

When Good Code Goes Bad - Workshop on Debugging December 2006

Basic Example

Transcript

The first thing we want to do is take a look at the problem. Now, what I have is the worst legitimate matrix multiplying algorithm. It's just a brute force $C = A \times B$. And we're using square matrices.

The details are easy enough to follow; it shouldn't distract from what it is that the code does, so it will make it a little easier to discuss the debugging issues involved.

The way that the problem is structured is not atypical code that we see. It's put together in a fairly straightforward but unstructured manner.

And this is the code we're going to be working with. I'm going to go through it once just to give you the idea of how long the code is. Then I want to talk in detail about what it does.

We have MatMul. It has very few declarations. These are just simply the integers required to specify the matrix, and then "start" and "finish" are a set of arrays used to describe how the matrix is partitioned.

We have the MPI initialization functions. This matrix is in a file called "matrix.dat". The very first element is the size of the matrix. The remaining elements, in order and in ASCII, are the array elements.

We figure out the size of the array, and in this step we read the array in. This section of code is the partitioning of the array. It's broken into even sized elements. If breaking it apart creates elements that can't be distributed evenly across all processors, then we simply tack one additional element onto each of the first N partitions.

In this section of code we set up the partitioned arrays, and then send them to each of the processors involved in the program.

This chunk of code here is the partitioning on the receiving processors. Each processor receives a section of the array. Each processor will then create a B array, which is basically a diagonal matrix with twos on the diagonal, and then a C array, which is initialized to zero.

This is the brute force matrix multiplication algorithm. We go along and see the receives for the array; we set up an output file. This should all be just processor one just sending its element of the array to processor zero. And this is processor zero receiving its element.

So, the code's very confusing. And it's important to note that because it has an influence on how debugging proceeds.