Лабораторная работа 9

14.04.2025

Технология OpenMP: распараллеливание циклов

OpenMP (Open Multi-Processing) – это набор директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах с общей памятью. Первый стандарт OpenMP был разработан в 1997 году.

Существует множество источников, по которым можно изучать технологию OpenMP. Например:

1. Сайт www.openmp.org – основной источник информации.
2. Гергель, В.П. Высокопроизводительные вычисления для многоядерных многопроцессорных систем, Глава 5.
3. https://learn.microsoft.com/ru-ru/cpp/parallel/openmp/openmp-in-visual-cpp?view=msvc-170
4. https://bisqwit.iki.fi/story/howto/openmp/ – быстрый старт.

Заголовочный файл <omp.h>.

Свойства проекта – необходимость поддержки OpenMP (Свойства проекта-C/C++-Язык-Поддержка Open MP).

Задание 1. Параллельный регион.

Директива *parallel* определяет параллельный регион – код, который будет выполняться несколькими потоками параллельно. Количество потоков определяется во время выполнения, обычно исходя из количества процессоров вычислительной системы. Каждый из потоков выполняет код, который находится в параллельном регионе. После параллельного региона потоки снова объединяются в один. Например:

```
#pragma omp parallel
  {
    // Код внутри этой области выполняется параллельно
  printf("Hello!\n");
  }
```

Этот код создает группу потоков. Каждый поток выполняет один и тот же код – выводит текст «Привет!\n» столько раз, сколько потоков создано.

Полезные функции:

- *omp_get_num_threads()* возвращает количество потоков в параллельной области;
- *omp_set_num_threads()* задает количество потоков в предстоящих параллельных регионах;
- *omp_get_thread_num()* возвращает номер потока, из которой функция вызвана;
- *omp_get_wtime()* возвращает время в секундах.

Выполнить:

- Запустите приведенную выше программу.
- Добавьте вывод номера потока.
- Синхронизируйте конкурентное использование std::cout.

Отчет:

- Текст программы.
- Окно работы программы.

Задание 2. Параллельный цикл.

Конструкция *parallel for* разделяет итерации цикла между потоками. Например:

```
#pragma omp parallel for
  for (int n = 0; n < 10; ++n)
  {
    printf(" %d ", n);
  }
```

Приведенный выше цикл превращается в код, эквивалентный следующему:

```
#pragma omp parallel
  {
    int this_thread = omp_get_thread_num(),
        num_threads = omp_get_num_threads();
    int my_start = (this_thread) * 10 / num_threads;
    int my_end = (this_thread + 1) * 10 / num_threads;
    for (int n = my_start; n < my_end; ++n)
      printf(" %d ", n);
  }
```

Таким образом, каждый поток получает отдельный участок цикла, и они выполняют свои участки параллельно.

Выполнить:

- Запустите приведенную выше программу.
- Проверьте экспериментально, как распределяются итерации цикла между потоками.
- Проведите вычислительные эксперименты, изменяя размер данных и количество потоков, со следующим кодом:

```
// Параллельная инициализация таблицы
  const double M_PI = 3.141592653589;
  const int size = 256;
  double sinTable[size];

#pragma omp parallel for
  for (int n = 0; n < size; ++n)
    sinTable[n] = std::sin(2 * M_PI * n / size);

  // Таблица инициализирована
```

Отчет:

- Текст программы.
- Таблицы с результатами экспериментов.
- Выводы.

Задание 3. Планирование цикла.

Алгоритм планирования цикла *for* можно явно контролировать. Например:

```
#pragma omp parallel for schedule(static)
  for (int n = 0; n < 10; ++n)
    printf(" %d ", n);
```

Выполнить:

- Изучите алгоритмы планирования цикла.
- Проверьте экспериментально, как распределяются итерации цикла между потоками для вариантов:

#pragma omp parallel for for schedule(static)
#pragma omp parallel for schedule(dynamic)
#pragma omp parallel for schedule(guided)

Отчет:

- Выводы.

Задание 4. Взаимное исключение.

Директива *atomic* обычно используется для обновления счетчиков и других простых переменных, к которым одновременно обращаются несколько потоков:

```
  int counter = 0,
      value = 2,
      n = 10000;
#pragma omp parallel for
  for (int i = 0; i < n; ++i)
  {
#pragma omp atomic
    counter += value;
  }
  printf("counter = %d\n", counter);
```

Выполнить:

- Запустите приведенную выше программу с директивой atomic и без нее.

Директива *critical* указывает, что код выполняется только в одном потоке одновременно. Например:

```
  max = a[0];
#pragma omp parallel for num_threads(4)
  for (i = 1; i < SIZE; i++)
  {
    if (a[i] > max)
    {
#pragma omp critical
      {
        // compare a[i] and max again because max
        // could have been changed by another thread after
        // the comparison outside the critical section
        if (a[i] > max)
          max = a[i];
      }
    }
  }
```

Выполнить:

- Реализуйте поиск минимального значения в одномерном массиве.
- Проведите вычислительные эксперименты, изменяя размер данных и количество потоков. Результаты оформите в виде таблицы.

Отчет:

- Текст программы.
- Таблицы с результатами экспериментов.

Задание 5. Разделение данных между потоками.

В директиве *parallel* можно указать, какие переменные являются общими для разных потоков, а какие нет. По умолчанию все переменные являются общими, кроме тех, которые объявлены в параллельном блоке. Например:

```
  int a = 0,
      b = 0;
#pragma omp parallel for private(a) shared(b)
  for (a = 0; a < 50; ++a)
  {
#pragma omp atomic
    b += a;
  }
```

В этом примере явно указано, что `a` является *private* (каждый поток имеет свою собственную копию) и `b` является *shared* (каждый поток обращается к одной и той же переменной).

Директива *reduction* позволяет накапливать общую переменную без директивы *atomic*, но необходимо указать тип накопления. Это часто приводит к более быстрому выполнению кода, чем при использовании директивы *atomic*.

Функция для вычисления факториала с директивой *reduction*:

```
int factorial(int number)
{
  int fac = 1;
#pragma omp parallel for reduction(*:fac)
  for (int n = 2; n <= number; ++n)
    fac *= n;
  return fac;
}
```

В начале параллельного блока делается приватная копия переменной. В конце параллельного блока приватная копия атомарно объединяется с общей переменной с помощью заданного оператора.

Функция для вычисления факториала без директивы *reduction*:

```
int factorial(int number)
{
  int fac = 1;
#pragma omp parallel for
  for (int n = 2; n <= number; ++n)
  {
#pragma omp atomic
    fac *= n;
  }
  return fac;
}
```

Выполнить:

- Проведите эксперименты с функциями для вычисления факториала.
- Задача – вычисление числа π. Реализуйте в разных версиях. Проведите вычислительные эксперименты. Результаты оформите в виде таблицы.
- Задача – вычисление скалярного произведения двух векторов. Реализуйте в разных версиях. Проведите вычислительные эксперименты. Результаты оформите в виде таблицы.

Отчет:

- Тексты программ.
- Таблицы с результатами экспериментов.