Iterative Algorithm for Multidimensional Pareto Frontiers Intersection Determination

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*—A processor forms the basis of almost most of today's electronic devices. In embedded systems, the emphasis is put not only on high performance but also on the small size and low power consumption. Application-specific instruction set processors present a solution that may be optimized for specific applications by different modifications of their parameters where the trade-offs among the parameters may be represented by a Pareto frontier. In this paper, we propose a novel method of Pareto frontier merging to allow the optimization of a processor for a whole set of applications rather than a single one. We provide an experimental evaluation of the method on a model of a RISC-V processor and we show that the proposed method provides better approximation of the source Pareto frontiers than the state-of-the-art methods.**

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

I. Introduction

The area of Internet of Things (IoT) [1] is growing and the used electronic devices are usually based on a processor that offers sufficient computing performance as well as high flexibility. One possibility is to use a general purpose processor (GPP). 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). These processors are optimized for a particular purpose and offer high power efficiency as well as high computing performance in the specific applications. The advantage of ASIPs is that they are designed for a specific application with different parameters such as power consumption, performance and the chip area taken into account. The ASIP optimization is usually based on changing the key parameters of the processor such as the number of registers, caches, number of computation units, or on the instruction set modifications. 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) [3]. ADLs provide a more abstract way of the processor description (i.e. the designer does not have to pay much attention to hardware details). There exist various tools for automatic processor generation based on its abstract description. As an example, we can mention several of them. The *Synopsys ASIP Designer* [4] 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 [5]. In our experimental work, we use *Codasip Studio* provided by the Codasip company [6]. 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 [7] based Codix Berkelium processor [8]). It is possible to generate various processor configurations and test their usability for a selected application.

978-1-7281-3427-7/20/31.00/31.00

In our previous work [9], we have proposed a framework for searching the most suitable configurations of processor parameters and compiler flags for a selected application. The framework is based on a simulation of the processor and the evaluation of the obtained results. A Pareto frontier of the possible solutions is the main output 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 among 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 two proposed solutions were the main topics of [10]. In this paper, we propose a novel solution of this problem that provides significantly better results in approximation of the source Pareto frontiers as demonstrated in section IV.

The paper is organized as follows. Section II briefly describes the Pareto optimization task and Pareto frontier intersection. In section III, a new solution of the presented problem is proposed. Experimental evaluation of the proposed method is presented in section IV. Finally, section V presents our conclusions and future research directions.

II. The Pareto Frontier Optimization

Multi-criteria optimization [11], [12] 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 [12]. 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. This is illustrated in Fig. 1. There are two charts representing Pareto frontiers for application A (orange) and application B (red). As we can see on the numbered squares representing the individual solutions, the frontiers are disjoint. The green frontiers in both graphs then represent a joint frontier of solutions for both applications that meet the objectives the best. The goal of this paper is to propose algorithms for the construction of such a joint Pareto frontier. In order to explain the principle of the joint Pareto frontier, we introduce a formal description in [10].

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 [13] introduces a problem of constructing a Pareto frontier optimizing hydraulic actuation systems. It considers two settings of the 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 authors of [14] introduce an analysis of a generalized Pareto frontier, which focuses on novel norm balancing algorithms based on cost-reward formulations on multiple access channels (MACs) and the

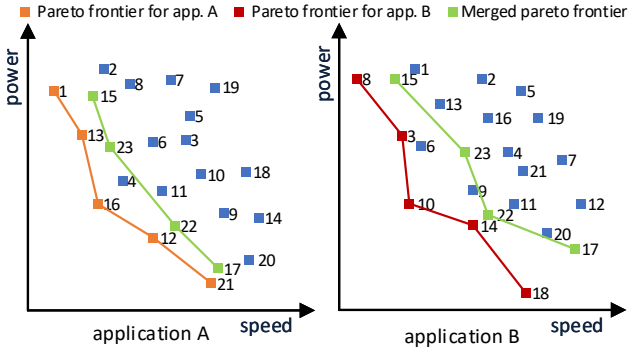

Fig. 1: The example of Pareto frontier merging.

results are extended to decode-and-forward (DF) relay systems. The Pareto frontier intersection principle has been also used in [15] to optimize competing concept alternatives in the area of turbine engine performance. However, the Pareto frontier intersection is considered differently than in our research. It is a process of joining different solutions considering the same objectives but with only a single problem setting.

III. Solution for Pareto Frontier Intersection

We have identified two ways to solve the *Pareto Frontier Intersection* problem in previous paper [10]. In this section a new *Onion Peeling* solution is proposed which is based on an iterative recalculation of the Pareto frontier. The previously developed methods published in [10] that we use for comparison are briefly introduced in sections III-B and III-C.

A. An Iterative Algorithm – Onion Peeling

In this section, we present an algorithm which we named after the process of an onion peeling. As the peeling of an onion is done by removing its layers one after another, the algorithm peels the Pareto frontiers one after another in order to obtain a set of solutions that creates a joint Pareto frontier. The algorithm finds the Pareto frontiers for all the combinations of the problem settings and objectives (the space $A \times F$) and then, iteratively moves the discovered Pareto frontiers into a set P and computes new Pareto frontiers until the set P of potential solutions reaches a predefined size s. The principle is illustrated in Fig. 2 where the color layers are the Pareto frontiers that are successively constructed and moved into P. The size of P is then a total number of solutions for all the Pareto frontiers.

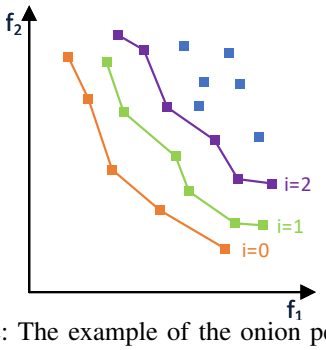

Fig. 2: The example of the onion peeling.

The algorithm is formally described in Algorithm 1. The individual Pareto frontiers are constructed and moved to the P_i sets of potential solutions iteratively for all settings of the problem. After a desired number of solutions in P_i sets was

obtained, the resulting set of solutions is constructed as the intersection of all the P_i sets. The output set M contains the solutions that were present in all the P_i sets of potential solutions. Then, M is a new Pareto frontier and although the contained solutions are not Pareto optimal, they are as close to the optimum as possible.

Algorithm 1: Iterative algorithm – onion peeling

```
   Data: Set A of optimization problem settings
         Size s of partial sets P_x
   Result: Set M of configurations representing the
           merged Pareto frontier
1  i := 0;
2  forall variants of the problem settings A do
3  |  set P_i := ∅;
4  |  while sizeof(P_i) < s do
5  |  |  calculate Pareto frontier;
6  |  |  move all configurations on Pareto frontier to set
   |  |    P_i;
7  |  end
8  |  i++;
9  end
10 M := ⋂_{j=0}^{i} P_j
```

The proposed algorithm may be modified at line 4 as the peeling process does not have to run until all the s solutions are found for the current P_i set. Instead, a fixed number of iterations may be used that corresponds to a fixed number of peeled layers. If the value of s (the size of the P_i sets) is high, constructing the intersection of the P_i sets after the whole peeling process may potentially lead to sub-optimal solutions to be moved to the M set due to uneven distribution of the solutions on the Pareto frontiers in the original sets. Therefore, the choice of s is important for the overall efficiency of the algorithm. It must not be too high to potentially produce sub-optimal solutions and not too small to produce smaller number of final solutions than desired. Another approach is to construct the intersection iteratively after every peeling step until M reaches the desired size. However, this approach may lead to higher time complexity of the algorithm.

B. Algorithm based on the Data Pre-processing

This method computes the Pareto frontier after all the objective values have been joined for all the problem settings. All the objectives are joined using a vector $G(\vec{x}) = (g_1(\vec{x}), g_2(\vec{x}), ..., g_N(\vec{x}))$ of functions $g_I(\vec{x}), g_{II}(\vec{x}), ..., g_n(\vec{x})$ and then, the Pareto optimal solutions are found based on the new objective values. The g functions determine a way of joining the objective values. This functions may use a weighted average, median or arithmetic average which we have used in our experiments.

C. Expanding the Number of Dimensions

This algorithm differs from the previous ones in changing the number of dimensions of the Pareto frontiers. 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. 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.

IV. Experimental Evaluation

Presented algorithms were evaluated using our platform for searching the Pareto frontier of processor configurations presented in [10] together with implementation details. Obtained results were compared with the previously presented results. 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 RISC-V processor Codix Berkelium [8] implemented by Codasip [6]. According to its specification [7], seven parameters of the Codix Berkelium processor have been selected that may be changed (EXTENSION_E, EXTENSION_M, EXTENSION_C, ENABLE_ICACHE, ICACHE_LINE_SIZE, ICACHE_SIZE, ENABLE_PARALLEL_MUL). The parameters (described in detail in [10]) represent the total of 252 hardware configurations. Additionally, we have chosen a small subset of compiler flags (-o0, -o1, -o2, -o3, -os, -ofast, -ffunction-sections, -fdata-sections, -funroll-loops, -fno-inline-functions, -ftrapv) that are frequently used. Therefore, the total number is 9072 configurations. Codasip Studio allow us to measure four metrics during the evaluation as the objective functions: 1) *the number of the processor cycles* – 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. For all the applications divided into the above mentioned groups, we evaluated the whole state space and constructed the Pareto frontiers for all the applications. The size of the frontier for the individual applications is shown in [10].

Using the presented algorithms, we joined the constructed frontiers for all the application groups. The iterative algorithm (onion peeling) requires the size s of the P_x set (Algoritmh 1) to be set in advance. During the experiments, we found out that this value cannot be constant for all the application groups as it may lead to an empty frontier for some applications. The used values of s, which were found experimentally, are listed in Table I in the s column as a percentage ratio of all configurations (size of the state space). Table I also contains the number of configurations on the joined Pareto frontier for all the presented algorithms in absolute values (the [-] columns) as well as percentages of the state space.

We have evaluated the proposed algorithms by ranking all the sets of solutions taken from the joined frontiers and comparing them with the original frontiers. The evaluation

TABLE I: Number of configurations on merged Pareto frontier obtained by proposed algorithms.

	Iterative alg. (onion peeling)			Data pre-proc. (average)		Dimensions expanding	
	s [%]	[-]	[%]	[-]	[%]	[-]	[%]
Addition	10	22	0.24%	26	0.27%	174	1.92%
Division	1	28	0.31%	26	0.27%	53	0.58%
Multiply	1	432	4.76%	10	0.11%	16	0.18%
Unrelated	2	24	0.26%	12	0.13%	49	0.54%

result is represented by a numeric *rating* which represents the distance of the merged frontier from the original one. 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 rating. The lower the rating is the closer the joined Pareto frontier is to the original frontier. The details of the rating process are also proposed in [10].

Table II contains all the ratings for all the selected application groups and the mentioned algorithms. The first column shows the ratings of the newly proposed iterative (onion peeling) algorithm; the remaining columns correspond to the previously published algorithms. As we may notice, the onion peeling algorithm achieves lower ratings than the remaining methods for all applications which means that the resulting Pareto frontiers better approximates the source Pareto frontiers.

TABLE II: Comparison of ratings for application groups (lower rating means better approximation of the orig. Pareto frontier).

		Iterative alg. (onion peeling)	Data pre-proc. (average)	Dimensions expanding
Integer addition	decode	15	70	92
	aniso_diff	20	90	143
	zip	7	12	25
	dhry	6	67	89
	faces	24	61	133
	Average	14	60	96
Division instructions	factor	1	14	65
	knapsack	4	9	28
	Average	3	12	47
Multiply instructions	matrix_p	1	1	6
	matrig_s	1	3	7
	matrix_t	1	6	6
	Average	1	3	6
Unrelated	aes	5	24	128
	decode	8	19	27
	matrix_p	1	3	7
	Average	5	15	54

The highlighted *Average* rows in Table II contain the average ratings of the joined Pareto frontiers for the whole application groups. The ratings show that the *iterative algorithm (onion peeling)* provided the best approximation for all four application groups. In context of the processor optimization task, this means that the resulting Pareto frontier for the whole group contains the processor configurations that are closer to the optimal configurations for the individual applications.

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 and on a method of constructing a compromise joined Pareto frontier for an application group. This paper focuses on a new algorithm for constructing a joined Pareto frontier that would represent a whole group of specified applications from multiple Pareto frontiers related to the individual applications. We have presented a new *iterative algorithm (onion peeling)* which processes a set of Pareto frontiers of the processor configurations for the individual applications. This algorithm was compared with the previously presented algorithms and was proven to provide the best results as we demonstrate on the ranking results of the obtained frontiers for the individual applications within a joined application group.

We will explore possible improvements of the presented algorithm in detail in our future research in order to increase the algorithm efficiency. Another possible followup of our work might be to develop another rating algorithm in order to verify the evaluation of the presented algorithms.

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

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

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

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

[7] 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.

[8] Codasip. (2019) RISC-V processors. [Online]. Available: https://www.codasip.com/risc-v-processors/

[9] 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.

[10] ——, "Multidimensional Pareto Frontiers Intersection Determination and Processor Optimization Case Study," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2019, pp. 597–600.

[11] 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.

[12] 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.

[13] 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.

[14] S. B. Roy, A. Madhukumar, and J. Joung, "On joint pareto frontier in multiple access and relay rate regions with rayleigh fading," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 5, pp. 3777–3786, 2017.

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