М.В. Якобовский
Е.Ю. Кулькова

РЕШЕНИЕ ЗАДАЧ НА МНОГОПРОЦЕССОРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ С РАЗДЕЛЯЕМОЙ ПАМЯТЬЮ

Москва
2004

Введение

На примере задачи вычисления определенного интеграла от аналитически заданной функции рассматриваются алгоритмы, минимизирующие время решения задачи на многопроцессорной вычислительной системе. Их использование обеспечивает снижение времени решения задачи относительно «наилучшего» доступного последовательного алгоритма. Определение эффективности параллельного алгоритма относительно собственной же последовательной версии, как правило, не корректно. Обладающий высокой эффективностью параллельный алгоритм может по времени выполнения проигрывать лучшему из доступных последовательных алгоритмов решения той же задачи. Далее рассматриваются алгоритмы, минимизирующие именно время решения задачи, оперирующей относительно небольшим объемом данных. Применяемые подходы могут быть полезны при решении широкого круга проблем.

Интегрирование одномерной функции на многопроцессорной системе с общей памятью

В качестве модельной задачи рассматривается проблема вычисления с точностью ε значение определенного интеграла (1):

$$J(A,B)=\int_{A}^{B} f(x)dx, \qquad (1)$$

Параллельные алгоритмы

Относительно несложно разработать параллельные алгоритмы решения обсуждаемой задачи на основе метода геометрического параллелизма и метода коллективного решения.

Метод геометрического параллелизма

Согласно методу геометрического параллелизма отрезок интегрирования разбивается на ***p*** частей, где ***p*** - число используемых процессоров. Далее, каждому из процессоров предоставляется обработка одного из интервалов, причем разные процессоры вычисляют значения интеграла на разных интервалах. Рассмотрим соответствующий параллельный алгоритм 5.

```
main()
{
…
 for(i=0;i<p;i++)
            StartParallelProcess
                  ( IntTrap04, A+(B-A)*i/p, A+(B-A)*(i+1)/p, &(s[i]) )

 WaitAllParallelProcess

 J=0
 for(i=0;i<p;i++)
      J+=s[i]
}
```

Алг. 5. Параллельный алгоритм: метод геометрического параллелизма

На данном этапе рассмотрения алгоритмов основное внимание уделяется изучению основных свойств алгоритмов, обнаружению присущего им внутреннего параллелизма. Подробности и методы порождения параллельных процессов остаются на втором плане. Ключевое слово ***StartParallelProcess*** в алгоритме 5, подразумевает, что процедура-функция *IntTrap04* будет запущена ***p*** раз, при этом каждой копии будут переданы параметры, определяемыми соответствующим значением *i*. Ключевое слово ***WaitAllParallelProcess*** предполагает, что стоящие после него инструкции программы не будут выполняться, до тех пор, пока все запущенные параллельные процессы не выполнятся полностью и частичные суммы не будут записаны в соответствующие ячейки массива *s*. Дальнейшее вычисление интеграла сводится к нахождению суммы элементов массива *s* и выполняется последовательно одним процессом.

Алгоритм 5 эффективен при условии равномерного распределения всего объема вычислений по отрезку интегрирования. Однако существует множество функций при интегрировании которых указанное условие не соблюдается. В их числе рассматриваемая нами на отрезке $[10^{-5},1]$ функция 5. При разбиении интервала $[10^{-5},1]$ на ***p*** равных частей практически весь объем вычислений, необходимых для опре

деления интеграла с точностью ε =10^{-5}, сосредоточен в первой части: $\left[10^{-5},\ 10^{-5}+\frac{1-10^{-5}}{p}\right]$.

Таблица 2

Параметры расчета интеграла на разных отрезках

p	интервал 1	интервал2	время1, с	время2, с
10	[1e-5, 0.10000900000]	[0.10000900000, 1]	37.679	0.004
100	[1e-5, 0.01000990000]	[0.01000990000, 1]	37.274	0.037
1 000	[1e-5, 0.00100999000]	[0.00100999000, 1]	36.989	0.369
10 000	[1e-5, 0.00010999900]	[0.00010999900, 1]	34.064	3.364
100 000	[1e-5, 0.00001999990]	[0.00001999990, 1]	18.869	18.822

Из таблицы **2** следует полная непригодность метода геометрического параллелизма для решения поставленной задачи даже на системе из двух процессоров. При разбиении на две ***равные*** части, на первую из них приходится практически вся вычислительная нагрузка. Для обеспечения равной вычислительной нагрузки на два процессора следует разбить интервал интегрирования на две ***неравные*** части, причем одна из них должна быть в 10^5 раз меньше другой.

Метод коллективного решения

Из таблицы **2** так же следует неприменимость и метода коллективного решения (Алг. 6).

```
main()
{
// управляющий процесс

// Предполагается, что число интервалов n не меньше числа процессоров
p:
// n≥p

// Порождение p параллельных процессов, каждый из которых
// выполняет процедуру slave

for(k=0;k<p;k++)
      StartParallel(slave #k)

// Передача порожденным параллельным процессам координат
// концов отрезков интегрирования для определения частичных сумм

i=0 // число интервалов, уже переданных для обработки

for(k=0;k<p;k++)
      {
      Send(slave #k, A+(B-A)*i/n, A+(B-A)*(i+1)/n)
      i++
      }

J=0   // J - значение интеграла на всем интервале [A,B]
      // s - значение частичной суммы

// Пока есть отрезки, не переданные для отработки, следует дождаться
```

```
// сообщения от любого из процессов slave, вычислившего
// частичную сумму на переданном ему отрезке, получить
// значение этой суммы, прибавить к общему значению интеграла
// и передать освободившемуся процессу очередной отрезок

while(i<n)
      {
      Recv(slave #any, s)
      J+=s
      Send(slave #any, A+(B-A)*i/n, A+(B-A)*(i+1)/n)
      i++
      }

// Получить результаты вычислений переданных отрезков
// и прибавить их к общей сумме

for(k=0;k<p;k++)
      {
      Recv(slave #any, s)
      J+=s
      }
}

slave()
{
// подчиненный процесс, вычисляющий значение интеграла на отрезке [a,b]


      while(1)
            {
            Recv(main,a,b)
            s=IntTrap04(a,b)
            Send(main,s)
            }
}
```

Алг. 6. Параллельный алгоритм: метод коллективного решения

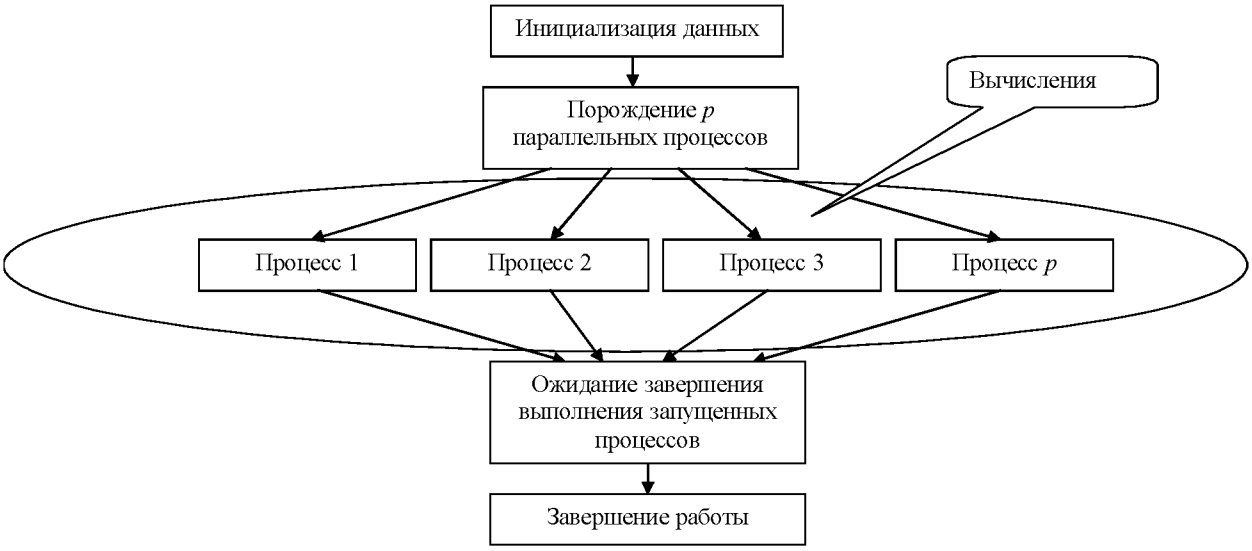

Рис. 2. График *f(x)*

Основные вычисления выполняются параллельными процессами Процесс1 … Процесс*p*, каждый из которых выполняет один и тот же алгоритм.