Laplace Exercise Solution Discussion

John Urbanic

Parallel Computing Scientist
Pittsburgh Supercomputing Center

Distinguished Service Professor
Carnegie Mellon University

Copyright 2026

Finished?

If you have finished, we can review a few principles that you have inevitably applied. If you have not, there won't be any spoilers here. It you want spoilers, you should look in ~training/Laplace at

laplace_mpi.f90

laplace_mpi.c

We have a lot more exercise time today, and you also have your accounts through at least next week. So don't feel pressured to give up or cheat.

Two things I know you did.

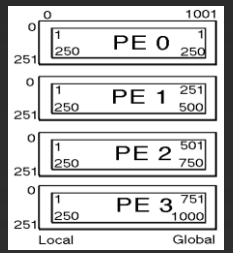
Even though I may not have been looking over your shoulder, I know that you had to apply the domain decomposition process that we discussed is universal to MPI programming. In this case you had to:

1) Identify the main data structures of the code:

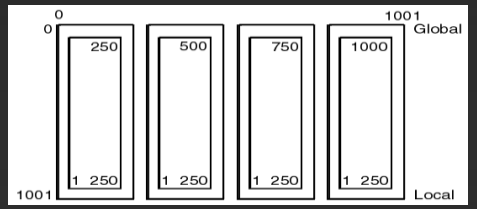
Temperature and *Temperature_last*

2) Decompose both those data structures with a forward looking strategy:

C



Fortran



PITTSBURGH SUPERCOMPUTING CENTER

Digging through all the code

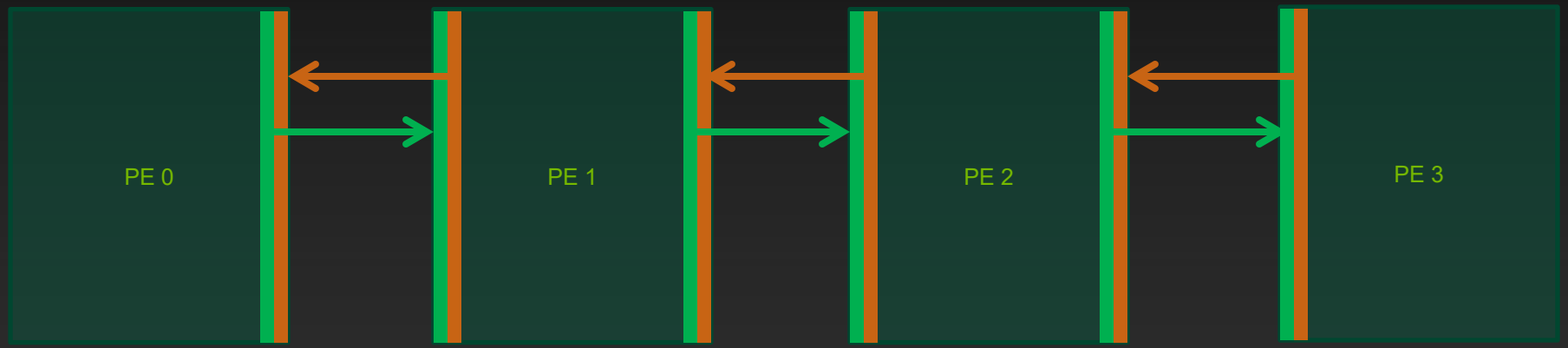
This code has subroutines, like all real codes. And, like all real codes, you have to follow the main data structures that you are modifying into those subroutines.

If you had to decompose it, you almost always have to tweak the code that involves that data structure.

Here, you had to modify the boundary conditions and you had to modify the IO

On the other hand, you did *not* have to modify the kernel, or real math, or effectively the science of the code. This is also typical of a real MPI port.

Careful of the deadlocks

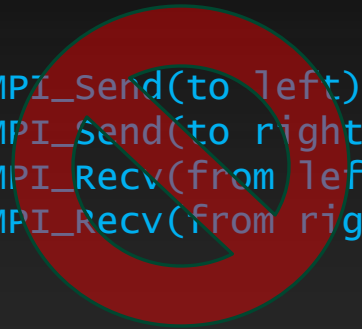


The classic “ghost zone” data exchange.

PITTSBURGH SUPERCOMPUTING CENTER

Two Blocking Methods

There are two similar ways of coding this that we might try:



```
MPI_Send(to left)
MPI_Send(to right)
MPI_Recv(from left)
MPI_Recv(from right)
```

```
MPI_Send(to left)
MPI_Recv(from right)
MPI_Send(to right)
MPI_Recv(from left)
```

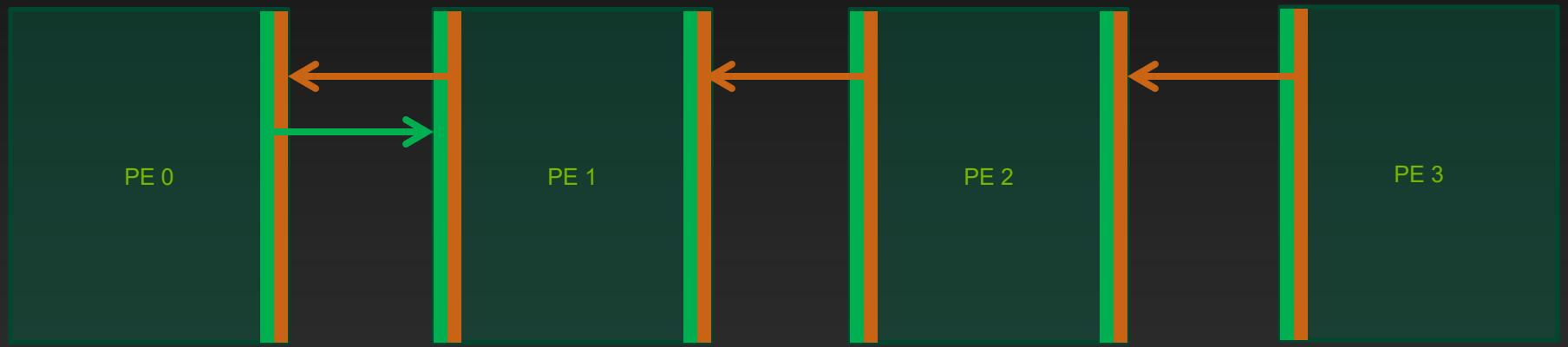
On Blue Waters they both worked OK solving the beginning 1000x1000 problem. But when we scaled up to the full competition size (10000x10000) one of them hangs. Where?

PE's 1-3 are blocking sending to the left, and PE 0 is blocking on the send to the right.

Hung…

PE's 1-3 are blocking sending to the left, and PE 0 is blocking on the send to the right.

```
MPI_Send(to left)
MPI_Send(to right)
MPI_Recv(from left)
MPI_Recv(from right)
```

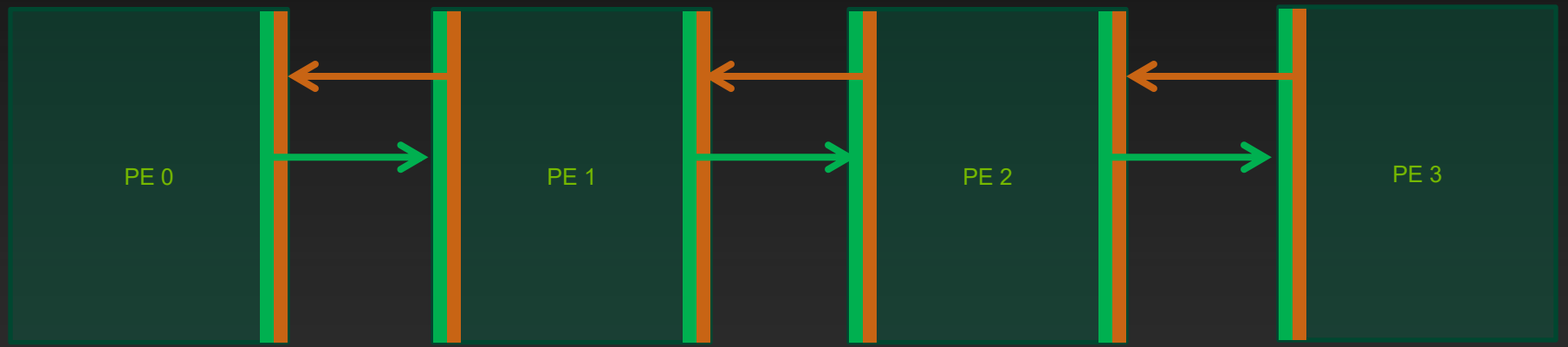


Is our other solution truly the answer? *Note that using MPI_Ssend() here would have caught this problem right away!*

Cascading Messages

```
MPI_Send(to left)
MPI_Recv(from right)
MPI_Send(to right)
MPI_Recv(from left)
```

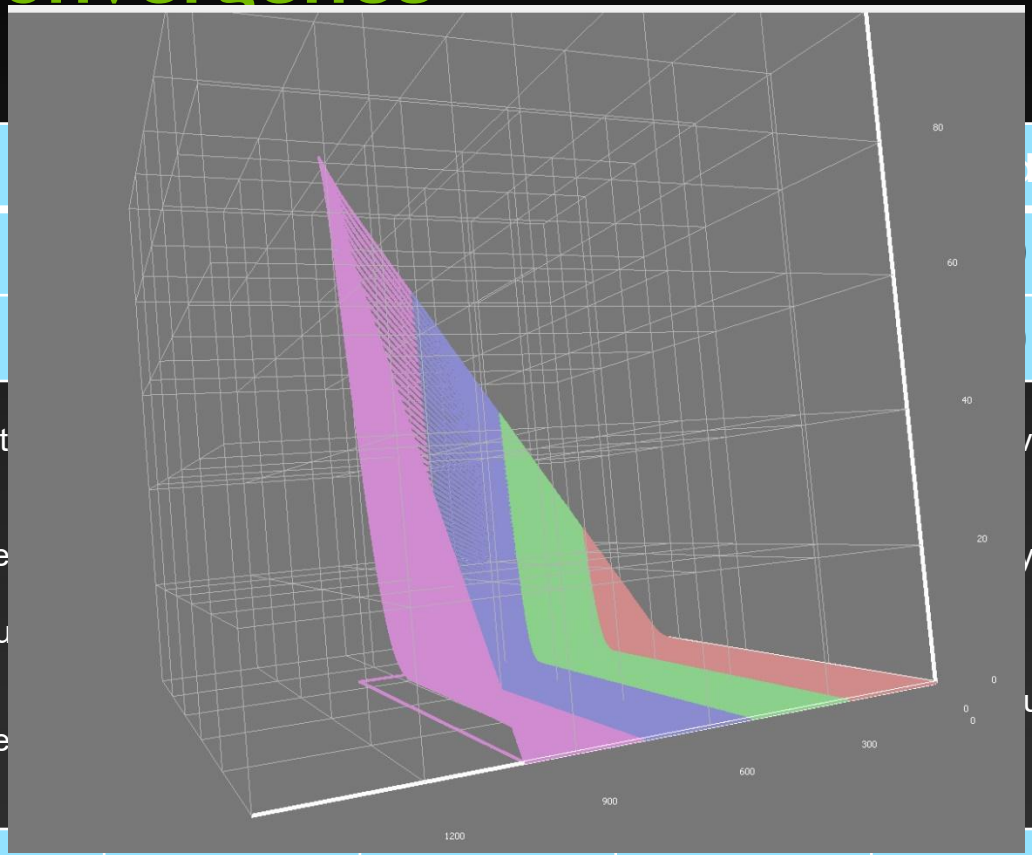
At least the second solution doesn't' hang. But it does results in a sequential process here that we don't really want.



Might want to use non-blocking…

PITTSBURGH SUPERCOMPUTING CENTER

Running to convergence



				dup
C				
Fortran				

Note that all versions converge on t ver, exact binary repeatability is usually not possible

Scaling off the node will typically be ype run at normal scale.

To run on 4 nodes you need to requ

The above was the default ("mpicc u should use the -O3 optimize option ("mpicc -O3 laplace

C (-O3)	3.5	0.7	0.6	5.0

PITTSBURGH SUPERCOMPUTING CENTER

Vs. OpenMP

For all our efforts, we only achieved a speedup of roughly what we could do with a few lines of OpenMP on the same 4 cores. Why would we ever use MPI on a problem of this type?

The answer is simply that OpenMP is limited to the size of the single largest node (by which we probably mean box, or blade, or perhaps even cabinet). Bridges has some *very* large nodes (12TB with 260 cores), and even then you would find the performance falls off when using all the cores.

Our MPI code can run across any networked collection of equipment we can assemble. On Bridges this can realistically be all 800+ nodes, each with many cores. And for MPI jobs, there are even larger machines out there.

And, as we will discuss with hybrid computing, you can combine both approaches quite comfortably. They are designed to do so.

MPI has lots of ways to make this even easier

Some trivial to just edit right in:

- *MPI_Sendrecv*

Some with little effort, but a bigger payoff:

- Defined Data Types: *MPI_Type_commit*
- Non-Blocking Messages: *Isend*
- Persistent Communications: *MPI_Send_Init*

And some that require more thought, but would be appropriate for an Exascale code:

- *MPI_Put*

We will get to all of these options in the Advanced MPI talk.

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.

PITTSBURGH SUPERCOMPUTING CENTER

Model Improvements

The Laplace code is a realistic serial-to-MPI example. We can extend this example even further into the world of real application codes with some modifications that you could pursue.

1) 3D. To make this code into a full 3D application would be a straightforward, if tedious, addition of indices to the arrays and loops. However, you would have to reconsider your data decomposition and your communication routines would now have to pass larger 2D "surfaces" (non-blocking messages) of ghost cell data instead of just strips (user defined datatypes).

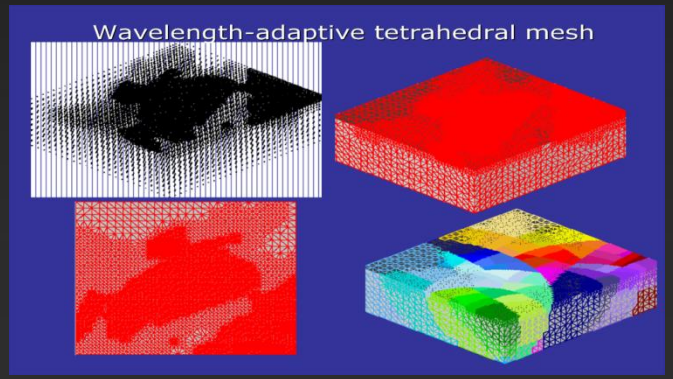
2) Data Decomposition….

Data Decomposition

Whether we go to 3D or not, it is desirable, and usually necessary, that the code be able to run on variable numbers of PEs. Sometimes you have 4 PEs, and sometimes you have 3,144,412 to use.

Furthermore, real problems are not usually so "cartesian". MPI excels at these additional constraints. Indeed, it is often the only practical option.

However, the techniques that enable this flexibility benefit greatly from some of the additional MPI routines that we have yet to cover.



Domain decompositions can be quite complex in applications with irregular shapes.