NTNU – Trondheim
Norwegian University of
Science and Technology

Department of Computer and Information Science

Examination paper for
TDT4200 Parallel Computing, Fall 2015

Academic contact during examination: Jørn Amundsen

Phone: 91897897

Examination date: Tuesday December, 1 2015

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

Permitted examination support material: code C, specified printed and hand-written support material is allowed. (This year (2015) no such support material is allowed). A specific basic calculator is allowed.

The exam accounts for 75% of the final grade, and the provided points show the maximal number of points that can be achieved on each assignment. Read the problem texts thoroughly. You can answer the questions in English or Norwegian.

For all multiple choice questions: Answer by writing the question-ID and one alternative, like this: "X1 c" where X1 is the question ID and c is your answer. You are awarded 3.0 points for a correct answer and 0 points if you do not answer. If your answer is wrong or you give more than one alternative, you will get -1.5 points.

Language: **English**

Number of pages: **7**

Checked by:

_____

Date Signature

A. Parallel Computing - multiple choice (Maximum 33 points)

1) The textbook describes what is a very common distinction between the three concepts concurrent, parallel and distributed computing. Which of the following three statements is most correct in this context?

   a) Concurrent computing describes parallel computing being executed by time multiplexing on a single processor, while distributed computing is a parallel computation where the data are distributed as evenly as possible over all the processing nodes in a single supercomputer.

   b) Both parallel and distributed computations are concurrent, but in parallel computing multiple tasks cooperate more closely than in distributed computing.

   c) Concurrent computing uses asynchronous message passing whenever it is possible, parallel computing uses mainly synchronous communication and barrier synchronization, and distributed computing is centered around data level parallelism.

2) Three very central concepts for parallel computing is multitasking, processes and threads. Which of the following three statements about these concepts is most correct?

   a) Multitasking is an implementation technique for task-based programming that is used to distribute processes and threads over all the cores in a multicore processor to improve load balancing.

   b) Multitasking is used by a process to let many threads access the memory to prefetch data into a process cache before the main computation starts.

   c) Multitasking can implement appearantly simultaneous (parallel) execution of multiple processes by sharing of a single processor, while threads are normally more "light-weight" and contained within processes and can share the same executable code.

3) Three central concepts when discussing speedup of parallel programs are Amdahls law, strong scaling and weak scaling. What is the most correct statement?

   a) Amdahls law assumes strong scaling.

   b) Weak scaling and strong scaling does the same assumptions about problem size, but strong scaling corresponds to 50% efficiency or better.

   c) Amdahls law is outdated because all problems can be increased in size if we get access to a larger computer, as assumed in weak scaling.

4) Assume a multicore processor (CPU) with 8 cores and private L1 caches. Which of these three descriptions is the best explanation of the purpose of a bus snooping cache coherence mechanism in this context?

   a) It snoops the bus for cache misses from a core and checks if the requested data are stored in a register in any of the other cores.

   b) To be able to keep several copies of a data-word in different L1-caches when data are read, and to keep the data consistent by use of update or invalidation operations when data are written.

   c) To detect when the memory traffic is too high by monitoring the temperature of the bus.

5) Two-dimensional arrays can be stored in row-major or column-major order. C uses row-major order, and we assume a standard dekstop computer with a multicore CPU. When using two nested for loops for processing all the elements in such an array it is important to:

   a) Loop over columns in the innermost loop to harvest the advantage of data being prefetched since cache-lines are longer than one data-word.

   b) Disable prefetching of data to the cache to avoid an increased miss-rate.

   c) Copy each row into a local memory buffer to reduce the cache traffic**.**

6) Assume a communication channel with latency L and bandwidth B. The time used to exchange a message with length N over the channel can be expressed as:

   a) L + B + N/2 (/ denotes division)

   b) (L*N)/B (* denotes multiplication)

   c) L + N/B

7) Assume a shared memory system with 4 cores. When writing a parallel program for such a system it is important to avoid false sharing since:

   a) It can result in a situation where one core reads false or wrong values since the data has been incorrectly updated by another core.

   b) It will reduce performance.

   c) It can result in a message being sent to an incorrect receiver, and hence delay its arrival at the correct destination.

8) When running a parallel computation with a very large memory footprint (memory usage) the use of a virtual memory system is of special importance since:

   a) It allows a dynamic extension of the physical memory during operation (so called hot-swap).

   b) It allows the programmer to use as large a memory as the application needs. If 64-bit adresses becomes too small, the complete memory system will automatically expand to 128 bit adresses via hyper extendable memory-buffers, flex-busses and cooling by pre-installed synthetic snake oil.

   c) It automatically manages the data used by the application by moving it to and from the disks in much the same way as on-chip cache systems speeds up access to main memory.

9) SPMD is an abbreviation for Single Program Multiple Data, and most of our programs in this course follow that pattern. Flynn's taxonomy for multiprocessors define the abbreviations SIMD and MIMD. The relation between Flynn's taxonomy and SPMD is that:

   a) A SPMD computation can execute efficiently on a MIMD computer.

   b) A SPMD computation can not contain program sequences using SIMD instructions.

   c) SPMD and SIMD are two different names on the same concept.

10) Assume a program using POSIX Threads (pthreads). The main thread has performed two calls to the function pthread_create(), and no call to pthread_join(). How many threads are running at the same time?

a) 1

b) 2

c) 3

11) When more than one thread wants to update a shared variable this must be coordinated in a critical section that ensures atomic updates (mutual exclusion, or mutex for short). Such a critical section can be implemented using busy-waiting. The advantage of instead using the mutex-primitives (function calls) offered by the pthreads API is that:

a) The code becomes more portable since it can be used together with OpenMP or MPI.

b) They will reduce the amount of CPU-cycles used by cores or CPUs waiting to get access to the protected variable.

c) The waiting time becomes much shorter since the mutex will enforce a specific serialization of the accesses.

B. MPI (Maximum 12 points)

Implement a ring pass by sending an integer value as shown in the figure below with asynchronous MPI.

The following MPI functions might be useful:

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest,
       int tag, MPI_Comm comm)

int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm
comm, 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_Isend(const void *buf, int count, MPI_Datatype datatype, int dest, int tag,
MPI_Comm comm, MPI_Request *request)
```

C. OpenMP (Maximum 10 points)

Write OpenMP code to integrate a function with the Trapezoidal rule,

$$f(x) \approx h[f(x_0 = a)/2 + f(x_1) + f(x_2) + ... + f(x_{(n-1)}) + f(x_n = b)/2] \quad ,$$

given a, b, n and f(x). In the above, the subinterval length h is determined by a, b and n. The following OpenMP pragmas and functions might be useful:

```
#pragma omp critical

#pragma omp parallel if(scalar-expression) num_threads(integer-expression) default(shared|
none) private(list) firstprivate(list) shared(list) copyin(list) reduction(redution-
identifier:list) proc_bind(master|close|spread)

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

#pragma omp parallel for if(scalar-expression) num_threads(integer-expression)
default(shared|none) private(list) firstprivate(list) shared(list) copyin(list)
reduction(redution-identifier:list) schedule(kind[,chunk_size]) proc_bind(master|close|
spread) collapse(n) ordered

void omp_set_num_threads(int num_threads);

int omp_get_num_threads(void);

int omp_get_max_threads(void);

int omp_get_thread_num(void);
```

D. CUDA (Maximum 10 points)

Below is a program that creates a simple 2D image of waves on an ocean.

```
#include <stdio.h>
#include <math.h>

# define BLOCK_SIZE  8
# define GRID_SIZE   2

// **** Missing CUDA kernel ****

int main()
{
    float *h_image, *d_image;
    int size;

    size = BLOCK_SIZE*GRID_SIZE;
    h_image = (float *)malloc(size*size*sizeof(float));
    cudaMalloc(&d_image, size*size*sizeof(float));

    dim3 dimBlock( BLOCK_SIZE, BLOCK_SIZE );
    dim3 dimGrid( GRID_SIZE, GRID_SIZE );

    waves<<<dimGrid, dimBlock>>>(d_image, size);
    cudaMemcpy(h_image, d_image, size*size*sizeof(float), cudaMemcpyDeviceToHost);

    for (int x=0; x<size; x++) {
        for (int y=0; y<size; y++)
            printf("%.2f ", h_image[x*size+y]);
        printf("\n");
    }

    cudaFree( d_image );
    free( h_image );

    return EXIT_SUCCESS;
}
```

Both the height and the width of the image is “size” and only one float is used for each pixel. Implement the missing CUDA kernel using the cos() and sin() functions so that there are 4 times as many waves in the x direction than in the y direction.

E. OpenCL (Maximum 10 points)

Implement an OpenCL kernel that simulates the movement of particles in one dimension. A session when running the complete program could for instance look like this:

```
$ ./particles
positions before: 38.88 25.70 9.60 13.33 3.98 18.50 15.32 45.67 4.05 42.59
positions after: 38.77 25.99 9.24 12.93 -11.97 18.99 15.65 46.37 18.86 42.79
```

The program only performs one iteration. One can for instance see that the two particles with the positions 3.98 and 4.05 are very close to each other and that they repel each other, so that their end position is far from their original position.

The force between two particles is calculated by the expression $1/(x_0 - x_i)$, where x_0 is the position of the particle to be moved and x_i is the position of the other particle, as shown in the figure below:

The kernel must loop through all particles and add the force contributions from the other particles to find the resultant force on a particle. The distance that each particle is moved in an iteration is equal to its resultant force. If two particles have the same position then the corresponding force contribution is ignored (to avoid division by 0).

Replace the question marks with the missing code and/or keywords:

```
? void move_particles(? float *x, int n)
{
  ?
}
```