САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Параллельные вычисления

Проектирование многопоточных программ

Михаил Моисеев

Санкт-Петербург
2012

Создание параллельных программ на основе OpenMP

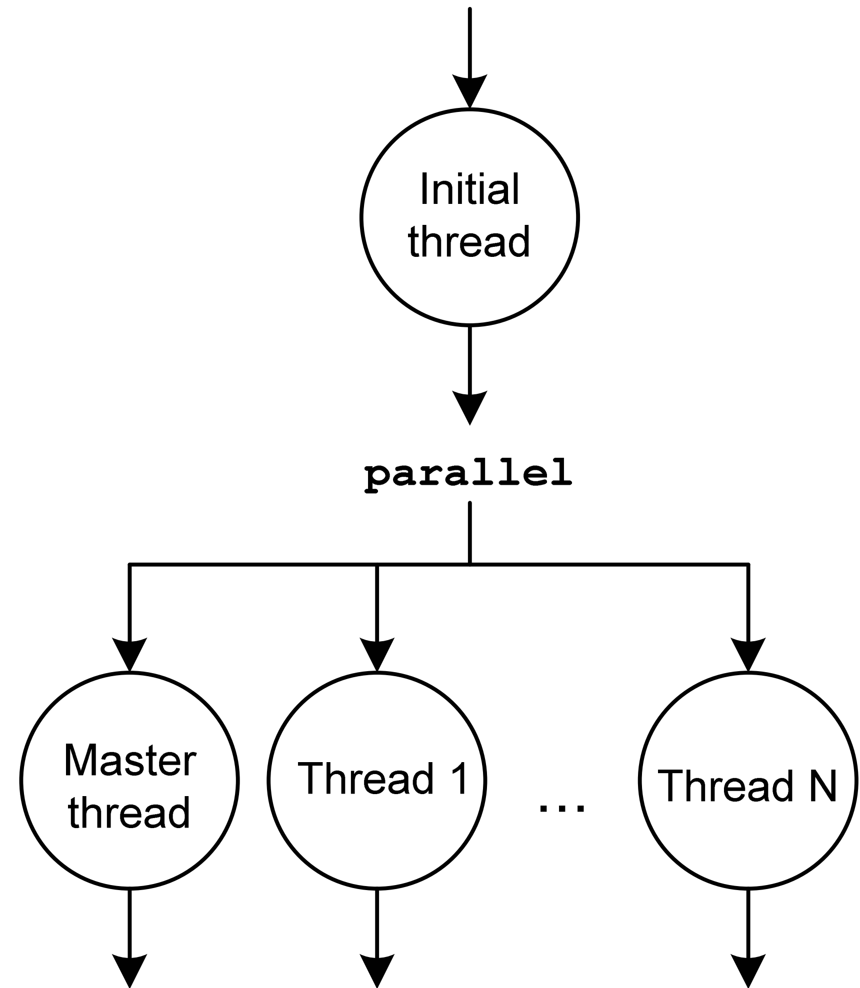
- OpenMP (Open Multi-Processing) – технология создания параллельных программ с общей памятью (http://www.openmp.org)
- OpenMP позволяет создавать программы, параллельно выполняющие одинаковые операции для разных данных (Single Program Multiple Data)
- OpenMP разработан для языков C, C++, Fortran

Создание параллельных программ на основе OpenMP

- OpenMP представляет набор директив компилятора, библиотечных функций и переменных окружения
- Директивы OpenMP применяются к структурированным блокам
  - структурированный блок – простой или составной оператор программы (`{...}`), который имеет единственный вход и единственный выход
- Директивы OpenMP
  - для языков C/C++ – **`#pragma omp …`**
  - для языка Fortran – комментарий
- Поддержка компиляторами
  - gcc, Oracle Solaris Studio, MS Visual Studio, Intel compilers…

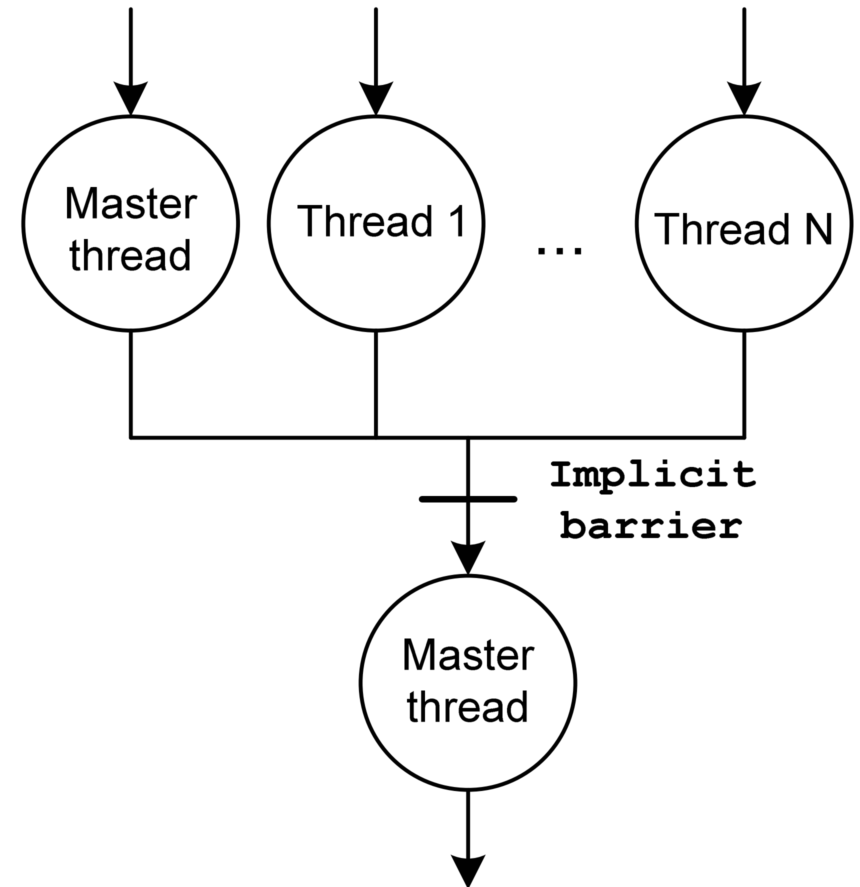
Модель параллельных вычислений OpenMP

- При старте программы запускает один начальный поток (Initial thread)
- При достижении директивы `parallel` порождается группа потоков, родительский поток также входит в эту группу как master thread
- С каждым потоком связывается задача, в которой выполняется код внутри директивы `parallel`
- Задачи в разных потоках параллельно обрабатывают разные данные



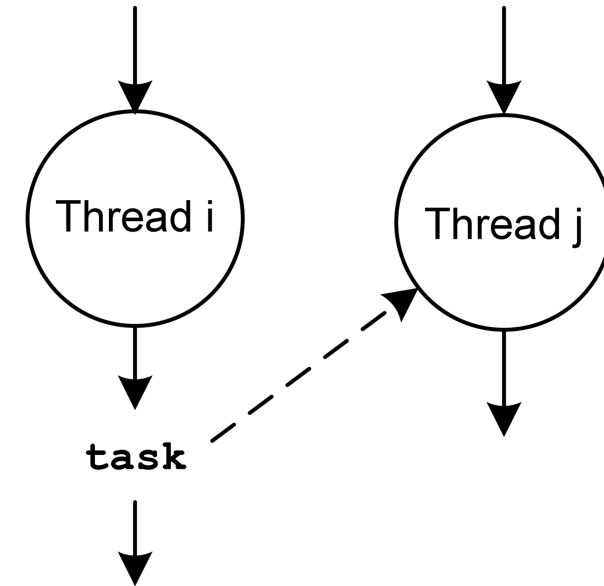
Модель параллельных вычислений OpenMP

- После завершения всех потоков продолжает выполнение master thread, для ожидания потоков используется барьер (неявно)
- Для того чтобы не ожидать завершение всех потоков можно использовать директиву `nowait`
- Программа может содержать произвольное число директив parallel
- Поддерживаются вложенные директивы `parallel` – внутри потока может создаваться своя группы потоков



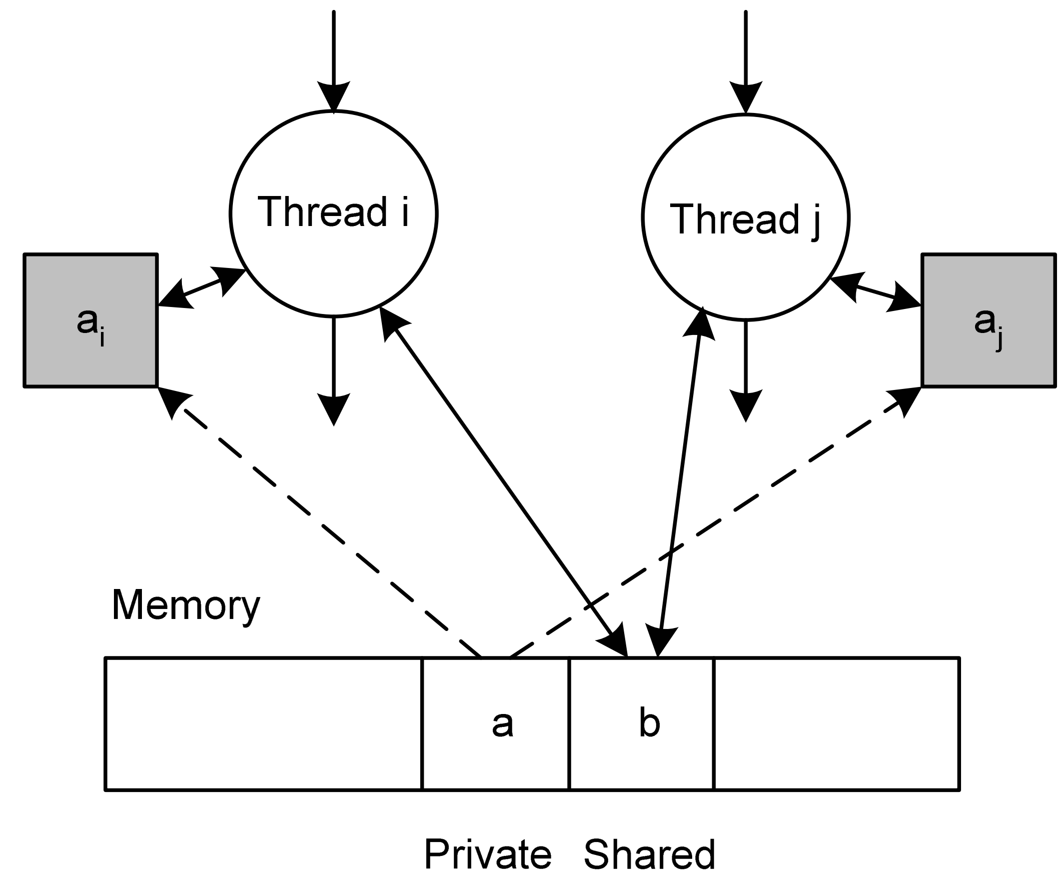
Модель параллельных вычислений OpenMP

- Директива `task` позволяет создавать явные задачи, которые будут выполняться в одном из потоков текущей группы
- Все задачи, созданные с помощью `task` завершаются до завершения потоков данной группы



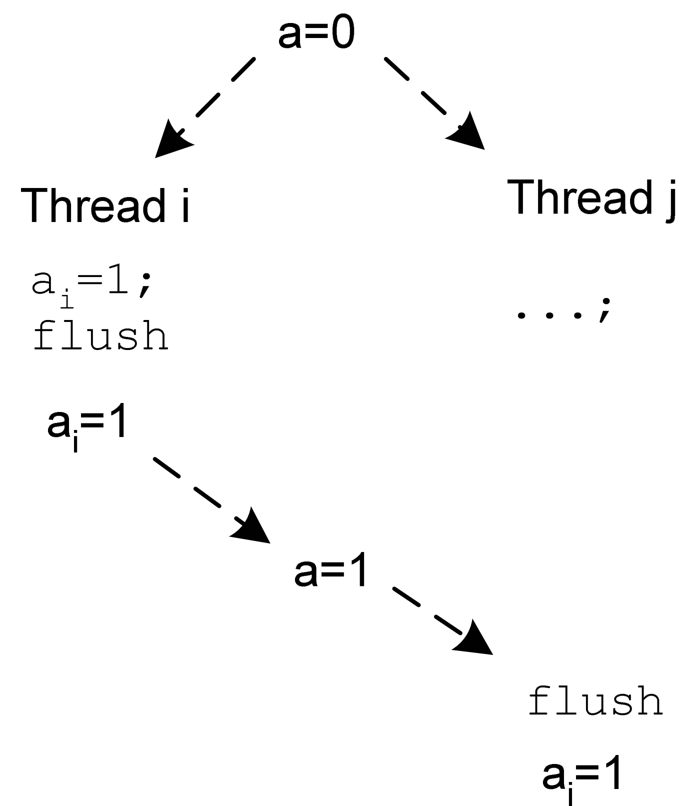
Модель памяти в OpenMP

- При создании потоков для переменной можно создать локальную копию для каждого потока (Private)
- Private переменные используются только в текущем потоке, могут инициализироваться значением оригинальной переменной, результат может быть сохранен в оригинальной переменной
- Shared переменные должны использоваться либо только на чтение, либо защищаться с помощью директив синхронизации



Модель памяти в OpenMP

- Relaxed-consistency модель памяти – значение локальных копий могут различаться в разных потоках
- Для синхронизации значения локальной копии переменной и переменной в памяти используется директива `flush`
- Директива `flush` определяет место последнего изменения переменной и выполняет запись локального значения в память или чтение значения из памяти в локальную копию (при изменении значения в другом потоке)
- Программист должен обеспечить отсутствие конкурентной модификации переменной для которой выполняется `flush`



Основные директивы OpenMP

- Создание и запуск группы параллельных потоков

```
#pragma omp parallel [clause[ [, ]clause] ...]
structured-block
```

```
clause: if(scalar-expression), num_threads(integer-expression),
default(shared | none), private(list), firstprivate(list),
shared(list), copyin(list), reduction(operator: list)
```

Пример создания параллельных потоков

```
#include <stdio.h>
#include <omp.h>
int main(){
   #pragma omp parallel num_threads(2)
   {
      if (omp_get_thread_num() == 0) {
         doSomething(); // In master thread
      } else {
         doSomethingElse(); // In other parallel thread
      }
   }
```

Пример разделяемых и локальных переменных

```
int main(){
    int x = 0; int y = 0;
    #pragma omp parallel num_threads(2) shared(x) private(y)
    {
        if (omp_get_thread_num() == 0) {
            x = 1;
            y = 1;
        } else {
            y = 2;
        }
    }
    printf("%d %d", x, y); // 1, 0
```

Директивы для циклов

- Распараллеливание итераций цикла в потоках текущей группы

```
#pragma omp for [clause[ [, ]clause] ...]
for-loops
clause:private(list), firstprivate(list), lastprivate(list),
reduction(operator: list), schedule(kind[, chunk_size]),
collapse(n), ordered, nowait
```

- Создание группы параллельных потоков для распараллеливания итераций цикла

```
#pragma omp parallel for [clause[ [, ]clause] ...]
for-loops
```

Ограничения на распараллеливаемые циклы

- `for(init-expr; test-expr; incr-expr) structured-block`
  - переменная цикла должна иметь тип signed int
  - операция сравнения должна иметь следующий формат

    `test-expr:: variable {<| <= | > | >=} const`
  - инкрементная часть цикла должно целочисленным сложением или вычитанием

    `incr-expr:: variable{+|-} const`
  - test-expr должно быть согласовано с incr-expr
  - запрещены выходы из цикла (за исключением `exit`), операторы goto и break, могут использоваться только для переходов внутри цикла, исключения должны перехватываться внутри цикла
- Пример распараллеливаемого цикла
  - `for(i = 0; i < 10; i++) {a[i] = b[i];}`

Пример распараллеливания итераций цикла

```
void simple(int n, float *a, float *b)
{
   int i;
   #pragma omp parallel for
   for (i=1; i<n; i++){  /* i is private by default */
      b[i] = (a[i] + a[i-1]) / 2.0;
   }
}
```

Зависимости данных и гонки в циклах

- При распараллеливании цикла не должно быть зависимости между итерациями цикла

```
void simple(int n, float *a, float *b)
{
    #pragma omp parallel for
    for (int i=1; i<n; i++){
        a[i+1] = i * a[i];
    }
}
```

Основные директивы OpenMP

- Распределение задач в потоках текущей группы

#pragma omp sections {section1, section2, …}

#pragma omp section *structured-block*

```
#pragma omp parallel {
  #pragma omp sections {
    #pragma omp section
       Task1();
    #pragma omp section
       Task2();
  }
}
```

Основные директивы OpenMP

- Выполнение конструкций указанного блока только в одном из потоков текущей группы

```
#pragma omp single [clause[ [, ]clause] ...]
structured-block
clause:private(list), firstprivate(list), copyprivate(list),
nowait
```

- Явное создание параллельной задачи

```
#pragma omp task [clause[ [, ]clause] ...]
structured-block
clause: if(scalar-expression), final(scalar-expression),
untied, default(shared | none), mergeable, private(list),
firstprivate(list), shared(list)
```

Пример работы директивы task

```
struct node {
   struct node *left;
   struct node *right;
}
void traverse( struct node *p ) {
   if (p->left)
   #pragma omp task // p is firstprivate by default
      traverse(p->left);
   if (p->right)
   #pragma omp task // p is firstprivate by default
      traverse(p->right);
   doSomething(p);
}
```

Пример работы директивы single

```
void single_example()
{
   #pragma omp parallel
   {
       #pragma omp single
          printf("Beginning doSomething1");
       doSomething1();
       #pragma omp single nowait
          printf("Finished doSomething1");
       doSomething2();
    }
}
```

Директивы синхронизации

- Ограничение на выполнение следующего блока одновременно только одним потоком

```
#pragma omp critical [(name)]
 structured-block
```

- Создание барьера **#pragma omp barrier**
- Ожидание завершения всех порожденных задач

```
#pragma omp taskwait
```

- Создание атомарного блока конструкций

```
#pragma omp atomic [read | write | update | capture]
expression-stmt | structured-block
```

- Синхронизация локальной копии переменной и переменной в памяти

```
#pragma omp flush [(list)]
```

Пример работы директивы critical

```
void critical_example(float *x)
{
   int ix_next;
   #pragma omp parallel shared(x) private(ix_next)
   {
       #pragma omp critical (critical_section_name)
           ix_next = dequeue(x);
       doSomething(ix_next);
    }
}
```

Пример работы директив atomic и flush

```
int main(){
   int x = 0;
   #pragma omp parallel num_threads(2)
   {
     if(omp_get_thread_num()==0) {
          #pragma omp atomic
             x = someValue;
      } else {
          while(x == 0){
             #pragma omp flush(x)
          }
          ...
      }
```

Пример работы директивы reduction

```
void reduction1(float *x, int *y, int n)
{
  int i, c; float a, d;
  a = 0.0; c = y[0]; d = x[0];
  #pragma omp parallel for private(i) shared(x, y, n) \
  reduction(+:a) reduction(min:c) reduction(max:d)
  for (i=0; i<n; i++) {
     a += x[i];
     if (c > y[i]) c = y[i];
     d = fmaxf(d,x[i]);
  }
}
```

Runtime Library Routines

- `void omp_set_num_threads(int num_threads)`
- `int omp_get_num_threads(void)`
- `int omp_in_parallel(void)`
- `int omp_get_thread_num(void)`