

When Good Code Goes Bad - Workshop on Debugging December 2006

Lessons Learned

Transcript

The code is, well, it's a problem. To start with, it's monolithic. You can't see any of the inherent structure of the code, because it just runs one long stream.

There are no comments. It would have been really helpful, for example, to note that NMAX was the size of the array that we were concerned with and MAX was the size of the array we allocated. It would be helpful to know that tag zero meant sending or receiving NMAX, tag one meant sending or receiving the initial A matrix and that tag two meant sending or receiving the result.

There are certain things that we should take away from this. To be effective about what you're doing when you're debugging, you have to figure out where the thing failed, and you can't really figure it out without properly bracketing suspect lines of code--function calls, loops, divides. You bracket those lines with some print statement saying, "This is where we're starting, here's where we should have completed." If you see the start, and you don't see the complete, you know where it failed.

You want to collect reasonable amounts of information. Now, one file I produced had several gigabytes of useless data. You don't really want to do that. But you do want to check things like the values you're passing to functions. Now, if you're passing a thousand by thousand matrix to a function, you don't want to necessarily want to write that out and then go through every one of the values. But you should check the size of the matrix. You should make sure that any simple to read scalar values are actually being passed properly and being received properly.

You might want to have a subset of matrices that you pass, so in the cases where we have a matrix being passed through an MPI send or receive, if you know that you're only passing a certain chunk of the matrix, then you might want to print out the first row or the first column of that matrix to make sure that you're passing something that you can verify is correct.

In general, really, you want to log the data. Printing it to the screen is okay if you can ferret out which lines came from which processor in any easy manner. Otherwise, you can use this built in notion that fortran has that you can declare a output unit and, without naming it, it will produce a file called "fort dot" whatever number you've assigned. Fort dot the error unit number.

Something I haven't covered yet. You want to include a flush on your prints. Now I didn't do that in this first step because this first step's confusing enough without adding too many details.

A computer, when it reaches some sort of print statement, doesn't actually print anything. It puts the data into a buffer. And then it eventually gets around to taking it out of the buffer and putting it onto the screen or putting it into a file.

If the computer crashes before it has an opportunity to print whatever's in its buffer, you've lost that message.

So, by using a flush, and it's simply "call flush" and the unit number, you will force the computer to print that data to the appropriate file or to the screen, and we'll see examples of that shortly.

One thing that we didn't do in this code, or talked about it very briefly, was something that saves endless hours of agony and a huge waste of time. Before you actually start executing code, if you're accepting input from a file or from a user, check to make sure the input makes sense. If you have a limit on your matrix size, as we saw, make sure you didn't exceed that. If you have values that are expected to be integer, make sure that characters weren't put in. And if you have values that were expected to be floats, in a certain range, check the range.

You may spend an additional millisecond of computation, but by spending that additional millisecond of computation and perhaps thirty or forty seconds of development time, you may save yourself running code that would fail for forty-eight hours on a thousand processors. So checking those input values is really pretty important.