

OS – Assignment 2 – Theoretical (& code) Part

Question n1 : Amdahl's Law

- (1) Given a program where 75% of the execution time is parallelizable, calculate the theoretical maximum speedup using 4 processors.
- (2) A program has a parallelizable portion of 60% and a sequential portion of 40%. How many processors are needed to achieve a speedup of 3?
- (3) If a program has 90% of its execution time parallelizable and the remaining 10% is sequential, calculate the speedup achieved with 32 processors.
- (4) For a program where 85% can be parallelized, what is the maximum theoretical speedup if there are an infinite number of processors?
- (5) How does the speedup change for a program with 80% parallelizable code if the number of processors is increased from 4 to 32? Provide a detailed calculation.
-

(1) Formula of speedup given by:

$$Speedup = \frac{1}{(1-P) + \left(\frac{P}{N}\right)}$$

For $P = 0.75$ and $N = 4$ the speedup is $\frac{1}{(1-0.75) + \frac{0.75}{4}} \cong 2.29$. The theoretical maximum speedup with 4 processors is ~ 2.29 .

(2) Number of processors needed to achieve a speedup of 3 with $P = 0.6$.

$$3 = \frac{1}{(1 - 0.6) + \frac{0.6}{N}}$$

$$N = -9$$

As we can see, a speedup of 3 with only 60% of the program being parallelizable code is not possible. When we try to check for maximum speed up, we can see that it's a speedup of 2.5.

(3) $P = 0.9$, $1 - P = 0.1$, $N = 32$

$$\text{thus, speedup} = \frac{1}{0.1 + \frac{0.9}{32}} \cong 7.81$$

Therefore, the speedup achieved with 32 processors is ~ 7.81

(4) $P = 0.85$, $1 - P = 0.15$

$$S(\infty) = \frac{1}{(1 - P) + 0}$$
$$S(\infty) = \frac{1}{1 - 0.85} \cong 6.67$$

Therefore, the maximum theoretical speedup if there are infinite processors is 6.67

(5)

$$S(4) = \frac{1}{(1 - 0.8) + \frac{0.8}{4}} \cong 2.5$$
$$S(32) = \frac{1}{(1 - 0.8) + \left(\frac{0.8}{32}\right)} \cong 4.44$$

The speedup change : $\frac{S(32)}{S(4)} = \frac{4.44}{2.5} \cong 1.776$

Therefore, the speedup changes by approximately 1.776 times when increasing the number of processors from 4 to 32 for a program with 80% parallelizable code.

Q2. Process vs. Thread (8 points)

In this question you have to measure how threads are lightweight comparing to processes. Measure how much time it takes to run a simple 'hello world' process and to run a simple 'hello world' thread.

Start to measure just before you launch the process\thread and measure immediately after they finish (remember how to wait for thread termination),

(1) Do the measurements 10 times and write the results in a 10x2 table within the answers PDF file (the source code is not required in this question)

(2) Write an explanation about the results given your current knowledge in threads and processes internals.

(1)

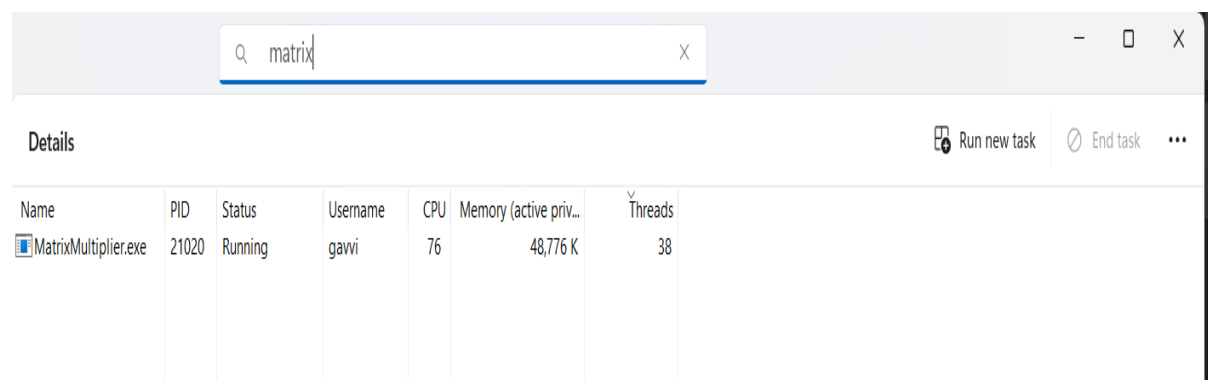
Run #	Process Execution Time (ms)	Thread Execution Time (ms)
1	210	9
2	24	5
3	19	5
4	29	12
5	28	7
6	20	5
7	21	5
8	23	7
9	24	5
10	18	6

(2) Processes, which run independently with their own memory space and resources allocated by the OS, exhibited longer execution times. This is due to significant overhead including memory allocation, context switching, and potentially disk I/O operations. In contrast, threads, sharing memory within the same process, started

swiftly with shorter execution times. Threads' efficiency stems from minimal setup compared to processes, highlighting their advantage for concurrent tasks within a program

(3) The Console.writeline isn't thread safe. When we print out the random numbers we can see that the prints from the threads overwrite the prints from the process. From that fact we can infer that the threads and process don't have a synchronized access to the Console, and their prints overwrite each other.

Q3.

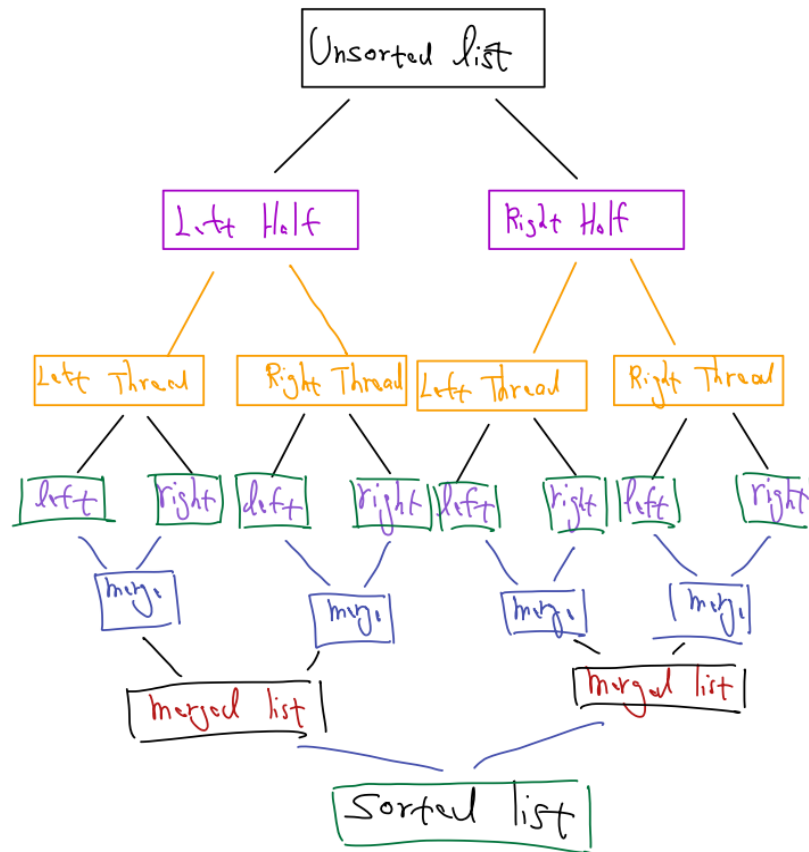


The screenshot shows the Windows Task Manager interface. At the top, there is a search bar with the text 'matrix' and a close button 'X'. Below the search bar, the 'Details' tab is selected. On the right side of the 'Details' tab, there are three buttons: 'Run new task', 'End task', and a three-dot menu. The main area displays a table with the following columns: Name, PID, Status, Username, CPU, Memory (active priv...), and Threads. The table contains one row of data for 'MatrixMultiplier.exe'.

Name	PID	Status	Username	CPU	Memory (active priv...)	Threads
MatrixMultiplier.exe	21020	Running	gawi	76	48,776 K	38

Q4:

- Multi-threaded merge-sort divides a list of strings into halves, sorting each recursively. If the list size is below a threshold ($nMin$), it sorts sequentially to avoid thread overhead. Larger lists spawn two threads to sort halves concurrently. After sorting, results are merged using *string.Compare*. This approach balances thread utilization and sorting efficiency, crucial for large datasets where parallel processing improves performance significantly.
- Example with $nMin = 4$** .(In our test we tried with both low $nMin$ such as 4, and high $nMin$ such as 1000, with higher values we got fast execution times)



c) + d) + e) are in the class submission MTMergeSort.cs
