Soft-core Processor Customization using the Design of Experiments Paradigm

David Sheldon, Frank Vahid*, Stefano Lonardi

Department of Computer Science and Engineering
University of California, Riverside
**Also with the Center for Embedded Computer Systems at UC Irvine*
{dsheldon,vahid,stelo}@cs.ucr.edu

Abstract

Parameterized components are becoming more commonplace in system design. The process of customizing parameter values for a particular application, called tuning, can be a challenging task for a designer. Here we focus on the problem of tuning a parameterized soft-core microprocessor to achieve the best performance on a particular application, subject to size constraints. We map the tuning problem to a well-established statistical paradigm called Design of Experiments (DoE), which involves the design of a carefully selected set of experiments and a sophisticated analysis that has the objective to extract the maximum amount of information about the effects of the input parameters on the experiment. We apply the DoE method to analyze the relation between input parameters and the performance of a soft-core microprocessor for a particular application, using only a small number of synthesis/execution runs. The information gained by the analysis in turn drives a soft-core tuning heuristic. We show that using DoE to sort the parameters in order of impact results in application speedups of 6x-17x versus an un-tuned base soft-core. When compared to a previous single-factor tuning method, the DoE-based method achieves 3x-6x application speedups, while requiring about the same tuning runtime. We also show that tuning runtime can be reduced by 40-45% by using predictive tuning methods already built into a DoE tool.

1. Introduction

Soft-core processors are becoming increasingly common in modern technology. A soft-core processor is a programmable processor that can be synthesized to a circuit, typically integrated into a larger system existing on an application-specific integrated circuit (ASIC) or field-programmable gate array (FPGA). Popular commercially available soft-cores include ARM [1], Tensillica [13], Microblaze [14], and Nios [2]. Several trends of modern technology catalyze soft-core usage, including stabler synthesis tools, higher-capacity ASICs and FPGAs, new commercial toolsets for application-specific instruction-set processors, and increasing demands for high-performance and low-power embedded processing.

Whereas traditional pre-fabricated processors must be optimized for good performance across an entire domain of applications, soft-cores can instead be customized to the particular applications they execute. For example, a particular application may perform best on a processor having a large cache and a floating-point unit, while another application may instead require a hardware multiplier and have no need for a cache. Nearly all soft-core providers therefore include parameters that may be customized by a soft-core user. Common parameters include instantiatable coprocessor units such as hardware floating-point, multiplier, divider, or barrel shifter units; cache architecture settings such as the use of one or two levels of cache, separate versus unified cache, cache size, line size, associativity, and write back policy; and processor micro-architecture features such as pipeline depth, bypass logic, or register-file size. The majority of the soft-core processors mentioned above contain more than ten customizable parameters, and the trend in newer versions is towards increasing the number of parameters. In this work, we only consider customizations that consist of tuning parameter values; such customizations are in a different category than are application-specific instruction-set extensions [13], which involve the design of new hardware and/or the introduction of new instructions. Extensions could be integrated into a tuning approach by pre-determining good possible extensions, and treating each possibility as a particular value of an “extension” parameter.

The process of customizing a soft-core by tuning parameter values can yield improved performance, lower-energy, and/or smaller size. A newly evolving size-constrained scenario involves dozens of FPGA soft-cores competing for limited hardware-resources [9], for which tuning will be important to make best use of that limited hardware.

Soft-core providers offer little or no automated support to assist users in customizing a soft-core’s parameters to a particular application, other than providing simulation tools. As a consequence, users must manually guess and simulate (or implement) a set of candidate configurations in order to find the best parameters for a particular application. Each such simulation may require tens of minutes, limiting the number of candidate configurations that can be examined. Some recent research addresses automated soft-core configuration using custom heuristics developed for a particular parameters [12]. However, those custom heuristics do not scale to handle a broader range of parameters, as shown below.

In this paper, we propose to map the soft-core tuning problem to a well-established scientific paradigm known as *Design of Experiments (DoE)* [8][10]. DoE is an eighty years old statistics discipline which aims to optimize parameterized physical phenomena through a carefully chosen set of experiments. In the DoE framework, one of the objectives is to minimize the number of experiments because each experiment is costly and/or time-consuming. The main idea is to design the set of experiments such that subsequent analysis of the resulting data provides maximum information about the impact of each parameter on the metrics of interest.

We claim that the work in this paper makes two distinct contributions. First, we will show that the DoE-based strategy yields a more robust soft-core customization methodology. The DoE-based method not only handles a wider variety of parameter types than previous approaches, but also results in better-customized soft-cores having improved performance than obtained by previous heuristics. Second, to the best of our knowledge, this work represents the first use of the DoE paradigm in design automation. With the increasing introduction on the market of highly parameterized intellectual property components for use in system-on-chip designs, the DoE paradigm may prove useful for a variety of design automation problems involving the customization of a large spectrum of parameterized components or platforms.

2. Previous Work

Kumar [6][7] showed the benefit of multi-core general-purpose processor chips having heterogeneous rather than homogenous cores. They considered superscalar processor parameters related to cache, instantiations of floating-point, multiply, and arithmetic-logic units, and sizes of the register file, translation lookaside buffer, and load/store queue, yielding 480 possible single-core configurations. Via exhaustive search, they showed that an optimally configured four-core system has up to 40% better performance for a given workload, versus the best homogeneous four-core system for that workload.

Givargis [5] developed a tuning approach for parameterized system-on-a-chip platforms, considering parameters related to cache, bus, processor voltage, and a few parameters in peripherals. They used a user's denotation of independent subsets of parameters to extensively prune the configuration space before searching dependent parameters exhaustively or using heuristics. They showed roughly 5x tradeoffs between power and performance for different applications.

Sekar [11] discussed trends toward highly parameterized platforms, including parameterized processor cores, peripherals, caches, etc., and then described a technique for dynamically tuning the voltage and frequency of the processor.

Yiannacouras [15][16] developed a framework for generating and customizing a soft-core for FPGAs, with parameters including hardware versus software multiplication, different shifter implementations, and pipeline depth. They showed 30% improvements obtained by optimally tuning soft-core parameters for a specific application, using exhaustive search to carry out the tuning. Their work motivates the need to develop efficient automated customization heuristics.

We previously [12] developed heuristics for soft-core parameter tuning. The approach assumed that synthesis and execution (or simulation) of soft-core configurations, rather than pure estimation approaches, is essential for accurate evaluation of FPGA soft-cores, due to the tremendous variation of soft-core performance for different applications and across the hundreds of different FPGA devices by an FPGA vendor. Because synthesis/execution runs are costly, requiring tens of minutes or more, we developed several tuning heuristics that utilized only about a dozen synthesis/execution runs, thus executing in 1-2 hours. We considered a Xilinx Microblaze soft-core processor whose parameters each involved the option of instantiating a hardware component, including a hardware multiplier, barrel shifter, divider, floating point unit, or a fixed-sized cache. That work showed 2x application speedups of a customized core versus a base core having no optional components instantiated.

Our previous best heuristic (as well as our other heuristics) used what we will call a "single factor" analysis, a common analysis approach. The heuristic was guided by the speedup versus the base core when instantiating exactly one (single factor) of the core's optional hardware components. The heuristic then sorted each component by the ratio of its speedup over size, yielding an "impact-ordered tree" of parameters, which the heuristic then descended (encountering two choices per tree level) to find a solution. While a single-factor analysis is effective for on/off-type parameters, such an approach lacks an obvious extension for parameters that have two non-zero values (which value would be the base value?) or that have three or more values. Furthermore, a single-factor analysis may be inaccurate if parameters are interdependent. For example, neither of two components may individually yield speedup, but the two together may; conversely, two components may individually each yield speedup, but instantiating both may yield little benefit beyond instantiating just one, due to functionality overlap. In contrast, the approach we introduce here performs a multi-factor analysis, supporting multi-valued parameters and considering interdependent parameters, as will be described.

3. The Design of Experiments Paradigm

Design of Experiments (DoE) [8][10] is a statistical paradigm whose objective is to design a small set of experiments that provides maximum information on how the experimental parameters, or *factors*, influence the output and interact with one another. The basic assumption in DoE is that experiments are costly (with respect to money and/or time) and thus only a few can be conducted. DoE was originally created by Sir Ronald Fisher in the 1920s for agriculture (in which factors might for example involve different watering schedules or fertilizer types, and output might be crop quality), and has since expanded to a wide variety of domains, including even management of workers.

DoE produces three types of statistics. The first identifies the extent of positive or negative impact that each factor has on the output. The second indicates whether each factor is beneficial to the experiment. The third describes how factors interact with one another. For a given number of allowed experiments, where each experiment involves setting each factor to a specific value and then measuring the output, DoE seeks to design experiments providing maximum quality of statistical information.

DoE experimental design is based on four main principles: *randomization, replication, blocking,* and *orthogonality*. Randomization means that the experiments are performed in a random order. Replication means that each experiment is repeated. Blocking is the process of grouping different experiments into a group and running those experiments in the same environment. Orthogonality is the process of creating the experiments so that one factor can be analyzed independently of the other factors. Orthogonality is the principle of main interest to our work.

Several experimental design methods have evolved to create a set of runs that satisfy the orthogonality, as well as the other, principles. The *factorial* method (called an *exhaustive* search method by the design automation community) examines all possible combinations of the factors, thus providing maximum information, but obviously being infeasible in most situations due to too many experimental runs. The *fractional factorial* method runs only one half to one eighth of the total number of runs of the factorial design, increasing feasibility to more situations, but still often impractical due to too many runs. In contrast, the *Taguchi* [10] and the *Plackett-Burman* [10] methods generate just a few more runs then the number of factors, thus using a linear rather than an exponential number of runs with respect to the number of factors, at the expense of less accurate information obtained about the factors.

DoE uses various statistical techniques to analyze the data resulting from a set of runs, namely *analysis of variance* (ANOVA) that separately analyzes the variation of each of the factors and *multiple regression* that determines whether a factor is statistically significant.

We observed that the soft-core customization problem can be mapped to the DoE paradigm. The parameters of a soft-core correspond to factors of the experiment, whereas the performance of an application running on the soft-core corresponds to the output of the experiment. The time required for synthesizing and executing/simulating an application on a configured soft-core corresponds to the cost of an experiment[1].

In what follows, we exploit DoE's strengths in experimental design and analysis methods to customize a soft-core for an application. In contrast to our previous single-factor analysis approach [12], a DoE approach considers interactions among parameters. We thus hypothesized that a DoE approach would yield better results than the previous single-factor method.

4. Experimental Methodology

In this section, we describe the features of our experimental setup for soft-core tuning using DoE methods. Our objective is to find a soft-core configuration that maximizes performance of a given application, subject to a size constraint.

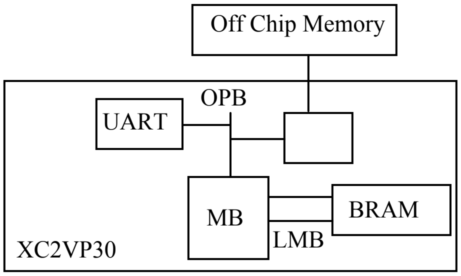
4.1 Soft-Core Framework

We use a Xilinx Microblaze soft-core processor on a Virtex 2 Pro device, illustrated in Figure 1. The system consists of a UART for external communication, an on-chip BRAM (Block RAM) accessed through the Xilinx's LMB (local memory bus), and access to off chip memory via the Xilinx's OPB (on-chip peripheral bus). We used Xilinx's Virtual Platform tools to simulate designs.

The soft-core parameters were:

- Instantiatable datapath units – The Microblaze has six on/off parameters: barrel shifter, floating-point unit, multiplier, divider, MSR (machine status register) instructions, and comparator. Each parameter could be set to on, which would instantiate the hardware unit, or to off, which would carry out the corresponding function in software.

Figure 1: Diagram of the Microblaze system being configured**.**



- Cache type – The Microblaze has two cache types for instruction and data caches: OPB (On-chip peripheral bus) cache and CacheLink cache. The OPB cache is the standard cache. The CacheLink cache is more advanced by using FSL (fast simplex link) technology to communicate with off-chip memory, as opposed to using the OPB, allowing other resources to use the bus. The CacheLink cache uses less BRAM (block RAM) for a given cache size at the expense of a higher LUT (lookup table) count.
- Cache size – Each cache's size may range from 0K to 64K. We considered sizes of 0K, 4K, 16K, and 64K.
- Instruction/data location – The instruction and data segments of an application can be stored in either of two memory locations: on-chip BRAM, or off-chip memory. When both instructions and data are stored off-chip, the size of the BRAM is reduced to 2K.
- Compiler optimizations – An application can be compiled using one of four optimization levels: none, O1, O2, or O3.

We define a *base Microblaze core* as a core configured with all instantiatable datapath units turned off, all caches off, instruction and data segments on chip, and no compiler optimization.

4.2 Equivalent LUTs

The term "size" must be qualified in the context of FPGAs. FPGAs contain different hardware resources, including lookup tables (LUTs), multipliers, and block RAM. However, having a single number for size is desirable during design optimization, so that two designs may be directly compared. The most straightforward method of creating a single size number is to combine the various hardware resource numbers of a particular design – number of LUTs used, number of multipliers used, bytes of block RAM used – using a weighted sum function. We use the equations shown in Figure 2, where weights for multipliers and block RAMs are determined from the total number or size of those items in a maximally configured soft-core, resulting in an "equivalent LUT" value. For configured Microblazes, we found this equivalent LUT approach to correlate almost perfectly (0.995) to the "equivalent gate" concept defined by Xilinx for general circuits. From this point forward, we use the term LUTs to refer to equivalent LUTs, unless otherwise stated.

Figure 2: Equations for calculating *Equivalent LUT* value of a configured MB System.

$$LUT_{Equivalent} = LUT_{Regular} + LUT_{Equivalent(Mult)} + LUT_{Equivalent(BRAM)}$$

$$LUT_{Equivalent(Mult)} = \#Mult_{Used} / \#Mult_{full\ MB} * LUT_{full\ MB}$$

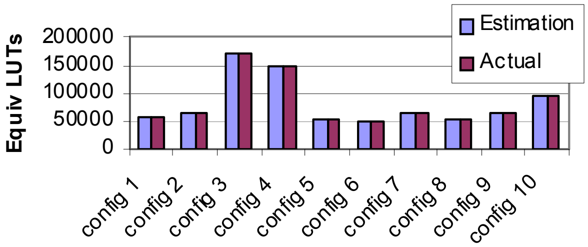
$$LUT_{Equivalent(BRAM)} = sizeBRAM_{Used} / sizeBRAM_{full\ MB} * LUT_{full\ MB}$$

4.3 Size Estimation

Although our DoE-based strategy aims to keep the number of soft-core configurations that must be synthesized to a minimum, we found it was still necessary to use some estimation to reduce the number of such synthesis runs during our own experiments. Synthesis time for one soft-core configuration can range from 10 minutes to over one hour, depending on the particular parameter

[1] Although in DoE, "costly" typically means costing thousands of dollars or requiring weeks or months of time, cost is a relative – in design automation, tens of minutes is costly.

Figure 3: Comparison between the actual LUTs and the estimated LUTs, showing less than 0.1% error in any case.



settings of the configuration. Running 20 or 30 configurations could thus require 1-2 days. In an attempt to reduce that time during our experiments, in which we were tuning ten distinct applications, we investigated the accuracy of adding incremental sizes (versus the base) of each parameter's hardware. In other words, if a multiplier adds X LUTs to the base core's size, and a divider adds Y LUTs, then we sought to determine the accuracy of estimating the size of a core having both a multiplier and divider by simply adding X + Y to the base core's size, rather than actually synthesizing a base core with a multiplier and divider.

For every relevant parameter, we synthesized a core configuration with just that parameter's hardware, and measured the size increment. Then, for 10 random core configurations, we synthesized those configurations, and compared the actual size to the size estimated by adding each parameter's size increment to the base core's size. Figure 3 shows that the estimation approach was very accurate, with an average error of only 0.03%, and with no configuration having an error higher than 0.1%. We thus have reasonable confidence that the estimation approach is valid.

4.4 Soft-Core Customization using DoE

We utilized the DOE Pro XL [3] tool for DoE. The tool allows a user to specify the number of factors (up to 27), the type of experimental design method (fractionalized factorial, Taguchi, etc.), and the number of experimental runs. The tool automatically generates a list of the experiments to run, showing the necessary factor settings of each experiment, as shown in Figure 5.

Like many DoE tools, the DOE Pro XL tool primarily supports factors with only two levels, due to two level factors enabling elegant statistical methods. The common technique used in DoE is thus to map multi-level factors to several two-level factors. For example, our cache size factor has four levels – 0K, 4K, 16K, and 64K. We map that factor to two two-level factors. The first two-level factor narrows down the cache size to either 0K or 4K (subset one), or 16K or 64K (subset two). The second two-level factor chooses between the smaller size or larger size cache in the subset chosen by the first factor. The four-level compiler optimization parameter can be mapped to two two-level factors similarly. Mapping all the soft-core parameters listed in Section 4.1 resulted in 16 two-level factors.

Not all factor combinations represent valid systems. We thus perform simple post-processing on the tool-generated experiments to detect and modify invalid systems.

The *base* system is the configuration where all the factors are set to the first of their two levels. This is the base configuration that we used later in an impact-ordered tree approach.

We chose to use a 20-run Plackett-Burman set of experiments, as the time for 20 runs corresponds to the approximate amount of time we believe to be acceptable in an FPGA soft-core tuning tool, namely about 2-3 hours, assuming each run requires about 5-10 minutes (note: the tool also could general 24 and 28 run experiments). We wrote a script to automatically run the 20 experiments specified by the tool and enter the results into the tool's table. The DOE Pro XL tool then analyzed those results automatically to generate the factor impact chart shown in Figure 4. That chart show that the instruction location factor has the biggest impact (67.42%), followed by first of the two instruction cache size factors (11.28%). We sorted factors by the ratio of impact/size, and generated an impact-

Figure 4: Factor impact chart output by DOE Pro XL.

ANOVA TABLE

			Cycles			
Source	SS	df	MS	F	P	% Contrib
I_LOC	86.3	1	86.3	169.626	0.001	67.42%
IC_SIZE_1	14.4	1	14.4	28.389	0.013	11.28%
DC_SIZE_1	7.1746	1	7.1746	14.106	0.033	5.61%
OPT_1	3.7225	1	3.7225	7.319	0.073	2.91%
BS	3.3171	1	3.3171	6.522	0.084	2.59%
MSR	3.2214	1	3.2214	6.334	0.086	2.52%
DC_SIZE_2	2.7988	1	2.7988	5.503	0.101	2.19%
IC_SIZE_2	1.8308	1	1.8308	3.600	0.154	1.43%
PCMP	1.5656	1	1.5656	3.078	0.178	1.22%
IC_TYPE	1.3164	1	1.3164	2.588	0.206	1.03%
OPT_2	0.4041	1	0.4041	0.794	0.438	0.32%
D_LOC	0.2017	1	0.2017	0.396	0.574	0.16%
DIV	0.0764	1	0.0764	0.150	0.724	0.06%
MUL	0.0678	1	0.0678	0.133	0.739	0.05%
DC_TYPE	0.0313	1	0.0313	0.062	0.820	0.02%
FPU	0.0001	1	0.0001	0.000	0.988	0.00%
Error	1.526	3	0.509			1.19%
Total	127.967	19				

Figure 5: Set of 20 runs generated by the DOE Pro XL tool.

Factor	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Row #	BS	FPU	MUL	DIV	MSR	PCMP	IC_TYPE	IC_SIZE_1	IC_SIZE_2	DC_TYPE	DC_SIZE_1	DC_SIZE_2	I_LOC	D_LOC	OPT_1	OPT_2
1	1	0	1	1	0	0	0	0	1	0	1	0	1	1	1	1
2	1	1	0	1	1	0	0	0	0	1	0	1	0	1	1	1
3	0	1	1	0	1	1	0	0	0	0	1	0	1	0	1	1
4	0	0	1	1	0	1	1	0	0	0	0	1	0	1	0	1
5	1	0	0	1	1	0	1	1	0	0	0	0	1	0	1	0
6	1	1	0	0	1	1	0	1	1	0	0	0	0	1	0	1
7	1	1	1	0	0	1	1	0	1	1	0	0	0	0	1	0
8	1	1	1	1	0	0	1	1	0	1	1	0	0	0	0	1
9	0	1	1	1	1	0	0	1	1	0	1	1	0	0	0	0
10	1	0	1	1	1	1	0	0	1	1	0	1	1	0	0	0
11	0	1	0	1	1	1	1	0	0	1	1	0	1	1	0	0
12	1	0	1	0	1	1	1	1	0	0	1	1	0	1	1	0
13	0	1	0	1	0	1	1	1	1	0	0	1	1	0	1	1
14	0	0	1	0	1	0	1	1	1	1	0	0	1	1	0	1
15	0	0	0	1	0	1	0	1	1	1	1	0	0	1	1	0
16	0	0	0	0	1	0	1	0	1	1	1	1	0	0	1	1
17	1	0	0	0	0	1	0	1	0	1	1	1	1	0	0	1
18	1	1	0	0	0	0	1	0	1	0	1	1	1	1	0	0
19	0	1	1	0	0	0	0	1	0	1	0	1	1	1	1	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6: Obtained application speedups when tuning a Microblaze soft-core using the Single Factor and the Design of Experiment approaches to build the impact ordered tree, with a size constraint of 10,000 LUTs**.**

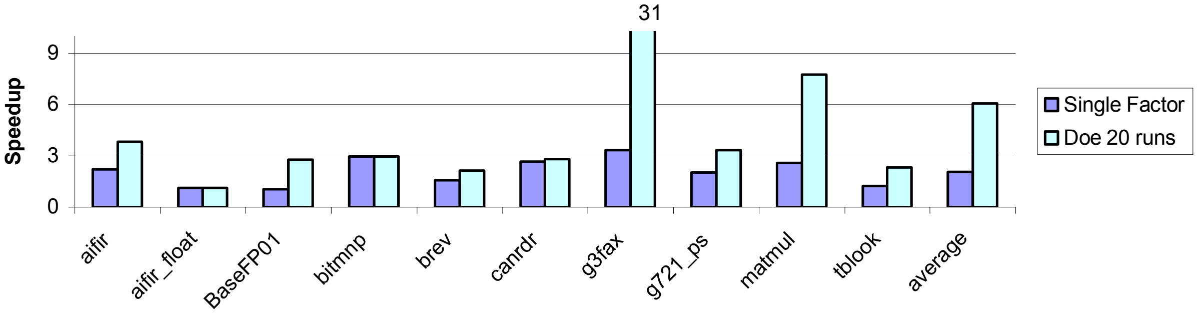
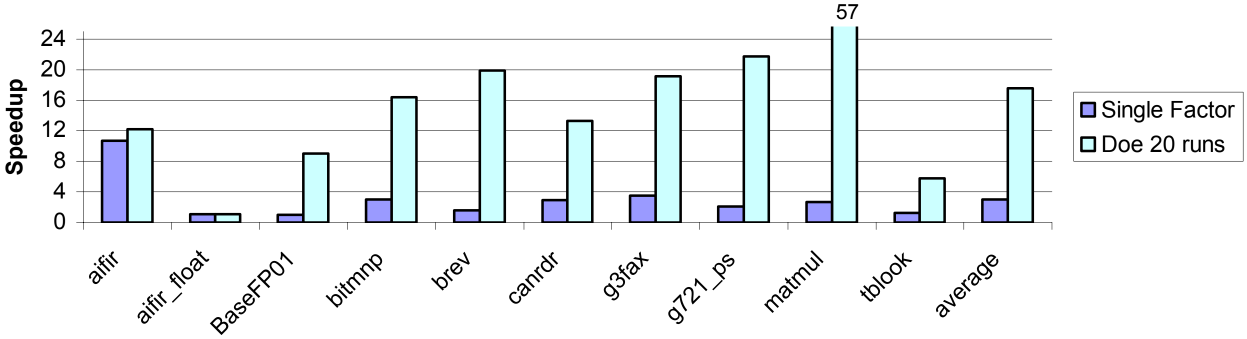


Figure 7: Obtained application speedups when tuning a Microblaze soft-core using the Single Factor and the Design of Experiment approaches to build the impact ordered tree, with a size constraint of 30,000 LUTs.



ordered tree based on those factors, as done in [12] (Note: for cache, we had to sometimes make one small departure from a strict impact ordering, to ensure that the decision to include a cache always preceded the decision of a cache's type). We then used a heuristic that descended the tree and picked the faster of the two configurations at a given level, subject to also satisfying a size constraint.

For comparison purposes, we also generated impact-ordered trees using our previous single-factor analysis method [12]. That approach considered only two-level parameters, and thus we applied that heuristic to our two-level factors.

The DoE approach required 37 runs per application (20 for the DoE part, and 17 for the impact-ordered tree part), while the single-factor analysis required 34 runs – roughly the same number for both approaches. Thus, the difference in upcoming results can be attributed mostly to quality of the search process, rather than to more runs by one approach or the other.

5. Results

5.1 DoE for Determining Impact Ordering

We ran the DoE and single-factor approach on 10 EEMBC benchmarks [4], for three size-constraint scenarios. Figure 6 shows the results comparing the Single Factor and the DoE approaches with a small size constraint 10,000 LUTs, while Figure 7 shows data for a size constraint of 30,000 LUTs. (A base Microblaze uses 7,376 LUTs). Results are shown in terms of the speedup obtained by running the application on a tuned core compared to running the application on a base core. The results show that, while both approaches yield speedups, the DoE approach vastly outperforms the single-factor analysis approach. Figure 6 shows on average that the Single Factor has an average speedup of 2x, while the DoE approach yields a 6x speedup; Figure 7's speedups are 3x versus 17x. Thus, DoE improves on a single-factor approach by 6/2=3x, and by 17/3=~6x.

We also obtained results with a 50,000 LUT size constraint. Because that size was large enough to support most optional core components, the approaches achieved the same speedups on all benchmarks.

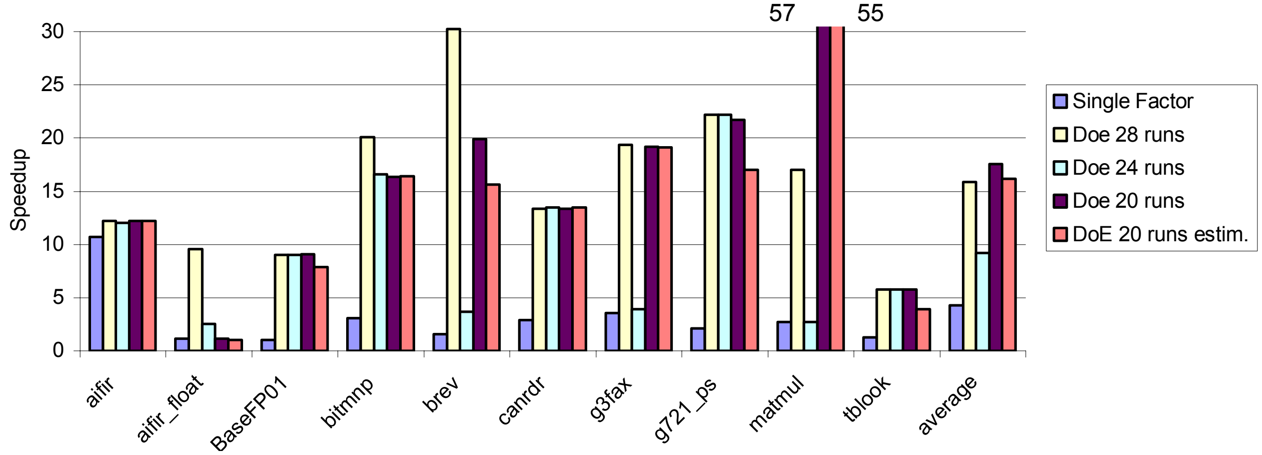
The success of the single-factor approach can depend strongly on selection of a good base system, and we were concerned that we might have chosen a poor base. We therefore modified the base case to place instruction and data segments off-chip (the most important factor in most application), and repeated the entire set of experiments. We found no significant difference in results compared to the results in Figure 6 and Figure 7. This latter analysis confirms that the DoE paradigm does not require the definition of a base system, instead considering multiple factors turned on (or at the high level) simultaneously.

We also generated DoE results using 24 and 28 run options provided by the DOE Pro XL tool. The data appears in Figure 8. We saw only mild improvements compared to the 20 run option, and in some cases slight worsening, due to the approaches being heuristic in nature

5.2 Using DoE's Built-In Tuning Approach

The previous section used DoE to determine the impact of each factor, and then used a tree-based heuristic to actually tune the parameter values. Interestingly, DoE tools often contain additional algorithms that further analyze the experimental data, and seek to predict the factor values that would predict the best output. These algorithms are based on sophisticated statistical

Figure 8: Application speedups obtained by the single-factor approach; by 20, 24, and 28-run DoE approaches for creating an impact-ordered tree; and by 20-run DoE approach using DoE's built-in statistical prediction of the best configuration.



prediction techniques whose details are beyond the scope of this paper[2]. Using the best predicted values by the DoE tool would eliminate the time required to run the impact-ordered tree tuning heuristic, thus reducing the total experiments from 37 down to just the 20 DoE-generated runs.

Figure 8 shows the speedups obtained using DoE's predicted best configuration ("estim."), compared to the DoE/impact-ordered tree/heuristic approach described earlier, for a 20-run DoE. The figure shows that the predicted configuration method is nearly as good as the impact-ordered tree heuristic. The single factor approach required 34 runs, the DoE 20-run approach followed by the tree heuristic required 37 runs, while the DoE with prediction approach required only 20 runs (with the prediction algorithm running in negligible time). This finding represents an additional benefit of using DoE for tuning parameterized soft-cores, namely that of about 40-45% faster tuning runtimes.

6. Conclusions

The well-established Design of Experiments paradigm can be exploited to tune a microprocessor soft-core to an application, yielding 6x-17x speedup compared to a base core. Those speedups are 3x-6x better than obtained by a previous non-DoE-based core tuning approach. The key benefit of DoE is the multi-factor analysis, proven to yield near-maximum information from a given small number of experimental runs. As modern technologies result in more parameterized components, applying DoE techniques to find the best parameter settings may become increasingly important in more areas of system-level design automation.

7. Acknowledgements

This work was supported in part by the National Science Foundation (CNS-0614957) and the Semiconductor Research Corporation (2005-HJ-1331), and by donations from Xilinx Corp.

References

[1] Arm http://www.arm.com.

[2] Altera Corp. Nios II Processors. http://www.altera.com/products/ip/processors/nios2/ni2-index.html, 2005.

[3] DOE Pro XL http://sigmazone.com/doepro_faqs.htm.

[4] EEMBC. http://www.eembc.org/, 2005.

[5] Givargis, T., F. Vahid. Platune: A Tuning Framework for System-on-a-Chip Platforms. IEEE Transactions on Computer Aided Design, Vol. 21, No. 11, Nov. 2002, pp. 1317-1327.

[6] Kumar, R., D. Tullsen, N. Jouppi. Core Architecture Optimization for Heterogeneous Chip Multiprocessors. International Conference on Parallel Architectures and Compilation Techniques, PACT, Seattle, April 2006.

[7] Kumar. R., D. Tullsen, P. Ranganathan, N. Jouppi, K. Farkas. Single-ISA Heterogeneous Multi-core Architectures for Multithreaded Workload Performance. In 31st International Symposium on Computer Architecture, ISCA-31, June 2004.

[8] McLean, R., V. Anderson, Applied Factorial and Fractional Designs. Marcel Dekker, Inc. New York, New York, 1984.

[9] Moyer, B., Tune *Multicore Hardware for Software*. *Xcell Journal*, Issue 58, 2006, pp 55-57.

[10] Petersen, R., *Design and Analysis of Experiments*. Mercel Dekker Inc. New York, New York, 1985.

[11] Sekar, K., Kanishka Lahiri, Sujit Dey. Dynamic Platform Management for Configurable Platform-Based System-on-Chips. Intl. Conf. on Computer-Aided Design (ICCAD), 2003.

[12] Sheldon, D., R. Kumar, R. Lysecky, F. Vahid, D. Tullsen. Application-Specific Customization of Paramaterized FPGA Soft-Core Processors. Intl. Conf. on Computer-Aided Design (ICCAD), 2006.

[13] Tensilica, Inc. The XPRES Compiler: Triple-Threat Solution to Code Performance Challenges. http://www.tensilica.com/pdf/XPRES-Triple-Threat_Solution.pdf, 2005.

[14] Xilinx, Inc. MicroBlaze Soft Processor Core. http://www.xilinx.com/, 2005.

[15] Yiannacouras, P., J. G. Steffan, J. Rose. Application-Specific Customization of Soft Processor Microarchitecture. FPGA 2006.

[16] Yiannacouras, P., J. Rose, J. G. Steffan. The Microarchitecture of FPGA-based soft processors International Conference on Compilers, Architecture, and Synthesis For Embedded Systems (CASES), 2005.

[2] And frankly, not entirely understood by this paper's authors. DOE Pro XL fully implements the statistical techniques, so it is not required that DoE users have a complete understanding – akin to a user of an integer linear program (ILP) solver not having a complete understanding of ILP solution methods.