Tapping the Full Potential of Multicore on Intelligent Machines with Parallel Algorithms

Amudha L[1], Nithya T. M[2], Ramya J[3], Infant Raj I[4]
[1,2,3,4]*Assistant Professor, Dept. of CSE, K. Ramakrishnan College of Engineering, Tiruchirappalli, TamilNadu, India*

Abstract **- The number of cores in a computing machine is doubling each generation. As the number of cores on multi-cores increase, all cores must be effectively utilized. So there should be a different approach in programming. One such solution is Parallel Programming or Parallel Computing.Parallel computing is a type of computation in which many calculations or the execution of processes are carried out simultaneously. Large problems can often be divided into smaller ones, which can then be solved at the same time. The main reason for the difficulty in improving the efficiency is due to the conventional step by step programming of the traditional programming languages. Parallel algorithms that run on parallel processors will be a better hope for performance improvement of today's fast computing devices. Today with the advent of numerous automated machines, and big data, conventional programming languages does not promise effective use of the hardware which in turn reduces the performance. Hence this paper, unveils the parallel programming languages that can be used for intelligent machines like robots, high speed computational devices, data analytics engine, etc**

Keywords **- Sequential, parallel computation, parallel algorithms, performance, speed.**

I. INTRODUCTION

A parallel language is able to express programs that are executable on more than one processor. Parallel processing is a great opportunity for developing high performance systems and solving large problems in many application areas. During the last few years parallel computers ranging from tens to thousands of computing elements became commercially available. They continue to gain recognition as powerful tools in scientific research, information management, and engineering applications. This trend is driven by parallel programming languages and tools that contribute to make parallel computers useful in supporting a broad range of applications. Many models and languages have been designed and implemented to allow the design and development of applications on parallel computers. Parallel programming languages (called also *concurrent languages*) allow the design of parallel algorithms as a set of concurrent actions mapped onto different computing elements. The cooperation between two or more actions can be performed in many ways according to the selected language. The design of programming languages and software tools for parallel computers is essential for wide diffusion and efficient utilization of these novel architectures. High-level languages decrease both the design time and the execution time of parallel applications, and make it easier for new users to approach parallel computers.

This paper is designed to explore about the different types of parallel programming languages that can improve the efficiency of utilizing the cores of a multi processing computer. Pipelining of tasks are needed for implementing parallel algorithms. The benefits of parallel programs over sequential programs is listed below,

1. Parallel programming executes the code efficiently,
2. Parallel programming saves time
3. Parallel programming scales with program size, hence produces solution for larger problems with ease.

II. INTELLIGENT MACHINES

Unlike a computer to solve computational problems, intelligent machines are developed to solve analytical and logical problems as human brain solves, based on continuous learning. Since brain computations are done in different manner from a digital computer. A digital computer does computations sequentially whereas, human brains makes use of the massive network of parallel and distributed computational elements called neurons. This large number of connections gives the human with powerful capability of learning. Motivated by this biological model, scientists today has decided to build computational systems that can process information in a similar way as the brain does. Astonishing developments in semiconductor technology has given new life to the machines, with the aid of artificial intelligence techniques. Machine with vision processes are made up of Extensive Graphics Processing Units (GPUs), multicore very Long Instruction Word Digital Signal Processors (VLIW DSPs).Any typical DSP multicore can be used for intelligent processing.

III. MULTICORE COMPUTATION

Multi core computation uses a single computing element with 2 or more cores (CPUs). Each such core is usually integrated into a separate IC chip. Also each core has dedicated memory and cache that helps in storage and runtime efficient storage.

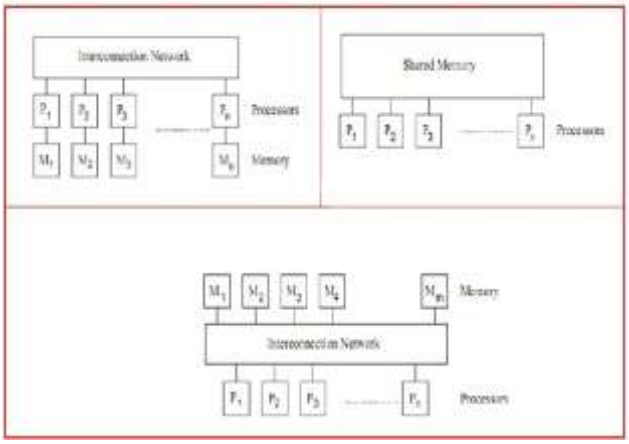

Fig 1. Multicore computing architecture

IV. PARALLEL ALGORITHMS

Some of the standard parallel algorithms are Map Scatter, Gather Reduction Scan Search, Sort.Parallel algorithms deals with shared memory. Certain issues like how processors are activated and how the shared memory is accessed is controlled by the parallel algorithm and its compiler.
The algorithmic building blocks of parallel algorithms are Map, Scatter, Gather and Scatter Reductions, Scan, Sort and Search.

1) MAP

Map is a parallel computing paradigm where simple operations are applied to all elements parallel. It is used to solve problems that can be decomposed into independent subtasks requiring no communication or synchronization between the computing elements. Fig.2 shows a graphical representation of solving Depth First Search traversal using map reduce concept.

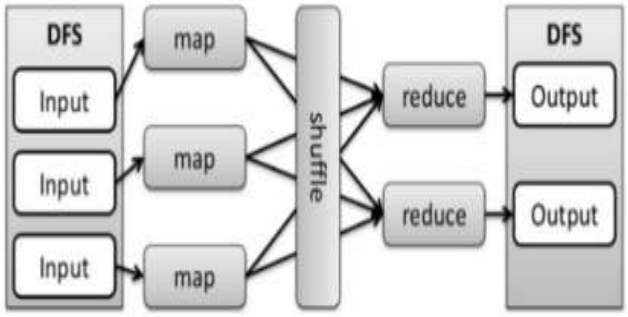

Fig2. Map reduce with DFS

2) SCATTER

Scatter module writes a single data item to multiple locations. The GPU support is provided at elementary level using global memory. Scatter is faster if locality and repeated access can be exploited. Gather module reads multiple data items to a single location.

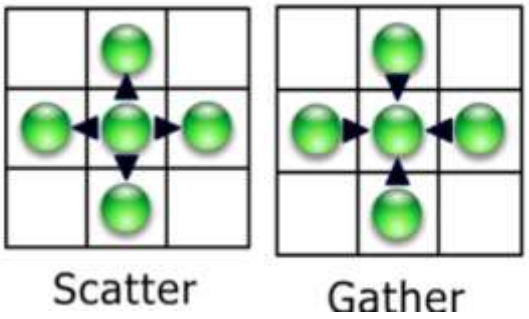

Fig 3. Scatter-Gather representation

3) REDUCTION

With identity I⁄Binary associative operator, Ordered set s = [a0, a1, ..., an-1] of n elements a_{n-1}⁄ ... ⁄ a_1⁄, s) returns a_0⁄ Reduce. For Example: reduce (+, [3 1 7 0 4 1 6 3]) = 25 Reductions common in parallel algorithms. Common operators used in reduce operation are +, ×, min and max.1D parallel reduction is a technique that adds two halves of list together repeatedly until we are left with a single value.

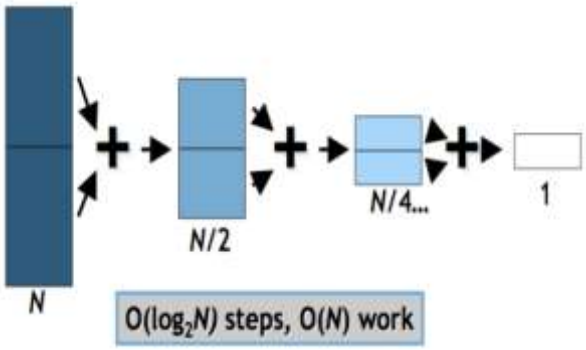

Fig 4. 1D Parallel Reduction

4) CUDA REDUCTION

Tree-based approach used within each thread block Need to be able to use multiple thread blocks. Cuda reduction helps to process very large arrays and to keep all multiprocessors on the GPU busy always. During cuda reduction each thread blockreduces a portion of the array.

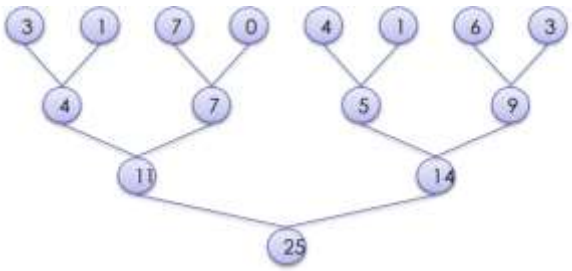

Fig. 5 Tree notation of CUDA GPU reduction

5) SCAN

Binary associative operator with identity I Ordered set s = [a0, a1, ..., an-1] of n elements a1), ..., (a0⁄, s) returns[I, a0, (a0 ⁄Scan(an-2)]⁄ .../ a1⁄

Example: scan(+, [3 1 7 0 4 1 6 3]) =[0 3 4 11 11 15 16 22]

Radix sort Sparse matrices Quicksort Polynomial evaluation String comparison Solving recurrences Lexical analysis Tree operations Stream compaction Histograms

V. EXPERIMENTAL EVALUATION

With the increase in number of cores, larger problems can be solved in milliseconds, which might take very large time if the same computation is done using sequential program with a single processor or even with multi-threading. To evaluate the processing power of the cores, this paper tests with the Bitonic Sort algorithm. The number of comparisons done by bitonic sort is greater than the number of comparisons done by merge sort or quick sort. But Bitonic sort is always better for parallel computational processors, since the algorithm uses a recursive approach in aligning the numbers in the sorted sequence.

Bitonic principle:

A sequence is called Bitonic if it is first increasing, then decreasing. In other words, an array arr[0..n-i] is Bitonic if there exists an index j where 0<=j<=n-1 such that, **x0 <= x1 …..<= xj andxj>= xj+1….. >= xn-1.**

1. A sequence, sorted in increasing order is considered Bitonic with the decreasing part as empty. Similarly, decreasing order sequence is considered Bitonic with the increasing part as empty.
2. A rotation of Bitonic Sequence is also bitonic.

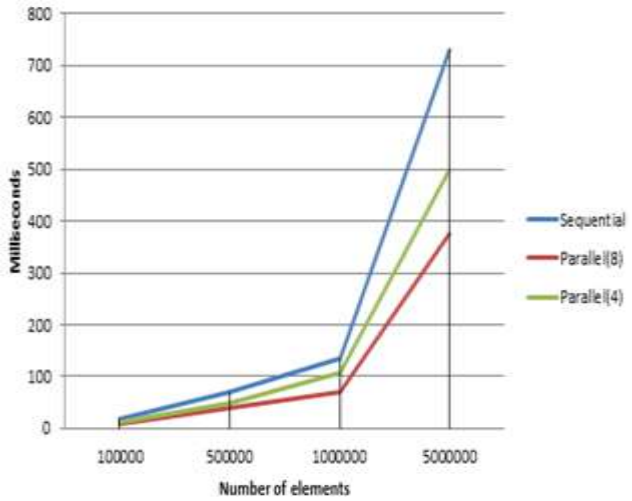

Fig. 6.Sequential Vs Parallel efficiency

Speed up: Speedup = the quotient between the speed of the parallel algorithm and the speed of the corresponding sequential algorithm.

A parallel algorithm is efficient if and only if it is fast and the product of parallel time and the number of processors is close to the time of the sequential algorithm. That is, if N is the number of cores and T_{seq} is the time to complete using sequential algorithm and T_{par} is the time taken to execute the algorithm using parallel algorithm, $T_{par} X N \approx T_{seq.}$

VI. EVALUATION RESULTS ON SORTING

The following chart is the pictorial representation of the comparing the speed of different types of sequential algorithms like merge sort, radix sort, quick sort with parallel algorithms IBR Bitonic sort and Bitonic sort. The graphs shows the radical increase in speed for the IBR bitonic sort algorithm for parallel programming.

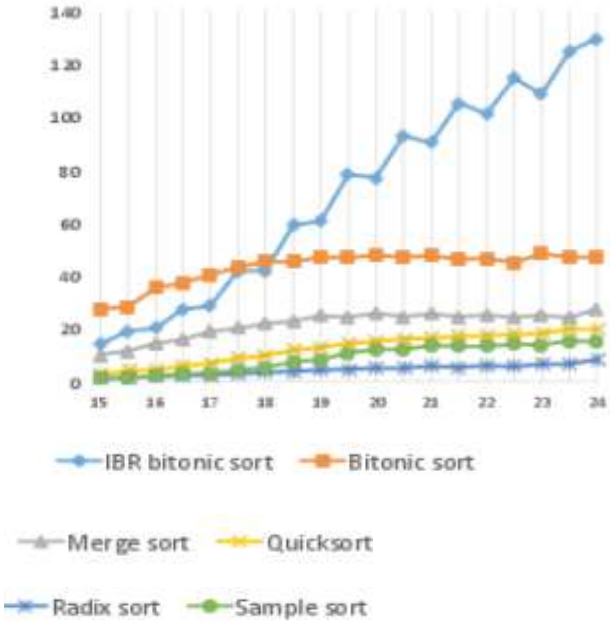

Fig 7. Comparison of sorting algorithms with parallel computation

VII. CONCLUSION

Recent work on parallel algorithms has focused on solving problems from domains such as pattern matching, data structures, sorting, computational geometry, combinatorial optimization, linear algebra, and linear and integer programming. However, some parallel computers cannot efficiently execute all algorithms, even if the algorithms contain a great deal of parallelism. Thus the magic lies in identifying the feasibility and performance gain that can be achieved with the type of parallel algorithm and the problem that is to be solved.

REFERENCES

[1]. Peters H, Schulz-Hildebrandt O, Luttenberger N. Fast in-place, comparison-based sorting with CUDA: A study with bitonic sort. Concurr.Comput. : Pract. Exper. May 2011; 23(7):681–693, doi:10.1002/cpe.1686. URL http://dx.doi.org/10.1002/cpe.1686.

[2]. Peters H, Schulz-Hildebrandt O, Luttenberger N. A novel sorting algorithm for many-core architectures based on adaptive bitonic sort. 26th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2012, Shanghai, China, May 21-25, 2012, 2012; 227–237, doi:10.1109/IPDPS.2012.30. URL http://dx.doi.org/ 10.1109/IPDPS.2012.30.

[3]. Dehne F, Zaboli H. Deterministic sample sort for GPUs. CoRR 2010; abs/1002.4464.URL http://dblp.uni-trier.de/db/journals/corr/corr1002.html#abs- 1002- 4464.

[4]. Harris M. Optimizing Parallel Reduction in CUDA. Technical Report, nVidia 2008. URLhttp://developer. download.nvidia.com/assets/cuda/files/reduction.pdf.

[5]. Cormen T H, Leiserson C E, Rivest R L, Stein C. Introduction to Algorithms, Third Edition. 3rd edn., The MIT Press, 2009.