Raven: A Data-Driven Approach for Automatically Generating Heterogenous Processors

Abstract

Heterogeneous CMPs with two types of cores are now available as commodity products, and recent proposals argue for increasingly heterogeneous CMPs as a means to continue to improve energy efficiency. However, both designing and exploiting increasingly heterogeneous processors raises many challenges. Design choices for resource heterogeneity span a vast design space and must balance diversity commensurate with workload variability without becoming intractable to verify and construct. At run time, scheduling applications on the appropriate core is key to profiting from heterogeneity.

In this paper, we present Raven, an analytical model for tailoring heterogeneous processor resources to a target workload based on architecture-independent properties of the applications. This is done by pre-selecting the architecture features that can be adjusted as well as selecting the range of those adjustments. We show that 8 primary features are sufficient to guide both core feature selection and runtime placement. Using Raven, we produce heterogeneous platforms that are an average of 10% more energy efficient than comparable two-way heterogeneous platforms and homogeneous processors alike.

1. Introduction

The demise of Dennardian scaling [8] has lead processor designers to increasingly shift their focus from raw, per-core performance toward energy efficiency as a primary metric. This transition has given rise to a multitude of multicore designs across many domains. Compared to their uniprocessor predecessors, the more modestly aggressive cores in modern multicore processors reap efficiency benefits not only from avoiding less rewarding regions of the superlinear relationship between peak core power and peak core performance, but also from the fact that there are many applications (e.g. SPECINT-like desktop programs) which will only rarely approach peak core utilization.

Recent proposals [22, 30] and products [1] aim to further capitalize on the latter of these effects by increasing processor heterogeneity. On a heterogeneous platform, each application can run on the least energy-expensive processor that still allows it to achieve a high fraction of its potential performance, increasing overall efficiency. However, the space of all potential heterogeneous processors is quite large, even when restricting the options to traditional core types. It is not yet clear how best to select the constituent cores in a heterogeneous CMP, nor how best to map incoming applications to those cores.

There are two prominent schools of thought for heterogeneous design strategies, but much room remains in the middle for new approaches. Current designs, such as ARM's Big.LITTLE [1], draw inspiration from early work [16, 17, 18] on heterogeneity that advocates limiting design costs by combining previously developed architectures. This, however, may limit the energy savings of the system, because it maps applications to cores whose resource diversity owes more to temporal changes in design restrictions than to a concerted effort to more efficiently serve particular classes of current or future applications. At the other extreme, proposals such as [6, 7, 10, 31] call for domain or even application-level specialization of cores, which maximizes energy efficiency, but vastly increases design effort and may produce systems that are overly sensitive to changing workloads. Taking insight from both ends of the design spectrum, the intuition is that there will be profitable ground in the middle: heterogeneous systems that consist of general purpose cores with traditional components, but where each core in the collection can optimize its performance/energy tradeoffs for a subset of applications.

In this paper, we present Raven, a high-level design approach for automatically selecting the degree and dimensions of heterogeneity in a CMP based on the architecture-independent features of a target workload. Raven constructs processor configurations out of a modest sized library of fixed components, but with many possible combinations. Lifting the restriction that every core be well-suited for executing the entire workload allows Raven to select individual cores with non-standard configurations, e.g. cores lacking a floating point unit and large caches due to the requirements of part of the workload, while still effecting general application support through the union of cores selected in a particular design. With Raven we can quickly predict what set of cores would be needed in a single CMP to exploit heterogeneity within a workload and to produce multiple CMP designs to better exploit heterogeneity among workloads from different domains.

At a high level, Raven works as follows: Raven considers a design space spanning 8 dimensions including cache size, issue width, and ALU allocation. We train on a small number of applications across a subset of the configurations within that design space. For each training application or application phase, we collect a vector of architecturally independent features gathered from MICA [14] and the associated energy-performance-area three-tuple on a given core. To design a particular heterogeneous multiprocessor, we pass the architecturally independent feature vectors of a target workload into a

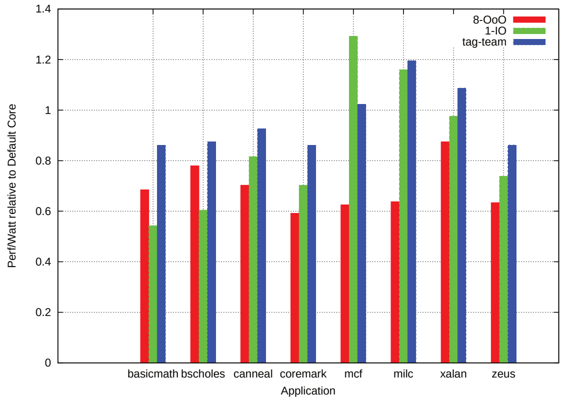

Figure 1: ***Performance-per-Watt Statistics:*** **An 8-wide out-of-order, 1-wide in-order, and basic heterogeneous processor span a wide range of performance/power tradeoff points, but the heterogeneous option is consistently more efficient than either extrema.**

trained Raven, as well as the maximum allowed degree of heterogeneity, K. Raven then partitions the input applications into up to K groups and, for each group, estimates the energy and performance of the group across all points in the design space. Finally, Raven picks the optimal performance/watt configuration across each group, selects that processor for inclusion in the CMP, and provides a distance-based mapping function between the architecture-independent features of any current or future application and the predicted-preferred core type.

The contributions of this work include the following:

- We describe Raven, a design approach for automatically selecting an energy-efficient set of heterogeneous cores, given an architecturally independent description of a workload.
- We show that Raven designs with greater degrees of heterogeneity offer superior energy efficiency compared to homogeneous or limited heterogeneity designs, improving on existing designs by up to 18%.
- We show that there is sufficient variation among workloads, and that different workloads warrant different heterogeneous solutions.
- We explore the sensitivity of Raven designs to workload variation and show that Raven performs well against other potential solutions.

The rest of the paper proceeds as follows. Section 2 provides background on the allure of and challenges posed by heterogeneous architectures, and discusses related work. Section 3 presents our design for Raven and Section 4 validates our approach. Section 5 evaluates the heterogeneous processors that Raven produces and Section 6 concludes.

2. Background and Related Work

The fundamental appeal of heterogeneous architectures lies in the fact that workload demands are inherently heterogeneous at several levels: Applications in different domains have different characteristics, different applications within a domain differ in their resource bottlenecks [15], and even within a single application, different phases provide different power/performance tradeoffs. Figure 1 illustrates the potential benefits of a heterogeneous processor. While an 8-wide out of order execution core may succeed in terms of raw performance, it lags behind even a 1-wide in order core in performance-per-watt metrics. On top of this, a simple two-way heterogeneous core (architecture details are provided in table 3) that has both types of execution models is able to outperform both on average by making simple energy-minded distinctions between the two cores.

2.1. Heterogeneity in General Purpose Multiprocessors

The body of work on single-ISA heterogeneous multicore processors [16, 17, 18] investigated the power/performance tradeoffs for asymmetric CMPs. By transitioning on phase boundaries between aggressive and simple cores, these designs were able to save up to 50% in power budget for a performance reduction of only 10%. That all said, the selection of which particular cores to include in a particular CMP design for a particular workload was not considered in depth.

More recently there have been efforts to place heterogeneous designs on physical silicon. [1] These processors combine two existing architectures with different execution models, but are designed to work as a heterogeneous unit, varying the execution on the processor based on the needs of the application. Some take this concept one step further by combining as many architectural components as possible to try and develop a single heterogeneous core [22]; capable of switching between a large out-of-order engine and a small in-order engine depending on the IPC of the application being run. Raven improves on previous approaches relying on collections of existing processor models by automatically selecting the appropriate degrees of asymmetry to best exploit a workload. The range of asymmetry is only limited by the number of architecture features one desires to change for a particular core design.

2.2. Heterogeneity in specialized architectures

As power constraints tighten, designers are increasingly integrating specialized coprocessors into general-purpose architectures. GPUs are an especially common addition, and are now found on-chip in everything from cell phones to servers. Many recent efforts [21] attempt to harness these heterogeneous platforms with language extensions like CUDA [23] and streaming frameworks such as Brook [5], but they focus primarily on highly-parallel code and loosely-coupled execution models. Even flexible heterogeneous processing frameworks such as Intel's EXOCHI [32] face deployment challenges in scaling to greater degrees of heterogeneity: EXOCHI's uniform abstraction for sequencing execution across heterogeneous execution engines still requires specialized compilers for each piece of target hardware. In contrast, Raven is designed to run existing, unmodified binaries and maintain a focus on a single-ISA design, rather than relying on multiple programming models to achieve maximum efficiency.

Recent efforts on generating internally diverse processors have focused on automating the production and use of specialized coprocessors [31, 25]. These automatically-generated coprocessors do not achieve the performance of hand-crafted accelerators, but they are very energy-efficient and can target nearly-arbitrary code. Raven on the other hand focuses on remaining a general purpose platform, allowing for any application that is designed for the underlying ISA to be executed. At the same time, it does not require processors to be so specific that they have little use outside of the applications they were trained under, and indeed this is a design choice that tends to be avoided when possible.

2.3. Heterogeneous scheduling

Scheduling a heterogeneous processor presents unique challenges but is essential in order to provide energy benefits from its varied set of cores. There are varying approaches on how best to perform this scheduling, from monitoring metrics during execution [29] to developing a programming system that dynamically schedules applications based on running the application through a virtual machine first [21]. These methods rely on focusing on the monitoring of applications during runtime, and making changes in scheduling based on the application's performance on the system.

One other way that scheduling is performed is by gathering the architecture signatures of an application to determine the best fit out of a given set of cores. [26] Rather than focus on a phase-driven analysis, scheduling is performed by looking at the typical behavior of the application given a set of parameters. Raven deploys a similar technique, using architecture independent metrics in order to make informed decisions on the type of core that would be a good fit for a given application. In either case it will need an efficient scheduling algorithm to take advantage of the cores that are designs so research in this area is important to the overall health of the concept of a heterogeneous processor.

2.4. Comprehensive Prediction

For the most part, the desire for a comprehensive prediction and core generation scheme for heterogeneous processors is a relatively new one. Efforts to date have been focused on making well-established changes to processor configurations or combining accelerators with current general-purpose cores to achieve the goal of heterogeneity as discussed earlier. The goals have changed as well; previously they been focused on raw performance improvements [17] or attempting to cluster applications themselves to aid in the design of broader accelerator-based systems [11], but not necessarily in a single-ISA framework. Raven attempts to solve this issue, providing a prediction model that can be broadened to any number of architectural features as needed while using architecture independent metrics to aid in the construction of heterogeneous cores.

3. Raven Design

Raven is a regression-based model that we have designed to perform two tasks. First, given a workload, Raven must be able to accurately select a covering set of core designs that collectively constitute a general purpose, energy efficient CMP. Second, given a previously generated CMP and a new application, Raven must accurately predict on which core on the CMP the application will most efficiently run.

Given any non-trivial set of heterogeneity dimensions, the search space for determining the best set of heterogeneous cores for a given workload gets very large very quickly. Moreover, even if an exhaustive search were tractable, real processors must execute applications developed after the processor's introduction and run existing applications on new inputs. Thus, the application phases used to train Raven are described solely in terms of their architecture-independent features, and in Section 5 we will closely examine the sensitivity of Raven-derived processors to workloads divergent from the original training set.

Below, we discuss the key steps in building a fast and accurate model for accurately selecting and mapping among heterogeneous cores.

Defining a design space One of the first steps in designing Raven revolves around determining the architecture features, or knobs, that the user will have available in the processor generation search space. Since the goals for a heterogeneous processor tend to revolve around some combination or power, energy, and/or area, care should be taken to select a set of knobs that addresses at least some of these concerns.

High impact changes, such as issue width and size and type of the pipeline, are currently the focus of current heterogeneous design efforts, since they can greatly influence both the energy consumption and the performance of a given core. Thus, they should nearly always be included in any search space. Other aspects such as cache structure and functional units should be strongly considered as well. Smaller power draws such as branch prediction and prefetchers can be modeled as well should the need arise, but the total number of dimensions and number of choices must remain modest in order to keep both hardware library design effort and model training times tractable.

Once the knobs are selected, one of the first steps to perform is to acquire power and area information based on a target architecture. These are numbers that are used by Raven in performing comparisons in the various knobs options and thus should be as accurate as possible to the architecture one is building on. For this, we have used the McPAT [19] power and area modeling framework to calculate the Δ-power/area changes in a processor depending on the knob settings (e.g. toggling the number of Integer ALUs while keeping all other values constant). This will aid in getting power and area information that is reasonably accurate for what the modeling data would provide without the need to run the framework for

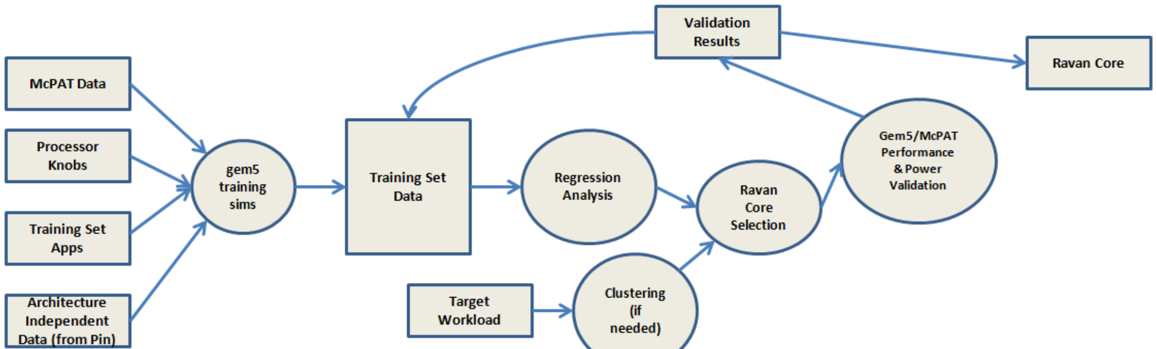

Figure 2: ***Procedural Flow of Raven Design:*** **Raven is designed so that processor designers can tailor their training sets to the types of applications they expect to see on their processors, then continue to train during subsequent core design runs.**

every single possible configuration.

Selecting an optimization function There are many optimization functions one may wish to choose for an approach like Raven. For expedience in evaluation, our current implementation of Raven optimizes for performance/Watt, but it could easily be extended to take the optimization function as an input parameter.

Selecting input parameters In order to make Raven scalable and easily usable, there needs to be a concise representation of the types of workloads the user will want to try to optimize for. The reason for this is that it influences the rest of the Raven process as well as gives an initial design space for our training sets and equation formulation. We choose a set of architecture-independent features because these can be easily gathered for any application to be tested on Raven. We acquire these architecture-independent datapoints using the Intel Pin [20] program and specifically the MICA pintool [14] for all applications within the workload. The goal would then be to pick the relevant data to the knobs that have been selected and select a training set based on these values. These values are also saved as they are inserted into the training set to help generate the performance equations.

Generation of Training Data and Regression The accuracy of Raven's predictions will depend on the representativeness of its training set. Modeling data as accurately as possible is key to getting accurate heterogeneous cores. To this end, we generate simpoints [27] for each of the applications in the workload irrespective of whether the apps are used in training either the regression or the processors. It is well understood that applications generally have multiple phases, so in order to have the most accurate representation of the workload simpoins will need to be gathered for each application in the workload. For all simulations we use gem5 [4], and we were able to generate the basic block vectors necessary for simpoint analysis as well as create the checkpoints necessary to speed up future runs. For all application runs we select a simpoint and run for 100 million instructions, and this is performed for all runs, be they for training set generation or processor validation.

In order to generate the training set data, a set of applications needs to be selected that best represents the workload as a whole. This can be determined by examining the data from MICA and looking for varying application data so the training set can cover an adequate surface space. Gem5 simulations are then run on the training applications, varying one of the knobs at a time while keeping the rest of the processor constant. Care should be taken to select some sort of default or "Midway" processor that can serve as a baseline for future calculations, and as a processor configuration that this training data revolves around.

Once each application has been processed for both the simulation-based processor data and the architecture independent variables, we combine the two into a single training set and send it through a linear regression program within the Weka machine learning suite [13]. The standard analysis will result in a single equation that contains both the various knobs (x_1, x_2,....x_j) and the various arch-independent data points (y_1, y_2,....y_k) resulting in:

$$perf = \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_1 y_1 + \beta_2 y_2 + \ldots + \alpha \quad (1)$$

This performance equation will estimate the performance across the knob space effectively while holding the application data points fixed for any particular core fitting exercise. While this may work for basic situations, additional complexities do arise in constructing these regression and we will address this in section 4.

Processor Prediction Once the above steps are completed, Raven is ready to generate processors. A workload can be selected at either a one process per core granularity or on the cluster level. Clustering is generally desirable in the case where there are more applications than available cores. To perform this, we run a K-means clustering on the application characteristics, but any clustering algorithm will do as long as the number of clusters is no more than the number of cores. In this particular case, we will then use these "meta apps" as the input for Raven rather than relying on the application that is the closest to the cluster median. The list of inputs into Raven are as follows:

- McPAT data for the Δ-changes in the area and power of the processor
- Linear regression for performance
- EITHER the application specific or cluster metadata for each core to be tested
- An equation denoting the goal (i.e. maximizing for performance/watt, performance/area) etc.
- The list of knobs that are being explored and the values within the knobs
- The configuration of the "default" core
- Number of processors to validate after Raven analysis completes

After setting up the necessary data structures, Raven performs an entire sweep of the knob search space, calculating the projected performance, area, and power for each possibility. The area and power metrics are determined by increasing the projected value based on the introduction of more components, while the performance is determined using the regression equations given in the input. We perform this exhaustive search because estimating for these knobs is still much quicker than performing individual simulations, while also avoiding any pitfalls of pruning the search space unnecessarily. Some basic pruning decisions are made (e.g. there is little need to model varying instruction window sizes on an in-order processor), but nothing that is a reasonable and unique core is left out. Once this search is complete, Raven will select the top processors that achieve at least the same predicted performance/watt as the default core, then outputs the results to a file for further analysis.

The cores that were selected by Raven are run through gem5 to get accurate performance measurements beyond that of what the model can perform. These numbers are coupled with McPAT data generated from the same processors to allow us to select the best core within the limited search space for the given goal. Raven then reports the processors for each application/cluster to the user. One final step consists of adding the results of the gem5 runs to the training set, which allows the user to continuously tune the regression to their workloads without having to retrain from scratch every time. While in the end we do perform gem5 simulations, they remain in the order of tens of simulations versus the order of tens of thousands for even basic heterogeneous core search spaces.

4. Refinement and Setup of Raven

In this section we describe the refinements made to Raven in order to tailor the tool for our purposes, and provide some insight on whether adjustments would need to be taken in similar situations. We then look at the steps we took to set up our experiments, thus validating that our design is sound and allowing for the evaluation of our Raven cores against other hardware.

4.1. Refinements

One of the key heuristic changes made was acknowledging that a single regression equation for performance can lead to incorrect predictions based on training set data. One key example that we encountered was the introduction of floating point applications and a floating point (FP) unit knob to the knob list. While the regression would accurately penalize FP operations that would run without an accompanying FP ALU in the processor, it would also assign the same penalty for integer only workloads that would try to save power by eliminating the unit. This sort of interdependence was not unique to integer vs floating point, but it had one of the largest impacts on the predicted performance. The solution to this involved two parts. First, The regression components were separated: requiring that any directly FP-dependent variables (both from knobs and independent variables) were omitted from the int-regression and the reverse was true for the FP-regression. Then, because the int regression would not over-provision performance due to a lack of knowledge, the two regressions are weighted based on the ratio of floating point operations to total operations.

$$perf = ((1 - FP_{inst}) * Reg_{int} - FP_{const}) + (FP_{inst}) * Reg_{fp} \tag{2}$$

This new equation gave much more accurate numbers for our out-of-order execution predictions, but the equation was still lacking for in-order executions. The model was not accurately accounting for memory-bound applications that were spending much of their time waiting for data from DRAM. Once again, the regression gave a performance boost for out-of-order execution, but did not take this particular scenario into account, due in part to the large variations in cache behavior between the applications. One architecture independent metric used was the average reuse distance of memory addresses, and noted that there was a direct correlation between this metric and the performance penalty of switching to in-order cores. The solution was then add a component to the regression, where we add a fraction of the data reuse variable only to in-order processors. This adjustment solved the issues with in-order performance while at the same time not affecting out-of-order performance, which was not affected by this issue.

$$Reg_x = Reg_x + (DataReuse * \beta) * ((inOrder == 1?)0 : 1) \tag{3}$$

Lastly, it was noted that the regressions did not use all of the architecture independent variables or had β coefficients that were negligible. We decided to keep these variables rather than discard them in order to create more informed clusters during the clustering phase of Raven.

Component	Settings
L1 Cache	16KB/2-way I-D, 32KB/2-way I-D, 64KB/4-way I-D
L2 Cache	64KB, 256KB, 1MB
Int ALUS	1, 3, 6
Mul/Div ALUS	1, 2
FP Units	0, 1, 2
Instruction Window Size (for OoO models)	64, 128, 200
Issue Width & Execution Model	1-IO, 2-IO, 4-IO, 4-OoO, 8,OoO

Table 1: Knob Selection for Raven Experiments

Application	Suite	Purpose
gcc	SPECInt 2006	Code Compilation
libquantum	SPECInt 2006	Quantum Simulation
namd	SPECFP 2006	Molecular Dyanmics
milc	SPECFP 2006	Lattice Computation
freqmine	PARSEC 2.1	Itemset Mining
swaptions	PARSEC 2.1	Monte Carlo Sim
fft	MiBench	Fast-Fourier Transform
stringsearch	MiBench	String Comparison

Table 2: Training Set Applications for Raven Experiments

4.2. Setup of Raven

With the above adjustments in place and the component design complete, we will now describe our setup for the experiments and evaluation to follow. The attempt was to have as broad of a spectrum of applications as possible. The idea is to perform both breadth testing by looking at a processor that was developed across all possible suites as well as doing some analysis on processors specifically designed with one processor in mind. As a result, we drew from the desktop, embedded, and HPC communities, selecting from the following benchmark suites:

- Desktop: SPEC2006 [28]
- HPC: Parsec 2.1 [3]
- Embedded: MiBench [12] and Coremark [9]

The next step was determining the knobs that we would iterate against. Since current heterogeneous designs focus on issue width and execution model, it seems like that would be a reasonable starting point. Our decision points for which issue width and execution model combinations to analyze were based on previous studies that look at the energy-performance tradeoffs of these types of processors [2]. The next focus is on high power components that could be scaled in various ways and have meaningful impacts on performance We ended up selecting the various ALUs, cache sizes, and the instruction window of the out-of-order models as our additional knobs to test with. Table 1 shows our detailed selection and settings for each knob.

The selection of the knobs also allowed us to see what kinds of architecture independent variables to use. It was clear that we needed a breakdown of what instructions were actually executing as well as detailed information related to the memory subsystem. We wanted to select a series of metrics that gave a detailed outlook as to how the application performs, even if some of the metrics are used only for clustering purposes. In the end we selected the following independent variables:

- Integer, Floating Point, Branch, and Memory Access Instruction Rates
- How much memory passes through the processor in a given simpoint (its footprint)
- Average data reuse and the percentage of data reuses in a short distance
- Register operand average
- Average register producer-consumer rate

With this information we can collect both our McPAT data and select a base processor to test the rest of our candidate processors against. While a detailed description of the processor can be found in table 3 we essentially selected the midpoints for all of our knobs and focused on a 4-wide OoO processor in order to provide an aggressive performance/watt target that did not penalize against compute-bound workloads (much like current consumer processors) This processor, henceforth known as Midway, is used to normalize our performance-energy-area tuple so we can focus on improving from a hypothetical homogeneous core model making basic decisions on optimizing for performance/watt rather than try to normalize around an extreme processor that may or may not be trying to achieve that goal.

Finally, as far as the training set goes, we selected two representative applications from Parsec and MiBench, as well as 2 from SPECInt and SPECFP. Table 2 highlights the processes chosen as well as their basic functionality. The goal was to give the training set a wide variety of potential scenarios so it can make reasonable approximations for new applications that may not fall exactly within the training set. With the heuristic adjustments made to the performance equation discussion in Section 4.1 the training set proved to be adequate for our evaluation.

5. Evaluation

In this section we will discuss the results of experiments performed to determine the viability of the Raven model. We first examine using a general-purpose target workload with Raven to construct the *Raven-G* CMP and evaluate Raven-G across the applications of several benchmarking suites. We then explore how optimizing for a domain-specific workload affects core selection and how sensitive those processors (*Raven-M* and *Raven-S*) are to new applications entering their workload. Lastly we look at workload-level statistics to show the overall effectiveness of Raven across all test cases.

Component	Midway	8-OoO	1-IO	Tag-Team:Large	Tag-Team:Small
Width and Execution Model	4-OoO	8-OoO	1-IO	4-OoO	2-IO
L1 Cache	32 KB-2 Way	64 KB-4 Way	16 KB-2 Way	32 KB-2 Way	32 KB-2 Way
L2 Cache	256 KB	1 MB	64 KB	1 MB	1 MB
Int ALUS	3	6	1	3	1
Mul/Div ALUS	1	2	1	1	1
FP Units	1	2	1	1	1
Instruction Window	128	200	N/A	128	N/A

Table 3: Baseline Processors (The *Midway* processor is what graphs and data are normalized to for all metrics.)

5.1. Experiment Set-Up

Processor	Target Applications
Raven-General (Raven-G)	basicmath, blackscholes, canneal, coremark, mcf, *milc*, xalan, zeusmp
Raven-MiBench (Raven-M)	adpcm, basicmath, blowfish, coremark, crc, *fft*, *stringsearch*
Raven-SPEC (Raven-S)	*gcc*, *libquantum*, mcf, *milc*, namd, povray, soplex, xalan

Table 4: Workloads for Evaluation Experiments (Applications in training set are *italicized*)

In order to test both the breadth and the depth of Raven's capabilities, we have constructed three target workloads, each containing eight applications. Table 4 lists the applications in each workload. While some of the training set applications are present in the target sets, the goal was to create processors based on a majority of new programs not present in the regressions. We cap maximum heterogeneity in our Raven cores to four core-types per Raven CMP. Since we have twice as many applications in the workload, we perform K-Means clustering with K=4 to produce up to four sets of meta-application characteristics, and use those values as the input to Raven. We then evaluate the selected Raven CMPs and our baseline CMPs on both the target workloads, and workloads disjoint from the target set constructed from other benchmarks in the same domain.

Table 3 details the baseline designs we compare our Raven cores against. We use these cores as the baseline in all experiments. They include the Midway processor that we use to normalize our metrics, an 8-wide out of order (OoO) core featuring the maximal resource allocations in all dimensions in our design space, a 1-wide in-order (IO) core featuring the minimum resource configuration in our design space, and a two processors combination, *tag-team*, that reflects current design philosophies for heterogeneous design by coupling a 4-wide OoO core and a 2-wide IO core. For tag-team results, we assume that every application is always scheduled to the core on which it achieves superior performance/watt.

We evaluate the different designs across several workloads using performance/watt as our optimization metric. The performance/watt metric is a good measure of fitness because it simultaneously offers an easy intuition on how these processors can either scale up in throughput or provide savings under a fixed budget.

5.2. Raven-G: General Workload

We start by running Raven over a general workload. Table 5 shows what the selection model chooses as the best processors. There are four distinct cores that differ primarily in two dimensions, reflecting the amount of floating point computation and the memory access patterns of the applications. Figure 3 shows the performance/watt results of running the target workload on all the baseline processors as well as Raven-G. Here we can see that across the workload, and for most applications, Raven-G provides an efficiency improvement over the other designs. When suboptimal choices are made, the penalties are low.

We next investigate whether the Raven-G processor can maintain its advantage over a larger workload. We select various applications from across our domain suites to form a "general" workload. We map these applications to the cores in Raven-G by a simple euclidian distance metric between the architecture independent feature vector for the application to be mapped and the mean architecture-independent feature vector from the cluster of target applications that drove core selection. The results of this are found in figure 4.

For certain workloads such as lbm, we do exceedingly well due to its placement on an in-order core, and even in cases where we perform worse than the Midway core we still outperform all other options, including the Tag-team heterogeneous core. While Raven-G is not always the most efficient design, it achieves its goal of "general" purpose by being more efficient on average than the other available cores. Raven-G, improves performance/Watt efficiency by 9% over the Midway core and by 18% over the tag-team core. While it is less surprising that we beat the extreme cores, they serve as a reminder that neither the maximal nor minimal extrema tend to provide superior efficiency. While the Midway core provides better energy efficiency that the Tag-team system, this is largely due to the large caches present in the heterogeneous system. These caches were designed with a worst-case scenario with general purpose applications in mind where applications need the large

Component	Raven-G:1	Raven-G:2	Raven-G:3	Raven-G:4
Target Applications	basicmath, blackscholes, zeusmp	milc	canneal, xalan, coremark	mcf
Width and Execution Model	4-OoO	4-IO	4-OoO	1-IO
L1 Cache	32KB-2 Way	32KB-2 Way	32KB-2 Way	16KB-2 Way
L2 Cache	64 KB	64 KB	256 KB	64 KB
Int ALUS	3	3	3	1
Mul/Div ALUS	1	1	1	1
FP Units	1	1	0	0
Instruction Window	64	N/A	64	N/A

Table 5: Processor Cores within Raven-G: Raven for General Workloads

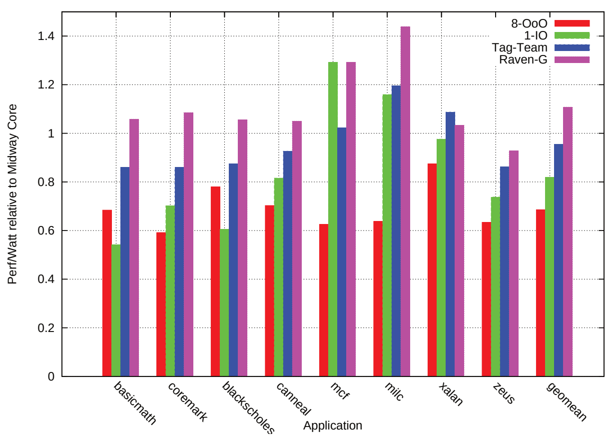

Figure 3: ***Raven-G Perf/Watt on Targeted Applications:*** **We evaluate the performance/Watt of a Raven processor designed for a general workload. Raven-G consistently provides better efficiencythan any of our other reference processors on its target workload.**

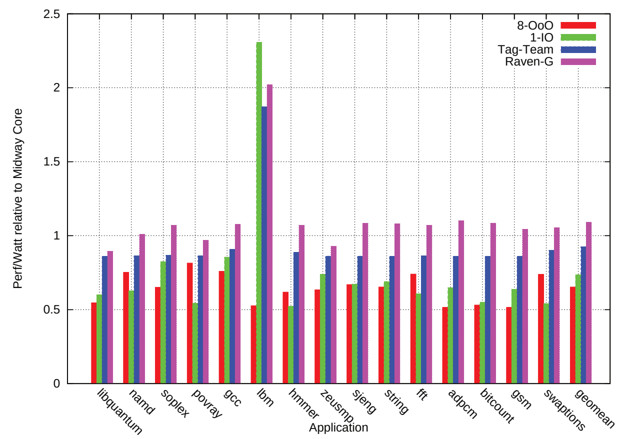

Figure 4: ***Raven-G Performance on broad suite:*** **Applications from all three domains performed favorably on the Raven-G processor as shown here. Efficiency did not suffer greatly running new applications, even when outliers such as lbm are removed.**

cache, something Midway and Raven do not account for given the workloads presented.

Collectively these results show that for a general workload, Raven-G can improve the performance/watt by a measurable amount with very little overhead in actually generating the core designs. Secondly, it shows that expanding the asymmetry of heterogeneous CMPs continues to improve performance/watt beyond current designs. Even simple decisions like removing a floating point unit for integer based workloads can have a significant improvement as experiments in McPAT showed a relatively large leakage wattage (around 0.25 W, which for an in-order core is a source for significant power savings).

5.3. Raven-S and Raven-E: Sensitivity Analysis

In order to see the effects of different single-domain suites on Raven, we decided to run sensitivity analysis experiments using the MiBench and SPEC benchmarks suites.

First, we tackle MiBench and its embedded system-minded applications. While these applications were used in the general test and performed relatively well, there are some key differences when they are tested alone. Many of these benchmarks have very similar independent characteristics, so it is more difficult to get a good sense of the sources of variation in these applications. Due to the similarities, the clustering to produce Raven-M ends up with greater overlap than in Raven-G, and Raven select three processors to cover the four clusters (two clusters mapped to the same processor), as outlined in table 6.

As a result of the vastly different workload from the general case, Raven selects all 2-wide in-order cores. Intuitively, this makes some sense; a series of embedded benchmarks should be expected to perform well on lightweight cores and see only limited improvement from stronger processors. Figure 5 shows the results of running the target applications on the Raven-M processor. Once again, for most cases Raven-M provides better efficiency than the other options, and the patterns are similar to the the results seen for Raven-G.

Component	Raven-M:1	Raven-M:2	Raven-M:3
Target Applications	adpcm, crc, blowfish, coremark	basicmath, fft	string
Width and Execution Model	2-IO	2-IO	2-IO
L1 Cache	32 KB-2 Way	32 KB-2 Way	64 KB-4 Way
L2 Cache	64 KB	64 KB	64 KB
Int ALUS	1	1	1
Mul/Div ALUS	1	1	1
FP Units	0	1	1

Table 6: Processor Cores within Raven-M: Raven for MiBench Workloads

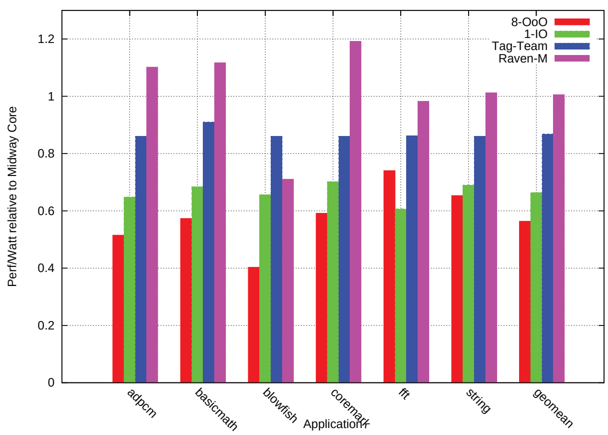

Figure 5: *Raven-M Perf/Watt on Targeted Applications:* Many of the MiBench benchmarks had similar characteristics, leading to a very lightweight set of cores. While performance hits were greater for some applications over others, there was still a similar trend to the Raven-G case.

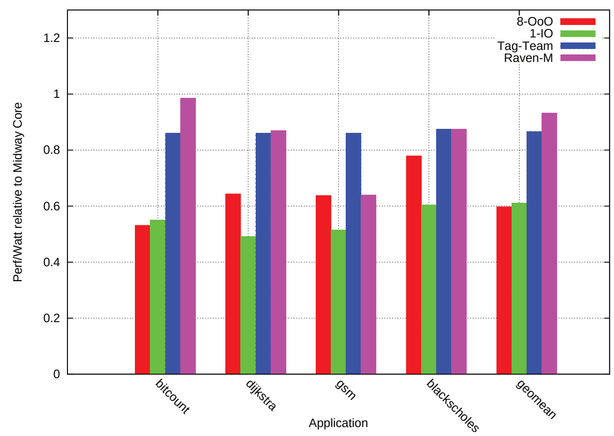

Figure 6: *Raven-M Perf/Watt on New Applications:* Even though the target applications proved to be similar, it is possible for multiple programs that are not the targets for Raven-M to not perform as well as the targeted applications. Since MiBench has low instruction count benchmarks with a limited number of behaviors, subtle changes in the architecture can have large effects on performance.

Looking at the performance of applications outside the target workload of the Raven-M processor, the story takes a slightly different turn, as shown on figure 6. Other applications within MiBench show that Raven-M continues to do as well or better than tag-team on average, but not to the degree we saw in the Raven-G experiment. We also included a separate offender application, blackscholes, from the PARSEC benchmark suite, to show what might happen if the workload begins to shift. It is worth noting that applications that we expect to perform well on in-order cores, such as lbm, milc, and mcf, were not selected for consideration but if the workloads shifted in a more memory intensive direction then it is possible that Raven-M would perform far above the expectations outlined here. Still, we perform 7% better than Tag-team, due in large part to the cache structure in that processor that is not present in Raven-M.

Lastly, we look at a Raven processor designed solely around SPEC. We designed our target workload based on previous work on subsetting SPEC [24] in order to try to capture the widest variety of applications within the benchmark suite with the fewest applications. In doing so, we ended up with cores that were nearly identical to the ones found in the Raven-G processor (see table 7). The Raven-S processor shows that it is possible for applications that have "strong" affinity toward particular setups (such as desiring a larger caches or floating point unit) to dominate the decision making process for Raven.

Figure 7 shows what happens when we run these cores on their target workload. We see a very similar story to the Raven-G cores with a couple of exceptions. Thanks to the introduction of libquantum and the fact it shares a core with milc, the core's benefit to milc was decreased by Raven, which led to a relative performance degradation over the Raven-G case. That being said, Raven-S still performed better than all other processors for milc, and libquantum saw significant gains over Raven's competitors.

Component	Raven-S:1	Raven-S:2	Raven-S:3	Raven-S:4
Target Applications	povray, soplex, namd	libquantum, milc	gcc, xalan	mcf
Width and Execution Model	4-OoO	4-IO	4-OoO	1-IO
L1 Cache	32KB-2 Way	16KB-2 Way	32KB-2 Way	16KB-2 Way
L2 Cache	64 KB	64 KB	256 KB	64 KB
Int ALUS	3	3	3	1
Mul/Div ALUS	1	1	1	1
FP Units	1	1	0	0
Instruction Window	64	N/A	64	N/A

Table 7: Processor Cores within Raven-S: Raven for SPEC Workloads

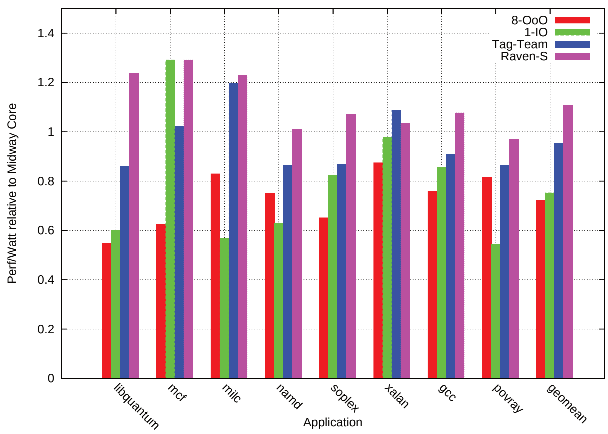

Figure 7: *Raven-S Perf/Watt on Targeted Applications:* We compare Raven-S to our reference processors over the Raven-S target workload. Once again, the SPEC benchmarks that are part of the target workload fare better on average on Raven-S than on the homogeneous cores or on tag-team. The cores selected were very similar to the ones selected for Raven-G.

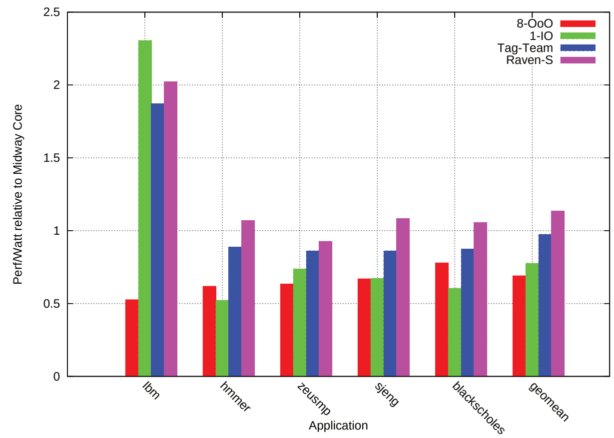

Figure 8: *Raven-S Perf/Watt on New Applications:* Performance-per-watt variations will occur with new applications, but because SPEC is more general and the target set covered a wider range of applications, the new programs had an impact similar to that seen on the Raven-G processor.

Once we check for the sensitivity of Raven-S we end up seeing the opposite end of the spectrum from Raven-M (see figure 8). Applications like lbm (which do very well on in-order cores), cause a large spike in the perf/watt capabilities of many of the processors on the graph. Across the rest of the applications we see similar performance as before, and we note that the outlier from the Raven-M sensitivity example, blackscholes, actually performs better than the Midway core on Raven-S. Overall we see similar improvements to the general core, getting a 14% improvement over Midway (thanks in part to the benefits seen by the memory-intensive apps, and a 16% improvement over its closest competitor, Tag-team.

5.4. Workload Comparisons

Throughout the design of the Raven processors, there have been clearly different cores generated depending on the the target workload provided. Even in the case where the SPEC benchmarks matched closely with the general workload, differences could easily occur based on what applications were selected and how they were clustered. A stark difference was seen between the Raven-M and Raven-G processors, and their performance difference is seen more clearly in figure 9. This is primarily an artifact of optimizing solely for performance/Watt. Future refinements to Raven's optimization function would easily allow us to incorporate notions of minimum performance requirements, or to optimize for alternative (e.g. performance^2/Watt) metrics. For the stated goal of energy-efficiency, the selected processors perform with reasonable tradeoffs.

There is a consistent advantage to using a Raven processor over any of the other options when considering the metric of performance/watt. Even with Raven-M where it performed worse than Midway, the difference was not unacceptable, and it is possible that through a different initial training set the

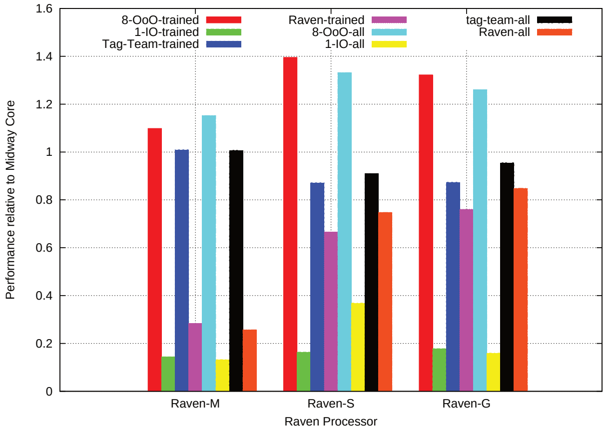

Figure 9: ***Varying Performance:*** **Between various types of processors, there are clear variations in actual performance. This variation can change depending on the workload and the designed cores. Since Raven optimizes solely for performance/Watt, it does not always ensure a strong minimum performance bound.**

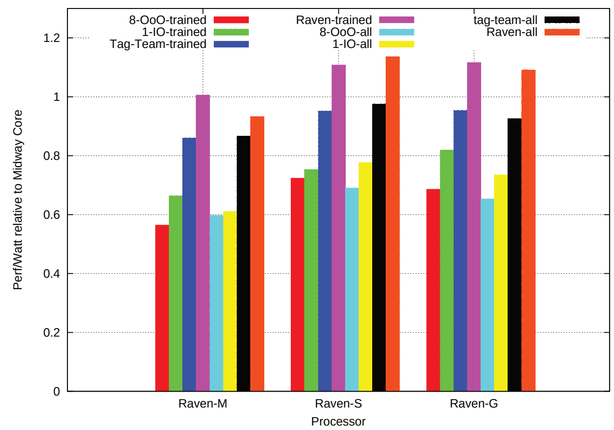

Figure 10: ***Final Workload Statistics:*** **Workload selection plays an important role in designing heterogeneous cores, and while there may be subtle variations in total workload performance when factoring new applications in, the trend shows that Raven processors remain efficient even for applications they neither targeted nor were trained on.**

performance equations can better handle workloads where all the applications share similar characteristics. Amongst the Raven-S and Raven-G processor competitors Raven did better both before and after the introduction of new applications, and this was done on a relatively small subset of the available architecture components. This shows both the promise of the model and the desire to push forward for discovering new heterogeneous designs that can capture both energy efficiency and general purpose applicability.

6. Conclusion

In this work, we have presented a model, Raven, that allows for heterogeneous core design without the need to exhaustively search for good cores for a given set of applications. This is done by creating a series of performance, area, and power equations that predict the performance of a set of potential heterogeneous cores. These are then validated for the user via gem5 simulations and McPAT power analysis to get more accurate values, and these results are then fed back into the regression training set to allow Raven to continue learning about new applications and what works for them. When such processors are actually tested, we generally see an efficiency improvement of between 5-10% over a homogeneous core that was designed with some energy savings in mind and a basic two-way heterogeneous core that reflects current heterogeneous design approaches.

References

[1] ARM, "Big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7," 2011. Available: http://www.arm.com/files/downloads/big.LITTLE_Final.pdf

[2] O. Azizi *et al.*, "Energy-performance tradeoffs in processor architecture and circuit design: a marginal cost analysis," in *Proceedings of the 37th annual international symposium on Computer architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 26–36. Available: http://doi.acm.org/10.1145/1815961.1815967

[3] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.

[4] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011. Available: http://doi.acm.org/10.1145/2024716.2024718

[5] I. Buck *et al.*, "Brook for gpus: stream computing on graphics hardware," in *ACM SIGGRAPH 2004 Papers*, ser. SIGGRAPH '04. New York, NY, USA: ACM, 2004, pp. 777–786. Available: http://doi.acm.org/10.1145/1186562.1015800

[6] N. Clark *et al.*, "An architecture framework for transparent instruction set customization in embedded processors," in *ISCA*. IEEE Computer Society, 2005, pp. 272–283.

[7] N. Clark, A. Hormati, and S. Mahlke, "Veal: Virtualized execution accelerator for loops," in *ISCA*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 389–400.

[8] R. Dennard *et al.*, "Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions," in *IEEE Journal of Solid-State Circuits*, October 1974.

[9] EEMBC, "Coremark." Available: http://coremark.org/

[10] N. Goulding-Hotta *et al.*, "The GreenDroid mobile application processor: An architecture for silicon's dark future," *IEEE Micro*, vol. 31, no. 2, pp. 86–95, Mar./Apr. 2011.

[11] A. Guha *et al.*, "The 10x10 foundation for heterogeneity: Clustering applications by computation and memory behavior," *University of Chicago, Tech. Rep. TR-2012-01*, 2012.

[12] M. R. Guthaus *et al.*, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop*, ser. WWC '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 3–14. Available: http://dx.doi.org/10.1109/WWC.2001.15

[13] M. Hall *et al.*, "The weka data mining software: an update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009. Available: http://doi.acm.org/10.1145/1656274.1656278

[14] K. Hoste and L. Eeckhout, "Microarchitecture-independent workload characterization," *Micro, IEEE*, vol. 27, no. 3, pp. 63–72, 2007.

[15] V. Kontorinis *et al.*, "Reducing peak power with a table-driven adaptive processor core," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42. New York, NY, USA: ACM, 2009, pp. 189–200. Available: http://doi.acm.org/10.1145/1669112.1669137

[16] R. Kumar *et al.*, "Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction," in *MICRO*. IEEE Computer Society, 2003, p. 81.

[17] R. Kumar, D. M. Tullsen, and N. P. Jouppi, "Core architecture optimization for heterogeneous chip multiprocessors," in *PACT*. New York, NY, USA: ACM Press, 2006, pp. 23–32.

[18] R. Kumar *et al.*, "Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance," in *ISCA*. IEEE Computer Society, 2004, p. 64.

[19] S. Li *et al.*, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42. New York, NY, USA: ACM, 2009, pp. 469–480. Available: http://doi.acm.org/10.1145/1669112.1669172

[20] C.-K. Luk *et al.*, "Pin: building customized program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, ser. PLDI '05. New York, NY, USA: ACM, 2005, pp. 190–200. Available: http://doi.acm.org/10.1145/1065010.1065034

[21] C. Luk, S. Hong, and H. Kim, "Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2009, pp. 45–55.

[22] A. Lukefahr *et al.*, "Composite cores: Pushing heterogeneity into a core," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 317–328. Available: http://dx.doi.org/10.1109/MICRO.2012.37

[23] J. Nickolls *et al.*, "Scalable parallel programming with cuda," *Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008. Available: http://doi.acm.org/10.1145/1365490.1365500

[24] A. Phansalkar, A. Joshi, and L. K. John, "Subsetting the spec cpu2006 benchmark suite," *SIGARCH Comput. Archit. News*, vol. 35, no. 1, pp. 69–76, Mar. 2007. Available: http://doi.acm.org/10.1145/1241601.1241616

[25] J. Sampson *et al.*, "Efficient complex operators for irregular codes," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, 2011, pp. 491–502.

[26] D. Shelepov *et al.*, "Hass: a scheduler for heterogeneous multicore systems," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 2, pp. 66–75, Apr. 2009. Available: http://doi.acm.org/10.1145/1531793.1531804

[27] T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications," in *PACT*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 3–14.

[28] Standard Performance Evaluation Corporation, "SPEC CPU 2006 benchmark specifications," 2006, SPEC2006 Benchmark Release.

[29] K. Van Craeynest *et al.*, "Scheduling heterogeneous multi-cores through performance impact estimation (pie)," *SIGARCH Comput. Archit. News*, vol. 40, no. 3, pp. 213–224, Jun. 2012. Available: http://doi.acm.org/10.1145/2366231.2337184

[30] S. Variable, "A multi-core cpu architecture for low power and high performance," *Whitepaper-http://www. nvidia. com*, 2011.

[31] G. Venkatesh *et al.*, "Conservation cores: reducing the energy of mature computations," in *ASPLOS '10: Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*. New York, NY, USA: ACM, 2010, pp. 205–218.

[32] P. H. Wang *et al.*, "Exochi: architecture and programming environment for a heterogeneous multi-core multithreaded system," in *Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*, ser. PLDI '07. New York, NY, USA: ACM, 2007, pp. 156–166. Available: http://doi.acm.org/10.1145/1250734.1250753