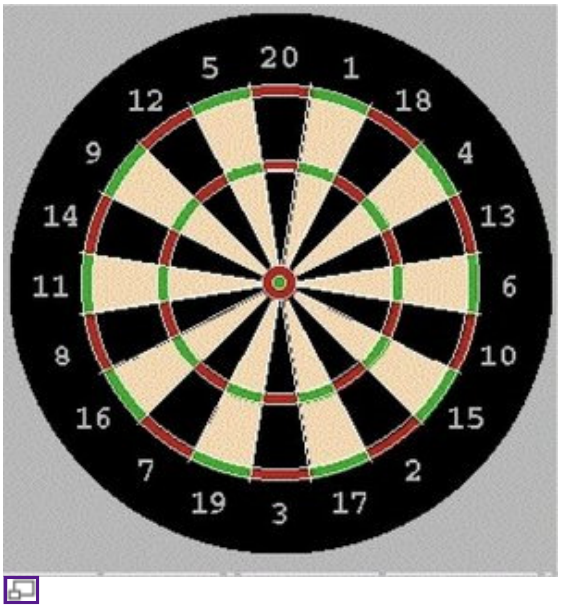
Parameter Space

From BCCD 2.2

Jump to: navigation, search

This tutorial will walk you through an investigation of parallel computing using a parameter space example. For instructions on how to compile and run MPI programs, please see Compiling and Running. This tutorial assumes that you have booted up the BCCD and have X running (to do this, enter `startx`). It uses mpirun, which means you need an implementation of MPI installed; the BCCD comes with both LAM MPI and MPICH but with MPICH running by default. For more information, see Running MPICH and Running LAM MPI).

Parameter Space Example: Scoring Well in 301



The parameter space model is a study of the dart game "301" In the game of 301, you start at 301 points and work your way down to zero. The board is divided into 20 wedges, with the bull's-eye in the middle worth 25 points. The ring around the outer edge is a double score, the inner ring is a triple score, and the exact center of the bull's-eye is a double score. You get three darts per turn, and except for the beginning and end of the game (which require hitting the "double" ring) the goal is to score as high as possible.

The origin of this model began in the dart league at Eammon's in Loudonville New York. It was a friendly league, with lots of beginners, and lots of free advice for beginners. One piece of advice I heard given to many players was that if you missed a lot, then you should aim at the 1, as when you miss you will hit either the 20 or the 18. I wasn't so sure about this advice, so I made the following assumption. "Typical dart throws will have a random direction from some 'aim point' and a normal distance from some 'aim point.'"

I set up a Monte Carlo model to test each spot on the board to determine the average score of a three dart throw if I aimed at that point, and ran it with different accuracy levels, where the accuracy level was defined as the standard deviation of the normal distribution in distance from the "aim point."

And then I waited. Monte Carlo models work by running random events multiple times and averaging the results. If you want high accuracy, you may have to run the model many times.

The code here is a reproduction of that model, designed to allow the user to visualize the solution as it progresses. It is

also designed to break the model up into pieces that can be solved in parallel by different computers.

To run the program, first move into the Param_space directory by executing `cd ~/Param_space`. Next the executable needs to be "made" by running `make`. This will create the executable `Param_space`. (If you're using the most recent version of the BCCD, this will show up as a green-colored file, meaning it's executable.)

Next we need to copy this executable to all the nodes that will be running it (If you have not yet set up your nodes for remote access, make sure you have logged into each machine as bccd, started the heartbeat (pkbcast) program, run `bccd-allowall`, and run `bccd-snarfhosts`. You must do this before continuing.) A new automated script now exists to successfully copy executables across BCCD nodes without compromising other user's runs. It is called 'bccd-syncdir', and is run with the following command:

```
bccd-syncdir ~/Param_space ~/machines
```

where `~/Param_space` is the directory which holds the executable, and `~/machines` is the machinefile created previously with 'bccd-snarfhosts', which contains a list of all the nodes in your cluster. This creates a unique directory in /tmp which holds your executable directory across all nodes. The name of this directory is unique and chosen by the first 8 characters of your current host's public key. Make note of this directory and move into with `cd <your directory>`.

Now run it using the mpirun command. To run the program on 1 processor with a grid of 100 radial values and 100 angular values, averaging 200 times and displaying the results on the screen as a graph while keeping track of the running time, type the following:

```
time mpirun -np 1 -machinefile ~/machines ./Param_space     101           101       200
             #cpus                                       #radial+1    #angular+1 #averages
```

Run the model with 1 processor, and then increase the number of processors. What happens to the running time as you increase the number of processors? Is this what you expected?

(efficiency can be measured in many ways, but typically can be expressed by taking the running time with 1 processor, and dividing it by the running time with P processors *P)

```
efficiency = time(1)/(time(P)*P)
```

Increase the size of the model by either using more grid points or averaging more times. What happens to the efficiency?

Does this model allow you to answer the question of where to aim on the board?

Retrieved from "http://bccd.net/ver2_2/wiki/index.php/Parameter_Space"

- About BCCD
- Recent changes
- Random page
- Help

Search

Go Search

Toolbox

- What links here
- Related changes
- Special pages
- Printable version
- Permanent link