Возможности новых версий OpenMP
Обзор самого интересного

Предупреждение – описанные возможности есть в стандарте, но далеко не всегда реализованы в конкретной версии компилятора. В частности, на учебном сервере большинство указанных в данной презентации конструкций не работает

Куда движется стандарт и кто его развивает?

OpenMP Architecture Review Board (OpenMP ARB)

Текущие приоритеты – многоязыковой параллелизм высокого уровня, производительность, продуктивность и переносимость.

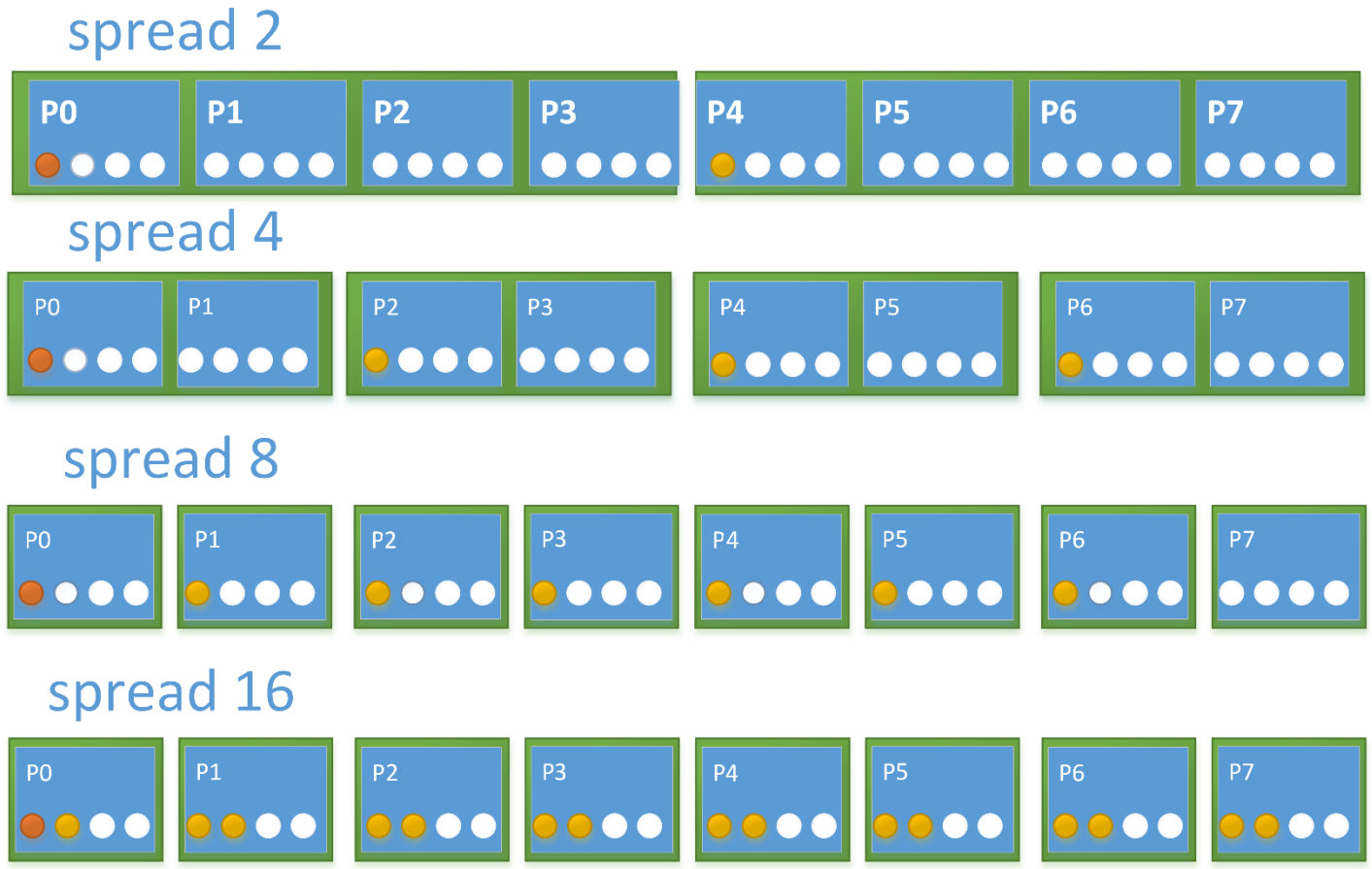
Эволюционные улучшения

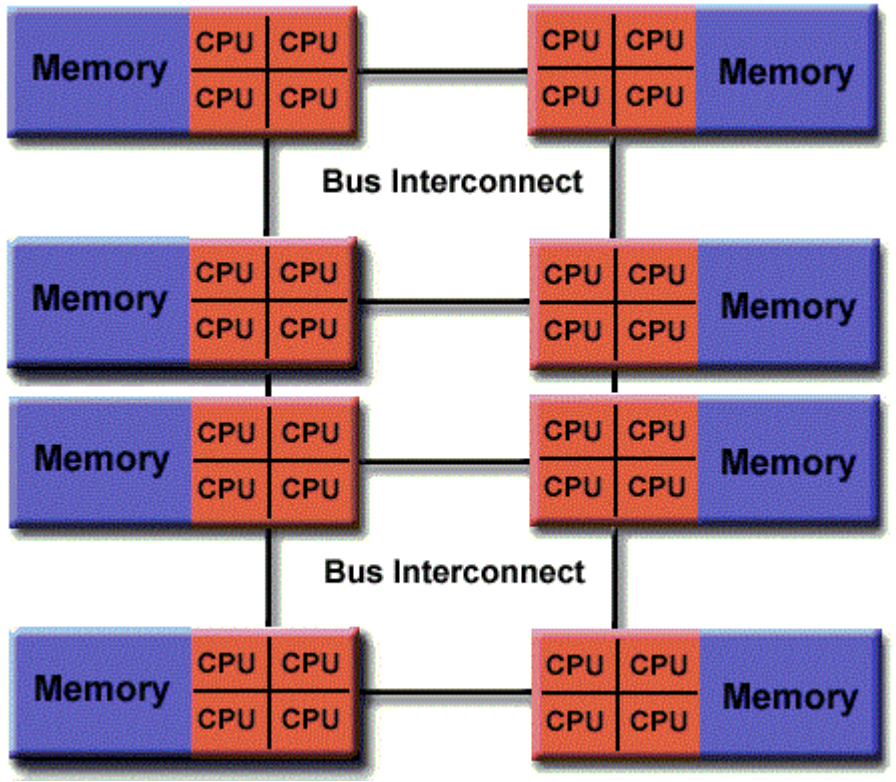
- В стандарте много эволюционных улучшений – поддержка новых версий языков программирования (в первую очередь, последних версий C++ (C++ 17, C++ 20 ожидается), Fortran 2008)
- Дополнение существующих конструкций в соответствии с потребностями программистов (пример – inout для depend, taskyeld, collapse для вложенных циклов, собственные операторы для reduce и др.)
- Адаптация под существующие архитектурные особенности современных вычислительных систем (например – адаптация к тому, что современные системы в массе своей стали ccNUMA - Cache coherent Non-uniform memory access – «архитектура с несимметричным (неравномерным) доступом к памяти с когерентностью кешей» - proc_bind (варианты master, close, spread, также true и false)

Пример proc_bind

export OMP_PROC_BIND= "spread“ //максимально раскидать по процессорам

#pragma omp parallel proc_bind(spread) num_threads(N)



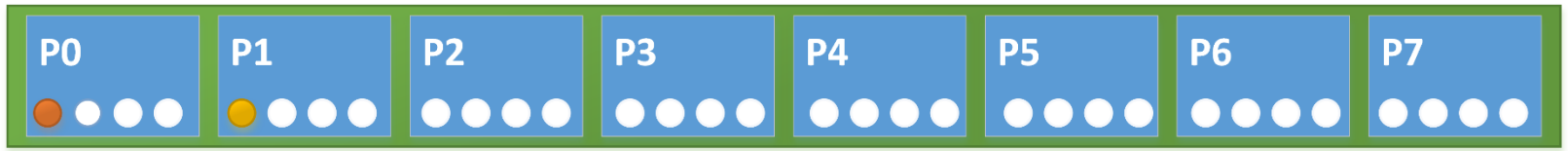


Пример proc_bind

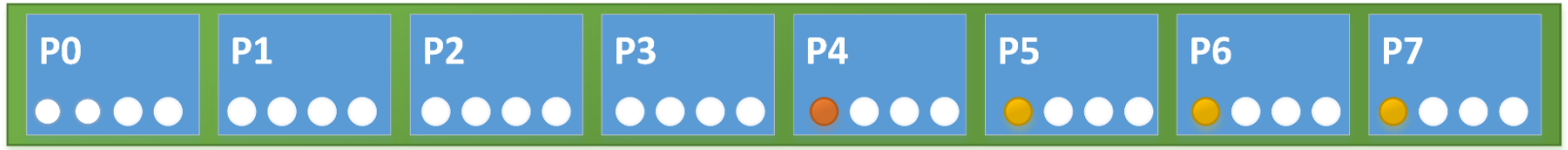
export OMP_PROC_BIND= "close“//поближе друг к другу

#pragma omp parallel proc_bind(close) num_threads(N)

close 2



close 4



close 8

Пример proc_bind

export OMP_PROC_BIND= "master“ //все мастер-потоку

#pragma omp parallel proc_bind(master) num_threads(N)

master 2

master 4

master 8

OMP_PROC_BIND

Глобальная переменная OMP_PROC_BIND (master, close, spread, а также false – все политики размещения игнорируются или true – тогда можно задавать через #pragma omp parallel proc_bind()

Посмотреть конфигурацию – lscpu, cat /proc/cpuinfo

(ОС Linux)

Пример:

```
#pragma omp parallel proc_bind(spread)
{
   #pragma omp parallel proc_bind(close)
   {
//какой-то код, приоритеты: внутренний блок – важна локальная общность данных, внешний – частота и наибольшая пропускная способность памяти
   }


}
```

OMP_PLACES

Глобальная переменная OMP_PLACES (threads, cores, sockets)

Также есть возможность явной нумерации

export OMP_PLACES = threads

Каждый поток закреплён к своему потоку процессора

export OMP_PLACES = cores

Каждый поток закреплён к своему ядру процессора, может мигрировать между потоками своего ядра

export OMP_PLACES = sockets

Каждый поток закреплён к процессору в отдельном сокете, может мигрировать между ядрами и гиперпотоками своего сокета

OMP_PLACES - прямая нумерация

Прямое указание

export OMP_PLACES "{0,1,2,3},{4,5,6,7},{8,9,10,11}, … {28,29,30,31} "

Или

export OMP_PLACES "{0:3}:8:4"

Формат шаблона:

{<границы>}:<число вхождений>:<шаг>

//оба варианта на слайде одинаковы

//через глобальные переменные – удобно конфигурировать

Пример улучшения - collapse

- Когда нужна конструкция – при наличии вложенных циклов, особенно если число итераций по внешнему циклу относительно небольшое, из-за чего страдает балансировка
- Что даёт – позволяет распределять между потоками итерации не только внешнего цикла, но и внутреннего (внутренних) (collapse(n))

```
#pragma omp parallel for collapse(2)
  for (i = 0; i < imax; i++) {
    for (j = 0; j < jmax; j++)
          {// какой-то код
          }
  }
```

Пример улучшения – ordered depend (sink)

```
#pragma omp for ordered (2)
for (int i = 0; i < M; i++)
  for (int j = 0; j < N; j++)
{
  a[i][j] = foo (i, j);
#pragma omp ordered depend (sink: i – 1, j) depend (sink: i, j – 1)
  b[i][j] = boo (a[i][j], b[i – 1][j], b[i][j – 1];
#pragma omp ordered depend (source)
  bzz (a[i][j], b[i – 1][j]);
}
```

Только внешний цикл распараллеливается. ordered depend (sink) блокирует выполнение итераций i, j до тех по, пока в итерациях i-1 и j-1 не достигнет следующей директивы ordered depend (source)

В данном случае проигнорирует, ибо к моменту i,j i, j-1 выполнена.

Принципиально новые возможности стандарта

- Стандарт активно адаптируется под новые архитектурные веяния, в том числе такие, как векторные инструкции, модель устройств (которая, в принципе, расширяется на GPU, Intel MIC и другие, выгрузка вычислений на ускорители называется offload)
- В стандарте появляются новые парадигмы – так как задачи вместо секций и т.д., так как в целом акцент смещается с доработки существующего ПО (в тяжёлых случаях – legacy code) на разработку нового ПО
- Вопрос о распространении стандарта на distributed-memory – стоит, есть множество научных статей на эту тему, однако как будет реализован – вопрос пока открытый. Теоретически можно было бы из существующих конструкций реализовать через модель device – однако, много ограничений – в частности, несимметричность модели.