Running Jacobi using OpenACC on LBNL's Perlmutter system

Lecture 4

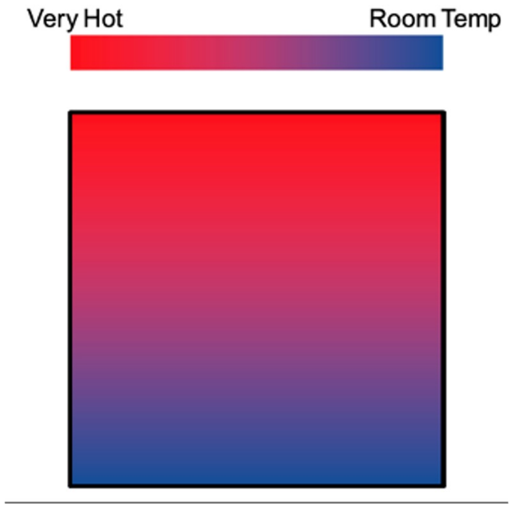
Sunita Chandrasekaran
Associate Professor, University of Delaware
PDC Summer School, Aug 2023
Materials also prepared by Dr. Felipe Cabarcas,
Postdoctoral Fellow, UDEL

Table of content

Table of content

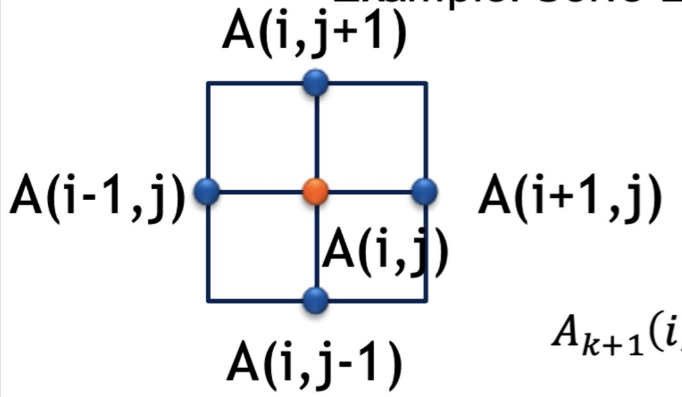
Laplace Heat Transfer

- A simple simulation of heat distributing across a metal plate
- Apply a consistent heat to the top of the plate
- Simulating the heat distribution across the plate



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: $\boldsymbol{\nabla^2 f(x, y) = 0}$



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

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

Iterate until converged

Iterate across matrix elements

Calculate new neighbors

```
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]);
    error = fmax( error, fabs(Anew[j][i] - A[j][i]));
  }
}
```

Compute max error for convergence

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

Swap input/output arrays

Profiling Sequential Code

Profile your code to obtain detailed information about how does the code runs:

- Total runtime
- runtime of routines
- Hardware counters

Identify portions that took longer to execute. These are the portions that you will want to parallelize.

```
LLVM
$ clang -Ofast -fopenmp -fno-inline -pg -o jacobi-serial jacobi.c
Jacobi relaxation Calculation: 4096 x 4096 mesh
    0, 0.250000
  100, 0.002397
  200, 0.001204
  300, 0.000804
  400, 0.000603
  500, 0.000483
  600, 0.000403
  700, 0.000345
  800, 0.000302
  900, 0.000269
 total: 25.557923 s
```

to use gprof add **-pg** to compile the application

Serial code with Nvidia nvc, performs similar to LLVM

```
NVC
$ nvc -O3  -o jacobi-serial jacobi.c
Jacobi relaxation Calculation: 4096 x 4096 mesh
    0, 0.250000
  100, 0.002397
  200, 0.001204
  300, 0.000804
  400, 0.000603
  500, 0.000483
  600, 0.000403
  700, 0.000345
  800, 0.000302
  900, 0.000269
 total: 23.364053 s
```

Table of content

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

Parallelize first loop next

```
#pragma acc parallel loop copy(A[:m*n],Anew[:m*n])
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]);
    error = fmax( error, fabs(Anew[j][i] - A[j][i]));
  }
}
```

```
#pragma acc parallel loop copy(A[:m*n],Anew[:m*n])
for( int j = 1; j < n-1; j++)
{
  for( int i = 1; i < m-1; i++ )
  {
    A[j][i] = Anew[j][i];
  }
}
```

Parallelize second loop

We didn't detail *how* to parallelize the loops, just *which* loops to parallelize.

Build and run the code

- Using Perlmutter
- Module load nvhpc/23.1
- Target which architecture you want to use to compile and execute the code; for example
- nvc -fast -acc=gpu -Minfo=all <source_code.c> -o <executable>
  - -acc=gpu: denotes that the target gpu
  - -fast: an optimization flag that you can add to your compilation command
  - -Minfo=all: gives you information about what parts of the code were accelerated
- Check for
  - “Generating NVIDIA GPU code”
  - Proof that your code generated GPU code

NVC

```
$ nvc -fast -acc=gpu -Minfo=all -o jacobi-acc-loop jacobi.c
initialize:
     41, Generated vector simd code for the loop
calcNext:
     49, Generating copy(A[:n*m]) [if not already present]
         Generating NVIDIA GPU code
         51, #pragma acc loop gang /* blockIdx.x */
             Generating implicit reduction(max:error)
         53, #pragma acc loop vector(128) /* threadIdx.x */
     49, Generating implicit copy(error) [if not already present]
         Generating copy(Anew[:n*m]) [if not already present]
     53, Loop is parallelizable
swap:
     64, Generating copy(A[:n*m],Anew[:n*m]) [if not already present]
         Generating NVIDIA GPU code
         66, #pragma acc loop gang /* blockIdx.x */
         68, #pragma acc loop vector(128) /* threadIdx.x */
     68, Loop is parallelizable
main:
    111, initialize inlined, size=10 (inline) file jacobi.c (37)
          41, Loop not fused: function call before adjacent loop
              Generated vector simd code for the loop
    119, Loop not vectorized/parallelized: potential early exits
    134, deallocate inlined, size=2 (inline) file jacobi.c (76)
```

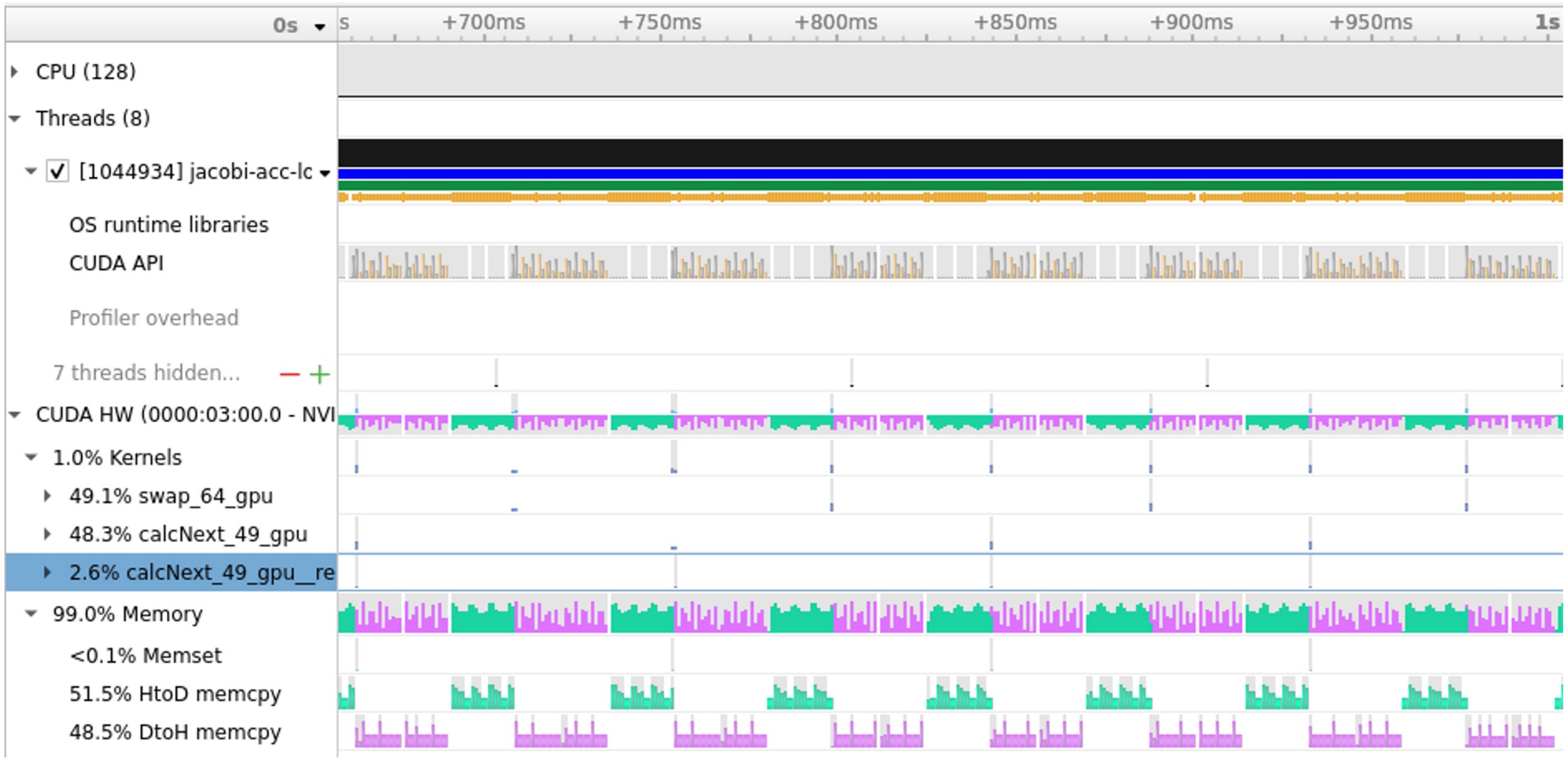
```
Jacobi relaxation
Calculation: 4096 x 4096
mesh
    0, 0.250000
  100, 0.002397
  200, 0.001204
  300, 0.000804
  400, 0.000603
  500, 0.000483
  600, 0.000403
  700, 0.000345
  800, 0.000302
  900, 0.000269
 total: 84.040213 s
```

Accelerated code using parallel and no data clauses takes 84.04 on GPUs
about 4 times slower than serial

Using NSight System

- nsys profile --gpu-metrics-device=all -o prof-${FILE}-nvc ./${FILE}
- FILE=<your file name>
- Download https://developer.nvidia.com/nsight-systems
- View your files

- Note the memory usage bars
- Not an optimized code
- Data being copied to and from GPU to and from the CPU all the time – hurts performance



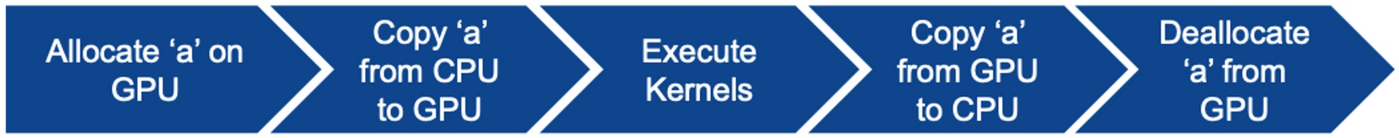
```
$ nvidia-smi
Fri Jun 23 06:46:07 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100-SXM...  On   | 00000000:03:00.0 Off |                    0 |
| N/A   26C    P0    50W / 400W |      0MiB / 40960MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A100-SXM...  On   | 00000000:41:00.0 Off |                    0 |
| N/A   25C    P0    49W / 400W |      0MiB / 40960MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA A100-SXM...  On   | 00000000:82:00.0 Off |                    0 |
| N/A   26C    P0    53W / 400W |      0MiB / 40960MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   3  NVIDIA A100-SXM...  On   | 00000000:C1:00.0 Off |                    0 |
| N/A   25C    P0    52W / 400W |      0MiB / 40960MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+


+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

What was missing in the previous code?

Do data clauses to copy data from CPUs to GPUs

Let’s see how to add data clauses to our code



OpenACC Data Clauses

- **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.
- **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.
- **create(list)** - Allocates memory on GPU but does not copy.
- **delete(list)** - Deallocate memory on the GPU without copying. (Unstructured Only)
- **present(list)** - Data is already present on GPU from another containing data region.

```
#pragma acc data copyout(a[0:N]), copyin(b[0:N])
{
 #pragma acc parallel loop present(a,b)
 for (int i=0; i<N; i++)
     a[i] = b[i] + 1;
}
```

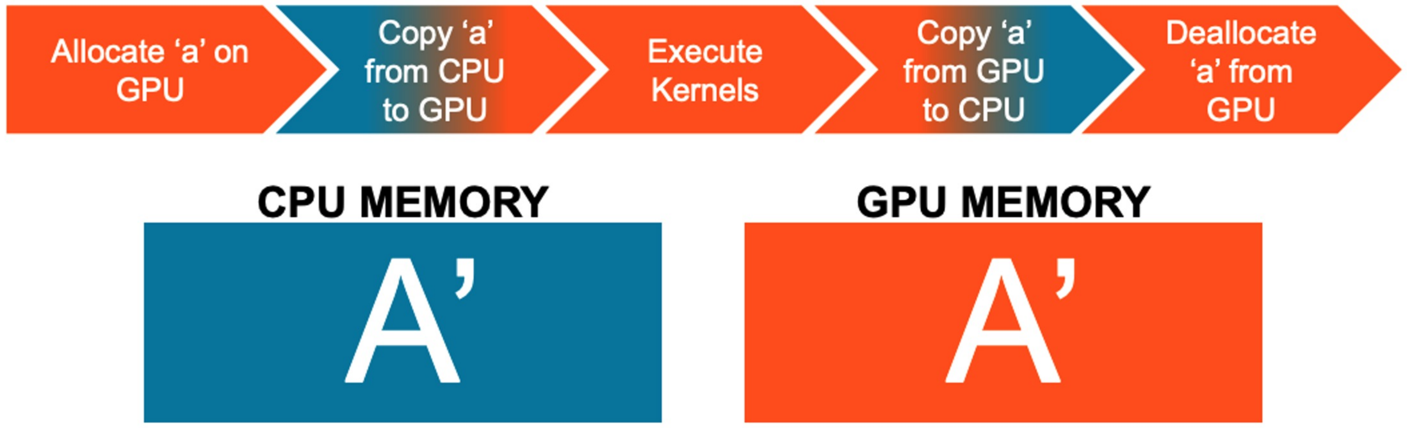
```
const int N=100;
#pragma acc data copy(a[0:N])
{
 #pragma acc parallel loop present(a)
 for (int i=0; i<N; i++)
     a[i] = a[i] + 1;
}
```

```
#pragma acc data copyout(a[0:N]), create(b[0:N])
{
 #pragma acc parallel loop
 for (int i=0; i<N; i++)
     b[i] = i * 2.0;
```

Table of content

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.

Principal use: For many important data structures in your code, this is a logical default to input, modify and return the data.



ARRAY SHAPING

- Sometimes the compiler needs help understanding the *shape* of an array
- The first number is the start index of the array
- In C/C++, the second number is how much data is to be transferred
- In Fortran, the second number is the ending index

```
copy(array[starting_index:length])
```

C/C++

```
copy(array(starting_index:ending_index))
```

Fortran

BASIC DATA MANAGEMENT

Multi-dimensional Array shaping

```
copy(array[0:N][0:M])
```

C/C++

```
copy(array(1:N, 1:M))
```

Fortran

```
#pragma acc data copy(A[:n*m]) create(Anew[:n*m])
while ( error > tol && iter < iter_max )
{
   error = 0.0;
```

Create data on the GPUs

```
#pragma acc parallel loop reduction(max:error) copy(A[:m*n],Anew[:m*n])
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]);
     error = fmax( error, fabs(Anew[j][i] - A[j][i]));
   }
}
```

Parallelize and max *reduction*

```
#pragma acc parallel loop copy(A[:m*n],Anew[:m*n])
for( int j = 1; j < n-1; j++)
{
  for( int i = 1; i < m-1; i++ )
  {
    A[j][i] = Anew[j][i];
  }
}
```

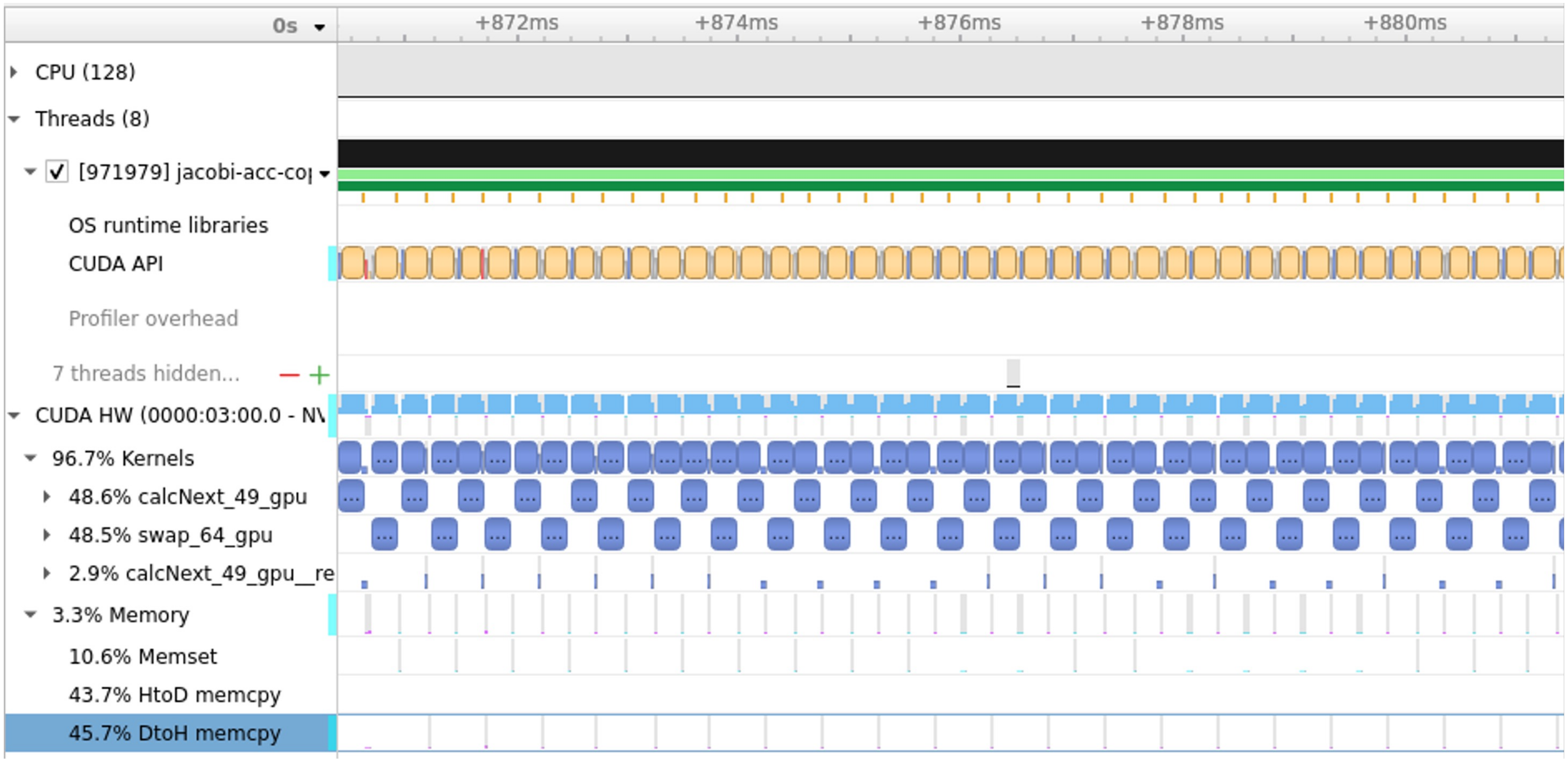
Parallelize second loop

```
$ nvc -fast -acc=gpu -Minfo=all -o jacobi-acc-copy jacobi.c
initialize:
     41, Generated vector simd code for the loop
calcNext:
     49, Generating copy(A[:n*m]) [if not already present]
         Generating NVIDIA GPU code
         51, #pragma acc loop gang /* blockIdx.x */
             Generating reduction(max:error)
         53, #pragma acc loop vector(128) /* threadIdx.x */
     49, Generating implicit copy(error) [if not already present]
         Generating copy(Anew[:n*m]) [if not already present]
     53, Loop is parallelizable
swap:
     64, Generating copy(A[:n*m],Anew[:n*m]) [if not already present]
         Generating NVIDIA GPU code
         66, #pragma acc loop gang /* blockIdx.x */
         68, #pragma acc loop vector(128) /* threadIdx.x */
     68, Loop is parallelizable
main:
    111, initialize inlined, size=10 (inline) file jacobi.c (37)
          41, Loop not fused: function call before adjacent loop
              Generated vector simd code for the loop
    119, Generating create(Anew[:m*n]) [if not already present]
         Generating copy(A[:m*n]) [if not already present]
         Loop not vectorized/parallelized: potential early exits
    134, deallocate inlined, size=2 (inline) file jacobi.c (76)
```

```
Jacobi relaxation
Calculation: 4096 x 4096
mesh
    0, 0.250000
  100, 0.002397
  200, 0.001204
  300, 0.000804
  400, 0.000603
  500, 0.000483
  600, 0.000403
  700, 0.000345
  800, 0.000302
  900, 0.000269
 total: 1.589625 s
```

Accelerated code using parallel and data clauses taking 1.58s on GPUs

Using Nsight System



We reduced data movement to/from GPU to host

Results

Serial Code takes **23.364053s** on GPUs

Accelerated code using parallel and NO data clauses on main loop takes **84.040213s** on GPUs

Accelerated code using parallel and data clauses take **1.589625s** on GPUs

What to avoid?

```
#pragma acc data copy(A[:n*m]) create(Anew[:n*m])
while ( error > tol && iter < iter_max )
{
  error = 0.0;

#pragma acc parallel loop reduction(max:error) copy(A[0:n*m]) copy(Anew[0:n*m])

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]);
    error = fmax( error, fabs(Anew[j][i] - A[j][i]));
  }
}

#pragma acc parallel loop copy(Anew[0:n*m]) copy(A[0:n*m])

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

Copying **to and from** GPU for every iteration

Add data copy to the main loop; this would avoid the copy on every iteration

Table of content

```
#pragma acc data copy(A[:n*m]) create(Anew[:n*m])
while ( error > tol && iter < iter_max )
{
   error = 0.0;
```

Create data on the GPUs

```
#pragma acc parallel loop reduction(max:error) copy(A[:m*n],Anew[:m*n])
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]);
     error = fmax( error, fabs(Anew[j][i] - A[j][i]));
   }
}
```

Parallelize and max *reduction*

```
#pragma acc parallel loop copy(A[:m*n],Anew[:m*n])
for( int j = 1; j < n-1; j++)
{
  for( int i = 1; i < m-1; i++ )
  {
    A[j][i] = Anew[j][i];
  }
}
```

Parallelize second loop

Use multicore flag

Set cores

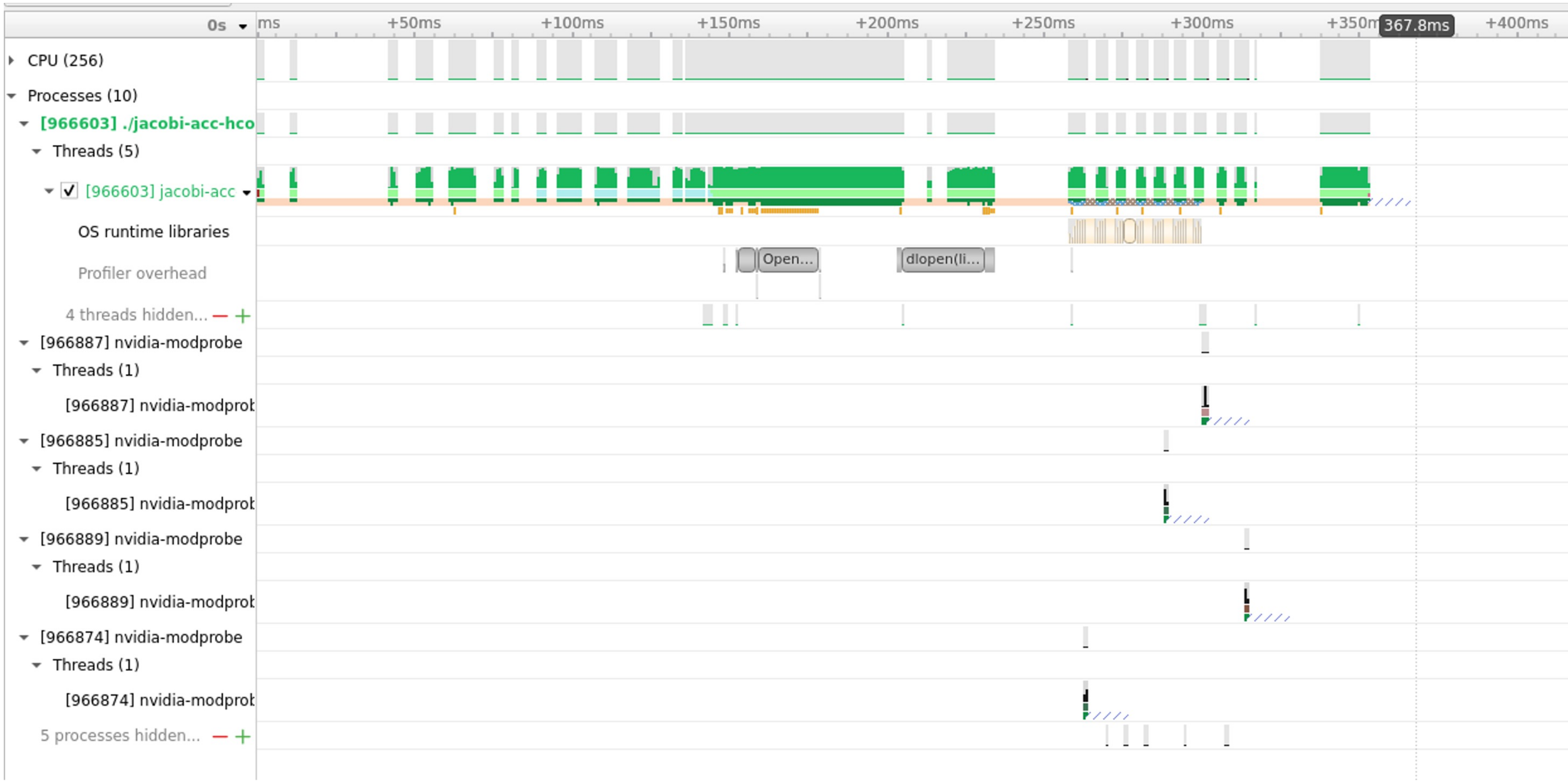
```
$ nvc -fast -acc=multicore -Minfo=all -o jacobi-acc-hcopy jacobi.c
initialize:
     41, Generated vector simd code for the loop
calcNext:
     49, Generating Multicore code
         51, #pragma acc loop gang
     51, Generating reduction(max:error)
     53, Loop is parallelizable
         Generated vector simd code for the loop containing
reductions
swap:
     64, Generating Multicore code
         66, #pragma acc loop gang
     68, Loop is parallelizable
         Memory copy idiom, loop replaced by call to __c_mcopy8
main:
    111, initialize inlined, size=10 (inline) file jacobi.c (37)
          41, Loop not fused: function call before adjacent loop
              Generated vector simd code for the loop
    119, Loop not vectorized/parallelized: potential early exits
    134, deallocate inlined, size=2 (inline) file jacobi.c (76)
```

```
$ export ACC_NUM_CORES=64
$ ./jacobi-acc-hcopy
Jacobi relaxation
Calculation: 4096 x 4096
mesh
    0, 0.250000
  100, 0.002397
  200, 0.001204
  300, 0.000804
  400, 0.000603
  500, 0.000483
  600, 0.000403
  700, 0.000345
  800, 0.000302
  900, 0.000269
 total: 0.852060 s
```

Parallelized code using parallel construct took 0.852s on 64 core CPU

AMD EPYC 7763 64-Core Processor

Using Nsight System



Results

- Serial Code takes **23.364053s** on a single core CPU
- Accelerated code using parallel and no data clauses on main loop takes **84.040213s** on NVIDIA A100 GPUs
- Accelerated code using parallel and data clauses take **1.589625s** on NVIDIA A100 GPUs
- Parallelized code using parallel construct took **0.852060s** on 64 core AMD EPYC multicore CPUs

Is something looking odd?

Increasing the size of the mesh size

- Was the GPU fed with enough compute to do?
- Try nvidia-smi to determine GPU occupancy

4096 x 4096 GRID

Accelerated code using parallel and data clauses take **1.589625s** on GPUs

Parallelized code using parallel construct took **0.852060s** on 64 core multicore CPUs

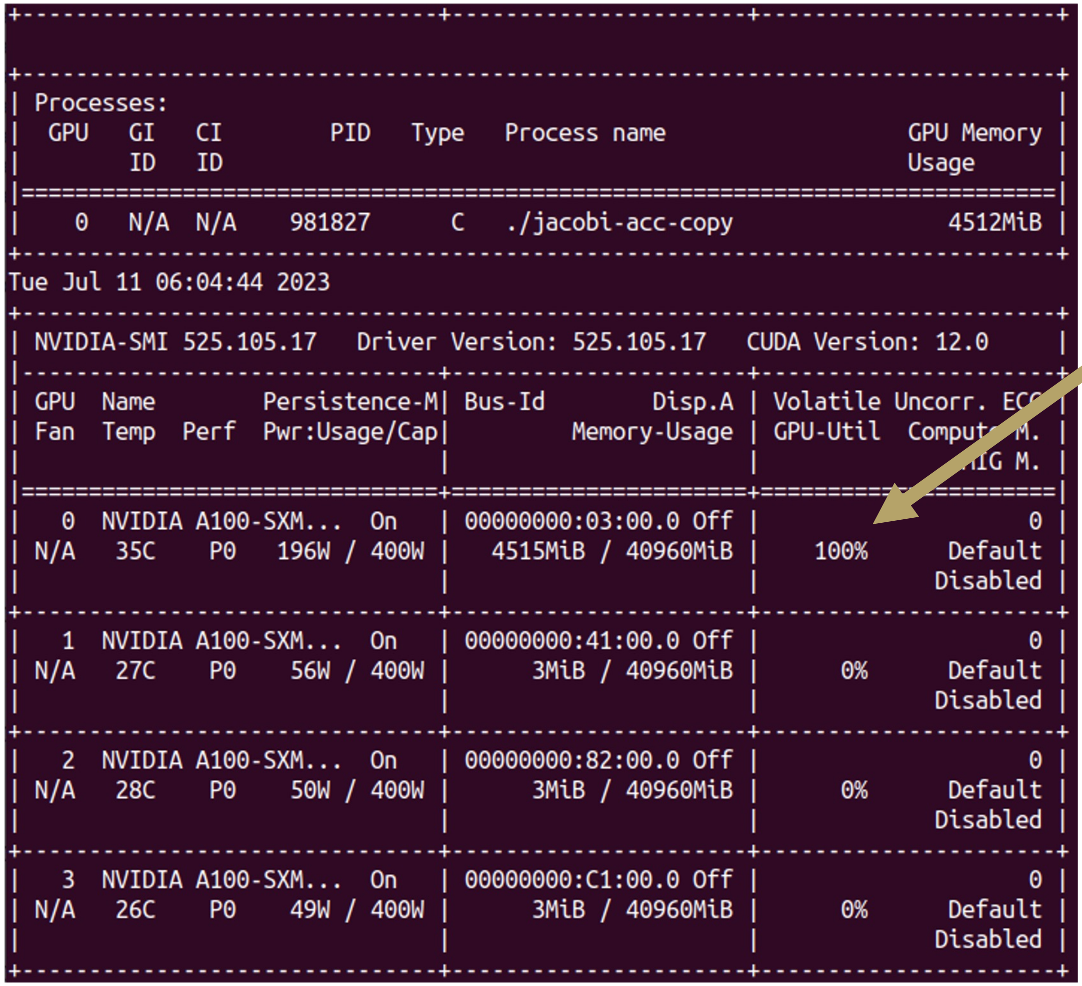
16384 x 16384 GRID

Accelerated code using parallel and data clauses take **9.582822 s** on GPUs

Parallelized code using parallel construct took **356.332373 s** on 64 core multicore CPUs

Increasing the size of the mesh size

For grid size 16384 x 16384



```
+-----------------------------------------------------------------------------+
+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    981827      C   ./jacobi-acc-copy                4512MiB |
+-----------------------------------------------------------------------------+
Tue Jul 11 06:04:44 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100-SXM...  On   | 00000000:03:00.0 Off |                    0 |
| N/A   35C    P0   196W / 400W |   4515MiB / 40960MiB |    100%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A100-SXM...  On   | 00000000:41:00.0 Off |                    0 |
| N/A   27C    P0    56W / 400W |      3MiB / 40960MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA A100-SXM...  On   | 00000000:82:00.0 Off |                    0 |
| N/A   28C    P0    50W / 400W |      3MiB / 40960MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   3  NVIDIA A100-SXM...  On   | 00000000:C1:00.0 Off |                    0 |
| N/A   26C    P0    49W / 400W |      3MiB / 40960MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
```