Средства и системы параллельного программирования

#6. Task-based подход в OpenMP

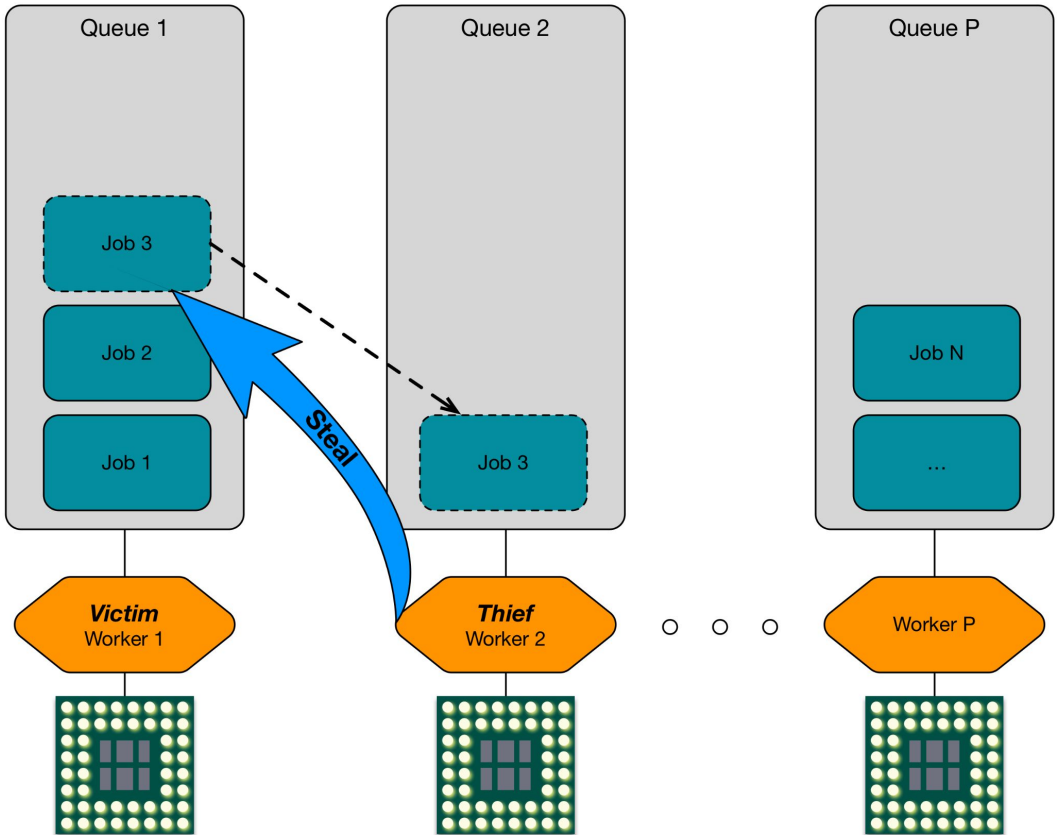
Sections (worksharing construct)

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

Возможность создать отдельные блоки кода, выполняющимися потоками в параллельной области.

Распределение секций по потокам определено реализацией и, по большому счёту, случайно.

Work stealing



Сложные случаи для распараллеливания

```
while ( <expr> ) {
    ...
}
```

```
void myfunc( <args> )
{
    ...
    myfunc( <newargs> );
    ...
}
```

Tasks

#pragma omp task [clause[[,] clause] ...] new-line

    structured-block

Поток, создающий task, может выполнить код в блоке самостоятельно, а может передать выполнение кода другому потоку.

Полезные clause:

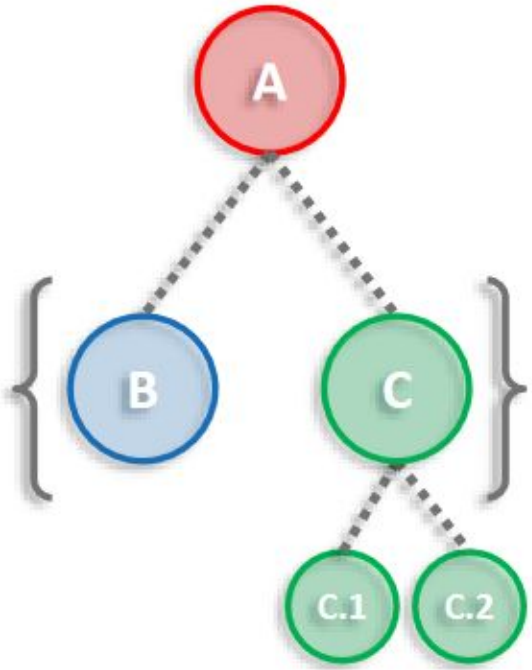
if([task :] scalar-expression) - позволяет заставить поток тут же выполнить блок (т.н. undeferred task) при scalar-expression, сводящемуся к false.

private(list), firstprivate(list), shared(list)

Task synchronization

```
#pragma omp taskwait
```

директива для выполнения ожидания всех дочерних task. Работает только для детей (первого поколения), не работает для task вложенности больше 1.



#pragma omp barrier

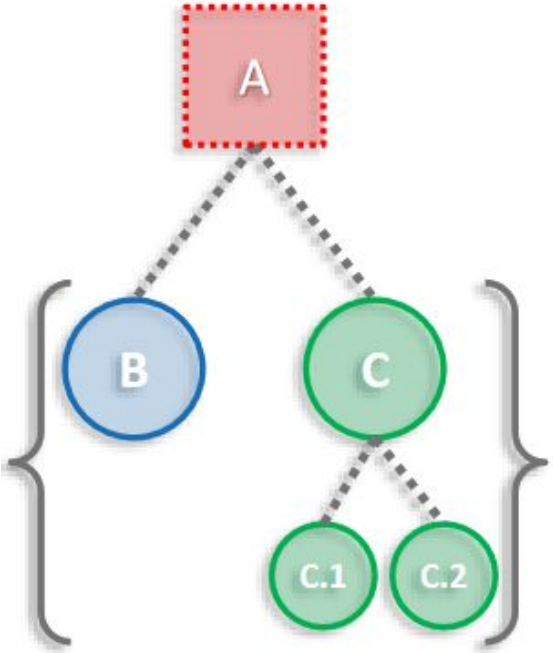
внутри parallel блока, если мы не находимся в work-sharing блоке, барьер может быть использован для ожидания выполнения всех task.

Task synchronization

#pragma omp taskgroup [clause[[,] clause]...]

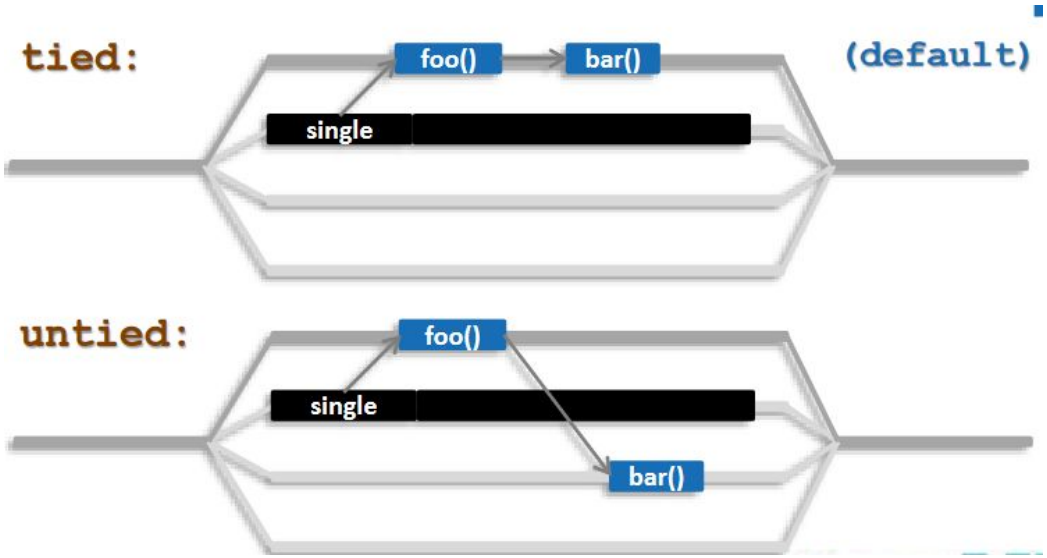
{structured-block}

Директива taskgroup позволяет создать группу из task и контролировать выполнение всех task, порождённых в ней (в том числе сколь угодно вложенных)



Tied vs untied tasks

```
#pragma omp parallel
#pragma omp single
{
    #pragma omp task untied
    {
    foo();
    #pragma omp taskyield
    bar();
    }
}
```



taskloop

#pragma omp taskloop [clause[[,] clause] ...] new-line

for-loops

Директива, позволяющая в цикле создавать наборы task

Полезные clause:

`grainsize(grain-size)` - число итераций для каждого task

`num_tasks(num-tasks)` - число task, по которым раскидались итерации

Depend

`depend([depend-modifier,]dependence-type : locator-list)`

Позволяет выстроить “граф” из task исходя из зависимостей по переменным

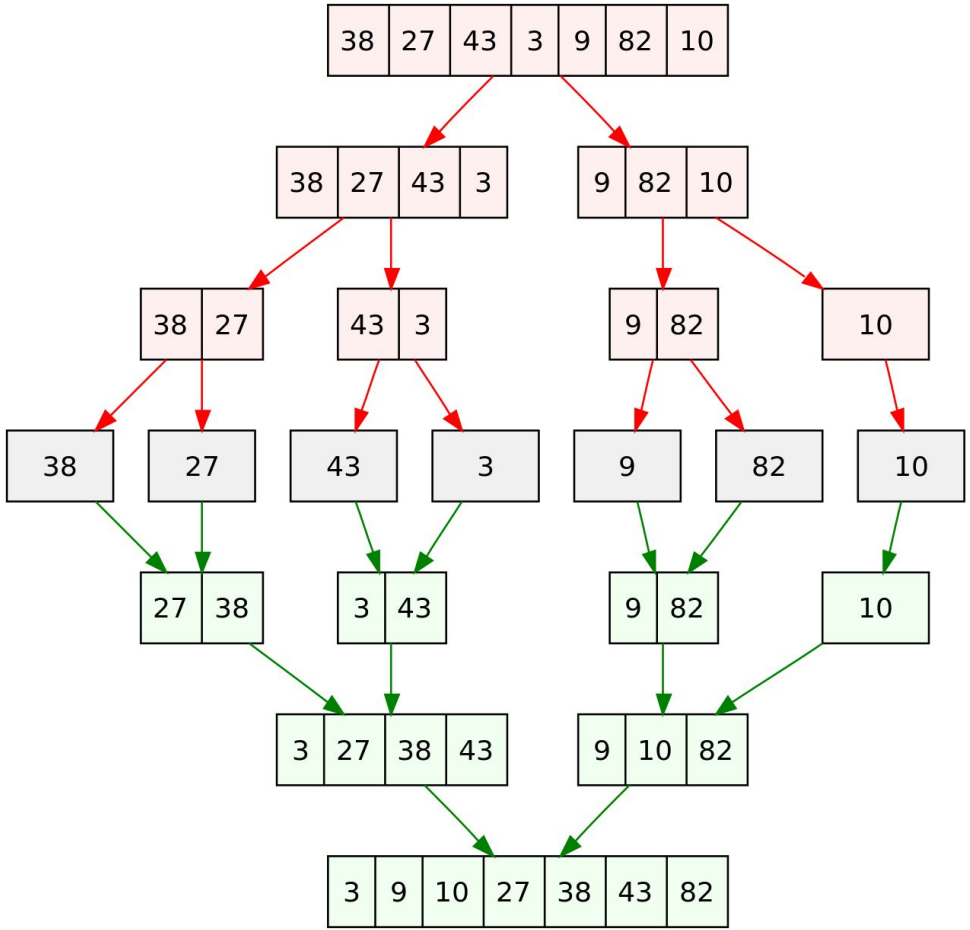
Возможные значения для dependence-type:

in Input variable(s)

out Output variable(s)

inout Input and output variable(s)

Задание


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

Задание

Дан массив A из N целых чисел в случайном порядке. Нужно упорядочить массив A в порядке возрастания. Алгоритм состоит из трёх шагов:

- разбить массив a на куски (чанки)
- Отсортировать каждый чанк массива (любым алгоритмом сортировки). Чанки следует сортировать параллельно друг относительно друга
- Слить чанки в единый упорядоченный массив, используя параллельный алгоритм слияния

Задание - реализовать параллельную сортировку слиянием **с помощью OpenMP tasks** (и, возможно, sections).

Задание (продолжение)

Требования к коду и отчёту:

- Программа должна принимать на вход целочисленные N, p (число потоков)
- Составить график зависимости T(p), S(p), E(p) при фиксированном большом N. На графике T(p) отразить время работы qsort() из stdlib.h
- **При N>1000000 многопоточный вариант сортировки должен работать не медленнее qsort() из stdlib.h (макс. допускается 105% от времени qsort())**

Дедлайн: 4.11, 11.11