Lecture 20 – NVHPC Hands-on

Prof. Brendan Kochunas

NERS/ENGR 570 - Methods and Practice of Scientific Computing (F22)

COLLEGE OF ENGINEERING
NUCLEAR ENGINEERING & RADIOLOGICAL SCIENCES
UNIVERSITY OF MICHIGAN

Outline

- Announcements
- (Labs 9 and 10) Simulated Annealing Problem with OpenMP
- (HW3) SpMV with CUDA

Learning Objectives: By the end of Today's Lecture you should be able to

- (*Skill*) write a better OpenMP implementation of Simulated Annealing
- (*Skill*) use the OpenMP TARGET directive for GPU offload
- (*Skill*) write simple CUDA kernels (maybe)

Simulated Annealing

Instructions

- Pair/Group up (so we don't have a bunch simultaneous jobs on GL)
  - Login to (Log into? Log in to?) Great Lakes
  - Copy down laplace_serial.c into new directory for today's lecture

- Establish host Scaling
  - Login to (Log into? Log in to?)
  - `$ srun -A ners570f22_class --pty --partition=standard --time=00:60:00 --nodes=1 --ntasks-per-node=1 --cpus-per-task=36 --mem=100gb  /bin/bash`

- Work on GPU implementations
  - Start a new interactive job on GL
  - `$ srun -A ners570f22_class --pty --partition=gpu --time=00:60:00 --gpus=1  /bin/bash`

My Host Speedup

Number of Threads	gcc -Ofast -march=native -fopenmp (Time s)	nvc -fast –mp (Time s)	GCC Speedup	NVC Speedup
1	5.101	5.556	1.0	1.0
2	2.325	2.591	2.2	2.1
4	1.181	1.277	4.3	4.4
8	0.710	0.776	7.2	7.2
16	0.382	0.370	13.4	15.0
24	0.155	0.192	32.9	28.9
32	0.129	0.163	39.6	34.0
36	0.125	0.159	40.9	35.1

My OpenMP Solution

```
59      // do until error is minimal or until max steps
60      while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {
61
62          dt = 0.0; // reset largest temperature change
63  #pragma omp parallel default(shared) private(i,j)
64  {
65
66          // main calculation: average my four neighbors, compute local max dt
67          #pragma omp for
68          for(i = 1; i <= ROWS; i++) {
69              for(j = 1; j <= COLUMNS; j++) {
70                  Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] + Temperature_last[i-1][j] +
71                                              Temperature_last[i][j+1] + Temperature_last[i][j-1]);
72              }
73          }
74
75
76          // copy grid to old grid for next iteration
77          #pragma omp for reduction(max:dt)
78          for(i = 1; i <= ROWS; i++){
79              for(j = 1; j <= COLUMNS; j++){
80                dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
81                Temperature_last[i][j] = Temperature[i][j];
82              }
83          }
84  }
85          // periodically print test values
86          if((iteration % 100) == 0) {
87              track_progress(iteration);
88          }
89
90          iteration++;
91      }
```

Now we do GPUs

```
$ srun -A ners570f22_class --pty --partition=gpu --
time=00:60:00 --gpus=1  /bin/bash
```

First GPU Implementation (Threaded)

```
59      // do until error is minimal or until max steps
60      while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {
61
62          dt = 0.0; // reset largest temperature change
63
64  #pragma omp target map(Temperature_last) map(Temperature) map(dt)
65  {
66          // main calculation: average my four neighbors, compute local max dt
67          #pragma omp parallel for
68          for(i = 1; i <= ROWS; i++) {
69              for(j = 1; j <= COLUMNS; j++) {
70                  Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] + Temperature_last[i-1][j] +
71                                              Temperature_last[i][j+1] + Temperature_last[i][j-1]);
72              }
73          }
74
75          // copy grid to old grid for next iteration
76          #pragma omp parallel for reduction(max:dt)
77          for(i = 1; i <= ROWS; i++){
78              for(j = 1; j <= COLUMNS; j++){
79                dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
80                Temperature_last[i][j] = Temperature[i][j];
81              }
82          }
83  }
84          // periodically print test values
85          if((iteration % 100) == 0) {
86              track_progress(iteration);
87          }
88
89          iteration++;
90      }
91
```

GPU With Teams

```
59 #pragma omp target data map(Temperature) map(Temperature_last)
60     // do until error is minimal or until max steps
61     while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {
62
63         // main calculation: average my four neighbors, compute local max dt
64 #pragma omp target teams distribute parallel for
65         for( i = 1; i <= ROWS; i++) {
66             for(j = 1; j <= COLUMNS; j++) {
67                 Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] + Temperature_last[i-1][j] +
68                                             Temperature_last[i][j+1] + Temperature_last[i][j-1]);
69             }
70         }
71
72         dt = 0.0; // reset largest temperature change
73
74         // copy grid to old grid for next iteration
75 #pragma omp target teams distribute parallel for reduction(max:dt)
76         for(i = 1; i <= ROWS; i++){
77             for(j = 1; j <= COLUMNS; j++){
78               dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
79               Temperature_last[i][j] = Temperature[i][j];
80             }
81         }
82
83         // periodically print test values
84         if((iteration % 100) == 0) {
85             track_progress(iteration);
86         }
87
88         iteration++;
89     }
```

Explicitly maps arrays
For entire while loop

Spawns thread teams
And distributes iterations to those teams

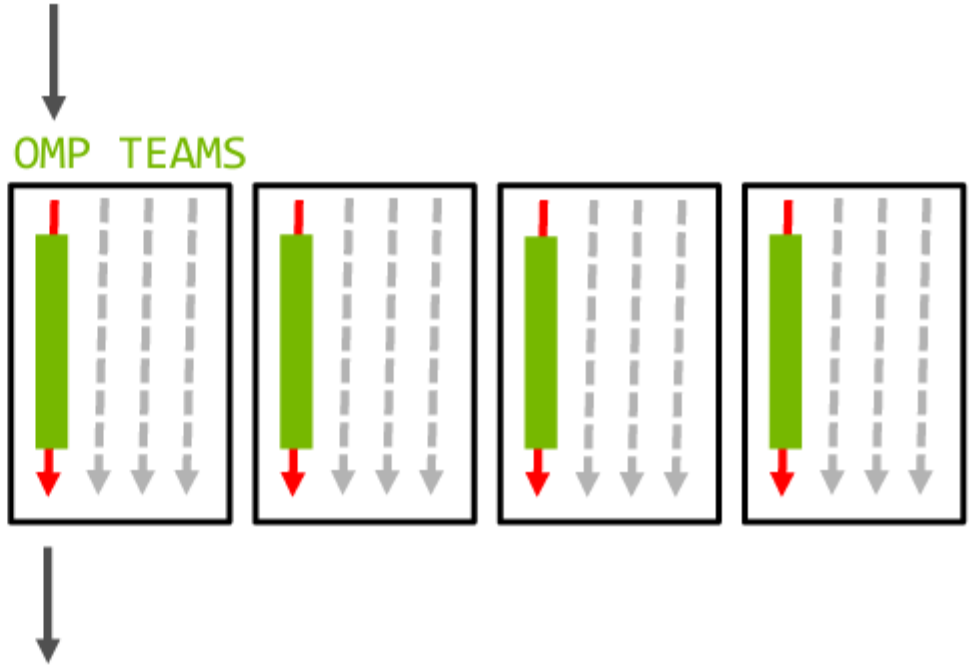
Worksharing within teams

The Teams Directive

TEAMS Directive

To better utilize the GPU resources, use many thread teams via the TEAMS directive.

- Spawns 1 or more thread teams with the same number of threads
- Execution continues on the master threads of each team (redundantly)
- No synchronization between teams

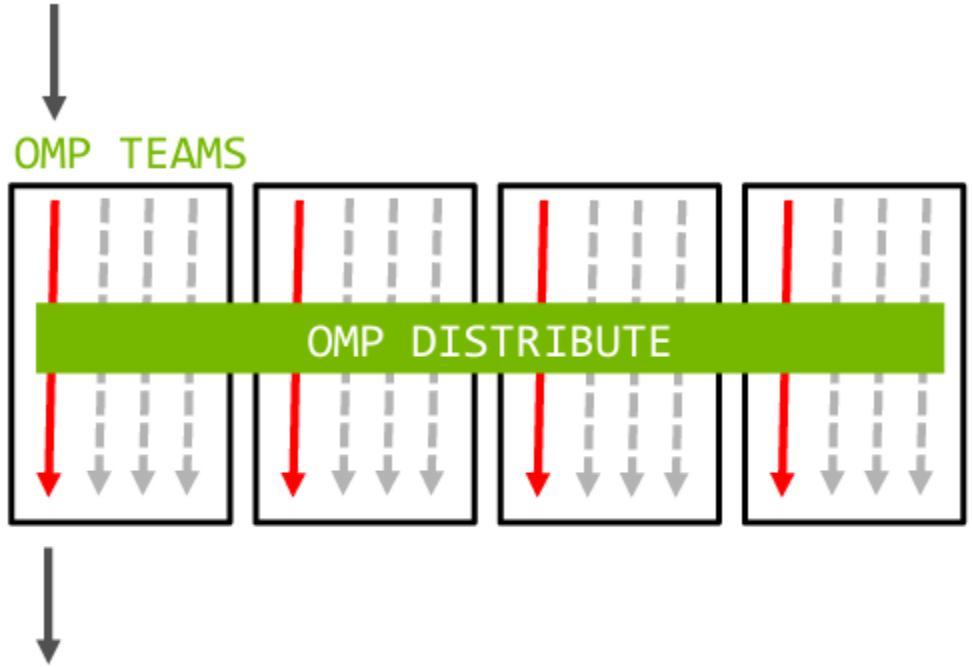


The Distribute Directive

DISTRIBUTE Directive

Distributes the iterations of the next loop to the master threads of the teams.

- Iterations are distributed statically.
- There's no guarantees about the order teams will execute.
- No guarantee that all teams will execute simultaneously
- Does not generate parallelism/worksharing within the thread teams.



GPU Teams with Splitting loops

```
59 #pragma omp target data map(Temperature) map(Temperature_last)
60     // do until error is minimal or until max steps
61     while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {
62
63         // main calculation: average my four neighbors, compute local max dt
64 #pragma omp target teams distribute
65         for(i = 1; i <= ROWS; i++) {
66 #pragma omp parallel for
67             for(j = 1; j <= COLUMNS; j++) {
68                 Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] + Temperature_last[i-1][j] +
69                                             Temperature_last[i][j+1] + Temperature_last[i][j-1]);
70             }
71         }
72
73         dt = 0.0; // reset largest temperature change
74
75         // copy grid to old grid for next iteration
76 #pragma omp target teams distribute reduction(max:dt)
77         for(i = 1; i <= ROWS; i++){
78 #pragma omp parallel for reduction(max:dt)
79             for(j = 1; j <= COLUMNS; j++){
80               dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
81               Temperature_last[i][j] = Temperature[i][j];
82             }
83         }
84
85         // periodically print test values
86         if((iteration % 100) == 0) {
87             track_progress(iteration);
88         }
89
90         iteration++;
91     }
```

Distribute “i” loops over teams

Workshare “j” loops over threads

GPU Teams with loop collapsing

```
59 #pragma omp target data map(Temperature) map(Temperature_last)
60     // do until error is minimal or until max steps
61     while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {
62
63         // main calculation: average my four neighbors, compute local max dt
64 #pragma omp target teams distribute parallel for collapse(2)
65         for(i = 1; i <= ROWS; i++) {
66             for(j = 1; j <= COLUMNS; j++) {
67                 Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] + Temperature_last[i-1][j] +
68                                             Temperature_last[i][j+1] + Temperature_last[i][j-1]);
69             }
70         }
71
72         dt = 0.0; // reset largest temperature change
73
74         // copy grid to old grid for next iteration
75 #pragma omp target teams distribute parallel for reduction(max:dt) collapse(2)
76         for(i = 1; i <= ROWS; i++){
77             for(j = 1; j <= COLUMNS; j++){
78               dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
79               Temperature_last[i][j] = Temperature[i][j];
80             }
81         }
82
83         // periodically print test values
84         if((iteration % 100) == 0) {
85             track_progress(iteration);
86         }
87
88         iteration++;
89     }
```

Transforms 2-loops
Into 1-loop for
Increased parallelism

GPU Teams with loop Splitting and Scheduling

```
59 #pragma omp target data map(Temperature) map(Temperature_last)
60     // do until error is minimal or until max steps
61     while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {
62
63         // main calculation: average my four neighbors, compute local max dt
64 #pragma omp target teams distribute
65         for(i = 1; i <= ROWS; i++) {
66 #pragma omp parallel for schedule(static,1)
67             for(j = 1; j <= COLUMNS; j++) {
68                 Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] + Temperature_last[i-1][j] +
69                                             Temperature_last[i][j+1] + Temperature_last[i][j-1]);
70             }
71         }
72
73         dt = 0.0; // reset largest temperature change
74
75         // copy grid to old grid for next iteration
76 #pragma omp target teams distribute reduction(max:dt)
77         for(i = 1; i <= ROWS; i++){
78 #pragma omp parallel for reduction(max:dt) schedule(static,1)
79             for(j = 1; j <= COLUMNS; j++){
80               dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
81               Temperature_last[i][j] = Temperature[i][j];
82             }
83         }
84
85         // periodically print test values
86         if((iteration % 100) == 0) {
87             track_progress(iteration);
88         }
89
90         iteration++;
91     }
```

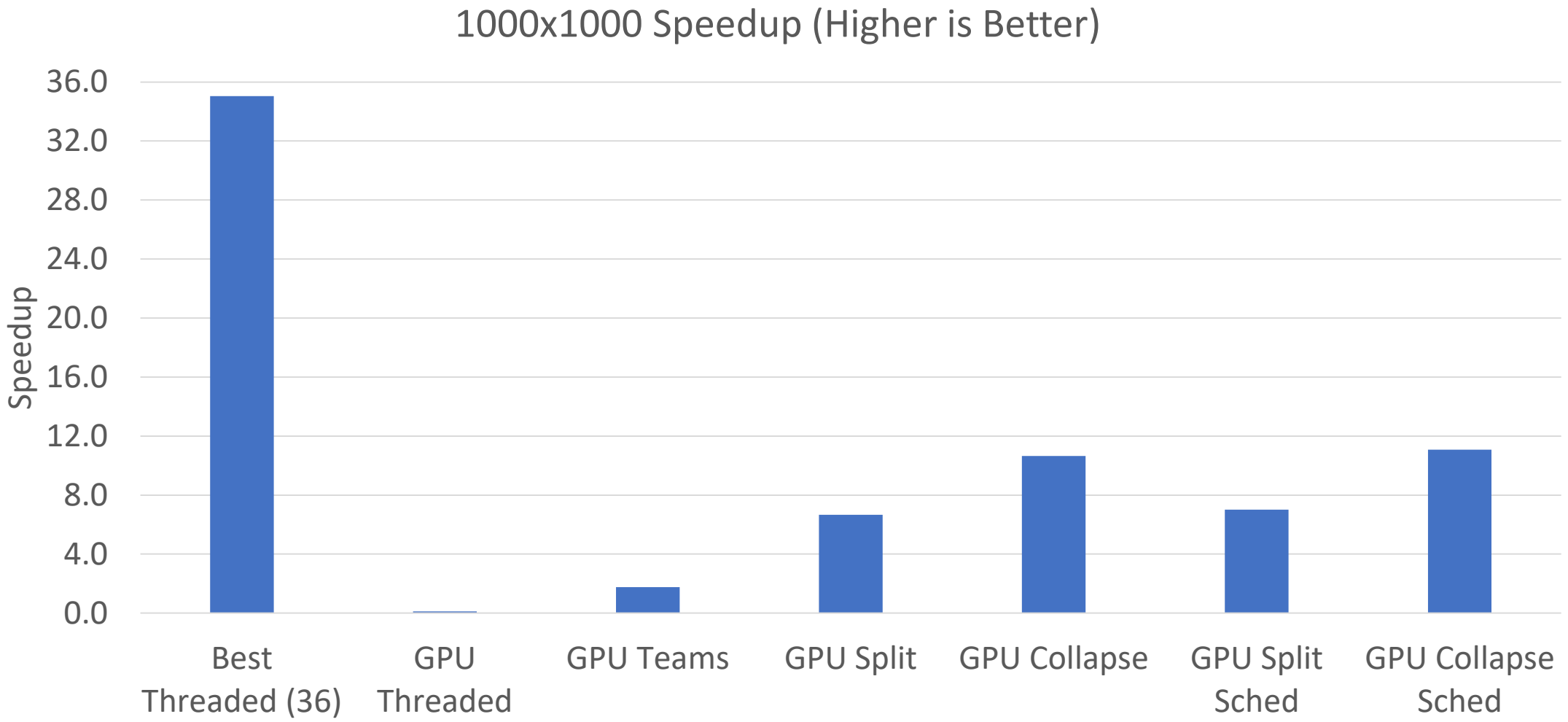
Align loop iterations with threads

GPU Teams with loop collapsing and Scheduling

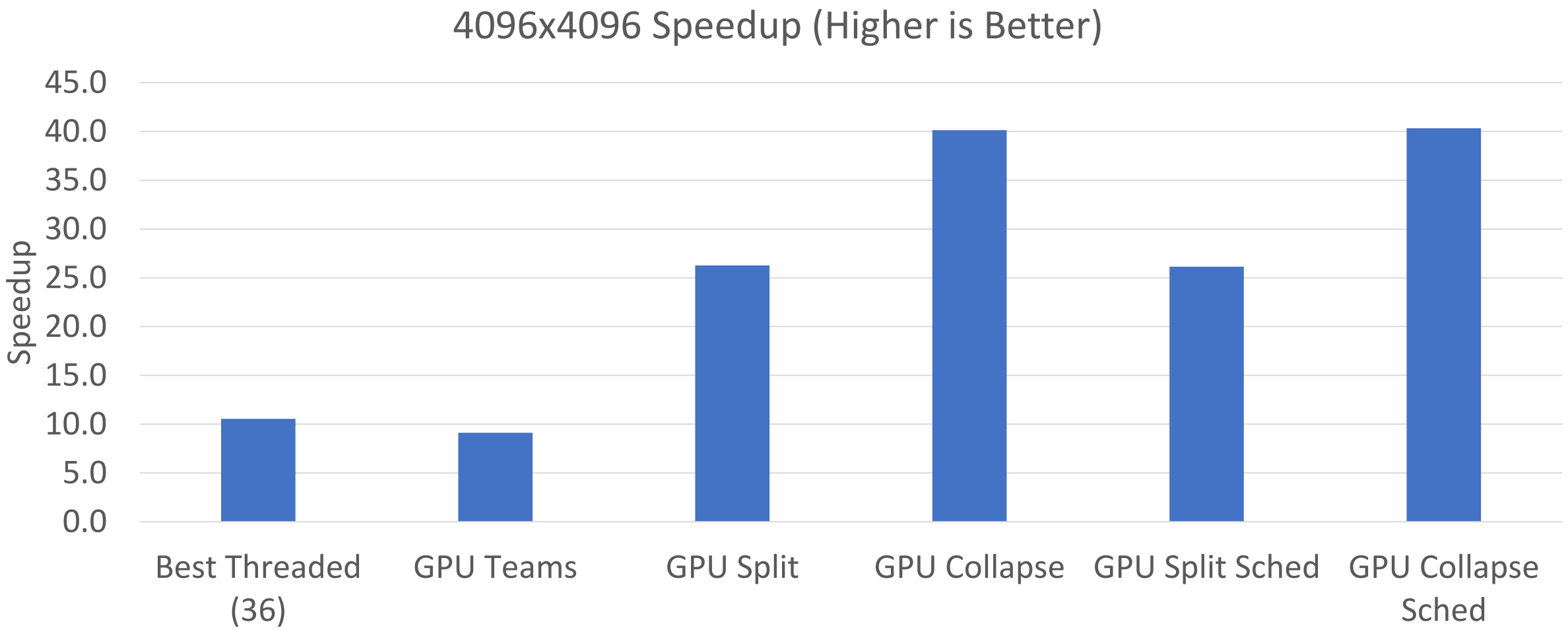
```
59 #pragma omp target data map(tofrom:Temperature) map(Temperature_last)
60     // do until error is minimal or until max steps
61     while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {
62
63         // main calculation: average my four neighbors, compute local max dt
64 #pragma omp target teams distribute parallel for collapse(2) schedule(static,1)
65         for(i = 1; i <= ROWS; i++) {
66             for(j = 1; j <= COLUMNS; j++) {
67                 Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] + Temperature_last[i-1][j] +
68                                             Temperature_last[i][j+1] + Temperature_last[i][j-1]);
69             }
70         }
71
72         dt = 0.0; // reset largest temperature change
73
74         // copy grid to old grid for next iteration
75 #pragma omp target teams distribute parallel for reduction(max:dt) collapse(2) schedule(static,1)
76         for(i = 1; i <= ROWS; i++){
77             for(j = 1; j <= COLUMNS; j++){
78               dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
79               Temperature_last[i][j] = Temperature[i][j];
80             }
81         }
82
83         // periodically print test values
84         if((iteration % 100) == 0) {
85             track_progress(iteration);
86         }
87
88         iteration++;
89     }
90
```

Align loop iterations with threads

Summary of Speedup



What if we have a Larger Problem?



SpMV ELLPACK Format