Курс: « Архитектура и программное обеспечение терафлопных вычислителей»

март-май 2008 г.

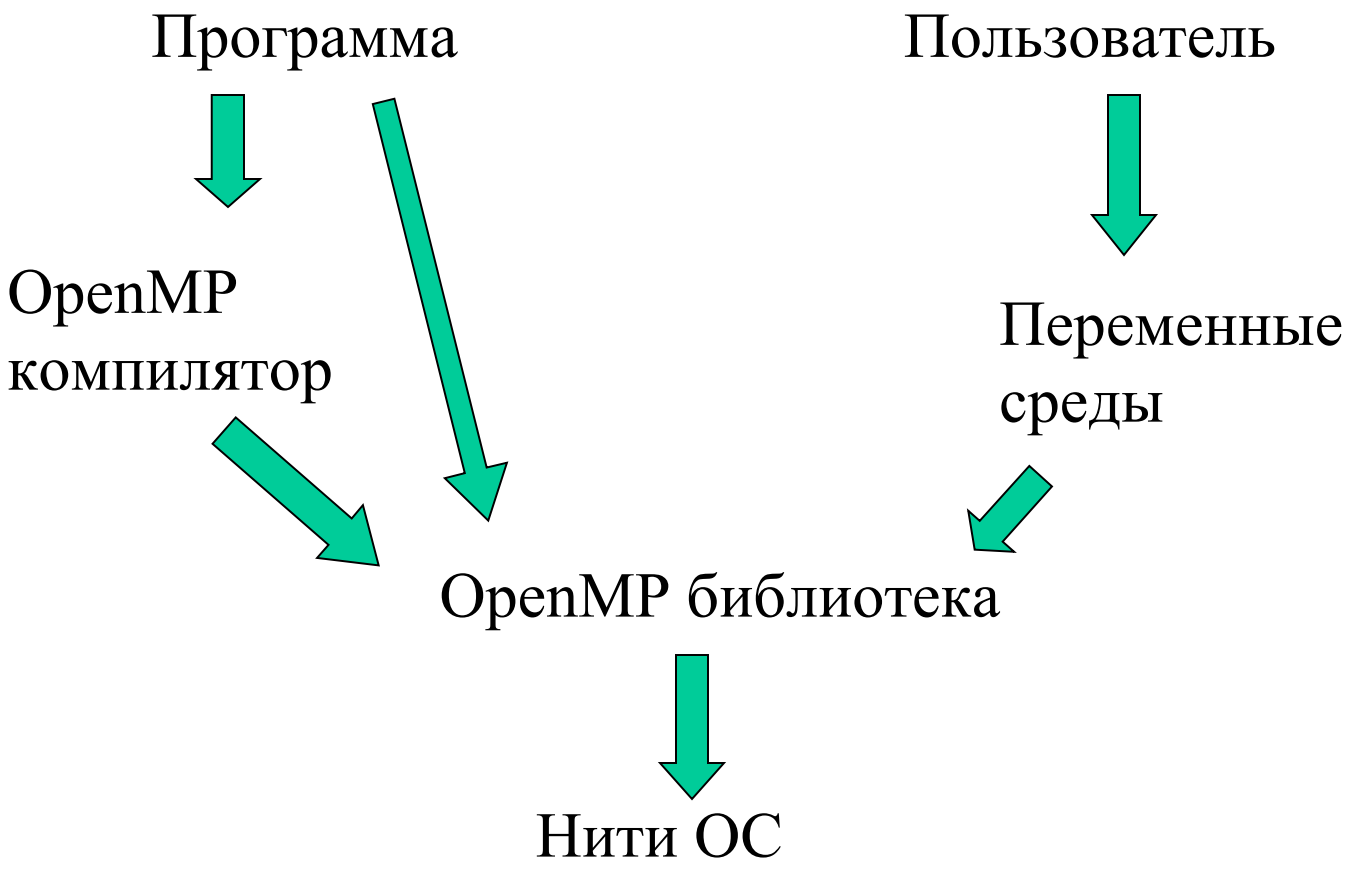
Лекция: **Параллельное программирование для систем с общей памятью с использованием технологии OpenMP. Программные средства Интел для отладки параллельных (OpenMP) программ**

Лектор: Попова Н.Н.

OpenMP

- Расширение последовательных языков путем введения параллельных конструкций в язык – директив OpenMP
- Если компилятор не распознает OpenMP директиву, программа сохраняет функциональность Parallelization is orthogonal to the functionality
  - If the compiler does not recognize the OpenMP directives, the code remains functional (albeit single-threaded)
- Базируется на идее нитевого программирования для систем с общей памятью
- Включает в свой состав: параллельные конструкции, библиотеку функций, а также специальные переменные окружения
- Индустриальный стандарт
  - Поддерживаетс Intel, Microsoft, Sun, IBM, HP, …
    В ряде случаев поведение зависит от реализации

Архитектура OpenMP



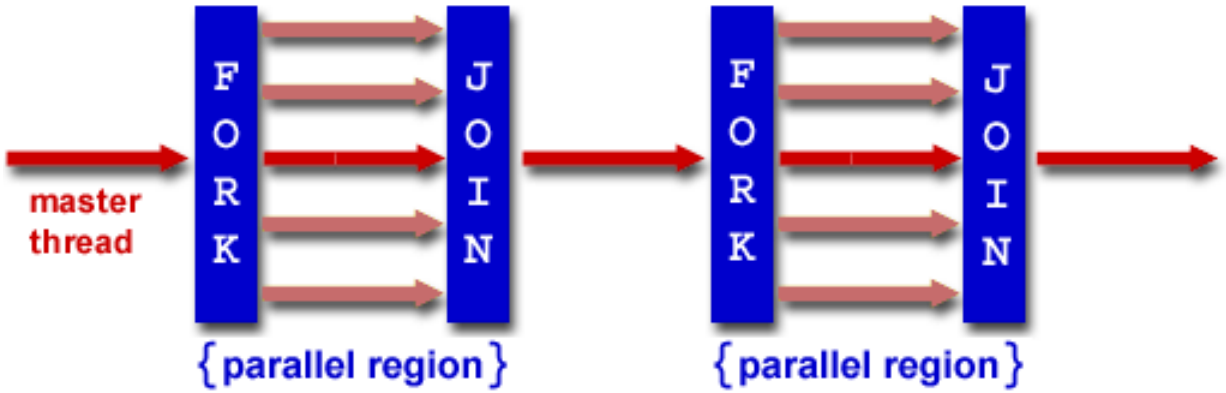
```
#include <stdio.h>
#include <omp.h>

main ()  {
int nthreads, tid;

#pragma omp parallel private(nthreads, tid)
  {
   tid = omp_get_thread_num();
   printf("Hello World from thread = %d\n", tid);
   if (tid == 0)
    {
     nthreads = omp_get_num_threads();
     printf("Number of threads = %d\n", nthreads);
    }
  }

}
```

Модель выполнения OpenMP программы



Работа с вычислительным пространством

- Мастер-нить имеет номер 0
- Число нитей, выполняющих работу определяется:
  - переменная окружения **OMP_NUM_THREADS**
  - вызов функции **omp_set_num_threads**()
- Определение «своих» координат в вычислительном пространстве:
  - свой номер: **omp_get_thread_num**()
  - число нитей: **omp_get_threads_num**()

OpenMP модель памяти

- Модель разделяемой памяти
  - Нити взаимодействуют через разделяемые переменные
  - Разделение определяется синтаксически
  - Любая переменная, видимая более чем одной нити, является разделяемой
  - Любая переменная, видимая только одной нити, является private
- Возможны Race conditions
  - Требуется синхронизация для предотвращения кофликтов
  - Возможно управление доступом (shared, private) для минимизации накладных расходов на синхронизацию

Синтаксис директив OpenMP

#pragma omp directive_name [clause[clause ...]] newline
<структурный блок >

Действия, соответствующие директиве, применяются непосредственно к структурному блоку, расположенному за директивой. Структурным блоком может быть любой оператор, имеющий единственный вход и единственный выход.

Понятие параллельной области.

Пример структурного блока и параллельной области

```
# pragma omp parallel
{
    id = omp_thread_num();
    res(id) = lots_of_work(id);
}
……
int lots_of_work(int tid)
{
    ……
}
```

Директива **parallel**

Инициация параллельного выполнения структурного блока программы.

#pragma omp parallel [clause [[,]...]
<stuctured block>

clause: **if** (scalar_expression)
private (list)
shared (list)
default (shared | none)
firstprivate (list)
reduction (operator: list)
copyin (list)
num_threads (integer_expression)

Опции директивы **parallel**
опция private

Опция **private (<*список*>)**

По умолчанию переменные, видимые в области, объемлющей блок параллельного исполнения, являются общими (**shared**).
Переменные, объявленные внутри блока по умолчанию считаются закрытыми (**private**).
Опция **private** задает список закрытых переменных.

Только shared-переменные в объемлющей параллельном блоке могут быть аргументами опции **private**

Опция **firstprivate**

Опция **firstprivate** обладает той же семантикой, что и опция **private**. При этом, все копии переменной инициализируются значением исходной переменной до входа в блок.

Опция **lastprivate**

Опция **lastprivate** обладает той же семантикой, что и опция **private**. При этом, значение переменной после завершения блока параллельного исполнения определяется как ее значение на последней итерации цикла или в последней секции для work-sharing конструкций.

Опция **default**

Опция **default** задает опцию по-умолчанию для переменных. Пример:

#pragma omp parallel default(private)

Опция **shared**

Опция **shared** задает список общих переменных.

#pragma omp parallel default(private) shared(x)

опция **reduction**

Опция **reduction** определяет, что на выходе из параллельного блока переменная получит «редукционное» (выполненное по всем нитям) значение. Пример:

#pragma omp for reduction(+ : x)

Допустимы следующие операции: **+, *, -, &, |, ^, &&, ||**

Директива threadprivate

#omp threadprivate (список **глобальных** переменных)

переменные становятся общими для всех тредов:

```
static int a;

f() {
   printf(“%d\n”, a);
}

main() {
#omp threadprivate(a)

#omp parallel
  {
  a = omp_num_thread();
  f();
  }
}
```

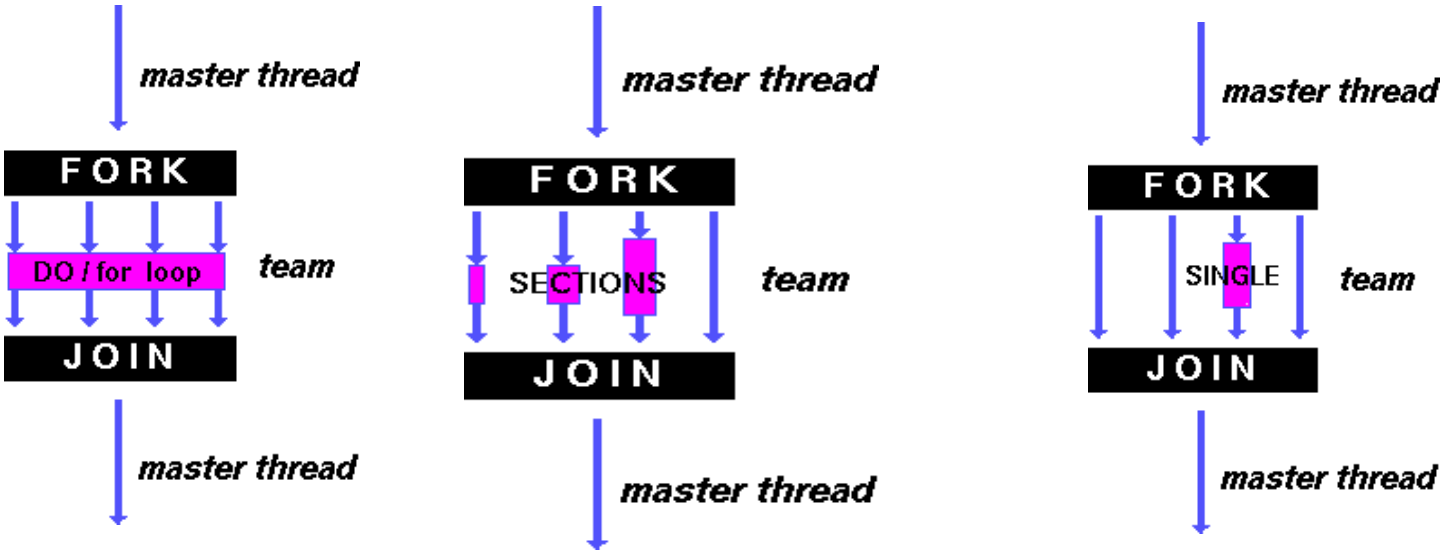
Опция copyin

Опция **copyin** директивы **parallel** определяет порядок инициализации **threadprivate**-переменных: эти переменные инициализируются их значениями в **master**-нити в начале выполнения структурного блока.

Директивы управления распределением вычислений по нитям (Work-Sharing)

Распределение вычислений по нитям:

for **sections** **single**



Директива **for**

#pragma omp for [clause ...]

clause:

schedule (type [,chunk_size])
ordered
private (list)
firstprivate (list)
lastprivate (list)
reduction (operator: list)
nowait

Директива for

Директива предшествует циклу **for** канонического типа:

for(init-expr ; var logical-op b; incr-expr)

init_expr var = expr

logical_op >, <, >=, <=

Incr-expr ++ - += -=

incr_expr ::=	var ++
	++ var
	var --
	-- var
	var += incr
	var -= incr
	var = incr + var
	var = var + incr
	var = var – incr
var	переменная целого типа
incr, lb, b	инварианты цикла целого типа

Пример

```
#include <omp.h>
#define CHUNKSIZE 100
#define N 1000
main ()
 {
   int i, chunk;
   float a[N], b[N], c[N];
   /* Some initializations */
    for (i=0; i < N; i++) a[i] = b[i] = i * 1.0;
    chunk = CHUNKSIZE;
#pragma omp parallel shared(a,b,c,chunk) private(i)
  { #pragma omp for
       for (i=0; i < N; i++){
          c[i] = a[i] + b[i];
           printf (“Thread %d execute loop iteration %d \n”,
           omp_get_thread_num(),i);
     }
   } /* end of parallel section */
 }
```

Пример

```
#include <omp.h>
#define CHUNKSIZE 100
#define N 1000
main ()
 {
   int i, chunk;
   float a[N], b[N], c[N];
   /* Some initializations */
    for (i=0; i < N; i++) a[i] = b[i] = i * 1.0;
    chunk = CHUNKSIZE;
#pragma omp parallel shared(a,b,c,chunk) private(i)
  { #pragma omp for schedule(dynamic,chunk) nowait
     for (i=0; i < N; i++){
          c[i] = a[i] + b[i];
           printf (“Thread %d execute loop iteration %d \n”,
           omp_get_thread_num(),i);
     }
   } /* end of parallel section */
 }
```

Опция **shedule** директивы **for**

Опция **shedule** допускает следующие аргументы:

static - распределение осуществляется статически;

dynamic - распределение осуществляется динамически (нить, закончившая выполнение, получает новую порцию итераций);

guided - аналогично dynamic, но на каждой следующей итерации размер распределяемого блока итераций равен примерно общему числу оставшихся итераций, деленному на число исполняемых нитей, если это число больше заданного значения chunk, или значению chunk в противном случае (крупнее порция – меньше синхронизаций)

runtime - распределение осуществляется во время выполнения системой поддержки времени выполнения (параметр chunk не задается) на основе переменных среды

Особенности опции **schedule** директивы **for**

- аргумент chunk можно использовать только вместе с типами static, dynamic, guided
- по умолчанию chunk считается равным 1
- распараллеливание с помощью опции runtime осуществляется используя значение переменной OMP_SCHEDULE

Пример.

```
setenv  OMP_SCHEDULE  “guided,4”
```

```
#include <omp.h>
#define CHUNK   10
#define N       100
main ()  {
int nthreads, tid, i, n, chunk;
float a[N], b[N], c[N];
for (i=0; i < N; i++)
  a[i] = b[i] = i * 1.0;
 n = N;
 chunk = CHUNK;
```

```
#pragma omp parallel shared(a,b,c,n,chunk) \
private(i,nthreads,tid)
  {
        tid = omp_get_thread_num();
#pragma omp for schedule(dynamic,chunk)
        for (i=0; i < n; i++){
             c[i] = a[i] + b[i];
             printf("tid= %d i= %d  c[i]= %f\n", tid, i, c[i]);
        }

        if (tid == 0){
             nthreads = omp_get_num_threads();
             printf("Number of threads = %d\n", nthreads);
        }
  }
}
```

Директива sections

```
#pragma omp sections [clause ...]
          structured_block
```

```
clause:                 private (list)
                        firstprivate (list)
                        lastprivate (list)
                        reduction (operator: list)
                        nowait
```

```
{
  #pragma omp section
      structured_block
  #pragma omp section
      structured_block
}
```

```
#include <omp.h>
#define N     50
main ()
{
int i, n, nthreads, tid;
float a[N], b[N], c[N];
for (i=0; i < N; i++)
  a[i] = b[i] = i * 1.0;
n = N;
#pragma omp parallel shared(a,b,c,n) private(i,tid,nthreads)
{
  tid = omp_get_thread_num();
  printf("Thread %d starting...\n",tid);
```

```
#pragma omp sections nowait
   {
   #pragma omp section
     for (i=0; i < n/2; i++)
       {
       c[i] = a[i] + b[i];
       printf("tid= %d i= %d c[i]= %f\n",tid,i,c[i]);
       }
   #pragma omp section
     for (i=n/2; i < n; i++)
       {
       c[i] = a[i] + b[i];
       printf("tid= %d i= %d c[i]= %f\n",tid,i,c[i]);
       }
   }
```

```
  if (tid == 0)
    {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
    }

  }
}
```

Директива **single**

#pragma omp single *[clause ...]*
structured_block

clause:
private *(list)*
firstprivate *(list)*
nowait

Директива single определяет: последующий блок будет выполняться только одной нитью

Пример директива **single**

```
#pragma omp single shared(a,b) private (i)
{
  #pragma omp single
   {  a=0;
      printf("Single construct executed by thread
%d\n",
      omp_get_thread_num());
    }/*Barrier is automatically inserted here  */
  #pragma omp for
    for (i=0; i<n; i++)
    b[i] = a;
}/* end of parallel region  */
```

Только одна нить инициализирует shared переменную a

Комбинированные parallel work-sharing директивы

#pragma omp parallel for *[clause ...]*
structured_block

#pragma omp parallel sections *[clause ...]*
structured_block

#pragma omp single *[clause ...]*
structured_block

Комбинированные директивы реализуются более эффективно по сравнению с аналогичной реализацией в виде:

```
#pragma omp parallel
 {
   #pragma omp  for [clause ...]
         structured_block
 }
```

Директивы синхронизации

- master
- critical
- barrier
- atomic
- flush
- ordered

#pragma omp master

определяет секцию кода, выполняемого только master-тредом

#pragma omp critical [(name)]

определяет секцию кода, выполняемого только одним тредом в данный момент времени

#pragma omp barrier

определяет секцию кода, выполняемого только одним тредом в данный момент времени

#pragma omp atomic
<expr-stmt>

<expr-stmt> ::==

x *binop* = expr

x ++

++ x

x --

-- x

#paragma omp flush [var-list]

<expr-stmt> ::==

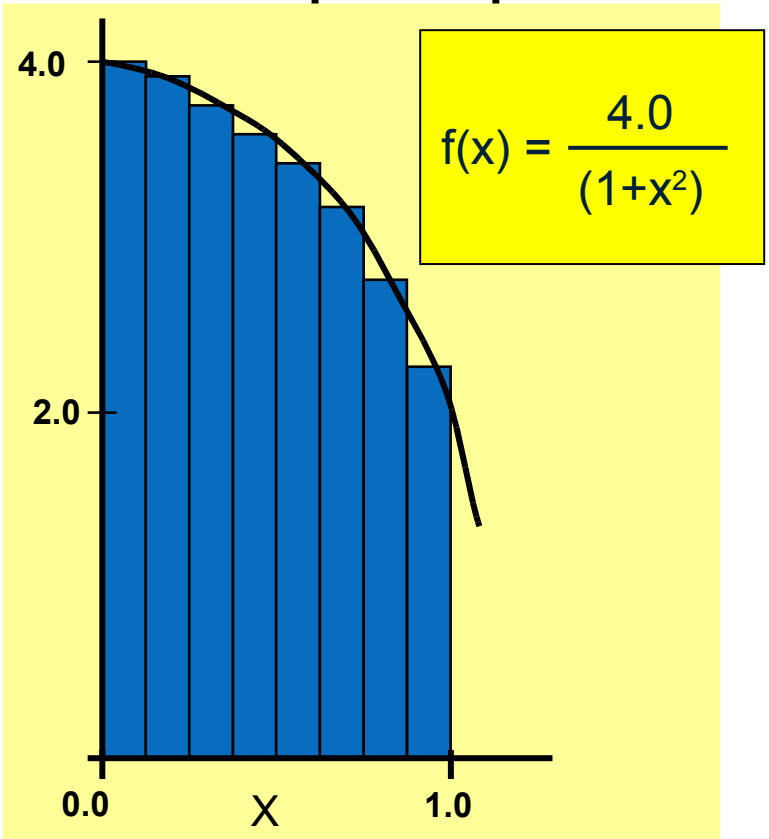
x *binop* = expr
x ++
++ x
x --
-- x

Неявный flush: barrier, вход и выход из critical, ordered, выход из parallel, for, sections, single

Функции OpenMP

- **omp_set_dynamic(int)/omp_get_dynamic()**
  - Динамическое создание нитей
- **omp_set_num_threads(int)/omp_get_num_threads()**
  - Установка числа нитей
  - Вызывается в последовательном участке кода
  - Также может быть установлено, используя **OMP_NUM_THREADS**
- **omp_get_num_procs()**
  - Число доступных процессоров
- **omp_get_thread_num()**
- **omp_set_nested(int)/omp_get_nested()**
  - Установка/проверка вложенного параллелизма
- **omp_in_parallel()**
  - Нить находится в параллельной области?
- **omp_get_wtime()**
  - Определение времени

Пример: численное интегрирование




```
static long num_steps=100000;
double step, pi;

void main()
{  int i;
   double x, sum = 0.0;

   step = 1.0/(double) num_steps;
   for (i=0; i< num_steps; i++){
      x = (i+0.5)*step;
      sum = sum + 4.0/(1.0 + x*x);
   }
   pi = step * sum;
   printf("Pi = %f\n",pi);
}
```

Computing Pi through integration

```
static long num_steps=100000;
double step, pi;

void main()
{  int i;
   double x, sum = 0.0;

   step = 1.0/(double) num_steps;
   for (i=0; i< num_steps; i++){
      x = (i+0.5)*step;
      sum = sum + 4.0/(1.0 + x*x);
   }
   pi = step * sum;
   printf("Pi = %f\n",pi);
}
```

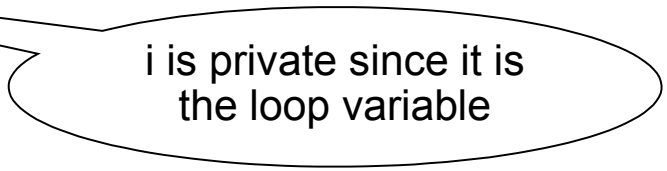
- Какие переменные shared?
- private?
- reductions?

Вычисление Pi

```
static long num_steps=100000;
double step, pi;

void main()
{  int i;
   double x, sum = 0.0;

   step = 1.0/(double) num_steps;
   #pragma omp parallel for \
      private(x) reduction(+:sum)
   for (i=0; i< num_steps; i++){
      x = (i+0.5)*step;
      sum = sum + 4.0/(1.0 + x*x);
   }
   pi = step * sum;
   printf("Pi = %f\n",pi);
}
```



Компиляция OpenMP-программ

Regatta

Опции компилятораOpenMP

- **xlc_r** – C компилятор,
- **xlC_r** – C++ компилятор,
- Помимо опции ***–o*** для компиляторов xlc применимы все стандартные опции (флаги ***-I include_dir, -L library_dir, -l library_name, -c, -g*** - описание см. выше в разделе о MPI).
- Возможно также использование нестандартных ключей компиляторов IBM:
- **-qsmp=omp** обеспечивает возможность писать программы с использованием OpenMP стандарта.
- **-qsmp=auto** для автоматического распараллеливания программ для SMP архитектур**,**

Трансляция OpenMP- программ

Пример:

xlc_r -qsmp=omp -O -qarch=pwr4 -q64 -o my_c_program *.c

Компиляция всех, находящихся в данной директории *.c файлов в исполняемый 64-х разрядный файл с именем my_c_program, оптимизиция его под имеющуюся архитектуру, и учтет все директивы OpenMP.

Запуск на счет OpenMP-программ (Regatta)

ompsubmit [<параметры ompsubmit>] <имя задачи – название исполняемого файла> [<параметры задачи>]

(параметры, заключенные в [] не являются обязательными)

Параметры ompsubmit те же, что и mpisubmit (см. таблицу).

Чтобы запустить hello на восьми процессорах с максимальным временем выполнения, равным 1 минуте, надо выполнить команду:

ompsubmit -w 01:00 -n 4 hello