Boyao Zhu & Jiaxin Yang

CMSE 822 Parallel Computing

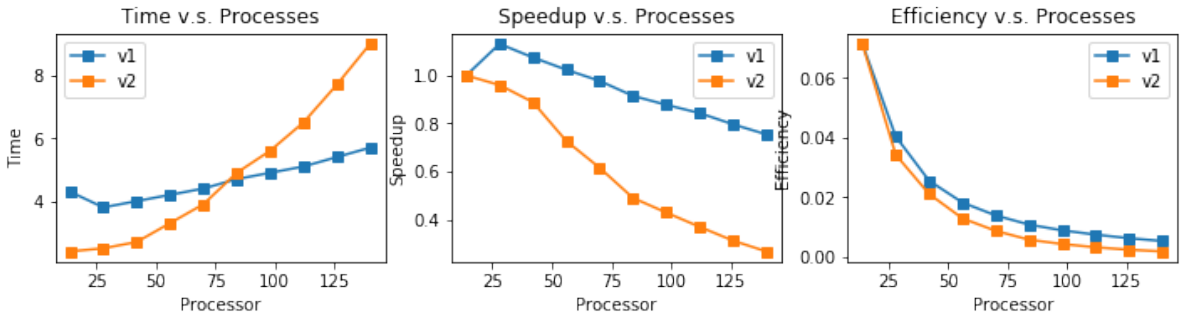
Homework 6

Part 1

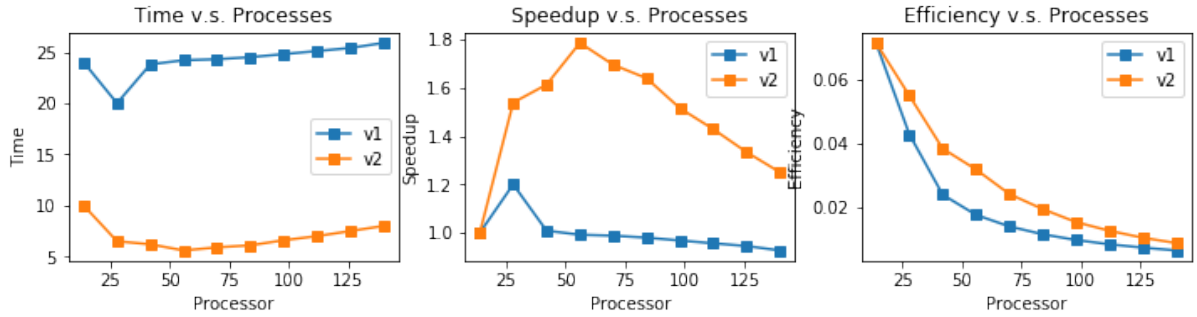
(a)

We test our programs on HPCC, starting from 14 cores (single socket) and using up to 140 cores (5 nodes). The program sizes N are 100 millions, 500 millions and 2 billions ($1 \times 10^8, 2 \times 10^8, 2 \times 10^9$), respectively. The total execution times, efficiencies and speedups are shown below.

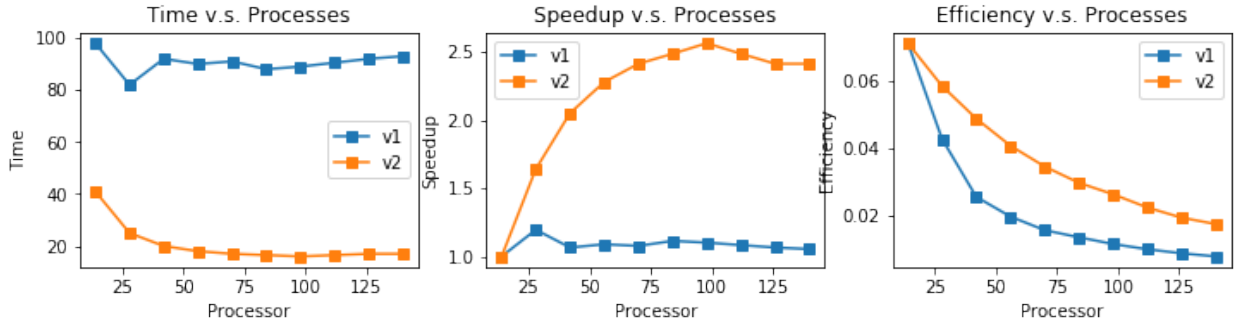
$N = 10^8$



$N = 5 \times 10^8$



$N = 2 \times 10^9$



Both two versions don't scale well and perform worse with increasing process (p). We believe it is due to the competition between increasing communications and decreasing local calculations. Moreover, we observe that v2 outperforms v1 as N, the number of random number, grows. This can be understood that in v1, root process bears all preparation, and this increases as N increases. But, in v2, loads are evenly distributed among all processes.

(b)

$$N = 2 \times 10^9$$

	Generate time		Bin time		Distribute time		Local sort time		Gather time	
#processes	V1	V2	V1	V2	V1	V2	V1	V2	V1	V2
14	42.07	2.95	13.11	0.95	6.53	0.95	30.20	27.68	3.51	7.22
28	42.20	1.50	13.68	0.67	7.05	0.94	14.68	13.54	3.52	7.43
42	41.54	0.98	29.63	0.81	7.44	1.24	9.25	8.88	3.44	6.83
56	42.44	0.79	30.29	0.60	7.55	1.78	6.55	6.62	3.30	6.79
70	42.85	0.60	31.27	0.54	7.59	2.44	5.59	5.33	3.21	6.48
84	41.83	0.47	31.22	0.44	8.21	3.10	4.38	4.21	3.18	6.46
98	41.40	0.44	31.66	0.32	8.63	3.82	3.84	3.22	3.16	6.52
112	42.12	0.39	31.72	0.30	9.11	4.28	3.23	2.99	3.14	6.44
126	41.88	0.33	30.22	0.27	9.15	4.99	2.87	2.48	3.16	6.38
140	42.01	0.29	31.31	0.24	10.01	5.58	2.67	2.16	3.15	6.40

In this part, we measure the execution time of different phases, which consists of array generation, bin determination, array distribution, local sort and result gather. To make results numerically readable, We set $N = 2 \times 10^9$, which is characteristic enough for our analysis. The most significant differences between v1 and v2 locate in the first three phases. Indeed, v1 spends much more time than v2 on these parts. And these phases contribute significantly to total execution time, especially with large process number. In v1 root process bears a heavy load, which seriously hurts the performance. However, v2 processes share almost the same load, which increases its efficiency. In addition, both v1 and v2 show similar performance in the phase of local sort, because each process works independently. V1 outperforms v2 in gathering

because less communication occurs in v1's gathering, and v2 has a better load balance and faster than v1. Other results are attached below.

$N = 1 \times 10^8$

	Generate time		Bin time		Distribute time		Local sort time		Gather time	
#processes	V1	V2	V1	V2	V1	V2	V1	V2	V1	V2
14	2.67	0.16	0.79	0.05	0.58	0.26	1.69	1.39	0.17	0.32
28	2.98	0.08	0.87	0.02	0.80	0.46	0.83	0.67	0.17	0.38
42	2.72	0.06	1.11	0.03	1.00	0.58	0.48	0.48	0.25	0.41
56	3.17	0.05	1.35	0.02	1.24	0.77	0.42	0.37	0.35	0.66
70	3.19	0.05	1.29	0.02	1.51	0.97	0.35	0.26	0.20	0.51
84	3.34	0.04	1.27	0.02	1.67	1.18	0.29	0.26	0.21	0.56
98	3.01	0.03	1.30	0.01	1.93	1.46	0.26	0.23	0.30	0.64
112	3.55	0.03	1.36	0.01	2.16	1.18	0.23	0.20	0.25	0.68
126	3.59	0.02	1.54	0.01	2.54	2.29	0.20	0.17	0.27	0.75
140	3.79	0.02	1.43	0.01	2.63	2.35	0.19	0.17	0.24	0.78

$N = 5 \times 10^8$

	Generate time		Bin time		Distribute time		Local sort time		Gather time	
#processes	V1	V2	V1	V2	V1	V2	V1	V2	V1	V2
14	13.19	0.78	3.85	0.24	1.90	0.40	9.01	7.17	0.75	1.51
28	14.25	0.42	4.10	0.14	2.31	0.51	4.81	4.13	1.12	1.56
42	14.55	0.31	6.37	0.13	2.55	0.66	3.32	3.03	0.93	1.45
56	14.70	0.21	6.16	0.10	2.81	0.82	2.41	1.88	0.95	1.51
70	20.04	0.17	5.85	0.08	2.80	0.99	1.60	1.50	0.84	1.52
84	13.95	0.15	6.18	0.07	3.09	1.25	1.37	1.30	0.91	1.63
98	14.21	0.13	5.84	0.07	3.38	1.45	1.19	1.66	0.87	1.66
112	14.99	0.12	6.34	0.06	3.67	1.84	1.06	1.08	0.89	1.73
126	15.37	0.11	6.47	0.05	4.32	2.38	1.43	1.00	0.88	5.37
140	16.45	0.10	6.39	0.05	4.32	2.42	0.94	0.94	0.77	1.92

P.S.

In this Homework, we had 2 versions of bucket_sort_v2.c, which was coded separately. One compared bucket_sort_v1.c with the first version of bucket_sort_v2.c, which contribute to part I. While the other used second version of bucket_sort_v2.c to compare with bucket_sort_v3, which contribute to part II. So the data of bucket_sort_v2 may looks different in 2 parts.

Question 2

- 1. Take N = 2,000,000,000 as an example

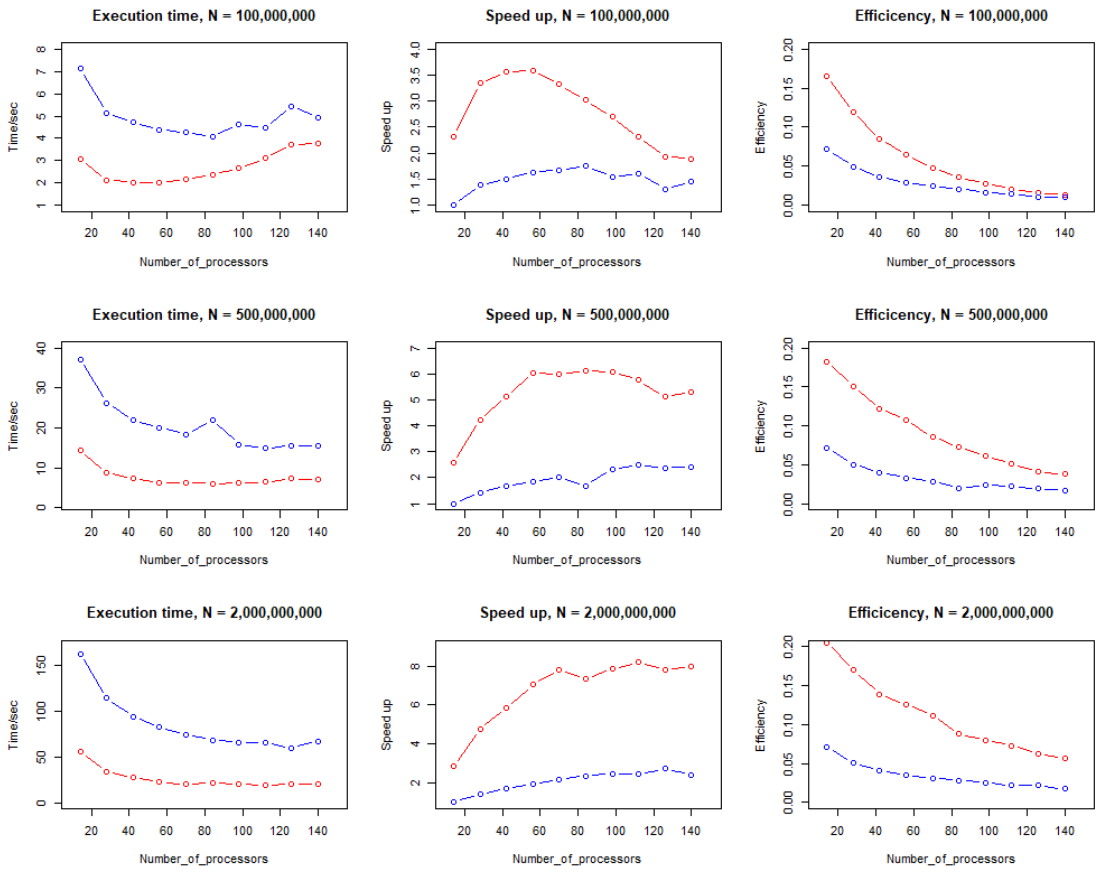
#processors	Uniform Min time	Uniform Max time	Uniform Ave time	Squared Min time	Squared Max time	Squared Ave time
14	50.16549	51.65397	50.81503	158.54366	162.31880	160.36575
28	27.94675	28.79809	28.46914	111.85467	114.24249	113.23436
42	21.06843	22.09723	21.77419	92.45437	93.94243	93.06452
56	20.01567	20.59854	20.20952	81.08576	82.68083	81.76584
70	17.60986	18.70543	18.22532	72.65468	74.44119	73.98913
84	15.28403	15.96547	15.66551	65.76578	68.63640	67.06846
98	15.04654	15.35739	15.24398	65.95401	67.57456	66.75654
112	15.43076	15.83027	15.65156	64.88226	66.19994	65.35743
126	15.22105	15.59836	15.32782	59.64563	61.67731	60.26554
140	14.84854	15.24089	15.04316	61.65474	64.22437	62.64248

We can observe that the squared input costs more time that uniformed one.

- 3. Just use

$$S = 12 \times \#Processors \times log_2(N)$$

The red curve represents version 3 and the blue curve represents version 2.



Based on the plot, by selecting better pivots, the version 3 beats version 2 in any conditions. So the load imbalance problem is remedied.

When N is small, the speed up goes up at the beginning and then goes down as we increase the number of processors, which may be due to the time spent in selecting pivots.

- 4. N = 100,000,000

#processors	Execution time V2	Execution time V3	Local sort time V2	Local sort time V3	Select pivots time V3
14	7.141098	3.085823	6.335912	2.00092	0.035096
28	5.140282	2.128991	4.271277	1.005166	0.070987
42	4.74025	2.006507	3.481234	0.717346	0.107974
56	4.387656	1.989261	2.929638	0.528984	0.140733
70	4.259611	2.149182	2.64103	0.584744	0.065518
84	4.084136	2.364185	2.335471	0.407525	0.101423
98	4.623136	2.636741	2.129685	0.602431	0.084465
112	4.464304	3.095392	2.080492	0.368805	0.09612
126	5.443174	3.703375	1.96101	0.382304	0.109076
140	4.937406	3.776036	1.987116	0.358639	0.132404

N = 500,000,000

#processors	Execution time V2	Execution time V3	Local sort time V2	Local sort time V3	Select pivots time V3
14	37.145258	14.503446	34.179177	10.381325	0.041856
28	26.267568	8.788078	23.775689	5.347536	0.068065
42	21.917821	7.237915	19.360209	3.830571	0.103321
56	20.120534	6.151884	16.119834	2.718232	0.139668
70	18.309004	6.193875	14.493686	2.453819	0.092163
84	21.948996	6.067357	13.389144	2.233362	0.156086
98	15.85829	6.129294	12.17381	2.037888	0.064928
112	14.823311	6.440016	11.002445	1.977828	0.152674
126	15.579125	7.238188	11.233935	1.884998	0.195586
140	15.465686	6.984677	11.256443	1.851637	0.122507

N = 2,000,000,000

#processors	Execution time V2	Execution time V3	Local sort time V2	Local sort time V3	Select pivots time V3
14	162.318804	56.528666	149.430138	41.348024	0.071783
28	114.242489	34.158568	103.505128	21.558882	0.097482
42	93.942427	27.755692	84.396301	14.689283	0.107031
56	82.680829	23.076407	72.2637	11.184059	0.143749
70	74.441186	20.716727	64.627486	9.710804	0.098239
84	68.636397	22.07807	59.13009	8.890311	0.146015
98	65.954008	20.625565	54.759571	8.033759	0.156309
112	66.199937	19.82552	51.926315	7.665162	0.123765
126	59.645625	20.761564	48.400635	7.184525	0.249126
140	67.224372	20.323457	49.469146	7.094715	0.233717

From the table, version 3 just costs less than half of second while reduces the time in local sort greatly. When the number of numbers goes up to 2 billion, the time spent in local sort is reduced by about 7 times. So the time spent in selecting pivots is worthy.