Excursions into Parallel Programming with MPI

Michael Chen

November 2, 2007

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. The solution can be found through parallel processing, the act of linking together multiple computers and cutting run-time by using all of them efficiently. MPI (Message Passing Interface) is one of the most crucial and effective ways of programming in parallel, despite its difficulty to use at times. However, there has been no effective alternate to this date, and continues as the de facto standard of parallel programming. By exploring this rapidly-expanding field, novel approaches can be found to the most complex of problems, even modeling climate change. **Keywords:** Message passing interface

1 Introduction

1.1 Scope of Study

MPI involves many different aspects and a variety of ways to solve problems. Because of its adaptability, it is able to match the type of problem the programmer needs to solve. For example, this could be a pipeline method to solve problems that are sequential in nature, or a divide and conquer method to filter through and divide a large task among multiple processors.

1.2 Expected results

Parallel programming is a field that will have much use in the future. As a programmer, it is important to reach into bodies of knowledge that will become important in time. Though at the time, I can only produce embarrassingly parallel programs, a thorough understanding of MPI will allow me to eventually approach ”grand challenge” problems.

Many hopes for the future lie in this field. Molecular biology, strong artificial intelligence, and ecosystem simulations are just a few of the multitude of applications which will surely require parallel computing. Though the expected results may not come in a short-term scale, MPI is an essential skill set that could very well pave the way for numerous advances in science, mathematics, and engineering.

2 Background and review of current literature and research

Parallel programming is the concept that multiple processors can be used to split up a task and then combine the separate parts to enhance processing speed. This has been explained in many books such as Introduction to Parallel Programming as well as Parallel Programming in MPI. Parallel programming is most effectively used for projects that require high computational power. Project ideas that have been explored include taking images taken from aircraft and placing them on a map in terms of longitude and latitude, (or from the DARPA Grand Challenge) symbolic computations for recognizing speech and facial features.

3 Development

3.1 Requirements and Limitations, Overview, Development plan

The technology demand of MPI will be no problem to meet, since at TJHSST, all the computers in the Systems Research Lab are compatible with MPI, and have enough processing power to suffice for any computational power I will be using. Finally, a thought to consider is an implementation of a graphical interface to view the parallel programming and the processes it is going through more easily, since MPI can and often does become convoluted, difficult to write, and difficult to debug in more advanced stages.

3.2 Research Theory

Although I will start with programs like embarrassingly parallel computations, throughout the course of the year, I will move through different aspects of coding including pipeline computation, divide and conquer algorithms, and check-pointing. Though my knowledge is still vague in these areas, they are areas with great amounts of reference material that I can access. As I work my way through different examples, I will try to eventually reach a stage where I can consider ?grand challenge? problems.

3.3 Testing and analysis

Because of the nature of MPI, it is not restricted to certain capabilities, though it does excel at some. Thus, a variety of possible displays and processes can be run, and depending on the specifics of each programming, different debugging and error analyses are required. However, random inputs should be adequate for most cases, or user-defined inputs for specific cases that have problems. MPI itself is fairly standard, so there will always be controls to compare my results to.

4 Results, Discussion, Conclusion, Recommendations

4.1 Expected Results

Once again, I am hoping to learn about parallel programming more through MPI, a field that will become more important in the future. Even now, there are many research teams comparing and contrasting different ways of using MPI. For example, one research I read was exploring the possibility of a graphical interface called MPI-Delphi for workstation networks that allows quick and easy access for programmers. This is essential in complicated tasks known as "grand challenges." Even at the professional level, I read about testing done on blocking v. non-blocking coordinate checkpointing, another method of MPI. So whether or not the research I do yields a substantial result this year, or if I find a direction to follow, the knowledge and skills I obtain will become indispensable in the future.

4.2 Results

WILL BE ADDED

5 Appendices

Sample Code from Wind Velocity Lab:

```
for(angle=0.0;angle<=maxangle;angle+=0.5) //parent process
for (wvel=-50.0;wvel<=wmax;wvel+=0.1)
{
//if all child processes are busy, wait to receive info
if (flag>=size)
{
MPI_Recv(args,3,MPI_DOUBLE,k,tag,MPI_COMM_WORLD,&status);
printf("receiving! endx:%3.2f\n", args[2]);
flag--;
}
//after receiving, send info to the same process
args[0]=wvel;
args[1]=angle;
args[2]=1337.0;
MPI_Send(args,3,MPI_DOUBLE,k,tag,MPI_COMM_WORLD);
//move to the next process and go around the for loop
k++;
flag++; if (k>=size)
k=1;
//making sure not to exceed number of computers available
printf("sending! angle:%3.2f,wvel:%3.2f\n",angle,wvel);
}
//this section of code makes sure that all values have been received
int recall=0;
int ori=k-1;
if (ori<1)
```

```
ori=size;
while (recall==0)
{
if (size==2)
{
MPI_Recv(args,3,MPI_DOUBLE,k,tag,MPI_COMM_WORLD,&status);
printf("%3.2f\n",args[2]);
break;
}
MPI_Recv(args,3,MPI_DOUBLE,k,tag,MPI_COMM_WORLD,&status);
printf("%3.2f\n",args[2]);
k++;
if (k>=size)
k=1;
if (k==ori)
recall=1;
}
//this portion of code ends all child processes
args[0]=0.0;
args[1]=0.0;
args[2]=666.0;
int n;
for (n=1;n<size;n++)
MPI_Send(args,3,MPI_DOUBLE,n,tag,MPI_COMM_WORLD);

--------------------------------------------------------------------------------

MPI_Recv(args,3,MPI_DOUBLE,0,tag,MPI_COMM_WORLD,&status);
double vx = v0*cos(args[1]*M_PI/180);
double vy = v0*sin(args[1]*M_PI/180);
double vw = -args[0];
//checks to see if parent process ends this process
if (args[2]==666.0)
break;
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);
```

MORE WILL BE ADDED

6 Literature Cited

P. Pacheco, Parallel Programming with MPI, M. Kauffman: San Francisco, California, 1997.

MORE WILL BE ADDED

7 Acknowledgements

Special thanks to Mr. Latimer for help throughout the year

Also thanks to Mr. Torbert for help with MPI with programs like wind velocity and Mandelbrot