Four Metrics to Evaluate Heterogeneous Multicores

Four Metrics to Evaluate Heterogeneous Multicores [1] [2] [3]

Erik Tomusk[4], University of Edinburgh
Christophe Dubach, University of Edinburgh
Michael O'Boyle, University of Edinburgh

Semiconductor device scaling has made single-ISA heterogeneous processors a reality. Heterogeneous processors contain a number of different CPU cores that all implement the same instruction set architecture (ISA). This enables greater flexibility and specialization, as runtime constraints and workload characteristics can influence which core a given workload is run on. A major roadblock to the further development of heterogeneous processors is the lack of appropriate evaluation metrics. Existing metrics can be used to evaluate individual cores, but to evaluate a heterogeneous processor, the cores must be considered as a collective. Without appropriate metrics, it is impossible to establish design goals for processors, and it is difficult to accurately compare two different heterogeneous processors.

We present four new metrics to evaluate user-oriented aspects of sets of heterogeneous cores: *localized non-uniformity*, *gap overhead*, *set overhead*, and *generality*. The metrics consider sets rather than individual cores. We use examples to demonstrate each metric, and show that the metrics can be used to quantify intuitions about heterogeneous cores.

CCS Concepts: •**General and reference** → **Metrics; Evaluation;** •**Computer systems organization** → *Heterogeneous (hybrid) systems;*

Additional Key Words and Phrases: Localized non-uniformity, gap overhead, set overhead, generality, effective speed, single-ISA

1. INTRODUCTION

The amount of heterogeneity available in mobile consumer devices is increasing rapidly. Mobile processors that implement two different types of CPU cores have become common [Greenhalgh 2011; NVIDIA Corp. nd; Samsung Electronics Co. Ltd. nd], and a processor that has three types of CPU cores and a total of 10 cores has been announced [Media Tek Inc. 2015]. These processors implement the same instruction set architecture (ISA) with different microarchitectures, enabling power and performance trade-offs at runtime while supporting an existing code base.

The potential benefits of heterogeneity have led to considerable research interest in design space exploration (DSE) for single-ISA heterogeneous processors. Most DSE methods aim to quickly find a set of power- and performance-optimal core types from a design space [Lee and Brooks 2007; Kang and Kumar 2008; Turakhia et al. 2013]. A heterogeneous processor is not, however, simply a collection of optimal cores. A power and performance Pareto-optimal set can contain hundreds of types of cores, and the processor designer must select some of these for implementation. Selection is a combinatorially complex problem. The designer can be expected to make a small number of intuitive evaluations, but a rigorous selection methodology that is consistent and works at scale requires well-defined metrics. Metrics exist for comparing individual cores and for evaluating system-level throughput, but there has been little work on comparing sets of cores intended for consumer devices.

[1] New paper, not an extension of a conference paper

[3] This work has made use of the resources provided by the Edinburgh Compute and Data Facility (ECDF) (http://www.ecdf.ed.ac.uk/).

[4] Erik Tomusk is partly supported by the ARM Centre of Excellence at the University of Edinburgh and mentored by John Goodacre at ARM. The opinions and findings in this material are those of the authors, and do not necessarily reflect those of ARM.

E.g., it is simple to compare cores in terms of speed and power consumption, but if there are two processors, one with a slow and a fast core, and the other with a slightly slower and a slightly faster core, it is not clear what metric will adequately compare the two. In servers, throughput and efficiency are appropriate metrics, because the tasks and the operating environment are well-defined. Consumer devices face widely varying loads and power limits, and users are much more concerned about responsiveness and battery life than about system efficiency or throughput.

We will introduce four new metrics to quantitatively evaluate heterogeneous cores. The overarching goal of these metrics is to build on existing work in DSE and to enable the development of algorithms that select cores from Pareto-optimal sets. Our metrics focus on the set of cores collectively rather than on the particular features of any single core. They are intended for power-limited consumer devices, not throughput-oriented servers. The metrics are:

— **Localized non-uniformity** to analyze the flexibility afforded by a set of cores,
— **Gap overhead** to evaluate wastefulness and the effects of adding core types,
— **Set overhead** to measure the cost of using one set of cores instead of another, and
— **Generality** to evaluate the level of specialization of a set of cores.

When analyzing heterogeneous processors, great care must be taken not to confuse independent issues. Our metrics are intended to evaluate a selection of core types. They are not intended to evaluate considerations such as *uncore* components (e.g., networks-on-chip or NoCs), scheduling algorithms, or even the business case for heterogeneity. If a designer finds benefits from, e.g., eight core types, but then finds that the NoC hides these benefits, then this should motivate NoC research. Similarly, if an operating system designer struggles to schedule to eight core types, this demonstrates the need for better scheduling algorithms. Finally, the engineering cost of, e.g., eight heterogeneous cores might be prohibitive in a given market, but this can change quickly as markets change and design automation improves. A designer must understand the effects of each system component. Our metrics evaluate one part of the system—the cores.

In section 2, we will provide further motivation for our metrics. Sections 3-6 define the metrics and demonstrate them with examples. Section 7 provides more thorough use cases for each metric, and demonstrates a method of comparing heterogeneity and DVFS (dynamic voltage and frequency scaling). The examples are simple by design to show that each metric's definition lines up with intuition. This increases confidence that the metrics will continue to be valid in circumstances that are too complex to reason about by intuition alone. To demonstrate the relationship of our metrics to more conventional techniques, section 8 compares two of our metrics to execution speed. Section 9 summarizes related work, and section 10 introduces possible future work. Section 11 concludes. The appendix summarizes the source of the example data.

2. MOTIVATION

We will first motivate the need for new metrics by way of example. ED^2 and similar metrics are often used when energy-efficiency is important, but we will show that ED^2 cannot effectively evaluate a set of heterogeneous cores. We will then summarize the four proposed metrics, and describe the goals we have when formulating the metrics.

2.1. ED^2 Example

ED^2—energy-delay-squared product—is a measure of energy efficiency [Martin et al. 2002]. ED^2 is often used to evaluate cores, as it captures the trade-off between energy and execution time. It is therefore easy to assume that the cores on a heterogeneous processor should minimize ED^2. Some cores would be slow and consume very little energy, others

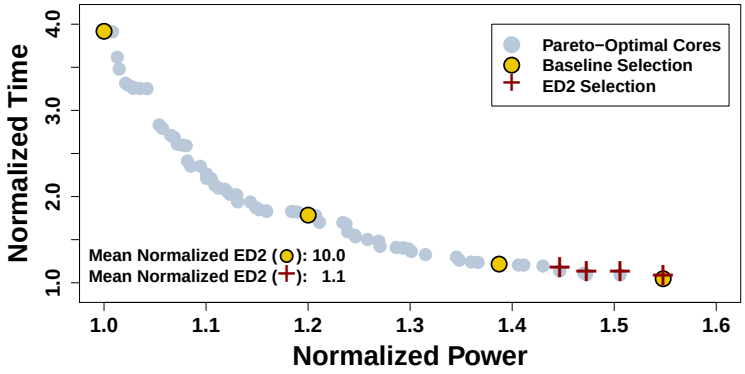

Fig. 1. The ED^2-optimized selection has a favorable mean normalized ED^2 (1.1), but the cores do not give good coverage of the design space. The *Baseline* selection contains both low-power and high-performance cores and covers the design space, but its mean normalized ED^2 is poor (10.0). Smaller values on the axes are better. The complete set of Pareto-optimal cores is shown for reference. The data is for benchmark bm_1.

would be fast and consume large amounts of energy, but all cores would have a similar ED^2. This assumption is evident in, for example, the definition of an *energy-efficient microarchitecture* by Zyuban and Kogge [2000]. We have also made this assumption in our earlier work [Tomusk and O'Boyle 2013].

2.1.1. Problem summary. Figure 1 shows the trade-off between power and execution time in a heterogeneous design space. The gray points are all the power-performance Pareto-optimal cores in the space—there are no cores that have both lower power and better performance than these. We select two sets of four cores each from the Pareto-optimal set: the *Baseline* selection and the ED^2 selection. The *Baseline* selection contains a broad spread of cores, ranging from low-power to high-performance. This selection can run tasks quickly when there is a large amount of power available to the processor, and it can continue running tasks on the slower cores even when power is tightly constrained. The cores in the ED^2 selection are selected to minimize ED^2—i.e., to maximize efficiency. As can be seen from figure 1, efficiency is maximized at the high-power end of the Pareto-optimal set. It is clear from inspection that the *Baseline* selection is a better set of heterogeneous cores for a mobile device. If power is limited even slightly, then none of the cores in the ED^2 selection can be used. Despite this intuition, the efficiency (ED^2) of the *Baseline* selection is nine times worse that the efficiency of the ED^2 selection. We conclude that *ED^2 optimization does not guarantee a good heterogeneous processor.*

2.1.2. Example details. The data used for figure 1 is described in more detail in the appendix. Figure 1 plots CPU cores from our data set for benchmark bm_1 (an AES encryption benchmark). The power, time, and ED^2 values used in the plot are expressed in normalized, unit-less quantities—they have been divided by the lowest values for bm_1 on any core in our data set. 2.0 on the time axis, for example, means that a core takes twice as long to execute bm_1 than the fastest core for bm_1. The average ED^2 of the four cores in the ED^2 selection is only 1.1—10% worse than the best ED^2 possible in the design space. The average ED^2 for the cores in the *Baseline* selection is 10.0. This is 900% worse than the best possible, or 9× worse than the ED^2 selection. Again, this highlight that the features that lead to a good set of heterogeneous cores can easily cause a poor aggregate ED^2.

We expect that a processor designer would use a design space exploration (DSE) methodology to find a set of Pareto-optimal cores, and a core selection algorithm along with our metrics to choose some Pareto-optimal cores for implementation. Our sets of cores—the *Baseline* selection, the ED^2 selection, as well as other selections throughout the paper—

have been picked manually for illustration purposes. The *Baseline* selection represents a reasonable set of heterogeneous cores. We will use it and other selections to demonstrate how our metrics can be used.

2.1.3. ED^2 limitations. The simple explanation for why ED^2 does not help select heterogeneous cores is that there are situations where optimal efficiency is not advantageous. It can often be necessary to decrease power consumption or increase performance at the expense of efficiency. The greater problem with ED^2, however, is that it assumes that energy can be traded for execution time. While this is true for the asynchronous circuits that ED^2 was originally designed to evaluate, it is not necessarily true for whole CPU cores. In our simulations, we have observed that faster microarchitectures are more power-hungry but generally consume less energy, as the increase in execution speed more than amortizes the greater power consumption. The same pattern can also be seen in other works [Givargis et al. 2001; Choi et al. 2004; Raghavan et al. 2013; Czechowski et al. 2014].

The ED^2 example can be summarized as follows: In a heterogeneous processor, it is desirable that the cores provide a broad range of power and performance points so that the best core can be chosen at runtime. If cores are selected based on one metric, like ED^2 or IPC (instruction per cycle), then cores will be selected from only one corner of the design space, preventing flexibility at runtime. Metrics that evaluate heterogeneous cores must take into account the spread of cores in the design space.

2.2. Summary of Metrics

Each of our four metrics evaluates a different aspect of the spread of cores selected from a Pareto-optimal frontier.

Localized non-uniformity measures how clustered the cores are. The ED^2 cores in figure 1, for example, are all in a tight cluster. At runtime, a scheduler will decide which core to use, but because the cores are so similar, this will have very little real effect on performance.

Gap overhead measures the amount of time wasted because of the "gaps" between cores. The *Baseline* selection in figure 1 has four cores and three gaps between them. If there were more cores, then the gaps would be smaller, and a runtime scheduler would have more fine-grained control over power and performance.

Set overhead extends gap overhead to compare two sets of cores. It determines how much more time is wasted if using one set of heterogeneous cores rather than another.

Generality measures whether the selected cores are relevant to all types of workloads, or whether some of the cores are specialized. It evaluates how coverage of the design space changes from workload to workload.

2.3. Goals for Metrics

As described above, the overall goal of the metrics is to evaluate coverage of the design space. We have four further goals for the form that the metrics' definitions should take:

Goal 1: Avoid constants and tuning
Goal 2: Use relative instead of absolute values
Goal 3: Have no dependence on a baseline architecture
Goal 4: Have an intuitive interpretation

Goal 1: If one must choose values for constants or perform some other tuning on a metric, then one can never be certain whether the metric is actually evaluating the processor, or whether it is only evaluating its own tuning. This severely limits the broader applicability of a metric, since a user must prove that constants have been given correct values (where "correct" is poorly defined). It also makes comparing results from two different studies difficult, because both must tune the metric similarly. Our metrics do not depend on constants.

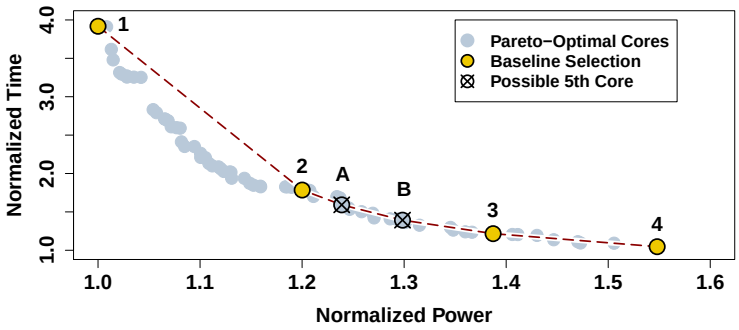

Fig. 2. Non-uniformity helps determine that core B is a better addition to the *Baseline* selection than core A when considering benchmark bm_1. The dashed line shows Euclidean distance between cores.

Goal 2: DSE is reliant on simulators and power models, since DSE precedes processor implementation. The output from these tools is generally accurate in relative terms, but not in absolute terms. I.e., if the size of a microarchitectural structure is doubled, then the relative power for the core can be expected to track appropriately (as reported by a tool like McPAT [Li et al. 2009]). In contrast, the absolute power in Watts is implementation-dependent, and is difficult to predict across a design space. Goal 2 states that metrics should not rely on the absolute correctness of a number from a tool. Metrics should be relative—that is, unit-less quantities.

Goal 3: Goal 3 follows from goal 2. For metrics to be relative, they must compare against some baseline. Works on homogeneous processors generally use a baseline architecture, but in the heterogeneous case, it is unclear what that architecture should be. Our approach is to treat the limits of the design space as the baseline and to express all measurements with respect to each benchmark's best possible values (see section 2.1.2 above). This will become more clear in sections 3-6.

Goal 4: Finally, goal 4 states that metrics should be intuitive to understand. As systems become more complicated, greater effort must be made to ensure that humans can continue to reason about the processors being designed and sanity-check the algorithms used during design.

3. LOCALIZED NON-UNIFORMITY

Our first metric is *localized non-uniformity*. Localized non-uniformity measures how well a selection of cores covers the design space. We describe the intuition behind the metric and provide the mathematical definition. We then discuss how the metric works, and also compare it to other similar metrics.

3.1. Intuition

We begin with the *Baseline* selection of four power-performance Pareto-optimal cores from section 2.1. Let us assume that the processor designer has the option of adding one more core to the selection: either core A or core B. This is shown in figure 2. The two candidate cores are plotted with crossed circles, and the complete set of Pareto-optimal cores is shown in gray for reference. Axes are as described in section 2.1.

Intuitively, there are two reasons to select B over A. The first is flexibility at runtime. B roughly bisects the gap between cores *2* and *3*. The runtime scheduler can trade off power and performance in regular increments, leading to a smoother user experience. In contrast, there is a large performance drop between *3* and A, and only a small drop between A and *2*. The second reason is engineering effort. A almost duplicates *2*. The effort to design A would be better spent designing a different core.

The intuition, then, is that a metric is required that measures whether cores are uniformly spaced along the Pareto-optimal frontier, or whether the cores appear in clusters. The metric should consider worst-case behavior, such as the gap between cores *1* and *2*. It cannot, however, be dominated by worst-case behavior, because then it cannot help select between *A* and *B*. In the next section, we will define *localized non-uniformity* as a measure of how tightly clustered a set of points is.

3.2. Definition

Localized non-uniformity is measured over a 1-dimensional distribution of points. We start with the coordinates of the points in 2-dimensional, normalized, unit-less space (see section 2.1), and flatten them to one dimension using Euclidean distance. I.e., we order the points by one of the axes and place the first point at the 1D origin. Since the points are Pareto-optimal, it does not matter which axis is chosen for ordering. We then measure Euclidean distance between the first and second point, and use the distance as the 1D offset for the second point. This is done for all points. The dashed lines in figure 2 show the Euclidean distance between cores. Equation 1 defines localized non-uniformity, ב (*kaph*), for this 1-dimensional distribution:

$$\text{ב}_{bm} = \frac{(u_1 - R_{min}) + (R_{max} - u_N) + \sum_{i=2}^{N-1} d_i}{R_{max} - R_{min}} \qquad d_i = \left| u_i - \frac{u_{i-1} + u_{i+1}}{2} \right| \tag{1}$$

$$\text{ב}_{bm} = \frac{R_d + \sum_{i=2}^{N-1} d_i}{R_{max} - R_{min}} \qquad R_d = (u_1 - R_{min}) + (R_{max} - u_N) \tag{2}$$

ב_{bm} is localized non-uniformity for benchmark *bm*.
R_d
u_i is the 1-dimensional coordinate of point i.
N is the number of points.
R_{min} is the lower-bound of the range over which ב is calculated.
R_{max} is the upper-bound of the range.
d_i is the distance from point i to the midpoint between its two neighbors.

The value of ב_{bm} is guaranteed to fall in the range [0,1], where 0 is a perfectly uniform distribution, and 1 is a single, tight cluster of points. The range that ב_{bm} is calculated over is defined separately from the points, as u_0 and u_{N-1} are not necessarily the smallest and largest possible values for u.

3.3. Discussion

As the name suggests, localized non-uniformity measures how far points are from being uniformly distributed, but it does so by taking into account only a point's neighbors and not the entire distribution. Equation 1 calculates how far the first point is from the start of the range and how far the last point is from the end. For every other point, it calculates how far it is from being uniformly distributed with respect to its two neighbors (how far it is from the midpoint). The most noteworthy aspect of this formulation is that it is easy to normalize—the sum of all the distances will never be greater than the size of the full range. The formulation also ensures translation-invariance—a set of points will continue to have the same ב value even if it is shifted within the range $[R_{min}, R_{max}]$. Finally, whether points appear clustered or uniformly distributed depends on how large the range $[R_{min}, R_{max}]$ is, and equation 1 reflects this.

Localized non-uniformity is most useful for ranking different selections of the same number of core types in preparation for further analysis. For example, taking the four cores plus core *A* in figure 2 gives $\text{ב} = 0.39$. If core *B* is used instead of *A*, $\text{ב} = 0.33$.

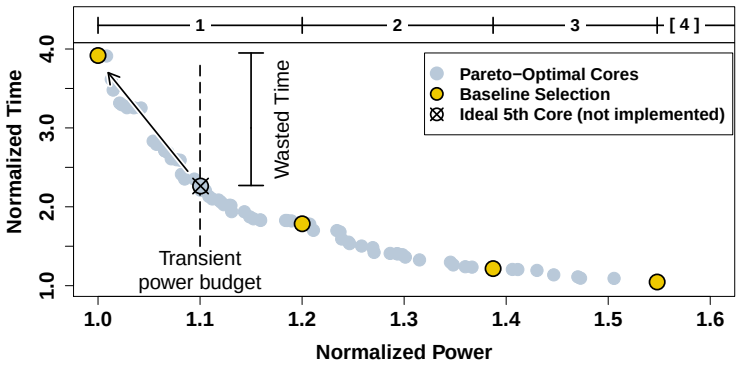

Fig. 3. If the amount of runtime power available to bm_1 is at the dashed vertical line, then execution must take place on the slowest core. The difference in execution time between the ideal, unimplemented core and the slowest core is overhead. Intervals (i in equation 3) are shown along the top.

3.4. Comparison to Alternatives

Localized non-uniformity is similar to various diversity metrics used in genetic algorithms. These metrics are generally inadequate for our use case, since they are used to direct an evolutionary search rather than to evaluate selections from the search result. The extent and uniformity of a distribution are quantified with ϵ-coverage and δ-uniformity by Sayın [2000], but this approach uses two metrics to measure what $\supset$ expresses in one. It is unlikely that converting the power-performance selection problem into a trade-off between ϵ and δ will help a processor designer select cores. Other similar metrics are entropy-based diversity [Farhang-Mehr and Azarm 2002] and Δ-nonuniformity [Deb et al. 2002], but as these are difficult to normalize, they are not readily usable for comparisons. Diversity also requires the user to select a grid to measure density, violating our goal 1. Δ relies on the average distance between points, so a cluster of points can be "averaged away."

Our localized non-uniformity is also similar in intent to the Kolmogorov-Smirnov statistical test (KS test) [Conover 1999, p. 456]. However, the KS test is completely dominated by worst-case behavior. In figure 2, for example, the KS test would focus on the gap between cores *1* and *2*, and would not be able to help select between A and B.

4. GAP OVERHEAD

Gap overhead is a metric for quantifying how wasteful a selection of heterogeneous cores is on average. Localized non-uniformity in the previous section measured how regular the gaps between selected cores are. Gap overhead measures the average effect of the gaps between cores. We first describe the intuition behind the metric, then present its definition, and discuss how it can be used.

4.1. Intuition

One of the primary motivators for heterogeneity is the ability to select an appropriate power-performance point for a task at runtime. If a task has a high priority, then it can be allocated a large amount of power so that it can run quickly. Often, however, it will be the case that only a limited amount of power is available to a task, since there may be other tasks running, the task might have a low priority, or the processor may be close to its thermal limit, etc. In such cases, it is desirable to run the task as quickly as possible without exceeding the amount of power available to it. It is highly unlikely that a heterogeneous processor contains a core that can execute the given task using exactly the amount of power available to it. In most cases, the task must drop down to a slightly slower core, which incurs

a slowdown *in addition* to the slowdown created by the power limit. Gap overhead quantifies this additional slowdown.

Figure 3 illustrates gap overhead with bm_1 and the *Baseline* selection of four cores. In this particular instance, it so happens that there is a small amount of power available to bm_1, the *transient power budget*. The ideal core for bm_1 (the crossed circle) is not implemented, and execution must drop to a lower-power core, as shown with the arrow. As a result, bm_1 runs slower than it ideally would, thereby wasting time. Gap overhead averages this wasted time for all possible transient power budgets.

4.2. Definition

Equation 3 gives the definition of gap overhead. Gap overhead is based on the intervals between cores, as shown in figure 3.

$$\mathrm{GO}_{bm} = \sum_{i=1}^{N} \alpha_i \left(\frac{\max(Y_i) - \overline{Y_i}}{\overline{Y_i}} \right) \qquad \alpha_i = \frac{\max(X_i) - \min(X_i)}{X_{max} - X_{min}} \tag{3}$$

GO_{bm}	is gap overhead for benchmark bm.
X	is the constrained resource (power in our example).
Y	is the wasted resource (time in our example).
$\min(X_i)$	is the minimum value of X in interval i.
$\max(X_i)$	is the maximum value of X in interval i.
$\max(Y_i)$	is the maximum value of Y in interval i.
α_i	is a weighting factor for interval i.
X_{min}	is equivalent to $\min(X_1)$—the lower bound of the range over which gap overhead is calculated.
X_{max}	is the upper bound of the range over which gap overhead is calculated. See below for details.
$\overline{Y_i}$	is an estimate for the average of resource Y in interval $[\min(X_i), \max(X_i)]$ (or $[\min(X_N), X_{max}]$ for the final interval). See below for details.
N	is the number of intervals between the core types that are Pareto-optimal for bm, plus a final interval $[X_N, X_{max}]$.

GO_{bm} calculates how much of the Y resource is wasted in each interval in the average case, and takes a weighted average across all intervals. It is calculated using absolute values for X and Y (X and Y are not normalized). GO_{bm} uses only those cores in the selection that are Pareto-optimal for bm. We refer to the x-axis parameter (power) as the *constrained resource*, since power imposes a hard limit that cannot be exceeded. The y-axis parameter (time) is the *wasted resource*, as this is what is wasted as a result of limits on the constrained resource. The division by $\overline{Y_i}$ removes the units and makes the overhead relative (goal 2). $\overline{Y_i}$ is an estimate for the average value of the wasted resource in interval i. We define $\overline{Y_i}$ as the arithmetic mean of the Y values of all known Pareto-optimal cores in interval i, even if they are not part of the set of selected cores. $\overline{Y_i}$ could also be determined by extrapolation or from first principles, but care must be taken to avoid compromising goal 1. α_i provides a weight for each interval. The range over which GO is calculated is $[X_1, X_{max}]$. X_{max} is the maximum amount of power that will ever be available. For our examples, we define it as the maximum power consumed by any benchmark on the most powerful core, but we expect that a designer will be able to use known physical limits instead.

4.3. Discussion

Gap overhead measures how much extra time execution takes on average because all possible cores cannot be implemented. It helps answer the question, "How much would the average case benefit if another core were added; i.e., should another core be added?" For the example

in figure 3, gap overhead is 0.25—assuming that bm_1 can happen to have any amount of power available to it within the valid range, and assuming all power values are equally likely, then on average, bm_1 will take 25% longer to execute than it would in the ideal case. If the fifth core shown in figure 3 were also implemented, gap overhead would drop to 0.15. By measuring gap overhead, a designer can determine the average benefit of adding another core type, and whether the engineering effort of another core can be justified.

While we have defined α in equation 3 to represent a flat probability distribution for available power, a designer could modify α to weight regions of the design space differently depending on the benchmark. For example, one benchmark might tend to have a small amount of power available because of its low priority, while a high-priority benchmark always has a large amount of power available. The designer could then determine what effect adding a core has on different benchmarks.

One should be aware of two considerations when using gap overhead: First, when evaluating the effects of adding cores, the minimum power core must stay constant. Otherwise, the range over which gap overhead is calculated changes, breaking comparability. Second, gap overhead measures the average case, and is therefore not as sensitive to extremes as localized non-uniformity. For example, returning to figure 2, gap overhead is 0.26 regardless of whether A or B is chosen. Gap overhead and non-uniformity should be used together: GO works best when comparing different numbers of cores; ↄ works best when comparing different selections of the same number of cores.

5. SET OVERHEAD

Set overhead extends gap overhead to measure how much more wasteful one set of heterogeneous cores is compared to another. We describe the intuition behind set overhead and provide the definition. We then discuss set overhead and compare it to gap overhead.

5.1. Intuition

Gap overhead (section 4) quantifies how much of a resource, such as time, is wasted because a heterogeneous processor cannot implement all possible cores. Set overhead extends this concept to compare the cores on two different heterogeneous processors. Let us assume two competing algorithms that both select four core types to implement. The first algorithm picks the cores in our *Baseline* selection. The second algorithm makes an *Alternative* selection. Next, let us assume a runtime scenario like the one in section 4.1—a scheduler determines the maximum amount of power available to a task, and the task must run as quickly as possible while staying within the power budget. Since the two selections contain different cores, one will run the task slower—i.e., with more time wasted—than the other. This quantity is *set overhead*.

Figure 4 demonstrates set overhead. For the demonstration, we have assumed that the hypothetical algorithm that made the *Alternative* selection was not able to find cores as close to the Pareto frontier as the algorithm that made the *Baseline* selection. The amount of power available to bm_1 at a given time is shown by the dashed line. The ideal, but unimplemented core for the given power budget is shown using the crossed circle. From inspection, it can be seen that in this case, the core in the *Alternative* selection is slower, and the *Alternative* selection wastes more time than the *Baseline* selection. *Set overhead* quantifies this additional waste across the range of transient power budgets that is common between the two selections.

5.2. Definition

Set overhead is defined similarly to gap overhead, but the comparison to the complete Pareto-optimal set is replaced with a comparison to another selection of cores.

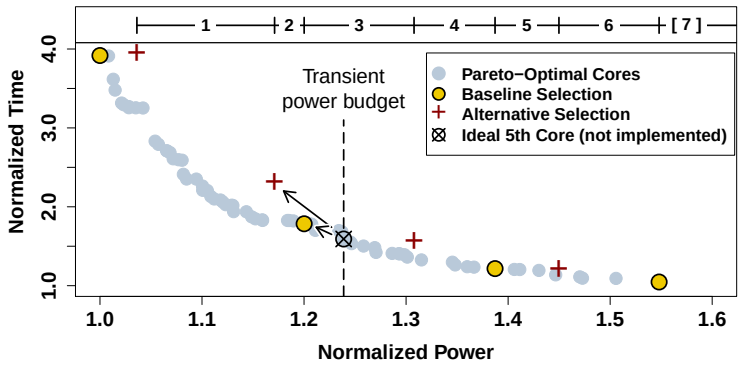

Fig. 4. Set overhead compares two selections of cores. If the amount of runtime power available to bm_1 is at the dashed vertical line, execution will take place on the cores indicated by the arrows. For the power budget shown, the *Alternative* selection wastes more time than the *Baseline* selection. Intervals for equation 4 are shown along the top.

$$\mathrm{SO}_{bm} = \sum_{i=1}^{N} \alpha_i \left(\frac{\max(Y_{A,i}) - \max(Y_{B,i})}{\overline{Y_i}} \right) \qquad \alpha_i = \frac{\max(X_i) - \min(X_i)}{X_{max} - X_{min}} \tag{4}$$

SO_{bm}	is set overhead for benchmark bm.
X	is the constrained resource (power in our example).
Y	is the wasted resource (time in our example).
$\min(X_i)$	is the minimum value of X in interval i.
$\max(X_i)$	is the maximum value of X in interval i.
$\max(Y_{S,i})$	is the maximum value of resource Y for selection S for interval i.
S	is a selection of cores (in our example, either the *Baseline* selection B, or the *Alternative* selection A).
α_i	is a weighting factor for interval i.
X_{min}	is the lower bound of the range over which set overhead is calculated. It is given by $\max(X_{B,1}, X_{A,1})$.
X_{max}	is the upper bound of the range over which set overhead is calculated.
$\overline{Y_i}$	is an estimate for average Y in interval i.
N	is the number of intervals between cores, taking both sets into account. See below for details.

Like gap overhead, SO_{bm} uses absolute (not normalized) X and Y values, and is calculated for each interval between cores, where the intervals are weighted using α and then summed. The definitions of $\min(X_i)$, $\max(X_i)$, X_{max}, and $\overline{Y_i}$ are also identical. The differences are in how the intervals are defined, the definition of X_{min}, and the overhead calculation before the multiplication with α. For set overhead, the intervals are calculated with respect to all core types, regardless of which selection a core belongs to. If the minimum power values for the two selections B and A are not identical, the first interval is discarded (see figure 4). In this interval, only one processor can run a workload, and a comparison is therefore undefined. The subtraction $(\max(Y_{A,i}) - \max(Y_{B,i}))$ evaluates for each interval how much more time one selection of cores takes compared to the other. This is then expressed relative to the idealized time estimate, $\overline{Y_i}$. Since intervals are defined by two sets of cores, either $\max(Y_{B,i})$ or $\max(Y_{A,i})$ will always fall outside i. In figure 4, for example, if available power is in $i = 3$, then the usable core from the *Alternative* selection (A) is the lower-bound of interval $i = 2$.

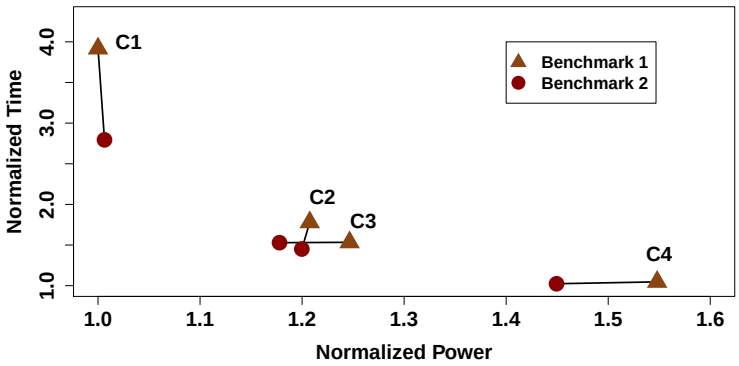

Fig. 5. Four cores are plotted by power and time for bm_1 (triangles) and also for bm_2 (circles). The ordering of cores changes depending on the benchmark, as identified by the black lines. Power and time are normalized to each benchmark's best value.

5.3. Discussion

It is important to note that set overhead is not just a difference of gap overhead values. Set overhead compares two selections, whereas gap overhead compares a selection to the limit set containing all Pareto-optimal cores. For the example in figure 4, the *Baseline* GO is 0.25, the *Alternative* GO is 0.26, but the set overhead is 0.03—in the average case, the *Alternative* selection is 3% slower than the *Baseline* for bm_1. From inspection, one might expect SO to be larger. However, the *Alternative* selection compensates for its slower cores with better coverage of the lower half of the power range, and the speed difference between the sets is negligible in interval 1.

The designer must remember that SO is evaluated only over the common power range—i.e., the probability that available power falls below interval 1 is forced to zero. If this were not the case, then at low levels of available power, one set of cores would complete a task in finite time, while the other set would require infinite time. SO would also be infinite. The designer could well be interested in the fact that one selection can operate at lower power levels than the other, but this is orthogonal to SO.

6. GENERALITY

The final metric, *generality*, evaluates the extent to which core types are specialized to only some workloads, or are generally applicable to all workloads. We provide the intuition behind the metric, give a definition, and expand on its implications.

6.1. Intuition

When determining which core to run a given workload on, there are two extremes to single-ISA heterogeneous processors. In one extreme are processors for which the given workload could reasonably be run on any core type, dependent only on runtime requirements. The cores on such a processor can be said to be *general*. In the other extreme are processors for which a given workload could only reasonably be run on one core type; running on any other type would cause a loss of both power and performance. These cores can be described as *specialized*. Our first three metrics only use the Pareto-optimal cores for a given benchmark. For example, for GO, if there is a core C that is Pareto-optimal for bm_1 but not for bm_2, then it will factor into the GO calculations for bm_1 but not for bm_2. It is then up to the designer to decide how to weight GO for bm_1 and GO for bm_2 when calculating an average GO. In contrast, generality evaluates all cores across all benchmarks using one summary number.

The generality metric measures the ordering of cores. If one were to order all cores by, e.g., runtime power, then the ordering might be the same for all benchmarks, or it might be different (see *monotonicity* in, e.g., Kumar et al. [2006]). The latter is illustrated in figure 5. Four core types, $C_{1\text{-}4}$, are plotted using normalized power and execution time for bm_1. The same four cores are also plotted when executing bm_2. To aid the illustration, this example uses a set of cores different from our earlier *Baseline* selection. It can be seen that the order of C_2 and C_3 is dependent on the benchmark. When all core types take on the same order for all benchmarks, then there exists a set of circumstances that makes each core useful to each benchmark. If, however, there is no clear ordering, it indicates that some cores' microarchitectures are specifically tuned to some tasks but not to others—a core will be finely tuned to one type of workload, but the fine-tuning makes it perform poorly for other workload types. The generality metric quantifies how general a selection of cores is.

6.2. Definition

We use ר (*resh*) for the generality of a processor. It is derived from Spearman's rank correlation coefficient (Spearman's ρ; see e.g. Conover [1999, p. 314]; we use the R implementation [R Core Team 2013]). ר is defined as follows:

$$ר = \frac{1}{W(W-1)/2} \sum_{i=1}^{W-1} \sum_{j=i+1}^{W} \rho\Big(o(X_i), o(X_j)\Big) \tag{5}$$

ר	is generality.
W	is the number of workloads, or benchmarks.
X_i	is the set of constrained metric values (power values in our example) when the core types execute workload i.
$o(\mathcal{S})$	returns the permutation that will sort the elements in set $\mathcal{S}$ into ascending order.
$\rho(P_1, P_2)$	measures Spearman's rank correlation coefficient between permutations P_1 and P_2.

For each benchmark, we order core types by increasing power, and calculate ρ for all pairwise combinations of benchmarks. ρ compares the order of items in two sets. It ranges from 1.0 when the order is the same, to -1.0 when the order is reversed. For ר, ρ values are summed and divided by the number of comparisons. When ר $= 1.0$, the processor is monotonic. As the selection of cores becomes more specialized and generality decreases, ר decreases as well.

6.3. Discussion

Unlike GO, כ, and SO, ר does not need to be minimized or maximized. Processors with both high and low generality are viable, and ultimately it is up to the designer to choose the required level of generality based on the processor's target application. We expect that a if heterogeneous processor is to only have a few core types, then a high generality will be desirable. On the other hand, if very many core types are employed, ר is likely to be small simply because finding many cores that are appropriate for all benchmarks is difficult. Generality can help a designer determine the point at which adding core types to a heterogeneous processor no longer benefits all benchmarks. It can also be used to compare the specialization of two different heterogeneous processors.

7. EXAMPLE USE CASES

We will now present several processor design scenarios, and will demonstrate how our metrics can help a designer evaluate alternative selections of cores and identify sources of potential problems with the selections. These examples use the *Baseline* set of four cores (section 2.1.1), the *Alternative* set of four cores (section 5.1), a *2-Core* set, and an *8-Core*

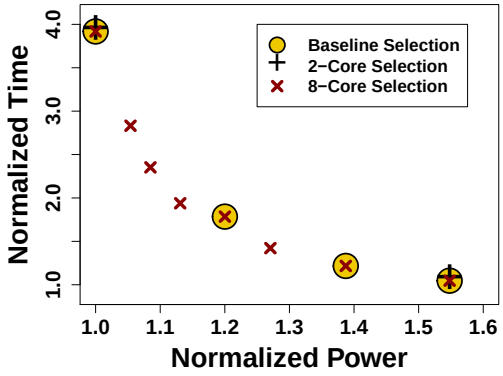

Fig. 6. The *Baseline* selection, and *2-* and *8-Core* selections, shown for bm_1

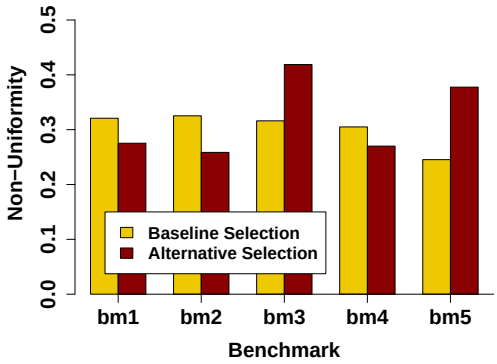

Fig. 7. Localized non-uniformity, ↄ, shows that clustering behavior is worst for benchmark bm_3 running on the *Alternative* selection of cores.

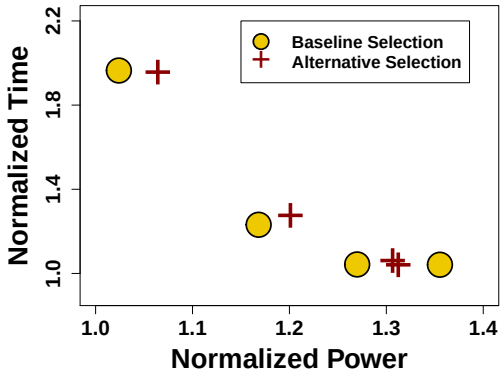

Fig. 8. *Baseline* and *Alternative* selections for benchmark bm_3. The two fastest cores in the *Alternative* set have almost identical behavior.

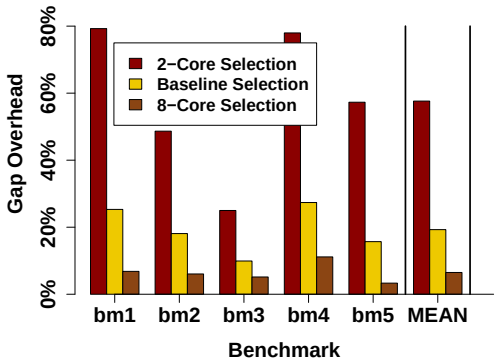

Fig. 9. When moving from two to four to eight cores (figure 6), the average amount of time wasted (gap overhead) decreases from 58% to 19% to 6%.

set. The *2-Core*, *Baseline*, and *8-Core* sets are shown in figure 6. Examples that compare the *Baseline* and *Alternative* selections demonstrate a comparison between two equally sized sets of different cores. Examples that use the *2-Core*, *Baseline*, and *8-Core* sets demonstrate the effects of adding core types to a set. We use the same two benchmarks as the previous sections, bm_1 and bm_2, along with three more benchmarks, $bm_{3\text{-}5}$. The dataset that these cores and benchmarks are drawn from is detailed in the appendix. It is important to note that while we discuss design problems in terms of a designer making decisions, we expect that a human designer will use an automated design space exploration and core selection methodology. The reason quantitative metrics are needed is so that such methodologies can automatically evaluate billions of alternatives and only report the most interesting ones to the human designer. We will present a use case for each of our four metrics. We will then compare a heterogeneous set of cores to DVFS (dynamic voltage and frequency scaling), and use the comparison to demonstrate the difference between the two overhead metrics. For completeness, we include an analysis using the ubiquitous ED^2 metric, which shows that our metrics have greater descriptive power.

7.1. Identifying Redundancy with Localized Non-uniformity

Recall that in section 3.1, we argued that closely clustered cores are redundant—there is little benefit to designing two different cores with almost identical power and performance. The localized non-uniformity metric, ↄ, can be used to quickly identify the worst cases of redundancy. As an example, figure 7 shows ↄ for the *Baseline* selection and the *Alternative* selection for all five benchmarks. Note that ↄ is different for each selection-benchmark combination, as each benchmark interacts differently with each core's microarchitecture. The *Baseline* selection has a ↄ less than 0.33 for all benchmarks. This gives the designer confidence that if the *Baseline* selection is sufficiently uniform for, e.g., bm_1, then it is sufficiently uniform for other benchmarks as well.

The *Alternative* selection has better uniformity than the *Baseline* selection for bm_1, bm_2, and bm_4. However, ↄ for bm_3 and bm_5 on the *Alternative* selection is particularly poor—0.42 and 0.38, respectively. The designer will wish to investigate why non-uniformity is so high for these benchmarks, and whether the use of the *Alternative* selection in a processor is justified. Figure 8 shows the *Baseline* and *Alternative* selections for bm_3. The reason for the high non-uniformity value is obvious: the two fastest cores in the *Alternative* selection have almost identical behavior. Ideally, with four different cores, a workload could be run at four different power-performance points, but the *Alternative* selection can run bm_3 at only three power-performance points. This result can help the processor designer choose between the *Baseline* and *Alternative* selections. It may be the case that bm_3 must have access to four distinct power-performance points, or it may be that the processor's priorities are elsewhere and the *Alternative* selection is sufficient for bm_3.

Given only two sets of cores and only five benchmarks, a designer could find the worst-case clustering of cores simply by visual inspection. However, as the number of selections and benchmarks increases, a human designer will quickly be overwhelmed, while the localized non-uniformity metric can easily scale to any number of benchmarks, any number of cores in a set, and any number of sets.

7.2. Adding Core Types with Gap Overhead

Gap overhead, GO, measures how much more time execution takes because a processor cannot implement cores at all possible power-performance points to match all possible transient power budgets at runtime. GO can be used to determine when to stop adding cores—if an extra core would only marginally reduce gap overhead, then the designer may decide that the engineering effort required to implement the core outweighs its benefits. As an example, figure 9 shows gap overhead for the *2-Core*, *4-Core* (*Baseline*), and *8-Core* selections from figure 6. With two cores, gap overhead is quite high—execution of bm_1 and bm_4 takes nearly 80% longer than theoretically possible. Moving to four and then eight cores, GO drops significantly, and average GO at eight cores is only 6%. The benefit of using four core types rather than two is obvious, but there are diminishing returns to using eight types rather than four. It is up to the designer to determine when the effort to engineer an additional core is no longer justified by a reduced GO. We expect that the designer would use a weighted average of GO values, and would give benchmarks that represent high-priority or frequently executed tasks a higher weight, as the time wasted by these tasks is more important.

GO can also be used to determine which cores to add to a set and in what order. Figure 10 shows GO for the *2-*, *4-*, and *8-Core* selections, as well as for intermediate numbers of cores. For example, there are two paths between the *2-Core* and *4-Core* selections, depending on the order that cores are added to the *2-Core* set. One of these paths is clearly better. One 3-core selection has a GO of 51%, while for the other, GO is 33%. Similarly, for five cores, GO ranges from 15% to 22%. This information is useful to a designer for two reasons: First, the designer may simply wish to use GO to select a core to add to a set. Second,

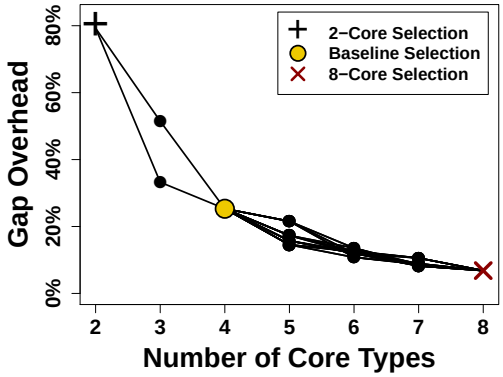

Fig. 10. Adding core types always reduces gap overhead (GO), but some core types reduce overhead more than others. GO is shown for bm_1.

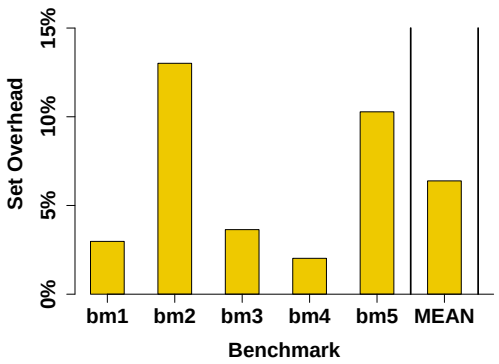

Fig. 11. Set overhead (time wasted) for benchmarks 1-5 when using the *Alternative* selection instead of the *Baseline* selection (shown in figure 4).

the designer may have already decided to implement, e.g., a 4-core processor, but wishes to ship a 3-core processor as an intermediate product while the fourth core is being finalized. GO helps determine which core best complements an existing set and should therefore be designed first.

7.3. Comparing Selections with Set Overhead

Set overhead, SO, measures the average percentage of time wasted because one set of heterogeneous cores is used instead of another. SO can compare two selections even when the selections are composed of completely different cores. For example, a designer may wish to compare two sets, where cores in one set use a more advanced technological feature and are more difficult to design. SO can help the designer determine whether the benefits of the feature justify its costs. To demonstrate SO, we compare the *Alternative* selection to the *Baseline* selection (both shown in figure 4). A visual inspection suggests that the *Alternative* selection is slower, but for the designer to be able to perform an informed cost-benefit analysis, it is important to know how much better the *Baseline* selection is. Figure 11 shows the set overhead of using the *Alternative* selection instead of the *Baseline* selection for the five benchmarks. Set overhead is below 5% for three of the benchmarks, but is as high as 13% for bm_2. Average SO is 6%. I.e., on average, the average task will only be 6% slower on the *Alternative* set than on the *Baseline* set, but bm_2 will be 13% slower. If bm_2 represents a low-priority task, then it may be possible to use the *Alternative* set and save on engineering effort. If bm_2 represents an important task, then the *Alternative* set of cores can be ruled out.

7.4. Identifying Workload Divergence with Generality

The generality metric, ך, compares the ordering of cores to determine whether they are useful to all workloads. If ך is less than 1.0, it indicates to the designer that benchmark behavior has diverged, and different benchmarks are responding differently to the various core types. As an example, for the *2-Core* and *4-Core* (*Baseline*) selections from figure 6, ך = 1.0, but for the *8-Core* selection, ך = 0.76. This shows that it is difficult to select eight cores that are all generally applicable—with eight cores, it is almost inevitable that some benchmarks should never be run on some cores. If a designer adds a new core type to a selection, and ך drops below 1.0, then the designer can know that each new core type will no longer be useful to every workload type. In and of itself, this is not a sufficient reason to stop adding cores. It does, however, highlight to the designer that there will be greatly diminished returns from adding more core types, as each subsequent type will benefit a

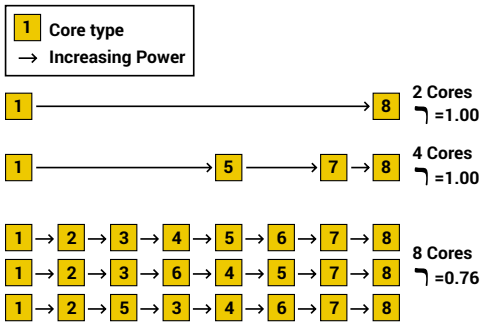

Fig. 12. The *2-Core* and *4-Core* selections (figure 6) can be ordered by increasing power. For eight cores, there are three possible orderings, depending on the benchmark. ר evaluates the divergence in the orderings.

shrinking subset of workloads. Depending on the design goals for the processor, this may or may not be acceptable.

Generality can also be used to gauge how difficult a set of cores is to schedule for. If ר is 1.0, then the runtime scheduler will always know the order of cores from low power to high performance, because the order is the same for all workload types. As ר decreases, it will be increasingly difficult for a scheduler to determine where to schedule tasks. We illustrate this in figure 12 for the *2-*, *4-*, and *8-Core* sets from figure 6. In the *2-* and *4-Core* cases, there is only one way to order cores from low power to high performance, regardless of which benchmark is considered. In the *8-Core* case, there are three possible orderings, depending on which benchmark is used. If, for example, a task running on core 5 must be moved to a more powerful core, it is not immediately obvious whether core 6 should be considered. For some benchmarks, core 6 consumes more power than core 5, but for others, it consumes less. Again, this does not mean that a processor where $ר < 1.0$ should never be designed, but it indicates to the designer that more effort will be required from the operating system developers to take advantage of the processor.

Due to these two considerations, the generality metric acts to temper the drive to add more core types. GO will always show that adding a core type provides at least a marginal improvement to a set; ר shows that the extra core may not be desirable.

7.5. Comparing DVFS and Heterogeneity

This example demonstrates how a designer can use gap overhead and set overhead to choose whether to implement a heterogeneous processor or a homogeneous processor with DVFS. DVFS, or dynamic voltage and frequency scaling, is a method of reducing the power and performance of a core at runtime (see, e.g., Burd and Brodersen [2000]). It is a more mature technology than heterogeneity, but Lukefahr et al. [2014] have shown it to be less effective.

We assume a scenario where a designer must choose between implementing a heterogeneous processor and implementing a homogeneous processor with DVFS. Figure 13 shows the cores for these two alternatives for benchmark bm_2. For the first alternative, we have selected three heterogeneous cores from the Pareto-optimal set. For the second alternative, we have applied DVFS to the highest-power heterogeneous core by extrapolating the voltage and frequency data published by Lukefahr et al. [2014]. We assume 10 DVFS steps, as shown by the black dots along the dashed line.

7.5.1. Gap overhead. The first step in the comparison is to measure gap overhead. Since the two alternatives make use of different technologies, the estimate, $\overline{Y}$, for calculating gap overhead need not be the same for both calculations. For the heterogeneous case, we use the complete set of cores to determine $\overline{Y}$, as described in section 4.2. For the DVFS case,

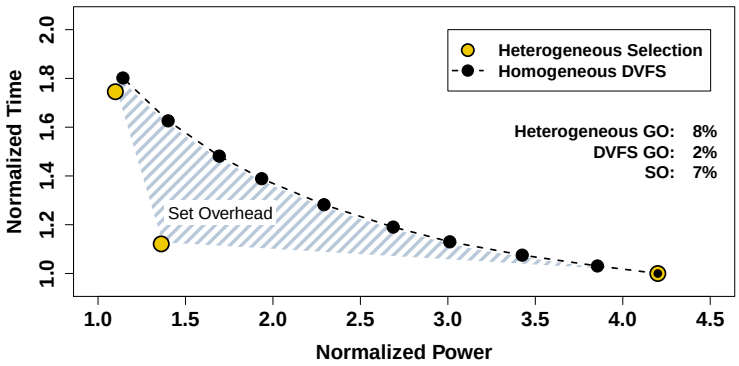

Fig. 13. Set overhead can be used to measure how much faster, on average, a heterogeneous set of cores is compared to a fast, DVFS-enabled core.

we calculate $\overline{Y}$ from first principles. The heterogeneous selection has a gap overhead of 8%; the DVFS core's gap overhead is 2%. In both cases, gap overhead is small, and the benefits of adding additional heterogeneous cores or DVFS steps is minimal. Gap overhead quantifies what is in this case obvious from visual inspection: that the 10 DVFS steps closely approximate an infinite number of DVFS steps, while the three heterogeneous cores are not as close an approximation of the complete Pareto-optimal set of cores.

7.5.2. Set overhead. Gap overhead does not, however, help determine whether the designer should choose to implement the three heterogeneous cores, or the one fast core with DVFS. Gap overhead is lower for the DVFS core even though the heterogeneous set is visibly faster. Instead, comparing the two alternatives requires set overhead, SO. In this case, $\overline{Y}$ in equation 4 is again based on the complete set of cores. SO shows that on average, the DVFS core wastes 7% more time than the heterogeneous set. The shaded region in figure 13 illustrates set overhead.

Set overhead represents the average waste across the entire range of power values. Figure 14 shows the speedup of the heterogeneous selection over the DVFS core at several discrete power levels. If available power is high, then the heterogeneous selection and DVFS core are equally fast. If available power (normalized) is 4.0, then the heterogeneous selection is nearly 10% slower than the DVFS core. However, as available power decreases, the relative performance of the heterogeneous selection improves considerably. When only very little power is available, the heterogeneous selection can be 40% faster than DVFS.

7.5.3. Summary. Knowing the above results, the designer can reason about the return on the engineering effort of implementing either a heterogeneous or DVFS-enabled processor. In this case, if the processor is to be used in a power-limited setting, the heterogeneous option is preferable to the DVFS option. The example shows that GO and SO complement each other. GO helps the designer understand the design space as the number of core types changes. SO helps the designer compare competing sets of cores. SO does not require both processors to contain the same number of core types, and it remains valid when one or even both processors are not heterogeneous. While this example compares heterogeneity and DVFS, the methodology is identical for a heterogeneous processor whose cores are also capable of DVFS. The designer would simply need to supply the metrics with the combinations of core types and DVFS levels rather than just core types or just DVFS levels.

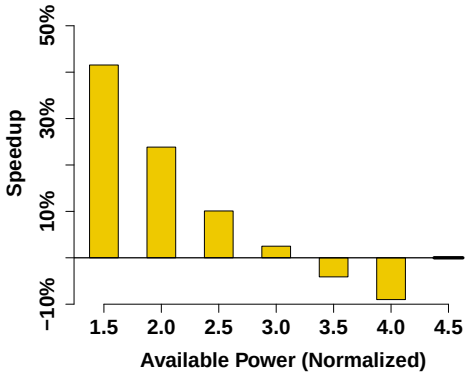

Fig. 14. The speedup of the heterogeneous selection over the DVFS core varies depending on how much power is available. See figure 13.

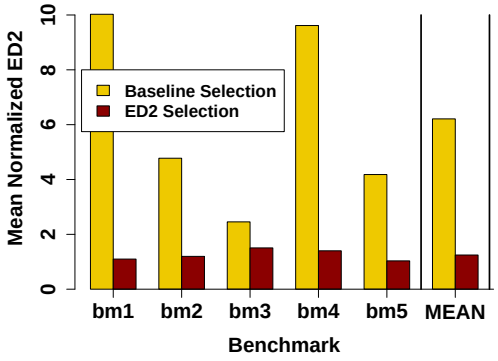

Fig. 15. While the *Baseline* selection contains a better set of cores, average ED^2 is much lower for all benchmarks in the ED^2 selection (cp. figure 1). Smaller ED^2 values are better.

7.6. Limitations of ED^2

For completeness, we extend the example from section 2.1 to demonstrate that the ED^2 efficiency metric does not accurately evaluate a set of heterogeneous cores. Figure 15 shows the mean normalized ED^2 for the five benchmarks running on the *Baseline* selection and on the ED^2 selection from figure 1. For bm_1 and bm_4, mean ED^2 of the *Baseline* selection is nearly 10× greater than the minimum possible. On average, mean ED^2 is five times worse on the *Baseline* selection. Based on this result, a designer could easily conclude that the ED^2 selection is up to ten times better than the *Baseline* selection. However, as noted in section 2.1, the ED^2 selection has many high-power cores and no low-power option. It is difficult to justify designing a heterogeneous processor like this. The reason ED^2 leads to a false conclusion is that in this example, the cores with very low power are not as efficient as some of the faster ones. As a result, a selection optimized for ED^2 excludes these cores even though a heterogeneous processor would benefit from having some low-power cores.

This example also helps illustrate why our metrics should be used instead of ED^2 for comparisons, such as the one with DVFS above. ED^2 can show a set of heterogeneous cores as more efficient than a single core with DVFS, but this will be an incomplete analysis. If the heterogeneous set lacks either slow or fast cores, then under real power constraints, it could be slower than the DVFS option. Localized non-uniformity and set overhead can be used to understand such scenarios.

8. EFFECTIVE SPEED ANALYSIS

In section 7, we demonstrated that a designer can use our metrics to quantify various features of selections of heterogeneous cores rather than relying only on subjective intuition. In this section, we will show that the gap and set overhead metrics correlate to the effective speed of a selection of cores. We will also argue that the overhead metrics provide more insight than an analysis of execution speed.

Existing metrics for measuring the execution speed of a processor, such as IPC (instructions per cycle) or ANTT (average normalized turnaround time, see Eyerman and Eeckhout [2008]) are independent of the amount of power that a processor consumes. These metrics can be used to measure maximum execution speed or execution speed under a fixed power budget. The gap and set overhead metrics assume a power budget that varies probabilistically during runtime, and to accurately compare the overhead metrics to execution speed, speed must also be calculated using a probabilistic power distribution. We calculate the *effective speed* of a set of heterogeneous cores under a changing power budget. The effective

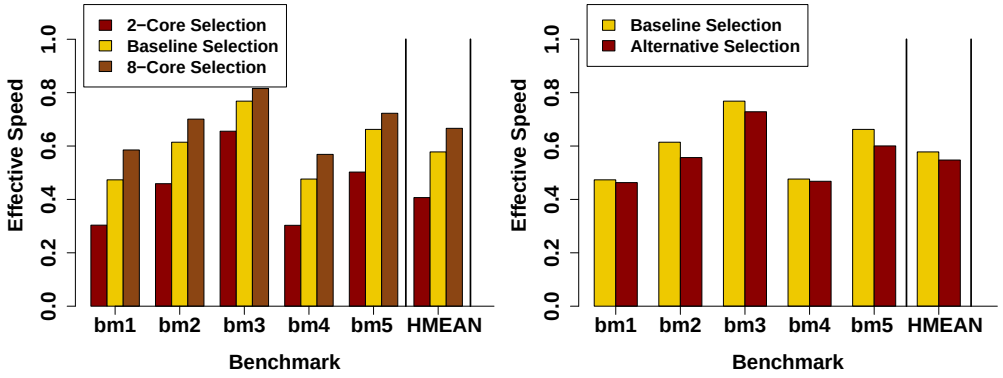

Fig. 16. Effective speed is shown for the *2-*, *4-*, and *8-Core* selections (left), and the *Baseline* and *Alternative* selections (right). Larger values are better. Effective speed confirms the results from GO (figure 9) and SO (figure 11)—more core types improve speed, and the *Baseline* set is faster than the *Alternative* set.

speed is a weighted average of the speed of each core in a set. It is normalized to the fastest possible core. The weights for the average come from the available power distribution—the likelihood that a core will be used. For example, if a selection contains the fastest possible core, and the power distribution guarantees that this core is always used, then effective speed is 1.0. If the slowest core in a set is ten times slower than the fastest core, and the power distribution guarantees that the slowest core is always used, then the effective speed of the set is 0.1.

We calculate the effective speed of the *2-Core*, *8-Core*, *Baseline*, and *Alternative* selections of cores from section 7. Since effective speed is a rate, we use the harmonic mean to average individual benchmark results. Figure 9 showed that under tight power constraints, the amount of time wasted can be reduced by increasing the number of core types. Similarly, figure 16 (left) shows that the effective speed of a set increases with more core types. With two, four, and eight cores, effective speed is 0.40, 0.58, and 0.67, respectively. Figure 11 showed that execution on the *Alternative* selection takes longer than on the *Baseline* selection. Figure 16 (right) confirms this, showing that the average effective speeds for the *Baseline* and *Alternative* selections are 0.58 and 0.55, respectively. We can conclude that a design methodology that minimizes the overhead metrics maximizes execution speed.

We advocate for the use of the gap and set overhead metrics instead of effective speed, because the overhead metrics are simpler to interpret (see goal 4 in section 2.3). For example, if gap overhead is 10%, then the designer immediately knows by how much it is theoretically possible to reduce overhead. In contrast, it is difficult to determine if an effective speed value of 10% is good. 10% could indicate a desirable feature, like the presence of some very low-power cores. It could alternatively indicate an undesirable feature, such as the lack of medium- and high-performance cores.

9. RELATED WORK

One of the earliest works on single-ISA heterogeneity is by Kumar et al. [2003], who use four generations of Alpha cores to reduce the power consumption of memory-bound programs. "big.LITTLE" processing from ARM couples low-power and high-performance cores, and is the first widely available, single-ISA heterogeneous platform [Greenhalgh 2011]. There is commercial interest in extending big.LITTLE technology to three types of cores [Media Tek Inc. 2015]. Kumar et al. [2006] classify heterogeneity as either *monotonic* or *non-monotonic*. Monotonicity is similar to our ך metric. If ך = 1.0, then a set of cores is monotonic; otherwise it is non-monotonic. Monotonicity is a binary classification, whereas ך can be used to differentiate levels of generality.

There is a large body of work on efficiently searching design spaces. These works often try to minimize simulation time using algorithms that carefully select cores to simulate. Kang and Kumar [2008] describe a search methodology for maximizing the throughput of heterogeneous cores under a power and area constraint. A search with similar goals is implemented by Turakhia et al. [2013], but for multi-threaded workloads. Van Craeynest and Eeckhout [2013] explore the trade-off between throughput and turnaround time in a heterogeneous design space. Liu et al. [2011] search a design space by repeatedly synthesizing logic circuits. Givargis et al. [2001] and Pham et al. [2013] consider the design space of an entire system-on-chip rather than a set of cores. There have also been a number of works on using analytical models to explore design spaces [Lee and Brooks 2006; Lee and Brooks 2007; Karkhanis and Smith 2007]. Design space exploration (DSE) is a prerequisite to our work. We assume that a designer can find candidate cores in a design space, and then use our metrics to help select some of these candidates for implementation.

Some authors have combined DSE with core selection. Navada et al. [2013] use a genetic algorithm to select up to four core types to maximize performance. Annavaram et al. [2005] trade off the number of low-power and high-power cores to ensure that program execution stays within a processor's power budget. Guevara et al. [2014] select heterogeneous cores for a data center with the goal of minimizing risk at runtime. Similarly to our metrics, this work recognizes the need for runtime flexibility.

In our examples, we have only considered heterogeneous processors where all cores implement the same ISA (instruction set architecture). There have been works on designing specialized instruction set extensions [Venkatesh et al. 2010; Venkatesh et al. 2011], and even using two completely different ISAs on a processor [DeVuyst et al. 2012]. The design spaces of these processors are significantly more complex than single-ISA spaces. There is, however, no reason why our metrics could not be extended to evaluate selections from these much larger spaces.

A number of different metrics have been used to evaluate processors, both homogeneous and heterogeneous. $\mathrm{IPS}^3/\mathrm{W}$ (instructions-per-second-cubed per Watt) is a common measure of efficiency [Lee and Brooks 2007; Dubach et al. 2010; Navada et al. 2013]. $\mathrm{IPS}^3/\mathrm{W}$ is the inverse of ED^2. Alioto et al. [2012] considers a broader set of $\mathrm{E}^i\mathrm{D}^j$ metrics. However, we have shown that cores selected for ED^2 tend to converge to a small regions of the design space. Using an alternative $\mathrm{E}^i\mathrm{D}^j$ metric will simply lead to convergence in a different region. Another common efficiency measure is EPI—energy per instruction. Annavaram et al. [2005] use EPI to tune a selection of cores to both serial and parallel regions of benchmarks. Karkhanis and Smith [2007] consider trade-offs between four pairs of metrics: area–CPI (cycles per instruction), area–EPI, CPI–EPI, and area–(EPI $\times$ CPI). We have demonstrated our metrics using a power-performance trade-off, but a designer could easily use another pair of metrics. EPI–CPI may be a particularly good fit. The localized non-uniformity and gap overhead metrics were first proposed in a poster abstract [Tomusk et al. 2014].

There is ongoing discussion on the best metrics for evaluating the throughput and fairness of processors. Snavely and Tullsen [2000] define the *weighted speedup* (WS) metric for evaluating SMT and multicore processors. Luo et al. [2001] use the harmonic mean of speedups and the standard deviation of throughputs as fairness metrics. Eyerman and Eeckhout [2008] argue for the use of STP (system throughput; another name for WS) and ANTT (average normalized turnaround time). STP is described as a "system-oriented metric," while ANTT is a "user-oriented metric;" i.e., throughput is important on the system level, but an individual user is more concerned about turnaround time. Van Craeynest and Eeckhout [2013] explore the trade-off between STP and ANTT in a space of heterogeneous processors. Michaud [2013] discusses the inconsistencies caused by some definitions of throughput metrics, and Eyerman et al. [2014] introduce a methodology and several associated metrics for evaluating the throughput of systems. Our metrics are intended for consumer devices

and are therefore of the "user-oriented" rather than the "system-oriented" type. While a data center operator must maximize throughput and work-per-energy, a smartphone user is far more interested in a responsive device with a long battery life, even if it does not always maximize throughput and efficiency.

Selecting heterogeneous cores is similar to statistical sampling used to maintain elitism in multiobjective optimization algorithms (see, e.g., Zitzler and Thiele [1999]). In this context, Sayın [2000] defined ϵ-coverage and δ-uniformity to measure the breadth and uniformity of a selection, respectively. Farhang-Mehr and Azarm [2002] defined an entropy-based diversity metric, and Deb et al. [2002] defined Δ-nonuniformity. All these metrics have a goal similar to our localized non-uniformity. However, these metrics are intended to merely guide an optimization algorithm. They are either difficult to normalize or require the user to tune them, making them unsuitable for comparing selections of cores. The Kolmogorov-Smirnov test (KS test) [Conover 1999, p. 456] can also be used to evaluate the quality of a sample, but as noted in section 3.4, it is not as sensitive as localized non-uniformity.

We have focused on the design aspects of heterogeneous multicores, though as noted in section 7.4, generality can be used to gauge how easy it is to schedule to a heterogeneous processor. An energy-aware scheduler for heterogeneous processors and processors with DVFS is presented by Lukefahr et al. [2014]. Alsafrjalani and Gordon-Ross [2014] propose a scheduler that uses a learning phase to determine which heterogeneous core to schedule a task to. Li et al. [2010] study scheduling for a processor where some cores support additional instruction set extensions.

10. FUTURE WORK

We have introduced four novel metrics for evaluating heterogeneous processors. Possible avenues for future work include incorporating empirical data into the metrics and extending the scope of the metrics.

In defining localized non-uniformity, gap overhead, and set overhead, we assumed that available power has a flat probability density—all values are equally likely. A designer could instead use per-benchmark probability density functions (PDFs) for available power. The PDFs could be based on task priority and thermal considerations. Some workloads might be time-critical, for example, and would always have access to large amounts of power. A processor for a set-top box can expect to have more power available overall than a processor for a smartphone. Incorporating PDFs would simply involve adjusting the interval weights (α-values) in equations 3 and 4, and adjusting the X-coordinates in equation 1. A designer could use different PDFs to target processors for different markets.

We have demonstrated our metrics on a design space where every core can execute every benchmark. As noted in section 9, however, there is growing interest in specialized accelerators that are very efficient for one, well-defined task. Our metrics can also be used to evaluate CPU cores together with these accelerators. If an accelerator cannot execute a certain benchmark, then that accelerator will simply not factor in to the metric calculation for that benchmark. When evaluating processors with large numbers of accelerators, it will be crucial to ensure that tasks that cannot be accelerated are not underrepresented, but that there is adequate CPU resource to run them.

11. CONCLUSION

We have argued that a substantial roadblock to the development of heterogeneous processors is the lack of robust, quantitative metrics for evaluating sets of heterogeneous CPU cores. Such metrics are required both for selecting cores for a processor, as well as for comparing alternative selections. The engineering effort required to implement a processor is enormous, and designers must weigh the benefits of designing an improved processor against the engineering cost. To enable the development of selection algorithms and to motivate further research into metrics for heterogeneity, we have defined four quantities. **Localized**

Table I. Example Design Space

Parameter	Value range	Parameter	Value range
Data cache size	16kB - 64kB	Load queue entries	8 - 64
Data cache ways	1 - 4	Store queue entries	8 - 64
Instruction cache size	4kB - 64kB	Reorder buffer	16 - 128
Instruction cache ways	1 - 4	Branch predictor counter bits	1 - 3
Integer registers	50 - 256	Branch predictor entries	2^{10} - 2^{13}
Floating point registers	96 - 256	Branch target buffer entries	2^{10} - 2^{13}
Issue queue entries	16 - 64	Branch target buffer tag bits	16 - 18

non-uniformity (כ) gauges worst-case behavior and helps rank different selections by coverage of the design space. **Gap overhead** (GO) measures wastefulness and the benefits of adding more core types. **Set overhead** (SO) can be used to compare two sets of cores, homogeneous, heterogeneous, or both. **Generality** (ר) gauges the amount of specialization in a set of cores, and can also be used for ranking. The four metrics are independent—a designer may choose to use one, some, or all of them based on design requirements. This list is by no means definitive. We hope that these metrics motivate the development of further analysis techniques for heterogeneous processors.

APPENDIX

A. EXAMPLE DATA

In sections 3-8, we demonstrated our four metrics using example data. Here, we will provide more details on the source of this data.

We use the gem5 cycle-accurate simulator [Binkert et al. 2011] and McPAT power model [Li et al. 2009] to simulate cores from the design space summarized in table I. The space contains billions of permutations of cores. We randomly select 3000, and simulate five benchmarks from the EEMBC Digital Entertainment benchmark suite [Poovey et al. 2009] on these cores. The benchmarks are *AES*, *CJPEG*, *HUFFDE*, *MPEG4ENCODE*, and *RGB2CMYK*, and correspond to $bm_{1\text{-}5}$ in the text. These benchmarks are examples of workloads that are run on the types of consumer processors that we expect our metrics will be used to evaluate.

Of the 3000 core types, 76 are power-performance Pareto-optimal for bm_1 and 75 are optimal for bm_2. 29 are Pareto-optimal for both. The 76 Pareto-optimal cores make up the Pareto-optimal set shown in gray in figures 1-4. From these Pareto-optimal cores, we manually choose some for our examples. A real DSE algorithm would probably have a more effective way of finding Pareto-optimal cores than random search, and it would automate the selection of cores from the Pareto-optimal set. We hope that our metrics contribute to the development and refinement of such algorithms.

REFERENCES

Massimo Alioto, Elio Consoli, and Gaetano Palumbo. 2012. From energy-delay metrics to constraints on the design of digital circuits. *International Journal of Circuit Theory and Applications* 40, 8 (2012), 815–834. DOI:http://dx.doi.org/10.1002/cta.757

Mohamad Hammam Alsafrjalani and Ann Gordon-Ross. 2014. Dynamic scheduling for reduced energy in configuration-subsetted heterogeneous multicore systems. In *12[12] IEEE International Conference on Embedded and Ubiquitous Computing*. IEEE, 17–24. DOI:http://dx.doi.org/10.1109/EUC.2014.12

Murali Annavaram, Ed Grochowski, and John Shen. 2005. Mitigating Amdahl's Law through EPI throttling. In *Proceedings of the 32[nd] International Symposium on Computer Architecture*. 298–309. DOI:http://dx.doi.org/10.1109/ISCA.2005.36

Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad

Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 simulator. *SIGARCH Computer Architure News* 39, 2 (Aug 2011), 1–7. DOI:http://dx.doi.org/10.1145/2024716.2024718

Thomas D. Burd and Robert W. Brodersen. 2000. Design issues for dynamic voltage scaling. In *Proceedings of the International Symposium on Low Power Electronics and Design*. ACM, 9–14. DOI:http://dx.doi.org/10.1145/344166.344181

Kihwan Choi, Wonbok Lee, Ramakrishna Soma, and Massoud Pedram. 2004. Dynamic voltage and frequency scaling under a precise energy model considering variable and fixed components of the system power dissipation. In *IEEE/ACM International Conference on Computer Aided Design*. 29–34. DOI:http://dx.doi.org/10.1109/ICCAD.2004.1382538

W. J. Conover. 1999. *Practical nonparametric statistics*. (3rd ed.). John Wiley & Sons, Inc.

Kenneth Czechowski, Victor W. Lee, Ed Grochowski, Ronny Ronen, Ronak Singhal, Richard Vuduc, and Pradeep Dubey. 2014. Improving the energy efficiency of Big Cores. In *Proceedings of the 41st ACM/IEEE International Symposium on Computer Architecture*. 493–504. DOI:http://dx.doi.org/10.1109/ISCA.2014.6853219

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (April 2002), 182–197. DOI:http://dx.doi.org/10.1109/4235.996017

Matthew DeVuyst, Ashish Venkat, and Dean M. Tullsen. 2012. Execution migration in a heterogeneous-ISA chip multiprocessor. In *Proceedings of the 17th Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 261–272. DOI:http://dx.doi.org/10.1145/2150976.2151004

Christophe Dubach, Timothy M. Jones, Edwin V. Bonilla, and Michael F. P. O'Boyle. 2010. A predictive model for dynamic microarchitectural adaptivity control. In *Proceedings of the 43rd IEEE/ACM International Symposium on Microarchitecture*. 485–496. DOI:http://dx.doi.org/10.1109/MICRO.2010.14

Stijn Eyerman and Lieven Eeckhout. 2008. System-level performance metrics for multiprogram workloads. *IEEE Micro* 28, 3 (May 2008), 42–53. DOI:http://dx.doi.org/10.1109/MM.2008.44

Stijn Eyerman, Pierre Michaud, and Wouter Rogiest. 2014. Multiprogram throughput metrics: A systematic approach. *ACM Transactions on Architectire Code Optimization* 11, 3, Article 34 (Oct 2014), 26 pages. DOI:http://dx.doi.org/10.1145/2663346

Ali Farhang-Mehr and Shapour Azarm. 2002. Diversity assessment of Pareto optimal solution sets: An entropy approach. In *Congress on Evolutionary Computation*, Vol. 1. 723–728. DOI:http://dx.doi.org/10.1109/CEC.2002.1007015

Tony Givargis, Frank Vahid, and Jörg Henkel. 2001. System-level exploration for Pareto-optimal configurations in parameterized systems-on-a-chip. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*. 25–30. DOI:http://dx.doi.org/10.1109/ICCAD.2001.968593

Peter Greenhalgh. 2011. *Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7*. White paper. ARM Ltd. http://www.arm.com/ja/files/downloads/big.LITTLE_Final.pdf

Marisabel Guevara, Benjamin Lubin, and Benjamin C. Lee. 2014. Strategies for anticipating risk in heterogeneous system design. In *IEEE 20th International Symposium on High Performance Computer Architecture*. 154–164. DOI:http://dx.doi.org/10.1109/HPCA.2014.6835926

Sukhun Kang and Rakesh Kumar. 2008. Magellan: A search and machine learning-based framework for fast multi-core design space exploration and optimization. In *Proceedings of the Conference on Design, Automation and Test in Europe*. 1432–1437. DOI:http://dx.doi.org/10.1145/1403375.1403721

Tejas S. Karkhanis and James E. Smith. 2007. Automated design of application specific superscalar processors: An analytical approach. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*. ACM, 402–411. DOI:http://dx.doi.org/10.1145/1250662.1250712

Rakesh Kumar, Keith I. Farkas, Norman P. Jouppi, Parthasarathy Ranganathan, and Dean M. Tullsen. 2003. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*. 81–92. DOI:http://dx.doi.org/10.1109/MICRO.2003.1253185

Rakesh Kumar, Dean M. Tullsen, and Norman P. Jouppi. 2006. Core architecture optimization for heterogeneous chip multiprocessors. In *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*. ACM, 23–32. DOI:http://dx.doi.org/10.1145/1152154.1152162

Benjamin C. Lee and David M. Brooks. 2006. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 185–194. DOI:http://dx.doi.org/10.1145/1168857.1168881

Benjamin C. Lee and David M. Brooks. 2007. Illustrative design space studies with microarchitectural regression models. In *Proceedings of the 13th IEEE International Symposium on High Performance Computer Architecture*. 340–351. DOI:http://dx.doi.org/10.1109/HPCA.2007.346211

Sheng Li, Jung Ho Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, and N.P. Jouppi. 2009. 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*. 469–480.

Tong Li, Paul Brett, Rob Knauerhase, David Koufaty, Dheeraj Reddy, and Scott Hahn. 2010. Operating system support for overlapping-ISA heterogeneous multi-core architectures. In *IEEE 16th International Symposium on High Performance Computer Architecture*. 1–12. DOI:http://dx.doi.org/10.1109/HPCA.2010.5416660

Hung-Yi Liu, Ilias Diakonikolas, Michele Petracca, and Luca Carloni. 2011. Supervised design space exploration by compositional approximation of Pareto sets. In *Proceedings of the 48th Design Automation Conference*. ACM, 399–404. DOI:http://dx.doi.org/10.1145/2024724.2024818

Andrew Lukefahr, Shruti Padmanabha, Reetuparna Das, Ronald Dreslinski Jr., Thomas F. Wenisch, and Scott Mahlke. 2014. Heterogeneous microarchitectures trump voltage scaling for low-power cores. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation Techniques*. ACM, 237–250. DOI:http://dx.doi.org/10.1145/2628071.2628078

Kun Luo, Jayanth Gummaraju, and Manoj Franklin. 2001. Balancing thoughput and fairness in SMT processors. In *IEEE International Symposium on Performance Analysis of Systems and Software*. 164–171. DOI:http://dx.doi.org/10.1109/ISPASS.2001.990695

Alain J. Martin, Mika Nyström, and Paul I. Pénzes. 2002. ET^2: A metric for time and energy efficiency of computation. In *Power Aware Computing*, Robert Graybill and Rami Melhem (Eds.). Springer US, 293–315. DOI:http://dx.doi.org/10.1007/978-1-4757-6217-4

Media Tek Inc. 2015. MediaTek launches the MediaTek Helio X20: The world's first mobile SoC featurint tri-cluster CPU Architecture. (12 May 2015). Retrieved 2015-05-18 from http://www.mediatek.com/en/news-events/mediatek-news/mediatek-launches-the-mediatek-helio-x20-the-worlds-first-mobile-soc-featuring-tri-cluster-cpu-architecture/

Pierre Michaud. 2013. Demystifying multicore throughput metrics. *Computer Architecture Letters* 12, 2 (July 2013), 63–66. DOI:http://dx.doi.org/10.1109/L-CA.2012.25

Sandeep Navada, Niket K. Choudhary, Salil V. Wadhavkar, and Eric Rotenberg. 2013. A unified view of non-monotonic core selection and application steering in heterogeneous chip multiprocessors. In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*. IEEE Press, 133–144. http://dl.acm.org/citation.cfm?id=2523721.2523743

NVIDIA Corp. n.d. Tegra 3 Multi-Core Processors. (n.d.). Retrieved Accessed 2014-02-17 from http://www.nvidia.com/object/tegra-3-processor.html

Nam Khanh Pham, Amit Kumar Singh, Akash Kumar, and Khin Mi Mi Aung. 2013. Incorporating energy and throughput awareness in design space exploration and run-time mapping for heterogeneous MPSoCs. In *Euromicro Conference on Digital System Design*. IEEE, 513–521. DOI:http://dx.doi.org/10.1109/DSD.2013.61

Jason A. Poovey, Markus Levy, Shay Gal-On, and Thomas M. Conte. 2009. A benchmark characterization of the EEMBC benchmark suite. *IEEE Micro* 29, 5 (Sep 2009), 18–29. DOI:http://dx.doi.org/10.1109/MM.2009.74

R Core Team. 2013. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. http://www.R-project.org/

Arun Raghavan, Laurel Emurian, Lei Shao, Marios Papaefthymiou, Kevin P. Pipe, Thomas F. Wenisch, and Milo M.K. Martin. 2013. Computational sprinting on a hardware/software testbed. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 155–166. DOI:http://dx.doi.org/10.1145/2451116.2451135

Samsung Electronics Co. Ltd. n.d. Samsung Exynos. (n.d.). Retrieved Accessed 2014-02-17 from http://www.samsung.com/global/business/semiconductor/minisite/Exynos/products5octa_5420.html

Serpil Sayın. 2000. Measuring the quality of discrete representations of efficient sets in multiple objective mathematical programming. *Mathematical Programming* 87, 3 (2000), 543–560. DOI:http://dx.doi.org/10.1007/s101070050011

Allan Snavely and Dean M. Tullsen. 2000. Symbiotic jobscheduling for a simultaneous multithreaded processor. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*. 234–244. DOI:http://dx.doi.org/10.1145/378993.379244

Erik Tomusk, Christophe Dubach, and Michael O'Boyle. 2014. Measuring flexibility in single-ISA heterogeneous processors. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation Techniques*. ACM, 495–496. DOI:http://dx.doi.org/10.1145/2628071.2628125

Erik Tomusk and Michael O'Boyle. 2013. Weak heterogeneity as a way of adapting multicores to real workloads. In *Proceedings of the 3rd International Workshop on Adaptive Self-Tuning Computing Systems*. ACM, Article 5, 3 pages. DOI:http://dx.doi.org/10.1145/2484904.2484909

Yatish Turakhia, Bharathwaj Raghunathan, Siddharth Garg, and Diana Marculescu. 2013. HaDeS: Architectural synthesis for heterogeneous dark silicon chip multi-processors. In *Proceedings of the 50th Annual Design Automation Conference*. Article 173. DOI:http://dx.doi.org/10.1145/2463209.2488948

Kenzo Van Craeynest and Lieven Eeckhout. 2013. Understanding fundamental design choices in single-ISA heterogeneous multicore architectures. *ACM Transactions on Architecture and Code Optimization* 9, 4, Article 32 (Jan 2013), 23 pages. DOI:http://dx.doi.org/10.1145/2400682.2400691

Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. 2010. Conservation cores: Reducing the energy of mature computations. In *Proceedings of the 15th Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 205–218. DOI:http://dx.doi.org/10.1145/1736020.1736044

Ganesh Venkatesh, Jack Sampson, Nathan Goulding-Hotta, Sravanthi Kota Venkata, Michael Bedford Taylor, and Steven Swanson. 2011. QsCores: Trading dark silicon for scalable energy efficiency with quasi-specific cores. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44 '11)*. ACM, 163–174. DOI:http://dx.doi.org/10.1145/2155620.2155640

Eckart Zitzler and Lothar Thiele. 1999. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation* 3, 4 (Nov 1999), 257–271. DOI:http://dx.doi.org/10.1109/4235.797969

Victor Zyuban and Peter M. Kogge. 2000. Optimization of high-performance superscalar architectures for energy efficiency. In *Proceedings of the International Symposium on Low Power Electronics and Design*. 84–89. DOI:http://dx.doi.org/10.1109/LPE.2000.155258