Hybrid MPI/OpenMP Parallel Programs

Students: Mark Randles
Supervisor: Dr. H. Rajaei
Dept. of Computer Science
Bowling Green State University, Ohio
Spring 2004

Abstract

There are many ways to parallelize a problem in computer science. One of the more fascinating ways to parallelize a problem is a hybrid or multilevel parallelization using two or more parallel systems. The purpose of this paper is to explain and explore a hybrid parallelization of two problems in computer science. One problem can benefit from the hybrid parallelization and the other benefits, but it is not a good fit.

Table of Contents

1.Introduction

1.1 Purpose

The purpose of this project is to study and develop programs using a hybrid of common high performance parallel libraries and systems, in particular OpenMP and MPI.

1.2 Goals and Objectives

To develop a program or suite of programs that will be used to measure the performance between MPI, OpenMP, and their hybrid implementation.

1.3 Background

MPI and OpenMP each are the excellent systems to write parallel systems, however each requires a complimentary partitioning scheme and different systems. However under certain situations a hybrid of their respective methods is a much better fit, the either one or the other.

From the available research, the best solutions for hybridizing are thus that either have two levels of vectorization or thus that can be divided into both course and fine granularity. The generally accepted method of using the two systems together is to do a course partition using MPI and then a fine partition using OpenMP threads. This was the method I used for solving the problems.

2.Project Phases

The phases of this project are outlined below.

Week 1: Research. Find methods of hybridizing the separate systems.

Week 2: Development. Develop a problem and begin to write the code to test the various methods of parallelization.

Week 3: Con't Development. Finish the code development and begin testing.

Week 4: Testing & Writing. Finish program testing and write the report.

3.Project Details

There are a variety of problems that can be parallelized with the hybrid method. For this project i picked two problems. One is a very good example of proper parallelization with the hybrid methods. The first problem picked was a previous problem during the semester, very large matrix multiplication. The second problem picked was to simulate and explosion of a distinct set of particles. Their ballistic motion would be tracked in 3 dimensions.

The first problem was quite easy to parallelize. Both a MPI and a OpenMP version of the program had already been written, so the development of a hybrid program was quite easy.

The second problem too much longer to develop a solution for. After the problem was defined, a single processor solution was developed. From that a OpenMP solution was developed and tested. The MPI solution was very different from the single processor and OpenMP code, so it took a bit of time to write. Also it was discovered that this problem was a bit difficult to parallelize as it relied on having an easy and efficient way to output it's data. After the MPI solution was satisfactorily written, it was parallelized with OpenMP code. After the MPI and hybrid programs were done they each were tested for two different data sets.

Each program was tested on the Itanium2 cluster at OSC, all OpenMP implementations were tested using the Altix 3000 attached to that system. The result times were derived from the Unix time output of each program and recored appropriately. The speedup was calculated using a single processor time derived from a serial job on that same cluster.

4.Results and Analysis

4.1Matrix Results and Analysis

(Data and charts in Appendix A)

The results retrieved from the various runs of this program are to be expected. Before a detailed analysis is performed one must understand the dataset. The hybrid solution was run for 3 to 31 node, each processor allowing for a max of 2 OpenMP threads. The MPI solution was run for 3 to 17 node, with a max of one processor per node. The OpenMP solution was run for 3 to 15 OpenMP threads. The reasons for the various numbers is thus, on the MPI cluster there are 64 parallel nodes, of which it's easy to get at most 31 at any one time. On the OpenMP Altix 3000 there are 32 processors, of which it's easy to get 15 at any one time. The MPI numbers were gather from my original run in Lab 1 which i used a max of 17 nodes.

As you can see from the graph, the OpenMP solution is the slowest. This is caused by the architecture to allow for SMP on the Altix 3000 machine. essentially each processors is a separate computer which shares it memory across a network interface. This is no where near as fast as a true SMP machine. However for small datasets the performance loss is negligible compared to the ease of parallelization.

For the MPI and the hybrid solution the differences in performance is negiliable. The hybrid solution is in fact faster then the MPI solution. However they are very close. This is mostly caused because the overhead of the OpenMP threads spawned in the hybrid solution nearly negate any gains from the parallel processing they do. If the dataset was doubled the effect of the OpenMP would be much more apparent.

The final verdict is that the matrix multiplication problem is the ideal problem for parallelization and to benefit from OpenMP/MPI hybridizing.

4.2Explosion Results and Analysis

(Data and charts in Appendix B)

This was a much harder problem to parallelize. A big part of it was trying to get the data back from the children processors. Each particle in the explosion was

held in a structure with it's position, velocity, and acceleration stored as vectors, and the particles mass stored for future use. However MPI had difficulty passing these structures. Also the dataset that was required to process was quite large, and the children nodes had trouble passing that voluminous amount of data back to the parent in an efficient manner.

Only the preliminary OpenMP performance data is available at the writing of this report. The job that would provide me with the final data has been queue for quite a bit of time, and isn't set to run for a few days yet. However from the preliminary results, it appears that like in the matrix multiplication problem, the OpenMP solution would be slower then the MPI and Hybrid solutions.

For the MPI and hybrid solutions, a different approach was used for the course partitioning. The problem called for the position to be calculated for a set number of “frames”. For the OpenMP solution the frames loop was not parallelized, but rather the inner loops parallelized. This was probably not the most optimal solution, but it was the solution that best fit the OpenMP model of fork-and-join. However MPI is much better suited for a course parallelization, so the frames loop was divided up between the nodes instead of the particle list. This also cut down on the communication while the processing was preceding, as all the processes could sync their datasets after all the processing was done. However this created quite the memory burden on each node, as the node must store the entire particle set for each frame.

As a cheat to keep the memory usage down on the parent node, and to keep the network traffic to a minimum, each child node was to output it's own data in sequence, the sequence was determined by the parent, which sent a flag to each child which triggered the data dump, and after the data dump the child flagged the parent to continue. This was to insure that only one node would output it's data at the same time. However it was discovered that this was dog slow, at least 10x slower the actually doing the computation, so for the purposes of actual performance numbers, the data output was suspended.

The actual hybrid solution to the problem used nearly an exact hybrid of the two approaches. There was the same

course partitioning as was used in the MPI solution, along with the fine partitioning that was used in the OpenMP solution. This lended to a very odd speedup curve. They hybrid solution was slower the the MPI solution! The exact reason for this is not know at this time. The times themselves are very close together, however the hybrid solution is a decided factor slower when the speedup is calculated. However the curve for the hybrid solution is much smoother then that of the MPI solution, so it can be inferred that the slowdown is not related to the MPI network code, but there must be some sort of slowdown in the OpenMP vectorized code.

After all the data was gathered, it becomes apparent that this problem benefits from parallelization, but is a very bad candidate for fine parallelization either through the hybrid approach or through vanilla OpenMP parallelization. It might be possible to reorganize this problem so it would benefit from a fine parallelization, but i think that it would be a very specific subset of this problem that would benefit.

5.Proposal Of Future Work

For future work on this problem it would be interesting to see some of the solutions to other problems. It would also be interesting to modify the MPI code for the explosion simulation, so that there were more environmental factors tallied into the physics, such as collision detection, uneven terrain, angular momentum, etc. This would most likely change the dynamic of the partitioning as, it would easier to divide the dataset into static partitions and have each child compute the approaite physics, using OpenMP threads to divide the task down further after the course partition is sent to the child.

6.Concluding Remarks

This was a unique problem to work on, in addition to working on OpenMP and MPI implementations of the same problems. Some interesting challenges were laid forth in both partitioning and efficiently gathering data from the children. Also in the world of high performance computation this will be an exciting area of study, as SMP systems become more economically feasible to use in clusters, as well as provide an efficient means of speeding problems without the associated cost of more hardware nodes or the computation cost of network latency.

7.References

(I realize this is not a proper citation. However it is very difficult to cite specific sources, as many are PDF files.)

Ohio Supercomputer Center. *Using the Linux/Athlon Cluster at OSC.* Date Unknown.

Ohio Supercomputer Center. *Using the Pentium 4 Cluster at OSC.* Date Unknown.

Ohio Supercomputer Center. *Using the Itanium2 Cluster (McKinley) Cluster at OSC.* Date Unknown.

Ohio Supercomputer Center. *Multilevel Parallel Programming.* Date Unknown.

Ohio Supercomputer Center. *Intermediate MPI: A Practical Approach for Programmers (Day 4).* Date Unknown.

All Ohio Supercomputer Center (OSC) sources are available through http://oscinfo.osc.edu.