# Data-Level Parallelism

## *Bonus Assignment in IS2202 Computer Systems Architecture, 2022*

### Instructions

• The assignment must be solved *individually*.

• Solutions should be properly motivated, with a few short sentences and/or formulas.

• Answer the question in your own words. Copying complete answers from other sources is not acceptable. You may, however, quote other sources as part of an explanation or discussion. In that case, include a proper reference to the source and describe your interpretation.
*Specifically, the description of your interpretation must be longer than the quote.*

• For any material that is not your own: if you quote it or use it in any way, you must include a proper reference. This includes books, research papers, Internet sources, other students, and other material that is not your own.

• A proper reference describes the source clearly and concisely, so that any reader instantly can find the relevant page of the source. Author, title, date, and URL must be present, if available.

• All submitted reports will be automatically checked for plagiarism.

• Solutions may be given in English or Swedish, but not a mix of the languages.

• Only hand in PDF files from computer generated sources. *All text must be machine-readable.* Scanned text is not allowed.

• Solutions must be handed in through Canvas. Submissions by e-mail are not accepted.

• You need to score at least 24 points to qualify for a bonus point at the exam.

• The deadline is given in Canvas. After the deadline, submission is closed and no further submissions are possible.

## Parameters for problems

• You have a Swedish civic registration number (personnummer) of the form YyMmDd-XXXX.

Use the following formulas to calculate the parameters in the problems on the following pages.

• Loop Length L1 =100 + Y+y+M+m+D+d = _____

• Memory Instruction MOH = 8+d = _____ (ccs)

• ADD and MUL instruction DOH = 4+D = _____ (ccs)

• Picture x_size=1920*(1+D) = _____ (pixels)

• Picture y_size=1080*(1+M) =_____ (pixels)

• Number of CUDA threads NrThreads = 32*(1+M) =_____ (Threads)

## Problem 1. Data Level Parallelism - 10 points

Consider the following C-code:

```
for (i = 0; i < L1; i=i+1) {
    C[i] = X[i]* Y[i]-W[i]*Z[i];
    D[i] = X[i]* Z[i]+Y[i]*W[i];
    B[i]=C[i]*C[i]+D[i]*D[i];
}
```

a) Write down the VMIPS code for the loop. The target vector machine has vector length 64. Assume that all variables are double-precision. (4p)

b) How long is the estimated execution time in clock cycles including overhead? Memory instructions has an overhead of *MOH* clock cycles, ADD and MUL instructions have an overhead of *DOH* clock cycles. (4p)

c) Convert the C-code to CUDA code. Use *NrThreads* CUDA Threads per thread block. (2p)

## Problem 2. VMIPS vs GPU - 20 points

The Sobel Operator is used in Image processing to do edge-detection. It applies two separate kernels to calculate the x and y gradients in the image. The length of this gradient is then calculated and normalized to produce a single intensity approximately equal to the sharpness of the edge at that position. The algorithm can be broken down into its constituent steps:

1. Iterate over every pixel in the image
2. Apply the x gradient kernel
3. Apply the y gradient kernel
4. Find the length of the gradient using Pythagoras' theorem
5. Normalize the gradient length to the range 0-255
6. Set the pixels to the new values

The gradient kernels are two 3x3 matrixes:

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

X-gradient kernel

| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

Y-gradient kernel

在CUDA中，由于CUDA核函数在GPU上并行执行，因此它不支持动态多维数组的索引（例如gradient_kernel_x[j + 1][i + 1]），也不支持直接访问主机内存中的数据（例如image[(y + j) * x_size + (x + i)]）。可以通过使用一维数组来存储核函数中使用的数据，并在内核函数中手动计算索引来解决这个问题

CUDA支持多维线程块和网格
需要将二维索引映射到一维索引，以便在一维数组中访问数据

The c-code for applying the gradients (on a gray-scale) image is

```
for (y=1;y<y_size-1;y++) {
   for (x=1;x<x_size-1;x++) {
      pixel_value_x=0.0;
      pixel_value_y=0.0;
      for(j=-1;j<=1;j++) {
         for(i=-1;i<=1;i++) {
            pixel_value_x+=gradient_kernel_x[j+1][i+1]*image[y+j][x+i];
            pixel_value_y+=gradient_kernel_y[j+1][i+1]*image[y+j][x+i];
         }
      }
      // approximate sqrt(x^2+y^2) with |x|+|y|
      pixel_value=abs(pixel_value_x)+abs(pixel_value_y);
      new_pixel_value = 255 * (pixel_value>threshold);
      edge_image[y][x]=new_pixel_value;
   }
}
```

在这种情况下，循环内的操作不需要考虑线程块。这是因为每个线程都将独立地计算图像的一个像素点，而不需要与其他线程进行通信或同步。因此，循环内的 i 和 j 变量只是用来迭代处理每个像素点的周围区域，并不涉及线程块的计算。

where the threshold value decides if the generated gradient length value indicates an edge or not.

a) The target machine is a 32 vector length 32 bit Single Precision VMIPS architecture. Assume that all pixels are 8 bit integers, (but stored as 32 bit integers to allow for the fancy VMIPS addressing modes). Write down the VMIPS code for the Sobel operator. (10p)

b) Write down the CUDA code for a GPU implementation of the gradient function. Use *NrThreads* CUDA Threads per thread block. (5p)

c) The algorithm is applied to an HD-image of size (*x_size* * *y_size*) pixels. Would it be more beneficial to run the algorithm on the VMIPS Machine, or on a GPU using *NrThreads* CUDA threads? Analyze the algorithm and explain the difference between the two implementations in max 10 sentences. (5p)