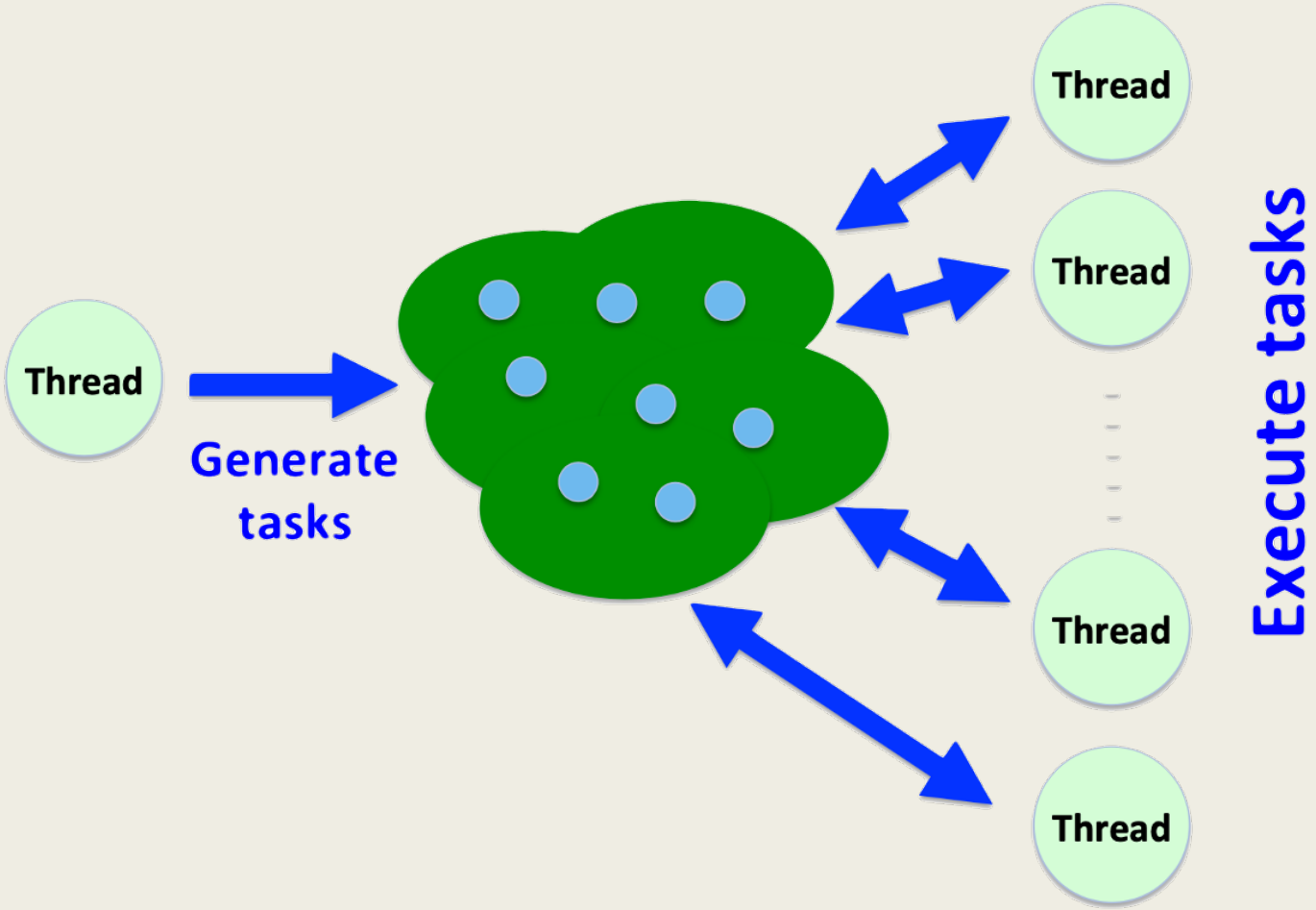
Проблемы OpenMP 2.x.x

- Общие механизмы работают эффективно в большинстве случаев...
- но хочется большего контроля, например: циклы с конечным числом итераций в `runtime`, конечное число параллельных секций...
- Такой подход не работает в некоторых случаях, например: связанные списки, рекурсивные алгоритмы и т.п.
- К тому же, если решение и находилось, оно хоть и было приемлемым, но было буквально уродливым

Расширение OpenMP 3.0 - задачи

- До появления задач определенные типы параллелизма было невозможно реализовать
- Основная идея: разделение общей задачи на подзадачи, т.к. тем самым улучшается читаемость и приобретается новый опыт программирования
- Задачи могут быть, и часто будут, вложенными *(не для слабонервных!)*

Расширение OpenMP 3.0 - задачи



Общие механизмы OpenMP. Задачи

```
#pragma omp task
[shared|private|firstprivate|default(shared|none)]
```

- Любые созданные задачи могут быть исполнены независимо
- Когда любой поток/задача обнаруживают определение задачи, она немедленно генерируется
- Момент начала исполнения задачи зависит от системы выполнения
- Выполнение может быть немедленным или отложенным
- Завершение может быть форсировано механизмами синхронизации
- Данные и код упаковываются в область задачи

Общие механизмы OpenMP. Задачи – синхронизация

Ожидание завершения всех задач на любых потоках:

```
#pragma omp barrier
```

Ожидание завершения всех дочерних (child) задач,

не потомков (descendants)!:

```
#pragma omp taskwait
```

Общие механизмы OpenMP. Задачи – пример

Выводить на экран «A car race» или «A race car»

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

int main(int argc, char *argv[]) {
  printf("A ");
  printf("race ");
  printf("car ");
  printf("\n");
  return(0);
}
```

Общие механизмы OpenMP. Задачи – пример

bash

```
$ gcc hello.c
$ ./a.out
A race car
$
```

Общие механизмы OpenMP. Задачи – пример

Выводить на экран «A car race» или «A race car»

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

int main(int argc, char *argv[]) {
  #pragma omp parallel
  {
    printf("A ");
    printf("race ");
    printf("car ");
  }
  printf("\n");
  return(0);
}
```

Общие механизмы OpenMP. Задачи – пример

bash

```
$ gcc -fopenmp hello.c
$ OMP_NUM_THREADS=2 ./a.out
A race car A race car
$ OMP_NUM_THREADS=2 ./a.out
A A race race car car
$ OMP_NUM_THREADS=2 ./a.out
A race A car race car
$ OMP_NUM_THREADS=2 ./a.out
A race A race car car
$
```

Общие механизмы OpenMP. Задачи – пример

Выводить на экран «A car race» или «A race car»

```
int main(int argc, char *argv[]) {
  #pragma omp parallel
    #pragma omp single
    {
      printf("A ");
      printf("race ");
      printf("car ");
    }
  printf("\n");
  return(0);
}
```

Общие механизмы OpenMP. Задачи – пример

bash

```
$ gcc -fopenmp hello.c
$ OMP_NUM_THREADS=2 ./a.out
A race car
$
```

Общие механизмы OpenMP. Задачи – пример

Выводить на экран «A car race» или «A race car»

```
int main(int argc, char *argv[]) {
  #pragma omp parallel
    #pragma omp single
    {
      printf("A ");
      #pragma omp task
        printf("race ");
      #pragma omp task
        printf("car ");
    }
  printf("\n");
  return(0);
}
```

Общие механизмы OpenMP. Задачи – пример

bash

```
$ gcc -fopenmp hello.c
$ OMP_NUM_THREADS=2 ./a.out
A race car
$ OMP_NUM_THREADS=2 ./a.out
A race car
$ OMP_NUM_THREADS=2 ./a.out
A car race
$
```

Общие механизмы OpenMP. Задачи – пример

... добавить в конце фразу «is fun to watch»

```
int main(int argc, char *argv[]) {
  #pragma omp parallel
    #pragma omp single
    {
      printf("A ");
      #pragma omp task
        printf("race ");
      #pragma omp task
        printf("car ");
      printf(“is fun to watch “);
    }
  printf("\n");
  return(0);
}
```

Общие механизмы OpenMP. Задачи – пример

bash

```
$ gcc -fopenmp hello.c
$ OMP_NUM_THREADS=2 ./a.out
A is fun to watch race car
$ OMP_NUM_THREADS=2 ./a.out
A is fun to watch race car
$ OMP_NUM_THREADS=2 ./a.out
A is fun to watch car race
$
```

Общие механизмы OpenMP. Задачи – пример

… добавить в конце фразу «is fun to watch»

```
int main(int argc, char *argv[]) {
  #pragma omp parallel
    #pragma omp single
    {
      printf("A ");
      #pragma omp task
        printf("race ");
      #pragma omp task
        printf("car ");
      #pragma omp taskwait
        printf(“is fun to watch “);
    }
  printf("\n");
  return(0);
}
```

Общие механизмы OpenMP. Задачи – пример

bash

```
$ gcc -fopenmp hello.c
$ OMP_NUM_THREADS=2 ./a.out
A race car is fun to watch
$ OMP_NUM_THREADS=2 ./a.out
A race car is fun to watch
$ OMP_NUM_THREADS=2 ./a.out
A car race is fun to watch
$
```

Общие механизмы OpenMP. Задачи – пример №2

Работа со связанным списком

```
while(my_pointer) {
    (void) do_independent_work (my_pointer);
    my_pointer = my_pointer->next;
} // End of while loop
```

- До введения задач реализация была осложнена:
  - *Посчитать общее кол-во итераций*
  - *Заменить while loop на for loop для применения директивы*

Общие механизмы OpenMP. Задачи – пример №2

Работа со связанным списком

```
my_pointer = listhead;
#pragma omp parallel
{
  #pragma omp single
  {
    while(my_pointer) {
      #pragma omp task firstprivate(my_pointer)
        (void) do_independent_work(my_pointer);
      my_pointer = my_pointer->next;
    }
  } // End of single
} // End of parallel region
```

Общие механизмы OpenMP. Задачи – пример №2

Работа со связанным списком

```
my_pointer = listhead;
#pragma omp parallel
{
  #pragma omp single nowait
  {
    while(my_pointer) {
      #pragma omp task firstprivate(my_pointer)
        (void) do_independent_work(my_pointer);
      my_pointer = my_pointer->next;
    }
  } // End of single - no implied barrier (nowait)
} // End of parallel region - implied barrier
```

Общие механизмы OpenMP. Задачи – условия

```
#pragma omp task if(clause)
```

- Если скалярное выражение принимает значение false (0):
  - *Текущее задание приостанавливается*
  - *Данное новое задание начинается немедленно*
  - *Родительское задание продолжается, когда новое задание будет завершено*

Полезно применять для оптимизации (напр., создания мелких сервисных подзадач)

Общие механизмы OpenMP. Задачи – расписания

- Каждая задача привязана ***(tied)*** к потоку, который начал ее выполнение – не обязательно поток-создатель.
- Ограничения:
  - *Только поток, к которому привязана задача, может ее исполнять / продолжить исполнять*
  - *Задача может быть приостановлена только через директивы* `taskwait`, `barrier`, `taskyield`
  - Если задача приостановлена с помощью ***`barrier`***, исполняющий поток может переключиться только на прямого потомка (child), привязанного к нему
- Задачи, созданные с условием ***`untied`*** никогда не привязываются к потокам
  - *Приостановленная задача может быть продолжена другим потоком*
  - *Может быть приостановлена в любой момент (балансировка нагрузки)*

Общие механизмы OpenMP. Задачи – расписания

```
#pragma omp taskyield
```

- Означает возможность приостановить текущую задачу, чтобы отдать управление другой задаче при необходимости

Является подсказкой для среды выполнения и/или механизмом предотвращения deadlocks.

Общие механизмы OpenMP. Задачи – расписания

#pragma omp taskyield

```
void something_useful();
void something_critical();

void foo(omp_lock_t * lock, int n) {
  for(int i = 0; i < n; i++)
    #pragma omp task
    {
      something_useful();
      while( !omp_test_lock(lock) ) {
        #pragma omp taskyield
      }
      something_critical();
      omp_unset_lock(lock);
    }
}
```