Excursions into Parallel Programming

By Michael Chen

November 1, 2007

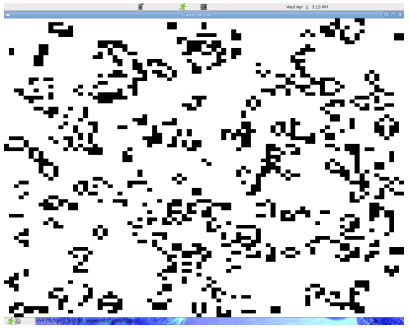
TJHSST Computer Systems Lab 2007-2008

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.

Introduction:

-MPI has 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 with latency v. processing Power: where is the middle point?



Running simulation of game of life with 9 processors

Sample Code:

```
void move()
{
if (xs>0)
  MPI_Send(arr,arrsize*arrsize,MPI_INT,rank-1,tag,MPI_COMM_WORLD);
if (xs<sqrt(size)-1)
  MPI_Send(arr,arrsize*arrsize,MPI_INT,rank+1,tag,MPI_COMM_WORLD);
if (ys>0)
  MPI_Send(arr,arrsize*arrsize,MPI_INT,rank-
sqrt(size),tag,MPI_COMM_WORLD);
if (ys<sqrt(size)-1)
MPI_Send(arr,arrsize*arrsize,MPI_INT,rank
+sqrt(size),tag,MPI_COMM_WORLD);

//after sending, all computers waiting for other processes
if (xs>0)
{

MPI_Recv(rec,arrsize*arrsize,MPI_INT,rank-1,tag,MPI_COMM_WORLD,&st
atus);
for (x=amount;x>=1;x--)
  for (y=ya;y<ya+xymax;y++)
    arr[xa-amount][y]=rec[xa-amount][y];
}
if (xs<sqrt(size)-1)
{
MPI_Recv(rec,arrsize*arrsize,MPI_INT,rank
+1,tag,MPI_COMM_WORLD,&status);
for(x=amount;x>=1;x--)
  for(y=ya;y<ya+xymax;y++)
arr[xa+xymax+amount-1][y]=rec[xa+xymax+amount-1][y];
}
if (ys>0)
{
  MPI_Recv(rec,arrsize*arrsize,MPI_INT,rank-
sqrt(size),tag,MPI_COMM_WORLD,&status);
for (y=amount;y>=1;y--)
  for (x=xa;x<xa+xymax;x++)
    arr[x][ya-amount]=rec[x][ya-amount];
}
if (ys<sqrt(size)-1)
{
MPI_Recv(rec,arrsize*arrsize,MPI_INT,rank
+sqrt(size),tag,MPI_COMM_WORLD,&status);
for (y=amount;y>=1;y--)
for (x=xa;x<xa+xymax;x++)
  arr[x][ya+xymax+amount-1]=rec[x][ya+xymax+amount-1];
}
}
```

Portion of move() method from the Game of Life

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.

Results and Conclusions:

-Many real-life applications
  -blocking vs. non-blocking checkpointing
  -supercomputing
-With the game of life, more factors will have to be considered than 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

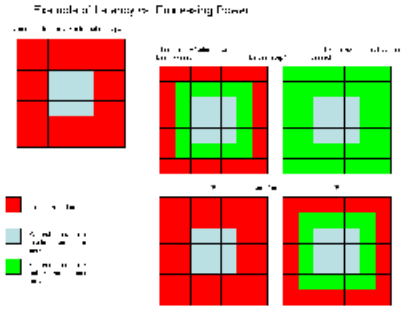


Diagram of latency vs. processing