задачу оптимального вложения параллельной программы в ВС с иерархической организацией. В качестве показателя оптимальности вложения будем использовать время (3) выполнения параллельной программы

$$T(X) = \max_{i \in V}\left\{ \sum_{j=1}^{M} \sum_{p=1}^{N} \sum_{q=1}^{N} x_{ip}\ x_{jq}\ d_{ij} / b_{z(p,q)} \right\} \to \min_{(x_{ij})} \quad (4)$$

при ограничениях:

$$\sum_{j=1}^{N} x_{ij} = 1, \quad i = 1,2,\ldots,M, \quad (5)$$

$$\sum_{i=1}^{M} x_{ij} \le 1, \quad j = 1,2,\ldots,N, \quad (6)$$

$$x_{ij} \in \{0,1\}, \quad i \in V, \ j \in C. \quad (7)$$

Ограничения (5), (7) гарантируют назначение каждой ветви параллельной программы на единственную ЭМ, ограничения (6) обеспечивают назначение на машину не более одной ветви.

Задача (4) – (7) относится к дискретной оптимизации и является трудноразрешимой. Предложим приближённые алгоритмы её решения.

3.4. Иерархический метод вложения параллельных программ в ВС

Для решения задачи вложения параллельных программ в иерархические распределённые ВС разработан метод, основанный на решении задачи разбиении взвешенного графа на *k* непересекающихся подмножеств. Рассмотрим последнюю задачу.

3.5. Задача оптимального разбиения графа на *k* непересекающихся подмножеств

Пусть дан взвешенный граф $G' = (V', E')$; требуется отыскать разбиение вершин $V' = \{1, 2, \ldots, n\}$ графа на *k* непересекающихся подмножеств V'_1, V'_2, …, V'_k с целью минимизации максимальной суммы весов рёбер, инцидентных любому подмножеству разбиения. Количество элементов в каждом из подмножеств не должно превышать заданного числа *s*.

Обозначим $E'(i, j)$ – множество рёбер, соединяющих вершины из подмножеств V_i' и V_j', $i, j \in \{1, 2,\ldots, k\}$

$$E'(i, j) = \{(u, v) \in E' : u \in V_i', v \in V_j', i \neq j\};$$

$c(u, v, i, j)$ – вес ребра, инцидентного вершинам из разных подмножеств $u \in V_i'$ и $v \in V_j'$,

$$c(u, v, i, j) = w(u, v)\, W(i, j),$$

где $w(u, v)$ – вес ребра $(u, v) \in E'$, $W(i, j)$ – весовой коэффициент для рёбер, соединяющих вершины из подмножеств i и j.

Учитывая ограничения, накладываемые на подмножества V_i', получаем задачу оптимального разбиения графа на k непересекающихся подмножеств

$$F(V_1', V_2', ..., V_k') = \max_{i=1,...,k} \left\{ \sum_{j=1}^{k} \sum_{(u,v) \in E'(i, j)} c(u, v, i, j) \right\} \to \min_{(V_1', V_2', ..., V_k')} \tag{8}$$

при ограничениях:

$$V_1' \cap V_2' \cap \ldots \cap V_k' = \varnothing, \tag{9}$$

$$V_1' \cup V_2' \cup \ldots \cup V_k' = V', \tag{10}$$

$$|V_i'| > 0, \quad i = 1, 2, ..., k, \tag{11}$$

$$|V_i'| \leq s, \quad i = 1, 2, ..., k. \tag{12}$$

На рис. 3 приведён пример взвешенного графа. На рис. 4 представлен возможный вариант разбиения графа на три подмножества.

Суммарный вес рёбер, инцидентных первому подмножеству V_1', равен $w(1, 5)W(1, 2) + w(6, 8)W(1, 3) + w(2, 3)W(1, 3) = 22$; для второго подмножества V_2' – 40; для третьего подмножества V_3' – 38. Значение целевой функции $F(V_1', V_2', V_3') = 40$.

Известно [26], что задача (8) – (12) разбиения взвешенного графа на k непересекающихся подмножеств является трудноразрешимой.

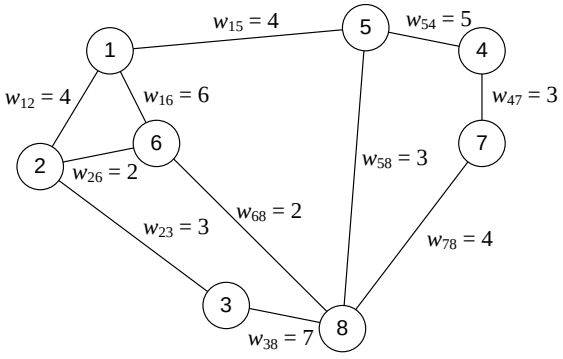


Рис. 3. Пример взвешенного графа

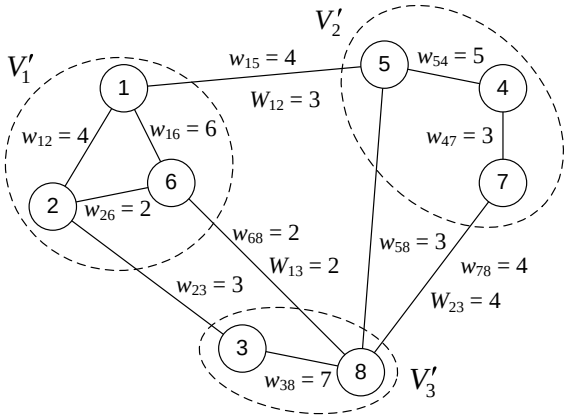


Рис. 4. Разбиение взвешенного графа на три подмножества:
$k = 3$; $s = 3$; $F(V'_1, V'_2, V'_3) = 40$

3.6. Метод вложения параллельных программ в ВС

Предложим метод решения задачи вложения параллельных программ в иерархические распределённые ВС. В основе метода лежит процедура разбиения информационного графа параллельной программы на подмножества с параллельными ветвями, обменивающимися большими объёмами данных, и их вложения в ЭМ, связанные быстрыми каналами связи.

Суть метода. Для исходных данных задачи (4) – (7) одним из известных методов [27–33] строится решение задачи (8) – (12) разбиения взвешенного графа на k непересекающихся подмножеств. В задаче (8) – (12) полагаем, что $V' = V$, $k = [(M-1)/c_{L1}]+1$, $s = c_{L1}$, $c(u, v, i, j) = d_{uv}/b_{g(L,i,j)}$. Решение – k подмножеств V'_1, V'_2, …, V'_k.

По решению задачи (8) – (12) строится решение задачи (4) – (7). Вложение $f : V \to C$ формируется следующим образом. Ветви с номерами из подмножества V'_i назначаются на ЭМ из множества C_{Li}, $i = 1, 2, ..., k$. Значение целевой функции T задачи (4) – (7) не зависит от выбора номера ЭМ во множестве C_{Li}. Это объясняется тем, что каналы связи между элементарными машинами в подмножестве C_{Li} имеют одинаковую производительность.

Обозначим описанный метод TMGP (Task Map Graph Partitioning). В листинге 1 приведён его псевдокод.

В псевдокоде метода используется структура данных очередь (*queue*) [34, 35], для которой определены операции *Enqueue*(*Q*, *i*) – добавление элемента *i* в конец очереди *Q*; *Enqueue*(*Q*, *Q'*) – добавление элементов очереди *Q'* в конец очереди *Q*; *Dequeue*(*Q*) – извлечения первого элемента из очереди *Q*; *QueueSize*(*Q*) – определение количества элементов в очереди *Q*. Трудоёмкость перечисленных операций составляет *O*(1).

Здесь и далее при описании методов и алгоритмов используются обозначения, принятые в [35].

Листинг 1. Псевдокод метода TMGP

Входные данные: $G = (V, E)$ – информационный граф параллельной программы; N, L, n_l, n_{lk}, C, C_{lk}, c_{lk} – описание иерархической распределённой ВС ($l \in \{1, 2, \ldots, L\}$, $k \in \{1, 2, \ldots, n_l\}$).

Выходные данные: $x[i, j]$ – вложение; $x[i, j] = 1$, если ветвь i назначена на ЭМ j, иначе $x[i, j] = 0$.

1 $k \leftarrow [(M-1)/c_{L1}]+1$

2 $s \leftarrow c_{L1}$

3 $(p_1, p_2, \ldots, p_M) \leftarrow PartGraph(G, k, s)$

4 **for** $i \leftarrow 1$ **to** M **do**

5 $c \leftarrow Dequeue(C_{p_i})$

6 $x[i,c] \leftarrow 1$

7 **end for**

Поясним смысл основных этапов работы метода. Функция *PartGraph* реализует решение задачи (8) – (12) одним из известных методов и возвращает список p_1, p_2, …, p_M номеров подмножеств разбиения, к которым принадлежат вершины графа G; $p_i \in \{1,2,..., k\}$ – номер подмножества, которому принадлежит вершина i информационного графа G. C_j – очередь номеров ЭМ из подмножества C_{Lj}, $j \in \{1, 2,..., n_L\}$.

На рис. 5 приведён пример вложения информационного графа задачи в вычислительный кластер с иерархической организацией. Разбиение графа получено приближённым решением задачи (8) – (12) с параметрами: $M = 12$, $c_{L1} = 4$, $k = [(M-1)/c_{L1}]+1 = 3$, $s = 4$.

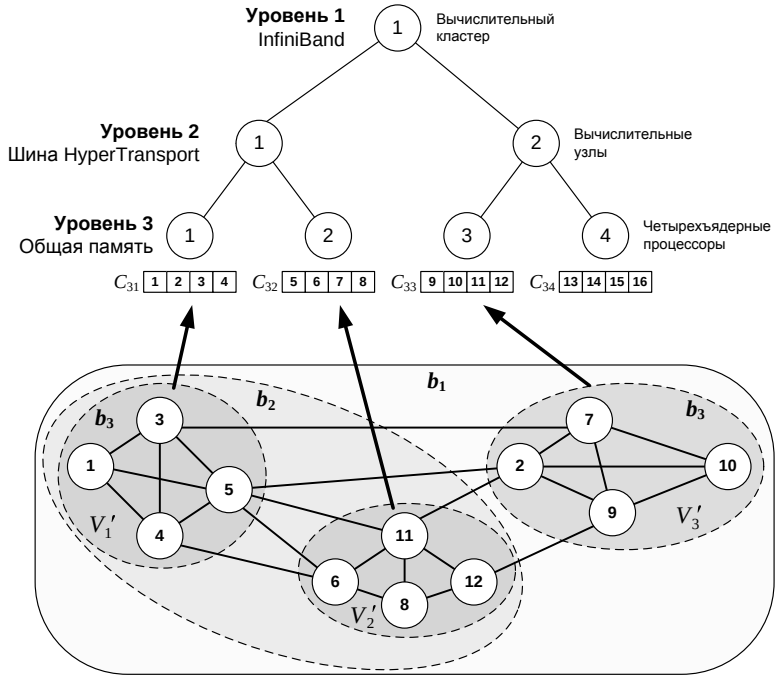


Рис. 5. Пример вложения информационного графа в иерархическую распределённую ВС: $N = 16$; $L = 3$; $M = 12$; $b_1 = 2$ ГБ/с; $b_2 = 6$ ГБ/с; $b_3 = 8$ ГБ/с

Важно отметить, что производительность каналов связи между элементарными машинами, входящими в подмножества C_{31}, C_{32}, C_{33}, C_{34}, различна. Для рассматриваемого примера функция $c(u,v,i,j) = w(u,v)\,W(i,j) = d_{uv}/b_{g(L,i,j)}$ строилась следующим образом: $w(u,v) = d_{uv}$, $W(i,j) = 1/b_{g(L,i,j)}$, $u,v \in \{1, 2,..., 12\}$, $i, j = \{1, 2, 3, 4\}$, $W(1, 2) = 1/(6 \cdot 2^{30})$, $W(1, 3) = 1/(2 \cdot 2^{30})$, $W(1, 4) = 1/(2 \cdot 2^{30})$, $W(2, 3) = 1/(2 \cdot 2^{30})$, $W(2, 4) = 1/(2 \cdot 2^{30})$, $W(3, 4) = 1/(6 \cdot 2^{30})$.

Обозначим количество операций, требуемых для выполнения функции *PartGraph*, через T_{GP}. Очевидно, что вычислительная сложность метода TMGP равна $T_{TMGP} = O(T_{GP} + M)$.

Задача оптимального разбиения взвешенного графа на k непересекающихся подмножеств трудноразрешима. Для её решения существуют [27–33] точные и приближённые методы и алгоритмы, характеризующиеся различными временны́ми и пространственными сложностями. Большинство известных алгоритмов имеют практические ограничения на размер обрабатываемых графов. Интерес представляют многоуровневые алгоритмы разбиения графов [28–33], позволяющие получать субоптимальные решения задачи (8) – (12) и характеризующиеся невысокой вычислительной сложностью.

3.7. Многоуровневые методы разбиения графов

Рассмотрим основные этапы работы многоуровневых методов разбиения графов (рис. 6). Имеется исходный граф $G_0 = (V_0, E_0)$, который требуется разбить на k непересекающихся подмножеств так, чтобы целевая функция (8) приняла субминимальное значение.

1) Этап сжатия графа (coarsening). Исходный граф G_0 преобразуется в последовательность графов G_1, G_2, …, G_m меньших размерностей, таких, что $|V_0| > |V_1| > \ldots > |V_m|$. Графы меньших размерностей формируются построением наибольших паросочетаний (maximal matching) и стягиванием согласованных вершин в мультивершины. На рис. 7 приведён пример стягивания вершин графа G_{i-1} и формирования графа G_i.

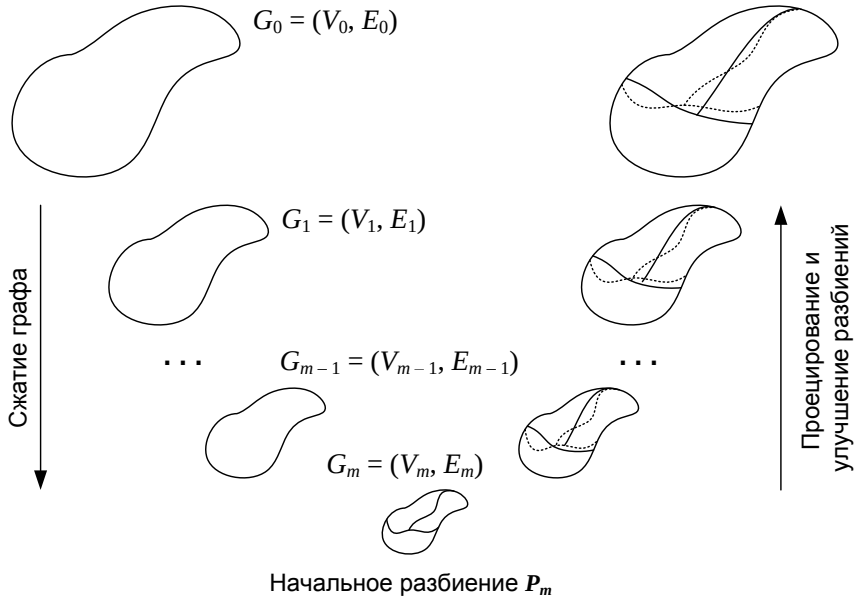
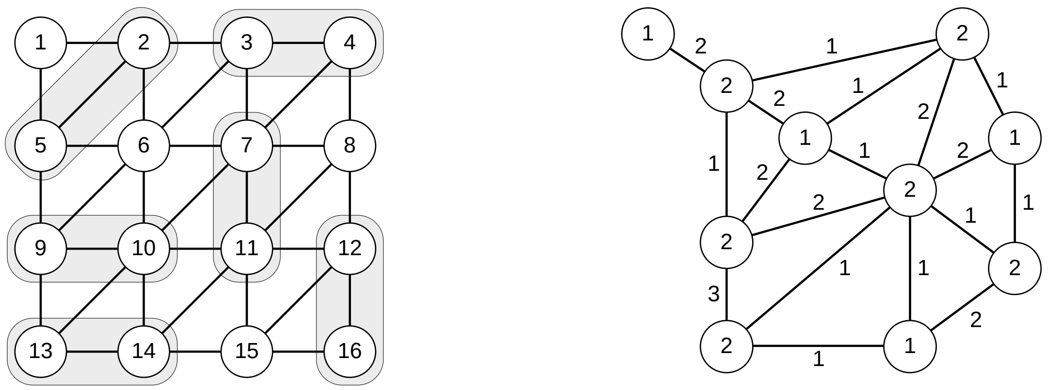


Рис. 6. Схема работы многоуровневых алгоритмов разбиения графов:
——— – исходное разбиение P_i; ---------- – улучшенное разбиение P_{i-1}

2) Этап начального разбиения (partitioning). Граф $G_m = (V_m, E_m)$ малой размерности разбивается одним из известных алгоритмов на k непересекающихся подмножеств. Результат – это разбиение P_m. На данном этапе применяются алгоритмы рекурсивной бисекции графа, а для формирования разделения графа на две части используются комбинаторные и спектральные методы [33]. Бисекция графа – это разделение его вершин на два непересекающихся подмножества.

3) Этап улучшения разбиений (refinement). Разбиение P_m проецируется на граф $G_{m-1} = (V_{m-1}, E_{m-1})$. Получается разбиение P_{m-1}, которое улучшается (в смысле показателя (8)) известными алгоритмами локальной оптимизации. Процедура проецирования разбиения и его улучшения продолжается, пока не будет достигнут исходный граф G_0. В основе большинства алгоритмов улучшения разбиений лежит модификация нетрудоёмкой эвристики Кернигана – Лина (Kernighan-Lin) [36].



а *б*

Рис. 7. Пример сжатия графа:
а – исходный граф с выделенными рёбрами паросочетания; *б* – сжатый граф с мультивершинами

Основное достоинство многоуровневых алгоритмов – это возможность обрабатывать графы больших размерностей за приемлемое время.

3.8. Алгоритм вложения параллельных программ в иерархические распределённые ВС

На основе метода TMGP разработан алгоритм вложения параллельных программ в иерархические распределённые ВС. Разбиение информационных графов программ осуществляется алгоритмом, предложенным в работе [31].

Обозначим алгоритм TMMGP (Task Map Multilevel Graph Partitioning). Опишем его основные шаги.

На первом этапе осуществляется сжатие информационного графа $G = (V, E)$ задачи и построение последовательности G_1, G_2, …, G_m. Сжатие осуществляется до тех пор, пока размеры графа G_m не позволят эффективно применить к нему алгоритм рекурсивной бисекции (recursive bisection). Эмпирически установлено [29–31], что сжатие целесообразно осуществлять, пока в графе G_m не останется M' мультивершин

$$M' = \max\left\{\frac{M}{40 \log_2 k}, 20\,k\right\}.$$

Сжатие графа $G_i = (V_i, E_i)$ осуществляется стохастическим алгоритмом HEM (Heavy Edge Matching) [30]. Вершины графа посещаются в случайном порядке. Если вершина $u \in V_i$ не посещалась, то среди её смежных необработанных вершин отыскивается вершина $v \in V_i$ с максимальным значением веса ребра $(u, v) \in E_i$. Ребро $(u, v) \in E_i$ включается в паросочетание, вершины u, v отмечаются как посещённые. Трудоёмкость алгоритма HEM равна $O(|E_i|)$. Оценим количество итераций сжатия графов алгоритмом HEM для достижения графа G_m: $|V_m| \le M'$.

При сжатии графов и переходе от графа G_i к графу G_{i+1} количество вершин в последнем уменьшается вдвое. Справедливо неравенство

$$|V_i| > |V_{i+1}| \ge |V_i|/2.$$

Нетрудно заметить, что количество итераций m сжатия графов есть m-й член геометрической прогрессии со знаменателем $1/2$ и первым членом, равным 1. Отсюда по определению m-го члена геометрической прогрессии

$$M' = M\,(1/2)^{m-1},\ m = \log_2 \frac{M}{M'} + 1.$$

На втором этапе рекурсивной бисекцией графа G_m на k подмножеств формируется начальное разбиение P_m. Бисекция реализуется алгоритм GGP (Graph Growing Partitioning) [31, 33]. В основе алгоритма GGP лежит идея обхода графа G_m в ширину начиная с периферической вершины [33]. Чтобы разбить граф на k подмножеств, требуется выполнить порядка $\log_2 k$ его бисекций. Так как граф G_m имеет небольшую размерность, то время реализации рекурсивной бисекции с использованием алгоритма GGP незначительно и требует выполнения порядка $O(|E| \cdot \log_2 k)$ операций.

На третьем этапе разбиение P_i, $i = m, m-1, \ldots, 1$ проецируется на граф G_{i-1} и улучшается нетрудоёмким алгоритмом BKL (Boundary Kernighan-Lin), основанным на эвристике Кернигана – Лина (Kernighan-Lin, KL), а также на её менее трудоёмкой модификации – эвристике FM (Fiduccia-Mattheyses) [36]. Трудоёмкость алгоритма BKL равна $O(|E_i|)$ [31].

В листинге 2 приведён псевдокод алгоритма TMMGP.

Листинг 2. Псевдокод алгоритма TMMGP

Входные данные: $G = (V, E)$ – информационный граф параллельной программы; N, L, n_l, n_{lk}, C, C_{lk}, c_{lk} – описание ВС ($l \in \{1, 2, \ldots, L\}$, $k \in \{1, 2, \ldots, n_l\}$).

Выходные данные: $x[i, j]$ – вложение; $x[i, j] = 1$, если ветвь i назначена на ЭМ j, иначе $x[i, j] = 0$.

```
1   k ← [(M −1)/c_L1]+1
2   s ← c_L1
3   M' ← max{ M/(40 log2 k), 20 k }
4   m ← log2 M/M' +1
5   for i ← 1 to m do
6     G_i ← CoarseGraphHEM(G_{i−1})
7   end for
8   P_m ← RecursiveBisectionGGP(G_m, k, s)
9   for i ← m to 1 do
10    P_{i−1} ← ProjectPartition(G_{i−1}, P_i)
11    P_{i−1} ← RefinePartitionBKL(G_{i−1}, P_{i−1})
12  end for
13  for i ← 1 to M do
14    c ← Dequeue(C_{P0,i})
15    x[i,c] ← 1
16  end for
```

Поясним смысл основных функций, используемых в листинге 2. Функция *CoarseGraphHEM*(G_{i-1}) реализует сжатие графа G_{i-1} алгоритмом HEM; функция *RecursiveBisectionGGP*(G_m, k, s) осуществляет рекурсивную бисекцию графа G_m на k под-

множеств; функция *ProjectPartition*(G_{i-1}, P_i) проецирует разбиение P_i на граф G_{i-1}; функция *RefinePartitionBKL*(G_{i-1}, P_{i-1}) улучшает разбиение P_{i-1} графа G_{i-1} алгоритмом BKL.

Рассмотрим работу алгоритма TMMGP на примере вложения параллельной MPI-программы NAS Parallel Benchmark Conjugate Gradient (NPB CG) (рис. 8), реализующей решение системы линейных алгебраических уравнений (СЛАУ) методом сопряжённых градиентов, в вычислительный кластер на базе двухъядерных процессоров Intel Xeon 5150 (рис. 9). Коммуникационная среда кластера имеет иерархическую организацию. Первый уровень коммуникационной среды образован сетью связи стандарта Gigabit Ethernet, второй уровень – средствами доступа многоядерных процессоров к общей памяти. Каждый узел укомплектован парой двухъядерных процессоров.

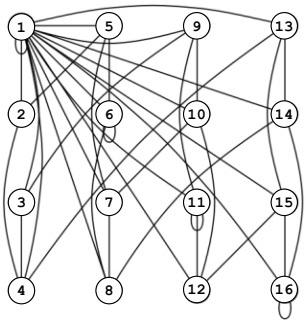


Рис. 8. Информационный граф параллельной программы NPB CG: $M = 16$

Для графа на рис. 9 задача разбиения $(8)-(12)$ решалась со следующими параметрами: $k = [(M-1)/c_{L1}]+1 = (15/4)+1 = 4$, $s = 4$. Сжатие и улучшение разбиений графа не осуществлялось, так как $M' = 80$ и $m < 0$. На рисунке изображён конечный результат рекурсивной бисекции графа алгоритмом GGP.

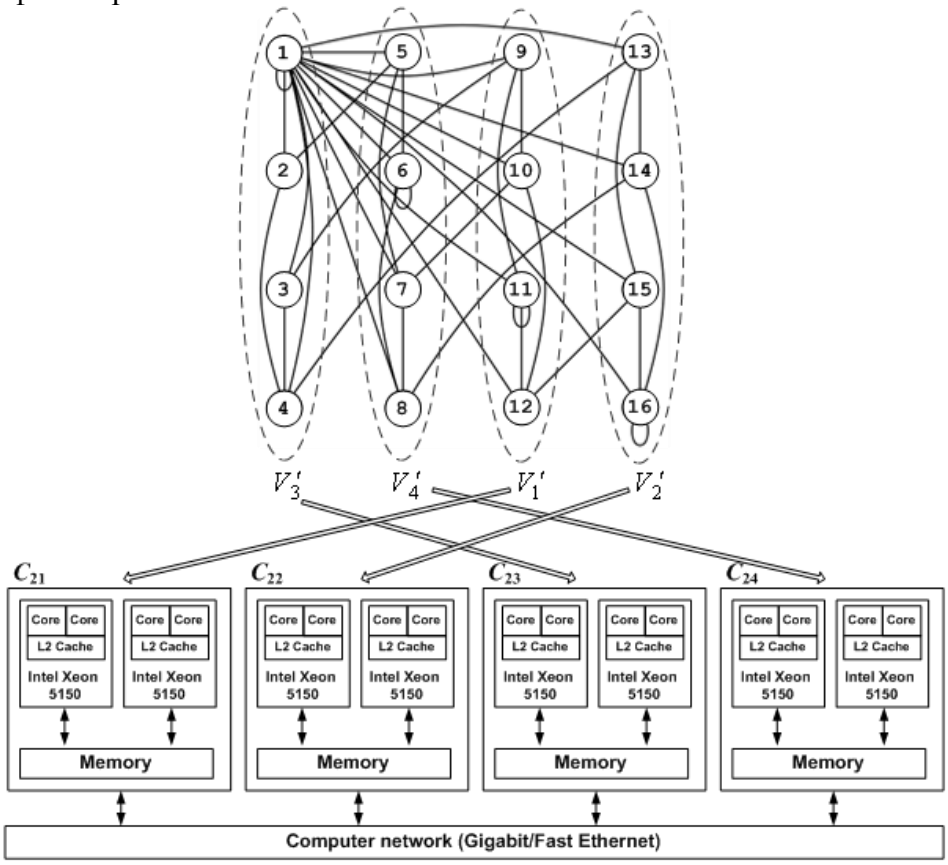


Рис. 9. Вложение параллельной программы в вычислительный кластер:
$M = 16$; $N = 16$; $L = 2$; $c_{21} = c_{22} = c_{23} = c_{24} = 4$

Результирующее вложение

$$X = \{x_{1,9} = 1, x_{2,10} = 1, x_{3,11} = 1, x_{4,12} = 1, x_{5,13} = 1, x_{6,14} = 1, x_{7,15} = 1, x_{8,16} = 1, x_{9,1} = 1, x_{10,2} = 1, x_{11,3} = 1, x_{12,4} = 1, x_{13,5} = 1, x_{14,6} = 1, x_{15,7} = 1, x_{16,8} = 1\}.$$

Справедливо утверждение.

Утверждение 1. *Вычислительная сложность алгоритма TMMGP равна*

$$T_{TMMGP} = O(|E| \log_2 k).$$

Доказательство. Трудоёмкость метода определяется вычислительной сложностью трёх последовательно выполняемых этапов: сжатие графа, начальное разбиение и улучшение разбиений. Обозначим через T_1 трудоёмкость этапа сжатия графа. Для сжатия графов G_0, G_1, …, G_m требуется выполнить $m + 1$ раз алгоритм HEM. С переходом от графа к графу количество рёбер сокращается в два раза. Для графа G_i справедливо $|E_i| = |E| / 2^i$, $i = 1, 2, ..., m$. Тогда трудоёмкость сжатия графов (строки 5–7) равна

$$T_1 = \frac{|E|}{2^0} + \frac{|E|}{2^1} + ... + \frac{|E|}{2^m} = |E| \sum_{i=0}^{m} \frac{1}{2^i} = |E| \left(2 - \frac{2}{2^m} \right) = |E| \left(\frac{2M - M'}{M} \right), \frac{2M - M'}{M} \le 2.$$

Следовательно, $T_1 = O(|E|)$.

Трудоёмкость формирования начального разбиения равна $T_2 = O(|E| \log_2 k)$.

Трудоёмкость проецирования и улучшения разбиений алгоритмом BKL равна $T_3 = O(|E|)$. Доказательство аналогично рассуждениям о вычислительной сложности этапа сжатия графов G_0, G_1, …, G_m.

Суммарная трудоёмкость алгоритма TMMGP равна

$$T_{TMMGP} = T_1 + T_2 + T_3 = O(|E| + |E| \log_2 k + |E|) = O(|E| \log_2 k).$$

Утверждение доказано.

3.9. Вложения параллельных программ в подсистемы иерархических ВС

Рассмотрим модификацию математической модели коммуникационной среды иерархической распределённой ВС, описанной в п. 3.1. Допустим существование конфигураций подсистем, в которых элементы одного уровня могут содержать различное количество элементарных машин. Другими словами, отменим выполнение условия (2). Следовательно, $\exists k_1, k_2 \in \{1, 2, \ldots, n_L\}$:

$$c_{Lk_1} \ne c_{Lk_2}. \quad (13)$$

На рис. 10 приведён пример подсистемы с иерархической организацией. Серым цветом обозначены ЭМ, недоступные для использования в данный момент, например, вследствие их выхода из работоспособного состояния или использования для решения других параллельных задач.

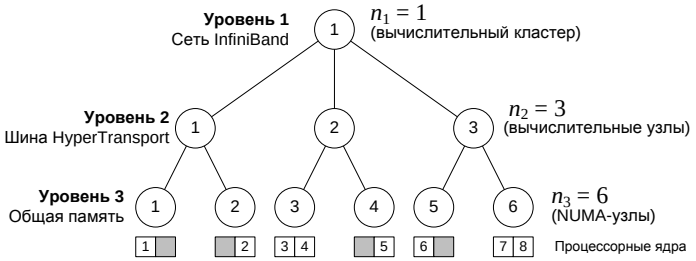


Рис. 10. Пример подсистемы с иерархической организацией:
$N = 8$; $L = 3$; $c_{31} = 1$; $c_{33} = 2$

Такие подсистемы могут возникать и в результате работы алгоритмов выделения ЭМ для параллельных программ. В приведенном примере видно, что $c_{31} \neq c_{33}$, следовательно, конфигурация системы удовлетворяет условию (13).

Предложим алгоритм поиска приближённых решений задачи (4) – (7) вложения параллельных программ в подсистемы, удовлетворяющие условию (13).

3.10. Эвристический алгоритм вложения параллельных программ в подсистемы иерархических ВС

Алгоритм основан на распределении смежных ветвей параллельной программы, обменивающихся большими объёмами данных, по ЭМ, которые объединены быстрыми каналами связи. Входными данными алгоритма являются информационный граф $G = (V, E)$ параллельной программы и описание подсистемы.

Средняя скорость передачи данных от элементарной машины $p \in \{1, 2, \ldots, N\}$ до остальных машин подсистемы характеризуется средним геометрическим значением B_p пропускных способностей каналов связи между ней и остальными машинами. Значение B_p может быть рассчитано по определению

$$B_p = \left(\prod_{\substack{q \in C \\ q \neq p}} b_{z(p,q)} \right)^{\frac{1}{N-1}} = \exp\left(\frac{1}{N-1} \sum_{\substack{q \in C \\ q \neq p}} \ln(b_{z(p,q)}) \right).$$

Оценка объёмов данных, передаваемых между ветвью $i \in V$ и смежными ветвями, аналогично осуществляется через среднее геометрическое значение D_i объёмов данных d_{ij}, $i, j \in \{1, 2, \ldots, M\}$, передаваемых между параллельными ветвями.

$$D_i = \left(\prod_{j \in Adj(i)} d_{ij} \right)^{\frac{1}{|Adj(i)|}} = \exp\left(\frac{1}{|Adj(i)|} \sum_{j \in Adj(i)} \ln d_{ij} \right),$$

где $Adj(i)$ – множество ветвей, смежных с ветвью i.

Будем считать, что информационный граф $G = (V, E)$ представлен списками смежности. Через Q_i обозначим список смежности вершины $i \in \{1, 2, \ldots, M\}$.

Опишем основные этапы работы алгоритма.

Для каждой элементарной машины $p \in \{1, 2, \ldots, N\}$ рассчитывается значение B_p. Номера элементарных машин сортируются по невозрастанию значений B_p и в таком порядке помещаются в очередь C.

Для упорядочивания последовательности из n целых чисел, лежащих в интервале $[0, k]$, используется алгоритм сортировки подсчётом (Counting Sort), вычислительная сложность которого оценивается величиной $\Theta(k + n)$ [35].

Для каждой вершины $i \in \{1, 2, \ldots, M\}$ информационного графа параллельной программы рассчитывается значение D_i. Аналогично, номера ветвей сортируются по невозрастанию значений D_i и в таком порядке помещаются в очередь Q. По значениям D_i сортируются и вершины в списках смежности Q_i, $i = 1, 2, \ldots, M$.

Осуществляется обход вершин информационного графа $G = (V, E)$. Из очереди Q извлекается вершина с максимальным значением D_i, она и её смежные ветви, взятые в порядке

невозрастания значений D_i, распределяются на свободные ЭМ с максимальными значениями B_p (первые в очереди C). После чего из очереди Q извлекается очередная необработанная вершина, и процедура вложения текущей и смежных с ней ветвей повторяется.

Обозначим алгоритм TMGT (Task Map Graph Traversal). В листинге 3 приведён его псевдокод.

Листинг 3. Псевдокод алгоритма TMGT

Входные данные: $G = (V, E)$ – информационный граф программы; N, L, n_l, n_{lk}, C, C_{lk}, c_{lk}, b_{pq} – описание подсистемы ($l \in \{1, 2, \ldots, L\}$, $k \in \{1, 2, \ldots, n_l\}$, $p, q \in \{1, 2, \ldots, N\}$).

Выходные данные: $x[i, j]$ – вложение; $x[i, j] = 1$, если ветвь i назначена на ЭМ j, иначе $x[i, j] = 0$.

```
1   for p ← 1 to N do                          /* Расчёт компонент вектора B */
2     Enqueue(C, p)
3     B[p] ← 0
4     for q ← 1 to N do
5       if p ≠ q then
6         B[p] ← B[p] + ln(b_pq)
7     end for
8     B[p] ← exp(1 / (N – 1) * B[p])
9   end for
10  CountingSort(C)          /* Сортировка номеров ЭМ по значениям B[p] */
11  for i ← 1 to M do                          /* Расчёт компонент вектора D */
12    Enqueue(Q, i)
13    mapped[i] ← 0
14    D[i] ← 0
15    for (для) каждого j ∈ Adj(i) do
16      D[i] ← D[i] + ln(d_ij)
17    end for
18    D[i] ← exp(1 / |Adj(i)| * D[i])
19  end for
20  CountingSort(Q)       /* Сортировка номеров ветвей по значениям D[i] */
21  for i ← 1 to M do                    /* Сортировка списков смежности */
22    CountingSort(Q_i)
23  end for
24  while QueueSize(Q) > 0 do              /* Обход информационного графа */
25    i ← Dequeue(Q)
26    if mapped[i] = 0 then                        /* Если ветвь i невложена */
27      Enqueue(Q', i)
28      Enqueue(Q', Q_i)
29      while QueueSize(Q') > 0 do          /* Вложение смежных ветвей */
30        i ← Dequeue(Q')
31        if mapped[i] = 0 then
32          c ← Dequeue(C)
33          x[i, c] ← 1
34          mapped[i] ← 1
35        end if
36      end while
37    end if
38  end while
```

Рассмотрим работу алгоритма на примере вложения информационного графа (рис. 11) в подсистему с иерархической организацией (рис. 12).

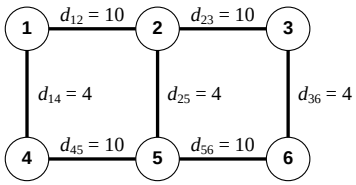


Рис. 11. Пример информационного графа параллельной программы:
$M = 6$; $Q_1 = (2, 4)$; $Q_2 = (1, 3, 5)$; $Q_3 = (2, 6)$; $Q_4 = (5, 1)$; $Q_5 = (4, 6, 2)$; $Q_6 = (5, 3)$

Выпишем значения $b_{pq} = b_{z(p,q)}$, $p, q \in \{1, 2, \ldots, 7\}$ для системы, приведённой на рис. 12.

$$\left\| b_{pq} \right\| = \begin{pmatrix} 0 & 6 & 2 & 2 & 2 & 2 & 2 \\ 6 & 0 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 0 & 8 & 2 & 2 & 2 \\ 2 & 2 & 8 & 0 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 0 & 8 & 6 \\ 2 & 2 & 2 & 2 & 8 & 0 & 6 \\ 2 & 2 & 2 & 2 & 6 & 6 & 0 \end{pmatrix} \tag{14}$$

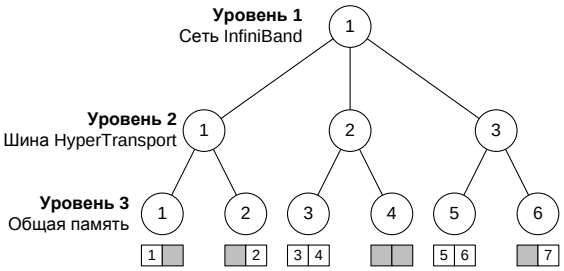


Рис. 12. Пример подсистемы с иерархической организацией:
$N = 7$; $L = 3$; $b_1 = 2$ ГБ/с; $b_2 = 6$ ГБ/с; $b_3 = 8$ ГБ/с

Рассчитаем по (14) вектор средних геометрических значений пропускных способностей каналов связи между ЭМ подсистемы:

$$B = (2.40;\ 2.40;\ 2.52;\ 2.52;\ 3.03;\ 3.03;\ 2.88).$$

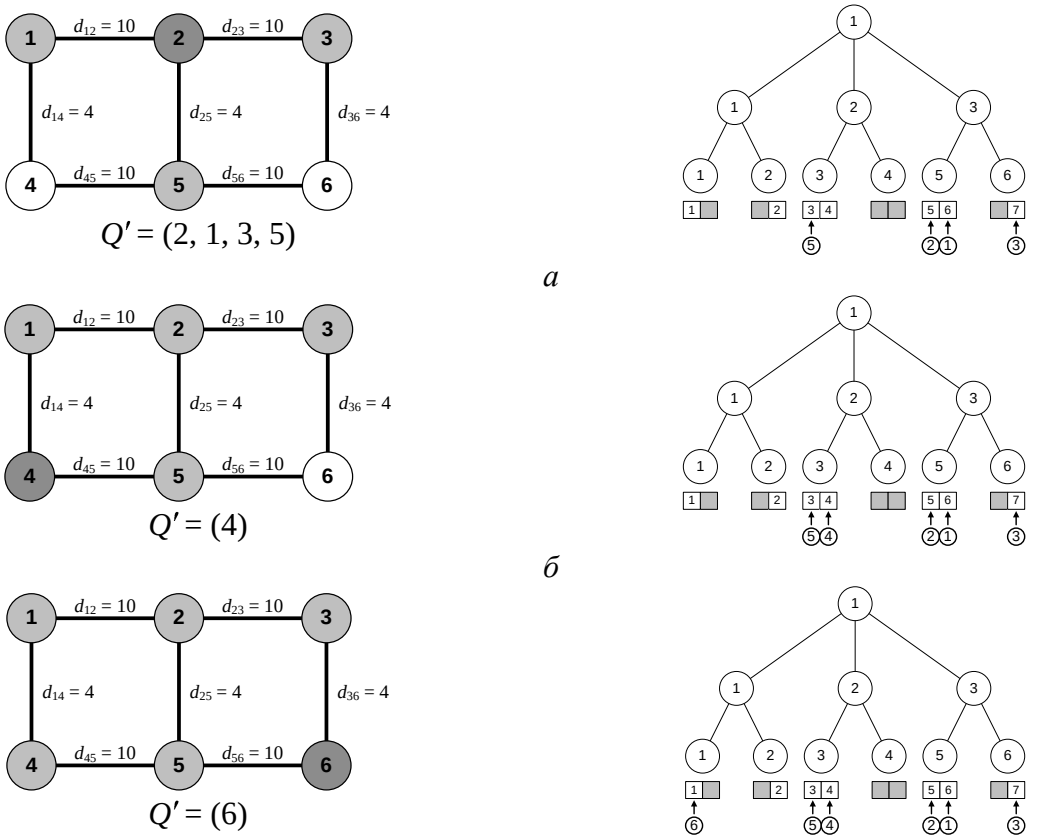
Аналогично запишем вектор D для ветвей параллельной программы:

$$D = (6.32;\ 7.37;\ 6.32;\ 6.32;\ 7.37;\ 6.32).$$

Упорядоченные списки номеров параллельных ветвей Q и элементарных машин C примут следующий вид:

$$Q = (2, 5, 1, 3, 4, 6),\ C = (5, 6, 7, 3, 4, 1, 2).$$

На рис. 13 представлены промежуточные результаты работы алгоритма между итерациями цикла 24–38. Для каждой итерации на графе серым цветом отмечены обработанные вершины. Начальная вершина итерации отмечена тёмно-серым цветом.



а

б

в

Рис. 13. Пример работы алгоритма TMGT. Обход графа (строки 24–38):
а – первая итерации; *б* – вторая итерация;
в – третья итерация

Конечный результат работы алгоритма – вложение X:

$$X = \{x_{16} = 1, x_{25} = 1, x_{37} = 1, x_{44} = 1, x_{53} = 1, x_{61} = 1\}.$$

Вычислим значение целевой функции (4) для построенного вложения.

$$T(X) = \max_{i \in V} \left\{ \sum_{j=1}^{M} \sum_{p=1}^{N} \sum_{q=1}^{N} x_{ip}\ x_{jq}\ d_{ij} \,/ b_{z(p,q)} \right\},$$

$$t_1 = \frac{13}{4}, \quad t_2 = \frac{59}{12}, \quad t_3 = \frac{11}{3}, \quad t_4 = \frac{13}{4}, \quad t_5 = \frac{33}{4}, \quad t_6 = 7, \ ([t_i] = \text{байт·с/ГБ}).$$

$$T(X) = \max_{i \in V} \{t_i\} = \frac{33}{4}.$$

Относительно вычислительной сложности алгоритма TMGT справедливо утверждение.

Утверждение 2. *Вычислительная сложность алгоритма TMGT равна*

$$T_{TMGT} = O(N^2 + M|E|).$$

Доказательство. Введём обозначения: T_1 – трудоёмкость вычисления компонент вектора B (строки 1–9); T_2 – трудоёмкость сортировки номеров ЭМ по значениям соответствующих компонент вектора B (строка 10); T_3 – трудоёмкость расчёта компонент вектора D (строки 11–19); T_4 – трудоёмкость сортировки номеров параллельных ветвей (строка 20); T_5 – трудоёмкость сортировки списков смежности (строки 21–23); T_6 – трудоёмкость обхода информационного графа (строки 24–38).

Трудоёмкость алгоритма составляет $O(T_1 + T_2 + T_3 + T_4 + T_5 + T_6)$.

Трудоёмкость расчёта векторов B и D определяется трудоёмкостью расчёта среднего геометрического значения n чисел, которая оценивается величиной $O(n)$. Поэтому $T_1 = O(N^2)$, $T_3 = O(M|E|)$. Сортировка подсчётом n неотрицательных целых чисел требует выполнения порядка $O(n)$ операций, следовательно, $T_2 = O(N)$ и $T_4 = O(M)$, $T_5 = O(M|E|)$.

В строках 24–38 реализуется обход графа, трудоёмкость которого эквивалентна трудоёмкости алгоритма обхода графа в ширину, представленного списками смежности, и составляет $T_6 = O(M + |E|)$.

По условию задачи $N \geq M$, следовательно, трудоёмкость алгоритма

$$T_{TMGT} = O(N^2 + N + M|E| + M + M|E| + M + |E|) = O(N^2 + M|E|)$$

Утверждение доказано.

Предложенные алгоритмы имеют полиномиальную трудоёмкость и позволяют строить приближённые решения задачи оптимального вложения параллельных программ в иерархические распределённые ВС.

4. Моделирование работы алгоритмов

4.1. Инструментарий оптимизации вложения параллельных MPI-программ

На основе предложенных алгоритмов вложения параллельных программ в иерархические распределённые ВС создано программное средство MPITaskMap, позволяющее организовывать выполнение MPI-программ с субоптимальным распределением их ветвей по ЭМ системы. На рис. 14 приведена схема работы пакета MPITaskMap.

Исполняемый код MPI-программ и её паспорт, в котором содержится описание информационного графа, поступают на вход пакета MPITaskMap. В зависимости от текущих настроек и конфигурации ВС, применяется один из алгоритмов вложения. Далее формируется сценарий запуска программы с субоптимальным распределением ветвей по ЭМ и осуществляется её выполнение.

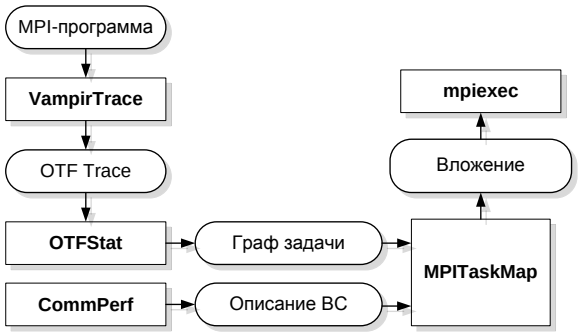


Рис. 14. Схема работы пакета MPITaskMap

Описание ВС формируется при помощи разработанной утилиты CommPerf, которая осуществляет оценку производительности каналов связи между элементарными машинами ВС на уровне протокола MPI.

Для анализа производительности MPI-программ и автоматизации формирования их информационных графов разработано программное средство OTFStat. Поддерживаются протоколы выполнения MPI-программ в формате Open Trace Format (OTF). Созданное средство позволяет получать информацию о структуре обменов между параллельными ветвями, дисбалансе их загрузки и коэффициентах накладных расходов.

Пакет MPITaskMap разработан на языке программирования C для операционной системы GNU/Linux.

4.2. Организация моделирования алгоритмов

Время работы и качество формируемых алгоритмами TMMGP и TMGT вложений сравнивалось с циклическим (Task Map Round Robin – TMRR) и линейным (Task Map Linear – TML) алгоритмами формирования вложений, которые реализованы в современных системах управления ресурсами (TORQUE, Sun GridEngine, IBM LoadLeveler и др.) и библиотеках стандарта MPI (MPICH2, OpenMPI).

Алгоритм TMRR распределяет параллельные ветви по ЭМ системы из N вычислительных узлов в следующем порядке: первая ветвь на первый вычислительный узел, вторая на второй, …, ветвь N на узел N, ветвь $N + 1$ на узел 1 и т. д. Алгоритм TML распределяет ветви на первые M свободных ЭМ.

Результаты численного моделирования проверялись натурными экспериментами по вложению тестовых параллельных MPI-программ в действующие вычислительные кластеры с иерархической организацией коммуникационных сред. В качестве тестовых параллельных MPI-программ были взяты программы из пакетов оценки производительности распределённых ВС NAS Parallel Benchmarks (NPB) и High Performance Linpack (HPL). Выбор программ обусловлен покрытием ими основных схем межмашинных обменов – в программах присутствуют как дифференцированные обмены, так и коллективные [1].

На рис. 15 представлен информационный граф теста HPL, реализующего решение системы линейных алгебраических уравнений (СЛАУ) методом LU-факторизации. На рис. 16 – информационный граф теста NPB Conjugate Gradient (NPB CG), осуществляющего решение СЛАУ методом сопряжённых градиентов. На рис. 17 – информационный граф теста NPB Multigrid, строящего решение трехмерного уравнения Пуассона многосеточным методом. Информационные графы сформированы средствами пакета OTFStat.

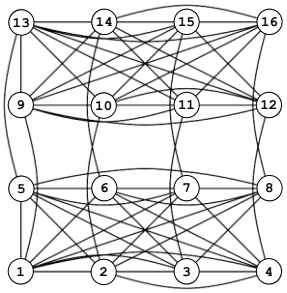


Рис. 15. Информационный граф теста HPL:
$M = 16$; PMAP = 0; BCAST = 5

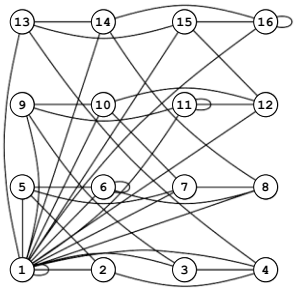


Рис. 16. Информационный граф теста NPB CG:
$M = 16$; CLASS = B

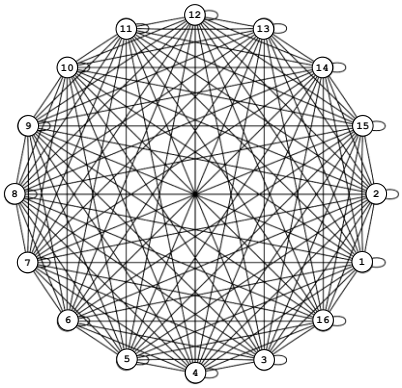


Рис. 17. Информационный граф теста NPB Multigrid:
$M = 16$; CLASS = B

Натурные эксперименты по вложению тестовых параллельных MPI-программ проводились на действующих вычислительных кластерах Xeon10 и Xeon16 – сегментах пространственно-распределённой мультикластерной вычислительной системы Центра параллельных вычислительных технологий ГОУ ВПО “СибГУТИ” и Лаборатории вычислительных систем Института физики полупроводников Сибирского отделения РАН.

Оба кластера укомплектованы двухпроцессорными узлами и имеют коммуникационную среду с иерархической организацией.

Кластер Xeon10 укомплектован 6 двухпроцессорными узлами на базе AMD Opteron 248. Коммуникационная среда кластера включает два уровня: первый уровень – сеть связи стандарта Gigabit Ethernet , второй – средства доступа процессоров к их общей памяти.

Кластер Xeon16 включает 4 двухпроцессорных узла. Каждый узел построен на базе двухъядерных процессоров Intel Xeon 5150. Первый уровень коммуникационной среды кластера образован сетями связи стандартов Gigabit и Fast Ethernet, второй уровень – средства доступа процессорных ядер к их общей памяти.

4.3. Моделирование работы алгоритма TMMGP

На рис. 18 и 19 приведены графики зависимости времени работы алгоритма от количества вершин и рёбер в информационном графе параллельной программы. Рассматривались графы следующих структур: 2*D*-решётка, звезда, кольцо, линейка. На рис. 18 представлены результаты вложения в ВС $N = 16384$, $L = 3$, $c_{L1} = 2$. Время работы алгоритма приведено для процессора Intel Xeon E5345.

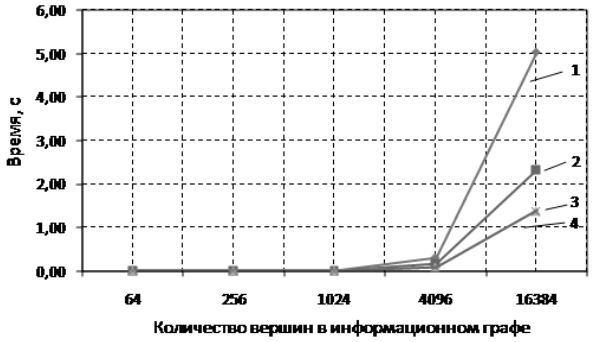


Рис. 18. Зависимость времени работы алгоритма TMMGP от количества *M* вершин в информационном графе: $c_{L1} = 2$;
1 – 2*D*-решётка; 2 – звезда; 3 – кольцо; 4 – линейка

На рис. 19 результаты вложения в систему $N = 16384$, $L = 3$, $c_{L1} = 8$.

Из графиков видно, что время работы алгоритма незначительно и для большемасштабных ВС сопоставимо с системным временем подготовки подсистемы для запуска ветвей параллельной программы. Графики подтверждают результаты утверждения 1. Видно, что с уменьшением *k* количества подмножеств разбиения время работы алгоритма уменьшается.

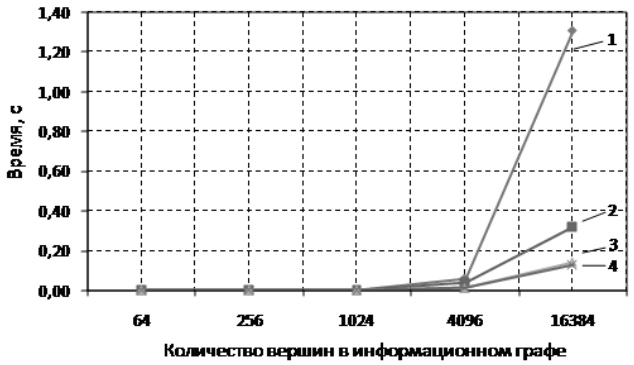


Рис. 19. Зависимость времени работы алгоритма TMMGP от количества M вершин в информационном графе: $c_{L1} = 8$;
1 – 2D-решётка; 2 – звезда; 3 – кольцо; 4 – линейка

Оценка качества формируемых алгоритмом TMMGP вложений осуществлялась для информационных графов со структурами: 2D-решётка, звезда, кольцо и линейка. Для сравнения приведены результаты работы алгоритмов TML и TMRR. В табл. 1 приведены результаты вложений параллельных программ, состоящих из $M = 512$ ветвей, в систему $N = 16384$, $L = 3$, $c_{L1} = 4$, $b_1 = 2 \cdot 2^{30}$, $b_2 = 6 \cdot 2^{30}$, $b_3 = 8 \cdot 2^{30}$.

Таблица 1. Результаты вложений параллельных программ, состоящих из $M = 512$ ветвей, в систему $N = 16384$, $L = 3$, $c_{L1} = 4$, $b_1 = 2 \cdot 2^{30}$, $b_2 = 6 \cdot 2^{30}$, $b_3 = 8 \cdot 2^{30}$

Структура информационного графа	Значение целевой функции T, с		
	$T(X_{TML})$, с	$T(X_{TMRR})$, с	$T(X_{TMMGP})$, с
2D-решётка	462	466	166
Звезда	125568	126525	125560
Кольцо	312	333	312
Линейка	312	333	312

В табл. 2 отражены результаты вложений параллельных программ ($M = 512$) в систему с параметрами: $N = 16384$, $L = 7$, $c_{L1} = 4$, $b_1 = 0.5 \cdot 2^{30}$, $b_2 = 1 \cdot 2^{30}$, $b_3 = 2 \cdot 2^{30}$, $b_4 = 3 \cdot 2^{30}$, $b_5 = 4 \cdot 2^{30}$, $b_6 = 6 \cdot 2^{30}$, $b_7 = 8 \cdot 2^{30}$.

Таблица 2. Результаты вложений параллельных программ ($M = 512$) в систему с параметрами: $N = 16384$, $L = 7$, $c_{L1} = 4$, $b_1 = 0.5 \cdot 2^{30}$, $b_2 = 1 \cdot 2^{30}$, $b_3 = 2 \cdot 2^{30}$, $b_4 = 3 \cdot 2^{30}$, $b_5 = 4 \cdot 2^{30}$, $b_6 = 6 \cdot 2^{30}$, $b_7 = 8 \cdot 2^{30}$

Структура информационного графа	Значение целевой функции T, с		
	$T(X_{TML})$, с	$T(X_{TMRR})$, с	$T(X_{TMMGP})$, с
2D-решётка	1712	1816	316
Звезда	432168	488700	423997
Кольцо	1062	1083	1062
Линейка	1062	1083	1062

Анализ результатов моделирования показал, что алгоритм TMMGP позволяет формировать эффективные вложения для насыщенных информационных графов с неоднородными весами рёбер. Возможности алгоритма по вложению параллельных программ со структурами информационных графов в виде линейки, кольца и звезды ограничены. Видно, что качество вложения зависит от разброса производительности каналов связи на различных уровнях коммуникационной среды. Чем больше уровней в коммуникационной среде и значительней разброс производительности каналов связи, тем выше эффективность вложений, формируемых алгоритмов TMMGP.

На втором этапе проводилась серия натурных экспериментов по вложению алгоритмом TMMGP промышленных MPI-программ NPB CG, NPB MG и HPL в действующие вычислительные кластеры. В табл. 3 приведены результаты выполнения MPI-программ с различными вариантами вложений на различных сетях связи кластера Xeon16. Время работы алгоритма TMMGP на всех тестовых задачах на процессоре Intel Core 2 Duo 2.13 GHz не превышало $5 \cdot 10^{-3}$ с.

Таблица 3. Результаты выполнения MPI-программ с различными вариантами вложений на различных сетях связи кластера Xeon16

Сеть связи	Время выполнения MPI-программы, с		
	$T(X_{TMRR})$, с	$T(X_{TMMGP})$, с	$T(X_{TMRR}) / T(X_{TMMGP})$
High-Performance Linpack			
Fast Ethernet	1108.69	911.81	1.22
Gigabit Ethernet	263.15	231.72	1.14
NPB Conjugate Gradient			
Fast Ethernet	726.02	400.36	1.81
Gigabit Ethernet	97.56	42.05	2.32
NPB Multigrid			
Fast Ethernet	23.94	23.90	1.00
Gigabit Ethernet	4.06	4.03	1.00

Данные в табл. 3 подтверждают результаты численного моделирования. Видно, что качество вложения в значительной степени зависит от структуры информационного графа задачи, а также от разброса значений показателей производительности на различных уровнях коммуникационной среды. Важно отметить, что существует класс параллельных программ, для которых возможности по оптимизации вложения существенно ограничены. К этому классу относятся программы, информационные графы которых являются полными и имеют однородные по объёму передаваемых данных связи между ветвями. Примером может служить тест NPB MG.

4.4. Моделирование работы алгоритма TMGT

Для исследования алгоритма TMGT формировались тестовые подсистемы с иерархической организацией коммуникационных сред. Рассматривались следующие конфигурации систем: $N \in \{2048, 4096, 8192\}$, $L \in \{2, 3, \ldots, 10\}$, $c_{L1} \in \{2, 4, 8\}$. Подсистемы формировались путём случайного равновероятного распределения параллельных задач различных рангов по ЭМ и их исключения из состава системы.

На рис. 20 приведён график зависимости времени работы алгоритма TMGT от количества ветвей в параллельной программе и структуры информационного графа. Результаты приведены для тестовой конфигурации ВС $N = 4096$ (количество доступных ЭМ), $L = 4$, $c_{L1} = 4$, $b_1 = 1 \cdot 2^{30}$, $b_2 = 2 \cdot 2^{30}$, $b_3 = 6 \cdot 2^{30}$, $b_4 = 8 \cdot 2^{30}$. Время работы алгоритма указано для процессора Intel Xeon E5345.

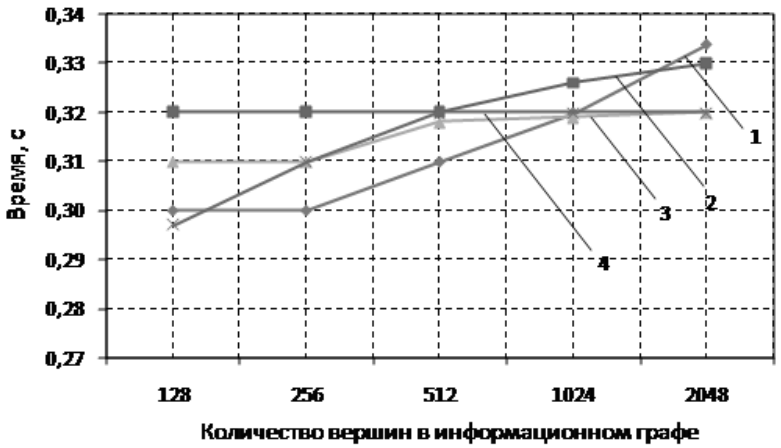


Рис. 20. Зависимость времени работы алгоритма TMGT
от количества M вершин в информационном графе:
1 – 2D-решётка; 2 – звезда; 3 – кольцо; 4 – линейка

На рис. 21 приведена зависимость времени выполнения алгоритма от количества ветвей (структура графа – 2D-решётка) в параллельной программе и ранга подсистемы.

Из графиков видно, что время работы алгоритма TMGT находится в зависимости от ранга подсистемы и количества рёбер в информационном графе параллельной программы. В целом, время работы алгоритма приемлемо для практического использования на подсистемах с рангом порядка десятка тысяч ЭМ.

Рис. 21. Зависимость времени работы алгоритма TMGT
от количества M вершин в информационном графе:
1 – N = 2048; 2 – N = 4096; 3 – N = 8192

В табл. 4 отражены результаты вложения параллельных программ (M = 128) в систему N = 4096, L = 4, c_{L1} = 8, $b_1 = 1 \cdot 2^{30}$, $b_2 = 2 \cdot 2^{30}$, $b_3 = 6 \cdot 2^{30}$, $b_4 = 8 \cdot 2^{30}$.

Качество формируемых алгоритмом TMGT вложений зависит от структуры информационного графа программы. В среднем на тестовых наборах алгоритм TMGT формировал вложения по значению целевой функции в 1.1 – 1.5 раза лучше алгоритма TML и 1.1 – 1.6 раз лучше алгоритма TMRR.

Таблица 4. Результаты вложения параллельных программ (M = 128) в систему N = 4096, L = 4, c_{L1} = 8, $b_1 = 1 \cdot 2^{30}$, $b_2 = 2 \cdot 2^{30}$, $b_3 = 6 \cdot 2^{30}$, $b_4 = 8 \cdot 2^{30}$

Структура информационного графа	Значение целевой функции T, с		
	$T(X_{TML})$, с	$T(X_{TMRR})$, с	$T(X_{TMGT})$, с
2D-решётка	1062	1333	790
Звезда	238520	251583	210104
Кольцо	562	583	312
Линейка	561	580	310

В табл. 5 приведены результаты натурного эксперимента по вложению тестовых MPI-программ (см. моделирование алгоритма TMMGP) в подсистему кластера Xeon16.

Таблица 5. Результаты натурного эксперимента по вложению тестовых MPI-программ в подсистему кластера Xeon16

Сеть связи	Время выполнения MPI-программы, с		
	$T(X_{TMRR})$, с	$T(X_{TMGT})$, с	$T(X_{TMRR})$ / $T(X_{TMGT})$
High-Performance Linpack			
Fast Ethernet	120.92	84.35	1.43
Gigabit Ethernet	25.10	19.75	1.27
NPB Conjugate Gradient			
Fast Ethernet	727.00	390.35	1.86
Gigabit Ethernet	101.14	41.64	2.42
NPB Multigrid			
Fast Ethernet	23.9	23.9	1.00
Gigabit Ethernet	4	4	1.00

5. Заключение

Учёт иерархической организации коммуникационных сред современных распределённых ВС и структур информационных графов задач при их вложении в вычислительные системы позволяет существенно сократить время выполнения параллельных программ.

Как видно из результатов моделирования, качество вложения в значительной степени зависит и от структуры информационного графа задачи. Например, граф теста NPB Multigrid является полным и имеет однородные по объёму передаваемых данных связи. Возможности по оптимизации вложения этого теста ограничены.

Разработанные алгоритмы вложения ориентированы на использование в системах управления ресурсами иерархических распределённых вычислительных систем, включая SMP-кластеры на базе многоядерных процессоров.

Литература

1. Хорошевский В. Г. Архитектура вычислительных систем. – М. : МГТУ им. Н. Э. Баумана, 2008. – 520 с.
2. Евреинов Э. В., Хорошевский В. Г. Однородные вычислительные системы – Новосибирск : Наука. Сибирское отд-е, 1978. – 319 с.

3. Евреинов Э. В., Косарев Ю. Г. Однородные универсальные вычислительные системы высокой производительности. – Новосибирск: Наука. Сибирское отд-е, 1966. – 308 с.
4. Bokhari S. H. On the mapping problem // IEEE Transactions on Computers. – 1981. – Vol. 30, № 3. – P. 207 – 214.
5. Agarwal T. Topology-aware task mapping for reducing communication contention on large parallel machines // Parallel and Distributed Processing Symposium. – 2006. – P. 10
6. Тарков, М.С. Вложение структур параллельных программ в структуры живучих распределённых вычислительных систем // Автометрия. – 2003. – Том 39, № 3. – С. 84 – 96.
7. Lee C., Bic L. On the mapping problem using simulated annealing // Proceedings of Computers and Communications. – 1989. – P. 40 – 44.
8. Tarkov M. S., Mun Y., Choi J., Choi H.-I. Mapping parallel programs onto distributed computer systems with faulty elements // Lecture Notes in Computer Science: Computational Science. – 2001. – P. 148 – 157.
9. Ucar B. Task assignment in heterogeneous computing systems // Journal of Parallel and Distributed Computing. – 2006. – Vol. 66, № 1. – P. 32 – 46.
10. Yu H., Chung I. H., Moreira J. Topology Mapping for Blue Gene/L Supercomputer // Proceedings of ACM/IEEE Conference Supercomputing. – 2006. – P. 52.
11. Монахов О. Г., Монахова Э. А. Параллельные системы с распределённой памятью: управление ресурсами и заданиями. – Новосибирск : Изд-во ИВМиМГ СО РАН, 2001. – 168 с.
12. Cho S. Y., Park K. H. Dynamic task assignment in heterogeneous linear array networks for metacomputing // Heterogeneous Computing Workshop. – 1994. – P. 66 – 71.
13. Ercal F., Ramanujan J., Sadayappan P. Task allocation onto a hypercube by recursive mincut bipartitioning // Journal of Parallel and Distributed Computing. – 1990. – Vol. 10, № 1, – P. 35 – 44.
14. Jacob J. C., Lee S.-Y. Task Spreading and Shrinking on Multiprocessor Systems and Networks of Workstations // IEEE Transactions on Parallel and Distributed Systems. – 1999. – Vol. 10, № 10. – P. 1082 – 1101.
15. Jose A. An approach to mapping parallel programs on hypercube multiprocessors // Proceedings of Workshop Parallel and Distributed Processing. – 1999. – P. 221 – 225.
16. Kalinowski T. Solving the mapping problem with a genetic algorithm on the MasPar-1 // Proceedings of the First International Conference on Massively Parallel Computing Systems. – 1994. – P. 370 – 374.
17. Ahmad I., Kafil M. A Parallel Algorithm for Optimal Task Assignment in Distributed Systems // Proceedings of the 1997 Advances in Parallel and Distributed Computing Conference. – 1997. – P. 284.
18. Bouvry P. Mapping and load balancing on distributed memory systems // Proceedings of Microprocessing. – Budapest, 1994.
19. Chen Hu. MPIPP: An Automatic Profileguided Parallel Process Placement Toolset for SMP Clusters and Multiclusters // Proceedings of the 20th annual international conference on Supercomputing. – 2006. – P. 353 – 360.
20. Chen S., Eshaghian M. A fast recursive mapping algorithm // Concurrency: Practice and Experience. – 1995. – Vol. 7, № 5. – P. 391 – 409.
21. Coli M., Palazzari P. Global execution time minimization by allocating tasks in parallel systems // Proceedings of the 3rd Euromicro Workshop on Parallel and Distributed Processing. – 1995. – P. 91.
22. G. Bhanot et al. Optimizing task layout on the Blue Gene/L supercomputer // IBM Journal of Research and Development. – 2005. – Vol. 49, № 2. – P. 489 – 500.
23. Hluchy L., Dobrovodsky M., Dobrucky M. Static Mapping Methods for Processor Networks [Электронный ресурс] // CiteSeer.IST, 2008. – Режим доступа: `http://citeseer.ist.psu.edu/472693.html` (дата обращения: 29.04.2009).

24. Salcedo-Sanz S., Hu Y., Yao X. Hybrid meta-heuristics algorithms for task assignment in heterogeneous computing systems // Computers and Operations Research. – 2006. – Vol. 33, № 3. – P. 820 – 835.
25. Hockney R. The communication challenge for MPP: Intel Paragon and Meiko CS-2 / R. Hockney. – Parallel Computing. – 1994. – Vol. 20, № 3. – P. 389 – 398.
26. Garey M., Johnson D., Stockmeyer L. Some simplified NP-complete graph problems. – Theoretical Computer Science. – 1976. – Vol. 1. – P. 237 – 267.
27. Bui T. N., Moon B. R. Genetic algorithm and graph partition // IEEE Transactions on Computers. – 1996. – Vol. 45, № 7. – P. 841 – 855.
28. Hendrickson B., Leland R. A multilevel algorithm for partitioning graphs // Proc. of ACM/IEEE conference on Supercomputing. – San Diego : IEEE Press, 1995.
29. Karypis G., Kumar V. Multilevel *k*-way partitioning scheme for irregular graphs // Journal of Parallel and Distributed computing. – 1998. – Vol. 48. – P. 96 – 129.
30. Karypis G., Kumar V. Analysis of multilevel graph partitioning // Proc. of ACM/IEEE conference on Supercomputing. – San Diego: IEEE Press, 1995.
31. Karypis G., Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs // Journal on Scientific computing. – 1998. – Vol. 20, № 1. – P. 359 – 392.
32. Khan M. S., Li K. F. Fast graph partitioning algorithms // Proc. of Int. IEEE conference “Communications, Computers, and Signal Processing”. – Victoria: IEEE Press, 1995. – P. 337 – 342.
33. Schloegel K., Karypis G., Kumar V. Graph partitioning for high-performance scientific simulations // Sourcebook of parallel computing. – San Franciso: Morgan Kaufmann Publish, 2003. – P. 491 – 541.
34. Ахо А. В., Хопкрофт В. Д., Ульман Дж. Д. Структуры данных и алгоритмы; пер. с англ. под ред. А. А. Минько. – Издательский дом “Вильямс”, 2001. – 384 с.
35. Кормен Т. Х. [и др]. Алгоритмы: построение и анализ, 2-е издание; пер. с англ. под ред. И. В. Красикова. – М.: Издательский дом “Вильямс”, 2005. – 1296 с.
36. Fiduccia C. M., Mattheyses R. M. A linear-time heuristic for improving network partitions // Proc. of conference “Design Automation”. – 1982. – P. 175 – 181.

Статья поступила в редакцию 08.04.2009.

Курносов Михаил Георгиевич
к.т.н. старший преподаватель кафедры вычислительных систем ГОУ ВПО “СибГУТИ”, (630102, Новосибирск, ул. Кирова, 86) тел. (383) 269-82-86, e-mail: mkurnosov@gmail.com.

Algorithms for mapping parallel programs onto hierarchical distributed computer systems

M. G. Kurnosov

In most cases modern distributed computer systems (computer clusters and MPP systems) have hierarchical organization and non-uniform communication channels between elementary machines (computer nodes, processors or processor cores). Execution time of parallel programs significantly depends on how they map to computer system (on what elementary machines parallel processes are assigned and what channels for inter-process communications are used). The general problem of mapping a parallel program into a distributed computer system is a well known NP-hard problem and several heuristics have been proposed to approximate its optimal solution. In this paper an algorithm for mapping parallel programs into hierarchical distributed computer systems based on task graph partitioning is proposed. The software tool for mapping MPI applications into multicore computer clusters is considered. The quality of this algorithm with the NAS Parallel Benchmarks is evaluated.
Keywords: parallel programs mapping, task allocation, task assignment, MPI, distributed computer systems.