Hybrid Programming

John Urbanic
Parallel Computing Specialist
Pittsburgh Supercomputing Center

Assuming you know basic MPI

- This is a rare group that can discuss this topic meaningfully.
- I have mentioned MPI 3.0's "improvements" to its hybrid capabilities. These are primarily tying up loose ends and formally specifying that things work as you would expect, and as they largely do. Your MPI 1/2 knowledge will be more than sufficient here.

Hybrid OpenACC Programming (Fast & Wrong)

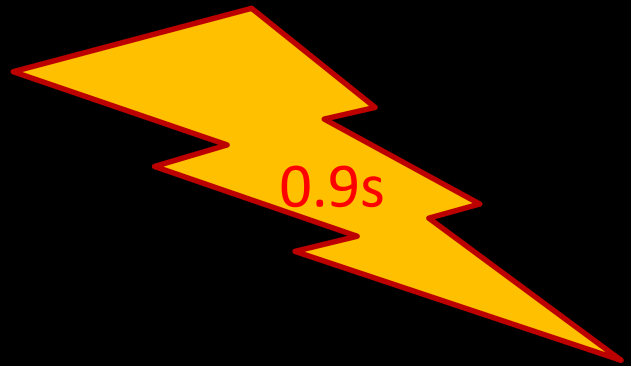
```
#pragma acc data copy(Temperature_last), create(Temperature)
while ( dt_global > MAX_TEMP_ERROR && iteration <= max_iterations ) {

    #pragma acc kernels
    for(i = 1; i <= ROWS; i++) {
        for(j = 1; j <= COLUMNS; j++) {
            Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] + Temperature_last[i-1][j] +
                                        Temperature_last[i][j+1] + Temperature_last[i][j-1]);
        }
    }

    if(my_PE_num != npes-1){
        MPI_Send(&Temperature[ROWS][1], COLUMNS, MPI_DOUBLE, my_PE_num+1, DOWN, MPI_COMM_WORLD);
    }

    if(my_PE_num != 0){
        MPI_Recv(&Temperature_last[0][1], COLUMNS, MPI_DOUBLE, my_PE_num-1, DOWN, MPI_COMM_WORLD, &status);
    }

    if(my_PE_num != 0){
        MPI_Send(&Temperature[1][1], COLUMNS, MPI_DOUBLE, my_PE_num-1, UP, MPI_COMM_WORLD);
    }

    if(my_PE_num != npes-1){
        MPI_Recv(&Temperature_last[ROWS+1][1], COLUMNS, MPI_DOUBLE, my_PE_num+1, UP, MPI_COMM_WORLD, &status);
    }

    dt = 0.0;

    #pragma acc kernels
    for(i = 1; i <= ROWS; i++){
        for(j = 1; j <= COLUMNS; j++){
            dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
            Temperature_last[i][j] = Temperature[i][j];
        }
    }

    MPI_Reduce(&dt, &dt_global, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
    MPI_Bcast(&dt_global, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    if((iteration % 100) == 0) {
        if (my_PE_num == npes-1){
            #pragma acc update host(Temperature)
            track_progress(iteration);
        }
    }

    iteration++;
}
```

MPI routines using host data

Hybrid OpenACC Programming (Fast & Wrong)

```
#pragma acc data copy(Temperature_last), create(Temperature)
while ( dt_global > MAX_TEMP_ERROR && iteration <= max_iterations ) {

    #pragma acc kernels
    for(i = 1; i <= ROWS; i++) {
        for(j = 1; j <= COLUMNS; j++) {
            Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] + Temperature_last[i-1][j] +
                                        Temperature_last[i][j+1] + Temperature_last[i][j-1]);
        }
    }

    if(my_PE_num != npes-1){
        MPI_Send(&Temperature[ROWS][1], COLUMNS, MPI_DOUBLE, my_PE_num+1, DOWN, MPI_COMM_WORLD);
    }

    if(my_PE_num != 0){
        MPI_Recv(&Temperature_last[0][1], COLUMNS, MPI_DOUBLE, my_PE_num-1, DOWN, MPI_COMM_WORLD, &status);
    }

    if(my_PE_num != 0){
        MPI_Send(&Temperature[1][1], COLUMNS, MPI_DOUBLE, my_PE_num-1, UP, MPI_COMM_WORLD);
    }

    if(my_PE_num != npes-1){
        MPI_Recv(&Temperature_last[ROWS+1][1], COLUMNS, MPI_DOUBLE, my_PE_num+1, UP, MPI_COMM_WORLD, &status);
    }

    dt = 0.0;

    #pragma acc kernels
    for(i = 1; i <= ROWS; i++){
        for(j = 1; j <= COLUMNS; j++){
            dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
            Temperature_last[i][j] = Temperature[i][j];
        }
    }

    MPI_Reduce(&dt, &dt_global, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
    MPI_Bcast(&dt_global, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    if((iteration % 100) == 0) {
        if (my_PE_num == npes-1){
            #pragma acc update host(Temperature)
            track_progress(iteration);
        }
    }

    iteration++;
}
```

MPI routines using host data



Hybrid OpenACC Programming (Slow and Right)

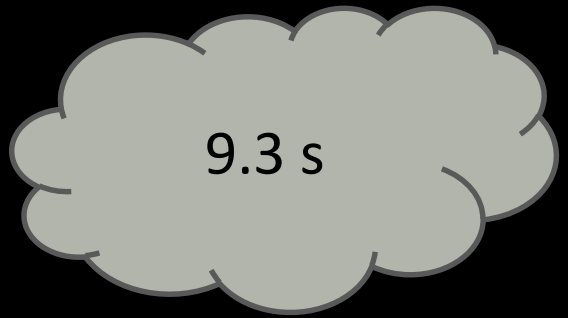
```
#pragma acc data copy(Temperature_last), create(Temperature)
while ( dt_global > MAX_TEMP_ERROR && iteration <= max_iterations ) {

    #pragma acc kernels
    for(i = 1; i <= ROWS; i++) {
        for(j = 1; j <= COLUMNS; j++) {
            Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] + Temperature_last[i-1][j] +
                                        Temperature_last[i][j+1] + Temperature_last[i][j-1]);
        }
    }

    #pragma acc update host(Temperature, Temperature_last)

    if(my_PE_num != npes-1){
        MPI_Send(&Temperature[ROWS][1], COLUMNS, MPI_DOUBLE, my_PE_num+1, DOWN, MPI_COMM_WORLD);
    }

    if(my_PE_num != 0){
        MPI_Recv(&Temperature_last[0][1], COLUMNS, MPI_DOUBLE, my_PE_num-1, DOWN, MPI_COMM_WORLD, &status);
    }

    if(my_PE_num != 0){
        MPI_Send(&Temperature[1][1], COLUMNS, MPI_DOUBLE, my_PE_num-1, UP, MPI_COMM_WORLD);
    }

    if(my_PE_num != npes-1){
        MPI_Recv(&Temperature_last[ROWS+1][1], COLUMNS, MPI_DOUBLE, my_PE_num+1, UP, MPI_COMM_WORLD, &status);
    }

    #pragma acc update device(Temperature, Temperature_last)

    dt = 0.0;

    #pragma acc kernels
    for(i = 1; i <= ROWS; i++){
        for(j = 1; j <= COLUMNS; j++){
            dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
            Temperature_last[i][j] = Temperature[i][j];
        }
    }

    MPI_Reduce(&dt, &dt_global, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
    MPI_Bcast(&dt_global, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    if((iteration % 100) == 0) {
        if (my_PE_num == npes-1){
            #pragma acc update host(Temperature)
            track_progress(iteration);
        }
    }

    iteration++;
}
```

Update data entering and leaving MPI section

Hybrid OpenACC Programming (Slow and Right)

```
#pragma acc data copy(Temperature_last), create(Temperature)
while ( dt_global > MAX_TEMP_ERROR && iteration <= max_iterations ) {

    #pragma acc kernels
    for(i = 1; i <= ROWS; i++) {
        for(j = 1; j <= COLUMNS; j++) {
            Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] + Temperature_last[i-1][j] +
                                        Temperature_last[i][j+1] + Temperature_last[i][j-1]);
        }
    }

    #pragma acc update host(Temperature, Temperature_last)

    if(my_PE_num != npes-1){
        MPI_Send(&Temperature[ROWS][1], COLUMNS, MPI_DOUBLE, my_PE_num+1, DOWN, MPI_COMM_WORLD);
    }

    if(my_PE_num != 0){
        MPI_Recv(&Temperature_last[0][1], COLUMNS, MPI_DOUBLE, my_PE_num-1, DOWN, MPI_COMM_WORLD, &status);
    }

    if(my_PE_num != 0){
        MPI_Send(&Temperature[1][1], COLUMNS, MPI_DOUBLE, my_PE_num-1, UP, MPI_COMM_WORLD);
    }

    if(my_PE_num != npes-1){
        MPI_Recv(&Temperature_last[ROWS+1][1], COLUMNS, MPI_DOUBLE, my_PE_num+1, UP, MPI_COMM_WORLD, &status);
    }

    #pragma acc update device(Temperature, Temperature_last)

    dt = 0.0;

    #pragma acc kernels
    for(i = 1; i <= ROWS; i++){
        for(j = 1; j <= COLUMNS; j++){
            dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
            Temperature_last[i][j] = Temperature[i][j];
        }
    }

    MPI_Reduce(&dt, &dt_global, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
    MPI_Bcast(&dt_global, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    if((iteration % 100) == 0) {
        if (my_PE_num == npes-1){
            #pragma acc update host(Temperature)
            track_progress(iteration);
        }
    }

    iteration++;
}
```

Update data entering and leaving MPI section

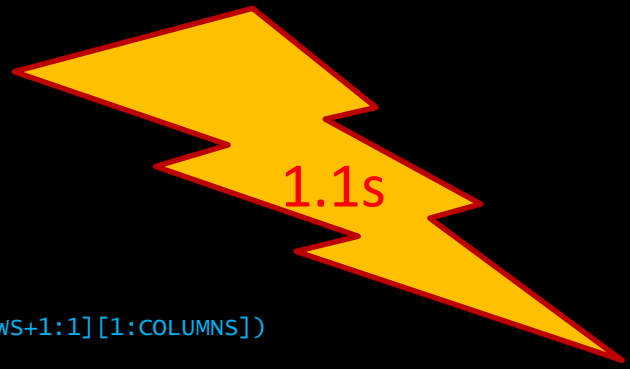


Hybrid OpenACC Programming (Much Better)

```
#pragma acc data copy(Temperature_last), create(Temperature)
while ( dt_global > MAX_TEMP_ERROR && iteration <= max_iterations ) {

    #pragma acc kernels
    for(i = 1; i <= ROWS; i++) {
        for(j = 1; j <= COLUMNS; j++) {
            Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] + Temperature_last[i-1][j] +
                                        Temperature_last[i][j+1] + Temperature_last[i][j-1]);
        }
    }

    #pragma acc update host(Temperature[1:1][1:COLUMNS],Temperature[ROWS:1][1:COLUMNS])

    if(my_PE_num != npes-1){
        MPI_Send(&Temperature[ROWS][1], COLUMNS, MPI_DOUBLE, my_PE_num+1, DOWN, MPI_COMM_WORLD);
    }

    if(my_PE_num != 0){
        MPI_Recv(&Temperature_last[0][1], COLUMNS, MPI_DOUBLE, my_PE_num-1, DOWN, MPI_COMM_WORLD, &status);
    }

    if(my_PE_num != 0){
        MPI_Send(&Temperature[1][1], COLUMNS, MPI_DOUBLE, my_PE_num-1, UP, MPI_COMM_WORLD);
    }

    if(my_PE_num != npes-1){
        MPI_Recv(&Temperature_last[ROWS+1][1], COLUMNS, MPI_DOUBLE, my_PE_num+1, UP, MPI_COMM_WORLD, &status);
    }

    #pragma acc update device(Temperature_last[0:1][1:COLUMNS], Temperature_last[ROWS+1:1][1:COLUMNS])

    dt = 0.0;

    #pragma acc kernels
    for(i = 1; i <= ROWS; i++){
        for(j = 1; j <= COLUMNS; j++){
            dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
            Temperature_last[i][j] = Temperature[i][j];
        }
    }

    MPI_Reduce(&dt, &dt_global, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
    MPI_Bcast(&dt_global, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    if((iteration % 100) == 0) {
        if (my_PE_num == npes-1){
            #pragma acc update host(Temperature)
            track_progress(iteration);
        }
    }

    iteration++;
}
```

Hybrid OpenACC Programming (Much Better)

```
#pragma acc data copy(Temperature_last), create(Temperature)
while ( dt_global > MAX_TEMP_ERROR && iteration <= max_iterations ) {

    #pragma acc kernels
    for(i = 1; i <= ROWS; i++) {
        for(j = 1; j <= COLUMNS; j++) {
            Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] + Temperature_last[i-1][j] +
                                        Temperature_last[i][j+1] + Temperature_last[i][j-1]);
        }
    }

    #pragma acc update host(Temperature[1:1][1:COLUMNS],Temperature[ROWS:1][1:COLUMNS])

    if(my_PE_num != npes-1){
        MPI_Send(&Temperature[ROWS][1], COLUMNS, MPI_DOUBLE, my_PE_num+1, DOWN, MPI_COMM_WORLD);
    }

    if(my_PE_num != 0){
        MPI_Recv(&Temperature_last[0][1], COLUMNS, MPI_DOUBLE, my_PE_num-1, DOWN, MPI_COMM_WORLD, &status);
    }

    if(my_PE_num != 0){
        MPI_Send(&Temperature[1][1], COLUMNS, MPI_DOUBLE, my_PE_num-1, UP, MPI_COMM_WORLD);
    }

    if(my_PE_num != npes-1){
        MPI_Recv(&Temperature_last[ROWS+1][1], COLUMNS, MPI_DOUBLE, my_PE_num+1, UP, MPI_COMM_WORLD, &status);
    }

    #pragma acc update device(Temperature_last[0:1][1:COLUMNS], Temperature_last[ROWS+1:1][1:COLUMNS])

    dt = 0.0;

    #pragma acc kernels
    for(i = 1; i <= ROWS; i++){
        for(j = 1; j <= COLUMNS; j++){
            dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
            Temperature_last[i][j] = Temperature[i][j];
        }
    }

    MPI_Reduce(&dt, &dt_global, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
    MPI_Bcast(&dt_global, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    if((iteration % 100) == 0) {
        if (my_PE_num == npes-1){
            #pragma acc update host(Temperature)
            track_progress(iteration);
        }
    }

    iteration++;
}
```



Hybrid OpenMP Programming
(Most "complex" version: MPI_THREAD_MULTIPLE)

```
#include <mpi.h>
#include <omp.h>

//Last thread of PE 0 sends its number to PE 1

main(int argc, char* argv[]){

   int provided, myPE, thread, last_thread, data=0, tag=0;
   MPI_Status status;

   MPI_Init_thread(&argc, &argv, MPI_THREAD_MULTIPLE, &provided);
   MPI_Comm_rank(MPI_COMM_WORLD, &myPE);

   #pragma omp parallel firstprivate(thread, data, tag, status)
   {
        thread = omp_get_thread_num();
        last_thread = omp_get_num_threads()-1;

        if ( thread==last_thread && myPE==0 )
             MPI_Send(&thread, 1, MPI_INT, 1, tag, MPI_COMM_WORLD);
        else if ( thread==last_thread && myPE==1 )
             MPI_Recv(&data, 1, MPI_INT, 0, tag, MPI_COMM_WORLD, &status);

        printf("PE %d, Thread %d, Data %d\n", myPE, thread, data);
   }

   MPI_Finalize();
}
```

```
% export OMP_NUM_THREADS=4
% aprun -n3 -N1 -d4 a.out
PE 0, Thread 0, Data 0
PE 1, Thread 0, Data 0
PE 2, Thread 0, Data 0
PE 2, Thread 3, Data 0
PE 0, Thread 3, Data 0
PE 1, Thread 3, Data 3
PE 0, Thread 2, Data 0
PE 2, Thread 2, Data 0
PE 1, Thread 2, Data 0
PE 0, Thread 1, Data 0
PE 1, Thread 1, Data 0
PE 2, Thread 1, Data 0
```

Output for 4 threads run on 3 PEs

Mix and Match

- PGI Compile:

```
mpicc –acc laplace_hybrid.c
mpf90 –acc laplace_hybrid.f90
mpicc –mp –acc laplace_hybrid.c
etc…
```

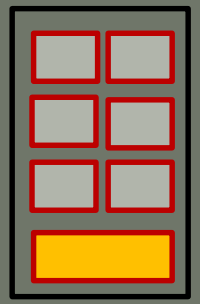
- Running:

```
interact ?
     –n 4
     -N1 –n4
     -p GPU –N1 –n4
     -p GPU –N4 –n4
     -N1 –n28
     -N4 –n112
     etc…
```

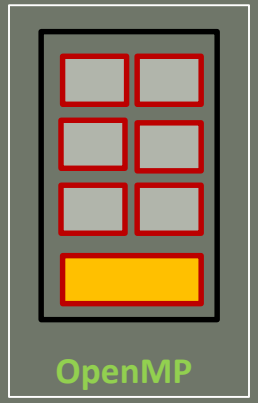
Bottom Line…

- Each one of these approaches occupies its own space.
- If you understand this, you will not be confused as to how they fit together.
- Once again…

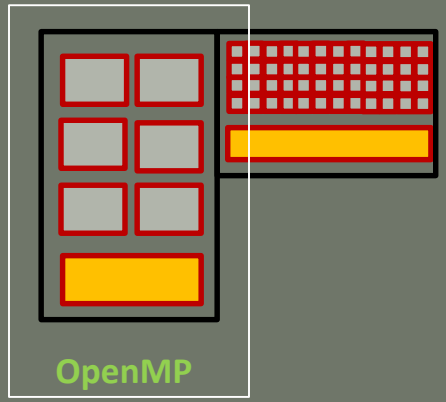
In Conclusion...



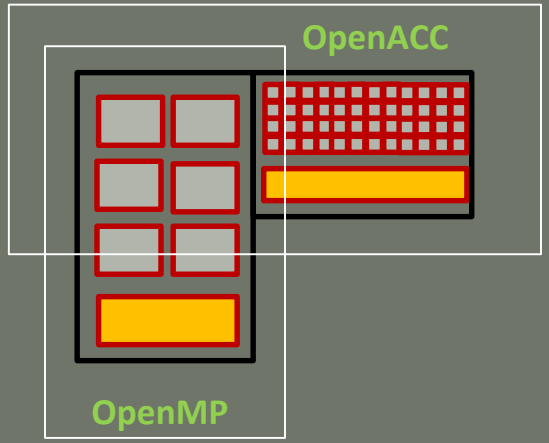
In Conclusion…



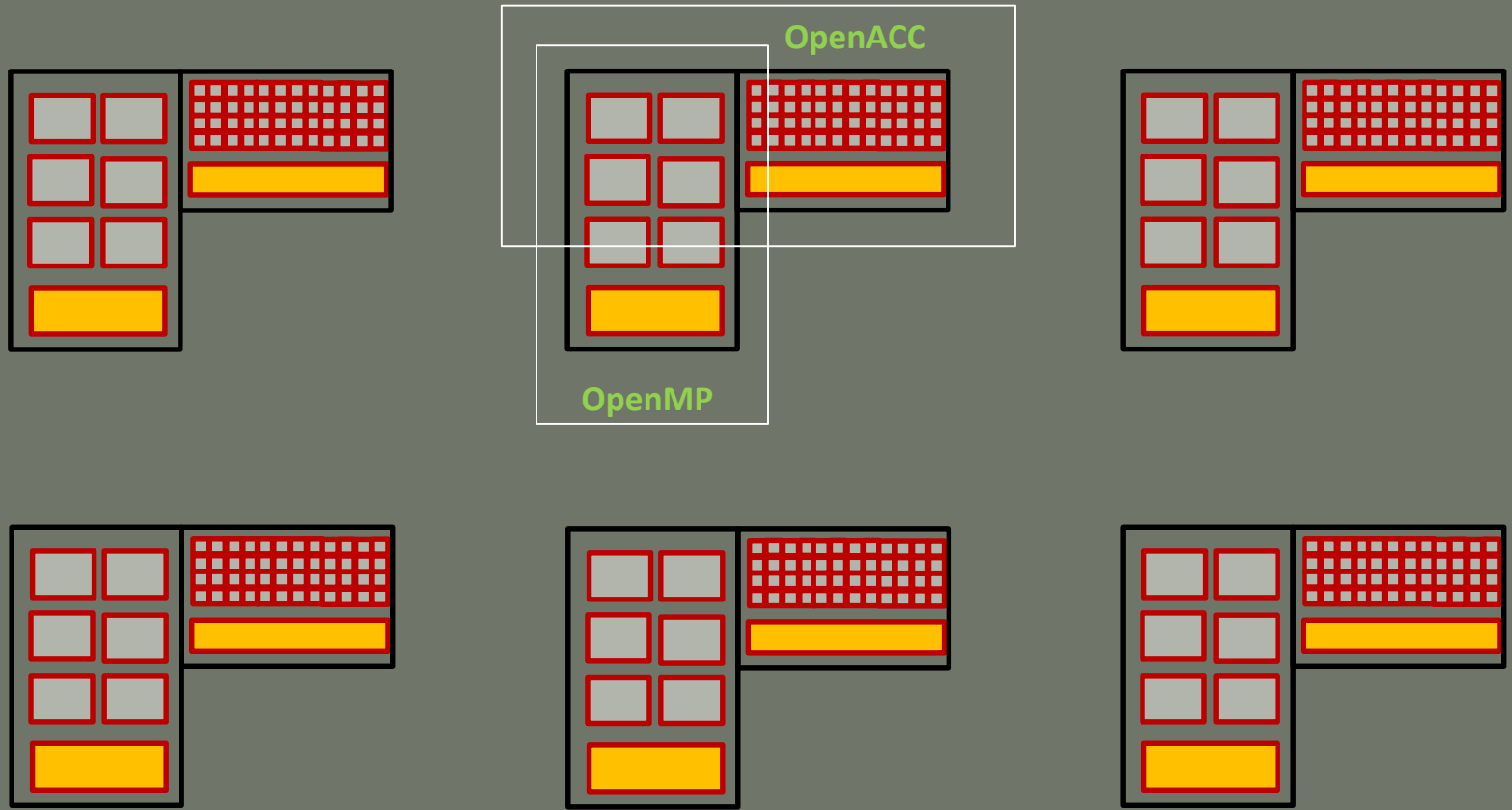
In Conclusion…



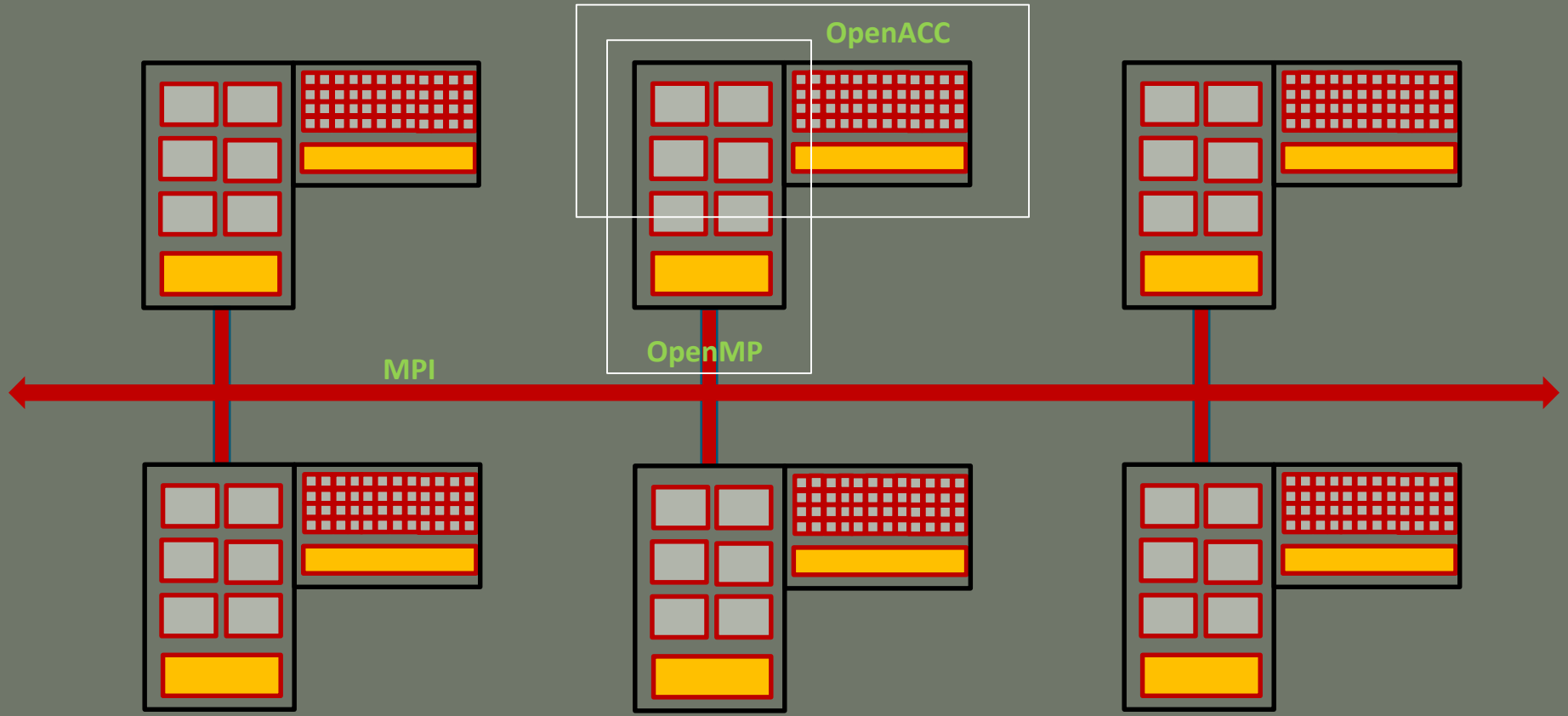
In Conclusion…



In Conclusion…



In Conclusion…

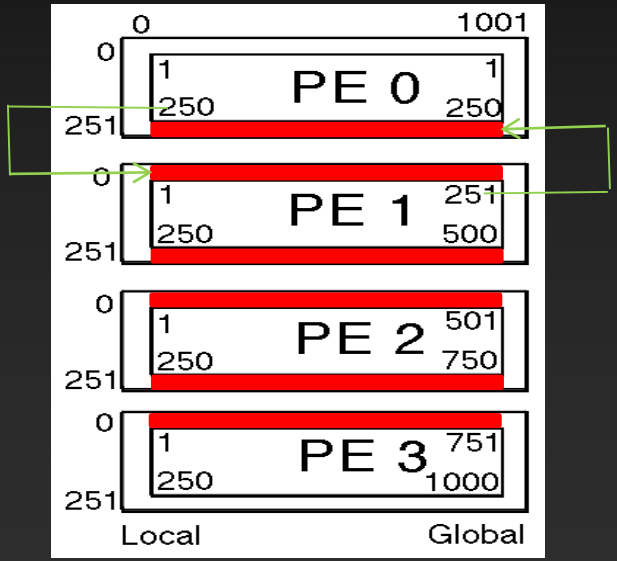


Hybrid Challenge Head Start

John Urbanic
Parallel Computing Scientist
Pittsburgh Supercomputing Center

Simplest Decomposition for C Code

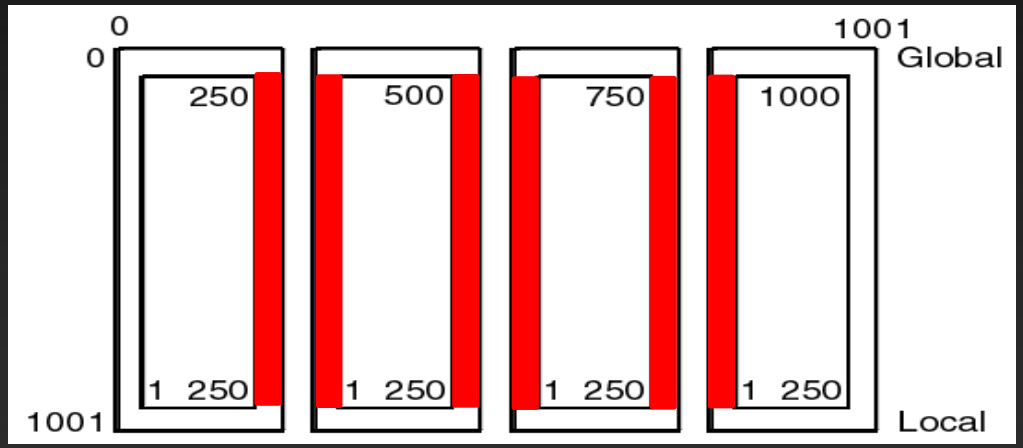
In the parallel case, we will break this up into 4 processors. There is only one set of boundary values. But when we distribute the data, each processor needs to have an extra row for data distribution, these are commonly called the "ghost cells".



The program has a local view of data. The programmer has to have a global view of data. The ghost cells don't exist in the global dataset. They are only copies from the "real" data in the adjacent PE.

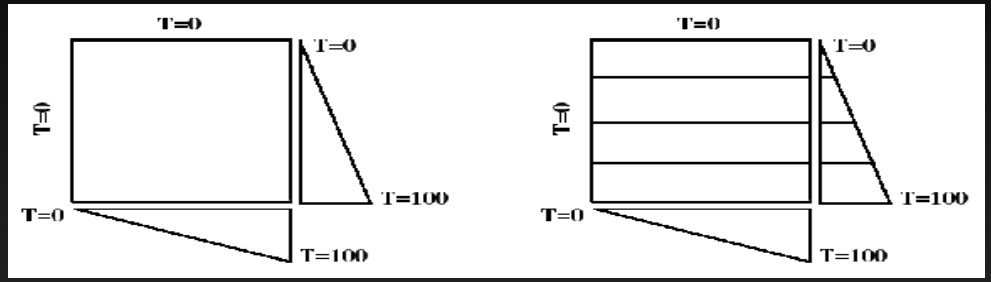
Simplest Decomposition for Fortran Code

Then we send strips to ghost zones like this:

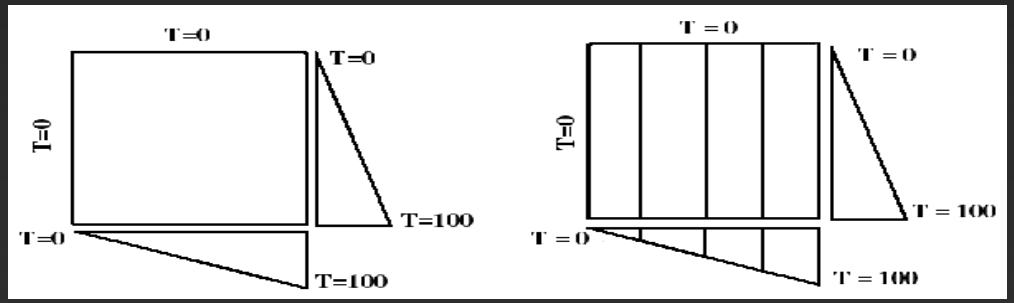


Same ghost cell structure as the C code, we have just swapped rows and columns.

Boundary Conditions



Both C and Fortran will need to set proper boundary conditions based upon the PE number.



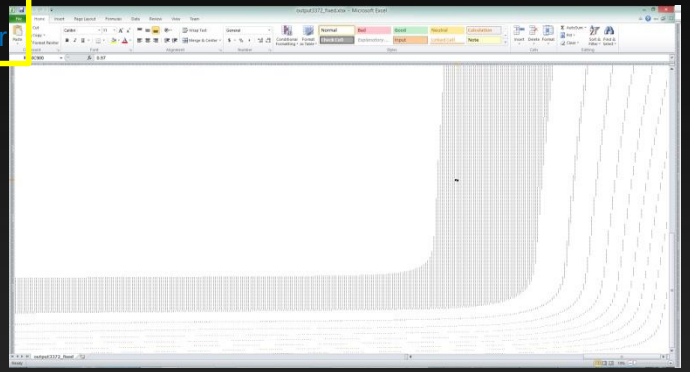
How do you know you are correct?

Your solution converges
at 3372 timesteps!

How do you know you are correct?

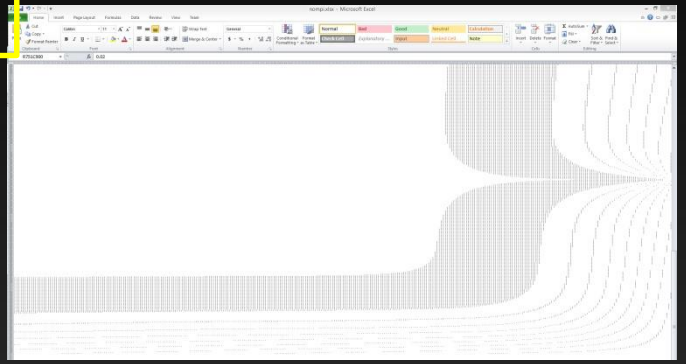
How do you know you are correct?

Bottom Right Corner



Working MPI Solution

Bottom Right Corner

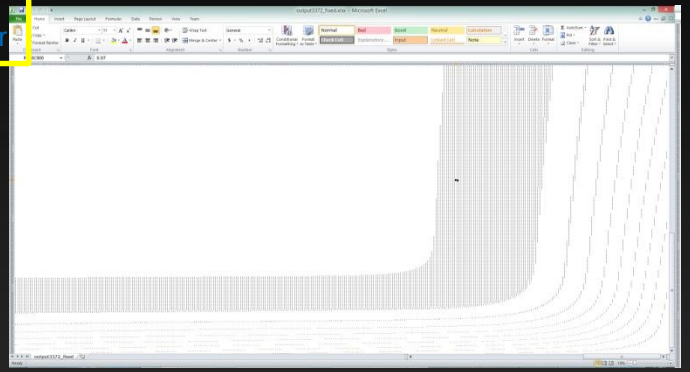


MPI Routines Disabled

Both converge at 3372 steps!

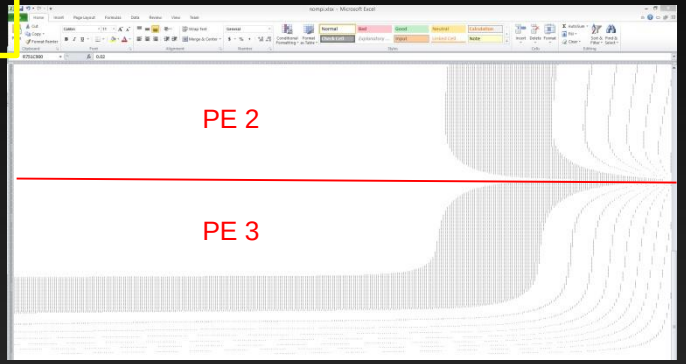
How do you know you are correct?

Bottom Right Corner



Working MPI Solution

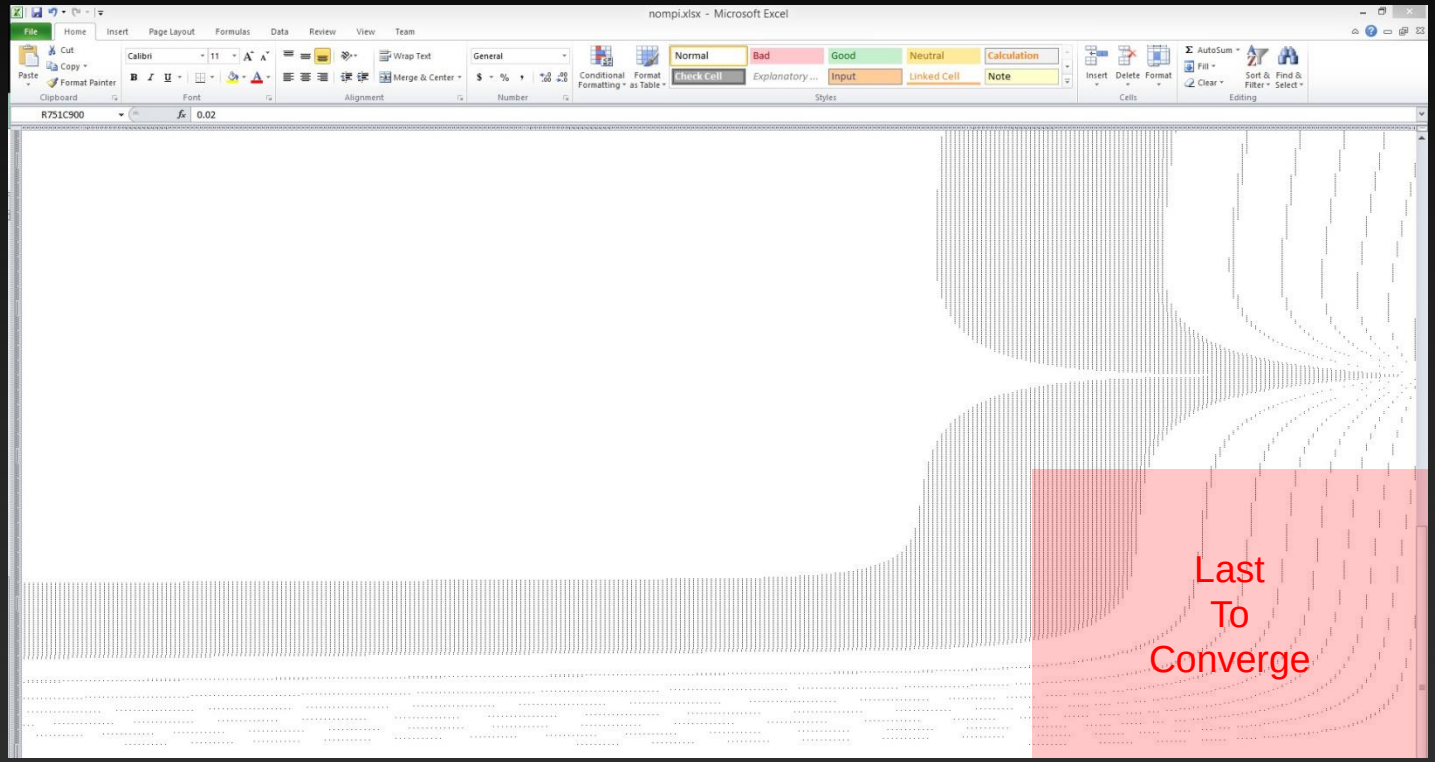
Bottom Right Corner



MPI Routines Disabled

Both converge at 3372 steps!

All the action is here.



Check for yourself.

```
void output(int my_pe, int iteration) {

  FILE* fp;
  char filename[50];

  sprintf(filename,"output%d.txt",iteration);

  for (int pe = 0; pe<4; pe++){
    if (my_pe==pe){

      fp = fopen(filename, "a");

      for(int y = 1; y <= ROWS; y++){
        for(int x = 1; x <= COLUMNS; x ++){
          fprintf(fp, "%5.2f ",Temperature[y][x]);
        }
        fprintf(fp,"\n");
      }

      fflush(fp);
      fclose(fp);

    }
    MPI_Barrier(MPI_COMM_WORLD);

  }

}
```

- Human Readable
- 1M entries
- Visualize. I used Excel (terrible idea).

Check for yourself.

```
void output(int my_pe, int iteration) {

  FILE* fp;
  char filename[50];

  sprintf(filename,"output%d.txt",iteration);

  for (int pe = 0; pe<4; pe++){
    if (my_pe==pe){

      fp = fopen(filename, "a");

      for(int y = 1; y <= ROWS; y++){
        for(int x = 1; x <= COLUMNS; x ++){
          fprintf(fp, "%5.2f ",Temperature[y][x]);
        }
        fprintf(fp,"\n");
      }

      fflush(fp);
      fclose(fp);

    }
    MPI_Barrier(MPI_COMM_WORLD);

  }

}
```

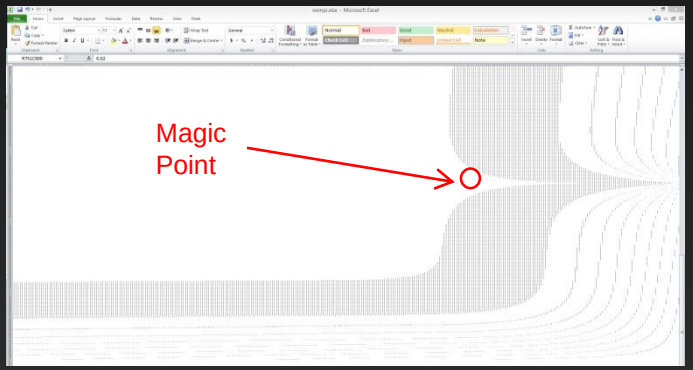
- Human Readable
- 1M entries
- Visualize. I used Excel (terrible idea).

C:

```
if (my_PE_num==2)
      printf("Global coord [750,900] is %f \n:", Temperature[250][900]);
```

Fortran:

```
if (mype==2) then
        print*, 'magic point', temperature(900,250)
endif
```



- If about 1.0, probably good
- Otherwise (like 0.02 here) probably not

Running to convergence

3372 iterations to converge to 0.01 max delta.

	Serial (s)	4 Cores	4 Nodes	Speedup
C	16.0	4.9	4.8	3.3
Fortran	16.0	4.1	3.9	4.0

Note that all versions converge on the same iteration. This kind of repeatability should be expected. However, exact binary repeatability is usually not possible due simply to floating point operation reordering.

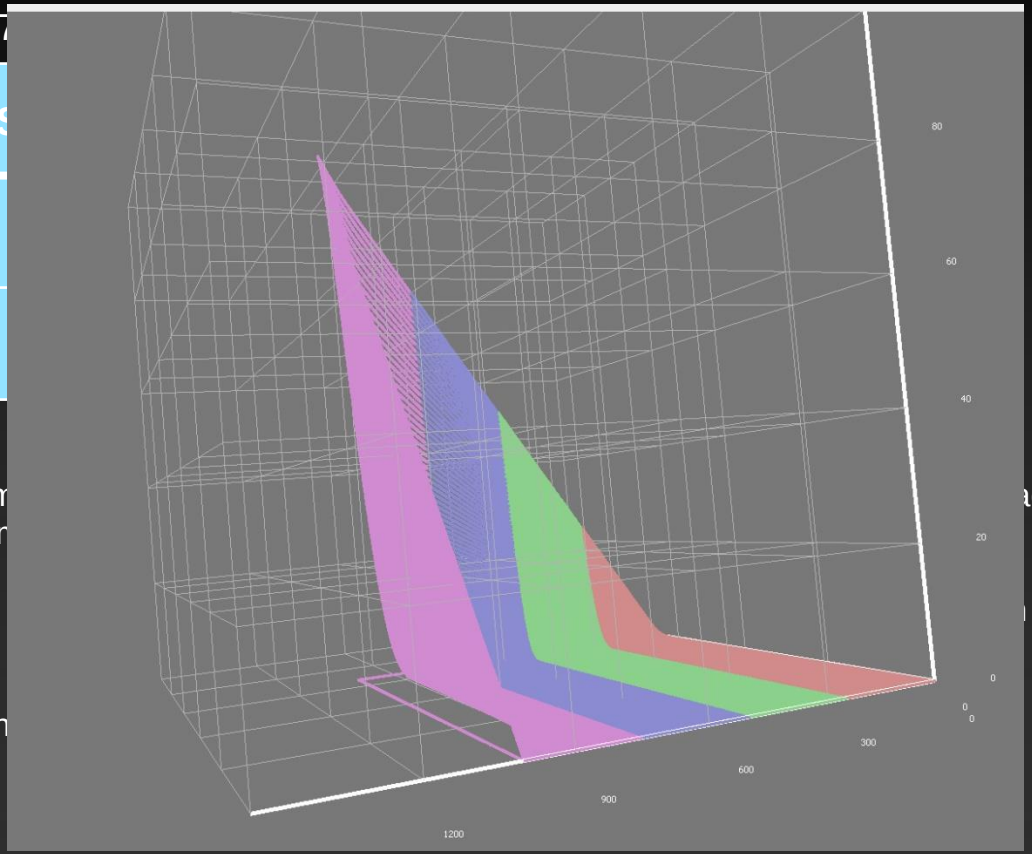
Scaling off the node will typically be much better than scaling on the node for a well written problem of this type run at normal scale.

To run on 4 nodes you need to request 4 nodes from the queue: **interact -N 4 -n 4**

Running to convergence

337

C	
Fortran	



Note that all versions converge on the sam ...act binary repeatability is usually not possible due sir...

Scaling off the node will typically be much ... at normal scale.

To run on 4 nodes you need to request 4 n...

Running to convergence

3372 iterations to converge to 0.01 max delta.

	Serial (s)	4 Cores	4 Nodes	Speedup
C	16.0	4.9	4.8	3.3
Fortran	16.0	4.1	3.9	4.0

Note that all versions converge on the same iteration. This kind of repeatability should be expected. However, exact binary repeatability is usually not possible due simply to floating point operation reordering.

Scaling off the node will typically be much better than scaling on the node for a well written problem of this type run at normal scale.

To run on 4 nodes you need to request 4 nodes from the queue: **interact -N 4 -n 4**

Send_init and Recv_init as used by a *Summer Boot Camp Hybrid Challenge* winner

```
call MPI_Send_Init(temperature(1,columns), rows, MPI_DOUBLE_PRECISION, right, 1r, MPI_COMM_WORLD, request(1), ierr)
call MPI_Recv_Init(temperature_last(1,0), rows, MPI_DOUBLE_PRECISION, left, 1r, MPI_COMM_WORLD, request(2), ierr)
// 8 of these as winning solution did a 2D (left, right, up, down) decomposition on 10,000 x 10,000 size problem
      .
      .

do while ( dt_global > max_temp_error .and. iteration <= max_iterations)

         do j=1,columns
            do i=1,rows
               temperature(i,j)=0.25*(temperature_last(i+1,j)+temperature_last(i-1,j)+ &
                                      temperature_last(i,j+1)+temperature_last(i,j-1) )
            enddo
         enddo
      .
      .
         call MPI_StartAll(8,request,statuses)

         dt=0.0
      .
         do j=1,columns
            do i=1,rows
               dt = max( abs(temperature(i,j) - temperature_last(i,j)), dt )
               temperature_last(i,j) = temperature(i,j)
            enddo
         enddo
      .
      .
         call MPI_WaitAll(8,request,statuses,ierr)
      .
      .
enddo
```

Allow communications to overlap with the temperature_last update and maximum delta search.

Make sure all is complete before using this data in the next iteration.

Rules and Regulations of the 2nd Annual IHPCSS Challenge

Trophy bears no relationship to reality.

General Rules

- Due Thursday midnight (!)
- 4 Nodes of Bridges
- Use any combination of MPI, OpenACC, Python and OpenMP
- How fast can you run a 10K x 10K Laplace code to convergence?

Some Specifics

- Can't change kernel (Must retain two core loops source)
- Can change number of MPI processes (Does not have to be 112 or 4)
- 1 Source File
- 1 Combined Environment/Compile/Submit/Execute script
  - to make it easy for us to run your solutions!
- Mail to d.henty@epcc.ed.ac.uk by deadline
  - Mail a ping earlier if you want to be informed of any developments

Rules For Lawyers

- No libraries
- Don’t mess with timer placement
- ?

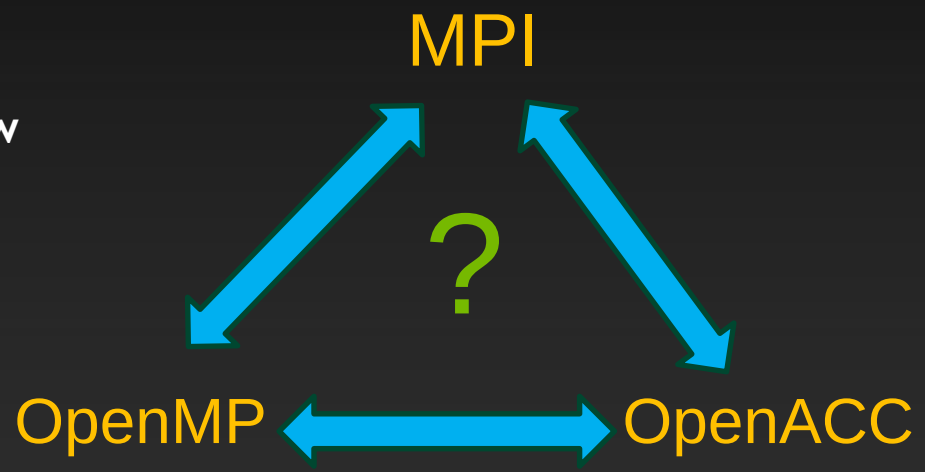
Reality Checks

- Serial code converges at 3578 time steps. Yours should too.
- As we know, this is not enough to verify correctness. You should find point [7500][9950] in C and (9950,7500) in Fortran converges to 17 (±1) degrees.
- As discussed, the 10K result differs from the 1K result.*
  - Plugging in Gauss-Seidel or Successive Over Relaxation (SOR) would be easy and interesting. But, not for our contest.

http://www.cs.berkeley.edu/~demmel/cs267/lecture24/lecture24.html is a brief analysis of these issues.

Suggested Things to Explore

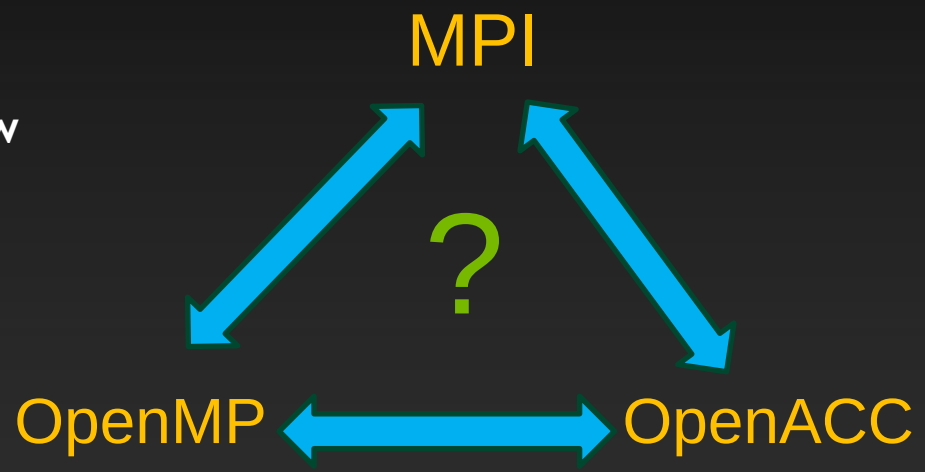
- Compiler flags
  - -fast
- Compiler
  - see Bridges documentation for how to use different modules
- MPI Environment Variables
  - man mpi
- Thread placement
  - google for KMP_AFFINITY



Suggested Things to Explore

Blue Waters User Guide is your friend!
https://bluewaters.ncsa.illinois.edu/user-guide

- Compiler flags
  - -fast
- Compiler
  - see Bridges documentation for how to use different modules
- MPI Environment Variables
  - man mpi
- Thread placement
  - google for KMP_AFFINITY



Decision

- On Thursday evening we will take the top self-reported speeds and run them in an interactive session
- Timings not within 10% of self-reported time will be disqualified
- Codes should print out “test point” at [7500][9950] for C, (9950,7500) for Fortran at conclusion of run.
- Best of two runs for each finalist will determine winner