OpenACC Fundamentals

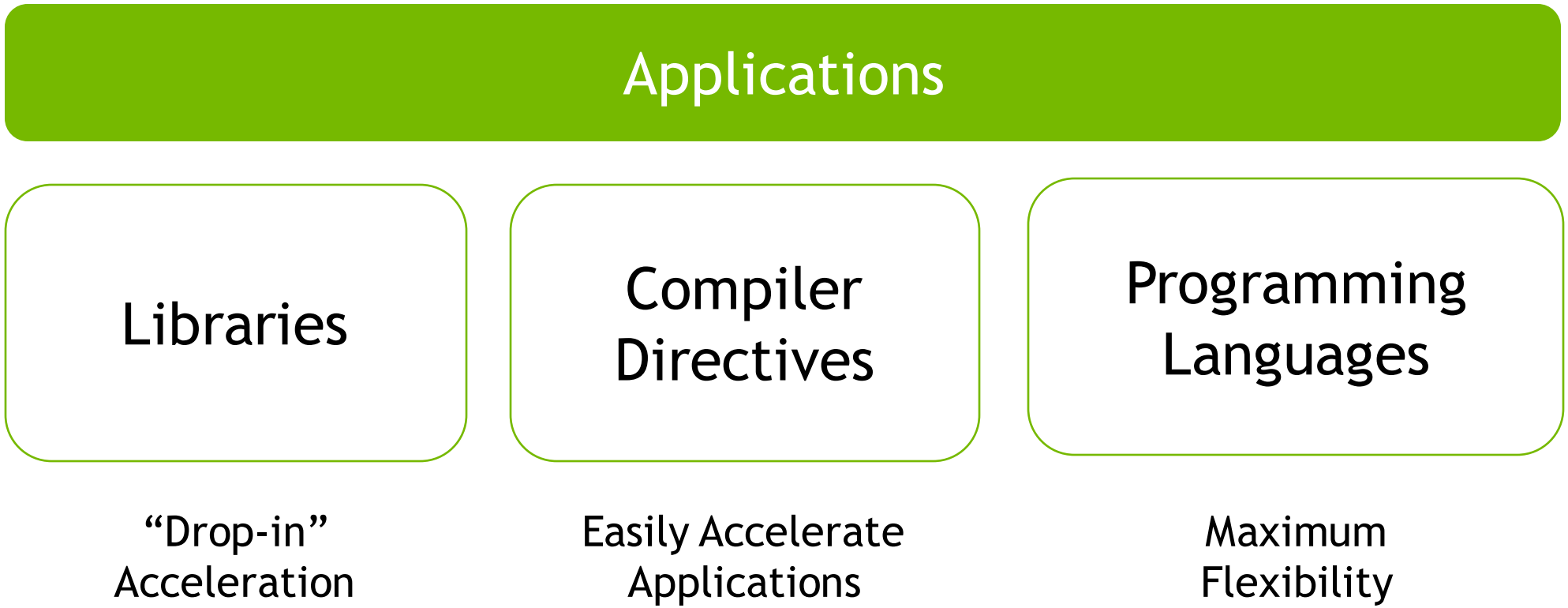
Jeff Larkin <jlarkin@nvidia.com>, November 16, 2016

AGENDA

What is OpenACC?

OpenACC by Example

3 Ways to Program GPUs



OpenACC Directives

Manage Data Movement

Initiate Parallel Execution

Optimize Loop Mappings

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

OpenACC
Directives for Accelerators

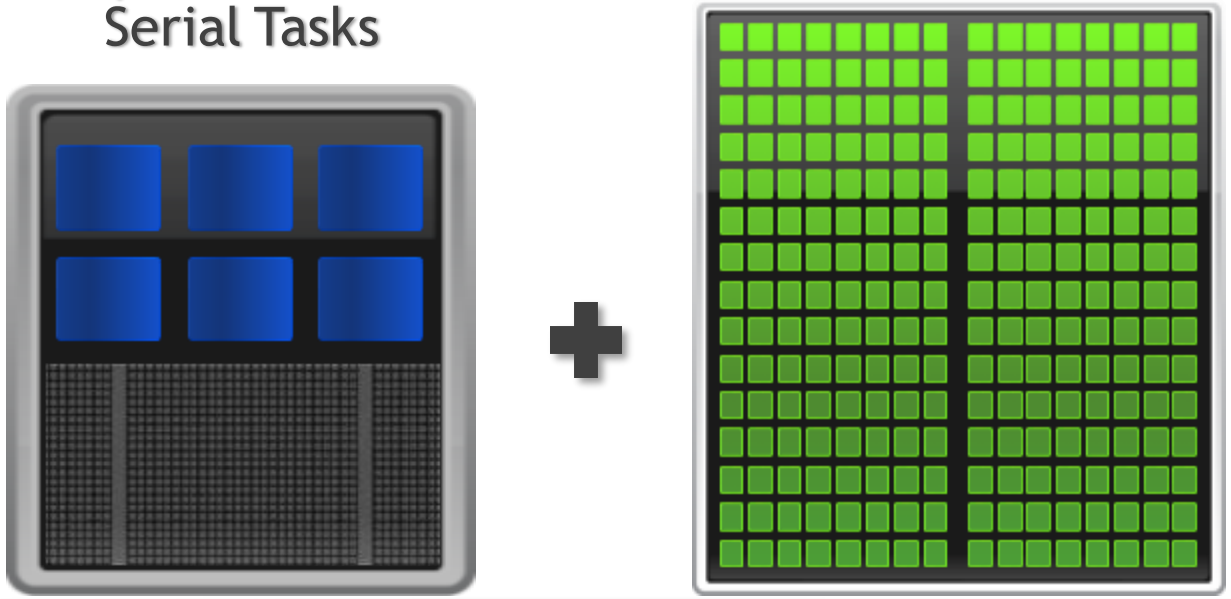
- Incremental
- Single source
- Interoperable
- Performance portable
- CPU, GPU, MIC

Accelerated Computing

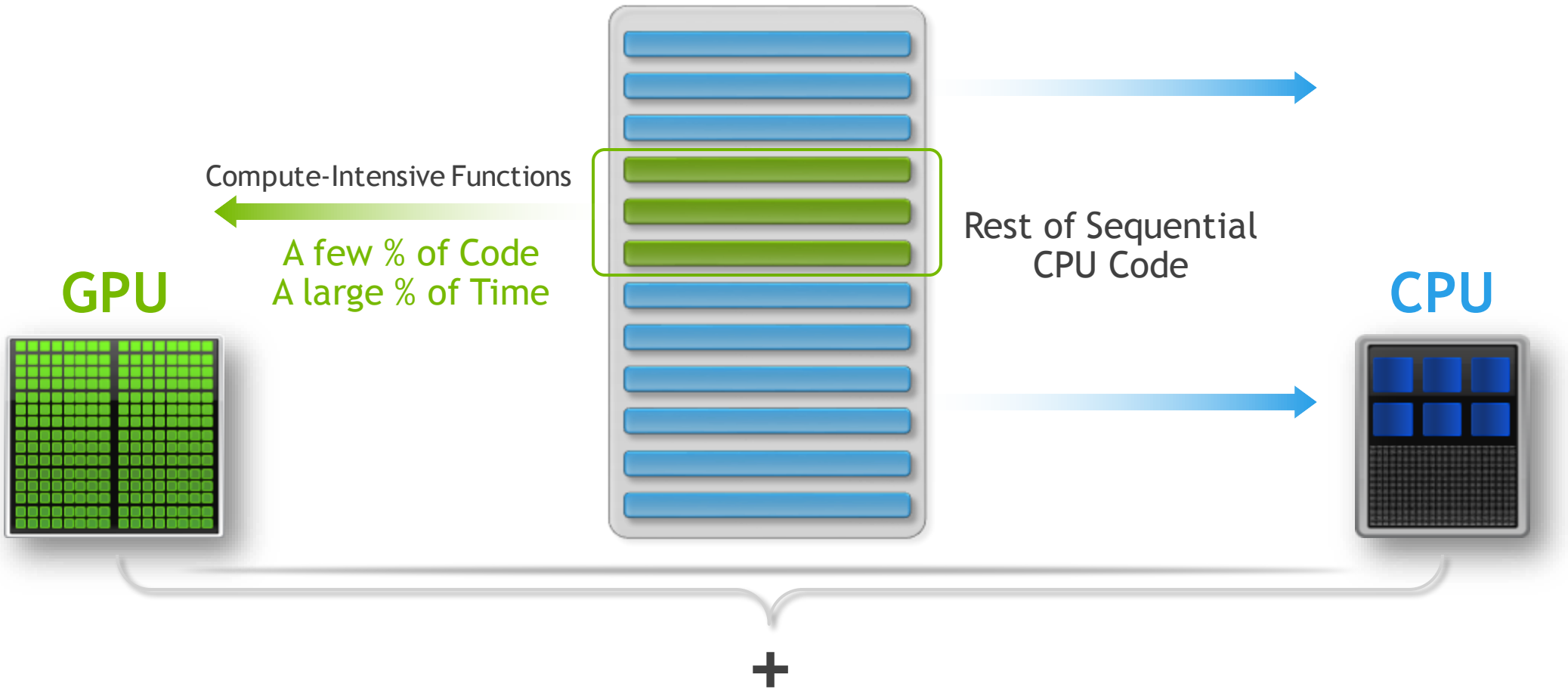
10x Performance & 5x Energy Efficiency for HPC

CPU
Optimized for
Serial Tasks

GPU Accelerator
Optimized for
Parallel Tasks

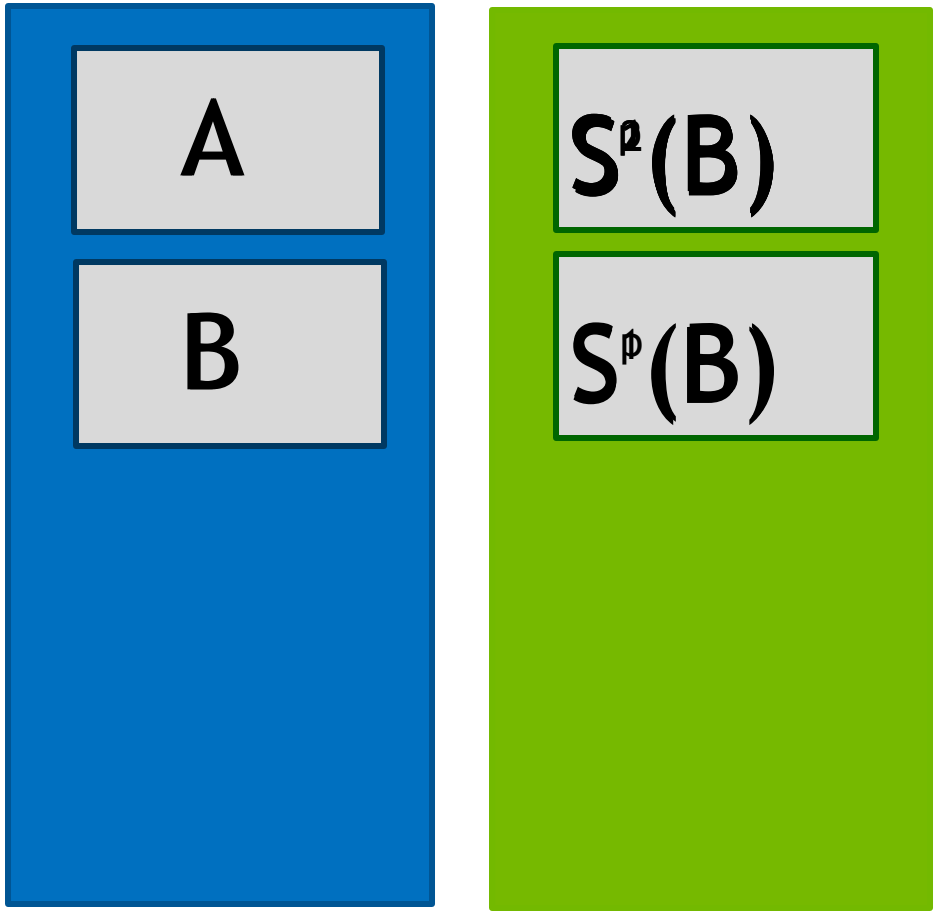


What is Accelerated Computing?



OpenACC Example

```
#pragma acc data \
        copy(b[0:n][0:m]) \
        create(a[0:n][0:m])
{
for (iter = 1; iter <= p; ++iter){
  #pragma acc kernels
  {
  for (i = 1; i < n-1; ++i){
    for (j = 1; j < m-1; ++j){
      a[i][j]=w0*b[i][j]+
              w1*(b[i-1][j]+b[i+1][j]+
                  b[i][j-1]+b[i][j+1])+
              w2*(b[i-1][j-1]+b[i-1][j+1]+
                  b[i+1][j-1]+b[i+1][j+1]);
  }
  for( i = 1; i < n-1; ++i )
    for( j = 1; j < m-1; ++j )
      b[i][j] = a[i][j];
  } }
}
}
```

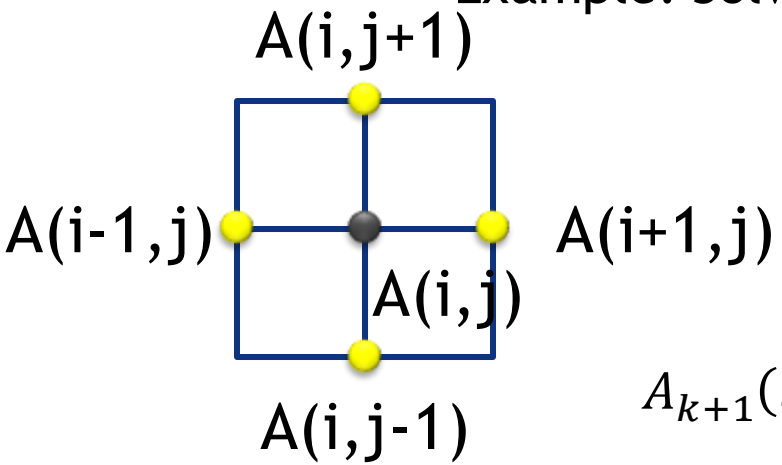


Example: Jacobi Iteration

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

Look For 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

Max Reduction required

Independent loop iterations

OPENACC DIRECTIVE SYNTAX

C/C++

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

…often followed by a structured code block

Fortran

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

…often paired with a matching end directive surrounding a structured code block:

```
!$acc end directive
```

Don’t forget acc

OpenACC Parallel Directive

Generates parallelism

```
#pragma acc parallel
{
}
```

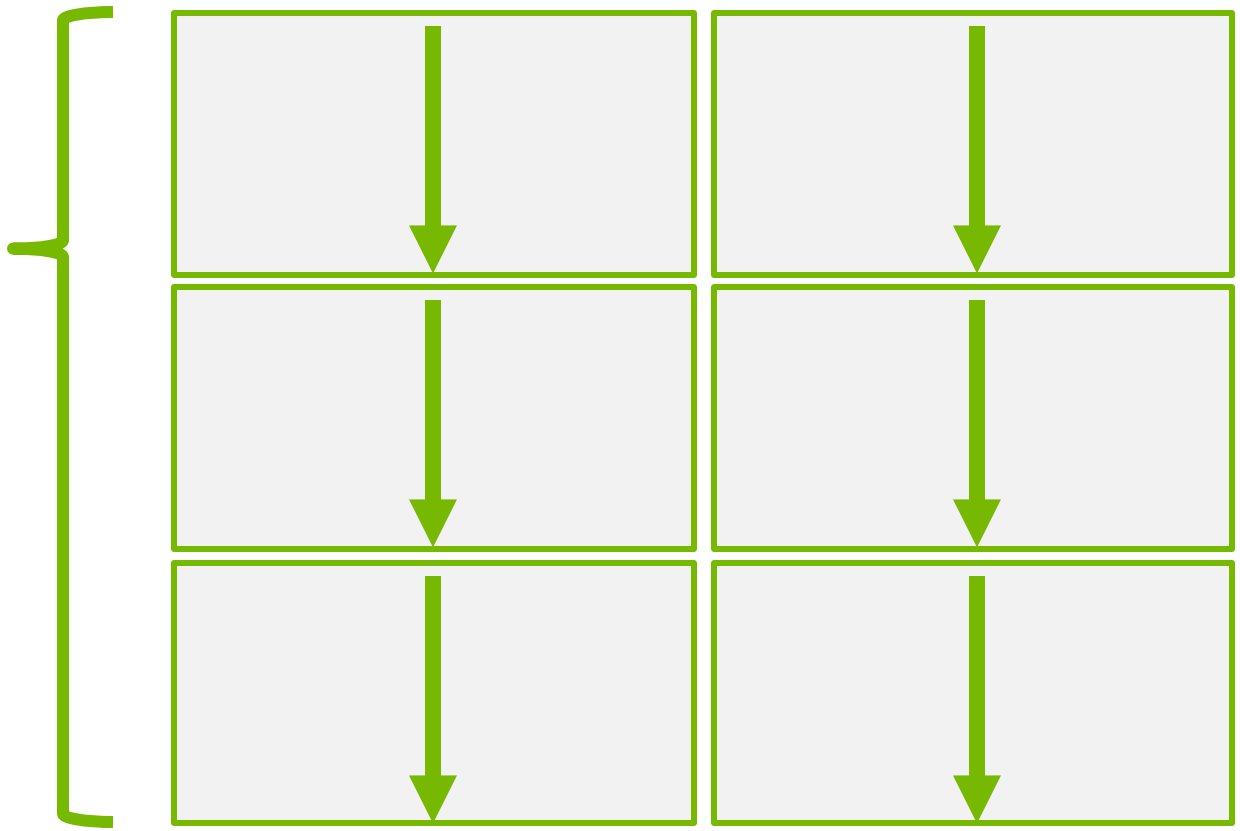
When encountering the *parallel* directive, the compiler will generate *1 or more parallel gangs*, which execute redundantly.

OpenACC Parallel Directive

Generates parallelism

```
#pragma acc parallel
{
}
```

When encountering the *parallel* directive, the compiler will generate *1 or more parallel gangs*, which execute redundantly.

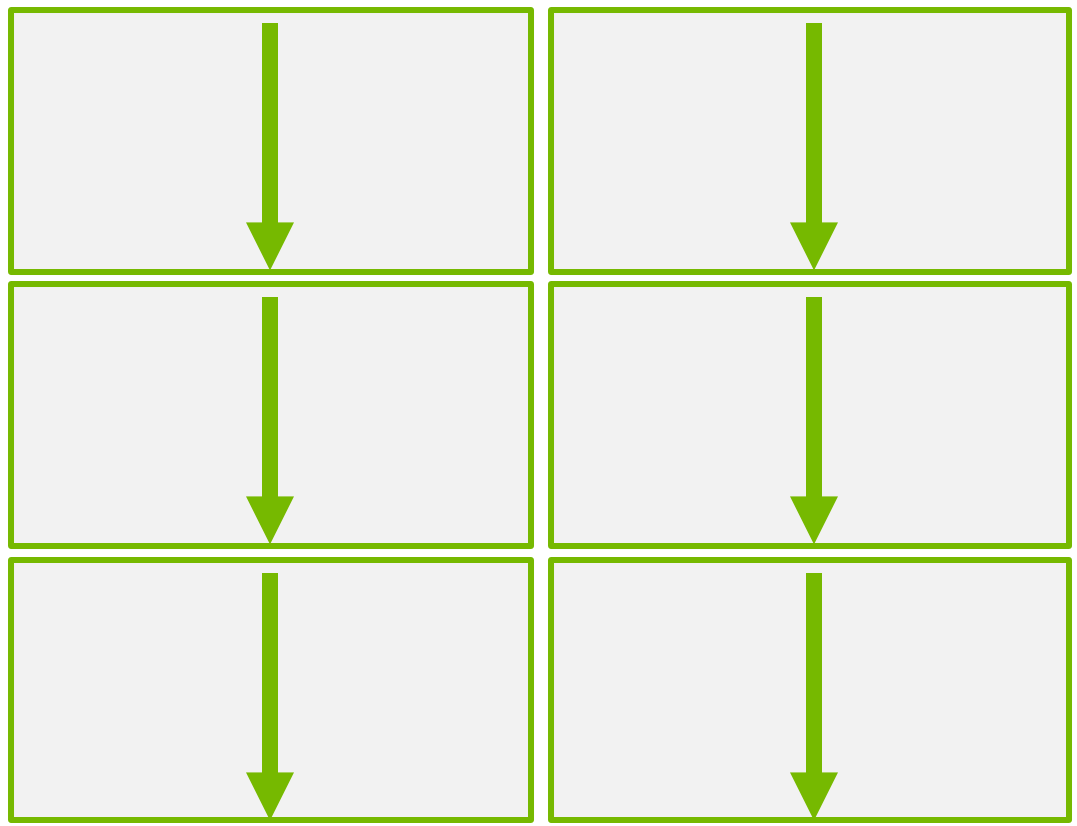


OpenACC Loop Directive

Identifies loops to run in parallel

```
#pragma acc parallel
{
  #pragma acc loop
  for (i=0;i<N;i++)
  {
  }
}
```

The *loop* directive informs the compiler which loops to parallelize.

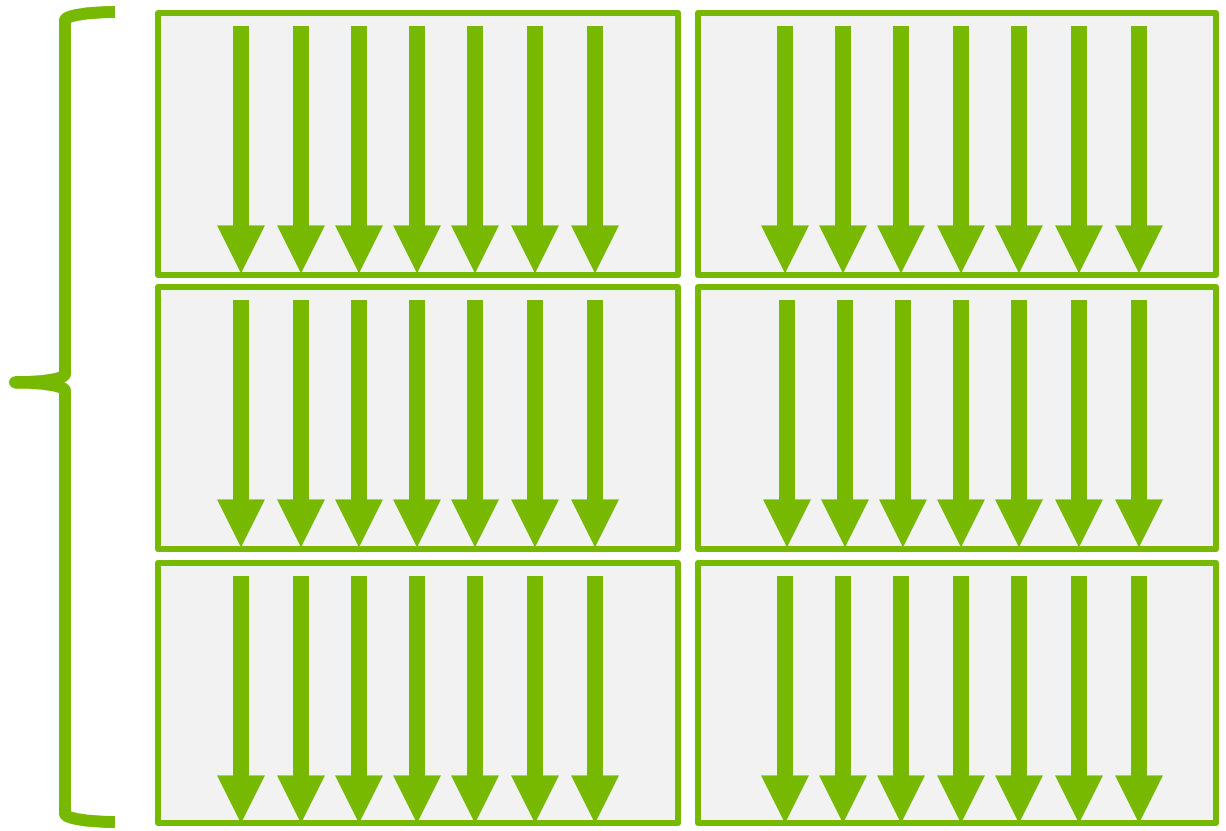


OpenACC Loop Directive

Identifies loops to run in parallel

```
#pragma acc parallel
{
  #pragma acc loop
  for (i=0;i<N;i++)
  {
  }
}
```

The *loop* directive informs the compiler which loops to parallelize.

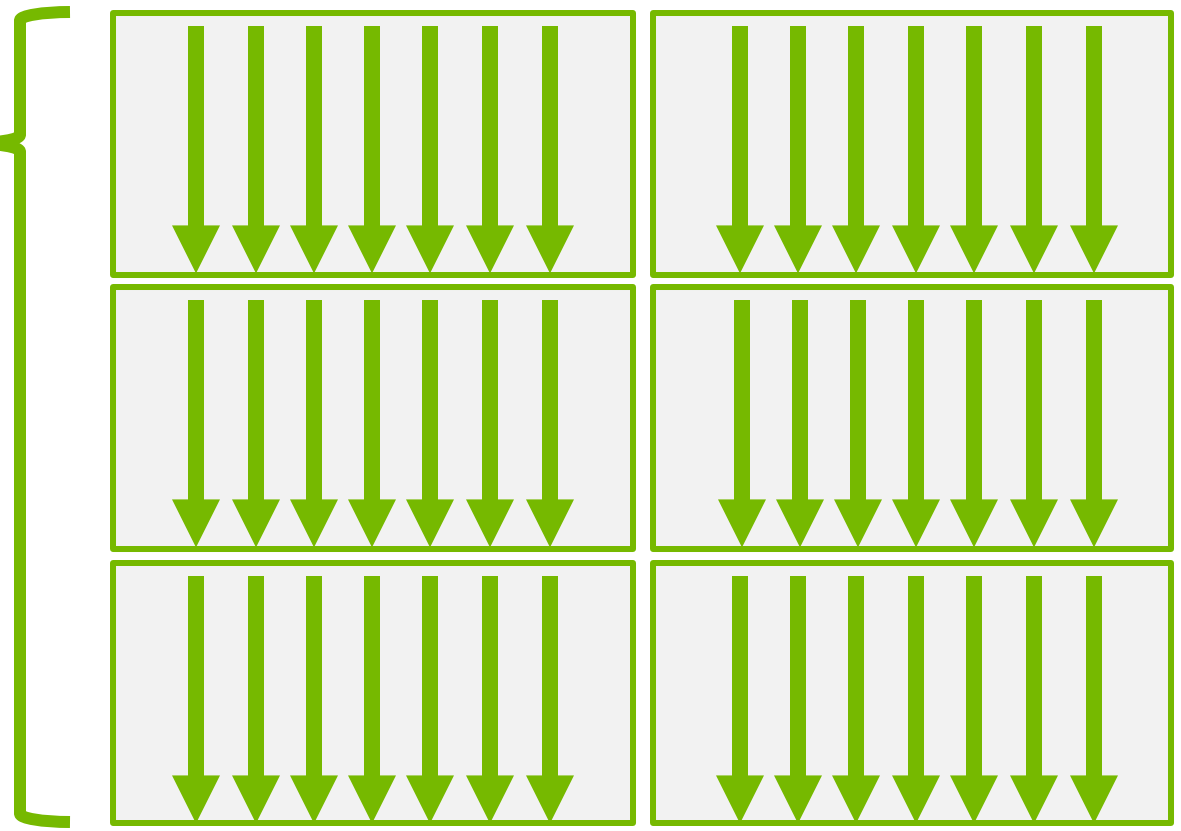


OpenACC Parallel Loop Directive

Generates parallelism and identifies loop in one directive

```
#pragma acc parallel loop
  for (i=0;i<N;i++)
  {
  }
```

The *parallel* and *loop* directives are frequently combined into one.



PARALLELIZE WITH OPENACC

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

#pragma acc parallel loop reduction(max:err)
  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 parallel loop
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }

  iter++;
}
```

Parallelize loop on accelerator

Parallelize loop on accelerator

* A *reduction* means that all of the N*M values for err will be reduced to just one, the max.

BUILDING THE CODE

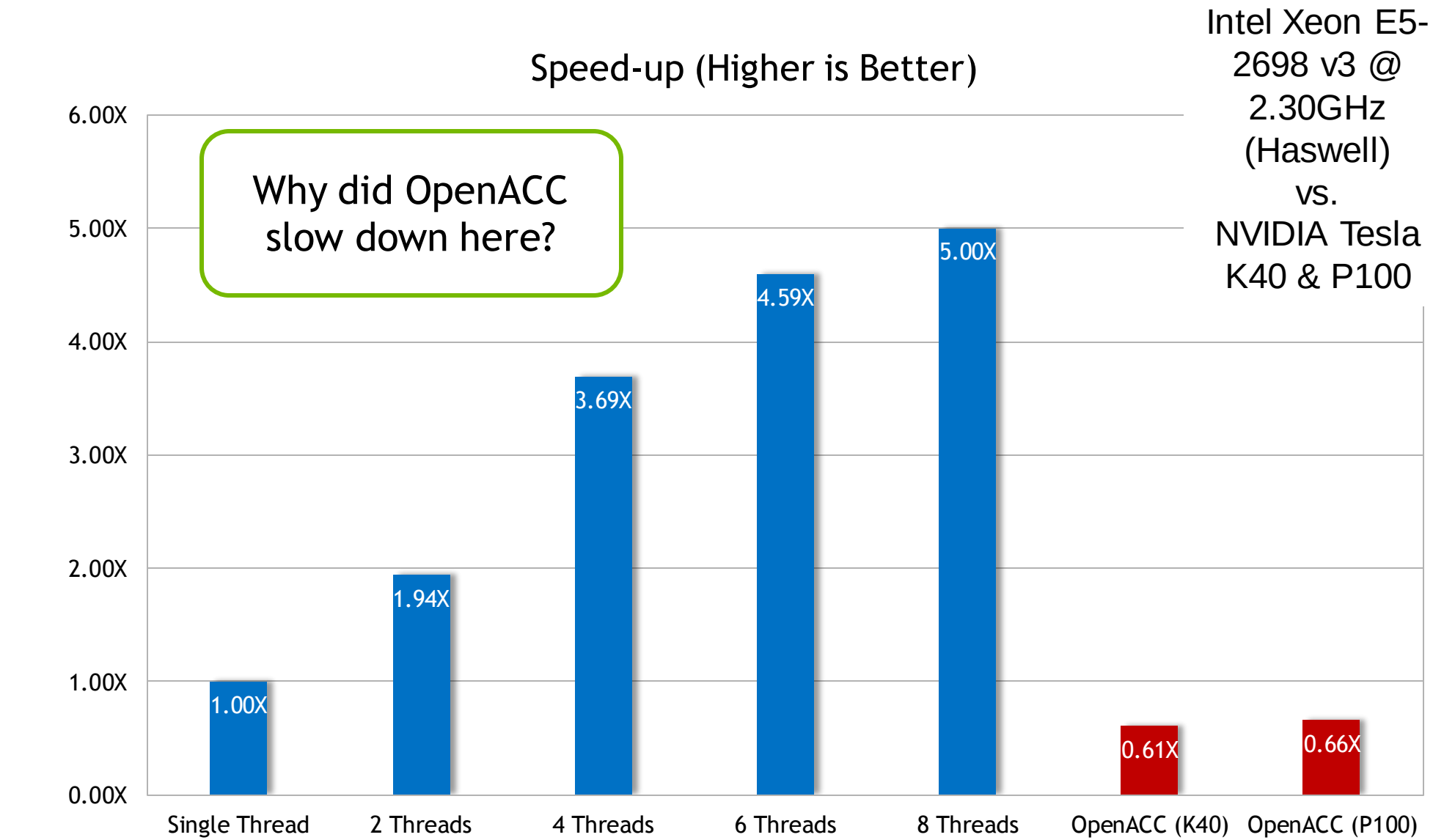
```
$ pgcc -fast -acc -ta=tesla -Minfo=all laplace2d.c
main:
     40, Loop not fused: function call before adjacent loop
         Generated vector sse code for the loop
     51, Loop not vectorized/parallelized: potential early exits
     55, Accelerator kernel generated
         55, Max reduction generated for error
         56, #pragma acc loop gang /* blockIdx.x */
         58, #pragma acc loop vector(256) /* threadIdx.x */
     55, Generating copyout(Anew[1:4094][1:4094])
         Generating copyin(A[:][:])
         Generating Tesla code
     58, Loop is parallelizable
     66, Accelerator kernel generated
         67, #pragma acc loop gang /* blockIdx.x */
         69, #pragma acc loop vector(256) /* threadIdx.x */
     66, Generating copyin(Anew[1:4094][1:4094])
         Generating copyout(A[1:4094][1:4094])
         Generating Tesla code
     69, Loop is parallelizable
```

BUILDING THE CODE

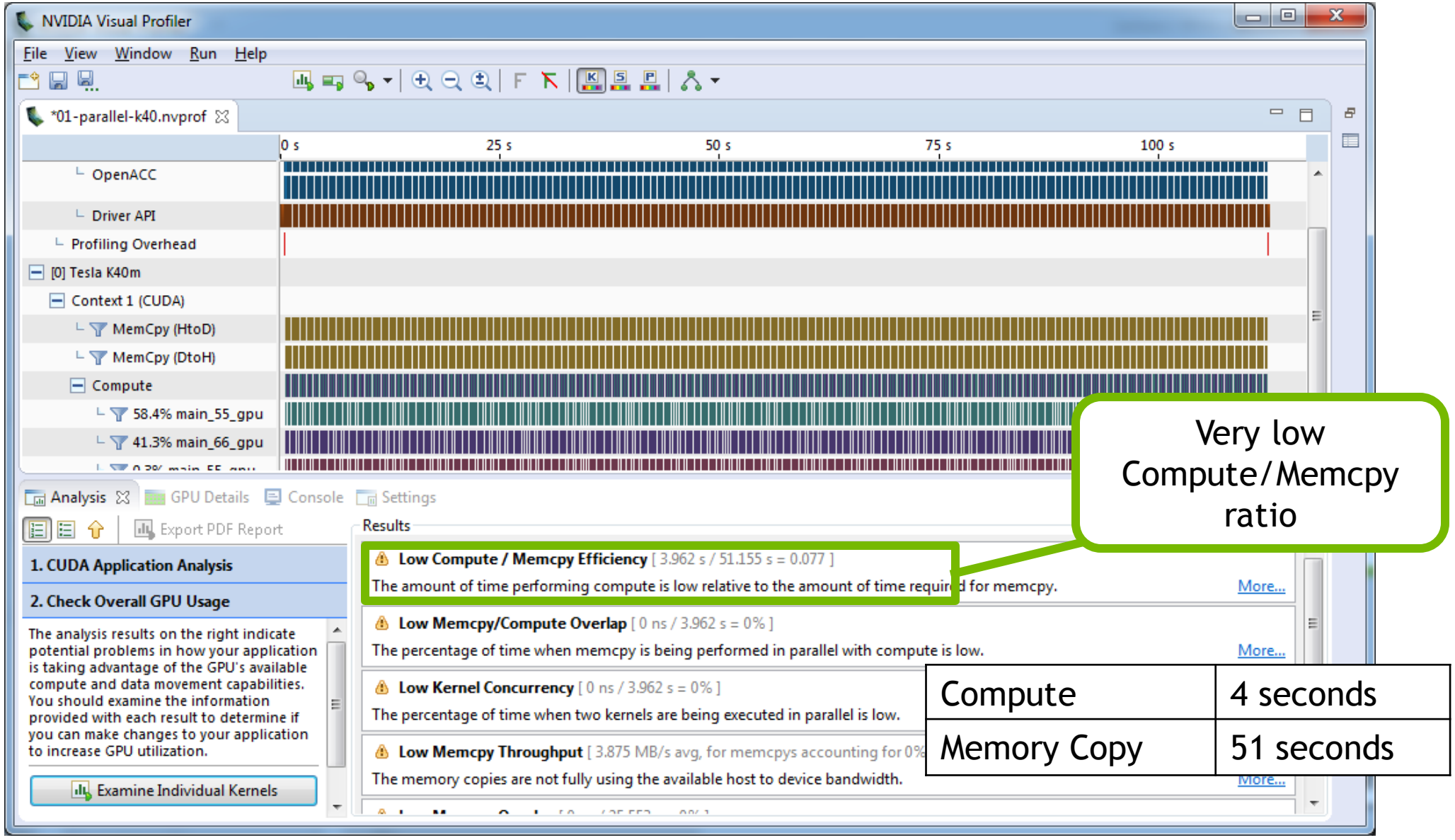
```
$ pgcc -fast -acc -ta=tesla -Minfo=all laplace2d.c
main:
     40, Loop not fused: function call before adjacent loop
         Generated vector sse code for the loop
     51, Loop not vectorized/parallelized: potential early exits
     55, Accelerator kernel generated
         55, Max reduction generated for error
         56, #pragma acc loop gang /* blockIdx.x */
         58, #pragma acc loop vector(256) /* threadIdx.x */
     55, Generating copyout(Anew[1:4094][1:4094])
         Generating copyin(A[:][:])
         Generating Tesla code
     58, Loop is parallelizable
     66, Accelerator kernel generated
         67, #pragma acc loop gang /* blockIdx.x */
         69, #pragma acc loop vector(256) /* threadIdx.x */
     66, Generating copyin(Anew[1:4094][1:4094])
         Generating copyout(A[1:4094][1:4094])
         Generating Tesla code
     69, Loop is parallelizable
```

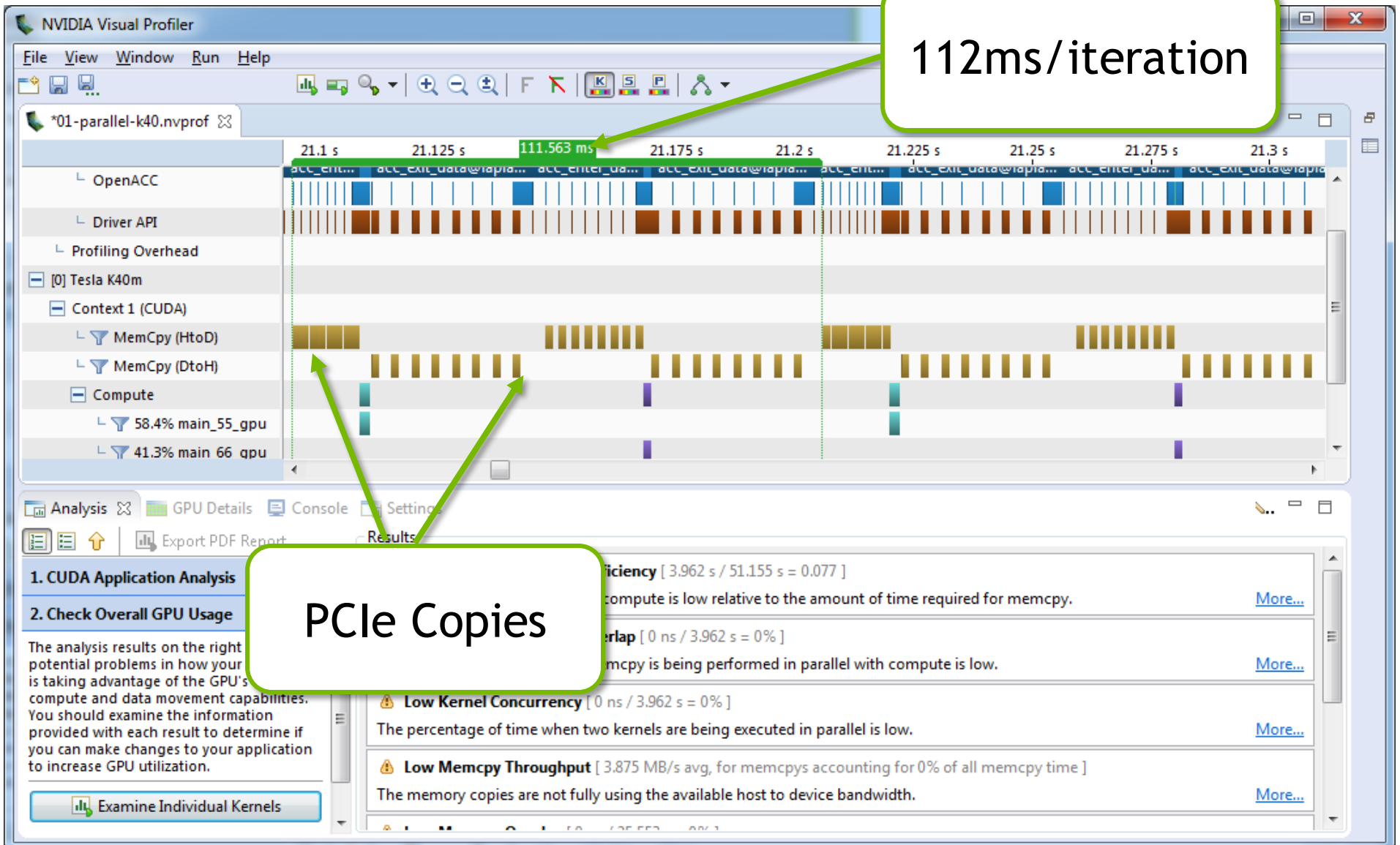
BUILDING THE CODE

```
$ 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, Loop not vectorized/parallelized: potential early exits
     55, Accelerator kernel generated
         55, Max reduction generated for error
         56, #pragma acc loop gang /* blockIdx.x */
         58, #pragma acc loop vector(256) /* threadIdx.x */
     55, Generating copyout(Anew[1:4094][1:4094])
         Generating copyin(A[:][:])
         Generating Tesla code
     58, Loop is parallelizable
     66, Accelerator kernel generated
         67, #pragma acc loop gang /* blockIdx.x */
         69, #pragma acc loop vector(256) /* threadIdx.x */
     66, Generating copyin(Anew[1:4094][1:4094])
         Generating copyout(A[1:4094][1:4094])
         Generating Tesla code
     69, Loop is parallelizable
```



Compiler: PGI 16.10


NVIDIA Visual Profiler
File View Window Run Help
*01-parallel-k40.nvprof
0 s
25 s
50 s
75 s
100 s
OpenACC
Driver API
Profiling Overhead
[0] Tesla K40m
Context 1 (CUDA)
MemCpy (HtoD)
MemCpy (DtoH)
Compute
58.4% main_55_gpu
41.3% main_66_gpu
Analysis
GPU Details
Console
Settings
Export PDF Report
1. CUDA Application Analysis
2. Check Overall GPU Usage
The analysis results on the right indicate potential problems in how your application is taking advantage of the GPU's available 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
Results
Low Compute / Memcpy Efficiency [3.962 s / 51.155 s = 0.077]
The amount of time performing compute is low relative to the amount of time requir d for memcpy.
More...
Low Memcpy/Compute Overlap [0 ns / 3.962 s = 0%]
The percentage of time when memcpy is being performed in parallel with compute is low.
More...
Low Kernel Concurrency [0 ns / 3.962 s = 0%]
The percentage of time when two kernels are being executed in parallel is low.
Low Memcpy Throughput [3.875 MB/s avg, for memcpys accounting for 0%
The memory copies are not fully using the available host to device bandwidth.
More...
Very low Compute/Memcpy ratio
Compute 4 seconds
Memory Copy 51 seconds


NVIDIA Visual Profiler
File View Window Run Help
*01-parallel-k40.nvprof
112ms/iteration
21.1 s
21.125 s
111.563 ms
21.175 s
21.2 s
21.225 s
21.25 s
21.275 s
21.3 s
OpenACC
Driver API
Profiling Overhead
[0] Tesla K40m
Context 1 (CUDA)
MemCpy (HtoD)
MemCpy (DtoH)
Compute
58.4% main_55_gpu
41.3% main_66_gpu
Analysis
GPU Details
Console
Export PDF Report
Results
PCIe Copies
1. CUDA Application Analysis
2. Check Overall GPU Usage
The analysis results on the right potential problems in how your is taking advantage of the GPU's 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
iciency [3.962 s / 51.155 s = 0.077]
compute is low relative to the amount of time required for memcpy.
More...
rlap [0 ns / 3.962 s = 0%]
mcpy is being performed in parallel with compute is low.
More...
Low Kernel Concurrency [0 ns / 3.962 s = 0%]
The percentage of time when two kernels are being executed in parallel is low.
More...
Low Memcpy Throughput [3.875 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 parallel loop
```

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 host

A, Anew resident on accelerator

```
  ...
}
```

Evaluate Data Locality

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

#pragma acc parallel loop
  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 parallel loop
  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?

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 parallel loop
...

#pragma acc parallel loop
...
}
```

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`

Add Data Clauses

```
#pragma acc data copy(A) create(Anew)
while ( err > tol && iter < iter_max ) {
  err=0.0;
#pragma acc parallel loop
  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 parallel loop
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }
  iter++;
}
```

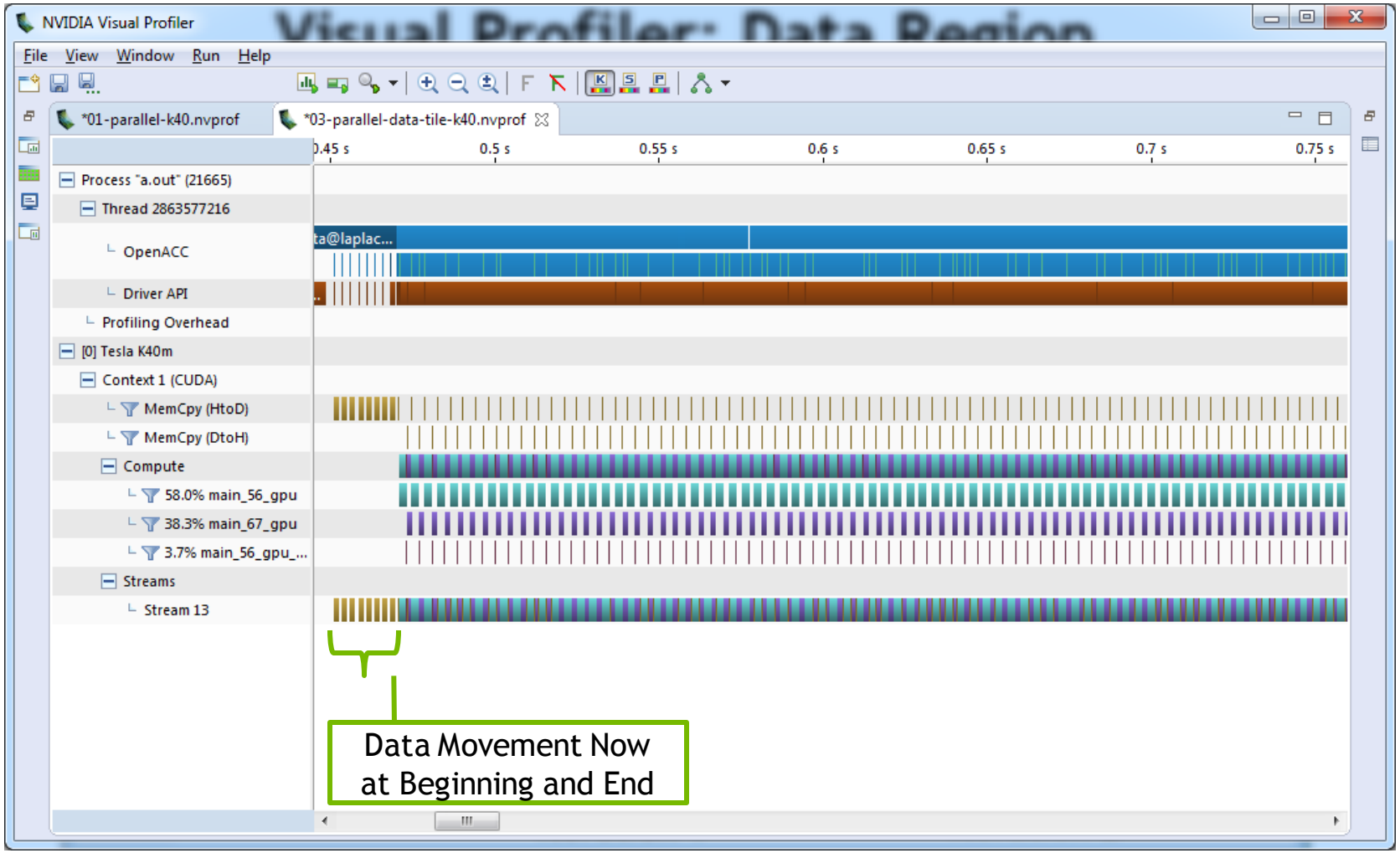
Copy A to/from the accelerator only when needed.

Create Anew as a device temporary.

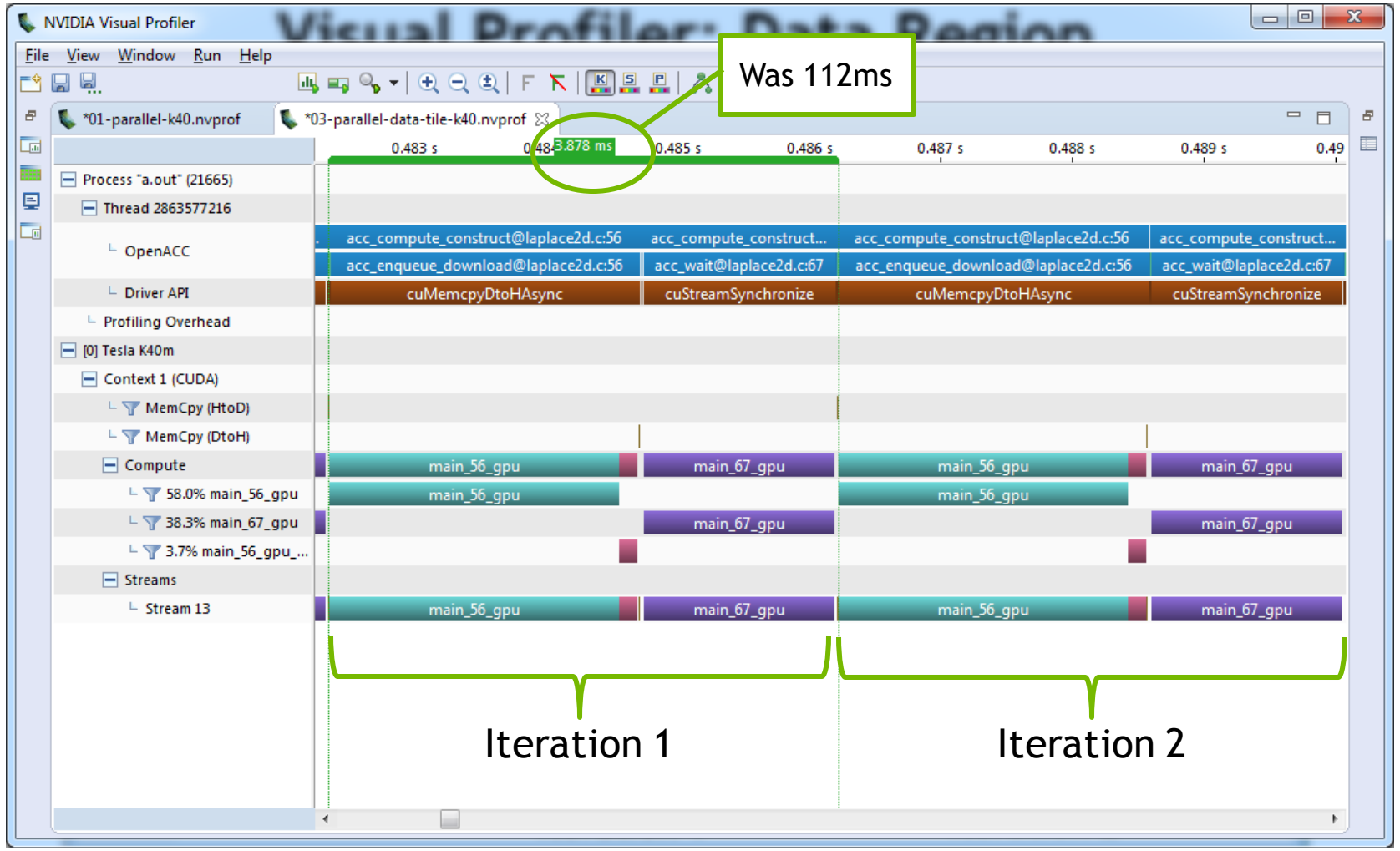
Rebuilding the code

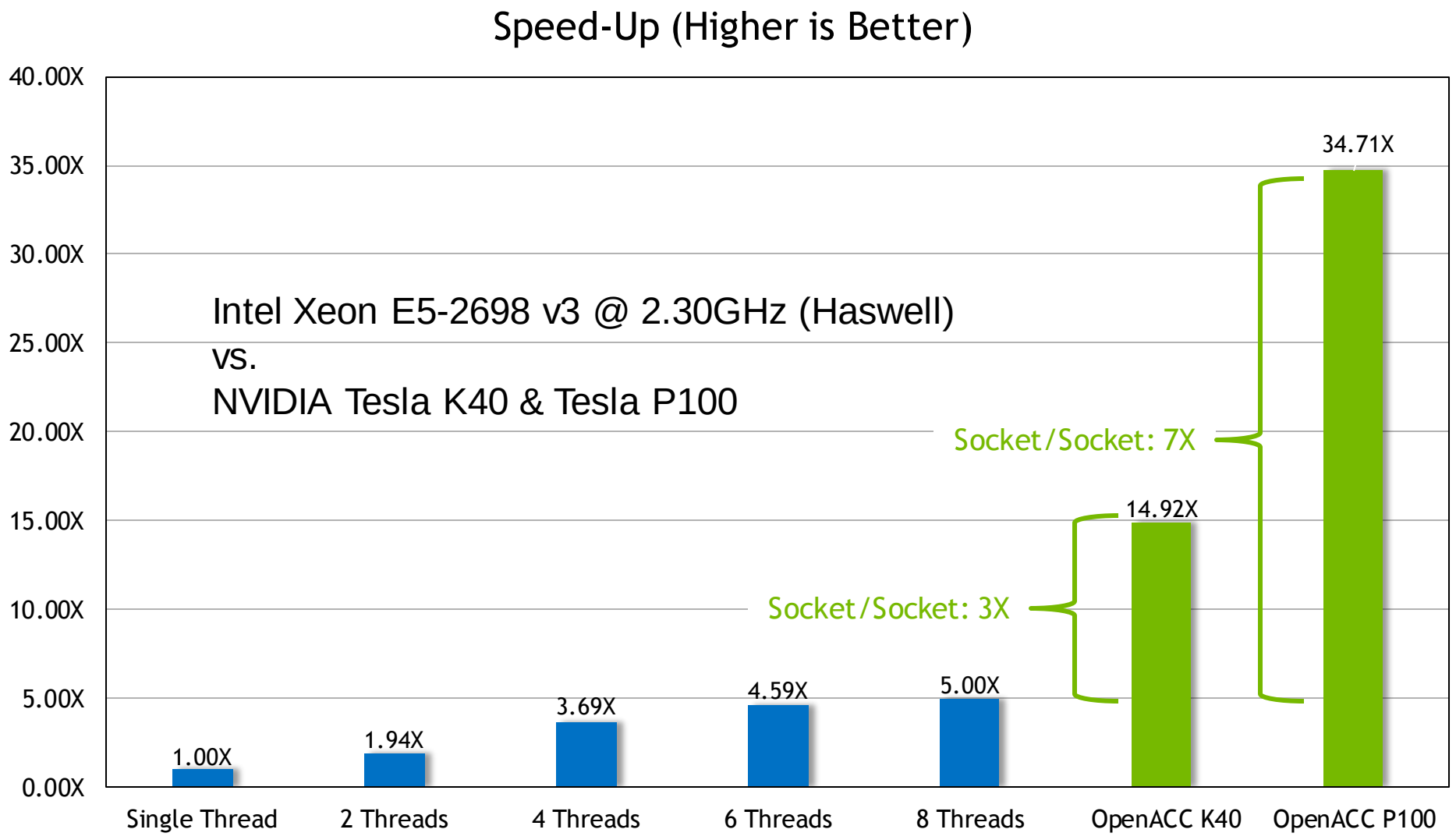
```
$ pgcc -fast -acc -ta=tesla -Minfo=all laplace2d.c
main:
     40, Loop not fused: function call before adjacent loop
         Generated vector sse code for the loop
     51, Generating copy(A[:][:])
         Generating create(Anew[:][:])
         Loop not vectorized/parallelized: potential early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



Visual Profiler: Data Region





Compiler: PGI 16.10

The loop Directive

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

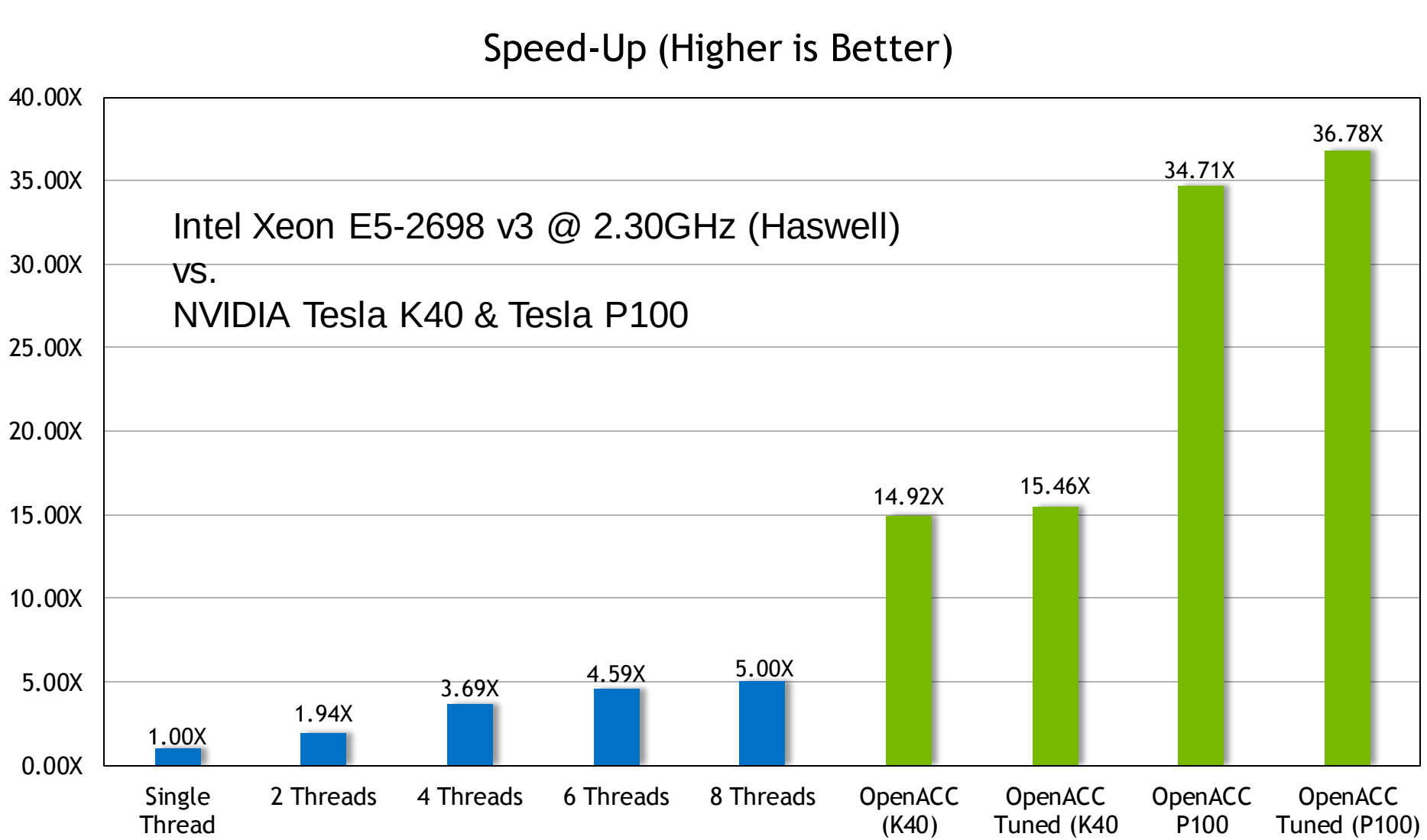
- `independent` – all iterations of the loop are independent
- `collapse(N)` – turn the next N loops into one, flattened loop
- `tile(N[,M,…])` - break the next 1 or more loops into *tiles* based on the provided dimensions.

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

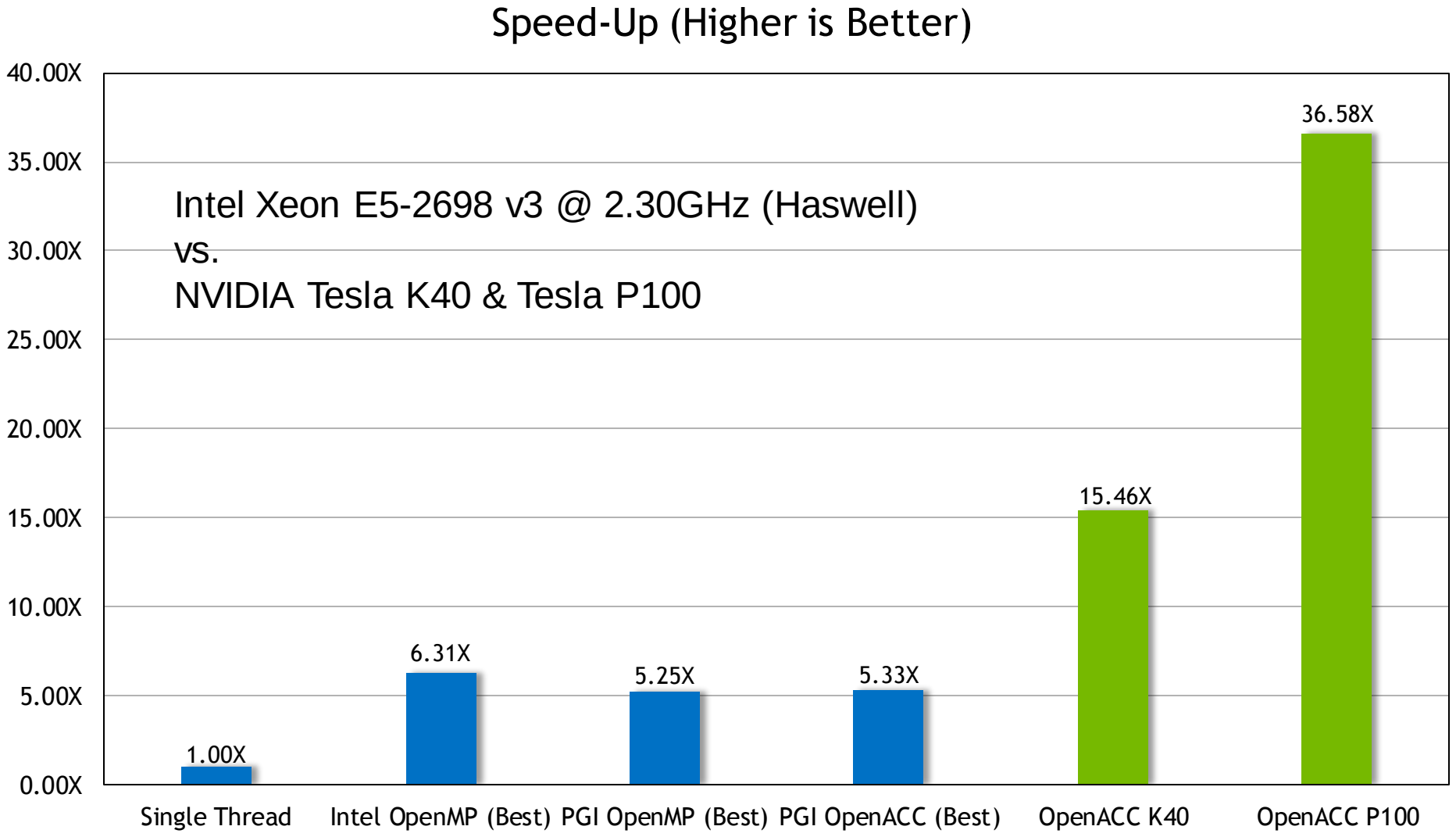
Optimize Loop Performance

```
#pragma acc data copy(A) create(Anew)
while ( err > tol && iter < iter_max ) {
  err=0.0;

#pragma acc parallel 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 parallel 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.



Compiler: PGI 16.10



Intel C Compiler 16, PGI 16.10 (OpenMP, K40, & P100), PGI 15.10 (multicore)

Next Lecture

Friday – OpenACC Pipelining