"CERE": A CachE Recommendation Engine: Efficient Evolutionary Cache Hierarchy Design Space Exploration

Conference Paper · August 2014

DOI: 10.1109/HPCC.2014.97

2014 IEEE International Conference on High Performance Computing and Communications (HPCC), 2014 IEEE 6th International Symposium on Cyberspace Safety and Security (CSS) and 2014 IEEE 11th International Conference on Embedded Software and Systems (ICESS)

"CERE": a CachE Recommendation Engine

Efficient Evolutionary Cache Hierarchy Design Space Exploration

Gabriel Yessin*§, Abdel-Hameed A. Badawy*†¶, Vikram Narayana*‖, David Mayhew‡††, and Tarek El-Ghazawi* **
*NSF Center for High-Performance Reconfigurable Computing (CHREC), Electrical & Computer Engineering Department, The George Washington University, Washington, DC 20052
†Electrical Engineering Department, Arkansas Tech University, Russellville, AR 72801
‡College of Technology & Innovation, Department of Engineering, Arizona State University, Tempe, AZ 85281
§gyessin@gwu.edu, ¶abadawy@atu.edu, ‖vikram@gwu.edu, ††David.Mayhew@asu.edu, **tarek@gwu.edu

***Abstract*—Design Space Exploration is a critical step in chip design. Unfortunately, it takes significant amounts of time and resources to explore a fraction of the design space. Herein, we present a heuristic, evolutionary approach (Genetic Algorithm) to exploration that significantly cuts down on the time and resources, obtaining a near optimal design. We demonstrate the real-world utility of our tool-chain "CERE" by rapidly and efficiently designing the cache hierarchy which maximizes the performance of a web-browser navigating to a set of famous websites running on a single ARM core. We rapidly traverse 134,136 possible configurations. At about two days per simulation on the Gem5 full system simulator, the entire space would have taken 268,272 CPU-Days or ≃734 CPU-Years (≃3.7 Years on a cluster of a hundred dual core machines) to brute force search. "CERE" provided results in ≃4.5 days and used ≃17.5 CPU-Days on our cluster. We ran the configurations that "CERE" chose through gem5 to verify that "CERE" made the right choices and we were able to observe a 17.1% speedup going from the "best" hierarchy relative to the "Worst" hierarchy.**

I. Introduction

As we move into the digital age and get connected more and more, email, social media, e-commerce, and the Internet as a whole have become an integrated and integral parts of our routine in our day-to-day lives. We are connected all the time with our mobile devices and the ability to search for information and acquire knowledge on everything we encounter has never been more. The expectations of mobile users are to have fast real-time access to information through their mobile devices' browsers. Users get frustrated quickly if the information they are looking for takes more than a few seconds to load into their web-browser. According to a survey published by the US Department of Labor [13], the most commonly reported task for the 77 million workers who used a computer at work in October 2003 was accessing the Internet or using e-mail [13]. This means that the web-browser is one of the most used applications in any computer system especially in mobile environments. Research has shown that the average web page loading time experienced by users of the top 2000 websites is ≃10 seconds (median 8.4 seconds). PhoCusWright [35] market research and industry intelligence has found recently that the average computer user is often unwilling to wait for more than three seconds for a web page to load, with ≃57% of users abandoning a web page before the four second mark [35]. The performance expectations of the users can cost loss of business and limit the growth of e-commerce and other sectors of the mobile digital commerce [39]. Architecture designers can adapt the architecture of the new lines of mobile devices to cater to the users by improving the performance of browsing the web.

There has been a recent trend in computing with movement towards heterogeneous multi-cores, or more specifically, weakly heterogeneous multi-cores [40]. They are weakly heterogeneous in that they are identical in ISA and most major microarchitectural features, but vary in some key features. A key example of this architecture is NVIDIA's Tegra 3 and the Tegra 4's variable SMP architecture [23], [29]. The NVIDIA's 4-PLUS-1 architecture makes use of four high-power cores and a separate low-power companion core. The companion core is used to save on power when the system does not require the power of all four cores, such as for displaying already-rendered web-pages. The companion core is nearly identical to the other cores except that it runs at a much lower frequency and is made using a special low-power manufacturing process. The net result is performance equivalent to the high-power quad cores, but with less net power consumption at a cheaper cost to build and less silicon requirements [23], [29]. We are driven here to design the cache hierarchy of a single core of the multi-cores of a mobile device to specialize in web browsing.

Architectural studies have always relied on cycle-accurate simulators such as SimpleScalar [5], SESEC [34], Graphite [27], Simics [25], Gems [26], M5 [11], [12], and Gem5 [9]. Design space exploration studies have used cycle-accurate simulators as well [21], [40]. Searching the entire possible design space is normally prohibitive in terms of the time and resources it would take to solve such a problem. Design space exploration studies can be viewed as an optimization problem as we are after the architecture parameters that optimize the utilization of the silicon area of the chip to produce the best performance within a reasonable power budget and with the minimal cost to build the chip. Design space exploration to be effective and relevant, designers and researchers have to explore benchmarks that are realistic and useful in order to derive the next wave of architectures that will derive architectural research or new released hardware to the market.

Genetic Algorithm (GA) [17], [18] is an artificial intelligence technique that is bio-inspired and is used as a search heuristic. It mimics the process of natural selection. This heuristic is customarily used to solve optimization and search problems. Genetic algorithm is an evolutionary algorithm (EA), which generates solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. Researchers and engineers have used GA in various fields.

In GA, a population of valid solutions (individuals) to an optimization problem evolves toward better solutions. Each possible solution has a set of properties normally called "chromosomes" which can be altered to form newer individuals. The initial population is randomly generated. In each iteration, (an iteration is called a generation) the fitness of every individual in the population is evaluated. The fitness function is normally chosen as the value of the objective function being optimized. The better individuals in terms of fitness are stochastically selected from the current population, and each individual's chromosomes are modified via crossover and mutation to form new individuals and a new generation. The new generation is used in the next iteration of GA. The algorithm terminates when either a maximum number of generations (10 generations in our case) has been reached, or a good enough fitness value has been achieved before the maximum number of generations have been reached [17], [18].

Normally, GA can search huge spaces to find an optimal or near optimal solution to an optimization problem. One of the challenges of GA is to find a good and suitable fitness function. In our evaluation of cache hierarchies we use the average memory access time (AMAT) as the fitness function for GA. We used cache simulations coupled with CACTI [38], [41]. CACTI is an integrated cache and memory access time, cycle time, area, leakage, and dynamic power model. Thus, CACTI estimates cache access times to determine the AMAT of a particular cache hierarchy. It requires only $\simeq$0.7% of the time it would take a full system simulation to determine execution time. It would also require only a few megabytes of memory compared to the greater than 4 GBs of memory a full system simulation would require, reducing search time significantly while utilizing computational resources far more efficiently.

Hardware design can lend itself to be a very suitable candidate for GA as the heuristic we can use to determine the best architecture parameters in an optimization problem. We formulate our design space optimization problem to be solved using synergistic harmony of several different tools to search the huge space we want to explore without spending excessive time and resources. We utilize GA to efficiently find the solution to build a specialized general purpose processor that is optimized for web-browsing. As of submitting this paper, we know of no user satisfaction-oriented design space exploration parameter studies that use GA and cycle-accurate architectural simulators.

The contributions of this paper are as follows:

1) We specialize the cache hierarchy of a single core ARM (embedded processor) to be optimized for web browsing to meet modern day.
2) We rely on full system cycle accurate simulations for verification purposes only.
3) We are able to explore the design space in a mere 1% of the time it would have taken to using brute force search techniques to find the best solution.
4) We show the utility of GA and AMAT as a fitness function in design space exploration in a very large design space.

The rest of this paper is structured as follows: Section II provides a summary of the most relevant works that our work builds upon and is most similar to; section III discusses the methodology used for the generation of simulations and for the collection and analysis of results; section IV we show and discuss the results collected for the various architectures, and section V we summarize the paper's conclusions.

II. Related Work

This work builds up on the work in [45]. In this paper, we explore a much larger design space. We depend on Artificial Intelligence techniques (GA) to efficiently and quickly search the design space to identify the optimal or near optimal configuration for the cache hierarchy. We also dropped using Google and Craigslist from the websites we run in Bbench [20]. Both were trivially small compared to the others. In [45], we have shown that the system performance highly correlates to AMAT (average difference of 11.54%, standard deviation of 10.92%) on almost the same set of benchmarks but with a much smaller design space. The cache simulation takes approximately 15 minutes per configuration on our cluster, with minimal memory usage, whereas detailed performance simulations take about 2 days each and about 4GB of memory. This represents a 192x speedup without much loss of accuracy for our purposes. We rely on this finding in that paper to utilize AMAT as the fitness function for our GA.

Design Space Exploration (DSE) Studies: Design Space Exploration is a daunting task. Exploring significant design spaces using full system simulations is almost impossible. Some of the previous research involving design space exploration has only looked into the effects of varying only one parameter, was limited to the running of synthetic benchmarks (which are of little to no importance to users and user-satisfaction), and/or testing out new architectural parameters [8], [46]. This shows that research with regards to leveraging the full capabilities of performance modeling in modern architecture simulation infrastructure is not easily doable. Modern architecture simulators support a lot of interesting features that allow faithful simulation of the system *e.g.* full system simulation (*i.e.* operating system interactions). This is especially true with regards to Gem5 [9], the simulator used in this study. Gem5 has been proven to be accurate enough for our purposes in this paper [14]. Yet, we recognize the importance of utilizing these simulator yet we also recognize that it is hard to do that for significant workloads nor for a lot of design point.

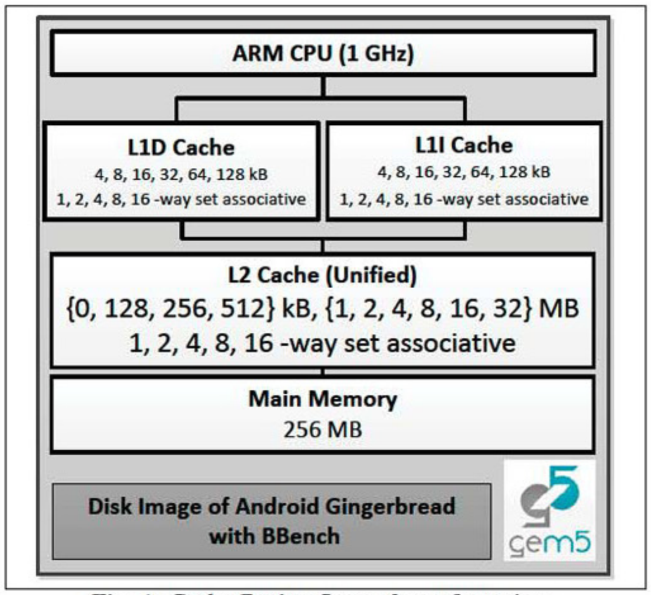
AMAT for DSE: Donald Yeung and his students have explored the design space for many-core architectures using their reuse distance profiling framework [43], [44]. In another study, they used reuse distance profiles to explore the design of caches for a multi-core processor [42]. The advantage of this framework is that it does not rely on detailed architectural simulators to be able to predict the performance under core count scaling, problem scaling in an architecture agnostic fashion. In [42], they built a performance prediction model using AMAT computed from the cache miss rates at each cache size computed from the reuse distance profiles whether it is private or shared. In [6], they have used a model similar to the one in [42] to chose optimizations that require profiling. The choices made by the reuse distance based model matched with good accuracy the results obtained from a brute force search of the possible space using detailed simulations using the M5 simulator.

Our work is similar to theirs in the use of AMAT. We use AMAT as the fitness function in our GA to explore the design space. Reuse distance profiles are idealized in how they compute AMAT. They cannot accurately compute cache conflicts except through a Qasem and Kennedy cache conflict model [33]. Also, the effects of cache associatively cannot be accounted for accurately since reuse distance assumes a fully associative cache, whereas we use Dinero IV [16] to predict the cache performance, where cache associativity behavior is taken into account. One major advantage of the reuse distance modeling is that with one single run, we can compute the reuse distance profile showing the entire space for all cache sizes and using a model we can compute the AMAT at any given cache design point. On the other hand, we have to run a separate cache simulation for each cache configuration GA is considering. We will consider in our future work using reuse distance profiles to compute the fitness function for GA using the AMAT model of reuse distance profiles and comparing its performance to Dinero IV as is in our framework here.

GA for DSE: Genetic Algorithm (GA) has been heavily used for search and optimization problems. We are not the first to see the utility of GA for architecture space exploration. It has been used in many architectural studies. In particular, researchers have used GA for multi-objective design space exploration (DSE) for System-on-Chip (SoC) architectures [3], [30], for application specific processors (ASP) [37], for system-level Multi-Processor SoC (MP-SoC) [22], for embedded computer systems [2], and for multimedia embedded systems [7].

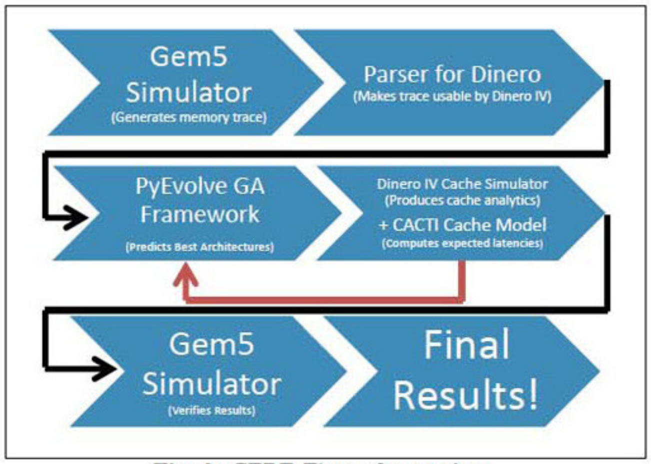
Ascia, Catania, and Palesi [4] have investigated a GA design space exploration framework for parameterized system-on-a-chip platforms. They use the Platune simulator. We use an embedded system processor (ARM), which is more relevant to the mobile market we are targeting. They only simulate one level of cache on a MIPS RISC processor running embedded windows. They also showed result for a VLIW processor which does have two-levels of cache. Their benchmarks, while valid, are very small, targeted ones, *e.g.* jpeg compression, they do not target a large application like we do (web browser running real, popular websites). They calculated the Pareto front.

Gordon-Ross, Vahid, and Dutt [19] use a heuristic approach to tune two-level caches, but they did not use CACTI to compute the cache latency. They estimate the latency and it stays constant for different cache configurations. Also, they did not use a real applications. They do discuss the power savings that can be had by properly designing the cache levels.



ARM CPU (1 GHz)
L1D Cache
4, 8, 16, 32, 64, 128 kB
1, 2, 4, 8, 16 -way set associative
L1I Cache
4, 8, 16, 32, 64, 128 kB
1, 2, 4, 8, 16 -way set associative
L2 Cache (Unified)
{0, 128, 256, 512} kB, {1, 2, 4, 8, 16, 32} MB
1, 2, 4, 8, 16 -way set associative
Main Memory
256 MB
Disk Image of Android Gingerbread with BBench
gem5

Fig. 1: Cache Design Space & configuration.



Gem5 Simulator
(Generates memory trace)
Parser for Dinero
(Makes trace usable by Dinero IV)
PyEvolve GA Framework
(Predicts Best Architectures)
Dinero IV Cache Simulator
(Produces cache analytics)
+ CACTI Cache Model
(Computes expected latencies)
Gem5 Simulator
(Verifies Results)
Final Results!

Fig. 2: CERE Flow of operation.

III. Methodology

In this section, we discuss the collection of tools we have synergistically combined to create the framework of "CERE". We discuss the design space to be explored, the tools, and the benchmarks we used in this paper. The methodology here is the same as the methodology used in [45] with the exception of the GA tools and usage alongside the much larger design space that we explore here.

A. The Design Space

In this paper, we examine the cache hierarchy design space for a single arm core. Our goal is to specialize this core to run the web-browser as efficiently as possible to meet the needs of the users. We explore a two level cache hierarchy. The L1 cache is a split cache with an instruction L1 cache (IL1) and a data L1 cache (DL1). The size of each is in powers of two from 4kB to 128kB. The cache sizes we explore are as follows: 4kB, 8kB, 16kB, 32kB, 64kB, and 128kB (*i.e.* 6 different sizes each for IL1 and DL1). The second level cache size is also in powers of two from 128kB to 32MB. We also include the case of no second level cache as well. The cache sizes we explore are as follows: 0kB, 128kB, 256kB, 512kB, 1MB, 2MB, 4MB, 8MB, 16MB, and 32MB (*i.e.* 10 different sizes for the unified L2 cache). We consider cache associativities from direct-mapped to 16-way set associative caches with a factor of 2 increments. Namely, we explore the following cache associativities 1, 2, 4, 8, and 16 (*i.e.* 5 associativities for each of the caches in the system). The associativity of each cache is independently varied irrespective of the associativity of other caches in the hierarchy. This cache design space represents a total of 33,534 possible unique cache configurations that are to be explored either exhaustively or through heuristics. Note that associativity does not matter at all with a 0kB L2 cache. Figure 1 shows the architecture being simulated and the different parameters for each cache. Note that when the L2 cache size is 0kB, this means that we will not have an L2 cache in the system and the L1 caches will be connected directly to the main memory of the system.

B. The simulation Infrastructure

We use Gem5, a full system simulator [9], [10] that runs a full fledged operating system. Our simulations were run on the ARM CPU architecture provided by Gem5 running Android OS 2.3 (Gingerbread) and the native browser provided therein. Simulations were carried out using the detailed model, which models a modern, Out-of-Order ARM processor core running at 1 GHz core frequency [36]. We chose ARM due to its greater than 95% dominance in the smartphone market, 10% in Mobile Computers, and 35% in digital TVs and set-top boxes as of 2010 [28]. Android OS 2.3 (Gingerbread) is used for several reasons. Currently it is the second most used Android OS in the market. It is running on 19% of the devices running Android [1]. Android's market share is 75% of all smartphones worldwide [24]. Furthermore, Android 2.3 (Gingerbread) is the most stable Android OS that we were able to run on the simulator (Gem5). The Android 3.X (Ice Cream Sandwich) is not as stable on Gem5. We do not expect the results nor the conclusions will change if we use a different version of the Android OS.

Figure 2 outlines the flow among the different tools we use in this paper. We use Gem5 for several purposes. Gem5 provides a detailed full system Android over ARM architectural simulation. It runs the Bbench web-browsing benchmarks [20]. Gem5 generates the memory traces for each of the websites we run from the Bbench suite. These memory traces are fed through a preprocessing Perl script to make the trace usable by Dinero IV [16]. We use the PyEvolve open-source evolutionary framework which provides us with a generic extensible framework for GA implementation [31], [32].

The major part of what GA needs is the fitness function which is used to determine how fit the individuals relative to each other. We use AMAT as the fitness function as we have explained in Section II and as Figure 3 suggests. We will explain this figure more in the next section. Dinero IV generates the cache statistics such as miss rates *etc.* for each cache hierarchy that the GA framework considers. To compute a realistic AMAT value, we need to compute accurately the access latency for each cache the framework considers. We use CACTI [38], [41] to determine the latency for accessing each cache hierarchy that GA considers. Note that, the individuals in each generation are not different all the time. A non-trivial number of individuals will be the same across generations. For these individuals, we do not need to run neither the cache simulation, the CACTI computation, nor the fitness function computation which would cut significantly on the computational time to find the optimal solution on top of the savings we obtain from not having to exhaustively search the entire space nor to use Gem5 to evaluate fitness.

The cycle of GA, Dinero IV and CACTI repeat until a stopping condition is reached. The stopping condition is either reaching a maximum number of generations or reaching a satisfactory solution. Gem5 is used to run a handful of full system simulations to verify the performance of the selected best designs.

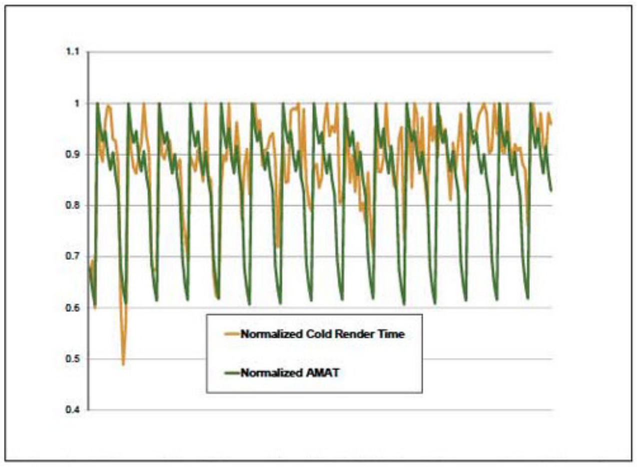


Fig. 3: Comparing Normalized AMAT to Normalized Render Time of Websites.

TABLE I. RANKING OF BENCHMARK WEBSITES ON ALEXA.COM GLOBALLY, IN THE US AND AMONG THE TOP 500

Website	Amazon	Ebay	MSN	Twitter
Global Rank	12	24	33	8
US Rank	5	8	25	9
Top 500	10	22	34	8

The parameters of GA that we used in this study are as follows:

1) Rank Selection:
There are many methods for selecting the best individuals, for example rank selection, roulette wheel selection, Boltzman selection, tournament selection, elitism, and some others [18]. Rank selection works by first ranking the population and then every individual receives fitness from the ranking. The worst will have fitness 1, second worst 2 *etc.* and the best will have fitness N (equal to number of individuals in the population). All individuals will have a chance to be selected. We chose this selection methods because of that. But this method can suffer from slower convergence;

2) Number of Generations is 10;
3) Number of individuals per generation is 60;
4) Mutation Rate is 10%
   Mutation is a genetic operator used to maintain genetic diversity from one generation to the next [18].
5) Crossover rate is 90%.
   Crossover is a genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next [18].

In trying to find what parameters to use GA, we resorted to using what other researchers have used in prior works that used GA. We increased the number of individuals per generation from 50 to 60 to be more conservative.

C. Benchmarks

Bbench was chosen because it is a fully self-contained web-rendering benchmark that represents many of the popular websites existing today [20]. Of the sites in Bbench, we chose to run Amazon, eBay, MSN, and Twitter, which are number 10, 22, 34, and 8 respectively on Alexa.com's top 500 list of the world's most popular websites in the past three month as of April 2014. Table I shows the ranking of each of the websites we run globally, in the US, and among the top 500 most visited websites. These websites represented a good cross-section of the type of websites available on the web representing E-commerce, news, and social media.

Due to the fact that the results of Bbench are generated in-browser and due to sandboxing of the Android browser for security reasons, the results could only be gathered through analyzing the simulator's snapshots of the frame buffer output. This necessitate using some automation techniques to determine the render time of each website. We use the same methodology as in [45] to acquire the results.

With 4 benchmarks (4 websites), the whole space evaluated comes to $33,534 \times 4$ (*i.e.* $134,136$) potential data points. At about 2 days per simulation, this would have take 268,272 CPU-Days or $\simeq$734 CPU-Years ($\simeq$3.7 Years on a 100 node dual core cluster) to get the full simulation results for the entire space.

IV. Experimental Results

In this section, we will present the results. We will quantify the advantages that GA achieves. We will compare that the estimated time of exploration of the space using brute force methods and the evolutionary techniques presented here.

AMAT correlation to Performance:
While AMAT and fitness scores (fitness score is derived from AMAT) are not perfect estimators of runtime, we will show below that they are a very reasonable estimator for the system runtime to chose among different cache hierarchies. AMAT can be a fair comparison point between competing architectures especially when the caches are the major difference between the architectures.

Figure 3 shows the relative performance of normalized AMAT in comparison to normalized render times for the Bbench websites and the cache hierarchy design space of [45]. The system performance is highly correlated to AMAT. This graph shows an average difference of 11.54% with a standard deviation of 10.92%. We rely on this result as the basis for using AMAT as the fitness function for the GA framework to save us time without sacrificing a lot on the accuracy of the estimates of performance it represents. Note that we intentionally eliminated the legends on the X-axis of the figure in order to display the graph but it would look like the X-axis of Figures 5 and 6.

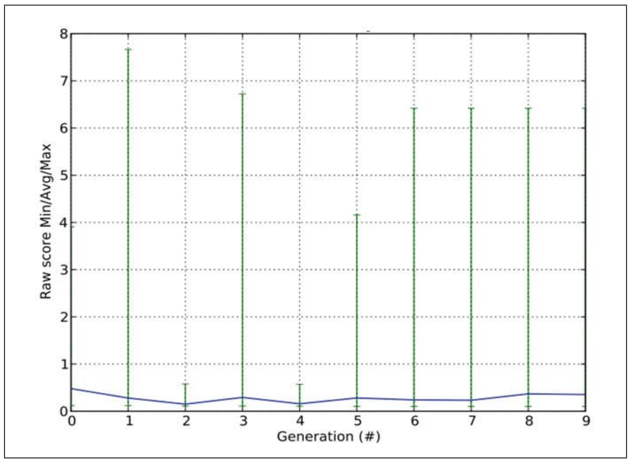


Fig. 4: GA Raw Fitness Minimum, Maximum, and Average for each Generation.

GA Results:
Figure 4 shows a snapshot of the 10 GA populations we ran. It shows that moving from generation 0 to 9, average scaled population fitness went from 0.93 to 0.15 (lower is better). Additionally, the worst 5 individuals in history existed in generation 0 (initial population). The best 5 individuals existed in generations 4, 5, and 6. The gene pool slightly worsened after generation 6. This is blamed on a combination of mutations and some poor selections. We could bias the selection process by forcing the best individuals from this generation to transfer through to the next generation. This is known in the literature for GA as elitism in selection [15].

Figure 5 shows the performance results of the best five individuals and the worst five individuals that GA examined. Due to the fact that GA evaluates each individual according to the fitness function used which is AMAT in this work, we notice in this figure that the top five "Best" and "Worst" architectures are not exactly ordered according to their performance. They are ordered according to how GA sees their respective fitness values. The X-axis of Figures 5 and 6 encode the architecture parameters for the cache being simulated as follows: "IL1 Cache Size" : "IL1 Associativity" : "DL1 Cache Size" : "DL1 Associativity" : "L2 Cache Size" : "L2 Associativity".

The results reported in Figure 5 are our verification data that GA really picks good choices and can identify the best hierarchies even though it never ran any full system simulations. In general, the top 5 individuals performed fairly better than the worst 5 individuals when evaluated using the Gem5 simulator. The average runtime of the 5 individuals with the worst fitness was $1950.95ms$ and the average of the best 5 individuals was $1813ms$, giving an average of 6.5% decrease in runtime, the true worst runtime to true best runtime of the Gem5 simulations was $2661ms$ to $2206ms$ for a 20.6%

decrease in runtime. We also note that the top choice of GA is really the 2nd best choice and it is within less than 2.2% of the true best runtime. It is also interesting that the best performing architecture among the worst architectures according to GA would rank fourth among top 5 best architectures. Among the best 5 architectures, the best and worst architectures are within less than 10% of each other. Likewise, among the top worst 5 architectures, the best and worst among them are within 12% of each other.

There are some very interesting observations. First, the top 5 best architectures share the L2 cache size of 128kB which is basically, the smallest non-zero L2 cache size that we examine in our design space. This suggests that the web-browser does benefit from an L2 cache but it does not require the biggest cache. It rather benefits more from a smaller and faster cache at the L2 level. Second, we observe that the associativity of L1 caches both instruction and data (IL1 and DL1) is fixed at one. The best level first level caches for our web-browser workloads are direct mapped caches. Third, we observe that the first level data caches are kept constant in size at 4kB direct mapped cache for all the top 5 best architectures. The first level instruction caches though varied from 4kB to 16kB but the top three best architectures were all fixed at 16kB direct mapped IL1.

Another way to think of these results and put them in perspective is that the top three best architecture are of the exact same hierarchy except for the associativity at the L2 cache where it varied from direct mapped to 2-way to 4-way set associative caches. The hierarchy was configured with a 16kB IL1 direct mapped first level instruction cache, a 4kB DL1 direct mapped first level data cache, and a 128kB L2 cache and the associativity of the L2 cache varied from direct mapped to 2-way to 4-way set associative caches.

Figure 6 shows the performance results of the five best and five worst individuals similar to Figure 5 with the addition of a set of four design points that represent some intuitive cache hierarchies. We chose design points with maximum cache sizes for each cache level (128kB IL1, 128kB DL1, and 32MB L2) to represent the conventional wisdom of bigger is better. We chose three cache associativities of 1 (*i.e.* direct mapped caches), 8, and 16 (*i.e.* maximum associativity simulated in this study) for all levels of cache. This gives us three design sizes with the maximum possible cache size that are direct mapped, with 8 and 16 associativities respectively. From the results in [45] we chose to use the case with no L2 cache *i.e.* 0 bytes of L2 cache and the smallest L1 instruction and data caches since these would hypothetically represent the fastest caches to access especially with direct mapped associativity. We note that the no L2 cache is not a good solution at all counter to the conclusions in [45]. This particular no L2 cache design point is ≃40% worse than the best design point. We also note that the largest caches that we considered here even though they did not compete on the best cache designs yet the best among them is only ≃13% worse than the best design point overall.

Our tool-chain "CERE" provides results in ≃4.5 days. It used ≃17.5 CPU-Days. The breakdown of this time is as follows: a two day period for Gem5 simulations, half a day for running GA and evaluating an individual's fitness takes ≃20 minutes. Multiple evaluations can be simultaneously obtained on multi-core machines with little additional overhead. We use previously computed fitness evaluations for repeated architectures to save even more time. Also, we have to add at the beginning of all of this work the time it takes to run Gem5 to obtain the memory traces for each Bbench website we run. We have to note that this is a one-time cost that does not repeat and fortunately, it is independent of the cache hierarchy parameters and is only application specific.

All in all, "CERE" evaluated 212 unique architectures. This represents a mere 0.63% of the entire design space we have considered. Simulating these 212 architectures would have taken ≃1696 CPU-Days or ≃4.7 CPU-years. Unless, we have a huge server farm, we cannot investigate this design space via full system simulations to provide the fitness evaluation. The heuristic approach have given us a reduction in resources required of about 99% without sacrifice on accuracy or quality of the solution.

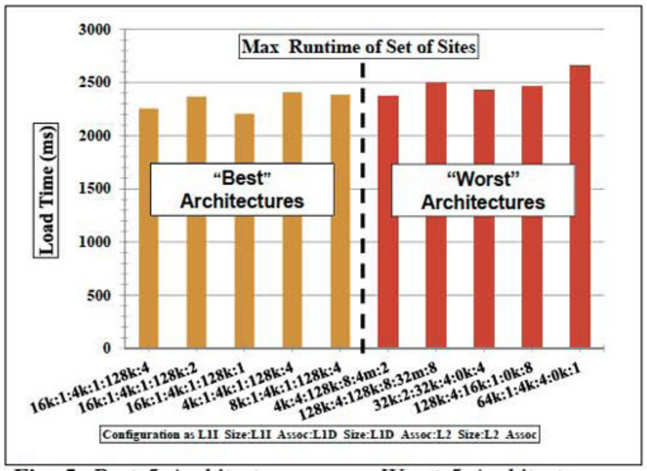


Fig. 5: Best 5 Architectures versus Worst 5 Architectures as GA predicted.

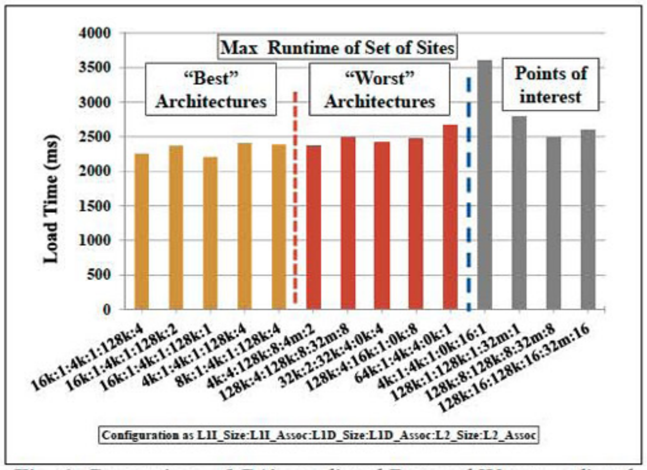


Fig. 6: Comparison of GA's predicted Best and Worst predicted Architectures versus Conventional wisdom.

V. Conclusions & Future Work

We have presented an evolutionary framework and a toolchain "CERE" that can explore a huge design space composed of thousands of design possibilities while skipping the lengthy and costly full system architectural simulation. We use a mere 1% of the resources that we would have used if we did not use AMAT as the fitness function for GA. "CERE" allowed us to examine less than 1% of the design points that we would have otherwise needed to examine exhaustively. The "CERE" computed best architectural choice for the cache hierarchy design performed within 2% of the true best choice. We can safely conclude that bigger is not always better. We also can safely conclude that cache hierarchy cycle times that are commonly used in architectural studies that are not computed using CACTI can lead to potentially inaccurate and unrealistic results.

By all means, we have looked at a small design space even though the design space is comprised of more than 134,000 possible design points. In the future, we would be interested in looking at a much larger space by including more benchmarks, larger design space and using reuse distance profiles to estimate the AMAT instead of using Dinero IV cache simulations.

We would also be interested in Parallel GA algorithm variations which would cut on the time it takes to run GA and thus we can afford to run it for a larger number of generations and larger population size. We would also consider designs of multi-core cache hierarchies for embedded as well as non-embedded systems. Also, a direct extension of this work is to consider the web-browser benchmarks (Bbench) behavior on desktop and laptop systems and use that to drive the design of the cache hierarchies for such systems.

We think that we should do a sensitivity study on how the accuracy of the solutions predicted using GA would change with changes in the GA parameters. Also, we might look into finding the optimal solution for the particular problem and find how close or far is our GA predicted solution to the real optimal solution. This might take too long since it would require exhaustively running full system simulations for all design points.

We believe that using the power consumption estimates, that CACTI emits with the latency statistics of each cache hierarchy, as another objective optimization function with the performance numbers can lead to better design choices. We can optimize for the Energy Delay product or the Energy Delay Squared product as the fitness function for GA.

Acknowledgment

We would like to thank the reviewers for their valuable suggestions and comments. This work was supported in part by Advanced Micro Devices, Inc. and the I/UCRC Program of the National Science Foundation under Grant Nos. IIP-0706352 and IIP-1161014.

References

[1] Android Developers, "Dashboards," April 2014. [Online]. Available: http://developer.android.com/about/dashboards/index.html

[2] G. Ascia, V. Catania, A. G. Di Nuovo, M. Palesi, and D. Patti, "Performance evaluation of efficient multi-objective evolutionary algorithms for design space exploration of embedded computer systems," *Applied Soft Computing*, vol. 11, no. 1, pp. 382–398, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494609002427

[3] G. Ascia, V. Catania, and M. Palesi, "An evolutionary approach for pareto-optimal configurations in soc platforms," in *SoC Design Methodologies*. Springer, 2002, pp. 157–168. [Online]. Available: http://link.springer.com/chapter/10.1007/978-0-387-35597-9_14

[4] ——, "A ga-based design space exploration framework for parameterized system-on-a-chip platforms," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 4, pp. 329–346, 2004. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1324695

[5] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling," *IEEE Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[6] A.-H. A.-S. Badawy, "Locality transformations and prediction techniques for optimizing multicore memory performance," Ph.D. dissertation, University of Maryland, College Park, August 2013. [Online]. Available: http://drum.lib.umd.edu/handle/1903/14781

[7] C. Baloukas, J. L. Risco-Martin, D. Atienza, C. Poucet, L. Papadopoulos, S. Mamagkakis, D. Soudris, J. Ignacio Hidalgo, F. Catthoor, and J. Lanchares, "Optimization methodology of dynamic data structures based on genetic algorithms for multimedia embedded systems," *Journal of Systems and Software*, vol. 82, no. 4, pp. 590–602, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121208002045

[8] M. D. H. Basu and M. M. Switch, "Reducing Memory Reference Energy with Opportunistic Virtual Caching," in *the 39th Annual International Symposium on Computer Architecture*, Portland, OR, 2012.

[9] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Computer Architecture News*, vol. vol - 39, pp. pp – 1 – 7, May 2011.

[10] ——, "The gem5 simulator," *SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, Aug 2011. [Online]. Available: http://doi.acm.org/10.1145/2024716.2024718

[11] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The M5 simulator: Modeling networked systems," *IEEE Micro special issue on Computer Architecture Simulation*, July/August 2006.

[12] N. L. Binkert, E. G. Hallnor, and S. K. Reinhardt, "Network-oriented full-system simulation using M5," in *the 6th Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, February 2003.

[13] Bureau of Labor Statistics, US Department of Labor, "Most common uses for computers at work," Sept 2005. [Online]. Available: http://www.bls.gov/opub/ted/2005/aug/wk5/art05.htm

[14] A. Butko, R. Garibotti, L. Ost, and G. Sassatelli, "Accuracy evaluation of GEM5 simulator system," in *the 7th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, York, UK, 2012.

[15] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=996017

[16] J. Edler and M. D. Hill. (1998) Dineroiv trace driven uniprocessor cache simulator. Online. [Online]. Available: http://pages.cs.wisc.edu/~markhill/DineroIV/

[17] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine learning*, vol. 3, no. 2, pp. 95–99, 1988.

[18] ——, *Genetic algorithms in search, optimization, and machine learning*. Addison-wesley Reading Menlo Park, 1989, vol. 412.

[19] A. Gordon-Ross, F. Vahid, and N. Dutt, "Automatic tuning of two-level caches to embedded applications," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 1, Feb 2004, pp. 208–213 Vol.1.

[20] A. Gutierrez, R. Dreslinski, T. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver, "Full-system analysis and characterization of interactive smartphone applications," in *2011 IEEE International Symposium on Workload Characterization (IISWC)*, Nov 2011, pp. 81–90.

[21] L. Hsu, R. Iyer, S. Makineni, S. Reinhardt, and D. Newell, "Exploring the cache design space for large scale CMPs." *SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 24–33, 2005.

[22] Z. J. Jia, A. D. Pimentel, M. Thompson, T. Bautista, and A. Núñez, "Nasa: A generic infrastructure for system-level mp-soc design space exploration," in *Embedded Systems for Real-Time Multimedia (ESTIMedia), 2010 8th IEEE Workshop on*. IEEE, 2010, pp. 41–50. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5666979

[23] Kingsley-Hughes, "Nvidia Tegra 4 processor details leaked — ZDNet," 2014. [Online]. Available: http://www.zdnet.com/nvidia-tegra-4-processor-details-leaked-7000008961/

[24] R. Llamas, K. Restivo, and M. Shirer, "Android Marks Fourth Anniversary Since Launch with 75.0 percent Market Share in Third Quarter, According to IDC," Nov. 2012. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS23771812#.UR3XBaXBOSo

[25] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, and G. Hallberg, "SIMICS: A full system simulation platform," *IEEE Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[26] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset." *SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 92–99, 2005.

[27] J. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," in *2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA'10)*, 2010, pp. 1–12.

[28] T. P. Morgan, "ARM Holdings eager for PC and server expansion," Online, February 2011. [Online]. Available: http://www.theregister.co.uk/2011/02/01/arm_holdings_q4_2010_numbers/

[29] NVIDIA Corporation, "Variable SMP (4-PLUS-1 (TM)) - A Multi-Core CPU Architecture for Low Power and High Performance," 2011.

[30] M. Palesi and T. Givargis, "Multi-objective design space exploration using genetic algorithms," in *Proceedings of the Tenth International Symposium on Hardware/Software Codesign (CODES'2002)*. IEEE, 2002, pp. 67–72.

[31] C. S. Perone, "Pyevolve: a python open-source framework for genetic algorithms," *ACM SIGEVOlution*, vol. 4, no. 1, pp. 12–20, 2009.

[32] ——. (2009, May) Pyevolve v0.5. Online. [Online]. Available: http://pyevolve.sourceforge.net/

[33] A. Qasem and K. Kennedy, "Evaluating a model for cache conflict miss prediction," Technical Report CS-TR05-457, Department of Computer Science, Rice University., Houston, TX, Tech. Rep., July 2005. [Online]. Available: http://symposium.cs.txstate.edu/papers/tr05.pdf

[34] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos, "SESC simulator," January 2005, http://sesc.sourceforge.net.

[35] C. Rheem, "Consumer Response to Travel Site Performance," *PhoCusWright*, 2010.

[36] A. Saidi, "ARM Implementation - gem5," May 2011. [Online]. Available: http://www.m5sim.org/ARM_Implementation

[37] A. Sengupta, R. Sedaghat, and Z. Zeng, "Multi-objective efficient design space exploration and architectural synthesis of an application specific processor (asp)," *Microprocessors and Microsystems*, vol. 35, no. 4, pp. 392–404, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141933111000366

[38] P. Shivakumar and N. P. Jouppi, "Cacti 3.0: An integrated cache timing, power, and area model," Technical Report 2001/2, Compaq Computer Corporation, Tech. Rep., 2001. [Online]. Available: ftp://gatekeeper.pa.dec.com/gatekeeper/pub/compaq/WRL/research-reports/WRL-TR-2001.2.pdf

[39] Strangeloop Networks Inc., "2012 Annual State of the Union: E-Commerce Page Speed and Website Performance," Jan 2012. [Online]. Available: http://www.strangeloopnetworks.com/assets/PDF/downloads/2012-Annual-State-of-the-Union-Report.pdf

[40] E. Tomusk and M. O'Boyle, "Weak Heterogeneity as a way of Adapting Multicores to Real," in *the 3rd International Workshop on Adaptive Self-tuning Computing Systems*, Edinburgh, 2013.

[41] S. J. Wilton and N. P. Jouppi, "Cacti: An enhanced cache access and cycle time model," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 5, pp. 677–688, 1996. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=509850

[42] M.-J. Wu and D. Yeung, "Identifying optimal multicore cache hierarchies for loop-based parallel programs via reuse distance analysis," in *the 2012 ACM SIGPLAN Workshop on Memory Systems Performance and Correctness*, ser. MSPC '12. New York, NY, USA: ACM, 2012, pp. 2–11. [Online]. Available: http://doi.acm.org/10.1145/2247684.2247687

[43] ——, "Efficient reuse distance analysis of multicore scaling for loop-based parallel programs," *ACM Transaction on Computer Systems*, vol. 31, no. 1, pp. 1:1–1:37, Feb 2013. [Online]. Available: http://doi.acm.org/10.1145/2427631.2427632

[44] M.-J. Wu, M. Zhao, and D. Yeung, "Studying multicore processor scaling via reuse distance analysis," in *the 40th International Symposium on Computer Architecture*, ser. ISCA-XL, June 2013. [Online]. Available: http://maggini.eng.umd.edu/pub/wu-isca13.pdf

[45] G. Yessin, L. Riha, T. El-Ghazawi, and D. Mayhew, "Application-specific processors for web-browsing: An exploration and evaluation of the design space," in *2013 IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors (ASAP 2013)*, IEEE. Washington, DC: IEEE, June 2013, pp. 87–90.

[46] D. Zoni, S. Corbetta, and W. Fornaciari, "HANDS: Heterogeneous Architectures and Networks-on-Chip Design and Simulation," in *the 2012 ACM/IEEE international symposium on Low power electronics and design*, Redondo Beach, CA, 2012.