Example programs for DeinoMPI

PI

There are several PI calculating examples provided. The algorithm for calculating PI simply divides the area under a curve into a large number of rectangles, each process in the job calculates the area of a subset of the rectangles and then the sum of all the rectangles is collected at the root and printed out to the console. This example does not require much processing power so you should not expect the total run time of the job to decrease with increasing number of processes. It is meant to show an example of a common practice to generate parallel algorithms, namely breaking up a problem into smaller pieces, dividing those pieces evenly amongst the participating processes and then collecting the result back to the root. The PI example is repeated several times to demonstrate some features of DeinoMPI:

- CPI
    - This is a C program that calculates PI
- ICPI
    - This is a C program that calculates PI and allows the user to specify how many iterations should be used and thus how accurate the result should be.
- FPI
    - This is a FORTRAN program that calculates PI
- CXXPI
    - This is a C++ program that calculates PI
- WCPI
    - This is a C program that calculates PI and uses wide character (UNICODE) strings.
- WCXXPI
    - This is a C++ program that calculates PI and uses wide character strings.
- WICPI
    - This is a C program that calculates PI, uses wide character strings and uses input from the user to specify to what accuracy PI should be calculated.

NetPipe

The netpipe example is a highly accurate ping-pong bandwidth calculating program between two processes. It calculates the bandwidth and latency between two processes using increasing message sizes starting from zero byte messages and ending once a single message takes more than 1/10th of a second to transmit. There are command line arguments that can be used to control the output. Execute the program with one process and no arguments to see a help message explaining the optional arguments. The most common usage is simply “mpiexec –n 2 netpipe.exe”

MPPTest

MPPTest is a performance measuring tool. It can measure ping-pong performance, bisectional-bandwidth, and other collective operation measurements. Execute “mpptest –help” to get a usage message explaining all the optional arguments mpptest understands.

Mandlebrot suite

There are several projects involved in the mandlebrot example. This example is divided up into two parts. There is the visualizer that displays the graphical window with the visual output and allows the user to interact with the set using the mouse. The second part is the parallel engine that is used to calculate the data and send the results to the visualizer. There is one visualizer and several examples of the parallel engine.

pman_vis

The visualizer is an MPI program that uses only one process so it doesn't need to be started using mpiexec. Simply execute pman_vis.exe and it will bring up a graphical interface. It does not do any work itself so nothing is displayed until it is connected to a separate parallel worker engine. There are two mechanisms for connecting to a worker engine and the workers determine which method should be used. The visualizer can connect to the root process in the engine using a socket or it can use an MPI port. When the engine is run it prints out either the socket or the MPI port to connect to. Copy this information and paste it into the File::Connect dialog of the visualizer. The visualizer can only connect to one engine and when the visualizer is exited both the visualizer and the engine shut down. If you want to test several engines you will have to restart the visualizer for each engine you want to test. Once the visualizer is connected to an engine it immediately draws the default mandlebrot set. Once this set is drawn you can use the mouse to drag a rectangle and zoom in to any location. You can also input the rectangle bounds directly using the File::Enter Point menu option. Right-clicking in the output window will cause the visualizer to reset the view and zoom all the way back out to the default bounds. You can run the visualizer in demo mode by inputing demo points in the File::Enter demo points menu. There are some examples in the cool.points file that can be loaded from the File::Enter demo points menu. Once the demo points have been loaded the visualizer automatically draws the selected areas of the mandlebrot set. It pauses for a moment after drawing each location and then moves to the next. Select File::Stop demo to stop cycling through the demo locations.

pmandel

The pmandel parallel engine uses a socket to connect to the visualizer and a single MPI_COMM_WORLD to partition out the work. Run this engine like this: "mpiexec –n 4 pmandel". Then connect to this engine using the socket option in the visualizer.

pmandel_service

The pmandel_service parallel engine uses an MPI port to connect to the visualizer and a single MPI_COMM_WORLD to partition out the work. Run this engine like this: "mpiexec –n 4 pmandel_service". Then copy the MPI port that it prints to the console and paste it into the MPI port option of the visualizer connect dialog.

pmandel_spawn

The pmandel_spawn parallel engine uses a socket to connect to the visualizer and spawns worker processes to partition out the work. Run this engine like this: "mpiexec –n 1 pmandel_spawn". This will generate one master and three workers. If you want to control the number of workers then run the example like this: “mpiexec –n 1 pmandel_spawn –n 10”. This will create one master and 10 workers. Then connect to this engine using the socket option in the visualizer.

pmandel_spaserv

The pmandel_spaserv parallel engine uses an MPI port to connect to the visualizer and spawns worker processes to partition out the work. Run the engine the same way you would the pmandel_spawn example: “mpiexec –n 1 pmandel_spaserv –n 10”. Then copy the MPI port that is printed to the console and paste it into the connect dialog in the visualizer.

pmandel_fence

The pmandel_fence parallel engine uses a socket to connect to the visualizer and a single MPI_COMM_WORLD to partition out the work. Run this engine like this: “mpiexec –n 4 pmandel_fence”. Then connect to this engine using the socket option in the visualizer. This engine uses the MPI-2 functions, MPI_Fence, MPI_Put, and MPI_Get to communicate the data between the root process and rest of the processes.