Проект комиссии Президента
по модернизации и технологическому развитию экономики России
«Создание системы подготовки высококвалифицированных кадров
в области суперкомпьютерных технологий и
специализированного программного обеспечения»

УТВЕРЖДАЮ
Председатель экспертного совета
системы НОЦ СКТ, член-корр. РАН
В.В. Воеводин
_____
"_____"_____201__г.

Конспект лекций дисциплины

«Параллельное программирование для многопроцессорных систем
с общей и распределенной памятью»

«**010400.62 – Прикладная математика и информатика**»

Разработчики: Лаева В.И., Трунов А.А.
Рецензент: проф. Вшивков В.А.

Москва

OpenMP. Лекция № 2

Директива for (parallel for)

C/C++

```
#pragma omp parallel [опции]
{
  #pragma omp for [опции]
  for (i = инвариант цикла; i {<,>,==,<=,>=} инвариант цикла;
          i изменяется на инвариант цикла)
{...}
}
```

или, если цикл **for** внутри **parallel** только один

```
#pragma omp parallel for [опции]
for (инициализация; условие выхода; постобработка)
{...}
```

Fortran

```
!$OMP parallel [опции]
  !$OMP do [опции]
  do i = инвариант цикла, инвариант цикла
     end do
     !$OMP end do [опции]
!$OMP end parallel
```

или, если цикл **for** внутри **parallel** только один

```
!$OMP parallel do [опции]
  do i = инвариант цикла, инвариант цикла
     end do
!$OMP end parallel do
```

Фрагмент программы (parallel for):

```
typedef double scalar_t;
#define N 32
int main(int argc, char *argv[])
{
     scalar_t a[N], b[N], c[N];
     for (int i = 0; i < N; i++) {a[i] = 1.0; b[i] = 2.0;}
     #pragma omp parallel for
          for (int i = 0; i < N; i++)
               c[i] = a[i] + b[i];
     for (int i = 0; i < N; i++)
          printf("c[%d] = %.2f\n", i, c[i]);
     return 0;
}
```

Директива for относится только к внешнему циклу, при наличии вложенных циклов. По умолчанию почти все переменные при входе в параллельную область остаются общими (shared). Исключение: счётчики циклов становятся собственными (приватными, локальными, private). Переменные, порождённые внутри параллельной области, становятся

собственными. Явно назначить класс переменных по умолчанию можно с помощью опции default(). Рекомендуется явно указывать классы переменных, применяя опцию default(none). Статические (static) переменные языка C, размещённые в параллельной области являются общими (shared). В языке Fortran по умолчанию общими являются элементы COMMON-блоков.

Фрагмент программы:

```
// умножение матрицы A и вектора x
scalar_t A[M][N], x[N], y[M];
      // y = A * x
#pragma omp parallel for default(none) shared(A,x,y)
     for (int i = 0; i < M; i++)
         for (int j = 0; j < N; j++)
             y[i] += A[i][j] * x[j];
```

Требования к параллельным циклам:

- корректная программа не должна зависеть от распределения итераций цикла по потокам;
- нельзя использовать побочный выход из цикла (оператор типа break), но можно использовать оператор продолжения цикла типа continue;
- количество итераций цикла должно быть чётко фиксировано до начала цикла, поэтому в инициализаторах, условиях останова и для приращения счётчика цикла используются только инварианты цикла;
- зависимости между итерациями должны быть разрешены программистом, не компилятором.

Опции распределения работы в параллельных циклах:

- **private**, **firstprivate**, **lastprivate**, **reduction**
- **schedule(**тип, [блок]) - задание распределения итераций цикла по потокам
- **collapse(n**) - n тесно вложенных циклов ассоциируются с директивой for; для циклов образуется общее пространство итераций; по умолчанию n = 1 (OpenMP версии 3.0)
- **ordered** - опция, позволяющая использовать директиву **ordered** в теле цикла (директива ordered позволяет выполнять операторы в порядке, в котором они идут согласно номеру итерации)
- **nowait** - опция позволяет не выполнять неявную барьерную синхронизацию в конце цикла

Использование опции schedule для распределения итераций между потоками:

- **static** (статическое распределение) - блочно-циклическое распределение (0-й поток выполняет 1-й блок итераций, ..., затем 0-й поток выполняет очередной блок итераций, ...); если размер блока не указан, то пространство итераций делится непрерывные части приблизительно одинакового размера

- **dynamic** (динамическое распределение) - изначально потоки получают блоки фиксированного размера, очередной освободившийся от работы поток получает следующий блок итераций
- **guided** (ведомое распределение) - разновидность динамического распределения, при котором размер блока итераций уменьшается с некоторого начального значения до указанного размера блока пропорционально кол-ву ещё нераспределённых итераций
- **auto** (автоматическое распределение) - определяется компилятором и/или во время выполнения; параметр блок(**chunk**) не задаётся
- **runtime** (распределение времени выполнения) - определяется во время выполнения с использованием переменной окружения **OMP_SCHEDULE** или функции **omp_set_schedule**()

Использование опции ordered для упорядочивания итераций:

```
#define K 2
int nt = omp_get_max_threads();
#pragma omp parallel default(none) shared(nt)
{
        int tid = omp_get_thread_num();
        #pragma omp for
        for (int i = 0; i < K*nt; i++)
             printf("Итерация %d (Поток %d)\n", i, tid);

        #pragma omp for ordered
        for (int i = 0; i < K*nt; i++)
             #pragma omp ordered
             printf("Итерация %d (Поток %d)\n", i, tid);
}
```

Директивы sections/section предназначены для реализации конечного (неитеративного) параллелизма. Каждый поток при входе в область программы, определяемую данной директивой выполняет свою секцию. Директива **sections** располагается внутри параллельной области, директивы **section** располагаются только внутри **sections.**

- C/C++

```
#pragma omp sections [опции]
{
     #pragma omp section
     {…}
     #pragma omp section
     {…}
     ...
}
```

- Fortran

```
!$omp sections [опции]
          !$omp section
          <операторы>
```

```
          !$omp section
           <операторы>
          …
!$omp end sections
```

Фрагмент программы:

```
#pragma omp parallel sections default(none)
{
     #pragma omp section
     {
        printf(«Секция#1. Поток #%d\n", omp_get_thread_num());
     }
     #pragma omp section
     {
        printf(«Секция#2. Поток #%d\n", omp_get_thread_num());
     }
     #pragma omp section
     {
        printf(«Секция#3. Поток #%d\n", omp_get_thread_num());
     }
}
```

Опции директивы sections:

- **private(**список**)** - собственные переменные (начальные значения не определены);
- **firstprivate(**список**) -** начальные значения переменных берутся из мастер-потока;
- **lastprivate(**список**) -** конечные значения переменных для мастер-потока берутся из последней секции;
- **reduction(**оператор:список_общих_переменных**)** – редукция;
- **nowait** – отмена неявной барьерной синхронизации;

Директива workshare предназначена для реализации конечного (неитеративного) параллелизма только на языке Fortran. Каждый оператор в пределах секции выполняется своим потоком, синхронизация между операторами выполняется автоматически.

```
!$omp workshare
<набор независимых операторов,
выполняемых параллельно>
!$omp end workshare [nowait]
```

Фрагмент программы:

```
double precision :: A(N,N), B(N,N), C(N,N), D(N,N)
!$omp parallel workshare default(none) shared(A,B,C,D)
C = A + B
D = A * B
!$omp end parallel workshare
```

Директива **single**

Блок операторов, относящийся к данной директиве, будет выполнен только 1-м потоком. При этом, номер потока не определён: может меняться от запуска программы к запуску и может меняться от одной директивы к другой в пределах программы. Директива используется, когда только один поток должен выполнить операцию (например, обновить общую переменную), но номер этого потока не важен. Остальные потоки ожидают завершения работы потока, выполняющего single.

- C/C++

```
#pragma omp single [опции]
{
      <операторы, исполняемые одним потоком>
}
```

- Fortran

```
!$omp single [опции]
     <операторы, исполняемые одним потоком>
!$omp end single [nowait] [copyprivate]
```

Фрагмент программы:

```
int nt;
#pragma omp parallel default(none) shared(nt)
{
     int tid = omp_get_thread_num();
     #pragma omp single
     {
    nt = omp_get_num_threads();
    printf(«Кол-во потоков: %d. Получил поток #%d\n", nt,
tid);
    printf("******************\n");
     }
     printf(«Поток #%d. Кол-во потоков: %d\n", tid, nt);
}
```

Опции директивы **single**

- **private(**список**), firstprivate(**список**)**;
- **copyprivate(**список**)-** копирование переменных списка в собственные переменные всех потоков, описанные в начале параллельной области (опция используется только директивой single); не используется с опцией **nowait**, т.к. для неё требуется синхронизация;
- **nowait** – отмена неявной барьерной синхронизации.

Директива **master**

Блок операторов, относящийся к данной директиве, будет выполнен только мастер-потоком (имеет номер 0). Остальные потоки пропускают данный блок. Неявная барьерная синхронизация отсутствует. Также отсутствует обновление данных (flush). Стоит отметить, что, как правило, нет причин использовать **master** вместо **single.**

- C/C++

```
#pragma omp master
{
      <операторы, исполняемые мастер-потоком>
}
```

- Fortran

```
!$omp master
     <операторы, исполняемые мастер-потоком>
!$omp end master
```

Фрагмент программы

```
#pragma omp parallel default(none) shared(nt)
{
     int tid = omp_get_thread_num();
     #pragma omp master
     {
        nt = omp_get_num_threads();
        printf(«Кол-во потоков: %d. Получил поток #%d\n", nt,
tid);
        printf("******************\n");
     }
     #pragma omp barrier
     #pragma omp for schedule(static, 1)
     for(int i = 0; i < nt; i++)
        printf(«Поток #%d. Кол-во потоков: %d\n", tid, nt);
}
```

Директива **threadprivate**

Время жизни собственных переменных потока ограничено параллельной секцией – вне секции переменные потока не существуют. Если требуется сохранить собственные переменные между параллельными областями – используется директива **threadprivate**.

C/C++

```
#pragma omp threadprivate(список_переменных)
```

Fortran

```
!$omp threadprivate(список_переменных)
```

Переменные объявленные как **threadprivate** не могут использоваться в опциях директив OpenMP, кроме **copyin**, **copyprivate**, **schedule**, **num_threads**, **if**. Кол-во потоков должно оставаться одинаковым для всех параллельных областей (нельзя использовать динамическое кол-во потоков). Начальные значения переменных не определены до использования опции **copyin,** которая поддерживается только директивой **parallel**. Директива должна быть объявлена после объявления переменных из её списка.

- C/C++: переменные объявленные как **threadprivate** должны иметь областью видимости весь файл/пространство имён(namespace) или быть статическими (static);

- Fortran: переменные должны быть глобальными (принадлежать COMMON-блокам).

Фрагмент программы:

```
int accum = 1;
#pragma omp threadprivate(accum)
int main(int argc, char *argv[])
{
omp_set_dynamic(0);
#pragma omp parallel default(none) copyin(accum)
{
   int tid = omp_get_thread_num();
   accum *= (tid + 1);
}
#pragma omp parallel default(none)
{
   accum *= 2;
   printf("accum = %d\n", accum);
}
return 0;
}
```

Оторванные (orphaned) директивы

Стандарт OpenMP не ограничивает применение директив распределения работы и синхронизации параллельной областью. Т.е. директивы **for**, **single**, **barrier**, **critical**, ... могут быть "оторваны", т.е. находится вне параллельной области. Можно, например, помещать такие директивы внутрь функций. Если функция будет выполняться в последовательной области, то директива будет проигнорирована. Если функция будет выполняться в параллельной области, то директива будет выполнена.

Фрагмент программы:

```
void dowork(void)
{
#pragma omp for
for (...)
      {...}
}

// вызов функции в последовательной области
dowork();

#pragma omp parallel
{
     dowork();
}
```