2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)

# Composite-ISA Cores: Enabling Multi-ISA Heterogeneity Using a Single ISA

Ashish Venkat
*University of Virginia*
*venkat@virginia.edu*

Harsha Basavaraj  Dean M. Tullsen
*University of California, San Diego*
{*hbasavar* | *tullsen@cs.ucsd.edu*}

***Abstract*—Heterogeneous multicore architectures are comprised of multiple cores of different sizes, organizations, and capabilities. These architectures maximize both performance and energy efficiency by allowing applications to adapt to phase changes by migrating execution to the most efficient core. Heterogeneous-ISA architectures further take advantage of the inherent ISA preferences of different application phases to provide additional performance and efficiency gains.**

**This work proposes *composite-ISA cores* that implement composite feature sets made available from a single large superset ISA. This architecture has the potential to recreate, and in many cases supersede, the gains of multi-ISA heterogeneity, by leveraging a single composite-ISA, exploiting greater flexibility in ISA choice. Composite-ISA CMPs enhance existing performance gains due to hardware heterogeneity by an average of 19%, and have the potential to achieve an additional 31% energy savings and 35% reduction in Energy Delay Product, with no loss in performance.**

***Keywords*-heterogeneous architectures; multicore processors; ISA; RISC; CISC; energy efficiency; single-thread performance**

## I. Introduction

Modern processors increasingly employ specialization to improve the execution efficiency of domain-specific workloads [1]–[6]. Additionally, some take advantage of microarchitectural heterogeneity by combining large high-performance cores and small power-efficient cores on the same chip, to create efficient designs that cater to the diverse execution characteristics of general-purpose mixed workloads [7]–[14]. Heterogeneous-ISA architectures [15]–[17] further expand the dimensions of hardware specialization by synergistically combining microarchitectural heterogeneity with ISA heterogeneity, exploiting the inherent ISA affinity of an application.

ISA heterogeneity provides the hardware architect and the compiler designer with finer control over features such as register pressure, predication support, and addressing mode availability, that could each significantly impact the overall throughput of the code [16], [18]–[20]. Multiple independent explorations have shown that ISA heterogeneity substantially improves execution efficiency in both chip multiprocessor (CMP) [15]–[22] and datacenter environments [23], [24], and these effects are particularly pronounced in scenarios where microarchitectural heterogeneity alone provides diminishing returns [16], [19].

However, in prior work, the ISA-dependent features had to be selected at a very coarse level. For example, the hardware designer might either choose x86 that comes with perhaps one key feature of interest but a lot of other overheads, or Thumb which forgoes several non-essential features but also a few others that are of significance. Such coarse selection greatly restricts both the ability to assign threads to the best core, but even more so it restricts the processor architect in identifying the best combination of hardware features to assign to a globally optimal set of heterogeneous cores.

Furthermore, despite their potential for greater performance and energy efficiency, the deployment of heterogeneous ISAs on a single chip is non-trivial. First, integration of multiple vendor-specific commercial ISAs on a single chip is fraught with significant licensing, legal, and verification costs and barriers. Second, process migration in a heterogeneous-ISA CMP necessitates the creation of fat binaries, and involves expensive binary translation and state transformation costs due to the difference in encoding schemes and application-binary interfaces (ABI) of fully disjoint ISAs.

This work proposes composite-ISA architectures that employ compact cores implementing fully custom ISAs derived from a single large superset ISA. If such an architecture can just match the performance and energy of the multi-ISA designs, this is a significant win due to the elimination of multi-vendor licensing, testing and verification issues, fat binaries, and high migration overheads. However, our results show that this design can, in fact, significantly outperform fully heterogeneous-ISA designs, due to greatly increased flexibility in creating cores that mix and match specific sets of features.

To derive fully custom ISAs with diverse capabilities, we first construct a well-defined baseline superset ISA that offers a wide range of customizable features: register depth (programmable registers supported), register width, addressing mode complexity, predication, and vector support. Ignoring some permutations of these features that are not viable or unlikely to be useful, this still gives the processor designer 26 custom ISAs that are potentially useful. Thus, by starting with a single superset ISA, the designer has far more choice and control than selecting from among existing commercial ISAs.

This work features a massive design space exploration that sweeps through all viable combinations of the customizable ISA features, along with an extensive set of customizable microarchitectural parameters, to identify optimal designs. A major contribution of this work is the isolation of specific ISA features that improve single thread performance and/or increase multi-programmed workload throughput/efficiency, and an extensive study of their effect on important architectural design choices that enable efficient transistor investment on the available silicon real estate.

In this work, we construct the superset ISA using extensions and mechanisms completely consistent and compatible with the existing Intel x86 ISA. However, we note that greater levels of customization can be achieved by creating a new (superset) ISA from scratch. We start with x86 because it not only already employs a large set of the features we want, but it has a clear history and process for adding extensions. In this work, we present detailed compiler techniques to extend the x86 backend to both support and exploit the extra features that we add and to customize existing features. We also describe migration strategies (with negligible binary translation costs) to switch between the composite ISAs that implement overlapping feature sets. In addition, this work features a comprehensive analysis of the hardware implications of the custom feature set options, including a full synthesized RTL design of multiple versions of the x86 decoder.

Due to the constraint of a single baseline superset ISA, the derived custom ISAs can never incorporate all traits of distinct vendor-specific ISAs (such as the code compression of Thumb). However, the greater flexibility and composability of our design results in greater overall efficiency. In designs optimized for multithreaded mixed workloads, we gain 18% in performance and reduce the energy-delay product (EDP) by 35% over single-ISA heterogeneous designs, without sacrificing most of the benefits of a single ISA. In designs optimized for single

2378-203X/19/$31.00 ©2019 IEEE
DOI 10.1109/HPCA.2019.00026

thread performance/efficiency, we achieve a speedup of 20% and an EDP reduction of 28% on average, over designs that only employ hardware heterogeneity. Further, we match and in fact supersede (by 15%) the gains of a heterogeneous-ISA CMP with vendor-specific ISAs, while effectively eliminating vendor-specific ISA licensing issues, fat binary requirements, binary translation, and state transformation costs.

## II. Related Work

Kumar, et al. [10]–[12] introduced single-ISA heterogeneous multicore architectures. These architectures employ cores of different sizes, organizations, and capabilities, allowing an application to dynamically identify and migrate to the most efficient core, thereby maximizing both performance and energy efficiency. Single-ISA heterogeneity has been well studied and applied broadly in embedded [4], [8], [14], general-purpose [10], [11], [25], GPU [3], [7], autonomous driving [26], and datacenter [27]–[29] environments. Architects have further explored several microarchitectural [30]–[38] and scheduling techniques [13], [39]–[55] to better harness the gains due to single-ISA heterogeneity.

Heterogeneous-ISA architectures [15]–[17], [20]–[22], [56] explore yet another dimension of heterogeneity. These architectures allow cores that are already microarchitecturally heterogeneous to further implement diverse instruction sets. By exploiting ISA affinity, where different code regions within an application inherently favor a particular ISA, they realize substantial performance and efficiency gains over hardware heterogeneity alone. Prior work shows that ISA affinity is beneficial not just in general-purpose environments, but could potentially enable significant energy efficiency in datacenter environments [23], [24], [57]. Wang, et al. [58] further enable offloading of binary code regions in a heterogeneous-ISA client/server environment. Furthermore, considerable prior work implements replicated OS kernel support for heterogeneous-ISA architectures [24], [59], [60].

Blem, et al. [61] claim that modern ISAs such as ARM and x86 have a similar impact in terms of performance and energy efficiency, but that work only compares similarly register pressure-constrained ISAs (ARM-32 and x86-32), turns off machine-specific tuning, ignores feature set differences (e.g., Thumb), and makes homogeneous hardware assumptions, unlike the work on heterogeneous-ISA architectures [16], [20], [21], [23], [24]. Akram and Sawalha [18], [19] perform extensive validation of the conflicting claims and conclude that the ISA does indeed have a significant impact on performance.

The RISC-V architecture provides extensive support in terms of both instruction set design and customizations for hardware accelerators [62], [63]. Although we choose our baseline ISA to be x86, the techniques we describe are not limited to x86. The RISC-V ISA allows enough flexibility to carve out similar axes of customization that we explore in this work, and thus would also be a reasonable host ISA for a composite-ISA architecture. On a fixed-length ISA like RISC-V, we expect to retain most of the benefits due to diversity in register depth/width, predication, and addressing-mode complexity. However, there may be additional effects due to the difference in code density–some of which manifested in the multi-vendor heterogeneous-ISA work that considered both fixed-length and variable-length ISAs [16].

There has been significant design space exploration [12], [16], [64]–[67] studies in related work.Intel's QuickIA [68], Fabscalar [69], [70], OpenPiton [71], [72], the heterogeneous-ISA JuxtaPiton [73], and Alladin [74], [75] further allow the exploration of heterogeneous architectures of varying complexity. We do not include the exploration of GPUs and other accelerator designs in our search space since we target diversity in execution characteristics and our primary goal is to achieve the gains of multi-vendor ISA-heterogeneity [16], [19], [20], [24] using a single ISA.

## III. ISA Feature Set Derivation

In this section, we describe our superset ISA. It resembles x86, but with an additional set of features that can be customized along 5 different dimensions: register depth, register width, opcode and addressing mode complexity, predication, and data-parallel execution. We further study the code generation impact, processor performance, power, and area implications of each dimension.

**Register Depth.** The number of programmable registers exposed by the ISA to the compiler/assembly programmer constitutes an ISA's register depth. The importance of register depth as an ISA feature is well established due to its close correlation to the actual register pressure (number of registers available for use) in any given code region [76]–[79]. While most compiler intermediate representations allow for a large number (potentially infinite) of virtual registers, the number of architectural registers is limited, resulting in spills and refills of temporary values into memory, and limiting the overall instruction-level parallelism [18], [19].

Register depth not only affects efficient code generation, but significantly impacts machine-independent code optimizations due to (register pressure sensitive) redundancy elimination and re-materialization (re-computation of a temporary value to avoid spills/refills) [80]–[84]. For example, decreasing the register depth from 32 to 16 registers in our custom feature sets results in a increase of 3.7% in stores (spills), 10.3% in loads (refills), 3.5% increase in integer instructions and 2.7% in branch instructions (rematerialization) on the SPEC CPU2006 benchmarks compiled using the methodology described in Section IV.

Furthermore, the backend area and power is strongly correlated to the ISA's register depth, impacting the nature and size of structures such as the reorder buffer and the physical register file – even with register renaming (i.e., dynamically scheduled cores) the physical register file still scales partially with ISA register depth. In our superset ISA, we allow register depth to be customized to 8, 16, 32, and 64 registers. A composite-ISA design that customizes each core with a different register depth alleviates the register pressure of impacted code regions by migration to a core with greater register depth, and at the same time saves power by enabling smaller microarchitectural structures in other cores.

**Register Width.** Like register depth, the register width of an architecture impacts performance and efficiency in several different ways. First, wider data types implies wider pointers allowing access to larger virtual memory and avoiding unnecessary memory mapping to files. However, wide pointers potentially expand the cache working set, thereby negatively impacting performance [85]. Second, wider registers can often be addressed as individual sub-registers enhancing the overall register depth of the ISA. Most compilers' register allocators take advantage of sub-registers (e.g., *eax*, *ax*, *al* etc) and perform

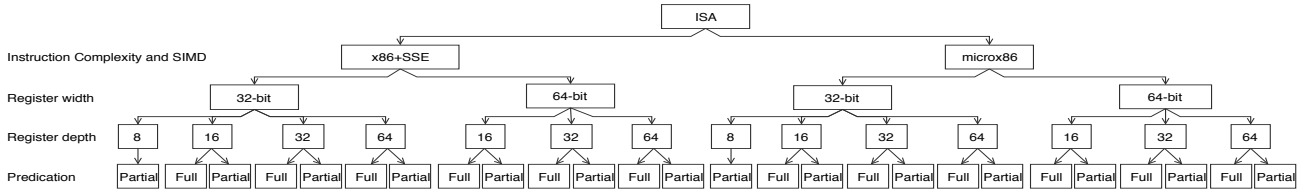


Figure 1. Derivation of Composite Feature Sets from a Superset ISA.

aggressive live range splitting and sub-register coalescing [81], [83], [86]–[89]. Third, emulating data types wider than the underlying ISA/core's register width not only requires more dynamic instructions, but could potentially use up more registers and thereby adversely impact register pressure.

Finally, wider registers imply larger physical register files in the pipeline which impacts both core die area and overall power consumption. In our experiments, doubling the register width from 32 bits to 64 bits impacts processor power by as much as 6.4% across different register depth organizations. Our superset ISA supports both 32-bit and 64-bit wide registers like x86, but we modify the instruction encoding to eliminate any restrictions on the addressing of a particular register, sub-register, or combination thereof.

**Instruction Complexity.** The variety of opcodes and addressing modes offered by an instruction set controls the mix of dynamic instructions that enter and flow through the pipeline. Incorporating a reduced set of opcodes and addressing modes into the instruction set could significantly simplify the instruction decode engine, if chosen carefully. In fact, by excluding instructions that translate to more than one micro-op, we can save as much as 9.8% in peak power and 15.1% in area. However, for some code regions, such a scheme could increase the overall code size, potentially impacting both the overall instruction cache accesses and instruction fetch energy.

To derive such a reduced feature set, we carve out a subset of opcodes and addressing modes from our superset ISA that can be implemented using a single micro-op, essentially creating custom cores that implement the x86 micro-op ISA, albeit with variable-length encoding. The reduced feature set, called *microx86* in this work, adheres to the *load-compute-store* philosophy followed by most RISC architectures. As a result, we could view this option as RISC vs CISC support. While one could conceive a more aggressive low-power implementation of *microx86* that implements fewer opcodes and further recycles opcodes for a more compact representation [90]–[94], we keep all the same opcodes, and thus follow x86's existing variable-length encoding and 2-phase decoding scheme. This not only maintains consistency with existing implementations of x86, but prevents us from incurring the binary translation costs associated with multi-vendor heterogeneous-ISA designs. However, this does mean we cannot completely replicate the instruction memory footprint of a theoretically minimal representation.

**Predication.** Predication converts control dependences into data dependences in order to eliminate branches from the instruction stream and consequently take pressure off the branch predictor and associated structures [95]–[99], while also removing constraints on the compiler's instruction scheduler. Modern ISA implementations of predication can be classified into three categories: (a) partial predication that allows only a subset of the ISA's instructions to be predicated, (b) full predication that allows any instruction to be predicated using a predefined set of

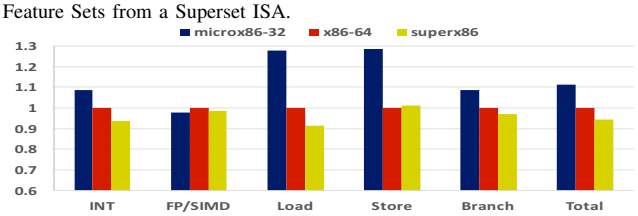


Figure 2. SPEC2006 Instr. Mix (normalized to x86-64)

predicated registers, and (c) conditional execution that allows any instruction to be predicated using one condition code register.

The x86 ISA already implements partial predication via CMOVxx instructions that are predicated on condition codes. In this work, we add full predication support to our superset ISA, allowing any instruction to be predicated using any available general-purpose register using the if-conversion strategy described in Section IV. While predication eliminates branch dependences, aggressive if-conversion typically increases the number of dynamic instructions, thus placing more pressure on the instruction fetch unit and the instruction queue. In our custom feature sets that offer predication, we observe an average increase of 0.6% in the number of dynamic instructions with a reduction of 6.5% in branches.

**Data-Parallel Execution.** Most modern instruction sets offer primitives to perform SIMD operations [85], [100]–[102] to take advantage of the inherent data-level parallelism in specific code regions. The x86 ISA already supports multiple feature sets that implement a variety of SIMD operations. We include the SSE2 feature set in our superset ISA that can compute on data types that are as wide as 128 bits as implemented in the gem5 simulator [103]. Furthermore, we constrain our microx86 implementations to not include SSE2 since more than 50% of SIMD operations rely on 1:n encoding of macro-op to micro-op. In our composite-ISA design, cores that do not implement SSE2 save 7.4% in peak power and 17.3% in area. They execute a precompiled scalarized version of the code when available, and in most cases, migrate code regions that enter intense vector activity to cores with full vector support.

In summary, we derive 26 different custom feature sets along the five dimensions described above.We exclude full predication from our 32-bit feature sets that have only 8 registers since they suffer from high register pressure. In fact, LLVM's predication profitability analysis seldom turns on predication with 8 registers. Similarly, we constrain 64-bit ISAs to support a register depth of at least 16.

Figure 2 shows the dynamic instruction (micro-op) breakdown for the SPEC CPU2006 benchmarks on three different custom ISAs: (a) the 32-bit version of microx86 with a register depth of 8 and no additional features (smallest feature set in our exploration), (b) the x86-64 ISA with SSE and no other customizations, and (c) the superset ISA which implements all the features described above. Due to the high register pressure in microx86-32, it incurs an average of 28% higher memory references than x86-64 and an overall expansion of 11% in the number of micro-ops.

Also compared to the x86-64, we find that the superset ISA, owing to the diverse set of custom features added, sees an average reduction of 8.5% in loads (spill elimination), 6.3% in integer instructions (aggressive redundancy elimination), and 3.2% in branches (predication).

## IV. Compiler and Runtime Strategy

In this section, we describe our compilation strategy that generates code to efficiently take advantage of the underlying custom feature sets, and our runtime migration strategy that allows code regions to seamlessly migrate back and forth between different custom feature sets, without the overhead of full binary translation and/or state transformation.

### A. Compiler Toolchain Development

Compilation to a superset ISA or a combination of custom feature sets allows different code regions to take advantage of the variety of custom feature sets implemented by the underlying hardware. For example, code regions with high register pressure could be compiled to execute on a feature set with greater register depth, and code regions with too many branches could be compiled to execute predicated code. We leverage the LLVM MC infrastructure [89] to efficiently encode the right set of features supported by the underlying custom design and further propagate it through various instruction selection, code generation, and machine-dependent optimization passes. We further take advantage of the MC code emitter framework to encode feature sets such as register depth and predication that require an extra prefix.

To convert the existing x86 backend to that of the superset ISA, we first include the additional 48 registers in the ISA's target description and further associate code density costs with it. This enables the register allocator to always prioritize the allocation of a register that requires fewer prefix bits to encode it. Furthermore, we allow each of these registers to be addressed as a byte, a word, a doubleword, and a quadword register, thereby smoothly blending into the register pressure tracking and subregister coalescing framework.

We next implement full predication in x86 by re-purposing the existing machine-dependent *if-conversion* framework of LLVM that implements if-conversions in three scenarios: (a) *diamond* – when a true basic block and a false basic block split from an entry block and rejoin at the tail, (b) *triangle* – when the true block falls through into the false block, and (c) *simple* – when the basic blocks split but do not rejoin, such as a break statement within a conditional. For every such pattern in the control flow graph, if-conversion is performed if determined profitable. The profitability of if-conversion is based on branch probability analysis, the approximate latency of machine instructions that occur in each path, and the configured pipeline depth. We further add the if-conversion pass as a pre-scheduling pass for the X86 target – this allows the scheduling pass to take full advantage of the large blocks of unscheduled code created by the if-conversion.

To implement *microx86*, we modify the existing x86 backend to exclude all opcodes, addressing modes, and prefixes that decode into more than one micro-op during machine instruction selection. While LLVM's instruction selector, for the most part, identifies a replacement *ld-compute-st* combination for the excluded addressing mode, certain IR instructions such as tail jumps and tail call returns require explicit instruction lowering.

For each code region (where we roughly define a region as the set of code that dominates a simpoint phase), the compiler must now make a global (or regional) decision about which features to use and which to skip. Further, the compiler should make these decisions with some knowledge of the features of the cores for the processor on which it will run. While we find that LLVM already makes profitable decisions for several features we explore (e.g., register depth and predication), near-optimal instruction selection (microx86 vs x86) would still need a well-defined heuristics approach. Because we examine so many core combinations in our design space exploration, we assume the compiler makes good instruction selection decisions about which features to include in compilation in all cases. So despite the fact that features included in compilation are not static for a region (across experiments), we can still see clear trends in code affinity for features.

We find that the highly register-constrained benchmark *hmmer* is consistently compiled to use all 64 registers across all code regions. In contrast, only one phase of the benchmark *bzip2* is compiled with a register depth of 64, with the rest of the seven regions typically compiled with a register depth of 32. The benchmark *lbm* exhibits low register pressure and prefers a register depth of 16. While most code regions prefer microx86, when constrained by fewer than 32 registers, x86 is preferred since the complex addressing modes somewhat alleviate the register pressure, as evidenced in the benchmarks *sjeng* and *mcf*. Similarly, the compiler employs predication in four regions of the benchmark *milc*, but does not find it to be profitable in two other regions of the same benchmark. Furthermore, due to the high register pressure and lack of free registers on *hmmer*, the compiler harnesses the full suite of complex addressing modes offered by x86 and seldom employs predication. These decisions propagate through the rest of the optimization passes resulting in carefully optimized code for the underlying feature set.

### B. Migration Strategy

Process migration across overlapping custom feature sets could involve two scenarios. In an upgrade scenario, a process is compiled to use only a subset of the features implemented by the core to which it is migrated, and therefore can resume native execution immediately after migration (no binary translation or state transformation). Conversely, in a downgrade scenario, the core to which a process is migrated implements only a subset of the features being used by the running code, which necessitates minimal binary translation of unimplemented features. We outline the following low-overhead mechanisms to handle feature downgrades.

Owing to the overlapping nature of the feature sets (same opcode and instruction format), feature emulation entails only a small set of binary code transformations, in contrast to full blown cross-ISA binary translation. First, when we downgrade from *x86* to *microx86*, we perform simple addressing mode transformations by translating any instruction that directly operates on memory into a set of simpler instructions that adhere to the *ld-compute-st* format. Second, we downgrade to a feature set with a smaller register depth by translating higher (unimplemented) registers into memory operands using a register context block [15], [104], [105], a commonly used technique during binary translation to register pressure-constrained ISAs. Third, we perform long-mode emulation [16], [105] and use *fat pointers* implemented using *xmm* registers in order to emulate wider types on a 32-bit core. Finally, we employ simple reverse if-conversions to translate

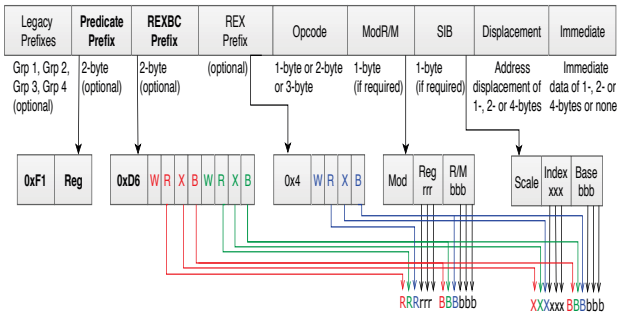


Figure 3. Customizations to x86 Encoding

predicated code back to control-dependences. As demonstrated in Section VII, feature downgrades are extremely rare and often inexpensive.

## V. Decoder Design

In this section, we describe our customizations to the x86 instruction encoding and decoder implementation to support the 26 feature sets derived in the Section III. We show that, due to the extensible nature of the x86 ISA, the decoder implementation requires minimal changes to support the new features and has a small overall peak power and area impact.

### A. Instruction Encoding

Feature extensions to the x86 instruction set are not uncommon. In accordance with its code density and backward compatibility goals, major feature set additions to x86 (e.g., REX, VEX, and EVEX) have been encoded by exploiting unused opcodes and/or by the addition of new (optional) prefix bytes. We use similar mechanisms to encode the specific customizations we propose as shown in Figure 3.

To double/quadruple the register depth of x86-64, we add a new prefix – REXBC, similar to the addition of the REX (register extension) prefix that doubled both the register width and depth of x86-32, giving rise to the x86-64 ISA. In particular, the REXBC prefix encodes 2 extra bits for each of the 3 register operands (input/output register, base, and index), which is further combined with 4 bits from the REX, MODRM, and SIB bytes, to address any of the 64 programmable registers. Furthermore, we use the remaining 2 bits of the REXBC prefix to lift restrictions in x86 that do not allow certain combinations of registers and subregisters to be used as operands in the same instruction. Finally, we exploit an unused opcode **0xd6** to mark the beginning of a REXBC prefix. Similarly, to support predication, we use a combination of an unused opcode **0xf1** and a predicate prefix. To efficiently support diamond predication (described in Section IV), we use the predicate prefix to encode both the nature (true/not-true) of the conditional (bit 7) and the register (bits 0-6) the instruction is predicated on.

All of the insights in this paper apply equally (if not more so) to a new superset ISA designed from scratch – such an ISA would allow much tighter encoding of these options.

### B. Decoder Analysis

Figure 4 shows the step-by-step decoding process of a typical x86 instruction. Owing to the variable length encoding, x86 instructions go through a 2-phase decode process. In the first phase, an instruction-length decoder (ILD) fetches and decodes raw bytes from a prefetch buffer, performs length validation, and further queues the decoded macro-ops into a macro-op buffer. These macro-ops are fused into a single macro-op when viable,

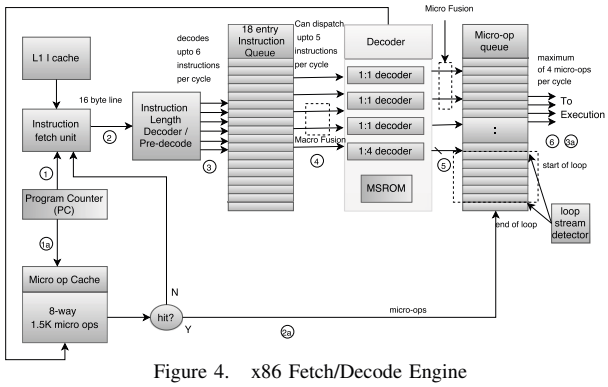


Figure 4. x86 Fetch/Decode Engine

and further fed as input into one of the instruction decoders that decode it into one or more micro-ops. The decoded micro-ops are subjected to fusion again, to store them in the micro-op queue and the micro-op cache [106]–[108] in a compact representation, and are later unfused and dispatched as multiple micro-ops. The micro-op cache is both a performance and power optimization that allows the engine to stream decoded (and potentially fused) micro-ops directly from the micro-op cache, turning off the rest of the decode pipeline until there is a micro-op miss.

In our RTL implementation, we model an ILD based on the parallel instruction length decoder described by Madduri, et al [109]. The ILD has 3 components: (a) an instruction decoder that decodes each byte of an incoming 8-byte chunk as the start of an instruction, decoding prefixes and opcodes, (b) a length calculator that speculatively computes the length of the instruction based on the decoded prefixes and opcodes, and (c) an instruction marker that checks the validity of the computed lengths, marks the begin and end of an instruction, and detects overflows into the next chunk.

Since our customizations affect the prefix part of the instruction, we modify the eight decode subunits of the instruction decoder to include comparators that generate extra decode signals to represent the custom register depth and predicate prefixes. These decode signals propagate through the speculative instruction length calculator and the instruction marker requiring wider multiplexers in the eight length subunits, the length control select unit, and the valid begin unit. These modifications to the instruction length decoder result in an increase of 0.87% in total peak power and 0.65% in area for our superset ISA.

Furthermore, we increase the width of the macro-op queue by 2 bytes to account for the extra prefixes. Since predication support and greater register depth in our superset ISA could potentially require wider micro-op ISA encoding, we increase the width of the micro-op cache and the micro-op queue by 2 bytes. Finally, for our *microx86* implementations, we replace the complex 1:4 decoder with another simple 1:1 decoder and forgo the microsequencing ROM. From our analysis, a decoder that implements our simplest feature set *microx86-32* consumes 0.66% less peak power and takes up 1.12% lesser area than the x86-64 decoder, and our superset decoder consumes 0.3% more peak power and takes up 0.46% more area than the x86-64 decoder. These variances do not include the increases or savings from the ILD.

## VI. Methodology

Table I shows our design space that consists of 5 dimensions of ISA customizations and 19 micro-architectural dimensions.

| ISA Parameter | Options |
|---|---|
| Register depth | 8, 16, 32, 64 registers |
| Register width | 32-bit, 64-bit registers |
| Instruction/Addressing mode complexity | 1:1 macroop-microop encoding (load-store x86 micro-op ISA), 1:n macroop-microop encoding (fully CISC x86 ISA) |
| Predication Support | Full Predication like IA-64/Hexagon vs Partial (cmov) Predication |
| Data Parallelism | Scalar vs Vector (SIMD) execution |
| **Microarchitectural Parameter** | **Options** |
| Execution Semantics | Inorder vs Out-Of-Order designs |
| Fetch/Issue Width | 1, 2, 4 |
| Decoder Configurations | 1-3 1:1 decoders, 1 1:4 decoder, MSROM |
| Micro-op Optimizations | Micro-op Cache, Micro-op Fusion |
| Instruction Queue Sizes | 32, 64 |
| Reorder Buffer Sizes | 64, 128 |
| Physical Register File Configurations | (96 INT, 64 FP/SIMD), (64 INT, 96 FP/SIMD) |
| Branch Predictors | 2-level local, gshare, tournament |
| Integer ALUs | 1, 3, 6 |
| FP/SIMD ALUs | 1, 2, 4 |
| Load/Store Queue Sizes | 16, 32 |
| Instruction Cache | 32KB 4-way, 64KB 4-way |
| Private Data Cache | 32KB 4-way, 64KB 4-way |
| Shared Last Level (L2) Cache | 4-banked 4MB 4-way, 4-banked 8MB 8-way |

Table I
FEATURE EXPLORATION SPACE

After careful pruning of configurations that are not viable (e.g., 4-issue cores with a single INT/FP ALU) or unlikely to be useful (full predication with 8 registers), this results in 26 different custom ISA feature sets, 180 microarchitectural configurations, and 4680 distinct single core design points, that each are spread across a wide range of per-core peak power (4.8W to 23.4W) and area (9.4mm$^2$ to 28.6mm$^2$) distributions. The goal of our feature set exploration is to find an optimal 4-core multicore configuration using the fully custom feature sets derived out of the superset ISA. Our objective functions that evaluate optimality include both performance and energy delay product (EDP), for both multithreaded and single-threaded workloads. Our workloads include 8 SPEC CPU2006 benchmarks further broken down into 49 different application phases using the SimPoint [110], [111] methodology.

We use the gem5 [103] simulator to measure performance in both our inorder and out-of-order cores. We modify the gem5 simulator to include micro-op cache and micro-op fusion support in order to measure the impact of our customizations in light of existing micro-op optimizations. However, we do not employ micro-op fusion in our *microx86* ISA because each instruction only decomposes into one micro-op and the micro-op fusion unit doesn't yet combine micro-ops from different macro-ops. Our implementations of the micro-op cache and fusion are consistent with guidelines mentioned in the Intel Architecture Optimization Manual [112].

We perform a full RTL synthesis using the Synopsys Design Compiler to measure the decoder area and power overheads of each of the customizations we employ in our feature sets, as described in Section V. We also use the McPAT [113] power modeling framework to evaluate the power and area of the rest of the pipeline. The peak power and area measurements we obtain out of McPAT simulations are key parameters to our exploration of this massive design space, which involves 196,560 gem5+McPAT simulations (approx. 28,080 simulations per benchmark) resulting in 49,733 core hours on the 2 petaflop XSEDE Comet cluster at the San Diego Supercomputing Center [114]. Furthermore, our multicore design search involves finding an optimal 4-core multicore out of a 102.5 trillion combinations that run all permutations of simpoint regions, for several different objective functions and constraints. To again make the search tractable, the results we report for the fully custom feature set design are local optima, and thus conservative estimates.

| microx86-8D-32W | microx86-32D-64W | x86-16D-64W |
|---|---|---|
| **Thumb-like Features** | **Alpha-like Features** | **x86-64-like Features** |
| Load/Store Architecture | Load/Store Architecture | CISC Architecture |
| Register Depth: 8 | Register Depth: 32 | Register Depth: 64 |
| Register Width: 32 | Register Width: 64 | Register Width: 64 |
| No SIMD support | No SIMD support | SIMD support |
| | CMOV Support | CMOV Support |
| **Exclusive Features:** | **Exclusive Features:** | **Exclusive Features:** |
| FP Support | None | None |
| **Thumb-specific Features:** | **Alpha-specific Features:** | **x86-specific Features:** |
| Code Compression | 2-address instructions | None |
| Fixed-length instructions (one-step decoding) | Fixed-length instructions (one-step decoding) | |
| | More FP Registers | |

Table II
X86-IZED VERSIONS OF THUMB, ALPHA, AND X86-64

Finally, process migration in a composite-ISA architecture could potentially involve binary translation of unimplemented features in case of a feature downgrade. We measure this cost by performing feature emulation for each checkpointed code region that represents a simpoint, on artificially-constrained cores that only implement a subset of the features the simpoint was compiled to. We also report the overall cost of migration on our designs optimized for multiprogrammed workload throughput where threads often contend for the cores of their preference, and therefore may not always run on the core of their choice.

## VII. RESULTS

### A. Performance and Energy Efficiency

This section elaborates on the findings of our design space exploration. We identify custom multicore designs that benefit from both hardware heterogeneity and feature set diversity, providing significant gains over designs that exploit only hardware heterogeneity. Furthermore, these designs recreate the effects or in most cases, surpass the gains offered by a fully heterogeneous-ISA CMP that implements a completely disjoint set of vendor-specific ISAs and requires sophisticated OS/runtime support. We conduct multiple searches through our design space to model different execution scenarios and budget constraints.

In each search, we identify three optimal 4-core multicore designs: (1) homogeneous x86-64 CMPs that employ cores that implement the same ISA and the same microarchitecture, (2) x86-64 CMPs that exploit hardware heterogeneity alone, and (3) composite-ISA x86-64 CMPs that exploit hardware heterogeneity and full ISA feature diversity.

In addition, we also identify two intermediate design points of interest: (1) heterogeneous-ISA CMPs [16] that implement three fixed, disjoint, vendor-specific ISAs (x86-64, Alpha, Thumb), and (2) composite-ISA CMPs that exploit hardware heterogeneity and a limited form of feature diversity via three x86-based fixed feature sets that resemble the above vendor-specific ISAs. We use the latter design as a vehicle to demonstrate that vendor-specific ISA heterogeneity can be recreated to a large extent by carving out custom feature sets from a single sufficiently-diverse superset ISA. Table II offers a more detailed comparison.

Thus, we compare against a number of interesting configurations, but two are most revealing. Since we seek to replicate the advantage of multi-ISA heterogeneity over single-ISA heterogeneity, the single-ISA heterogeneous result is our primary baseline for comparison. However, the multi-vendor (x86, Thumb, Alpha) result represents our "goal" that we strive to match, yet

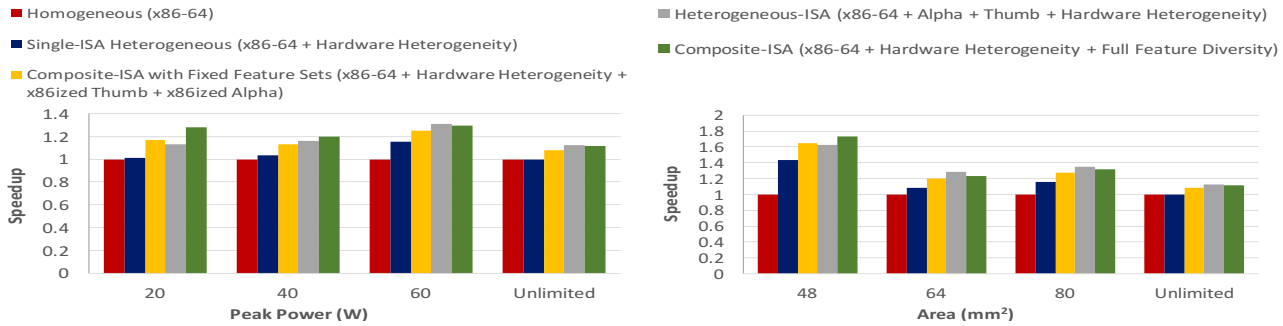


Figure 5. Multi-programmed workload throughput comparison (higher is better)

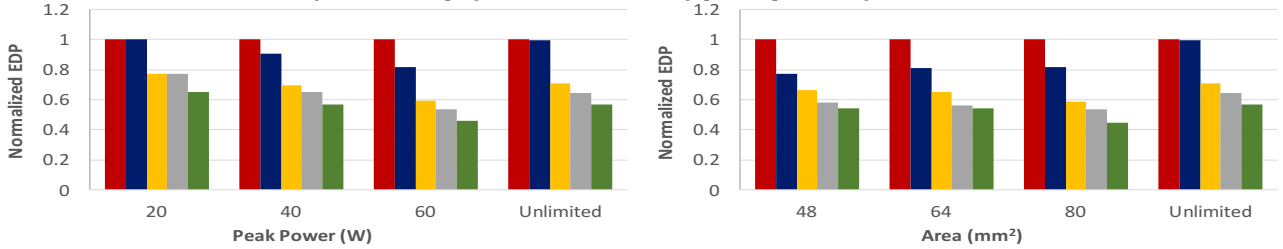


Figure 6. Multi-programmed workload EDP comparison (lower is better)

| Core Complexity | Reg Width | Reg Depth | Predication | IO/OOO | Issue Width | Branch Pred | INT Reg | FP Reg | ROB | IQ | INT ALU | INT MUL | FP/SIMD | LSQ | L1 Sz/Assoc | L2 Sz/Assoc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Peak Power Budget: 20W | | | | | | | | | | | | | | | | |
| 0 μx86 | 32 | 8 | P | I | 2 | L | 64 | 16 | 64 | 32 | 3 | 1 | 1 | 16 | 32kB/4 | 1MB/4 |
| 1 μx86 | 64 | 16 | P | I | 2 | T | 64 | 16 | 64 | 32 | 3 | 1 | 1 | 16 | 32kB/4 | 1MB/4 |
| 2 μx86 | 32 | 64 | F | I | 2 | T | 64 | 16 | 64 | 32 | 3 | 1 | 1 | 16 | 32kB/4 | 1MB/4 |
| 3 x86 | 32 | 64 | F | I | 1 | L | 64 | 16 | 64 | 32 | 1 | 1 | 1 | 16 | 32kB/4 | 1MB/4 |
| Peak Power Budget: 40W | | | | | | | | | | | | | | | | |
| 0 μx86 | 32 | 16 | P | O | 2 | T | 192 | 160 | 128 | 64 | 3 | 1 | 1 | 16 | 32kB/4 | 1MB/4 |
| 1 μx86 | 32 | 16 | P | O | 4 | T | 192 | 160 | 128 | 64 | 3 | 1 | 1 | 32 | 64kB/4 | 2MB/8 |
| 2 μx86 | 64 | 32 | P | O | 2 | T | 192 | 160 | 128 | 64 | 3 | 1 | 1 | 16 | 32kB/4 | 1MB/4 |
| 3 x86 | 32 | 64 | F | I | 2 | T | 64 | 16 | 64 | 32 | 3 | 1 | 1 | 16 | 64kB/4 | 1MB/4 |
| Peak Power Budget: 60W | | | | | | | | | | | | | | | | |
| 0 μx86 | 32 | 16 | P | O | 4 | T | 192 | 160 | 128 | 64 | 6 | 2 | 1 | 32 | 64kB/4 | 2MB/8 |
| 1 μx86 | 32 | 64 | P | O | 2 | T | 192 | 160 | 128 | 64 | 3 | 1 | 1 | 16 | 64kB/4 | 2MB/8 |
| 2 μx86 | 64 | 64 | P | O | 2 | T | 192 | 160 | 128 | 64 | 3 | 1 | 1 | 16 | 64kB/4 | 2MB/8 |
| 3 μx86 | 64 | 64 | F | O | 4 | T | 192 | 160 | 128 | 64 | 6 | 2 | 1 | 32 | 64kB/4 | 2MB/8 |
| Unlimited Peak Power and Area Budget. | | | | | | | | | | | | | | | | |
| 0 μx86 | 32 | 16 | P | O | 4 | T | 192 | 160 | 128 | 64 | 6 | 2 | 1 | 32 | 64kB/4 | 2MB/8 |
| 1 μx86 | 64 | 16 | P | O | 4 | T | 192 | 160 | 128 | 64 | 6 | 2 | 1 | 32 | 64kB/4 | 2MB/8 |
| 2 μx86 | 64 | 64 | F | O | 4 | T | 192 | 160 | 128 | 64 | 6 | 2 | 1 | 32 | 64kB/4 | 2MB/8 |
| 3 x86 | 32 | 64 | F | O | 4 | T | 192 | 160 | 128 | 64 | 6 | 2 | 1 | 32 | 64kB/4 | 2MB/8 |

Table III
COMPOSITE-ISA MULTICORE ARCHITECTURAL COMPOSITION (OPTIMIZED FOR MULTI-PROGRAMMED THROUGHPUT)

| Core Complexity | Reg Width | Reg Depth | Predication | IO/OOO | Issue Width | Branch Pred | INT Reg | FP Reg | ROB | IQ | INT ALU | INT MUL | FP/SIMD | LSQ | L1 Sz/Assoc | L2 Sz/Assoc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Peak Power Budget: 20W | | | | | | | | | | | | | | | | |
| 0 μx86 | 32 | 32 | P | I | 1 | T | 64 | 16 | 64 | 32 | 1 | 1 | 1 | 16 | 32kB/4 | 1MB/4 |
| 1 x86 | 32 | 8 | P | I | 1 | T | 64 | 16 | 64 | 32 | 1 | 1 | 1 | 16 | 32kB/4 | 2MB/8 |
| 2 x86 | 64 | 16 | P | I | 1 | L | 64 | 16 | 64 | 32 | 1 | 1 | 1 | 16 | 32kB/4 | 1MB/4 |
| 3 x86 | 32 | 64 | F | I | 1 | L | 64 | 16 | 64 | 32 | 1 | 1 | 1 | 16 | 32kB/4 | 1MB/4 |
| Peak Power Budget: 40W | | | | | | | | | | | | | | | | |
| 0 μx86 | 64 | 16 | P | O | 2 | T | 192 | 160 | 128 | 64 | 3 | 1 | 1 | 16 | 32kB/4 | 2MB/8 |
| 1 μx86 | 32 | 64 | P | O | 1 | G | 192 | 160 | 128 | 64 | 1 | 1 | 1 | 16 | 64kB/4 | 2MB/8 |
| 2 x86 | 32 | 16 | P | I | 2 | T | 64 | 16 | 64 | 32 | 3 | 1 | 1 | 16 | 32kB/4 | 2MB/8 |
| 3 x86 | 32 | 64 | F | I | 2 | L | 64 | 16 | 64 | 32 | 3 | 1 | 1 | 16 | 32kB/4 | 2MB/8 |
| Peak Power Budget: 60W | | | | | | | | | | | | | | | | |
| 0 μx86 | 32 | 32 | P | O | 2 | L | 192 | 160 | 128 | 64 | 3 | 1 | 1 | 16 | 32kB/4 | 1MB/4 |
| 1 μx86 | 32 | 32 | F | O | 2 | G | 192 | 160 | 128 | 64 | 3 | 1 | 1 | 16 | 32kB/4 | 2MB/8 |
| 2 x86 | 64 | 16 | P | I | 2 | T | 64 | 16 | 64 | 32 | 3 | 1 | 1 | 16 | 32kB/4 | 2MB/8 |
| 3 x86 | 32 | 64 | F | I | 2 | L | 64 | 16 | 64 | 32 | 3 | 1 | 1 | 16 | 32kB/4 | 2MB/8 |
| Unlimited Peak Power and Area Budget. | | | | | | | | | | | | | | | | |
| 0 μx86 | 32 | 16 | P | O | 4 | T | 192 | 160 | 128 | 64 | 6 | 2 | 1 | 32 | 64kB/4 | 2MB/8 |
| 1 μx86 | 32 | 32 | P | O | 2 | T | 192 | 160 | 128 | 64 | 3 | 1 | 1 | 16 | 64kB/4 | 2MB/8 |
| 2 μx86 | 64 | 32 | P | O | 2 | T | 192 | 160 | 128 | 64 | 3 | 1 | 1 | 16 | 32kB/4 | 2MB/8 |
| 3 x86 | 32 | 64 | F | I | 4 | L | 64 | 16 | 64 | 32 | 3 | 1 | 1 | 32 | 64kB/4 | 2MB/8 |

Table IV
COMPOSITE-ISA MULTICORE ARCHITECTURAL COMPOSITION (OPTIMIZED FOR MULTI-PROGRAMMED EFFICIENCY)

with essentially a single ISA and far fewer costs.

Figure 5 compares the performance and energy efficiency of the five optimal designs listed above, optimized to provide multi-programmed workload throughput when constrained under different peak power and area budgets. There are four major takeaways from this experiment. First, the designs that exploit feature diversity alongside hardware heterogeneity consistently and significantly outperform the designs that exploit only hardware heterogeneity. This is because hardware heterogeneity tends to diminish when the amount of available chip real estate becomes too small or too generous. Second, we find in the resulting optimal architectures that in an especially tightly power/area constrained environment, every feature present in the superset ISA is implemented by at least one core in the composite-ISA design. When the constraints become more relaxed, the composite-ISA designs continue to implement at least 10 out of the 12 features described in Section III. Third, the composite-ISA designs that implement the x86-ized versions of the vendor-specific ISAs trail slightly but generally match the gains of the fully heterogeneous-ISA designs. Fourth, the composite-ISA design with full ISA feature set diversity not only matches but frequently outperforms the fully heterogeneous-ISA design. This indicates that any loss due to the lack of specific ISA encoding and simplified hardware (e.g. decoders) is more than compensated for by the increased flexibility of the composable ISA features.

Overall, composite-ISA designs outperform single-ISA heterogeneous designs by 17.6% on average, and by 30% under tight power constraints. A significant chunk of these benefits arise from greater feature diversity. In fact, composite-ISA configurations with full feature diversity but uniform microarchitecture improve performance by an average of 9.3%

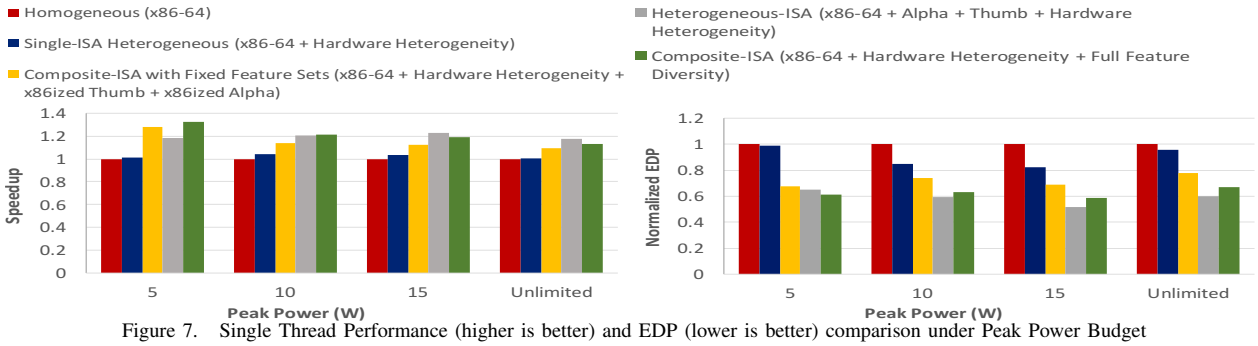


Figure 7. Single Thread Performance (higher is better) and EDP (lower is better) comparison under Peak Power Budget

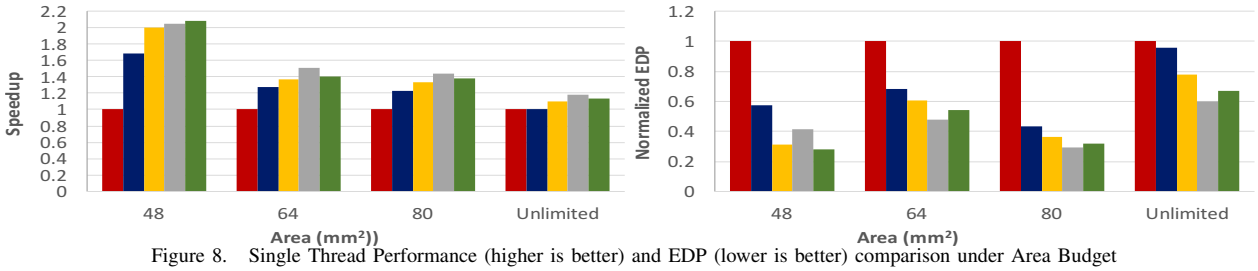


Figure 8. Single Thread Performance (higher is better) and EDP (lower is better) comparison under Area Budget

for multiprogrammed and 12.8% for single-threaded workloads, against a uniform-ISA (x86-64) and uniform microarchitecture baseline.

Table III gives the derived optimal designs for some of the experiments from Figure 5. As expected, we see the hardware heterogeneity of previous work, including variation in dynamic scheduling, queue sizes, ROB size, ALU count, and branch predictor (local (L), gshare (G), or tournament (T). However, we also see significant (perhaps even greater) use of the ISA design space, with variation in complexity, register width, register depth, and predication, not only as constraints vary, but within a single multicore design. For example, with a 20W power budget, we have designs with both the simple (microx86) and complex decoders, two register widths, 3 register depths, and both full and partial predication. It is interesting to note that for that tight power budget, the x86 design has a large number of registers but is otherwise conservative (scalar issue, narrow registers), while the microx86 cores use the saved power budget for more aggressive features like wider issue and tournament predictors. Overall, we see no clear dominant choice for any of our ISA features, but rather each optimal design taking good advantage of the choice of ISA parameters.

In Figure 6, we compare designs optimized to provide multi-threaded workload energy efficiency, measured as Energy Delay Product (EDP), while being constrained under different peak power and area budgets. We observe significant energy savings due to full ISA customization – an average of 31% savings in energy and 34.6% reduction in EDP over single-ISA heterogeneous multicore designs. This result was not necessarily expected, as the Thumb architecture still provides significant advantages over our most conservative microx86 core. However, many codes cannot use Thumb because of its limited features, while the composite-ISA architecture can combine microx86 with a variety of other features and make use of it far more often.

Table IV gives the derived optimal designs for experiments from Figure 6. We again see the designs taking good advantage of both hardware heterogeneity and ISA heterogeneity. While the choice of decoder complexity is clearly still a boon in this

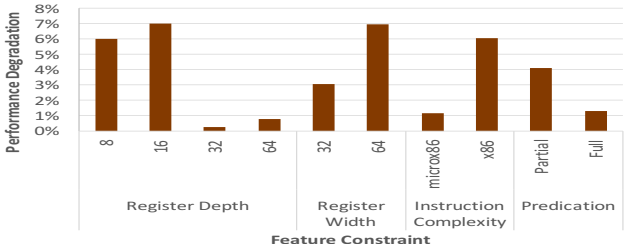


Figure 9. Performance Degradation over Composite-ISA Designs optimized for multi-programmed workload throughput at 48mm$^2$ budget, and under different Feature Constraints

design space, we see more designs that use the full decoder. So while the microx86 ISA coupled with superscalar is a good tradeoff for performance-optimized designs, for example, the EDP-optimized designs favored more x86 scalar designs. In nearly all designs, however, having the choice was the key to reaching the optimal design point.

We next evaluate our designs optimized to provide high single thread performance and energy efficiency. That is, while the prior results optimized for four threads running on four cores, this exploration optimizes for one thread utilizing four cores via migration. When the designs are constrained by peak power budgets, we model them after the dynamic multicore topology [115] where only one core is active at any given point of time, while the rest of the cores are powered off.

Figure 7 compares designs optimized to provide high single thread performance and energy efficiency. Note that the peak power budgets are tighter in this case since we assume only one core to be powered on at a time. Due to the low power constraints, hardware heterogeneity provides only marginal improvements in performance and EDP. However, we observe that every feature of the superset ISA again manifests in at least one of the feature sets implemented by the composite-ISA design, allowing applications to migrate across different cores in order to take advantage of any required ISA feature. We observe an average speedup of 19.5% and an average EDP reduction of 27.8% over single-ISA heterogeneous designs. Moreover, owing to the many low-power and feature-rich *microx86* options

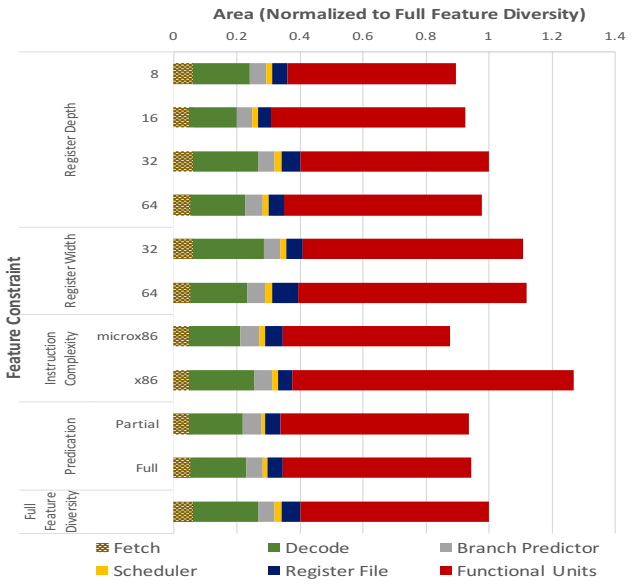


Figure 10. Transistor Investment by Processor Area normalized over that of Composite-ISA Designs optimized for multi-programmed workload throughput at 48mm$^2$ budget, and under different Feature Constraints

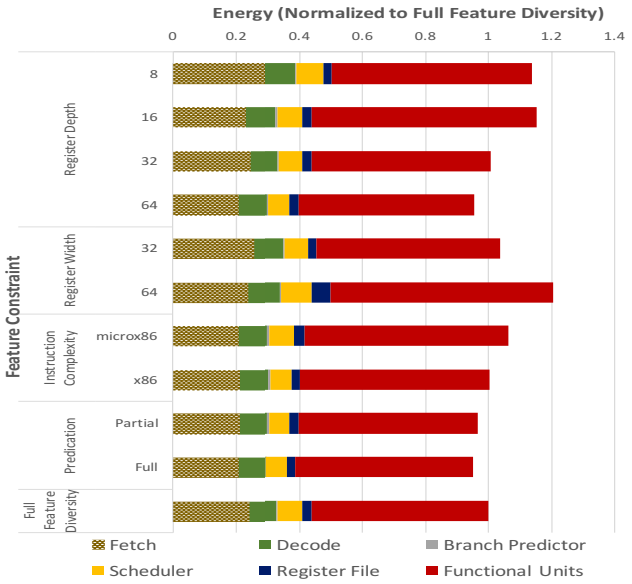


Figure 11. Processor Energy Breakdown normalized over that of Fully Custom Designs optimized for multi-programmed workload throughput at 48mm$^2$ budget, and under different Feature Constraints

available, we manage to outperform the fully heterogeneous-ISA design that implements vendor-specific ISAs, by 14.6% and reduce EDP by 3.5% under tight (5W) power constraints. The x86-ized versions of the fully heterogeneous-ISA designs again trail, but generally match up well, to the performance levels offered by vendor-specific ISA-heterogeneity. Once again, the composite-ISA design overall match the performance results of the fully heterogeneous-ISA design, while trailing slightly behind in terms of EDP.

When designs are constrained by area budget, the optimal multicore is typically composed of multiple small cores and one large core that maximizes single thread performance. In Figure 8, we evaluate the single thread performance and energy efficiency of the optimal designs under different area budgets. We observe that the composite-ISA designs sport two out-of-order *microx86* cores even under the most tightly area-constrained environment implementing different register depth and width features in each of them, allowing applications to migrate across cores and take advantage of the specific ISA features. While the fully heterogeneous-ISA design offers similar capabilities due to the area-efficient thumb cores, we note that migration across thumb and x86-64 cores is non-trivial and incurs significant overhead in comparison to the simpler migration across the two overlapping x86-based ISAs in our case. Moreover, under tight constraints, we are able to design more effective composite-ISA architectures due to the greater design options available (e.g., more efficient combination of 32-bit and 64-bit designs), saving an extra 13.2% in EDP when compared to fully heterogeneous-ISA designs.

Overall, the composite-ISA design consistently outperforms the single-ISA heterogeneous design, resulting in an average speedup of 20% and an EDP reduction of 21%.

### B. Feature Sensitivity Analysis

Owing to their feature-rich nature, composite-ISA CMPs consistently offer significant performance and energy efficiency benefits, even in scenarios where hardware heterogeneity provides diminishing returns. One of the major goals of this design space exploration is to identify specific ISA features that contribute toward these benefits, and further help architects make more efficient design choices. However, since ISA features typically manifest as components of a larger feature set a core implements, it is generally non-trivial to measure the effect of a specific feature in isolation.

In this section, we perform additional searches through our design space in order to understand the impact an ISA feature has over performance, energy, and transistor investment, by removing one axis of feature diversity at a time. As an example, consider the search for an optimal composite-ISA CMP that optimizes for multi-programmed throughput under an area budget of 48mm$^2$, but constrained to only include designs that limit the number of architectural registers to 16 in all cores. If register depth is an important feature, the optimal design from this search is expected to perform worse than the one chosen through an unconstrained search.

Figure 9 shows the result of this experiment. We make several inferences. First, constraining all cores to implement fewer than 32 architectural registers negates a significant chunk of the performance gain due to feature diversity. Most optimal designs typically employ two or more cores with a register depth of at least 32, and seldom employ cores with fewer than 16 registers. Second, the best performing designs typically include a mix of both 32-bit and 64-bit cores. While 64-bit cores are more efficient at computing on wider data types, 32-bit cores employ smaller hardware structures, saving area for other features. Designs that exclude any one of them incur 3-7% loss in performance. Third, most optimal designs employ both microx86 and x86 cores. While constraining cores to only include microx86 cores marginally affects performance, excluding them limits performance considerably. Finally, most optimal designs include both partially predicated and fully predicated cores.

We will examine these 10 constrained-optimal designs further. For example, this will allow us to compare the four-core design where all cores are *microx86* with the design where all cores are

*x86*. Figure 10 shows the transistor investment for the processor part of the real estate for each of the best designs from the above experiment. These designs were all optimized for the same area budget, but here we plot combined core area, without caches; therefore, longer bars imply that the design needed to spend more transistors on cores and sacrifice cache area to get maximum performance. We make the following observations. First, the design that constrains all cores to *microx86* takes up the least combined core area. However, owing to the area efficiency of *microx86*, it is the only design among the 10 that employs all out-of-order cores, each sporting a tournament branch predictor. Second, the design constrained to exclude *microx86* takes up the highest processor area, investing most of its transistors on functional units. Note that we always combine SIMD units with *x86* cores, and that the *microx86* cores lack any SIMD units. Third, the 64-bit-only optimal design spends more transistors on the register file and the scheduler than any other design. In that design, two out of four 64-bit cores are configured with a register depth of 64, with the remaining two configured with 32.

Figure 11 shows the processor energy breakdown by stages for each of the best designs from the above experiment. We find that the energy breakdown shows significant deviation from the corresponding area breakdown. While the decoder requires a greater portion of processor area than the fetch unit, it is the fetch unit that expends more energy during run-time since the decode pipeline is only triggered upon a micro-op cache miss, and instructions are streamed out of the micro-op cache for the most part. Interestingly, the design that constrains all cores to be configured with a register depth of 8 spends significant energy in the Fetch stage. This is due to the artificial instruction bloat caused by spills, refills, and rematerializations – a direct consequence of high register pressure. Furthermore, although the x86-only designs invested significantly in SIMD units, the energy spent by the functional units is not nearly as proportional. This is due to relatively infrequent vector activity. Finally, the 64-bit-only design continues to have high register file and scheduler energy.

### C. Feature Affinity

In this section, we study the feature affinity of the eight applications we benchmark, in two specific execution scenarios. In the first scenario, we consider a composite-ISA heterogeneous design optimized for single thread performance under a peak power budget of 10W. Recall that in such a design, hardware heterogeneity provides only marginal benefits, and most of the gains come from the fact that an application is free to migrate across different cores that each implement a diverse feature set. Therefore, such a design captures the true ISA affinity of an application.

Figure 12 shows the feature affinity in terms of the fraction of time an application spends executing on a particular feature set. First, we find that the multicore design that optimizes single thread performance exhibits significant feature diversity. In fact, by analyzing the component features of the multicore, we find that *all* features from our superset ISA have been used. Second, we find that there is significant variance in feature set preference across different applications. There is no single best feature set that is preferred by all applications. Third, we observe that there is some variance in feature set preference even within a single application's internal phases. We find that

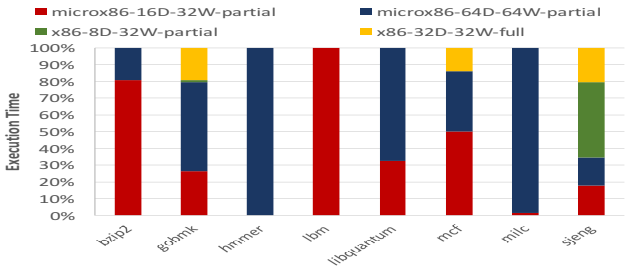


Figure 12. Execution Time Breakdown on the best composite-ISA CMP optimized for Single Thread Performance under 10W Peak Power Budget

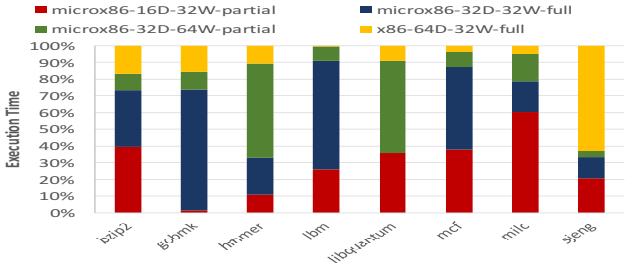


Figure 13. Execution Time Breakdown on the best composite-ISA CMP optimized for Multi-programmed Throughput under 48mm$^2$ Area Budget

most applications migrate to a core with a different feature set at least once. Fourth, the benchmark *hmmer* that exhibits significant register pressure tends to always execute on a feature set with a register depth of 64. Interestingly, we found that phases with considerable irregular branch activity, due to indirect branches and function pointer calls, prefer full predication since it eases the pressure on the branch predictor by converting some of the control flow into data flow. This is evidenced by the benchmarks *sjeng* and *gobmk*.

In the second scenario, we consider a composite-ISA design optimized for multi-programmed workload throughput under an area budget of 48mm$^2$, in which case applications typically contend for the best feature set preference, and may sometimes execute on feature sets of second preference. Figure 13 shows the results of this experiment. In sharp contrast to the design optimized for single thread performance, where applications had clear preferences, we find that *all* applications in the multi-programmed workload execute on *all* feature sets at some point of time. However, we are still able to make some high level inferences about feature affinity. For example, the benchmark *sjeng* continues to show a clear preference to *x86* over *microx86*, and both benchmarks *sjeng* and *gobmk* prefer to execute on fully predicated ISAs during phases of irregular branch activity.

### D. Migration Cost Analysis

Process migration across composite-ISA cores can involve two scenarios. In a feature upgrade scenario, the core which a process migrates to already implements a superset of the features the process was compiled to, in which case, there is *zero* binary translation or state transformation costs. On the other hand, in a feature downgrade scenario, the core to which the process migrates implements only a subset of the features the process is compiled to, necessitating minimal translation of unimplemented features. We first discuss the cost of a feature downgrade for any arbitrary code region, and then measure its performance impact on a design optimized for multi-programmed workload throughput.

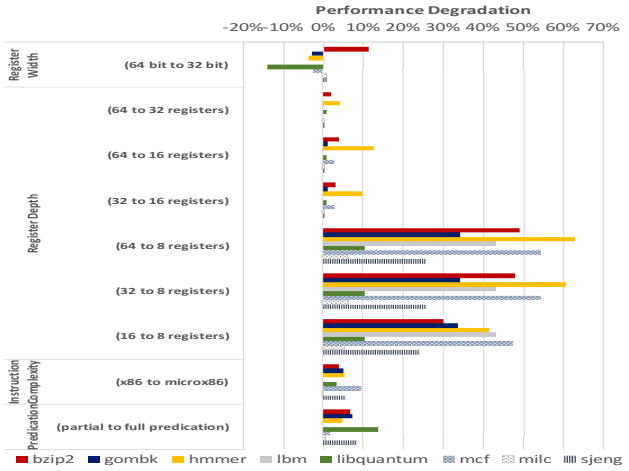


Figure 14. Feature Downgrade Cost

We measure feature downgrade costs by running each code region that corresponds to a simpoint on an artificially constrained core that only implements a subset of the features the simpoint was compiled to. Figure 14 shows the result of this experiment. We make several important observations here. First, when we downgrade from 64-bit to 32-bit cores, most of the emulation cost is negated due to the cache-efficient 32-bit cores. In fact, we achieve a speedup for some applications when we downgrade them from 64-bit to 32-bit feature sets. Second, since most applications use 32 or fewer registers, there is little emulation cost incurred due to a register depth downgrade from 64 to 32 registers. While we incur some overhead (an average of 2.7%) when we downgrade to a feature set that implements only 16 registers, there is significant overhead (an average of 33.5%) in migrating to a feature set that implements only 8 registers. In all cases, we find that the benchmark *hmmer* incurs the highest emulation overhead due to a register depth downgrade, concurring with our prior feature affinity analysis. Third, we incur an average of 5.5% overhead when we downgrade to a feature set without full predication. Finally, downgrade from *x86* to *microx86* comes at a cost of 4.2% on average. Note that we turn on full vectorization and our compiler emits SSE code for all floating-point benchmarks. However, any reasonable scheduler will not schedule simpoints with significant vector activity on *microx86* cores without SIMD support so the lack of vector support is not significantly exposed in these experiments. We limit the *x86-to-microx86* translation to simple addressing-mode transformations (as described in Section 4.2).

In our design space analysis, we needed to assume optimal selection of compiler features to make the search tractable. In the next experiment, we explore a single instantiation of one of our optimal core configurations, and a single instance (single set of features) of each compiled binary. The set of features chosen for the binary is the most common one selected for that application (among all possible scheduling permutations). We then run (again, for all permutations of our benchmark set), an experiment with four cores and four applications for 500 billion instructions. Every time one of the applications experiences a phase change that would cause us to re-shuffle job-to-core assignments, we assume a migration cost for each application that moves, and a possible downgrade cost over the next interval for each job that moves to a core that doesn't fully support the compiled features. Migration cost is measured via simulation for each binary and set of features not supported.

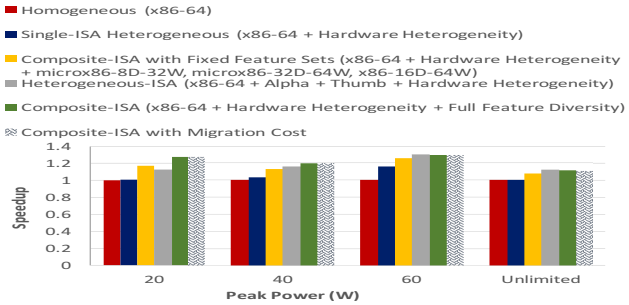


Figure 15. Multi-threaded Workload Throughput with Downgrade Cost

Figure 15 compares the designs optimized for multi-programmed workload throughput with migration cost included. Recall that in such a design, threads often contend for the best core and may not always run on their core of first preference. We observe that the performance degradation due to migrations across composite ISAs is a negligible 0.42%, on average (max 0.75%), virtually preserving all of the performance gains due to feature diversity. We attribute this to the fact that feature downgrades are infrequent and when there is one, the cost of software emulating it is minimal, due to the overlapping nature of feature sets. In fact, out of the 1863 migrations that occur in this experiment, only 125 migrations required downgrade from 64-bit to 32-bit, 171 from a register-depth of 64 to 32, 177 from a register-depth of 64 to 16, and 8 from x86 to microx86.

In summary, composite-ISA heterogeneous designs consistently outperform and use far less energy than single-ISA heterogeneous designs, generally matching or exceeding multi-vendor heterogeneous-ISA designs.

## VIII. Conclusion

This paper presents a composite-ISA architecture and compiler/runtime infrastructure that extends the advantages of multi-vendor heterogeneous-ISA architectures. It enables the full performance and energy benefits of multi-ISA design, without the issues of multi-vendor licensing, cross-ISA binary translation, and state transformation. It also gives both the processor designer and the compiler a much richer set of ISA design choices, enabling them to select and combine features that match the expected workload. This provides richer gains in efficiency than highly optimized but inflexible existing-ISA based designs. This architecture gains an average of 19% in performance and over 30% in energy savings over single-ISA heterogeneous designs. Further, it matches and in many cases outperforms multi-vendor heterogeneous-ISA designs, essentially with a single ISA.

## Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful insights. This research was supported in part by NSF Grant CNS-1652925, NSF/Intel Foundational Microarchitecture Research Grant CCF-1823444, and DARPA under the agreement number HR0011-18-C-0020. In addition, this work used the Extreme Science and Engineering Discovery Environment (XSEDE) Comet cluster at the San Diego Supercomputing Center through allocation CCR170018, which is supported by NSF Grant ACI-1548562.

## References

[1] "2nd Generation Intel Core vPro Processor Family," tech. rep., Intel, 2008.

[2] "The future is fusion: The Industry-Changing Impact of Accelerated Computing.," tech. rep., AMD, 2008.

[3] "The Benefits of Multiple CPU Cores in Mobile Devices," tech. rep., NVidia, 2010.

[4] "Qualcomm snapdragon 810 processor: The ultimate connected mobile computing processor," tech. rep., Qualcomm, 2014.

[5] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, (Piscataway, NJ, USA), pp. 13–24, IEEE Press, 2014.

[6] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, (New York, NY, USA), pp. 1–12, ACM, 2017.

[7] "Variable SMP A Multi-Core CPU Architecture for Low Power and High Performance," tech. rep., NVidia, 2011.

[8] P. Greenhalgh, "big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7," tech. rep., ARM, 2011.

[9] M. Hill and M. Marty, "Amdahl's Law in the Multicore Era," *Computer*, July 2008.

[10] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-ISA Heterogeneous Multi-core Architectures: The Potential for Processor Power Reduction," in *International Symposium on Microarchitecture*, Dec. 2003.

[11] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-ISA Heterogeneous Multi-core Architectures for Multithreaded Workload Performance," in *International Symposium on Computer Architecture*, June 2004.

[12] R. Kumar, D. M. Tullsen, and N. P. Jouppi, "Core Architecture Optimization for Heterogeneous Chip Multiprocessors," in *International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2006.

[13] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling Heterogeneous Multi-Cores through Performance Impact Estimation (pie)," in *International Symposium on Computer Architecture*, June 2012.

[14] "Apple 2017: The iphone x (ten) announced," 2017.

[15] M. DeVuyst, A. Venkat, and D. M. Tullsen, "Execution migration in a heterogeneous-isa chip multiprocessor," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2012.

[16] A. Venkat and D. M. Tullsen, "Harnessing ISA diversity: Design of a heterogeneous-ISA chip multiprocessor," in *Proceedings of the International Symposium on Computer Architecture*, 2014.

[17] A. Venkat, S. Shamasunder, H. Shacham, and D. M. Tullsen, "HIPStR: Heterogeneous-isa program state relocation," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, 2016.

[18] A. Akram and L. Sawalha, "the impact of isas on performance"," in *Proceedings of the 14th Annual Workshop on Duplicating, Deconstructing and Debunking (WDDD) associated with ISCA*, June 2017.

[19] A. Akram, "A study on the impact of instruction set architectures on processors performance," 2017.

[20] S. K. Bhat, A. Saya, H. K. Rawat, A. Barbalace, and B. Ravindran, "Harnessing energy efficiency of heterogeneous-isa platforms," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 2, pp. 65–69, 2016.

[21] A. Venkat, *Breaking the ISA Barrier in Modern Computing*. PhD thesis, UC San Diego, 2018.

[22] W. Lee, D. Sunwoo, C. D. Emmons, A. Gerstlauer, and L. K. John, "Exploring heterogeneous-isa core architectures for high-performance and energy-efficient mobile socs," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pp. 419–422, ACM, 2017.

[23] J. Nider and M. Rapoport, "Cross-isa container migration," in *Proceedings of the 9th ACM International on Systems and Storage Conference*, SYSTOR '16, (New York, NY, USA), pp. 24:1–24:1, ACM, 2016.

[24] A. Barbalace, R. Lyerly, C. Jelesnianski, A. Carno, H.-R. Chuang, V. Legout, and B. Ravindran, "Breaking the boundaries in heterogeneous-isa datacenters," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, pp. 645–659, 2017.

[25] "2nd generation intel core vpro processor family," tech. rep., Intel, 2008.

[26] "Intel go autonomous driving solutions," tech. rep., Intel, 2017.

[27] D. Wong and M. Annavaram, "Knightshift: Scaling the energy proportionality wall through server-level heterogeneity," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 119–130, IEEE Computer Society, 2012.

[28] J. Mars and L. Tang, "Whare-map: heterogeneity in homogeneous warehouse-scale computers," in *ACM SIGARCH Computer Architecture News*, vol. 41, pp. 619–630, ACM, 2013.

[29] V. Petrucci, M. A. Laurenzano, J. Doherty, Y. Zhang, D. Mosse, J. Mars, and L. Tang, "Octopus-man: Qos-driven task management for heterogeneous multicores in warehouse-scale computers," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pp. 246–258, IEEE, 2015.

[30] A. Lukefahr, S. Padmanabha, R. Das, F. M. Sleiman, R. Dreslinski, T. F. Wenisch, and S. Mahlke, "Composite cores: Pushing heterogeneity into a core," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 317–328, IEEE Computer Society, 2012.

[31] S. Padmanabha, A. Lukefahr, R. Das, and S. Mahlke, "Dynamos: dynamic schedule migration for heterogeneous cores," in *Proceedings of the 48th International Symposium on Microarchitecture*, pp. 322–333, ACM, 2015.

[32] A. Lukefahr, S. Padmanabha, R. Das, F. M. Sleiman, R. G. Dreslinski, T. F. Wenisch, and S. Mahlke, "Exploring fine-grained heterogeneity with composite cores," *IEEE Transactions on Computers*, vol. 65, no. 2, pp. 535–547, 2016.

[33] A. Lukefahr, S. Padmanabha, R. Das, R. Dreslinski Jr, T. F. Wenisch, and S. Mahlke, "Heterogeneous microarchitectures trump voltage scaling for low-power cores," in *Proceedings of the 23rd international conference on Parallel architectures and compilation*, pp. 237–250, ACM, 2014.

[34] M. Gupta, V. Sridharan, D. Roberts, A. Prodromou, A. Venkat, D. Tullsen, and R. Gupta, "Reliability-aware data placement for heterogeneous memory architecture," in *High Performance Computer Architecture (HPCA), 2018 IEEE International Symposium on*, pp. 583–595, IEEE, 2018.

[35] S. Srinivasan, N. Kurella, I. Koren, and S. Kundu, "Exploring heterogeneity within a core for improved power efficiency," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 1057–1069, 2016.

[36] H. R. Ghasemi, U. R. Karpuzcu, and N. S. Kim, "Comparison of single-isa heterogeneous versus wide dynamic range processors for mobile applications," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*, pp. 304–310, IEEE, 2015.

[37] S. Padmanabha, A. Lukefahr, R. Das, and S. Mahlke, "Trace based phase prediction for tightly-coupled heterogeneous cores," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 445–456, ACM, 2013.

[38] M. A. Kinsy, S. Khadka, M. Isakov, and A. Farrukh, "Hermes: Secure heterogeneous multicore architecture design," in *Hardware Oriented Security and Trust (HOST), 2017 IEEE International Symposium on*, pp. 14–20, IEEE, 2017.

[39] K. Van Craeynest and L. Eeckhout, "Understanding fundamental design choices in single-isa heterogeneous multicore architectures," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 9, no. 4, p. 32, 2013.

[40] K. Van Craeynest, S. Akram, W. Heirman, A. Jaleel, and L. Eeckhout, "Fairness-aware scheduling on single-isa heterogeneous multi-cores," in *Parallel Architectures and Compilation Techniques (PACT), 2013 22nd International Conference on*, pp. 177–187, IEEE, 2013.

[41] S. Akram, J. B. Sartor, K. V. Craeynest, W. Heirman, and L. Eeckhout, "Boosting the priority of garbage: Scheduling collection on heterogeneous multicore processors," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 13, no. 1, p. 4, 2016.

[42] A. Naithani, S. Eyerman, and L. Eeckhout, "Reliability-aware scheduling on heterogeneous multicore processors," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pp. 397–408, IEEE, 2017.

[43] N. Markovic, D. Nemirovsky, O. Unsal, M. Valero, and A. Cristal, "Thread lock section-aware scheduling on asymmetric single-isa multi-core," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 160–163, 2015.

[44] N. Markovic, D. Nemirovsky, V. Milutinovic, O. Unsal, M. Valero, and A. Cristal, "Hardware round-robin scheduler for single-isa asymmetric multi-core," in *European Conference on Parallel Processing*, pp. 122–134, Springer, 2015.

[45] D. Nemirovsky, T. Arkose, N. Markovic, M. Nemirovsky, O. Unsal, and A. Cristal, "A machine learning approach for performance prediction and scheduling on heterogeneous cpus," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2017 29th International Symposium on*, pp. 121–128, IEEE, 2017.

[46] L. Sawalha, S. Wolff, M. P. Tull, and R. D. Barnes, "Phase-guided scheduling on single-isa heterogeneous multicore processors," in *Digital System Design (DSD), 2011 14th Euromicro Conference on*, pp. 736–745, IEEE, 2011.

[47] A. Adileh, S. Eyerman, A. Jaleel, and L. Eeckhout, "Mind the power holes: Sifting operating points in power-limited heterogeneous multicores," *IEEE Computer Architecture Letters*, vol. 16, no. 1, pp. 56–59, 2017.

[48] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," in *ACM SIGPLAN Notices*, vol. 48, pp. 77–88, ACM, 2013.

[49] A. Naithani, S. Eyerman, and L. Eeckhout, "Optimizing soft error reliability through scheduling on heterogeneous multicore processors," *IEEE Transactions on Computers*, vol. 67, no. 6, pp. 830–846, 2018.

[50] J. Chen and L. K. John, "Efficient program scheduling for heterogeneous multi-core processors," in *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*, pp. 927–930, IEEE, 2009.

[51] J. Chen and L. K. John, "Energy-aware application scheduling on a heterogeneous multi-core system," in *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*, pp. 5–13, IEEE, 2008.

[52] J. A. Winter and D. H. Albonesi, "Scheduling algorithms for unpredictably heterogeneous cmp architectures," in *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pp. 42–51, IEEE, 2008.

[53] J. A. Winter, D. H. Albonesi, and C. A. Shoemaker, "Scalable thread scheduling and global power management for heterogeneous many-core architectures," in *Parallel Architectures and Compilation Techniques (PACT), 2010 19th International Conference on*, pp. 29–39, IEEE, 2010.

[54] E. Forbes and E. Rotenberg, "Fast register consolidation and migration for heterogeneous multi-core processors," in *Computer Design (ICCD), 2016 IEEE 34th International Conference on*, pp. 1–8, IEEE, 2016.

[55] E. Forbes, Z. Zhang, R. Widialaksono, B. Dwiel, R. B. R. Chowdhury, V. Srinivasan, S. Lipa, E. Rotenberg, W. R. Davis, and P. D. Franzon, "Under 100-cycle thread migration latency in a single-isa heterogeneous multi-core processor," in *Hot Chips 27 Symposium (HCS), 2015 IEEE*, pp. 1–1, IEEE, 2015.

[56] A. Venkat, A. Krishnaswamy, K. Yamada, and P. R. Shanmugavelayutham, "Binary translator driven program state relocation," Sept. 15 2015. US Patent 9,135,435.

[57] "Ibm's power chips hit the big time at google," tech. rep., PCWorld, 2016.

[58] W. Wang, P.-C. Yew, A. Zhai, S. McCamant, Y. Wu, and J. Bobba, "Enabling cross-isa offloading for cots binaries," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 319–331, ACM, 2017.

[59] A. Barbalace, M. Sadini, S. Ansary, C. Jelesnianski, A. Ravichandran, C. Kendir, A. Murray, and B. Ravindran, "Popcorn: Bridging the programmability gap in heterogeneous-isa platforms," in *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, (New York, NY, USA), pp. 29:1–29:16, ACM, 2015.

[60] T. Li, P. Brett, R. Knauerhase, D. Koufaty, D. Reddy, and S. Hahn, "Operating System Support for Overlapping-ISA Heterogeneous Multi-Core Architectures," in *International Symposium on High Performance Computer Architecture*, Jan. 2010.

[61] E. Blem, J. Menon, and K. Sankaralingam, "Power Struggles: Revisiting the RISC vs. CISC Debate on Contemporary ARM and x86 Architectures," in *International Symposium on High Performance Computer Architecture*, Feb. 2013.

[62] A. S. Waterman, *Design of the RISC-V Instruction Set Architecture*. University of California, Berkeley, 2016.

[63] A. Raveendran, V. B. Patil, D. Selvakumar, and V. Desalphine, "A risc-v instruction set processor-micro-architecture design and analysis," in *VLSI Systems, Architectures, Technology and Applications (VLSI-SATA), 2016 International Conference on*, pp. 1–7, IEEE, 2016.

[64] J. Huh, D. Burger, and S. W. Keckler, "Exploring the design space of future cmps," in *Parallel Architectures and Compilation Techniques, 2001. Proceedings. 2001 International Conference on*, pp. 199–210, IEEE, 2001.

[65] M. Monchiero, R. Canal, and A. Gonzalez, "Power/performance/thermal design-space exploration for multicore architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 5, pp. 666–681, 2008.

[66] M. Monchiero, R. Canal, and A. González, "Design space exploration for multicore architectures: a power/performance/thermal view," in *Proceedings of the 20th annual international conference on Supercomputing*, pp. 177–186, ACM, 2006.

[67] L. Strozek and D. Brooks, "Energy-and Area-Efficient Architectures through Application Clustering and Architectural Heterogeneity," *ACM Transactions on Architecture and Code Optimization*, 2009.

[68] N. Chitlur, G. Srinivasa, S. Hahn, P. K. Gupta, D. Reddy, D. Koufaty, P. Brett, A. Prabhakaran, L. Zhao, N. Ijih, *et al.*, "Quickia: Exploring heterogeneous architectures on real prototypes," in *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*, pp. 1–8, IEEE, 2012.

[69] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi, and E. Rotenberg, "Fabscalar: composing synthesizable rtl designs of arbitrary cores within a canonical superscalar template," in *ACM SIGARCH Computer Architecture News*, vol. 39, pp. 11–22, ACM, 2011.

[70] N. Choudhary, S. Wadhavkar, T. Shah, H. Mayukh, J. Gandhi, B. Dwiel, S. Navada, H. Najaf-abadi, and E. Rotenberg, "Fabscalar: Automating superscalar core design," *IEEE Micro*, vol. 32, no. 3, pp. 48–59, 2012.

[71] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, M. Matl, and D. Wentzlaff, "Openpiton: An open source manycore research framework," *ACM SIGOPS Operating Systems Review*, vol. 50, no. 2, pp. 217–232, 2016.

[72] M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, J. Balkind, A. Lavrov, M. Shahrad, S. Payne, and D. Wentzlaff, "Piton: A manycore processor for multitenant clouds," *Ieee micro*, vol. 37, no. 2, pp. 70–80, 2017.

[73] K. Lim, J. Balkind, and D. Wentzlaff, "Juxtapiton: Enabling heterogeneous-isa research with RISC-V and SPARC FPGA soft-cores," *CoRR*, vol. abs/1811.08091, 2018.

[74] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *ACM SIGARCH Computer Architecture News*, vol. 42, pp. 97–108, IEEE Press, 2014.

[75] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "The aladdin approach to accelerator design and modeling," *IEEE Micro*, vol. 35, no. 3, pp. 58–70, 2015.

[76] G. J. Chaitin, M. A. Auslander, A. K. Chandra, J. Cocke, M. E. Hopkins, and P. W. Markstein, "Register allocation via coloring," *Computer languages*, vol. 6, no. 1, pp. 47–57, 1981.

[77] M. Auslander and M. Hopkins, "An overview of the pl. 8 compiler," in *ACM SIGPLAN Notices*, vol. 17, pp. 22–31, ACM, 1982.

[78] F. C. Chow and J. L. Hennessy, "The priority-based coloring approach to register allocation," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 12, no. 4, pp. 501–536, 1990.

[79] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck, "Efficiently computing static single assignment form and the control dependence graph," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 13, no. 4, pp. 451–490, 1991.

[80] P. Briggs, K. D. Cooper, and L. Torczon, "Rematerialization," *ACM SIGPLAN Notices*, vol. 27, no. 7, pp. 311–321, 1992.

[81] D. Bernstein, M. C. Golumbic, Y. Mansour, R. Y. Pinter, D. Q. Goldin, H. Krawczyk, and I. Nahshon, "Spill code minimization techniques for optimizing compliers," in *ACM SIGPLAN Notices*, vol. 24, pp. 258–263, ACM, 1989.

[82] R. Gupta and R. Bodík, "Register pressure sensitive redundancy elimination," *CC*, vol. 99, pp. 107–121, 1999.

[83] K. D. Cooper and L. T. Simpson, "Live range splitting in a graph coloring register allocator," in *International Conference on Compiler Construction*, pp. 174–187, Springer, 1998.

[84] P. Zhao and J. N. Amaral, "To inline or not to inline? enhanced inlining decisions," in *International Workshop on Languages and Compilers for Parallel Computing*, pp. 405–419, Springer, 2003.

[85] ARM Limited, *ARM Cortex-A Series Programmers Guide for ARMv8-A*.

[86] L. George and A. W. Appel, "Iterated register coalescing," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1996.

[87] G.-Y. Lueh, T. Gross, and A.-R. Adl-Tabatabai, "Fusion-based register allocation," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 2000.

[88] P. Bergner, P. Dahl, D. Engebretsen, and M. O'Keefe, "Spill code minimization via interference region spilling," *ACM SIGPLAN Notices*, 1997.

[89] C. Lattner and V. Adve, "Llvm: A compilation framework for lifelong program analysis & transformation," in *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*, p. 75, IEEE Computer Society, 2004.

[90] B. C. Lopes, R. Auler, L. Ramos, E. Borin, and R. Azevedo, "Shrink: Reducing the isa complexity via instruction recycling," in *ACM SIGARCH Computer Architecture News*, 2015.

[91] E. Borin, M. Breternitz, Y. Wu, and G. Araujo, "Clustering-based microcode compression," in *Computer Design, 2006. ICCD 2006. International Conference on*, pp. 189–196, IEEE, 2007.

[92] E. Borin, G. Araujo, M. Breternitz Jr, and Y. Wu, "Structure-constrained microcode compression," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2011 23rd International Symposium on*, pp. 104–111, IEEE, 2011.

[93] E. Borin, G. Araujo, M. Breternitz, and Y. Wu, "Microcode compression using structured-constrained clustering," *International Journal of Parallel Programming*, vol. 42, no. 1, pp. 140–164, 2014.

[94] A. Baumann, "Hardware is the new software," in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, HotOS '17, 2017.

[95] J. R. Allen, K. Kennedy, C. Porterfield, and J. Warren, "Conversion of control dependence to data dependence," in *Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pp. 177–189, ACM, 1983.

[96] S. A. Mahlke, D. C. Lin, W. Y. Chen, R. E. Hank, and R. A. Bringmann, "Effective compiler support for predicated execution using the hyperblock," in *ACM SIGMICRO Newsletter*, vol. 23, pp. 45–54, IEEE Computer Society Press, 1992.

[97] S. A. Mahlke, R. E. Hank, R. A. Bringmann, J. C. Gyllenhaal, D. M. Gallagher, and W.-m. W. Hwu, "Characterizing the impact of predicated execution on branch prediction," in *Proceedings of the 27th annual international symposium on Microarchitecture*, pp. 217–227, ACM, 1994.

[98] S. A. Mahlke, R. E. Hank, J. E. McCormick, D. I. August, and W.-M. W. Hwu, "A comparison of full and partial predicated execution support for ilp processors," *ACM SIGARCH Computer Architecture News*, vol. 23, no. 2, pp. 138–150, 1995.

[99] D. I. August, W.-m. W. Hwu, and S. A. Mahlke, "A framework for balancing control flow and predication," in *Microarchitecture, 1997. Proceedings., Thirtieth Annual IEEE/ACM International Symposium on*, pp. 92–103, IEEE, 1997.

[100] M. J. Flynn, "Some computer organizations and their effectiveness," *IEEE transactions on computers*, vol. 100, no. 9, pp. 948–960, 1972.

[101] R. Duncan, "A survey of parallel computer architectures," *Computer*, vol. 23, no. 2, pp. 5–16, 1990.

[102] Intel, *Intel 64 and IA-32 Architectures Software Developers Manual*.

[103] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The M5 Simulator: Modeling Networked Systems," *Micro, IEEE*, 2006.

[104] P. Smith and N. C. Hutchinson, "Heterogeneous Process Migration: The Tui System," *Software-Practice and Experience*, 1998.

[105] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator," in *USENIX Technical Conference*, Apr. 2005.

[106] B. Solomon, A. Mendelson, R. Ronen, D. Orenstien, and Y. Almog, "Micro-operation cache: A power aware frontend for variable instruction length isa," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 5, pp. 801–811, 2003.

[107] M. Taram, A. Venkat, and D. Tullsen, "Mobilizing the micro-ops: Exploiting context sensitive decoding for security and energy efficiency," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 624–637, IEEE, 2018.

[108] M. Taram, A. Venkat, and D. M. Tullsen, "Context-sensitive fencing: Securing speculative execution via microcode customization," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[109] V. Madduri, R. Segelken, and B. Toll, "Method and apparatus for variable length instruction parallel decoding," in *United States Patent Application 10/331,335*.

[110] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002.

[111] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder, "Using SimPoint for Accurate and Efficient Simulation," in *ACM SIGMETRICS Performance Evaluation Review*, June 2003.

[112] Intel Corporation, *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. No. 248966-018, March 2009.

[113] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "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*, MICRO 42, (New York, NY, USA), pp. 469–480, ACM, 2009.

[114] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, *et al.*, "Xsede: accelerating scientific discovery," *Computing in Science & Engineering*, vol. 16, no. 5, pp. 62–74, 2014.

[115] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark Silicon and the End of Multicore Scaling," in *International Symposium on Computer Architecture*, June 2011.