NTNU
Kunnskap for en bedre verden

KANDIDAT

10034

PRØVE

TDT4200 1 Parallelle beregninger

Emnekode	TDT4200
Vurderingsform	Skriftlig eksamen
Starttid	12.12.2022 15:15
Sluttid	12.12.2022 19:15
Sensurfrist	12.01.2023 23:59
PDF opprettet	04.12.2023 11:14

Front Page

Oppgave	Tittel	Oppgavetype
ℹ	Front Page / Forside	Informasjon eller ressurser

Section 1 -- True/False --- Note: Negative scores for wrong answers!

Oppgave	Tittel	Oppgavetype
1	Performance - T/F	Sant/usant
2	Fastest memory - T/F	Sant/usant
3	Caches T/F	Sant/usant
4	Heap T/F	Sant/usant
5	Superscalar - T/F	Sant/usant
6	Amdahl´s Law T/F	Sant/usant
7	FFT & Bit-rev. - T/F	Sant/usant
8	Switch stmt -- T/F	Flervalg
9	Global Sum T/F	Sant/usant
10	MPI basic -- T/F	Sant/usant
11	MPI derived datatype - T/F	Sant/usant
12	MPI Collectives & tags - T/F	Sant/usant
13	Branching - T/F	Sant/usant
14	Pthread mutex - T/F	Sant/usant
15	Thread safety of strtok T/F	Sant/usant
16	CUDA Warps - T/F	Sant/usant
17	CUDA thread block T/F	Sant/usant

18	CUDA - Register spilling - T/F	Flervalg
19	HIP and OpenCL	Sant/usant
20	Comments to T/F questions	Langsvar

Section 2 - MPI & Distributed Memory

Oppgave	Tittel	Oppgavetype
21	MPI - Make efficient	Langsvar
22	MPI - What is wrong?	Langsvar
23	Describe MPI error	Langsvar
24	MPI_Cart_shift	Fyll inn tekst
25	MPI Deadlock?	Langsvar

Section 3 - Shared Memory, OpenMP, PThreads, etc

Oppgave	Tittel	Oppgavetype
26	(8%) OpenMP	Sammensatt
27	(3%) Parallel prog.	Flervalg (flere svar)
28	(2%) Looping and efficiency	Flervalg
29	(2%) Latency & Bandwidth	Flervalg
30	(2%) Elster´s Algorithm	Flervalg (flere svar)
31	PageRank	Langsvar
32	PageRank and Cilk	Flervalg

Section 4 - CUDA

Oppgave	Tittel	Oppgavetype

33	CUDA Programming: Grayscale Image Convolution	Programmering
34	Grayscale Blur: CUDA Grid Calculation	Flervalg (flere svar)
35	Grayscale Blur: Grid Size Efficiency	Langsvar
☑	Grayscale Blur: Further Optimization (10pts)	Skjema
36	CUDA Blurr grading	Langsvar

COURSE REFLECTIONS

Oppgave	Tittel	Oppgavetype
37	Course reflections - multiple choice	Flervalg (flere svar)
38	Bonus question and comments	Langsvar

1 Performance - T/F

Processor speed is more important than data locality with respect to performance
Select an alternative (-0.5 for wrong answer):

- ○ True
- ◉ False

Maks poeng: 1

2 Fastest memory - T/F

L1 cache is the fastest memory available on Intel processors
Select an alternative (-0.5 for wrong answer):

- True
- False

Maks poeng: 1

3 Caches T/F

There are generally **fewer conflict misses** in a **set-associative cache** than a **direct-mapped cache.**
Select one alternative (-0.5 for wrong answer):

- False
- True

Maks poeng: 1

4 Heap T/F

The heap is used to keep track of active function parameters
Select one alternative (-0.5 for wrong answer):

- True
- False

Maks poeng: 1

5 Superscalar - T/F

Superscalar performance is typically due to caching
Select an alternative (-0.5 for wrong answer):

- True
- False

Maks poeng: 1

6 Amdahl´s Law T/F

Amdahl's Law is named after Gene Amdahl in the 1960s who observed that the serial parts quickly dominate the execution times when parallelising codes. For instance, if the serial part of the code takes 10% of the time, Amdahl's Law tells you that

$T_{parallel} = \frac{0.9 \cdot T_{serial}}{p} + 0.1 \cdot T_{serial} = \frac{18}{p} + 2$ for $T_{serial} = 20$ seconds
and $\text{Speedup} = \frac{T_{serial}}{T_{parallel}} = \frac{20}{\frac{18}{p}+2}$.

As **p** gets larger, the Speedup approaches 20/2 = 10, even for many thousands of cores!

In general. if a fraction **r of our serial program remains unparallizable, Amdahl´s Law state that we cannot get a better speedup that 1/r!**

Thus, if the serial part of our program takes only 5 % of the time, the max speed-up is 20.
Select one alternative (2 pts for correct answer, -1 for wrong):

- False
- True

Maks poeng: 2

7 FFT & Bit-rev. - T/F

Both the FFT and Elster's Bit-reversal algorithm are O (N log N)
Select an alternative (-0.5 for wrong answer):

- True
- False

Maks poeng: 1

8 Switch stmt -- T/F

When optimizing a **switch statemen**t for best performance one should sort the statements in ascending (increasing) order with the amount of time they take to compute.
Select one alternative (-0.5 pts for wrong answer)

- True
- False

Maks poeng: 1

9 Global Sum T/F

Parallelizing a global a sum is often done in a tree structure, especially on hypercubes.
Select one alternative (-0.5 for wrong answer) :

- True
- False

Maks poeng: 1

10 MPI basic -- T/F

MPI programs are Single Program Multiple Data (SPMD) rather than SIMD or MIMD (Flynn's Taxonomy)

Select an alternative (-0.5 for wrong answer) :

- (•) True
- ( ) False

Maks poeng: 1

11 MPI derived datatype - T/F

MPI_Type_create_struct is used to build an **MPI derived datatype**

Select one alternative (-0.5 for wrong answer):

- ( ) False
- (•) True

Maks poeng: 1

12 MPI Collectives & tags - T/F

MPI collectives may use tags

Select an alternative (-0.5 for wrong answer):

- (•) False
- ( ) True

Maks poeng: 1

13 Branching - T/F

Branching may lead to performance issues on both CPUs and GPUs
Select an alternative (-0.5 for wrong answer) :

- True
- False

Maks poeng: 1

14 Pthread mutex - T/F

Variables of type **pthread_mutex_t** need to be **initialized** by the system before they are used
Select one alternative (-0.5 for wrong answer):

- True
- False

Maks poeng: 1

15 Thread safety of strtok T/F

The C string library function **strtok** splits an input string into substrings. It can be called recursively. Implementing it by using a static **char*** variable that refers to the string that was passed on the first call, is **thread safe**.
Select one alternative (-0.5 for wrong answer) :

- False
- True

Maks poeng: 1

16 CUDA Warps - T/F

CUDA Warps use the SIMT/SIMD model
Select an alternative (-0.5 pts for wrong answer)

- (•) True
- ( ) False

Maks poeng: 1

17 CUDA thread block T/F

A CUDA **thread block** may be distributed across several SMs.
Select one alternative (-0.5 for wrong answer) :

- (•) False
- ( ) True

Maks poeng: 1

18 CUDA - Register spilling - T/F

Since an SM may be able to run more than one warp at the same time, performance overall may improve despite register spilling affecting single thread´s performance.
Select one alternative (-0.5 pts for wrong answer) :

- (•) False
- ( ) True

Maks poeng: 1

19 HIP and OpenCL

HIP and OpenCL can both be used on systems with AMD GPUs. CUDA is closer to HIP than OpenCL.

Select one alternative (-0.5 for wrong answer):

- (●) True
- ( ) False

Maks poeng: 1

20 Comments to T/F questions

If you find one or more of the questions from this section vagues, you may explain one or more of your T/F answers below. Make sure to refer to the T/F problem´s title. We may or may not consider these explainations.

Reminder: Use of Google, ChatGTP or similar tools are not allowed during this exam.

For task 1: Location, Location, Location. read/write speeds are the bottleneck of modern day computing

For task 5: Assuming Superscalar Performance refers to the use of multistage pipelining, and not Superlinear Speedups. Because Superlinear Speedups are usually due to more efficient caching

For task 13: Although CPU's use Branch Prediction as a remedy for branching, people who write compilers generally want to avoid frequent branching. On GPU's branching is particularly costly when it causes each of the branching alternatives to be executed serially (in groups). Hence, branching is a slowdown

Ord: 90

Maks poeng: 10

21 MPI - Make efficient

How would you make the code below more efficient?

```c
int rank, comm_size;
MPI_Comm_rank ( MPI_COMM_WORLD, &rank );
MPI_Comm_size ( MPI_COMM_WORLD, &comm_size );
double data;
if ( rank == 0 ) {
        data = 3.14;
        for ( int i = 1; i < comm_size; ) {
                MPI_Send ( &data, 1, MPI_DOUBLE, i, 0, MPI_COMM_WORLD
);
        }
} else {
        MPI_Recv ( &data, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE );
}
```

Fill in your answer here by giving showing a more efficient code + explaination.

int rank, comm_size;
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_size (MPI_COMM_WORLD, &comm_size);
double data;

if (rank == 0) {
  data = 3.14; //assuming rank 0 needs to "calculate" the value before it broadcasts it
}

int status = MPI_Bcast (&data, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
//Instead of using Send/Recv to send the data to all the other nodes, we can use broadcast instead to send to all of them at once. We don't need to check for which rank we are here because MPI_Bcast is default set to receive data unless the rank matches the root rank in the BCast command.

Ord: 100

Maks poeng: 6

22 MPI - What is wrong?

```
### What is wrong in this code snippet from an MPI program that tries
### to broadcast from rank 0 to all other processes?
int rank;
MPI_Comm_rank ( MPI_COMM_WORLD, &rank );
int data = 0;
if ( rank == 0 ) {
        data = 3;
        MPI_Bcast ( &data, 1, MPI_INT, 0, MPI_COMM_WORLD );
}
```

Fill in your answer here

There is no need to check that you are the "correct rank" to be able to receive from MPI_Bcast.
As a result, only rank 0 reaches the line with MPI_Bcast, and none of the other ranks are able to receive the broadcast. Resulting in a gridlock.

Ord: 46

Maks poeng: 5

23 Describe MPI error

```
### What is the error in this code snippet from an MPI program?
int rank;
MPI_Comm_rank ( MPI_COMM_WORLD, &rank );
float data;
if ( rank == 0 ) {
        data = 50.5;
        MPI_Send ( &data, 1, MPI_FLOAT, 1, MPI_ANY_TAG, MPI_COMM_WORLD
);
} else if ( rank == 1 ) {
        MPI_Recv ( &data, 1, MPI_FLOAT, 0, MPI_ANY_TAG, MPI_COMM_WORLD,
MPI_STATUS_IGNORE );
}
```

Fill in your answer here

First: Using MPI_Sendrecv would be easier.

I can't seem to find anything else.

Obviously this only sends from rank 0 to rank 1, and doesn't scale, but that isn't something that's fundamentally wrong.

Ord: 33

Maks poeng: 4

24 MPI_Cart_shift

The following unfinished MPI code should initialise a cartesian communicator and retrieve information about the position and neighbors of each rank.

Fill in the blanks so that:

- The cartesian communicator *cart* is **periodic in the x axis.**
- The variables *north, south, west,* and *east* are set to the rank of the north, south, west, and east neighbours respectively.

The signature for MPI_Cart_shift:

MPI_Cart_shift(MPI_Comm communicator, int direction, int displacement, int *source, int *destination);

```
#include <mpi.h>
int main ( int argc, char** argv )
{
        MPI_Comm cart;
    int rank, comm_size;
        int dims[2] = {0, 0}
        int periodic[2], coords[2];
        int north, south, west, east;
        MPI_Init ( &argc, &argv );
        MPI_Comm_size ( MPI_COMM_WORLD, &comm_size );
        MPI_Dims_create ( comm_size, 2, dims );
        periodic[0] = _; // Fill in the blank
        periodic[1] = _; // Fill in the blank
        MPI_Cart_create ( MPI_COMM_WORLD, 2, dims, periodic, 0, &cart
);
        MPI_Cart_shift ( cart, _, _, _, _ ); // Fill in the blank
        MPI_Cart_shift ( cart, _, _, _, _ ); // Fill in the blank
    MPI_Comm_rank ( cart, &rank );
    MPI_Cart_coords ( cart, rank, 2, coords );
        ...

        MPI_Finalize();
        return 0
}
```

periodic[0] = 0

periodic[1] = 1

MPI_Cart_shift(cart, 1 , dims[1] - 1 ,

&rank , &west);

MPI_Cart_shift(cart, 1 , dims[1] - 1 ,

%rank , &east);

Maks poeng: 10

25 MPI Deadlock?

```
#include <mpi.h>
int main ( int argc, char** argv )
{
        ...

        int recv_buffer[40];
        int send_buffer[10];
        for ( int i = 0; i < 10; i++ ) {
                send_buffer[i] = i;
        }
        if ( coords[0] != 0 ) {
                MPI_Sendrecv ( send_buffer, 10, MPI_INT, north, 0,
&recv_buffer[0], 10, MPI_INT, north, 0, cart, MPI_STATUS_IGNORE );
        }
                MPI_Sendrecv ( send_buffer, 10, MPI_INT, south, 0,
&recv_buffer[10], 10, MPI_INT, south, 0, cart, MPI_STATUS_IGNORE );
        if ( coords[1] != 0 ) {
                MPI_Sendrecv ( send_buffer, 10, MPI_INT, west, 0,
&recv_buffer[20], 10, MPI_INT, west, 0, cart, MPI_STATUS_IGNORE );
    }
        MPI_Sendrecv ( send_buffer, 10, MPI_INT, east, 0,
&recv_buffer[30], 10, MPI_INT, east, 0, cart, MPI_STATUS_IGNORE );
        ...
}
```

Assume the cartesian communicator is set up as described in the MPI_Cart-Shift problem. Above is code for the processes to communicate with their immediate neighbours.

Will this code lead to a deadlock? Why/why not? State your assumptions. Fill in your answers here:

The first thing I notice is that all of the Sendrecv commands have the same destination as source, which I don't know if is an intended mistake or not. If it is intentended, there would be no deadlocks, as the node is only communicating with itself and therefore only dependent on itself.

Assuming this is not the case, we need to discuss the situation of a deadlock caused by a race condition.

Assuming you have a subgrid of 4 nodes:

a b

c d

There is a (rare) possibility that a->b (Meaning sendrecv from a to b) happens at the same time as b->c, c->d, and d->a.

Ord: 107

Maks poeng: 5

26 (8%) OpenMP

(2%) In OpenMP there is no guarantee of fairness in mutual exclusion constructs. This means that it's possible that a thread can be blocked forever when using a #pragma omp critical inside a while loop.

Select an alternative

- (●) True
- ( ) False

(2%) OpenMP threads share stack and program counter

Select an alternative

- (●) False
- ( ) True

(2%) False sharing occurs when threads use data in the same cache line, but do not use the same variable when accessing the data, so that the behaviour of the thread with respect to memory access is the same as if there were sharing a variable.

How can false sharing be avoided during matrix-vector multiplication?

Select one alternative

- ( ) Avoid padding the resulting y vector.
- ( ) It is not possible to avoid false sharing when doing matrix multiplication in OpenMP
- ( ) Make sure the resulting y vector is small enough
- (●) Have each thread use private storage during multiplication loop, then update shared storage when done

(2%) Why does the following code exhibit poor performance on 4 or more cores?

```
#pragma omp parallel for shared(a,b,sum) private(I,tmp)
for (i = 0; i < n; i++) {
  tmp = a[i] * b[i];
  #pragma omp atomic
  sum = sum + tmp;
}
```

Select one alternative

- 4 cores are too many for parallelising global sums
- synchronisation need to be added
- #pragma omp for needs to use global tmp
- #pragma atomic serialises the summation

Maks poeng: 8

27 (3%) Parallel prog.

Writing efficient parallel programs usually involve
Select one or more alternatives:

- coordinating the work of the cores
- communications among the cores
- load balancing
- synchronisation of the cores

Maks poeng: 3

28 (2%) Looping and efficiency

Given an array defined as double m[2048][2048] and the following two snippets 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.

Weight: 2% (-0.5% for wrong answer)

Which is the most efficient option?

- (●) 1
- ( ) 2
- ( ) They are equally efficient

Maks poeng: 2

29 (2%) Latency & Bandwidth

What is the difference between **Latency** and **Bandwidth?**

Select one alternative:

- ( ) a) Latency reflects how long the total communication is, whereas bandwidth the communication rate
- (●) b) Latency reflects that time elapsing between a source starting to send data until the data starts arriving at destination, whereas bandwidth is the rate which a link can transmit data
- ( ) c) Latency is how long it takes before the message is ready to be sent, whereas bandwidth is how long it takes to send the message
- ( ) a) and b) above

Maks poeng: 2

30 (2%) Elster´s Algorithm

One or more of the following is true for Elster´s Algorithm
Select one or more alternatives that are TRUE regarding Elster´s algorithm:

- [ ] Uses doubly nested loops
- [ ] Loop controls can be done with shift operations
- [ ] O(N log N)
- [x] O(N) with a high scaling factor due to complexity
- [ ] Linear with a low scaling factor
- [ ] May have tail recursion

Maks poeng: 2

31 PageRank

Larry Page and Sergey Brin developed PageRank at Stanford University in 1996 as part of a research project about a new kind of search engine. They later founded a company.

What was the name of the company?

The mathematics of PageRank are entirely general and apply to any graph or network in any domain. The eigenvalue problem behind PageRank's algorithm was independently rediscovered and reused in many scoring problems.

Mention at least two other use of PageRank (other than for search engines) and why parallelzing PageRank may or may not be important for that application.

Fill in your answer here

PageRank could also be used in computation of eigenvectors and for complex matrix calculations.

Ord: 14

Maks poeng: 4

32 PageRank and Cilk

Cilk, like OpenMP is a general purpuse paralle programming language that is based on C/C++ that allows for multithreading. OpenCilk 2.0 was released in July 2022.

In the following PageRank code, the variables contribution and rank are arrays of doubles, and in_degree and out_degree are arrays of integers. neighbor is a two dimensional array that stores the edges. neighbor[i][j] is the jth neighbor of the ith node.

```
1  void pagerank(double * rank, double * contribution,
2                int ** neighbor, int * in_degree,
3                int * out_degree, int num_vertices) {
4      for (size_t iter = 0; iter < 10; iter++) {
5          cilk_for (size_t i = 0; i < num_vertices; i++){
6              for (int j = 0; j < in_degree[i]; j++){
7                  rank[i] += contribution[neighbor[i][j]];
8              }
9          }
10         for (size_t i = 0; i < num_vertices; i++){
11             contribution[i] = rank[i]/out_degree[i];
12             rank[i] = 0.0;
13         }
14     }
15 }
```

For each of the following code modifications designed to improve performance, select the appropriate option to specify whether it is safe to make the indicated change, whether it is safe if a reducer is used, or whether it is unsafe. **Note: “safe” means that the output must be exactly the same as for the original code. You also get negative scores (-1.5/2) for each wrong answer.**

A document describing cilk_for is provided under PDF-dokument, but should not be need to answer correctly.

Replacing the **for** in line 4 with **cilk_for** is
Select one alternative

- Safe
- Unsafe

Replacing the **for** in line 6 with **cilk_for** is
Select one alternative

- Safe
- Unsafe

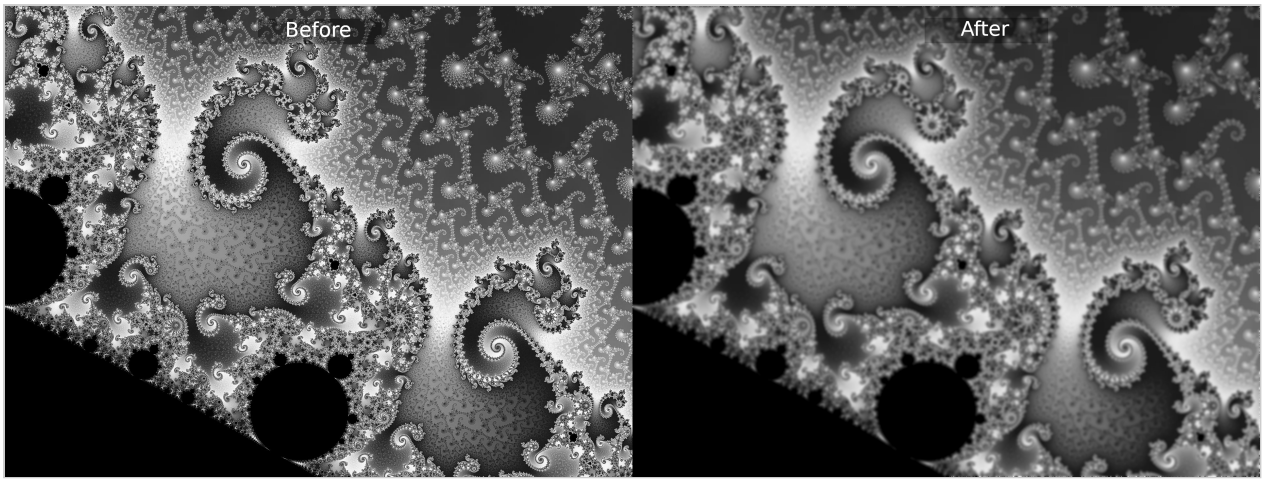
Replacing the **for** in line 10 with **cilk_for** is

Select one alternative:

- Unsafe
- Only safe with reducer
- Safe

Maks poeng: 6

33 CUDA Programming: Grayscale Image Convolution



The *blur* function given in the attached PDF iterates over a *GrayscaleImage* and updates the intensity values of each pixel with the average of itself and its eight neighbors.

$$l(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} l(x+i, y+i)$$

A picture before and after blurring the image over 20 iterations can be seen in the illustration:

Each pixel is an *unsigned char* with values ranging from 0 (black) to 255 (white).
Rewrite the **blur** function into a CUDA kernel named **blur_gpu**.

It should:
- Be callable from host
- Process **exactly one** pixel per thread.

```
typedef struct {
    int width:
    int height;
    unsigned char* data;
} GrayscaleImage

__global__
void
blur_gpu(GrayscaleImage *image, GrayscaleImage* result)
{
    int width = image->width;
    int height = image->height;
    unsigned char *data = image->data;
    unsigned char *result_data = result->data;

    unsigned int sum = 0;

    //get the global index
    // The index of the pixel at coordinates (i,j) //Assuming row major
    size_t pixel_index =
        blockIdx.y * blockDim.y * gridDim.x +
        threadIdx.y * gridDim.x  + threadIdx.x;

    size_t pixel_index_north =
        blockIdx.y * blockDim.y * gridDim.x +
        (threadIdx.y - 1) * gridDim.x  + threadIdx.x;
    size_t pixel_index_south =
        blockIdx.y * blockDim.y * gridDim.x +
        (threadIdx.y + 1) * gridDim.x  + threadIdx.x;

    if (threadIdx.y > 0 && threadIdx.y < height - 1 && threadIdx.x > 0 && threadIdx.x < width - 1)
        {
        sum += data[pixel_index_north - 1] + data[pixel_index_north] + data[pixel_index_north + 1];
        sum += data[pixel_index - 1] + data[pixel_index] + data[pixel_index + 1];
```

```
34          sum += data[pixel_index_south - 1] + data[pixel_index_south] + data[pixel_index_south + 1];
35      }
36      float divisor = 1.0f / 9.0f;// Write the resulting pixel's value to the output image
37
38      result_data[pixel_index] = (unsigned char) (divisor * sum);
39
40
41  }
42
43
44
```

Maks poeng: 10

34 Grayscale Blur: CUDA Grid Calculation

Assume thread block dimensions of $(32, 32, 1)$. Which of the following alternatives will calculate a grid size that is guaranteed to process the entire image of size *width* x *height*? Select **at most** three alternatives. You will also be given a negative point for each wrong answer.

Select one or more alternatives:

- [x] { width / 32 + 1, height / 32 + 1, 1 }
- [ ] { floor((float) width / 32.0f, floor((float) height / 32.0f, 1 }
- [ ] { width / 32, height / 32, 1 }
- [x] { ceil ((float) width / 32.0f), ceil((float) height / 32.0f, 1 }
- [ ] { ceil(width / 32), ceil(height / 32), 1 }

Maks poeng: 3

35 Grayscale Blur: Grid Size Efficiency

Consider the previous question (Grayscale Blur: CUDA Grid Calculation).
Of the alternatives you picked, which method is the most efficient?
State the alternative and describe **briefly** why you think this is the case.
State your assumptions.

Fill in your answer here

In general, we do not want to generate bigger grid dimensions than needed.

I'll assume that in terms of integer calculations; 33/32 = 1, 31/32 = 0

For the example:
{ width / 32 + 1, height / 32 + 1, 1 }
We see that if either width or height are perfectly divisible by 32, we would get an extra column/row in the grid

With:
{ ceil ((float) width / 32.0f), ceil((float) height / 32.0f, 1 }
...we see that we only use extra grid if the dimensions are not perfectly divisible by 32.

Hence, the most efficient alternative is:
{ ceil ((float) width / 32.0f), ceil((float) height / 32.0f, 1 }

Ord: 119

Maks poeng: 8

Grayscale Blur: Further Optimization (10pts)

To iteratively blur the image, the kernel may be called multiple times from the host, synchronizing and swapping image pointers between each iteration.

Instead of iteratively calling the kernel multiple times, we would like the kernel to be able to perform all iterations before terminating.

We would like to change this:

```
int main() {
    // ... setup ...
    for ( int i = 0; i < NUM_ITERATIONS; i++ ) {
        blur_gpu<<<grid_dim, thread_dim>>>(d_img, d_result);
        // ... error handling ...
        swap((void **) &d_img, (void **) &d_result);
    }
}
```

Into this:

```
int main() {
    // ... setup ...
    blur_gpu<<<grid_dim, thread_dim>>>(d_img, d_result,
NUM_ITERATIONS);
}
```

How can the kernel be rewritten to perform all iterations on the GPU? (The kernel should process all iterations before terminating). Your answer should briefly summarize the change(s) you would have to make.

the kernel can perform iteratively (with a loop), but should include a barrier /synchronize at the end of the loop so that that the result would still be correct.
The swap function would be at the end of the loop, but also an additional time outside it to change back to the correct pointers before writing to the buffer.

What challenges are you likely to encounter when rewriting the kernel? Shortly summarize the challenges and why they might cause problems if left unhandled. (Hint: There are dependencies between iterations.)

As stated by the hint, there are dependencies between iterations, which means a barrier is necessary. Each kernel is self contained, which is why this wasn't an issue in the original code. If this barrier is not implemented, we would have a race condition, and cause nondeterminism.

What could be used to remedy the challenge(s)? (Hint: Problem Set 5 - Graded CUDA)

A grid synchronization. Alternatively a custom synchronization / custom dependency where each block is only dependent on a neighbouring block, but considering the complexity cost and the chance for human error as well as how small the speedup would be, a grid synchronization seems like the better solution

The above will be scored as part of the next problem, where you may add further comment.

The following questions do not need to be answered to get a full score, but are meant for further thought. No extra points will be given!

Fill in your answer here

Does your solution impose constraints on your implementation? If so, shortly describe the constraint(s).

If you answered that there are constraints in your implementation. Shortly describe, in words, how the kernel can be rewritten such that it works for any image size.

What performance impact would your approach have? Shortly summarize both positive and negative impacts.

36 CUDA Blurr grading

Grading of CUDA Blurr and additional comments you may have re. the CUDA problems in this section.

Fill any additional comments here

Unsure if we were supposed to call the kernel anywhere, or otherwise do any setup with device side buffers.

This also then means I was unsure if we needed to provide a calculation for grid / block dimensions.

it would be nice to have a CUDA quick reference too, the same way we have an MPI Quick Reference.

Ord: 58

Maks poeng: 9

37 Course reflections - multiple choice

In this problem we are curious about how you selected and reflected over the content and problems sets in this course. No points will be deducted, but rather 3 points given if you answer all of these.

We want to improve and really appreciate your honesty and will not count your answers against you. In fact, we will only look at this problem AFTER the rest of your exam is graded.

Which TWO of the followineg statements are the closest to your MAIN reasons for taking this course? Select TWO only.

- [ ] I found it online/in the course catalogue and found it interesting
- [x] I have to take it as part of my major (e.g. Alg/HPC)
- [ ] A former student strongly recommended this course to me
- [ ] This course seemed to have the most energetic and motivating instructor
- [x] I like courses with lots of programming
- [ ] I like GPUs and wanted to learn more about CUDA
- [ ] It was the most interesting course taught in English
- [ ] I wanted to take a course related to sustainability

What were the BEST 3 aspects of this course? Pick max. 3.

- [ ] Access to real paralle clusters and GPU hardware
- [ ] Graded problem sets that count 50% of final grade
- [x] Learning CUDA programming
- [ ] Enthusiastic instructor
- [x] Interesting problem sets, that were painful but taught me a lot.
- [ ] Interesting problem sets that were highly relevant for sustainability.
- [ ] Clear syllabus and great on-line resources
- [ ] Learning about parallel computing, performance and energy efficiency
- [ ] Great TA support
- [ ] In class instruction, not just videos /remote teaching

Before reading any further, how relevant do you think this course was with respect to sustainability

- [ ] Relevant
- [x] Very relevant
- [ ] A must-take course for people in STEM (science, technology, engineering or math) interested in sustainability!
- [ ] Not relevant
- [ ] Somewhat relevant

Pick the aspect of the course that seems most relevant for sustainability

- [ ] HPC systems use a lot of electric power, so parallel computing is not really sustainable
- [ ] Optimizing for locality, reduces moving of data which is energy costly
- [x] Programming in parallel utilizes the computing systems better, so likely more energy efficient
- [ ] Problem set on shallow water equation that is highly relevant for sustainability

What were the most interesting aspects and/or truthful statement regarding the Problem sets. Select one or more alternatives

- [x] Like learning CUDA the most
- [x] That they were graded so the exam only counts 50% which releaves a lot of stress
- [ ] Working on the same problem, the shallow water equation, in several programming enviroments
- [ ] Liked learning MPI the most
- [ ] The shallow water equations since they relate so closely to sustainability
- [ ] The shallow water equations since they relate so closely to other numerical simulations I expect to work with
- [ ] The problems sets t
- [ ] Liked learning OpenMP/Pthreads the most

Select all the statements that you think mostly aligns with your opinion of TDT4200.

- [x] I learned a lot. Love the power of GPUs and would recomment this class
- [ ] I wish this class had been taught in Norwegian
- [ ] Too much work / time-consuming problem sets!
- [ ] Prefer to study on my own, so did not attend to most classes
- [ ] The problem sets took too much time compared to what I got out of them
- [x] I may recommend this class --- Will have to see how this exams goes...
- [ ] Did not attend most classes due to work conflicts
- [ ] Instructor talked too fast.
- [ ] Would probably not recemmend this class. Content was not as expected
- [ ] Did not attend many classes due to time clash with another course
- [ ] Yes, love courses with enthusiastic instructors that make me feel like coming to class
- [ ] Clear material available -- never felt like I had to come to class
- [ ] Would have been nice if the lectures were at a different time so I could have attended class more often

Maks poeng: 3

38 Bonus question and comments

Comment on how useful was TDT4200 regarding making you aware that computing is related to sustainability.

Also add any further comments or quesitions regarding this exam below.

Please add your reflections here. You can get 2 bonus points for this.

I now know that task parallelism leads to more sustainable computing, as it is energy efficient to use proper load balancing. Additionally, superlinear speedups show that we can use a lower number of total clock cycles to compute something when done in parallel, which clearly means it is also more energy efficient.

Non related to energy efficiency, parallel computations are a much more effective use of computing power.

Comment

I'm really confused as of to what question 24 actually asks me to do

Ord: 83

Maks poeng: 2