Introduction to Accelerated Computing with OpenACC

Jeff Larkin, OLCF Users' Group Meeting 2016

AGENDA

NVIDIA Introduction

Introduction to Accelerated Computing

Accelerated Computing with OpenACC

Next Steps

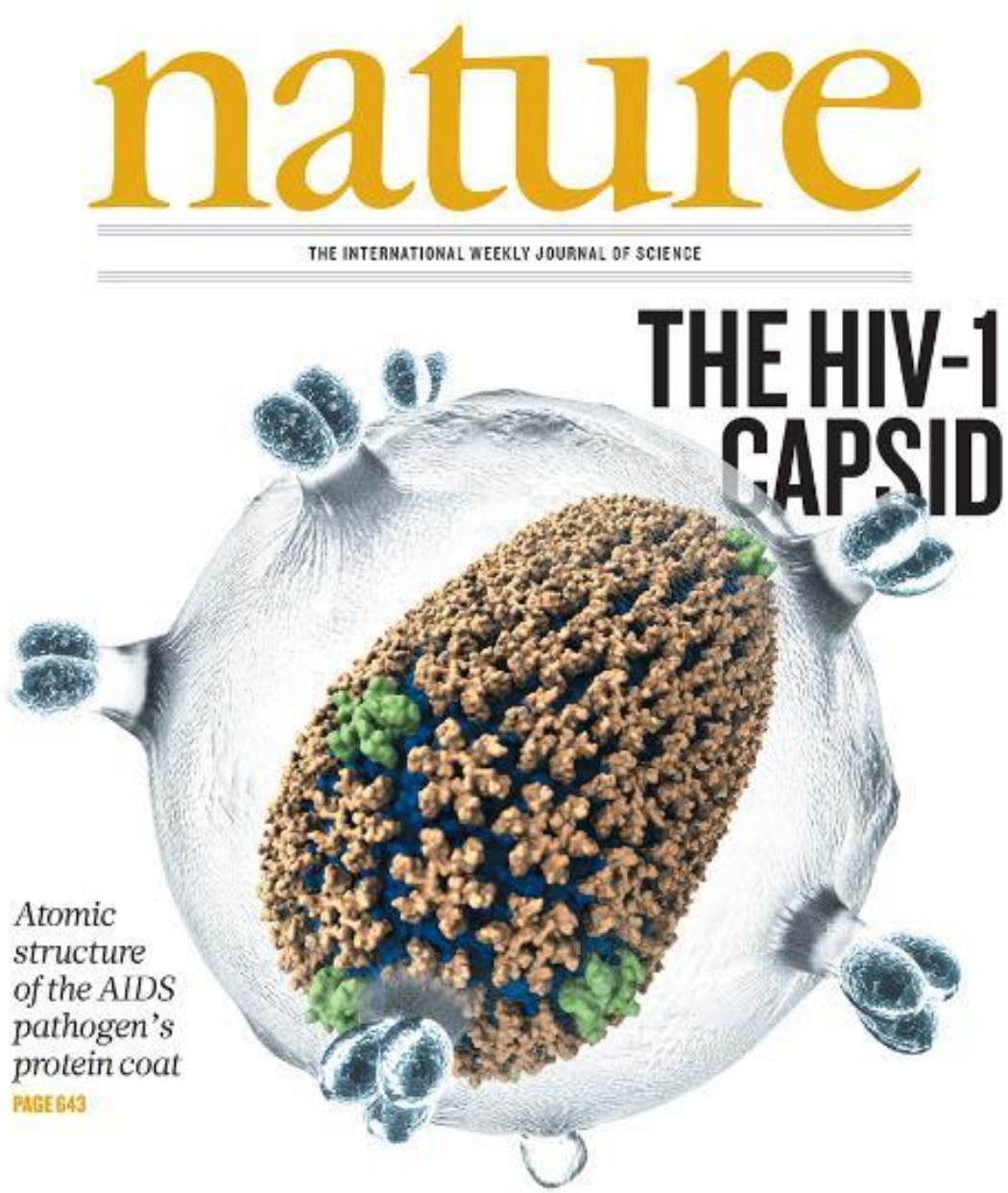
NVIDIA Introduction

THE WORLD LEADER IN VISUAL COMPUTING

Tesla Accelerates Discoveries

Using a supercomputer powered by the Tesla Platform with over 3,000 Tesla accelerators, University of Illinois scientists performed the first all-atom simulation of the HIV virus and discovered the chemical structure of its capsid — "the perfect target for fighting the infection."

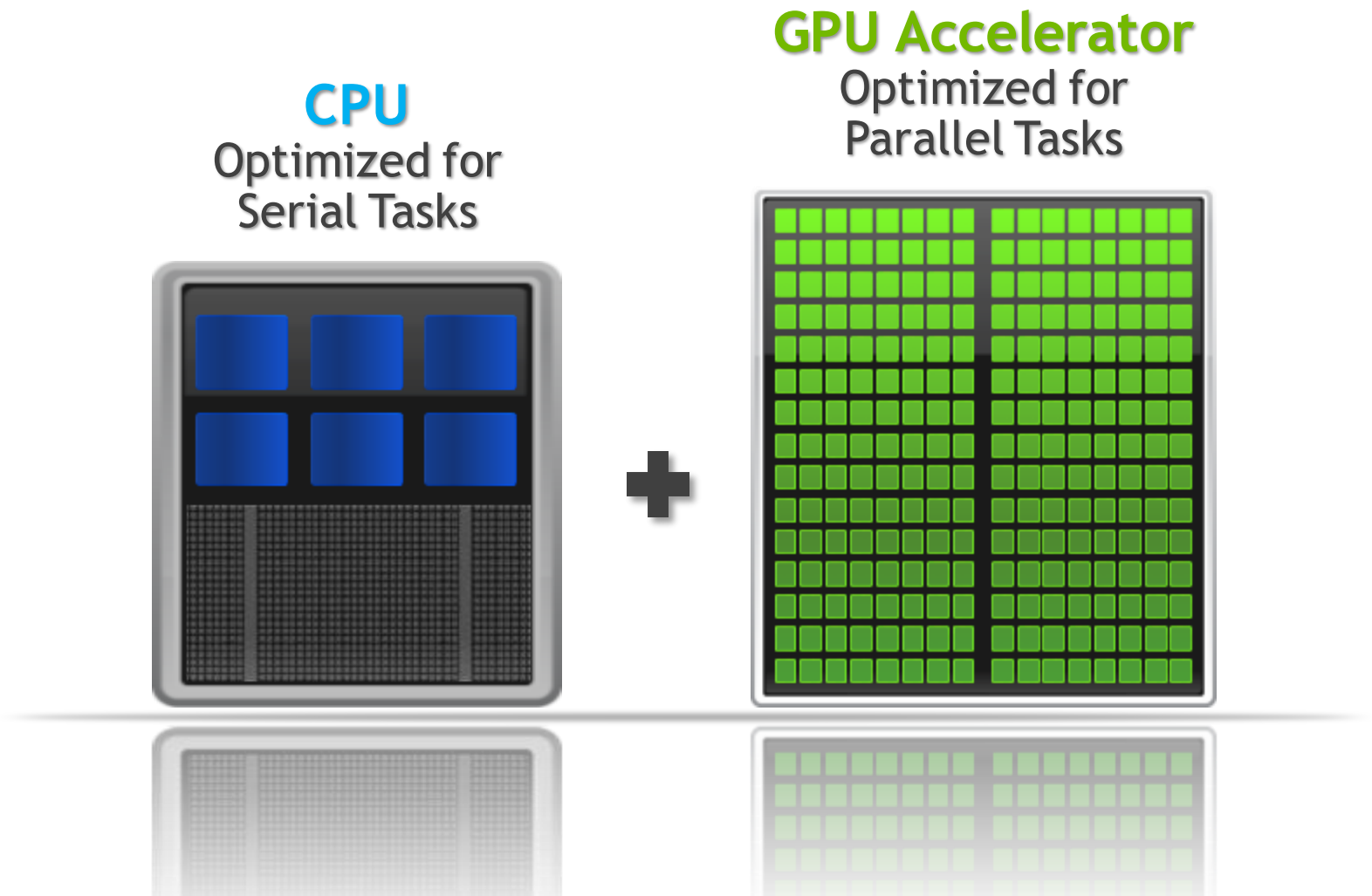
Without GPU, the supercomputer would need to be 5x larger for similar performance.



Introduction to Accelerated Computing

ACCELERATED COMPUTING

10X PERFORMANCE & 5X ENERGY EFFICIENCY FOR HPC

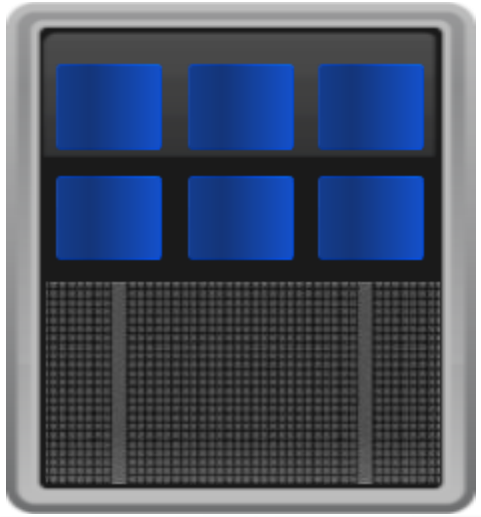


ACCELERATED COMPUTING

10X PERFORMANCE & 5X ENERGY EFFICIENCY FOR HPC

CPU

Optimized for
Serial Tasks



CPU Strengths

- Very large main memory
- Very fast clock speeds
- Latency optimized via large caches
- Small number of threads can run very quickly

CPU Weaknesses

- Relatively low memory bandwidth
- Cache misses very costly
- Low performance/watt

ACCELERATED COMPUTING

10X PERFORMANCE & 5X ENERGY EFFICIENCY FOR HPC

GPU Strengths

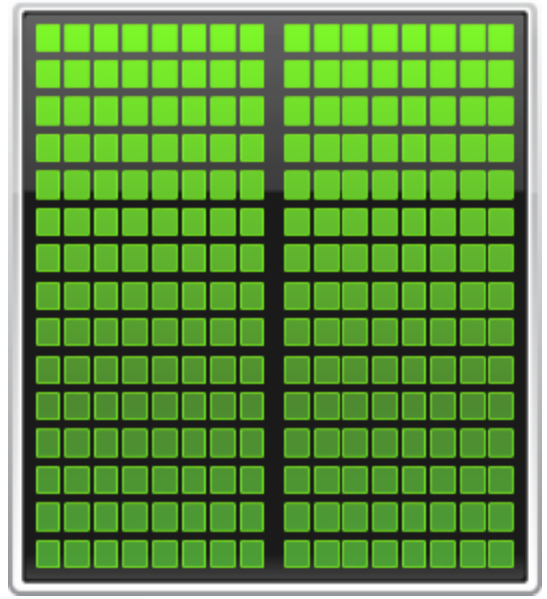
- High bandwidth main memory
- Latency tolerant via parallelism
- Significantly more compute resources
- High throughput
- High performance/watt

GPU Weaknesses

- Relatively low memory capacity
- Low per-thread performance

GPU Accelerator

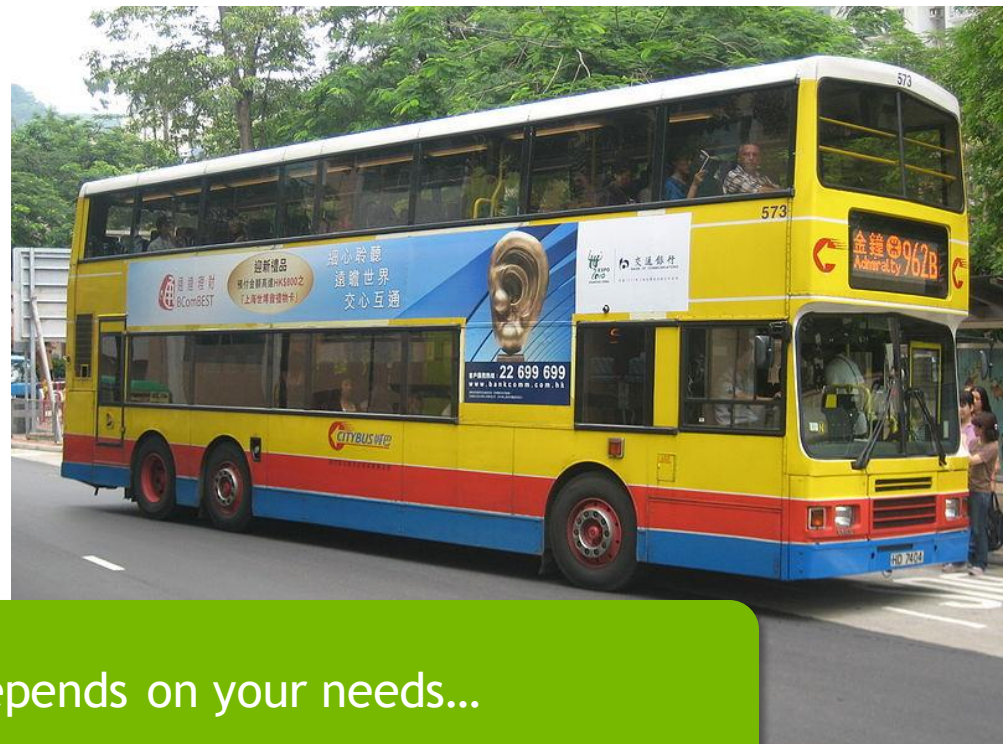
Optimized for Parallel Tasks



Speed v. Throughput

Speed

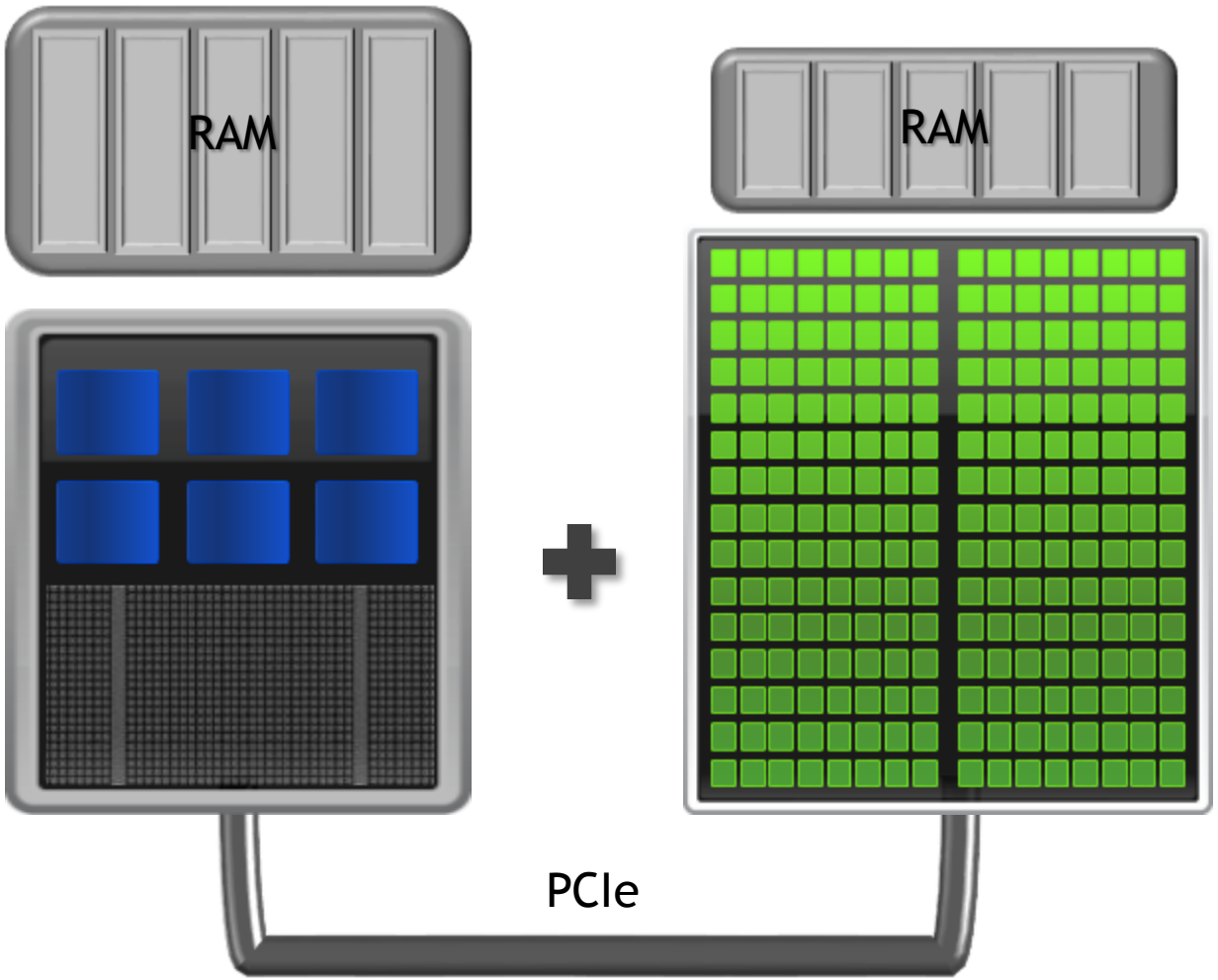
Throughput



Which is better depends on your needs…

*Images from Wikimedia Commons via Creative Commons

Accelerator Nodes



CPU and GPU have distinct memories

- CPU generally larger and slower
- GPU generally smaller and faster

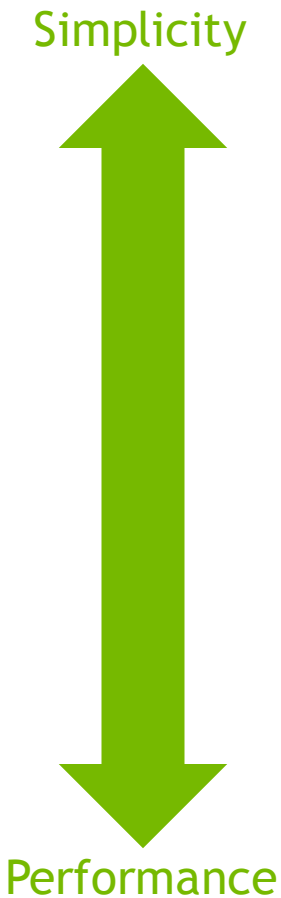
Execution begins on the CPU

- Data and computation are offloaded to the GPU

CPU and GPU communicate via PCIe

- Data must be copied between these memories over PCIe
- PCIe Bandwidth is much lower than either memories

Simplicity & Performance



Accelerated Libraries

Little or no code change for standard libraries; high performance

Limited by what libraries are available

Compiler Directives

High Level: Based on existing languages; simple and familiar

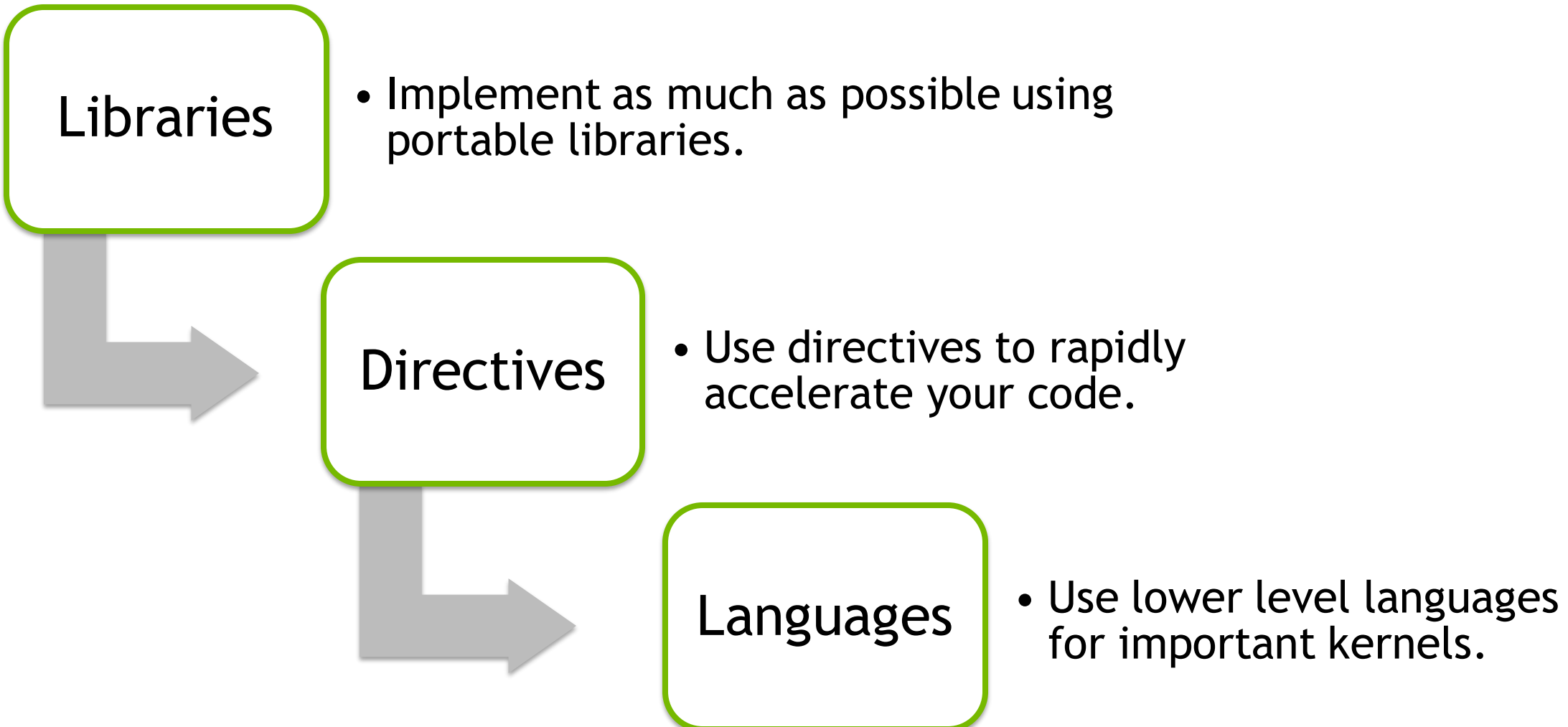
High Level: Performance may not be optimal

Parallel Language Extensions

Expose low-level details for maximum performance

Often more difficult to learn and more time consuming to implement

Code for Simplicity & Performance



GPU Accelerated Libraries

"Drop-in" Acceleration for Your Applications

Domain-specific
Deep Learning, GIS, EDA, Bioinformatics, Fluids

cuDNN | NVBIO | gpp Geometry Performance Primitives | Sundog Software Triton Ocean SDK

Visual Processing
Image & Video

NVIDIA CODEC SDK | FFmpeg | OpenCV | NVIDIA NPP

Linear Algebra
Dense, Sparse, Matrix

SuiteSparse CHOLMOD | MAGMA | [HiPLAR] | NVIDIA cuBLAS, cuSPARSE

Math Algorithms
AMG, Templates, Solvers

AmgX | IMSL RogueWave | Thrust & ArrayFire | cuSOLVER

developer.nvidia.com/gpu-accelerated-libraries

OpenACC

Simple | Powerful | Portable

Fueling the Next Wave of Scientific Discoveries in HPC

```
main()
{
  <serial code>
  #pragma acc kernels
  //automatically runs on GPU
  {
    <parallel code>
  }
}
```

University of Illinois
PowerGrid- MRI Reconstruction

70x Speed-Up
2 Days of Effort

RIKEN Japan
NICAM- Climate Modeling

7-8x Speed-Up
5% of Code Modified

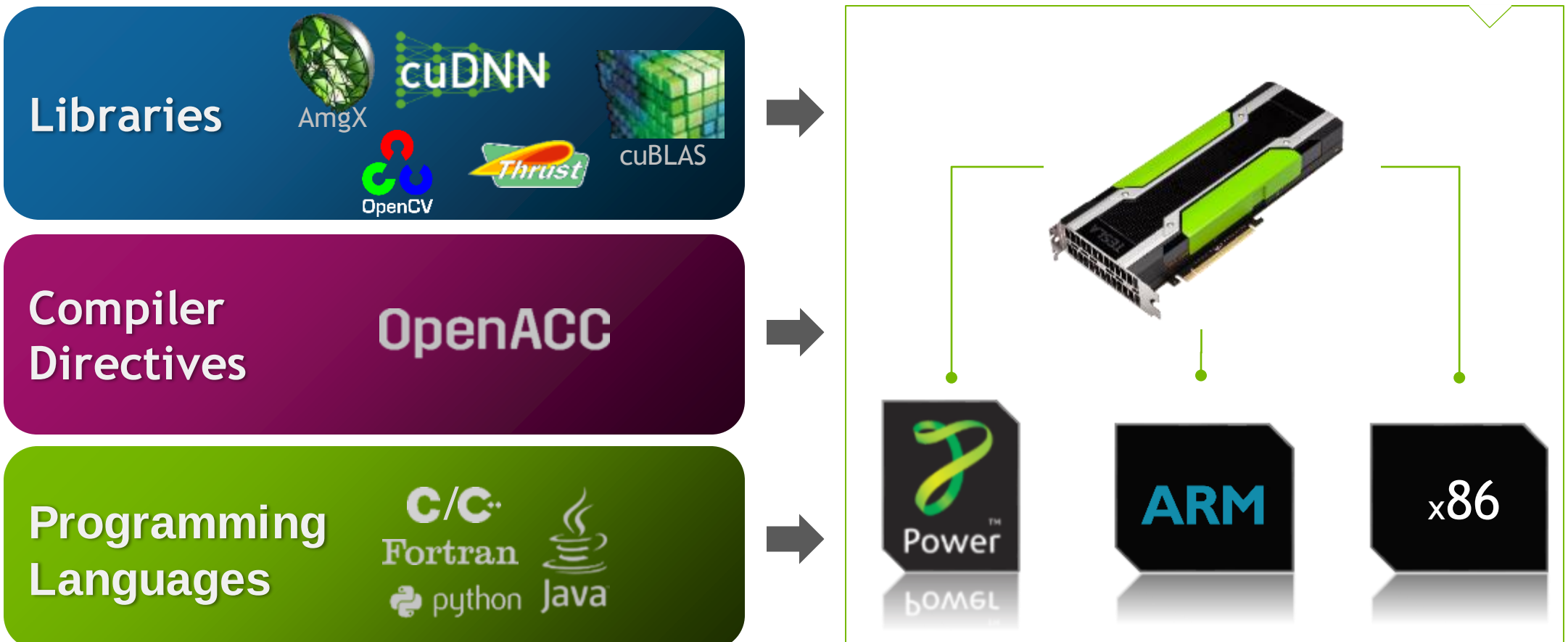
8000+

Developers

using OpenACC

http://www.cray.com/sites/default/files/resources/OpenACC_213462.12_OpenACC_Cosmo_CS_FNL.pdf
http://www.hpcwire.com/off-the-wire/first-round-of-2015-hackathons-gets-underway
http://on-demand.gputechconf.com/gtc/2015/presentation/S5297-Hisashi-Yashiro.pdf
http://www.openacc.org/content/experiences-porting-molecular-dynamics-code-gpus-cray-xk7

Common Programming Models Across Multiple CPUs



GPU Developer Eco-System

Numerical Packages

MATLAB
Mathematica
LabView

Debuggers & Profilers

CUDA-GDB
NV Visual Profiler
NVIDIA Nsight
Visual Studio
Allinea
TotalView

Languages & Directives

C
C++
Fortran
Java
Python
OpenACC
OpenMP

Cluster Tools

GPUDirect RDMA
Datacenter
GPU Manager

Libraries

FFT
BLAS
SPARSE
LAPACK
NPP
Video
Imaging

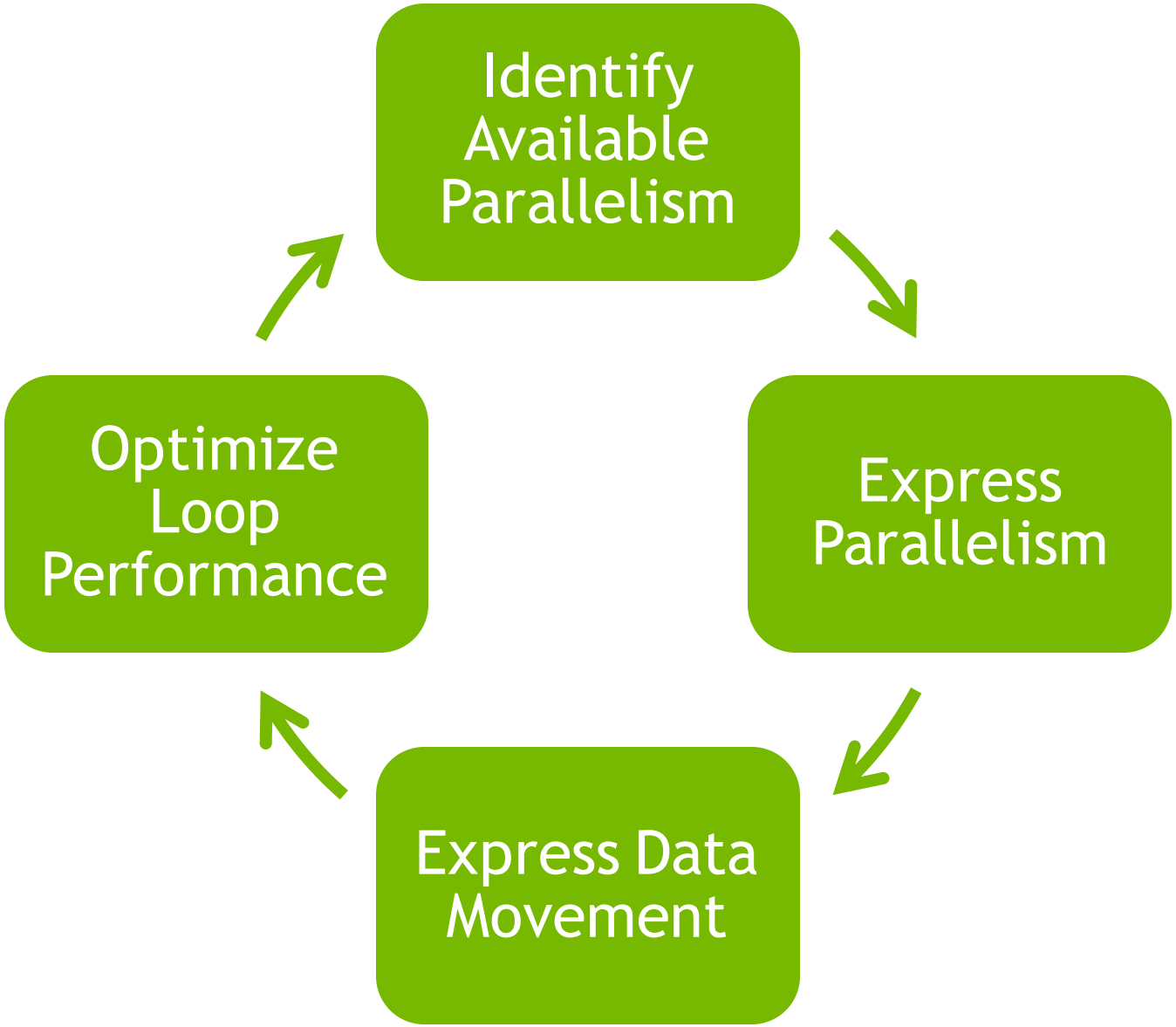
Consultants & Training

acceleware · GASS · STONE RIDGE TECHNOLOGY · ANEO · GPU Tech · TECH-X · ArrayFire · HPC · EM Photonics · SCALABLE GRAPHICS · WIPRO

OEM Solution Providers

DELL · hp · IBM · CRAY · ASUS · SUPERMICRO · sgi · FUJITSU · BULL · CISCO · lenovo 联想 · NEC

Accelerated Computing with OpenACC

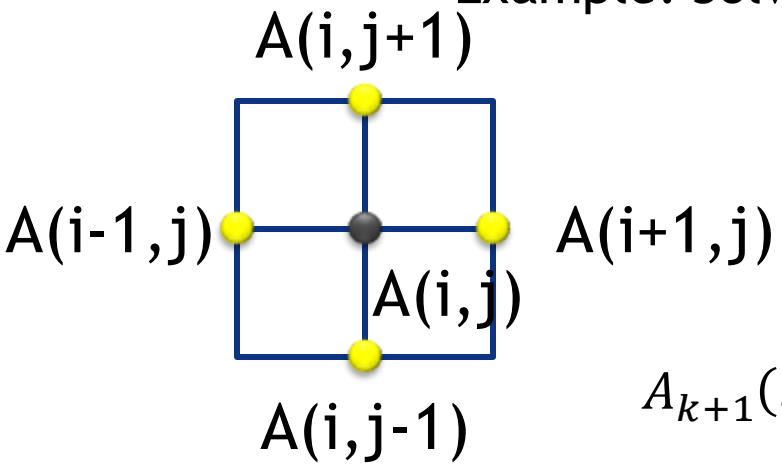

Identify Available Parallelism
Express Parallelism
Express Data Movement
Optimize Loop Performance

Example: Jacobi Iteration

Iteratively converges to correct value (e.g. Temperature), by computing new values at each point from the average of neighboring points.

Common, useful algorithm

Example: Solve Laplace equation in 2D: $\nabla^2 f(x, y) = 0$



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

Jacobi Iteration: C Code

```
while ( err > tol && iter < iter_max ) {
  err=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]);

      err = max(err, 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++;
}
```

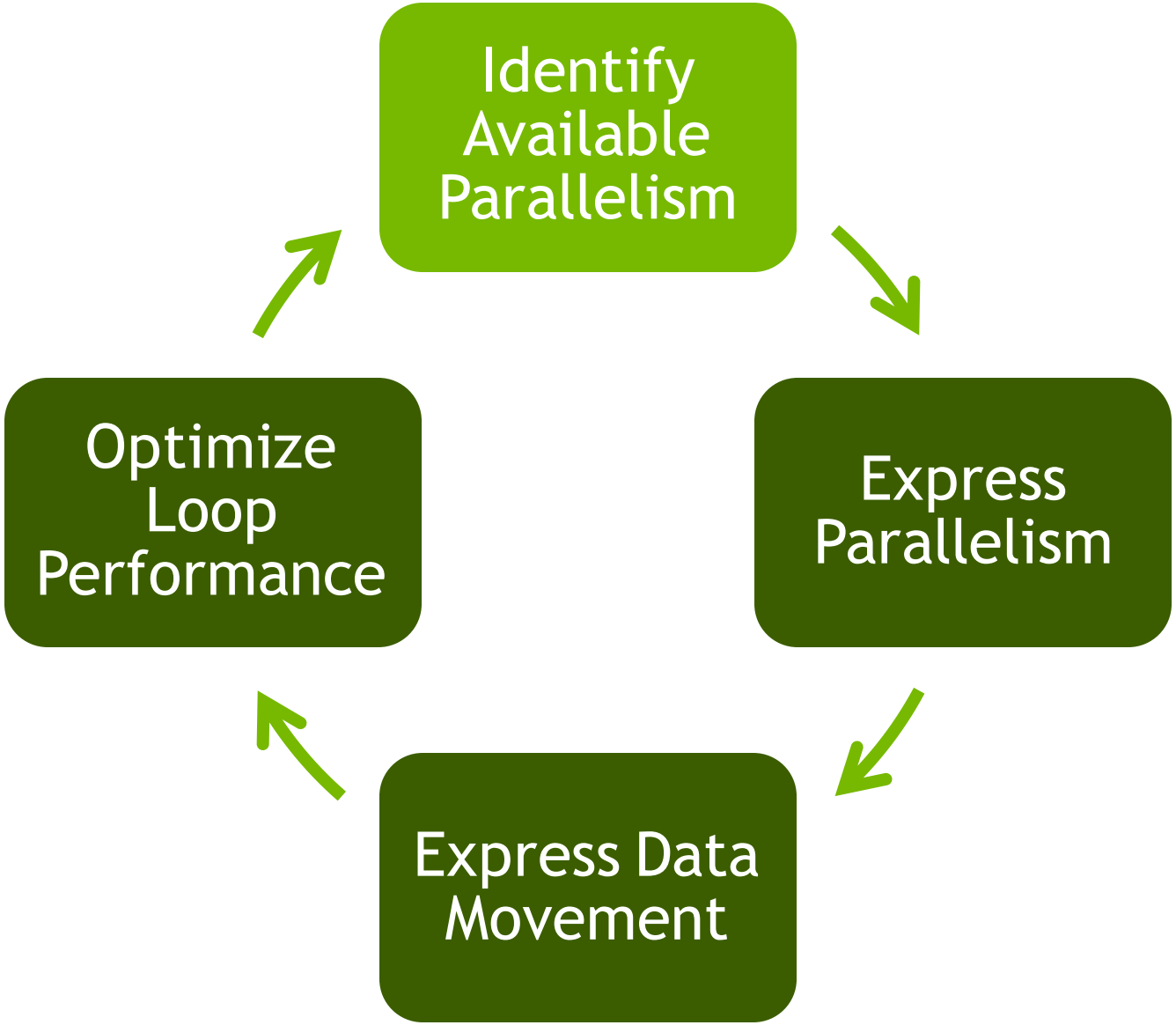
Iterate until converged

Iterate across matrix elements

Calculate new value from neighbors

Compute max error for convergence

Swap input/output arrays


Identify Available Parallelism
Express Parallelism
Express Data Movement
Optimize Loop Performance

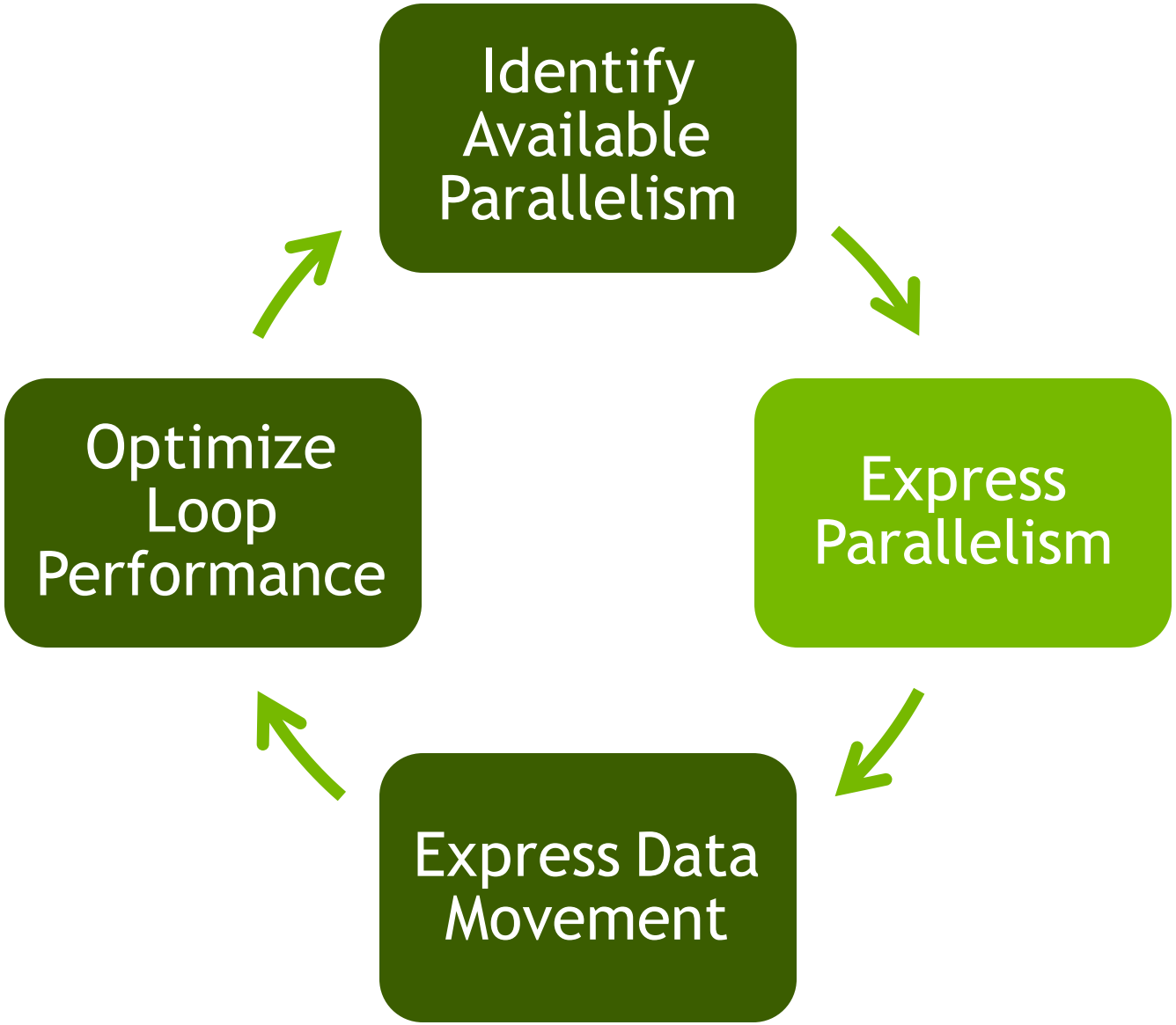
Identify Parallelism

```
while ( err > tol && iter < iter_max ) {
  err=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]);

      err = max(err, 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++;
}
```

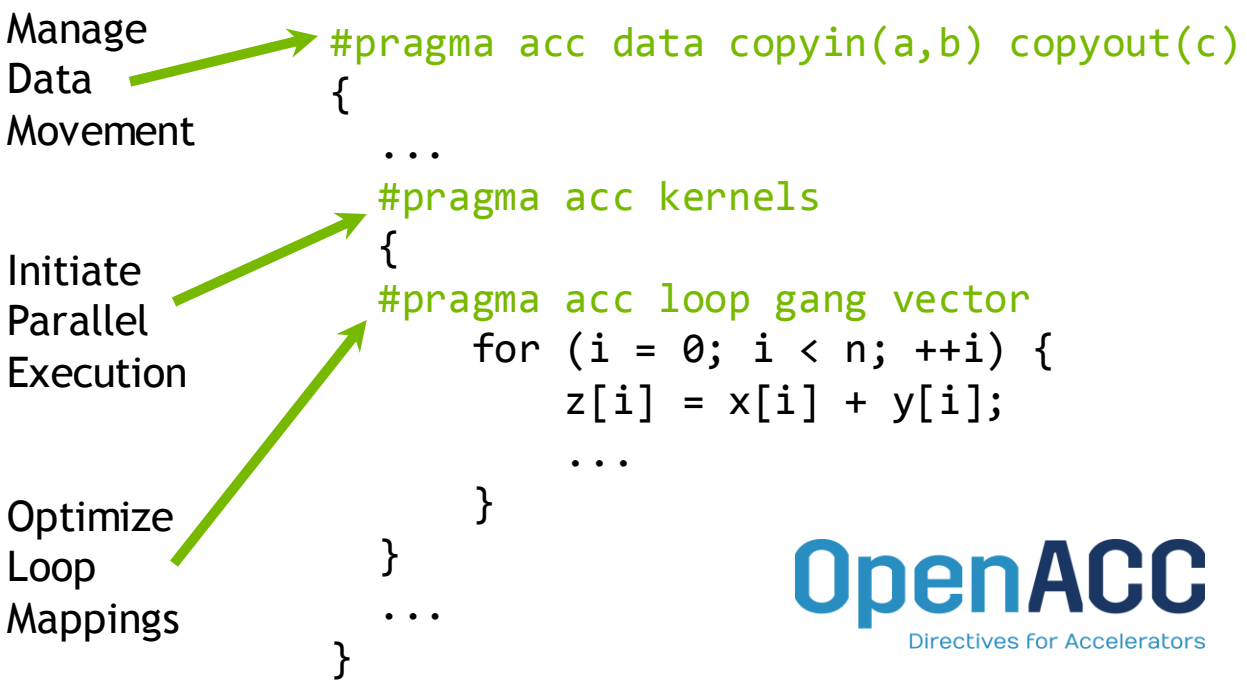
Data dependency between iterations.

Independent loop iterations

Independent loop iterations


Identify Available Parallelism
Express Parallelism
Express Data Movement
Optimize Loop Performance

OpenACC Directives



OpenACC
Directives for Accelerators

- Incremental
- Single source
- Interoperable
- Performance portable
- CPU, GPU, MIC

OpenACC kernels Directive

Identifies a region of code where I think the compiler can turn *loops* into *kernels*

```
#pragma acc kernels
{
for(int i=0; i<N; i++)
{
  x[i] = 1.0;
  y[i] = 2.0;
}

for(int i=0; i<N; i++)
{
  y[i] = a*x[i] + y[i];
}
}
```

kernel 1

kernel 2

The compiler identifies 2 parallel loops and generates 2 kernels.

OpenACC kernels Directive (Fortran)

Identifies a region of code where I think the compiler can turn *loops* into *kernels*

```
!$acc kernels
do i=1,N
  x(i) = 1.0
  y(i) = 2.0
end do

y(:) = a*x(:) + y(:)

!$acc end kernels
```

kernel 1

kernel 2

The compiler identifies 2 parallel loops and generates 2 kernels.

Loops vs. Kernels

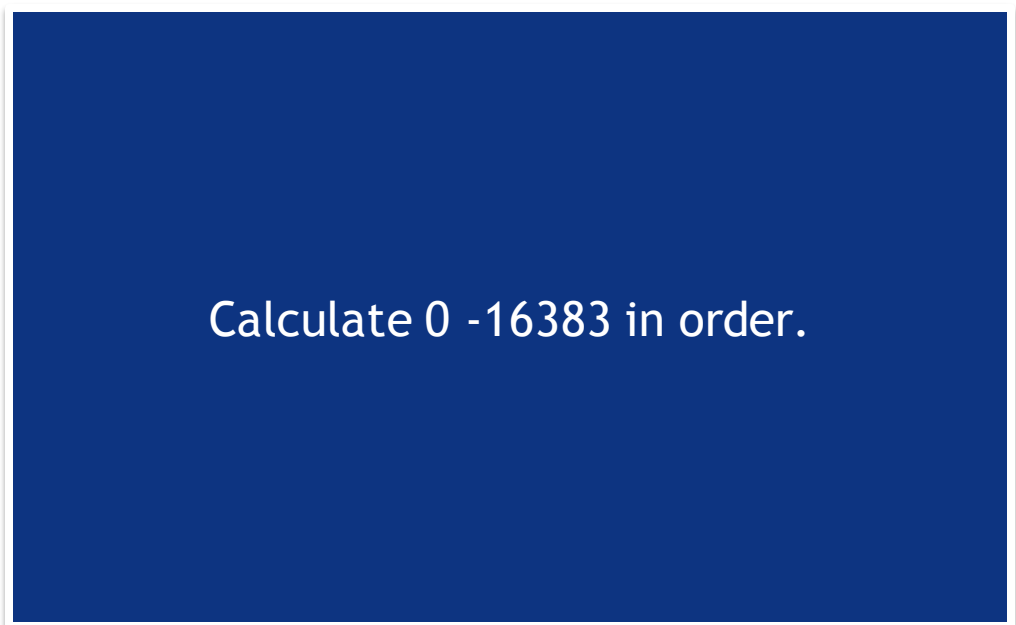
```
for (int i = 0; i < 16384; i++)
{
  C[i] = A[i] + B[i];
}
```

```
function loopBody(A, B, C, i)
{
  C[i] = A[i] + B[i];
}
```

Loops vs. Kernels

```
for (int i = 0; i < 16384; i++)
{
  C[i] = A[i] + B[i];
}
```

```
function loopBody(A, B, C, i)
{
  C[i] = A[i] + B[i];
}
```



Loops vs. Kernels

```
for (int i = 0; i < 16384; i++)
{
  C[i] = A[i] + B[i];
}
```

Calculate 0 -16383 in order.

```
function loopBody(A, B, C, i)
{
  C[i] = A[i] + B[i];
}
```

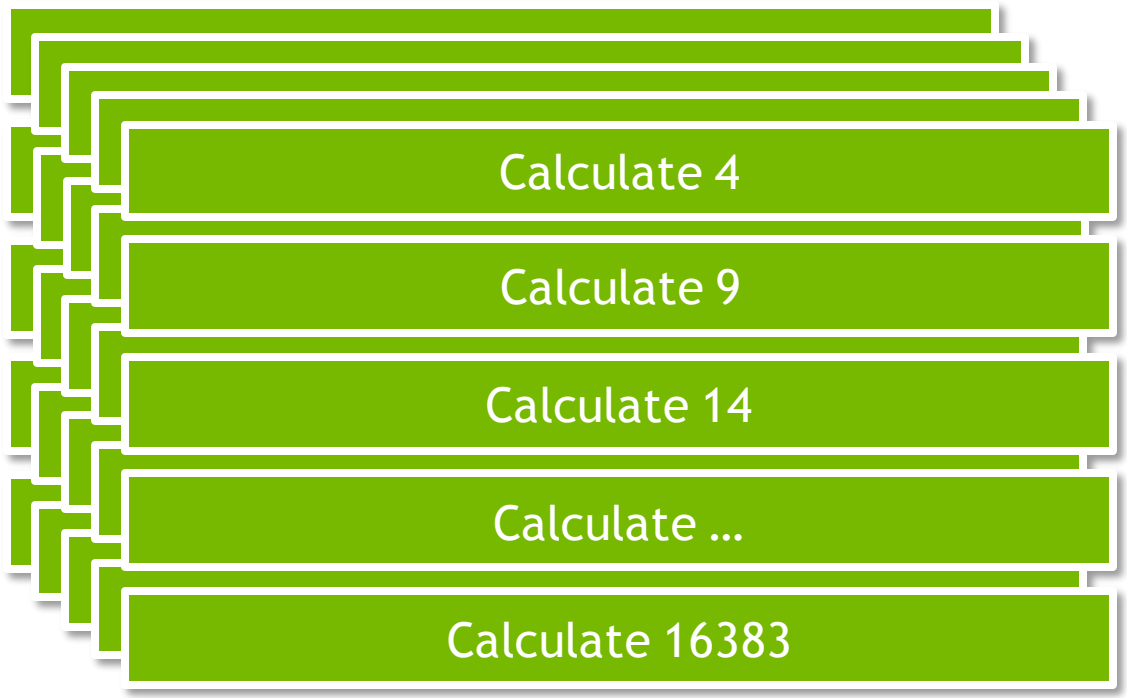
Calculate 0

Loops vs. Kernels

```
for (int i = 0; i < 16384; i++)
{
  C[i] = A[i] + B[i];
}
```

Calculate 0 -16383 in order.

```
function loopBody(A, B, C, i)
{
  C[i] = A[i] + B[i];
}
```



The Kernels Directive

Identifies a region of code where I think the compiler can turn *loops* into *kernels*

```
#pragma acc kernels
{
for (int i = 0; i < 16384;
          i++)
{
  C[i] = A[i] + B[i];
}
}
```

The Compiler will…

1. Analyze the code to determine if it contains parallelism
2. Identify data that needs to be moved to/from the GPU memory
3. Generate kernels
4. Run on the GPU

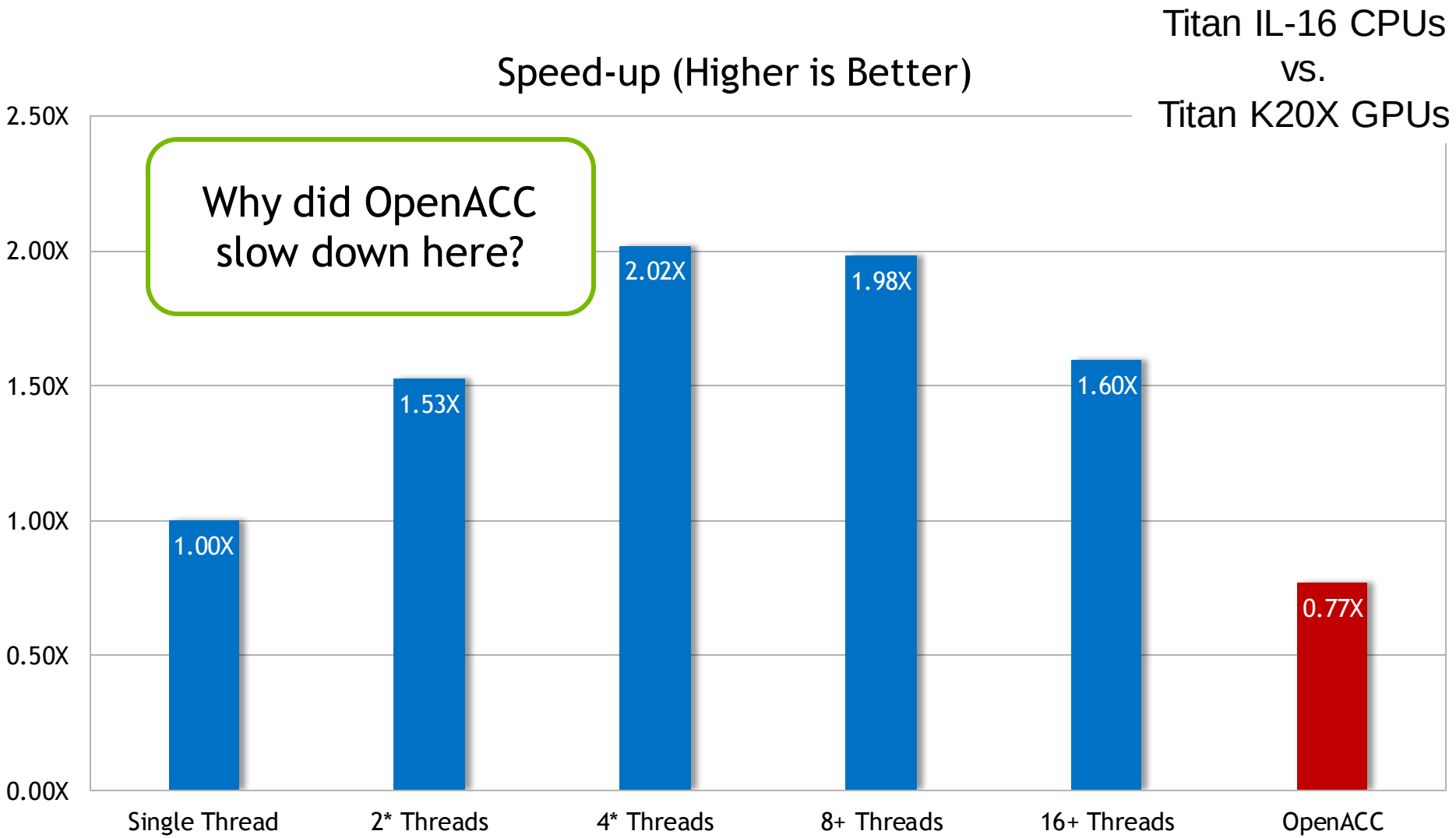
Parallelize with OpenACC kernels

```
while ( err > tol && iter < iter_max ) {
  err=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]);

      err = max(err, 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++;
}
```

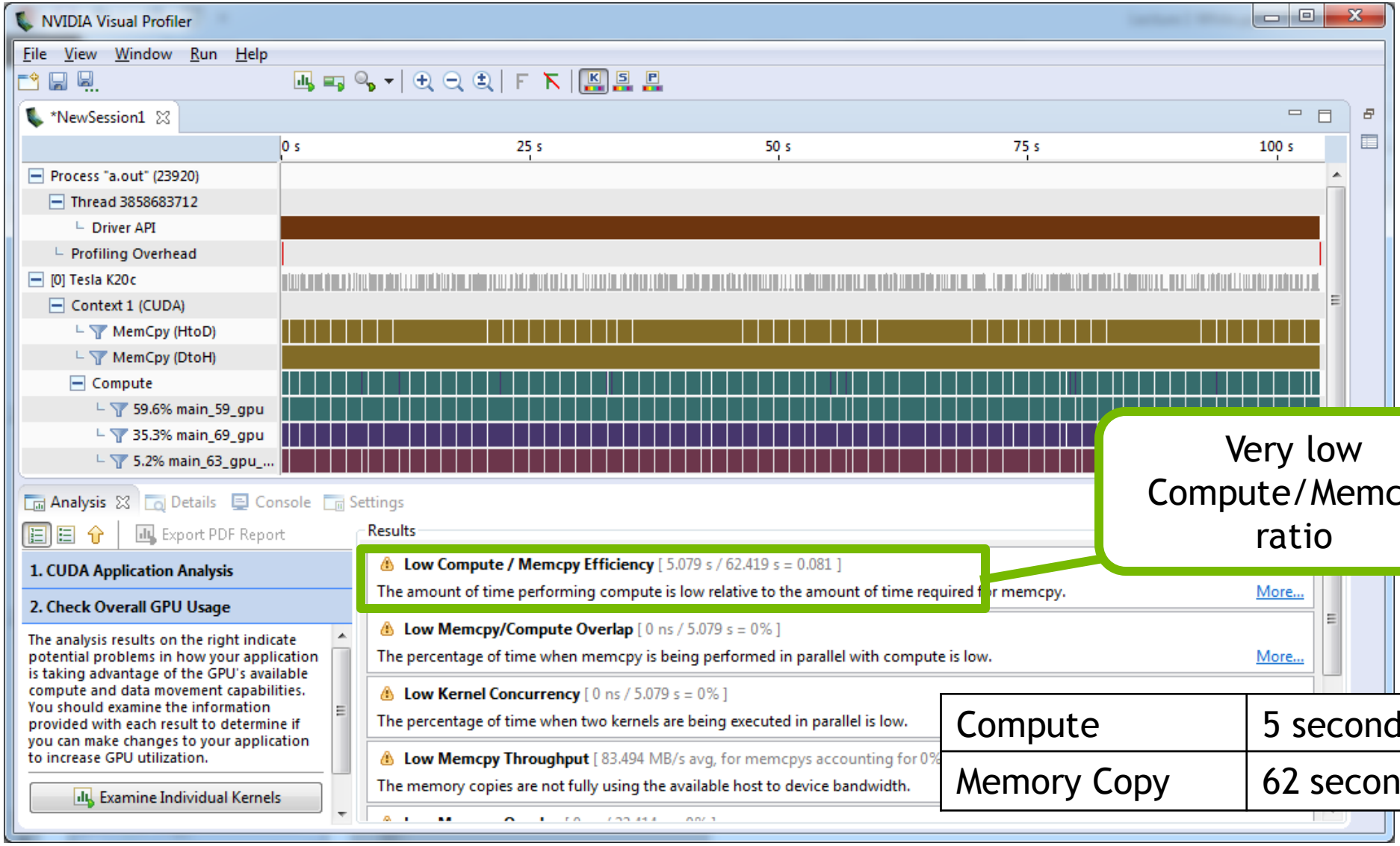
Look for parallelism within this region.

Building the code

```
$ pgcc -fast -ta=tesla -Minfo=all laplace2d.c
main:
     40, Loop not fused: function call before adjacent loop
         Generated vector sse code for the loop
     51, Loop not vectorized/parallelized: potential early exits
     55, Generating copyout(Anew[1:4094][1:4094])
         Generating copyin(A[:][:])
         Generating copyout(A[1:4094][1:4094])
         Generating Tesla code
     57, Loop is parallelizable
     59, Loop is parallelizable
         Accelerator kernel generated
         57, #pragma acc loop gang /* blockIdx.y */
         59, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
         63, Max reduction generated for error
     67, Loop is parallelizable
     69, Loop is parallelizable
         Accelerator kernel generated
         67, #pragma acc loop gang /* blockIdx.y */
         69, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
```

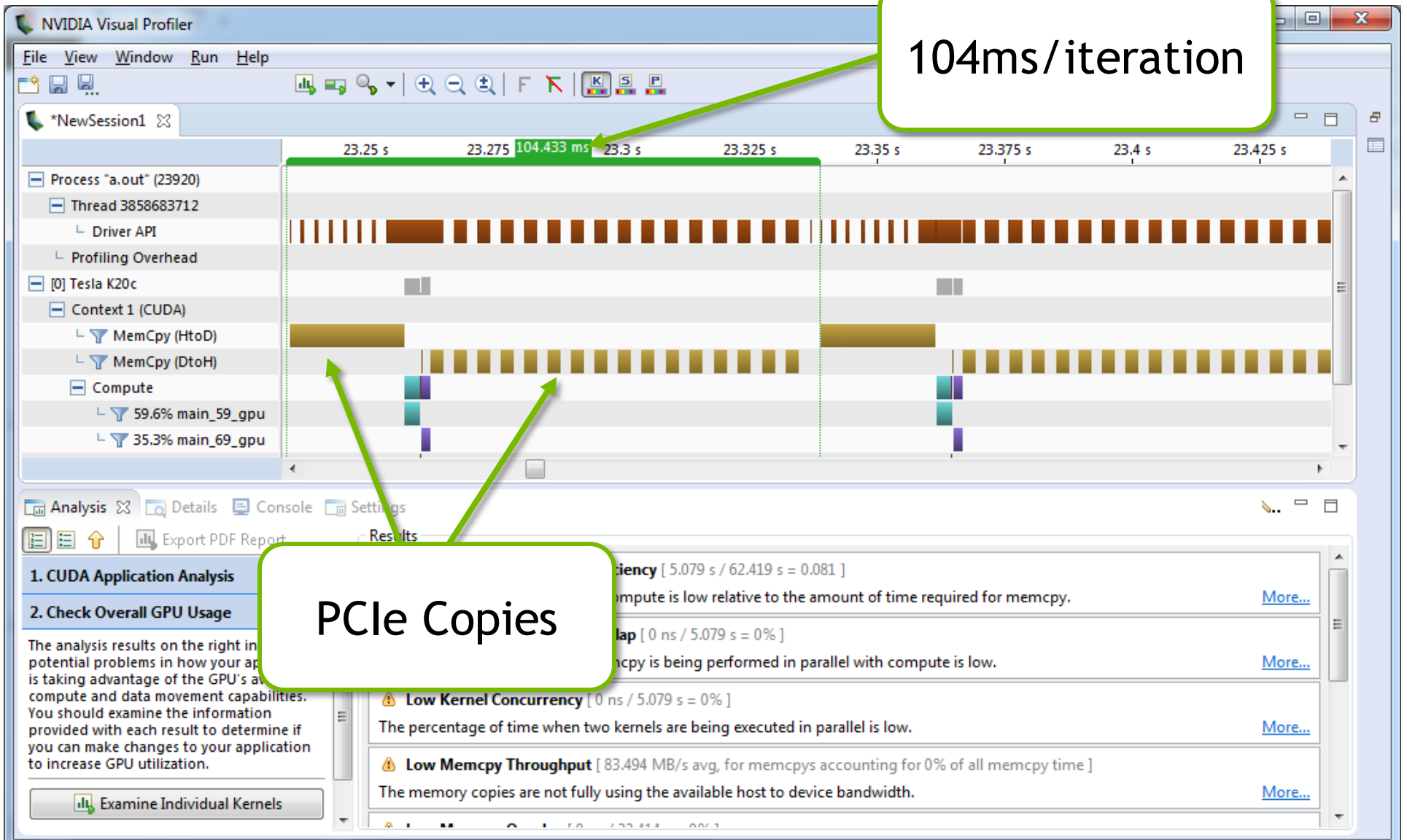


** Compute Module Mode, + Default mode*



Very low Compute/Memcpy ratio

Compute	5 seconds
Memory Copy	62 seconds


NVIDIA Visual Profiler
File View Window Run Help
*NewSession1
104ms/iteration
23.25 s
23.275
104.433 ms
23.3 s
23.325 s
23.35 s
23.375 s
23.4 s
23.425 s
Process "a.out" (23920)
Thread 3858683712
Driver API
Profiling Overhead
[0] Tesla K20c
Context 1 (CUDA)
MemCpy (HtoD)
MemCpy (DtoH)
Compute
59.6% main_59_gpu
35.3% main_69_gpu
Analysis
Details
Console
Export PDF Report
Results
PCIe Copies
1. CUDA Application Analysis
2. Check Overall GPU Usage
The analysis results on the right in
potential problems in how your ap
is taking advantage of the GPU's a
compute and data movement capabilities.
You should examine the information
provided with each result to determine if
you can make changes to your application
to increase GPU utilization.
Examine Individual Kernels
iency [5.079 s / 62.419 s = 0.081]
mpute is low relative to the amount of time required for memcpy.
More...
lap [0 ns / 5.079 s = 0%]
ncpy is being performed in parallel with compute is low.
More...
Low Kernel Concurrency [0 ns / 5.079 s = 0%]
The percentage of time when two kernels are being executed in parallel is low.
More...
Low Memcpy Throughput [83.494 MB/s avg, for memcpys accounting for 0% of all memcpy time]
The memory copies are not fully using the available host to device bandwidth.
More...

Excessive Data Transfers

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

A, Anew resident on host

Copy

Copy

A, Anew resident on accelerator

```
#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]);
    err = max(err, abs(Anew[j][i] -
                       A[j][i]);
  }
}
...
```

These copies happen every iteration of the outer while loop!

A, Anew resident on host

A, Anew resident on accelerator

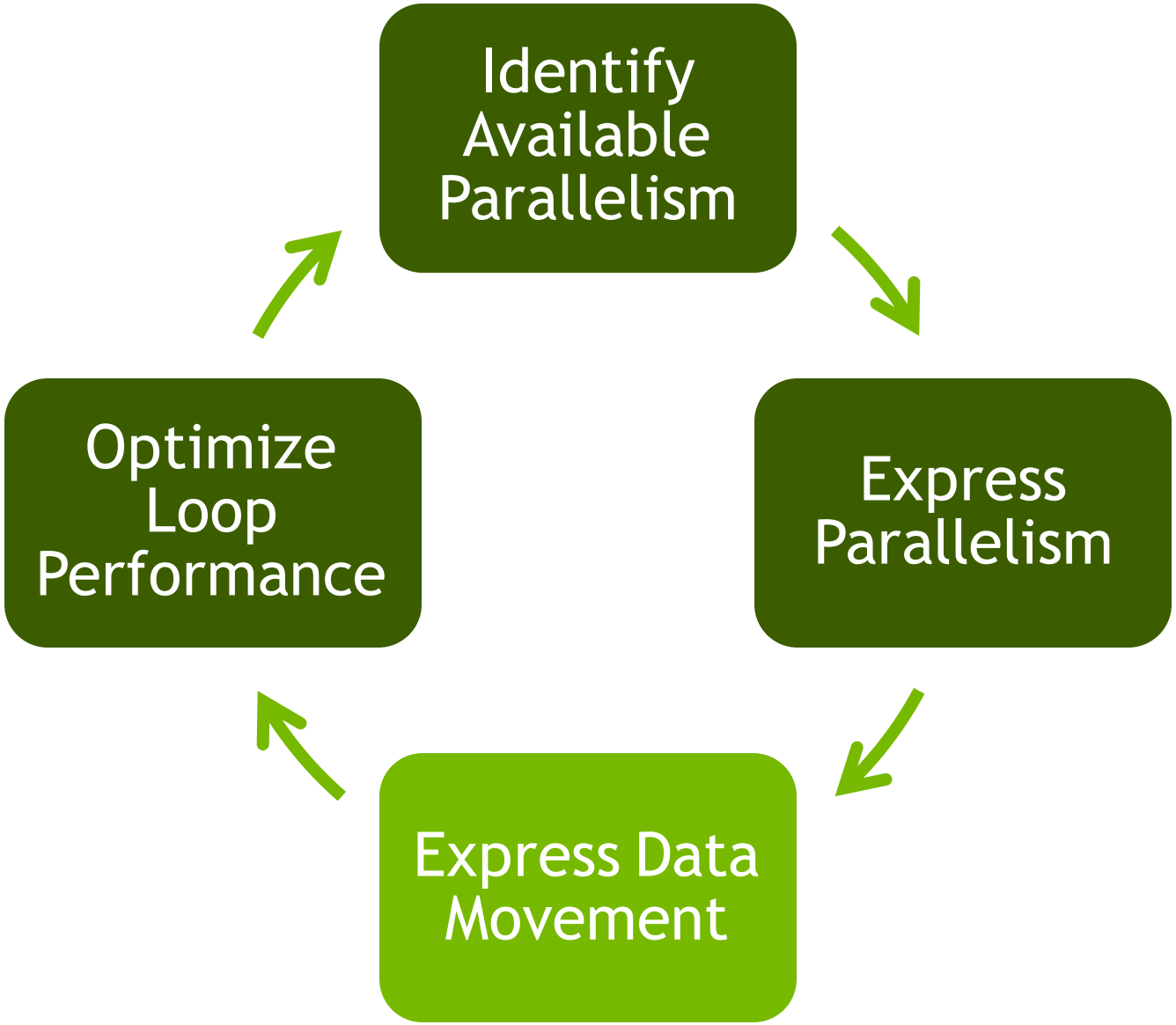
```
  ...
}
```

Identifying Data Locality

```
while ( err > tol && iter < iter_max ) {
  err=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]);

      err = max(err, 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++;
}
```

Does the CPU need the data between these loop nests?

Does the CPU need the data between iterations of the convergence loop?


Identify Available Parallelism
Express Parallelism
Express Data Movement
Optimize Loop Performance

Data regions

The **data** directive defines a region of code in which GPU arrays remain on the GPU and are shared among all kernels in that region.

```
#pragma acc data
{
#pragma acc kernels
...

#pragma acc kernels
...
}
```

Data Region

Arrays used within the data region will remain on the GPU until the end of the data region.

Data Clauses

`copy ( list )`	Allocates memory on GPU and copies data from host to GPU when entering region and copies data to the host when exiting region.
`copyin ( list )`	Allocates memory on GPU and copies data from host to GPU when entering region.
`copyout ( list )`	Allocates memory on GPU and copies data to the host when exiting region.
`create ( list )`	Allocates memory on GPU but does not copy.
`present ( list )`	Data is already present on GPU from another containing data region.
`deviceptr( list )`	The variable is a device pointer (e.g. CUDA) and can be used directly on the device.

Array Shaping

Compiler sometimes cannot determine size of arrays

Must specify explicitly using data clauses and array “shape”

C/C++

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

Fortran

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

Note: data clauses can be used on `data`, `parallel`, or `kernels`

Express Data Locality

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

      err = max(err, 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++;
}
```

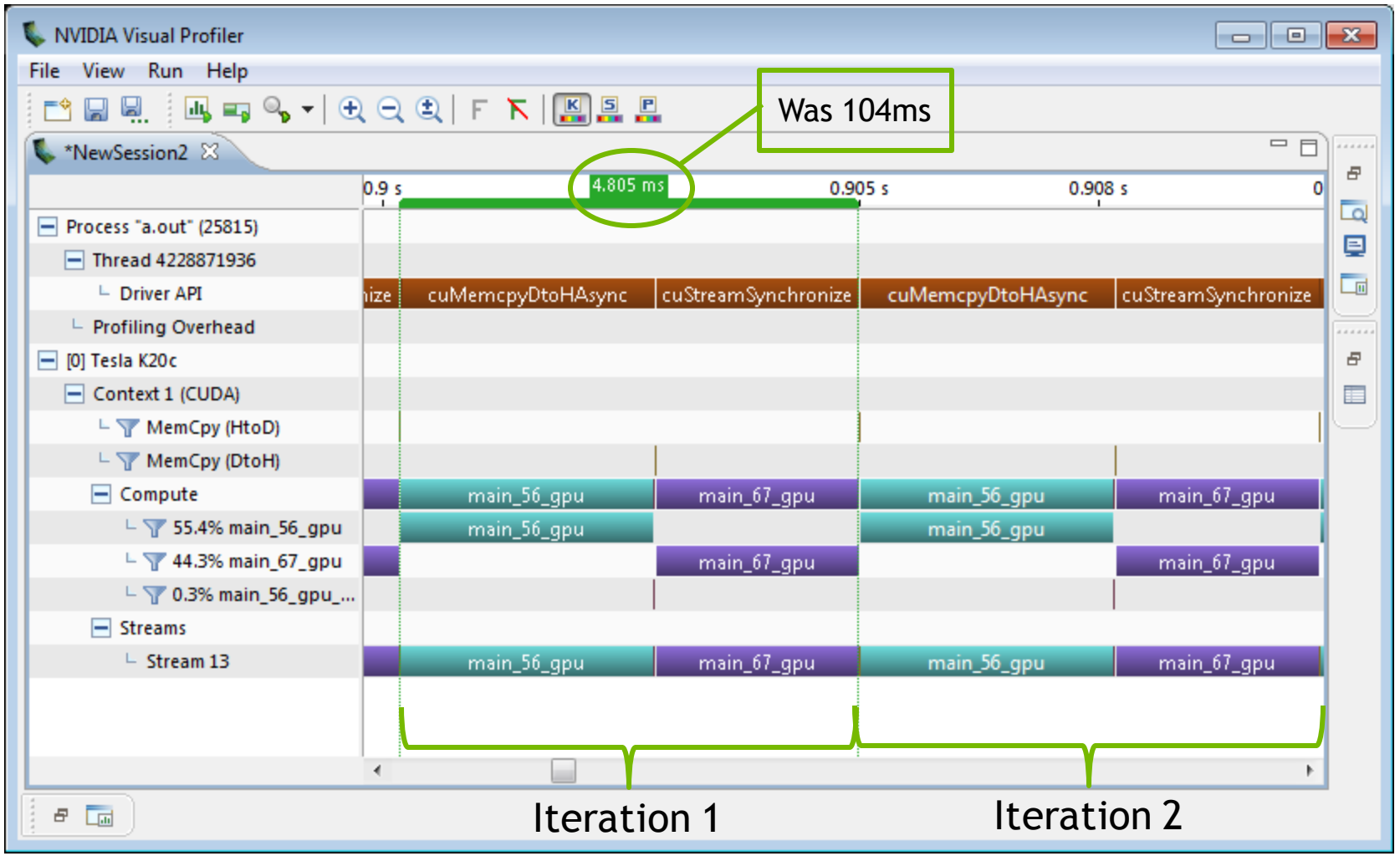
Copy A to/from the accelerator only when needed.

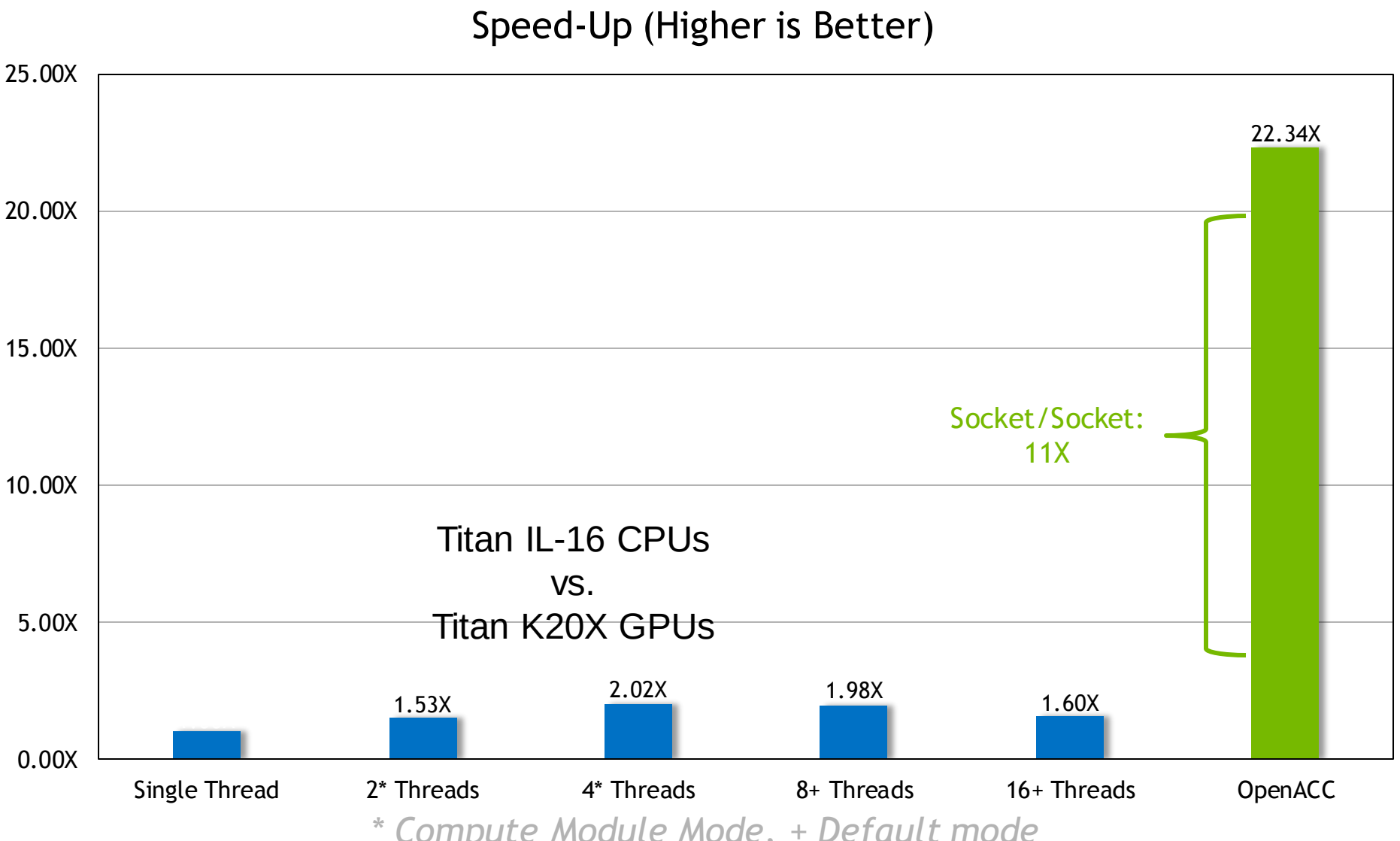
Create Anew as a device temporary.

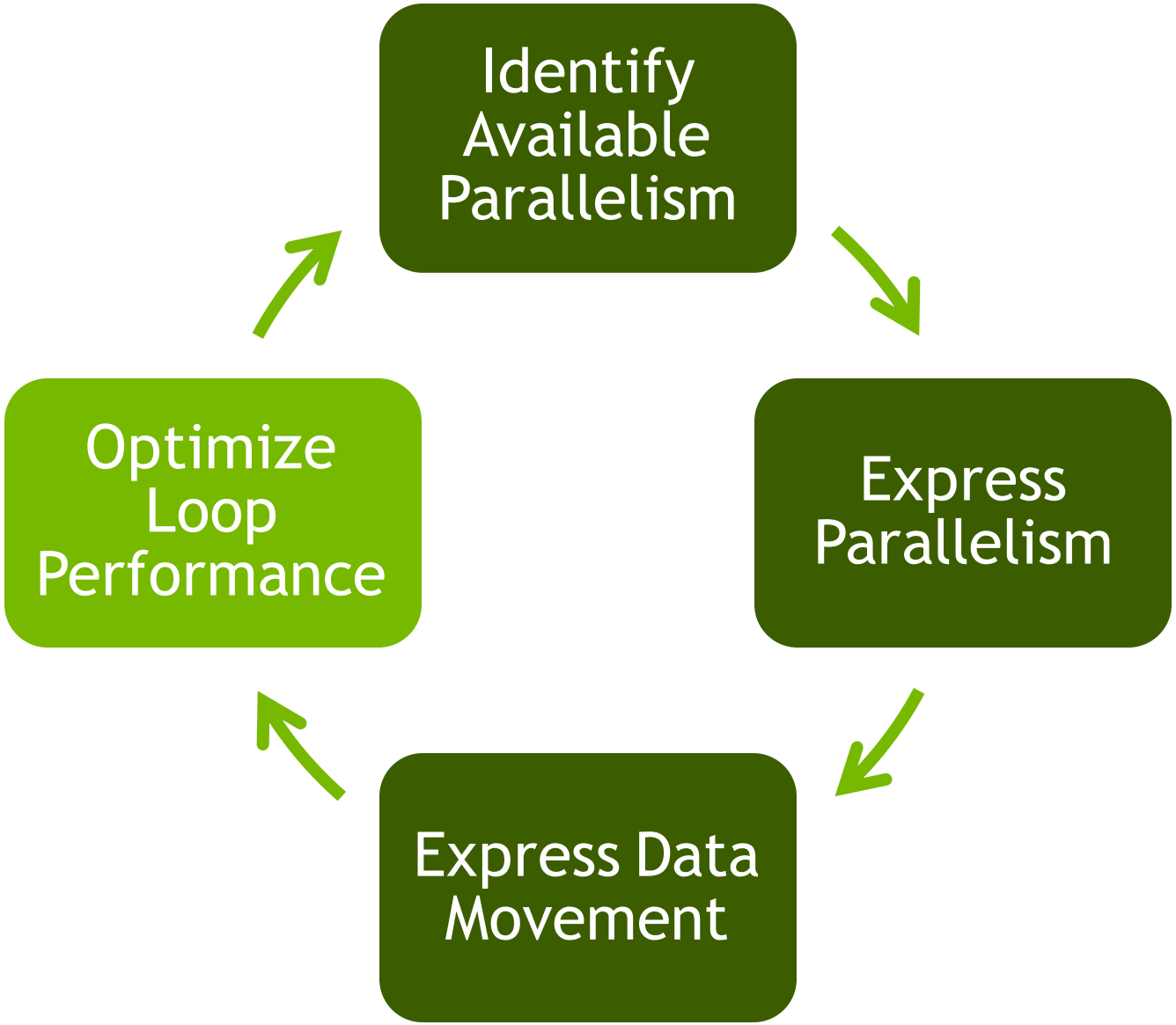
Rebuilding the code

```
$ pgcc -fast -acc -ta=tesla -Minfo=all laplace2d.c
main:
     40, Loop not fused: function call before adjacent loop
         Generated vector sse code for the loop
     51, Generating copy(A[:][:])
         Generating create(Anew[:][:])
         Loop not vectorized/parallelized: potential early exits
     56, Accelerator kernel generated
         56, Max reduction generated for error
         57, #pragma acc loop gang /* blockIdx.x */
         59, #pragma acc loop vector(256) /* threadIdx.x */
     56, Generating Tesla code
     59, Loop is parallelizable
     67, Accelerator kernel generated
         68, #pragma acc loop gang /* blockIdx.x */
         70, #pragma acc loop vector(256) /* threadIdx.x */
     67, Generating Tesla code
     70, Loop is parallelizable
```

Visual Profiler: Data Region




Speed-Up (Higher is Better)
25.00X
20.00X
15.00X
10.00X
5.00X
0.00X
22.34X
Socket/Socket:
11X
Titan IL-16 CPUs
vs.
Titan K20X GPUs
1.53X
2.02X
1.98X
1.60X
Single Thread
2* Threads
4* Threads
8+ Threads
16+ Threads
OpenACC
* Compute Module Mode, + Default mode


Identify Available Parallelism
Express Parallelism
Express Data Movement
Optimize Loop Performance

The loop Directive

The `loop` directive gives the compiler additional information about the *next* loop in the source code through several clauses.

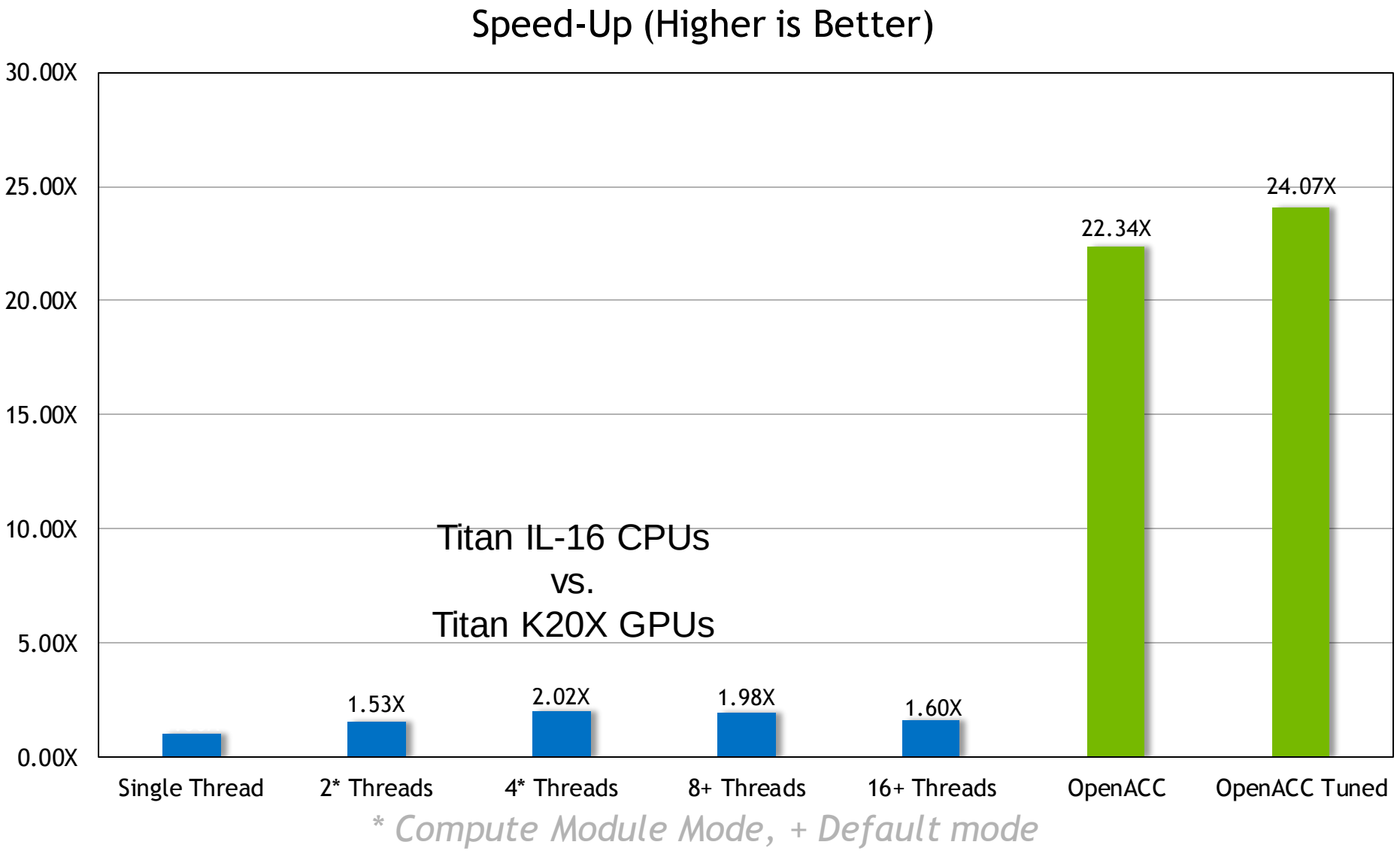
- `independent` – all iterations of the loop are independent
- `collapse(N)` – turn the next N loops into one, flattened loop
- `tile(N[,M,…])` - break the next 1 or more loops into *tiles* based on the provided dimensions.

These clauses and more will be discussed in greater detail in a later class.

Optimize Loop Performance

```
#pragma acc data copy(A) create(Anew)
while ( err > tol && iter < iter_max ) {
  err=0.0;
#pragma acc kernels
{
#pragma acc loop device_type(nvidia) tile(32,4)
  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]);

      err = max(err, abs(Anew[j][i] - A[j][i]));
    }
  }
#pragma acc loop device_type(nvidia) tile(32,4)
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }
}
  iter++;
}
```

"Tile" the next two loops into 32x4 blocks, but only on NVIDIA GPUs.


Speed-Up (Higher is Better)
30.00X
25.00X
20.00X
15.00X
10.00X
5.00X
0.00X
Titan IL-16 CPUs
vs.
Titan K20X GPUs
1.53X
2.02X
1.98X
1.60X
22.34X
24.07X
Single Thread
2* Threads
4* Threads
8+ Threads
16+ Threads
OpenACC
OpenACC Tuned
* Compute Module Mode, + Default mode

Next Steps

Online Training

https://nvidia.qwiklab.com/

NVIDIA offers a variety of hands-on labs online.

- OpenACC, Libraries, CUDA, Deep Learning

You can obtain free credits to take these labs by

- Visiting https://developer.nvidia.com/qwiklabs-signup and
- Signing up with the promo code OPENACC

NVIDIA Online OpenACC Courses

2015 course videos and slides are available online, the homework labs are in qwiklabs.

https://developer.nvidia.com/openacc-courses

Advanced OpenACC Course began May 19th.

https://developer.nvidia.com/openacc-advanced-course

2016 OpenACC course begins in September