Excursions into Parallel Programming

By Michael Chen

November 1, 2007

TJHSST Computer Systems Lab 2007-2008

Abstract:

With more and more computationally-intense problems appearing through the fields of math, science, and technology, a need for better processing power is needed in computers. MPI (Message Passing Interface) is one of the most crucial and effective ways of programming in parallel, and despite its difficulty to use at times, it has remained the *de facto* standard. But technology has been advancing, and new MPI libraries have been created to keep up with the capabilities of supercomputers. Efficiency has become the new keyword in finding the number of computers to use, as well as the latency when passing messages. The purpose of this project is to explore some of these methods of optimization in specific cases like the game of life problem and heat dissipation.

Introduction:

MPI is adaptable and can solve a variety of problems
-Divide and Conquer, Pipeline, Synchronous
-High computational power, used in every field for intense calculations (AI, molecular biology, ecosystems)
-Expansions in library to adjust for use with supercomputers
-Efficiency becoming an issue: Latency v. Processing Power

Procedures and Methodology:

-Uses C, C++, and Fortran
-Flexible, not restricted to certain capabilities
-Start with non-mpi code, then convert
-Works very case by case, no general testing program
-Optimization of code depends on a computer's specific latency and its processing power. Depending on which works better, the number of computers best used as well as amount of passing in code used changes.

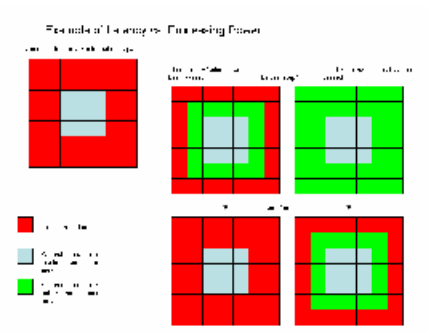
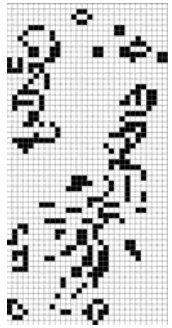
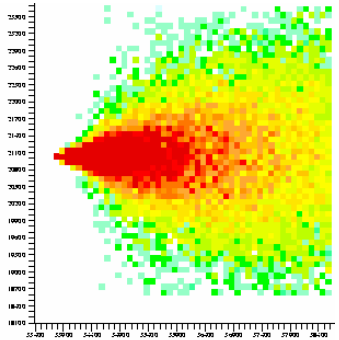


Diagram of latency vs. processing





Game of life and similar problems (heat dispersion)

Results and Conclusions:

-Many real-life applications
-blocking vs. non-blocking checkpointing
-supercomputing
-With wind velocity program, optimum number of computers is eight, though this is a simple example.
-With the game of life, more factors will have to be considered that just number of computers: how often to pass information, and how much information to pass.
-Moving the program from theoretical to the practical
-Latency algorithms for testing possible

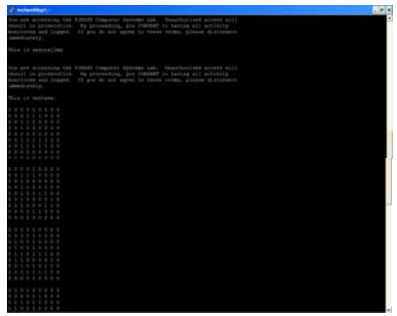
Sample Code and Test Runs:

```
while (py>=0) //as long as the projectile has height
  {
      //recalculates acceleration and accounts for it
      ax = (vx-vw)*(vx-vw)*rf;
      //acceleration of wind depends on direction of wind
      //with respect to the direction of the projectile
      if (vw<vx)
          ax=-ax;
      //checks if wind velocity is working with/against gravity
      if (vy>0.0)
          ay=-g-rf*vy*vy;
      else
          ay=-g+rf*vy*vy;
      vx+=(ax*dt);
      vy+=(ay*dt);
      px+=vx;
      py+=vy;
  }
  //sets value as the ending location of projectile and sends it
  args[2]=px;
  MPI_Send(args,3,MPI_DOUBLE,0,tag,MPI_COMM_WORLD);
                                                          }
```

Sample code from child process in wind velocity lab.

```
//all processors make a move (only in their area)
step(xa,ya,xymax,size);
//corners are not covered, need to swap around the pass statements
if (xs>0)
    MPI_Send(arr,max*max,MPI_INT,rank-1,tag,MPI_COMM_WORLD);
if (xs<sqrt(size)-1)
    MPI_Send(arr,max*max,MPI_INT,rank+1,tag,MPI_COMM_WORLD);
if (ys>0)
    MPI_Send(arr,max*max,MPI_INT,rank-
sqrt(size),tag,MPI_COMM_WORLD);
if (ys<sqrt(size)-1)
 MPI_Send(arr,max*max,MPI_INT,rank+sqrt(size),tag,MPI_COMM_WORLD);
//after sending, all computers waiting for other processes, then processes
them
```

Sample code from Game of Life lab



Sample test run from Game of Life Lab