УДК 519.8, 519.6

В.А. ИВАЩЕНКО, Р.Ю. ЛОПАТКИН, В.В. КУПРИЕНКО

СПОСОБ ВЫБОРА АЛГОРИТМА РАЗБИЕНИЯ ГРАФА ДЛЯ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ

Анотація. *Запропоновано спосіб визначення потенційно найбільш ефективного алгоритму розбиття заданого графа для розподілених обчислень, який спирається на результати аналізу статистичної залежності величини розрізу, що отримується (для того або іншого алгоритму розбиття графа) від метрик графа. Експерименти щодо використання запропонованого способу перед початком розрахунків демонструють його ефективність у підвищенні швидкодії розподіленої програми.*
Ключові слова: *прогнозування, оптимізація швидкодії, розбиття графа, метрики графів, розподілені обчислення.*

Аннотация. *Предложен способ определения потенциально наиболее эффективного алгоритма разбиения заданного графа для распределенных вычислений, который опирается на результаты анализа статистической зависимости величины получаемого разреза (для того или иного алгоритма разбиения графа) от метрик графа. Эксперименты по использованию предложенного способа перед началом расчетов демонстрируют его эффективность в повышении быстродействия распределенной программы.*
Ключевые слова: *прогнозирование, оптимизация производительности, разбиение графа, метрики графов, распределенные вычисления.*

Abstract. *A method of definition of the most effective partition algorithm of a specified graph for distributed calculation based on the analytical data of statistical dependence between the received value (for one or another graph partition algorithm) and graph metric is suggested. The experiments on the method mentioned above before the calculations demonstrate its efficiency in improving the performance of distributed applications.*
Keywords: *forecasting, performance optimizing, graph partition, graph metrics, distributed calculation.*

1. Введение

Постановку задачи поиска минимального разбиения (иногда употребляется термин «минимальный разрез») графа можно сформулировать следующим образом. Пусть дан неориентированный невзвешенный граф $G = (V, E)$, где V – множество его вершин, E – множество ребер этого графа. Необходимо разделить множество V на k непересекающихся подмножеств $\overline{V} = (V_1,...,V_k)$ $(V_i \cap V_j = \varnothing,\ \forall i, j = 1..k,\ i \neq j;\ V = V_1 \cup ... \cup V_k)$ таким образом, чтобы выполнялось условие

$$\begin{cases} \min(B_i) \geq B^*, \forall i = 1..k, \\ \sum_{i,j} cut(V_i, V_j) \to \min, i = 1..k-1,\ j = i+1..k, \end{cases} \quad (1)$$

где $B_i = |V_i| / |V|$;

$cut(V_i, V_j)$ – метрика, обозначающая количество ребер, соединяющих вершины из множества V_i с вершинами из множества V_j (далее будем использовать термин «величина разреза» для ее обозначения). Очевидно, что $0 \leq B^* \leq 1/k$. Как правило, ставится строгое ограничение $B^* = 1/k$, но допускается отклонение B^* на малую величину от максимально возможного: $B^* \in [1/k - \varepsilon; 1/k]$ (например, если количество вершин графа нечетное, то первое условие выполнить невозможно).

ISSN 1028-9763. Математичні машини і системи, 2011, № 4

Рассмотрим только частный случай задачи (1) при $k = 2$, т.е. так называемую задачу половинного разбиения графа (англ. Graph bipartition problem):

$$\begin{cases} |V_1| \approx |V_2|, \\ cut(V_1, V_2) \to \min. \end{cases}$$

Следует отметить, что решение более общей задачи $(k > 2)$ можно находить рекурсивно, используя половинное разбиение графа [1].

Задача (1) имеет важное практическое применение в области параллельных вычислений, где особое внимание уделяется скорости вычислений, хотя, например, при проектировании печатных плат [2] фактор скорости не является критичным.

К сожалению, единственный известный алгоритм, позволяющий находить точное решение рассматриваемой задачи (1), – это алгоритм решения путем перебора всех возможных разбиений графа, сложность которого растет как $O(k^n)$, где k – количество компонент, а n – размер компоненты [1, 3]. По этой причине разработано достаточно большое количество эвристических алгоритмов, решающих задачу разбиения графа с определенной точностью (качеством) за приемлемое время. Если говорить об абсолютной оценке точности, то это теоретическая оценка: $\varepsilon = c^* - c$, где c^* – точное значение минимального разреза графа, а c – величина разреза, полученного с помощью алгоритма. Но на практике, если необходимо сравнить качество двух алгоритмов, то обычно сравнивают метрики, которые характеризуют разбиения, получаемые с помощью каждого из этих алгоритмов [4]. Например, это может быть величина разреза $c(G) = \sum_{i,j} cut(V_i, V_j)$. Будем говорить, что один алгоритм более качественный, чем другой, если для одного и того же графа он позволяет найти разрез меньшей величины, чем другой алгоритм.

Естественно, что каждый алгоритм имеет свои характеристики, влияющие на качество получаемого разбиения (точность алгоритма) и его быстродействие [1]. Значения этих характеристик варьируются для каждого из алгоритмов применительно к определенным классам графов. Следовательно, выбранный алгоритм разбиения графа для распределенных (параллельных) вычислений имеет существенное влияние на общее время выполнения всей программы. Поэтому на практике возникает вопрос: выбор какого алгоритма обеспечит наилучшее конечное быстродействие программы? Как правило, выбор алгоритма происходит по каким-либо субъективным предпочтениям исследователя или подбирается эмпирическим путем, что не всегда приводит к наилучшему результату. Основная задача исследования, описанного в данной статье, заключается в поиске критерия выбора оптимального алгоритма разбиения графа. Такой критерий можно получить на основе сравнительного анализа математических моделей процессов выполнения параллельной программы при условии, что в каждом из них используется свой конкретный алгоритм разрезания графа для распараллеливания вычислений.

Для решения поставленной задачи нами разработан способ определения потенциально наиболее эффективного алгоритма разбиения для заданного графа. В нем используются результаты анализа статистической зависимости величины получаемого разреза (с помощью того или иного алгоритма) от определенных метрик графа. На основании этих данных имеется возможность прогнозирования ожидаемого разбиения графа, который в свою очередь можно использовать как параметр модели для прогнозирования быстродействия распределенной программы.

2. Обзор методов

Выделяют следующие основные классы методов прогнозирования быстродействия распределенной программы:

1. *Аналитико-статистические методы*. Это класс неточных методов, который используется скорее для грубой оценки времени выполнения программы (в виде подсказки).

2. *Прогнозирование на основе профилирования*. Эти методы предполагают сбор данных о производительности программы во время ее выполнения и прогнозирование ее быстродействия на основе этих данных.

3. *Методы имитационного моделирования*. Методы этой группы предполагают разработку имитационной модели программы и прогнозирование быстродействия путем «проигрывания» этой модели [5]. В этих методах может использоваться история статистических данных времени вычислений.

4. *LINPACK-тесты*. Данные методы можно применить как для определения производительности вычислительной техники, так и для прогнозирования времени выполнения параллельной программы.

Отметим, что точность методов прогноза растет в порядке их перечисления – от первой до четвертой группы, однако их скорость, наоборот, уменьшается. Методы, которые используют в прогнозировании, данные времени вычислений однозначно не подходят для решения рассматриваемой задачи из-за того, что выбор алгоритма необходимо сделать еще до начала вычислений. В этом случае вполне подходящим может быть аналитико-статистический метод, поскольку, во-первых, он позволяет делать прогноз быстродействия программы до ее выполнения, а во-вторых, при сравнении производительности двух алгоритмов можно получить более точный ответ, чем при прогнозировании времени работы каждого из них по отдельности. Это возможно за счет того, что в данном случае можно не включать в модель некоторые (не влияющие на преимущество алгоритма) характеристики вычислительной системы, что позволит также исключить их влияние на общую ошибку вычислений.

Таким образом, данный подход обеспечит возможность быстро определять (основываясь на метриках графа, которые можно довольно быстро посчитать, возможно, в параллельном потоке во время считывания графа), какой из алгоритмов для заданного графа обеспечит наилучшее быстродействие всей параллельной программы.

3. Основная часть

Согласно [6], время выполнения параллельной программы можно представить в виде функции от характеристик используемой вычислительной техники, среды исполнения и характеристик самой параллельной программы. Поскольку в рассматриваемой задаче вопрос состоит в том, чтобы выбрать лучший алгоритм разбиения графа (как составную часть программы) для исполнения на одной и той же машине, то необходимо выбирать тот алгоритм, для которого значение этой функции минимальное. Как будет показано дальше, влияние некоторых величин имеет один и тот же вклад в случае использования любого из алгоритмов и поэтому при сравнении может быть сокращено (исключено из модели), как не влияющей на преимущество любого из алгоритмов.

Время работы параллельной программы с учетом затрат времени на разбиение графа можно представить в виде следующей модели:

$$T_{pr} = T_0 + T_{main} = T_0 + \sum T_i \,, \qquad (2)$$

где T_{pr} – общее время работы программы;

T_0 – время, затраченное на разбиение графа (этап распараллеливания программы);

T_{main} – время работы основной части программы (собственно вычислений, для ускорения которых проводилось разбиение графа);

T_i – время вычислений на i-м шаге основной программы.

Время, затрачиваемое на выполнение одного шага параллельной программы, можно представить как [6]

$$T_i = T_{com(i)} + T_{calc(i)} + T_{wait(i)},$$

где $T_{com(i)}$ – суммарное время передачи и приема данных по каналам связи на i-м шаге;

$T_{calc(i)}$ – суммарное время, затраченное на пересчет всех вершин одним процессом;

$T_{wait(i)}$ – суммарное время ожидания процессами друг друга на i-м шаге.

Более подробно распишем время, затрачиваемое на осуществление одного шага программы, учитывая, что временем задержек можно пренебречь:

$$T_i = T_{com(i)} + T_{calc(i)} + T_{wait(i)} = v \cdot c / w + |V| / 2 \cdot x \cdot p, \quad (3)$$

где v – объем передаваемых данных (байт), w – скорость передачи данных по каналу связи (б/с), p – производительность (оп/с), x – количество операций, выполняемых при пересчете одной вершины графа. То есть, произведем некоторые упрощения: будем считать, что для пересчета любой из вершин необходимо выполнить одно и то же количество операций.

Рассмотрим два абстрактных алгоритма: A_1 с характеристиками (T_{10}, c_1) – время выполнения и величина получаемого с его помощью разреза соответственно, A_2 с характеристиками (T_{20}, c_2). Допустим, что

$$T_{10} < T_{20}, \text{ но } c_1 > c_2. \quad (4)$$

То есть, алгоритм A_2 дает более качественный разрез, чем A_1, но за большее время.

Рассмотрим случай, когда $T_i = const$. Тогда формула (2) принимает вид $T_{pr} = T_0 + N \cdot T_i$, где N – количество шагов основной части программы. $T_{pr1} = T_{10} + N \cdot T_{1i}$ – время работы параллельной программы при условии использования алгоритма A_1, а $T_{pr2} = T_{20} + N \cdot T_{2i}$ – время работы параллельной программы при условии использования алгоритма A_2.

Согласно (3), при выполнении условия (4) $T_{1i} > T_{2i}$, что делает возможным выполнение равенства $T_{10} + N \cdot T_{1i} = T_{20} + N \cdot T_{2i}$. Решая его относительно переменной N, получаем

$$N^* = \frac{T_{20} - T_{10}}{T_{1i} - T_{2i}} \quad (5)$$

– номер шага, начиная с которого программа, где использован более медленный, но более качественный алгоритм разрезания графа, начинает опережать по быстродействию программу, в которой использован более быстрый, но менее качественный алгоритм (рис. 1).

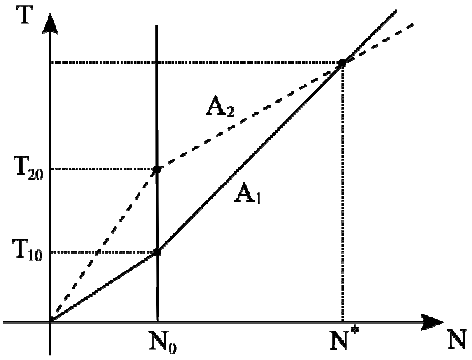

Рис. 1. Зависимость быстродействия параллельной программы от выбора алгоритма разбиения графа

На данном рисунке N_0 – нулевой шаг программы (время завершения алгоритма разбиения

графа). В соответствии с данной моделью, в зависимости от количества итераций программы, имеет смысл выбрать тот или иной алгоритм.

Подставляя (3) в (5), получаем

$$N^* = \frac{T_{20} - T_{10}}{(v \cdot c_1 / w + |V|/2 \cdot x \cdot p) - (v \cdot c_2 / w + |V|/2 \cdot x \cdot p)} = \frac{(T_{20} - T_{10}) \cdot w}{v(c_1 - c_2)}.$$

Согласно данной модели, N^* зависит от разницы времени выполнения алгоритмов разбиения графа, пропускной способности сети (канала коммуникации процессоров), объема передаваемых данных и разницы величин разрезов, получаемых с помощью этих алгоритмов. Время выполнения алгоритма зависит от производительности конкретной вычислительной машины и может быть записано как $T = o / p$, где o – оценка количества операций, совершаемых алгоритмом разбиения графа. Получаем

$$N^* = \frac{(o_2 - o_1) \cdot w}{p \cdot v(c_1 - c_2)}. \quad (6)$$

Чтобы иметь возможность прогнозировать N^*, необходимо получить оценки величин o_1, o_2, c_1, c_2, а также характеристики вычислительной техники – w, p. Для получения эмпирических зависимостей величины разреза (c) и количества операций (o) от метрических характеристик графа [7] необходимо сделать оценку математического ожидания количества операций $M[o]$ и получаемого разреза $M[c]$.

Чтобы найти эти оценки, было проведено по 60 экспериментов с использованием каждого из двух алгоритмов Levelized Nested Dissection (LND) [1] и Greedy Graph Growing Partitioning Algorithm (GGGP) [8]. (Такой выбор был не случайным, поскольку, как известно, второй из них имеет более высокую сложность по сравнению с первым, но качество его аппроксимации лучше). В экспериментах в качестве метрик использовались размер графа $sz = |V|$ и средняя степень вершины (связность) графа $con = \frac{1}{sz \cdot (sz-1)} \sum_{i,j=1,sz} G_{i,j}$ [7]. Поскольку остальные метрики графа, которые могут повлиять на результат, не рассматривались, то на экспериментальные данные был наложен ряд ограничений, сводящий к минимуму влияние других метрик:

1) равномерное распределение ребер графа;

2) равномерная связность: степень всех вершин графа должна быть примерно одинаковая.

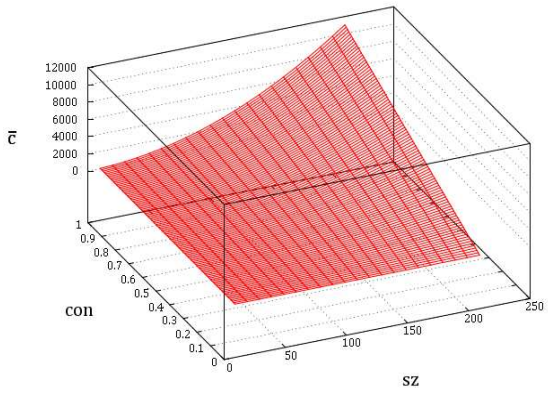

Рис. 2. Усредненная зависимость величины получаемого разреза (c) от связности (con) и размера графа (sz) для алгоритма LND

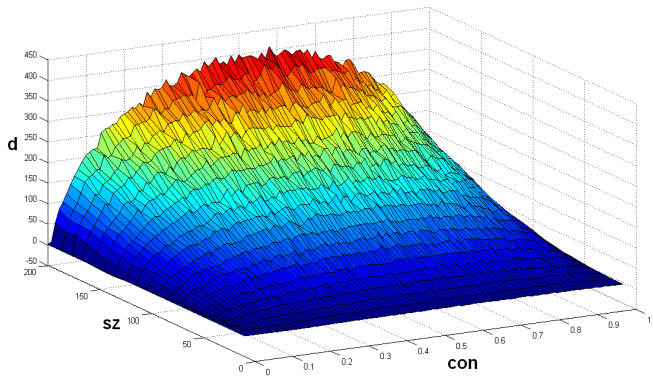

Рис. 3. Преимущество алгоритма GGGP над LND

Результаты экспериментов для каждого из алгоритмов – это табличные данные $c_i(con, sz)$. Область, в которой проводились эксперименты: $con \in \{0.1,0.2,...,1\} \bigcup sz \in \{i \in [10;210], i \bmod 10 = 0\}$. На рис. 2 на примере алгоритма LND показана усредненная зависимость величины разреза от связности и размера графа.

Согласно этому графику, величина разреза c растет при росте размера графа sz и его связности con.

На рис 3. показана зависимость $d(A_1, A_2) = \bar{c}(A_1, |V|, con(G)) - \bar{c}(A_2, |V|, con(G))$ – разности усредненных значений разрезов от связности графа и его размера, получаемых с помощью алгоритмов LND и GGGP.

Полученный результат можно объяснить тем, что если связность графа стремится к нулю $(con \to 0)$, то и величина разреза, получаемого при использовании каждого из алгоритмов, стремится к нулю. А разница нулей равна нулю. Аналогично, если связность графа стремится к единице $(con \to 1)$, то и величина нормированного разреза, получаемого при помощи каждого из алгоритмов, тоже стремится к единице, поскольку, как известно, любой разрез полного графа равен максимальному разрезу этого графа. А разница единиц равна нулю. Кроме того, имеет место рост разницы разрезов с ростом размера графа. Это можно объяснить тем, что соотношение разрезов может оставаться приблизительно одинаковым, но при этом будет иметь место рост разницы ненормированных значений разрезов. Таким образом, можно сделать вывод, что при максимально и минимально возможных связностях все алгоритмы разбиения графов дают практически одинаковые результаты, а также, чем больше размер графа, тем больший выигрыш в абсолютной величине разреза получаем с помощью более качественного алгоритма.

После этого была осуществлена аппроксимация усредненных данных поверхностями, представляющими собой произведение полиномов. Выбирается именно суперпозиция полиномов, поскольку по своей природе величина разреза зависит полиномиально как от размера графа, так и от его связности. Для обоих рассмотренных алгоритмов аппроксимация полиномом четвертого порядка обеспечивает достаточную точность приближения. Например, c_1 и c_2 можно выразить в виде следующих полиномиальных оценок:

$$\tilde{c}_1(con, sz) = 1{,}855 - 3{,}986 \cdot con + 2{,}649 \cdot con^2 - 0{,}355 \cdot sz + 0{,}115 \cdot con \cdot sz + 0{,}236 \cdot sz \cdot con^2 + \\ + 3{,}5 \cdot 10^{-5} \cdot sz^2 + 0{,}25 \cdot sz^2 \cdot con - 2{,}7 \cdot 10^{-5} \cdot sz^2 \cdot con^2,$$

$$\tilde{c}_2(con, sz) = 3.947 + 36.230 \cdot con - 36.351 \cdot con^2 - 0.260 \cdot sz - 3{,}305 \cdot con \cdot sz + 3{,}311 \cdot sz \cdot con^2 - \\ - 0{,}002 \cdot sz^2 + 0{,}233 \cdot sz^2 \cdot con + 0{,}018 \cdot sz^2 \cdot con^2.$$

Значение каждой из них – прогноз ожидаемого разреза для графа размером sz вершин и связностью con.

Аналогично были найдены оценки необходимого количества операций для каждого рассмотренного алгоритма (рис. 4):

$\tilde{o}_1(n) = 0{,}946 \cdot n^3 - 0{,}308 \cdot n^2 - 20{,}577 \cdot n + 235{,}36$ – оценка необходимого количества операций для GGGP;

$\tilde{o}_2(n) = 2{,}255 \cdot n^2 + 12{,}868 \cdot n - 12{,}969$ – оценка необходимого количества операций для LND.

Итак, в качестве параметров модели используются:

1) метрики входного графа G;

2) предполагаемое количество шагов основной программы – N.

При соблюдении описанных выше ограничений можно воспользоваться следующими правилами для выбора наиболее оптимального алгоритма разбиения графа:

– если необходимо сделать выбор в пользу одного из двух алгоритмов, то можно просто воспользоваться решением уравнения (6);

– если количество алгоритмов больше двух, то необходимо выбирать тот из них, для которого значение $T_0 + N \cdot T_i = T_0 + N(v \cdot c / w + |V|/2 \cdot x \cdot p)$ минимальное. Причем в таком случае часть этого выражения $|V|/2 \cdot x \cdot p$ можно приравнять нулю, поскольку она вносит одинаковый вклад для любого из алгоритмов.

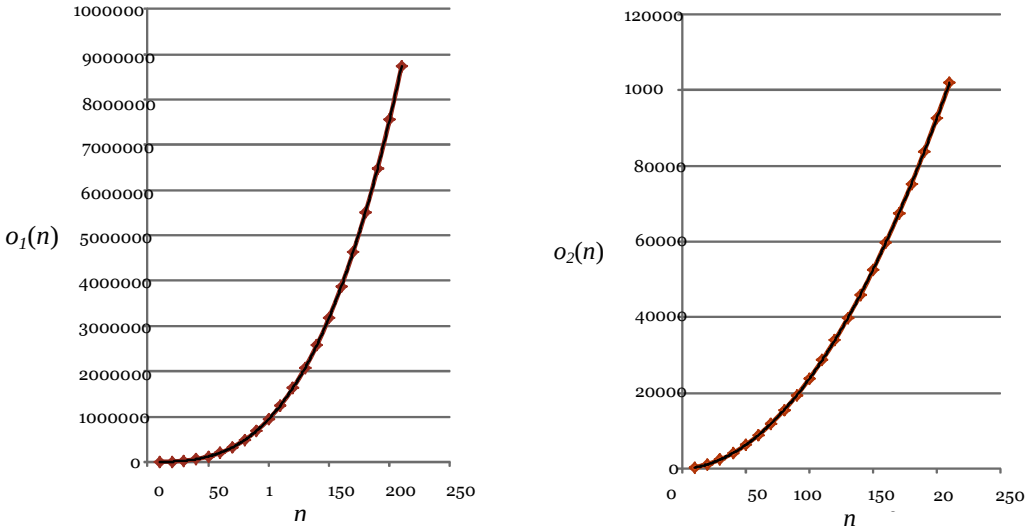

Рис. 4. Зависимость количества операций, выполняемых алгоритмом разбиения графа, от размера графа

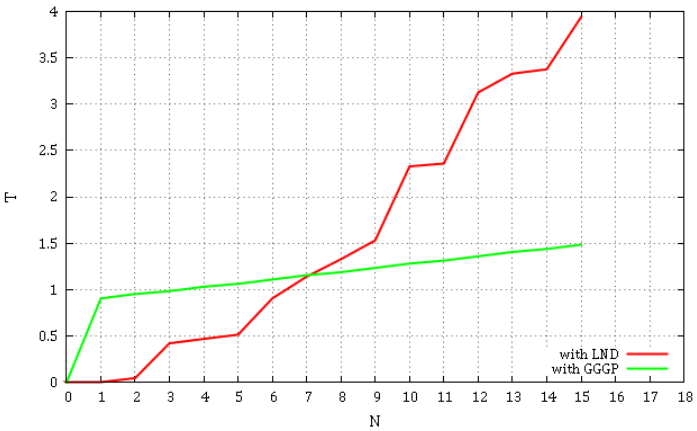

Рис. 5. Сравнение быстродействия параллельной программы при использовании различных алгоритмов разбиения графа

4. Экспериментальная проверка

Была проведена проверка предложенного способа выбора алгоритма на примере элементарной программы, реализованной с использованием MPI-технологии. Сначала входной граф был разбит с помощью алгоритма LND и измерено время выполнения параллельной программы, потом точно такие же действия были выполнены с использованием алгоритма GGGP. Результат одного из экспериментов с этой программой показан на рис. 5.

Спрогнозированный момент времени, когда программа, использующая LND, опередит программу, использующую GGGP, равен 7,993. После серии экспериментов полученное экспериментальным путем и усредненное $\overline{N}^* = 7{,}429$, что достаточно хорошо согласуется с прогнозируемым результатом.

5. Выводы

Прогнозирование влияния выбранного алгоритма разбиения графа на время выполнения параллельной программы является актуальной и важной практической задачей. Расчеты показали, что основное влияние на общее время выполнения параллельной программы

имеют две основные характеристики алгоритма разбиения графа: время его выполнения и «качество» его работы. А для конкретного графа и алгоритма разбиения графа не очевидна ни оценка этих характеристик, ни их влияние на общее время выполнения параллельной программы.

По результатам исследований предложен способ выбора алгоритма разбиения графа для распараллеливания вычислений в зависимости от метрических характеристик графа. Основным достоинством и перспективностью применения такого подхода есть возможность прогнозирования эффективности выполнения параллельных программ без необходимости их предварительного выполнения. Это, в свою очередь, позволяет ускорять время работы этих программ за счет выбора оптимального алгоритма для разбиения задачи на подзадачи.

На данный момент предложенный способ применим для узкого класса графов, у которых ограничением является равномерная связность (примерно одинаковая степень каждой из вершин графа). В дальнейшем нами планируется расширить применимость предложенного способа путем введения дополнительных метрик, а также провести более глубокие исследования точности предложенного подхода.

СПИСОК ЛИТЕРАТУРЫ

1. Schloegel K. Graph partitioning for high-performance scientific simulations / K. Schloegel, G. Karypis, V. Kumar // Sourcebook of parallel computing. – Morgan Kaufmann Publishers Inc., 2003. – P. 491 – 541.
2. Caldwell A.E. Design and Implementation of the Fiduccia-Mattheyses Heuristic for VLSI Netlist Partitioning / A.E. Caldwell, A.B. Kahng, I.L. Markov// Lecture Notes in Computer Science. – Springer, 1999. – P. 182 – 198.
3. Srikant Y.N. The compiler design handbook: optimizations and machine code generation / Y.N. Srikant, P. Shankar. – CRC Press, 2008. – 784 p.
4. Hendrickson B. Graph partitioning models for parallel computing / B. Hendrickson, T.G. Kolda // Parallel Computing. – 2000. – N 26 (12). – P. 1519 – 1534.
5. Simulaton-based performance prediction for Large Parallel Machines / G. Zheng, T. Wilmarth, P. Jagadishprasad [et al.] // International Journal of Parallel Programming. Special issue: The next generation software program. – 2005. – Vol. 33, Is. 2. – P. 183 – 207.
6. Performance Prediction Methods /A. Szymańska-Kwiecień, J. Kwiatkowski, M. Pawlik [et al.] // Proc. of the International Multiconference on Computer Science and Information Technology. – Wisła, 2006. – P. 363 – 370.
7. Субъективные метрики оценки онтологий / Т.А. Гаврилова, В.А. Горовой, Е.С. Болотникова [и др.] // Материалы Всероссийской конф. с междунар. участием «Знания-Онтологии-Теории» (ЗОНТ-2009). – Новосибирск: Институт математики СО РАН, 2009. – Т. 1. – С. 178 – 186.
8. Karypis G. A fast and high quality multilevel scheme for partitioning irregular graphs / G. Karypis, V. Kumar // SIAM Journal on Scientific Computing. – 1999. – Vol. 20, N 1. – P. 359 – 392.

Стаття надійшла до редакції 01.06.2011