Директивы OpenACC

Простой пример: SAXPY

```
void saxpy(int n, float a, float *x, float *restrict y)
{
#pragma acc kernels
  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}

...
// Perform SAXPY on 1M elements
saxpy(1<<20, 2.0, x, y);
...
```

Конструкция kernels

Каждый цикл запускается как отдельное ядро на GPU.

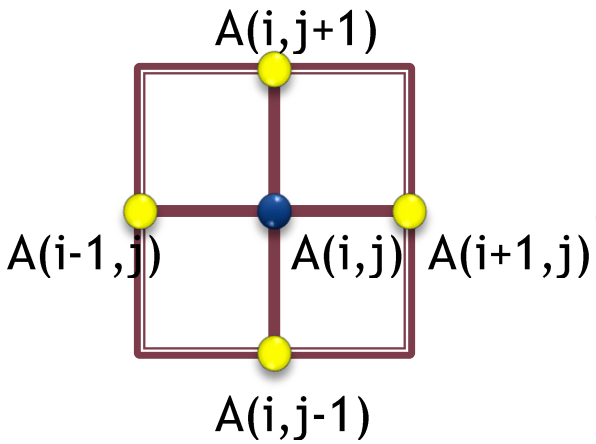
```
#pragma acc kernels
{
  for (int i = 0; i < n; ++i)  {
       a[i] = 0.0
       b[i] = 1.0
       c[i] = 2.0
  }

   for (int i = 0; i < n; ++i)  {
       a(i) = b(i) + c(i)
   }
}
```

kernel 1

kernel 2

Пример 2: метод Якоби

- Итеративно вычисляется среднее значение соседних точек на сетке. Часто используется для решения уравнения Лапласа $\nabla^2 f(x, y) = 0$



$$A_{k+1}(i,j) = \frac{A_k(i-1,j) + A_k(i+1,j) + A_k(i,j-1) + A_k(i,j+1)}{4}$$

Код метода Якоби

```
while ( error > tol && iter < iter_max )
{
  error=0.0;

  for( int j = 1; j < n-1; j++) {
    for(int i = 1; i < m-1; i++) {


      Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                           A[j-1][i] + A[j+1][i]);


      error = max(error, abs(Anew[j][i] - A[j][i]);
    }
  }

  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }

  iter++;
}
```

Повторять, пока не сойдется

Повторять для всех элементов матрицы

Среднее соседних элементов

Вычисление ошибки

Перезапись

Распараллеливание с помощью OpenMP

```
while ( error > tol && iter < iter_max ) {
  error=0.0;

#pragma omp parallel for shared(m, n, Anew, A) reduction(max : error)
  for( int j = 1; j < n-1; j++) {
    for(int i = 1; i < m-1; i++) {

      Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                           A[j-1][i] + A[j+1][i]);

      error = max(error, abs(Anew[j][i] - A[j][i]);
    }
  }

#pragma omp parallel for shared(m, n, Anew, A)
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }

  iter++;
}
```

OpenACC. Первая попытка

```
while ( error > tol && iter < iter_max ) {
  error=0.0;

#pragma acc kernels
  for( int j = 1; j < n-1; j++) {
    for(int i = 1; i < m-1; i++) {

      Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                           A[j-1][i] + A[j+1][i]);

      error = max(error, abs(Anew[j][i] - A[j][i]);
    }
  }

#pragma acc kernels
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }

  iter++;
}
```

Ядро 1

Ядро 2

Производительность

Execution	Time (s)	Speedup
CPU 1 OpenMP thread	69.80	--
CPU 2 OpenMP threads	44.76	1.56x
CPU 4 OpenMP threads	39.59	1.76x
CPU 6 OpenMP threads	39.71	1.76x
OpenACC GPU	162.16	0.24x FAIL

CPU: Intel Xeon X5680
6 Cores @ 3.33GHz

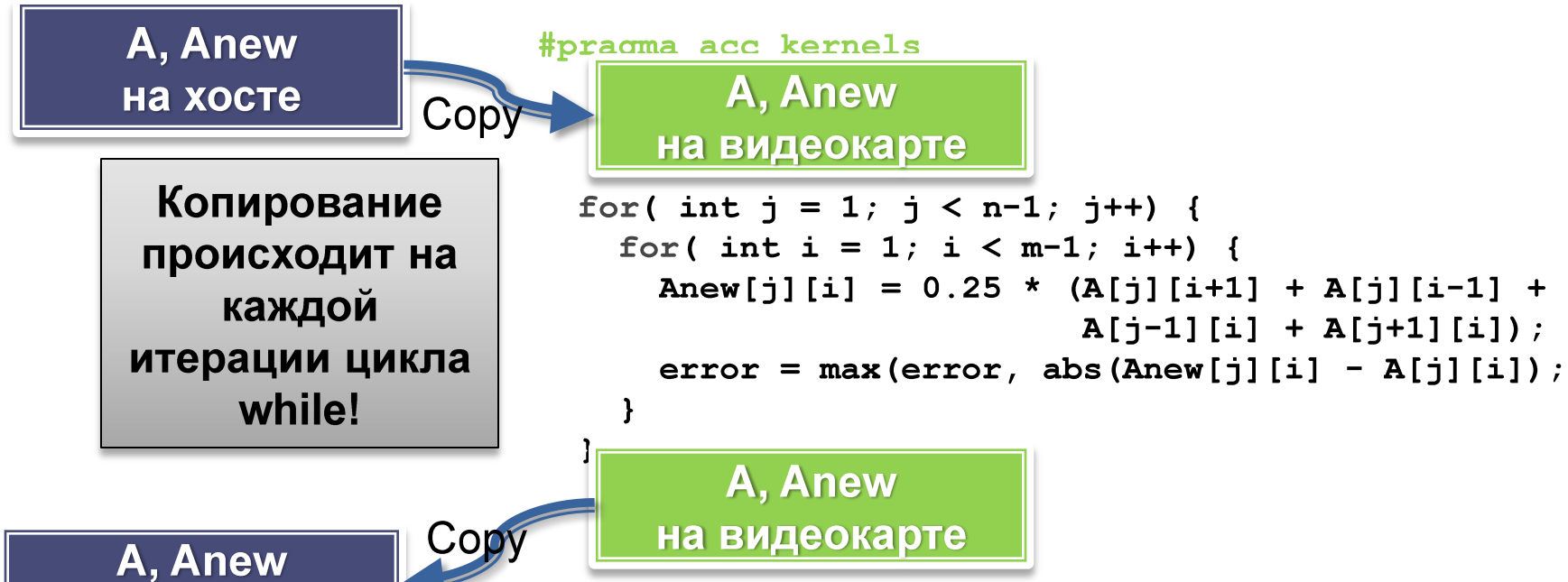
GPU: NVIDIA Tesla M2070

Speedup vs. 1 CPU core

Speedup vs. 6 CPU cores

Причина ошибки: избыточные пересылки данных

```
while ( error > tol && iter < iter_max )
{
  error=0.0;
```




```
  ...
}
```

Конструкция data

```
#pragma acc data [действие …]
```

Действия:

`copy ( list )`	Выделить память на GPU, скопировать массив с хоста перед указанным блоком и скопировать обратно - после
`copyin ( list )`	Выделить память на GPU, скопировать массив с хоста перед указанным блоком.
`copyout ( list )`	Выделить память на GPU, скопировать массив с GPU после указанного блока.
`create ( list )`	Выделить память на GPU.
`present ( list )`	Данные уже есть на GPU.

Размер массива

- Компилятору иногда сложно определить размер массива
  - Тогда нужно явно указать его размер

```
#pragma acc data copyin(a[0:size-1]), copyout(b[s/4:3*s/4])
```

OpenACC. Попытка 2

```
#pragma acc data copy(A), create(Anew)
while ( error > tol && iter < iter_max ) {
  error=0.0;

#pragma acc kernels
  for( int j = 1; j < n-1; j++) {
    for(int i = 1; i < m-1; i++) {

      Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                           A[j-1][i] + A[j+1][i]);

      error = max(error, abs(Anew[j][i] - A[j][i]);
    }
  }

#pragma acc kernels
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }

  iter++;
}
```

Копирование A перед и после цикла while. Выделение памяти для Anew на GPU

Производительность

Execution	Time (s)	Speedup
CPU 1 OpenMP thread	69.80	--
CPU 2 OpenMP threads	44.76	1.56x
CPU 4 OpenMP threads	39.59	1.76x
CPU 6 OpenMP threads	39.71	1.76x
OpenACC GPU	13.65	2.9x

CPU: Intel Xeon X5680
6 Cores @ 3.33GHz

GPU: NVIDIA Tesla M2070

Speedup vs. 1 CPU core

Speedup vs. 6 CPU cores