Lichen: Leveraging Coupled Heterogeneity

Prakhar Diwan[1], Nirmal Kumar Boran[*,2], and Virendra Singh[1]

[1]Indian Institute of Technology Bombay, India
{prakhar.diwan,viren}@ee.iitb.ac.in

[2]National Institute of Technology Calicut, India
nirmalkboran@nitc.ac.in

Abstract—**Heterogeneous chip multiprocessors (CMPs) have shown significant promise for improving energy efficiency. They utilize cores varying across heterogeneity dimensions such as execution semantics and microarchitectural resources. Recently, instruction set architecture (ISA) has emerged as another crucial dimension, with heterogeneous-ISA CMPs achieving significant performance and energy efficiency benefits over single-ISA counterparts. However, heterogeneous-ISA CMPs utilized coarse-grained and uniform-length program phases, across different heterogeneity dimensions, limiting the potential benefits; as each dimension has corresponding optimal phase length. Accurately scheduling such non-uniform length program phases is critical to realize this potential. We propose Lichen, a method to improve energy efficiency of heterogeneous-ISA CMPs by leveraging diversity between ISA and execution semantics. Lichen uses finer switching between different ISA cores and execution modes (in-order/out-of-order) within them by dynamic morphing; and accurately schedules fine-grained program phases. Overall, Lichen achieves 15.36% energy savings with 6.35% performance degradation over heterogeneous-ISA CMPs, leading to 9.63% reduction in energy delay product (EDP), with minimal hardware overhead. The proposed CART algorithm attains 85.44% and 91.35% accuracy in predicting optimal ISA core and execution mode, respectively.**

Index Terms—**Heterogeneous architectures, Energy efficiency, ISA, Execution Semantics, Dynamic scheduling**

I. Introduction

Advances in VLSI technology have led to smaller transistors, allowing Moore's law to continue with increasing transistor counts on a single chip. However, the reduction in transistor threshold voltages has plateaued, limiting the number of active transistors that can operate within the power budget. Heterogeneous CMPs [1] were introduced to optimize energy efficiency by strategically using chip area, integrating cores with varying resources on the same chip. These consisted multiple cores with different performance and power characteristics. The applications were dynamically migrated to optimal core, which was most efficient in meeting their runtime demands. So far heterogeneity has been exploited across numerous dimensions such as core sizes [1] and execution schemes [2], [3], achieving better energy-efficiency than homogeneous CMPs. Heterogeneous CMPs are widely utilized in industry [4], showcasing their effectiveness.

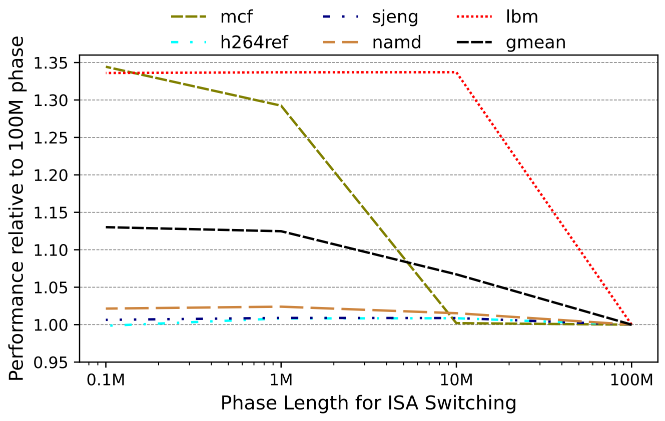
However, these architectures were constrained to single-ISA due to large migration overheads highlighted by Tui system [5], thereby neglecting the potential diversity across different ISAs, which are designed with different objectives such as reducing code size or lowering hardware complexity. DeVuyst et al. [6] proposed heterogeneous ISA CMP and faster migration methods for switching execution between different ISA cores. Cross-ISA migration was performed at equivalence points (where memory state was consistent across different ISA executables) or dynamic binary translation was done till an equivalence point was reached, latter costing significant latency. Venkat et al. [7] performed design space exploration for heterogeneous-ISA CMPs, across different ISAs, execution semantics, cache hierarchies and various other microarchitectural features. They observed different program phases performed better with different ISAs and named this ISA affinity. This was due to factors like register pressure and code density, which varied depending on program execution in different ISAs. However, they utilized oracular scheduling and uniform phase length ($\sim 100M$ instructions) for analysing different heterogeneity dimensions. Heterogeneity dimensions such as ISA [8] and execution semantics [2] exhibit more effectiveness at finer granularities, with optimal phase length being possibly different for each dimension. Hence, heterogeneous-ISA CMPs were unable to realize the full potential of different heterogeneity dimensions.

We chose two heterogeneity dimensions: ISA and execution semantics. We picked two widely used ISAs: ARM and X86 for our experiments, and evaluated five benchmarks from SPEC2006 [9] suite, namely *mcf, lbm, namd, sjeng,* and *h264ref*. A dual-core system, comprising X86 and ARM out-of-order cores was used, with its configuration in Table II.

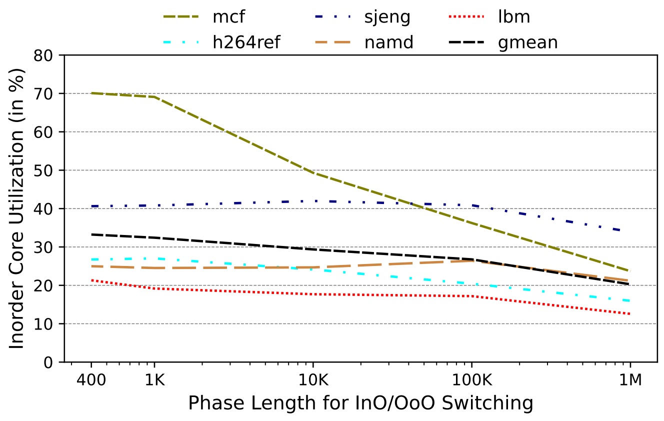
First, we observed that ISA affinity varied at finer phases due to ISA-impacted parameters such as register pressure and code density changing at smaller granularities. To find the optimal phase length for switching between different ISA cores, we performed phase length sensitivity analysis, shown in Figure 1a. We assumed oracular scheduling for switches between different ISA cores and used prior cross-ISA migration technique [6], however, we migrated only at equivalence points (function call sites for our work). We observed significant speedup for *lbm* and *mcf* benchmarks at finer phases. However, at very fine granularity ISA-impacted parameters rapidly fluctuated, leading to large number of switches. This incurred significant overhead, thereby, saturating performance gains (indicated by *gmean*) at $\sim 0.1M$ instructions. Hence, we select function call sites spaced by atleast $1M$ instructions for switching between different ISA cores in our proposal.

Second, we analysed execution semantics for heterogeneous-ISA CMPs. We observed that in-order execution can perform close to out-of-order execution for low-instruction level parallelism (ILP), memory-intensive or branch-heavy program phases. It has been shown that using tightly-coupled heterogeneous cores [2] or dynamically polymorphing out-of-order core into in-order core [3] led to lower overhead for migrating between in-order and out-of-order modes. To find the optimal phase length for

*Corresponding Author



(a) Instruction Set Architecture: ARM/X86



(b) Execution Semantics: in-order(InO)/out-of-order(OoO)

Fig. 1: Analysis of finer phase lengths for ISAs and in-order/out-of-order switching in heterogeneous-ISA CMPs

in-order/out-of-order switching in heterogeneous-ISA CMP, we observed in-order core's utilization at various phase lengths, shown in Figure 1b. For switching between different ISA cores, we used a phase length of $1M$ instructions, as found best for performance from Figure 1a. During the single-ISA execution of $1M$ instruction phases, we morphed between out-of-order and in-order modes as per prior work [3] and allowed 5% performance degradation compared to only out-of-order execution. We assumed oracular scheduling for switches between different ISAs and in-order/out-of-order modes. Figure 1b depicted that in-order utilization increases for all benchmarks. However, utilization saturated for $1K$ instructions phase length due to rising number of migrations between in-order and out-of-order modes after it. Hence, we utilize $1K$ instructions for switching between InO and OoO modes in our proposal, *Lichen*. Overall, *Lichen* makes the following contributions:

- It uses optimal phase lengths for switching across different ISA cores and execution modes, thereby, exploiting both heterogeneity dimensions for energy efficiency.
- It proposes scheduling mechanism (CART-decision tree based) to accurately map non-uniform length program phases across different ISA cores and execution modes.

II. Related Work

Various works [1]–[3], [6]–[8], [10]–[13] have been proposed related to heterogeneous architectures. Kumar et al. [1] have shown that single ISA heterogeneous CMPs can significantly reduce power compared to homogeneous CMPs. Over the years, the industry has utilized heterogeneous computing to improve energy efficiency [4]. It started with ARM's big.LITTLE [4] for mobile space, which consisted of larger out-of-order cores along with smaller in-order cores. Lukefahr et al. [2] proposed Composite Cores, an architecture that reduced migration overheads by accommodating heterogeneity (in-order and out-of-order execution) within a core. Sudarshan et al. [3] presented the idea of dynamically morphing an out-of-order core into an in-order core as per the program's runtime requirements, already existent in modern processors for debug [14]. DeVuyst et al. [6] introduced heterogeneous-ISA CMPs (with shared main memory) and cross-ISA migration technique, proposing compiler modifications to maintain consistent memory image. Since the stack section is optimized according to ISA, runtime transformations were performed using compiler metadata. Blem et al. [10] claimed that ISA being RISC or CISC does not impact systems' performance and energy efficiency. Venkat et al. [7] showcased the benefits of ISA diversity by extending heterogeneous-ISA to a more diverse set of ISAs (Thumb, Alpha, X86), achieving significant performance and energy efficiency improvements. They also extended migration techniques across these ISAs, with overhead averaging 100 μs. To achieve heterogeneous-ISA gains from single ISA, [13] proposes a heterogeneous CMP consisting of cores built on composite instruction feature sets. Barbalace et al. [15] extended the energy benefits of ISA heterogeneity to server space by presenting system software for execution migration between ARM and X86 servers. Function-wise scheduling [8] achieved significant performance benefits for heterogeneous-ISA CMPs by proposing a low overhead cross-ISA migration scheme. However, it's heuristic suffered from low-accuracy in predicting the affine ISA core. To derive maximum benefits from heterogeneous-ISA CMPs, efficient scheduling algorithms have been proposed [11], [12]. These schedulers model inactive-ISA core's performance based on the microarchitectural parameters of active-ISA core and schedule program phases on the predicted affine ISA core.

III. Lichen

A. Overview

Heterogeneous-ISA CMP achieve significant performance and energy efficiency benefits over single-ISA counterparts. However, it incurs two drawbacks: (a) oracular scheduling, which is impractical and (b) inefficient exploitation of heterogeneity dimensions, due to coarse-grained switching and uniform phase lengths. We overcome these limitations through *Lichen*, a scheme which leverages heterogeneity in ISA (ARM(v7)/X86) and execution semantics (in-order/out-of-order), saving substantial energy with negligible overhead.

Lichen effectively utilizes heterogeneous-ISA CMPs. It exploits ISA affinity for performance at finer granularity ($1M$ instructions) compared to prior proposal [7] where they did it for $100M$ instructions. This is achieved due to lower cross-ISA migration overhead by migrating only at function call sites, where the memory state is consistent between different ISA executables. We add a 20-bit dynamic instruction saturating counter to indicate cross-ISA migration point. It is reset after every heterogeneous-ISA phase, regardless of migration decision. The migration decision is taken by scheduler based on affine ISA predicted, described further in Section III-B. Given the counter has crossed $1M$ count and scheduler decides to migrate to another-ISA core, the executing ISA core waits till the next function call is committed. Once it is committed it performs cross-ISA migration, described in Section III-C1. The inactive-ISA core is power-gated to conserve energy.

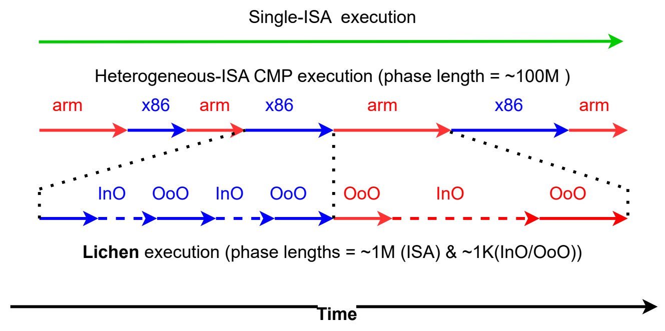

Single-ISA execution
Heterogeneous-ISA CMP execution (phase length = ~100M)
arm
x86
arm
x86
arm
x86
arm
InO
OoO
InO
OoO
OoO
InO
OoO
Lichen execution (phase lengths = ~1M (ISA) & ~1K(InO/OoO))
Time

Fig. 2: Lichen switches ISAs and execution semantics over time to effectively use both the heterogeneity dimensions

We observed that heterogeneity between in-order and out-of-order modes is harnessed effectively at finer phases, even for heterogeneous-ISA execution from Figure 1b. For reduced migration overheads between in-order and out-of-order modes, we leverage the morphing capability of superscalar out-of-order cores [3] into in-order cores at runtime. This morphing capability is already present in many Intel processors for debug [14], hence incurring negligible hardware overhead over out-of-order cores. During the in-order mode, the microarchitectural structures facilitating out-of-order execution are powered off (clock-gated), saving upon power consumption. Overall, *Lichen* switches between in-order and out-of-order modes at $1K$ instruction phases within single-ISA phases of $1M$ instructions, shown in Figure 2; hence, decoupling the scheduling of ISA and execution semantics. It allows 5% performance degradation compared to only out-of-order execution. We restrict ending $0.1M$ instructions to execute on out-of-order mode to estimate performance on inactive-ISA core, with $0.9M$ instruction mark indicated through the same 20-bit dynamic instruction counter. Migration and scheduling between in-order and out-of-order modes within single-ISA phase is explained in Sections III-C2 and III-B2.

B. Scheduling

We decouple the scheduling for *Lichen* into: 1) predicting affine-ISA core at $1M$ instruction granularity and 2) deciding efficient execution mode (in-order/out-of-order) every $1K$ instructions within single-ISA phases of $1M$ instructions. We utilize 9 microarchitectural parameters of active-ISA core and active-mode to predict the ISA affinity and performance of in-active mode, respectively. These are shown in Table I, most are readily available via performance counters in modern processors [16]. Instruction-level parallelism and memory-level parallelism were estimated using methods from prior works [2], [11]. We performed exploration across a larger set of parameters and pruned them to these parameters, based on weights allocated in decision models. Upon analysing different algorithms for scheduling, we found CART decision tree to be optimal for both the tasks, as also shown later in Section V-B. It is a supervised learning algorithm that constructs a tree-like model during the training phase, which is then utilized to predict new samples during the inference phase. During training, the decision tree algorithm partitions the original dataset S into subsets and continues this process recursively until each subset cannot be further divided. At each step of this partitioning process, the decision tree algorithm selects the best splitting strategy based on certain criteria. We have considered splitting strategy as *entropy*, maximum depth of tree as 3, minimum number of samples required to be at a leaf node as 5, and random state set to 100 for permuting the features randomly at each split. Overall, we utilize 6 different CART models for scheduling between different ISAs (2: ARM and X86) and execution semantics (4: ARM OoO, ARM InO, X86 OoO and X86 InO). The former 2 models are trained using data from $0.1M$ instructions phases, whereas, the latter 4 using $1K$ instructions phases. However, same tree structure is re-used across all 6 models due to their mutually exclusive operation (described further in following subsections); with 6 sets of coefficients stored in memory-mapped programmable registers. The models are trained offline and make decision online for the reported results in Section V.

Parameter	Purpose
L1-I misses	Stalls due to cache misses
L1-D misses	
L2 misses	
ROB full events	Execution pipeline stalls
IQ full events	
ILP	Parallelism of the executing program
MLP	
Branch mispredictions	Performance impact of branch predictor
Float instruction count	ISA-specific parameters

TABLE I: Microarchitectural parameters used by scheduler

1) ISA: The decision for ISA scheduling is taken after every $1M$ instruction phase. The micro-architectural parameters for the last $0.1M$ instructions of each $1M$ instruction phase are taken to predict the better performing ISA (affine ISA) for next phase. We assume that the behaviour of program does not change rapidly, as also observed by assessing ISA-impacted parameters and performance using affine ISA across different phase lengths in Figure 1a. We utilized 30,000 phases of $1M$ instructions, using with last $0.1M$ for scheduler input and utilized the simulated performance on both ISA cores for labelling the instruction phases; with 70% of these phases used for training and the rest for testing.

2) Execution Semantics: We re-utilize the CART decision tree algorithm for predicting the efficient execution scheme, given 5% performance penalty over only out-of-order execution. We load the model with different coefficients and utilize the same microarchitectural parameters but at a finer granularity of $1K$ instructions within a single ISA phase of $1M$ instructions. We reused data from ISA scheduler at finer phases (1K instructions) to train the InO/OoO scheduler, with 70% data used for training and remaining for testing. The data was labelled using the simulated performance of both InO and OoO cores for an ISA. The microarchitectural parameters of current phase are used to decide the execution scheme for upcoming phase. The last $0.1M$ instructions of $1M$ instruction single-ISA phase are constrained to be executed only using out-of-order mode, to accurately predict the affine ISA and reuse the CART decision tree model.

C. Migration

1) Between different ISA cores: We support heterogeneous-ISA execution by building a *fat* binary, using unified address space and common address translation scheme as in prior works [6], [7]. *Fat* binary consists of target-independent data sections, target-specific code sections and migration metadata. Migration metadata consists of compilation information about function call sites, variable locations, frame layouts, caller-saved and callee-saved registers' spill areas for both ISAs and mapping between program objects that are created using an intermediate representation. During migration, the aim is to transform the ISA-specific program state as per the target ISA. By keeping the number of objects, relative order, size, alignment and padding rules identical in each section between ISAs; transformation costs are reduced. For ARM and X86 ISAs, we use the same endianness and stack growth direction; and choose little-endian and downward direction. Every function's definition is placed at an identical virtual address along with the order of functions in binary being identical across ISAs, to avoid fixing function pointers post-migration. This requires identical function sizes across ISAs, which is achieved by padding NOP instructions. With these compiler modifications, only the register state and stack portions require to be transformed on migration. Local variables and return addresses of open function calls comprise the stack portion, which must be modified on migration, along with the register state for target ISA's execution. We use LLVM (llc and clang) [17] for analysis of function stack-frames for stack transformation. With different calling conventions and number of architectural registers, variables may be expected at different locations across ARM and X86 codes. Hence, we transform the architecture-specific program state (stack and register state) using migration metadata. However, unlike prior works [6], [7] we utilize only function call sites as migration points, avoiding costly dynamic binary translation.

2) Between in-order/out-of-order modes: When switching from out-of-order to in-order mode, pipeline is not drained and instructions are fetched after last committed instruction. From in-order to out-of-order mode switch, the start and end pointers of ROB are initialized to point to same entry. Since both in-order and out-of-order execution are supported within the same core through polymorphing [3], [14], architectural state transfer and warming up of microarchitectural structures (caches and branch predictors) is not required. We utilise identical superscalar width between both modes, hence, require only clock gating out-of-order execution facilitating structures such as re-order buffer (ROB), load store queue (LSQ) and register alias table (RAT), when switching between out-of-order to in-order/vice-versa.

IV. Experimental Setup

Dual Core System with ARM and X86 cores	
OoO/InO Core (ARM & X86)	Superscalar Width=2, 3 GHz Clock , 108-entry ROB, 16-entry LQ, 16-entry SQ, 36-entry IQ
Functional Units ARM/X86	IntALU=2, IntMultDiv=1, FPALU=0/1, FPMultDiv= 0/1, SIMD Units=0/1
L1 I-Cache	32 KB, 8-way, 2 cycle latency
L1 D-Cache	32 KB, 8-way, 2 cycle latency
Last Level Cache (L2)	2 MB, 16-way, 15 cycle latency

TABLE II: System Configuration

We consider two ISAs in this work - ARM(v7) and X86. Benchmarks were compiled for X86 and cross-compiled for ARM, similar to prior works. We use a dual-core heterogeneous-ISA system, consisting ARM and X86 out-of-order cores with configurations listed in Table II. Private 2-level cache hierarchies [7] are used for each core. The clock rate for both the cores is set to 3GHz and 32nm technology node used for analysis. We simulated SPEC CPU2006 [9] benchmarks on the system using gem5 architectural simulator [18]. Cross-ISA migration was performed as in prior works [6], however, switching only at function call sites (equivalence points). Switching between in-order and out-of-order mode and vice-versa was performed as explained in Section III-C2. We performed power analysis using mcPAT [19].

V. Results

We present the results for *Lichen* relative to heterogeneous-ISA chip multiprocessor (HISACMP), using oracle and proposed scheduling schemes. Oracle indicates perfect assignment of all phases to corresponding optimal ISA core (based on performance) and execution mode (based on performance penalty ¡ 5%). HISACMP uses oracle scheduling for switching to affine-ISA core, every $100M$ instructions. We allow 5% performance penalty for *Lichen* relative to out-of-order execution using affine ISA for $1M$ instruction phases.

A. Affinity towards different ISAs & Execution Semantics

We obtained the affinity of benchmarks towards ISA and execution semantics by observing the execution time using each ISA core and mode, given oracular scheduling. Benchmarks *soplex, bzip2, mcf, sjeng, lbm* and *milc* utilize ISA diversity, executing using ARM and X86 ISAs for substantial time. Figure 3 depicts that floating-point benchmarks *soplex,* *lbm* and *libquantum* are affined towards X86, as it natively supports floating point operations. *libquantum* and *milc* utilize the SIMD functionality available in X86, being heavily affined to it. High-ILP benchmarks *sjeng* and *bzip2* are affine towards ARM due to lower register pressure it offers. Benchmarks *mcf* and *gobmk* significantly use in-order modes due to their memory-intensive and branch-heavy nature, which favour in-order execution due to pipeline stalls and heavy misprediction penalties in out-of-order mode. Benchmarks *sjeng* and *bzip2* also showcase substantial use of in-order mode.

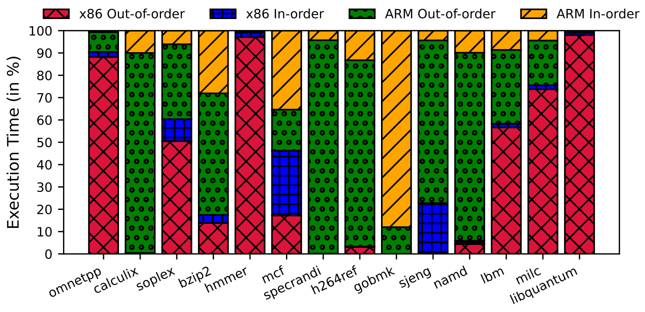


Fig. 3: *Lichen*'s use of different ISAs and execution semantics

B. Scheduler Accuracy

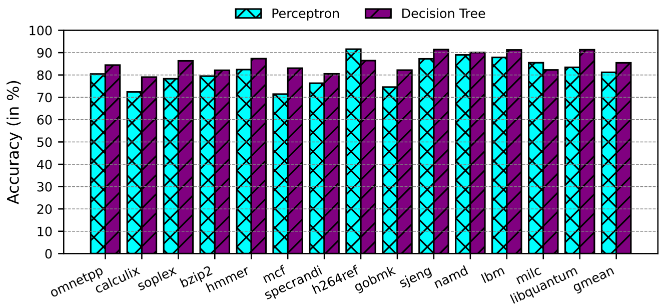


Fig. 4: Affine ISA prediction accuracy for all benchmarks

1) ISA: We define *accuracy* as the percentage of time when the scheduler chooses the most affine ISA. Migration decision accuracies of current state-of-the-art perceptron [11] and CART decision tree (our proposal) are plotted in Figure 4. Apart from *milc* and *h264ref* benchmarks, decision tree showcased better accuracy compared to perceptron. Overall, it achieved an accuracy of 85.44%, performing better than perceptron, which showed 81.19% accuracy. The accuracies are averaged across runtime usage of both ISA models.

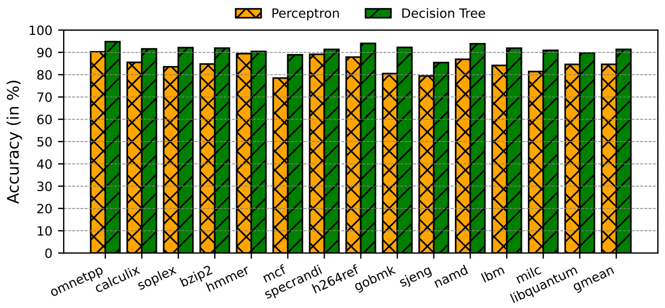


Fig. 5: Prediction accuracy between InO and OoO modes

2) Execution Semantics: For predicting the performance of inactive mode (in-order/out-of-order), we analysed perceptron, linear regression [2] and CART decision tree algorithms. We observed that perceptron and CART decision tree provide more accurate performance predictions than linear regression. Hence, we compare accuracies of perceptron and CART decision tree in Figure 5. We observed that CART decision tree outperformed perceptron in predicting inactive mode's performance across all benchmarks, achieving 91.35% accuracy compared to perceptron's 84.65%. The overall accuracies were observed across the usage of ARM OoO, ARM InO, X86 OoO and X86 InO models at runtime.

C. Performance

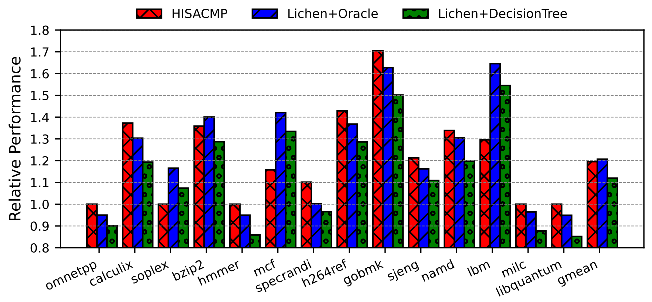


Fig. 6: Performance of *Lichen* w.r.t. X86 OoO core

We analysed performance of *Lichen* relative to X86 OoO core (with configuration as in Table II), shown in Figure 6. *Lichen* works with performance penalty of 5% relative to corresponding heterogeneous-ISA out-of-order only execution. Still, as it utilizes ISA affinity at finer granularities ($1M$ instructions), we see speedups for *soplex, mcf, bzip2* and *lbm* benchmarks. A slowdown for *specrandi* benchmark was observed due to high number of cross-ISA migrations performed for inadequate performance gains. For other benchmarks, we found expected 5% slowdown relative to HISACMP, indicating saturation ISA affinity gains from them. Overall, *Lichen* achieves 0.93% and 20.67% speedup over HISACMP and X86 OoO core with oracular scheduling. With CART-based scheduling, it incurred slowdown of 6.35% over HISACMP due to affine ISA mispredictions, but still achieved 11.97% speedup over X86 OoO core.

D. Energy Savings

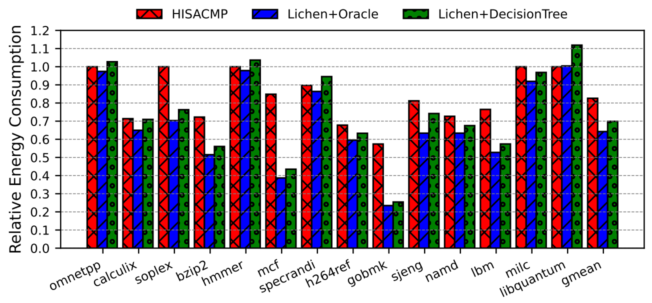


Fig. 7: Energy Consumption of *Lichen* w.r.t. X86 OoO core

Figure 7 highlights the energy consumption of *Lichen* over single X86 OoO core. Significant energy savings for benchmarks with heavy in-order mode utilization (from Figure 3) such as *mcf, gobmk* and *bzip2* were achieved. Benchmarks *soplex* and *lbm* achieve energy savings owing to significant speedup due to ISA affinity exploited at finer phases. We observed 22.21% energy savings and 23.01% energy delay product reduction on average for *Lichen* with oracular schedule, relative to HISACMP. Overall, *Lichen* with CART-based scheduling achieves 30.08% and 15.36% energy savings relative to X86 OoO core and HISACMP, respectively.

E. Performance Degradation Sensitivity Analysis

In Figure 8, we show the analysis across different performance penalties used for *Lichen*, assuming oracular schedule. We observed that for 1% performance penalty relative to *Lichen* operating using out-of-order execution, 5.14% speedup and 18.87% energy savings were achieved over HISACMP. With 10% penalty, *Lichen* is capable of achieving 25.25% energy savings due to 38.26% in-order mode utilization, sacrificing only 4% performance relative to HISACMP.

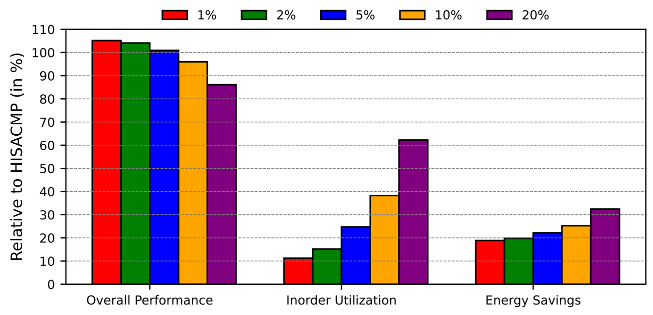


Fig. 8: Performance degradation sensitivity analysis for *Lichen*

F. Hardware Overhead

Lichen requires a 20-bit dynamic instruction counter, used to schedule migration between different ISA cores at $1M$ instruction phases. The CART decision tree used to predict affine ISA requires the listed performance counters in Table I (most already present in modern processors [16]), 54 16-bit memory-mapped programmable registers (for storing model coefficients), a subtractor and 4 comparators. For in-order/out-of-order scheduling within single ISA phase, we reuse the CART decision tree model loaded with different coefficients, hence, re-utilizing associated hardware, but use performance counters' data at finer phases of $1K$ instructions.

VI. Conclusion

This work focused on achieving energy efficiency within heterogeneous-ISA CMPs. We proposed leveraging heterogeneity dimensions of ISA and execution semantics (in-order/out-of-order) at corresponding optimal finer granularities. Our proposal *Lichen*, allows finer-grained switching between different-ISA cores, exploiting ISA affinity. It uses existent out-of-order to in-order morphing feature (for debug) at runtime, reducing the migration overhead and using fine-grained opportunities present. We observed that the interaction between heterogeneity dimensions was synergistic in nature, achieving 15.36% energy savings, 6.35% performance loss and 9.63% EDP reduction over heterogeneous-ISA CMPs. Our proposed CART decision tree based scheduling achieved 85.44% and 91.35% accuracy for predicting the optimal ISA and execution scheme (in-order/out-of-order), respectively.

Acknowledgement

This work was supported in part by Indo Japanese Joint Lab Grant and AI powered adaptive cyber defence framework sponsored by NSCS, Government of India.

References

[1] R. Kumar et al., "Single-isa heterogeneous multi-core architectures: the potential for processor power reduction," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, 2003, pp. 81–92.

[2] A. Lukefahr et al., "Composite cores: Pushing heterogeneity into a core," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 317–328.

[3] S. Srinivasan et al., "On dynamic polymorphing of a superscalar core for improving energy efficiency," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*. Los Alamitos, CA, USA: IEEE Computer Society, oct 2013, pp. 495–498.

[4] B. Jeff, "Big.little system architecture from arm: saving power through heterogeneous multiprocessing and task context migration," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 1143–1146.

[5] P. Smith et al., "Heterogeneous process migration: The tui system," *Software: Practice and Experience*, vol. 28, no. 6, pp. 611–639, 1998.

[6] M. DeVuyst et al., "Execution migration in a heterogeneous-ISA chip multiprocessor," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVII. New York, NY, USA: ACM, 2012, pp. 261–272.

[7] A. Venkat et al., "Harnessing isa diversity: Design of a heterogeneous-isa chip multiprocessor," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 121–132.

[8] N. K. Boran et al., "Fine-grained scheduling in heterogeneous-isa architectures," *IEEE Computer Architecture Letters*, vol. 20, no. 1, pp. 9–12, 2021.

[9] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, Sep. 2006.

[10] E. Blem et al., "Power struggles: Revisiting the risc vs. cisc debate on contemporary arm and x86 architectures," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, 2013, pp. 1–12.

[11] N. K. Boran et al., "Classification based scheduling in heterogeneous isa architectures," in *2020 24th International Symposium on VLSI Design and Test (VDAT)*, 2020, pp. 1–6.

[12] P. Diwan et al., "Mist: Many-isa scheduling technique for heterogeneous-isa architectures," in *2024 37th International Conference on VLSI Design and 2024 23rd International Conference on Embedded Systems (VLSID)*, 2024, pp. 348–353.

[13] A. Venkat et al., "Composite-isa cores: Enabling multi-isa heterogeneity using a single isa," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 42–55.

[14] D. Koufaty et al., "Bias scheduling in heterogeneous multi-core architectures," in *Proceedings of the 5th European Conference on Computer Systems*, ser. EuroSys '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 125–138.

[15] A. Barbalace et al., "Breaking the boundaries in heterogeneous-isa datacenters," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 645–659.

[16] D. Terpstra et al., "Collecting performance data with papi-c," in *Tools for High Performance Computing 2009*, M. S. Müller, M. M. Resch, A. Schulz, and W. E. Nagel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 157–173.

[17] C. Lattner, "Llvm and clang: Next generation compiler technology," ser. The BSD Conference, 2008.

[18] N. Binkert et al., "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1–7, aug 2011.

[19] S. Li et al., "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 469–480.