NTNU – Trondheim
Norwegian University of
Science and Technology

Department of Computer and Information Science

Examination paper for TDT4200 Fall 2014

Academic contact during examination: Dr. Anne C. Elster (Instructor)

Phone: +47 981 02 638

Examination date: Monday Dec 8, 2014

Examination time (from-to): 09:00-13:00

Permitted examination support material: C – Extra material attached to exam paper only.

[Beskjed til eksamensvakter: ALLE SVAR MAA ANGIS AV KANDIDATENE PAA VEDLAGTE EKSAMENSARK SOM LEVERES INN SAMMEN MED OMSLAG OG NOTATARK. NOTATARK VIL GENERELT IKKE BLI RETTET.]

Other information: ALL ANSWERS NEED TO BE WRITTEN ON THIS EXAM´s NUMBERED PAGES WHERE INDICATED AND THESE NUMBERED SHEETS NEED TO BE TURNED IN FOR GRADING. YOU MAY ONLY USE THE EXTRA NUMBERED SHEETS ATTACHED FOR THE PROGRAMMING PROBLEMS.

Please fill in your candidate number on each sheet before turning them in.

YOU MAY NOT KEEP OR DISTRIBUTE ANY COPIES OF THIS EXAM

Language: English

Number of pages (front page excluded): 13

Number of pages enclosed:

Checked by:

_____

Date Signature

1. **WARM-UPS – TRUE/ FALSE (20 %)**

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

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) Data locality is a major energy and performance challenge.....….....TRUE/FALSE

b) MPI exposes data locality.....TRUE/FALSE

c) NUMA shared memory systems are not affected by data locality ..... TRUE/FALSE

d) Caches may be used to overcome locality issues….....TRUE/FALSE

e) All branch optimizations can be handled by a branch predictor ….....TRUE/FALSE

f) OpenMP is used for programming shared-memory systems..... TRUE/FALSE

g) A 5D Hypercube has 32 nodes ..... TRUE/FALSE

h) Task parallelism is typically a form of domain decomposition ..... TRUE/FALSE

i) Pthread programs use compiler directives ….....TRUE/FALSE

j) Elster's Serial Algorithm for BitReversal is linear i.e. O(N)..... TRUE/FALSE

1. CONTINUED

k) MPI_Bcast may use MPI_ANY_TAG..... TRUE/FALSE

l) Recursion benefits from on-chip registers available for stack .....TRUE/FALSE

m) Applications with output images are well suited for GPU computing...TRUE/FALSE

n) Constant memory in CUDA is read-only from host.....TRUE/FALSE

p) CUDA threads may access any registers in a given warp.....TRUE/FALSE

State assumption: _____

q) synchthreads( ) may be used to synchronize across threadblocks in CUDA

….....TRUE/FALSE

State assumption:_____

r) Team is the OpenCL-equivalent of a CUDA warp ..... TRUE/FALSE

s) OpenCLprogramming is currently limited to GPUs.....TRUE/FALSE

t) GPUs rely on branch prediction for performance ..... TRUE/FALSE

2. PARALLEL COMPUTING BASICS (10%)

2.1 Explain the 3 components that constitute the "BRICK WALL" described in Lecture 3 (Hint: the term was coined by Prof. David Patterson at UC Berkeley.)

i) Power Wall: _____

ii) Memory Wall: _____

iii) ILP Wall: _____

2.2 What are the Pros and Cons of Replicated vs. Distributed grids?

Replicated:

Pros: _____

Cons: _____

Distributed:

Pros: _____

Cons: _____

2.3 To parallelize a simple for loop on a CPU you would use:

i) MPI ii) OpenMP iii) Pthreads iv) CUDA v) OpenCL

State assumption for above choice: _____

3. MULTIPLE CHOICE QUESTIONS (8%)

Note that multiple choices may be correct. Negative score will be given within each question. i.e. if say one correct and one wrong circled for a given question and only one was correct, then 0 will be given.

3.1 How do you get rid of branches (circle all that apply)?

(a) Use SIMD instruction for masking optimizations

(b) Break out of loops

(c) Optimize loops

(d) Optimize Switch instructions

(e) Consider conditional branches that have been executed more than once

3.2. Elster's algorithm is a:

(a) Optmization scheme for multi-core

(b) FFT-based algorithm

(c) Linear Bit-reversal algorithm

(d) Fast sorting algorithm

3.3. The differences between OpenMP and OpenMPI is:

(a) One is a standard, the other an extention of that standard.

(b) The first is a library the second C-extension implementations

(c) The second one parallelizes distributed processes across machines, the first one not.

(d) The second one uses message passing, the first one shared memory directives

3.4 MPI_Allgather uses tags

(a) allways

(b) sometimes

(c) never

(d) root only

3.5 One may avoid busy-waiting by using?

(a) semaphores

(b) monitors

(c) mutex locks

(d) all of the above

4. More Parallel Computing Concepts including CUDA (12%)

4.3Draw a 3D hypercube with Greycode encoding of the nodes -- i.e. let the node number of connected nodes vary by only one bit).

4.2 Elster used recursion to get a serial algorithm from O(N logN) to O(N).

a) How fast in terms of big-Oh would a parallel version of the classical O(N logN) be?

__O(_____)

b) How did Elster's algorithm parallelize for multicore systems? Describe the trick used to get the above algorithm suitable for multi-core systems? (Hint: Similar method used in Atlas and FFTW)

_____

_____

4.3 What is one of the main differences between OpenCL and CUDA?

_____

4.4. Register allocation on NVIDIA GPUs

(a) is done when individual variables get assigned

(b) are assigned globally across SMs

(c) allows reading across warps on both Fermi and Kepler

(d) all of the above

4.5 CUDA instruction __constant__

(a) is read-only

(b) has global scope

(c) is initialized by host

(d) all og the above

4.6 If warps are executing in an arbitrary overlapped fashion you can use

(circle correct:) barrier() __syncthreads() mem_fence() if needed to correct the behavior

Exercise 5, MPI (5%)

Consider the following code, where a rank receives data from another rank, and then processes it, in a loop:

```
for(int i = 0; i < 100; i++){
    MPI_Recv(&r, n, MPI_INT,  s, i, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    process(r);
}
```

Rewrite the code to overlap computation and communication. The following MPI functions can be useful:

```
int MPI_Recv( void *buf, int count, MPI_Datatype datatype, int source,
      int tag, MPI_Comm comm, MPI_Status *status )

int MPI_Wait (MPI_Request  *request, MPI_Status   *status)

int MPI_Irecv( void *buf, int count, MPI_Datatype datatype, int source,
      int tag, MPI_Comm comm, MPI_Request *request )

int MPI_Sendrecv( void *sendbuf, int sendcount, MPI_Datatype sendtype,
      int dest, int sendtag,
      void *recvbuf, int recvcount, MPI_Datatype recvtype,
      int source, int recvtag, MPI_Comm comm, MPI_Status *status )
```

Exercise 6, OpenMP (5%)

Consider the following code, where we try to find the maximum value in the array a.

```
int max = a[0];

#pragma omp parallel for
for(int i = 0; i < N; i++){
      if(a[i] > max){
            max = a[i];
      }
}
```

Rewrite the code to fix the race condition. Your fixed version should achieve good parallel performance. The following OpenMP directives and functions can be useful:

```
#pragma omp parallel [clause[ [, ]clause] ...]
      structured-block

 clause:
      if(scalar-expression)
      num_threads(integer-expression)
      default(shared | none)
      private(list)
      firstprivate(list)
      shared(list)
      copyin(list)
      reduction(reduction-identifier: list)
      proc_bind(master | close | spread)


#pragma omp for [clause[ [, ]clause] ...]
      for-loops

clause:
      private(list)
      firstprivate(list)
      lastprivate(list)
      reduction(reduction-identifier: list)
      schedule(kind[, chunk_size])
      collapse(n)
      ordered
      nowait


#pragma omp parallel for [clause[ [, ]clause] ...]
      for-loop

clause: Any accepted by the parallel or for directives,
except the nowait clause, with identical meanings and
restrictions.


#pragma omp single [clause[ [, ]clause] ...]
      structured-block

clause:
      private(list)
      firstprivate(list)
      copyprivate(list)
      nowait


#pragma omp critical [(name)]
      structured-block


#pragma omp atomic [read | write | update | capture]
      expression-stmt


int omp_get_num_threads(void);
int omp_get_thread_num(void);
```

Exercise 7, CUDA (10%)

Consider the following reduction kernel.

```
__global__ void reduce(int *g_idata, int *g_odata) {
      extern __shared__ int sdata[];

      // each thread loads one element from global to shared mem
      unsigned int tid = threadIdx.x;
      unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;
      sdata[tid] = g_idata[i];
      __syncthreads();

      // do reduction in shared mem
      for(unsigned int s=1; s < blockDim.x; s *= 2) {
            if (tid % (2*s) == 0) {
                  sdata[tid] += sdata[tid + s];
            }
            __syncthreads();
      }

      // write result for this block to global mem
      if (tid == 0) g_odata[blockIdx.x] = sdata[0];
}
```

Rewrite the kernel to improve performance by implementing the following two optimizations:

1. Let each thread load two elements from global memory and add them before storing them in shared memory.
2. Reduce the warp divergence in the reduction loop.

Do not implement additional optimizations, they will not give additional credit.

Exercise 8, OpenCL (5%)

Consider the following OpenCL kernel:

```
__kernel reorder(__global int* a, __global float*  b, __global float* c){
      float id = get_global_id(0);
      int index = a[id];
      c[id] = b[index];
}
```

The kernel is launched with 1D work-groups with size 256. The size of the array `b` is also 256, and the values in `a` are between 0 and 255. Improve the performance of the kernel by using coalesced reads from global memory to shared memory for `b`.

Shared memory arrays, where the size is known at compile time are declared with the following syntax:

```
__local float localBuffer[1024];
```

(for a buffer of 1024 floats).

In addition, the function `barrier()` synchronizes work-items in the same work-group.
(space for problem 8, extra space for other problems if needed)

EXTRA SHEET for grading