2022.1 Multicore Computing, Project #3

Problem 2

Document

소프트웨어학부

20176342 송민준

(a) in what environment (e.g. CPU type, memory size, OS type ...) the experimentation was performed

CPU : AMD Ryzen 5 2600X Six-Core Processor (12 CPUs), ~3.6GHz

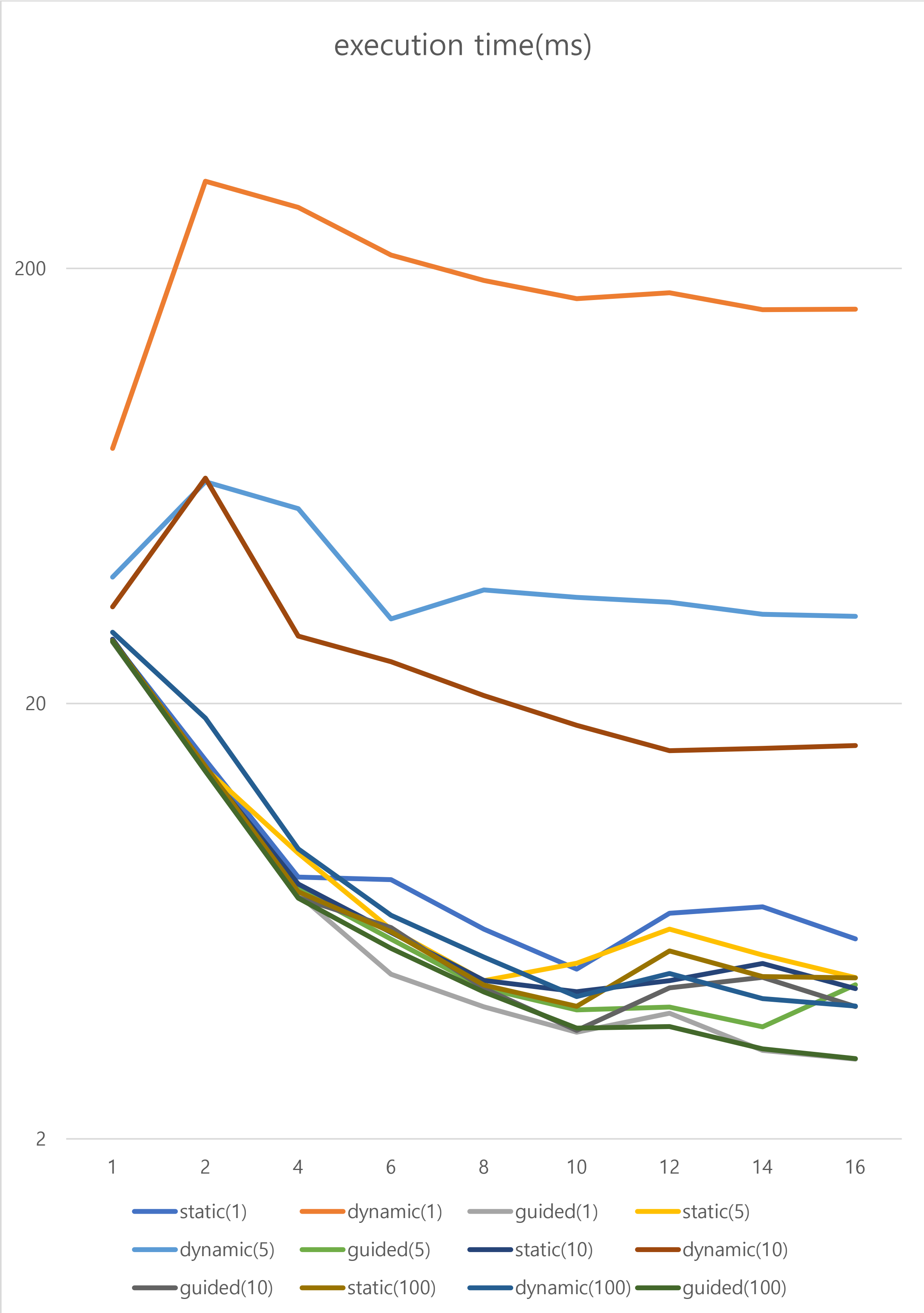
Memory : DDR4 16384MB RAM

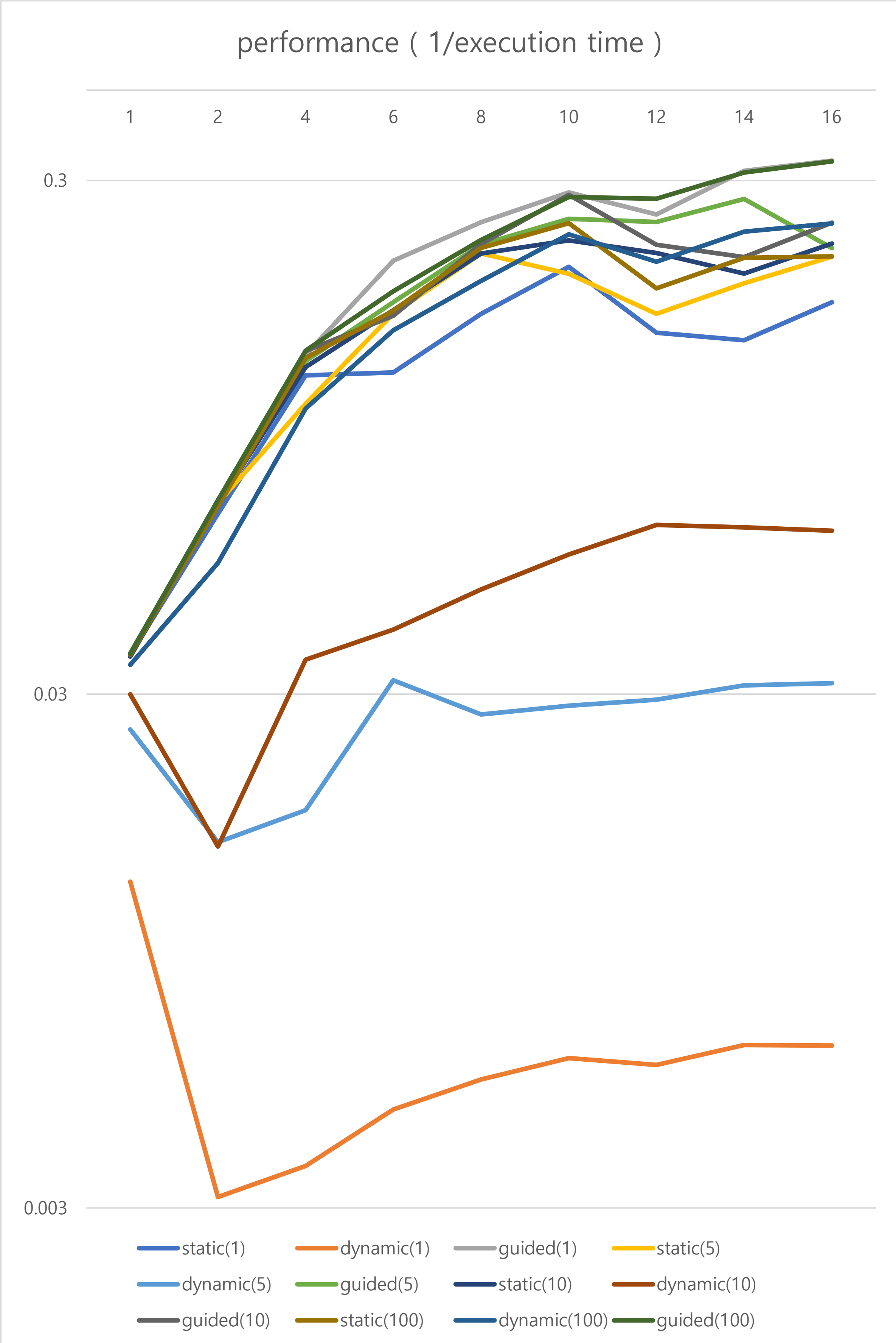
OS : Windows 10

(b) tables and graphs that show the execution time (unit:milisecond) for thread number = {1,2,4,6,8,10,12,14,16}.

Exec time(unit: ms)	Chunk Size	1	2	4	6	8	10	12	14	16
static	1	27.911	14.853	7.985	7.881	6.062	4.909	6.597	6.828	5.757
Dynamic		77.214	317.279	276.202	214.503	187.557	170.392	175.765	160.635	161.095
guided		28.108	13.994	7.282	4.779	4.019	3.516	3.885	3.195	3.051
static	5	27.947	14.169	9.067	6.058	4.623	5.066	6.062	5.289	4.693
Dynamic		39.048	64.676	56.081	31.32	36.51	35.1	34.193	32.061	31.739
guided		27.729	14.081	7.496	5.755	4.441	3.958	4.018	3.62	4.518
static	10	28.14	14.225	7.702	5.962	4.626	4.359	4.617	5.06	4.425
Dynamic		33.361	65.971	28.573	24.976	20.859	17.824	15.604	15.786	16.022
guided		27.744	13.998	7.163	6.117	4.432	3.553	4.446	4.7	4.029
static	100	27.926	14.358	7.391	5.98	4.51	4.037	5.406	4.719	4.685
Dynamic		29.202	18.511	9.271	6.53	5.228	4.246	4.798	4.197	4.039
guided		27.83	13.981	7.151	5.477	4.351	3.593	3.62	3.217	3.059

Exec time(unit: ms)	Chunk Size	1	2	4	6	8	10	12	14	16
static	1	0.035828168	0.0673265	0.1252348	0.1268875	0.1649621	0.2037075	0.1515841	0.1464558	0.1737016
Dynamic		0.012951019	0.0031518	0.0036205	0.0046619	0.0053317	0.0058688	0.0056894	0.0062253	0.0062075
guided		0.03557706	0.0714592	0.1373249	0.2092488	0.2488181	0.2844141	0.2574003	0.312989	0.3277614
static	5	0.035782016	0.0705766	0.1102901	0.165071	0.2163098	0.1973944	0.1649621	0.1890717	0.2130833
Dynamic		0.025609506	0.0154617	0.0178314	0.0319285	0.0273898	0.02849	0.0292458	0.0311905	0.031507
guided		0.036063327	0.0710177	0.1334045	0.1737619	0.2251745	0.2526529	0.24888	0.2762431	0.2213369
static	10	0.035536603	0.0702988	0.1298364	0.167729	0.2161695	0.2294104	0.2165909	0.1976285	0.2259887
Dynamic		0.029975121	0.0151582	0.0349981	0.0400384	0.0479409	0.0561041	0.0640861	0.0633473	0.0624142
guided		0.036043829	0.0714388	0.1396063	0.1634788	0.2256318	0.2814523	0.2249213	0.212766	0.2482005
static	100	0.035808924	0.0696476	0.1352997	0.1672241	0.2217295	0.2477087	0.1849797	0.2119093	0.2134472
Dynamic		0.03424423	0.0540219	0.1078632	0.1531394	0.1912777	0.2355158	0.2084202	0.2382654	0.247586
guided		0.035932447	0.0715256	0.1398406	0.1825817	0.2298322	0.278319	0.2762431	0.3108486	0.3269042


execution time(ms)
200
20
2
1
2
4
6
8
10
12
14
16
static(1)
dynamic(1)
guided(1)
static(5)
dynamic(5)
guided(5)
static(10)
dynamic(10)
guided(10)
static(100)
dynamic(100)
guided(100)


performance (1/execution time)
1
2
4
6
8
10
12
14
16
0.3
0.03
0.003
static(1)
dynamic(1)
guided(1)
static(5)
dynamic(5)
guided(5)
static(10)
dynamic(10)
guided(10)
static(100)
dynamic(100)
guided(100)

(c) The document should also contain explanation on the results and why such results can be obtained.

First of all, looking at the performance graph first, it can be seen that dynamic load balancing is generally slow. Dynamic load balancing is also slowest when chunk size is 1, and then the speed improves as chunk size gets larger.

Dynamic load balancing is the slowest because the more you split the job, the more overhead you get when you get a pi. And if you grow the chunk size, the overhead of splitting the job is reduced, which makes it faster.

Dynamic load balancing can significantly reduce performance when increasing from one thread to two because of the heavy overhead of dividing the work into chunks, and if you have one thread, you don't have to do it. This results in poor performance when the number of threads increases from one to two.

Static, dynamic, and guided performance increases as the number of threads increases, and performance no longer increases when more than 12 threads are added.

In all cases, there are approximately 12 threads where the context switching overhead of the thread is larger.

And the guided load balancing method was the fastest in almost all cases. This is because guided makes chunk size larger from the beginning, making all threads busy, and then gradually decreasing chunk size.

Static load balancing was slower than guided load balancing because the distribution of work was not fair (some threads took a long time to calculate, and some threads were resting because they were done quickly).

Because static load balancing is not a fair distribution of work in the first place, performance tends to increase as threads increase, regardless of chunk size.