An Introduction to OpenACC Part II

Wei Feinstein

HPC User Services@LSU

LONI Parallel Programming Workshop 2015

Louisiana State University

Roadmap

- Recap of OpenACC
- OpenACC with GPU-enabled library
- Code profiling
- Code tuning with performance tool
- Programming on multi-GPUs

Large Scale of applications

Genomics

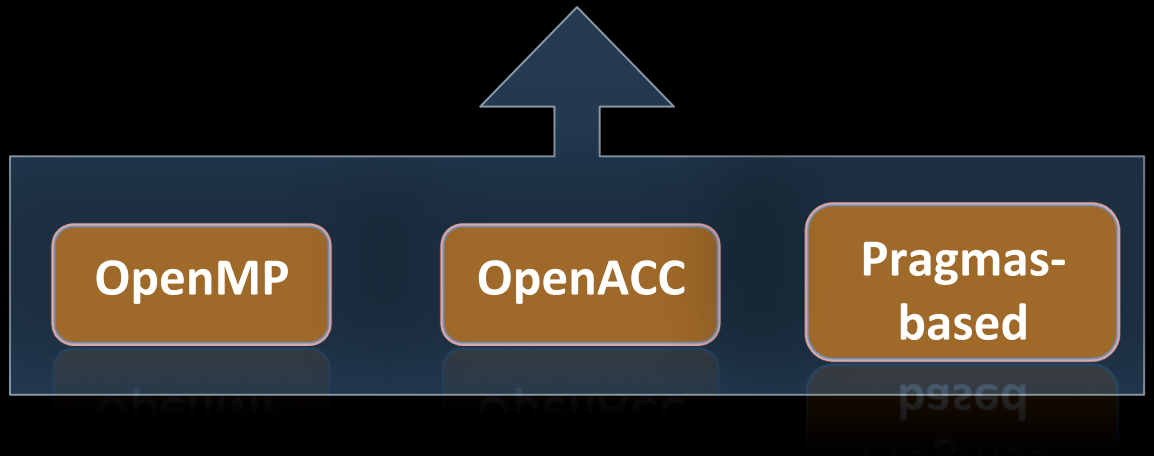
Deep Learning

Costal Storm Prediction

Multi-core CPU

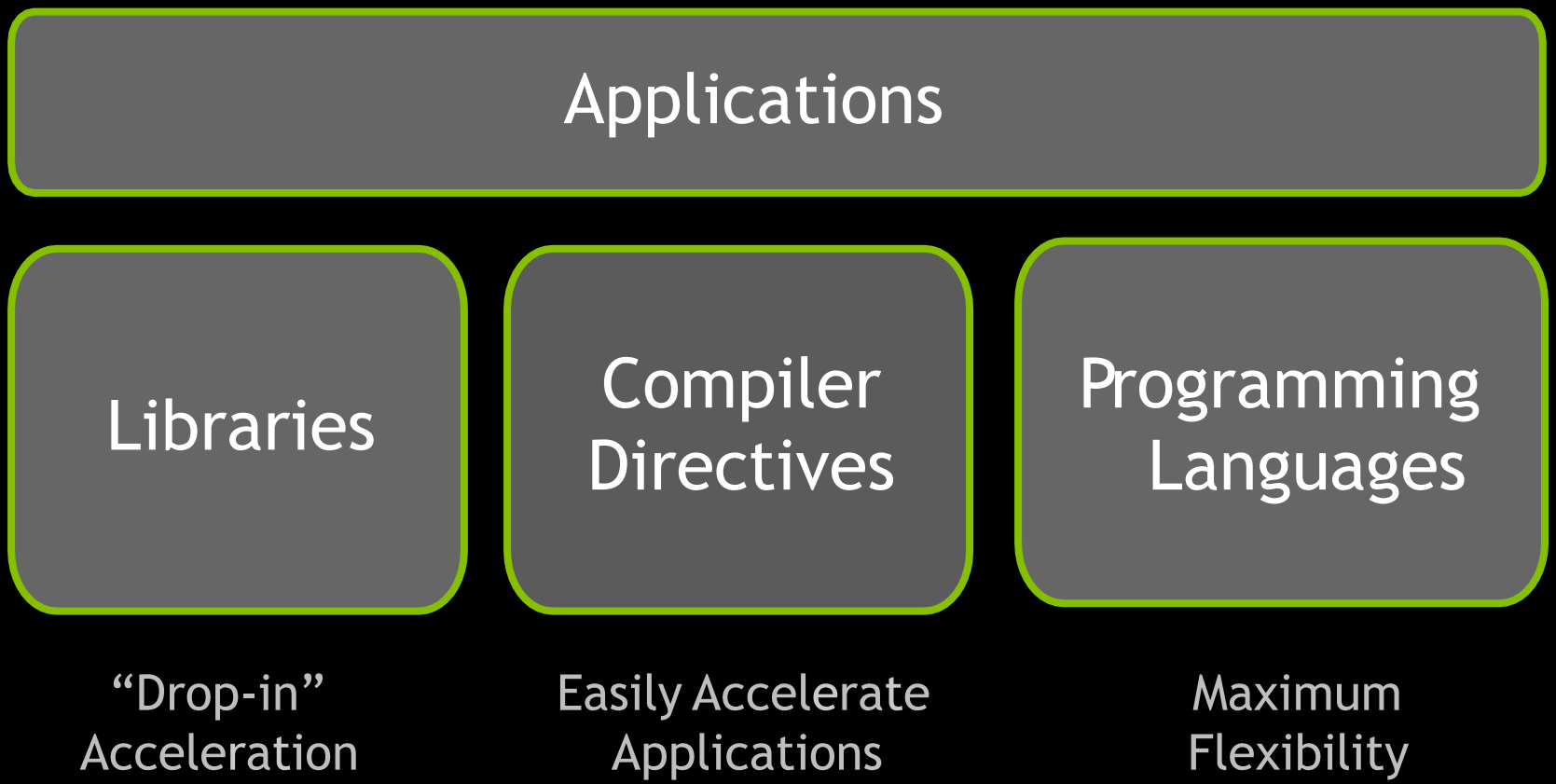
GPU

Many-core CPU

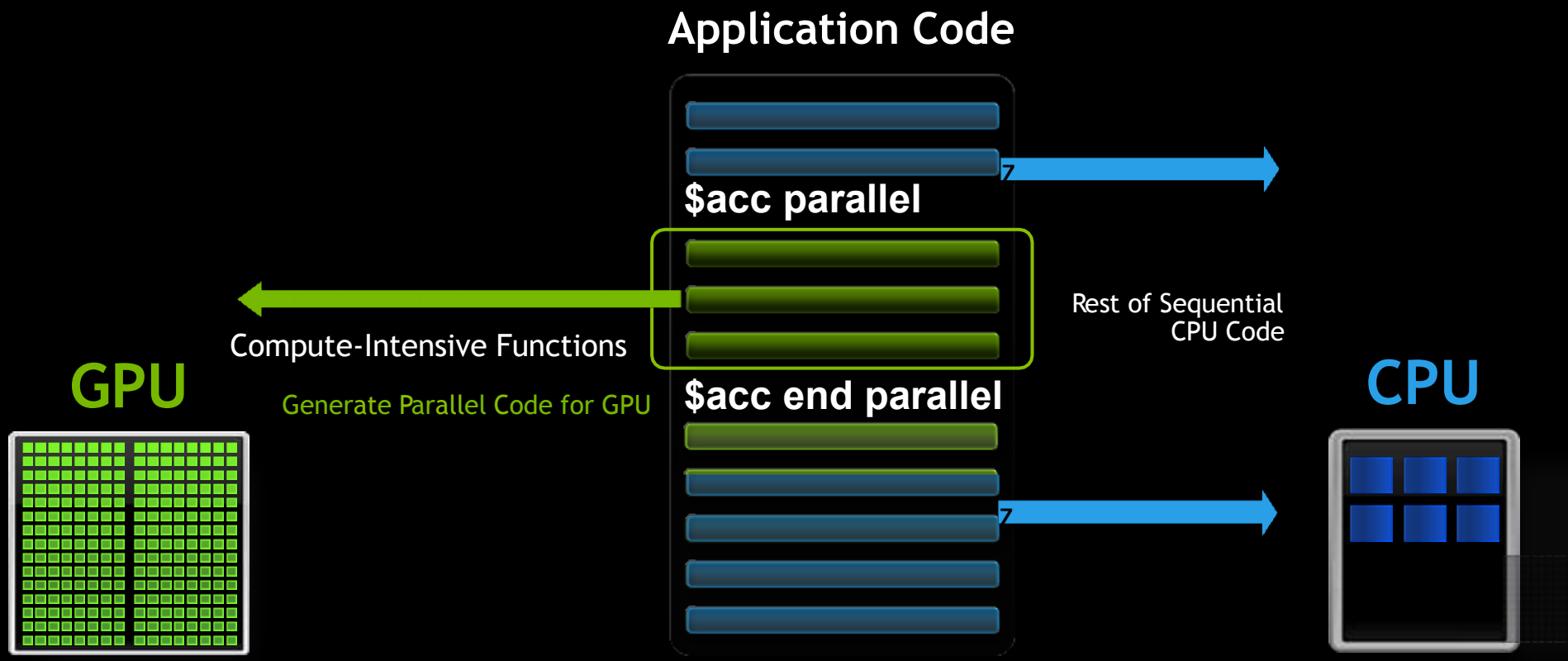


Compiler-directives
high programming level

Heterogeneous Programming on GPUs



OpenACC Execution Model



OpenACC Memory Model

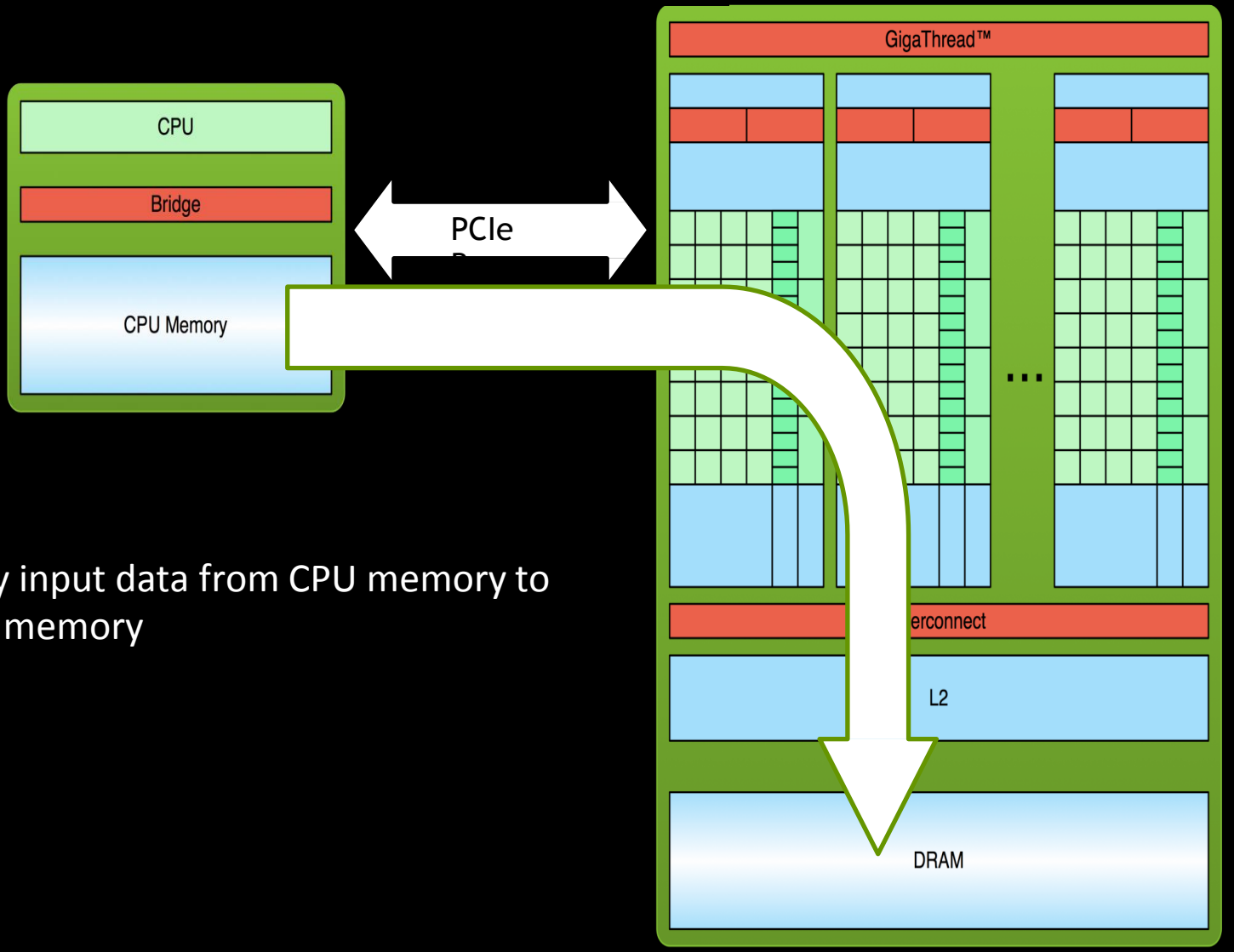
Two separate memory spaces between host and accelerator

- Data transfer by DMA transfers
- Hidden from the programmer in OpenACC, so beware:
  - Latency
  - Bandwidth
  - Limited device memory size

Accelerator:

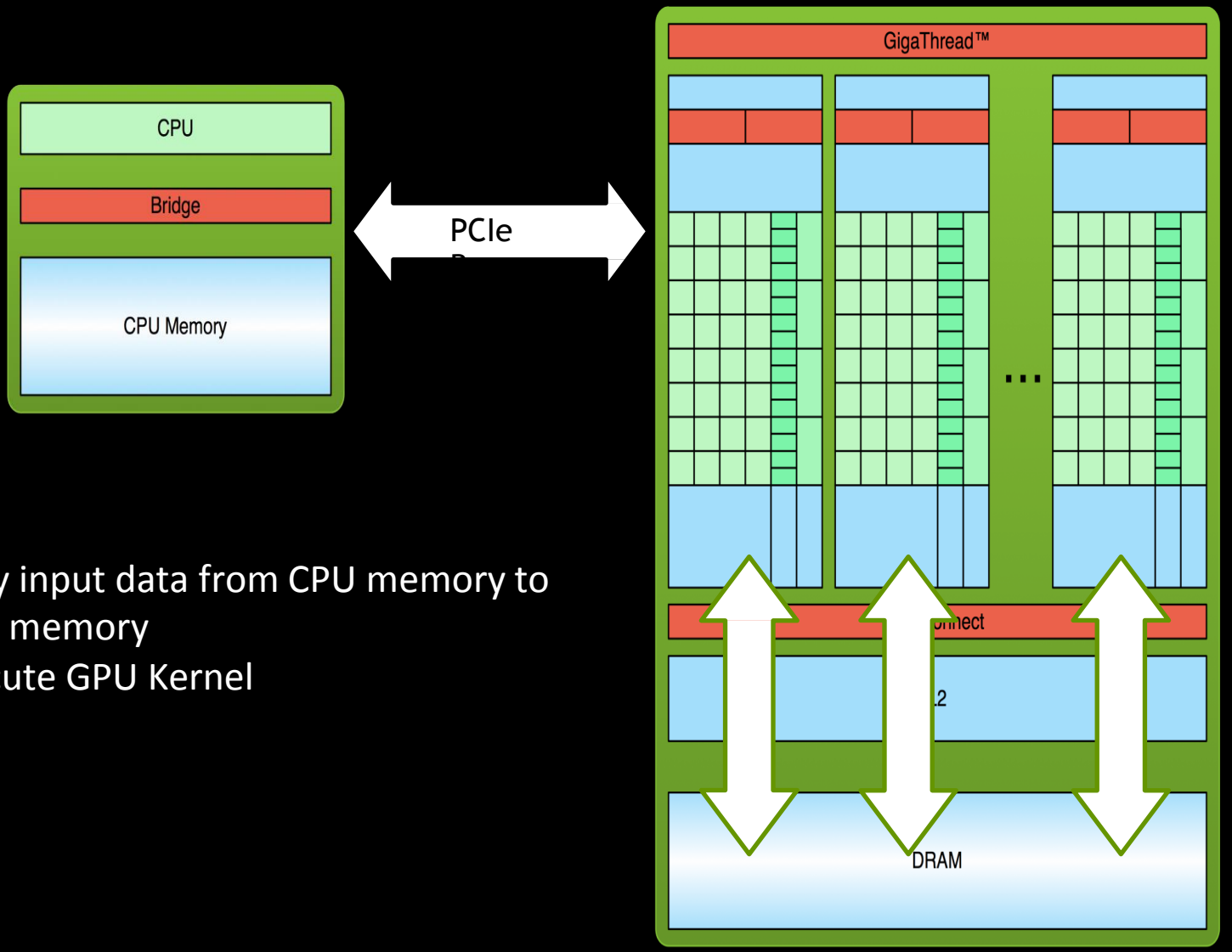
- No guarantee for memory coherence → beware of race conditions
- Cache management done by compiler, user may give hints

Data Flow



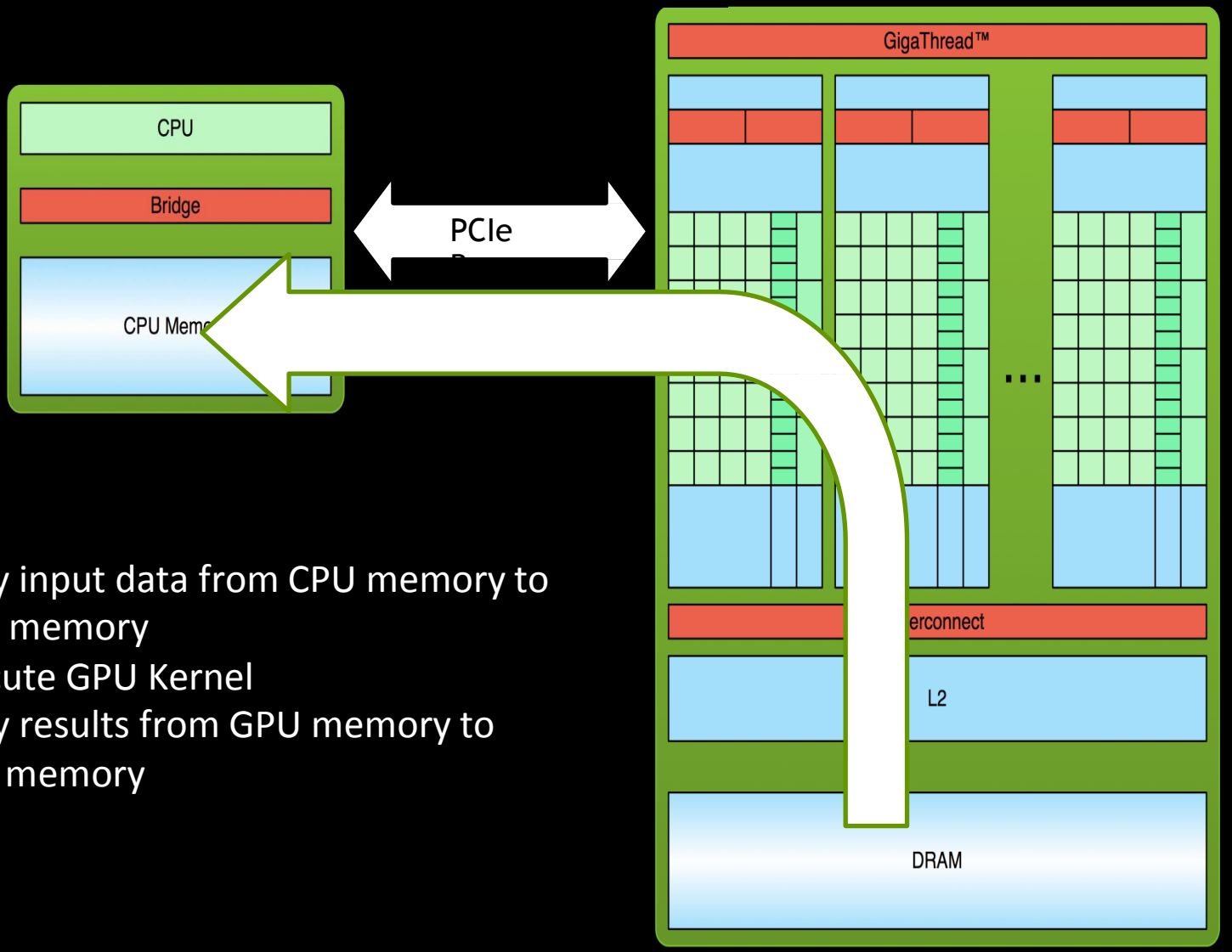
1. Copy input data from CPU memory to GPU memory

Data Flow



1. Copy input data from CPU memory to GPU memory
2. Execute GPU Kernel

Data Flow



1. Copy input data from CPU memory to GPU memory
2. Execute GPU Kernel
3. Copy results from GPU memory to CPU memory

Basic OpenACC directives

C/C++

```
#pragma acc directive-name [clause [[,] clause]...]
```

Fortran

```
!$acc directive-name [clause [[,] clause]...]
```

“Kernels / Parallel” Constructs

- Kernels

  C/C++

```
#pragma acc kernels [clauses]
```

  Fortran

```
!$acc kernels [clauses]
```

- Parallel

  C/C++

```
#pragma acc parallel loop [clauses]
```

  Fortran

```
!$acc parallel loop  [clauses]
```

“Data” Construct

Data: management of data transfer between host and device

C/C++

```
#pragma acc data [clauses]
```

Fortran

```
!$acc data [clauses]
```

“host_data” Construct

C/C++

```
#pragma acc kernels host_data use_device(list)
```

Fortran

```
!$acc kernels host_data use_device(list)
```

- Make the address of device data available on host
- Specified variable addresses refer to device memory
- Variables must be present on device
- Can only be used within a data region

OpenACC compilers

- PGI compiler for C, C++ and Fortran
- Cray CCE compilers for Cray systems
- CAPS compilers

OpenACC Standard

GPU Tools

Code performance increases with the deployment of GPU tools.

GPU languages

GPU directives

GPU libraries

CPU only

SAXPY

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

Saxpy_serial

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

int main(){
…
// Initialize vectors x, y
#pragma acc kernels
      for (int i = 0; i < n; ++i) {
        x[i] = 1.0f; y[i] = 0.0f;
      }
// Perform SAXPY
 saxpy_acc(n, a, x, y);
}
  …
```

Saxpy_openacc_v1

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

int main(){
…
// Initialize vectors x, y
#pragma acc parallel loop
      for (int i = 0; i < n; ++i) {
        x[i] = 1.0f; y[i] = 0.0f;
      }
// Perform SAXPY
 saxpy_acc(n, a, x, y);
}
  …
```

Parallel the loop

Saxpy_openacc_v2

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

int main(){
…
// Initialize vectors x, y
#pragma acc data create(x[0:n]) copyout(y[0:n])
   #pragma acc parallel loop
     for (int i = 0; i < n; ++i) {
       x[i] = 1.0f; y[i] = 0.0f;
     }
// Perform SAXPY
 saxpy_acc(n, a, x, y);
}
  …
```

Data management

Parallel the loop

cublasSaxpy from cuBlAS library

```
void cublasSaxpy( int          n,
                  const float  *alpha,
                  const float  *x,
                  int          incx,
                  float        *y,
                  int          incy)
```

- A function in the standard Basic Linear Algebra Subroutines (BLAS) library
- cuBLAS: GPU-accelerated drop-in library ready to be used on GPUs.

Saxpy_openacc_v2

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

int main(){
…
// Initialize vectors x, y
#pragma acc data create(x[0:n]) copyout(y[0:n])
   #pragma acc parallel loop
     for (int i = 0; i < n; ++i) {
       x[i] = 1.0f; y[i] = 0.0f;
     }
// Perform SAXPY
 saxpy_acc(n, a, x, y);
}
  …
```

Saxpy_cuBLAS

```
extern void
cublasSaxpy(int,float,float*,int,float*,int);

int main(){
…
// Initialize vectors x, y
#pragma acc data create(x[0:n]) copyout(y[0:n])
   #pragma acc parallel loop
     for (int i = 0; i < n; ++i) {
       x[i] = 1.0f; y[i] = 0.0f;
     }
// Perform SAXPY
  #pragma acc host_data use_device(x,y)
    cublasSaxpy(n, 2.0, x, 1, y, 1);
  …
```

http://docs.nvidia.com/cuda

Saxpy_openacc_v2

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

int main(){
…
// Initialize vectors x, y
#pragma acc data create(x[0:n]) copyout(y[0:n])
   #pragma acc parallel loop
     for (int i = 0; i < n; ++i) {
       x[i] = 1.0f; y[i] = 0.0f;
     }
// Perform SAXPY
 saxpy_acc(n, a, x, y);
}
  …
```

Saxpy_cuBLAS

```
extern void
cublasSaxpy(int,float,float*,int,float*,int);

int main(){
…
// Initialize vectors x, y
#pragma acc data create(x[0:n]) copyout(y[0:n])
   #pragma acc parallel loop
     for (int i = 0; i < n; ++i) {
       x[i] = 1.0f; y[i] = 0.0f;
     }
// Perform SAXPY
  #pragma acc deviceptr (x,y)
    cublasSaxpy(n, 2.0, x, 1, y, 1);
  …
```

http://docs.nvidia.com/cuda

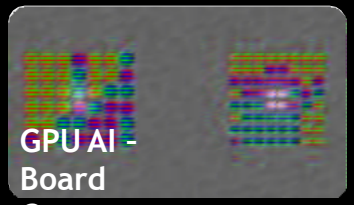
GPU Accelerated Libraries
"Drop-in" Acceleration for your Applications

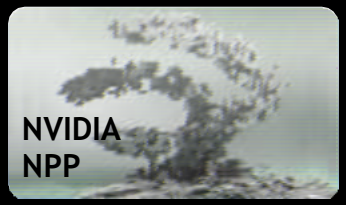
Linear Algebra
FFT, BLAS, SPARSE, Matrix

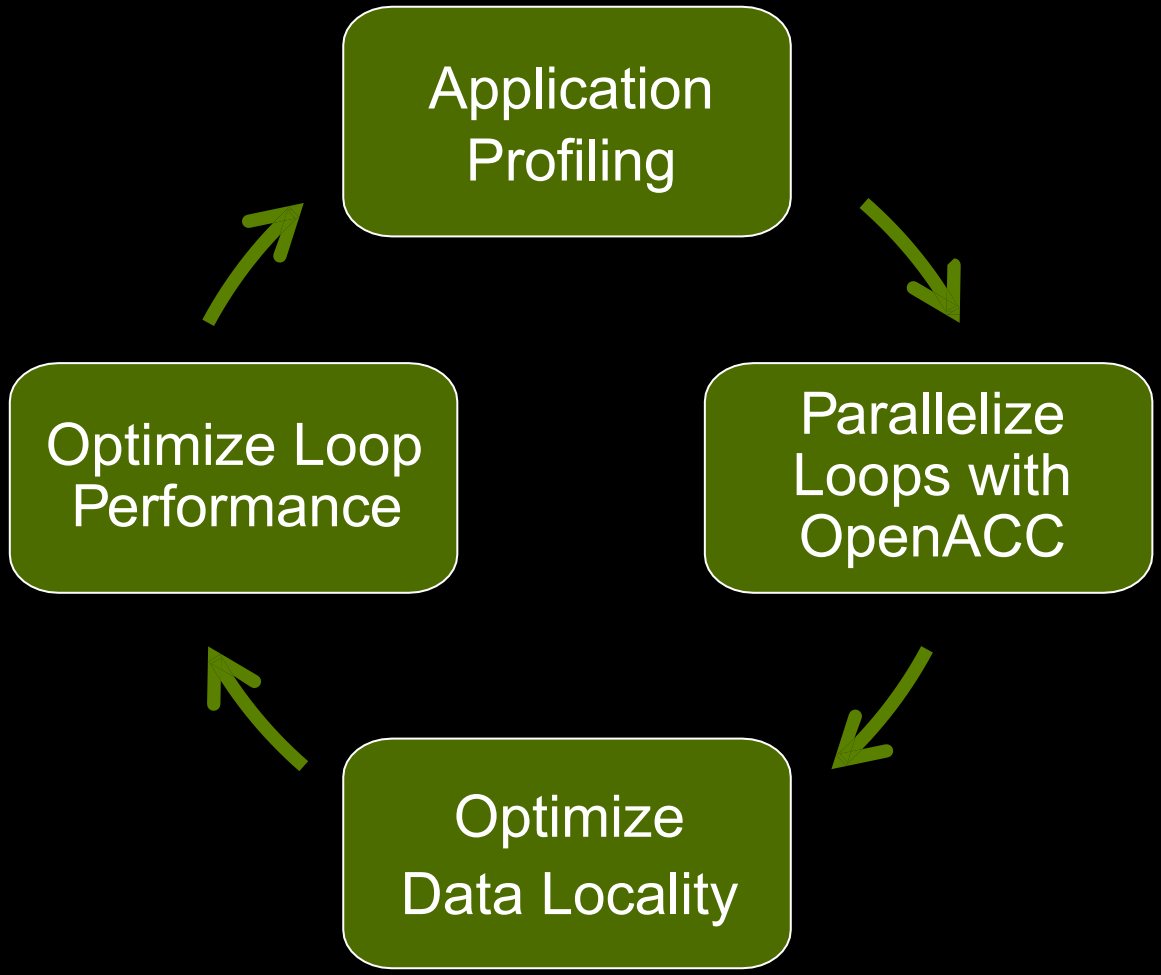
Numerical & Math
RAND, Statistics

Data Struct. & AI
Sort, Scan, Zero Sum

Visual Processing
Image & Video

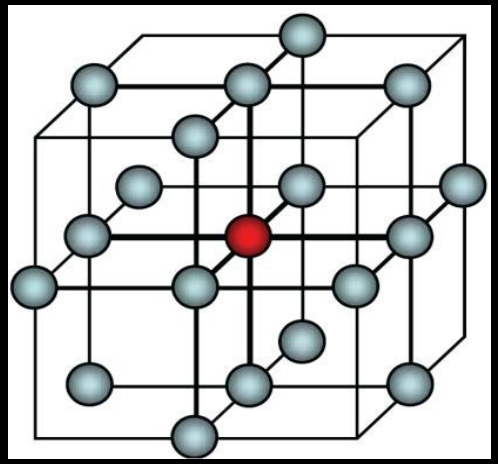


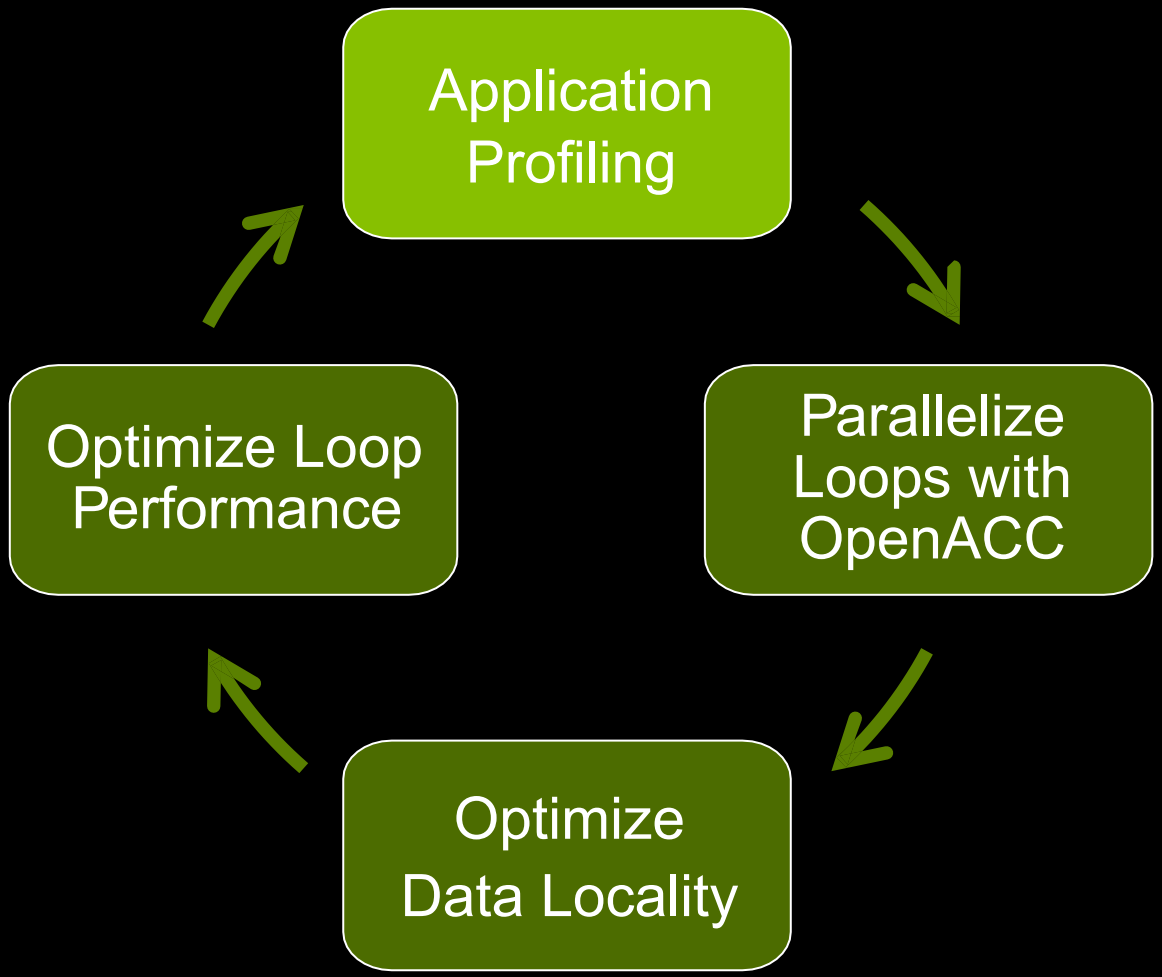



Application Profiling
Parallelize Loops with OpenACC
Optimize Data Locality
Optimize Loop Performance

The Himeno code

- 3D Poisson equation solver
  - Iterative loop evaluating 19-point stencil
  - Memory intensive, memory bandwidth bound
- Fortran and C implementations are available from http://accc.riken.jp/2467.htm
- The scalar version for simplicity
  - We will discuss the parallel version using OpenACC




Application Profiling
Parallelize Loops with OpenACC
Optimize Data Locality
Optimize Loop Performance

Application Profiling

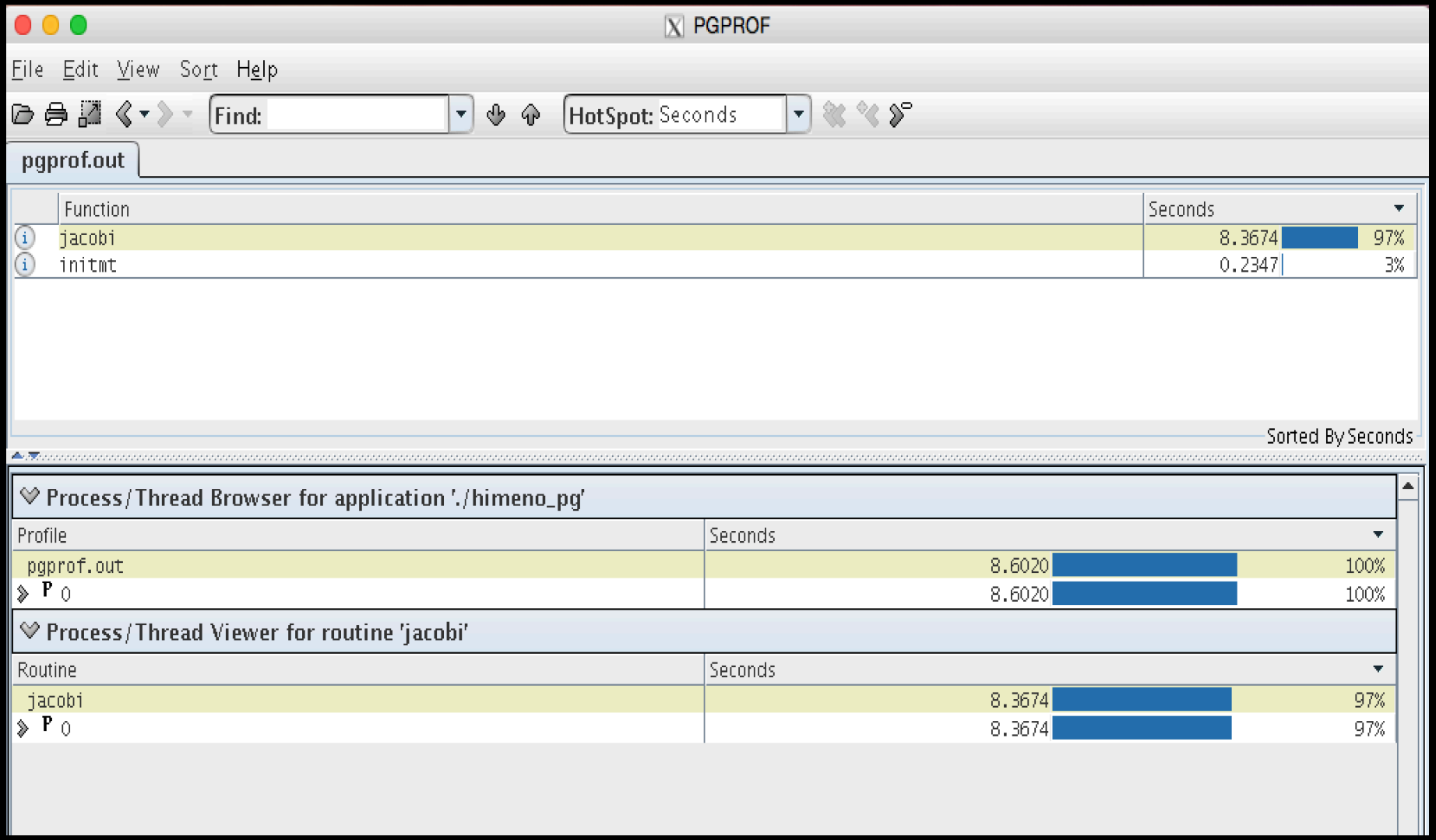
- pgprof - PGI performance profiler

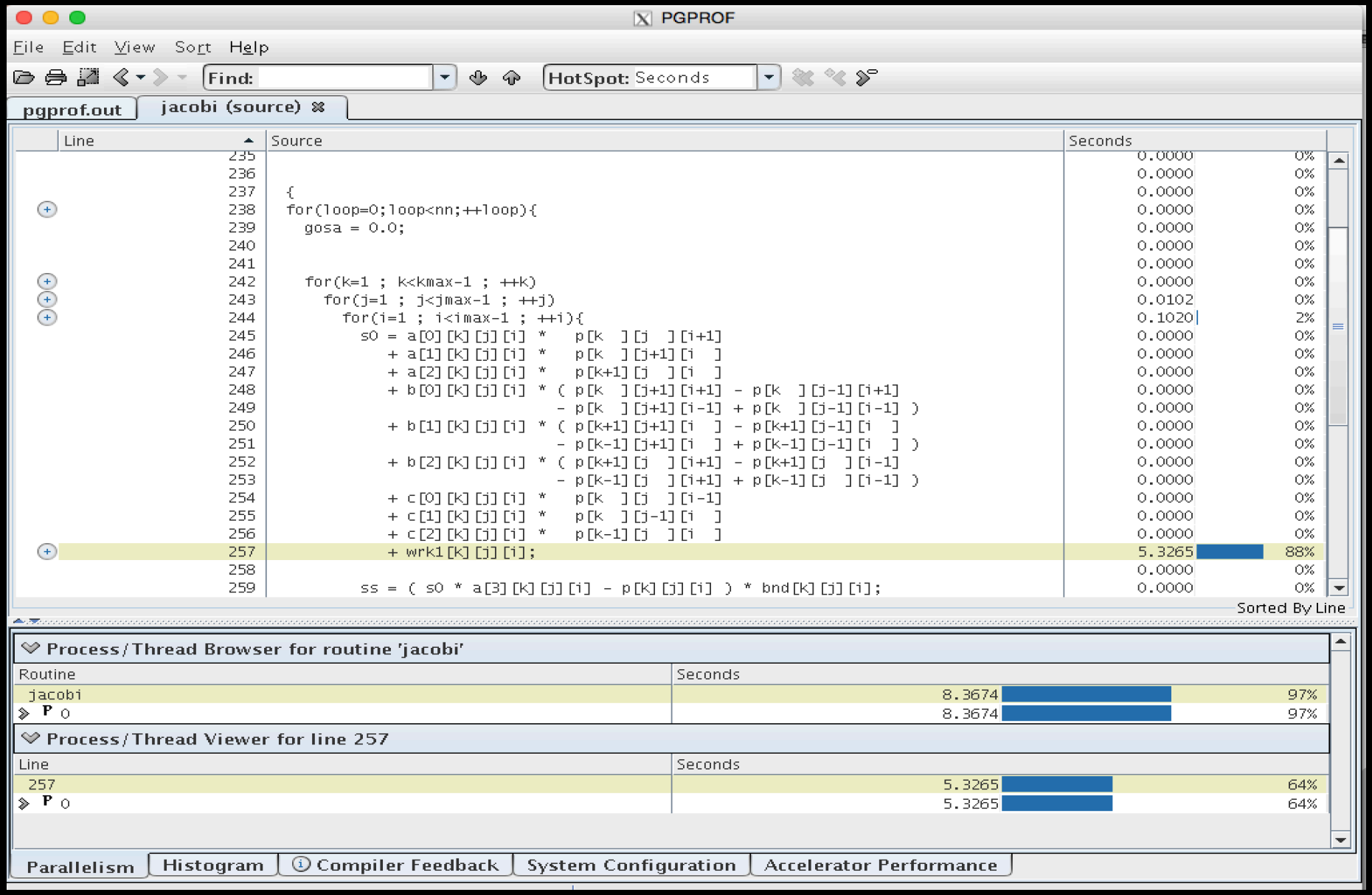
```
pgcc –Minfo=ccff –o yourcode_exe yourcode.c
pgcollect  yourcode_exe
pgprof –exe yourcode_exe
```

- gprof - GNU command line profiler

```
gcc –pg –o yourcode_exe yourcode.c
./yourcode_exe
gprof  yourcode_exe gmon.out > yourcode_pro.output
```

- nvprof - command line profiler -nvprof


PGPROF
File Edit View Sort Help
Find:
HotSpot: Seconds
pgprof.out
Function
Seconds
jacobi
8.3674
97%
initmt
0.2347
3%
Sorted By Seconds
Process/Thread Browser for application './himeno_pg'
Profile
Seconds
pgprof.out
8.6020
100%
P 0
8.6020
100%
Process/Thread Viewer for routine 'jacobi'
Routine
Seconds
jacobi
8.3674
97%
P 0
8.3674
97%


PGPROF
File Edit View Sort Help
Find:
HotSpot: Seconds
pgprof.out
jacobi (source)
Line
Source
Seconds
235 0.0000 0%
236 0.0000 0%
237 { 0.0000 0%
238 for(loop=0;loop<nn;++loop){ 0.0000 0%
239 gosa = 0.0; 0.0000 0%
240 0.0000 0%
241 0.0000 0%
242 for(k=1 ; k<kmax-1 ; ++k) 0.0000 0%
243 for(j=1 ; j<jmax-1 ; ++j) 0.0102 0%
244 for(i=1 ; i<imax-1 ; ++i){ 0.1020 2%
245 s0 = a[0][k][j][i] * p[k][j][i+1] 0.0000 0%
246 + a[1][k][j][i] * p[k][j+1][i] 0.0000 0%
247 + a[2][k][j][i] * p[k+1][j][i] 0.0000 0%
248 + b[0][k][j][i] * (p[k][j+1][i+1] - p[k][j-1][i+1] 0.0000 0%
249 - p[k][j+1][i-1] + p[k][j-1][i-1]) 0.0000 0%
250 + b[1][k][j][i] * (p[k+1][j+1][i] - p[k+1][j-1][i] 0.0000 0%
251 - p[k-1][j+1][i] + p[k-1][j-1][i]) 0.0000 0%
252 + b[2][k][j][i] * (p[k+1][j][i+1] - p[k+1][j][i-1] 0.0000 0%
253 - p[k-1][j][i+1] + p[k-1][j][i-1]) 0.0000 0%
254 + c[0][k][j][i] * p[k][j][i-1] 0.0000 0%
255 + c[1][k][j][i] * p[k][j-1][i] 0.0000 0%
256 + c[2][k][j][i] * p[k-1][j][i] 0.0000 0%
257 + wrk1[k][j][i]; 5.3265 88%
258 0.0000 0%
259 ss = (s0 * a[3][k][j][i] - p[k][j][i]) * bnd[k][j][i]; 0.0000 0%
Sorted By Line
Process/Thread Browser for routine 'jacobi'
Routine Seconds
jacobi 8.3674 97%
P 0 8.3674 97%
Process/Thread Viewer for line 257
Line Seconds
257 5.3265 64%
P 0 5.3265 64%
Parallelism
Histogram
Compiler Feedback
System Configuration
Accelerator Performance

Application Profiling

- pgprof - PGI visual profiler

```
pgcc –Minfo=ccff –o yourcode_exe yourcode.c
pgcollect  yourcode_exe
pgprof –exe yourcode_exe
```

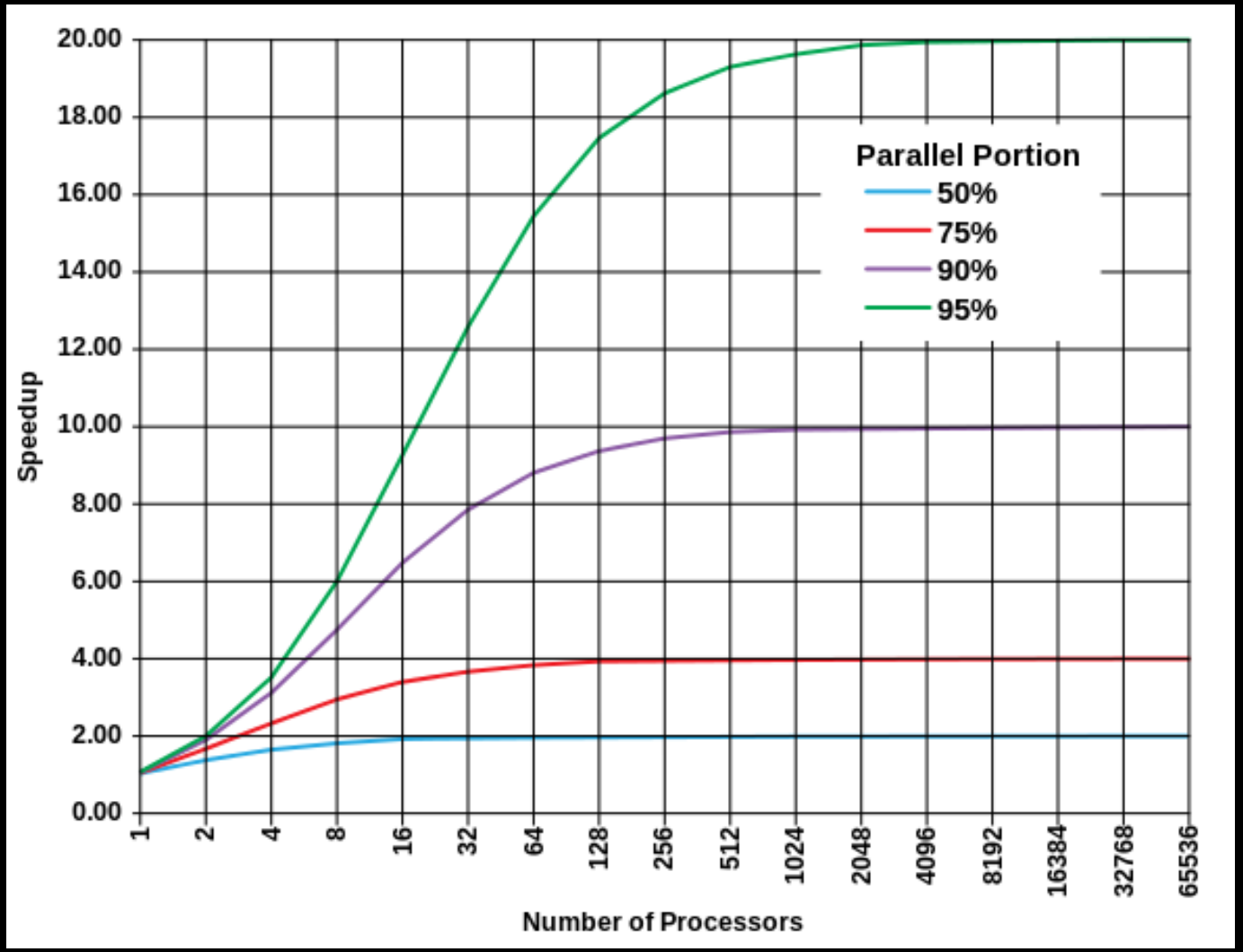
- gprof - GNU command line profiler

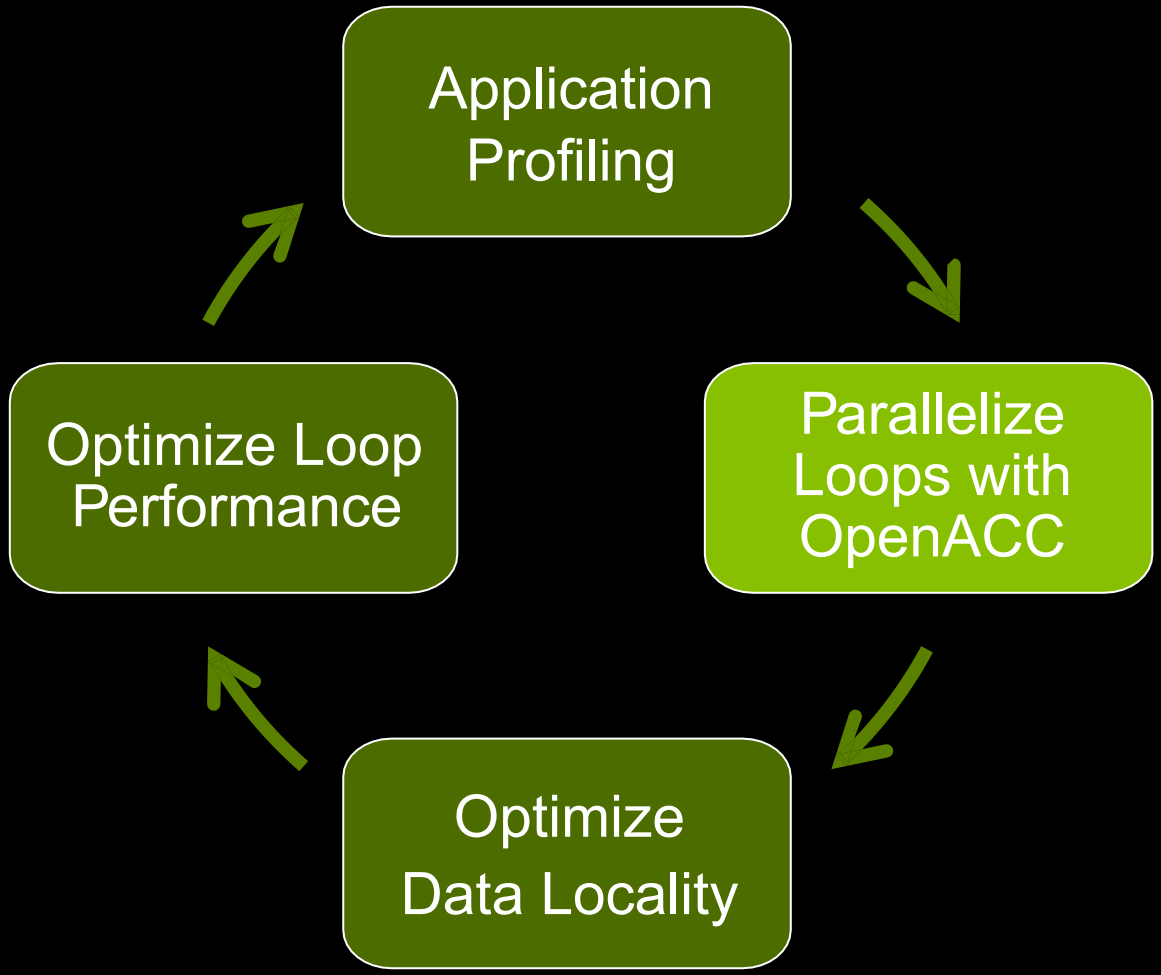
```
gcc –pg –o yourcode_exe yourcode.c
./yourcode_exe
gprof  yourcode_exe gmon.out > yourcode_pro.output
```

- nvprof - command line profiler -nvprof

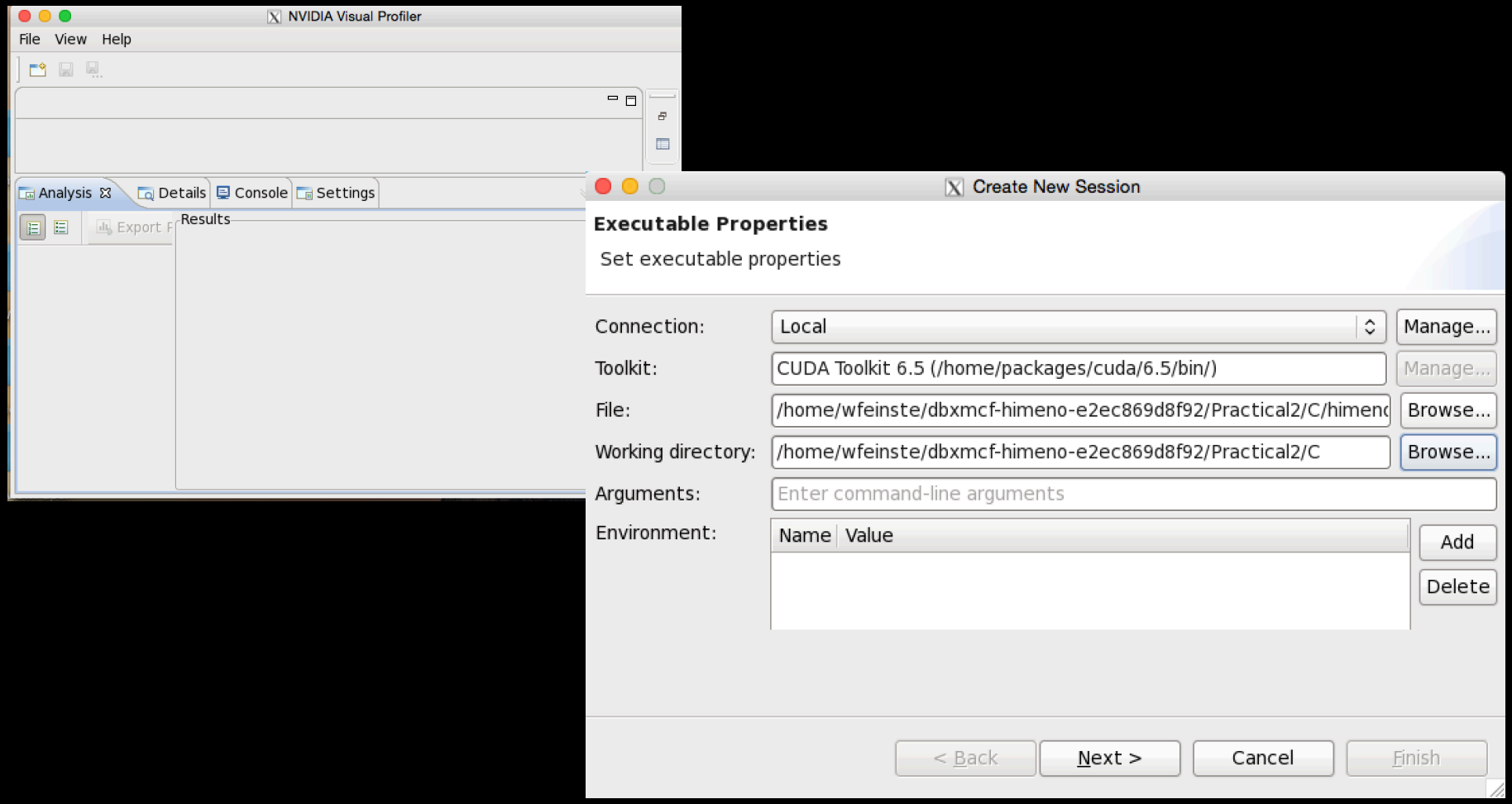
```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 42.39      0.39     0.39      986     0.40     0.40  dp_
 26.09      0.63     0.24   129471     0.00     0.00  cal_tmscore_
 16.30      0.78     0.15     1004     0.15     0.53  get_score_
  7.61      0.85     0.07   132633     0.00     0.00  u3b_
  7.61      0.92     0.07     1006     0.07     0.38  tmsearch_
  0.00      0.92     0.00      497     0.00     0.00  make_sec_
  0.00      0.92     0.00      378     0.00     0.00  get_ngl_
  0.00      0.92     0.00       90     0.00     0.00  getbest_
  0.00      0.92     0.00       90     0.00     9.18  make_iter_
  0.00      0.92     0.00       25     0.00     0.00  filter_
  0.00      0.92     0.00       18     0.00    51.04  caltmsc_
  0.00      0.92     0.00       18     0.00     0.00  fillinvmap_
  0.00      0.92     0.00       18     0.00     0.00  get_initial3_
  0.00      0.92     0.00       18     0.00     0.00  get_score1_
  0.00      0.92     0.00       18     0.00     0.00  recomputefmatrix_
  0.00      0.92     0.00        6     0.00     0.00  fragdp_
  0.00      0.92     0.00        1     0.00   920.01  MAIN__
  0.00      0.92     0.00        1     0.00     0.00  assignssp_
  0.00      0.92     0.00        1     0.00   918.66  calbesttm_
  0.00      0.92     0.00        1     0.00   919.26  fragscan_
  0.00      0.92     0.00        1     0.00     0.00  smooth_
  0.00      0.92     0.00        1     0.00   919.26  super_align_

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.
```

Amdahl's Law

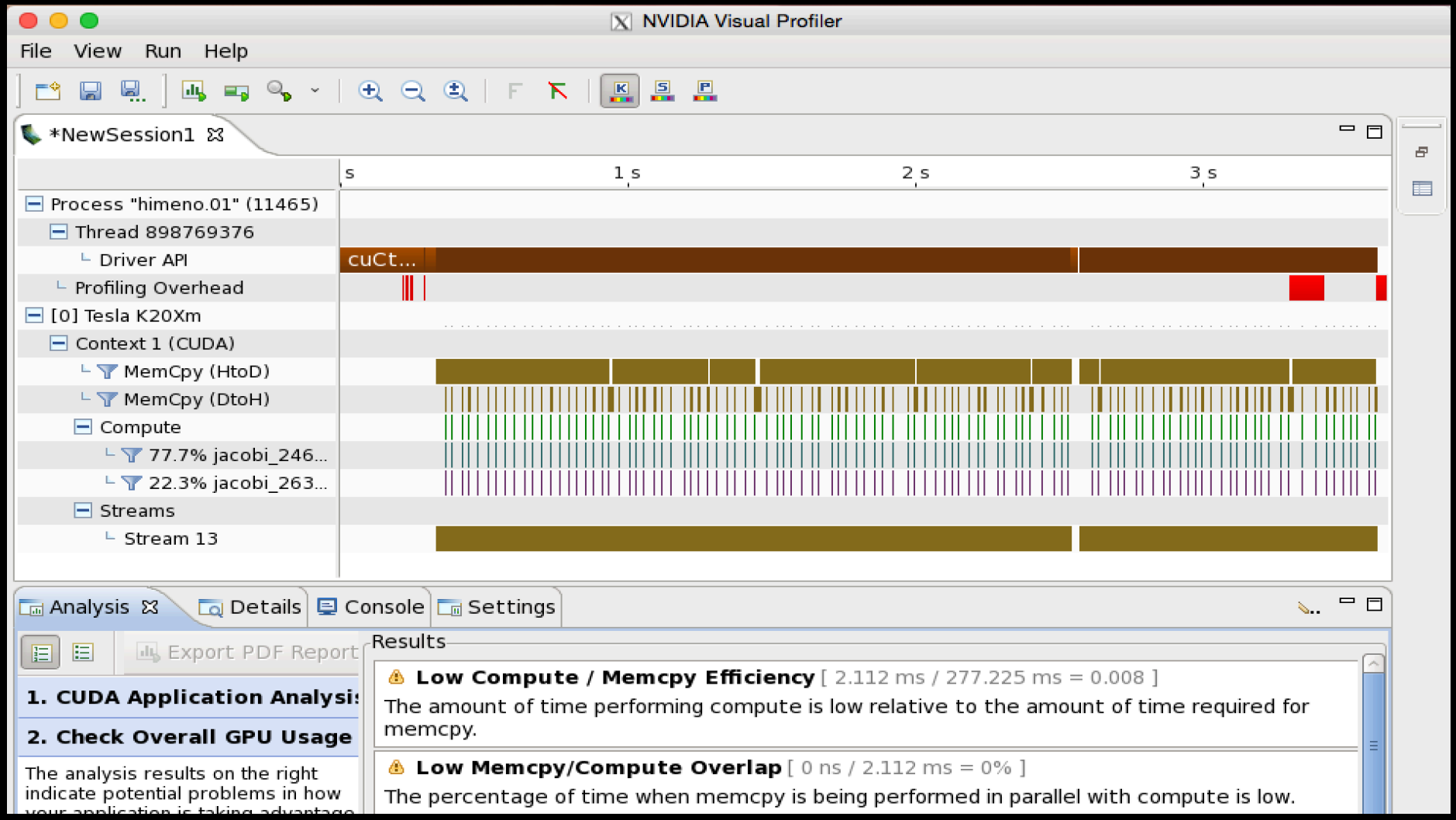

Speedup
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
Parallel Portion
50%
75%
90%
95%
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Number of Processors


Application Profiling
Parallelize Loops with OpenACC
Optimize Data Locality
Optimize Loop Performance

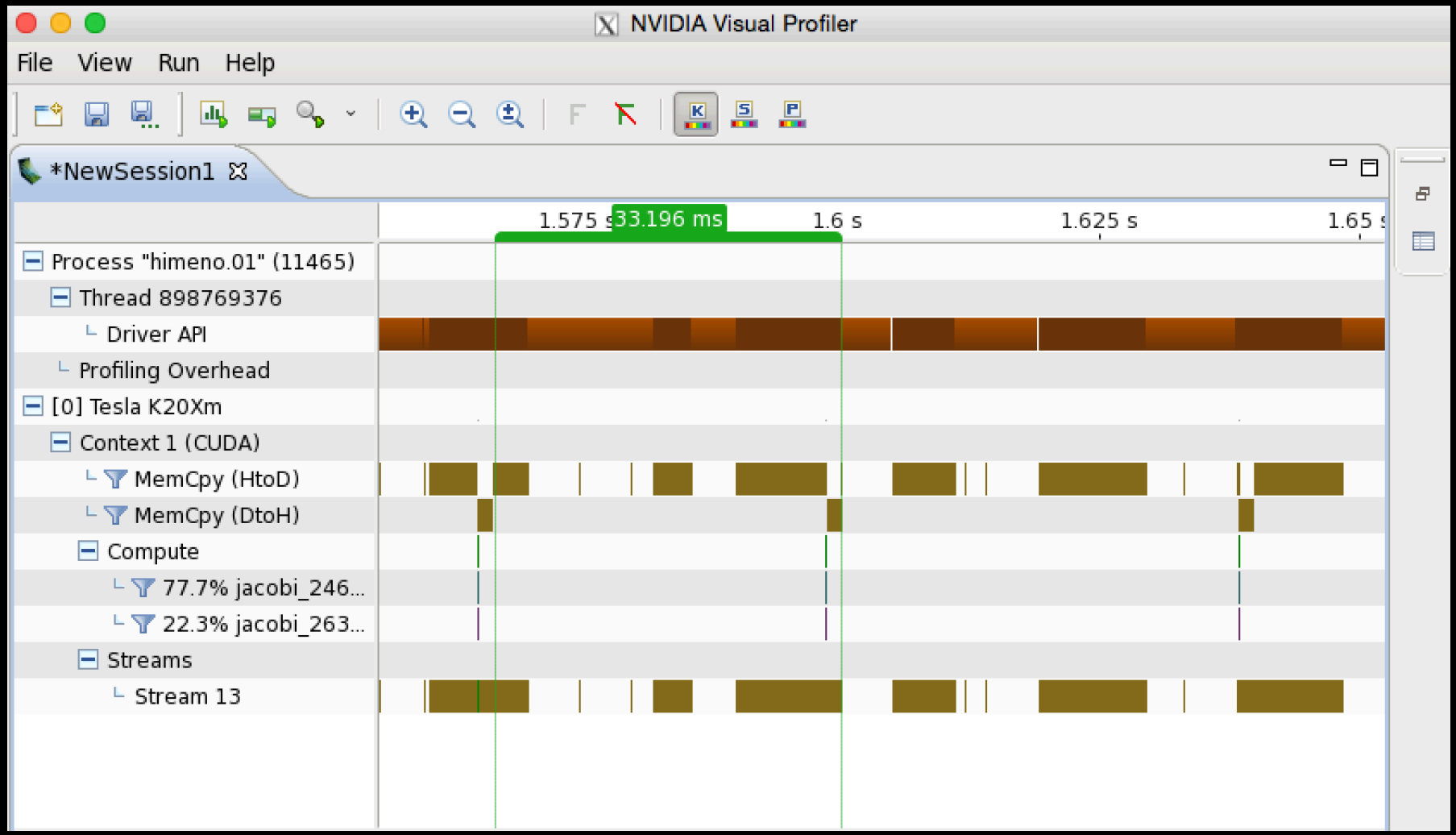
Performance Profiling via NVVP



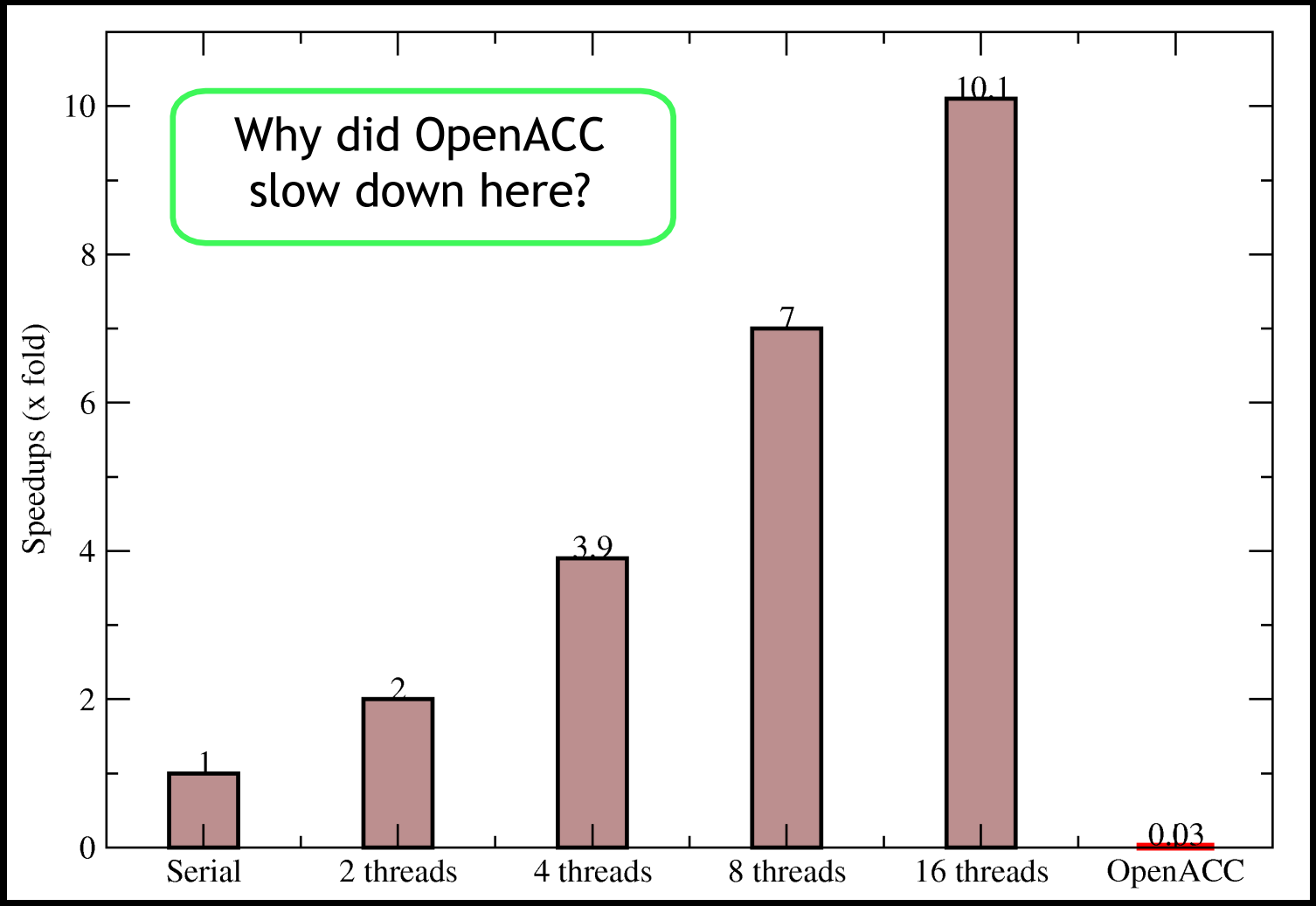
Performance Profiling via NVVP

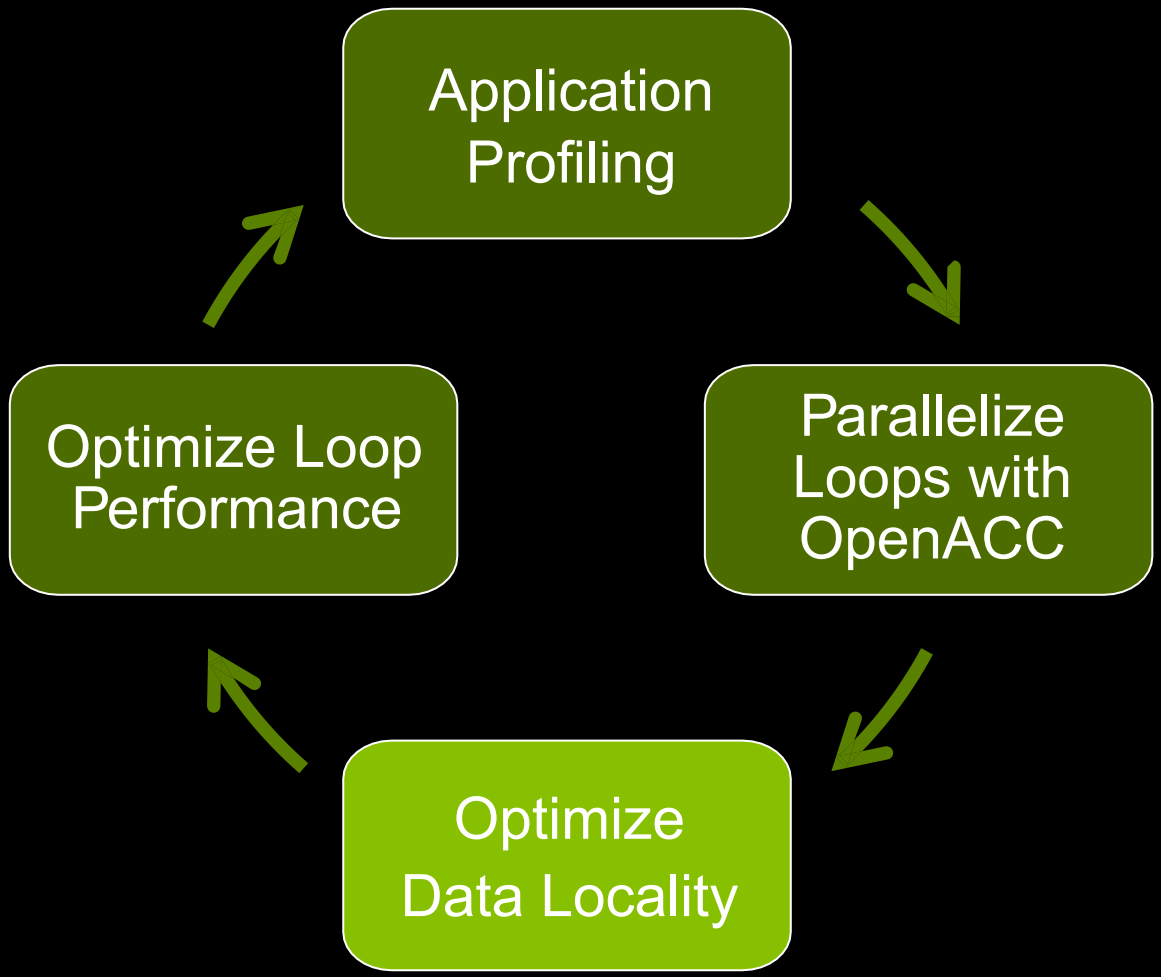


Performance Profiling via NVVP



Performance

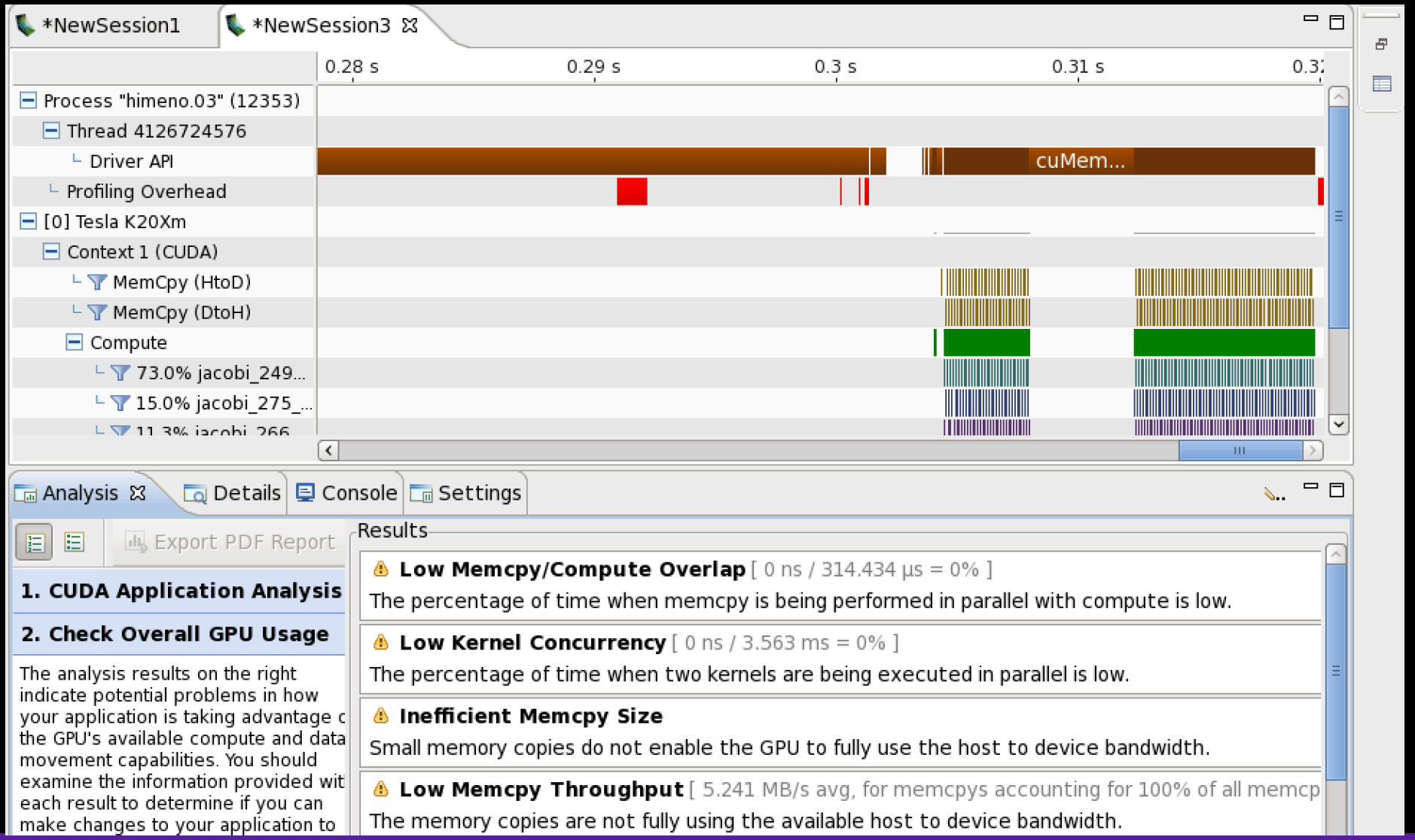



Application Profiling
Parallelize Loops with OpenACC
Optimize Data Locality
Optimize Loop Performance

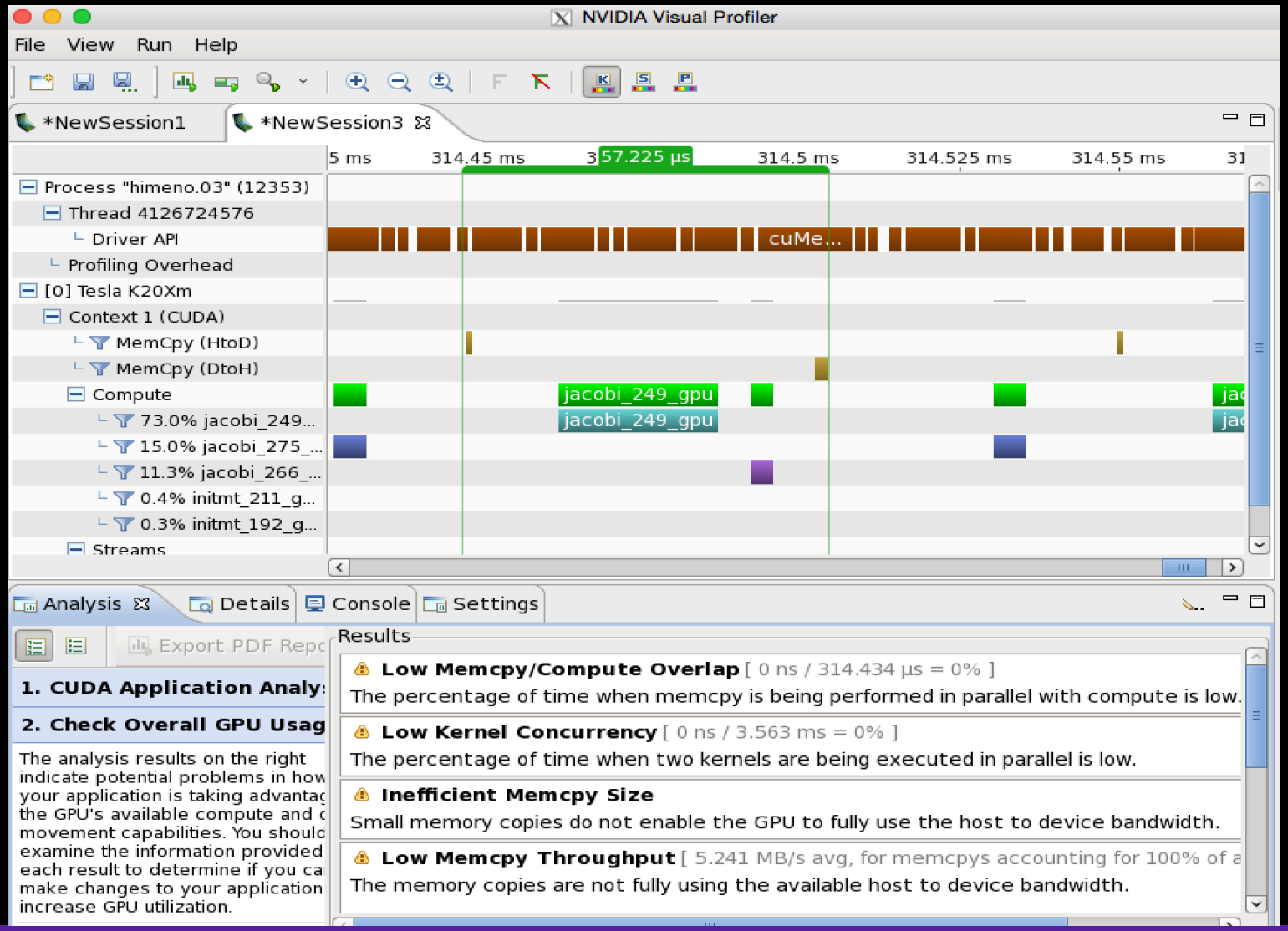
Data transfer

- Data movement is expensive causing bottleneck to performance
- Minimize data movement
- Data caching
  - #pragma acc data copyin/copyout/copy
    - Allocate memory on device and copy data from host to device, or back , or both
  - #pragma acc data present

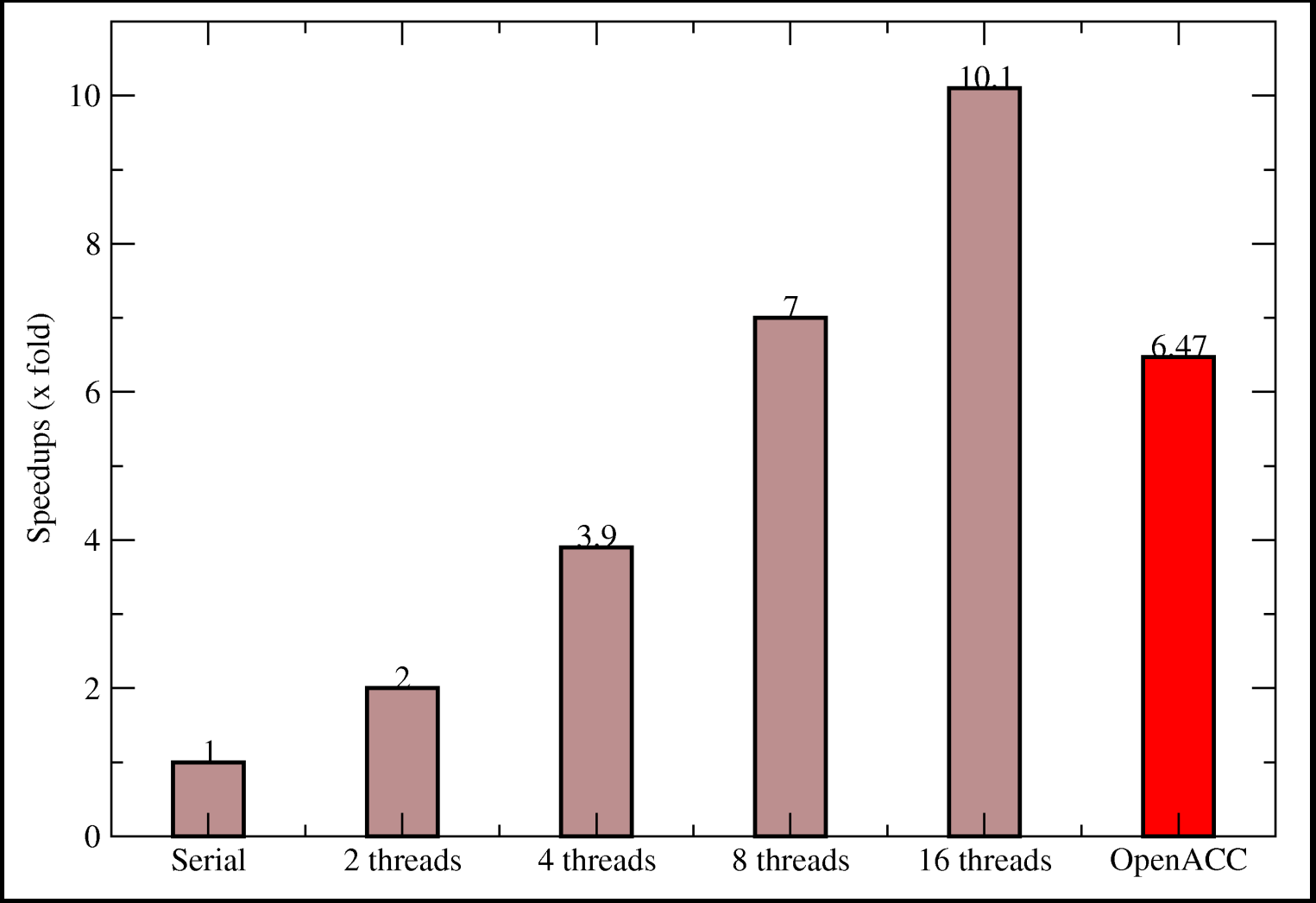
Improved performance with better data locality

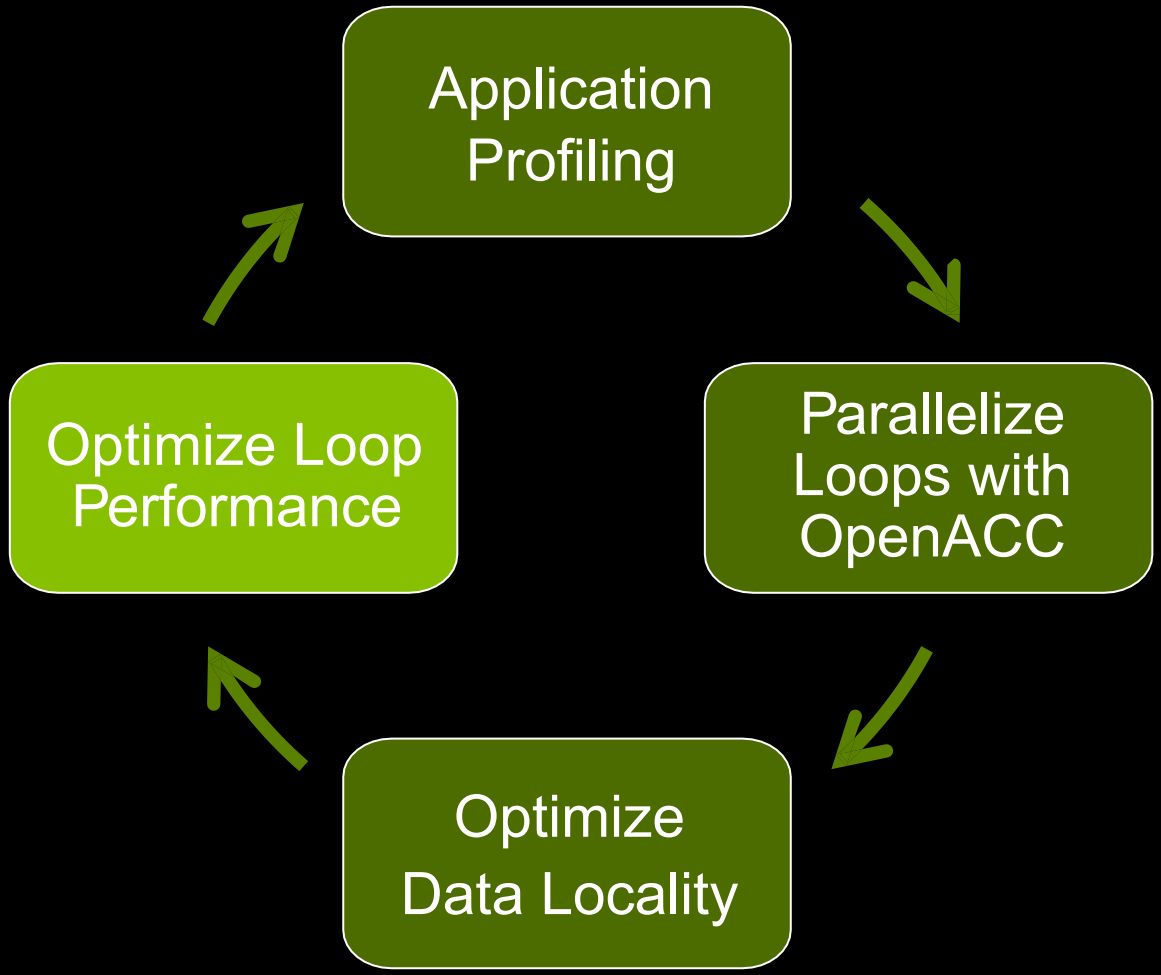


Improved performance with better data locality



Improved performance with better data locality




Application Profiling
Parallelize Loops with OpenACC
Optimize Data Locality
Optimize Loop Performance

Three Levels of Parallelism

OpenACC provides more detailed control over parallelization via gang, worker, and vector clauses

- Gang:
  - Share iterations across the gangs (grids) of a parallel region
- Worker:
  - Share iterations across the workers (warps) of the gang
- Vector:
  - Execute the iterations in SIMD mode

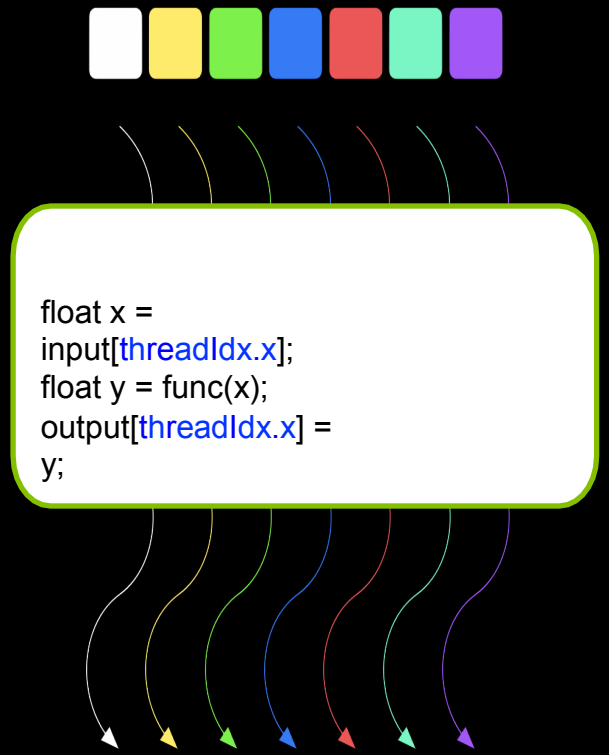
CUDA Kernels: Parallel Threads

A kernel is a function executed on the GPU as an array of threads in parallel

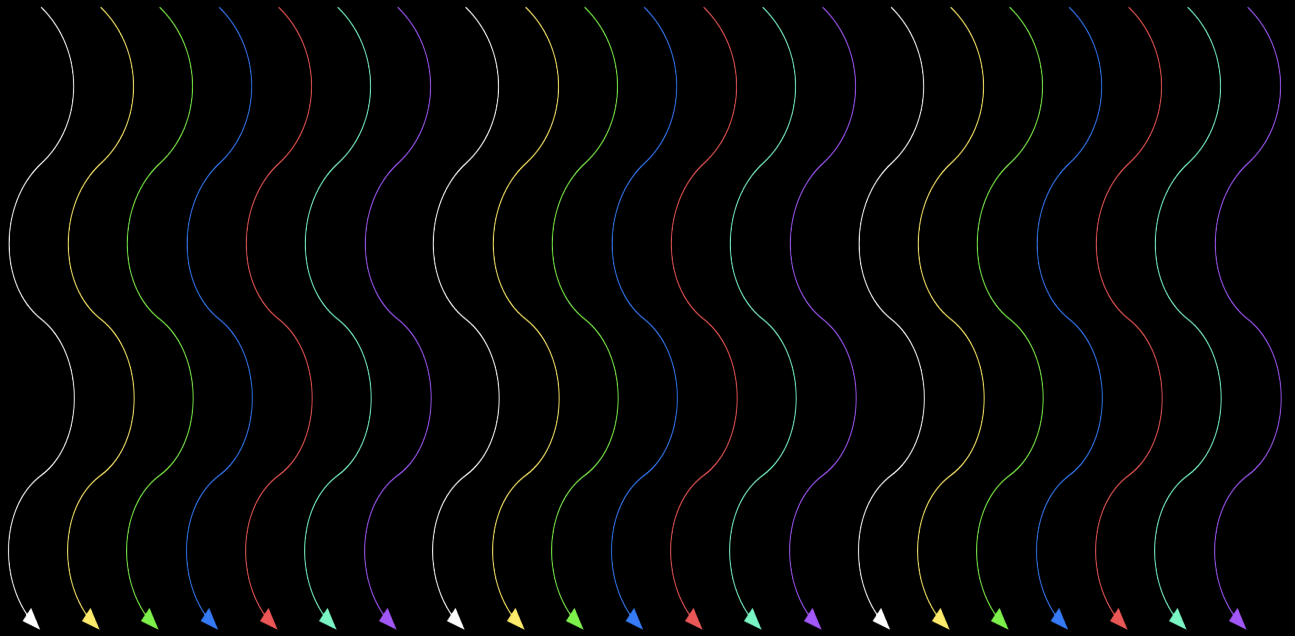
All threads execute the same code, can take different paths

Each thread has an ID Select

- input/output data
- Control decisions

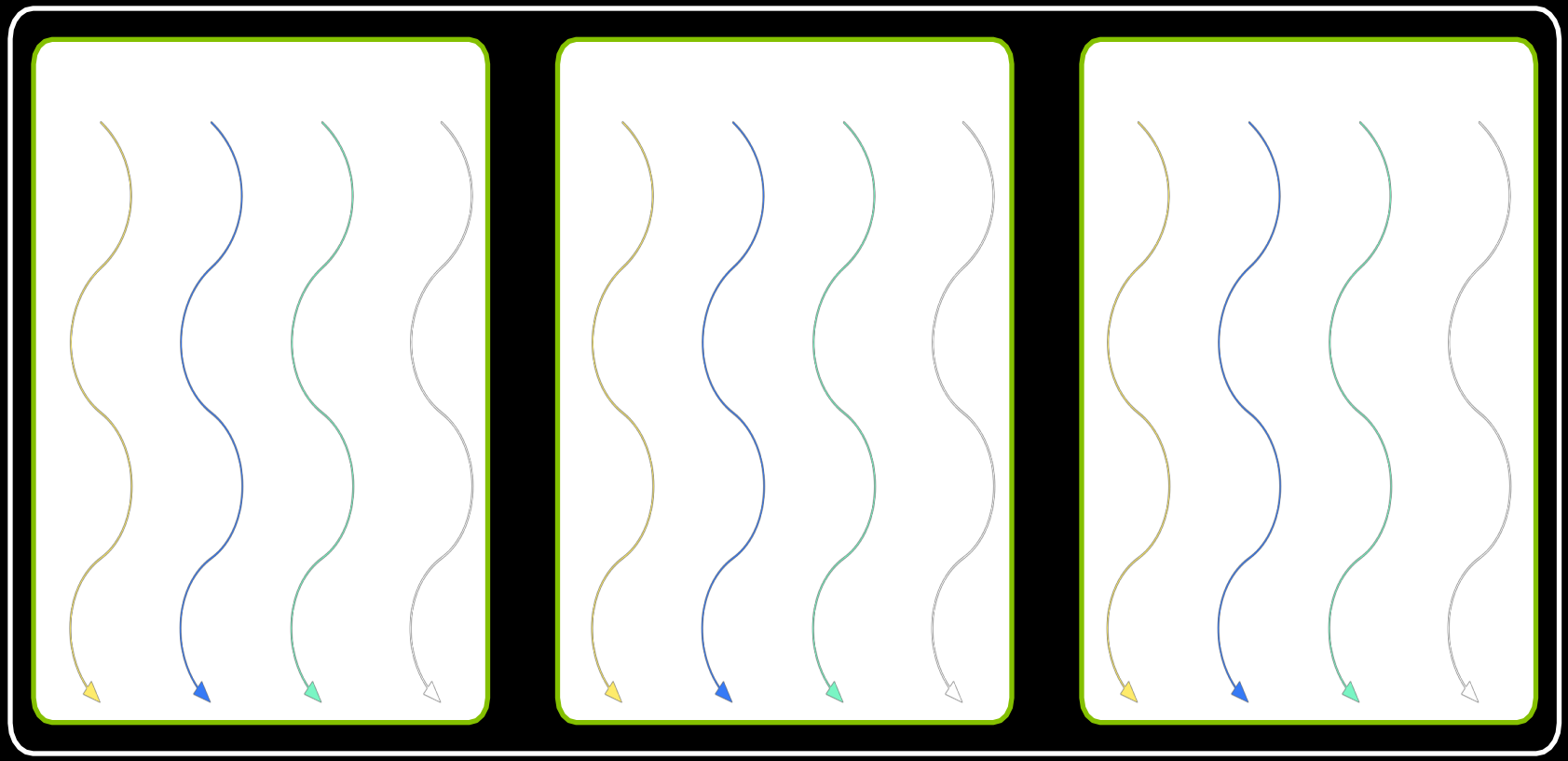


CUDA Kernels: Subdivide into Blocks



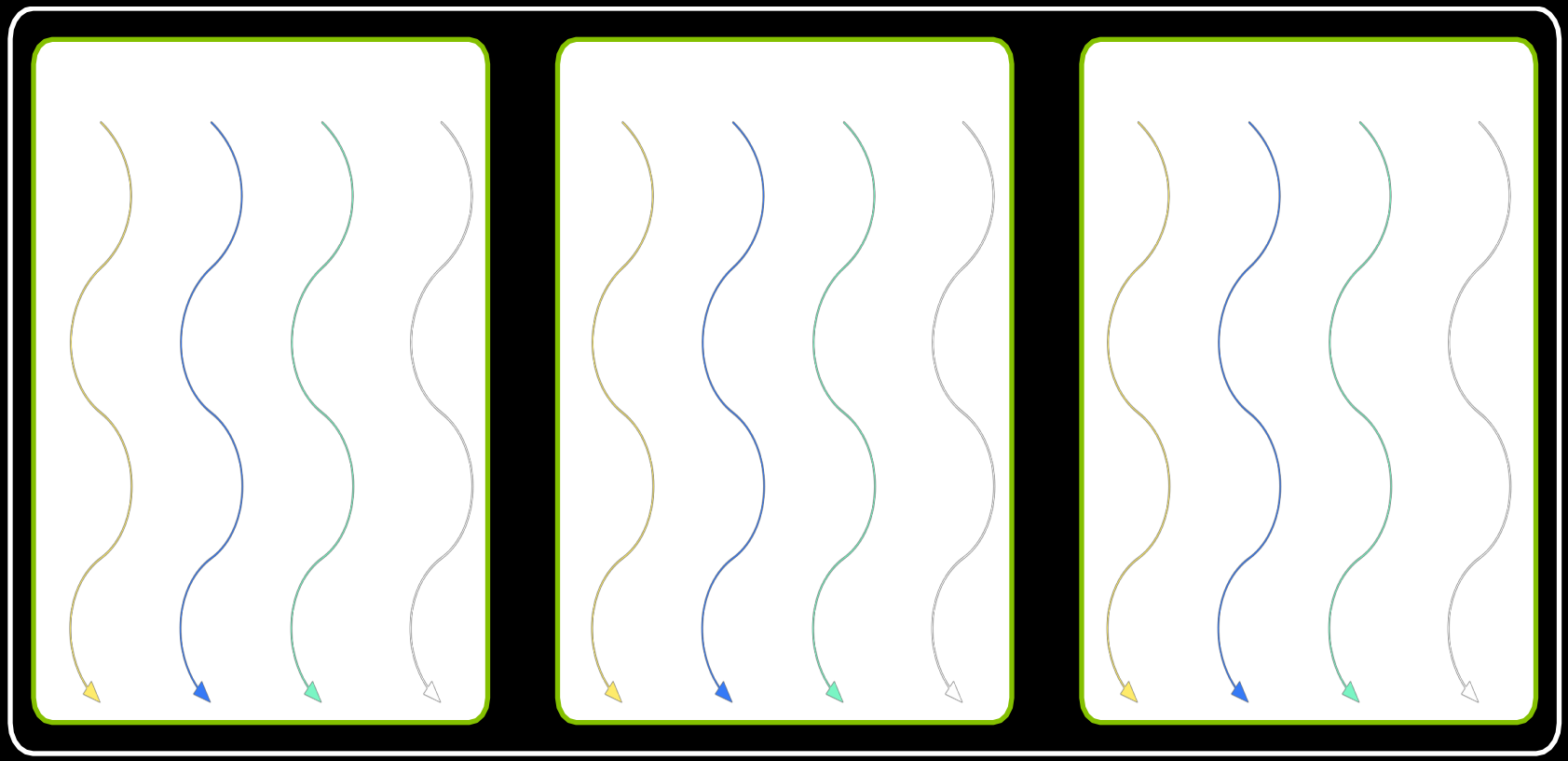
- Threads are grouped into blocks

CUDA Kernels: Subdivide into Blocks



- Threads are grouped into blocks
- Blocks are grouped into a grid

CUDA Kernels: Subdivide into Blocks



- Threads are grouped into blocks
- Blocks are grouped into a grid
- A kernel is executed as a grid of blocks of threads

MAPPING OPENACC TO CUDA

OpenACC Execution Model on CUDA

The OpenACC execution model has three levels:
gang, worker, and vector

For GPUs, the mapping is implementation-dependent.
Some possibilities:

gang==block, worker==warp, and vector==threads of a warp

Depends on what the compiler thinks is the best mapping for the problem

OpenACC Execution Model on CUDA

The OpenACC execution model has three levels:
gang, worker, and vector

For GPUs, the mapping is implementation-dependent.

...But explicitly specifying that a given loop should map to gangs, workers, and/or vectors is optional anyway

- Further specifying the *number* of gangs/workers/vectors is also optional
- So why do it? To tune the code to fit a particular target architecture in a straightforward and easily re-tuned way.

Three Levels of Parallelism

C/C++

```
#pragma acc parallel [num_gangs()/
num_workers()/vectoor_length()]
```

Fortran

```
!$acc parallel [num_gangs()/
num_workers()/vectoor_length()]
```

C/C++

```
#pragma acc loop[(num_gangs)/
(num_workers)/(vectoor_length)]
```

Fortran

```
!$acc parallel [(num_gangs)/
(num_workers)/(vectoor_length)]
```

Multiple GPUs card on a single node?

Device Management

- Internal control variables (ICVs):
  - *Acc-device-type-var* → Controls which type of accelerator is used
  - *Acc-device-num-var* → Controls which accelerator device is used
- Setting ICVs by API calls
  - *acc_set_device_type()* *acc_set_device_num()*
- Querying of ICVs
  - *acc_get_device_type() acc_get_device_num()*

Device Management

`acc_get_num_devices`

- Returns the number of accelerator devices attached to host and the argument specifies type of devices to count

C:

- `int acc_get_num_devices(acc_device_t)`

Fortran:

- `Integer function acc_get_num_devices(devicetype)`

Device Management

`acc_set_device_num`

- Sets ICV `ACC_DEVICE_NUM`
- Specifies which device of given type to use for next region Can not be called in a parallel, kernels or data region

C:

- `Void acc_set_device_num(int,acc_device_t)`

Fortran:

- `Subroutine acc_set_device_num(devicenum,devicetype)`

Device Management

- `Acc_get_device_num`
  - Return value of ICV `ACC_DEVICE_NUM`
  - Return which device of given type to use for next region
  - Can not be called in a parallel, kernels or data region
- C:
  - `Void acc_get_device_num(acc_device_t)`
- Fortran:
  - `Subroutine acc_get_device_num(devicetype)`

Directive-based programming on single node with multi-GPU cards

SAXPY Code

```
    // initialization
    for (i = 0; i < n; i++){
        x[i] = 1.0; y[i] = 2.0;
    }
    // calculation
    for (i = 0; i < n; i++){
        y[i] = a*x[i] + y[i]*2;
    }
}
```

Directive-based programming on single node with multi-GPU cards

```
// get # of GPU cards on this node
int gpu_ct=acc_get_num_devices(acc_device_nvidia);
// create one thread for each GPU kernel
#pragma omp parallel private(tid) num_threads(gpu_ct)
{
  // Obtain thread id
  tid = omp_get_thread_num();
  // assign one kernel to one OpenMP thread
  acc_set_device_num(tid +1 , acc_device_nvidia);
  #pragma acc data create(x[0:n],y[0:n]) copyin(a)
    #pragma acc kernels loop
        for (i = 0; i < n; i++){
            x[i] = 1.0; y[i] = 2.0;
        }
    #pragma acc kernels loop
        for (i = 0; i < n; i++){
            y[i] = a*x[i] + y[i]*2;
        }
 }//end of omp parallel
```

Directive-based programming on single node with multi-GPU cards

- OpenACC only supports one GPU
- Hybrid model:
  - OpenACC + OpenMP to support multi-GPU parallel programming
- Limitations
  - Lack direct device-to-device communications

Conclusions

- OpenACC is a powerful programming model using compiler directives
- Progressive, productive code porting
- Portable and easy to maintain
- Interoperability
- Advanced features provide deeper control

Introduction to OpenACC
PartII Lab

Getting Started

Connect to shelob cluster:

ssh username@shelob.hpc.lsu.edu

Extract the lab to your account:

tar xzvf /home/user/himeno.tar.gz

Change to the lab directory:

cd himeno

Request a interactive node

qsub –I –A allocation –lwalltime=2:00:00 –lnodes=1:ppn=16

Login in to the interactive node

ssh –X shelobxxx

Exercise 1

Goal: code profiling to identify the target for parallelization (use your own code would be great)

cd Practical1

pgprof: PGI visual profiler

pgcc –Minfo=ccff mycode.c –o mycode

pgcollect mycode

pgprof –exe mycode

Exercise 1

Goal: code profiling to identify the target for parallelization (use your own code would be great)

gprof: GNU profiler

gcc mycode.c –o mycode –pg

./mycode

gprof mycode gmon.out >mycode_profile.output

Exercise 2

Goal: Identify hot spots in your code to improve performance

cd Practical2

Compile source code (e.g. version 1)

make ver=01

Check performance at command line (-Minfo=accel is turned on)

./himeno.01

Use nvvp visual profiler by typing:

nvvp

Exercise 3

Goal: use nvvp to fine tune your code for better performance via using more OpenACC directives

cd Practical3

Compile the source code (e.g. version 1)

make ver=01

Use nvvp visual profiler by typing:

nvvp

Exercise 4

Goal: use OpenACC with GPU-enabled library

cd Practical4

pgprof: PGI visual profiler

pgcc –Minfo=ccff mycode.c –o mycode

pgcollect mycode

pgprof –exe mycode

Exercise 5

Goal: use multi-GPUs cards on a single node

cd Practical5