Wu Y, Chen YJ, Chen TS *et al.* An elastic architecture adaptable to various application scenarios. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 29(2): 227–238 Mar. 2014. DOI 10.1007/s11390-014-1425-x

# An Elastic Architecture Adaptable to Various Application Scenarios

Yue Wu[1,2] (伍　岳), Yun-Ji Chen[1] (陈云霁), *Member*, *CCF*, *ACM*, *IEEE*, Tian-Shi Chen[1] (陈天石)
Qi Guo[3] (郭　崎), and Lei Zhang[1] (张　磊), *Member*, *CCF*, *ACM*, *IEEE*

[1] *State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences Beijing 100190, China*

[2] *University of Chinese Academy of Sciences, Beijing 100049, China*

[3] *IBM Research-China, Beijing 100193, China*

E-mail: {wuyue, cyj, chentianshi}@ict.ac.cn; joyguoqi@gmail.com; zlei@ict.ac.cn

Received November 14, 2013; revised January 8, 2014.

**Abstract** The quantity of computer applications is increasing dramatically as the computer industry prospers. Meanwhile, even for one application, it has different requirements of performance and power in different scenarios. Although various processors with different architectures emerge to fit for the various applications in different scenarios, it is impossible to design a dedicated processor to meet all the requirements. Furthermore, dealing with uncertain processors significantly aggravates the burden of programmers and system integrators to achieve specific performance/power. In this paper, we propose elastic architecture (EA) to provide a uniform computing platform with high elasticity, i.e., the ratio of worst-case to best-case performance/power/performance-power trade-off, which can meet different requirements for different applications. It is achieved by dynamically adjusting architecture parameters (instruction set, branch predictor, data path, memory hierarchy, concurrency, status&control, and so on) on demand. The elasticity of our prototype implementation of EA, as Sim-EA, ranges from 3.31 to 14.34, with 5.41 in arithmetic average, for SPEC CPU2000 benchmark suites, which provides great flexibility to fulfill the different performance and power requirements in different scenarios. Moreover, Sim-EA can reduce the EDP (energy-delay product) for 31.14% in arithmetic average compared with a baseline fixed architecture. Besides, some subsequent experiments indicate a negative correlation between application intervals' lengths and their elasticities.

**Keywords** architecture design, configurable, elasticity, energy-delay product reduction

## 1 Introduction

The continuous advancing of computer processor designing during the past decades enables millions of computer applications to emerge. An application may have different constraints on execution expenditure in different scenarios. For example, the power consumption requirements of a same application can be significantly different for hand-held terminals and desktop computers.

Under this circumstance, more and more processors, which have similar functionalities, have been devised to meet different performance-power trade-off requirements in different scenarios. For instance, the processor products of Intel had increased from 5 types to near 30 types between 1999 and 2009①. Such repetitions consume considerable cost in processor design and manufacture. Furthermore, the diversity of processors makes the performance and power of an application non-deterministic, which is inconvenient for programmers and system integrators.

To address this problem, we seek for a uniform platform to tackle applications under different requirements more efficiently. That is, this platform can solve applications with great flexibility in performance and power. To quantitatively evaluate the flexibility of an architecture, we propose a quantitative measure called elasticity, which means the ratio of the worst-case and best-

Regular Paper

This work is partially supported by the National Natural Science Foundation of China under Grant Nos. 61003064, 61100163, 61133004, 61222204, 61221062, 61303158, the National High Technology Research and Development 863 Program of China under Grant No. 2012AA012202, the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDA06010403, and the Ten Thousand Talent Program of China.

A preliminary version of this paper was published in the Proceedings of NPC 2012.

① Intel 1999 annual report, http://www.intel.com/content/dam/doc/report/history-1999-annual-report.pdf, January 2014, and Intel 2009 annual report, http://www.intc.com/intelAR2009/, January 2014.

case responses[②] when executing an application on the architecture. Conventional architectures have certain degrees of elasticities, however, their room of adjusting responses is quite narrow. For instance, the performance elasticity of a modern processor with dynamic voltage/frequency scaling (DVFS) with the frequency ranging from 1GHz to 2GHz is 2 for all applications. Therefore, it might be hard for a conventional architecture to meet more sophisticated requirements (e.g., maintaining a high performance-power tradeoff) when facing various applications.

In this paper, we propose Elastic Architecture (EA), which offers high elasticities with respect to performance, power, and performance-power tradeoffs for different application scenarios by employing reconfigurable architectural features. Through dynamically configuring these features, the thread-level parallelism (TLP) and instruction-level parallelism (ILP) of the processor can be adjusted on demand. For example, in multi-core systems that explicitly exploit the TLP, we can intuitively shut down several cores to save power once the performance requirement is met. Moreover, the reconfiguration of multi-core architecture around hardware failure is also a promising solution to provide enough system availability, reliability and dependability for fault-sensitive, even life-critical scenarios, i.e., the failure may result in death or severe damage. In comparison with conventional architecture that cannot adapt to concrete applications and scenarios, the high elasticity of EA not only allows it to achieve optimal or near-optimal response over an individual application, but also makes it possible to meet sophisticated performance/power requirements under different application scenarios.

Based on above ideas, we implement a prototype design of EA, named Sim-EA. Sim-EA can be reconfigured into more than 70 000 000 architecture instances. We present the first part of the experiment by evaluating the elasticity of performance-power tradeoff with EDP (energy-delay product) of the EA, using 26 benchmarks of SPEC CPU2000. The arithmetic average EDP elasticity of Sim-EA is 5.41 and the mean EDP reduction (defined as the percentage of EDP reduction of the best-case response to the fixed baseline response) is 31.14%. The results indicate that selected reconfigurable components have crucial impacts on the performance-power tradeoff, and the EA can provide near-optimal EDPs for all benchmark applications. In the later part of the experiments, we focus on how the interval length will impact elasticity and EDP reduction when executing an application interval by interval (i.e., segment by segment), using 16 segments of applications from SPEC CPU2006. The interval lengths are set into three levels, which are 1, 3, and 10 million instructions (per interval) respectively, from the fine-grained to the coarse-grained. Both elasticity and EDP reduction show a monotonic negative correlation with the interval length. To be exactly, the arithmetic average elasticity goes from 17.56 (with interval length of 10 million instructions) to 19.24 (3 million) and up to 22.25 (1 million); while the EDP reduction goes from 33.93% (10 million) to 37.61% (3 million), and finally to 42.41% (1 million). The results indicate that shorter application intervals get higher elasticities and larger percentages of EDP reduction.

The main contributions of this paper can be summarized as follows. First, the elasticity is proposed to measure quantitatively the flexibility of a processor. Second, the elastic architecture, which can adapt to the scenario requirements through dynamically reconfiguring architecture parameters, is introduced. Third, the advantage of the EA has been demonstrated by our experimental results carried out on a prototype of EA.

The rest of the paper is organized as follows. Section 2 introduces the concept and critical design issues of EA. Section 3 presents the experimental methodology. Section 4 empirically evaluates our implementation of EA, as Sim-EA with respect to elasticity and EDP reduction on a baseline architecture. Section 5 discusses utilizing application clusters to reduce the reconfiguration overheads. Section 6 briefly reviews some related work. Section 7 concludes the paper and discusses the future work.

## 2 Elastic Architecture

In this section, we first present the concept of elastic architecture. After that, we introduce an implementation of elastic architecture. Some additional implementation issues are also discussed in this section.

### 2.1 Concept

We say a CPU architecture is an elastic architecture (EA), if its main features, which include instruction set, branch predictor, data path, memory hierarchy, concurrency, status&control, and so on, can be dynamically reconfigured on demand.

The EA is CPU-based so as to achieve generality. It exhibits elasticity in different aspects, such as instruction set, performance, power. For an application with high-performance requirement, the EA can work as a high-performance CPU. For an application with low-

[②] Response is a term widely used in design space exploration, which refers to some kind of execution expenditure (e.g., performance, power, performance-power trade-off).

power requirement, it can work as a low-power CPU. For an application without specific requirement, it can select a running mode with a high performance-power tradeoff. The concrete configurable features of an EA may include:

1) The configurable instruction set mainly requires a configurable instruction decoder, which is sufficient to support multi instruction sets. It enables an EA to support applications developed for different computer families.

2) The configurable branch predictor guarantees the EA can adopt suitable branch prediction strategies (e.g., global history predictor, local history predictor) for applications with different types of branch behaviors.

3) The configurable data path provides flexible computational ability for the EA. Through configuring data path, the number and functionality of computational units can be adjusted to meet specific performance and power requirements.

4) The configurable memory hierarchy is crucial to the EA, since the memory hierarchy may consume half of the area in a state-of-the-art CPU. There are many important memory hierarchy parameters, such as cache size, cache line size, cache way, cache replacement strategies. Each of these parameters has a non-negligible impact on the performance and power.

5) The configurable concurrency includes not only TLP, but also ILP. Concretely, the concurrency configurations can include core number, core interconnection issue width, instruction window size, and so on.

6) The configurable status&control include voltage adjustment, frequency adjustment, kernel-model resource adjustment, and other miscellaneous CPU configurations.

Fig.1 illustrates the concept of EA, in which a central configuration module controls the other modules (the instruction fetch&decode model, execution engine, computational units, memory access unit, status and core interconnection) through several configuration bus (instruction configuration bus, branch prediction configuration bus, concurrency configuration bus, data path configuration bus, memory hierarchy configuration bus, and status&control configuration bus).

It is worth noting that an EA does not require that all the above features of the CPU are reconfigurable, since it may bring unnecessary difficulty to implementation. In other words, for a certain EA, it always consists of a part of fixed features and a part of reconfigurable features. Hence, determining which part to reconfigure and reconfigure to what extend should cautiously trade off the design complexity and obtained benefits. An empirical guideline to determine the reconfigurable features is that such reconfigurable parts can provide large elasticity for applications, which can offer great adaptivity to a wide range of application scenarios.

### 2.2 Implementation of EA

To demonstrate the feasibility and merit of EA, we implement Sim-EA, which is a prototyping of EA, on a SimpleScalar-like C simulator[1]. As shown in Table 1,

**Table 1.** Reconfigurable Parameters in Processor with EA

| Abbreviation | Parameter | Value |
|---|---|---|
| WIDTH | Fetch width | 2, 4, 6, 8 |
| FUNIT | FPALU/FPMULT units | 2, 4, 6, 8 |
| IUINT | IALU/IMULT units | 2, 4, 6, 8 |
| L1IC | L1-ICache | 8 KB∼256 KB: step 2* |
| L1DC | L1-DCache | 8 KB∼256 KB: step 2* |
| L2UC | L2-UCache | 256 KB∼4 096 KB: step 2* |
| ROB | ROB size | 16∼256: step 16+ |
| LSQ | LSQ size | 8∼128: step 8+ |
| GSHARE | GShare size | 1 K∼32 K: step 2* |
| BTB | BTB size | 512∼4 096: step 2* |
| Total | 10 parameters | 70 778 880 options |

Note: "step 8(16)+" means the parameter is a finite arithmetic sequence the common difference of which is 8(16); "step 2*" means the parameter is a finite geometric sequence the common ratio of which is 2.

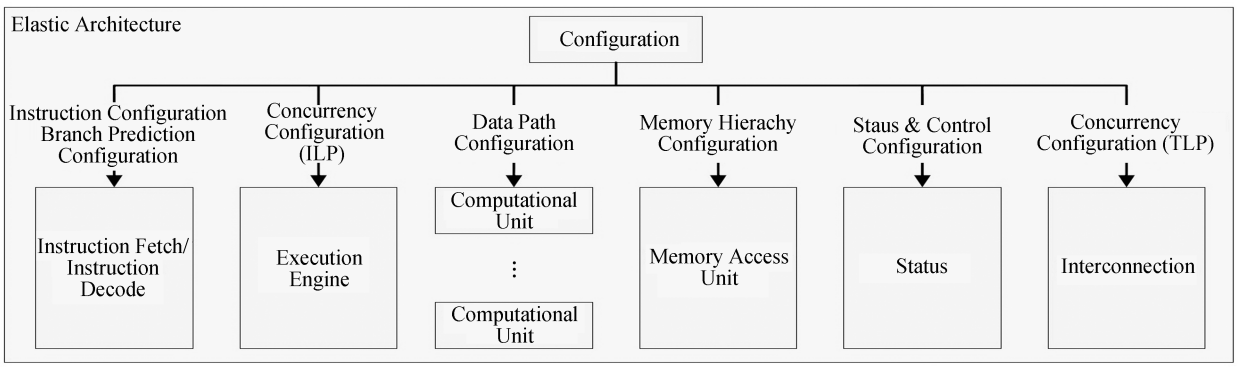



Fig.1. In an elastic architecture, the instruction fetch&decode model, execution engine, computational units, memory access unit, status, and interconnection can be dynamically configured.

there are 10 configurable parameters in the EA, which include the issue width (WIDTH), the number of floating-point functional unit (FUNIT), the number of integer functional unit (IUNIT), the size of L1 data cache (L1DC), the size of L1 instruction cache (L1IC), the size of L2 cache (L2UC), the size of gshare branch history table (GSHARE), the size of branch target buffer (BTB), the size of reorder buffer (ROB), and the size of load store queue (LSQ).

We choose the above parameters mainly for two reasons. Firstly and the most importantly, these parameters provide significant elasticity in our implementations. They are critical to influence elasticity by controlling the overall performance/power of the processor, i.e., most of such parameters are closely related to the ILP, e.g., WIDTH, FUNIT, IUNIT, GSHARE, BTB and ROB. Secondly they can be conveniently configured. Among these parameters, WIDTH, FUNIT, IUNIT, ROQ and LSQ can be reconfigured in about 10 cycles. Once the pipeline is flushed, they can take effect immediately. L1DC, L1IC, L2UC, GSHARE and BTB need relatively long time to be reconfigured, since the corresponding RAMs are needed to be flushed before reconfiguring these parameters. The detailed costs of reconfiguring such parameters are shown in Table 2.

**Table 2.** Reconfiguration Costs of Configurable Parameters

| Parameter | Reconfiguration Costs |
|---|---|
| WIDTH | $\approx$ 10 cycle (flush pipeline) |
| FUNIT | $\approx$ 10 cycle (flush pipeline) |
| IUNIT | $\approx$ 10 cycle (flush pipeline) |
| L1DC | $\approx$ 2 000 cycle (flush L1D cache) |
| L1IC | $\approx$ 2 000 cycle (flush L1I cache) |
| L2UC | $\approx$ 10 000 cycle (flush L2 cache) |
| GSHARE | $\approx$ 200 cycle (flush gshare table) |
| BTB | $\approx$ 100 cycle (flush branch target buffer) |
| ROB | $\approx$ 10 cycle (flush pipeline) |
| LSQ | $\approx$ 10 cycle (flush pipeline) |

WIDTH relates to several pipeline stages of a processor, including fetch, decode, dispatch, writeback, and commit. In Sim-EA, these stages should be bounded by WIDTH. Concretely, in the fetch stage, the PC increment and the number of fetched instructions should be less than the given WIDTH. In the dispatch stage, the number of dispatched instructions should be less than the configured issue width. Similarly, in the writeback stage and commit stage, the result bus and commit bus should be no wider than the given WIDTH.

FUNIT and IUNIT mainly relate to the issue queues of the dispatch stage. The number of selected instructions to be issued per cycle is also bounded by FUNIT and IUNIT.

To configure the sizes of L1 data cache, L1 instruction cache, L2 cache, branch history table, and branch target buffer, the most important support is the flexible RAM. In Fig.2, we use L1 data cache to illustrate how to achieve 4 KB$\sim$64 KB configurable data cache. We use 16 small RAMs, each of which has a size of 4 KB. When the data cache is configured to be 4 KB, only ram00 is used. When the data cache is configured to be 16 KB, we can use ram00, ram01, ram02, and ram03 to form a 4-way cache. When the data cache is configured to be 64 KB, all RAMs should be activated.

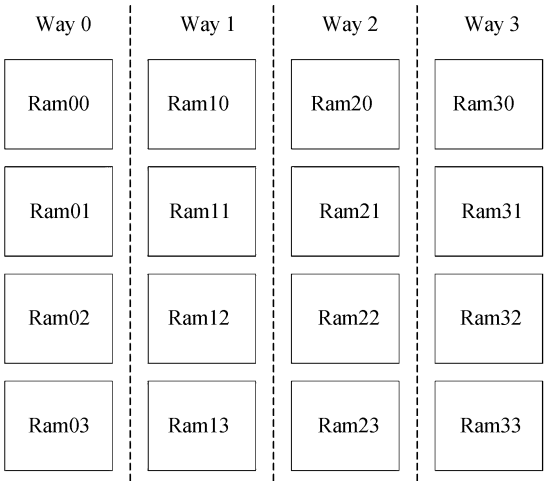



Fig.2. Configurable L1 data cache. Each ram is 4 KB.

Implementing configurable reorder buffer or load store queue is not so difficult as imagined. Take load store queue as an example. The size of the load store queue is used to calculate whether the queue is full or not, to allocate new queue entry, and to select the appropriate queue entry to control data bus (such as result bus). In practice, the impacts of queue size on these operations can be easily encapsulated through modulization.

### 2.3 Additional Implementation Issues

In practice, the implementation of an EA may meet two main problems.

One problem is about area. Intuitively, an EA may introduce additional area to support reconfiguration. We argue that an EA does not have to bring significant area cost in comparison with a fixed architecture whose parameters are the same with the maximal parameters supported by the EA. As we have mentioned in Subsection 2.2, all of the ten parameters involved by Sim-EA can be implemented with little effort. They need neither a large number of registers, nor additional RAM. Furthermore, they do not introduce complex combina-

tional circuit. Considering the advantage of flexibility, paying a little extra area on elasticity is cost-efficient.

The other problem is about frequency. Obviously, introducing additional logic for elasticity may decrease the maximal frequency of a processor. However, such decrease is very small. Any configurable feature can be finally boiled down to a multiplexor, which selects the output from several functionality modules according to the enable signal determined by the configuration (as shown in Fig.3). When the enable signal of a multiplexer is fixed to a certain value (e.g., 2'b00), the latency from the S0 port to the Out port is only around 10 ps in 28 nm process. Hence, for a processor with 3GHz maximal frequency, the impact of elasticity is only 3%.

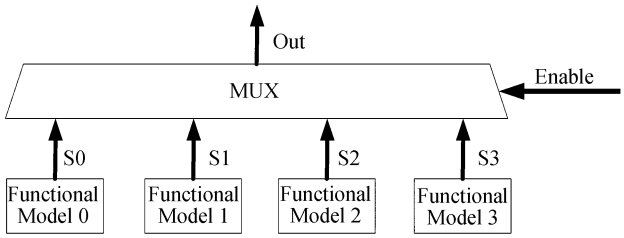



Fig.3. Multiplexer to choose which functional module to use.

## 3 Experimental Methodology

In this section, we present the experimental methodology utilized in our experiments. To evaluate the EDP elasticity and the EDP reduction of Sim-EA, we employ directly the 26 benchmarks of SPEC CPU2000 as the representative applications for real life applications in different fields. As shown in Table 1, 10 crucial features of Sim-EA are variable, resulting in a design space with more than 70 M potential design architectures. As defined in Section 1, the EDP elasticity is the ratio of the worst-case and best-case EDP, which means that we should determine such two extreme architectures (i.e., achieving the best and worst EDPs) from the design space for each application. Furthermore, we also want to obtain the EDP reduction compared with a baseline architecture, which also requires us to obtain the optimal architectures from such a design space for each application.

Actually, the above problem, which is called the *design space exploration* problem, has haunted architects in last decade[2-4]. Apparently, the traditional brute-force first-simulate-then-compare way falls short of such a large design space due to extremely slow simulation speed. To efficiently explore this design space, we utilize predictive modeling techniques to significantly reduce the number of design architectures need to simulate[5-6]. Concretely, we only simulate a small portion of the whole design space to obtain corresponding performance and power responses. The obtained responses, along with simulated architectures, form the sample for building predictive models via machine learning techniques (e.g., model tree algorithm[7-8]), which is often referred as the *training phase*. Afterward, in the so-called *predicting phase* such models can be employed to predict the performance/power responses of new architecture without tedious simulations. The detail process of predictive modeling is illustrated in Fig.4. Typically, in comparison with the traditional brute-force approach, predictive modeling can achieve at least 100x, even 10 000x or larger speedup in design space exploration.

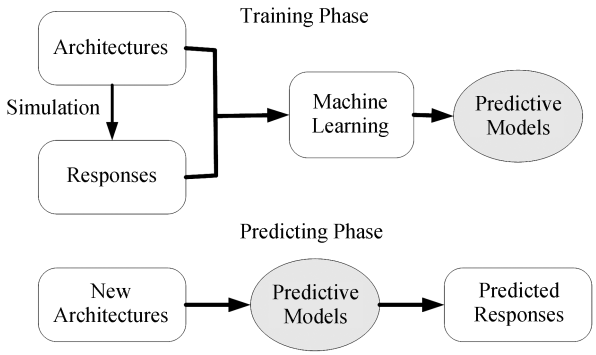



Fig.4. Framework of predictive modeling technique for design space exploration.

For fair comparison, we employ average cycle-per-instruction (CPI) to measure the performance of each architecture in the design space. In addition to the performance metric, we also estimate the average power consumption of each architecture. The goal to find an optimal architecture for each application can be precisely interpreted as to find the architecture with the best performance-power tradeoffs. One of the most widely used metrics on performance-power tradeoffs can be formulated as $\text{CPI}^2 \times power$, which corresponds to the EDP[9] as mentioned before. Apparently, an architecture with smaller EDP indicates that it has better performance-power tradeoffs. In other words, we determine the architectures with the minimal EDP for each application during the early design of Sim-EA via predictive modeling techniques.

## 4 Experimental Results

### 4.1 Elasticity

As the first step of experiments, for each of the 26 benchmark applications we employ predictive modeling techniques to explore the design space. For each application, we first randomly sample 500 design architectures as the training set, that is, we simulate each application with 500 different design architectures to ob-

tain the corresponding performance/power responses. Then, such information is collected as the training data to build two predictive models, i.e., model trees, for performance and power, respectively. After that, the performance/power with respect to a given architecture can be rapidly deduced by such models. Since we can easily know the performance/power responses with the help of predictive models, it is straightforward to find the two architectures with the best and worst EDP for each application.

Once we can obtain the best and worst case EDP of Sim-EA, we can show the elasticity over 26 applications in SPEC CPU2000 in Fig.5. We observe that the elasticity ranges from 3.31 (*sixtrack*) to 14.34 (*art*), and the arithmetic average elasticity is 5.41. Briefly, an application corresponding to larger elasticity implies that the application is more sensitive to Sim-EA, which provides larger freedom for the EA to dynamically reconfigure the architecture features on demand. For example, the elasticity of *art* is 14.34, which indicates that *art* is the most sensitive (among all the investigated 26 applications) to 10 reconfigurable parameters in Sim-EA. Moreover, the average elasticity as 5.41 indicates that the selected 10 reconfigurable parameters offer considerable freedom for Sim-EA to obtain an appropriate EDP for real-life scenarios. Similar situations can be observed when using performance or power as the alternative response for estimating the elasticity (i.e., performance elasticity or power elasticity).

### 4.2 EDP Reduction

By integrating the optimal architectures (obtained for the 26 benchmarks respectively) as the candidate running modes, Sim-EA can reconfigure its architecture to achieve promising EDP over different applications. To demonstrate the effectiveness of Sim-EA, a baseline architecture is employed in the experimental comparison. Intuitively, architecture with larger issue width, larger cache size and so on, can always achieve better performance, i.e., smaller CPI. However, power consumption will also increase with aggressive designs. Therefore, the default architecture in SimpleScalar Tool Suite, as shown in Table 3, is employed as the baseline. Although this baseline architecture is somewhat conservative, its power consumption is also very low compared with nowadays aggressive superscalar designs. Thus, it may also achieve better EDP compared with many existing commercial processors, especially for some performance insensitive applications.

**Table 3.** Baseline Architecture

| Parameter | Value |
|---|---|
| WIDTH | 4 |
| FUNIT | 4 |
| IUNIT | 4 |
| L1IC | 16 KB |
| L1DC | 16 KB |
| L2UC | 128 KB |
| ROB | 16 |
| LSQ | 8 |
| GSHARE | 2 048 |
| BTB | 2 048 |

Fig.6 shows the EDP reduction of Sim-EA compared with the baseline architecture, i.e., $(1 - EDP_{\text{Sim-EA}}/EDP_{\text{baseline}}) \times 100\%$. It can be observed that, for all applications, Sim-EA can reduce the EDP

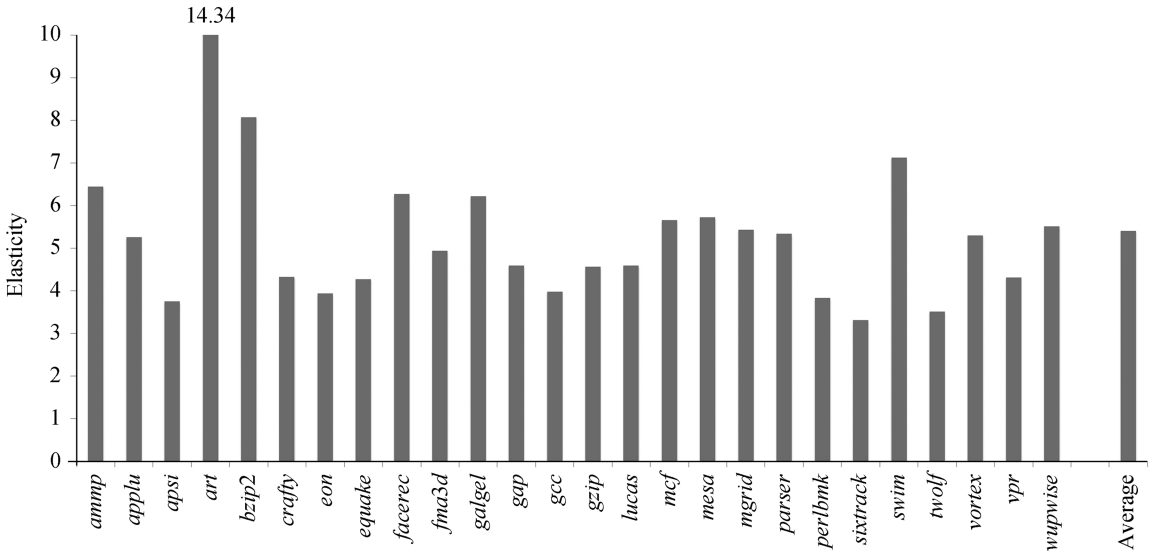



Fig.5. EDP elasticities of Sim-EA over different applications. The arithmetic average elasticity is 5.41, which indicates that 10 variable architectural parameters in Sim-EA are indeed crucial parameters to EDP that should be determined for reconfiguration.

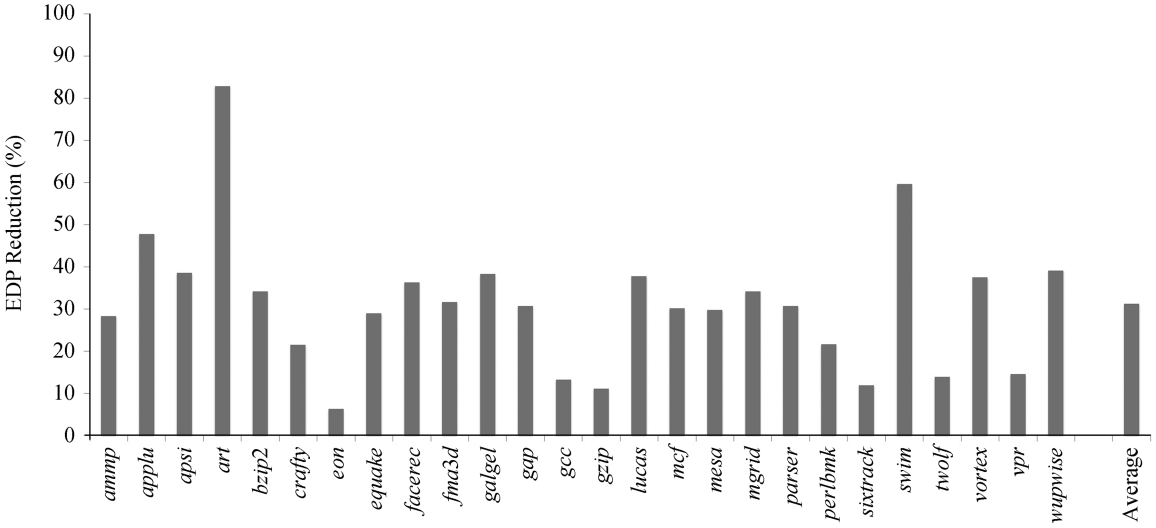



Fig.6. EDP reduction over the baseline architecture. The arithmetic average EDP reduction is 31.14%, ranging from 6.26% (*eon*) to 82.84% (*art*) for 26 applications in SPEC CPU2000.

of the baseline architecture significantly, and the arithmetic average EDP reduction is 31.14%. The benefit of Sim-EA is highlighted by application *art*, which can reduce 82.84% EDP compared with the baseline architecture. However, for *eon*, it can only achieve 6.26% EDP reduction, an insignificant improvement on EDP. To be specific, the CPI and power of Sim-EA on *eon* are 0.79 and 14.43 respectively, while the CPI and power of the baseline are 0.69 and 19.14, respectively. In fact, the baseline architecture has already been at the Pareto Frontier of the performance-power tradeoff function[10], and it is a near-optimal architecture with respect to EDP. In this case, the EDA reduction of Sim-EA is not significant.

### 4.3 Elasticity and EDP Reduction of Application Intervals

Similar to the former subsection, for each of the 16 application intervals (Table 4) which are selected from SPEC CPU2006 benchmark suit and each interval length (chosen from 1, 3, 10 million instructions), 500 random sampled architecture instructions are simulated to build the predicting model and explore the design space, to help find the best and worst instructions for each case. The results are shown in Fig.7 for elasticity and Fig.8 for EDP reduction. (Notice that two different groups of benchmarks are used, which means comparisons between the results of this subsection and the former subsection are meaningless.) Additionally, as shown in Fig.7 and Fig.8, for each application interval, the elasticity/EDP reduction changes as the interval length changes.

Fig.7 and Fig.8 show the range of elasticity/EDP reduction of the 16 application intervals. The elasticity

**Table 4.** Application Intervals from SPEC CPU2006

| Application | Serial Number of Input Parameter Set | Interval (Million Instructions) |
|---|---|---|
| 400.perlbench | 1st | 103∼104 |
| 400.perlbench | 2nd | 299∼300 |
| 400.perlbench | 5th | 202∼203 |
| 401.bzip2 | 1st | 83∼84 |
| 401.bzip2 | 2nd | 454∼455 |
| 403.gcc | 1st | 220∼221 |
| 433.milc | 1st | 191∼192 |
| 435.gromacs | 1st | 220∼221 |
| 436.cactusADM | 1st | 220∼221 |
| 437.leslie3d | 1st | 145∼146 |
| 444.namd | 1st | 220∼221 |
| 445.gobmk | 1st | 183∼184 |
| 445.gobmk | 5th | 150∼151 |
| 453.povray | 1st | 220∼221 |
| 456.hmmer | 1st | 277∼278 |
| 483.xalancbmk | 1st | 322∼323 |

ranges from 5.67 to 83.95, an incredible large value, which demonstrates the interval 463_1_221 is quite sensitive to different architecture instructions. The minimum of EDP reduction is 0.005 8, the interval picked from *gobmk*, indicating that the baseline instruction is almost as good as the best-case during the execution of the 10 million instructions. An intriguing phenomenon in both the figures is that as the interval length becomes bigger, elasticity and EDP reduction are reducing.

## 5 Discussions

As validated by previous experiments, Sim-EA can achieve optimal architectures for each application,

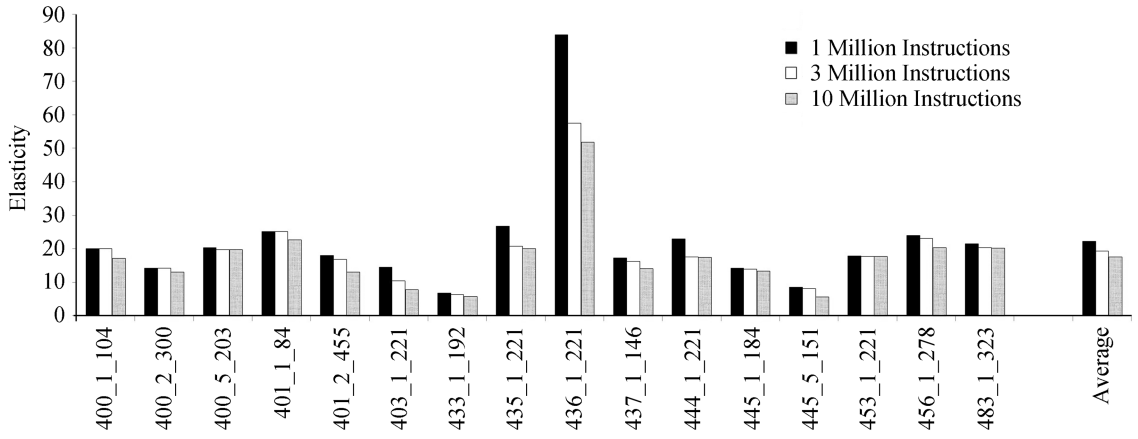



Fig.7. For each application interval, the elasticity of Sim-EA decreases as the interval length increases.

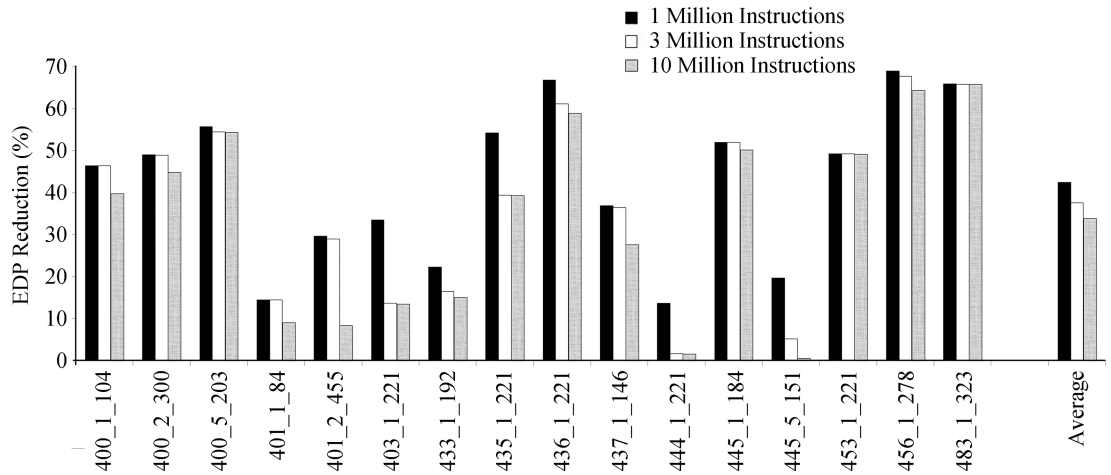



Fig.8. For each application interval, the EDP reduction of Sim-EA decreases as the interval length increases.

while the conventional processor (with fixed architecture) fails. However, in our implementation, Sim-EA should reconfigure the architectural parameters for each application. Actually, by detailed investigation on the optimal architectures for each application, we observe that there exists similarity among the optimal architectures of applications. As an extreme example, in Sim-EA, both *parser* and *twolf* achieve the best EDPs on the same architecture, which means that *twolf* may also benefit from the optimization on the architecture for *parser*, and it is not necessary to carry on reconfiguration between these two applications. Furthermore, considering the optimal architectures of other two applications as *mcf* and *parser*, the only difference between their optimal architectures is the number of LSQ, i.e., 16 (*mcf*) and 32 (*parser*), which shows that the optimal architecture for *mcf* is also the near optimal one for *parser* (only increasing 1% compared with the optimal EDP of *parser*). Therefore, *mcf* and *parser* can share the same optimal architecture in industrial design, which can reduce the reconfiguration overheads. We can also see that the similarity varies significantly among applications. Table 5 gives the optimal archite-

**Table 5.** Example of Applications with Significantly Different Optimal Architectures

| Parameter | *apsi* | *mgrid* |
|---|---|---|
| WIDTH | 8 | 4 |
| FUNIT | 8 | 2 |
| IUNIT | 8 | 2 |
| L1IC | 16 KB | 8 KB |
| L1DC | 8 KB | 8 KB |
| L2UC | 4 096 KB | 256 KB |
| ROB | 32 | 128 |
| LSQ | 32 | 64 |
| GSHARE | 1 024 | 1 024 |
| BTB | 512 | 4 096 |

ctures of *apsi* and *mgrid*, where nearly all parameters (8 of 10) need to reconfigure between these two applications. Otherwise, the optimal architecture for one application may significantly harm the responses of the other one, i.e., the best architecture of *mgrid* can increase 28.4% EDP on *apsi*.

Based on the above observations on the similarities among applications, we may utilize statistical/machine learning techniques to classify applications into different application clusters. In each cluster, we only need to select one architecture as the representative architecture for all applications in this cluster. In fact, the concept of application cluster has already been taken into account in performance evaluation for several decades. A famous example is the SPEC project[11]. It aims at selecting the most representative applications from distinct application clusters, so as to offer balanced quantitative evaluations for a computer that may be applied to applications from various clusters. Each benchmark in SPEC CPU2000 can be considered as a representative application of an application cluster consisting of applications from similar application domains. Recently, a number of investigations have been dedicated to the classifications of applications based on architecture-independent program characteristics[12-15], which offer alternative ways of defining different application clusters. Actually, the fact that existing researches on defining application clusters are commonly based on the analysis of application characteristics, gives us some hints to determine application clusters for the design of efficient EA.

One potential criterion of defining application clusters for EA is that, whether or not applications belonging to the same cluster have similar responses when being executed by a same computer architecture. If an application cluster can be appropriately defined to meet the above criterion, by the optimal architecture deduced from the representative application we are able to find optimal/near-optimal architecture for all the applications in this application cluster. Therefore, when encountering a new application, after its characteristics have been extracted and assigned to a specific cluster with "similar" applications, we only need to reconfigure the architecture to the representative architecture of this cluster, and it is not necessary to reconfigure the architecture *in* this cluster, which also can obtain the near optimal responses of this application.

## 6 Related Work

*Adaptive Systems.* Over the past decades, in contrast to fine granularity reconfigurable systems in the gate level, many studies try to tackle with the adaptivity by adjusting different micro-architecture features, e.g., issue queue[16], reorder buffer, register file[17], pipeline[18], and cache size[19]. Besides, as multi-core becomes the mainstream architecture, many proposals try to address the resource contention problem when executing several programs on different cores by partitioning the cache size and bandwidth[20-21] according to different application requirements, which is another kind of adaptive systems. However, all these researches only focus on limited number of architectural features that can be easily changed.

Recently, a table-driven adaptive processor core has been proposed to reduce the peak power[22]. In this scheme, the design space is much smaller than ours and the resultant elasticity is greatly restricted. Core fusion is a reconfigurable CMP where groups of independent small cores can be dynamically fused into a large superscalar CPU, to adapt to application diversity[23]. Although it can improve the performance of single-threaded programs, it cannot provide so much flexibility as EA due to the restriction of conventional multi-core architecture. For example, EA can reduce the number of functional units of a single core to achieve less power consumption or less EDP. The most closet proposal to our work is in [24], where Dubach *et al.* developed an adaptive micro-architecture that can tailor resources to the specific requirements of different program phases based on predictive modeling techniques. However, in our study, we quantitatively analyzed the elasticity of EA. Moreover, their adaptive system omits many implementation details in industrial design, while we presented detailed analysis of the reconfiguration overheads, which can be further reduced by application cluster techniques. Our former work in [25] first proposed the measurement of elasticity and the main frame of EA, but the detailed implementation were not fully presented, such as the discussion of flexible L1 data cache and the analysis of area & frequency cost brought by additional configuration logical units. Moreover, the elasticity and EDP reduction of application intervals rather than the whole execution were not evaluated.

*Design Space Exploration.* The design space exploration is the first and centric challenge during the microprocessor design. Due to the extremely slow simulation speed and exponential number of design architectures, it is impossible to explore the entire design space by full extent simulation. To reduce the simulation overheads, many fast simulation techniques have been proposed, e.g., statistical sampling[26-27], statistical simulation[28-29] and benchmark subsetting[14-15]. However, these fast simulation techniques only consider to reduce the simulated instructions for each design ar-

chitecture, and the resultant simulation speedup is restricted. Rather than reducing the simulated instructions for each architecture, predictive modeling reduces the number of architectures for simulation, which can significantly improve the simulation efficiency. In EA, we also need to utilize existing design space exploration techniques to determine the optimal architecture for each application cluster. Moreover, we hope EA can automatically detect the optimal architectures for each application. In fact, the predictive modeling is quite suitable for this task since prior knowledge is not necessary during model construction. It is clearly that the design space exploration technique is only one of the critical techniques to achieve software adaptivity.

*Program Characteristics.* A fairly large body of work exists on analyzing program characteristics for performance prediction or reducing the simulation costs. SimPoint, which is widely accepted for phase simulation, is built based on detailed analysis on the similarity of program characteristics such as cache miss rate, branch misprediction, and IPC[26]. Apart from the analysis of program signatures (microarchitecture dependent characteristics), microarchitecture-independent characteristics are also investigated by many proposals for performance prediction[12-13], benchmark subsetting[14-15], and compiler optimization[30]. The above program characterizations are concerned with single-threaded workloads. To adapt to parallel programs, Eyerman *et al.* proposed system metrics for multi-programmed workloads[31]. Besides, Wang *et al.* also proposed several program features for performance prediction in order to determine the optimal parallelism and scheduling policy[32]. However, unlike the characteristics for serial programs, the investigations on the inherent characteristics for parallel programs are far from enough. Moreover, even for proposed characteristics on single-threaded applications, we still need further investigation to determine the close relationship between program characteristics and optimal architectures to facilitate the design of EA.

## 7 Conclusions and Future Work

In 1964, IBM SYSTEM/360 was designed as the first computer family to cover all the complete range of applications[33-35]. An application designed for one member of a computer family can also run on another member of the computer family. Computer family greatly facilitates programmers and end users, thus has been widely accepted. Currently, most processors can be classified to some family, such as the x86 family, Itanium family, Power family, MIPS family, SPARC family, and ARM family. With the ever developing of computer industry, the programmers and end users have put more and more requirements on processors. When running different applications, they hope that the processor can adapt to specific requirements on performance/power. Such requirements cannot be satisfied with a processor, and may lead to excessive market segmentation of processors, which increases the overall cost of processor design and manufacture.

A promising solution is to make the processor elastic, which has many configurable features (e.g., instruction set, data path, memory hierarchy, concurrency). In this paper, we proposed a prototype design of EA, which is named Sim-EA. Experimental results show that with respect to SPEC CPU2000 benchmark suite, the elasticity of Sim-EA, ranges from 3.31 to 14.34, with 5.41 in arithmetic average, which provides great flexibility to fulfill the different performance/power requirements in different scenarios. Moreover, Sim-EA can also significantly reduce the energy-delay product for 31.14% in arithmetic average compared with a baseline fixed architecture. The correlation between application intervals' lengths and their elasticities also indicates that a proper reconfigure frequency may greatly improve elasticity, and furthermore improve performance/power of processors.

In the future development of EAs, it is possible that more architecture features are designed to be reconfigurable, resulting in advanced EAs. An advanced EA is said to be *downward-compatible* with a former EA if the advanced one can reconfigure all architecture features that can be reconfigured by the former. Driven by the rapid development of processor industry, it can be predicted that a new concept called *computer tribe* will debut, which is the set of consecutively-developed processors adopting downward-compatible EAs.

## References

[1] Austin T, Larson E, Ernst D. SimpleScalar: An infrastructure for computer system modeling. *Computer*, 2002, 35(2): 59-67.

[2] Kunkel S, Eickemeyer R, Lip M *et al.* A performance methodology for commercial servers. *IBM Journal of Research and Development*, 2000, 44(6): 851-872.

[3] Kumar R, Zyuban V, Tullsen D M. Interconnections in multicore architectures: Understanding mechanisms, overheads and scaling. In *Proc. the 32nd Annual Int. Symp. Computer Architecture* (*ISCA2005*), June 2005, pp.408-419.

[4] Li Y, Lee B, Brooks D *et al.* CMP design space exploration subject to physical constraints. In *Proc. the 12th IEEE Symposium on High Performance Computer Architecture* (*HPCA2006*), February 2006, pp.17-28.

[5] İpek E, McKee S A, Caruana R *et al.* Efficiently exploring architectural design spaces via predictive modeling. In *Proc. the 12th International Conference on Architectural Support for Programming Languages and Operating Systems* (*ASPLOS-XII*), October 2006, pp.195-206.

[6] Lee B, Brooks D. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proc. the 12th Int. Conf. Architectural Support for Pro-*

*gramming Languages and Operating Systems* (*ASPLOS-XII*), October 2006, pp.185-194.

[7] Quinlan J R. Learning with continuous classes. In *Proc. the 5th Australian Joint Conference on Artificial Intelligence* (*AI1992*), November 1992, pp.343-348.

[8] Wang Y, Witten I. Induction of model trees for predicting continuous classes. In *Proc. the 9th European Conference on Machine Learning* (*ECML1997*), April 1997, pp.128-137.

[9] Gonzalez R, Horowitz M. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 1996, 31(9): 1277-1284.

[10] Mariani G, Avasare P, Vanmeerbeeck G *et al.* An industrial design space exploration framework for supporting runtime resource management on multi-core systems. In *Proc. the Conference on Design, Automation and Test in Europe* (*DATE2010*), March 2010, pp.196-201.

[11] Hennessy J L, Patterson D A. Computer Architecture: A Quantitative Approach (3rd edition). San Francisco, USA: Morgan Kaufmann Publishers Inc., 2002.

[12] Hoste K, Phansalkar A, Eeckhout L *et al.* Performance prediction based on inherent program similarity. In *Proc. the 15th Int. Conf. Parallel Architectures and Compilation Techniques* (*PACT2006*), Sept. 2006, pp.114-122.

[13] Hoste K, Eeckhout L. Microarchitecture-independent workload characterization. *IEEE Micro,* 2007, 27(3): 63-72.

[14] Joshi A, Phansalkar A, Eeckhout L *et al.* Measuring benchmark similarity using inherent program characteristics. *IEEE Transaction on Computers*, 2006, 55(6): 769-782.

[15] Phansalkar A, Joshi A, John L K. Subsetting the SPEC CPU2006 benchmark suite. *SIGARCH Computer Architecture News*, 2007, 35(1): 69-76.

[16] Folegnani D, González A. Energy-effective issue logic. In *Proc. the 28th Annual International Symposium on Computer Architecture* (*ISCA2001*), June 2001, pp.230-239.

[17] Abella J, González A. On reducing register pressure and energy in multiple-banked register files. In *Proc. the 21st Int. Conf. Computer Design* (*ICCD2003*), Oct. 2003, pp.14-20.

[18] Hughes C J, Srinivasan J, Adve S V. Saving energy with architectural and frequency adaptations for multimedia applications. In *Proc. the 34th Annual ACM/IEEE Int. Symp. Microarchitecture*, Dec. 2001, pp.250-261.

[19] Balasubramonian R, Albonesi D, Buyuktosunoglu A, Dwarkadas S. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *Proc. the 33rd Annual ACM/IEEE Int. Symp. Microarchitecture*, Dec. 2000, pp.245-257.

[20] Qureshi M K, Patt Y N. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proc. the 39th Annual IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2006, pp.423-432.

[21] Liu F, Jiang X, Solihin Y. Understanding how off-chip memory bandwidth partitioning in chip multiprocessors affects system performance. In *Proc. the 16th Int. Symp. High Performance Computer Architecture*, January 2010.

[22] Kontorinis V, Shayan A, Tullsen D M, Kumar R. Reducing peak power with a table-driven adaptive processor core. In *Proc. the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, December 2009, pp.189-200.

[23] Ipek E, Kirman M, Kirman N, Martínez J F. Core fusion: Accommodating software diversity in chip multiprocessors. In *Proc. the 34th Annual Int. Symp. Computer Architecture*, June 2007, pp.186-197.

[24] Dubach C, Jones T M, Bonilla E V *et al.* A predictive model for dynamic microarchitectural adaptivity control. In *Proc. the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, December 2010, pp.485-496.

[25] Chen Y J, Chen T S, Guo Q, Xu Z W, Zhang L. An elastic architecture adaptable to millions of application scenarios. In *Proc. the 9th IFIP Int. Conf. Network and Parallel Computing*, Sept. 2012, pp.188-195.

[26] Sherwood T, Perelman E, Hamerly G, Calder B. Automatically characterizing large scale program behavior. In *Proc. the 10th Int. Conf. Architectural Support for Programming Languages and Operating Systems*, Oct. 2002, pp.45-57.

[27] Wunderlich R E, Wenisch T F, Falsafi B, Hoe J C. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In *Proc. the 30th Annual Int. Symp. Computer Architecture*, June 2003, pp.84-97.

[28] Yi J J, Lilja D J, Hawkins D M. Improving computer architecture simulation methodology by adding statistical rigor. *IEEE Transaction on Computers*, 2005, 54(11): 1360-1373.

[29] Genbrugge D, Eeckhout L. Chip multiprocessor design space exploration through statistical simulation. *IEEE Transaction on Computers*, 2009, 58(12): 1668-1681.

[30] Chen Y, Huang Y J, Eeckhout L *et al.* Evaluating iterative optimization across 1000 datasets. In *Proc. the ACM SIGPLAN Conf. Programming Language Design and Implementation*, June 2010, pp.448-459.

[31] Eyerman S, Eeckhout L. System-level performance metrics for multiprogram workloads. *IEEE Micro*, 2008, 28(3): 42-53.

[32] Wang Z, O'Boyle M F. Mapping parallelism to multi-cores: A machine learning based approach. In *Proc. the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Feb. 2009, pp.75-84.

[33] Blaauw G A, Brooks F P. The structure of SYSTEM/360: Part I: Outline of the logical structure. *IBM Systems Journal*, 1964, 3(2): 119-135.

[34] Stevens W Y. The structure of SYSTEM/360: Part II: System implementations. *IBM Systems Journal*, 1964, 3(2): 136-143.

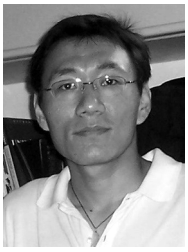
[35] Amdahl G. The structure of SYSTEM/360: Part III: Processing unit design considerations. *IBM Systems Journal*, 1964, 3(2): 144-164.

**Yue Wu** received the B.S. degree in statistics from University of Science and Technology of China, Hefei, in 2006. He is currently a Ph.D. candidate of Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His main research interests include computer architecture, design space exploration, and computational intelligence.

**Yun-Ji Chen** graduated from the Special Class for the Gifted Young, University of Science and Technology of China, Hefei, in 2002. He received the Ph.D. degree in computer science from Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, in 2007. He is currently a professor at ICT. His research interests include parallel computing, microarchitecture, hardware verification, and computational intelligence.

**Tian-Shi Chen** received the B.S. degree in mathematics from the Special Class for the Gifted Young, University of Science and Technology of China (USTC), Hefei, in 2005, and the Ph.D. degree in computer science from Department of Computer Science and Technology, USTC, in 2010. He is currently an associate professor at ICT. His research interests include computer architecture, parallel computing, and computational intelligence.

**Qi Guo** received the B.E. degree in computer science from Department of Computer Science and Technology, Tongji University, Shanghai, in 2007, and the Ph.D. degree in computer science from Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), in 2012. He currently is a staff researcher at IBM Research-China, Beijing. His research interests include computer architecture, VLSI design and verification.



**Lei Zhang** received the B.E. degree in computer sciences from University of Electronic Science and Technology of China in 2003, and his Ph.D degree in computer architecture from Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS) in 2008. He is currently an associate professor at ICT. His research interests include multi-core architecture, network-on-chip, fault-tolerant computing, cyber-physical systems and applications.