SCOC IP Cores for Custom Built Supercomputing Nodes

Venkateswaran Nagarajan*, Rajagopal Hariharan‡, Vinesh Srinivasan‡, Ram Srivatsa Kannan‡, Prashanth Thinakaran‡, Vigneshwaren Sankaran‡,Bharanidharan Vasudevan‡, Ravindhiran Mukundrajan †, Nachiappan Chidambaram Nachiappan†, Aswin Sridharan†, Karthikeyan Palavedu Saravanan†, Vignesh Adhinarayanan†and Vignesh Veppur Sankaranarayanan†
*Director, ‡WARFT Research Trainee, †Previously affiliated with WARFT
Waran Research Foundation [WARFT], India
Email: waran@warftindia.org

***Abstract*—A high performance and low power node architecture becomes crucial in the design of future generation supercomputers. In this paper, we present a generic set of cells for designing complex functional units that are capable of executing an algorithm of reasonable size. They are called Algorithm Level Functional Units (ALFUs) and a suitable VLSI design paradigm for them is proposed in this paper. We provide a comparative analysis of many core processors based on ALFUs against ALUs to show the reduced generation of control signals and lesser number of memory accesses, instruction fetches along with increased cache hit rates, resulting in better performance and power consumption. ALFUs have led to the inception of the SuperComputer On Chip (SCOC) IP core paradigm for designing high performance and low power supercomputing clusters. The proposed SCOC IP cores are compared with the existing IP cores used in supercomputing clusters to bring out the improved features of the former.**

***Keywords*-Supercomputing; heterogeneous cores; SCOC IP Cores; complex functional units; ALFUs**

I. Introduction

The future of computational sciences depends on the deliverance of exascale computing power in the foreseeable future. However, global energy crisis places a huge constraint in achieving this goal with traditional off-the-shelf processors. The onus of delivering exascale computing by overcoming the energy consumption concerns lies with computer architects. A well known solution to obtain high energy efficiency is the use of application specific custom processors. However, the design effort and the high cost overhead associated with ASICs prevent them from mainstream use. Therefore, the twin constraints – energy efficiency and infrastructure costs, force us to explore new frontiers in the design space where custom specialization co-exists with an existing architecture.

In order to exploit this young design space, a novel node architecture model [1] based on the use of large functional units and other architectural elements was proposed. In this paper, we present the design and analysis of a more generic set of cells used in building Algorithm Level Functional Units (ALFUs). These ALFUs are capable of executing a complete algorithm of reasonable size driven by a single instruction. The corresponding Algorithm Level Instruction Set Architecture (ALISA) is a superset of other instruction sets such as vector instructions, CISC and VLIW which are used in various multi-core/many core processors. The term *ALFU based design*, frequently used in this paper, refers to the design of heterogeneous many core processors using ALFUs and scalars. The ALFUs are designed for a wide variety of numeric, semi-numeric and non-numeric algorithms.

Algorithm Level Functional Units (ALFUs) are designed by hardwiring a set of scalars based on the characteristics of an algorithm. A balanced mix of these ALFUs and scalars can be used to execute applications that are computationally intensive. The use of such units provides a variety of advantages such as a reduction in overall power consumption and increased performance. A significant drop in the number of control sequences associated with each instruction, memory access, the number of instruction fetches and the overall control complexity are be observed. Additionally, the cache performance is improved.

The simultaneous execution of multiple applications without space or time sharing (SMAPP) at the supercomputing cluster level would enable cost sharing, while avoiding substantial performance sharing amongst multiple users [2]. A new class of IP cores called SuperComputer On Chip (SCOC) IP cores for low power supercomputing clusters based on ALFUs is introduced in this paper. The design of low power, yet high performance many core processors which support the simultaneous execution of multiple applications without space or time sharing (SMAPP) at the supercomputing cluster level is simplified by the use of the SCOC IP cores. These IP cores are customizable to a great extent and can be designed for any architectural set. Section V elucidates the design of SCOC IP cores for architectures based on the CUBEMACH design paradigm [3].

The use of large functional units that provide ASIC-like functionality to the cores has earlier been advocated in the works of [1] and [4]. Our paper provides the design methodology and a systematic analysis of the use of ALFUs in heterogeneous many core processors.

978-0-7695-4767-1/12 $26.00 © 2012 IEEE
DOI 10.1109/ISVLSI.2012.80

Table I
TYPES OF CELLS USED TO DESIGN ALFUS

Cell	Input	Output	Functionality	Employed In
DACSRAM_A	A_i, B_i	C_i	$C_i = A_i \oplus B_i$	Adder,max/min finder
DACSRAM_B	A_i, B_i	C_i	$C_i = A_i + B_i$	Multiplier, inner product
DACSRAM_C	A_i, B_i	C_i	$C_i = A_i B_i$	Comparator unit,sorter BFS and DFS
DAASRAM	A_i, B_i, C_i	Sum, C_i	$Sum = A_i \oplus B_i \oplus C_i$ $Carry = (A_i \oplus B_i)C_i + A_i B_i$	Multiple operand adder,KL graph unit
DACSRAM_A1	A_i, B_i	P_i, G_i	$P_i = A_i + B_i, G_i = A_i B_i$	Adder/subtractor,comparator,Matrix adder
DACSRAM_A2	A_i, B_i	A_i, B_i, C_i	$Sum = A_i \oplus B_i \oplus C_i$	Adder/subtractor,matrix multiplier
DACSRAM_B1	$G_{i,j}, G_{j+1,k}, P_{i,j}, P_{j+1,k}$	$G_{i,k}, P_{i,k}$	$P_{i,k} = P_{i,j}.P_{j+1,k}$ $G_{i,k} = (G_{ij}.P_{j+1,k}) + G_{j+1,k}$	Adder/subtractor,comparator, sorter
DACSRAM_B2	$P_{i,j}, G_{i,j}, C_i$	$C_{i,j}$	$C_i = (P_{i,j}.C_i) + G_{i,j}$	Adder/subtractor,crout unit

II. RELATED WORK

It is important not to confuse the ALFUs with accelerators. Processors relying on accelerators for higher performance seldom have binary compatibility and require a standalone module for decoupling the Instruction Set Architecture [5]. Unlike accelerators that are stand-alone units, the ALFUs are an integral part of the processor itself, thereby eliminating the issues of compatibility that the accelerators bring in.

Another interesting work that can be compared with the ALFU is the Dynamically Specialized Datapaths [6].The overheads due to switching in the DySER blocks are ones that ALFUs do not face and hence the ALFUs are expected to offer better performance. The fusion of various instructions into Macro-ops would mean that the Instruction Decode units would still have to actually decode as many instructions even though they have been fused into a single Macro-op. A single ALISA instruction triggers the execution of an ALFU. Hence, the fetch and decode complexities of the ALISA is lesser.

Lawrence Berkeley National Laboratory recently adopted the Tensilica System-On-Chip IP cores which are partially customizable, to design a supercomputing cluster for climate modeling [7]. The Xtensa IP cores are customizable with the provision of adding a single application specific block and its associated instructions. The SCOC IP cores, on the other hand provide complete customization of the design of all the ALFUs or scalars, the communication backbone and even the compiler/instruction scheduler.

III. ARCHITECTURE OF ALGORITHM LEVEL FUNCTIONAL UNITS

A set of cells capable of performing basic operations has been designed which forms the building blocks of ALFUs. The cells developed and their functionalities are shown in Table I.

ALFUs are designed for a wide class of algorithms (Numeric, Semi-Numeric and Non-Numeric). Some of the

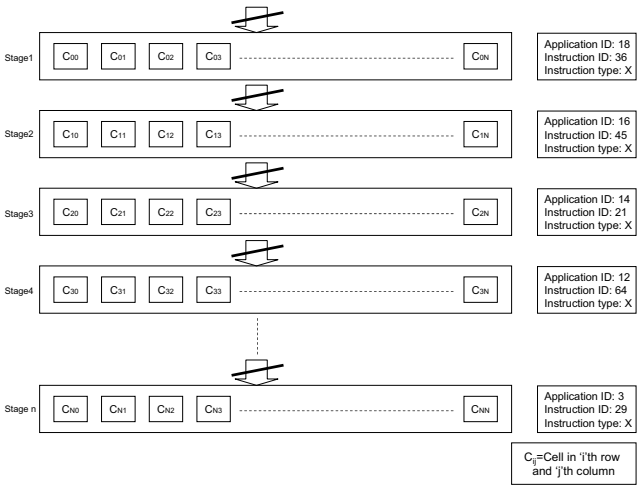

Figure 1. Cell based generic ALFU architecture

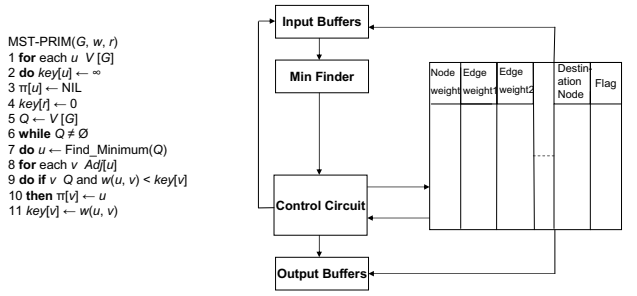

Figure 2. ALFU architecture of the Minimum Spanning Tree Algorithm

cells that have most commonly been used for designing ALFUs are shown in Table I.

A set of appropriate cells are selected from the cell library to build pipelinable stages with suitable interconnection nets to form the ALFUs and scalars. The cell based generic architecture of an ALFU is shown in Figure 1.

Understandably, the cell with maximum delay decides the delay of the particular stage. Suitable latching is provided to match the delays of the cells. The cell with maximum delay across all the stages decides the pipelining rate of the ALFU. By suitable arrangement of the cells, the pipelining delay of the ALFU can be reduced.

A. ALFU for Minimum Spanning Tree Algorithm

The ALFU designed for the Minimum Spanning Tree (MST) algorithm shown in Figure 2 is a table based architecture. The control circuit present in the ALFU generates the appropriate control signals to enable searching the table for the corresponding node, check for formation of cycles and choose the shortest edge from a selected entry. The table contains the set of node and edge weights, whose values are compared against the source node, checked for

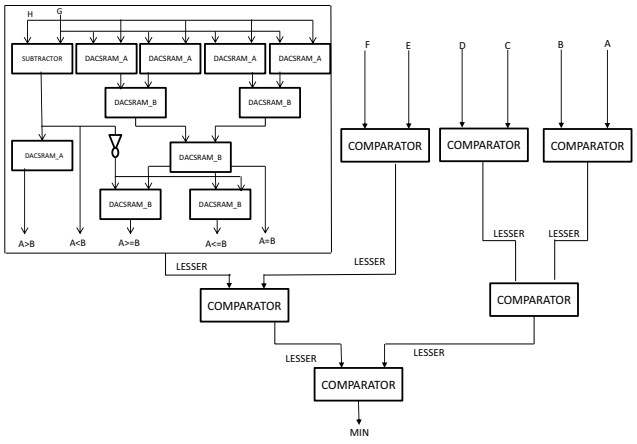

Figure 3. Cell level architecture for Minimum Operand Finder Unit

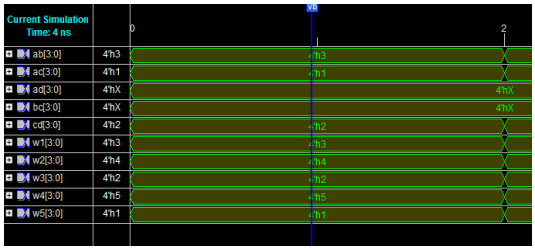

Figure 4. The working of the MST architecture was verified functionally.

formation of cycles and the shortest edge is chosen using the Minimum Operand Finder shown in Figure 3. This is iterated taking every node of the graph as source node to find the Prim's Minimum Spanning Tree. This effectively minimizes the number of control sequences associated with the corresponding instructions. The architecture was verified for a small problem size using Verilog HDL as shown in Figure 4.

The number of pipeline stages of the ALFUs implies that the pipelining depth is variable as per requirement to suit simultaneous execution of multiple instructions from multiple applications in a single ALFU. The description of other ALFU architectures can be found in [8].

IV. Algorithm Level Functional Units : Performance and Power Analysis

A. Impact of ALFUs on instruction complexity

A single ALISA instruction is inherently equivalent to several dependent and independent ALU instructions. In this sense, a single ALISA instruction is equivalent to several VLIW or vector instructions. In effect, ALISA instructions are such that a single instruction is equivalent to 10s or even 100s of scalar instructions. A single ALISA instruction computes a complete algorithm, it is a superset of all other ISAs, VLIW, vector or otherwise.

Table II
Generic set of equations for MOV and COMPUTE instructions associated with ALFUs and ALUs. Here, N is the problem size of the application and α is the problem size of the ALFU.

Algorithm	Type of Instruction	ALU	ALFU
Matrix Multiplication	MOV	N^3	$\frac{N^3}{\alpha^3}$
	COMPUTE	$N^2(N+1)$	$\frac{N^2(N+1)}{\alpha^2}$
Matrix Addition	MOV	N^2	$\frac{N^3}{\alpha^3}$
	COMPUTE	N^2	$\frac{N^2}{\alpha^2}$
Max/Min Finder	MOV	N	N
	COMPUTE	N	$\frac{N}{\alpha}$
Minimum Spanning Tree	MOV	N	N
	COMPUTE	N	$(\frac{N}{\alpha})$
Multiple Operand Adder	MOV	N	$N-1$
	COMPUTE	$\frac{N}{2}$	1
Inner Product	MOV	N	$\frac{N}{\alpha}$
	COMPUTE	$N+1$	$\frac{N}{\alpha}+1$
Odd Even Transposition Sorting	MOV	$\frac{3N}{2}(N+1)$	$\frac{3N}{2\alpha}(\frac{N}{\alpha}+1)$
	COMPUTE	$\frac{N(N-1)}{2}$	$N(\frac{N}{\alpha}-1)/2\alpha$
Kernighan Lin Graph Partitioning	MOV	$\frac{N^2+2N+16}{8}$	$\frac{\frac{N^2}{\alpha^2}+2\frac{N}{\alpha}+16}{8}$
	COMPUTE	$\frac{N^2+10N+8}{8}$	$\frac{\frac{N^2}{\alpha^2}+10\frac{N}{\alpha}+8}{8}$
Graph Traversal	MOV	$N-1$	$N/\alpha-1$
	COMPUTE	$N-1$	$N/\alpha-1$

The nature of the ALISA used in ALFU based processors implies that the number of instructions to be executed by the functional units to run a particular application is inherently lesser in comparison with ALU based processors. This would imply that the power consumed due to instruction fetch and decode in ALFU based processors is significantly lower. The circuitry used in association with instruction fetch and decode is used less often, amounting to lower power consumption. From the tabulated results in Table II, it observed that there is a drop in power consumption with respect to the instruction fetch.

The Table II, contains a generalized set of equations are derived for the number of ALISA instructions (COMPUTE) that are required to execute a particular algorithm using ALFU based and ALU based cores. Along with these equations, the number of instructions that is associated with move operations (MOV) are also estimated. Here, N represents the problem size of algorithm being executed and α represents the problem size for which the ALFU is designed.

Table III
COMPARISON OF COMPLETE SET OF CONTROL SEQUENCES OF ALFUS WITH MINIMUM NUMBER OF EXPLICIT CONTROL SEQUENCES OF ALUS

Algorithm	Problem Size	No. of control sequences	
		ALU	ALFU
Matrix Multiplication	2×2	54 (Pipelined)	38 (Pipelined)
		46 (Parallel)	32 (Parallel)
Matrix Addition	2×2	42 (Pipelined)	34 (Pipelined)
		40 (Parallel)	32 (Parallel)
Crouts	2×2	24	18
Matrix Inverse	3×3	86	64
Minimum Spanning Tree	8 Node	164	108
Max/Min Finder	8 Operands	22	17
Multiple Operand Adder	9 Operands	28	22
KL Graph Partitioning	4 Node	140	108
Inner Product	8 Operands	47	38
Sorting	8 Operands	29	22

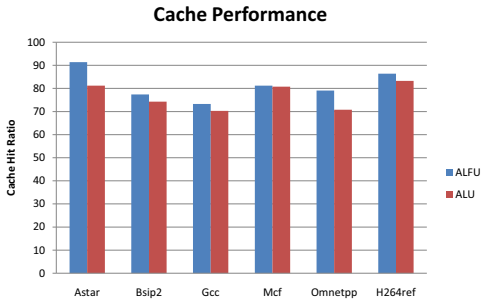

Figure 5. The cache hit ratio for each of the SPEC equivalent benchmarks that were provided as worload inputs to the WIMAC simulator [3]

B. Impact of ALFUs on control complexity

Another important aspect of the usage of ALFUs in many core processors is that the number of control sequences is considerably reduced in comparison with their ALU based counterparts. This is particularly because the ALFUs are made up of several hardwired scalar units. As a result, much of the control sequences get absorbed within the large functional unit.

By studying the nature of algorithms, the number of control sequences needed for those have been computed. The complete analysis of the control sequences associated with the ALFUs is done by estimating the control sequences associated for each operation. There are no specific benchmarks developed to estimate the number of control sequences associated with ALU instructions. So the minimum number of explicit control sequences that is associated with the execution on an algorithm in ALU based cores are considered for comparison. This was done in order to reduce the complexity of analysis and was found that the control sequences associated with the ALFU based processors was significantly lesser than the minimum set of explicit sequences that were considered themselves.

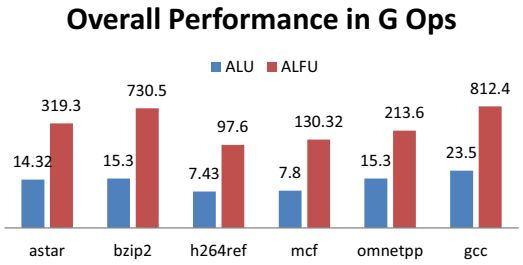

Figure 6. A comparison of the overall performance metric of ALFU based processors against their ALU based counterparts

C. Impact of ALFUs on cache hit in a heterogeneous many core processor

Cache with varying sizes (shown in Table IV) adopting the 4-way set associative mapping and a heuristic based replacement policy. As the data requirement of each ALFU is quite large because of the size of operands that each ALFU operates on is large. The increased hit ratios of ALFU based architecture over ALU is because of the existence of dependencies across instructions which get localized as a consequence of the hardwired scalar units. A set of cache replacement heuristics have been developed for the CUBEMACH design paradigm which suit the execution of multiple spplications simultaneously without space or time sharing [3]. This reduces number of conflict misses as well as capacity misses thereby showing a considerable improvement in cache hit ratios as compared to conventional ALU based heterogeneous many core processors.

D. Impact of ALFUs on the overall performance figures

The overall performance of the ALFU based cores is found to be higher than that of the ALU based cores. Figure 6 shows the simulation results of a CUBEMACH design paradigm [3] based architecture. The WIMAC simulator [3] has been used, whose workload inputs are SPEC equivalent Benchmarks. The overall performance of the ALFU based cores is understandably higher due to the reduced number of memory accesses from the use of ALFUs. The Figure 6 is not to scale.

E. Estimation of power consumption of individual ALFUs

Design of ALFU based heterogeneous cores needs to be done very meticulously, keeping in mind various constraints such as the interconnection between ALFUs, the power consumed by the ALFUs etc. A wide range of power estimation methods have been discussed in [9]. The method used by us is a generic model to estimate the dynamic power consumption of any functional unit. The development of tools that can effectively estimate the dynamic power consumption for different architectures would sufficiently simplify the task of the designer.

Table IV
CUBEMACH DESIGN PARADIGM BASED ARCHITECTURAL SPECIFICATION WHICH IS AN INPUT TO THE WIMAC SIMULATOR

		SPEC EQUIVALENT WORKLOAD					
Architectural Parameters		astar	gcc	bzip2	mcf	omnetpp	h264ref
Cores		4	4	4	4	4	4
Cache Size	L1	32kB	64kB	32kB	32kB	32kB	64kB
	L2	256kB	512kB	256kB	256kB	256kB	512kB
	L3	4MB	8MB	4MB	4MB	4MB	8MB
SubLocal Router Stages	Input	4	8	4	4	4	8
	Output	4	8	4	4	4	8
Local Router Stages	Input	8	12	8	8	8	12
	Output	4	6	4	4	4	6
Global Router Stages	Input	12	18	12	12	12	18
	Output	6	12	6	6	6	12
Instruction word Buffer Size		16kB	32kB	16kB	16kB	16kB	32kB
Network Based		16kB	32kB	16kB	16kB	16kB	32kB
Clock Frequency		800MHz					

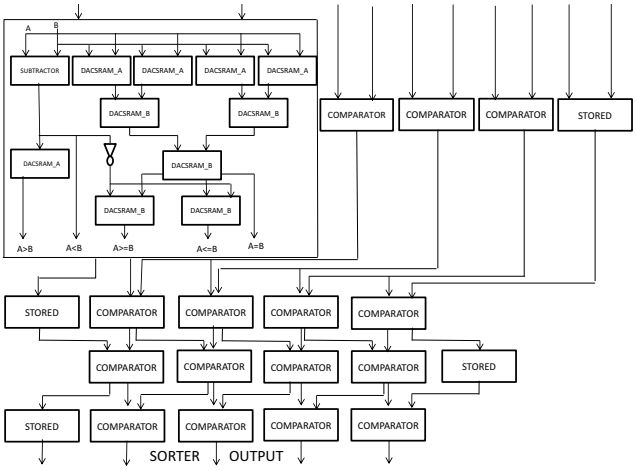

Figure 7. The architecture of the sorter ALFU has been illustrated above. The architecture shown here is an 8 element Batcher's Odd Even Algorithm based Sorter.

A probabilistic model has been developed to estimate the total activity factor of each ALFU, with the inputs to an ALFU being in any distribution of choice. The model has been evolved in a bottom up manner. As shown in Figure 1, the stages of the ALFUs are designed using the standard cells.

The results of the power analysis for a simple ALFU architecture has been shown. Figure 7 shows the architecture of a Batchers Odd Even Transposition Sorter. The architecture of the Sorter ALFU is scalable to any number of elements, but the architecture given in Figure 7 is an 8 element Sorter. Based on the model, activity factors of the cells used to make up the Sorter ALFU are obtained and the dynamic power consumption of the ALFU is estimated. The inputs to the ALFU are considered to be a set of stochastic variables based on a distribution of a particular type. The Table V shows the results of power estimation of the Sorter ALFU with the inputs to the ALFUs assumed to be normally distributed.

Table V
POWER ESTIMATION RESULTS FOR THE 8 NODE SORTER ARCHITECTURE GIVEN ABOVE.

Word-length	Activity Factor	Gate count	Number of idle gates
8	0.165367	1152	961
16	0.164885	2688	2244
32	0.164774	5760	4810
64	0.164748	11904	9942
128	0.164742	24192	3985

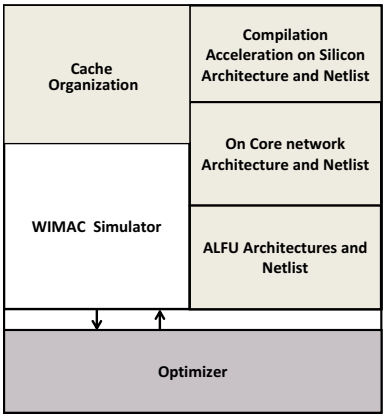

Figure 8. Structure of the proposed SCOC IP core

V. DESIGN OF SCOC IP CORES FOR HETEROGENEOUS MANY CORE PROCESSORS USING ALFUS

In the design of the ALFU based processors, the cost factor should not be a deterrent. A class of IP cores for heterogeneous many core processors can be designed using ALFUs and are called SuperComputer On Chip (SCOC) IP cores. These IP cores are a large library of scalable and customizable cores which can be designed at different levels of abstractions.

The structure of the IP core is as shown in Fig. 8 . The SCOC IP core comprises of three main elements; the architectural components of the CUBEMACH design paradigm [3] Compilation Accelerator on Silicon, On Core Network, on chip memory organization, ALFUs/scalars; WARFT India MAny Core (WIMAC) simulator; the Optimizer Engine. The SCOC IP core presented here is a plug and play module and will not require additional configuration.

The Tensilica Xtensa IP core that is used in the Green Flash supercomputing cluster, on the other hand, consists of the Base CPU, a cycle accurate simulator, the application specific datapath, the set of registers and Floating Point Unit.

The SCOC IP cores support simultaneous execution of multiple applications (SMAPP) at the supercomputing cluster level. This means that the pipeline of the ALFUs in the SCOC IP cores can contain instructions from multiple applications in different stages. SMAPP inherently cost

and hardware sharing across multiple applications run by multiple users or multiple applications run by a single user.

The Compilation Accelerator on Silicon (CAS) architecture that has been detailed in [10], is a customizable hardware code generator cum dynamic scheduler. The architecture specifications and the netlist of the CAS is an important part of the IP core. In comparison with the compiler in the Xtensa IP core used in the Green Flash supercomputer, which is a vectorizing compiler that is software based, the hardware based CAS offers a multitude of advantages. The instruction issue rate and scheduling rate is much higher due to the hardware instruction generator and dynamic scheduler.

The On Core Network (OCN) is a circuit switched network that forms the communication backbone across the many core processor. The OCN structure is based on the Multi-stage Interconnection Network (MIN) and is completely scalable and customizable in accordance with the specification.

A balanced mix of ALFUs and the scalars are employed for computation purposes. The design of ALFUs has already been elaborated in Section II. In comparison with the Green Flash supercomputing cluster, the Xtensa IP cores offer customization for only one application-specific block of the core. The SCOC IP core provides complete customization with respect to the kind of units that should be present in every core.

The aforementioned architectural components are provided as inputs to the WARFT India MAny Core Simulator (WIMAC), which is a cycle accurate simulator tuned for the CUBEMACH design paradigm. The WIMAC simulator is tightly coupled with an Optimizer Engine, based on Game Theory and Simulated Annealing that prunes the design space search of the CUBEMACH based architectures and also contributes to the core formation based on the KL graph partitioning algorithm.

VI. Conclusion

Future generation supercomputers would ideally have high performance per watt. To achieve this, processors should be designed with ASIC-like efficiency. Algorithm Level Functional Units (ALFUs), proposed in this paper, can aid in achieving such high performance, while maintaining reasonable energy efficiency. The design of these ALFUs are completely cell based and is performed by hard wiring a suitable set of scalar units based on their respective parallel algorithms. The use of ALFU improves processor efficiency by generating reduced number of control signals, memory accesses and instruction fetches, along with better cache hit rates. The power consumption of the associated instruction fetch and control circuitry is also reduced significantly. Our experimental evaluations show that high improvement in performance can be observed when ALFU based cores are used instead of ALU based cores. Further, we also show how the ALFUs can be implemented in a CUBEMACH based architecture as SCOC IP cores and their effectiveness when we simultaneously execute multiple applications without space or time sharing.

References

[1] N. Venkateswaran, D. Srinivasan, M. Manivannan, T. P. R. Sai Sagar, S. Gopalakrishnan, V. Elangovan, K. Chandrasekar, P. K. Ramesh, V. Venkatesan, A. Babu, and Sudharshan, "Future generation supercomputers i: a paradigm for node architecture," *SIGARCH Comput. Archit. News*, vol. 35, no. 5, pp. 49–60, Dec. 2007. [Online]. Available: http://doi.acm.org/10.1145/1360464.1360466

[2] N. Venkateswaran, V. Elangovan, K. Ganesan, T. Sagar, S. Aananthakrishanan, S. Ramalingam, S. Gopalakrishnan, M. Manivannan, D. Srinivasan, V. Krishnamurthy, K. Chandrasekar, V. Venkatesan, B. Subramaniam, V. Sangkar, A. Vasudevan, S. Ganapathy, S. Murali, and M. Thyagarajan, "On the concept of simultaneous execution of multiple applications on hierarchically based cluster and the silicon operating system," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, april 2008, pp. 1 –8.

[3] "Cubemach design paradigm simulator," June 2012. [Online]. Available: http://www.warftindia.org/cas/cas.pdf

[4] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," in *Proceedings of the 37th annual international symposium on Computer architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 37–47. [Online]. Available: http://doi.acm.org/10.1145/1815961.1815968

[5] N. Clark, A. Hormati, and S. Mahlke, "Veal: Virtualized execution accelerator for loops," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ser. ISCA '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 389–400. [Online]. Available: http://dx.doi.org/10.1109/ISCA.2008.33

[6] V. Govindaraju, C.-H. Ho, and K. Sankaralingam, "Dynamically specialized datapaths for energy efficient computing," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, feb. 2011, pp. 503 –514.

[7] D. Donofrio, L. Oliker, J. Shalf, M. F. Wehner, C. Rowen, J. Krueger, S. Kamil, and M. Mohiyuddin, "Energy-efficient computing for extreme-scale science," *Computer*, vol. 42, no. 11, pp. 62–71, Nov. 2009. [Online]. Available: http://dx.doi.org/10.1109/MC.2009.353

[8] "Description of alfu architectures," June 2012. [Online]. Available: http://www.warftindia.org/alfu_architecture.pdf

[9] F. N. Najm, "A survey of power estimation techniques in vlsi circuits," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 2, no. 4, pp. 446–455, Dec. 1994. [Online]. Available: http://dx.doi.org/10.1109/92.335013

[10] N. Venkateswaran, V. Srinivasan, R. S. Kannan, P. Thinakaran, R. Hariharan, B. Vasudevan, K. Saravanan, N. Nachiappan, A. Sridharan, V. Sankaran, V. Adhinarayanan, V. Sankaranarayanan, and R. Mukundrajan, in *IEEE Computer Society Annual Symposium on VLSI*, August 2012.