Norwegian University of Science and Technology NTNU

Department of Computer and Information Science

TDT 4200 Final Exam (Eksamen)
Parallel Computing [Parallelle beregninger/berekningar]
Tuesday, Dec 04, 2009 [tirdag 26/5-2009]
Time [tid]: 09:00 – 13:00

Instructional contacts during the final[faglige kontakter under eksamen] :

Anne C. Elster, 918-97-062 ; Jan Christian Meyer

ALL ANSWERS NEED TO BE WRITTEN ON THIS EXAM´s NUMBERED PAGES WHERE INDICATED AND THESE NUMEBERED SHEETS (not the MPI pecs attached) TURNED IN FOR GRADING. YOU MAY ONLY USE THE EXTRA NUMERED SHEET PROVIDED FOR THE PROGRAMMING PROBLEM. YOU MAY NOT KEEP OR DISTRIBUTE ANY COPIES OF THIS EXAM.

ALLE SVAR FØRES INN PÅ DE NUMERERTE ARKENE DER DET ER ANGITT PLASS, OG DE NUMERERTE ARKENE (bortsett fra vedlegget om MPI spesifiksjonen) SKAL INNLEVERES FOR SENSUR. KODEOPPGAVEN KAN KUN BENYTTE DET NUMERERTE EKSTRAARKET VEDLAGT. DET ER IKKE TILLATT Å BEHOLDE ELLER DISTRIBUERE KOPIER AV DETTE EKSAMENSSETTET!

Aids [hjelpemidler]:

Only attached "MPI Reference Card" is permitted as written aid. The note sheets should be turned in with the final exam. No other aids, including calculators, are permitted.

Kun vedlagte "MPI reference Card" er tillatt som skriftlig hjelpemiddel.. Notatark skal innleveres med besvarelsen. Ingen andre hjelpemidler, inkludert kalkulatorer, er tillatt.

Grades will be assigned by January 4 16, 2010. [Karakterer vil bli satt innen 4/1-2010 .]

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: _____ **Checked by: _____**

CANDIDATE Number/Kandidat nr.: _____

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

Circle your answers -- Note: You will get a -1% negative score for each wrong answers 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) **Programming in MPI forces you to think about memory issues** TRUE/FALSE
(Programering i MPI tvinger deg til å tenke på minne)

b) **MPI_Recv can use wildcards for destination and tag** TRUE/FALSE
(MPI_Recv kan bruke "åpne variabler" for destination og tag)

c) **SIMD instructions are offered on recent GPUs** TRUE/FALSE
(SIMD instruksjoner tilbys på moderne GPUer)

d) **Streaming is used to help overcome the memory bottleneck** TRUE/FALSE
("Streaming" blir brukt til å overvinne minneflaskehalsen)

e) **Blocked MPI calls return when they are locally complete** TRUE/FALSE
(Blokkerende MPI kall returnerer når de er ferdige lokalt)

f) **The FFT akgorithm is O(N^2)** TRUE/FALSE
(FFT-algorithmen er O(N^2)

g) Jacobi iterations typically converge faster than Gauss-Seidel TRUE/FALSE
(Jacobi-iterasjoner kovergerer vanligvis raskere en Gauss-Seidel

h) L1 cache misses may impact performance severely TRUE/FALSE
(L1 hurtigminne-"miss" kan ha en stor inflytelse på ytelse)

i) Software pre-fetchihng can avaoid cache issues TRUE/FALSE
(programvare "pre-fetching" kan avverge hurtig-minneproblemer)

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

k) CUDA can perform calculations with double precision TRUE/FALSE
(CUDA kan uteføre beregninger med dobbel presisjon)

l) CUDA is a set of compiler directives for massive parallelism TRUE/FALSE
(CUDA er et sett med kompilator direktiv for massiv paralllelisme)

m) Block and grid dimensions in CUDA can have up to three dimensions
TRUE/FALSE

n) Constant memory in CUDA is read-only from host

o) Shared memory in CUDA is faster than texture memory

2. **MPI BASICS. Fill in the blanks and circle TRUE or FALSE where indicated**.
(fyll inn og sett sirkler rundt true/false hvor indikert) **(10%)**

a. An MPI communciator is always needed TRUE/FALSE
(En MPI communicator er alltid nødvendig)

i. Why/why not? (Hvorfor/hvorfor ikke?)

b. MPI_ANY_TAG may always be used as an argument**.** **TRUE/FALSE**
(MPI_ANY_TAG kan alltid bli brukt som argument**)**

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

c. MPI_COMM_WORLD may always be used as an argument. **TRUE/FALSE**
Why/why not? (Hvorfor/hvorfor ikke?)

d. MPI collective operations do **NOT** use tags **TRUE/FALSE**

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

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

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

3. PARALLEL COMPTING BASICS (10%)

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

_____

_____

b) Amdah´s Law is given by (er gitt som: **) S(p) = p/(1+(p-1)f).**

i) What is S(p) as p → infinity? _____
(Hva blir S(p) hvis p → uendelig**)?**

ii) What is f? (Hva er f?)_____

iii) Mention two ways to overcome Amdahl´s Law:
(Nevn to måter å komme seg over Amdahl´s Law på:)

_____

_____

c) Why are Monte Carlo methods easy to parallelize?
(Hvorfor er Monte Carlo metoden lette å parallelisere?)

_____

_____

d) What is the SPMD model? Compare it to SIMD.
(Hva er SPMD modellen? Sammenlign den med SIMD)

_____

_____

5 OPTIMIZATIONS (optimeringer) (10%)

a) **List at least 4 main sources of performance problems with un-optimized codes.**:
[Nevn minst fire grunner til ytelsesproblemer av uoptimerte koder:]

i _____

ii _____

iii
_____

iv
_____

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

_____

_____

_____

c) Which of the following types of branches would one generally optimize:
(Hvilke av de følgende typer forgreninger bør en generelt optimere?)

i) Conditional branches executed for the first time

ii) Conditional branches that have been executed more than once.

iii) Call and Return

iv) Indirect calls & jumps (function pointers & jump tables)

v) Unconditional direct branches/jumps

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

_____

_____

6. PROGRAMMING AND GPUs (porgramering og GPUer) **(10%)**

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

_____ _____

b) **What is the main diference between MPI and OpenMP?**
(Hva er hovedforskjellen/hovudskilnaden på MPI og OpenMP?)

_____

_____

c) **What is the major bottleneck when using a GPU?**
(Hva er hovedflaksehasle ved bruk av CUDA)

---

d) **Why is branching bad for a GPU?** (Hvorfor er det ille men forgreninger på GPU?)

---

e) **When should one use shared memory in CUDA?**
(Når bør en bruke fellesminne i CUDA?

_____

_____

7. Performance & Load Balancing (Ytelse & Lastbalansering) (10%)

a) 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)_____

b) Name at least 4 static load-balancing techniques
(Nevn minst 4 statiske lastbalanseringsteknikker):

i) _____ii)_____

iii)_____ iv)_____

c) Which conditions need to be satisfied for termination in centralized dynamic load-balancing systems? (Hvilke to kriterier må være oppfylt for at en sentralt dynamisk lastballanert system skal terminere?)

---

d) List at least major differences between CPU and CPU
[Nevn minst tre forskjeller på CPU og GPU:]

i _____

ii _____

iii _____

8. CUDA PROGRAMMING (programering) (5%)

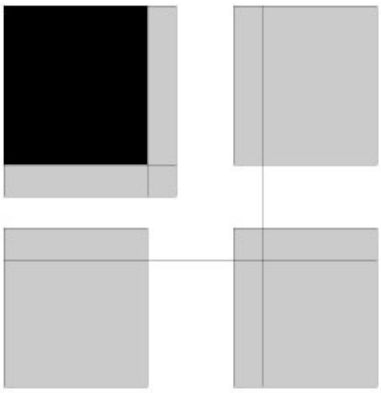
// What is the content of the ip array on device after the following code has executed?

```
 #include <stdio.h>

__global__ void kernel(int * ip) {

   int i = threadIdx.x + blockIdx.x*blockDim.x + blockIdx.y*blockDim.x*gridDim.x;

   ip[i] = blockIdx.x+blockIdx.y;
}
  int main(int argc, char *argv[]) {
   int * ip;

   dim3 d0  = dim3(3, 3);

  dim3 d1 = dim3(2);

  int i0 = d1.x * d0.x * d0.y * sizeof(int);

  cudaMalloc((void **) &ip, i0);

 kernel<<<d0, d1>>>(ip); }
```

IP ARRAY: __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __

9. MPI PROGRAMMING (programering) (15%)

Image decomposition with MPI



When splitting images for filtering applications, the size of the filter mask often determines a halo area (much like the stencil in PS3 did, but without periodicity). A 4-way split like this is illustrated in the figure on the right, note how the halo extends only towards the middle.

The program code below requires such a split in the function distribute_data, and your task is to write it. To restrict the problem, note that the program needs only to work with

- ***4 processes***
- ***1024x1024 images (N = 1024)***
- ***5-wide borders (B = 5)***

Bildeoppdeling med MPI

Når man deler opp bilder for filtreringsanvendelser bestemmer størrelsen på filtermasken ofte et speilet naboområde (svært likt hva stensilen forlangte i øving 3, men uten periodisitet). Figuren til høyre illustrerer en 4-veis oppdeling som dette, legg merke til hvordan naboområdene bare strekker seg mot bildets sentrum.

Programkoden nedenfor trenger en slik oppdeling i funksjonen distribute_data, og oppgaven din er å skrive den. For å begrense problemstillingen, legg merke til at programmet bare trenger å fungere med

- *4 prosesser*
- *bilder på 1024x1024 (N = 1024)*
- *kanttykkelse på 5 (B = 5)*

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

#define N 1024
#define LN (N/2)
#define B 5

int *source_data, rank, size;
int local_data[LN+B][LN+B];
```

```
void read_source_data ( int *src );
void distribute_data ( void );
void filter( void );
void collect_data ( void );

int
main ( int argc, char **argv )
{
   MPI_Init ( &argc, &argv );
   MPI_Comm_rank ( MPI_COMM_WORLD, &rank );
   MPI_Comm_size ( MPI_COMM_WORLD, &size );
   if ( size != 4 )
      { fprintf ( stderr, "This only works with 4 processes\n" ); exit ( EXIT_FAILURE ); }

   if ( rank == 0 )
      { source_data = malloc ( N*N*sizeof(int) ); read_source_data ( source_data ); }

   distribute_data();
   filter();
   collect_data ();
   if ( rank == 0 )  free ( source_data );
   MPI_Finalize();
}
void
distribute_data ( void )
{
```

```
}
```

9. b) More CUDA Programming (10%)

Fill-in-code task:

// The following code should increment values in host_data by 10. The incrementation is to be done on the GPU. Fill in the 3 blanks.
// E.g. if host data before kernel launch is (0 1 2 3 ...) then it should be (10 11 12 13 ...) after launch

```
#include <stdio.h>
 int table_length = 128*128;
 int * host_data; int * device_data;  __global__ void increment_kernel(int* data_table) {

       int i =                          // Fill in value

      data_table[i] += 10;
}

 void addValuesToHost(){
      for(int i=0; i<table_length; i++)
            host_data[i] = i;
 }


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

       dim3 block_dims = dim3(128, 1, 1);

      dim3 grid_dims  = dim3(128, 1);

      host_data = (int*) malloc(table_length * sizeof(int));

       addValuesToHost();

// Fill in code here




       increment_kernel<<<grid_dims, block_dims>>>(device_data);

      // Fill in code here



      for (int i1 = 0; i1 < table_length; i1++) printf("%d ", host_data[i1]);
}
```

10. OpenMP Programming (5%)

10 a) Parallelize the following function using OpenMP:

```
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;
}
```

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

EXTRA PAGE (Ekstra-ark)

EXTRA PAGE (Ekstra-ark)