CSC/ECE 506: Architecture of Parallel Computers
Problem Set 1
Due: Monday, June 2, 2025
Reflection due: Monday, June 9, 2025

Problem 1. (*20 points)* Consider an algorithm that runs for 240 seconds on a uniprocessor system with 60% of the work being parallelizable.

(a) With a multiprocessor system, how many processors would be needed to execute the algorithm in a total execution time of 1/2 the time of the uniprocessor system?

(b) What is the efficiency of the system in part (a)?

(c) Can any multiprocessor system execute the algorithm in 1/3 the time of the uniprocessor system? Explain.

(d) How many processors are needed to execute the algorithm in 108 seconds?

For parts (e), (f), and (g), assume that the algorithm is now improved so that 75% of the previously serial work is now parallelizable.

(e) What fraction of the total work is now parallelizable? What fraction is serial?

(f) How many processors will it take to execute the algorithm in 1/3 the time of the uniprocessor system after this improvement?

(g) What is the maximum speedup in this new system?

Problem 2. *(25 points)* Consider a system that has both CPUs and GPUs. Recall that GPUs have many slow PEs, but CPUs have fewer fast PEs.

Suppose you need to run a simple 2D finite-difference scheme on a matrix of 1024×1024 elements, all of which are 32-bit floating-point numbers. Now, at every step, each element in the matrix is updated by a weighted average of the four adjacent elements,

$$A[i,j\,] \coloneqq A[i,j\,] - w\,(A[i-1,j\,], A[i+1,j\,], A[i,j-1], A[i,j+1\,])$$

(If an element has fewer than 4 neighbors, ignore the non-existent adjacencies.)

The problem is to be run on this heterogeneous system.

	# of PEs	Time taken by a PE to update one element
CPU	4	1 ns
GPU	1024	10 ns

For simplicity's sake, ignore the overhead of thread management and data communication, and consider that all elements can be calculated in parallel without any data dependence.

(a) Based on this information, compute the following for a single sweep of the matrix:

i. (*3 points*) Time required when executed only on the CPU.
ii. (*3 points*) Time required when executed only on the GPU.
iii. (*4 points*) Relative speedup for GPU vs. CPU execution.

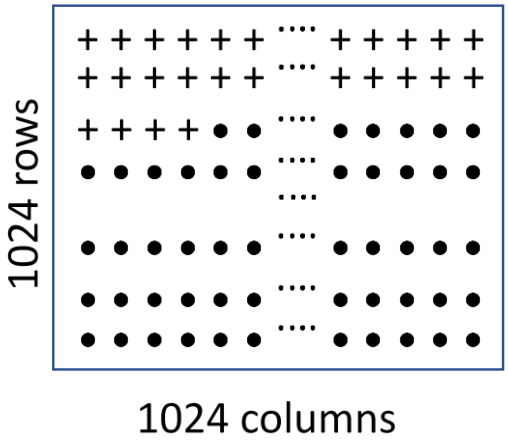
(b) Now, we want to perform the same sweep in a heterogeneous programming model—where all CPU and GPU PEs are used in parallel to achieve the minimum possible execution time.

i. (*3 points*) Calculate the number of elements computed by the CPU and GPU.

ii. (*3 points*) Calculate the total execution time

iii. (*4 points*) Compute the speedup—heterogeneous approach vs. GPU only and vs. CPU only.

iv. (*5 points*) What is the total amount of data (in bytes) exchanged between the CPU and GPU? Hypothesize that the CPU computes the first *N* elements (denoted by +) and the GPU computes the rest (denoted by •), as shown in the diagram at the right.



Problem 3. *(15 points)*Use the 2D finite-difference scheme from Problem 1:

$$A[i, j] := A[i, j] - w\,(A[i-1, j], A[i+1, j], A[i, j-1], A[i, j+1]).$$

This time, assume that the values are 64-bit floating-point numbers. With one element per processor and 1024 × 1024 elements, how much data is communicated per sweep?

Show how to map this computation onto 64 processors to minimize data traffic. Calculate the amount of data that needs to be communicated during each sweep.

This would be implemented by having each processor send data to the processor that is logically "below" it, to the processor that is "above" it, to the processor on the "left" and the processor on the "right," in four separate steps. With modern multiprocessors, it doesn't matter if the processor is actually placed in a 2D mesh, as it takes essentially the same time to send data anywhere in the processor array.

Problem 4. *(15 points) The* following code is a commonly used algorithm in image processing applications.

Consider an image *f* with width=ImageWidth and height=ImageHeight. *f* is a 2D grid of pixels. *k* is a kernel of width=2*w*+1 and height=2*h*+1 where (2*w*+1) < ImageWidth and (2*h*+1) < ImageHeight. The image *f* is processed using the kernel *k* to produce a new image *g* as shown:

```
for y = 0 to ImageHeight do
        for x = 0 to ImageWidth do
                sum = 0
                for i= -h to h do
                        for j = –w to w do
                                sum = sum + k[j,i] * f[x – j, y – i]
                        end for
                end for
                g[x y] = sum
        end for
end for
```

(a) Identify the read-only, R/W non-conflicting and R/W conflicting variables, if the **for** *y* loop is parallelized.

Read only	R/W non-conflicting	R/W conflicting

(b) Identify the read-only, R/W non-conflicting and R/W conflicting variables, if (only) the **for** *i* loop is parallelized. Assume that the **for** *i* tasks for the previous value of *x* must complete before the **for** *i* tasks of the current value of *x* are started.

Read only	R/W non-conflicting	R/W conflicting

(c) Identify the read-only, R/W non-conflicting and R/W conflicting variables, if the **for** *i* loop is parallelized. Assume that the **for** *i* tasks for the previous value of *x* do not have to complete before the **for** *i* tasks of the current value of *x* are started.

Read only	R/W non-conflicting	R/W conflicting