Norwegian University of Science and Technology NTNU

Department of Computer and Information Science

TDT 4200 Final Exam)
Parallel Computing
Saturday, Dec. 8, 2012
Time : 09:00 – 13:00

THIS EXAM IS GIVEN IN ENGLISH ONLY
[Oppgavesett for dette faget foreligger kun på Engelsk, slik det er beskrevet i Emner-på-Nett (EpN)]

Instructional contacts during the final

Anne C. Elster, mobil: 981 02 638

ALL ANSWERS NEED TO BE WRITTEN ON THES EXAM SHEETS WHERE INDICATED AND THESE SHEETS TURNED IN FOR GRADING.

NO EXTRA SHEETS WILL BE GRADED!

Aids [Hjelpemidler] :

One handwritten note sheet with Department stamp and candidate's name may be used. It needs to be turned in with the exam (may remove name before turn-in)

Attached "Summary of MPI Routines and Their Arguments" is permitted as written aid.

No other aids, including calculators, are permitted

Grades will be assigned medio January 2012.

It is NOT necessary to justify your answer on true/false questions, unless requested.

Written by: Anne C. Elster   Partially Checked by: Ruben Spaans

CANDIDATE NUMBER/Kandidatnr.: ______________________

1. WARM-UPS – TRUE/ FALSE (20 %)

Circle your answers -- **Note: You will get a -1% negative score for each wrong answer and 0 for not answering or circling both** TRUE and FALSE.

a) **OpenMP is used for programming shared-memory systems** TRUE/FALSE

b) **MPI programs may be SPMD** ……………… TRUE/FALSE

c) **L1 Cache is the fastest memory available** ……………….. TRUE/FALSE

d) **A 4D hypercube has 8 nodes** ………………… TRUE/FALSE

e) **Large Switch statements may be improved my moving** the **most used case last** TRUE/FALSE

f) **Domain decompositions is a form of task parallelism** TRUE/FALSE

g) **Data locality is a mayor performance challenge ………..** TRUE/FALSE

h) **Caching is used to help overcome the memory bottleneck** TRUE/FALSE

i) **Reduction operations are not parallelizable** ………………. TRUE/FALSE

j) **Tail recursion is hard to parallelize** ………………. TRUE/FALSE

k) **Pthread programs are implemented using compiler directives** TRUE/FALSE

l) **OpenMP uses compiler directives ……………………….** TRUE/FALSE

m) **Large SIMD programs are especially well-suited for GPUs** TRUE/FALSE

n) **GPUs use extensive branch prediction ………..** TRUE/FALSE

o) **CUDA threads may access any registers within a given warp** TRUE/FALSE

p) **CUDA warps use the SIMD model ……. ……………………** TRUE/FALSE

q) **CUDA supports global synchronization …………………….** TRUE/FALSE

r) **Constant memory in CUDA is read-only from host** TRUE/FALSE

s) **OpenCL is generally less verbose than CUDA …………………** TRUE/FALSE

t) **OpenCL is closer to OpenMP than thread APIs such as POSIX …** TRUE/FALSE

2. . PARALLEL COMPUTING BASICS (10%)

a) **Let l= latency and b= bandwidth, n = size. Which formula describes the message transmission time? (Circle the correct answer)**

i) b + n/l   ii) b + l/n   iii) l + n/b   iv) l+ b/n

b) **Amdahl´s Law says if a fraction r of a program isn't parallelizable, then the maximum speedup we can get is 1/r regardless of how many processes/threads/cores we use.**

If initialization and I/O takes 5% of the time, what is the best possible speedup? (I.e. How many cores can we maximally make use of according to this law?)

____

c) **How is Gustafson´s law related to scalability?**

____

d) **Efficiency E = n / (p(n/p) +1) = n/ (n+p) for n = problem size and p = no. of processes.**

What is the difference between strongly scalable and weakly scalable in terms of efficiency?

i) A program is strongly scalable if ________________________________

____

ii) A program is weakly scalable if: ____

____

iii) Is a program that obtains linear speed-up strongly scalable – Why/Why not?

____

e) **The two main methods for cache coherence are:**

i)__________________________________ ii)) ______________________________

f) What is a critical section?

____

3. MPI (12%)

a) **Suggest one reason why a MPI computation would execute slower on a system with two processors than on a system with one processor**.

____

____

b) **Identify one MPI routine that does not use a communicator as an argument.**

____

c) **The main difference between using MPI_Send + MPI_Recv and MPI_Sendrecv?**

MPI_Sendrcv ____

d) **The difference between MPI_Sendrecv and MPI_Sendrecv_replace is that**

MPI_Sendrecv_replace____

e) **Why is illegal in MPI to use the same buffer for I/O in MPI_Reduce (i.e. no aliasing)?**

____

3 MPI Continued

f) Below are three snippets from a MPI program, which will be run with two processes.

Which of them MIGHT cause a deadlock? Briefly explain why.

Snippet __________________

Because: ____

Snippet A:

```
int* source = (int*)malloc(sizeof(int)*1000);
int* dest = (int*)malloc(sizeof(int)*1000);

int other = (rank + 1) % 2; // Rank of other process

... // fill source with values
MPI_Recv(dest, 1000, MPI_INT, other, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
MPI_Send(source, 1000,  MPI_INT, other, 0, MPI_COMM_WORLD);
```

Snippet B:

```
int* source = (int*)malloc(sizeof(int)*1000); // assume filled with values
int* dest = (int*)malloc(sizeof(int)*1000);

int other = (rank + 1) % 2; // Rank of other process

  ... // fill source with values
MPI_Ssend(source, 1000, MPI_INT, other, 0, MPI_COMM_WORLD);
MPI_Recv(dest, 1000, MPI_INT, other, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

Snippet C:

```
int* source = (int*)malloc(sizeof(int)*1000); // assume filled with values
int* dest = (int*)malloc(sizeof(int)*1000);

int other = (rank + 1) % 2; // Rank of other process

  ... // fill source with values
MPI_Send(source, 1000, MPI_INT, other, 0, MPI_COMM_WORLD);
MPI_Recv(dest, 1000, MPI_INT, other, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

4. OPTIMIZATIONS – (8%)

Short Answer questions -- fill in the blanks (5%)

i) What is the difference between a DFT and an FFT?

The DFT ____

Whereas the FFT ____

ii) ATLAS is a (circle the one that best fits):

a. mapping program

b. performance analyzer

c. optimized BLAS library

d. optimized C compiler

iii) What advantage does intrinsics offer over autovectorization?

____

iv) When does it make sense to optimize/remove branches?

(circle ALLthat apply)

a. when the call statement is not matched

b. when the loop can be unrolled

c. when the conditional branch is executed for the first time

d. for conditional branches that have been executed more than once with which is frequently mis-predicted and take a significant amount of time

e. Indirect calls and jumps (funtion pointers and jump tabes)

v) What is the difference between temporal and spatial locality in the context of caches?

Temporal locality here is ____

Spatial locality here is ____

5. OpenMP (10%)

a) Name at least two differences between a process and a thread:

i) ____

ii) ____

b) The following code may be parallelize with OpenMP as shown:

```
for(int i = 0; i < N; i++){
        for(int j = i; j < N; j++){
            array[i*N + j] = sin(i) + cos(j);
        }
}
```

Alteraative A

```
#pragma omp parallel for
for(int i = 0; i < N; i++){
    for(int j = i; j < N; j++){
        array[i*N + j] = sin(i) + cos(j);
    }
}
```

Alternative B:

```
#pragma omp parallel for schedule(dynamic)
for(int i = 0; i < N; i++){
    for(int j = i; j < N; j++){
        array[i*N + j] = sin(i) + cos(j);
    }
}
```

If the above alternatives are run on a two-core processors, what (approx) speedups willl these alternatives achieve? Explain why.

i) Alternative A Speedup = _____

Because____

ii) Alternative B Speedup: ______

Because: ____

5. Continued -- More OpenMP

5 c) Consider the following code:

```
#pragma omp parallel for
for(int i = 0; i < 10; i++){
        for(int j = 0; j < 10; j++){
                array[i] += buffer[i* 10 + j];
                array[j] -= buffer[i* 10 + j];
        }
}
```

a) What is a race condition? ____

____

b) Explain why the code above has a race condition:

____

c) Consider the following attempt at removing the race condition:

```
#pragma omp parallel for
for(int i = 0; i < 10; i++){
        for(int j = 0; j < 10; j++){
                array[i] += buffer[i* 10 + j];

                #pragma omp critical
                {
                        array[j] -= buffer[i* 10 + j];
                }
        }
}
```

Will this solve the problem? Circle answer a. YES b. NO

6. PThreads (10%)

Consider the following code:

```
double* pos;
double* vel;

void* updatePosition(void* threadid){
   long tid = (long)threadid;

   for(int i = tid*500; i < tid*500 + 500; i++){
      pos[i] = pos[i] + vel[i] * 0.1;
   }
}

int main(){

   double pos = (double*)malloc(sizeof(double) * 1000);
   double vel = (double*)malloc(sizeof(double) * 1000);
    int dummy;

   pthread_t thread1, thread2;

   for(int i = 0; i < 1000000; i++){
      pthread_create(&thread1, NULL, updatePosition, (void*)0);
      pthread_create(&thread2, NULL, updatePosition, (void*)1);
      pthread_join(thread1, &dummy);
      pthread_join(thread2, &dummy);
   }
}
```

a) **Why is it slower than it could have been?** ______________________

____

b) **Change it, to remove this problem, and improve performance.**
Extra sheet to use on next page)

Extra sheet for 6 c), if needed

7. GPU Programming -- PART 1 (15%)

a) Name two types of overhead for CUDA programs.

i)____

ii)____

b) Name at least two differences between OpenCL and CUDA (other than naming conventions):

i)____

ii____

e) The following code is part of a CUDA program that sets some of the pixels of an image to black (0) on the GPU. Will this program be faster on the GPU than CPU?

i) Circle the right answer: a. YES b. NO

ii) Explain why/why not: ____

____

```
__global__ void kernel(int* image) {
  image[blockIdx.x*blockDim.x + threadIdx.x * 1000] = 0;
}

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

  int* image = (int*)malloc(sizeof(int)*10000*10000);
  readImage(image);

  int* deviceImage;
  cudaMalloc((void**)&deviceImage, 10000*10000*sizeof(int));

      cudaMemcpy(deviceImage, image, 10000*10000 * sizeof(int),
cudaMemcpyHostToDevice);

      kernel<<<10, 10>>>(deviceImage);

      cudaMemcpy(image, deviceImage, 10000*10000 * sizeof(int),
cudaMemcpyHostToDevice);
}
```

8. GPU – PART II (15%)

a) Why is shared memory faster than global memory on GPUs?

Answer: The shared memory is faster because ________________________________

____

b) In the following CUDA kernel, each block of threads will do a lot of processing on a part of the array "buffer". Improve the performance of the code *by using shared memory* by first loading the part of the array the block accesses into shared memory . Use the__shared__ double **.....** **syntax. Remember to store back to global memory.**

The size of "buffer" is 1000000. The kernel is launched with 1000 blocks of 1000 threads.
Add in the modifications by hand to the code below (do not use extra sheet).

```
__global__ void kernel(int* buffer){

  int threadId = blockIdx.x*blockDim.x + threadIdx.x;




  for(int i = 0; i < 1000; i++){
    double sum;
    for(int j = 0; j < blockDim.x; j++){


      sum += buffer[threadId] - buffer[blockIdx.x*blockDim.x + j];
    }

    __syncthreads();



    buffer[threadId]  *= sum;



    __syncthreads();

  }

}
```

8. CONTINUED -- GPU PART II

c) Consider the following CUDA kernel:

```
__global__ void kernel(int* buffer){
   int threadId = blockIdx.x*blockDim.x + threadIdx.x;

   if(threadId % 4 == 0){
      doSomething(buffer);
   }
   else if(threadId % 4 == 1){
      doSomethingElse(buffer);
   }
   else if(threadId % 4 == 2){
      doYetAnotherThing(buffer);
   }
   else{
      doSomethingCompletlyDifferent(buffer);
   }
}
```

i) What is the problem with this code? Why is this a problem?

____

____

____

____

____

ii) Outline how you could solve this problem for this specific kernel (in words not code):

____

____

____

____

EXTRA PAGE (no other extra page will be graded!)