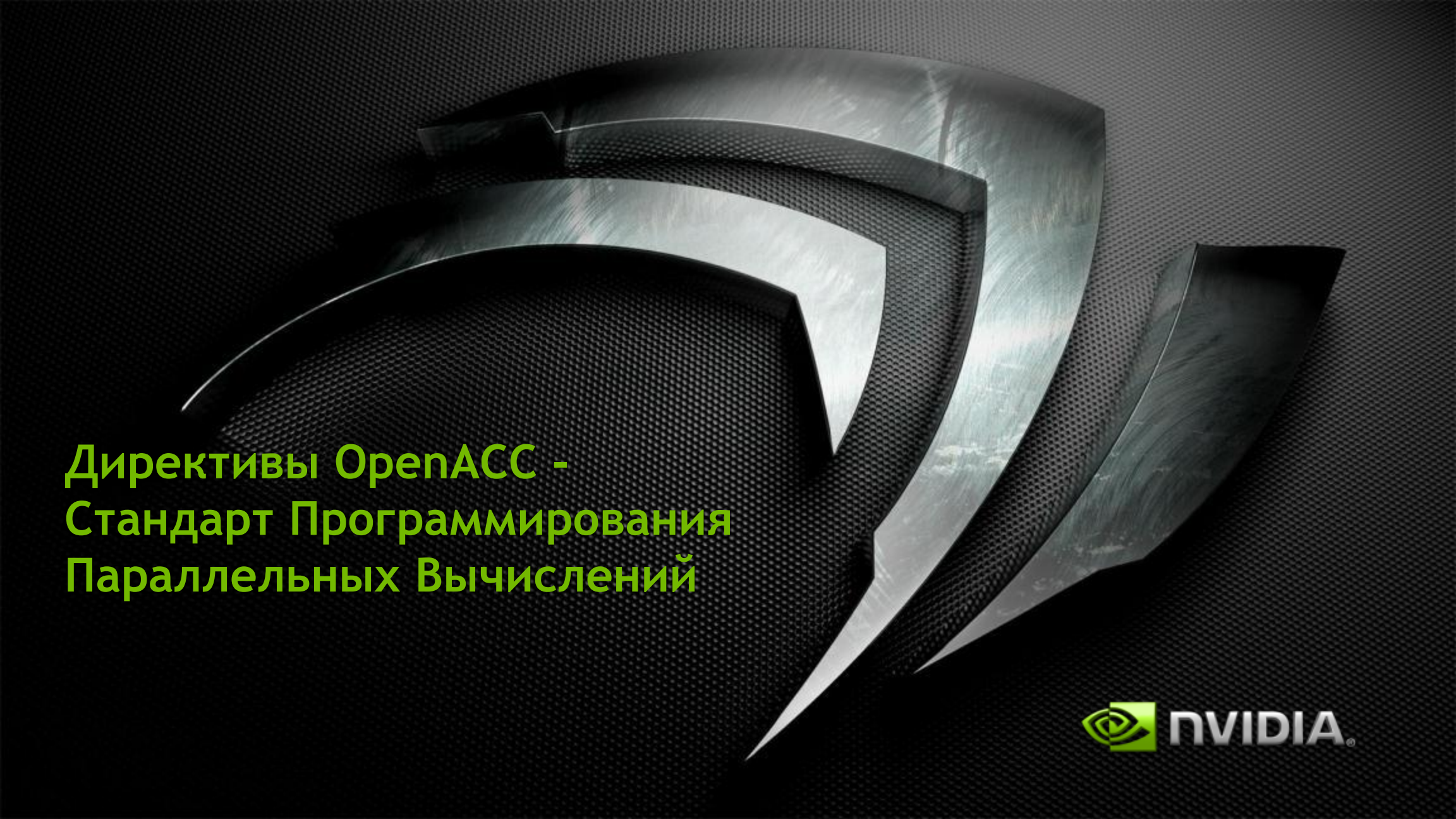
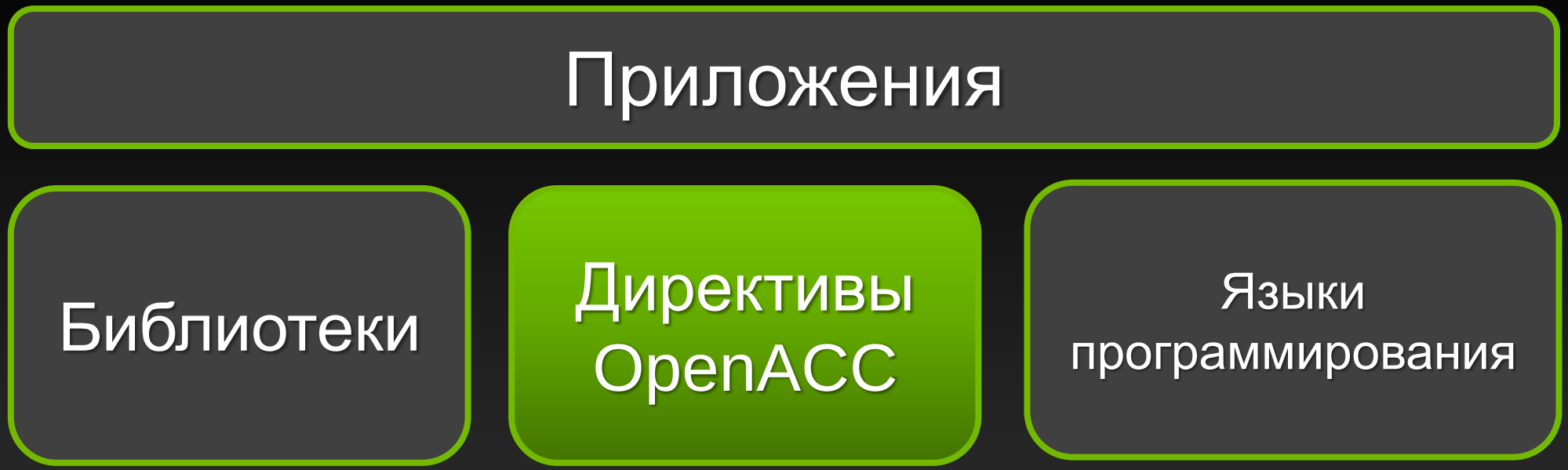
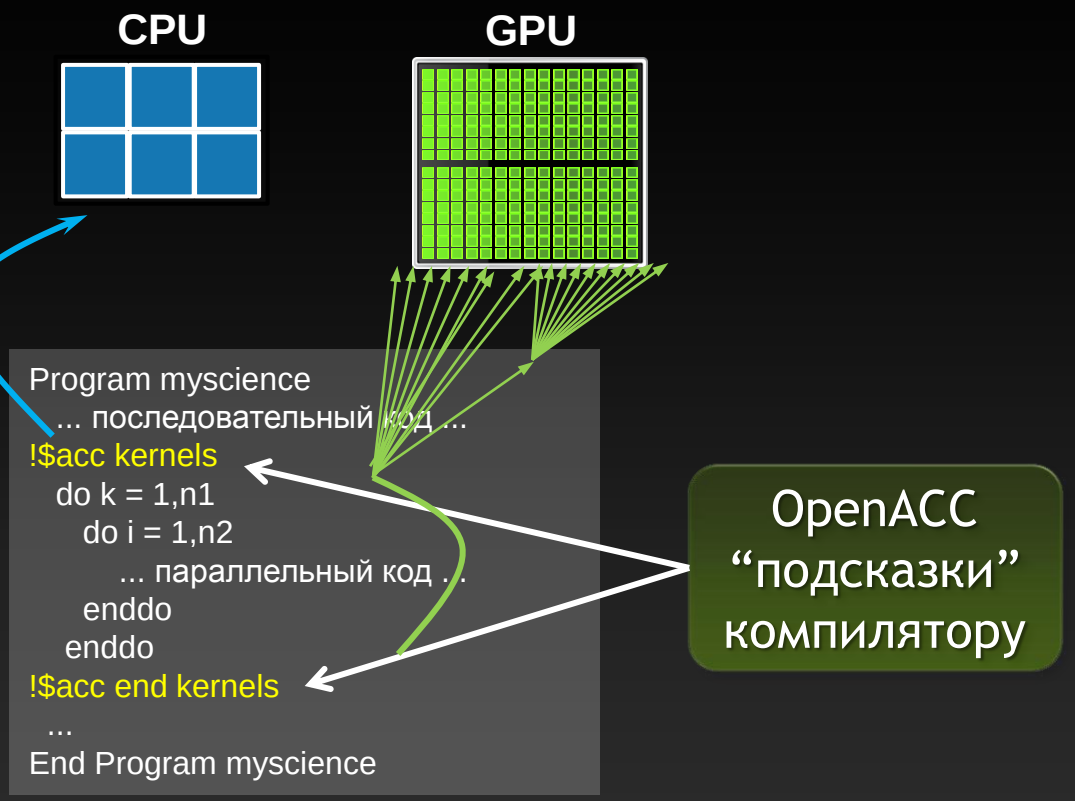
Директивы OpenACC - Стандарт Программирования Параллельных Вычислений



3 способа ускорения приложений



Директивы OpenACC



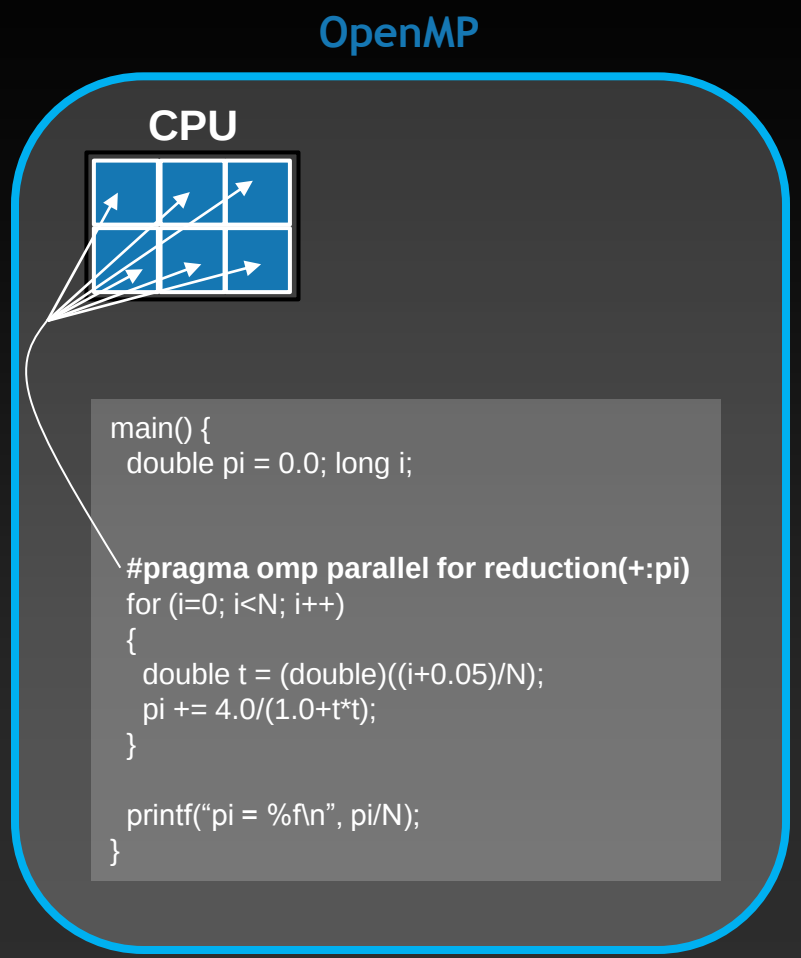
Ваш исходный Fortran или C код

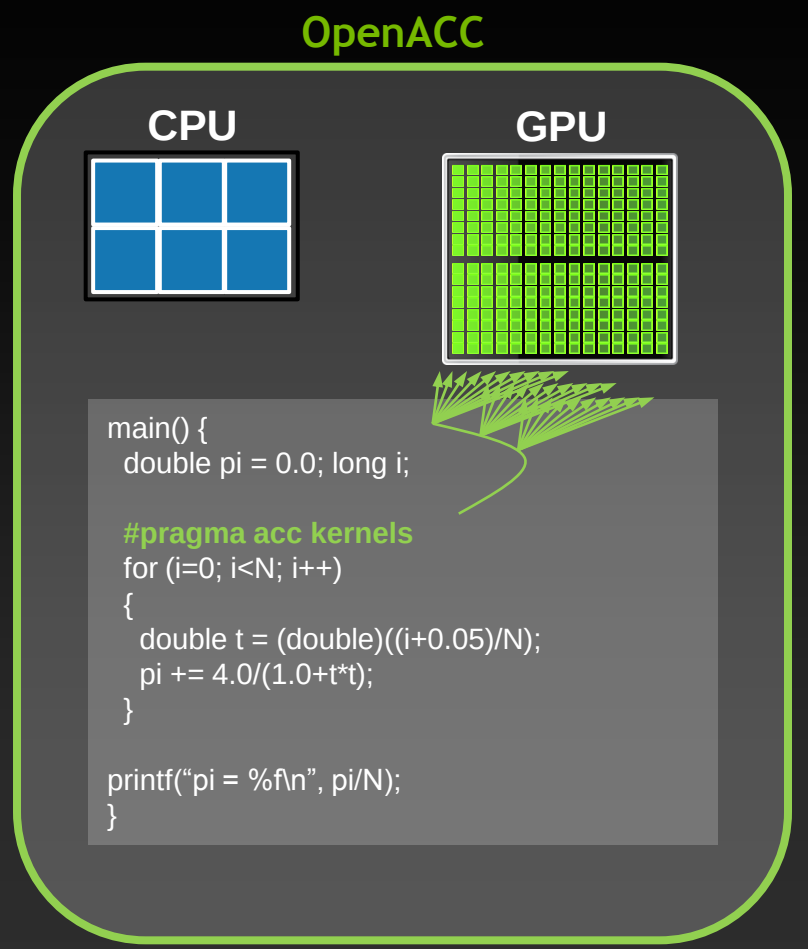
Простые “подсказки” компилятору

Компилятор параллелизует код

Код выполняется на ускорителе

Аналогично OpenMP





OpenACC

Открытый стандарт программирования параллельных вычислений

“OpenACC will enable programmers to easily develop portable applications that maximize the performance and power efficiency benefits of the hybrid CPU/GPU architecture of Titan.”

--Buddy Bland, Titan Project Director, Oak Ridge National Lab

“OpenACC is a technically impressive initiative brought together by members of the OpenMP Working Group on Accelerators, as well as many others. We look forward to releasing a version of this proposal in the next release of OpenMP.”

--Michael Wong, CEO OpenMP Directives Board

Стандарт OpenACC

NVIDIA

CRAY THE SUPERCOMPUTER COMPANY

PGI

CAPS

OpenACC

Программирование GPU с помощью директив

- **Легко:** Директивы – легкий способ ускорения вычислительно емких приложений
- **Открыто:** OpenACC – открытый стандарт, программирование GPU становится портируемым
- **Мощно:** Директивы предоставляют полный доступ к огромной вычислительной мощности GPU

OpenACC®
DIRECTIVES FOR ACCELERATORS

Высокоуровневое программирование, с возможностью низкоуровневого

- Директивы описывают параллельные участки кода на C, C++, Fortran
  - Компиляторы перекладывают выполнение этих участков с хоста на ускоритель
  - Код портируем на различные ОС, центральные процессоры, ускорители и компиляторы
- Создавайте гетерогенные программы
  - Не инициализируя явным образом ускоритель
  - Не осуществляя явное копирование данных или кода между хостом и ускорителем
- Програмная модель позволяет разработчикам начать с простого
  - Улучшать код с помощью дополнительных подсказок компилятору о выполнении циклов, расположении данных и других деталей оптимизации
- Совместимость с языками программирования GPU и библиотеками
  - Интероперабельность между директивами и CUDA C/Fortran, GPU библиотеками (CUFFT, CUBLAS, CUSPARSE, и т.д.)

Директивы: Легко и Мощно

Распознавание объектов в реальном времени

Мировой производитель навигационных систем

5x за 40 часов

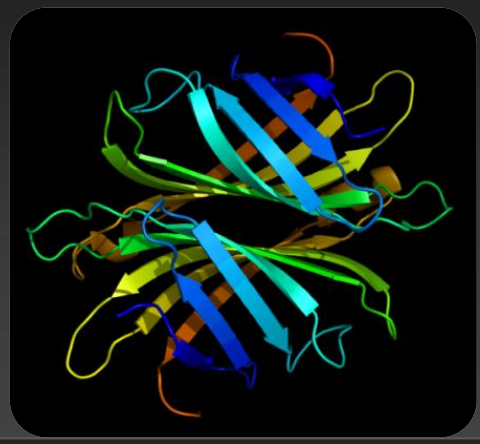
Оценка стоимости портфеля акций методом Монте-Карло

Глобальная консалтинговая компания

2x за 4 часа

Моделирование влияния растворителей на белки

Техаский университет в Сан-Антонио



5x за 8 часов

> “Optimizing code with directives is quite easy, especially compared to CPU threads or writing CUDA kernels. The most important thing is avoiding restructuring of existing code for production applications.”
>
> *-- Developer at the Global Manufacturer of Navigation Systems*

Небольшие усилия. Весомое улучшение.

Большая Нефтедобывающая Компания

3x за 7 дней

Решение миллиардов уровнений итеративно для моделирвоания добычи нефти в крупнейшем месторождении

UNIVERSITY of HOUSTON
CULLEN COLLEGE of ENGINEERING

Университет Хьюстона

Проф. М. Амин Каяли

20x за 2 дня

Исследования в области микромагнетизма, инновации в разработке магнитных носителей информации, сенсоров поля и биомагнетизма

THE UNIVERSITY OF MELBOURNE

Университет Мельбурна

Проф. Керри Блэк

65x за 2 дня

Понимание жизненных циклов популяции морского окуня в заливе Порт-Филипп

УГАТУ

УГАТУ

Проф. Артур Юлдашев

7x за 4 недели

Генерация стохастических геологических моделей нефтяных месторождений, обусловленных скважинными данными

COMPUTING FOR SCIENCE fs

GAMESS-UK

Др. Вилкинсон, Проф. Найдо

10x

Различные сферы применения, например исследования в области производства биотоплива, разработка молекулярных сенсоров

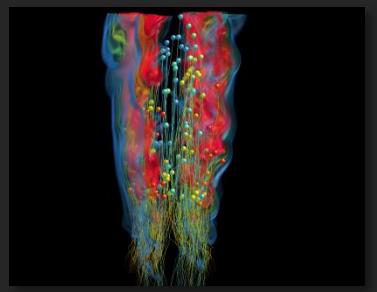
* Результаты получены с использованием компилятора PGI Accelerator

Выявление параллелизма

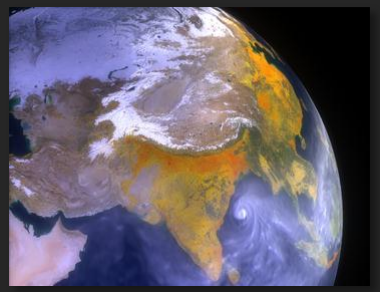
Когда средство оптимизации - директивы, тогда работа по оптимизации заключается в *выявлении параллелизма*, что само по себе улучшает качество и производительность кода

Пример: Работа по оптимизации приложений с использованием директив для нового суперкомпьютера Титан

S3D
Исследование горения топлива нового поколения



- **Тюнинг 3 подпрограмм (90% от общего времени выполнения программы)**
- **От 3x до 6x быстрее на CPU+GPU vs. CPU+CPU**
- **CPU версия была также ускорена на 50%**



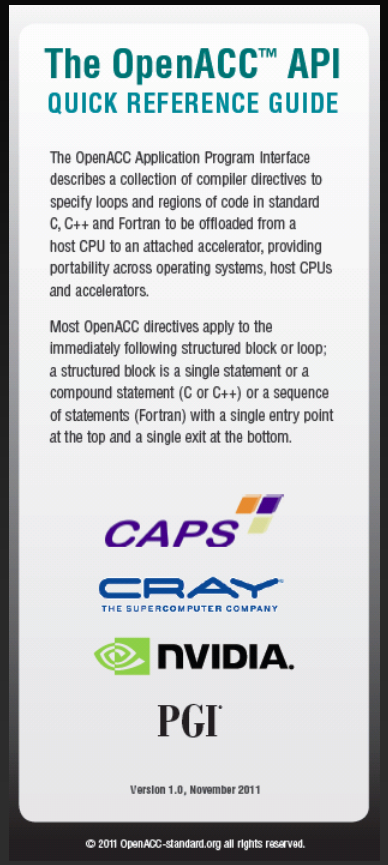
CAM-SE
Проработка вариантов адаптации к различным сценариям глобального изменения климата и смягчения последствий

- **Тюнинг одной подпрограммы (50% от общего времени выполнения программы)**
- **6.5x быстрее на CPU+GPU vs. CPU+CPU**
- **CPU версия была также ускорена на 100%**

OpenACC спецификации

- Полное описание стандарта OpenACC 3.1

http://www.openacc.org



Реализации стандарта

- PGI
  - PGI Accelerator with OpenACC: http://www.pgroup.com
  - Бесплатная пробная лицензия
- CAPS
  - CAPS OpenACC Compiler: http://www.caps-entreprise.com
  - Использует CUDA или OpenCL
- Cray
  - Cray Compiler Environment: http://www.cray.com/Products/XK6/Software.aspx

“Hello, world” в мире параллельных вычислений: SAXPY

SAXPY на C

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

...
// Вызвать SAXPY для 1M элементов
saxpy(1<<20, 2.0, x, y);
...
```

SAXPY на Фортране

```
subroutine saxpy(n, a, x, y)
  real :: x(:), y(:), a
  integer :: n, i
!$acc kernels
  do i=1,n
    y(i) = a*x(i)+y(i)
  enddo
!$acc end kernels
end subroutine saxpy


...
! Вызвать SAXPY для 1M элементов
call saxpy(2**20, 2.0, x, y)
...
```

Синтаксис

- Фортран

  `!$acc directive [clause [,] clause] …]`

  Часто идет в паре с закрывающей “end” директивой. Эта пара директив окружает блок кода

  `!$acc end directive`

- С

  `#pragma acc directive [clause [,] clause] …]`

  За директивей обычно следует блок кода

kernels: Ваша первая OpenACC директива

Каждый цикл исполняется как отдельное *ядро* (kernel) на GPU

```
!$acc kernels
      do i=1,n
         a(i) = 0.0
         b(i) = 1.0
         c(i) = 2.0
      end do

      do i=1,n
         a(i) = b(i) + c(i)
      end do
!$acc end kernels
```

Ядро 1

Ядро 2

Ядро:
параллельная функция, которая исполняется на GPU

Директива kernels

Фортран

```
!$acc kernels [clause …]
    блок кода
!$acc end kernels
```

C

```
#pragma acc kernels [clause …]
    { блок кода }
```

Clauses

```
if( condition )
async( expression )
и много других...
```

Подсказка для C: ключевое слово restrict

- Декларация намерений со стороны программиста компилятору
  - Применимо к указателю, например:

```
float *restrict ptr
```

  - Значение: “в течение времени жизни переменной `ptr`, только она или значения, напрямую полученное из этой перемнной (например `ptr + 1`) будут использованы для доступа к объекту, на который эта переменная указывает”*
- Ограничивает действие эффекта альясинга указателей
- OpenACC компиляторы часто требуют `restrict`, чтобы определить независимость итераций цикла
  - Иначе компилятор не в состоянии паралеллизовать цикл, в котором есть доступ по указателю `ptr`
  - Внимание: Если программист нарушает эту декларацию намерений, то результат неизвестен

http://en.wikipedia.org/wiki/Restrict

Полный код примера SAXPY

- Тривиальный пример
  - Использование kernels

```
#include <stdlib.h>

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

*restrict:
"Я обещаю, что y, x – не синонимы"

```
int main(int argc, char **argv)
{
  int N = 1<<20; // 1 million floats

  if (argc > 1)
    N = atoi(argv[1]);

  float *x = (float*)malloc(N * sizeof(float));
  float *y = (float*)malloc(N * sizeof(float));

  for (int i = 0; i < N; ++i) {
    x[i] = 2.0f;
    y[i] = 1.0f;
  }

  saxpy(N, 3.0f, x, y);

  return 0;
}
```

Сборка

- C:

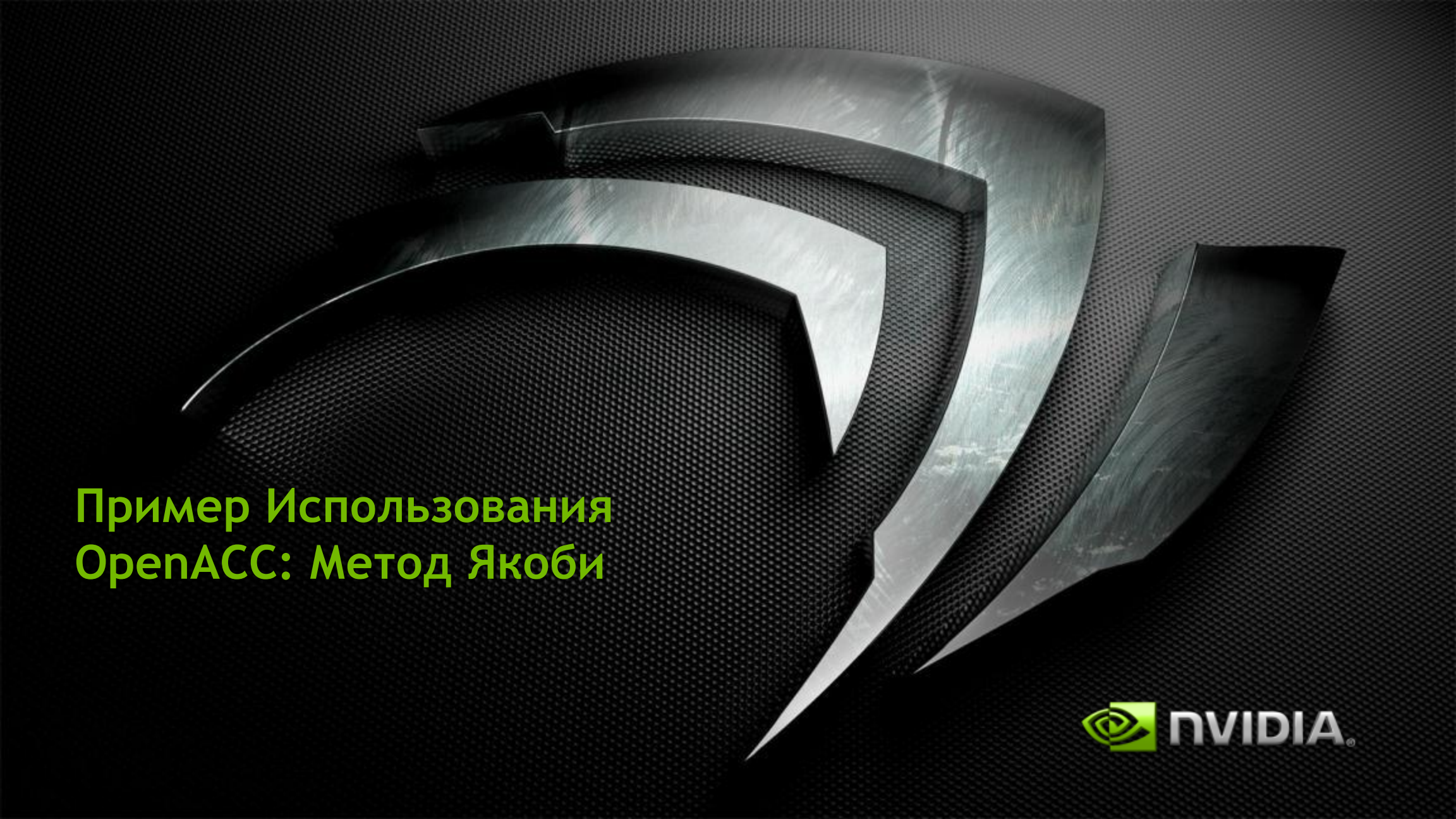
  ```
  pgcc -acc -ta=nvidia -Minfo=accel -o saxpy_acc saxpy.c
  ```

- Фортран:

  ```
  pgf90 -acc -ta=nvidia -Minfo=accel -o saxpy_acc saxpy.f90
  ```

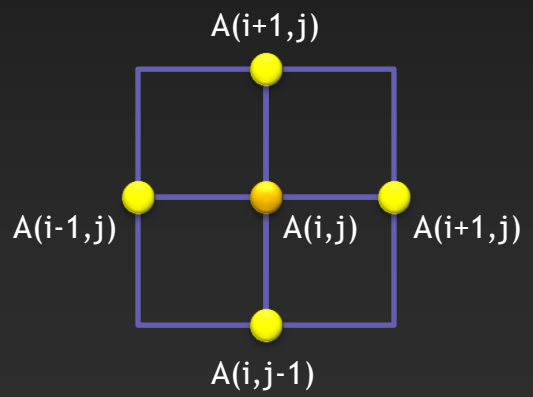
- Сообщения компилятора:

```
pgcc -acc -Minfo=accel -ta=nvidia -o saxpy_acc saxpy.c
saxpy:
      8, Generating copyin(x[:n-1])
         Generating copy(y[:n-1])
         Generating compute capability 1.0 binary
         Generating compute capability 2.0 binary
      9, Loop is parallelizable
         Accelerator kernel generated
          9, #pragma acc loop worker, vector(256) /* blockIdx.x threadIdx.x */
             CC 1.0 : 4 registers; 52 shared, 4 constant, 0 local memory bytes; 100% occupancy
             CC 2.0 : 8 registers; 4 shared, 64 constant, 0 local memory bytes; 100% occupancy
```


Пример Использования
OpenACC: Метод Якоби
NVIDIA

Пример: Метод Якоби

- Пример: Решаем уравнение Лапласа в 2D: $\nabla^2 f(x,y) = 0$
  - Итеративно сходится к решению уравнения, путем вычисления нового значения в точке путем усреднения значений в соседних точках.
  - Распространенный, полезный алгоритм



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

Метод Якоби: Код на C

```
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++;
}
```

Выполнять шаги до сходимости

Итерации по всем элементам матрицы

Вычислить новые значения из соседних

Обновить максимальное значение ошибки

Поменять местами входной и выходной массивы

Метод Якоби: Код на Фортране


```
do while ( err > tol .and. iter < iter_max )
  err=0._fp_kind

  do j=2,m-1
    do i=2,n-1


      Anew(i,j) = .25_fp_kind * (A(i+1, j  ) + A(i-1, j  ) + &
                                 A(i  , j-1) + A(i  , j+1))


      err = max(err, Anew(i,j) - A(i,j))
    end do
  end do

  do j=2,m-1
    do i=2,n-1
      A(i,j) = Anew(i,j)
    end do
  end do

  iter = iter +1
end do
```

Выполнять шаги до сходимости

Итерации по всем элементам матрицы

Вычислить новые значения из соседних

Обновить максимальное значение ошибки

Поменять местами входной и выходной массивы

OpenMP, код на C

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

#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++ ) {

      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++;
}
```

Параллелизация цикла по нитям CPU

Параллелизация цикла по нитям CPU

OpenMP, Код на Фортране

```
do while ( err > tol .and. iter < iter_max )
  err=0._fp_kind

!$omp parallel do shared(m,n,Anew,A) reduction(max:err)
  do j=2,m-1
    do i=2,n-1

      Anew(i,j) = .25_fp_kind * (A(i+1, j  ) + A(i-1, j  ) + &
                                 A(i  , j-1) + A(i  , j+1))

      err = max(err, Anew(i,j) - A(i,j))
    end do
  end do

!$omp parallel do shared(m,n,Anew,A)
  do j=2,m-1
    do i=2,n-1
      A(i,j) = Anew(i,j)
    end do
  end do

  iter = iter +1
end do
```

Параллелизация цикла по нитям CPU

Параллелизация цикла по нитям CPU

OpenACC, директива kernels

- Задача: Использовать OpenACC директивы `kernels` для параллелизации внутренних циклов в методе Якоби

- Вопрос: Можем ли мы получить ускорение лишь с использованием директив `kernels`?
  - По сравнению с 1 ядром CPU? А с 6 ядрами CPU?

OpenACC, Код на C

```
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++;
}
```

Выполнить GPU ядра для вложенных циклов

Выполнить GPU ядра для вложенных циклов

OpenACC, код на Фортране

```
do while ( err > tol .and. iter < iter_max )
  err=0._fp_kind

!$acc kernels
  do j=2,m-1
    do i=2,n-1

      Anew(i,j) = .25_fp_kind * (A(i+1, j  ) + A(i-1, j  ) + &
                                 A(i  , j-1) + A(i  , j+1))

      err = max(err, Anew(i,j) - A(i,j))
    end do
  end do
!$acc end kernels

!$acc kernels
  do j=2,m-1
    do i=2,n-1
      A(i,j) = Anew(i,j)
    end do
  end do
!$acc end kernels
  iter = iter +1
end do
```

Выполнить GPU ядра для вложенных циклов

Выполнить GPU ядра для вложенных циклов

C Makefile


```
CC       = pgcc
CCFLAGS  =
ACCFLAGS = -acc -ta=nvidia, -Minfo=accel
OMPFLAGS = -fast -mp -Minfo

BIN =  laplace2d_omp laplace2d_acc

all: $(BIN)

laplace2d_acc: laplace2d.c
        $(CC) $(CCFLAGS) $(ACCFLAGS) -o $@ $<

laplace2d_omp: laplace2d.c
        $(CC) $(CCFLAGS) $(OMPFLAGS) -o $@ $<

clean:
        $(RM) $(BIN)
```

Фортран Makefile

```
F90        = pgf90
CCFLAGS    =
ACCFLAGS = -acc -ta=nvidia, -Minfo=accel
OMPFLAGS = -fast -mp -Minfo

BIN =  laplace2d_f90_omp laplace2d_f90_acc

all: $(BIN)

laplace2d_f90_acc: laplace2d.f90
        $(F90) $(CCFLAGS) $(ACCFLAGS) -o $@ $<

laplace2d_f90_omp: laplace2d.f90
        $(F90) $(CCFLAGS) $(OMPFLAGS) -o $@ $<

clean:
        $(RM) $(BIN)
```

Сообщения компилятора (C)


```
pgcc  -acc -ta=nvidia -Minfo=accel -o laplace2d_acc laplace2d.c
main:
     57, Generating copyin(A[:4095][:4095])
         Generating copyout(Anew[1:4094][1:4094])
         Generating compute capability 1.3 binary
         Generating compute capability 2.0 binary
     58, Loop is parallelizable
     60, Loop is parallelizable
         Accelerator kernel generated
         58, #pragma acc loop worker, vector(16) /* blockIdx.y threadIdx.y */
         60, #pragma acc loop worker, vector(16) /* blockIdx.x threadIdx.x */
             Cached references to size [18x18] block of 'A'
             CC 1.3 : 17 registers; 2656 shared, 40 constant, 0 local memory bytes; 75% occupancy
             CC 2.0 : 18 registers; 2600 shared, 80 constant, 0 local memory bytes; 100% occupancy
         64, Max reduction generated for error
     69, Generating copyout(A[1:4094][1:4094])
         Generating copyin(Anew[1:4094][1:4094])
         Generating compute capability 1.3 binary
         Generating compute capability 2.0 binary
     70, Loop is parallelizable
     72, Loop is parallelizable
         Accelerator kernel generated
         70, #pragma acc loop worker, vector(16) /* blockIdx.y threadIdx.y */
         72, #pragma acc loop worker, vector(16) /* blockIdx.x threadIdx.x */
             CC 1.3 : 8 registers; 48 shared, 8 constant, 0 local memory bytes; 100% occupancy
             CC 2.0 : 10 registers; 8 shared, 56 constant, 0 local memory bytes; 100% occupancy
```

Производительность

CPU: Intel Xeon X5680
6 Cores @ 3.33GHz

GPU: NVIDIA Tesla M2070

Конфигурация	Время (с)	Ускорение
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

Ускорение по сравнению с 1 ядром CPU

Ускорение по сравнению с 6 ядрами CPU

Что пошло не так?

- Добавьте опцию компиляции `-ta=nvidia,time`

```
Accelerator Kernel Timing data
/usr/users/6/harrism/openacc-workshop/solutions/001-laplace2D-kernels/laplace2d.c
  main
    69: region entered 1000 times
        time(us): total=77524918 init=240 region=77524678
                  kernels=4422961 data=66464916
        w/o init: total=77524678 max=83398 min=72025 avg=77524
        72: kernel launched 1000 times
            grid: [256x256]  block: [16x16]
            time(us): total=4422961 max=4543 min=4345 avg=4422
/usr/users/6/harrism/openacc-workshop/solutions/001-laplace2D-kernels/laplace2d.c
  main
    57: region entered 1000 times
        time(us): total=82135902 init=216 region=82135686
                  kernels=8346306 data=66775717
        w/o init: total=82135686 max=159083 min=76575 avg=82135
        60: kernel launched 1000 times
            grid: [256x256]  block: [16x16]
            time(us): total=8201000 max=8297 min=8187 avg=8201
        64: kernel launched 1000 times
            grid: [1]  block: [256]
            time(us): total=145306 max=242 min=143 avg=145
acc_init.c
  acc_init
    29: region entered 1 time
        time(us): init=158248
```

4.4 секунды

66.5 секунды

8.3 секунды

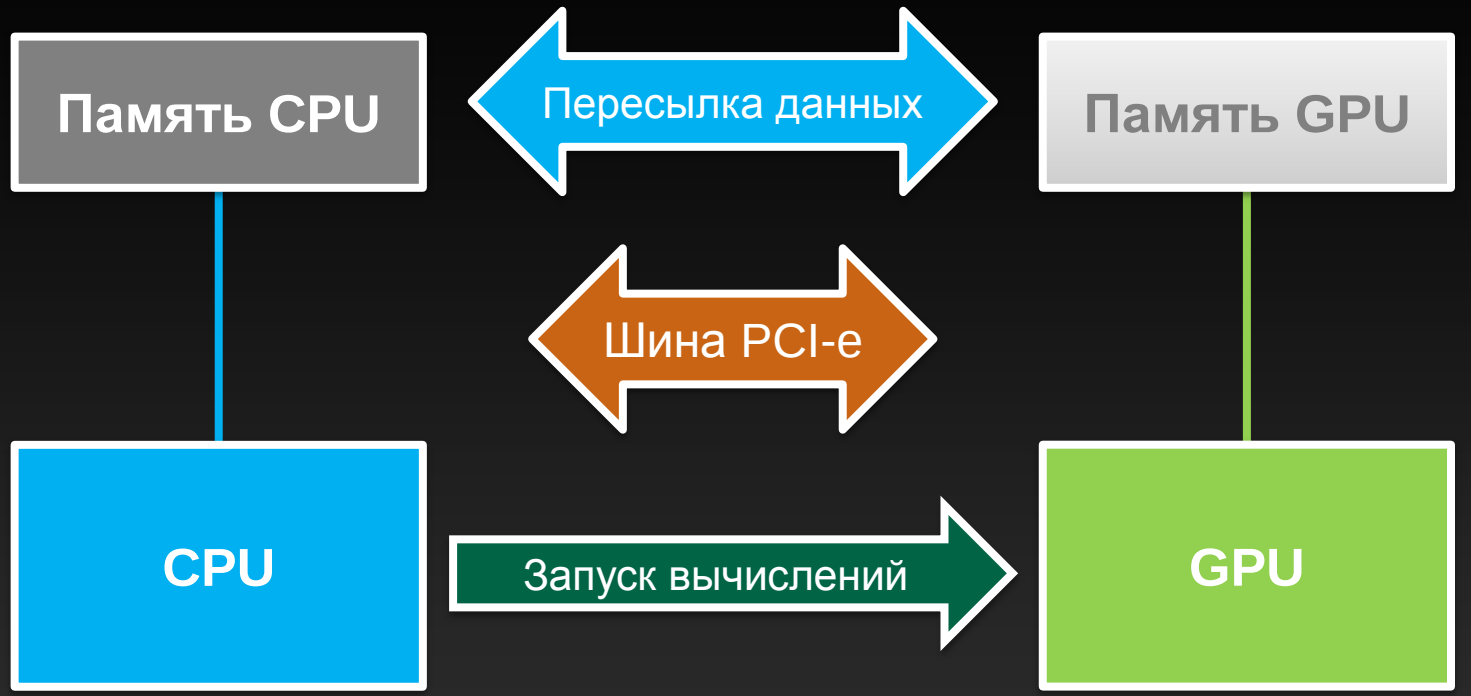
66.8 секунды

Большие затраты на пересылку данных!

Вычисления: 12.7 секунды

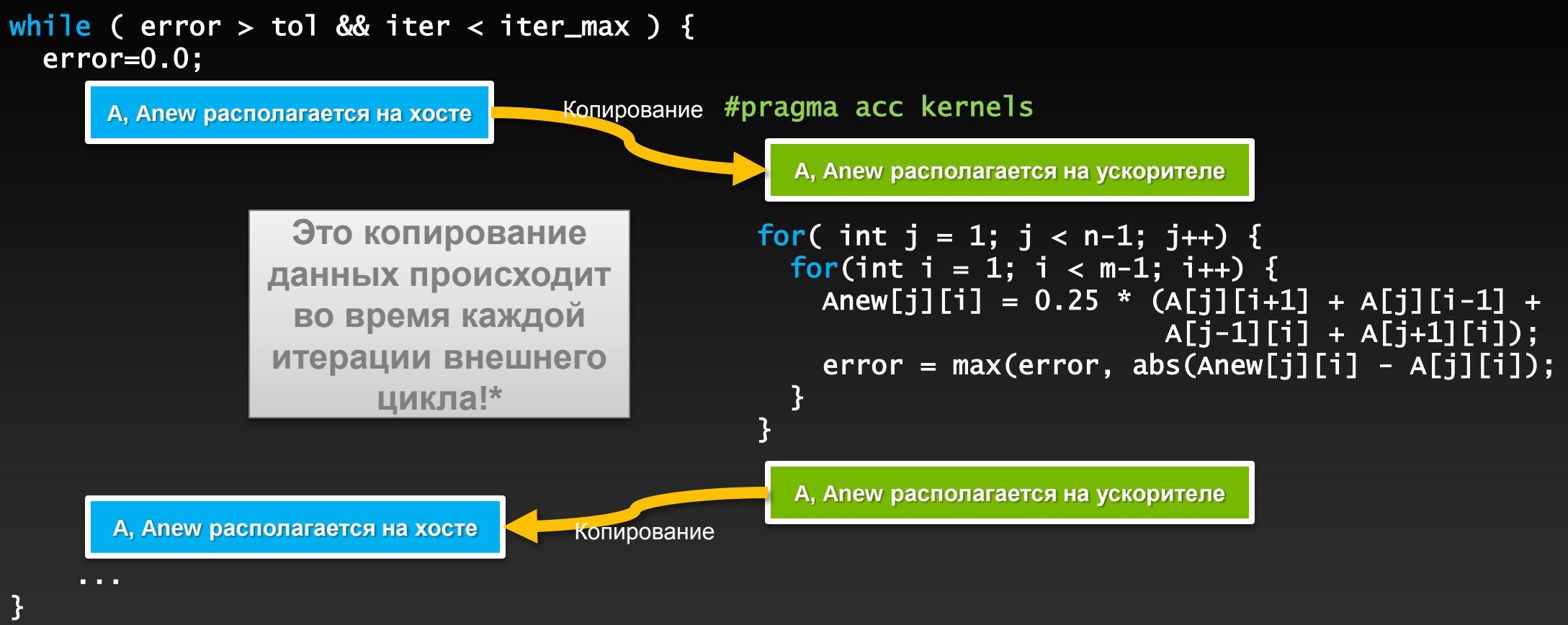
Пересылка данных: 133.3 секунды

Основные концепции



Для повышения эффективности отделите друг от друга передачу данных и вычисления

Избыточная передача данных



* Во внешнем цикле присутствует 2 директивы kernels, так что в каждой итерации выполняется 4 операции копирования данных!

Управление данными

Директива data

Фортран

```
!$acc data [clause …]
    блок кода
!$acc end data
```

C

```
#pragma acc data [clause …]
    { блок кода }
```

Общие операторы

```
if( condition )
async( expression )
```

Управление перемещением данных. Директивы могут быть вложенными.

Операторы, специфичные для данных

`copy ( list )`	Выделяет память на GPU и копирует данные с хоста на GPU в начале блока кода и обратно на хост в конце.
`copyin ( list )`	Выделяет память на GPU и копирует данные с хоста на GPU в начале блока кода.
`copyout ( list )`	Выделяет память на GPU и копирует данные с GPU на хост.
`create ( list )`	Выделяет память на GPU без копирования.
`present ( list )`	Данные уже находятся на ускорителе как результат работы внешней (по отношению к данной) директивы.

И `present_or_copy[in|out]`, `present_or_create`, `deviceptr`.

Размеры массивов

- Иногда компилятор не может определить размеры массива
  - Тогда необходимо явно их указать, используя операторы
- C

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

- Фортран

```
!$pragma acc data copyin(a(1:size)), copyout(b(s/4:3*s/4))
```

- Операторы управления данными могут быть использованы с директивами data, kernels и parallel

Директива update

Фортран

```
!$acc update [clause …]
```

C

```
#pragma acc update [clause …]
```

Операторы

```
host( list )
device( list )
```

```
if( condition )
async( expression )
```

Используется для обновления существующих данных после их изменения в соответствующей копии (например, для обновления копии данных на ускорителе после изменения данных на хосте)

Перемещает данные из GPU на хост и обратно.
Копирование данных может быть условным и/или асинхронным.

OpenACC, директива data

- Задача: Использовать OpenACC директивы data для минимизации копирования данных в примере с методом Якоби

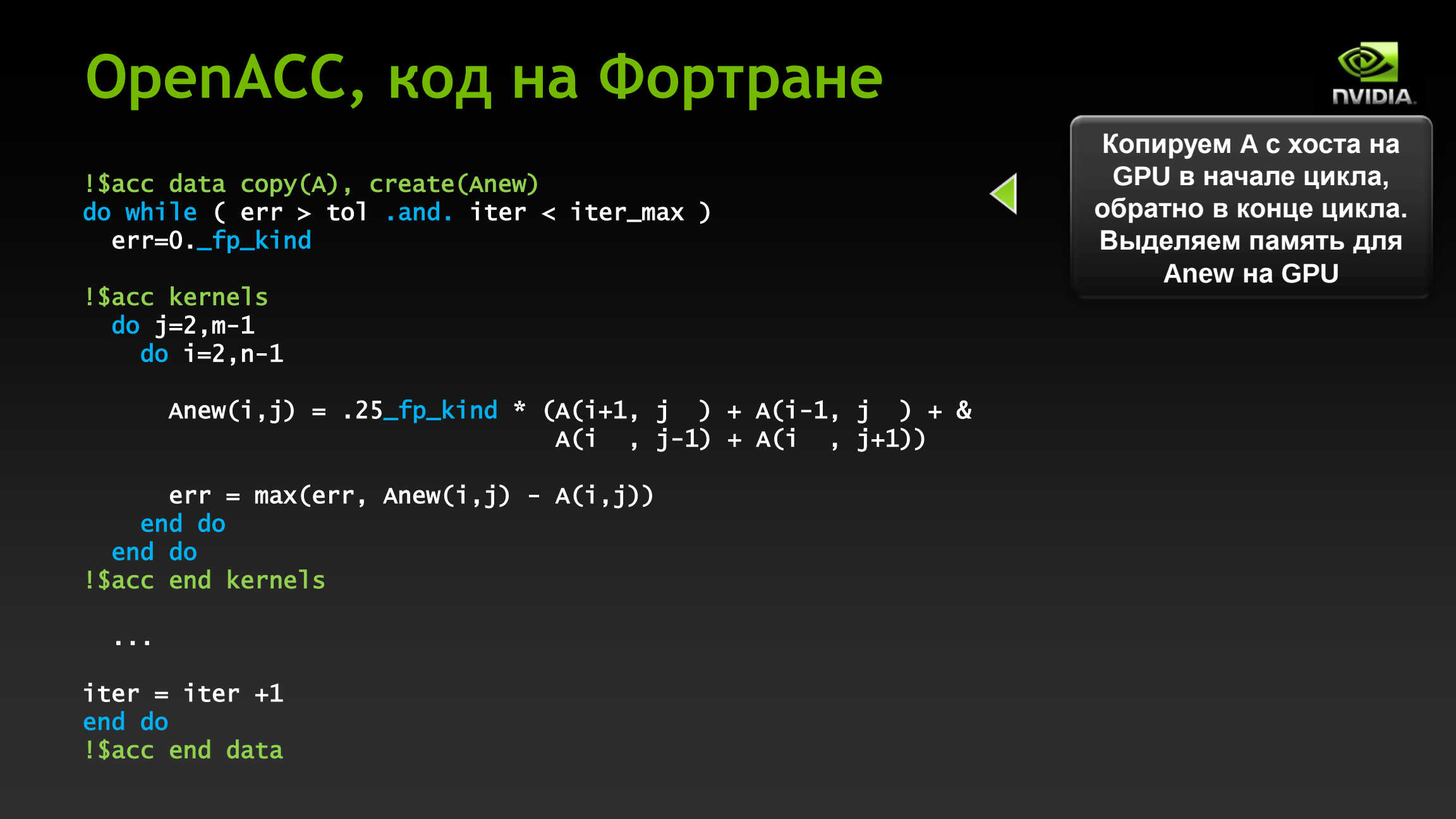
- Вопрос: Какое ускорение мы можем получить с использованием директивам data и kernels?
  - По сравнению с 1 ядром CPU? А с 6 ядрами CPU?

OpenACC, код на C

```
#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++;
}
```

Копируем A с хоста на GPU в начале цикла, обратно в конце цикла. Выделяем память для Anew на GPU

OpenACC, код на Фортране



```
!$acc data copy(A), create(Anew)
do while ( err > tol .and. iter < iter_max )
  err=0._fp_kind

!$acc kernels
  do j=2,m-1
    do i=2,n-1

      Anew(i,j) = .25_fp_kind * (A(i+1, j  ) + A(i-1, j  ) + &
                                 A(i  , j-1) + A(i  , j+1))

      err = max(err, Anew(i,j) - A(i,j))
    end do
  end do
!$acc end kernels

  ...

iter = iter +1
end do
!$acc end data
```

Копируем A с хоста на GPU в начале цикла, обратно в конце цикла. Выделяем память для Anew на GPU

Производительность

CPU: Intel Xeon X5680
6 Cores @ 3.33GHz

GPU: NVIDIA Tesla M2070

Конфигурация	Время (с)	Ускорение
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

Ускорение по сравнению с 1 ядром CPU

Ускорение по сравнению с 6 ядрами CPU

Тот же код выполняется за 9.78с на NVIDIA Tesla M2090 GPU

Выявление параллелизма в коде

- [Вложенные] циклы лучше всего подходят для параллелизации
- Небходимо большое количество итераций цикла для покрытия накладных расходов по запуску ядра на GPU и/или копирования данных
- Итерации цикла должны быть независимыми друг от друга
  - Подсказки компилятору: ключевое слово `restrict` (C), оператор `independent`
- Компилятор должен так или иначе знать размеры массивов
  - Можно явно указать размеры с помощью директив
- По возможности лучше избегать арифметики с указателями
  - Используйте индексированные массивы
- Функции внутри параллельных блоков кода должны быть inlinable