Final exam 4/27/2020 Parallel Programming COP6616

Open book, internet and notes.

Each question worth 2 points unless otherwise noted. Name ***Joseph Allen*** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

You have from 6 PM until 8PM to complete the exam. I must have a time stamp of 8 PM or earlier when you email the completed exam to me. Good luck and I enjoyed working with you all this term. I do not answer questions during the exam.

1. What is meant by a “loosely synchronous” paradigm when considering parallel programming using message passing?

This refers to the fact that the workers complete their work independently, but afterwards they need to be synchronized and the results need to be collected/aggregated.

1. The fundamental operations for message passing is the send and receive verbs. Identify the prototype of each of these operation.

The basic prototypes for these operations in openmpi are MPI\_Send(void\* message, int count, MPI\_Datatype datatype, int dest\_rank, int tag, MPI\_Comm comm) and MPI\_Recv(void\* message, int count, MPI\_Datatype datatype, int source\_rank, int tag, MPI\_Comm comm, MPI\_Status status)

1. Describe blocking concerns regarding send and receive operations. That is, which are typically blocking and which are not and how does one overcome the standard.

The basic MPI\_Send and MPI\_Recv operations are blocking by default, meaning that communication must complete before the programs proceed. On the contrary, MPI\_Isend and MPI\_Irecv are nonblocking and are simply requests to communicate. The program proceeds after these operations regardless of whether the communication completes or not.

1. What are the major problems when using non-buffered blocking sends?

On of the major problems with non-buffered blocking sends is that they might block forever if on of the processes fails or encounters an error and does not respond back. Another problem is that waiting on communication in general in programs introduces unnecessary latency.

1. Explain how message passing might result in deadlock.

Message passing might result in deadlock if process A is waiting to receive data from process B, which is itself waiting to receive data from A.

1. Approximately, how many verbs (instructions or function calls) are required to write a fully functional MPI program?

Four. Init, Send, Receive, and Finalize.

1. How does MPI\_Isend differ from MPI\_send?

MPI\_Isend is nonblocking while MPI\_send is blocking.

1. In general, what is the purpose of MPI\_Bcast?

The purpose is to *broadcast* data to all participating processes. For example, if performing parallel linear search with MPI, you might broadcast the search key to all processes from the master/root process.

1. Suppose process 0 has 15, process 1 has 17, process 2 has 11, process 3 has 12, process 4 has 17, and process 5 has 11. What would be returned by MPI\_MAXLOC(Value, Process)?.

(17, 1)

1. Compute the prefix sum of 1, 8, 3, 7, 2, 7, 2, 9

[1,9,12,19,21,28,30,39]

1. How many steps or instructions are required to compute the prefix sum of n numbers?

n

1. In what applications are prefix sums important?”

Prefix sums are important for the counting sort algorithm and the list ranking problem.

1. Identify the most significant differences between a thread and a process.

Threads can be thought of as “*lightweight”* processes since they can do much of what a process can while carrying less baggage. One main difference is that memory is shared between threads belonging to the same process, but processes always (ideally) have their memory all to themselves. Another difference is that typically processes are used to execute applications, but threads are used for smaller tasks.

1. How can threads support latency hiding?

Threads can support latency hiding because they are lightweight and enable fast context switches compared to processes. These fast context switches allow other work to be executed seamlessly should the thread be waiting on some message or data.

1. How do threads support load balancing?

Threads support load balancing because they can quickly be switched in to and out of execution on a processor, which allows them

1. Explain what is accomplished by the following instruction.

pthread\_create(&p\_threads[i], &attr, compute\_pi, (void \*) &hits[i])

This code creates a pthread to execute the function compute\_pi and stores the result in the hits array.

1. How does a programmer avoid false sharing?

The programmer can avoid false sharing by reducing the number of times cache is written to overall.

1. What is the concept of mutual exclusion?

Mutual exclusion is the concept of preventing simultaneous access to a shared resource.

1. How can a programmer create a body of code such that exactly one process executes it at any given moment?

The programmer can do this by using a critical section. For example, with openmp or openmpi one can define a section where only the process with rank/id 0 can execute.

1. What components are in a program that utilizes a GPU?

The main components of a program that utilizes a GPU are the ones that copy data to and from the CPU memory and the GPU memory. There are also components for the execution and synchronization of functions on data in the GPU’s memory.

1. Discuss and contrast speed up versus efficiency when considering parallel processing.

Speedup is determined by Amdahl’s law and measures the amount of execution time you save as you add more processors to a task. Efficiency however measures the impact of that speedup w.r.t. the number of processors added. Both speedup and efficiency decrease as more processors are added. When you add another processor, but only get minor speedup of 1%, you seriously decrease efficiency.

1. A compare-split operation takes (*ts+ twn/p)* . What is the n/p factor?

The n/p factor refers to the number of elements assigned to a processor.

1. What parallel computing model is GPU processing?

It is the “heterogeneous” or “hybrid” model of computing, since it is mostly used as an accelerator for compute-intensive applications.

1. Describe potential slowdowns that might (or will) occur when doing GPU processing.

One of the main potential slowdowns is the latency involved in copying all the data to and from the GPU, especially if it is connected over a network instead of locally.

1. Name 2 operations that all GPU programs must do.

1) memory allocate and 2) memory copy

1. Why does performance sometimes decline when adding more processors?

Performance can decline after a certain point since the amount of added overhead of distributing the data and initializing all the workers takes more time than was gained by executing the function in parallel in the first place. For example, splitting linear search on an array of 100 ints on 100 processors.

1. What level of decomposition occurs with fine granularity?

Nearly perfect task decomposition occurs with fine-grained parallelism. This is because the tasks are so fine-grained that they can be performed very quickly and the many tasks can be load balanced efficiently.

1. Give an example of a problem that would benefit from recursive decomposition.

The learning of complex functions in machine learning, since neural networks are a process of recursive decomposition. Alternatively, any sorting algorithm that is based on divide and conquer.

1. Why is the owner computes rule often desired?

It is often desired for its simplicity in terms of data allocation to processes/threads. Instead of having processes/threads operate on shared data, they each compute on their own partition of it.

1. Describe the difference between bandwidth and latency.

Bandwidth refers to the amount of information that can be sent over a channel in some amount of time, whereas latency refers to the delay in time for that communication to make its way across the channel.

1. Describe how superliner speed up might be achieved.

Super linear speedup might be achieved when a speedup of more than N is achieved when using N processors. This might happen due to lucky caching or in parallel backtracking algorithms.

1. How does openmp accomplish parallel execution?

Openmp accomplishes parallel execution on SMP machines using threads. From the developer perspective, it is accomplished using directives to the openmp compiler.

1. What distinguishes a NUMA computing system from other typical computing systems?

In NUMA systems the memory access time depends on the location of the memory relative to the processor (fast if in the processors memory and slower if in another processors memory). This is not the case for other typical systems, since they usually share the same memory.

1. Describe one solution to the cache coherence problem.

The directory-based solution to cache incoherence entails using a directory to hold the data being shared that also maintains the coherence between caches explicitly. When one processor caches something, the directory makes sure that change is reflected in all other caches.

1. Why is Moore’s Law important?

Moore’s Law is important because it has large economic implications for the future of the electronics and technology industries, as well as the USA and world in general. Computers are a huge part of daily life now, and a law that describes the rate at which they become more powerful is clearly important.

1. Why is Amdahl’s Law important?

Amdahl’s law is important because it tells us the speedup we can expect to gain by adding more processors to a task.

1. What is an atomic operation?

An atomic operation is an operation that executes completely independently of other threads or processes. Once started it must finish and that’s final.

1. What is the purpose of a “reduce” that might be used in MPI or openmp?

Reduce is often used to aggregate results of several processes/threads using some function such as add, min, max, etc. For example, you might want to add all the results together. You can do that using reduce.

1. What data structure is typically associated with exploratory decomposition?

A tree is typically used in exploratory decomposition, similar to classical searching algorithms.

1. Tall is the total time spent

Ts is the time spent for sequential execution.

What is the overhead for parallel execution when using p processors?

The overhead for parallel execution is (T\_all – T\_s) \* p

1. Describe how a program can be deadlocked when using message passing.

Message passing might result in deadlock if process A is waiting to receive data from process B, which is itself waiting to receive data from A. (question 5)

1. Can a program get deadlocked if non-blocking sends and receives are used?

Yes because even if the sends and receives are nonblocking, usually the point is to do something with that data. If some of the data is not received in error, it may wait forever for it.

1. Name 2 common mechanisms used to protect a critical section.

Semaphores and mutex locks

1. How does a scatter operation differ from a one-to-all broadcast?

Scatter differs from broadcast since it must be paired with a receive call as well. In other words, it requires more communication than a broadcast, which simply sends the data.

1. How can an All-to-All personalized communication be efficiently implemented?
2. We have had discussions regarding expected speed up of a variety of algorithms. Suppose I have n amount of data. I chose to use 4 processors and consequently divide the data evenly between each processor. If the algorithm runs in 32 time units sequentially and is an nlgn algorithm and perfect speed up is achieved, what is the expected execution time?

8 time units

1. In question 46, what is the efficiency?

Since perfect speedup was achieved, the efficiency is 1

1. In question 46, suppose perfect speed up is not achieved but rather it takes twice as long to execute as it would if perfect speed up was achieved. What is the efficiency in this case?

The efficiency in this case is E = S / N = (-2 / 4) = -0.5

1. Given the following code fragment consisting of 2 functions. Where is function “global” executed? Where is function main executed?

\_\_global\_\_ void mykernel(void){

// compute stuff

}

int main(void) {

mykernel<<<1,1>>>();

printf("Hello World!\n");

return 0;

}

“global” is not really a function, it is a keyword for the CUDA compiler. The associated mykernel function however is executed on the GPU. The main function is executed on the CPU as usual.

1. What is the concept of false sharing when dealing with a program executing on multiple processors?

The concept of false sharing describes how a single system in a distributed system may force another system to reload its cache needlessly after accessing a cache that contains data for both systems.