COMP 633 - Parallel Computing

Lecture 15
October 19, 2021

Programming Accelerators using Directives

Credits: Introduction to OpenACC and toolkit – Jeff Larkin, Nvidia – Oct 2015

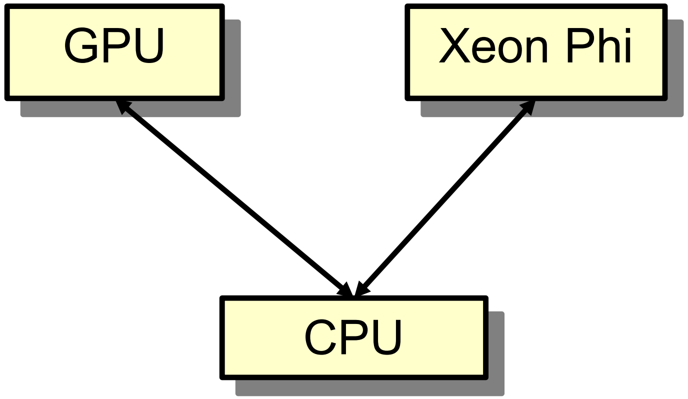
Heterogeneous Parallel Computers

- Composed of
  - CPU(s)
    - Low-latency processor optimized for sequential execution
    - large memory size and deep memory hierarchy
  - Accelerator(s)
    - high throughput SIMD or MIMD processors optimized for data-parallel execution
    - high performance limited memory size and small depth memory hierarchy

- Example
  - Multisocket compute server
    - Host: two-socket 20 – 40 Intel Xeon cores with 128 – 512 GB CC-NUMA shared memory
    - Accelerators: 1-8 accelerators (e.g. Nvidia Cuda cards, Intel Xeon Phi cards) connected via PCIe x16 interfaces (16GB/s)
      - host controls data to/from accelerator memory

Basic Programming Models

- Offload model
  - idea: offload computational kernels
    - send data to accelerator
    - call kernel(s)
    - retrieve data
  - accelerator-specific compiler support
    - Cuda compiler (`nvcc`) for Nvidia GPUs
    - Intel vectorizing compiler (`icc -mmic`) for Intel Xeon Phi KNL
      - `#pragma offload target(mic:n) in(…) out(…) inout(…)`
  - accelerator-neutral OpenCL → OpenACC
    - Cuda-like notation
    - OpenACC compiler can target Nvidia or Intel Xeon Phi (RIP)



Emerging Programming Models

- directive model
  - idea: identify sections of code to be compiled for accelerator(s)
    - data transfer and kernel invocation generated by compiler
  - accelerator-neutral efforts
    - OpenACC
      - ```
        #pragma acc parallel loop
           for (…) { … }
        ```
      - gang, worker, vector (threadblock, warp, warp in SIMT lockstep)
      - gcc 5, PGI, Cray, CAPS, Nvidia compilers
    - OpenMP 5.1
      - similar directives to (but more general than) OpenACC
      - includes all previous shared memory parallelization directives
      - implemented by gcc 11.2
        ```

Matrix multiplication

- library implementations

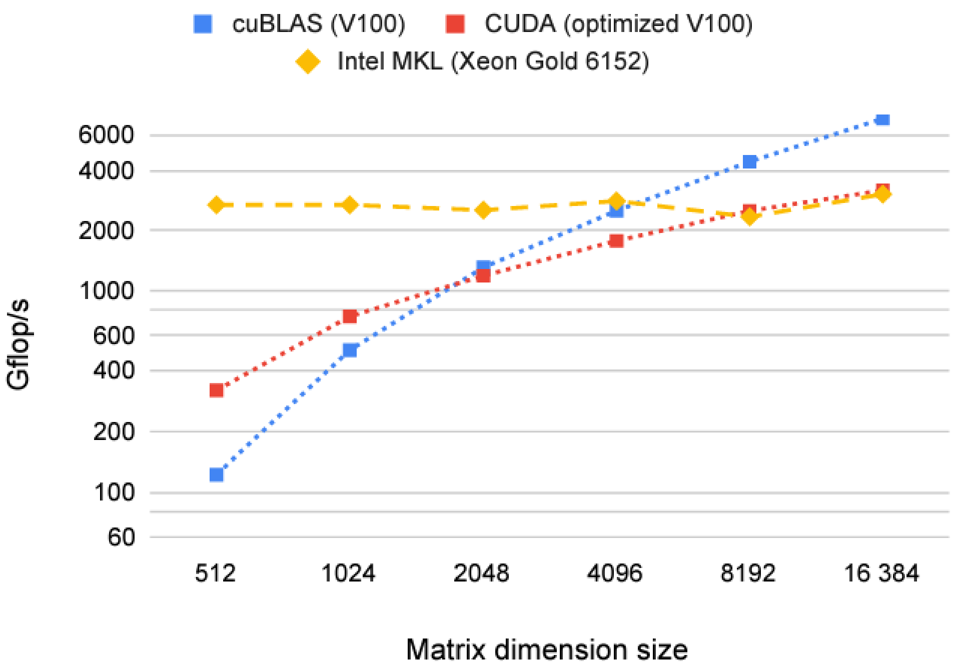


Figure 2. FP32 performance comparison of cuBLAS, Intel MKL and CUDA on Tesla V100

Matrix multiplication

- using parallelization directives

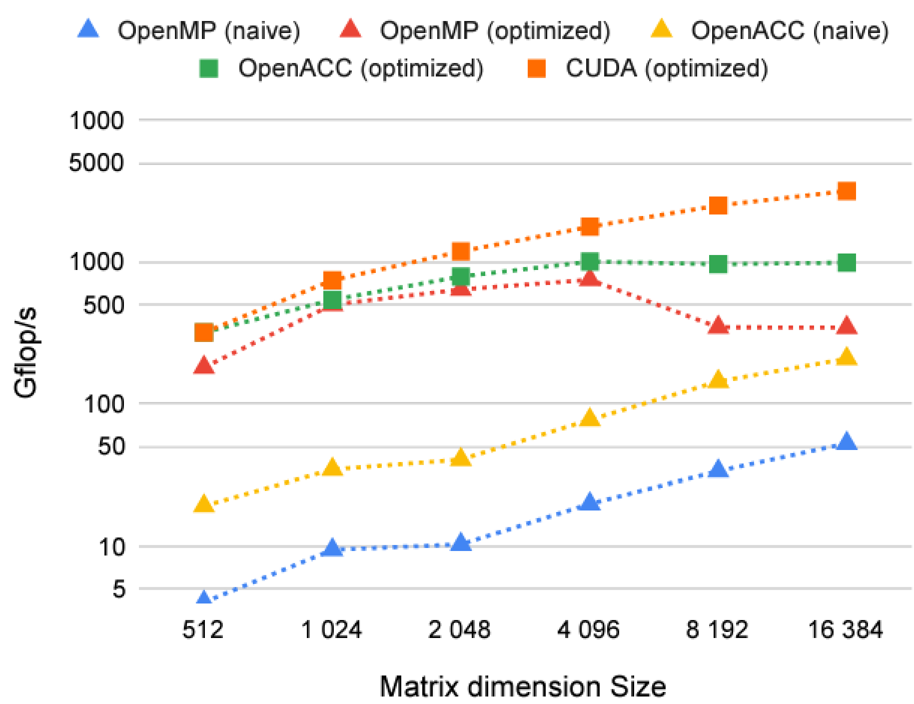


Figure 3. Dependence of FP32 performance on matrix sizes and programming model on Tesla V100

Introduction to OpenACC

Jeff Larkin, NVIDIA Developer Technologies

NVIDIA

Why OpenACC?

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

OpenACC Directives

Manage Data Movement

Initiate Parallel Execution

Optimize Loop Mappings

```
#pragma acc data copyin(a,b) copyout(c)
{
  ...
  #pragma acc parallel
  {
  #pragma acc loop gang vector
      for (i = 0; i < n; ++i) {
          z[i] = x[i] + y[i];
          ...
      }
  }
  ...
}
```

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

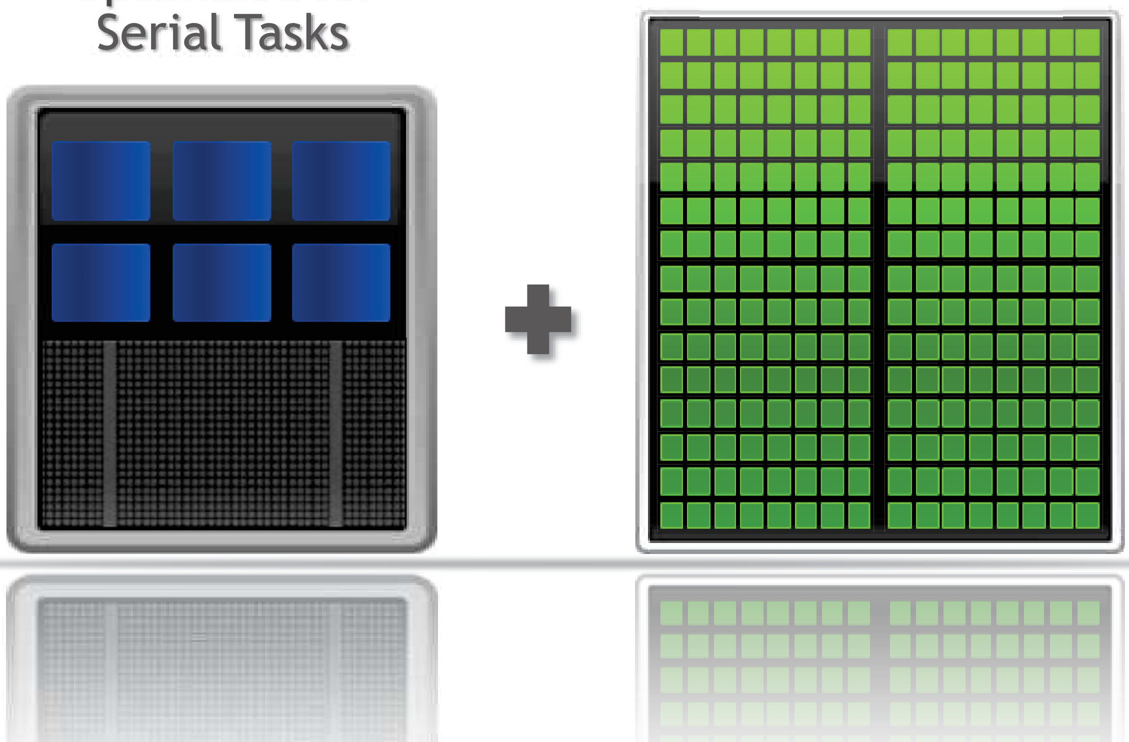
Accelerated Computing Fundamentals

Accelerated Computing

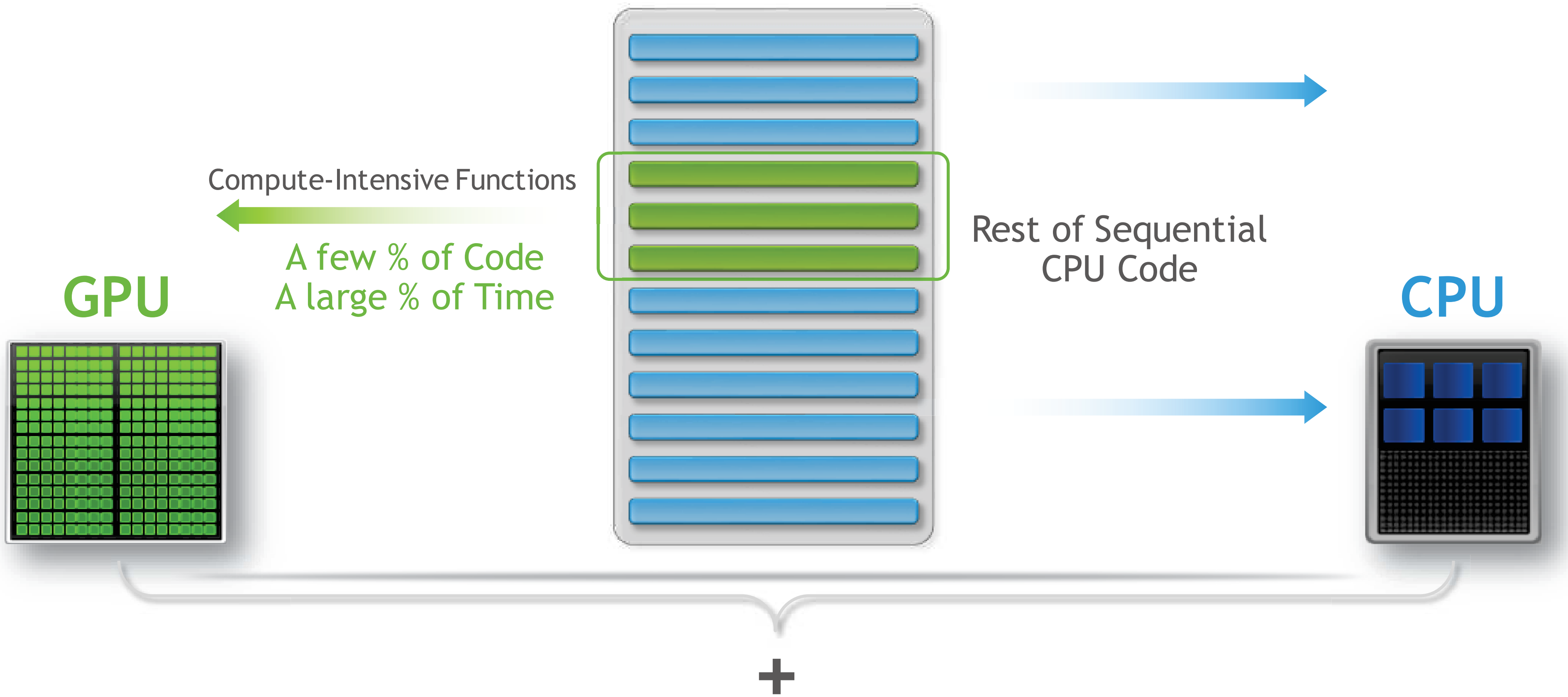
10x Performance & 5x Energy Efficiency for HPC

CPU
Optimized for
Serial Tasks

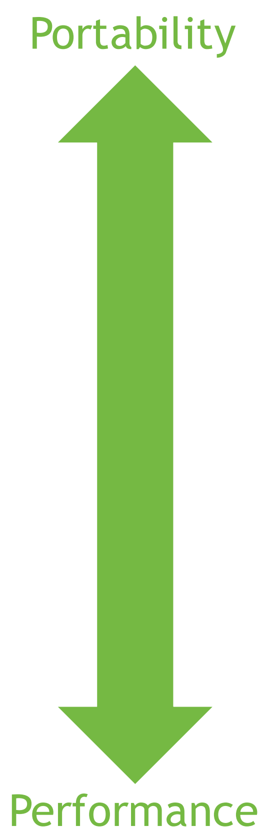
GPU Accelerator
Optimized for
Parallel Tasks



What is Heterogeneous Programming?



Portability & Performance



Accelerated Libraries

High performance with little or no code change

Limited by what libraries are available

Compiler Directives

High Level: Based on existing languages; simple, familiar, portable

High Level: Performance may not be optimal

Parallel Language Extensions

Greater flexibility and control for maximum performance

Often less portable and more time consuming to implement

Code for Portability & Performance

Libraries

- Implement as much as possible using portable libraries

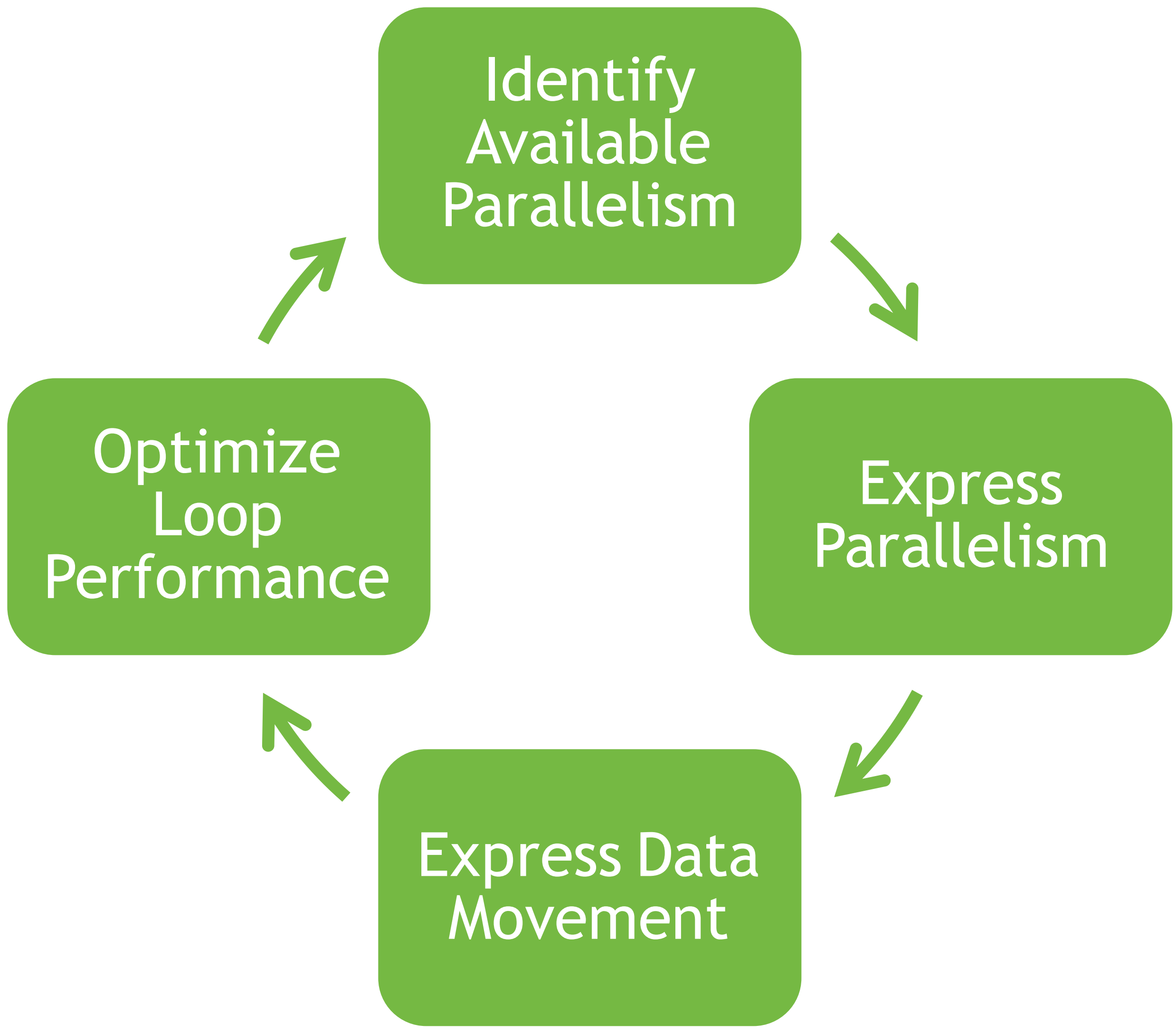
Directives

- Use directives for rapid and portable development

Languages

- Use lower level languages for important kernels

OpenACC Programming Cycle

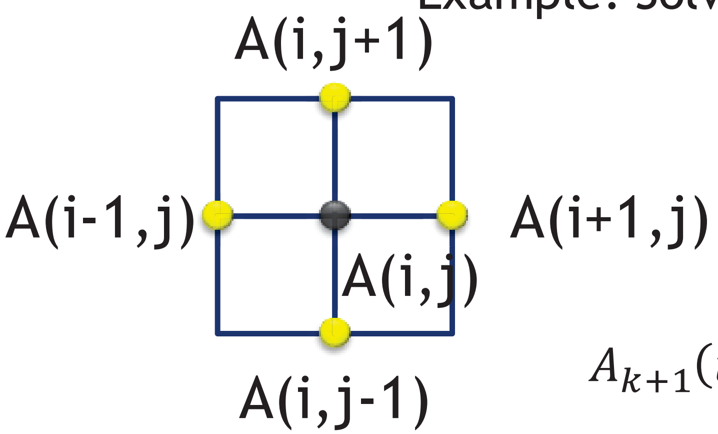

Identify Available Parallelism
Express Parallelism
Express Data Movement
Optimize Loop Performance

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++) {

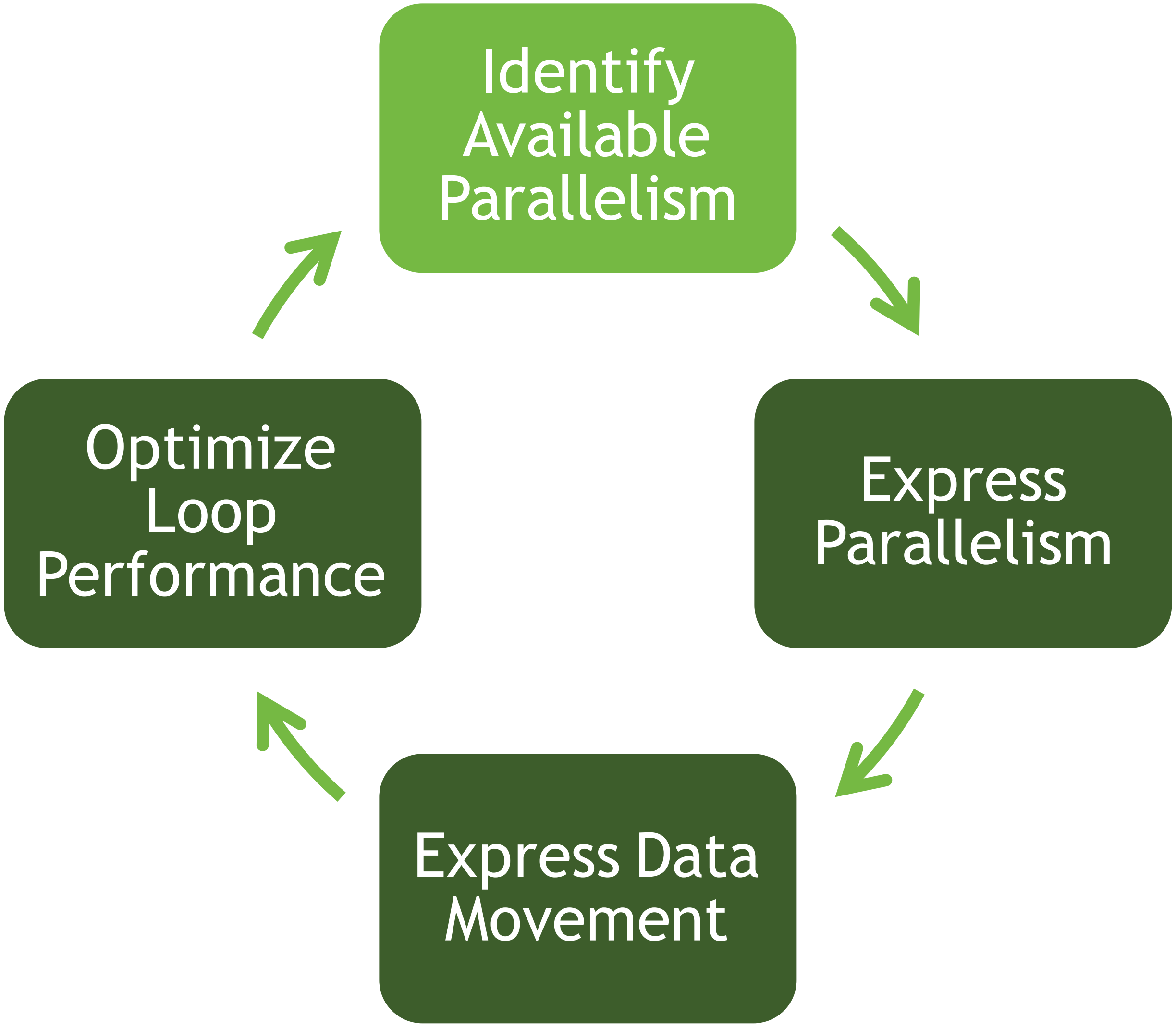
      Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                           A[j-1][i] + A[j+1][i]);

      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]));
    }
  }

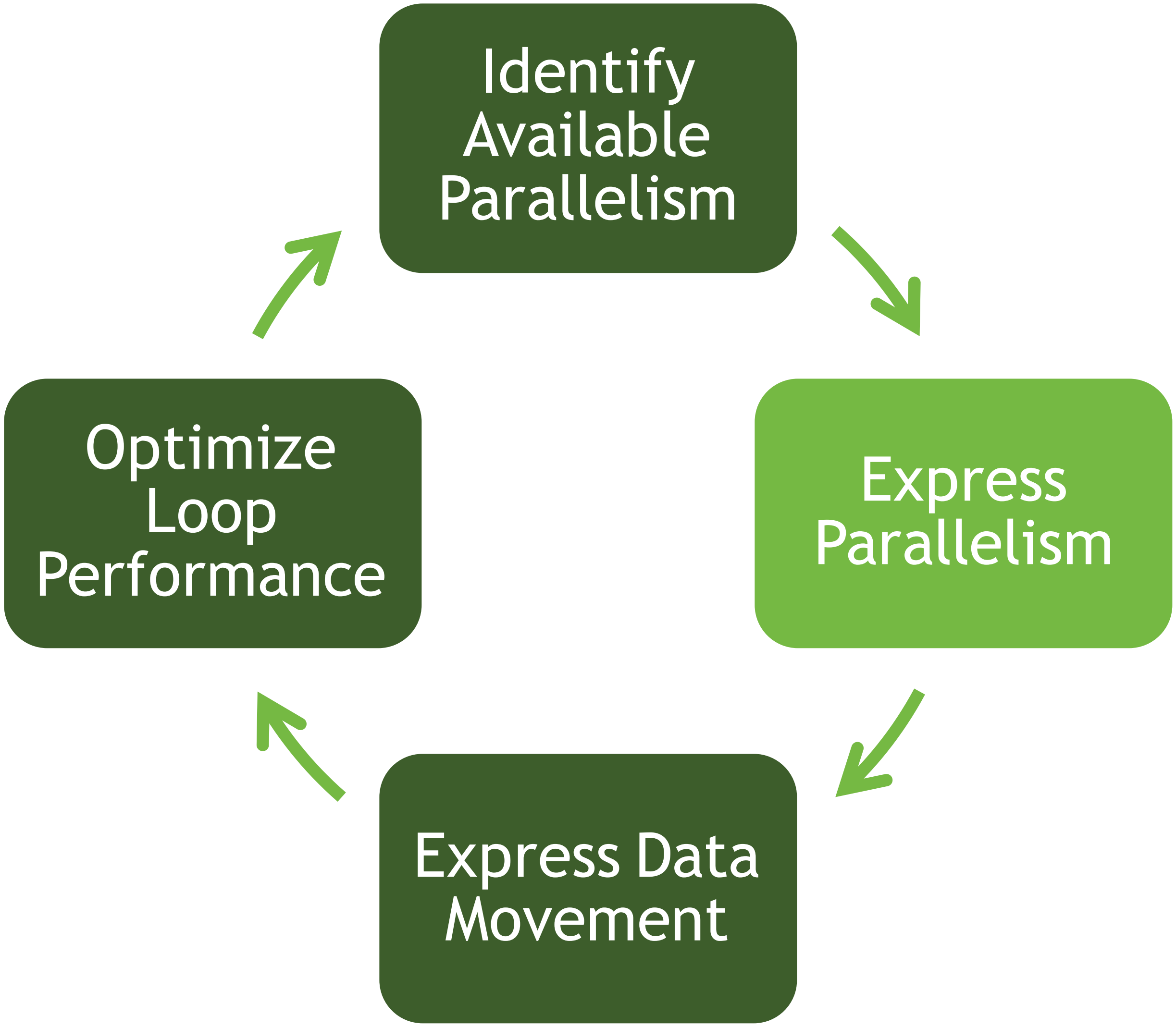
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }

  iter++;
}
```

Data dependency between iterations.

Independent loop iterations

Independent loop iterations



Identify Available Parallelism
Express Parallelism
Express Data Movement
Optimize Loop Performance

OpenACC kernels Directive

The kernels directive identifies a region that may contain *loops* that the compiler can turn into parallel *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.

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]);

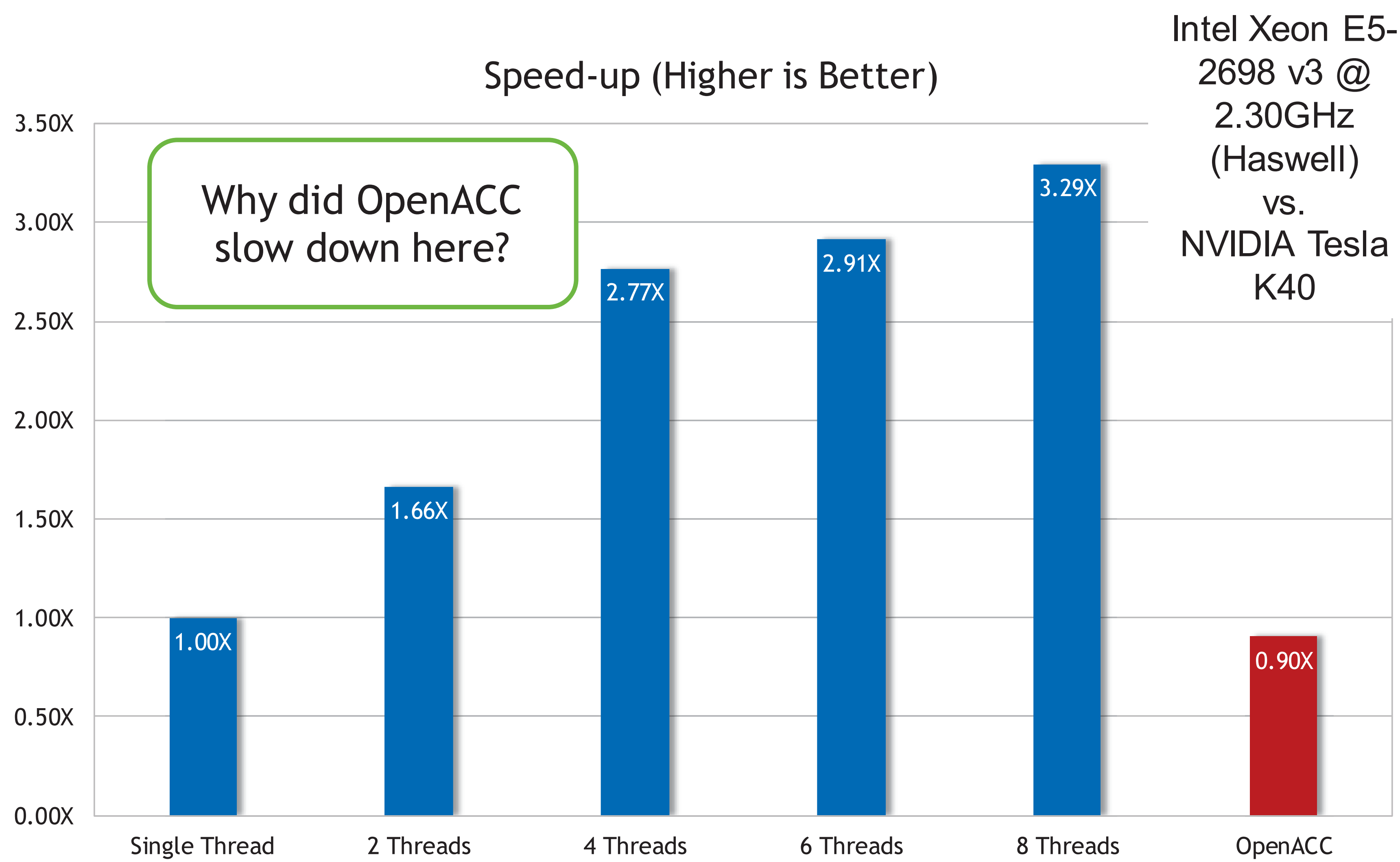
      err = max(err, abs(Anew[j][i] - A[j][i]));
    }
  }

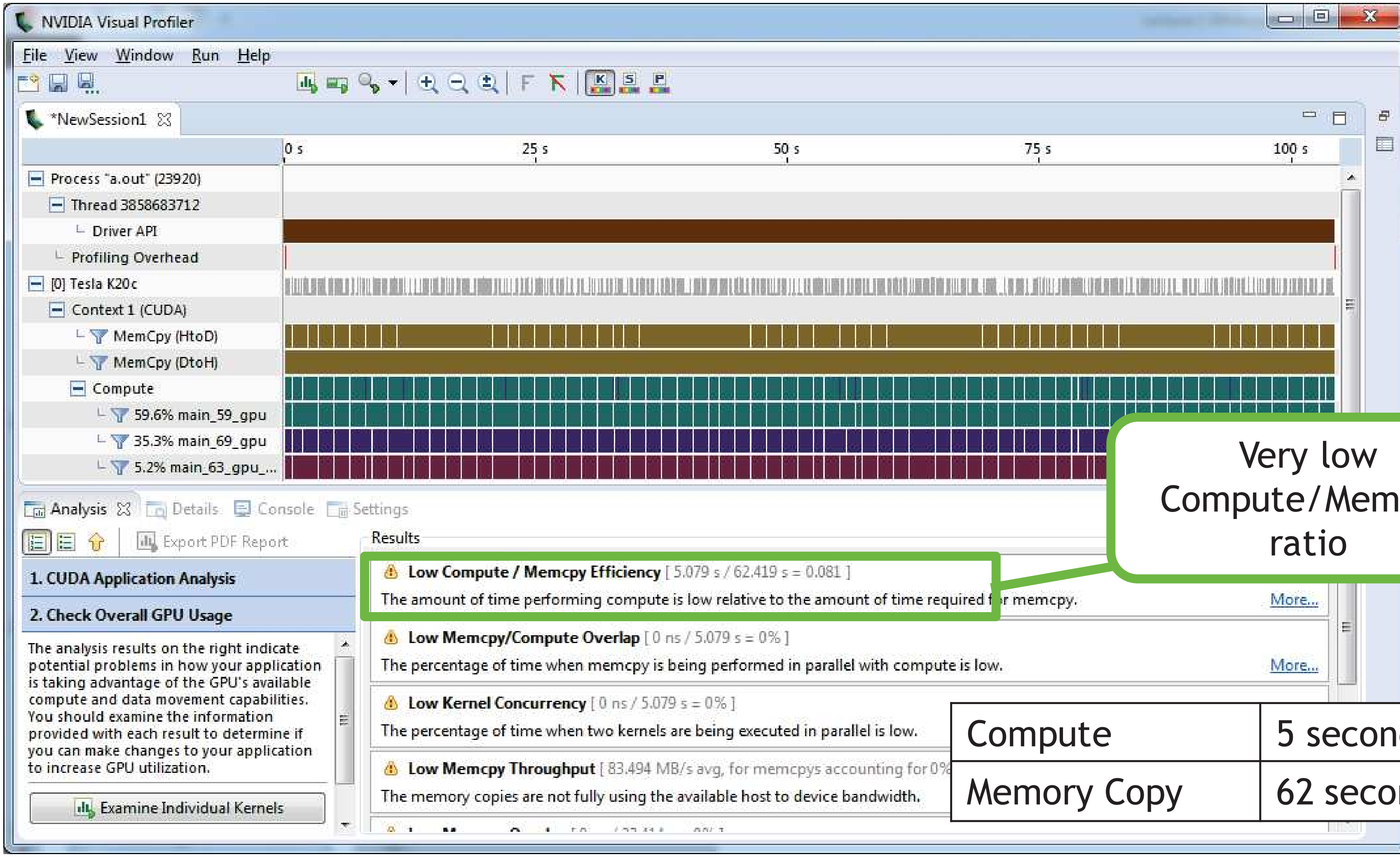
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }
}
  iter++;
}
```

Look for parallelism within this region.

Building the code

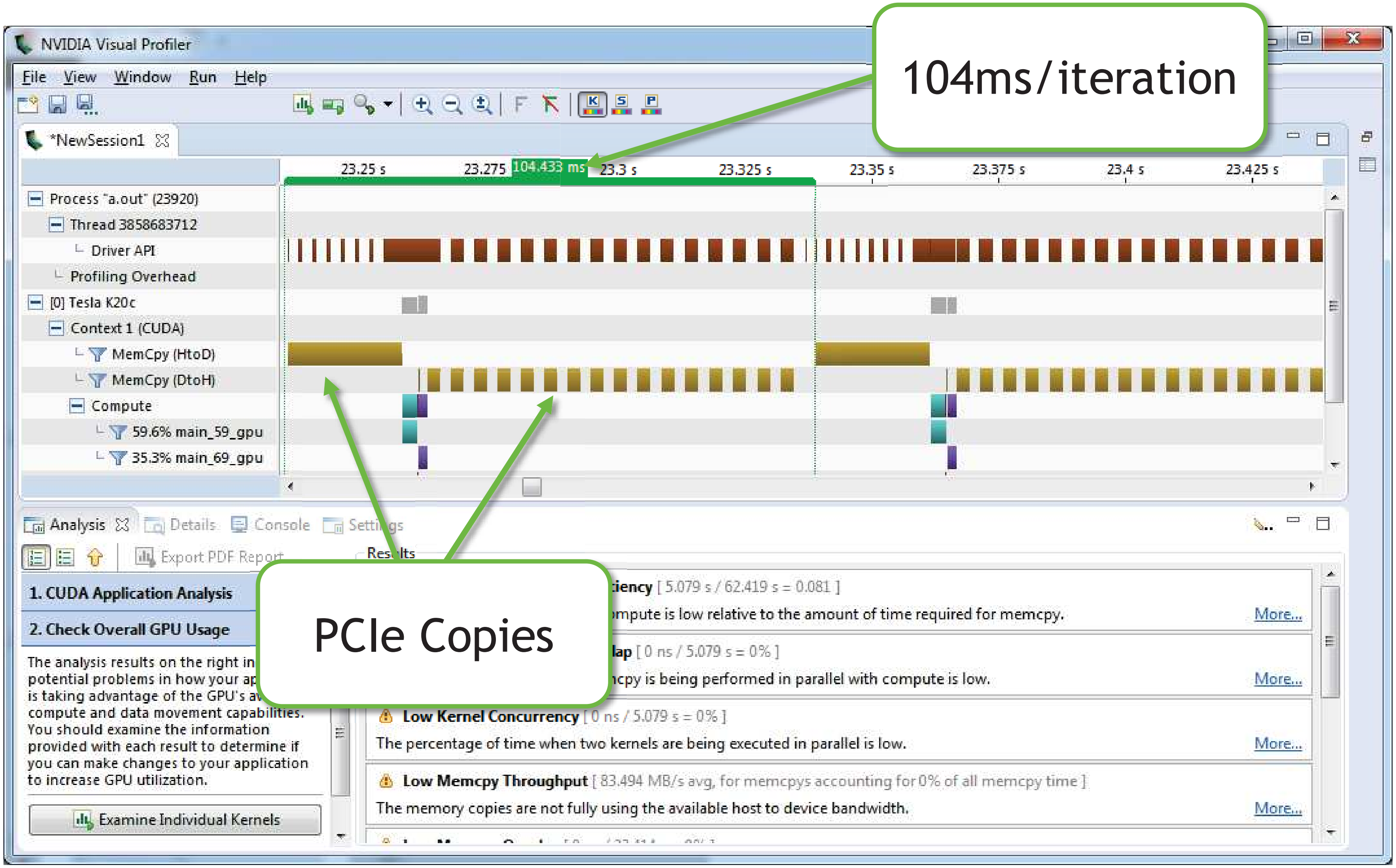
```
$ pgcc -fast -ta=tesla -Minfo=all laplace2d.c
main:
     40, Loop not fused: function call before adjacent loop
         Generated vector sse code for the loop
     51, Loop not vectorized/parallelized: potential early exits
     55, Generating copyout(Anew[1:4094][1:4094])
         Generating copyin(A[:][:])
         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 */
```


Speed-up (Higher is Better)
Intel Xeon E5-2698 v3 @ 2.30GHz (Haswell) vs. NVIDIA Tesla K40
Why did OpenACC slow down here?
3.50X
3.00X
2.50X
2.00X
1.50X
1.00X
0.50X
0.00X
1.00X
1.66X
2.77X
2.91X
3.29X
0.90X
Single Thread
2 Threads
4 Threads
6 Threads
8 Threads
OpenACC



Very low Compute/Memcpy ratio

Compute	5 seconds
Memory Copy	62 seconds


104ms/iteration
PCIe Copies
NVIDIA Visual Profiler
File View Window Run Help
*NewSession1
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
Settings
Export PDF Report
Results
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 av
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

```
#pragma acc kernels
```

A, Anew resident on accelerator

```
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!

Copy

A, Anew resident on accelerator

A, Anew resident on host

```
  ...
}
```

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]);

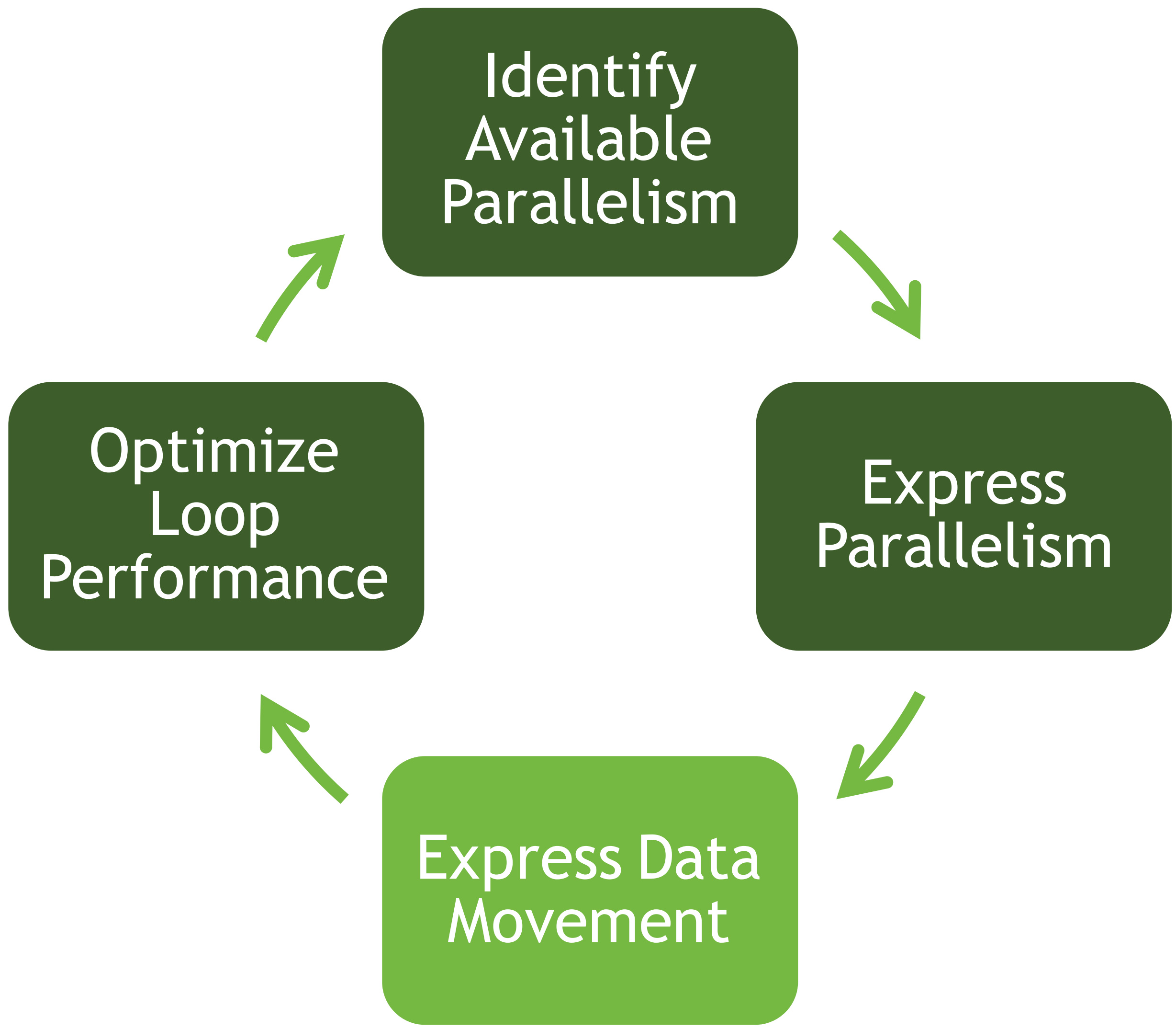
      err = max(err, abs(Anew[j][i] - A[j][i]));
    }
  }

  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++;
}
```

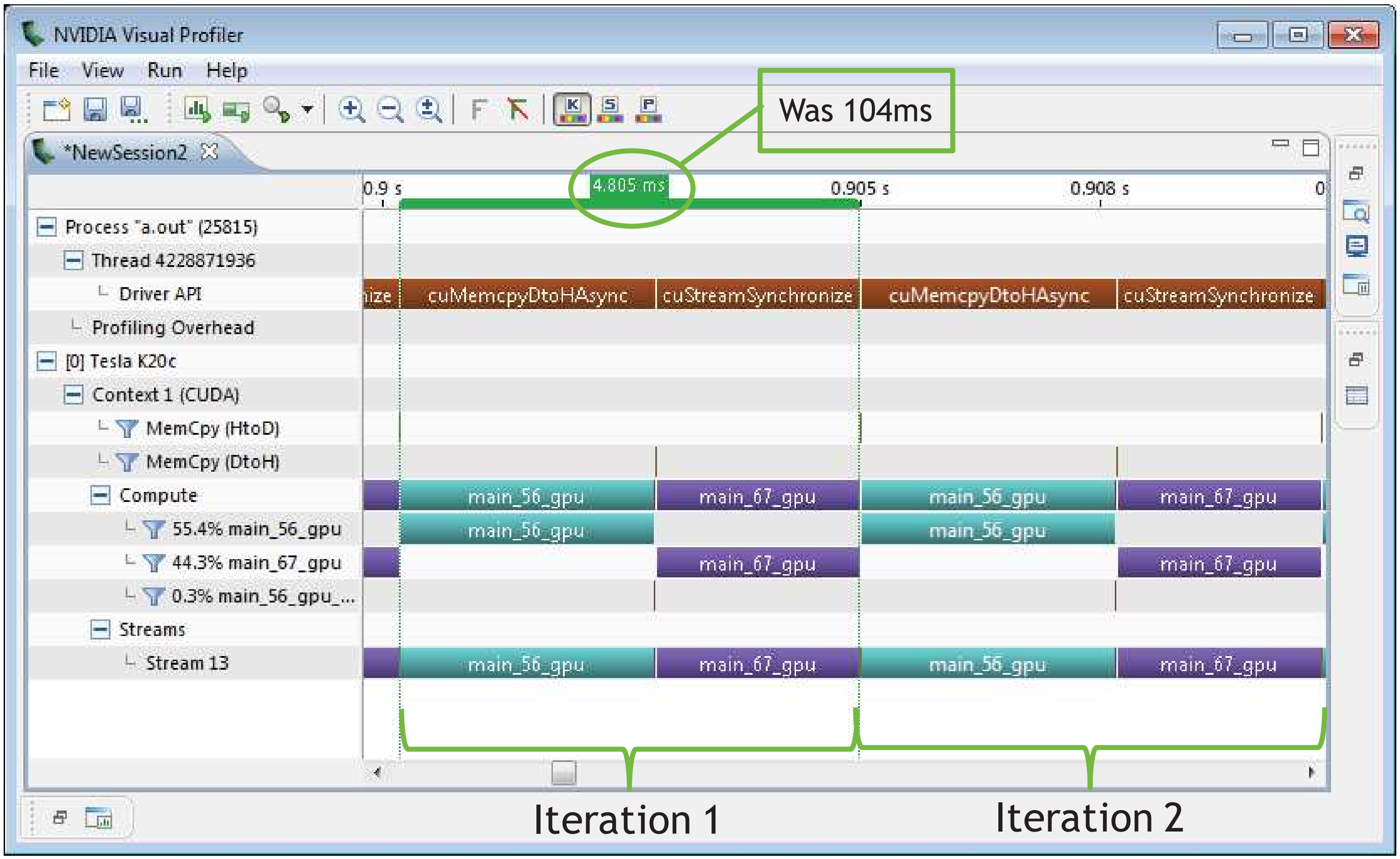
Copy A to/from the accelerator only when needed.

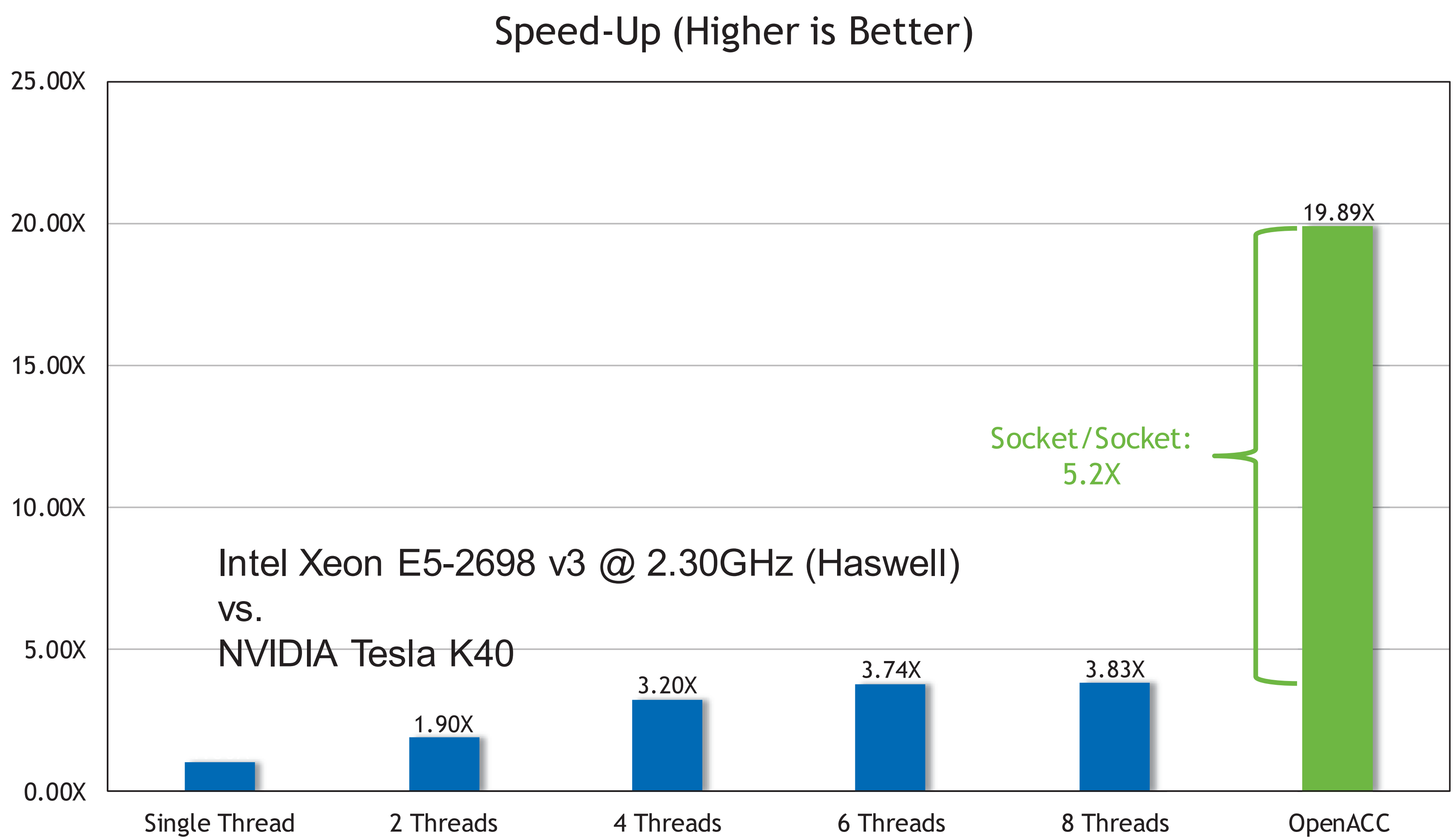
Create Anew as a device temporary.

Rebuilding the code

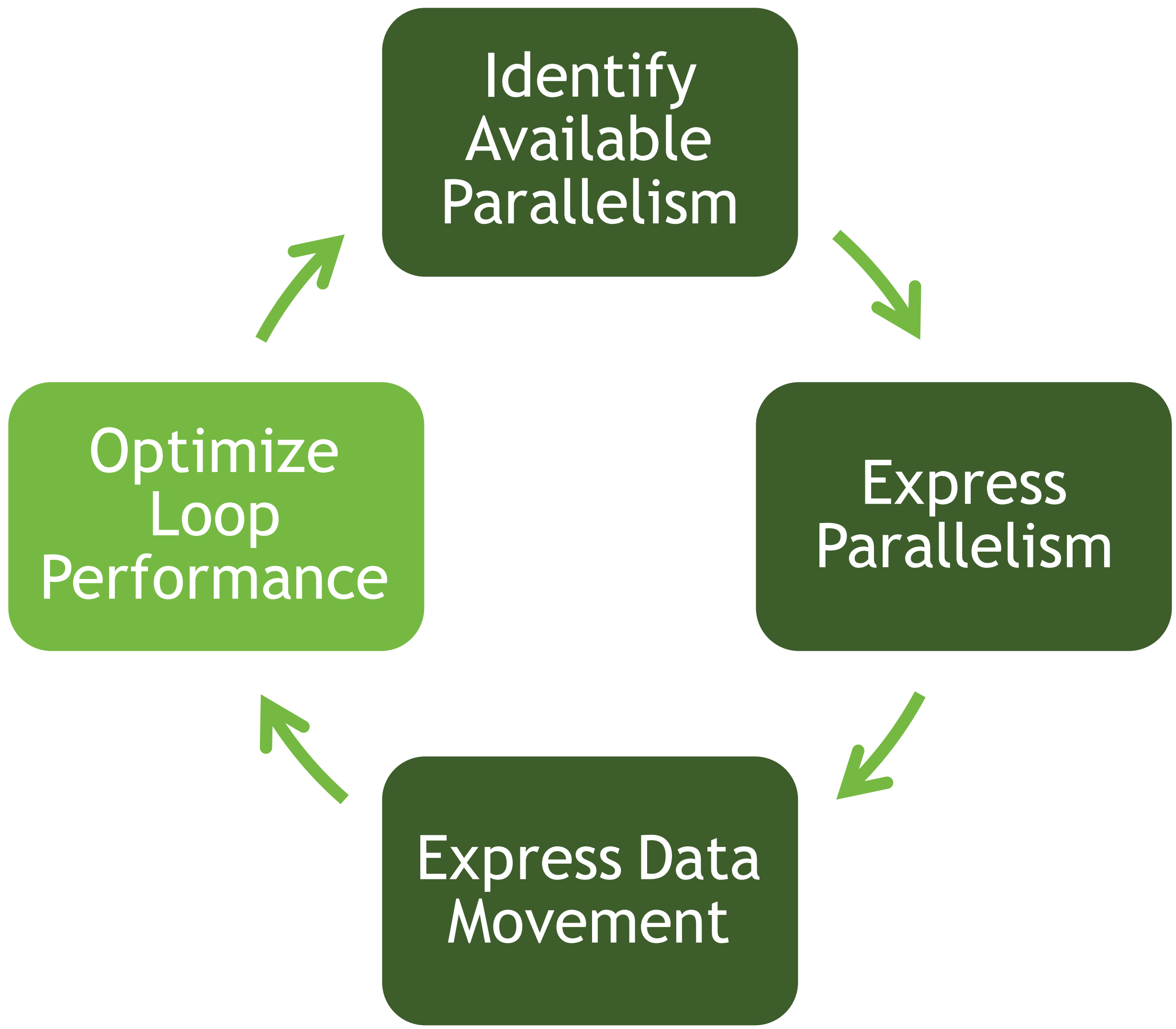
```
$ pgcc -fast -acc -ta=tesla -Minfo=all laplace2d.c
main:
     40, Loop not fused: function call before adjacent loop
         Generated vector sse code for the loop
     51, Generating copy(A[:][:])
         Generating create(Anew[:][:])
         Loop not vectorized/parallelized: potential early 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
19.89X
Socket/Socket:
5.2X
Intel Xeon E5-2698 v3 @ 2.30GHz (Haswell)
vs.
NVIDIA Tesla K40
1.90X
3.20X
3.74X
3.83X
Single Thread
2 Threads
4 Threads
6 Threads
8 Threads
OpenACC


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.

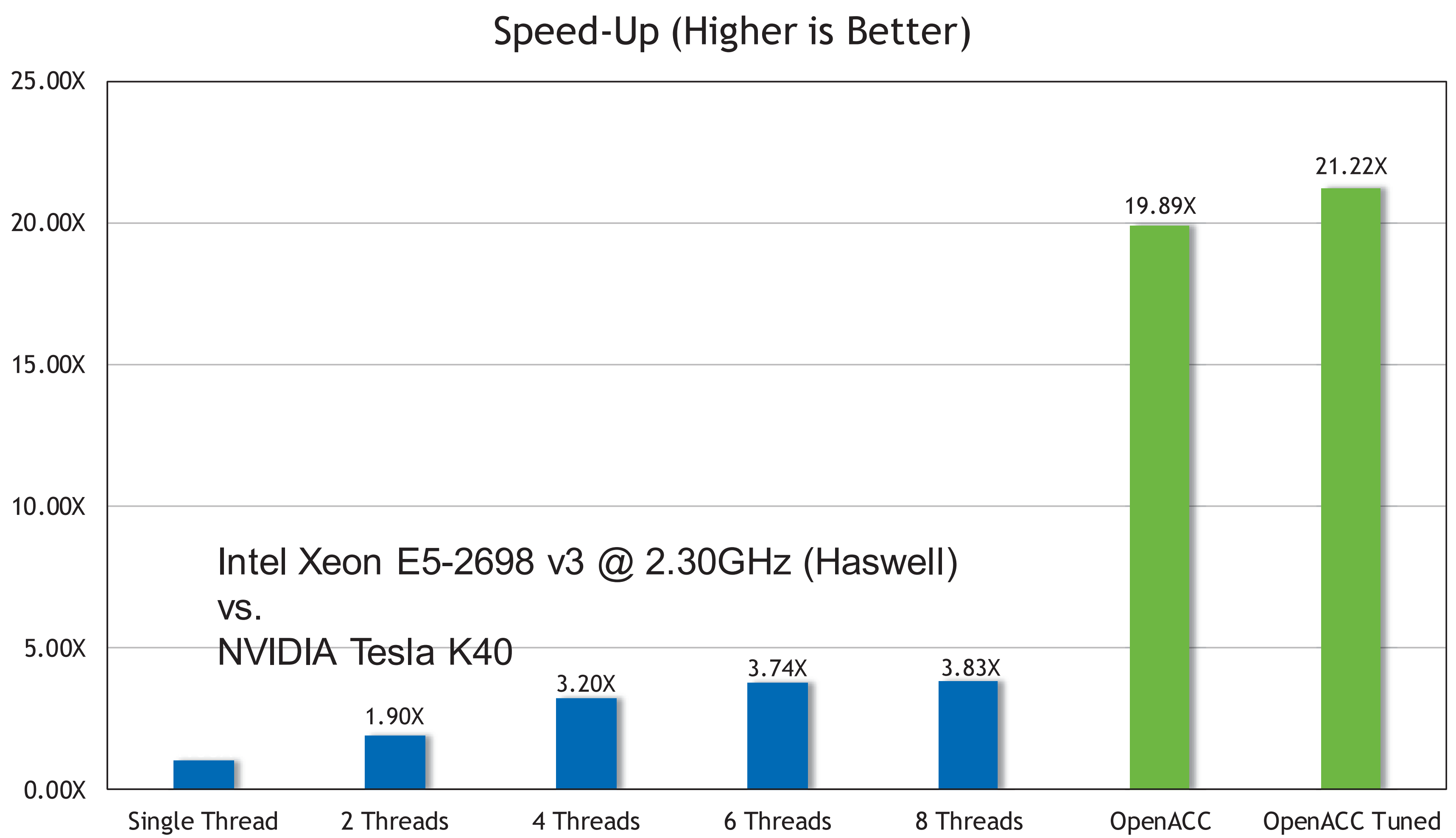
Optimize Loop Performance

```
#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)
25.00X
20.00X
15.00X
10.00X
5.00X
0.00X
Intel Xeon E5-2698 v3 @ 2.30GHz (Haswell)
vs.
NVIDIA Tesla K40
1.90X
3.20X
3.74X
3.83X
19.89X
21.22X
Single Thread
2 Threads
4 Threads
6 Threads
8 Threads
OpenACC
OpenACC Tuned

The OpenACC Toolkit

Introducing the New OpenACC Toolkit

Free Toolkit Offers Simple & Powerful Path to Accelerated Computing

http://developer.nvidia.com/openacc

PGI Compiler
Free OpenACC compiler for academia

NVProf Profiler
Easily find where to add compiler directives

GPU Wizard
Identify which GPU libraries can jumpstart code

Code Samples
Learn from examples of real-world algorithms

Documentation
Quick start guide, Best practices, Forums