**CSE 415: Introduction to Parallel Computing**

**Spring 2021 Final Exam**

Thursday, April 29, 2021

**Important information:**

* There are 100 points plus 8 bonus points.
* This is an open book, open notes exam, conducted via Zoom. You are expected to turn on your webcam & share your screen during the entire duration of the exam. If you do not have a working webcam, please use your smartphone as a webcam. Zoom chat will be used for Q&A.
* No calculators, cell phones or any electronic devices other than your computer (and smartphone as a webcam) are allowed during the exam. You can leave an answer in fractional form, if necessary.
* Time limit: 120 minutes (7:45 am – 9:45 am). You must return your completed exam in electronic form through your gitlab repo by 9:50 am.
* THE CONTENTS OF THIS EXAM ARE INTELLECTUAL PROPERTIES OF THE INSTRUCTOR AND MSU. YOU ARE NOT ALLOWED TO SHARE THIS EXAM OR ANY PORTION OF IT WITH ANYONE ELSE IN HARDCOPY OR ELECTRONIC FORM.

**Additional information:**

* Use the space underneath a question to write your answers. If the space is not enough, you may continue on the scrap pages at the end. When you do so, please put a mark to indicate that your answer is continued on the scrap pages.
* The ordering of questions is no indication of their relative difficulties.
* Don’t just state an answer, show your work so that you can get partial credit.

|  |  |
| --- | --- |
| NAME: |  |
| **Question #** | **Points** |
| **1** | /36 |
| **2** | /14 |
| **3** | /12 |
| **4** | /10 |
| **5** | /12 |
| **6** | /10 |
| **7** | /14 |
| **Total:** | **/108** |

**THIS PAGE IS INTENTIONALLY LEFT BLANK FOR DOUBLE SIDED PRINTING.**

1. **Short Answer Questions [6x6 = 36 pts]**

Provide brief (3-4 sentence long) answers to each question.

1. The L1 cache is smaller than the L2 cache, and if there is an L3, the L2 is smaller than the L3. Give a practical and a theoretical reason why this is so.

The practical reason in which the L1 cache is smaller than the L2 cache often times is due to the L1 cache being designed to maximize hit rate whereas the L2 cache is meant to minimize the miss penalty when an L1 miss happens. Also, the L1 cache is the smallest in physical size which reflects in the Mbytes. However, it is possible to simultaneously reference both the L1 and L2 cache meaning the search in the L2 cachce can be ran in parallel with that for the L1 cache.

1. A non-uniform memory architecture (NUMA) is better than a uniform memory architecture (UMA) from a performance point of view. Hence, modern high performance computers typically rely on a NUMA design. True or False? Explain.

NUMA is used typically in modern high performance computers for a few reasons, one of the biggest being the overall speed increase. NUMA improves performance over a single shared memory. NUMA is successful because a processor can access its own memory faster than its non-local memory.

1. Compare and contrast functional and data parallelism. Give a specific application or programming model example of each paradigm.

Functional and Data parallelism are both similar in the fashion that they are both simultaneous execution on multiple cores. The difference comes in where functional is of MANY different functions across the same OR different datasets where asynchronous computation is performed. Whereas Data parallelism is the SAME function across the elements of a dataset with synchronous computation. An example of data parallelism would be summing the contents of an array the size of N. One thread may sum the elements [N/2-1] while another thread on a different core will sum the elements [N/2]…[N-1], running in parallel.

1. Describe Amdahl’s Law and state its implication for parallel speedup.

Amdahl’s Law can be applied to estimate the speedup as a result of any performance optimization. Amdahl’s Law does have its implications, it assumes parallelizable parts can be run in parallel without any limits, and in practice, there are limits. In order for speedup in real applications with serial and parallel portions, heterogenous computing is required.

1. How does guided loop scheduling work in OpenMP? What advantage may guided scheduling offer over dynamic scheduling?

Guided loop scheduling in OpenMP divides iterations into chunks. A thread will execute a chunk then request another until all chunks have been executed. In dynamic scheduling these chunks are different sizes. Guided scheduling would have an advantage over Dynamic scheduling if the chunks were limited to a small size giving the most flexibility.

1. Describe the difference between blocking and non-blocking communication in MPI. Can the use of non-blocking communication operations help improving application performance? How?

Blocking communication is different from non-blocking communication in OpenMP based on what is returned. Blocking returns only when the buffer is ready to be used again whereas Non-Blocking returns right away. Non-Blocking communication can improve performance by overlapping communication with computation. For example, when scaling up to a larger system with inter-process communication, Non-Blocking communication will provide performance optimization.

**THIS PAGE IS INTENTIONALLY LEFT BLANK FOR YOUR ANSWERS TO Q1.**

**2. Parallel Algorithm Design [8+6 = 14 pts]**

The kernel given below performs a *sequential* matrix vector operation.

for i = 1:n {loop over rows of A}

for j = 1:n {loop over entries in x & the ith row of A}

y(i) = y(i) + A(i,j)\*x(j);

You would like to *parallelize* this operation over *t* threads, and you are trying to choose between a column-wise vs. a row-wise partitioning of your matrix among threads for parallelization. See the figures below which highlight the regions that each thread would be responsible for under column-wise (left) and row-wise partitioning (right), in this example for 4 threads.

**a)** Considering factors such as degree of parallelism, load-balancing, synchronizations and race conditions, which scheme would you choose? Why?

I would choose column partitioning to reduce disk IO.

**b)** For the scheme you choose, give the OpenMP parallel implementation for this operation.

n

i

x

A

column partitioning

row partitioning

**3. Amdahl’s Law [6+6 = 12 pts]**

**a)** Assume a code has a serial startup time of 1 second and a parallelizable portion that takes 1000 seconds to run on a single thread of execution. Calculate the parallel speedup and efficiency of this code when run on 50 processors and when run on 1000 processors.

On 50 processors: 1050/100

On 1000 processors: 2000/200

**b)** Consider an image segmentation application which reads in a batch of images, performs segmentation on each image and writes back the segmented images. You have recently upgraded the magnetic disk on your workstation with an SSD. Based on their specs, SSD is 5x faster than the disk in terms of read performance and it is 3x faster in terms of write performance.

Upon analyzing the performance profile of your application *after the upgrade*, you find out that 10% of the total application time is spent on reading the images in from the SSD, 70% is spent on processing them and the final 20% is spent in writing the results back to the SSD. Assuming speedups from hardware spec comparisons directly carry into the read/write phases of the applications, how much speedup have you obtained for this image segmentation application as a result of the SSD upgrade?

3/10

**4. OpenMP Parallelization [10 pts]**

A common sequential solution to identify the set of prime numbers in the sequence of integers from 1 to n is the Sieve of Eratosthenes. In this method, the series of all primes is generated starting from 2. The first number, 2, is prime and kept. All multiples of 2 are deleted because they cannot be prime. This process is repeated with each remaining number, up until but not beyond sqrt(n). A sequential implementation of this algorithm is as follows:

for (i=2; i<=n; i++)

prime[i] = true;

for (i=2; i<= sqrt(n); i++) {

if (prime[i]) {

for (j=i+i; j<=n; j=j+i) //multiples of i aren’t prime

prime[j] = false;

}

}

Give an efficient OpenMP parallel implementation of the Sieve of Eratosthenes method.

Int Eratosthenes(int LastNumber)

{

//Initialize

Char\* isPrime = new char[LastNumber + 1]

For (int I = 0; I <= LastNumber; i++)

isPrime[i] = 1;

//non-primes

For (int I = 2; i\*I <= LastNumber; 9++)

If (isPrime[i])

For (int j = i\*I; j <= LastNumber; j += i)

isPrime[j] = 0;

//Now count primes

Int found = 0;

For (int I = 2; I < LastNumber; i++)

found += isPrime[i];

delete[] isPrime;

return found;

**}**

**5. Parallel Programming [6+6 = 12 pts]**

**a)** Write a pseudocode OpenMP program in which multiple threads are launched, each generates a random number, and then the global maximum value is found. Do not worry about exact syntactical correctness, but do specify what OpenMP directives and any important options that need to be used.

Global\_Result = 0.0;

#pragma omp parallel num\_threads(thread\_count)

{

Double my\_result = 0.0;

My\_result += Local\_trap(double a, double b, int n);

#pragma omp critical

Global\_result += my\_result;

}

**b)** Write a pseudocode MPI program in which each process generates a random number and then process 0 finds the global maximum value. Do not worry about exact syntactical correctness, but do specify what MPI functions need to be called and any critical options that need to be passed to them.

Double sum = 0.0;

#pragma omp parallel for num\_threads(thread\_count)

Default(none) reduction(+:sum) private(k, factor)

Shared(n)

For(k = 0; k < n; k++) {

If (k % 2)

Factor = 1.0;

Else

Factor = -1.0;

Sum += Factor / (2 \* k + 1)

**6. Deadlocks [10 pts]**

Consider the code snippets below. Which one(s) will result in a deadlock? Explain briefly.

Option B will result in a deadlock.

Provide a fix for those with a deadlock.

**a)**

for (int p=0; p<nprocs; p++)

if (p!=procid)

MPI\_Send(sbuffer, buflen, MPI\_INT, p, 0, comm);

for (int p=0; p<nprocs; p++)

if (p!=procid)

MPI\_Recv(rbuffer, buflen, MPI\_INT, p, 0, comm, MPI\_STATUS\_IGNORE);

for (int p=0; p<nprocs; p++)

if (p!=procid)

MPI\_Recv(rbuffer, buflen, MPI\_INT, p, 0, comm, MPI\_STATUS\_IGNORE);

for (int p=0; p<nprocs; p++)

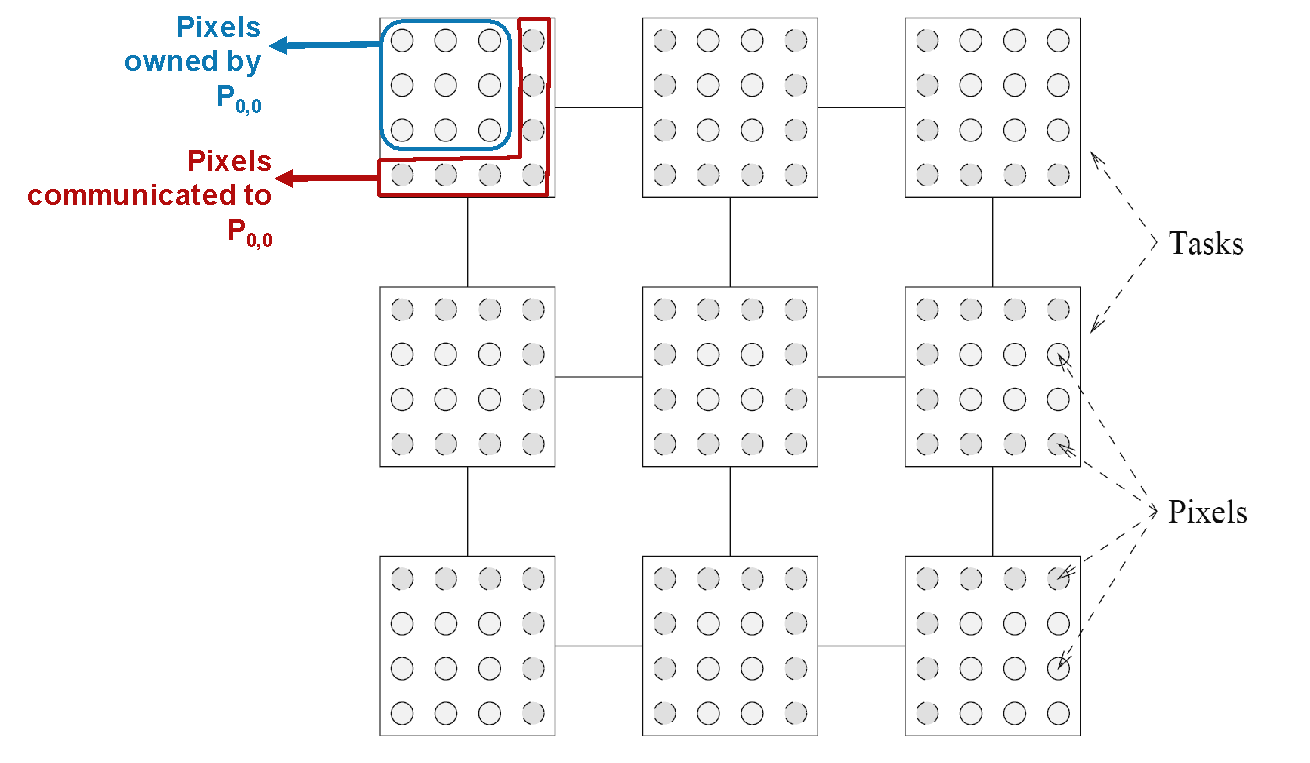
if (p!=procid)

MPI\_Send(sbuffer, buflen, MPI\_INT, p, 0, comm);

**b)**

**7. Communication Costs [4 + 10 = 14 pts]**

In image dithering, the color of each pixel in the *n* × *n* image is determined as the weighted average of its original value and the values of its neighboring pixels. We can decompose this computation by breaking the image into square regions and assign the task of processing each of these square regions to different processes. Note that in addition to the pixel values in its own square region, each process needs to access the values of the pixels surrounding its region. This 2-D block decomposition is illustrated for an 8x8 image in the figure below using a process grid of size 3x3. Those within solid circles represent the pixels owned by each process, and those within dashed circles represent the pixel values that must be communicated from neighbors.



1. Illustrate the 1-D block decomposition for the same 8x8 image using 4 processes.
2. For the general case of an *n* x *n* image and *p* processors (you can assume n >> p), compare the communication volume incurred by a 2D vs. 1D partitioning.

**Scrap page**