Multidimensional Pareto Frontiers Intersection Determination and Processor Optimization Case Study

Jakub Podivinsky, Ondrej Cekan, Martin Krcma, Radek Burget, Tomas Hruska, Zdenek Kotasek
Brno University of Technology, Faculty of Information Technology,
Centre of Excellence IT4Innovations
Bozetechova 2, 612 66 Brno, Czech Republic
Tel.: +420 54114-{1361, 1361, 1360, 1320, 1239, 1223}
Email: {ipodivinsky, icekan, ikrcma, burgetr, hruska, kotasek}@fit.vutbr.cz

***Abstract*—Almost all today's electronic devices are equipped with a processor. Different applications require and depend on different properties of the processor. For example, the fast growing field of Internet of Things depends on a long operation time of the devices when powered with batteries. Using general purpose processors has proved ineffective which led to a growing usage of Application-Specific Instruction-Set processors (ASIPs) which can be optimized for specific applications using different modifications of their properties (such as the number of registers, cache sizes, instruction set modifications, etc.). A suitable processor configuration can be hand-picked by a designer or by an automatic tool. The goal of this paper is to introduce a tool able to find a suitable processor configuration for multiple applications by constructing a compromise Pareto-optimal frontier of processor configurations. Experiments are based on a parametrizable RISC-V processor.**

Keywords*—*Pareto frontier, processor optimization, ASIP.

I. Introduction

The area of embedded systems is growing in use and importance in daily life due to the continuously increasing popularity of the Internet of Things (IoT) [1]. These systems are usually based on a processor that offers sufficient computing performance as well as high flexibility which allows the systems to be updated easily. One possibility is to use a general purpose processor (GPP). These processors generally offer high performance and flexibility allowing them to run a number of different applications. Manufacturing costs of these processors are quite low, but the processor power consumption and its physical dimensions have to be considered as well. Usually, embedded systems are designed for a long operation time when powered by batteries. Therefore, the low power consumption of the processor has to be considered [2] and the need of decreasing the GPPs' power consumption led to increasing the development and popularity of Application Specific Instruction-set Processors (ASIPs).

The advantage of ASIPs is that they are designed for a specific application with different parameters such as power consumption, performance and area (physical dimensions) taken into account. For example, an ASIP optimized for wireless communication was introduced in [3]. The ASIP optimization is usually based on changing the key parameters of the processor such as the number of registers, caches, number of slots, computation units configuration or instruction set modifications. The physical area of the processor can be decreased by removing unnecessary instructions while the performance can be increased by introducing new specific and optimized instructions. In this paper, we call the settings of all the mentioned parameters the *processor configuration*.

Processors can be modeled using different architecture description languages (ADLs) or hardware description languages (HDLs) [4]. ADLs [5] provide a more abstract way of the processor description There exist various tools for automatic processor generation based on its abstract description. The *Synopsys ASIP Designer* [6], [7] is a set of tools for ASIP design from a user-defined architecture to RTL description. The Cadence company provides configurable Xtensa LX7 Processor and its development tools [8]. In our experimental work, we use the Codasip Studio provided by the Codasip company [9]. Codasip Studio is a development tool for processor design; the designer is able to describe the architecture of a processor and its instruction set and then, to generate a corresponding toolchain (compiler, simulator, etc.) Codasip also offers predefined configurable processor cores (eg. RISC-V [10] based Codix-Bk processor [11]).

In our previous work [12], we have proposed framework for searching the most suitable configurations of processor parameters and the compiler flags for a selected application. The framework is based on the processor simulation and the evaluation of the obtained results. A Pareto frontier of the possible solutions is the main output of the proposed system which can be used by a designer for making the final decision. In some cases, there may be a requirement to find a processor configuration that is optimized for multiple different applications or for an entire application class. The parameters of such a configuration then represent a trade-off between the requirements of the individual applications. As the results of the individual optimization processes come in a form of discovered Pareto frontiers, we face a problem of joining multiple Pareto frontiers that have been discovered for the individual applications into a single set of suitable solutions. We call this problem a Multidimensional Pareto frontier intersection and the introduction to this problem as well as our proposed solutions are the main topics of this paper.

The paper is organized as follows. Section II describes the Pareto optimization task and presents the detailed description of the Pareto frontier intersection. In section III, solutions of the presented problem are proposed. Experimental results are presented in section IV. Finally, section V presents our conclusions and future research directions.

II. The Pareto Frontier Intersection

Multi-criteria optimization [13], [14] is a process of finding a vector $\vec{x} = (x_1, ..., x_n) \in X$ of decision variables (n is the number of decision variables) that exists in state space X

of a selected task, and which minimizes the vector $\vec{F}(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), ...f_N(\vec{x}))$ of *objective functions* where N is the number of objective functions, $\forall i \in \{1, ..., N\}, f_i(\vec{x}) \in \mathbb{R}$. The state space size is usually limited with several constraints $g_j(\vec{x}) \geq 0, j = 1, 2, ...M$. For most real problems, the optimization objectives are often contradictory and finding a single solution is usually not possible. Therefore, we may prefer to search for a set of suitable solutions that fit the objectives in an acceptable level and are not dominated by a specific objective at the same time. One of the used approaches is the Pareto optimization.

The underlying concept of the *Pareto optimization* is the Pareto dominance [14]. A solution $\vec{u} = (u_1, u_2, ...u_N) \in X$ is *Pareto dominant* over a solution $\vec{v} = (v_1, v_2, ...v_N) \in X$ when $\forall i \in \{1, ..., N\}, f_i(\vec{u}) \leq f_i(\vec{v}) \wedge \exists j \in \{1, ..., N\}, f_j(\vec{u}) < f_j(\vec{v})$. We say that a solution $\vec{u}$ is Pareto dominant over a solution $\vec{v}$ when $\vec{F}(\vec{u})$ is better than $\vec{F}(\vec{v})$ with respect to all the objectives $f_i(\vec{v})$ and there exists at least one objective $f_j(\vec{v})$ for which $\vec{F}(\vec{u})$ is sharply better than $\vec{F}(\vec{v})$. A solution is considered to be better than another one with respect to an objective, when the value of the corresponding objective function $f_i(\vec{v})$ (further called the *objective value*) is lower than the same objective value of the other solution. A solution $\vec{u} \in X$ is *Pareto optimal* when there is no other solution $\vec{v} \in X$ that is Pareto dominant over u. The set of all Pareto optimal solutions is called a *Pareto frontier*. In the processor optimization context, the objectives may represent, for example, a processor speed and power consumption. Then, the solutions represent different processor configurations which are interesting for potential usage are included in the Pareto frontier.

It may be often needed to construct a compromise Pareto frontier of solutions meeting the same objectives for different problem settings. In case of a processor, it may not be practical to fabricate different massively optimized processors for different application in the same domain. The goal then may be to find a processor adequately optimized for multiple applications in the same domain (for example a processor usable in both a digital watch and a hearing aid) when we are willing to tolerate some trade-offs in the objectives. We call the problem of joining a set of different Pareto frontiers into one frontier as *Pareto Frontier Intersection*. The result of this process is a Pareto frontier representing compromise solutions between different applications selected to be as optimal as possible. The goal of this paper is to propose algorithms for the construction of such a joint Pareto frontier. We consider a joint Pareto frontier to be a set composed of solutions with overall best objective values through all the problem settings. In order to explain the principle of the joint Pareto frontier, we introduce a formal description:

1) Let A be a set of problem settings.
2) Let C be a set of all possible solutions.
3) Let F be a set of all the objective functions.
4) All possible solutions meet all the objectives for all parameters to some objective value o:
$\forall a \in A \wedge \forall f \in F \wedge \forall c \in C : \exists o_{\{a,f\}}(c) \in \mathbb{R}$
5) $G : \mathbb{R}^m \to \mathbb{R}$ is a function that reduces all objective values to a single value through the space of $A \times F$:
$\forall c \in C \wedge \forall n_1...n_m \in A \times F \wedge m = |A \times F| : \exists G(o_{n_1}(c)..o_{n_m}(c)) = g_c; g_c \in \mathbb{R}$
6) O is an ordered sequence of reduced objective values of all solutions:
$O = \{g_{c_1}...g_{c_n} | g_{c_m} \leq g_{c_{m+1}}; g_{c_m} = G(c_m); n = |C|\}$
7) First x values of the O sequence create the joint Pareto frontier.

The description defines a set O that contains all the solutions ordered by their joint objective values. Considering the principles presented in the above sections, a solution is better when its objective value is lower. By joining all the objective values for all the parameters and using the resulting values to order the solutions, we obtain the generally better solutions in the beginning of the O sequence. A chosen number of solutions from the front of the sequence then can be considered as the joined Pareto frontier.

This problem has not been addressed very often in the literature, especially not in a form of an automatic algorithm as we propose. For example, paper [15] introduces a problem of constructing a Pareto frontier optimizing hydraulic actuation systems. It deals with a control error dependency on energy consumption and using a genetic algorithm, it finds a Pareto frontier. It considers two settings of this problem (a servo valve and a servo pump concept) which leads to two Pareto frontiers that are later joined manually and compared. The solution is not then automated. The Pareto frontier intersection principle has been also used in [16] to optimize competing concept alternatives in the area of turbine engine performance. In this article, the Pareto frontier intersection is considered differently than in our article. It is a process of joining different solutions considering the same objectives but with only one problem setting, while we deal with a number of different problem settings.

III. Proposed Solution for Pareto Frontier Intersection

We have identified two ways to solve the described problem. The first one is based on the average value calculation following the formal description. The second one extends the number of dimensions of the state space.

A. Direct Application of the Proposed Formal Description

This algorithm follows the formal description of the joint Pareto frontier in the Section II. This method computes the Pareto frontier after all the objective values have been joined for all the problem settings. All the objectives are then joined by the function G and then, the Pareto optimal solutions are found based on the new objective values. The vector F of objective functions $F_I(\vec{x}), F_{II}(\vec{x}), ..., F_n(\vec{x})$, which have different values for every setting, are joined using a vector G (Equation 2) of functions $g_I(\vec{x}), g_{II}(\vec{x}), ..., g_n(\vec{x})$:

$$F(\vec{x}) = G(F_I(\vec{x}), F_{II}(\vec{x}), ..., F_n(\vec{x})) \tag{1}$$

$$G(\vec{x}) = (g_1(\vec{x}), g_2(\vec{x}), ..., g_N(\vec{x})) \tag{2}$$

where $$g_z(\vec{x}) = \frac{f_{z\_I}(\vec{x}) + f_{z\_II}(\vec{x}) + ... + f_{z\_n}(\vec{x})}{n} \tag{3}$$

while the g functions determine a way of joining the objective values. These functions may use a weighted average, median or arithmetic average which is used in our experiments as shown in (3).

In case of the processor optimization, the objective functions are the parameters such as the processor computing power, power consumption or application memory consumption. The processor area although remains the same regardless of the considered application.

B. Expanding the Number of Dimensions

This algorithm does not change the objectives in any way. It creates a new state space with a higher number of dimensions by adding the same number of dimensions to every problem setting as the original state space had. Let us consider that the numbers of dimensions of the original state spaces corresponds to the number of objective functions and the number of state spaces corresponds to the number of different problem settings considered. For each problem setting, there is a state space of solutions with N dimensions and there were n settings. The new state space (and the new Pareto frontier constructed in that space) then has $N * n$ dimensions. The new objective function $F(x)$ will then be the following:

$$\begin{aligned}\vec{F}(\vec{x}) = (&f_{1\_I}(\vec{x}), f_{2\_I}(\vec{x}), ..., f_{N\_I}(\vec{x}),\\ &f_{1\_II}(\vec{x}), f_{2\_II}(\vec{x}), ..., f_{N\_II}(\vec{x}),\\ &...\\ &f_{1\_n}(\vec{x}), f_{2\_n}(\vec{x}), ..., f_{N\_n}(\vec{x}))\end{aligned} \tag{4}$$

The new Pareto frontier then includes all the problem settings and all the objective functions. The particular objective functions may be considered individually when searching for the best solution. On the other hand, joining a high number of problem settings may lead to a state space with an inconvenient number of dimensions making the results evaluation difficult. It may also happen that the majority of solution in the new space will become a part of the new Pareto frontier.

In case of the processor optimization and joining Pareto frontiers of different applications as the problem settings, the new space will not have $n*N$ dimension but only $n*(N-1)+1$ due to the fact that one of the objectives, the processor area which is the same for all of the applications, is shared by all the original spaces.

IV. Experimental Evaluation

We evaluated the proposed approaches using our platform. We have found the frontiers for several applications which we have divided into application classes according to their specific instructions utilization:

1) Integer addition – decoding a VOIP codec G.722.1 (*decode*), anisotropic diffusion image filtering (aniso_diff), decompressing a ZIP algorithm (*zip*), Dhrystone integer benchmark (*dhry*) and faces recognition (*faces*).
2) Division instructions – factorization of big integers (*factor*) and knapsack problem solver (*knapsack*).
3) Multiply instructions – matrix product of two matrices (*matrix_prod*), sorting a matrix (*matrix_sort*) and transposition of a matrix (*matrix_transpose*).
4) Unrelated – a group of unrelated applications contains decipher data using the AES 128 cipher (*aes*), *decode* and *matrix_prod*.

As the test case, we used the Codix Berkelium [11] processor which is a RISC-V processor implemented by Codasip [9]. We have selected seven parameters of the Codix Berkelium processor that may be changed. These parameters are governed by the user specification (User-Level ISA Specification) of the RISC-V processor [10]. The parameters (listed in Table I) represent the total of 252 hardware configurations of the processor which have to be taken into account during the optimization process. Moreover, there are a lot of standard flags of the used LLVM compiler [17] that can be set. We have chosen only a small subset of compiler flags (shown in Table II) that are frequently used for compiling applications. Therefore, the total number is 9072 configurations.

TABLE I: The changeable parameters of the Codix Berkelium.

EXTENSION_E	true, false
EXTENSION_M	true, false
EXTENSION_C	true, false
ENABLE_ICACHE	true, false
ICACHE_LINE_SIZE	16, 32, 64, 128
ICACHE_SIZE	4, 8, 16, 32, 64
ENABLE_PARALLEL_MUL	true, false

TABLE II: The subset of flags for the LLVM compiler.

-o0, -o1, -o2, -o3, -os, -ofast	optimization level
-ffunction-sections	functions in its own sections
-fdata-sections	data in its own sections
-funroll-loops	unroll loops whose number of iterations can be determined
-fno-inline-functions	disable the inlining of functions
-ftrapv	checks and traps the overflow for signed arithmetic operations

We use four metrics during the evaluation of the processor configurations as the objective functions: 1) *the number of the processor cycles* used when running the application (metric of performance), 2) *overall number of the application instructions* – metric of memory consumption, 3) *overall power consumption* estimate of the application execution and the 4) *area* estimation of the processor synthetized RTL design. The last metric remains the same for all the applications evaluated on the same processor configuration, which is suitable for our algorithm of Expanding the number of dimensions. The number of configurations on the frontier for the particular applications is shown in the Table III.

TABLE III: The size of the individual Pareto frontiers.

Application	# of configurations [-]	# of conf. [%]
decode	11	0.12%
aniso_diff	29	0.32%
zip	19	0.21%
dhry	16	0.18%
faces	20	0.22%
factor	14	0.15%
knapsack	18	0.20%
matrix_product	10	0.11%
matrig_sort	10	0.11%
matrix_transpose	9	0.10%
aes	4	0.04%

Using the presented algorithms, we joined the constructed frontiers in all the application groups. Table IV contains the number of configurations on the joined Pareto frontier for all the presented algorithms in absolute values as well as percentages of the state space.

TABLE IV: The size of the merged Pareto frontiers.

	Data pre-proc. (average)		Dimensions expanding	
	s [%]	[%]	[-]	[%]
Integer addition	26	0.27%	174	1.92%
Division instruction	26	0.27%	53	0.58%
Multiply instruction	10	0.11%	16	0.18%
Unrelated	12	0.13%	49	0.54%

The evaluation result is represented by a numeric rating which represents the distance of the merged frontier from the original one. To compute the rating, we can use a modified Onion peeling algorithm. For each evaluated application, we remove the layers of solutions (Pareto frontiers) from the original state space until the set of the removed solutions contains all the solutions that were part of the joined Pareto frontier. The number of removed layers then serves as the resulting rating. The lower the rating is, the closer the joined Pareto frontier is to the original frontier. The rating process for a single application is illustrated in Algorithm 1.

Algorithm 1: Iterative evaluation algorithm.

```
Data: The application $A$ against which the evaluation is
      performed
      A set $M$ of configurations on the Pareto frontier
      merged for the selected applications
Result: Rating $R$ of the set $M$ for the application $A$
$i$ := 0;
set $P := \varnothing$;
while $P \cap M \neq M$ do
    calculate the Pareto frontier;
    move all the configurations on the Pareto frontier to
     the set $P$;
    $i$++;
end
$R := i$;
```

The ratings of the joined Pareto frontiers for the individual applications are listed in Table V. The table contains all the ratings for all the selected application groups and all the presented algorithms. There are two interesting results in the table: The *Unrelated* application group inhomogeneity is obvious in the results for all the algorithms causing a significant variance of the joined Pareto frontier ratings for the individual applications. This is however an expected outcome for this application group. A similar variance can be found in the *Integer addition* application group results where the *aniso_diff* and the *faces* applications show significant difference. After a closer examination, we found out that the complexity of the *aniso_diff* and *faces* applications is the cause of this anomaly. Table V also contains average ratings of the joined Pareto frontiers for the whole application groups. These average ranks shows, that *data preprocessing algorithm* is better than *dimensions expanding algorithm* for all application groups.

TABLE V: Comparison of merged Pareto frontier ratings.

		Data pre-proc. (average)	Dimensions exp.
Integer addition	decode	70	92
	aniso_diff	90	143
	zip	12	25
	dhry	67	89
	faces	61	133
	Average	60	96
Division instructions	factor	14	65
	knapsack	9	28
	Average	12	47
Multiply instructions	matrix_p	1	6
	matrig_s	3	7
	matrix_t	6	6
	Average	3	6
Unrelated	aes	24	128
	decode	19	27
	matrix_p	3	7
	Average	15	54

V. Conclusion and Future Work

In this paper, we followed our previous research focused on finding an optimal configuration of an application specific instruction set processor for a selected application. This task involves finding a Pareto frontier of configurations of the selected processor worth a further evaluation for a single selected application. It shows up that it is economically inconvenient to deploy different processors for different applications but its is convenient to deploy a processor suitable for an entire group of applications. For this reason, we continued our research to develop a method of constructing a compromise joined Pareto frontier for an application group. This paper focuses on the algorithms capable of joining multiple Pareto frontiers related to the individual applications to a single one for the whole group. We have presented two algorithms – the algorithm based on expanding the state space dimensions and the algorithm based on data pre-processing (objectives) before the construction of the joined Pareto frontier.

During our research and experiments with the proposed algorithms, we have discovered multiple possible modificationss. We will explore these possibilities in detail in our future research in order to increase our algorithms efficiency.

Acknowledgment

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II); project IT4Innovations excellence in science - LQ1602.

References

[1] F. Wortmann and K. Flüchter, "Internet of things," *Business & Information Systems Engineering*, vol. 57, no. 3, pp. 221–224, 2015.

[2] Y. Pu, C. Shi, G. Samson, D. Park, K. Easton, R. Beraha, A. Newham, M. Lin, V. Rangan, K. Chatha *et al.*, "A 9-mm 2 Ultra-Low-Power Highly Integrated 28-nm CMOS SoC for Internet of Things," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 3, pp. 936–948, 2018.

[3] S. Shahabuddin, O. Silvén, and M. Juntti, "Programmable ASIPs for Multimode MIMO Transceiver," *Journal of Signal Processing Systems*, vol. 90, no. 10, pp. 1369–1381, 2018.

[4] P. Mishra and N. Dutt, *Processor description languages*. Morgan Kaufmann, 2011, vol. 1.

[5] P. Mishra and N. Dutt, "Architecture description languages for programmable embedded systems," *System-on-chip: next generation electronics*, p. 187, 2006.

[6] *ASIP Designer Application-Specific Processor Design Made Easy*, Synopsys, 3 2015.

[7] *DesignWare Processor IP Portfolio*, Synopsys, 5 2017.

[8] *Xtensa LX7 Processor*, Cadence, 9 2016.

[9] Codasip. (2019) Processors for the connected world. [Online]. Available: http://www.codasip.com

[10] A. Waterman and K. Asanovic, "The RISC-V instruction set manual," *volume I: User-level ISA, version 2.2, EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-54*, 2017.

[11] Codasip. (2019) Risc-v processors. [Online]. Available: https://www.codasip.com/risc-v-processors/

[12] J. Podivinsky, O. Cekan, M. Krcma, R. Burget, T. Hruska, and Z. Kotasek, "A Framework for Optimizing a Processor to Selected Application," in *East-West Design & Test Symposium (EWDTS), 2018 IEEE*. IEEE, 2018, pp. 564–574.

[13] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 992–1007, 2006.

[14] P. Ngatchou, A. Zarei, and A. El-Sharkawi, "Pareto Multi Objective Optimization," in *Intelligent systems application to power systems, 2005. Proceedings of the 13th international conference on*. IEEE, 2005, pp. 84–91.

[15] J. Andersson, "Applications of a multi-objective genetic algorithm to engineering design problems," in *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 2003, pp. 737–751.

[16] D. Rousis, "A pareto frontier intersection-based approach for efficient multiobjective optimization of competing concept alternatives," Ph.D. dissertation, Georgia Institute of Technology, 2011.

[17] LLVM Developer Group. (2007) The LLVM Compiler Infrastructure. https://llvm.org/.