Aleksas Murauskas 260718389

ECSE 420: Parallel Computing

Assignment 1

1. Describe briefly the following terms, expose their cause, and work-around the industry has undertaken to overcome their consequences:
   1. memory wall: The memory wall is the increasing gap between processor and memory performance. This is caused by a mismatch in processor and memory speed improvements, Processor speed doubles every 1.5 years, but memory access speed doubles every ten years. The growing divergence in the rates results in the von Neuman bottleneck. This bottleneck means that a CPU is forced to wait for data to me

The industry has begun using a small cache memory in between processors and the primary data storage in order to close the gap. Caches are much faster, and therefore reduce the time the CPU waits for data. Parallelism at the instruction level can also reduce the CPU’s wait time by performing actions on the memory before the CPU would use it.

* 1. frequency wall: The frequency wall is the inability to increase CPU clock frequency beyond an upper bound. The wall is created by the size and power required to operate the transistors within a processor. Switching time of a transistor is inversely proportional to the size of the transistor. Due to the physical limits in the creation of transistors, clock rates have plateaued at ~3.8 GHz.

To avoid this issue, the industry has increased the number of parallel cores within processors and thread level parallel computing. These systems operate at lower frequency but still increase performance.

* 1. power wall: The power wall is the inability to increase performance due an upper bound of thermal and electrical power in a processor. This is caused by fundamental constraints in power delivery and dissipation. The industry has begun to use forms of cooling the transistors to remove extra heat to improve power dissipation.

1. You should extend Amdahl’s and Gustafson-Barsis bound and make it slightly more realistic. Assuming the fixed overhead o in the communication and the setup of parallel processes, derive the expressions for both bounds that take the overhead into account. Visualize both bounds for a fixed overhead o.
2. There are two dominant models of how parent and children processes relate to each other in a shared address space. In the heavyweight process model, when a process creates the child process, it gets a private copy of the parent's image; that is, if the parent had allocated a variable x, then the child also finds a variable x in it address space that is initialized to the value to the value that the parent had for x when it created the child. However, any modifications that either process makes subsequently to its own copy of x are not visible to the other process. In the lightweight threads model, the child gets a pointer to the parent's image, so that it and the parent now see the same storage location for x. All data that any process or thread allocates is shared in this model, except that on a procedure's stack.
   1. Consider the problem of a process having to reference its process identifier PID in various parts of a program, in different routines called (in a call chain) by the routine at which the process begins execution. How would you implement this in the first model? In the second? Do you need private data per process, or could you do this with all data being globally shared?

In the Heavyweight model, the child program receives a copy of the parent’s image. Multiple Processes are used. A PID can be easily created in the child process’s separate memory, as well as a PID of its parent process.

In Lightweight, children and parents use the same memory space. This can be accomplished with threads. There needs to be private memory to record the PID or use a stack to store copies of the memory as needed by referencing the PID. Each memory input on the stack would correspond to the PID in the procedure stack. Some form of other memory storage is required to properly track PID.

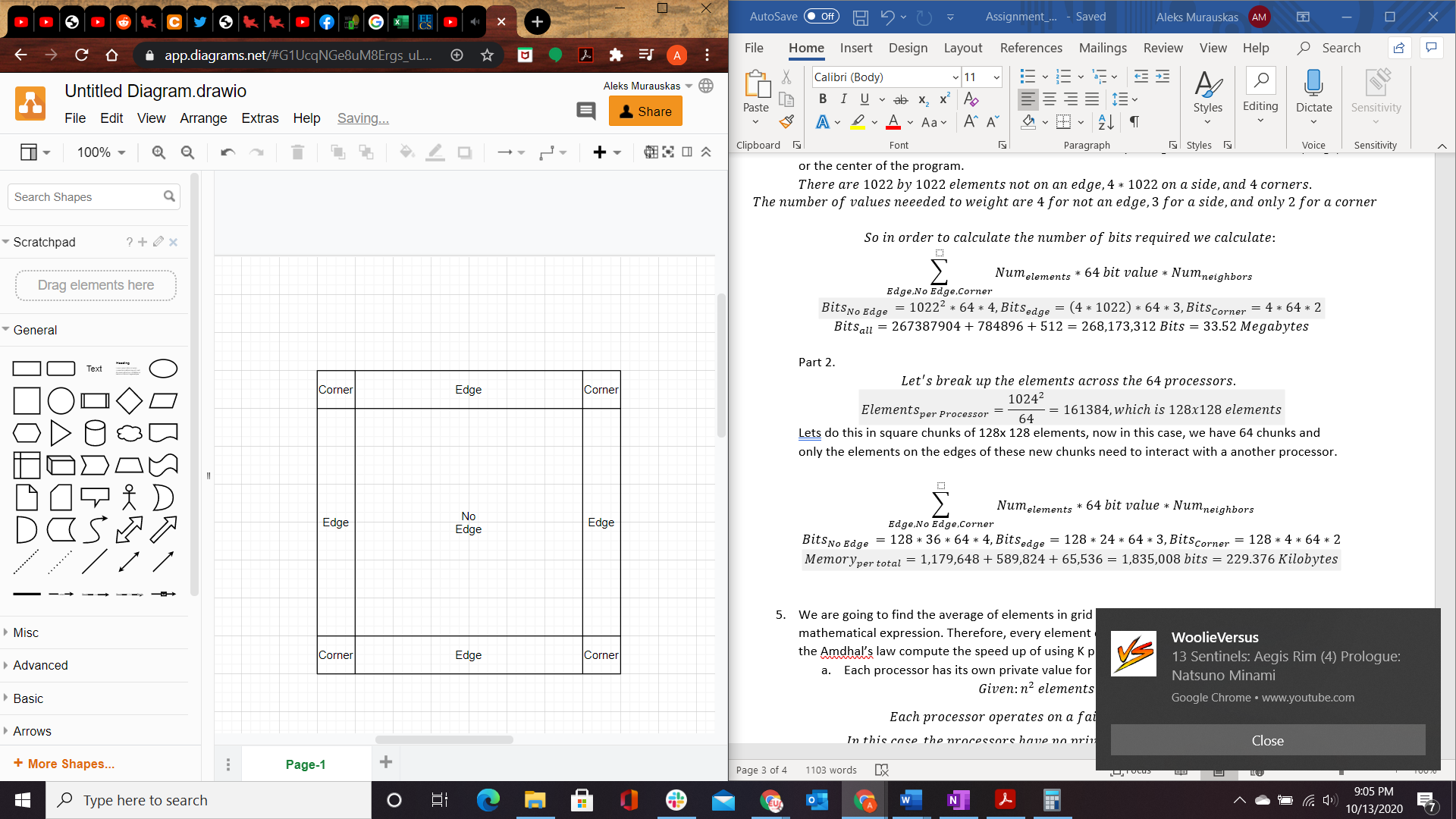
* 1. A program written in the former (process) model may rely on the fact that a child process gets its own private copies of the parents' data structures. What changes would you make to port the program to the latter (threads) model for data structures that are (i) only read by processes after the creation of the child and (ii) are both read and written?

In the case, where we read only. We can simply use a lightweight process, as the children will never change any of the information in the memory, only the parent will. As long as the process stack is properly followed a child should never read incorrect information as the parent is waiting for the children to complete.

In case two, we need to both read and write, the program needs to be careful as a child could overwrite information that is necessary for a another child to use, creating a cascade of errors.

1. Consider a simple 2D finite difference scheme where at each step every point in the matrix is updated by a weighted average of its four neighbors: A[i,j] = A[i,j]-w(A[i-1,j]+A[i+1,j]+A[i,j+1]+A[i,j-1]). All the values are 64 bit floating-point numbers. Assuming one element per processor and 1,024x1024 elements, how much data must be communicated per step? Explain how this computation could be mapped onto 64 processors so as to minimize data traffic. Compute how much data must be communicated per step.

We must remember That there are three distinct cases: Updating an element on a corner, edge, or the center of the program.



Part 2.

Lets do this in square chunks of 128x 128 elements, now in this case, we have 64 chunks and only the elements on the edges of these new chunks need to interact with a another processor.

1. We are going to find the average of elements in grid of (n\*n). Each element of this grid may be a mathematical expression. Therefore, every element of this grid requires computation. Based on the Amdhal’s law compute the speed up of using K processors in these two following situation:
   1. Each processor has its own private value for holding the sum.
   2. The processors have to use one shared value to keep tracking of the sum. That is, every processor should sum its result to the shared variable of sum.

In this case, there are private values, so the addition steps are reduced by a factor K.