Multicore computing
Project #3 – prob1

Department : Software
Name : Jeong Eui Chan
Student ID : 20195914

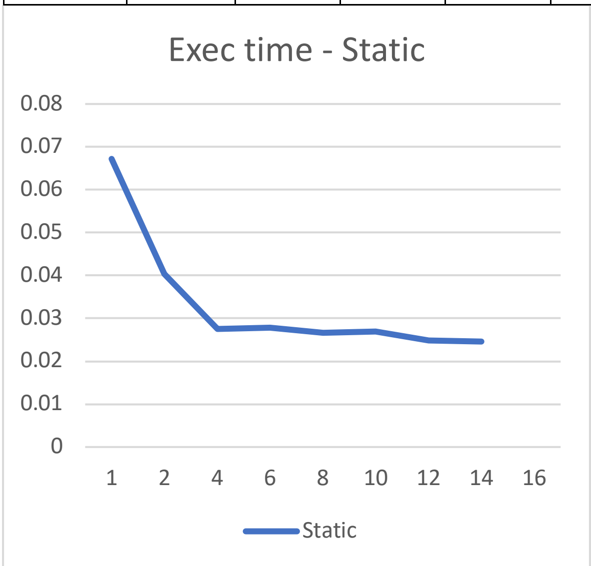
My computer system environment

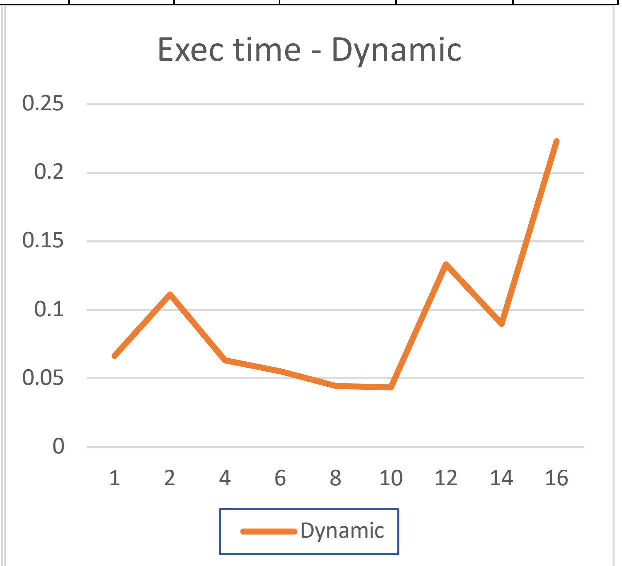
```
Last login: Tue May 23 22:48:50 on ttys000
 jeong-uichan  ~  system_profiler SPHardwareDataType
Hardware:

    Hardware Overview:

      Model Name: MacBook Air
      Model Identifier: MacBookAir10,1
      Model Number: Z1250002VKH/A
      Chip: Apple M1
      Total Number of Cores: 8 (4 performance and 4 efficiency)
      Memory: 16 GB
      System Firmware Version: 8422.100.650
      OS Loader Version: 8422.100.650
      Serial Number (system): FVFFC3ADQ6LT
      Hardware UUID: EE341825-7FB8-52C6-8F1A-E68A1656E926
      Provisioning UDID: 00008103-001539513CEA001E
      Activation Lock Status: Enabled
```

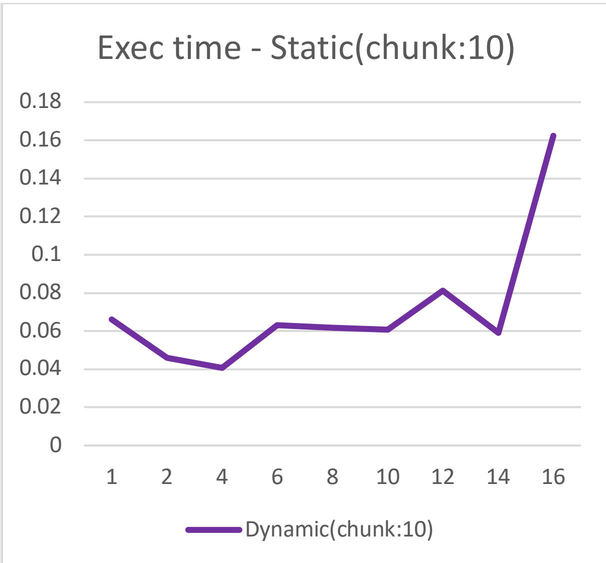
Both Problem1 and Problem2 were performed in an environment with the following specifications: The cpu of my computer uses an Apple M1 chip with 8 cores, consisting of 4 P-cores and 4 E-cores and The maximum clock speed of my cpu is 3.2GH. The memory capacity is 16GB, and the memory type used is LPDDR4. The operating system being used is macOS Ventrura version 13.2.1.

Exec time graph

Exec time	Chunk Size	1	2	4	6	8	10	12	14	16
Static	Default	0.0672	0.0403	0.0276	0.0278	0.0266	0.0270	0.0249	0.0246	0.0243
Dynamic	Default	0.0666	0.1112	0.0633	0.0552	0.0448	0.0435	0.1331	0.0896	0.2228
Static	10	0.0662	0.0401	0.0285	0.0281	0.0300	0.0240	0.0253	0.0252	0.0260
Dynamic	10	0.0662	0.0461	0.0407	0.0631	0.0618	0.0606	0.0811	0.0591	0.1624

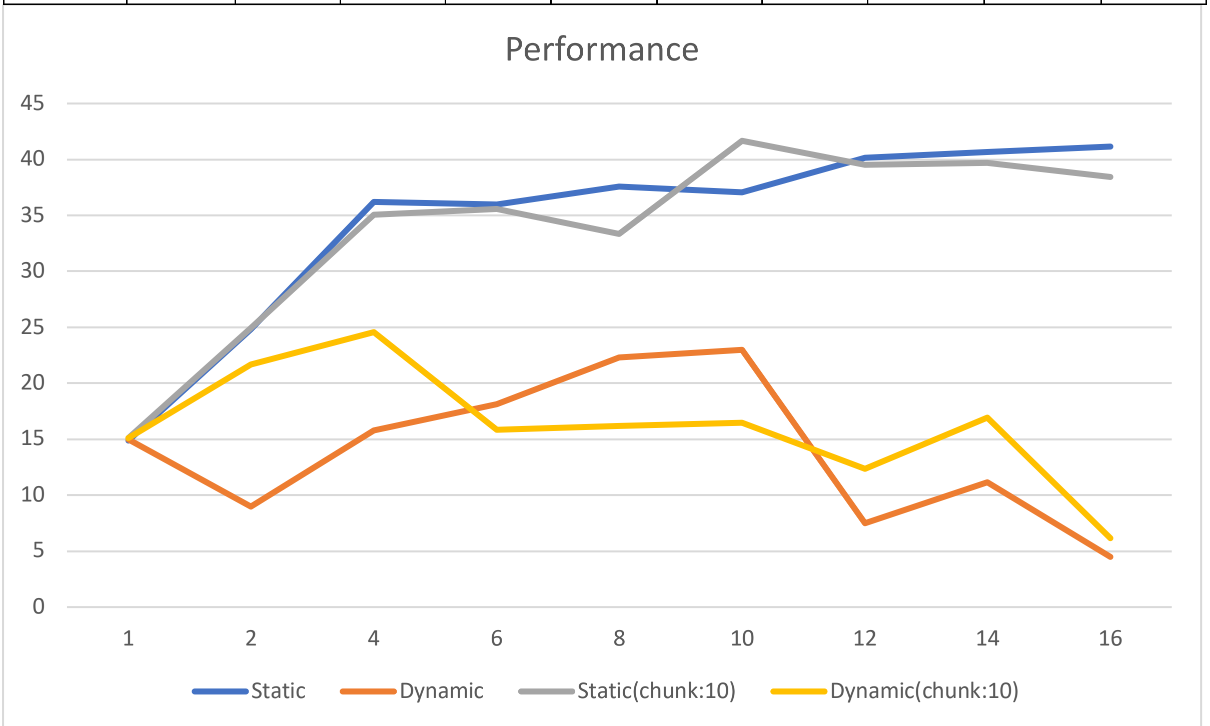






Performance graph

Exec time	Chunk Size	1	2	4	6	8	10	12	14	16
Static	Default	14.881	24.814	36.232	35.971	37.594	37.037	40.161	40.65	41.152
Dynamic	Default	15.015	8.993	15.798	18.116	22.321	22.989	7.513	11.161	4.488
Static	10	15.106	24.938	35.088	35.587	33.333	41.667	39.526	39.683	38.462
Dynamic	10	15.106	21.692	24.57	15.848	16.181	16.502	12.33	16.92	6.158



OpenMP's static scheduling evenly distributes iterations among threads.
Static scheduling has less runtime overhead compared to dynamic scheduling. It has low overhead because it does not require much synchronization during execution. Also, if the work done in each iteration of the loop is roughly the same, static scheduling distributes the work among all threads. Therefore, it provides better performance.

On the other hand, dynamic scheduling does not show better performance than static scheduling. Dynamic scheduling has higher runtime overhead than static scheduling because it requires frequent synchronization during loop execution. Each time a thread finishes processing an allocated iteration chunk, it must get a new chunk from the remaining iterations. Therefore, it requires additional overhead. Also, if the chunk size isn't chosen well, or if the last few chunks are smaller and are allocated to a few threads while other threads are idle, this can lead to poor resource utilization and poor performance. Dynamic scheduling can be better than static scheduling when the workload is not uniform and there are significant variations in the amount of work done in different iterations. The code executed in the task has a uniform workload, so static scheduling shows better performance than dynamic scheduling.

If you specify a chunk size of 10 in the static scheduling scheme, OpenMP distributes 10 repeating chunks to each thread at the beginning of the parallel region. Smaller chunk sizes can result in higher overhead because the number of chunks that need to be distributed across threads increases. However, in the case of static scheduling, overhead occurs at the beginning of the parallel section, so it does not affect loop execution. However, performance suffers due to the increased overhead.

Unlike static scheduling, in dynamic scheduling, a new chunk is allocated to a thread whenever processing of the current chunk is complete. Smaller chunk sizes, like 10, require new chunks to be fetched. Smaller chunk sizes, such as 10, result in higher overhead because the number of times a new chunk must be fetched increases. may lead to performance degradation. Also, each thread always fetches a new chunk when it's done working on the current chunk. However, the code I wrote doesn't get the big benefits of load balancing.