

Основы OpenMP

Nikita Panov
(nikita.v.panov@intel.com)

OpenMP: Что это?

- **Стандарт, позволяющий распараллеливать вычисления на системах с общей памятью.**
- **Преимущества:**
 - **Простота использования.**
 - **Переносимость.**
 - **Низкие накладные расходы.**
 - **Поддержка параллелизма по данным.**

Compiler directives:

C/C++

`#pragma omp directive [clause, ...]`

Fortran

`!$OMP directive [clause, ...]`

`C$OMP directive [clause, ...]`

`*$OMP directive [clause, ...]`

Параллельное исполнение

Parallel Regions

Основная конструкция OpenMP

`#pragma omp parallel`

```
#pragma omp parallel  
{  
    printf( "hello world from thread %d of %d\n",  
           omp_get_thread_num(),  
           omp_get_num_threads() );  
}
```

Параллельное исполнение

- Итак, большинство конструкций OpenMP – это директивы компилятора
- Основная конструкция OpenMP это `omp parallel [something]`

Serial Program:

```
void main()
{
    double Res[1000];

    for(int i=0;i<1000;i++) {
        do_huge_comp(Res[i]);
    }
}
```

Parallel Program:

```
void main()
{
    double Res[1000];
    #pragma omp parallel for
    for(int i=0;i<1000;i++) {
        do_huge_comp(Res[i]);
    }
}
```

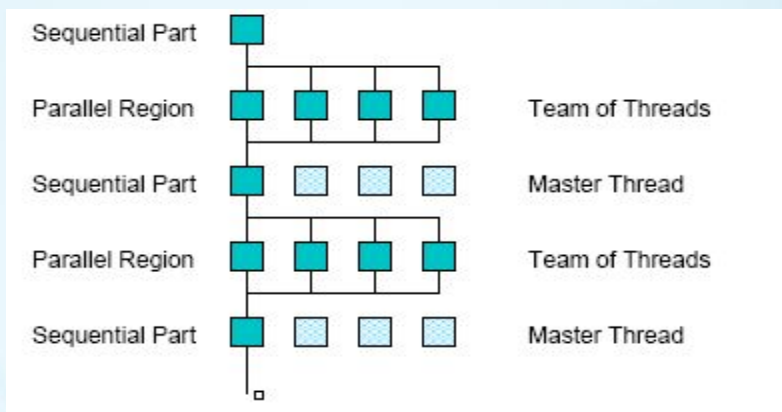
OpenMP подход к параллелизму

- OpenMP основано на модели разделяемой памяти.
- Вычислительная задача распределяется на потоки.
 - Переменные могут быть
 - **общими для всех потоков;**
 - **или каждый поток работает со своей независимой копией.**
- Неосторожное или неправильное использование переменных может привести к неверной работе распараллеленной программы.

OpenMP подход к параллелизму

Модель fork-join

- Исполнение программы начинается в одном потоке (master thread).
- Далее следует конструкция OpenMP
Основной поток создает параллельные потоки.
- После завершения параллельной секции
Ожидается завершение всех параллельных потоков.
- Основной поток продолжает исполнение.



Основные конструкции OpenMP

#pragma omp for

Каждый поток получает свою порцию работы – data parallelism.

#pragma omp section

Каждая такая секция будет исполняться в отдельном потоке – functional parallelism.

#pragma omp single

Последовательное (не параллельное) исполнение участка программы. Только один поток исполнит этот код.

OpenMP sections

```
#pragma omp sections [ clause [ clause ] ... ] new-line  
{  
[#pragma omp section new-line ]  
structured-block1  
[#pragma omp section new-line  
structured-block2 ]  
...  
}
```

OpenMP sections

Functional Parallelism

```
#pragma omp parallel
#pragma omp sections nowait
{
    thread1_work();
    #pragma omp section
    thread2_work();
    #pragma omp section
    thread3_work();
    #pragma omp section
    thread4_work();
}
```

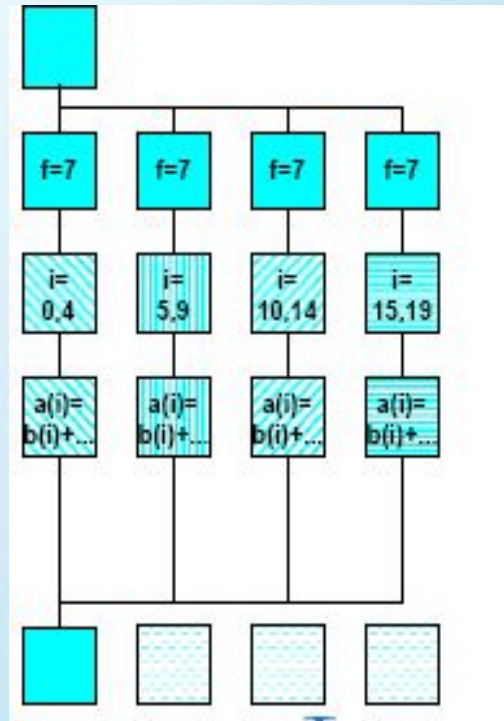
OpenMP for directive

```
#pragma omp for [ clause [ clause ] ...
```

Непосредственно следующий за директивой цикл будет исполнен параллельно (будет разделён по данным между исполнителями).

OpenMP for directive

```
#pragma omp parallel private(f)
{
    f=7;
    #pragma omp for
    for (i=0; i<20; i++)
        a[i] = b[i] + f * (i+1);
} /* omp end parallel */
```



OpenMP for directive

Возможны следующие модификаторы:

- `private(list);`
- `reduction(operator: list);`
- `schedule(type [, chunk]);`
- `nowait` (для `#pragma omp for`).

В конце цикла `for` происходит синхронизация, если не указана директива `nowait`.

`schedule` указывает, как итерации распределятся среди потоков (поведение по умолчанию зависит от реализации).

Переменные в OpenMP

`private (list)`

Будут созданы уникальные локальные копии перечисленных переменных.

`shared (list)`

Потоки разделяют одну переменную.

`firstprivate (list)`

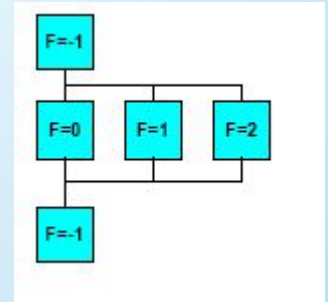
Создаются локальные переменные, инициализированные значением из основного потока.

`lastprivate (list)`

После завершения последней итерации значение переменной копируется в переменную основного потока.

...

По умолчанию все переменные `shared`, кроме локальных переменных в вызываемых функциях и счётчиков цикла.



Пример

```
int x;  
x = 0;           // Initialize x to zero  
  
#pragma omp parallel for firstprivate(x) // Copy value  
                // of x  
                // from master  
for (i = 0; i < 10000; i++) {  
    x = x + i;  
}  
printf( "x is %d\n", x ); // Print out value of x  
  
/* Actually needs lastprivate(x) to copy value back out to master */
```

OpenMP schedule clause

Разбиение итераций цикла по потокам или «диспетчеризация»

```
schedule( type [ , chunk ] )
```

type может принимать следующие значения:

`static`: Статическая диспетчеризация.
Каждый поток получает фиксированное количество итераций.

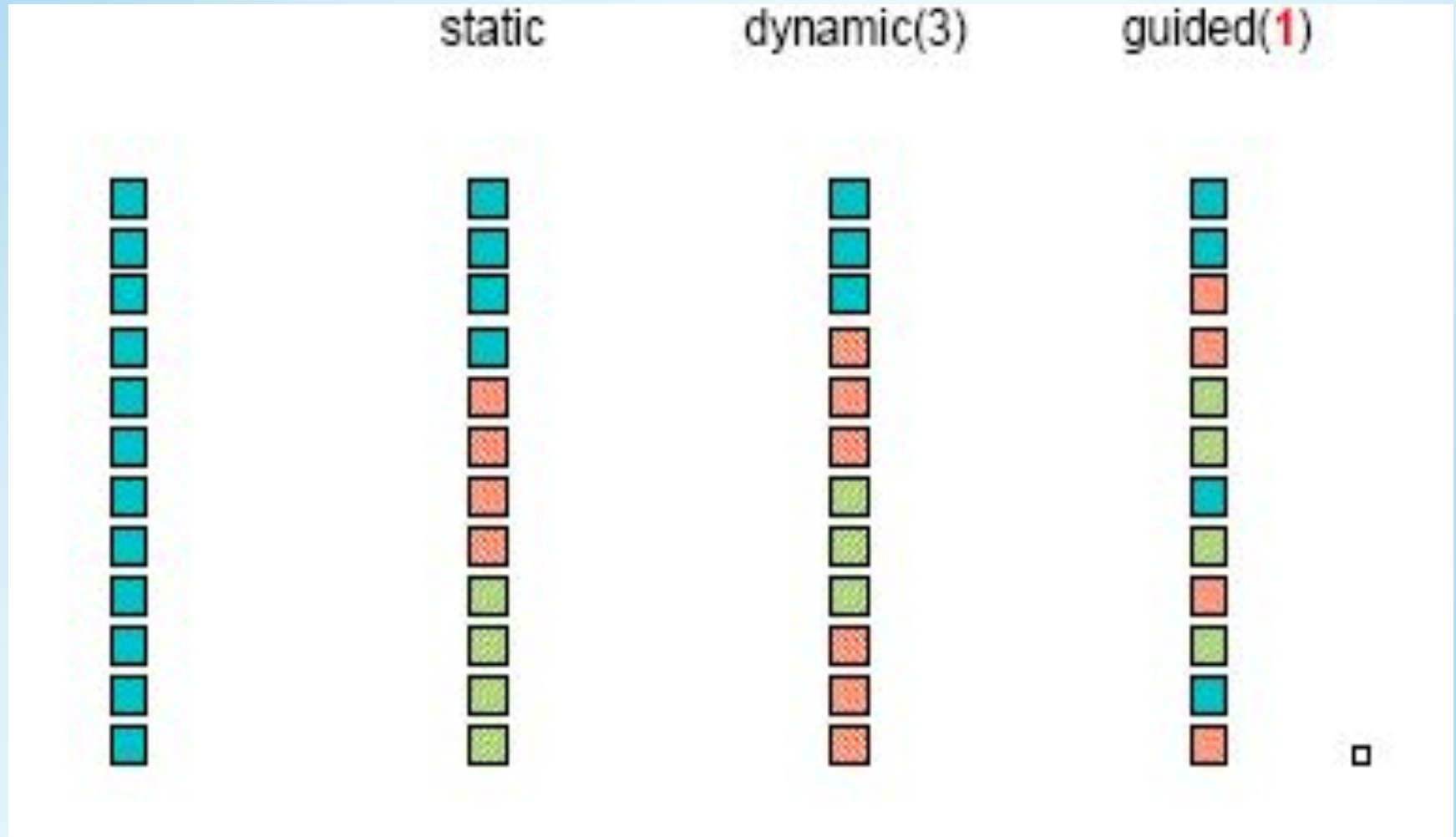
`dynamic`: Динамическая диспетчеризация.

`guided`: Фактически, постепенно
уменьшающееся количество итераций.

`runtime`: Тип диспетчеризации
определяется в момент исполнения, в
зависимости от значения переменных
окружения.

Loop scheduling

(планирование выполнения итераций цикла)



Основные функции OpenMP

```
int omp_get_num_threads(void);
```

```
int omp_get_thread_num(void);
```

...

<http://www.openmp.org/>

Синхронизация в OpenMP

Неявная синхронизация происходит в начале и конце любой конструкции *parallel* (до тех пор, пока не указана директива *nowait*).

Синхронизация в OpenMP

`critical` – критическая секция. Не может исполняться одновременно несколькими потоками.

`atomic` – специальная версия критической секции для атомарных операций (таких, как запись переменной).

`barrier` – точка синхронизации всех потоков.

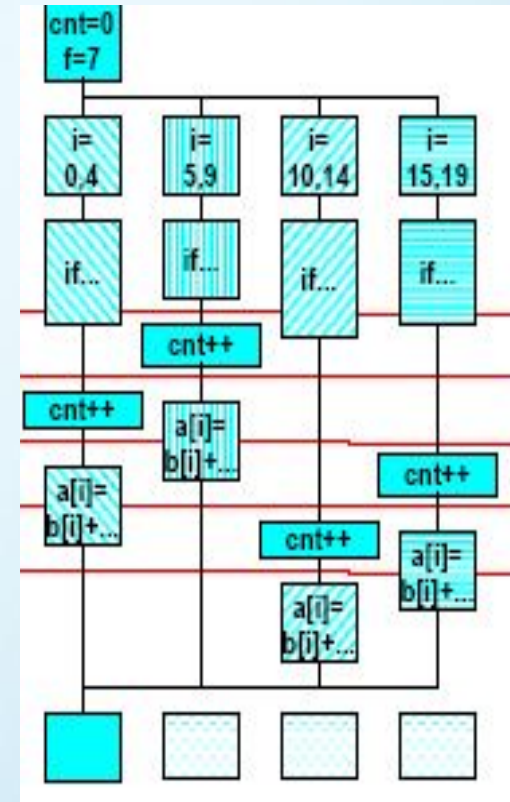
`ordered` – последовательное исполнение.

`master` – только основной поток исполняет следующий код.

...

OpenMP critical

```
cnt = 0;
f=7;
#pragma omp parallel
{
  #pragma omp for
  for (i=0; i<20; i++) {
    if (b[i] == 0) {
      #pragma omp critical
      cnt ++;
    } /* endif */
    a[i] = b[i] + f * (i+1);
  } /* end for */
} /*omp end parallel */
```



Полная информация

OpenMP Homepage: <http://www.openmp.org/>

Introduction to OpenMP - tutorial from WOMPEI 2000 [\(link\)](#)

Writing and Tuning OpenMP Programs on Distributed Shared Memory Machines
[\(link\)](#)

R.Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, R. Menon:

Parallel programming in OpenMP.

Academic Press, San Diego, USA, 2000, ISBN 1-55860-671-8

R. Eigenmann, Michael J. Voss (Eds):

OpenMP Shared Memory Parallel Programming.

Springer LNCS 2104, Berlin, 2001, ISBN 3-540-42346-X

Спасибо за внимание!