Multiprocessing Message Passing Application

CLE - Computação em Larga Escala
Assignment 2
May 2022

Mário Silva - 93430
Pedro Marques - 92926

Text Processing - Multiprocessing Implementation

Design and flow of the dispatcher process:

1. Read and process the command line.
2. Broadcast a message with the maximum number of bytes each chunk will have.
3. Obtain chunks and send them to the workers.
4. Wait for a response with the processing results from each worker that it sent a chunk.
5. Store the processing results obtained from the workers' response and repeat step 3 until there are no more chunks to be processed.
6. Send a message to the workers alerting when there isn't more work to be done.
7. Print final processing results.

Design and flow of the worker process:

1. Wait for a broadcasted message from the dispatcher with the maximum number of bytes of each chunk.
2. Wait for data to process.
3. Process the data.
4. Send to the dispatcher the processing results.
5. Repeat step 2 until the dispatcher says there is no more work to be done.

d

Text Processing - Results

Results for processing with **4096 bytes maximum** size **per processing chunk**, total of **20 tries**, using an **8-core 3.7GHz** AMD Ryzen™ 7 2700X Processor**.**

The **parallelization formula**, derived from **Amdahls Law,** is given as: S(n) = 1 / ((1-P) + P/n), where S(n) is the theoretical speedup, P is the **fraction of the algorithm that can be made parallel** and n is the number of CPU cores. We utilized this formula to find the **optimal value of P**, and, obtained, **P=94.5%**.

Input	Number of Processes	Average Elapsed Time (s)	Standard Deviation (s)	Actual Speedup	**Amdahl's Law Speedup (94.5% efficient)**
All Text Files	1	0.00355	0.001149	1	1
	2	0.001666	0.0002470	0.00355/0.001666 = 2.131	1.896
	4	0.000991	0.0001584	0.00355/0.000991 = 3.582	3.433
	8	0.000654	0.0000567	0.00355/0.000784 = 5.428	5.776

Command: *mpiexec -n 8 ./prog1 -f texts/text0.txt -f texts/text1.txt -f texts/text2.txt -f texts/text3.txt -f texts/text4.txt -m 4096*

Matrix Determinant - Multithreading Implementation

Design and flow of the Dispatcher process:

1. Read and process the command line.
2. For every file:
   a. Read the number and order of matrices in file.
   b. For every matrix:
      i. Send the matrix index, order, and the matrix itself.
      ii. Wait for the determinant from each worker.
      iii. Store the results.
3. Signal the workers to finalize.
4. Print results.
5. Finalize.

Design and flow of the Worker processes:

1. Get the current Worker Status.
   a. If no more work to be done, finalize.
2. While there is work to be done:
   a. Receive matrices sent by dispatch.
   b. Calculate the determinant.
   c. Send the results to the dispatcher.

Matrix Determination - Results

Timing results for processing files, total of **20 tries**, using an **8-core 3.7GHz** AMD Ryzen™ 7 2700X Processor.

File	Number of Processes	Average Elapsed Time (s)	Standard Deviation (s)	Actual Speedup	**Amdahl's Law Speedup (88.6% efficient)**
mat512_128 *and* ***mat512_256***	1	2.1678	0.0742	1	1
	2	1.4579	0.0532	2.167/1.4579 = 1.486	1.794
	4	0.6167	0.0376	2.167/0.6167 = 3.514	2.981
	8	0.6167	0.0148	2.167/0.6167 = 4.448	4.449

Command: *mpiexec -n 8 ./prog2 -f mat512_128.bin -f mat512_256.bin*

Conclusion

- ➢ Using **Amdahl’s law**, we can visualize how **a higher level of parallelization improves the performance** of a task at a fixed workload.
- ➢ The solutions developed for each problem have a **different parallelization factor**. The picture below depicts how this difference should affect the overall speedup of the programs.
- ➢ In compliance with this law, our results allow us to conclude that a **higher number of processes represents a better performance**, and tend to **stabilize speedup** with around **32 processors**.