Review on Parallel Sorting Algorithms

Abstract:

Sorting algorithms are most studied and used algorithms both in academia and indusry. To improve the running time of sorting procedures there is always needs for parallel algorithms and architectures. In this paper we have summarized literature reviews of parallel sorting algorithms. We have specifically covered hybid sorting algorithms in two tracks. First track is hybrid sorts that exploit the goodness of both shared memory and message passing architecture. Second track is where hybridization happens on algorithmic level and two or more algorithms are joined to get good performance. Moreover, we have also provided a literature review of some scheme that supports GPU based architecture for sorting.

1. Introduction:

Sorting is the class of algorithm that is clearly defined and heavily used. Notation of sorting starts with evolution of humankind and its numerical ability. With the evolution of computers and algorithms sorting became a basic subprocedure in countless number of algorithmic and implementation procedures. When the field of parallel processing emerges after that parallel sorting algorithms are adapted very quickly.

Parallelization is today's time is defined in two manners: Coarse and Fine-grain parallelism. Coarse grain refers to parallel processing using multiple processes and later one is used to refer parallel processing achieved by threads and hyperthreads. Also there are attempt to combine both that are summarized in separate section.

This paper is organized as follows: section one provides general introduction of sorting techniques and architectures. Section two presents summary of papers on parallel sorting that are having good impact on current research area. Secton three describes mixed types of algorithms both in dimension of architectre and algorithms of parallel sorting, followed by sorting algorithms that are specifically designed for fastest sorting by GPU in section four. At last, we provided conclusion and references in next sections.

2. Background and literature review:

This section presents literature survey of state of art parallel algorithm aiming for the task of sorting. Many scholars have done comparative analysis of open-source sorting procedues on various architecture, Also alot of researchers dig deep into algorithmic dimension and based on the architecture adjusted some sub-task of sorting to get higher throughput and speedup.

Difference between theoretical and practical performance is always there and to use any algorithm in real-time we need to evaluate the performance of algorithm before running that. This gap is bridged by Blelloch et. al. [1] by prediction of performance on connected machine model CM-2. This paper covers three most effective sorting algorithm radix sort, bitonic sort, and sample sort and verifies that actual performance is almost equal to predicted one. Authors have done characterization of primitive operations and time to do these operations on actual machine, after that algorithm is broke down in those operations and expected time is calculated accordingly.

Preparata et. al. [2] presented one of the earliest collection of parallel sorting setup for multiple processors. Authors divided the algorithm in to three phase namely count acquisition, rank determination and data rearrangement. Each one is doing calculation of number of smaller keys, rank and position of that element in actual sorted sequence.

Cheung et. al. [3] presented a common idea of divide and conquor mechanism of sorting to a new two stage parallel sorting. Authors divides the input to different sub-parts and sorted individually at different processors at first stage and combine these sorted sequences in second stage of execution. This paper uses general heap sort for each node at first stage.

A novel and smart approach to minimize communication cost is presented by Li et. al. [4] It uses the power of partition mechanism of known quick sort to move each element between processors just one time. This reduced the complications by efficient load balancing, ultimately good performance on both comparison and non-comparioson based algorithms.

Umeda et. al. [5] have done the comparison of three sorting algorithms using OpenMP. In their research they tried to parallelize the merge sort, quick sort and count sort and got interesting result. Authors implemented parallel version of their algorithms using OpenMP 3.0 and found that due to recursive in nature merge sort and quick sort gives the great speedup when parallelize using OMP TASK directive. On the other hand count sort does not show the same behavior. In case if count sort OMP DO gives better performance than OMP TASK.

Comparision between three different types are given in Figure 1, where Figure (a) is a plot between wall time and number of threads and Figure 1 (b) is providing the plot between speedup on y-axis and number of threads on x-axis for merge sort, quick sort and count sort.

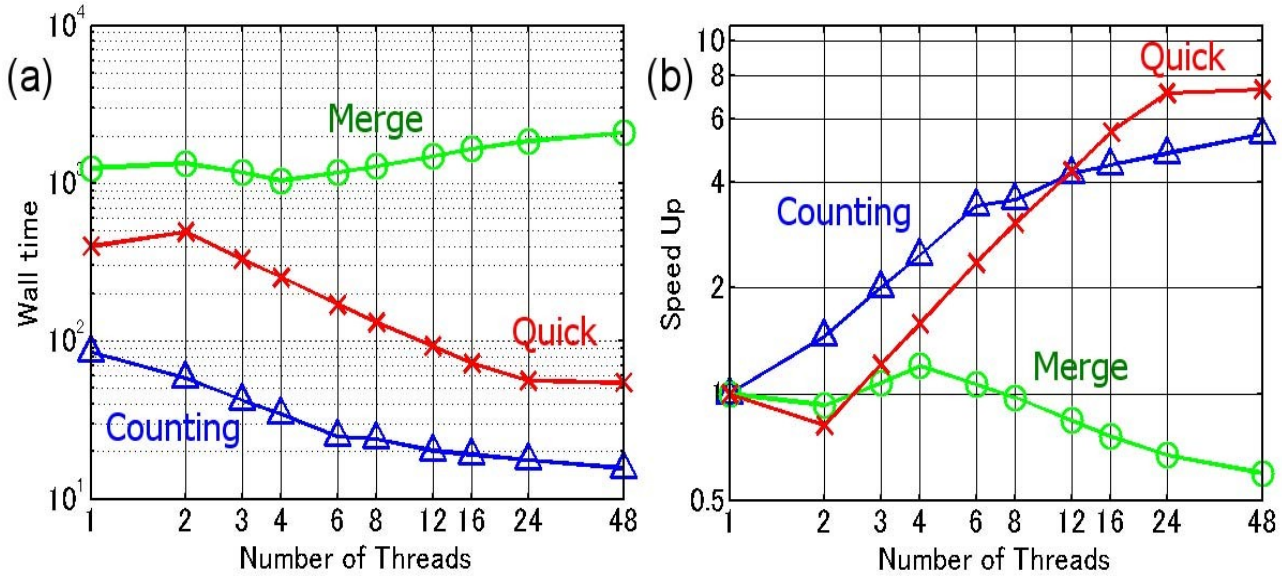

Figure 1: (a) Wall time and (b) speedup for the parallel merge sort, quick sort, and counting sort

Similarly researcher tries to compare the different algorithm using MPI that involves message passing mechanism. Durad et. al. [6] focuses on implementational aspect of parallel sorting mechanism and implemented a lot of algorithms on two different architectures SGI Virtue and computing cluster. Authors implemented binary sort, bitonic sort, hyper quick sort, merge sort, odd-even sort, quick sort, radix sort, shell sort and derived some observations. The execution time is inversely proportional to number of process and size of input. Radix sort's performance is most dependent on network communication while merge sort scales very well on the other hand, in case of odd-even merge sort at each process an abrupt behavior is there because of saturation in communication bandwidth of MPI.

Figure 2 is showing the analysis of execution time with respect to number of processing elements for two different architectures.

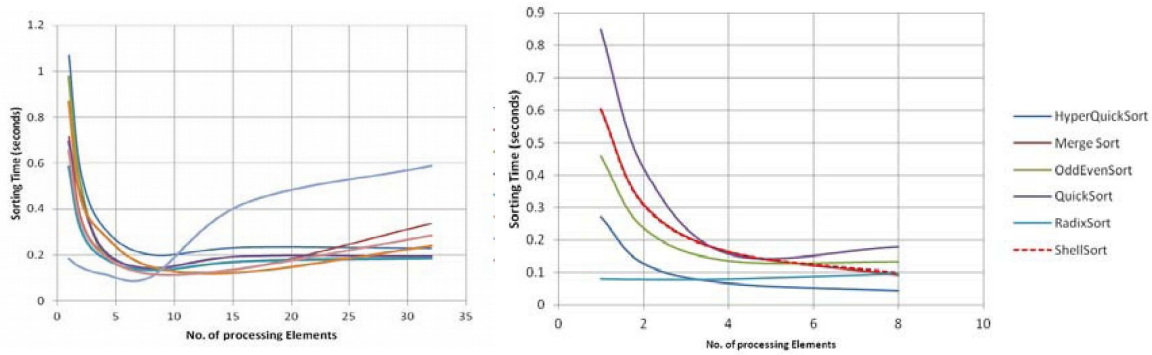

Figure 2: Execution time versus number of processing elements (Cluster System and SGI Virtue)

Another comparison based paper is by Pasetto et. al. [7] that covers both qualitatiove and qualitative aspect of performance of a large variety of architecture and sorting algorithms. Authors covered algorithms Mapsort, Mergesort, Tsigas-Zhang's Parallel Quicksort, Alternative Quicksort and STL sort with Nehalem and Westmere as evaluating architectures. It also compares the performance of direct sorting and key-pointer sorting as well.

Amato et. al. [8] have also presented an exhaustive comparative material covering three architectures namely MasPar MP1202(mesh-connected computer), nCUBE 2(hypercube multiprocessor) and Sequent Balance(distributed shared memory machine). It also evaluated two versions of bitonic sort along with sample sort, and parallel radix sort on each architecture.

3. Hybrid algorithms for sorting:

In this section, we will present the hybridization of different architectures and algorithms in view to achieve better efficiency and speedup. As we are all aware that combining the goods of two techniques always works quite well if assembled correctly. We have presented similar approaches here by various researchers.

3.1 Hybrid architecture proposals for parallel sorting:

One of the significant and notable work to achieve high performance by changing architecture is done by Radenski et. al. [9] that studied single algorithm for sorting on standalone and clustered systems. Authors stick to the merge sort as it represents a big class of divide and conquer algorithm. Implementation of this paper is done in OpenMP and MPI. First implementation is done using OpenMP on SMP systems, second is done using MPI that uses the computation nodes of cluster system. Third approach combines both by having coarse-grain parallelism using MPI with fine-grain parallelism using OpenMP within MPI processes and execute on clustered SMP systems. Performance evaluation is done on standalone Rocks cluster and AWS servers.

Figure 3 provides the comparative analysis of almost every possible configuration of paper by Radenski et. al. [9], where authors clearly explained serial, shared memeory, message passing and hybrid algorithm for a popular sorting algorithm merge sort. Authors also runned their code by varying different number of cores, CPUs, nodes, processes and threads. Figure 3 is given below for the results of cluster system named Rocks.

Program	OpenMP Threads	MPI Processes	Nodes Used	Cores Used	Average Rocks Time	Rocks Speedup
Serial			1	1	4.1	1.0
OpenMP	2		1	2	2.4	1.7
	4		1	4	1.6	2.6
	8		1	8	1.3	3.2
MPI		8	1	8	2.9	1.4
		16	2	16	2.2	1.9
		24	3	24	2.1	2.0
		32	4	32	1.9	2.2
		40	5	40	2.0	2.1
Hybrid	8	1	1	8	1.3	3.2
	8	2	2	16	1.5	2.7
	8	3	3	24	1.5	2.7
	8	4	4	32	1.9	2.2
	8	5	5	40	1.8	2.3

Figure 3: Performance results on a standalone Rocks cluster (in sec)

Alghamdi et. al. [10] similarly selected merge sort due to its property of being scalable for hybrid sorting on shared memory architecture, multiple processors connected with communication mechanism and clusters. Authors provided algorithms for sequential merge sort and quicksort, shared memory parallel merge sort, distributed memory parallel hybrid quicksort+merge sort and hybrid memory parallel merge sort using hybrid msd-radix and quicksort in cluster platforms.

Jose et. al. [11] provided an out-of-core sorting architecture based on message passing and PGAS (Partitioned Global Address Space). This architecture provides a good support for data intensive tasks. For the implementation purpose it uses MPI and OpenSHMEM PGAS. Figure 4 is given below to depict proposed Hybrid MPI+OpenSHMEM Application Stack.

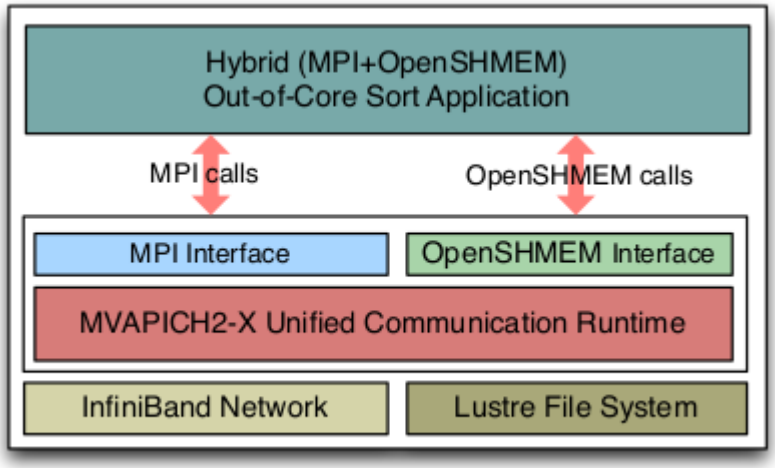

Figure 4: Hybrid MPI+OpenSHMEM Application Stack

Cheng et. al. [12] also exploit the power of parallel processing architecture by exploiting the memory structure in multicore systems. It allocates the memory for data chunks to L1 and L2 caches respectively and combines both thread level and process level parallel processing.

Todd et. al. [13] also tries to parallelize merge sort in a hybrid manner by using FIFO queue for merging procedure to reduce communication for merging data chunks. Algorithm runs faster as merge procedure runnes in an overlapped manner.

3.2 Hybrid algorithm proposals for parallel sorting:

There are a lot of attempt for combining different algorithm for efficient and speedy results. Axtmann et. al. [14] also presented an algorithm for modified versions of sample sort(AMS-sort and RLM-sort) with the solution bin packing problem and random data delivery procedure.

A variant of merge sort is proposed by Uyar et. al. [15]that creatively uses two thraeds for one merging procedure from both ends of array simultaneously. Implementation is done using fork-join model and CyclicBarrier library for java and claimed 20 to 30 % of speedup.

Similarly partition and concurrent merging (PCM) is defined by Herruzo et. al. [16] by taking intution from odd even merge sort and quick sort. PREZ algorithm works involving two processors that merge and sort two sequences simultaneously. Implementation is done by OpenMP at thread level for this paper.

4. GPU supported parallel sorting:

With the immense computation power given by Graphical Processing Units, researchers designed novel algorithm for sorting that exploit that power. In general these algorithm has two phase. In phase one data is distributed among GPU cores and local sorting got done there. After that in second step various procedures are designed to efficiently combine the data in sorted version from all GPU cores.

White et. al. [17] uses the bitonic sort for first phase to sort input chunks in parallel manner and then in second stage a reduction sorting network designed using MPI to merge the sorted subsequence from GPU clusters. Authors claim to have speedup rise of 8 to 9 percent.

Kumari et. al. [18] uses the radix sort on GPU nodes at first phase and uses parallel selection sort to get final sorted sequence. This paper also focused on Split and Concurrent Selection (SCS) mechanism.

Jan et. al. [19] provides comparative analysis of various parallel algorithm on graphical processing unit. Paper covers odd- even merge sort, rank sort and bitonic sort. It showed different speedup mertics for execution on CPU and GPU both.

Bitonic and radix sort’s parallel performance is studied by Yildiz et. al. [20] on multicore GPU systems. It also has provided the complete implementation of GPU and CPU version both and stated good results.

Yeng et. al. [21] provided a hybrid framework combining CUDA, OpenMP and MPI in detail using C1060 GPU nodes in GPU cluster. As of now this framework is not tested for parallel sorting but could provide amazing results.

Conclusion:

We have covered plenty of current state of art algorithms and techniques for parallel sorting on different type of setup. We have found out that the performance of any algorithm is very subjective on architecture and data upon which sorting needs to be done. Moreover It is found that hybrid approaches are better options as it combines the goodness of both message passing and shared memory architecture, Also there is a strong need to join both efficiently.

When talking about sorting using GPU we can say that multiple cores of GPUs became handy to sort data partially. Most of the work is going on defining an effective network that combines the data efficiently. Also since the data loading and unloading from GPU to CPU is heavy task, researchers also trid to minimize that.

References:

[1]. Blelloch, G.E., Leiserson, C.E., Maggs, B.M., Plaxton, C.G., Smith, S.J. and Zagha, M., 1998. An experimental analysis of parallel sorting algorithms. *Theory of Computing Systems*, *31*(2), pp.135-167.

[2]. Preparata, F.P., 1978. New parallel-sorting schemes. *IEEE transactions on Computers*, (7), pp.669-673.

[3]. Cheung, J., Dhall, S., Lakshmivarahan, S., Miller, L. and Walker, B., 1982, January. A new class of two stage parallel sorting schemes. In *Proceedings of the ACM'82 conference* (pp. 26-29).

[4]. Li, H. and Sevcik, K.C., 1994, August. Parallel sorting by over partitioning. In *Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures* (pp. 46-56).

[5]. Umeda, T. and Oya, S., 2015, December. Performance comparison of open-source parallel sorting with OpenMP. In *2015 Third International Symposium on Computing and Networking (CANDAR)* (pp. 334-340). IEEE.

[6]. Durad, M.H. and Akhtar, M.N., 2014, December. Performance analysis of parallel sorting algorithms using MPI. In *2014 12th International conference on frontiers of information technology* (pp. 202-207). IEEE.

[7]. Pasetto, D. and Akhriev, A., 2011, October. A comparative study of parallel sort algorithms. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion* (pp. 203-204).

[8]. Amato, N.M., Iyer, R., Sundaresan, S. and Wu, Y., 1996. A comparison of parallel sorting algorithms on different architectures. *Technical Report TR98-029, Department of Computer Science, Texas A&M University*.

[9]. Radenski, A., 2011. Shared memory, message passing, and hybrid merge sorts for standalone and clustered SMPs.

[10]. Alghamdi, T. and Alaghband, G., 2020. High Performance Parallel Sort for Shared and Distributed Memory MIMD. *arXiv preprint arXiv:2003.01216*.

[11]. Jose, J., Potluri, S., Subramoni, H., Lu, X., Hamidouche, K., Schulz, K., Sundar, H. and Panda, D.K., 2014, October. Designing scalable out-of-core sorting with hybrid MPI+ PGAS programming models. In *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models* (pp. 1-9).

[12]. Cheng, Z., Qi, K., Jun, L. and Yi-Ran, H., 2011, December. Thread-level parallel algorithm for sorting integer sequence on multi-core computers. In *2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming* (pp. 37-41). IEEE.

[13]. Todd, S., 1978. Algorithm and hardware for a merge sort using multiple processors. *IBM Journal of Research and Development*, *22*(5), pp.509-517.

[14]. Axtmann, M., Bingmann, T., Sanders, P. and Schulz, C., 2015, June. Practical massively parallel sorting. In *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures* (pp. 13-23).

[15]. Uyar, A., 2014, October. Parallel merge sort with double merging. In *2014 IEEE 8th International Conference on Application of Information and Communication Technologies (AICT)* (pp. 1-5). IEEE.

[16]. Herruzo, E., Ruiz, G., Benavides, J.I. and Plata, O., 2007, February. A new parallel sorting algorithm based on odd-even mergesort. In *15th EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing (PDP'07)* (pp. 18-22). IEEE.

[17]. White, S., Verosky, N. and Newhall, T., 2012, September. A cuda-mpi hybrid bitonic sorting algorithm for gpu clusters. In *2012 41st International Conference on Parallel Processing Workshops* (pp. 588-589). IEEE.

[18]. Kumari, S. and Singh, D.P., 2014, August. A parallel selection sorting algorithm on GPUs using binary search. In *2014 International Conference on Advances in Engineering & Technology Research (ICAETR-2014)* (pp. 1-6). IEEE.

[19]. Jan, B., Montrucchio, B., Ragusa, C., Khan, F.G. and Khan, O., 2012. Fast parallel sorting algorithms on GPUs. *International Journal of Distributed and Parallel Systems*, *3*(6), p.107.

[20]. Yildiz, Z., Aydin, M. and Yilmaz, G., 2013, November. Parallelization of bitonic sort and radix sort algorithms on many core GPUs. In *2013 International conference on electronics, computer and computation (ICECCO)* (pp. 326-329). IEEE.

[21]. Yang, C.T., Huang, C.L. and Lin, C.F., 2011. Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters. *Computer Physics Communications*, *182*(1), pp.266-269.

Assignment 3: Design Document

Introduction:

We have implemented a hybrid algorithm by taking intution from techniques presented in sections 3 and 4 where sorting is done in two phases. Two phase sorting is a popular and efficient strategy where data is divided into chunks and sorting is done on each by an independent process, thread or computation unit (in case of GPUs). After partial sorting in first step, merge procedure needs to be called by each for globally sort the data in second phase. This approach is useful where architecture supports efficient architecture to provide communication. We have used this approach in this assignment with the combination of three different algorithms since the data generation step is providing us an opportunity to minimize the communication between different processes.

We have also implemented search algorithm with minor modification from given skeleton, Need and reason for changes are presented in thelast section under the title notes.

Sorting Algorithms: Quick Sort, Merge Sort, Heap Sort

We have used two phase sorting algorithm where we have sorted the data locally in phase one and then combined them in phase two to get sorted sequence globally. For local sorting we have done quick sort, merge sort and heap sort and got the comparison result that quick sort is fastest among all. Time complexity of each is bounded by O(nlog n)

Local sorting is backed up by thread programming using OpenMP and each thread sort a subpart of process's data and then combined it. This strategy gives a boost in overall speedup.

In phase two global sorting is done to properly sort the data by rearranging the data according to its distribution. After this step we can call the search procedure for each process that is explained in the next section.

Searching Algorithm:

We have implemented a search procedure that is doing binary search for each procedure that is returning bool variable, which is has value true if search function found the key element and return false if search procedure is unable to get element.

To remove the problem induced by return of reference of local variable of function we have modified the skeleton of given search procedure by passing the reference to result variable.

We have implemented binary search within each process to get the desired element. We can use binary search in this case as data is sorted within each process. This search procedure is having complexity of O(log n).

Program Flow:

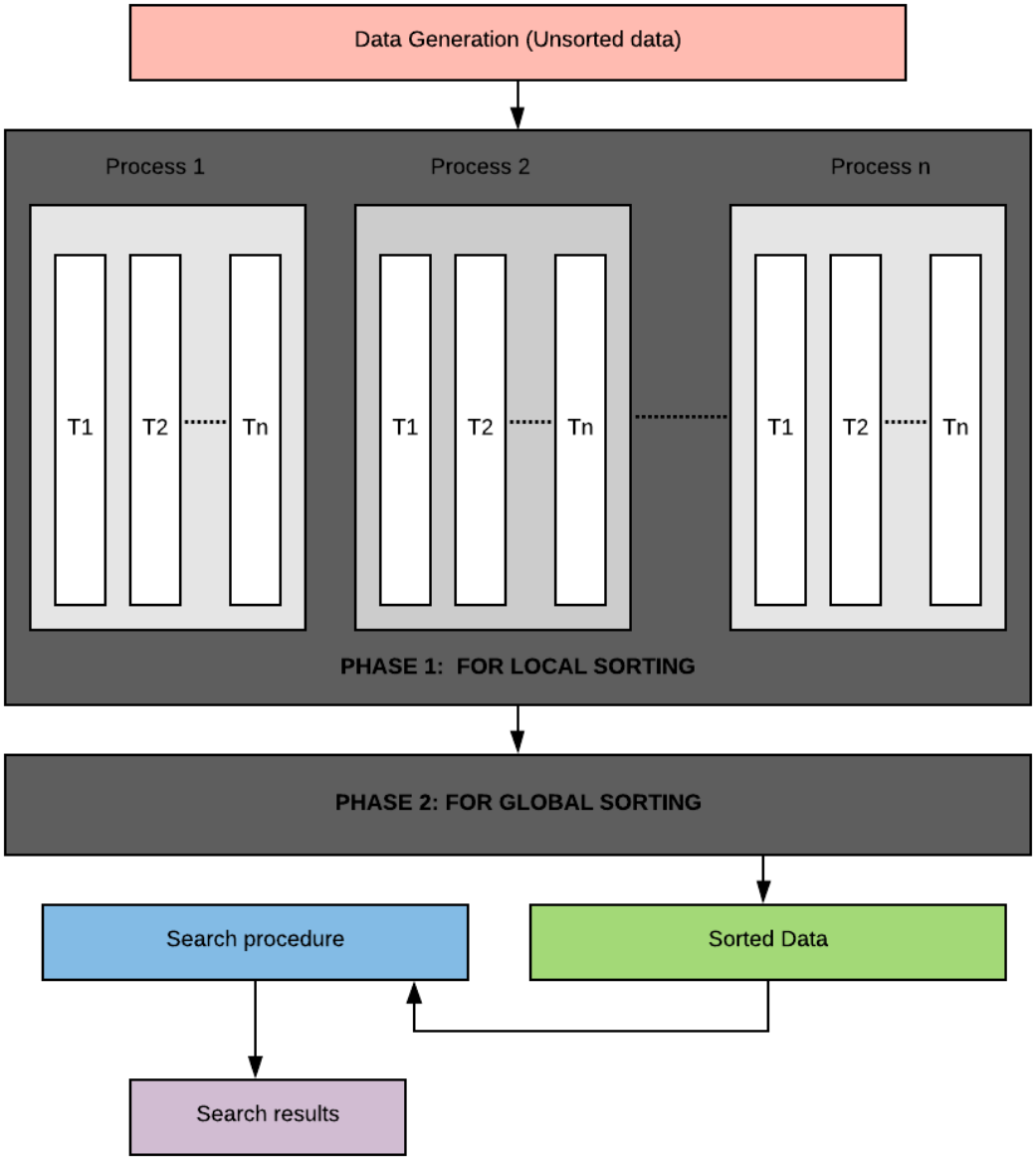

Figure 1: Program Flow

General program Flow is depicted in Figure 1 where sorting is done in two phase. At phase one sorting is done parallelly under each process using threads as depicted in Figure 2. After that in phase two data is globally sorted by using rearrange function in each process. Hence this program ensures the two level parallelism.

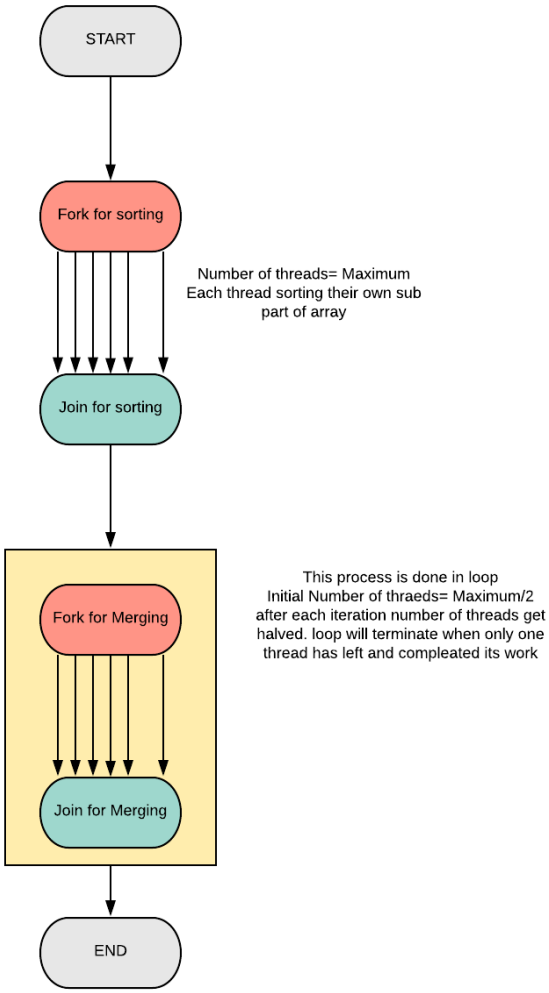

Figure 2: local sorting under each process

Implementation and Results:

We have implemented our algorithms using libraries OpenMP and MPI in c++. We have used openmp for availing thread-level programming that will be done under each process created and managed by mpi library.

Our implementation can be divided into two subphases for sorting and one procedure for searching. For the purpose of sorting design flow is given in figure 1 that shows the two phases in sorting. In phase one we have locally sorted the data by using threads in each process separately. Since we have understood the data organization we can rearrange the data to get sorted global data in phase two.

Program Execution:

Makefile is provided with the code. The program can be executed by giving following command.

Create executable file using:

mpiCC -fopenmp -c -o sort.o sort.cpp sort.h

ar -rsc libpsort.a sort.o

Run the tester code file using:

mpicxx -fopenmp tester.cpp libpsort.a

mpirun -n 64 a.out

Scaling:

Program is scaling properly. I have tested it with 2^30 data size and all of algorithms are working properly.

Notes:

Error in tester code is found and I am unable to resolve them with minor modifications hence previous tester code (give for assignment 1) is reused.

Search function parameter is changed due to design flaw that was present earlier. Model search function skeleton is returning the reference to the search key and warning is there while executing code (on linux machine with gcc compiler) for returning local variable. So I have changed the return type of search function from char* to bool that returns true if search is successful and false otherwise. For the task of getting search element we have passed the char* of same size to the function. This will remove the returning of local variable of function that can cause the problem in the future.