LITERATURE REVIEW

This chapter provides a bird's-eye view of parallel processing. Mainly this chapter is a Literature review of current and old literature. Different papers review like 2.1 Review Paper –I, 2.2 Review Paper –II, 2.3 Review Paper –III, 2.4 Review Paper –IV, 2.5 Review Paper –V, 2.6 Review Paper –VI, 2.7 Review Paper –VII, 2.8 Review Paper –VIII, 2.9 Review Paper –IX, 2.10 Review Paper – X, 2.11 Review Paper –XI, 2.12 Review Paper –XII.

Review Paper –I: C.Y. Lin, J.S. Liu, and Y.C. Chung, [42] , describe that Extended Karnaugh map Representation Scheme , the idea is based on the Karnaugh-map . When n=1 and 2 the TMR and EKMR schemes are the same. But in this paper more than two dimensional matrix multiplications explained. "Matrix operations are the core of many linear systems. Efficient matrix multiplication is critical to many numerical applications, such as climate modeling, molecular dynamics, computational fluid dynamics and etc. Much research work has been done to improve the performance of matrix operations. However, the majority of these works is focused on two-dimensional (2D) matrix. Very little research work has been done on three or higher dimensional matrix. Recently, a new structure called Extended Karnaugh Map Representation (EKMR) for n-dimensional (nD) matrix representation has been proposed, which provides better matrix operations performance compared to the Traditional matrix representation (TMR). The main idea of EKMR is to represent any nD matrix by 2D matrices. Hence, efficient algorithms design for nD matrices becomes less complicated. Parallel matrix operation algorithms based on EKMR and TMR are presented. Analysis and experiments are conducted to assess their performance. Both our analysis and experimental result show that parallel algorithms based on EKMR outperform those based on TMR..

Review Paper –II: C.Y. Lin, Y.C. Chung, and J.S. Liu, [43] describe that "Matrix operations are the core of many linear systems. Efficient matrix multiplication is critical to many numerical applications, such as climate modelling, molecular dynamics, computational fluid dynamics and etc. Much research work has been done to improve the performance of matrix operations. However, the majority of these works is focused on two-dimensional (20) matrix. Very little

research work has been done on three or higher dimensional matrix. Recently, a new structure called Extended Karnaugh Map Representation (EKMR) for n-dimensional (nD) matrix representation has been proposed, which provides better matrix operations performance compared to the Traditional matrix representation (TMR). The main idea of EKMR is to represent any nD matrix by 2D matrices. Hence, efficient algorithms design for nD matrices becomes less complicated. Parallel matrix operation algorithms based on EKMR and TMR are presented. Analysis and experiments are conducted to assess their performance. Both our analysis and experimental result show that parallel algorithms based on EKMR outperform those based on TMR..

Review Paper –III: B.B. Fraguela, R. Doallo, E.L. Zapata, [19] "Several fast sequential algorithms have been proposed in the past to multiply sparse matrices. These algorithms do not explicitly address the impact of caching on performance. We show that a rather simple sequential cache--efficient algorithm provides significantly better performance than existing algorithms for sparse matrix multiplication. We then describe a multithreaded implementation of this simple algorithm and show that its performance scales well with the number of threads and CPUs. For 10% sparse, 500 X 500 matrices, the multithreaded version running on

4--CPU systems provides more than a 41.1--fold speed increase over the well--known BLAS routine and a 14.6 fold and 44.6--fold speed increase over two other recent techniques for fast sparse matrix multiplication, both of which are relatively difficult to parallelize efficiently.

Review Paper –IV: A.J.C. Bik and H.A.G. Wijshoff[4], describe in his research distinguishes two types of test case prioritization: general and version-specific. In general test case prioritization, given program P and test suite T, test cases in T are prioritized with the goal of finding a test case order that will be useful over a sequence of subsequent modified versions of P. In contrast, in version-specific test case prioritization, given program P and test suite T, test cases

in T are prioritized with the intent of finding an ordering that will be useful on a specific version P' of P. Version specific prioritization is performed after a set of changes have been made to P and prior to regression testing P'.

Review Paper –V: Bryant Whittier York[34] In this paper, analyze single-node performance of sparse matrix-vector multiplication by investigating issues of data locality and fine-grained parallelism. We examine the data-locality characteristics of the compressed-sparse-row representation and consider improvements in locality through matrix permutation. Motivated by potential improvements in fine-grained parallelism, we evaluate modified sparse-matrix representations. The results lead to general conclusions about improving single-node performance of sparse matrix-vector multiplication in parallel libraries of sparse iterative solvers." portal.acm.org.

Review Paper –V: Hans P. Zima, Peter Brezany, Barbara Chapman, Piyush Mehrotra, and Andreas [116] it is worthwhile asking why the programmer should need to concerned at all with the parallelization of the code. In fact, the most ambitious parallel programming paradigm is to use parallelizing compilers, which, ideally, reschedule and distribute the computation in such a way that the source code, which can be written as if for a sequential machine, executes efficiently on the target machine.

Review Paper –VII: A.J.C. Bik and H.A.G. Wijshoff [5] had presented an integrated solution to model-based test generation and regression test selection. It includes traceability relationships required to support regression test selection are created during test case generation. Author had compared their approach to other approaches using a case study and the regression test evaluation framework. There approach assumed that models are primary artifacts in the software development process. Code generation tools should evolve and become reliable to the point that testing compliance of code with models at the adopted abstraction level will be unnecessary.

Review Paper –VIII: Margreet Louter-Nool [212] The parallel programming paradigm most relevant to the methodology advocated in this thesis is data-parallel programming. Data parallelism is a style of coding advocated particularly for SIMD massively parallel architecture; the term was coined in an influential 1986 paper by, who gave data-parallel solutions for several classic problems such as computing the sum of an array of numbers, general parallel prefix algorithms, regular-language parsing, and several graph algorithms. The data-parallel coding paradigm is intended to act on large amounts of data with comparatively few threads of control.

Review Paper –IX: Jack J. Dongarra, Fran Goertzel Gustavson, and A. Karp [144] The ENIAC's parallelism was relatively short-lived. The machine was completed in 1946, at which time the first stored program computers were already being designed. It was later realized that the ENIAC could be reorganized in the centralized fashion of these new computers, and that when this was done it would be much easier to put problems on the machine: Thus the first general purpose electronic computer, built with a parallel centralized architecture, operated for most of its life as a serial centralized computer.

Review Paper –X: M. Struik [227] The success of the data-parallel paradigm inspired both the implementation of computer languages which directly supported it, and formal semantic treatment of data-parallel constructs. There have also been a number of formal approaches to data parallel coding, many of which have been implicitly inspired by Backus' famous Turing award lecture, in which he advocated using algebraic primitives to model the interaction of complex operators.

Review Paper –XI: Sagan. Curves [112] In the theory of categorical data-types, data values are presumed to take their values from a particular category, which is usually presumed to have some additional structure, such as being Cartesian closed. Many constructions in standard programming languages, such as ordered-pair and arrays of arbitrary data-type, can then be expressed as operations on the underlying category. Since programming language functions can

be considered to be factors, algebraic tools can be utilized in deriving and verifying programs. There have also been a number of attempts to build a general theory of arrays [112,175, 183, 218], most of which are closely related to APL and to the Bird-Martens theory of lists. Although these methodologies are extremely general and powerful, they tend to sacrifice peak performance.

Review Paper –XII: Carl David Tolmfe Runge [44] The RW-400 Data System is a new design concept. It was developed to meet the increasing demand for information processing equipment with adaptability, real-time reliability and power to cope with continuously-changing information handling requirements. It is a polymorphic system including a variety of functionally-independent modules. These are interconnect able through a program-controlled electronic switching centre. If required, many pairs of modules may be independently connected, disconnected, and reconnected, in microseconds so as to meet continuously-varying processing requirements. The system can assume whatever configuration is needed to handle problems of the moment. Hence it is best characterized by the term 'polymorphic'(having many Shapes).

The cost of a parallel algorithm is defined as the product of the running time of the parallel algorithm and the number of processors used. Cost = Running time X Number of Processors If the cost of the parallel algorithm matches the lower bound of the best known sequential algorithm by a constant multiple factor then the algorithm is said to be cost optimal.

The algorithm for adding n numbers takes O(log n) steps on an n-1 processor tree. Thus the cost of parallel algorithm is given by O(n log n) whereas the sequential algorithm in this case takes O(n) times. Thus a parallel algorithm is not cost optimal. The efficiency of a parallel algorithm is defined as the ratio of the worst case running time of sequential algorithm to the cost of parallel algorithm. Parallel processing can be achieved using two public domain message passing system namely PVM and MPI. Message passing Interface (MPI) – MPI is a standard specification for a library of message passing functions. MPI specifies a public domain platform independent standard of message passing library which is portable. An MPI library consists of names, calling sequences and results of subroutines from FOTRAN 77

programs and functions to be called from C programs. Users write their programs in FOTRAN 77 and C and are compiled with ordinary compilers, which are linked to the MPI libraries.

A parallel program written in FOTRAN 77 or C using the MPI library could run without any change on a single PC, a workstation, a network of work station, a parallel computer from any vendor or in any operating system. The design of MPIs is based on four orthogonal concepts, which are message data types, communicators, communication operations and virtual topology. Compiling an MPI program – A program written using MPI format should be compiled using the parallel C compiler mpcc. mpcc function name.

Message Passing in MPI – Processes in MPI are heavy weighted and single threaded with separate address spaces. Since one process cannot directly access variables in another process's address space, message passing is used in inter-process communications. The routine MPI_Send and MPI with RECV are used in order to send/receive a message to or from a process. A message has two parts namely, the content of the message (message buffer)and the destination of the message (message envelope).

The MPI routine has six parameters, the first three specifies the message address, message count the message data type. MPI introduces data type identifier to support heterogeneous computing and to allow messages from non-contiguous memory locations. The last three parameters specify the destination process id of the process, tag and communicator respectively. These three parameters constitute together the message envelope. Point to Point Communications – MPI provides both blocking and non-blocking operations, and non blocking versions whose completions can be tested for and waited for explicitly. MPI also has multiple communications mode.

The standard mode corresponds to current practice in message passing systems. The synchronous mode required are send to block until the corresponding receive has occurred. In buffered mode a send assumes the availability of certain amount of buffer space which must be previously specified by the user program through a routine call MPI_Buffer_attached (buffer_size) that allocated a user buffer. The ready mode is a way for the programmer to notify the system that the corresponding receives has already received, so that the underlying system

can use a faster protocol. When all the processes in a group participate in a global communication operation, the resulting communication is called the collective communication.

In most parallel programs, each process communicates with only a few other processes and the pattern of communication within these processes are called an application topology. MPI allows the user to define a virtual topology. Communication within this topology takes place with the hope that the underlying network topology will correspond and expedite the message transfer. An example of virtual topology is the Cartesian or Mesh Topology.