ImpEDE: A Multidimensional Design-Space Exploration Framework for Biomedical-Implant Processors

Dhara Dave, Christos Strydis, Georgi N. Gaydadjiev
Computer Engineering Lab, Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands
E-mail: C.Strydis@tudelft.nl, D.Dave@student.tudelft.nl, G.N.Gaydadjiev@tudelft.nl

***Abstract*—The demand for biomedical implants keeps increasing. However, most of the current implant design methodologies involve custom-ASIC design. The SiMS project aims to change this process and make implant design more modular, flexible, faster and extensible. The most recent work within the SiMS context provides ImpEDE, a framework based on a multiobjective genetic algorithm, for automatic exploration of the design space of implant processors. The framework provides the processor designer with a Pareto front through which informed decisions can be made about specific implant families after analyzing their particular tradeoffs and requirements. A highly efficient, parallelized version of the genetic algorithm is also used to evolve the front and has as its objectives the optimization of power, performance and area. In addition, we illustrate the extensibility of our framework by modifying it to include a case study of a synthetic implant application with hard realtime deadlines.**

***Keywords*-Design-space exploration; simulation; optimization; genetic algorithm; biomedical microelectronic implant**

I. Introduction

From an engineering standpoint, medical implants constitute a highly resource-constrained class of embedded systems. A core component of an implant, its processor also inherits these limitations. Given this fact, currently existing, general-purpose processors are not ideal for implant applications. There is, instead, a need for new processors that are better suited for use in a wide range of implant applications. Accordingly, and as part of the ongoing SiMS project, effort has been put on developing a novel biomedical-implant processor. Optimal cache and branch-prediction subsystems for this processor have already been studied in [1], [2]. These studies have offered many design insights yet provide local and, by necessity, biased optimizations.

It is well-known that the global optimization of multiple design objectives – such as performance, power and area – across *all* processor subsystems is a non-trivial task. One must explore all possible processor configurations, compute the corresponding design objectives and find the *Pareto-dominant* solutions to be included in the final trade-off set. Since, typically, the design parameters that affect a processor are numerous, computing the behavior of all their possible combinations is quite hard, if not impossible. For example, by considering 13 processor *design parameters* represented by 36 (binary) bits, if one were to simulate all combinations, one would need to evaluate $2^{36} = 68,719,476,736$ different processor configurations to identify the true *Pareto front* – an unrealistically high number. To make things worse, in this new field of implant processors there is no established set of processor characteristics that would allow meaningful limiting of the above number of potential configurations.

To the best of our knowledge, none of the existing DSE tools are explicitly concerned with implantable systems. However, the field of biomedical, microelectronic implants is new and fast-progressing and, calls for particular design constraints such as ultra-low power consumption, high fault-tolerance levels and tight execution deadlines. What is needed is a fresh top-down approach to the field where implant applications are extensively profiled in a properly fine-tuned environment and the findings are used to drive an (automated, if possible) design-exploration effort for a suitable implant device. Setting up such an environment is a non-trivial problem as its specific parameters are either unknown or undisclosed, subject to tight proprietary controls. Building on our previous knowledge, with this work, we attempt to put together a multiobjective, DSE farmework, ImpEDE (Implant-processor, Evolutionary, Design-space Explorer). This framework provides careful investigation of the processor design space through the use of a particular *genetic-algorithm* (GA) variant called NSGA-II, along with cycle-accurate simulations, considering realistic design constraints imposed by our prior knowledge of the field. The implementation featured a very long computational time and, therefore, a parallelized version of the algorithm has also been implemented and described in the current document. Also, we attempt to fine-tune this environment to the goal at hand by providing the first – in our knowledge – tool to offer bounded DSE. We have made this tool freely available for further improvement, expansion and dissemination of information. Concisely, the contributions of this work are:

- A first yet educated attempt towards the systematic, automated and accurate design of implant processors;
- A fine-tuned toolset that delivers optimized implant-processor configurations across multiple first-order (e.g. performance, power) and second-order (e.g. hard real-time deadlines) objectives; and
- A freely available parallelized version that can be

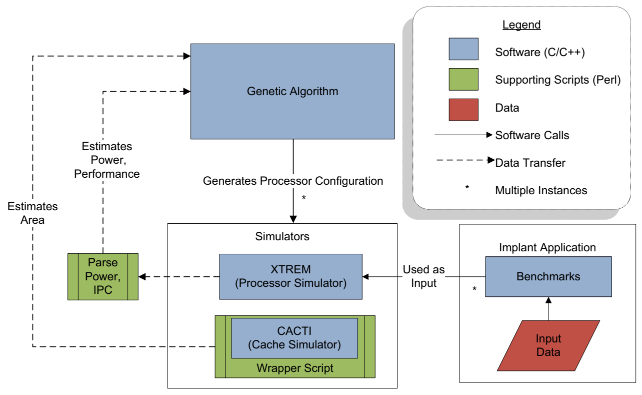


Figure 1: Framework organization.

expanded with additional design objectives and constraints and extended to applications classes other than implants.

The rest of the paper is organized as follows: section II briefly discusses other DSE toolsets available and explains the necessity for the current framework. Section III provides the overview and organization of ImpEDE while section IV discusses the performed customizations and selected parameters for making the framework ideal for architectural exploring of implant processors. Section V displays the validity of ImpEDE by presenting actual DSE results for our targeted implant processor and illustrates its flexibility by extending it with exploration under hard realtime constraints. Overall conclusions and future work are drawn in section VI.

II. Related work

A large number of design-space exploration tools have been presented in the past, targeting generic DSPs [3] to network processors [4] and, more recently, multicore processors [5], [6]. For an extensive overview of different approaches, the interested reader can refer to Matthias Gries [7].

One of the proposed methods is to employ Genetic Algorithms for design-space exploration purposes. Ascia et al. [8] have attempted to optimize processor subsystems yet the search-and-optimize algorithm they used is working only on a single objective at a time. Ghali and Hammani [9], on the other hand, have made use of a multiobjective setup, similar to ours, but address the optimization of Turbo decoders. Stijn et al. [10] have investigated various, automated, GA-based, single- and multi-objective DSE methods for custom processors, but focus on out-of-order flavors only.

III. Framework organization

Before introducing the DSE framework, we first have to identify the nature of the problem we attempt to solve: In designing our implant processor, we have formulated our problem as a multiobjective-minimization problem, with the objectives being processor *latency*, *average power consumption* and *area cost*. This is a set of first-order objectives typically optimized for in digital design. As the framework matures, we wish to add more objectives in our design effort such as *fault coverage* and more constraints such as *hard realtime deadlines*. In later sections, we will exemplify the latter by imposing a hard deadline on execution time (i.e. latency).

Since our objectives are minimizing area and power while maximizing performance, in order to convert our problem to a fully minimizing problem, we need to take the complement of performance as the objective encoded in our framework. We use IPC as the metric of performance. Therefore, we can simply use IPC*(-1) as the objective to be minimized[1]. For the rest, we use the metrics mm^2 and mW as the area and the power objectives, respectively. With these objectives in mind, we have designed the multiobjective, DSE framework shown in Fig. 1. At first, the selected GA (NSGA-II) generates valid processor configurations (i.e. a set of parameters) – encoded as "chromosome" – that are fed to a cycle-accurate, performance and power simulator (XTREM) and to a cache-area simulator (CACTI). The processor simulator also accepts as inputs implant-related benchmarks and assorted datasets (ImpBench). Then, both simulators execute and their resulting performance, power and area figures are fed back into the waiting GA which uses them to evaluate the *optimality* of the currently simulated processor configuration. This process is repeated a number of times equal to the preset chromosome *population*, then a few best-performing chromosomes – based on their fitness results – are *selected*, processed and propagated to the next round of optimizations, also known as *generation*. With each successive generation, increasingly better chromosomes are found and promoted; that is, we are approaching the true Pareto front for our DSE problem. Figure 2 shows the front found for two particular runs of our framework[2]. In the following subsections, we will describe in more detail the components of the framework as well as the choices made on the GA parameters such as the population size, the number of generations and the chromosome-selection policy used[3].

A. Genetic algorithm: NSGA-II

The classical single-objective optimization methods can be used to perform multiobjective optimization by reformulating the multiobjective optimization problem into a single-objective one. However, these methods suffer from several drawbacks such as requiring advanced knowledge of the design space and inability to find solutions for problems

[1]Note that cycles per instruction (CPI) could have also been used: since CPI is the inverse of IPC, it would give an identical relative ordering of processor configurations as IPC*(-1).

[2]Note that the Area axes are on a Logarithmic scale in all the figures in this paper.

[3]An extensive analysis can be found in [11].

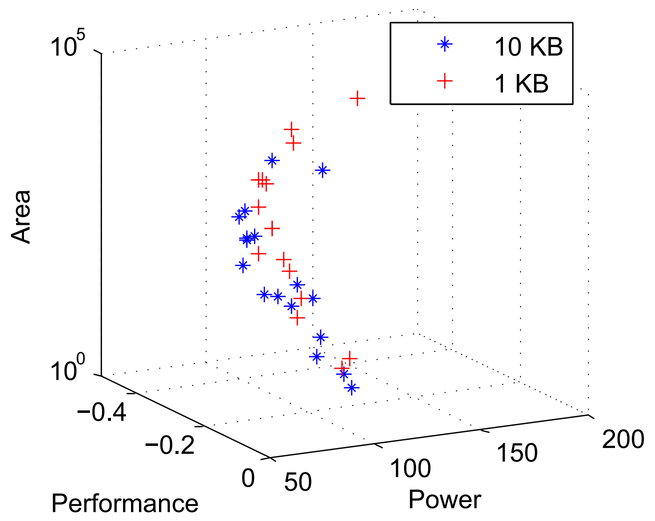


Figure 2: Framework-generated Pareto solutions (i.e. implant-processor instances) for two workload sizes.

having non-convex fronts. Therefore, we use an algorithm specially designed for multi-objective problems.

NSGA-II [12] evolves Pareto fronts using an elitist approach and uses density and crowding distance metrics to ensure well spread out points along the front, at the same time having a lower computational complexity than its predecessor, NSGA. Due to its superiority over other algorithms, popularity, ease of use and availability, we use it as our algorithm of choice.

B. Processor & cache simulators

In the current version of our DSE framework, evaluation of the performance and power consumption of a given chromosome (i.e. processor configuration) has been based on the XTREM [13] **simulator**. XTREM simulator is a cycle-accurate, microarchitectural, power and performance simulator for the Intel XScale core [14]. It has been selected for its straight-forward functionality but mostly for its accuracy in performance and power modeling. It exhibits an average performance error of 6.5% and an average power error of 4% compared to real hardware. We have extensively used XTREM in our previous studies on the implant processor and, in order to match our application field better, we had disabled many of XTREM's architectural parameters. In this case, however, we wish to allow for some degree of freedom in the processor design parameters so that the GA can explore a wider range of possible configurations. We have, thus, ended up with the (modified) XTREM characteristics summarized in Table I[4]. Performance/power figures have been checked and scaled properly with the changes. In the next section IV, we shall go through the selected simulator parameters and the way they have been encoded in the GA. It should be noted that flexible wrapper scripts have been used

[4]Values denoted with 'VAR' indicate adjustable parameters by the GA.

Table I: Architectural details of (modified) XTREM.

Feature	Value
ISA	32-bit ARMv5TE-compatible
Pipeline depth / width	7/8-stage pipeline / 32-bit
RF size	16 registers
Issue policy	in-order
Instruction window	single-instruction
I/D Cache L1 (separ.)	VAR size&assoc. (1-cc hit / 170-cc miss lat.)
BTB	VAR size, fully-assoc. / direct-mapped
Branch Predictor	VAR (4-cc mispred. lat.)
Ret. Address Stack	VAR size
I/D TLB (separ.)	1-entry / 1-entry
Write Buf. / Fill Buf.	2-entry / 2-entry
Mem. bus width	8-bit (1 mem. port)
INT/FP ALUs	1/1
Clock frequency	2 MHz
Implem. technology	0.18 μm @ 1.5 Volt

to provide the input to and capture the output of XTREM. As a result, the internal framework structure has been kept highly modular allowing for porting faster, more accurate or more powerful simulators in the future.

For evaluating the area cost of each chromosome, we have made the valid approximation that the subsystem dominating our envisioned implant processor is the cache (which holds also true for modern general-purpose processors). Furthermore, as can be seen from Table I, more adjustable parameters include some cache-like structure in them. Therefore, for quantifying each chromosome's area cost, we have used CACTI, a well-known, cache-area estimation tool. CACTI v3.2 has been primarily used since it is suitable for modeling simpler (older) cache-like structures (such as the BTB) and at an implementation technology identical to the one of the simulator (180 nm). However, our framework at present (Fig. 1) can also handle CACTI versions 4.1 and 6.0.

C. Biomedical benchmarks & workloads

Eight suitable **benchmark applications** have been used for execution on XTREM for evaluating different chromosomes. They comprise the ImpBench benchmark suite [15] and consist of *lossless data compression* algorithms, *symmetric-key encryption* algorithms and *data-integrity* algorithms as well as representative code based on *real biomedical applications*. The benchmarks represent anticipated common tasks running on future implant processors and exhibit varied characteristics, as shown in Table II.

Typical biomedical readouts are often highly periodic signals (e.g. heart beat) or stable signals (e.g. blood temperature) which can, under specific circumstances, display gradual or abrupt changes in value (e.g. a sudden muscle contortion). We have collected and used various representative workloads capturing both stable as well as rapidly changing patterns. The original data has been provided from the BIOPAC (R) Student Lab PRO v3.7 Software. Paper-size limitations do not allow for an extensive description of the various workloads; a concise overview of the workload

Table II: ImpBench [15] benchmarks. Columns denoted with a (*) indicate average values for $1-KB$ input workloads.

Benchmark type	Name	Size (KB)	#Instr.*	Sim. time* (sec)
Compression	miniLZO	16.30	199,163	3.07
	Finish	10.40	852,663	21.82
Encryption	MISTY1	18.80	1,268,465	16.65
	RC6	11.40	864,930	9.37
Data integrity	checksum	9.40	62,869	0.80
	CRC32	9.30	419,159	5.46
Real applications	motion	9.44	859,371	31.16
	DMU	19.50	36,808,268	37.25

Table III: Biomedical workloads with double-precision (8-Byte) data samples of sizes 1-KB and 10-KB.

Dataset name	size (Bytes)	Samples (#)	duration (sec)
Electromyogram II (EMGII)	1147 / 9605	144 / 1201	0,288 / 2,402
Electroencephalogram (EEGI)	984 / 9616	123 / 1202	0,615 / 6,010
Electrocardiogram (ECGI)	912 / 9615	114 / 1202	0,114 / 1,202
Respiratory Cycle I (RCI)	1192 / 9520	149 / 1191	1,490 / 11,910
Pulmonary Function I (PFI)	1184 / 9240	148 / 1155	1,480 / 11,550
Skin Temperature (AEP)	1120 / 9736	140 / 1217	0,700 / 6,085
Blood Pressure (BP)	1128 / 9545	141 / 1198	0,282 / 2,396

details is provided in Table III. Since reported literature [16] has revealed that typical implant data-memory sizes range from 1 KB to 10 KB, workloads of both sizes (1 KB and 10 KB) have been profiled.

Each chromosome represents a particular processor instance onto which each all benchmarks except *DMU* (which runs on a single, hard-coded input) are fed with each of the 7 workloads (of (1 KB or 10 KB)) and are executed. This accounts for a total of 50 benchmark runs for the 1 KB case and another 50 for the 10 KB case. As we shall see in subsection III-D, this is a substantial amount of (cycle-accurate) simulation time. In order to get practical results in our limited time frame and without loss of generality, for this paper we have selected and executed only the EMGII workload as it displays worst-case performance characteristics. In so doing, we have limited the number of runs to 8 per workload size. We, nonetheless, explore the design space for both sizes in order to investigate the effect workload size has on the Pareto-optimal solutions.

D. Parallelization & optimization

As shown in Table II, evaluating a single processor configuration (i.e. one GA individual) with a single input (EMGII) and a single data size (1KB), across all 8 benchmarks takes on an average 125.58 seconds[5]. Assuming the $10KB$ workloads run $10\times$ as slow as the $1KB$ workloads, except for the DMU benchmark and, considering optimization across all 7 Workload types, a full run of the GA with a population size of 20 and 200 generations will take approximately 343 days per result. We do not consider the GA run times in this calculation as the execution overhead of the parallelized GA was found to be negligible, contributing only 0.13 seconds per generation.

Since this run-time is quite prohibitive, we parallelized the evaluation stage of the GA so that different individuals are evaluated on idle CPUs of the group's laboratory machines. Hence, the speedup offered by our parallelized version is equal to $\sim P/\lceil P/N \rceil$, where P is the population size and N is the number of computers available. During the runtime of the GA, support scripts periodically search for and prepare free machines for the algorithm to use; these machines are used on the lowest priority in order to not disrupt regular usage. Therefore, this framework is expandable, modular and requires minimum dedicated resources.

IV. Framework fine-tuning

In the previous section, we went through the various building blocks of the DSE framework. Adjusting the framework to target implant processors requires, however, fine-tuning of the GA parameters and proper encoding (i.e. representation) of the chromosomes. In what follows, we go through such details that make our framework suitable for implant-processor design.

A. Chromosome encoding

Since GAs optimize the information encoded in the chromosomes, we needed to define a chromosomal representation for the processor parameters that the GA can work with. Each chromosome is encoded as shown in Table IV using a binary encoding. The table lists the processor variables chosen, their ranges as well as the encoding and decoding rules. The included variables are in agreement with the ones in Table I.

The processor *parameters* we chose to include in the search space depended both on what we wanted out of the processor and the capabilities and limitations of the simulators we had. As can be seen from Table IV, clock frequency has been encoded but was not used in this version of the GA; we found that running the simulator with different clock frequency values did not affect the results (although they are expected to be). For the purposes of this work, XTREM runs on a default clock frequency of 2 MHz, typical for implant processors.

The Write Buffer and the Fill Buffer included in XScale and shown in Table I help achieve better performance by hiding memory-access stalls when the core is running at a high clock frequency (e.g. 200 MHz). This is hardly the case for an implant processor; therefore, we did not encode the two buffers but rather fixed their sizes at the minimum supported by XTREM, i.e. exactly two entries. For similar reasons, Translation Lookaside Buffers (I/D-TLBs) have been excluded from the GA and fixed to a single-entry structure each.

We found that varying the I-cache latency above the default value of 1 often had XTREM crashing. Therefore,

[5]Measured on a dual-core, AMD Athlon(TM) XP 2400+ @ 2000.244 MHz, cache size 256 KB running Fedora 8 linux

Table IV: Processor design parameters considered in this work, encoded as 36 chromosomal bits.

Parameter		**Encoded**		**Decoding**	**Remarks**
Name (Ref)	**Range**	**Range**	**Bits**	**Formula**	
Core Clock Frequency ($Freq$)	[1...64]	[0...63]	6	$n+1$	Not used in current version
Branch Prediction ($Bpred$)	$Bimodal, Taken, notTaken$	[0...2]	2	-	$bit_0 = isBimod$, $bit_1 = isTaken$
Branch Target Buffer: Number of Sets (btb_nsets)	[32...128]	[0...5]	3	2^{n+5}	Only valid for $isBimod = TRUE$
Branch Target Buffer: Associativity (btb_assoc)	[1...32]	[0...5]	3	2^n	Only valid for $isBimod = TRUE$
Branch Prediction: Return Address Stack (RAS)	[0...8]	[0...4]	3	$floor(2^{n-1})$	-
L1 I/D-Cache: Number of Sets (I/D_nsets)	[1...8192]	[0...13]	4	2^n	-
L1 I/D-Cache: Block Size (I/D_bsize)	[8...32]	[0...2]	2	2^{n+3}	-
L1 I/D-Cache: Associativity (I/D_assoc)	[1...32]	[0...2]	3	2^n	-
L1 I/D-Cache: Replacement Policy (I/D_repl)	f, r, l	[0...2]	2	-	$bit_0 = isFifo$, $bit_1 = isRandom$
L1 D-Cache: Latency ($D\_latency$)	[1...16]	[0...4]	3	$floor(2^n)$	-
L1 I-Cache: Latency ($I\_latency$)	[1...16]	[0...4]	3	$floor(2^n)$	Not used in current version

we only include D-cache latency as a variable. Note also that the above discussion pertains only to L1 caches. At a stage where embedded applications hardly have even an L1 cache, we think having L2 caches would be an overkill, and therefore exclude them from our exploration. This is also in agreement with the results in [1].

B. Population size

The size of the population represents the maximum number of Pareto points the algorithm can find. If we pick too small a size for the population, we would have lesser tradeoffs available. On the other hand, the GA is $O(mN^2)$, where m is the number of objectives and N is the population size. Keeping these factors in mind, we chose a population size of 20, which also coincided with the number of machines we expected to be free at any given time. Therefore, the entire population could be evaluated at once in parallel.

C. Number of Generations

The number of generations represent the time the GA is allowed to reach the Pareto front. We observed that the GA converges rather rapidly to a front, then tries to increase its spread in the subsequent generations. After this, the GA reaches a sort of 'stable state' where it is unable to improve the spread without degrading the distance from the front, and vice versa. We can limit the number of generations at this phase, in order to reduce computation time. We use a reduced problem set - that of only optimizing the *checksum* benchmark, and run this for a large number of generations (1000) in order to approximate the number of generations needed, then use this as a guide to selecting the number of generations for the full problem.

We use Deb's metric (Δ) [17] to quantify the diversity (spread) of the solutions and Veldhuizen's Generational Distance (GD) [18] metric to quantify the distance of the solution front from the 'true' Pareto front[6]. We found both metrics to be very noisy, specially the diversity metric. Therefore, to make them easier to compare, we smooth the data using a moving average with a span of 20 generations. Figure 3 shows the resulting metrics. We observer that GD declines rapidly until about the 100th generation, after which it fluctuates around GD=0.4 for a long time until finally dropping to zero around generation 600. The rapid decline is of course due to the solution fronts converging towards the reference. After generation 51, it can be seen that the general trend is that as spread decreases, distance increases and vice versa. The behavior from generations 100-593 is also expected - as the algorithm searches for new solutions, the distance fluctuates. A minor rise in both GD and spread may not mean that the front is actually further away than the one previously – since we have finite points in the reference front, points that are the same distance from the *actual* front may give slightly varying distance values from the reference front. The behavior at the tail end reflects the fact that the reference solution is a combination of several solutions, and therefore also contains the final solution of the run in question. The apparent rapid convergence seen therefore, is in fact the algorithm moving towards its own final solution – a foregone conclusion. Therefore, we do not consider this region in our analysis.

Since after generation 100, the algorithm seems to oscillate between improving spread and distance at the cost of the other; without loss of precision in both quantities at the same time, we can stop the algorithm around generation 100. Since GAs are random by nature, in order to be sure of getting good results, we run every subsequent experiment for twice this time, i.e. – 200 generations. This also compensates for the smoothing we performed.

D. Mutation

The simplest and most recommended strategy is simply setting the mutation probability as $p_m = 1/n$ where n is the chromosome length [19]. This means that a single bit per chromosome is expected to change from the parent to the child population. Therefore, the child chromosome is likely to vary from the parent chromosome in exactly one attribute, that too by a hamming distance of one. We use this approach for setting the mutation probability in our implementation of the GA, i.e $p_m = 1/36$.

[6]Since we do not know the true front (by problem definition itself), we approximate it by computing a combined front consisting of mutually non-dominating points from the results of 10 separate runs of the algorithm. We call this the 'reference front'.

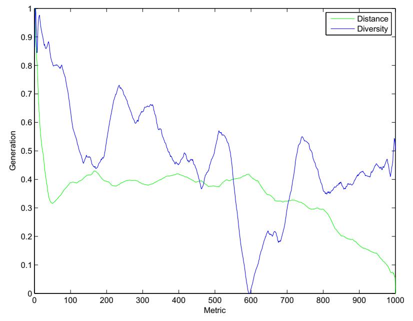
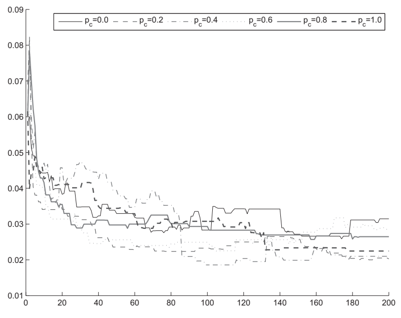
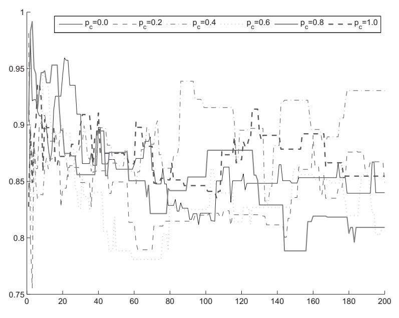


Figure 3: Smoothed distance and diversity metrics over 1000 generations (Benchmark: checksum)

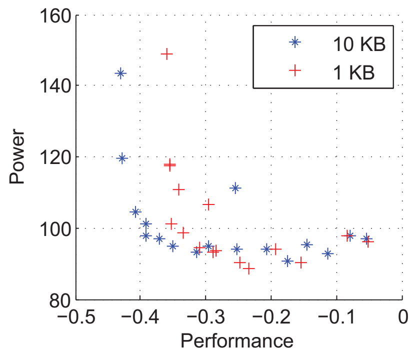


(a) Distance metric

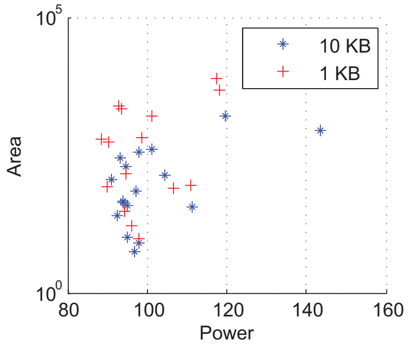


(b) Diversity metric

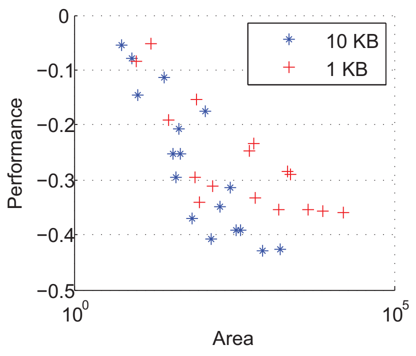
Figure 4: Distance and Diversity metrics for various crossover probabilities (P_c) over 200 generations (Benchmark:checksum)



(a) Performance-Power



(b) Power-Area



(c) Performance-Area

Figure 5: Baseline DSE results for 1 KB and 10 KB workloads running on all benchmarks.

E. *Crossover probability*

Used carefully, crossover can lead to much quicker evolution times, and is central to the idea of Genetic Algorithms examining more solutions in the more promising regions of the solution space. Therefore, it is important to set a good *crossover probability* P_c, which determines the percentage of chromosomes that undergo recombination at each generation. As in the case of number of generations, we used a reduced problem set – running only *checksum* with the $1-KB$ EMGII input for 200 generations with different crossover probabilities. Figure 4 shows the two metrics for the Pareto fronts resulting from each value of P_c [7]. Keeping in mind the discussion from Section IV-C, we see from the graphs that $P_c = 0.2$ and $P_c = 0.6$ seem to lead to the fastest convergence and best values for the two metrics over the course of the generations, and therefore $P_c = 0.2$ is chosen for subsequent runs.

V. Selected results & validation

In this section, the correct functionality of the framework is demonstrated. Also, an expansion of the framework with

[7]Note that in this case, these are the un-normalized, un-smoothed metrics. They appear less noisy due to the fact that the number of generations plotted is much lower than in Section IV-C

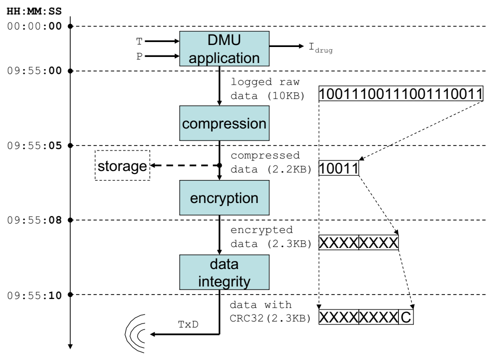


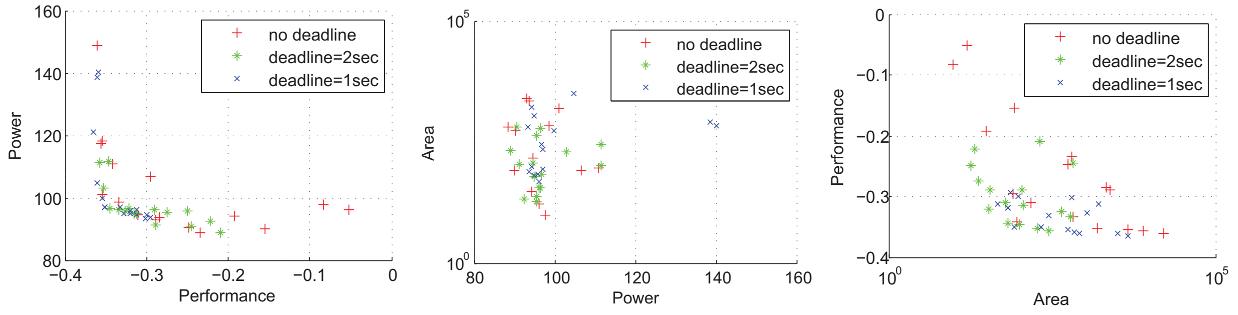
Figure 6: Conceptual block diagram of simulated implant application [20].

a hard realtime deadline leading to constrained design is illustrated.

A. *Implant-processor results*

Having prepared and fine-tuned the framework to the best of our knowledge, we move on to testing it by running it with all the benchmarks, once for each of the two workload sizes. We call these the *baseline* results. Figure 5 shows the projections of the Pareto front evolved on the 3 Cartesian planes (performance, power, area)).

We see from Fig. 5a and Fig. 5c that the $10-KB$ work-



(a) Performance-Power (b) Power-Area (c) Performance-Area

Figure 7: DSE results expanded with hard realtime deadlines of 2 $seconds$ and 1 $second$ for 10 KB workloads running on all benchmarks.

loads have a wider front w.r.t. performance. We anticipate this to be so because the bigger workload increases processor utilization by allowing the caches and branch predictor table to fill and, hence, minimizing processor stalls. This higher performance also leads to a corresponding increase in the power consumption as can be seen from the power axes in Fig. 5a and Fig. 5b.

On the contrary, we see slightly bigger area-utilization solutions for the smaller workload. Although measurements with more workload sizes are needed to draw safe conclusions, this area trend may again be due to ”cold-start” effects; that is, due to the fact that when small workloads are processed, poor CPU utilization occurs since the cache structures do not have enough time to fill, pushing the GA towards larger structures in the hopes of minimizing cache stalls.

B. *Framework expansion*

As this was one of the first steps towards designing an implant processor, we wanted to make sure that the framework was expandable, in order to facilitate the addition of new domain specific information into the framework as it gets available. In order to test this property of the framework, we devised a synthetic implant application with a hard realtime deadline. Indeed, many implant applications have realtime requirements so formulating a synthetic problem with such a constraint does not fall far from practice. We modified the DMU and Motion benchmarks (called StressDMU and StressMotion respectively), to represent a single iteration of these (repetitive) applications. Further, this iteration was chosen to be the worst-case iteration (in terms of instruction-cycle count) for each of the two original benchmarks.

Combined with data-integrity checks, compression and encryption, the stress benchmarks represent an *atomic* action for an implant application – from data collecting, processing, to transferring – as exemplified in Fig. 6. In a real application, this atomic action must be finished, for example, before the next set of input arrives. Therefore, we *constrain* the total time required for this combined operation as a hard realtime deadline.

Out of the ImpBench set, we chose checksum, miniLZO and RC6 as the data-integrity, compression and encryption algorithms, respectively. We obtain the simulated execution time of the processor configuration under investigation from the simulator output. In case the deadline is violated, the processor configuration is deemed to be unacceptable. On the other hand, if the deadline is met, we calculate the objectives of the configuration by combining with the rest of the benchmarks in the test suite (including ”normal” DMU and Motion). Therefore, the fitness metric remains the same as the baseline case.

Figure 7 shows the Pareto front evolved with a deadline of 1 second, and also with a slightly relaxed deadline of 2 seconds. As expected, the stricter deadline encourages processor configurations that have a higher performance, but at the same time take slightly more power and area.

VI. Conclusions & future work

Although reliability is one of the major reasons for the need to design processors specifically for implants, the present work does not directly address reliability. Instead, it starts off with the idea that processor design can be looked at from a black-box design perspective and provides a flexible and modular framework for doing so. Given such a framework, adding reliability as one of the optimization objectives is the next logical step, left as future work. We would also like to expand the simulator models with more parameters such as (off-chip) memory, effectively allowing for System-on-Chip exploration. In order to overcome the aforementioned limitations of XTREM, we are considering the XEEMU [21] simulator as a candidate for the next phase of the framework.

In conclusion, this paper introduces ImpEDE, a novel, multiobjective, framework that provides high-level DSE of biomedical-implant processors, populated by suitable biomedical benchmarks and assorted workloads. ImpEDE organization is described in detail and its functionality is fine-tuned based on our previous experience (e.g. processor parameter values and ranges) and new findings (e.g. crossover probability, workload size). Restricted by its simulator components, the current framework version can deliver (near) Pareto-optimal processor solutions, co-optimized across performance, power consumption and area utilization. In view of potentially more optimization goals and benchmarks, we have paid attention to making the framework modular and expandable. Furthermore, we have provided a parallelized, versatile version of the framework which offers execution speedup roughly equal to the number of processor available, without dedicated hardware resources. Last, it has been our intension to make the proposed framework a freely accessible tool, available to everyone online for further studies and improvements.

VII. Acknowledgements

This work has been partially supported by the ICT Delft Research Centre (DRC-ICT) of the Delft University of Technology and by Google Inc.

References

[1] C. Strydis and G. Gaydadjiev, "Suitable cache organizations for a novel biomedical-implant architecture," in *International Conference of Computer Design (ICCD'08)*, Lake Tahoe, California, USA, 12-15 October 2008, pp. 591–598.

[2] C. Strydis and G. N. Gaydadjiev, "Evaluating Various Branch-Prediction Schemes for Biomedical-Implant Processors," in *Proceedings of the 20th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP'09)*, Boston, USA, July 2009, pp. 169–176.

[3] M. Lorenz, R. Leupers, P. Marwedel, T. Drager, and G. Fettweis, "Low-energy DSP code generation using a genetic algorithm," in *Proceedings of International Conference on Computer Design (ICCD) 2001, Austin, Texas*, 2001, pp. 431–437.

[4] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli, "Design space exploration of network processor architectures," *Network Processor Design: Issues and Practices*, vol. 1, pp. 55–89, 2002.

[5] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis, "Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation," in *LCTES/SCOPES '02: Proceedings of the joint conference on Languages, compilers and tools for embedded systems*. New York, NY, USA: ACM, 2002, pp. 18–27.

[6] A. Pimentel, L. Hertzberger, P. Lieverse, P. van der Wolf, and F. Deprettere, "Exploring embedded-systems architectures with Artemis," *Computer*, pp. 57–63, 2001.

[7] M. Gries, "Methods for evaluating and covering the design space during early design development," *Integr. VLSI J.*, vol. 38, no. 2, pp. 131–183, 2004.

[8] G. Ascia, V. Catania, and M. Palesi, "Parameterised system design based on genetic algorithms," in *CODES '01: Proceedings of the ninth international symposium on Hardware/software codesign*. New York, NY, USA: ACM, 2001, pp. 177–182.

[9] K. Ghali and O. Hammani, "Embedded Processor Characteristics Specification Through Multiobjective Evolutionary Algorithms," in *IEEE International Symposium on Industrial Electronics (ISIE'03)*, 2003, pp. 907–912.

[10] S. Eyerman, L. Eeckhout, and K. De Bosschere, "Efficient design space exploration of high performance embedded out-of-order processors," in *DATE '06: Proceedings of the conference on Design, automation and test in Europe*. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2006, pp. 351–356.

[11] D. Dave, "Automated implant-processor design: An evolutionary multiobjective exploration framework," Master's thesis, TU Delft, 2010.

[12] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast elitist multi-objective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, 2000.

[13] G. Contreras *et al.*, "XTREM: A Power Simulator for the Intel XScale Core," in *LCTES'04*, 2004, pp. 115–125.

[14] *Intel XScale Microarchitecture for the PXA255 Processor: User's Manual*, Intel Corp., March 2003.

[15] C. Strydis, C. Kachris, and G. Gaydadjiev, "ImpBench - A novel benchmark suite for biomedical, microelectronic implants," in *To appear in International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS'08)*, Samos, Greece, 21-24 July 2008.

[16] C. Strydis *et al.*, "Implantable microelectronic devices: A comprehensive review," Computer Engineering, TU Delft, CE-TR-2006-01, Dec. 2006.

[17] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, LTD, 2001.

[18] D. A. V. Veldhuizen and G. B. Lamont, "Evolutionary computation and convergence to a pareto front," in *Genetic Programming 1998: Proceedings of the Third Annual Conference*. University of Wisconsin, Madison, WI, USA: Morgan Kaufmann, 22-25 Jul. 1998.

[19] T. Back, "Optimal mutation rates in genetic search," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, 1993, pp. 2–8.

[20] C. Strydis and G. Gaydadjiev, "The Case for a Generic Implant Processor," in *30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC'08)*, August 2008, pp. 3186–3191.

[21] Z. Herczeg, Á. Kiss, D. Schmidt, N. Wehn, and T. Gyimothy, "XEEMU: An improved XScale power simulator," *Lecture Notes in Computer Science*, vol. 4644, p. 300, 2007.