К. Ю. Богачёв

ОСНОВЫ параллельного программирования

ИЗДАТЕЛЬСТВО БИНОМ

К. Ю. Богачёв

ОСНОВЫ
параллельного программирования

3-е издание (электронное)

Москва
БИНОМ. Лаборатория знаний
2015

УДК 004.65
ББК 32.073
Б73

Богачёв К. Ю.

Б73 Основы параллельного программирования [Электронный ресурс] : учебное пособие / К. Ю. Богачёв. — 3-е изд. (эл.). — Электрон. текстовые дан. (1 файл pdf : 345 с.). — М. : БИНОМ. Лаборатория знаний, 2015. — (Математика). — Систем. требования: Adobe Reader XI ; экран 10".

ISBN 978-5-9963-2995-3

Данная книга представляет собой введение в методы программирования для параллельных ЭВМ.

Основной ее целью является научить читателя самостоятельно разрабатывать максимально эффективные программы для таких компьютеров.

Вопросы распараллеливания конкретных алгоритмов рассмотрены на многочисленных примерах программ на языке С. В основу книги положен курс лекций для студентов механико-математического факультета МГУ им. М. В. Ломоносова.

Для студентов, аспирантов, научных работников, программистов и всех, кто хочет научиться разрабатывать программы для параллельных ЭВМ.

УДК 004.65
ББК 32.073

Деривативное электронное издание на основе печатного аналога: Основы параллельного программирования : учебное пособие / К. Ю. Богачёв. — 2-е изд. — М. : БИНОМ. Лаборатория знаний, 2013. — 342 с. : ил. — (Математика). — ISBN 978-5-9963-1616-8.

ISBN 978-5-9963-2995-3

Оглавление

Предисловие

Параллельные ЭВМ, возникшие преимущественно для высокопроизводительных научных вычислений, получают все более широкое распространение: их можно встретить в небольших научных подразделениях и офисах. Этому способствуют как непрерывное падение цен на них, так и постоянное усложнение решаемых задач. Можно без преувеличения сказать, что параллельные компьютеры сейчас используются или планируются к использованию всеми, кто работает на передовых рубежах науки и техники:

- научными работниками, применяющими ЭВМ для решения **реальных** задач физики, химии, биологии, медицины и других наук, поскольку упрощенные модельные задачи уже рассчитаны на «обычных» ЭВМ, а переход к реальным задачам сопряжен с качественным ростом объема вычислений;
- программистами, разрабатывающими системы управления базами данных (СУБД), разнообразные Internet-серверы запросов (WWW, FTP, DNS и др.) и совместного использования данных (NFS, SMB и др.), автоматизированные системы управления производством (АСУП) и технологическими процессами (АСУТП), поскольку требуется обеспечить обслуживание максимального количества запросов в единицу времени, а сложность самого запроса постоянно растет.

В настоящий момент практически все крупные разрабатываемые программные проекты как научной, так и коммерческой

направленности либо уже содержат поддержку параллельной работы на компьютерах разных типов, либо эта поддержка запланирована на ближайшее время (причем промедление здесь часто вызывает поражение проекта в конкурентной борьбе).

Настоящая книга представляет собой введение в методы программирования параллельных ЭВМ. Основной ее целью является научить читателя самостоятельно разрабатывать максимально эффективные программы для таких компьютеров. Вопросы распараллеливания конкретных алгоритмов рассматриваются на многочисленных примерах. В качестве языка программирования использован язык C, как наиболее распространенный (и, заметим, единственный (не считая своего расширения C++), на котором можно реализовать все приведенные примеры). Программы, посвященные использованию параллелизма процесса и MPI, могут быть легко переписаны на языке FORTRAN-77. Для иллюстрации подпрограмма умножения двух матриц, дающая почти 14-кратное ускорение на одном процессоре, приведена на двух языках: C и FORTRAN-77.

Изложение начинается с изучения параллелизма в работе процессора, оперативной памяти и методов его использования. Затем приводится описание архитектур параллельных ЭВМ и базовых понятий межпроцессного взаимодействия. Для систем с общей памятью подробно рассматриваются два метода программирования: с использованием процессов и использованием задач (threads). Для систем с распределенной памятью рассматривается ставший фактическим стандартом интерфейс MPI. Для указанных систем приведены описания основных функций и примеры их применения. В описаниях намеренно выброшены редко используемые детали, чтобы не пугать читателя большим объемом информации (чем страдают большинство руководств пользователя).

Книга используется в качестве учебного пособия в основном курсе «Практикум на ЭВМ» на механико-математическом факультете МГУ им. М. В. Ломоносова по инициативе и при поддержке академика РАН Н. С. Бахвалова.

Порядок чтения

Книгу можно разделить на 5 достаточно независимых частей:

1. В главах 2, 3, 4 описан параллелизм в работе процессора и оперативной памяти, а также разнообразные приемы, используемые для повышения эффективности их работы. Эту информацию можно использовать для достижения значительного ускорения работы программы даже на однопроцессорном компьютере.
2. В главах 5 и 6 изложены основные понятия, используемые при рассмотрении параллельных программ, а также стандарты на операционные системы UNIX, установленные на подавляющем большинстве параллельных ЭВМ.
3. В главах 7 и 8 описаны основные функции для управления процессами и осуществления межпроцессного взаимодействия. Эти функции можно использовать для запуска многих совместно работающих процессов в системах с общей памятью, а также для разработки параллельного приложения для систем, не поддерживающих задачи (threads).
4. В главах 9 и 10 описаны основные функции для управления задачами (threads) и осуществления межзадачного взаимодействия. Эти функции можно использовать для разработки параллельного приложения в системах с общей памятью.
5. В главе 11 описаны основные функции Message Passing Interface (MPI). Эти функции можно использовать для разработки параллельного приложения в системах как с общей, так и с распределенной памятью.

Части расположены в рекомендуемом порядке чтения. Последние три независимы друг от друга и могут изучаться в произвольной последовательности. Главу 1, адресованную нетерпеливому читателю, при систематическом изучении рекомендуется разбирать по мере ознакомления с материалом основных частей книги.

Книгой можно воспользоваться и в качестве учебника. Для этого в конце приведены программа курса и список типовых экзаменационных задач. Эти материалы будут полезны и для самостоятельной подготовки.

1

Для нетерпеливого читателя

Для нетерпеливого читателя, желающего как можно быстрее научиться писать параллельные приложения, сразу же приведем пример превращения последовательной программы в параллельную. Для простоты рассмотрим задачу вычисления определенного интеграла от заданной функции и будем считать, что все входные параметры (концы отрезка интегрирования и количество точек, в которых вычисляется функция) заданы константами. Все использованные функции будут описаны в последующих главах.

1.1. Последовательная программа

Для вычисления приближения к определенному интегралу от функции f по отрезку $[a, b]$ используем составную формулу трапеций:

$$\int_a^b f(x)\,dx \approx h\big(f(a)/2 + \sum_{j=1}^{n-1} f(a+jh) + f(b)/2\big),$$

где $h = (b-a)/n$, а параметр n задает точность вычислений.

Вначале — файл `integral.c` с текстом последовательной программы, вычисляющей определенный интеграл этим способом:

```
#include "integral.h"
```

```
/* Интегрируемая функция */
double f (double x)
{
  return x;
}

/* Вычислить интеграл по отрезку [a, b] с числом точек
   разбиения n методом трапеций. */
double integrate (double a, double b, int n)
{
  double res;   /* результат */
  double h;     /* шаг интегрирования */
  int i;

  h = (b - a) / n;
  res = 0.5 * (f (a) + f (b)) * h;
  for (i = 1; i < n; i++)
    res += f (a + i * h) * h;
  return res;
}
```

Соответствующий заголовочный файл `integral.h`:

```
double integrate (double a, double b, int n);
```

Файл `sequential.c` с текстом последовательной программы:

```
#include <stdio.h>
#include "integral.h"

/* Все параметры для простоты задаются константами */
static double a = 0.;   /* левый конец интервала */
static double b = 1.;   /* правый конец интервала */
static int n = 100000000; /* число точек разбиения */

int main ()
{
```

```
  double total = 0.;    /* результат: интеграл */

  /* Вычислить интеграл */
  total = integrate (a, b, n);

  printf ("Integral from %lf to %lf = %.18lf\n",
          a, b, total);

  return 0;
}
```

Компиляция этих файлов:

```
cc sequential.c integral.c -o sequential
```

и запуск

```
./sequential
```

1.2. Ускорение работы за счет параллелизма

Для ускорения работы программы на вычислительной установке с p процессорами мы воспользуемся аддитивностью интеграла:

$$\int_a^b f(x)\,dx = \sum_{i=0}^{p-1} \int_{a_i}^{b_i} f(x)\,dx,$$

где $a_i = a + i * l$, $b_i = a_i + l$, $l = (b - a)/p$. Использовав для приближенного определения каждого из слагаемых $\int_{a_i}^{b_i} f(x)\,dx$ этой суммы составную формулу трапеций, взяв n/p в качестве n, и поручив эти вычисления своему процессору, мы получим p-кратное ускорение работы программы. Ниже мы рассмотрим три способа запуска p заданий для исполнения на отдельных процессорах:

1. создание новых процессов (раздел 1.3, с. 13),
2. создание новых задач (threads) (раздел 1.4, с. 18),
3. Message Passing Interface (MPI) (раздел 1.5, с. 21).

Первые два подхода удобно использовать в системах с общей памятью (см. раздел 2.5.1, с. 39). Последний подход применим и в системах с распределенной памятью.

1.3. Параллельная программа, использующая процессы

Рассмотрим пример параллельной программы вычисления определенного интеграла, использующей процессы. Описание ее работы приведено в разделе 8.4.6, с. 133, где рассматривается функция `pipe`; о функции `fork` см. раздел 7.1, с. 81. Файл `process.c` с текстом программы:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
#include "integral.h"

/* Все параметры для простоты задаются константами */
static double a = 0.;   /* левый конец интервала */
static double b = 1.;   /* правый конец интервала */
static int n = 100000000; /* число точек разбиения */

/* Канал вывода из главного процесса в порожденные.
   from_root[0] - для чтения (в порожденных процессах),
   from_root[1] - для записи (в главном процессе). */
static int from_root[2];

/* Канал вывода из порожденных процессов в главный.
   to_root[0] - для чтения (в порожденных процессах),
   to_root[1] - для записи (в главном процессе). */
static int to_root[2];

/* Функция, работающая в процессе с номером my_rank,
   при общем числе процессов p. */
void process_function (int my_rank, int p)
```

```
{
  char byte;
  /* длина отрезка интегрирования для текущего процесса*/
  double len = (b - a) / p;
  /* число точек разбиения для текущего процесса */
  int local_n = n / p;
  /* левый конец интервала для текущего процесса */
  double local_a = a + my_rank * len;
  /* правый конец интервала для текущего процесса */
  double local_b = local_a + len;
  /* значение интеграла в текущем процессе */
  double integral;

  /* Вычислить интеграл в каждом из процессов */
  integral = integrate (local_a, local_b, local_n);

  /* Ждать сообщения от главного процесса */
  if (read (from_root[0], &byte, 1) != 1)
    {
      /* Ошибка чтения */
      fprintf (stderr,
               "Error reading in process %d, pid = %d\n",
               my_rank, getpid ());
      return;
    }

  /* Передать результат главному процессу */
  if (write (to_root[1], &integral, sizeof (double))
      != sizeof (double))
    {
      /* Ошибка записи */
      fprintf (stderr,
               "Error writing in process %d, pid = %d\n",
               my_rank, getpid ());
      return;
```

```
    }
}

int main (int argc, char * argv[])
{
  /* Идентификатор запускаемого процесса */
  pid_t pid;
  /* Общее количество процессов */
  int p;
  int i;
  char byte;
  double integral = 0.;
  double total = 0.;    /* результат: интеграл */

  if (argc != 2)
    {
      printf ("Usage: %s <instances>\n", argv[0]);
      return 1;
    }

  /* Получаем количество процессов */
  p = (int) strtol (argv[1], 0, 10);

  /* Создаем каналы */
  if (pipe (from_root) == -1 || pipe (to_root) == -1)
    {
      fprintf (stderr, "Cannot pipe!\n");
      return 2;
    }

  /* Запускаем процессы */
  for (i = 0; i < p ; i++)
    {
      /* Клонировать себя */
      pid = fork ();
```

```
    if (pid == -1)
      {
        fprintf (stderr, "Cannot fork!\n");
        return 3 + i;
      }
    else if (pid == 0)
      {
        /* Процесс - потомок */
        /* Закрываем ненужные направления обмена */
        close (from_root[1]);
        close (to_root[0]);

        /* Проводим вычисления */
        process_function (i, p);

        /* Закрываем каналы */
        close (from_root[0]);
        close (to_root[1]);
        /* Завершаем потомка */
        return 0;
      }
    /* Цикл продолжает процесс - родитель */
  }

/* Закрываем ненужные направления обмена */
close (from_root[0]);
close (to_root[1]);

/* Получаем результаты */
for (i = 0; i < p ; i++)
  {
    /* Сигнализируем процессу */
    byte = (char) i;
    if (write (from_root[1], &byte, 1) != 1)
      {
```

```
          /* Ошибка записи */
          fprintf (stderr,
                   "Error writing in root process\n");
          return 100;
        }
      /* Считываем результат */
      if (read (to_root[0], &integral, sizeof (double))
          != sizeof (double))
        {
          /* Ошибка чтения */
          fprintf (stderr,
                   "Error reading in root process\n");
          return 101;
        }
      total += integral;
    }

  /* Закрываем каналы */
  close (from_root[1]);
  close (to_root[0]);

  printf ("Integral from %lf to %lf = %.18lf\n",
          a, b, total);

  return 0;
}
```

Компиляция этих файлов:

```
cc process.c integral.c -o process
```

и запуск

```
./process 2
```

где аргумент программы (в данном случае 2) — количество параллельно работающих процессов (обычно равен количеству имеющихся процессоров).

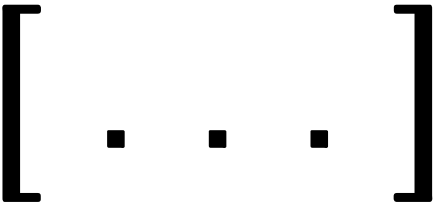
1.4. Параллельная программа, использующая задачи

Рассмотрим пример параллельной программы вычисления определенного интеграла, использующей задачи (threads). Описание ее работы см. в разделе 10.2, с. 168; описание использованных функций:

- `pthread_create` — см. раздел 9.1, с. 156;
- `pthread_join` — см. раздел 9.2, с. 157;
- `pthread_mutex_lock` — см. раздел 10.1.2, с. 162;
- `pthread_mutex_unlock` — см. раздел 10.1.4, с. 162.

Файл `thread.c` с текстом программы:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include "integral.h"

/* Все параметры для простоты задаются константами */
static double a = 0.;   /* левый конец интервала */
static double b = 1.;   /* правый конец интервала */
static int n = 100000000; /* число точек разбиения */

/* Результат: интеграл */
static double total = 0.;

/* Объект типа mutex для синхронизации доступа к total */
static pthread_mutex_t total_mutex
  = PTHREAD_MUTEX_INITIALIZER;

/* Общее количество процессов */
static int p;

/* Функция, работающая в задаче с номером my_rank */
void * process_function (void *pa)
{
```



Минимальные системные требования определяются соответствующими требованиями программы Adobe Reader версии не ниже 11-й для платформ Windows, Mac OS, Android, iOS, Windows Phone и BlackBerry; экран 10″

Учебное электронное издание

Серия: «Математика»

Богачёв Кирилл Юрьевич

ОСНОВЫ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

Учебное пособие

Ведущий редактор *М. С. Стригунова*
Художественный редактор *Н. А. Новак*
Технический редактор *Е. В. Денюкова*
Корректор *Н. Н. Ектова*
Оригинал-макет подготовлен *О. Г. Лапко* в пакете LaTeX 2ε

Подписано к использованию 19.03.15. Формат 125×200 мм

Издательство «БИНОМ. Лаборатория знаний»
125167, Москва, проезд Аэропорта, д. 3
Телефон: (499) 157-5272
e-mail: info@pilotLZ.ru, http://www.pilotLZ.ru

Данная книга представляет собой введение в методы программирования для параллельных ЭВМ.

Основной ее целью является научить читателя самостоятельно разрабатывать максимально эффективные программы для таких компьютеров.

Вопросы распараллеливания конкретных алгоритмов рассмотрены на многочисленных примерах программ на языке С. В основу книги положен курс лекций для студентов механико-математического факультета МГУ им. М. В. Ломоносова.

Для студентов, аспирантов, научных работников, программистов и всех, кто хочет научиться разрабатывать программы для параллельных ЭВМ.