MULTI OBJECTIVE DESIGN SPACE EXPLORATION FOR GRID ALU PROCESSOR

Reza Asgari[1], Ali Ahmadian Ramaki[2], Reza Ebrahimi Atani[3] and Asadollah Shahbahrami [4]

[1]Department of Computer Engineering, Guilan University, Rasht, Iran
rezaasgari@msc.guilan.ac.ir
[2]Department of Computer Engineering, Guilan University, Rasht, Iran
ahmadianrali@msc.guilan.ac.ir
[3]Department of Computer Engineering, Guilan University, Rasht, Iran
rebrahimi@guilan.ac.ir
[4]Department of Computer Engineering, Guilan University, Rasht, Iran
shahbahrami@guilan.ac.ir

ABSTRACT

Todays billions of transistors allow architects to design different huge processor to increasing parallel execution of instructions, which leads to increasing complexity of architecture. Establishing a trade-off between the complexity and performance to finding optimal design is a big challenge. In this paper we are going to focus on the design space exploration the Grid ALU Processor (GAP) to finding optimal design for it. To do so we address a multi-objective optimization method based on Pareto Front. This method helps us find best configuration of GAP at maximum performance and minimum complexity.

KEYWORDS

Grid ALU Processor, Design Space Exploration, Pareto Front

1. INTRODUCTION

Nowadays most of the researches in designing processors are focused on multicore architectures. These architectures are appropriate for programs that have the parallel nature and are not useful for sequential programs [1]. For parallel execution of sequential programs we need processors that change sequence of instructions to execute Independent instructions together. One of processors that raise parallel execution of instructions is Grid ALU processor that called GAP [2]. This processor has features of superscalar and reconfigurable systems with an array of Functional Units (FUs) to execute instructions simultaneously that reorder by a configure unit. One of the challenges of this processor (and all novel architectures) is to cope with the increasing complexity of designs.

All novel processor architectures like GAP expose lots of parameters, e.g. the number of processor cores, cache sizes, or memory bandwidth. Theses parameters form a huge design space. With multiple objectives and under specific constraints, as timing behavior or the availability and affordability of hardware resources, good points consisting of a combination of parameters have to be found. It is getting very hard for the system designers to cope with such increased

DOI : 10.5121/ijesa.2012.2303

complexity. For example, with 50 parameters each of them having 8 possible values, the number of possible configuration is 2^150 [20]. This means the designer must study 2^150 possible configuration to finding best design for processor. Then we need an approach to exploring huge space of these elements.

The goal of this paper is to purpose an approach to solving Design Space Exploration (DSE) problem automatically. This approach must find best configure of elements to design optimal processor that leads us to high performance whit less complexity. Unfortunately performance and complexity are in contrast, and then we need a method to establishing a trade-off between them simultaneously. Multi-objective optimization methods do this for us, so we are using a multi-objective optimization method based on Pareto Front. Pareto Front helps us to take all objectives into consideration simultaneously and maintains the diversity of solutions with ensuring that every element of Pareto Front is a good solution [17].

The rest of this paper continued as follows: Next section describes Background of this research. Section III describes the purposed method following by an implementation and evaluation in Section IV. Section V gives an overview of related works. At the end of the paper, the conclusion is provided.

2. BACKGROUND

In the following section, we give an overview of GAP. Also we introduce multi-objective optimization problem in part B.

2.1. Grid ALU Processor

The main goal of GAP architecture is parallel execution of the program instructions that written sequentially. The GAP combines elements of superscalar processor architectures with a coarse-grained reconfigurable array of functional units [2]. GAP architecture contains a two-dimensional Functional Units (FUs). Instructions are inserted into FUs with Instruction fetch, decode, configuration units, and execute. This processor uses a branch control unit for controlling conditions and ensures correct selection of branches as well as memory access control using load and store units. In addition for faster access to memory it deploys two caches memory which are referred to as instruction cache and data cache.

Fetch, decode and configuration units simultaneously work with FUs. The array is arranged in row and columns of ALUs (see the Figure. 1 [2]). Every column in the array is assigned to a single top register in the original GAP architecture. This leads to a number of columns in the array equal to the number of physical registers. Information flow in each column is top to down [1]. Each ALU can read the output information of previous row. ALUs in each row are synchronous and information in a row cannot transfer between ALUs.

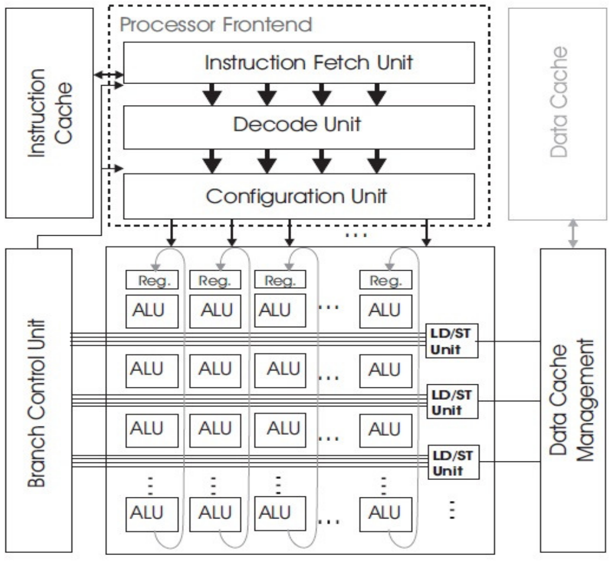

Figure 1. Architecture of the Grid ALU Processor [2]

The placement strategy in the ALUs is as follows: instructions that have no dependence on previous instructions (or instructions that related to them) put in a row and execute. Instructions that related to previous instructions can put in to next rows. When facing to a loop, later than placement of loop instructions, in the array for the next iterations of the loop not required to do fetch, decode and configuration steps and instruction can be executed immediately. Branch control unit makes sure misprediction penalty does not occur [1, 14].

The placement of instructions into the array is depicted by the following simple code fragment of pseudo machine instructions that adds 15 numbers out of subsequent memory cells followed by negating the result and depicted in (2).

Figure. 2 depicts the dependency graph of the 9 instructions, which can be recognized again at the placement of the instructions within the ALU array shown in fig. 3. The instructions 1 to 3 are placed within the first row of the array. Instruction 4 depends on R2 which is written in the first row and, therefore, it must be located in the second row. It reads the address as a result of instruction 2 and forwards the data received from memory into the column R4 which is the destination register of the load. The instructions 5 to 7 are placed in an analog way. Instruction 8 is a conditional branch that could change program flow. To sustain the hardware simplicity, conditional branches have to be placed below the lowest already arranged instruction. In this case, the branch has to be located after the third row. The last instruction must be placed after the branch in the fourth row. Hence, if the branch is taken, the GAP takes a snapshot of the register contents at the corresponding row and copies the data into the top registers (which is called “Top Regs” in Figure.3). In this case, instruction 9 is discarded. Otherwise, the sub is executed without any time loss [14].

2.2. Multi Objective Optimization

To understand, some of the most important concepts are listed below:

Multi-objective optimization is a process that usually can make two or more parameters optimal which conflict with each other simultaneously. In most cases where multi optimizations have to be performed there is not a single solution that simultaneously minimizes/maximizes each

objective. Indeed multi-objective optimization would ensure that with changes to one parameter, changes from other parameters do not cause any negative influence on the final result [21].
A general multi-objective problem can be mathematically depicted in equation (1).

$$\min/\max \vec{y} = \vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_m(\vec{x})] \qquad (1)$$
$$\text{Subject to: } \vec{x} = x_1, x_2, \ldots, x_n \in X$$
$$\vec{y} = y_1, y_2, \ldots, y_m \in Y$$

Where $\vec{x}$ is called the decision vector, X is the parameter pace, y is the objective vector and $\vec{Y}$ is the objective space $y_k = f_k(x)$ where k = 1, 2, 3, …, m.

One of the most popular methods that used to solving multi-objective optimization called Pareto Front. Pareto optimality is a concept that formalizes the trade-off between a given set of mutually contradicting objectives. A solution is Pareto optimal when it is not possible to improve one objective without deteriorating at least one of the other. A set of Pareto optimal solutions constitute the Pareto front [21].

3. Proposed Approach

To find an optimal design between possible designs we will use multi-objective optimization based on Pareto Front. The factors that inspect for optimization operation are Clock cycle Per Instruction (CPI) and design complexity. In the section A we talk about calculation of parameters used in optimization process and in section B we explain Pareto Front and how to make use of this method for our approach.

```
1.      move  R1,#15
2.      move  R2,#addr
3.      move  R3,#0
        loop:
4.        load  R4,[R2]
5.        add   R3,R3,R4
6.        add   R2,R2,#4
7.        sub   R1,R1,#1
8.        jnz  R1,loop  ; (end of loop ?)
9.        sub R1,R3,R1
                                        (2)
```

3.1. Calculation of Parameters

CPI is a division of CPU clock cycles to count numbers of executed instructions [18]. For calculation of clock cycles, we use a simulator program. We write the simulator in java programming language based on descriptions about GAP that represented in [1, 2, 14, 15, 16]. This simulator would be described in section IV.

Parameters that considered for definition of design are: 1) Number of FUs Rows={4, 5, 6,…, 32}, 2) Number of FUs Columns={4, 5, 6,…, 31}, 3) Cache Size={4k, 8k, 16k} and 4) Load/Store Units={Number of Rows}.

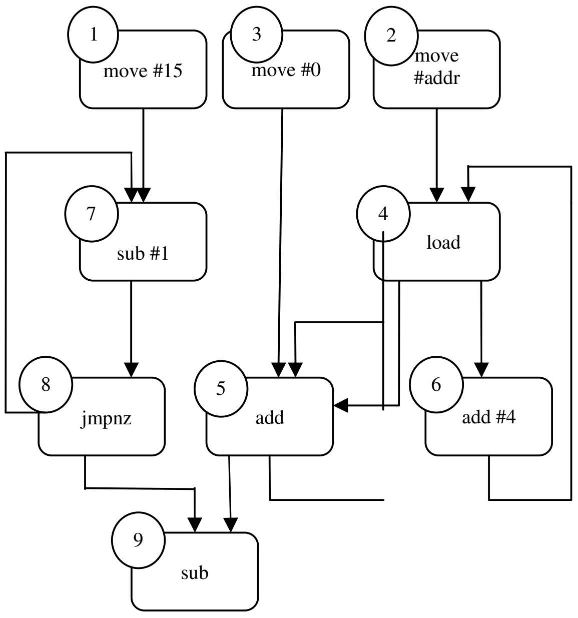

Figure 2. Dependency graph of the example instructions

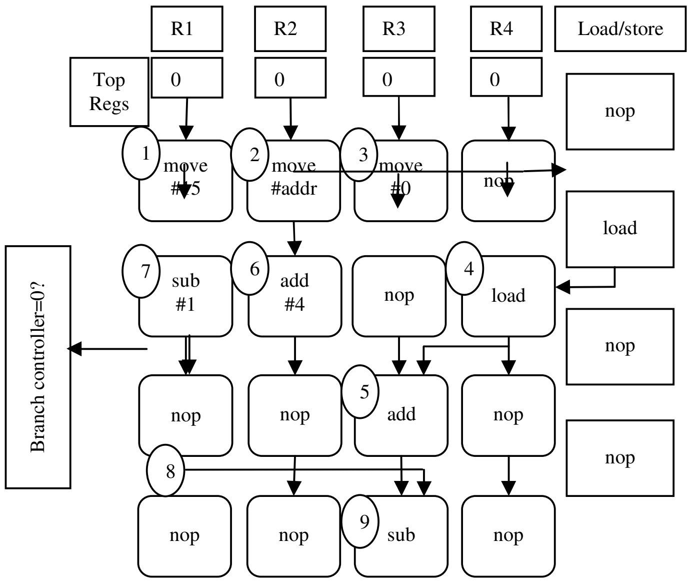

Figure 3. Placement of the example instruction

Combination of these parameters with different values constructs a set of all possible designs. After constructing the desired set, with execution of a program on simulator, the value of CPI calculated for every state. In this case we test JPEG program because of its potential loop-based structure [3], then GAP can execute it in ideal mode.

For calculation of design complexity we use (3) [2, 16]:

$$\text{Complexity} = C_{\text{ALUs}} + C_{\text{LSUs}} + C_{\text{cache}} \qquad (3)$$

In (3) C_X presents complexity of element X ∈ {ALUs, Load/Store, Cache}. The values of C_{ALUs}, C_{LSUs}, C_{cache} are respectively calculates from (4), (5) and (6). The variables that used here depicted in Table I.

$$C_{ALU} = (C_c*h_r + C_r*C_c*h_{ALU}) + (C_r*C_c*C_l*h_l) \quad (4)$$
$$C_{LSUs} = (C_r*h_{LSU}) + (C_r*C_l*h_l) \quad (5)$$
$$C_{cache} = \text{Cache Area}*h_c \quad (6)$$

Table 1. Values of Using Constants in Calculation of Complexity Design

Variable	Value	Comments
C_c	{4, 5, 6,…, 31}	Number of Columns
C_r	{4, 5, 6,…, 32}	Number of Rows
C_l	1	Number of Configuration Layers for FUs array, in this case we used just one layer
h_r	0.02	complexity of Register
h_{ALU}	1	FU comprising an ALU
h_l	0.02	Configuration Layer for a FU
h_{LSU}	3.50	complexity of Load/store Units
h_c	3 (1/mm^2)	complexity of Cache

For calculate h_r, h_{ALU}, h_l, h_{LSU}, and h_c we used ratio of area size of each components to area size of source component. We consider ALU as source component, and to calculate area size of components we used results of [22] (in this paper authors calculate area and delay for different hardware elements like cache, integer ALU and etc.). For exapmle h_r = (average area for register per bit / average area for integer ALU per bit) = (4.06e+4 / 2.41e+6) = 0.017 ≈ 0.02.

3.2. Approach

To find optimal design we will use Pareto Front. According to the definition of Pareto Front [17]:

We may define optimization criterion in a multi-objective problem on the basis of dominance concept as follows: for two decision vectors X1 and X2, X1 dominates X2 ($X1 \prec X2$) if and only if two conditions are satisfied [19]: 1) X1 is not worse than X2 in all objectives. 2) X1 is strictly better than X2, at least in one objective.

The decision vector of $X \in X_f$ is also called non-dominated in relation to $A \subseteq X_f$ if and only if $\exists$ a $\in A : X \prec a$ holds. X is Pareto optimum if and only if it is non-dominant in relation to X_f. Thus vertex X is regarded as optimum in perspective of being able to improve none of its objectives regardless of making other objective value worse [17].

To reduce the number of comparisons for finding Pareto optimal (with attention to (7) and (8)), instead of comparing each decision vector with total, first we divide decision vectors to sets that contain two members. In each set, decision vectors are compared and dominating vector goes to the next step, if none of the two vectors can dominate each other, two vectors go to the next step. In the next step, the place of decision vectors are randomly changed for comparison and again vectors are compared to each other. The process continues to have finally a set of decision vectors that non-dominate each other. Such set is the answer. This process depicted in (9). Pseudo code in (10) describes Dominate function that used in (9).

$$X_1 < X_2 \Leftrightarrow \forall i \in \{1, 2,\ldots, n\} : f_i(x_1) \leq f_i(x_2) \wedge \exists i \in \{1, 2,\ldots, n\} : f_i(x_1) < f_i(x_2) \quad (7)$$

$$\begin{aligned} X_1 < X_2 &\Leftrightarrow \forall i \in \{1, 2,\ldots, n\} : f_i(x_1) \leq f_i(x_2) \\ &\wedge \exists i \in \{1, 2,\ldots, n\} : f_i(x_1) < f_i(x_2) \\ X_2 < X_3 &\Leftrightarrow \forall i \in \{1, 2,\ldots, n\} : f_i(x_2) \leq f_i(x_3) \\ &\wedge \exists i \in \{1, 2,\ldots, n\} : f_i(x_2) < f_i(x_3) \\ f_i(x_1) \leq f_i(x_2) \leq f_i(x_3) &\Rightarrow f_i(x_1) \leq f_i(x_3) \\ f_i(x_1) < f_i(x_2) < f_i(x_3) &\Rightarrow f_i(x_1) < f_i(x_3) \Rightarrow x_1 < x_3 \end{aligned} \quad (8)$$

```
// Algorithm to finding optimal set of designs
Input: design rules
Output: optimal design of GAP

S = all possible design that return from GAP simulator
    While (elements of S are not changed) {
        for (i = 0 ; i< S.length ; i+=2)
            Add Dominate (S_i,S_i+1) to temp
        S = temp
        S = RandomExchange(S)
          //randomly exchange elements of array
    }
Return S
                                                   (9)
```

```
// Dominate (S_i,S_i+1)
Input: 2 decision vector a and b
Output: dominated vector

Int BetterInAnyObjective = 0;
for (i = 1; i < noObjectives && a[i] <= b[i]; i++)
    if (a[i] < b[i])
        BetterInAnyObjective = 1
if((i >= noObjectives) && (BetterInAnyObjective>0))
        return a
return a and b  //if a cannot dominate b
                                                   (10)
```

4. IMPLEMENTATION AND EVALUATION

To studying results of GAP architecture, we design a simulator in java programming language in which fetch, decode and configure units are synchronized by Wait() and Signal() functions. Executions of instructions placed in rows are also synchronous. Load and store units work as follows: all units can read simultaneously but if one needs to write must wait until all units that come with read operations finish their work. The task is managed by semaphores.

The simulator gives design properties and an assembly source code of a program (in this case we test JPEG program) as input, and then the simulator first configures the GAP architecture by design properties (such as number of rows and columns, cache size, design layer, number of registers for each column (one or two) and etc) and then runs the input program. After running the program, the simulator returns CPI of GAP for this program and architecture complexity as output. The steps of our algorithm show the process as follows (except step 1, other steps are doing automatically by algorithm):

1) Design GAP properties ranges by User
2) Create set of S = {all possible designs}
3) Create set of CC by sending each of $s_i \in S$ to simulator and calculate CPI and complexity for s_i
4) Send $c_i \in CC$ to the algorithm that explained in section IV, part B.
5) Return set of optimal design form the algorithm to the user.

To assess the performance of GAP architecture, we run JPEG program on Simple Scalar processor simulator and GAP simulator (the parameters of each simulator give in Table II), for each simulator we calculate the Instructions per Cycle (IPC). Simulation results show that on increasing the number of rows and columns in GAP, the value of IPC improves. For instance for a state with 32 columns when the number of rows is greater than 6, the value of IPC for GAP is better than Simple Scalar's IPC. The results of JPEG program execution on simulator of Simple Scalar processor and GAP processor simulator (with 31 columns and 4, 8, 16 and 32 rows) are showed in Figure. 4.

With attention to (3) and (4), the conclusion is that complexity and the number of columns and rows have a linear relationship that means on increasing the number of rows and columns, complexity increases. Figure. 5 depicts the correlation. The bigger is number of columns, the more is number of instructions which can be executed simultaneously.

This results in reduction of CPI. On the other hand, growing number of rows causes the loops of program put in rows and in second iteration of loop execution there is no need to fetch, decode and configure operations consequently it reduces as well the number of clock cycle needed for execution of the program. On the other hand with respect to figure. 6, it can be seen on increasing the complexity, the value of CPI decreases. Since the value of CPI however, is less the performance improves, hence we can conclude that increasing number of rows and columns improves performance of system. Now the question is that how much escalating number of rows and columns may ameliorate the performance?

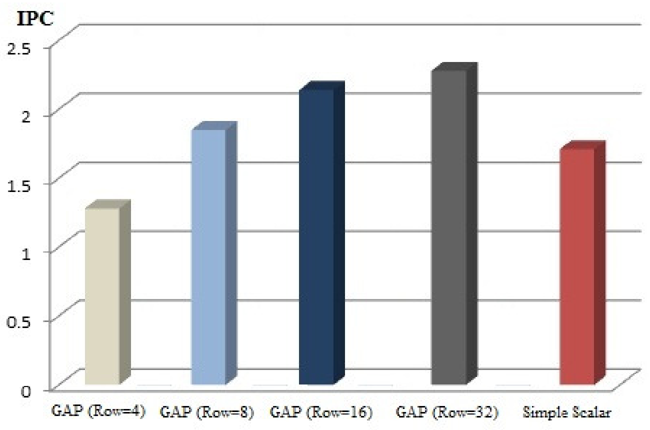

Figure 4. Comparing Simple Scalar and GAP(with 31 columns and 4, 8, 16 and 32 rows) whit IPC for the JPEG.

Table 2. Parameters of Simulators

Parameters	Simple Scalar	GAP
Integer ALUs	4	31 per row
Multipliers	1	1 per row
Brach prediction	bimod	bimod
Branch target buffer	512*4	none
Load / Store Unit	8	One per row

The results of simulation illustrate by increasing the number of columns to 31, the value of CPI then reduce but for values greater than 31 the rate of changes in CPI value is ignorable. By examining results (when the number of columns is at 31) we observe if the number of rows increases, CPI decreases at the same time. The CPI reduction goes on until the number of rows rises to 32. From this point onward, however, number of rows surges up, the rate of changes in CPI remains negligible.

As mentioned earlier, we are going to find a set of solutions that CPI and complexity remain optimal. By exploration of more than 3000 samples, the algorithm selects a set of optimal design consisting nearly 60 solutions as optimal design set. Figure. 7 delineates the correlation of complexity and CPI.

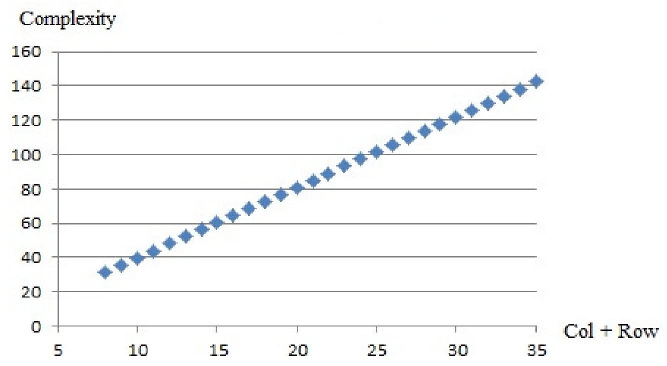

Figure 5. Correlation of complexity and number of Row (whit constant column number)

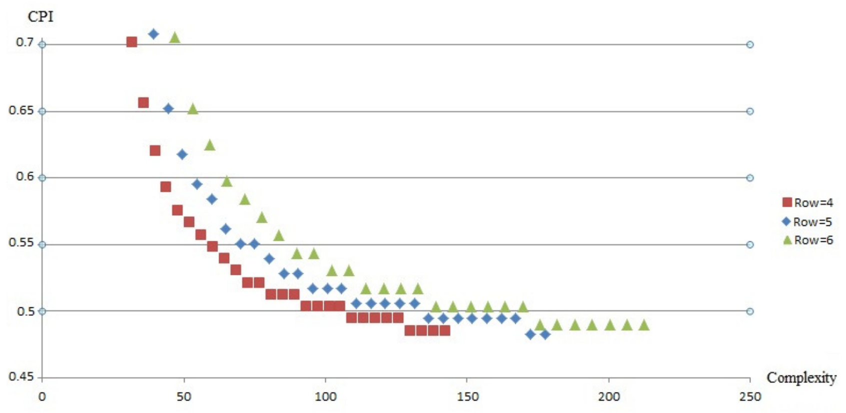

Figure 6. Correlation of complexity and CPI for Row = 4, 5, 6

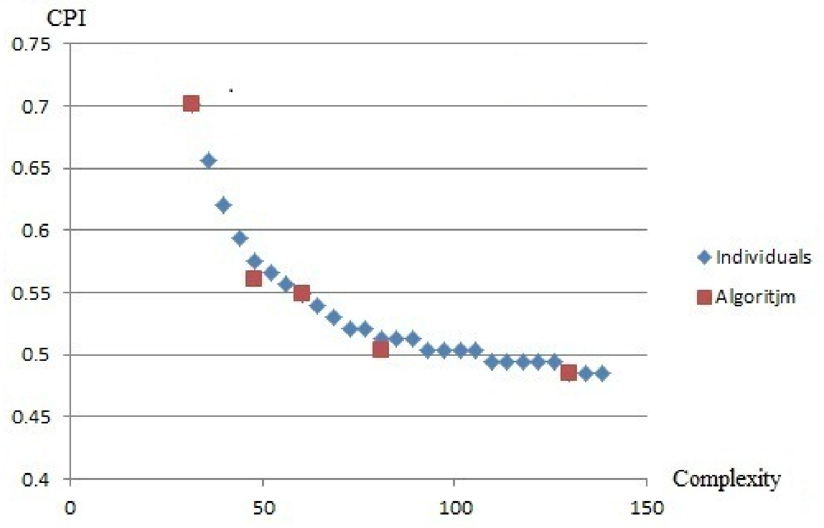

Figure 7. Correlation of complexity and CPI for some of the optimal designs and initial designs

5. Related Work

Several research works try to generalize design space exploration and provide frameworks for architecture exploration, although most of them do not bring design space exploration beyond parameter exploration. The Magellan framework for multicore exploration fuses power/area measurement and statistical exploration techniques, and exposes a larger range of multicore parameters [4]. ReSP enables the exploration of architectures composed of transaction-level SystemC components, as well as hardware/software trade-offs [5]. FADSE (is a Framework for Automatic Design Space Exploration) attempts to include as much ADSE algorithms as possible, to offer connectors to existing computer simulators (M5, GEMS, Multi2sim, NS3, etc) and to run in parallel the design space exploration process [20].

Most applications of reconfigurable architectures always require an additional processor as master. This processor has two tasks to fulfill: first, it has to provide configuration data to and, second, it has to take care about the activities of the reconfigurable system. Additionally, the presence and the architecture of the reconfigurable part must be taken into account at the software development phase. Example architectures with reconfigurable parts as coprocessor are the MOLEN [6], the GARP [7], and the MorphoSys [8] architectures. The XTENSA chip from Tensilica, the CHIMAERA [9] project, and the PRISC processor [10] use reconfigurable functional, units within the processor pipeline; see [11] for a more detailed overview.

ArchExplorer [12] is another automatic design space exploration (ADSE) tool. Researchers can upload their own hardware simulators for different architecture components in order to weigh the performance against other designs. The uploaded simulator has to be made compatible with the interface provided by the ArchExplorer developers. Having the simulator been uploaded, it automatically becomes part of the design space exploration tool and it is simulated and compared with other similar microarchitectures. The simulator continuously runs on the ArchExplorer servers. Any user can check any time what is the best configuration found so far for its simulated microarchitecture and also compare it with other implementations.

8. CONCLUSIONS

This paper tried to solving design space exploration problem. For achievement this goal we work on Grid ALU processor and study different configuration of this processor. To finding optimal configuration of GAP, we use multi-objective optimization based on Pareto Front that optimizes CPI and complexity of design.

The algorithm depicted in this paper automatically generates new designs of GAP and sends them to GAP simulator, after execution of JPEG program, the simulator returns CPI and complexity of each design to the algorithm. In last step, the algorithm finds optimal design based on Pareto Front. Section IV showed only 2% of possible designs were selected as optimal set and then we get to single out one design from this set proportional to our necessity.

In future work we are going to optimize the GAP architecture by making changes on ALUs configuration. Also we attempt to design a graphical user interface for our method to make it easier to use.

REFERENCES

[1] H. Calborean and L. Vintan, “An Automatic Design Space Exploration Framework for Multicore Architecture Optimizations”, 9th Roedunet International Conference, pp. 202-207, 2010.

[2] R. Jahr, T. Ungerer, H. Calborean, and L. Vintan, “Automatic Multi-Objective Optimization of Parameters for Hardware and Code Optimizations”, International Conference on High Performance Computing and Simulation, pp. 308 – 316, 2011.

[3] A. Shahbahrami, B. Juurlink, S. Vassiliadis, “Improving the Memory Behavoir of Vertical Filtering in the Discrete Wavelet Transform”, 3rd Conference on Computing frontiers, 2006.

[4] S. Kang and R. Kumar, ‘‘Magellan: A Search and Machine Learning-Based Framework for Fast Multicore Design Space Exploration and Optimization’’, Design, Automation, and Test in Europe, ACM Press, pp. 1432-1437, 2008.

[5] G. Beltrame, L. Fossati, and D. Sciuto, ‘‘Resp: A Nonintrusive Transaction-Level Reflective MPSOC Simulation Platform for Design Space Exploration’’, IEEE Trans. CAD of Integrated Circuits and Systems, vol. 28, no. 12, pp. 1857-1869, 2009.

[6] S. Vassiliadis, S. Wong, and S. D. Cotofana, "The Molen ρµucoded Processor", 11th International Conference on Field-Programmable Logic and Applications, Springer-Verlag Lecture Notes in Computer Science, Vol. 2147, pp. 275–285, 2001.

[7] J. Hauser and J. Wawrzynek. "Garp: A MIPS Processor with a Reconfigurable Coprocessor", IEEE Symposium on Field Programmable Custom Computing Machines, 1997.

[8] M. H. Lee, H. Singh, G. Lu, N. Bagherzadeh, F. J. Kurdahi, E. M. Filho, and V. C. Alves, "Design and Implementation of the Morphosys Reconfigurable Computing Processor", Journal of VLSI Signal Processing Systems 24(2–3), pp. 147-164, 2000.

[9] S. Hauck, T. Fry, M. Hosler, and J. Kao. "The Chimaera Reconfigurable Functional Unit", IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 87–96, 1997.

[10] R. Razdan and M. D. Smith. "A High-Performance Microarchitecture with Hardware-Programmable Functional Units", 27th Annual International Symposium on Microarchitecture, pp. 275–285, 1994.

[11] K. Compton and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software", ACM Computing Surveys, pp. 171–210, 2000.

[12] V. Desmet, S. Girbal, O. Temam, and B. France, "Archexplorer. org: Joint Compiler/Hardware Exploration for Fair Comparison of Architectures", Interact workshop at HPCA, 2009.

[13] B. Shehan, R. Jahr, S. Uhrig, and T. Ungerer, "Reconfigurable Grid Alu Processor: Optimization and Design Space Exploration", 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools, pp. 71-79, 2010.

[14] S. Uhrig, B. Shehan, R Jahr, and Theo Ungerer, "A Two-dimensional Superscalar Processor Architecture", Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, pp. 608-611, 2009.

[15] R. Nagarajan, K. Sankaralingam, D. Burger, and S. W. Keckler, "A Design Space Evaluation of Grid Processor Architectures", 34th ACM/IEEE International Symposium on Microarchitecture, MICRO-34 Proceedings, pp. 40-51, 2001.

[16] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, "A Comprehensive Memory Modeling Tool and its Application to the Design and Analysis of Future Memory Hierarchies", 35th Annual International Symposium on Computer Architecture, IEEE, pp. 51–62, 2008.

[17] R. Asgari, M.G. Moghaddam, S. S. Sahraei, and R. A. Ebrahimi, "An Approach to Ontologies Alignment Based Upon Pareto Front", 5th International Computer and Instructional Technologies Symposium, 2011.

[18] D. Patterson and J. Hennessy , "Computer organization and Design: the Hardware/Software Interface Textbook", 3rd ed, 2004.

[19] K. Deb, "Evolutionary Algorithms for Multi-Criterion Optimization in Engineering Design", Kanpur Genetic Algorithms Laboratory, 1999.

[20] H. Calborean and L. Vintan, "Framework for Automatic Design Space Exploration of Computer Systems", University of Sibiu, ISSN 1583-7149, 2011.

[21] C. Coello, G. Lamont, and D.A. Veldhuizen, "Evolutionary Algorithms for Solving Multi-Objective Problems(Second Edition)", Springer, 2007.

[22] S. Gupta, S. W. Keckler, and D. Burger, "Technology Independent Area and Delay Estimates for Microprocessor Building Blocks", University of Texas at Austin, Tech. Rep, 2000.

Reza Asgari

Reza Asgari was born in Iran, Ghazvin. He recieved his BSc degree from university of Guilan, Iran in 2011. He is now MSc student at university of Guilan, Iran. His research interests in operating system and database security.

Ali Ahmadian Ramaki

Ali Ahmadian Ramaki was born in Iran on August 10, 1989. He recieved his BSc degree from university of Guilan, Iran in 2011. He is now MSc student at university of Guilan, Iran. His research interests in computer security, network security and intelligent intrusion detection.

Reza Ebrahimi Atani

Reza Ebrahimi Atani received his BSc degree from university of Guilan, Rasht, Iran in 2002. He also recieved MSc and PhD degrees all from Iran University of Science and Technology, Tehran, Iran in 2004 and 2010 respectively. Currently, he is the faculty member and assistant professor at faculty of engineering, University of Guilan. His research interests in cryptography, computer security, network security, information hiding and VLSI design.

Asadollah Shahbahrami

Asadollah Shahbahrami received his BSc degree from Iran University of Science & Technology, Tehran, Iran in 1993. He also recieved MSc from Shiraz University, Shiraz, Iran in 1996 and PhD degree from Delft University of Technology, Delft, Netherland in 2008. Currently, he is the faculty member and assistant professor at faculty of engineering, University of Guilan. His research interests in advanced computer architecture, SIMD programming, digital image and video processing and reconfigurable architecture.