Performance Evaluation of Parallel Sorting in Shared and Distributed Memory Systems

Shaiful Alam Chowdhury
shaiful@ualberta.ca
Soham Sinha
soham@ualberta.ca

December 12, 2014

Abstract

In this paper, we investigate the performance of two different parallel sorting algorithms (PSRS and BRADIX) in both distributed and shared memory systems. These two algorithms are technically very different than each other. PSRS is a computation intensive algorithm whereas BRADIX relies mainly on communication among processors to sort data items. We observe that a communication intensive algorithm exhibits worse performance than a computation intensive algorithm, in terms of speedup, both in shared and distributed systems. This suggest that this type of algorithms, in general, can not take advantage of an increased number of processors.

1 Introduction

Different kinds of parallel sorting algorithms have been developed to address different issues related to parallel programming paradigm. In this paper, we present a comparative study between two important but strategically different parallel sorting algorithms: Load Balanced Parallel Radix Sort (BRADIX) [4] and Parallel Sorting by Regular Sampling (PSRS) [2]. We compare the performance of these two algorithms both in shared memory system and distributed memory system with different natures of input keys. We found that the area of employing these algorithms are different and distinct. Either of them perform better than other one depending on the computer system and nature of the input keys. We hope that our report can help scientific researchers to determine the proper algorithm they need for their specific problem with their available computer system.

We tested the Message Passing Interface (MPI) specific implementation of these two algorithms. In case of BRADIX, we have taken the code from GitHub [3] and developed the PSRS algorithm from scratch. For the performance evaluation in distributed memory system, we used the *coldlake* cluster at University of Alberta. For shared memory system, we used *st-francis*. Section 4 describes these issues in more detail.

As expected, we observe that BRADIX performs tremendously well in shared-memory system in terms of total execution time. However, in distributed memory system, BRADIX

cannot perform well enough because of lower granularity. PSRS performs way better in the latter case. In terms of speedup, PSRS always outperforms BRADIX. We also observe that duplicate key values have significant effect on PSRS runtime although BRADIX performs similarly with the presence or absence of duplicate keys.

2 Algorithms

In this section, we describe a brief overview of the sorting algorithms we chose to study.

2.1 Load Balanced Parallel Radix Sort (BRADIX)

A simple Parallel Radix Sort (PRS) is based on the sequential Radix sort technique, and it's not a comparative sorting methodology. Sequential Radix sort itself has an inherent quality of parallelizing the computation. PRS takes advantage of that inherent characteristic. Sequential Radix Sort groups the keys based on their least significant digit and continue grouping them with more significant digits. PRS initially follows the same technique as Sequential Radix Sort, i.e., grouping the keys based on the least significant digit(s). After that, it gets the keys as groups into certain number of buckets based on the digit in which the grouping has been performed. Now each processor is asked to take responsibility of a fixed number of buckets and perform sequential Radix Sort locally. This process continues with more significant digit(s). In PRS, instead of taking single digit to group the keys, a number of digits (g) is usually considered to make the process more efficient. In general, PRS follows the following four phases.

1. By scanning the keys using g bits, each processor calculate the number of keys in each bucket.

2. Each processor then moves the keys to appropriate buckets by doing the scanning again.

3. Now each processor sends the number of keys from each of its buckets to all other processors so that each processor has the actual numbers that it must receive from others.

4. At this phase, each processor collects the keys from other processors based on the processor and bucket ranks. For example, processor P_0 collects all the keys from bucket 0 of other processors.

PRS, however, might have serious load balancing problem when a particular processor is overloaded with keys while some other processors are experiencing light load of keys. To avoid this scenario, Load Balanced Parallel Radix Sort (BRADIX) is proposed by Andrew Sohn et al. [4]. In BRADIX, if some processors have lighter load than others, then they coordinate between each other to equalize their loads. The first two phases of BRADIX are the same as PRS. The main difference between PRS and BRADIX is in the third phase. Instead of having only the number of keys information for buckets that a processor is interested in, each processor now have the complete knowledge of every buckets in every processors, by

applying the *all-to-all* operations. This allows each processor to know where to collect the extra keys from, if it receives less number of keys than the number of keys to have perfect load balancing. Phase 4 is also modified a little to adjust the changes made in phase 3. An example of the algorithm has been demonstrated in Figure 1 taking the idea from the original paper [4]. In BRADIX, phase 1 and phase 2 are for computation whereas phase 3 and phase 4 are for communication. Each phase of this algorithm is repeated r times such that $r = \frac{n}{g}$, where n is the total number of bits of each key, and g is the group size for scanning. For a group size of 4 and for 32 bit keys, each phase in BRADIX is repeated 8 times.

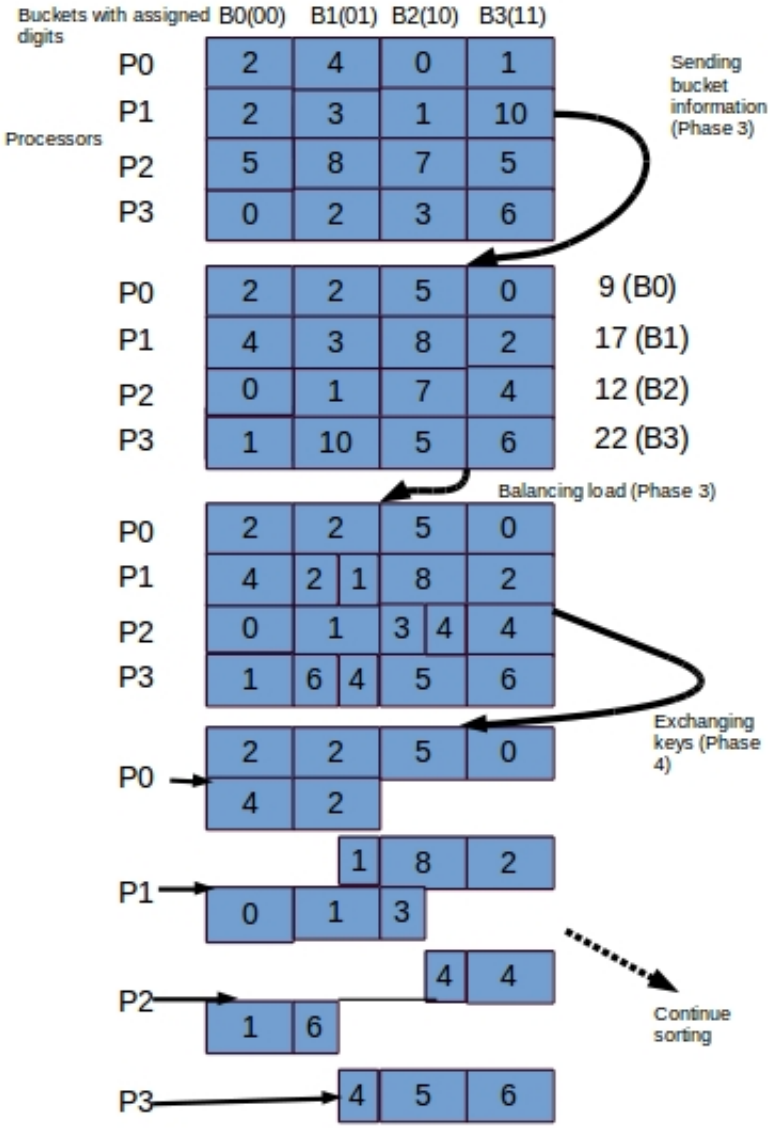


Figure 1: BRADIX demonstration for g=2 digits group

2.2 Parallel Sorting Using Regular Sampling (PSRS)

PSRS [2] involves sampling of data to divide the work between processors in an efficient manner. Without being elaborative, we present the four phases of the algorithm briefly:

1. Each of the processors have their own equally divided number of keys. They sort their keys locally following a sequential sorting algorithm (e.g., QuickSort). Each processor then select a fixed number of regular samples as described in [2].

2. One designated processor collects all the samples from all the other processors and sort these samples. If p is the number of total processors, then the processor choose $p-1$ pivots. These pivotes are broadcasted then to all other processors. After receiving the pivots, each processor partitions its locally sorted array using the pivots.

3. Each processor sends the partitions to other processors matching the partition number and the processor number. Each processor also keeps the partition which matches its own rank.

4. After receiving partitions from different processors, each processor merges all those sorted partitions.

It can be observed that in PSRS there is only one phase which is communication intensive, i.e., Phase 3. The communication involved in phase 2 is very insignificant. In contrast to BRADIX, all the phases in PSRS are executed only once.

3 Implementation and Validation

The actual BRADIX algorithm was proposed by Mr. Andrew Sohn and Mr. Yuetsu Kodama [4]. We tried to find the code written by the authors. However, because of the unavailability of the authors' own implementation, we used codes from [3]. We modified this code to include some parameters by which we can evaluate the algorithm in different scenarios according to our requirement. The PSRS algorithm was previously developed for our CMPUT 681 course assignment. We used that code with some modifications to compare it with BRADIX. Our implementation of PSRS uses QuickSort as in the original paper. For BRADIX algorithm, we have considered 8 bits as the group size. Both the codes are written in C language with MPICH library [1].

In order to validate the correctness of both the implementations, all the sorted keys were collected by the master processor in an array ***gather*** after phase 4. Two different approaches were applied to verify the correctness of the implementations of the PSRS and BRADIX.

First approach:

All the collected keys at the main processor will be in sorted order given that the implementations were correct. This means that the keys at the main processor has to be in non-decreasing order. As a result ***gather[i]*** has to be greater than or equals to ***gather[i-1]*** for all i.

Second approach:

Before phase 1, while generating the random keys, an integer array ***count1*** is used that counts the number of keys with the same value. For example, ***count1[5]*** indicates the number of keys with value 5. After phase 4, when all the items are collected by the master processor, another integer array ***count2*** is used for the same purpose. If no key was lost because of the send and receive operations by the different processors, ***count1[i]*** has to be equals to ***count2[i]*** for all i.

4 Experiment and Result Analysis

Our experiments, for distributed system, were conducted in the *coldlake* cluster generously provided by Prof. Paul Lu for this project. Coldlake has 12 nodes. The nodes have 2 or more cores with 1.7 GHz CPU clock-speed. For our experiments in shared memory system, we used *st-francis* graciously permitted to be used by Prof. Abram Hindle. This machine has 8 cores with 70GB of memory, 32K L1d and L1i cache, 256K L2 cache, and 12288K L3 cache.

In this section, we present our experimental results for BRADIX and PSRS in distributed and shared memory system. We conducted these experiments with randomly distributed fixed-length keys (5-digit integer numbers, i.e., 10,000 to 99,999) and uniformly distributed positive-valued keys where maximum value is the maximum allowed number of a 32-bit integer. Fixed-length keys are taken in order to observe how BRADIX and PSRS deal with duplicate keys. For all the results, the experiments were repeated 7 times, and the average of the last 5 runs has been taken.

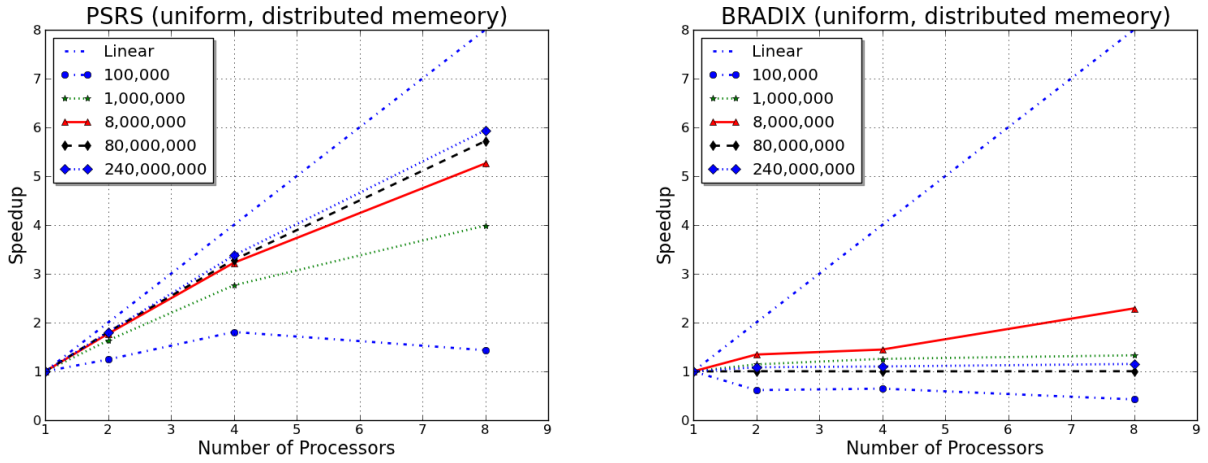
4.1 Distributed Memory System

4.1.1 Uniform key-distribution

The speed-up graphs for PSRS and BRADIX are given respectively in Figure 2 (a) and Figure 2 (b) for distributed memory with uniform key-distribution. It can be seen that while speedup for PSRS increases with more number of processors, BRADIX has a decreasing or same speedup (with very few exceptions) with the increment in number of processors. The reason behind the bad performance of BRADIX in distributed memory is rather simple; the granularity of the algorithm decreases with increasing number of processors. In the Phase 4 of BRADIX algorithm, large number of keys are being exchanged among all the processors. This communication-intensive phase is being run multiple times starting from least significant digits (8 bits at a time, as mentioned earlier) to most significant ones. As the number of processor increases, the need for communication also gets increased. The cost of communication over the network is significant, and that affects the granularity of the problem.

One of key points is to be noted from the Table 1 and Table 2 is that the actual time to complete sorting for a single processor in BRADIX is always less than PSRS with any number of keys. The reason is that Radix sort, in general, is a faster sorting method than other comparative sorting algorithms (QuickSort in case of our PSRS implementation). Another observation is that BRADIX sometime performs worse in multiple processors than in the single processor as observed from Table 2 for 100,000 keys. This scenario happens whenever

the overhead of communication becomes greater than the computation time. Although PSRS also has overhead of communication, but that does not exceed the computation cost in most of the times. This communication overheads decreases as the number of keys to sort increases. For example, for 8 number of processors, PSRS gets a maximum speedup of 5.94 with 240,000,000 number of keys.



(a) PSRS (b) BRADIX

Figure 2: Speedup with uniformly distributed keys in distributed memory system

Table 1: PSRS Execution time (in seconds) and speedup (uniform distribution and distributed memory)

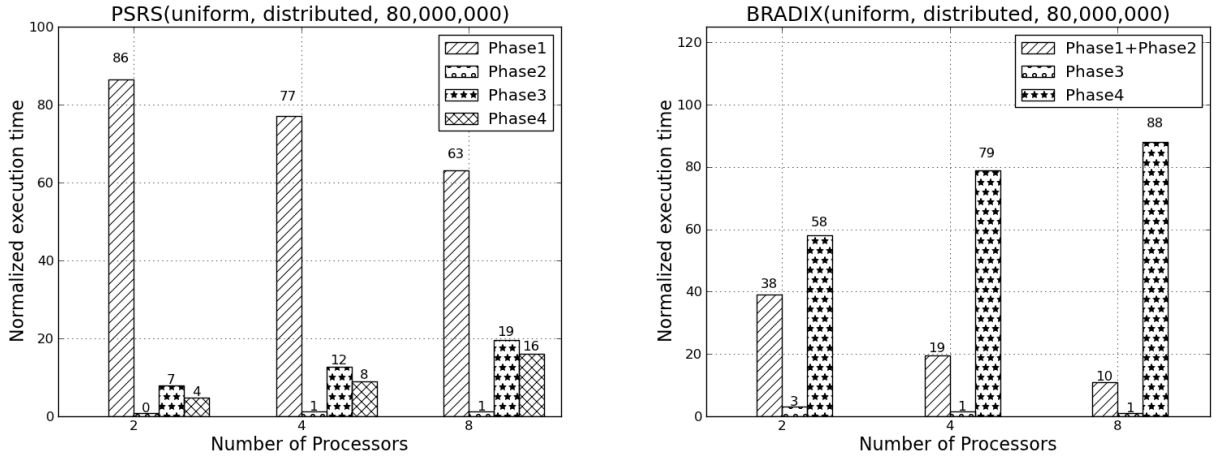
Sizes	**1 PE**		**2 PEs**		**4 PEs**		**8 PEs**	
	time (s)	speedup	time (s)	speedup	time (s)	speedup	time (s)	speedup
100,000	0.0184	–	0.0148	1.2437	0.0102	1.8082	0.0128	1.4341
1,000,000	0.2141	–	0.1311	1.6334	0.0775	2.7638	0.0537	3.9864
8,000,000	2.0093	–	1.135	1.7703	0.6231	3.2246	0.382	5.2605
80,000,000	22.5959	–	12.5151	1.8055	6.8788	3.2849	3.951	5.719
240,000,000	71.28	–	39.4852	1.8052	21.1148	3.3758	12.0001	5.94

The phase-by-phase analysis of PSRS and BRADIX, depicted respectively in Figure 3 (a) and 3 (b), shows the bottleneck of both algorithms. In case of PSRS, phase 1 is the time where large part of the sorting happens; so, phase 1 takes most of the time. However, as the number of processors increase, phase 3 increasingly takes more time to complete. As phase 3 is communication intensive in PSRS, increment in the number of processors result into more communications. So, phase 3 is the bottleneck of this algorithm. However, for BRADIX, phase 4 takes highest time, and the percentage of the running time of phase 4 also increases with the number of processors. Phase 4 is the particular phase where large number of keys are exchanged multiple times. The time taken in phase 4 is even more than the computation phases. So, with increasing number of processors, phase 4 slows down the performance of

Table 2: Balanced Parallel Radix Sort Execution time (in seconds) and Speed-up (uniform distribution and distributed memory)

Sizes	**1 PE**		**2 PEs**		**4 PEs**		**8 PEs**	
	time (s)	speedup	time (s)	speedup	time (s)	speedup	time (s)	speedup
100,000	0.0162	–	0.0263	0.6166	0.025	0.6471	0.038	0.4261
1,000,000	0.1585	–	0.1386	1.1443	0.1261	1.2572	0.1192	1.33
8,000,000	1.596	–	1.1859	1.3459	1.1025	1.4476	0.6965	2.2914
80,000,000	12.5655	–	12.5262	1.0031	12.5534	1.001	12.5244	1.0033
240,000,000	39.5041	–	36.4087	1.085	35.8664	1.1014	34.2745	1.1526

BRADIX significantly in distributed memory system.



(a) PSRS (b) BRADIX

Figure 3: Phase-by-phase time(%) for 80,000,000 keys (uniform-distribution, distributed memory)

4.1.2 Fixed-length keys

Fixed-length keys have numbers ranging from 10,000 to 99,999 in our experimentation. This allows us to generate duplicate input-keys and test the algorithms' performance in that scenario. BRADIX performs similarly with fixed-length keys in distributed-memory as with uniformly distributed keys. There's some improvement in the actual running time but the speed-up is not improved, which can be seen in Table 4. The improvement in running-time is mainly because of the radix sort's inherent strategy of sorting rather than leveraging the benefits of parallelism. Interestingly, we observe a super linear speedup for PSRS with fixed-length keys for certain number of inputs. PSRS gets a highest speed up of 25.978 with 80,000,000 keys for 8 processors as in Table 3. From Table 3, we can also observe that PSRS

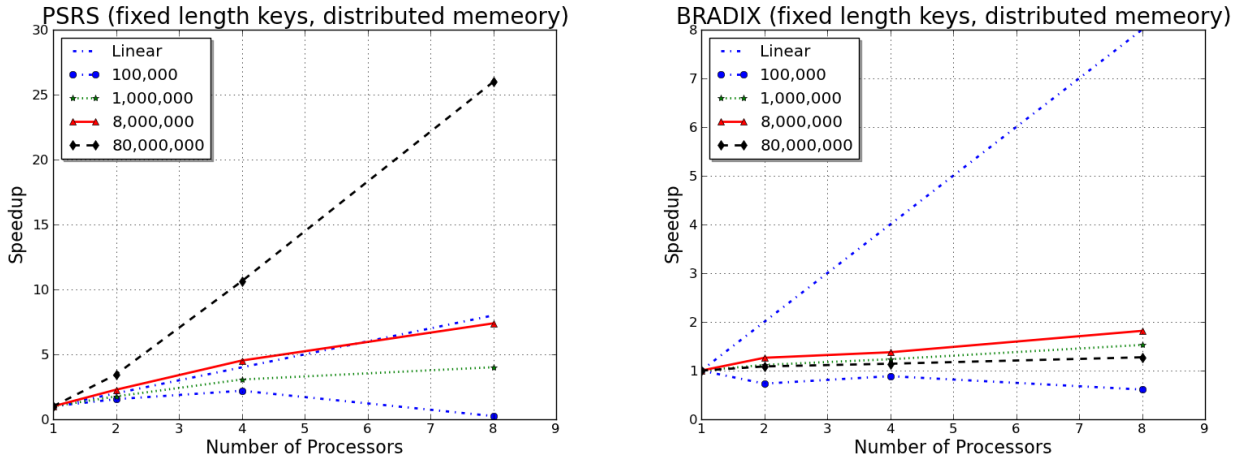
takes way more time to complete even with single processor for fixed-length keys than for uniformly distributed keys.

Table 3: PSRS Execution time (in seconds) and Speed-up (fixed length keys and distributed memory)

Sizes	**1 PE**		**2 PEs**		**4 PEs**		**8 PEs**	
	time (s)	speedup	time (s)	speedup	time (s)	speedup	time (s)	speedup
100,000	0.0186	–	0.012	1.5509	0.0085	2.1986	0.0748	0.2493
1,000,000	0.223	–	0.1264	1.764	0.0727	3.0655	0.0556	4.0094
8,000,000	2.907	–	1.2764	2.2774	0.6426	4.5239	0.3935	7.3874
80,000,000	142.01	–	41.1611	3.4501	13.3597	10.6297	5.4665	25.9781

Table 4: Balanced Parallel Radix Sort Execution time (in seconds) and Speed-up (fixed length keys and distributed memory)

Sizes	**1 PE**		**2 PEs**		**4 PEs**		**8 PEs**	
	time (s)	speedup	time (s)	speedup	time (s)	speedup	time (s)	speedup
100,000	0.0162	–	0.0221	0.7326	0.0183	0.8856	0.0265	0.6109
1,000,000	0.144	–	0.1293	1.1170	0.1171	1.2335	0.0946	1.5270
8,000,000	1.3003	–	1.0296	1.2628	0.9446	1.3764	0.7160	1.8160
80,000,000	11.7803	–	10.8509	1.0856	10.3213	1.1414	9.2508	1.2734
240,000,000	35.9576	–	31.6832	1.1349	29.9141	1.202	26.9889	1.3323



(a) PSRS (b) BRADIX

Figure 4: Speedup with fixed-length keys in distributed-memory system

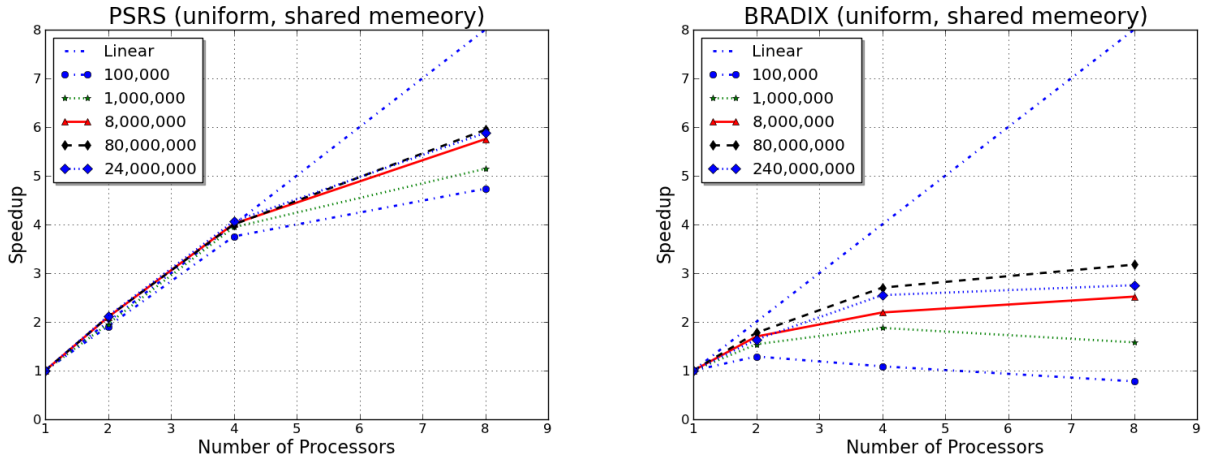
QuickSort in general performs worse if there are duplicate values in the input keys. As the number of processors increases, the load of duplicate values reduces. This improves the

performance of QuickSort very significantly. Figure 4 (a) for PSRS with fixed-length keys, shows that the speedup for 80,000,000 keys with 8 processor is surprisingly higher than linear speedup. This high speedup is also contingent upon the higher number of keys because we can only see this speedup with high number of keys.

4.2 Shared Memory System

4.2.1 Uniform key-distribution

For shared-memory machine with uniformly distributed keys, BRADIX performs better compared to itself in distributed memory machine. The speed-up graph in Figure 5 (b) is for BRADIX with uniform keys in shared-memory architecture. If we compare it to Figure 2 (b) which is for distributed memory, we can see an increment in the speed-up for same number of keys and same number of processors. In shared-memory machine, interprocess communication does not happen over the network. So, network-communication does not become the constraint of the algorithms. That is why the speed-up is improved in shared-memory architecture. Still, the speed-up is no where near the speed-up achieved by PSRS in shared memory as can be seen in Figure 5 (a). So, this tells us that the speed-up not only depends on the running time of the sequential algorithms and system architectures but also on the strategies of parallelism taken in the algorithms. Additionally, Table 5 tells us that BRADIX improves the running time significantly in shared-memory as well. It only takes 5.8846 seconds to sort 240,000,000 keys with 8 processors compared to nearly 34 seconds in distributed memory. This also attributes to the reduced cost of communication in shared-memory machines.



(a) PSRS (b) BRADIX

Figure 5: Speedup with uniformly distributed keys in shared memory system

Being a general purpose algorithm, PSRS in shared-memory performs quite similarly as in distributed memory. In Table 6, we can see almost similar speed-up as in Table 1 for

Table 5: Balanced Parallel Radix Sort Execution time (in seconds) and speedup (uniform distribution and shared-memory)

Sizes	**1 PE**		**2 PEs**		**4 PEs**		**8 PEs**	
	time (s)	speedup	time (s)	speedup	time (s)	speedup	time (s)	speedup
100,000	0.0062	–	0.0048	1.2902	0.0057	1.088	0.008	0.7798
1,000,000	0.054	–	0.0351	1.5406	0.0288	1.8765	0.0342	1.58
8,000,000	0.5667	–	0.3335	1.6995	0.2584	2.1932	0.225	2.5191
80,000,000	5.7266	–	3.2258	1.7753	2.1186	2.703	1.8032	3.1758
240,000,000	16.206	–	9.9053	1.6361	6.3563	2.5496	5.8846	2.754

Table 6: PSRS Execution time (in seconds) and speedup (uniform distribution and shared-memory)

Sizes	**1 PE**		**2 PEs**		**4 PEs**		**8 PEs**	
	time (s)	speedup	time (s)	speedup	time (s)	speedup	time (s)	speedup
100,000	0.0133	–	0.007	1.9079	0.0035	3.7549	0.0028	4.7348
1,000,000	0.1299	–	0.0663	1.9574	0.0329	3.944	0.0252	5.1438
8,000,000	1.199	–	0.5715	2.0978	0.2987	4.0145	0.2084	5.753
80,000,000	13.4723	–	6.4027	2.1042	3.3689	3.9991	2.2666	5.9437
240,000,000	42.96	–	20.2786	2.1185	10.5695	4.0645	7.3054	5.8806

uniformly distributed keys. The speed-up graph for PSRS in shared-memory is given in Figure 5 (b). However, if we look at the actual running time, we can see a significant decrease for PSRS in shared-memory. As interprocess communication is faster in shared memory than in the distributed memory, the running-time improves just like BRADIX. This can be confirmed by looking at Figure 6 and comparing to 3 (a) which are phase-by-phase analysis in both types of machines. We can see that Phase 3 which is the communication-intensive phase, takes significantly lower percentage of total time of the algorithm in shared-memory machine. The percentage almost remains same in spite of the increment in the number of processors. So, Phase 3 is not a bottleneck for PSRS with shared-memory architecture. Nevertheless, the actual running time taken by PSRS in shared-memory is not better than the BRADIX because of radix sort's inherent superiority in faster sorting strategy.

4.2.2 Fixed-length keys

With fixed-length key-values, PSRS performs well in shared-memory but with some constraints. We can see from Table 8 that the speedup is super linear here as well for 80,000,000 keys with 8 processors. However, the running time in shared memory is better than in distributed memory because of the less constraints in interprocess communications. BRADIX performs similarly with fixed-length keys as with uniformly distributed keys. Table 7 depicts the speedup and running time of BRADIX with fixed-length keys in shared-memory machine.

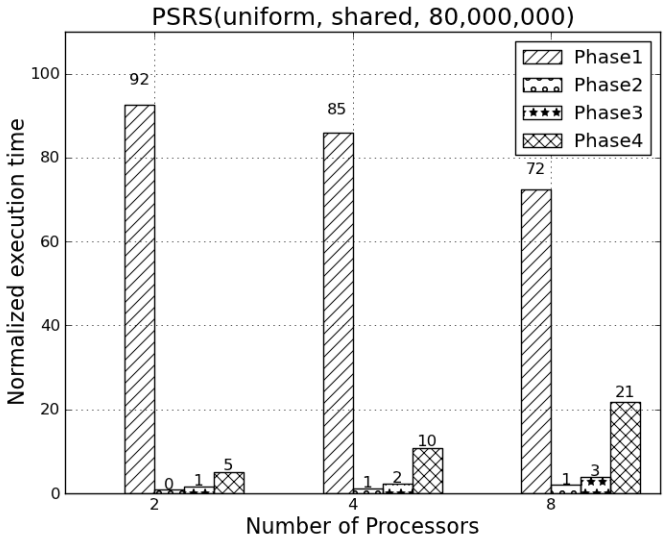


Figure 6: Phase-by-phase time(%) for PSRS for 80,000,000 keys (uniform-distribution, shared-memory)

Table 7: Balanced Parallel Sort Execution time (in seconds) and Speed-up (fixed length keys and shared-memory)

Sizes	**1 PE**		**2 PEs**		**4 PEs**		**8 PEs**	
	time (s)	speedup	time (s)	speedup	time (s)	speedup	time (s)	speedup
100,000	0.0065	–	0.0049	1.3256	0.0057	1.1402	0.0079	0.8168
1,000,000	0.0544	–	0.0358	1.5207	0.0276	1.9728	0.0341	1.5983
8,000,000	0.5339	–	0.3354	1.5921	0.2333	2.2887	0.2271	2.3508
80,000,000	5.6257	–	3.2244	1.7447	2.1458	2.6217	1.9022	2.9575
240,000,000	16.2991	–	9.8982	1.6467	6.6024	2.4688	5.5126	2.9568

Table 8: PSRS Execution time (in seconds) and Speed-up (fixed length keys and shared-memory)

Sizes	**1 PE**		**2 PEs**		**4 PEs**		**8 PEs**	
	time (s)	speedup	time (s)	speedup	time (s)	speedup	time (s)	speedup
100,000	0.0156	–	0.0082	1.8971	0.0045	3.4429	0.0032	4.9461
1,000,000	0.13	–	0.061	2.131	0.0325	4.0024	0.0242	5.3761
8,000,000	1.4685	–	0.5905	2.487	0.2858	5.1381	0.2057	7.1382
80,000,000	54.57	–	12.5736	4.34	4.6697	11.6859	2.293	23.799

5 Conclusion

In the analysis, we have made several important observations. The observations can be summarized as following:

I. PSRS is a generalized algorithm which can be used in any of the architectures with a guaranteed speedup. The performance of PSRS, however, depends mostly on phase 3 (communication). The cost for phase 3 increases if the number of processors is also increased. This effect, however, is not significant when the number of keys is significantly large.

II. BRADIX is not a feasible sorting algorithm in distributed memory. It does not improve the execution time significantly with the increment in the number of processors.

III. BRADIX is a good choice for shared memory machine in terms of execution time. The speedup, however, is not significant for this algorithm both in shared and distributed memory system.

IV. The number of duplicate keys has significant impact on the performance of PSRS. BRADIX, on the other hand, does not exhibit significant changes when sorting data with lots of duplicate keys.

The choice of the two algorithms should be depended on the characteristics of the input keys and the available system. The scalability of the algorithm should also be considered carefully. BRADIX may perform well for shared memory with lots of duplicate keys, but because of a moderate speedup, it may not give better performance after a certain number of processors. PSRS, however, can be categorized as an algorithm that shows significantly good performance in most of the cases.

References

[1] Mpich message passing interface. http://www.mpich.org/.

[2] Xiaobo Li, Paul Lu, Jonathan Schaeffer, John Shillington, Pok Sze Wong, and Hanmao Shi. On the versatility of parallel sorting by regular sampling. *Parallel Computing*, 19(10):1079–1103, 1993.

[3] Miguel Palhas. Parallel radix sort implementation in mpi. https://github.com/naps62/parallel-sort.

[4] Andrew Sohn and Yuetsu Kodama. Load balanced parallel radix sort. In *Proceedings of the 12th international conference on Supercomputing*, pages 305–312. ACM, 1998.