Memory Efficient Application Execution In MIP SCOC

Venkateswaran N*, Arvind M†, Karthik C†, Karthik G†, Vishwanath V† and Viswanath K†
*Director, WAran Research FoundaTion(WARFT)
Chennai - 600 033 Email: warf@vsnl.net
†Research Trainee, WARFT,
†Names Arranged in Alphabetical Order

***Abstract*— The Von-Neumann bottleneck was greatly reduced by the PIM architecture. The fine grain logical and physical integration of memory and processing elements inherent in the Memory In Processor (MIP)architecture, proposed earlier, further overcomes the bottleneck. The characteristic Algorithm Level Functional Units (ALFUs) account for a novel Algorithm Level Instruction Set Architecture (ALISA). ALISA is highly efficient over the conventional ALU-based instruction set architecture due to substantial reduction of memory access in application execution. Synthetic applications representing the real world problems are fabricated involving a random mix of computation and communication characteristics. These synthetic applications are employed in simulations to establish the superior performance of ALISA over the ISA of ALU-based PowerPC440. Results obtained by the extended simulation, on the MIP cluster further establish the superior memory access effectiveness of the MIP SCOC.**

I. Introduction

Evolution of novel architectural concepts is essential to successfully harness DSM technology which greatly aggravates the Von-Neumann bottleneck [1][2]. The concept of Processor In Memory (PIM) architecture[2][3]was the first step that moved away from the conventional system design approaches. In PIM-based architectures, memory and processing logic are placed on the same die. PIM exposes substantially greater memory bandwidth while imposing significantly low memory latency. Major supercomputer projects based on PIM are VIRAM, Gilgamesh, BlueGene/L and DIVA [4][5][6][7]. The Memory In Processor (MIP)[8][9] architecture was proposed with the objective of improving the memory-processor bandwidth and attaining TeraOPS peak performance at the node level itself. This is achieved by evolving an unconventional node and instruction set architecture which will harness the power of DSM technology giving rise to another class of processing paradigm - Super Computer On a Chip (SCOC) [8][9].

Section II discusses the various features of the MIP SCOC including its Instruction Set Architecture, compiler design and Algorithm Level Functional Units. Section III introduces the concept of synthetic application design. Section IV deals with evaluating the memory efficiency of the MIP architecture employing the synthetic applications. Section V extends the simulations to the cluster level.

II. MIP REVIEW

This review is based on the past papers on the Memory In Processor architecture[10][8][9]. With the DSM technology one can place hundreds of ALUs and Registers and evolve a powerful node. However such an approach is naive at best, as billions of low level instructions need to be executed and mapping within a node (resource allocation) becomes extremely complex. To reduce the above complexity, the node should posses a wide class of Algorithm Level Functional Units(ALFUs) like Matrix Multipliers and Graph theoretic units.In this context the Architecture of all these functional Units should be designed to have a memory like Cell-Based architecture.

Because of the increasing demand in the application areas like protein folding, galaxy simulation, computational fluid dynamics, it becomes mandatory to evolve supercomputing clusters whose node architecture is highly tuned towards the application. This to a large extent alleviates the time complexity which is currently predicted in petaflop years for applications running in general purpose supercomputers. But evolving such architectures for specific applications may be cost prohibitive. Under such conditions it is necessary to make the node architecture heterogeneous involving varied functional units covering wide applications. This necessarily leads to the evolution of a new concept of simultaneous multiple application mapping to efficiently harness the heterogeneous functionality of the cluster nodes.

Though the simultaneous multiple application mapping may appear complex, it is resolved by the higher level library design resulting in a simplified mapping process. On the other hand, the libraries for general purpose supercomputers are designed specific to every application and are bound to be complex due to their generic nature. Thus the general purpose supercomputers become unsuitable for handling simultaneous multiple applications.

All Functional Units are organized as 2-D Cell Arrays integrating 1 Bit SRAM as in Figure.1 with every cell of the Functional Units.

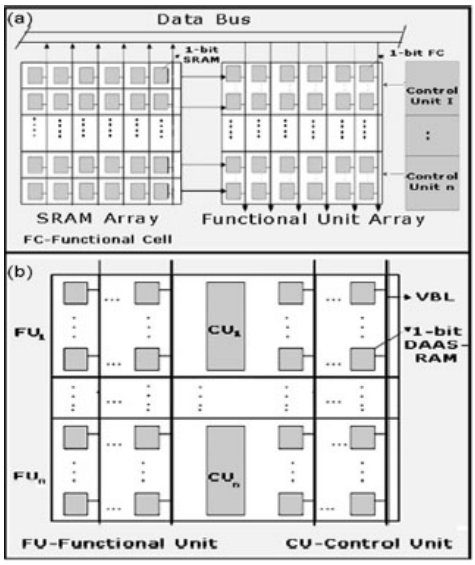

Fig. 1. a)Merging of 1 bit SRAM into Functional cell. b)MIP Architecture with memory integrated into the processor.

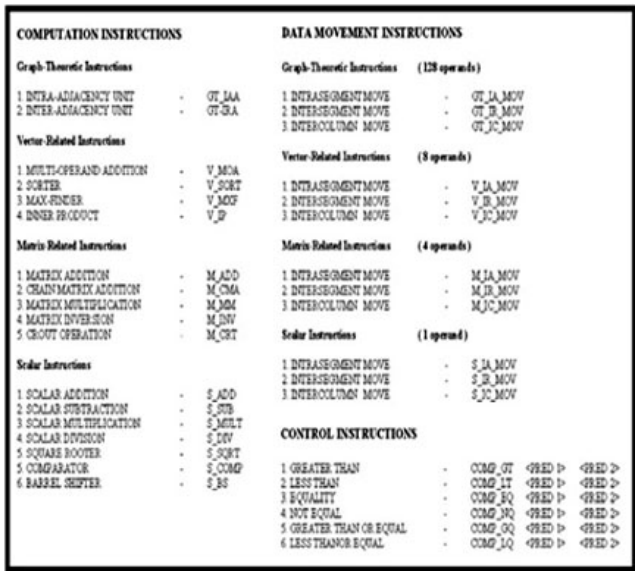

Fig. 2. MIP Instruction Set Architecture.

The control of both the processor and the memory are also integrated at the physical level. Based on extensive analysis[8] of wide class of Algorithms the different types of ALFUs are organized into Segments. These segments are logically grouped into columns. There are eight columns each having four segments. Inter and intra column communication is provided using three stage non-blocking Clos interconnection network[11].

A. MIP Instruction set Architecture

The MIP (Memory In Processor) Instruction Set Architecture is a higher order ISA designed to suite the algorithm level capabilities of the MIP SCOC functional units. The MIP SCOC ISA is referred as Algorithm Level Instruction Set Architecture (ALISA) and is presented in the Table shown in fig.2. The instructions of ALISA are categorized into Graph-theoretic, Vector-Related, Matrix Related, Scalar, ALFU - ALFU and SRAM - ALFU data transfer.

Every instruction in ALISA is a substitute for a large number of corresponding ALU instructions. Each instruction in ALISA is at a sub-algorithm level that gets executed on an ALFU directly. This brings down the corresponding number of memory fetches in MIP SCOC over the conventional ALU based node architecture of the current supercomputing clusters[12][13]. The design of ALISA makes the MIP Instruction Set Architecture simpler.

B. MIP Compiler On Silicon

In MIP SCOC, due to the presence of ALFUs, a rigorous mapping process is involved in scheduling ALISA instructions and balancing computations within the ALFUs population. This mapping in the MIP SCOC node is a complex process. To manage the complexity, a hardware based compiler is needed matching the MIP SCOC performance. Compilers for superscalar processors were the first to incorporate implicit detection of instruction level parallelism and scheduling the same. The most often used libraries will exist in the executable form. These executable libraries are bound to involve billions of instructions. Though the library exists in executable form, resource allocation (as in superscalars) during execution will be a time consuming process. In particular, if the resources within a node as in MIP SCOC become heterogeneous and large in number, a software based methodology will be too tedious a process for making optimal resource allocation. Hence, we resort to a hardware based solution (mapping and scheduling)for resource allocations in the MIP SCOC reducing the compilation process delay.

This hardware means is achieved by a set of eight Secondary Compilers On Silicon (SCOS)[14] mesh connected units driven by a Primary Compiler On Silicon (PCOS) unit as in Figure.3. These PCOS and SCOS units posses a MIP based design similar to that of ALFUs. In order to enable concurrent instruction issue to the various columns, the functionality of the COS has been partitioned into two levels specifically as the primary and secondary compilers on silicon. These SCOSs function under a common host-compiler (Primary COS - PCOS), which control the allocation of the sub-libraries. The problem is allocated by the hosts (in a cluster environment) to the PCOS in terms of built-in libraries. The PCOS takes up the incoming set of libraries, decomposes them into sub-libraries and passes them to the respective SCOSs for execution within a particular column.

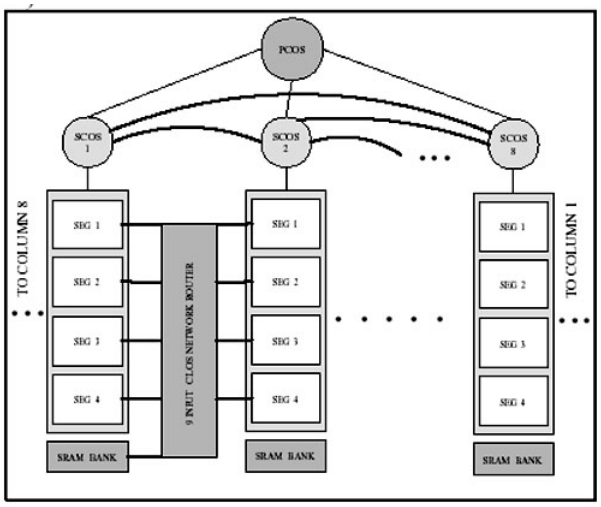

Fig. 3. A Basic MIP Node Architecture.

The PCOS takes care of mapping the problem to the different SCOSs that are under its control. The execution of the sublibraries is carried out by the SCOSs independent of each other and they communicate with the PCOS upon completion. As problem execution proceeds, communication between the SCOSs may be required to move the sub-library results across the columns for further execution. A mesh-connected topology is employed for linking the eight SCOSs for efficient execution.

1) PCOS Library specification: The library specification within the PCOS provide details for decoding and performing the necessary operation. This specification provides information about the operation to be performed, logical schedule time instants, dependencies and HLF units required. In the PCOS, the library is broken into sub-libraries. When multiple sub-libraries are associated with the same time-stamp, they are scheduled in parallel. Based on the SCOS resource constraints, PCOS might schedule the sublibraries over multiple cycles.

2) SCOS Sub-Library specification: The sub-libraries are executed within a specific column controlled by a SCOS. These sub-libraries break into HLF instructions that perform the required operation. A single instruction word as shown in fig.4 can trigger multiple HLF units simultaneously. Data movement within a column occurs due to dependency between the HLF instructions. Based on operation the appropriate Move instruction triggers transfer within or across segments.

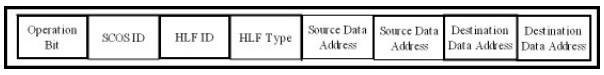

Fig. 4. HLF Instruction Word.

C. ALFU(Higher Level Functional Units)

The instruction control words for executing the algorithm level instructions in the respective ALFUs present in the column segments are issued by the different SCOSs. These instruction control words are generated with their associated operands and their corresponding HLF id. These instruction control words trigger the ALFU's local controller and the necessary operations are performed to carry out the execution of the algorithm level instruction. To simplify the decoding process of identification of HLF unit types, each HLF type has a field in the instruction control word shown in Fig. If a particular HLF unit is not being triggered for a particular operation, a 'NO-OP' fills up that field.

III. Synthetic application

The synthetic applications are random directed graphs as shown in fig.5, with nodes representing algorithms with edges depicting the dependency across these algorithms. These algorithms are chosen randomly from an algorithm set composed of a varied class of algorithms involving varying computational and communication complexities.

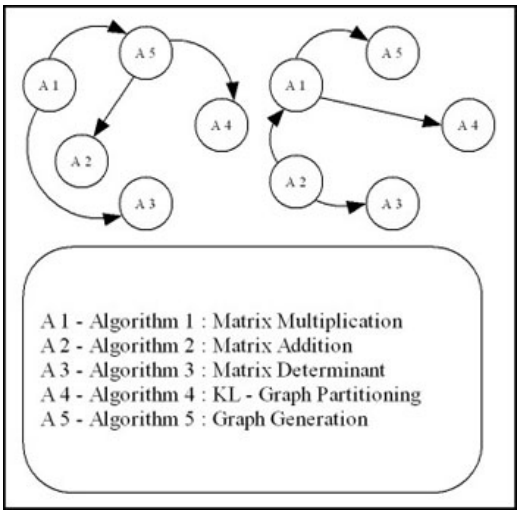

Fig. 5. Synthetic application generation.

Refer to Figs.7,8,9,10,11 illustrating the synthetic applications with random dependencies.

It is ensured that the random graph generated has computationally meaningful dependencies. Also appropriate and varied problem sizes are fixed for the algorithms. The nature of the algorithms considered for application synthesis cover matrix-oriented, graph theoretic, sorting based, vector and scalar related classes. These synthetic applications can be tuned to match real world applications. To effectively analyze the MIP and ALU based architectures with regard to overall computational and communication complexities and memory access, randomly dependent algorithm mixes are employed.

A 1 - Determinant
A 2 - LU Decomposition
A 3 - Determinant Using Lower Matrix
A 4 - Random Graph Gene\ration
A 5 - KL Graph Partitioning
A 6 - Large Addition
A 7 - Large Multiplication
A 8 - Square Root
A 9 - Matrix Multiplication
A 10 - Matrix Comparison
A 11 - Random Number Generation
A 12 - Sorting Algorithm

Fig. 6. List of Algorithms involved in developing Synthetic applications.

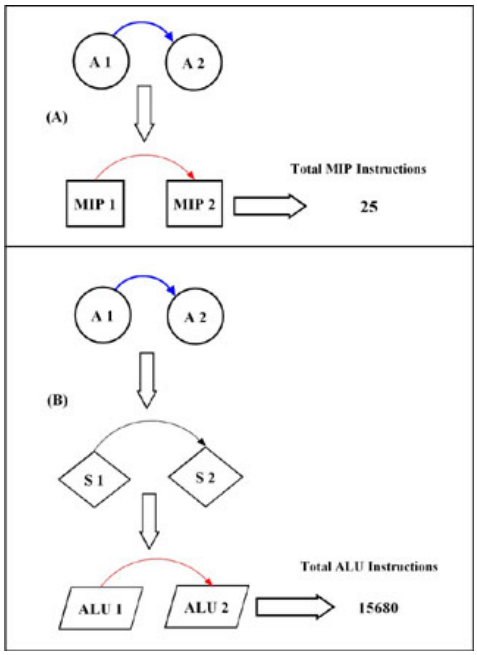

Fig. 7. Synthetic Application 1

As already stated in the introduction, the MIP SCOC has heterogeneous class of ALFUs, capable of executing a complex algorithm mix. These synthetic applications are mapped on to a MIP SCOC and we have estimated the number of MIP instructions being executed. The algorithm mix (synthetic applications) is disassembled corresponding to an ALU based processor architecture. The processor chosen is the IBM PowerPC-440. The memory access count (MIP POWER PC 440) is analyzed for widely varying synthetic applications, for different problem sizes of the algorithms involved. Refer the graph given Figs(simulation graphs).

These simulations are extended to the cluster level where every application running in the cluster is assigned a computational complexity based on that of the algorithms involved in the applications. This is expressed in terms of a vector depicted in fig.12, which denotes the number of operations in terms of matrix, graph, scalar, and vector, to be executed in the application.

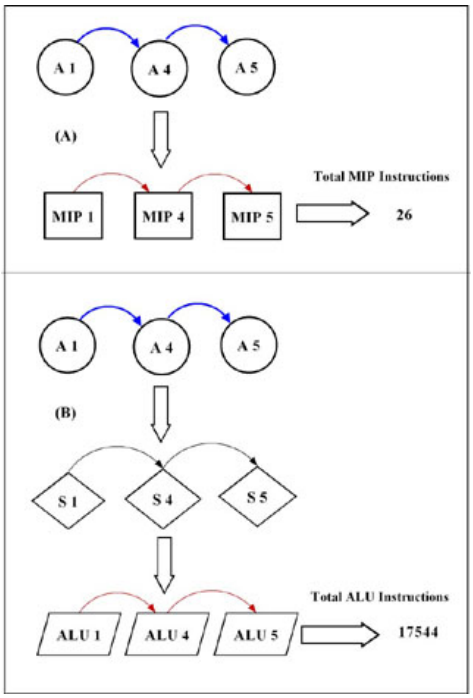

Fig. 8. Synthetic Application 2

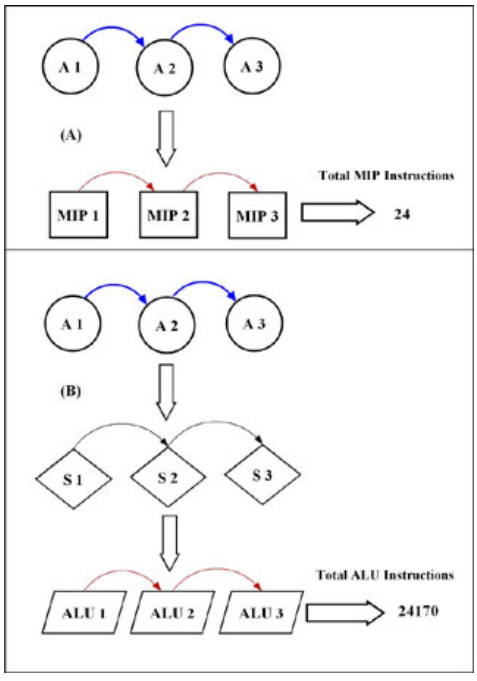

Fig. 9. Synthetic Application 4

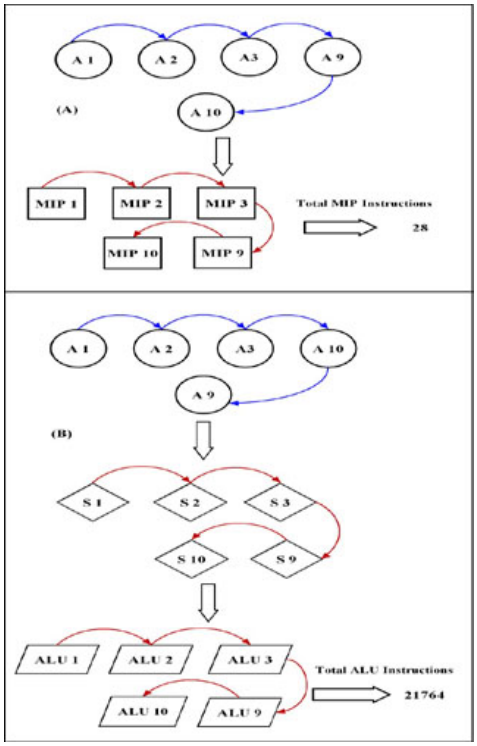

Fig. 10. Synthetic Application 3

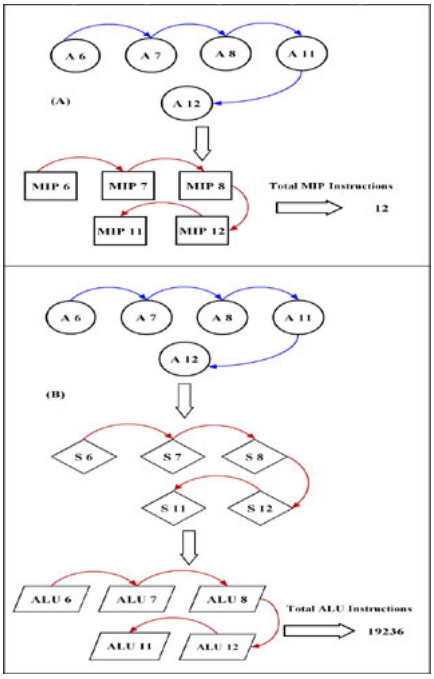

Fig. 11. Synthetic Application 5

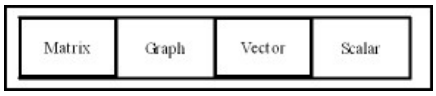

Fig. 12. Computational Complexity Vector.

IV. Simulation Results and Analysis

The integration of the high-speed memory with the processing elements of the HLF is so fine grained at the logic level, that the memory latency undergone by the processing elements is nearly annulled. The ease of mapping achieved through the abstractness of the units, which consummates into high performance is clearly portrayed in the previous media pub.

The instructions being at the algorithm level, the process of scheduling is also greatly simplified. These condensed set of instructions brings forth reduced instruction fetch cycles. The synthetic applications employed for simulation are listed in the figure.3.

These designed synthetic applications are used to illustrate the supremacy of the MIP architecture in reducing the memory fetch operations over the conventional ALU based architecture. The MIP simulator was employed to generate the equivalent MIP instructions of the synthetic applications. The tantamount of the same (ALU instructions), for the Powerpc-440 were generated using its disassembler.

With reference to graph given in fig.13 the corresponding synthetic application in fig.4 employs scalar and matrix oriented computations. The huge difference between the MIP and the ALU in instruction count is due to the matrix part of the algorithm. Whereas with regard to scalar operations, the difference is insignificant. On increasing the problem size, the variation between the two instruction counts does not show a proportional increase but rather increases multifold.

At the cluster level the problem size is very large and one can expect the difference between the number of instructions to be large enough in further decreasing the memory access. The second graph shown in fig.14 and the corresponding synthetic application(fig.7) are along the same lines. It is also observed that the type of matrix operations involved play a key role in deciding the difference between the ALU and MIP instruction count and hence the overall reduction in memory fetch operations. The matrix multiplication algorithm employed in the third synthetic application shown in the fig.9 plays a major role in reducing the overall memory fetch operations which is evident from the graph.15.

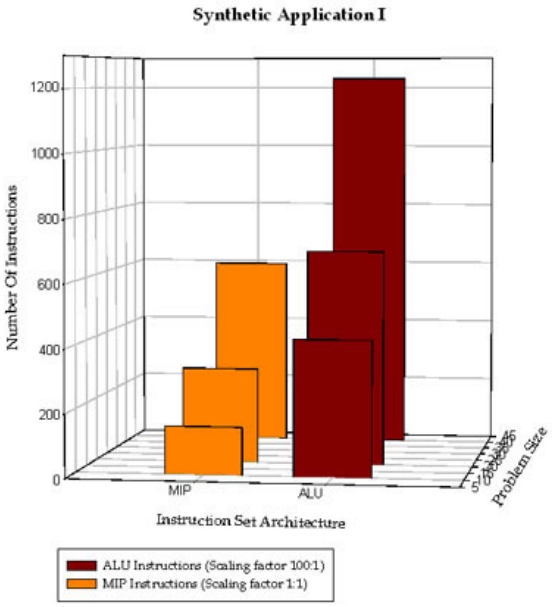

Fig. 13. Comparison between MIP and ALU instructions for different problem sizes. For ALU : 1 Unit = 100 ALU instructions
For MIP : 1 Unit = 1 MIP instruction.

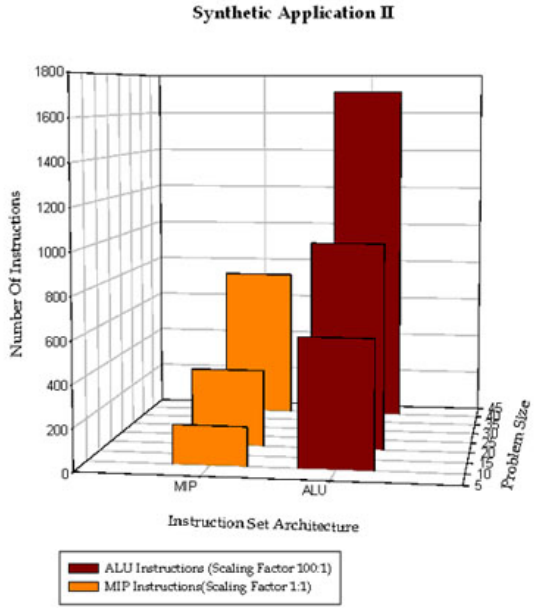

Fig. 14. Comparison between MIP and ALU instructions for different problem sizes. For ALU : 1 Unit = 100 ALU instructions
For MIP : 1 Unit = 1 MIP instruction.

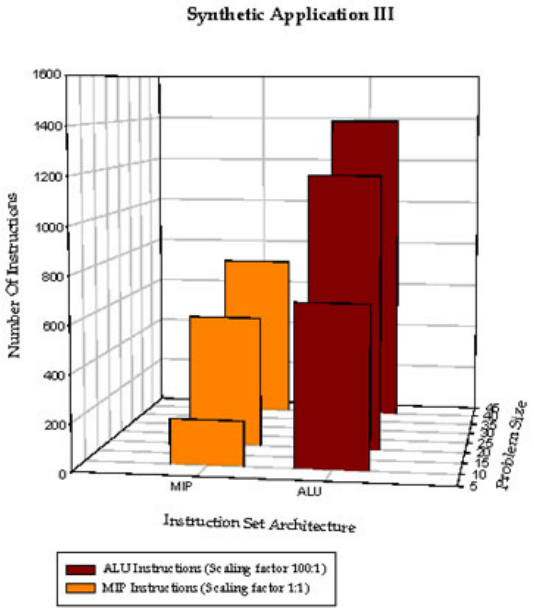

Fig. 15. Comparison between MIP and ALU instructions for different problem sizes. For ALU : 1 Unit = 100 ALU instructions
For MIP : 1 Unit = 1 MIP instruction.

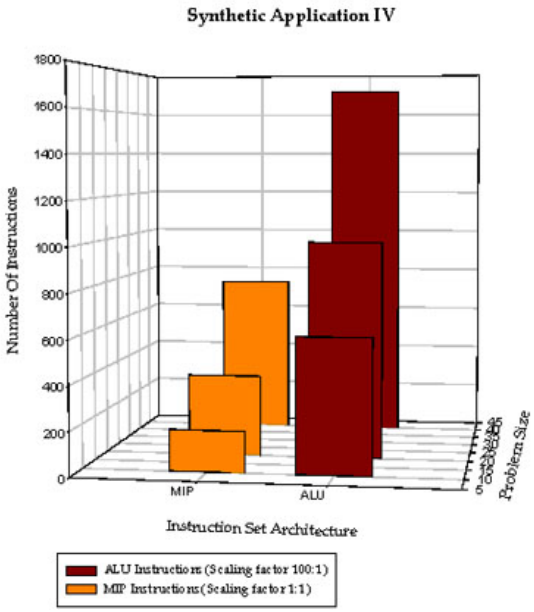

Fig. 16. Comparison between MIP and ALU instructions for different problem sizes. For ALU : 1 Unit = 100 ALU instructions
For MIP : 1 Unit = 1 MIP instruction.

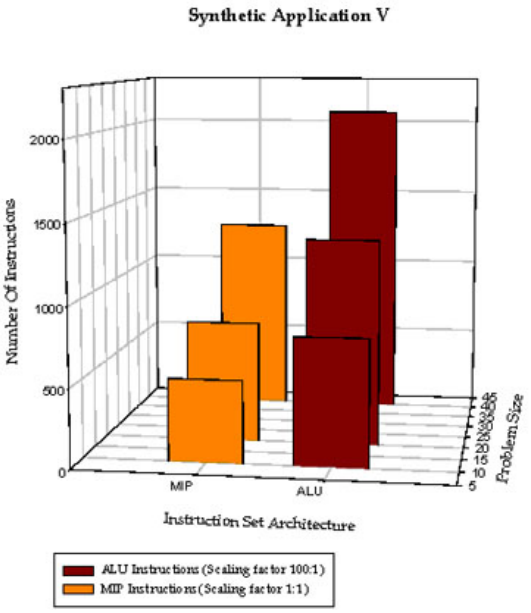

Fig. 17. Comparison between MIP and ALU instructions for different problem sizes. For ALU : 1 Unit = 100 ALU instructions
For MIP : 1 Unit = 1 MIP instruction.

Graphs depicting synthetic applications shown in fig.16 and 17 correspond to a completely different mix of algorithms. The algorithms of the synthetic application shown in fig.8, include matrix and graph theoretic operations. For this algorithm mix it is observed that, as the problem size increases the difference between the instruction counts of the ALU and MIP architectures increases significantly when compared to the other applications involving only matrix operations. The reason being the presence of the hardcore graph theoretic functional units such as intra and inter adjacency units.

With regard to the synthetic application shown in fig.10 the disparity in the instruction count for ALU and MIP architectures reduces since the algorithms involve large number of scalar operations such as sorting and square root. This is clearly evident from the graph shown in fig.17. From the simulations, it is observed that the influence of the succinct instructions of the ALISA leads to reduced memory fetch and hence higher performance. (Besides the reduced memory fetches, the drastic drop in the number of instructions simplifies the mapping of complex applications on to the MIP cluster in comparison with the conventional ALU based clusters.

V. Memory Efficient Execution Of Applications On MIP Cluster

The MIP SCOC cluster is envisaged for beyond hexaflops scale applications involving several hundreds of thousands of nodes driven by hierarchically placed multiple hosts.

The architecture of the MIP SCOC cluster has been proposed as a hybrid pyramid cluster. More details are presented in[9][10].

A single powerful general purpose host on a MIP SCOC cluster cannot run the entire system. It is an intricate task for a single host to meet the feed rate of the computing MIP-nodes and the complexity of mapping simultaneous multiple applications on to the MIP cluster. This paper does not concentrate on these issues and the genesis of these concepts are presented in [10][9].

A unique ID is generated for all algorithms for a given application. The dependency between the algorithms present within a particular application is analyzed. The typical characteristics of all algorithms are determined, based on the library requirements for those algorithms. This is done to assign a computational complexity measure for all algorithms and to construct a vector as shown in Fig.12, which characterizes the input application, thus extending support to mapping over diversified core types.

Each algorithm is mapped onto the MIP core(matrix, graph theoretic, vector)best suited to meet its performance demands providing massive parallelism.The best-fit strategy is being worked as a feasible solution for mapping applications on to the cluster. On completion of the execution of the algorithms present in the various nodes, the remaining algorithms (if any) are processed in the next fold.

The IBM PowerPc-440 was selected as the conventional ALU node to facilitate the comparison of the MIP cluster with that of the BlueGene cluster in the simulations.

VI. Conclusion

This paper highlights the memory efficiency of the Algorithm Level Instruction Set Architecture (ALISA) over the conventional ALU based ISAs. Synthetic applications analogous to real world applications have been chosen to establish the above stated fact. The node level simulations show a drastic decrease in the average memory fetch operations and when extended to the cluster, the same was reconfirmed. These applications could serve as effective benchmarks in evaluating the performance of supercomputers. The various strategies to map simultaneous multiple applications on to the MIP cluster are under investigation, the primary purpose of which is to design efficient application mixes capable of achieving a high performance accompanied with reduced memory fetches for a cluster with heterogeneous node architecture.

REFERENCES

[1] D. Burger and J.R. Goodman et al. Memory bandwidth limitations of future microprocessors. *Proceedings of 23rd Annual International Symposium Computer Architecture, ACM, pages 79-90*, August 1996.

[2] Dr. Peter M. Kogge. In pursuit of a petaflop : Overcoming the bandwidth latency wall with pim technology. 1999.

[3] Sunaga T and Peter M. Kogge et al. A processor in memory chip for massively parallel embedded applications. 1996.

[4] C.E.Kozyrakis. Vector iram. *I.E.E.E Comp. Elements Work*, June 1998.

[5] Gyan Banot et al. The blue gene/l supercomputer. June 2002.

[6] Jeff Draper and Jacqueline Chame et al. The architecture of the diva processing-in-memory chip. *International Conference on Supercomputing*, 2002.

[7] Thomas Sterling and Hans P. Zima. Gilgamesh: A multithreaded processor-in-memory architecture for petaflops computing.

[8] N. Venkateswaran, Aditya Krishnan, Niranjan Soundarajan, and Arrvindh shriraman. The mip project : Evolution of a novel supercomputing architecture. *MEDEA Workshop - PACT Conference*, September 2003.

[9] N. Venkateswaran, Aditya Krishnan, Niranjan Soundarajan, and Arrvindh shriraman. Memory in processor : A novel design paradigm for supercomputing architectures. *ACM SigArch Computer Architecture News*, June 2004.

[10] N Venkateswaran, Arrvindh Shriraman, and S. Niranjan Kumar. Memory in processorsupercomputer on a chip processor design and execution semantics for massive single chip performance. *Fifth Workshop on Massively Parallel Processing (WMPP), IPDPS*, 2005.

[11] S.LakshmiVarahan and S.K.Dhal. Analysis and design of parallel algorithms.

[12] Elie Krevat and Jose G. Castanos et al. Job scheduling for the blue gene/l system. *Euro-Par 2002*, 2002.

[13] K. Sankaralingam et al. Exploiting ilp, tlp, and dlp with the polymorphous trips architecture. *Proceedings of 30th International Symposium on Computer Architecture*.

[14] Niranjan Soundararajan. Hardware compilation for the mip s.c.o.c and hierarchically based multiple host system for the mip cluster. *WARFT Undergraduate Technical Thesis*, June 2004.