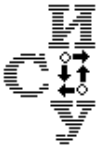

ЛИТЕРАТУРА

1. *Абрамов. О.В.* Параметрический синтез стохастических систем с учетом требований надежности. – М.: Наука, 1992.
2. Грид-технологии и вычисления в распределенной среде / Афанасьев А.П., Волошинов В.В., Посыпкин М.А., Сухорослов О.В., Хуторной Д.А. // Избранные доклады III Международной конференции «Параллельные вычисления и задачи управления» PACO'2006 / Ин-т пробл. упр. им. В.А. Трапезникова РАН, 2006. – С.5–16.
3. Параллельные алгоритмы построения области работоспособности / Абрамов О.В., Диго Г.Б., Диго Н.Б. Катуева Я.В. // Информатика и системы управления. – 2004. – № 2(8). – С.121–133.
4. *Абрамов О.В., Катуева Я.В., Назаров Д.А.* Оптимальный параметрический синтез по критерию запаса работоспособности. // Проблемы управления. – 2007. – № 6. – С.64-69.
5. *Катуева Я.В., Назаров Д.А.* Аппроксимация и построение областей работоспособности в задаче параметрического синтеза // Тр. междунар. симпозиума "Надежность и качество" / ПГУ. – Пенза, 2005. – С.130-134.
6. *Катуева Я.В., Назаров Д.А.* Алгоритмы анализа области работоспособности, заданной в матричной форме // Информатика и системы управления. – 2005. – №2(10). – С.118–128.
7. *Васильев Б.В., Козлов Б.А., Ткаченко Л.Г.* Надежность и эффективность радиоэлектронных устройств. – М.: Советское радио, 1964.

Статья представлена к публикации членом редколлегии О.В. Абрамовым.

УДК 681.3.062

© 2008 г. **И.А. Трещев**
(Комсомольский-на-Амуре государственный технический университет)

ПОСТРОЕНИЕ МНОГОПОТОЧНЫХ ПРИЛОЖЕНИЙ ДЛЯ РАСПАРАЛЛЕЛИВАНИЯ АЛГОРИТМОВ ПЕРЕБОРА

Предлагается модификация общей схемы распараллеливания алгоритмов для решения задач, допускающих решение методом перебора с возвратом. Модифицированная схема применяется для построения многопоточных приложений на компьютерах с SMP-архитектурой. Приводятся результаты тестирования этого метода на ряде классических задач перебора.

Введение

Метод перебора с возвратом предназначен для поиска конечных последовательностей элементов $(x_1, x_2,\ldots, x_n)$, связанных заданными отношениями. Этот метод формально был описан в пятидесятых годах прошлого столетия. Мы будем рассматривать подкласс класса задач, которые решаются методом перебора с возвратом. Опишем задачи этого подкласса. Более подробно они рассматриваются в учебном пособии [1].

Пусть заданы конечные множества A_1, A_2, …, A_n. Для произвольного целого $n > 0$ под *n-местным предикатом* мы будем понимать функцию:

$$P : A_1 \times A_2 \times \mathbf{K} \times A_n \to \{0,1\}, \quad (1)$$

принимающую значения 1 (истина) или 0 (ложь). *Предикатом* будем называть произвольную функцию:

$$Q : \bigcup_{n \geq 1} A_1 \times A_2 \times \cdots \times A_n \to \{0,1\}. \quad (2)$$

Постановка задачи

Даны конечные множества $A_1, A_2, \ldots, A_n$ и для каждого целого i задан i-местный предикат $P_i(x_1, x_2, \ldots, x_i)$. Предполагается, что для всех $i>1$ справедливы импликации

$$P_i(x_1, x_2, \mathbf{K}, x_i) = 1 \Rightarrow P_{i-1}(x_1, x_2, \mathbf{K}, x_{i-1}) = 1. \quad (3)$$

Требуется построить последовательности $x = (x_1, x_2, \ldots, x_n)$, где $x_i \in A_i$, при $1 \leq i \leq n$, для которых верны соотношения

$$P_n(x_1, x_2, \mathbf{K}, x_n) = 1, \quad (4)$$

$$Q(x_1, x_2, \mathbf{K}, x_n) = 1. \quad (5)$$

Например, для задачи Гаусса о ферзях [2] предикат (1) будут определять при $i<8$ расположения на первых i вертикалях, не бьющих друг друга ферзей. А предикат (2) принимает значение 1 тогда и только тогда, когда $n = 8$ (см. [1]).

Алгоритм перебора с возвратом позволяет перебирать последовательности, удовлетворяющие условию (4), и выбирать из них те, которые удовлетворяют условию (5). Идея алгоритма заключается в том, что за последней полученной последовательностью $(x_1, x_2, \ldots, x_{k-1})$ генерируется расширяющаяся ее последовательность $(x_1, x_2, \ldots, x_{k-1}, x_k)$ в соответствии с условием (3), а если таких расширений больше нет, то рассматривается следующий элемент $x_{k-1} \in A_{k-1}$. Если же такие элементы $x_{k-1} \in A_{k-1}$ исчерпаны, то производится возврат к последовательности $(x_1, x_2, \ldots, x_{k-2})$.

Замечание 1. Метод перебора с возвратом применяется также для поиска некоторой последовательности, удовлетворяющей соотношениям (4) и (5). В этом случае перебор заканчивается, если такая последовательность найдена.

Замечание 2. Согласно [3 – 5] метод перебора с возвратом (*backtrack*) годится для решения более широкого класса задач. Он называется также методом обхода в глубину (deep search method), или методом ветвей и границ (branch and bound method).

Сведение задачи к раскраске вершин дерева. Обозначим через

$$R_i \subseteq A_1 \times A_2 \times \mathbf{K} \times A_i, \quad (6)$$

подмножество последовательностей $(x_1, x_2, \ldots, x_i)$, удовлетворяющих условию:

$$P_i(x_1, x_2, \mathbf{K}, x_i) = 1. \quad (7)$$

Рассмотрим дерево, вершинами которого являются конечные последовательности, удовлетворяющие условию (6), а корнем служит пустая последовательность. Сыновьями узла $(x_1, x_2, \ldots, x_{k-1})$ будут вершины $(x_1, x_2, \ldots, x_{k-1}, x_k) \in R_k$.

Это дерево является подграфом дерева, у которого на уровне $k = 0$ находится корень, и из каждой вершины уровня $(k – 1)$ выходят $|A_k|$ ребер, имеющих метки $a \in A_k$.

Схемой программы или подпрограммы мы будем называть текст, состоящий из операторов языка C++ и выделенных курсивом комментариев, состоящих из русских слов и математических формул. Схема обычно нуждается в детализации, состоящей в замене комментариев операторами языка C++.

Рассмотрим задачу перебора последовательностей $(x_1, x_2,\ldots, x_n)$, удовлетворяющих условию (4). Последовательности $(x_1, x_2,\ldots, x_k)$ и $(x_1, x_2,\ldots, x_{k+1})$ соединяются ребром. Легко видеть (см., например, – [1]), что задача перебора сводится к задаче раскраски вершин построенного дерева. Следовательно, решения можно получить с помощью рекурсивной функции раскраски вершин графа, имеющей схему:

```
void rec(int k)
{
   for(y∈Ak)
   {
      if(Pk(x1,x2,…,xk-1,y))
      {
            xk=y;
            if(Q(x1,x2,…,xk-1,xk)) вывести решения x;
            rec(k+1);
      }
   }
}
```

Запись $y \in A_k$ означает, что y пробегает все элементы множества A_k.

Таким образом, с помощью данного алгоритма посещаются вершины дерева, представленного на рис. 1, методом обхода в глубину (сверху вниз и слева направо).

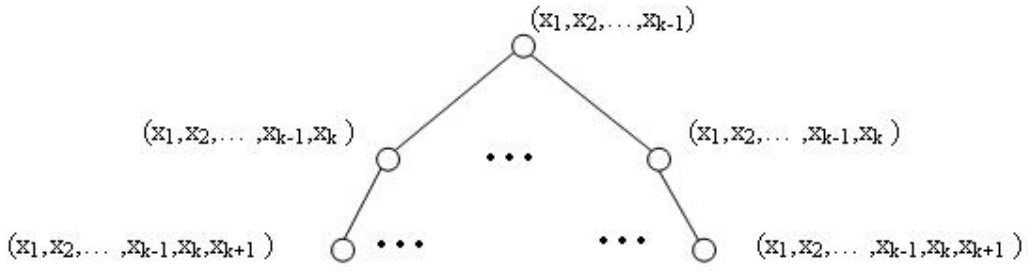

Рис. 1. Поддерево дерева решений.

Распараллеливание рекурсивных подпрограмм

В современных операционных системах возможность потока порождать другие потоки позволяет строить многопоточные программы из рекурсивно вызываемых подпрограмм. Вместо рекурсивного вызова функций для построения таких программ подставляется создание потоков, соответствующих этим функциям. Аргументы функций передаются через указатель на параметры функции [6, 7]. Для рассмотренного алгоритма перебора с возвратом можно предложить следующую схему распараллеливания:

```
struct arg
{
      int k;            //номер яруса в дереве решений
      int *X;           //последовательность предыдущего яруса
}
DWORD WINAPI rec(void *p)
{
   int *x = (arg *)p -> X;
   int k = (arg *)p -> k
   for(y ÎA_k)
   {
      if(P_k(x_1, x_2, …, x_k-1, y))
      {
            x_k=y;
            if(Q(x_1, x_2, …, x_k-1, x_k)) вывести решения x;
            arg *r1 = new arg;
            HANDLE H;
            r1->k = k+1;
            r1->X = new  int[k + 1];
            for (int i=0;i<=k;i++)
            {
                  r1->X[i]= x[i];
            }
            H = CreateThread(0,0,rec, (void *)r1,0,0);
            WaitForSingleObject(H,INFINITE);
            delete [](r1->X);
            delete r1;
      }
   }
}
```

Более глубокий анализ показывает, что выигрыш в быстродействии при использовании такой схемы отсутствует. Это связано с тем, что при запуске потока мы ожидаем его завершения, не выполняя никаких действий.

Отсюда вытекает, что параллельная реализация метода обхода в глубину не дает выигрыша во времени. Тем не менее, как будет показано в следующем параграфе, перебор последовательностей можно ускорить.

Параллельная реализация с помощью обхода в ширину

Рассмотрим метод распараллеливания, описанный в работах В.А. Беляева и Н.Е. Тимошевской [8, 9]. В его основе лежит метод обхода дерева в ширину. Для реализации этого метода воспользуемся следующей схемой:

```
struct arg
{
      int k;            //номер яруса в дереве решений
      int *X;           //последовательность предыдущего яруса
}

DWORD WINAPI rec(void *p)
{
   int *x = (arg *)p -> X;
   int k = (arg *)p -> k;
   N = | A_k |;          //количество элементов в k-ом множестве
   int Cur = 0;          //количество запущенных потоков
   arg *M[N];
   HANDLE *H[N];
   for(y ÎA_k)
   {
      if(P_k(x_1, x_2, …, x_k-1, y))
      {
            x[k] = y;
            if(Q(x_1, x_2, …, x_k-1, x_k)) вывести решения x;
            *M[Cur] = new arg;
```

```
            *H[Cur] = new HANDLE;
            M[Cur]->k = k+1;
            M[Cur]->X = new int[k+1];
            for (int i=0;i<k;i++)
            {
                  M[Cur]->X[i]= x[i];
            }
            *H[Cur] = CreateThread(0,0,rec, (void *)M[Cur],0,0);
            Cur++;
      }
   }
   for(i=0; i< Cur; i++)
   {
      WaitForSingleObject(*H[i],INFINITE);

      Delete [](M[i]->X);
      Delete M[i];
      Delete H[i];
   }
}
```

Замечание 3. На k-ом ярусе дерева решений может быть запущено $|A_k|$ потоков, где $|A_k|$ – число элементов соответствующего множества. Поэтому эффективная реализация требует достаточного числа процессоров.

Параллельная реализация перебора возрастающих последовательностей для заданного количества процессоров

В настоящее время все более популярными становятся системы с симметрично адресуемой памятью (поддержка технологии Hyper-Threading, многоядерные микропроцессоры). Для таких систем использование предложенной выше схемы реализации перебора с возвратом как поиска в ширину не совсем оправдано, поскольку «накладные расходы» по времени на создание большого количества потоков могут превзойти выигрыш от использования нескольких процессоров.

Мы предлагаем следующую модификацию этого метода. Наше решение заключается в том, чтобы подпрограмма перебора сама определяла, следует ли ей запуститься как потоку, если количество незанятых процессоров отлично от нуля, или как последовательной рекурсивной подпрограммой. Для доступа всех потоков к общей переменной, в которой будет храниться текущее количество занятых процессоров, следует использовать объекты синхронизации [6] (мьютексы, семафоры и события).

В зависимости от строения дерева решений для каждой конкретной задачи оптимальное количество потоков, необходимых для максимальной загрузки вычислительных узлов при использовании вышеописанной схемы, может быть различным. Рассмотрим, каким образом работа распределяется по потокам. После порождения нового потока начинается анализ новой ветви дерева решений, причем возможна ситуация, представленная на рис. 2.

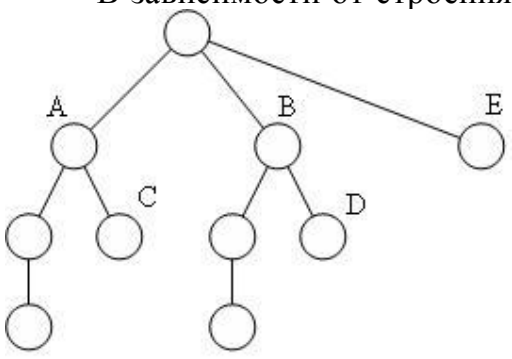

Рис. 2. Пример дерева решений.

Пусть у нас имеется 3 процессора, и в первый момент времени параллельно запусти-

лись расчеты в узлах A и B, а затем параллельно был запущен расчет в узле C; таким образом, все три процессора не простаивают. Теперь пусть потоки, выполняющие работу в узлах A и B, продолжают выполнение, а в узле C расчет заканчивается. В узлах D и E, в соответствии с алгоритмом, новые потоки для выполнения расчетов созданы не будут, хотя один процессор освободился, поскольку родительский процесс A ожидает окончания выполнения расчетов сначала для самого левого поддерева и, таким образом, один процессор простаивает, но если дерево решений будет сбалансировано и для каждого узла высота любого поддерева будет не больше высоты поддерева расположенного правее, то нагрузка распределится по потокам оптимально. От полученного эффекта простаивания процессоров можно избавиться и для произвольного дерева решений. Если родительский узел будет опрашивать каждый из своих дочерних узлов с некоторыми задержками, и в том случае, если произвольный дочерний узел завершил свою работу и при его запуске происходило порождение потока, информировать другие потоки, что освободился вычислительный узел и можно вновь порождать потоки, то следует отметить, что это неминуемо приведет к увеличению времени затрачиваемого на синхронизацию работы потоков.

Условия эксперимента и результаты тестирования многопоточных приложений [10] приведены ниже.

Условия проведения эксперимента и результаты тестирования

Тестирование разработанных многопоточных приложений производилось на HP Workstation xw8200 под управлением Microsoft Windows XP, оснащенных двумя микропроцессорами Intel Pentium 4 Xeon 3,6Ghz (1Mb L1 cache), с поддержкой технологии Hyper Threading, объем оперативной памяти 2Gb.

Нами были разработаны и протестированы многопоточные приложения для решения следующих классических задач, предполагающих использование алгоритма перебора с возвратом [11]: перебор возрастающих последовательностей, состоящих из 10 элементов – таких, что каждый элемент последовательности не превосходит 30; задача Гаусса о ферзях – рассматривается поле со стороной в 15 клеток; перебор разложений числа в сумму – рассматривается число 30 и слагаемые – 1, 2 , 3, 5, 7, 9, 10, 15; перебор гамильтоновых циклов в графе – рассматривается связный граф без петель, содержащий 12 вершин; генерация разбиений заданного множества, состоящего из 15 элементов.

Более подробное описание рассматриваемых задач можно найти в [1 – 3].

Результаты тестирования многопоточных приложений для описанных выше задач представлены на рис. 3 – 7 (время указано в миллисекундах). Тестирование проводилось путем пятикратного замера времени выполнения многопоточного приложения для каждого из возможных значений количества запускаемых потоков и вычисления среднего значения по формуле

$$T_i = \frac{1}{5}\sum_{k=1}^{5} w_k, \qquad (8)$$

где T_i – время выполнения многопоточного приложения, запускающего i потоков; w_k – k-е измерение времени выполнения многопоточного приложения, запускаю-

щего i потоков; i – количество потоков, $1 \le i \le 20$. Отметим, что в формуле (8) не учитывается время, затрачиваемое на создание и уничтожение потоков.

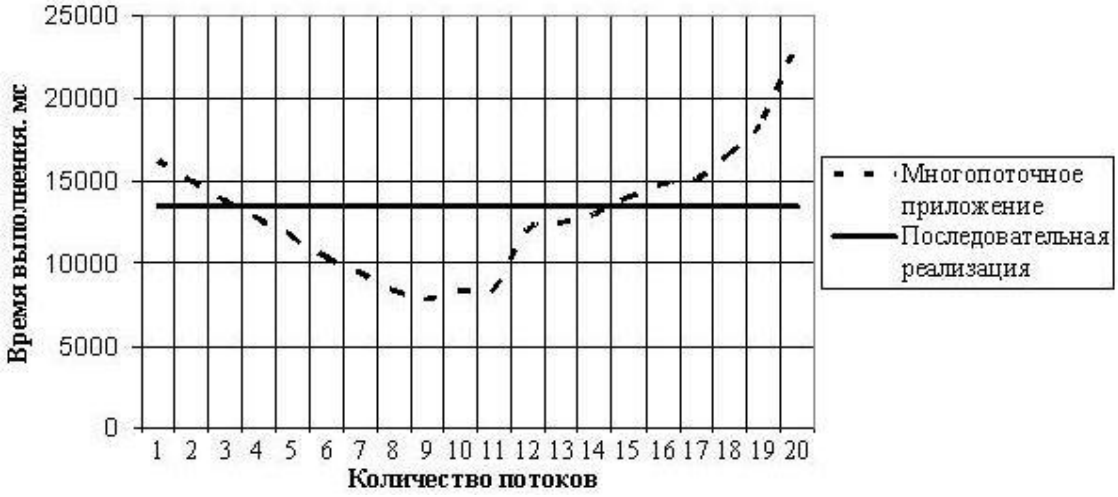

Рис. 3. График зависимости времени выполнения многопоточного приложения для задачи перебора гамильтоновых циклов в графе и количества потоков.

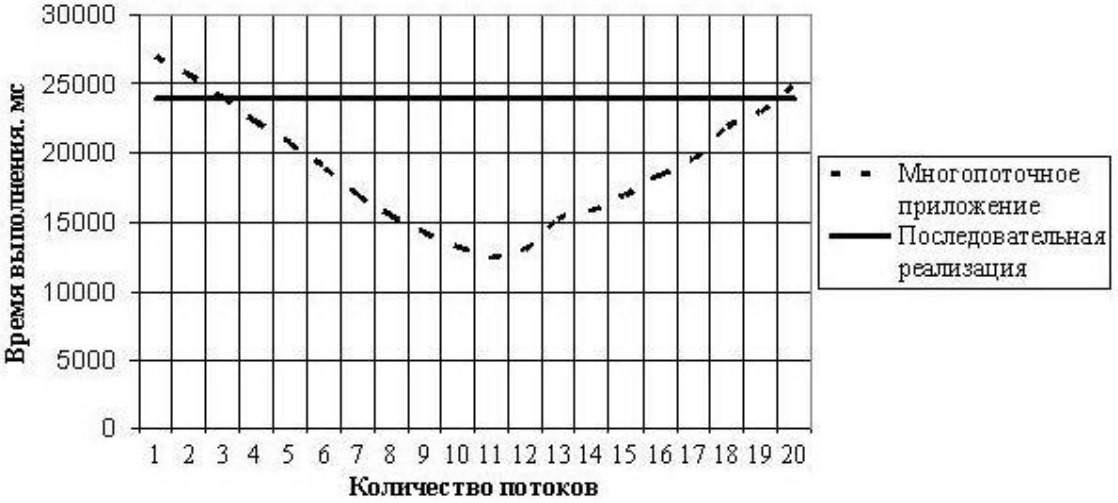

Рис. 4. График зависимости времени выполнения многопоточного приложения для задачи Гаусса о ферзях и количества потоков.

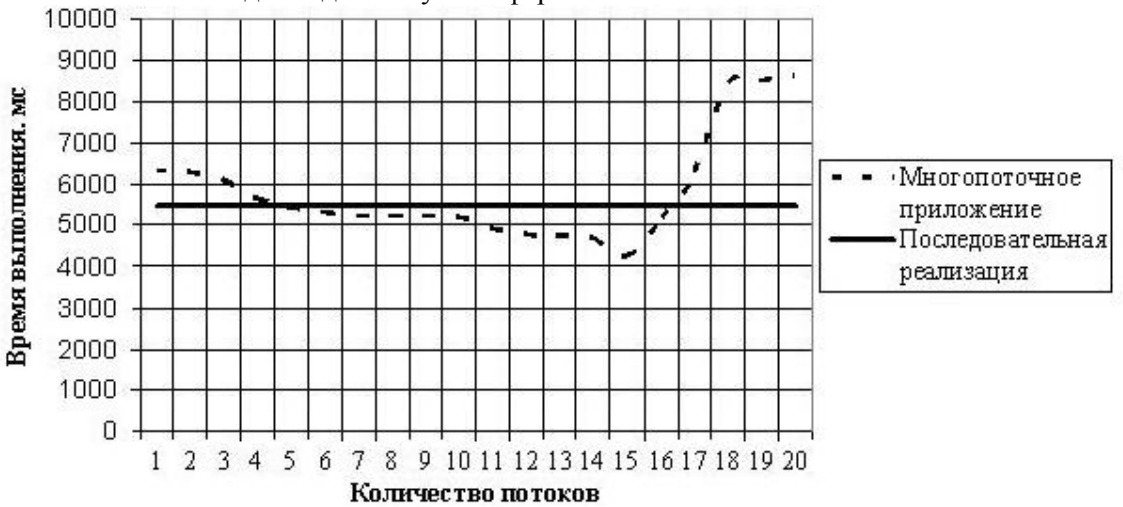

Рис. 5. График зависимости времени выполнения многопоточного приложения для задачи перебора разложений числа в сумму и количества потоков.

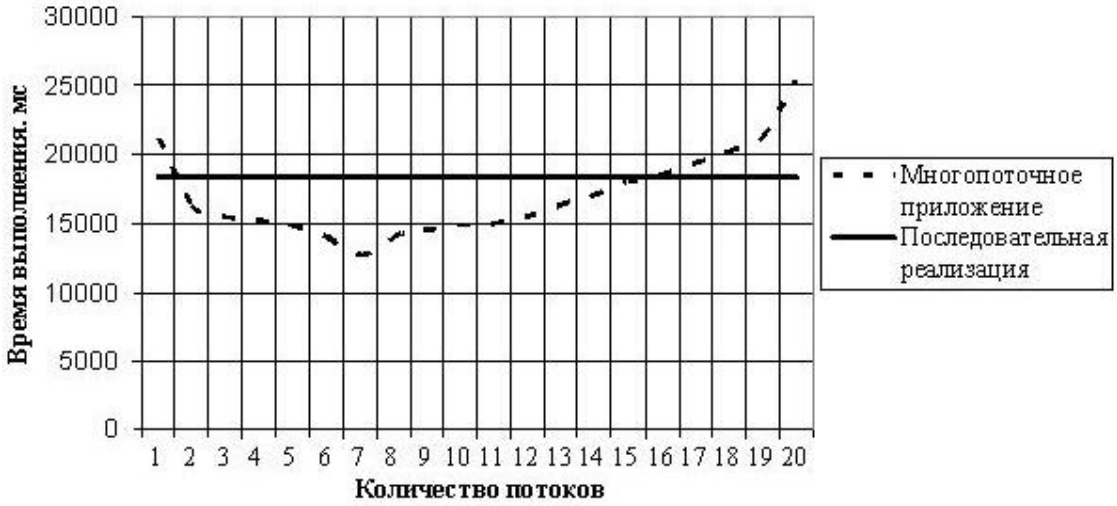

Рис. 6. График зависимости времени выполнения многопоточного приложения для задачи перебора возрастающих отображений и количества потоков.

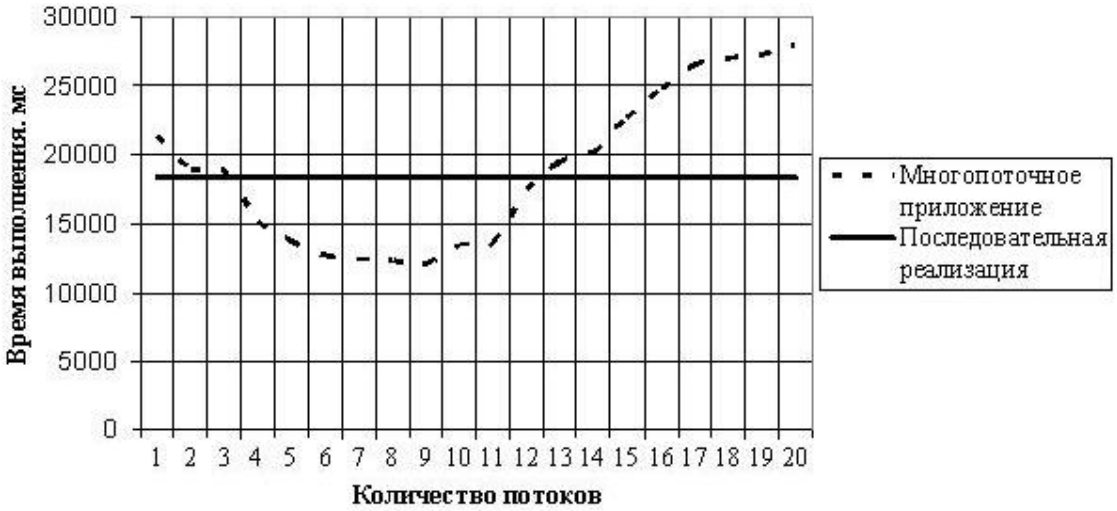

Рис. 7. График зависимости времени выполнения многопоточного приложения для задачи генерации разбиений заданного множества и количества потоков.

Заключение

Результаты, представленные на рис. 3 – 7, показывают, что оптимальное число потоков не всегда равно числу процессоров. Для задачи перебора возрастающих последовательностей, состоящих из 10 элементов – таких, что каждый элемент последовательности не превосходит 30, оно равно 7, для задачи Гаусса о ферзях для шахматного поля со стороной 15 клеток – 11, для задачи генерации разбиений заданного множества, состоящего из 15 элементов, – 9, для перебора разложений числа в сумму, рассматривается число 30 и слагаемые – 1, 2, 3, 5, 7, 9, 10, 15 – 15, для задачи перебора гамильтоновых циклов в графе рассматривается связный граф без петель, содержащий 12 вершин – 9. Также следует отметить, что на создание и синхронизацию работы потоков тратится вполне существенное для данных задач время, но при достаточном увеличении размера задачи, когда время работы потоков станет несравнимо больше времени, затрачиваемого на «накладные расходы», им можно будет пренебречь.

Ускорение (а значит и эффективность), полученное от использования параллельного алгоритма на многопроцессорной системе, зависит от исходных данных [12]. Для некоторых задач, как показывают графики, значение ускорения пропорционально числу используемых процессоров, но может и не превысить единицы: например, при поиске разложений числа в сумму. Пусть искомое решение представлено самым левым путем в дереве решений, тогда параллельный алгоритм, если не учитывать «накладных расходов», осуществит ровно столько же действий, что и последовательный. Противоположная ситуация возникает, если искомое решение представлено самым правым путем.

Таким образом, предложенная модификация метода перебора с возвратом легко масштабируема на произвольное число процессоров и позволяет получать ускорение вычислений для задач перебора, довольно близкое к теоретически достижимому.

ЛИТЕРАТУРА

1. *Хусаинов А.А., Михайлова Н.Н.* Структуры и алгоритмы обработки данных. Ч.2: Учеб. пособие. – Комсомольск-на-Амуре: Изд-во КнАГТУ, 2003.
2. *Вирт Н.* Алгоритмы и структуры данных. – М.: Мир, 1989.
3. *Нивергельт Ю., Фаррар Дж., Рейнгольд Э.* Машинный подход к решению математических задач. – М.: Мир, 1977.
4. *Ranadae A.* Optimal speedup for backtrack search on butterfly network // Mathematical Systems Theory. – 1994. – P.85-101.
5. *Rao V.N., Kumar V.* On the efficiency of parallel backtracking // IEEE Transactions on Parallel and Disributed Systems. – 1994. – Vol. 4, N 4.– P.427-437.
6. *Husainov A.A.* The study of distributed computing algorithms by multithread applications. – Preprint, 2004. – 17p. http://arxiv.org/abs//cs.DC/0404015.
7. *Хусаинов А.А., Михайлова Н.Н.* Архитектура вычислительных систем. Учеб. пособие. – Комсомольск-на-Амуре: Изд-во КнАГТУ, 2004.
8. *Беляев В.А., Тимошевская Н.Е.* Распараллеливание обхода дерева поиска для решения задачи о рюкзаке на кластерной системе // Высокопроизводительные параллельные вычисления на кластерных системах. Материалы международного научно-практического семинара / под ред. проф. Р.Г. Стронгина. – Нижний Новгород: Изд-во НГУ, 2001. – С.16-20.
9. *Тимошевская Н.Е.* Разработка параллельных алгоритмов обхода дерева // Высокопроизводительные параллельные вычисления на кластерных системах. Материалы третьего международного научно-практического семинара / под ред. проф. Р.Г. Стронгина. – Нижний Новгород: НГУ. – 2003. – С.198-206.
10. *Трещев И.А.* Программное обеспечение для перебора последовательностей на компьютерах с SMP архитектурой // Тезисы докладов XXXI Дальневосточной шк.-семинара. им. акад. Е.В. Золотова. – Владивосток: Дальнаука, 2006.
11. *Трещев И.А.* Свидетельство об официальной регистрации программы для ЭВМ №2006613475. Перебор последовательностей как раскраска вершин графа при обходе в ширину с использованием многопотоковых приложений на компьютерах с SMP архитектурой 6.10.2006
12. *Трещев И.А., Хусаинов А.А.* Метод перебора последовательностей как раскраска вершин графа при обходе в ширину на компьютерах с SMP архитектурой // Научно-техническое творчество аспирантов и студентов: материалы 36-й научно-технической конференции аспирантов и студентов. – Комсомольск-на-Амуре: КнАГТУ. – 2006. – Ч.1. – С.70-71.

Статья представлена к публикации членом редколлегии А.И. Олейниковым.