Norwegian University of Science and Technology NTNU

Department of Computer and Information Science

TDT 4200 Final Exam (Eksamen)
Parallel Computing [Parallelle beregninger/berekningar]
Wednesday, Dec. 7, 2011 [onsdag 7/12-2011]
Time [tid]: 09:00 – 13:00

Instructional contacts during the final[faglige kontakter/faglege kontaktar under eksamen]

Anne C. Elster, mobil: 981 02 638

ALL ANSWERS NEED TO BE WRITTEN ON THIS EXAM WHERE INDICATED AND THESE SHEETS TURNED IN FOR GRADING. YOU MAY USE THE EXTRA SHEETS PROVIDED FOR THE PROGRAMMING PROBLEM. YOU MAY NOT KEEP OR DISTRIBUTE ANY COPIES OF THIS EXAM.

DISSE EKSAMENSARKENE SKAL INNLEVERES OG ALLE SVAR FØRES INN DER DET ER ANGITT PLASS. KODEOPPGAVEN KAN BENYTTE EKSTRAARKENE VEDLAGT. DET ER IKKE TILLATT Å BEHOLDE ELLER DISTRIBUERE KOPIER AV DETTE EKSAMENSSETTET!

Aids [hjelpemidler]:

Only attached "Summary of MPI Routines and Their Arguments" is permitted as written aid. Any scratch sheets should be turned in with the final exam. No other aids, including calculators, are permitted.

Kun vedlagte "Summary of MPI Routines and Their Arguments" er tillatt som skriftlig hjelpemiddel.. Evt. kladdeark skal innleveres/innleverast med besvarelsen. Ingen andre hjelpemidler/el, inkludert kalkulatorer/ar, er tillatt/tilette.

Grades will be assigned by Jan 15, 2011. [Karakterer vil bli satt innen 15/1.2011.]

It is NOT necessary to justify your answer on true/false questions, unless requested.
If there are disagreements between the English and the Norwegian texts, the English text should be used as a guidline.

[Det er ikke nødvendig å avgi forklaring på TRUE/FALSE spørmål der det ikke er bedt om det. Skulle det være uoverstemmelser mellom den engleske og den norske teksten, skal den engelske teksten være førende.]

Written by: Anne C. Elster **Partially Checked by: Ivar Ursin Nikolaisen**

CANDIDATE NUMBER/Kandidatnr.: _____

1. WARM-UPS – TRUE/ FALSE [Sant/Ikke sant--sant/ikkje sant] (15 %)

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. [Sett sirkel rundt svarene -- NB: På denne oppgaven får dere -1% negativt poeng for hvert feilsvar, 0% poeng for å ikke svare eller å sirkle både ”TRUE”(sant) og ”FALSE” (ikke sant).]

a) **CUDA is a more portable al programming environement than OpenC** TRUE/FALSE
(CUDA et/eit mer portabelt prog. miljø enn OpenCL)

b) **Programming in MPI forces you to think about memory issues** TRUE/FALSE
(Porgramering i MPI tvinger deg til å tenke på minne)

c) **MPI_Send can use wildcards for source and tag** TRUE/FALSE
(MPI_Send kan bruke “åpne variabler” for source og tag)

d) **Most modern operating systems are multitasking**. TRUE/FALSE
(De fleste moderne operativsystem er “multitasking”)

e) **Modern GPUs take advantage of SIMD processing** TRUE/FALSE
(Moderne GPUer/ar tar/tek fordel av SIMD)

f) **Data locality does not matter on NUMA shared memory systems** TRUE/FALSE
(Lokasjon av data har ikke noe å si på NUMA fellesminnesystemer)

g) **Caching is used to help overcome the memory bottleneck** TRUE/FALSE
(“Caching” blir brukt til å overvinne minneflaskehalsen)

h) **Processors that support dynamic multiple issues are sometimes said to be superscalar** TRUE/FALSE
(Processorer/ar som støtter dynamisk “multiple issues” er noen/nokon ganger sagt å være “superscalar”)

i) **Radix sort parallelizes well** TRUE/FALSE
(Radixsortering paralleliseres bra)

j) With Pthread mutexes, the order in which the treads execute the code in the critical section is always sequential TRUE/FALSE
(Rekkefølg(j)en traåer eksekverer i med Prhead Mutex er alltid sekvensiell)

k) **Use of virtual memory never impacts performace** TRUE/FALSE
(Brukt av virtuelt minne gir aldri utslag på ytelse)

l) **Block and grid dimensions in CUDA can have up to 3 dimensions** TRUE/FALSE

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

n) **Shared memory in CUDA is faster than registers** TRUE/FALSE

o) **CUDA threads may access any registers in a given warp** TRUE/FALSE
(CUDA trader kan aksessere ethvert register I en gitt “warp”)

2. . PARALLEL COMPTING BASICS (15%)

a) Amdahl´s Law says if a fraction r of a program isn´t parallelizable, then the maximum speedup we can get is i/r regardless of how many processes/threads we use.

i) **If the I/O takes 5% of the time, how many processors could we maximally make use of according to this law?** _____

ii) **Show calculations for the above answer:**

iii) **What saves us from Amdahl´s law on modern 10 000 core+ supercomputers?**

_____

b) Explain the difference between *speedup* and *efficienc*y using T_serial and T_parallel on system with *p* processes:

c) Main reasons for super linear speedup is probably:
(Hovedgrunnene til mulig superlinær speedup er sansynligvis :)

_____

_____

d) **What are the two main differences between L1 and L2 cache?**
(Hva er to hovedforskjeller mellom L1 og L2 cache?)

_____

_____

e) List 3 differences between GPU and CPU (list 3 skilnader/forskjeller mellom GPU og CPU) :

_____ _____

_____

f) Absolute fastest and slowest datastorage on modern large supercomputer systems are:

Fastest: _____ **Slowest:** _____

3. PARALLEL PROGRAMMING LANGUAGES AND MODELS (15%)
(Parallel Programmeringsspråk og modeller/ar)

a) Name an advantage of OpenMP over POSIX Threads and vice versa?
(Nevn en fordel med OpenMP over POSIX-tråder og omvendt)

OpenMP advantage (fordel) _____

Pthread advantage(fordel)_____

b) What are the main two similarities between OpenCL and CUDA?
(Hva er to likheter mellom OpenCL og CUDA?)

_____

_____

c) i) The SPMD model versus SIMD. (SPMD modellen vis a vis SIMD)

i) Obtains parallelism through branching: _____

ii) CUDA warps use the _____ model.

iii) MPI follows the _____ model.

iv) Intel SSE instructions follow the _____ model

d) Fill in all that apply of: MPI, OpenMP, CUDA and/or OpenCL

i) Most widely used on share memory multiproessors:_____

ii) Most widely used for scalable cluster computing: _____

iii) Most widely used on AMD GPUs: _____

iv) Designed for use on heterogeneous clusters: _____

v) Makes you think about memory locality: _____

vi) Is the most similar to CUDA: _____

vii) Uses ANSI C extension _____

4) **OPTIMIZATIONS (optimeringer) (10%)**

a) **List at least 3 additional main techniques to increase performance with un-optimized codes.**: [Nevn minst 3 tilleggsmetoder for å øke ytlesen av uoptimerte koder:]

i) **USE optimized libraries such as BLAS, Atlas etc. wherever possible!**

ii_____

iii _____

iv _____

b) **Name at least 3 techniques discussed in class for removing branches**
(Nevn minst 3 teknikker for å fjerne forgreninger):

i) _____

ii) _____

iii) _____

c) What advantage does intrinsics offer over autovectorization?
(Hvilke fortrinn har "intrinsics" over auto-vektorisering?)

_____

d) Given an array defined as double m[2048][2048] and the following two snip- pets of code:

1)
```
for(i=0; i<2048; i++)
    for(j=0; j<2048; j++) {
        double value = m[i][j]; // do some calculations with value
    }
```

2)
```
for(j=0; j<2048; j++)
    for(i=0; i<2048; i++) {
        double value = m[i][j]; // do some calculations with value
    }
```
That is, the order of the two for loops are reversed

i) Which one of the two code snippets is most efficient? _____
ii) Explain why one of the snippets is faster

_____

_____

5) PROGRAMMING and PERFORMANCE (10%)

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

i) _____

ii) _____

b) **What are the following two libraries used for?**
(Hva brukes de følgende to bibliotekene til?)

Atlas: _____

PETSc: _____

_____

c) **What is the main diference between MPI_Test and MPI_Probe?**
(Hva er hovedforskjellen/hovudskilnaden på MPI_Test og MPI_Probe?)

_____

_____

e) It is **illegal** to **alias in/out** arguments in **MPI_Scatter*?*** **TRUE/FALSE**

Why/Why not? (hvorfor/hvorfor ikke?)

_____

d) GPUs can be used to accellerate performance of parallel codes. (GPUer kan bli brukt til å aksellerere ytelsen av parallelle koder.

Name two major issues with GPUs that limit their performance and explain what the challenge is: [Nevn to hovergrunner som bergrenser ytelsen til GPUer of forklar hva utfordringene er]

i)_____

ii)_____.

6. GPU sand GPU programming models (15%)

a) What is one of the main differences between a GPU and CPU?
(En av hovedskilnadene)forskjellene mellom GPU og CPU?)

---

b) What is a CUDA Bank conflict?

---

b) **Draw the CUDA device memory model by including** (Tegn mine-modelln til CUDA ved å ta med) **:shared memory, registers, threads, and global and Constant memory plus host and how they are related to**(oghvordan/ korleis de er relatert til) **grids, blocks and threads .**

c) **How do warps fit in this model?**

---

d) **OpenCL defines two types of memory objects: *buffers* and *image*s.**

i) _____ are equivalent to arrays in C created using malloc() and

ii) _____ are designed as opaque objects.

e) **NIVDIA has recently announced CUDA as a programming model for ARM devices. Would you use it over OpenCL? Why or why not?**

_____

7. VECTOR SUMMATION PROGRAMMING (programmering) (10%)

Code A: Given the following function:

```
float sumadd(float *out, float *a, float *b, int n) {
        float s = 0.0f;
        int i;


        for(i=0;i<n;++i) {
                out[i]=a[i] + b[i];
                s += out[i];
          }


        return s;
```

7a) Re-write the *sumadd* function as a CUDA kernel function right next to the above Code A

Hint: Code B of this exercise is a CUDA kernel program

7b) Parallelized Code A on left using OpenMP:

7c) Which conditions would the parameters out, a, b and n have to fulfill in order for the function to be easily optimized with SSE?

7 CONTINUED

Code B: And the CUDA sum reduction kernel:

```
__Shared__float Reduction_Sum[]

  unsigned int t = ThreadIdx.x;
  for unsigned int stride = 1; stride > blockDim.x; stride *= 2){
      __synchthreads();
  if (t % (2*stride) == 0)
      ReductionSum[t] += ReductionSum[t+stride]
  }
```

Code B clearly has thread divergence and spread of partial sums.

7d) Show this by drawing the memory access pattern for the above array for threads 0 through 7

7e)) Re-write this kernel to coalesce the partial sum threads in the left end of the array, that is, after the first sum, the partial sum should be in the first half of the array, etc:

```
{
```

8. MPI PROGRAMMING (programering) – Monte Carlo Method (10%)

Suppose we toss darts randomly at a square dartboard whose bullseye is at origin A and whose sides are 2 dm (deci meter). Inscribed in this square is a circle of radius 1 dm and area π square decimeter. If we always hit the board and hit it uniformly, the circle would be approximated by the equation:

(Antar at vi kaster piler i en firkantet blink hvor "midt I bkinken" er I origi. Og hvor sidene på frikanten er 2 dm. En sirkel på 1 dm er inne I sirkelen. Denne er π dm^2. Hvis I alltid treffer blinken og gjør dette i et uniform møsnster, kan sirkeleen bli approxmert som følger:

$$\frac{\text{Number in circle}}{\text{total number of tosses}} = \frac{\pi}{4}$$

We can then estimate π with a random number generator:

```
Number_in_circle = 0;
For (toss = 0; toss < number_of_tosses; toss++){
        x = random double between -1 and 1;
        y = random double between -1 and 1;
        distance_squared = x*x + y*y;
        if (dist_squared <= 1) number_in_circle++;
 }
pi_estimate = 4* number_in_circle / (double) number_of_tosses)
```

This is called a Monte Carlo Method since it uses randomness (the dart tosses).

a) Write an MPI program that uses a Monte Carlo Method to approvimate π.

Process 0 should read in the total number of tosses and broadcast it to the other processes. Use *MPI_Reduce* to find the global sum of the local variable *number_in_circle*, and have process 0 print the result. You may want to use *long long int*s for the number of hit in the circle and the number of tosses, since both have to be very large to get a reasonable approximations of π.

(Skriv et MPI program som bruker/nytter en Monte Carlo metode for å estimere π. Bruk MPI_Reduce til å finne/a den globale sum av den lokale variablen number_in_circle og ha process 0 til å skrive ut resultatet. Du bør antagelig bruke long long ints for number_of_hits i sirkelen og number_of_tosses, siden begge må være ganske store for å få et OK estimat av π.)

8a) Code repeated from last page to parallelize sing MPI:

```
Number_in_circle = 0;
For (toss = 0; toss < number_of_tosses; toss++){
        x = random double between -1 and 1;
        y = random double between -1 and 1;
        distance_squared = x*x + y*y;
        if (dist_squared <= 1) number_in_circle++;
}
pi_estimate = 4* number_in_circle / (double) number_of_tosses)
```

---- begin your code ----------

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>









MPI_Finalize();
}
```

Extra sheet for 8a) (Ekstra ark for 8a))

8b) Setting up vector types in MPI (10%)

You are given a two-dimensional data structure of 500 × 500 chars, and you want to transfer a subgrid of size 250 × 250 using MPI. Show how to define a MPI datatype for such a subgrid.

You are not required to make a complete computer program for this problem, just the required MPI statements for setting up the data type and an example on how to use it.

(Oppset av vektor-typer i MPI: Gitt at du har en 2D datastruktur med 500x500 chars, og du ønsker/ynskjar å øverføre et subgrid på 250x250 ved bruk av MPI. Vis hvordan du definerer en datatype for et slik subgrid. Du trenger ikke å skrive hele dataprogrammet for denne oppgaven(oppgåva), bare de nødvendige MPI kommandoene for å sette opp datatypen og et eksempel på hvordan/korleis den brukes.)

EXTRA PAGE (Ekstra-ark)