Master Educational Program
“Information technology in application

Mathematical computations with GPUs

Introduction to OpenACC

Alexey A. Romanenko
arom@ccfit.nsu.ru
Novosibirsk State University

Agenda

* What is OpenACC ?
* OpenACC API
* Examples…

Approaches to GPU programming

Application

Optimized libraries	Compiler directives	Programming languages (C/C++/FORTRAN)

Fast development

Maximum performance

Пример SAXPY на C: OpenMP

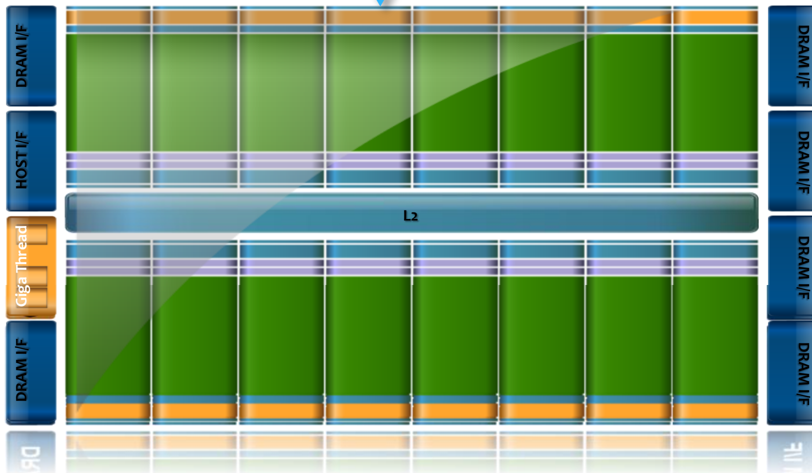
* Простота
* Открытый стандарт
* Высокая производительность

```
void saxpy(int n, float a, float *x,
           float *restrict y){
#pragma omp parallel for
  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);
...
```

Пример SAXPY на C: OpenACC

* Простота
* Открытый стандарт
* Высокая производительность

```
void saxpy(int n, float a, float *x,
           float *restrict y){
#pragma acc parallel
  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);
...
```



OpenACC API

* **Директивы компилятору указывают области параллельных вычислений** в языках C и Fortran
  * выгружает области с интенсивными вычислениями на GPU.
  * код не зависит от ОС, CPU, ускорителя, и компилятора
* Возможность создавать высокоуровневые **гибридные (CPU+ускоритель) программы**
  * без явной инициализации ускорителя
  * без явного копирования данных между CPU и ускорителем

OpenACC

* Программная модель позволяет программисту легко начать программировать GPU, обеспечивая компилятор подсказками:
  * данные для размещения на ускорителе и их характеристики
  * рекомендации по отображению циклов на ускоритель
  * и другие, связанные с производительностью, детали
* Совместим с другими языками программирования GPU и библиотеками:
  * взаимодействие с CUDA C/Fortran и GPU библиотеками
  * например CUFFT, CUBLAS, CUSPARSE, и т.д.

Спецификация v1.0

* Полная Спецификация OpenACC 1.0 доступна на сайте:
  http://www.openacc-standard.org
* Также доступна памятка по OpenACC
* Пробная версия компилятора:
  http://www.nvidia.com/object/openacc-gpu-directives.html
* Онлайн курсы и вебинары:
  http://www.nvidia.com/object/webinar.html

Модель исполнения OpenACC

* **CPU**
  * выполняет большую часть программы
  * выделяет память на ускорителе
  * инициирует копирование данных из памяти хоста в память ускорителя
  * отправляет код ядра на ускоритель
  * устанавливает ядра в очередь для исполнения на ускорителе
  * ожидает выполнения ядра
  * инициирует копирование данных из памяти ускорителя в память хоста
  * освобождает память ускорителя
* **Ускоритель**
  * исполняет ядра одно за другим
  * одновременно может передавать данные между хостом и ускорителем

Модель исполнения OpenACC

* Модель исполнения OpenACC имеет три уровня: **gang, worker, vector**
* На архитектуру эта модель отображается, как набор обрабатываемых элементов (PEs)
* Каждый PE содержит много рабочих и каждый рабочий может выполнять векторные инструкции
* Для GPU в большинстве случаев отображение происходит так: gang=block, worker=warp, vector=threads-in-warp
* Зависит от того, как компилятор посчитает нужным

Сравнение CUDA C/Fortran с OpenACC

* **CUDA C/Fortran:**
  + Высокая производительность кода, оптимизированного вручную
  + Портирование кода на GPU по частям
  - Поддержка только CUDA платформ
  - Необходимо иметь два различных варианта кода
* **OpenACC:**
  + Возможна хорошая производительность
  + Портирование кода на различные ускорители по частям
  + Поддержка не только CUDA платформ
  + Нужен только один вариант кода
  - Нельзя отследить решения компилятора
  - Почти все зависит от специфик реализации компилятора вендором

OpenACC API

Синтаксис директив

- **Fortran**

  !$acc *directive [атрибут [, атрибут] …]*

  структурированный блок

  !$acc end *directive*

- **C**

  #pragma acc *directive [атрибут [, атрибут] …]*

  структурированный блок

Компиляция

с помощью компилятора HMPP

* hmpp --openacc-target=CUDA gfortran <filename>
* hmpp --openacc-target=CUDA gcc <filename>

с помощью компилятора PGI

* pgfortran -acc -ta=nvidia <filename>
* pgcc -acc -ta=nvidia <filename>

с помощью компилятора CRAY

* ftn -h acc <filename>

Конструкция Parallel

Fortran

```
!$acc parallel [атрибут [, атрибут]… ]
        структурированный блок
!$acc end parallel
```

C

```
#pragma acc parallel [атрибут [, атрибут]… ]
        структурированный блок
```

Атрибуты конструкции Parallel

Основные атрибуты

- if (condition)
- async [(exp)]
- num_gangs (exp)
- num_workers (exp)
- vector_length(exp)
- private(list)
- firstprivate(list)
- reduction(operator:list)

атрибуты данных

- copy*(list)
- create(list)
- present(list)
- present_or_copy*(list)
- present_or_create(list)
- deviceptr(list)
- private(list)
- firstprivate(list)

- *<blank>|in|out

Ограничения в использовании конструкции Parallel

* Не может включать регионы parallel и kernels
* Нельзя делать условные входы и выходы в регион
* Не зависит от порядка указания атрибутов
* Условие в if должно быть только одно

Конструкция Kernels

Fortran

```
!$acc acc kernels [атрибут [, атрибут]… ]
          структурированный блок
!$acc end kernels
```

C

```
#pragma acc kernels [атрибут [, атрибут]… ]
          структурированный блок
```

Конструкция Kernels

```
!$acc kernels
  do i = 1,n
  do j = 1,n
     a(i,j) = 0.0
  enddo
  enddo
  do k = 1,n
     b(k) = 1.0
  enddo
!$acc end kernels
```

ядро 1

ядро 2

Атрибуты

Основные атрибуты

- if (condition)
- async [(exp)]

атрибуты данных

- copy*(list)
- create(list)
- present(list)
- present_or_copy*(list)
- present_or_create(list)
- deviceptr(list)
- private(list)
- firstprivate(list)

- *<blank>|in|out

Сравнение Parallel и Kernels

```
int a [10000];
int b [10000];
#pragma acc parallel
{
  for (int i=0; i<1000; i++)
    a[i] = i - 100 + 23;

  for (int j=0; j<1000; j++)
    b[j] = j - 10 + 213;
}
```

Сравнение Parallel и Kernels

```
arom@cuda:~/cuda/pgi$ pgcc -acc -ta=nvidia -Minfo=accel test10.c -o test10
main:
      7, Accelerator kernel generated
          9, #pragma acc loop vector(256) /* threadIdx.x */
         12, #pragma acc loop vector(256) /* threadIdx.x */
      7, Generating present_or_copyout(a[0:1000])
         Generating present_or_copyout(b[0:1000])
         Generating NVIDIA code
      9, Loop is parallelizable
     12, Loop is parallelizable
arom@cuda:~/cuda/pgi$ PGI_ACC_NOTIFY=3 ./test10
launch CUDA kernel  file=… function=main line=7 device=0 num_gangs=1 num_workers=1
vector_length=256 grid=1 block=256
download CUDA data  file=… function=main line=16 device=0 variable=b bytes=4000
download CUDA data  file=… function=main line=16 device=0 variable=a bytes=4000
```

Сравнение Parallel и Kernels

```
int a [10000];
int b [10000];
#pragma acc kernels
{
  for (int i=0; i<1000; i++)
    a[i] = i - 100 + 23;

  for (int j=0; j<1000; j++)
    b[j] = j - 10 + 213;
}
```

Сравнение Parallel и Kernels

```
arom@cuda:~/cuda/pgi$ pgcc -acc -ta=nvidia -Minfo=accel test10.c -o test10
main:
      7, Generating present_or_copyout(a[0:1000])
         Generating present_or_copyout(b[0:1000])
         Generating NVIDIA code
      9, Loop is parallelizable
         Accelerator kernel generated
          9, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
     12, Loop is parallelizable
         Accelerator kernel generated
         12, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
arom@cuda:~/cuda/pgi$ PGI_ACC_NOTIFY=3 ./test10
launch CUDA kernel  file=… function=main line=9 device=0 num_gangs=8 num_workers=1
vector_length=128 grid=8 block=128
launch CUDA kernel  file=… function=main line=12 device=0 num_gangs=8 num_workers=1
vector_length=128 grid=8 block=128
download CUDA data  file=… function=main line=16 device=0 variable=b bytes=4000
download CUDA data  file=… function=main line=16 device=0 variable=a bytes=4000
```

Конструкция Loop

Fortran

```
!$acc loop [атрибут [, атрибут]… ]
        do loop
```

C

```
#pragma acc loop [атрибут [, атрибут]… ]
        for loop
```

Атрибуты

* collapse(n)
* gang[(exp)]
* worker[(exp)]
* vector[(exp)]
* seq
* independent
* private(list)
* reduction(op:list)

Сравнение Independent и Seq

```
int a [10000];

#pragma acc kernels
{
#pragma acc loop independent
    for (int i=0; i<100; i++){
      #pragma acc loop independent
      {
        for (int j=0; j<100; j++)
          a[i*100 + j] = i - 100 + 23 + j;
      }
    }
}
```

Сравнение Independent и Seq

```
arom@cuda:~/cuda/pgi$ pgcc -acc -Minfo=accel -o c1 ./c1.c
main:
      5, Generating present_or_copy(a[0:])
         Generating NVIDIA code
      8, Loop is parallelizable
     11, Loop is parallelizable
         Accelerator kernel generated
          8, #pragma acc loop gang /* blockIdx.y */
         11, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
```

Сравнение Independent и Seq

```
int a [10000];

#pragma acc kernels
{
#pragma acc loop independent
    for (int i=0; i<100; i++){
      #pragma acc loop seq
      {
        for (int j=0; j<100; j++)
          a[i*100 + j] = i - 100 + 23 + j;
      }
    }
}
```

Сравнение Independent и Seq

```
arom@cuda:~/cuda/pgi$ pgcc -acc -Minfo=accel -o c2 ./c2.c
main:
      5, Generating present_or_copy(a[0:])
         Generating NVIDIA code
      8, Loop is parallelizable
     11, Loop is parallelizable
         Accelerator kernel generated
          8, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
```

Комбинированные директивы

Fortran

```
!$acc parallel loop [атрибут … ]
        do loop
[!$acc end parallel loop]

!$acc kernels loop [атрибут… ]
        do loop
[!$acc end kernels loop]
```

C

```
#pragma acc parallel loop
[атрибут… ]
        for loop

#pragma acc kernels loop
[атрибут… ]
        for loop
```

Отображение на блоки и нити CUDA

* Вложенные циклы создают многомерные сетки и блоки
* В большинстве случаев:
  gang -> blocks, vector -> threads
* Пример: 2-мерный цикл ->
  grid[100x200], block[16x32]

```
#pragma acc kernels loop gang(100), vector(16)
    for( … )
#pragma acc loop gang(200), vector(32)
        for( … )
```

16 нитей в длину

100 блоков в длину (ряд/направление Y)

32 нити в ширину

200 блоков в ширину (столбец/направление X)

Отображение на блоки и нити CUDA

```
n = 12800;
#pragma acc kernels
   for( int i = 0; i < n; ++i ) y[i] += a*x[i];
```

n/32 блока, каждый по 32 нити по умолчанию

```
#pragma acc kernels loop gang(100) vector(128)
   for( int i = 0; i < n; ++i ) y[i] += a*x[i];
```

100 блоков, каждый по 128 нитей, каждая нить выполняет одну итерацию цикла с использованием kernels

```
#pragma acc parallel num_gangs(100) vector_length(128)
{
   #pragma acc loop gang vector
   for( int i = 0; i < n; ++i ) y[i] += a*x[i]; }
```

100 блоков, каждый по 128 нитей, каждая нить выполняет одну итерацию цикла с использованием parallel

Отображение на блоки и нити CUDA

```
#pragma acc parallel num_gangs(100)
{
   for( int i = 0; i < n; ++i ) y[i] += a*x[i]; }
```

100 блоков, каждый по 32 нити по умолчанию

```
#pragma acc parallel num_gangs(100)
{
   #pragma acc loop gang
   for( int i = 0; i < n; ++i ) y[i] += a*x[i]; }
```

100 блоков, каждый по 32 нити по умолчанию

Отображение на блоки и нити CUDA

Каждая нить может выполнять одну итерацию цикла, или несколько, в зависимости от того, как определено отображение

```
#pragma acc kernels loop gang(100) vector(128)
   for( int i = 0; i < n; ++i ) y[i] += a*x[i];
```

100 блоков, каждый по 128 нитей

```
#pragma acc kernels loop gang(50) vector(128)
   for( int i = 0; i < n; ++i ) y[i] += a*x[i];
```

50 блоков, каждый по 128 нитей

Выполнение нескольких итераций одной нитью может увеличить производительность, снизив затраты на инициализацию

Атрибуты для области Data

Fortran

Синтаксис: array (начало : конец [, н:к] …)

Примеры: a(:, :), a(1:100, 2:n)

C

Синтаксис: array[начало : длина]

Примеры: a[2:n] // это значит a[2], a[3], … , a[2+n-1]

Использование региона Data

```
int a [1000];
int b [1000];

#pragma acc parallel
for (int i=0; i<1000; i++)
{
    a[i] = i - 100 + 23;
}

#pragma acc parallel
for (int j=0; j<1000; j++)
{
    b[j] = a[j] - j - 10 + 213;
}
```

Директивы ***parallel*** размещены отдельно и потребуют генерации ядер с отдельным копированием данных

Использование региона Data

```
main:
     11, Accelerator kernel generated
         12, #pragma acc loop vector(256) /* threadIdx.x */
     11, Generating present_or_copyin(a[0:])
         Generating NVIDIA code
     12, Loop is parallelizable
     17, Accelerator kernel generated
         18, #pragma acc loop vector(256) /* threadIdx.x */
     17, Generating present_or_copyin(b[0:])
         Generating present_or_copyin(a[0:])
         Generating NVIDIA code
     18, Loop is parallelizable
```

Использование региона Data

```
PGI_ACC_NOTIFY=3 ./test11
launch CUDA kernel  file=… function=main line=9 device=0 num_gangs=1
num_workers=1 vector_length=256 grid=1 block=256
download CUDA data  file=… function=main line=15 device=0 variable=a
bytes=4000
upload CUDA data  file=… function=main line=15 device=0 variable=a
bytes=4000
launch CUDA kernel  file=… function=main line=15 device=0 num_gangs=1
num_workers=1 vector_length=256 grid=1 block=256
download CUDA data  file=… function=main line=22 device=0 variable=a
bytes=4000
download CUDA data  file=… function=main line=22 device=0 variable=b
bytes=4000
```

Использование региона Data

```
int a [1000];
int b [1000];
#pragma acc data copyout (a[0:1000],b[0:1000])
{
#pragma acc parallel
for (int i=0; i<1000; i++)
{
    a[i]=i-100+23;
}

#pragma acc parallel
for (int j=0; j<1000; j++)
{
    b[j]=a[j]-j-10+213;
}
}
```

Директивы ***parallel*** размещены в рамках объемлющего региона ***data*** и потребуют генерации отдельных ядер, однако данные копируются один раз

Использование региона Data

```
pgcc -acc -ta=nvidia -Minfo=accel test11.c -o test11
main:
      9, Generating copyout(a[0:])
         Generating copyout(b[0:])
     11, Accelerator kernel generated
         12, #pragma acc loop vector(256) /* threadIdx.x */
     11, Generating NVIDIA code
     12, Loop is parallelizable
     17, Accelerator kernel generated
         18, #pragma acc loop vector(256) /* threadIdx.x */
     17, Generating NVIDIA code
     18, Loop is parallelizable
```

Использование региона Data

```
PGI_ACC_NOTIFY=3 ./test11
launch CUDA kernel  file=… function=main line=11 device=0 num_gangs=1
num_workers=1 vector_length=256 grid=1 block=256
launch CUDA kernel  file=… function=main line=17 device=0 num_gangs=1
num_workers=1 vector_length=256 grid=1 block=256
download CUDA data  file=… function=main line=24 device=0 variable=b
bytes=4000
download CUDA data  file=… function=main line=24 device=0 variable=a
bytes=4000
```

Остальные директивы

* **host_data**
  Делает адрес данных на ускорителе доступным для хоста
* **cache**
  Кэширует данные через программно-управляемый кэш.
  (CUDA разделяемая память)
* **update**
  Обновляет существующие данные после их изменения
* **wait**
  Ожидает выполнения асинхронных операций на ускорителе
* **declare**
  Указывает, что необходимо выделить память на ускорителе
  для использования в рамках региона data

Конструкция Host_Data

Fortran

```
!$acc host_data [атрибут [, атрибут]… ]
          структурированный блок
!$acc end host_data
```

C

```
#pragma acc host_data [атрибут [, атрибут]… ]
          структурированный блок
```

Директива Cache

Fortran

!$acc cache (list)

Пример: !$acc cache (arr (н1:к1, н2:к2))

C

#pragma cache (list) new-line

Пример: #pragma cache (arr[начало:длина])

Директива Update

Fortran

!$acc update атрибут [, атрибут]…

C

#pragma acc update атрибут [, атрибут]…

Атрибуты

* host(list)
* device(list)
* if(condition)
* async[(expr)]

Директива Wait

Fortran

!$acc wait [(expression)]

C

#pragma acc [(expression)]

Директива Declare

Fortran

!$acc declare [атрибут [, атрибут]…]

C

#pragma acc declare [атрибут [, атрибут]…]

Атрибуты данных

* copy*(list)
* create(list)
* present(list)
* present_or_copy*(list)
* present_or_create(list)
* deviceptr(list)
* device_resident(list)

* *<blank>|in|out

Подпрограммы runtime

Fortran
use openacc
#include “openacc_lib.h”

C
#include “openacc.h”

Функции

* acc_get_num_devices
* acc_set_device_type
* acc_set_device_num
* acc_async_wait
* acc_init
* acc_malloc
* acc_free
* …

Переменные среды и условная компиляция

* ACC_DEVICE device
  Тип ускорителя
* ACC_DEVICE_NUM num
  Номер ускорителя
* _OPENACC
  Директива препроцессору для условной компиляции

OpenACC v2.0
июнь 2013

* Дополнения:
  * entry data, exit data
  * routine
  * atomic
  * API
  * и т.д.

Example

Example

```
!Jacobi solver
        do while (change > tolerance)
            change = 0.0
            iter = iter + 1
!Parallel region to be executed on GPU
            do  j = 2, n-1
            do  i = 2, n-1
                newa(i,j)= w0 * a(i,j) + &
                    w1 * (a(i-1,j)+a(i,j-1)+a(i+1,j)+a(i,j+1)) + &
                    w2 * (a(i-1,j-1)+a(i-1,j+1)+a(i+1,j-1)+a(i+1,j+1))
               change = max(change, abs(newa(i,j) - a(i, j)))
            enddo
            enddo
            a(2:n-1,2:n-1) = newa(2:n-1,2:n-1)
!end of parallel region
        enddo
```

Example

```
!Jacobi solver
        do while (change > tolerance)
            change = 0.0
            iter = iter + 1
!$acc parallel
            do  j = 2, n-1
            do  i = 2, n-1
                newa(i,j)= w0 * a(i,j) + &
                    w1 * (a(i-1,j)+a(i,j-1)+a(i+1,j)+a(i,j+1)) + &
                    w2 * (a(i-1,j-1)+a(i-1,j+1)+a(i+1,j-1)+a(i+1,j+1))
               change = max(change, abs(newa(i,j) - a(i, j)))
            enddo
            enddo
            a(2:n-1,2:n-1) = newa(2:n-1,2:n-1)
!$acc end parallel
        enddo
```

Example

```
!$acc data copy(a, newa)
        do while (change > tolerance)
            change = 0.0
            iter = iter + 1
!$acc parallel reduction(max:change)
            do  j = 2, n-1
            do  i = 2, n-1
                newa(i,j)= w0 * a(i,j) + &
                    w1 * (a(i-1,j)+a(i,j-1)+a(i+1,j)+a(i,j+1)) + &
                    w2 * (a(i-1,j-1)+a(i-1,j+1)+a(i+1,j-1)+a(i+1,j+1))
               change = max(change, abs(newa(i,j) - a(i, j)))
            enddo
            enddo
!$acc end parallel
!$acc parallel
            a(2:n-1,2:n-1) = newa(2:n-1,2:n-1)
!$acc end parallel
        enddo
!$acc end data
```

Example

```
!$acc data copyin(a), create(newa)
        do while (change > tolerance)
            change = 0.0
            iter = iter + 1
!$acc parallel reduction(max:change)
            do  j = 2, n-1
            do  i = 2, n-1
                newa(i,j)= w0 * a(i,j) + &
                    w1 * (a(i-1,j)+a(i,j-1)+a(i+1,j)+a(i,j+1)) + &
                    w2 * (a(i-1,j-1)+a(i-1,j+1)+a(i+1,j-1)+a(i+1,j+1))
               change = max(change, abs(newa(i,j) - a(i, j)))
            enddo
            enddo
!$acc end parallel
!$acc parallel
            a(2:n-1,2:n-1) = newa(2:n-1,2:n-1)
!$acc end parallel
        enddo
!$acc end data
```

Example

```
!$acc data copyin(a), create(newa)
        do while (change > tolerance)
            change = 0.0
            iter = iter + 1
!$acc kernels loop reduction(max:change), gang(32), worker(8)
            do  j = 2, n-1
            do  i = 2, n-1
                newa(i,j)= w0 * a(i,j) + &
                    w1 * (a(i-1,j)+a(i,j-1)+a(i+1,j)+a(i,j+1)) + &
                    w2 * (a(i-1,j-1)+a(i-1,j+1)+a(i+1,j-1)+a(i+1,j+1))
               change = max(change, abs(newa(i,j) - a(i, j)))
            enddo
            enddo
!$acc end kernels loop
!$acc parallel
            a(2:n-1,2:n-1) = newa(2:n-1,2:n-1)
!$acc end parallel
        enddo
!$acc end data
```

Example

```
pgfortran -fast -acc -ta=nvidia -Minfo=accel -o jac_p jac.f90
jacobi:
     42, Generating copyin(a(1:n,1:n))
         Generating copyout(a(2:n-1,2:n-1))
         Generating local(newa(:,:))
     43, Loop is parallelizable
     44, Loop is parallelizable
         Accelerator kernel generated
         43, !$acc do parallel, vector(16) ! blockidx%y threadidx%y
         44, !$acc do parallel, vector(16) ! blockidx%x threadidx%x
             Cached references to size [18x18] block of 'a'
         48, Max reduction generated for cur_eps
     51, Loop is parallelizable
         Accelerator kernel generated
         51, !$acc do parallel, vector(16) ! blockidx%x threadidx%x
             !$acc do parallel, vector(16) ! blockidx%y threadidx%y
```

Example

* **Tesla T10 Processor**

[conqueror@tesla-cmc Tutorial2]$ ./J4.exe 1024

reached delta= 0.09998 in 3430 iterations for 1024 x 1024 array

time = 25.8760 seconds

* **Intel(R) Xeon(R) CPU E5620 @2.40GHz**

[conqueror@tesla-cmc sc1]$ ./j5 1024

Jacobi 1024 x 1024

JacobiHost converged in 3427 iterations to residual 0.099976

time(host) = 140.386185 seconds

Example

	N=400	N=512	N=1024
CPU Single CPU thread	6.7759	16.0250	140.3861
OpenMP (optimized)	1.8580	3.7771	29.6452
PGI OpenACC	6.8860	8.9890	9.2995
CUDA C (optimized)	3.8095	4.1140	6.5899