The Department of Electrical and Computer Engineering
The University of Alabama in Huntsville
Huntsville, AL 35899

CPE 434/534 Operating System Project
Performance Analysis of Distributed Applications on Local Machines / Alabama Supercomputer

Objective

The era of "big data" brings the necessities of processing the large scale data sets and distributed data sets. To this end, adopting applications in distributed and parallel computing is rapidly growing. Many parallelism techniques are used, such as threads, MPI, OpenMP and MapReduce, yielding different performance and usability characteristics. In this project, student will implement a simple sorting algorithm in a distributed and parallel manner to process a huge amount of data. Here, we choose **Mergesort** and a simple illustration is described in Fig.1.

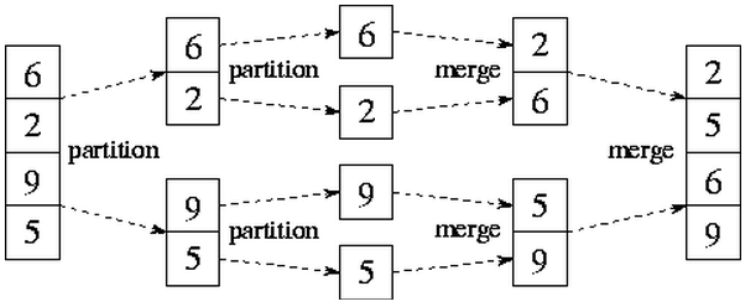

Fig. 1 A simple merge sort with the sequence [6,2,9,5].

Assignment

1. Undergraduate student (CPE434), you are required to write and implement **Two** programs. 1) Serial implementation of merge sort. 2) Distributed parallel implementation of merge sort using **pthreads** on local machines in EB246. You need to implement it using twenty EB246 local machines.
   Performance analysis includes comparison between serial implementation versus parallel implementations on local machines.

2. Graduate student (CPE534), you are required to write and implement **Three** programs. 1) Serial implementation of merge sort. 2) Distributed parallel implementation of merge sort using **pthreads** on local machines in EB246. 3) Distributed parallel implementation of merge sort using either **MPI or OpenMP** on Alabama Supercomputer (ASC).
   Performance analysis include: 1) comparison between serial implementation versus parallel implementations on local machines, 2) comparison between serial implementation versus parallel implementations on ASC.

Requirements

1. Use a random number generator to obtain a measurable data file as input file for sorting. You should vary the data size for analysis. The data size is varied from 50,000,000 to 140,000,000.
2. Distributed merge sort will work as follows. One central server distributes equal pieces of the data to 20 client machines. Server will start by reading in the full list of integers into an array. Then, the array is divided into twenty equal parts. The twenty equal parts are distributed out, one part per machine, and each

machine performs mergesort on the integers that it has received. After that, you should send back your results to the server and gather the final results.
3. For local machine running, you are required to use twenty EB246 machines which have 8 cores for sorting. Record the runtime for the program.
4. Performance analysis details: Vary the data sizes and draw a graph about the data size versus run time. Record the results for serial implementation and distributed parallel implementation in the report.

Deliverables

1. Source codes for serial Mergesort and distributed parallel Mergesort.
2. Project Report (includes all the graphs, explanations of implementations and comments).

Demo Date

Dec 1st, 2:30pm – 4:30 pm (EB216).
Dec 2nd, 9:30am – 12:00 pm (EB216).
If the above time slots do not work for you, please appoint another time by email.

Policy

Typically, it is an individual project. However, you can also finish it in a group with maximum of two people. In this case, each student in the group should explicitly state your individual contributions to the project during demo. All members in a group must show up during demo at the same time, fail to be present will result in a 50% penalty.

Guideline to Run Distributed Application Across multiple nodes on ASC

1) Compile your program on the dmc login node.
2) Write a script to load the same modules you used to compile then run your job.
3) Submit the job to the queue with the "run_script_mpi" command. When it prompts for where to run the job, you type in "dmc". Check the examples in the directory /opt/asn/doc/mpi and table 4.1 from reference [1].

Reference

1. Alabama Supercomputer User Manual: http://www.asc.edu/html/man.pdf
2. MPI Tutorial: http://booksite.elsevier.com/9780123742605/LS/Chapter_3.ppt
3. Pthreads Tutorial: http://booksite.elsevier.com/9780123742605/LS/Chapter_4.ppt
4. OpenMP Tutorial: http://booksite.elsevier.com/9780123742605/LS/Chapter_5.ppt

Due Date

Dec 2nd , 11:55pm, 2014, via angel dropbox.

Alabama Supercomputer Authority Temporary Class Account Information

We have made an account for you on the machines at the Alabama Supercomputer Center. These systems are an SGI Ultraviolet, and a locally architected fat node cluster called a Dense Memory Cluster (DMC).
Below is some information for new users of these machines. You can login, following the directions below, using the login and password provided by your instructor.

WARNING: This is a temporary account. The account and all the files in it will be deleted at the end of the semester.

The best source of information on using these computers is the HPC User Manual, which can be accessed on the web at http://www.asc.edu/html/man.pdf

The command ?ascdocs? shows a menu drive listing of locally written documentation. It displays text files, and gives paths for you to copy or download other types of files.

The computers can be accessed using secure shell (an encrypted form of telnet). Secure shell is installed on many Linux and Unix machines, and can be called with a command like this

```
ssh UserID@uv.asc.edu
ssh UserID@dmc.asc.edu
```

The first time you login, you will have to enter the password a second time, then set a new password.

Files can be copied to and from the DMC using secure copy "scp" or secure ftp "sftp". These programs are often included with secure shell software.

You can get help from the Alabama Supercomputer Center staff at hpc@asc.edu or (256) 971-7448 or (800) 338-8320. The staff can help with using the applications on the computers, accessing the system, etc.

In order to get the maximum possible utilization of the computers, computational jobs are executed through the Moab job queueing system. Moab can be accessed via PBS commands like "qstat" to check queue status.

Scripts are provided for more easily running jobs using many of the third party software packages. Most of these scripts are in the /apps/scripts directory and have names starting with "run". For example, a Gaussian 09 job can be submitted to a queue by typing

```
rung09 input_file_name
```

The queue script will prompt you for the name of the queue and any limits you may wish to set on time, memory or file size. It then prompts for a date and time to run the job, job name, and output directory.
All prompts can be satisfied by pressing "Enter" to give default values of the queue limits, immediate submittal, a default job name, and results being returned to the current directory. Only class accounts can submit jobs to the class queue.

Accounts have a disk quota limit. To see your disk quota, type "quota". Queue limits and interactive use limits can be seen by typing "qlimits".

Please contact the Alabama Supercomputer Center staff if you have any other questions or concerns about using the supercomputers.

David Young, Ph.D. Alabama Supercomputer Center
HPC Computational Specialist, CSC Alabama Research and Education Network
dyoung@asc.edu http://www.asc.edu/
(256) 971-7434 FAX: (256) 971-7491

```
/*
MPI Example - Hello World - C++ Version
FILE: hello_world_MPI.cpp

Compilation on uv.asc.edu

first set up environment by typing from the command line the
following two module load commands
   module load intel
   module load mpt


   to compile the program type

   icc hello_world_MPI.cpp -o hello_world_MPI -lmpi++ -lmpi

    to run on eight processors type

     mpirun -np 8 ./hello_world_MPI
*/

using namespace std;
#include <iostream>
#include <mpi.h>

int main (int argc, char *argv[]) {

   MPI_Status status;
   int nmtsks, rank;

   MPI_Init(&argc,&argv); // initalize MPI environment
   MPI_Comm_size(MPI_COMM_WORLD,&nmtsks); //get total number of processes
   MPI_Comm_rank(MPI_COMM_WORLD,&rank); // get process identity number

   cout << "Hello World from MPI Process #" << rank << endl;

  /* Only root MPI process does this */
  if (rank == 0) {
     cout << "Number of MPI Processes = " << nmtsks << endl;
  }

  /* Terminate MPI Program -- clear out all buffers */
  MPI_Finalize();

}
```

```
/*
  Pthreads Example - Hello World - C/C++ Version
File: hello_world_PTH.cpp

  first set up environment by typing from the command line the
  following two module load commands
   module load intel
   module load mpt

  Compilation on uv.asc.edu
     to compile the program
         icc hello_world_PTH.cpp -o hello_world_PTH -lpthread
     to run type
       ./hello_world_PTH
*/

using namespace std;
#include <iostream>

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#define NTHREADS 8 /* set number of threads to 8 */

pthread_mutex_t MUTEX; // globaly defined lock variable used to insure
                       // uninterrupted print operations from each thread

// This is the worker thread code
void *hello(void * arg) {
   int myid=*(int *) arg;

   // enter critical region
   pthread_mutex_lock(&MUTEX);

   cout << "Hello World, from PThread " << myid << endl;

   // exit critical region
   pthread_mutex_unlock(&MUTEX);

  return arg;
}
// this is the main thread's code -- it spawns the worker threads and
// then waits for all the worker threads to return before it exits
int main(int argc,char *argv[])
{
  int tid;                       // tid thread number
  pthread_t threads[NTHREADS];   // holds various thread info
  int ids[NTHREADS];             // holds thread args
  int errcode;                   // pthread error code

   // initialize mutex variable -- this variable is used to insure that
   // all couts are automic meaning that they are not interrupted
   pthread_mutex_init(&MUTEX,NULL);
```

```
   /* create the threads */
   for (tid=0; tid<NTHREADS; tid++) {
      ids[tid]=tid;
      errcode=pthread_create(
              &threads[tid],// thread information structure
              NULL,         // thread attributes -- NULL means assume defaults
              hello,        // function name that is to represent thread
              &ids[tid]);  // pthread created thread id for the created thread
      // check for error during thread creation
      if (errcode) {
         cerr << "Pthread Creation Error: " << strerror(errcode) << endl;
         exit(1);
      }
   }

   // print out number of threads
   pthread_mutex_lock(&MUTEX); // enter critical region
   cout << "Number of threads = " << NTHREADS << endl;
   pthread_mutex_unlock(&MUTEX); // exit critical region

   // wait for all threads to return before exiting main program (process)
   for (tid=0; tid<NTHREADS; tid++) {
      // wait for each thread to terminate
      errcode=pthread_join(
              threads[tid], //thread scheduler id information of selected thread
              NULL);        //thread return value -- not used in this case
      if (errcode) {
         cerr << "Pthread Join Error: " << strerror(errcode) << endl;
         exit(1);
      }
   }
   return(0);
}
```

```
/*
openMP Example - Hello World - C/C++ Version
FILE: hello_world_OMP.cpp

  first set up environment by typing from the command line the
  following two module load commands
   module load intel
   module load mpt

Compilation on uv.asc.edu

   to set number of threads through environment variable in bash shell
   (have to do this before you run the program)
   export OMP_NUM_THREADS=8

   to compile the program

      icc -o hello_world_OMP -openmp hello_world_OMP.cpp

   to run -- set number of threads as explained above and then type

     ./hello_world_OMP

*/

using namespace std;
#include <iostream>
#include <omp.h>

int main (int argc, char *argv[]) {

int nthreads, tid;

   // Fork a team of threads giving them their own copies of variables
   #pragma omp parallel private(nthreads, tid)
   {
      // Obtain thread number
      tid = omp_get_thread_num();

      // print out Hello World Message -- do this as a noninterruptable op
      #pragma omp critical
      {
         cout << "Hello World from thread = " << tid << endl;
      }
      // get number of threads
      nthreads = omp_get_num_threads();

      // print out No of Threads Message -- do this as a noninterruptable op
      #pragma omp critical
      {
         // Only master thread does this
         if (tid == 0) {
            cout << "Number of Threads = "  << nthreads << endl;
         }
      }
   }  // All threads join master thread and disband

}
```