Performance Modelling and Dynamic Scheduling on Heterogeneous-ISA Multi-Core Architectures

Nirmal Kumar Boran, Dinesh Kumar Yadav, and Rishabh Iyer

Computer Architecture and Dependable Systems Laboratory (CADSL),
IIT Bombay

Abstract. *Heterogeneous-ISA multi-core architectures have emerged as a promising design paradigm given the ever-increasing demands on single threaded performance. Such architectures comprise multiple cores that differ not just in micro-architectural parameters (e.g., fetch width, ROB size) but also in their Instruction Set Architectures (ISAs). These architectures extract previously latent performance gains by executing different phases of the program on the core (and ISA) best suited to it, as opposed to executing the entire program on a single ISA. In such a computing paradigm, maximum performance is only extracted when we ensure that at every point in the program's execution, the program runs on the core best suited to it. In this work, we propose a migration framework that practically and accurately decides when to migrate the program across different cores (and ISAs) to extract maximum performance gains. Under the covers, this framework combines a regression based performance modelling technique with a greedy scheduling algorithm. Our performance modelling technique leverages hardware performance counters prevalent in all major processors today to accurately estimate the performance of the program on different ISAs to within an error of 6%. Putting it together with our greedy scheduler enables our framework to achieve single thread performance speedups of 29.6% with respect to a baseline single ISA heterogeneous architecture.*

Keywords: Heterogeneous ISA multi-core chip · Single thread performance · Regression techniques · Execution migration.

1 Introduction

The history of microprocessor evolution throws light on the fact that the performance of single-threaded programs has always of prime importance. During the last century, researchers were successful in improving single threaded performance until saturation of the frequency scaling. Further performance enhancement required the addition of resources like increasing the size of the register file, ROB and instruction window which led to an increase in the processor's power dissipation. However, designers eventually hit the power wall [10], which led to a marked shift from single-core to multi-core architectures.

*Appears in and was awarded **Best Paper** at the 23rd International Symposium on VLSI Design and Test (VDAT), July 2019*

Multi-core architectures can be classified into three types: 1) Homogeneous, 2) Heterogeneous and 3) Dynamic. Homogeneous multi-core (HoMC) architectures [13] consist of several identical cores (both ISA & micro-architecture) and focus on extracting Thread Level Parallelism (TLP). Heterogeneous multi-core (HeMC) [6] architectures, on the other hand, improve the energy efficiency of the processor by leveraging variation in micro-architectural parameters (not ISA) across cores. They typically contain a combination of several small and a few aggressive cores and allowing them to exploit both TLP and Instruction Level Parallelism (ILP). Finally, to increase the energy efficiency further, Dynamic Core (DC) architectures [5] attempt to modify the micro-architectural parameters of cores on-the-fly, morphing from a single big core into many small cores (or vice-versa) depending upon the program.

Venkat et al. [3] [15] introduced the notion of Heterogeneous ISA multi-core (HeIMC) architectures. Unlike the previous designs, in which all the cores of the processor had the same ISA, different cores in the processor were now built using different ISAs (e.g., x86, ARM, MIPS). The insight behind this work is that ISA diversity can play a vital role in improving both performance and energy efficiency for single threaded programs. The authors show that different phases of the same program can display an affinity for different ISAs. This affinity arises due to a number of factors namely *code density, dynamic instruction count, register pressure* etc. of the program phase. Hence, if the program is migrated across cores with different ISAs, such that each phase is executed on the ISA it is most affine to, previously latent gains in single threaded performance can be achieved. The authors also describe compiler modifications [3] that make this migration practical.

However, Venkat et al. [3] [15] focus only on the prospective gains of HeIMC architectures and NOT on how to achieve them. To extract maximum performance gain, HeIMC architectures require that the program be migrated correctly, ensuring that each phase of the program is executed on the core (and ISA) it is most suited to. In fact, as we show in Section 2, incorrect decisions can lead to deterioration in single-threaded performance. Venkat et al. do not address this issue at all, assuming the existence of a prediction oracle which makes perfect decisions. This is naturally infeasible in practice.

To address this gap and close the loop on HeIMC architectures, we propose a framework that practically and accurately decides when to migrates the program across different cores (and ISAs) to extract maximum performance gain. Under the covers, this system combines a regression-based performance modelling technique with a greedy scheduling algorithm. The performance modelling technique practically estimates the performance of the current phase of the program by leveraging micro-architectural parameters obtained from hardware performance counters present in every major processor today [12]. By making accurate predictions and scheduling decisions, our system achieves single thread performance speedups of 29.6% with respect to a baseline single ISA HeMC architecture.

The remainder of the paper is organized as follows: Section 2 motivates the need for a novel performance modelling technique for HeIMC architectures. Section 3 explains the proposed method of performance modelling with the required details. Section 4 explains our scheduling mechanism and the trade-offs involved. Section 5 presents the results of our evaluation and finally, Section 6 concludes, with certain directions for future research.

2 Motivation and State-of-the-Art

In this section, we show the proposed performance benefits of HeIMC architectures and then describe why we need a novel performance modelling technique.

To demonstrate the ISA affinity exhibited by programs and the proposed benefits in single threaded performance, we divide the *'astar'* benchmark from SPEC CPU2006 [4] into 15 phases and execute it on ARM (RISC) and x86 (CISC) cores. The configurations used for both the cores is mentioned in Table-1. Apart from the number of General Purpose Registers (GPRs) which is an intrinsic property of the ISA, all micro-architecture parameters are identical across the two cores.

Figure 1 shows the execution time of each phase on both cores. The results show that by only changing the ISA, there is sufficient variability across the execution time of each phase. Further, neither core is dominant throughout, with x86 performing considerably better on phases 1, 7-14 and ARM performing considerably better on phases 2-6 and 15. Clearly, running the entire program solely on any one of these cores does not lead to optimal performance.

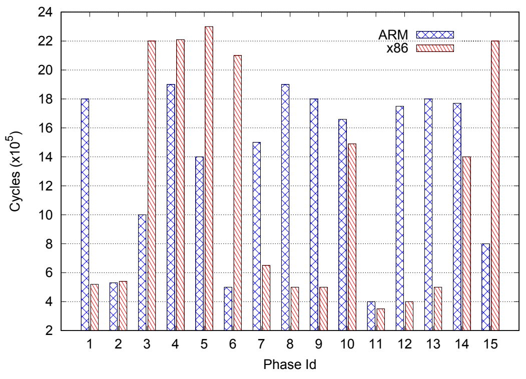

Fig. 1. Execution time of different phases of benchmark astar

Figure 2 shows the latent performance benefits that can be extracted by HeIMC architectures (accurate-migrations). When running each phase of the program on the core (and ISA) it is most suited to, the performance of the program can increase by up to 39% (This is an upper bound because it ignores migration overhead). On the other hand, if the performance modelling is flawed, and mispredicts where to execute each phase of the program, it can lead to unacceptable performance deterioration of up to 26.4%.

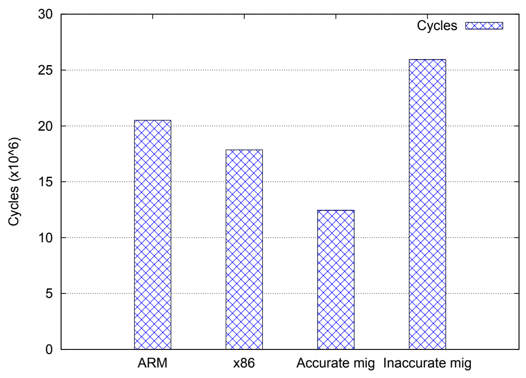

Fig. 2. Execution time when the program is run on ARM, x86 and execution time when each phase is run on the core most suited to it (”Accurate mig”) and least suited to it (”Inaccurate mig”)

Clearly, maximum performance benefits are only achieved when the program is migrated across the cores at the right time. Naturally, this requires a prediction and scheduling mechanism, which dictates when this migration is supposed to take place. Given this need for an accurate cross-ISA performance modelling technique, we now explain why prior work on the subject is inadequate.

The first work on performance modelling in multi-core architectures was by Kumar et al. [6] who used a sampling-based technique to predict migration of programs on different cores. In this approach, they would run a small section of the code on all available cores and run the remainder of the code on the core that performed best for the small section. This leads to poor resource utilization and does not scale with increasing core counts. Hence, researchers made attempts to use analytical models to model CPI of various programs on single ISA heterogeneous architectures. Craeynest et al. [14] introduced a regression-based performance impact estimator that used ILP and MLP (Memory level parallelism) as parameters to predict migration. This was taken further by Lukefahr et al. [9] and Pricopi et al. [11].

While techniques work well for HeMc architectures, they do not make accurate predictions for HeIMC architectures. This is because they are simply not designed to take into account factors that determine performance variation across ISAs, such as code density, register pressure and instruction mix since these factors are identical across cores of the same ISA. Boran et al. [2] demonstrate this by attempting to predict execution time across ISAs using a regression model which takes into account only micro-architectural parameters. *However, their approach leads to unacceptably high prediction errors which motivated us to build a better model that takes into account inter-ISA heterogeneity.* Thus, in this work, in addition to incorporating features that take into account inter-ISA heterogeneity, we also change our machine learning model, using linear regression which is more accurate than the General regression neural network(GRNN) and the model used in [2].

3 Performance Modelling

In this section, we describe, in detail, our technique for cross-ISA performance modelling. The goal of this technique is to utilize micro-architectural and ISA-specific parameters obtained from the core that the program is currently running on, to predict what the execution time of the program would be if run on a different core (and ISA). This prediction will then be fed to the scheduler which dictates when the migration should take place. From here-on, we will use the term *source ISA/core* to refer to the ISA/core that the program is currently running on, and the term *target ISA/core* to refer to the ISA/cores that the program can migrate to.

The work most closely related to ours [2] proposed a two-phased approach for performance modelling wherein they used the micro-architectural parameters from the source ISA to predict the same parameters on the target ISA. Then, using these predicted target ISA parameters, they predicted the execution time on the target ISA. Their approach has two main shortcomings:

– The decoupling of execution time prediction into two independent phases compounds prediction errors. In their work, Boran et al. [2] show that execution time predictions can be off by up to 54%.
– They do not take into account any parameters that characterize the inter-ISA heterogeneity, such as code density, register pressure or instruction mix, leading to high prediction errors.

We improve upon their model in the following ways:

– We eliminate the artificially enforced decoupling in the prediction of execution time. Consequently, we directly predict the execution time on the target ISAs using the parameters of the source ISA.
– We specifically introduce parameters that quantify the inter-ISA heterogeneity, specifically the instruction mix, dynamic instruction count and parallelism (ILP, MLP)
– We replace their regression model with a linear regression model, which performs better for single-level predictions.

3.1 Extracting Relevant Parameters

We now describe the list of parameters we use to predict the execution time on the target ISA and how we extract them. In this work, we use a total of thirteen parameters to predict execution time (number of execution cycles). The parameters include branch miss-predictions, L1-I-cache misses, L1-D-cache misses, L2 cache misses, Reorder Buffer full events, Instruction Queue full events, Store Queue full events, ILP, MLP, MSHR (Miss Status Handling Register) full events, instruction mix (Number of load instructions, Number of floating point instructions) and the dynamic instruction count.

L1-I-cache, L1-D-cache, and L2 cache misses capture the effects of the cache hierarchy on the execution time. Information regarding data-dependency induced stalls has been extracted by the occurrences of the ROB, Instruction Queue and Store Queue being full. ILP and MLP have been considered for determining the available parallelism given the specific ISA and core that the program is running on. Instruction mix and dynamic instruction count are chosen to quantify ISA specificity and finally, the behaviour of the branch predictor is captured using the branch misprediction parameter.

A common feature of all of these parameters is that their extraction is simple and practical. All parameters except ILP and MLP can be directly obtained from hardware performance counters prevalent in every major processor variant today. To calculate ILP and MLP we rely on schemes proposed by previous work [9]. ILP is estimated by using a hardware counter which maintains a running sum of the instructions in the issue stage whose execution requires the data from the other instructions currently under execution. This counter captures all the instructions which are stalled due to dependencies, thus providing us with an inverse measure. For MLP, we leverage a hardware counter that maintains a running sum of the number of MSHR entries at every cache miss. This gives us an estimate of MLP because during a cache miss, all misses currently being handled in parallel will have an entry in the MSHR. We take individual averages of the running sums maintained by both counters ($running\_sum/total\_instructions$) to estimate the ILP and MLP respectively.

3.2 Linear Regression Model

Given those parameters, we now describe our linear regression based performance model. In our evaluation Section 5, we show that it outperforms the GRNN model proposed in previous work. Given a source ISA in execution (ISA_A) and a target ISA (ISA_B), our linear regression model for estimation of the number of cycles is given by:

$$\begin{aligned} Cycle_B = & K + a_1.(L1DcacheMiss_A) + a_2.(L1IcacheMiss_A) \\ & + a_3(L2cacheMiss_A) + a_4.(IQFullEvents_A) \\ & + a_5.(SQFullEvents_A) + a_6.(ROBFullEvents_A) \\ & + a_7.(BranchMissPrediction_A) + a_8.(MLP_A) \\ & + a_9.(MSHRFullEvents_A) + a_{10}.(ILP_A) \\ & + a_{11}.(LoadCount_A) + a_{12}.(FloatInstruction_A) \\ & + a_{13}.(DynamicInstructionCount_A) \end{aligned} \tag{1}$$

where a_1 to a_{13} are regression coefficients, $Cycle_B$ is number of cycles for ISA_B and K is a constant

In this work, we consider three ISAs namely x86, ARM and Alpha. We build a total of 6 regression models: for each of the three ISAs, we predict the performance of the other 2. The idea here is that these models are built offline and incorporated into the processor with the coefficients of the model stored in special registers. Then, when programs are executed, these models continuously predict the performance on the other two ISAs and pass these predictions to the scheduler.

4 Scheduling

In this section, we describe the design of our scheduler. We first provide necessary background on how a cross-ISA compiled program looks like, before detailing our scheduling algorithm and the tradeoffs involved.

De Vuyst et al [3] describes compiler modifications that allow a program to efficiently migrate across cores. The generated binary possesses a number of *equivalence points* at which the memory state of the binary is consistent across different ISAs. This identical memory state allows for low-overhead migration of the program at one of these equivalence points. Note, migrating at any other point is expensive and requires dynamic binary translation on the target ISA till the next equivalence point is reached. Given the overhead of migration, these equivalence points are typically kept around 100M instructions apart. We call each such division of 100M instructions a *phase*. Figure 3 further illustrates equivalence points and phases.

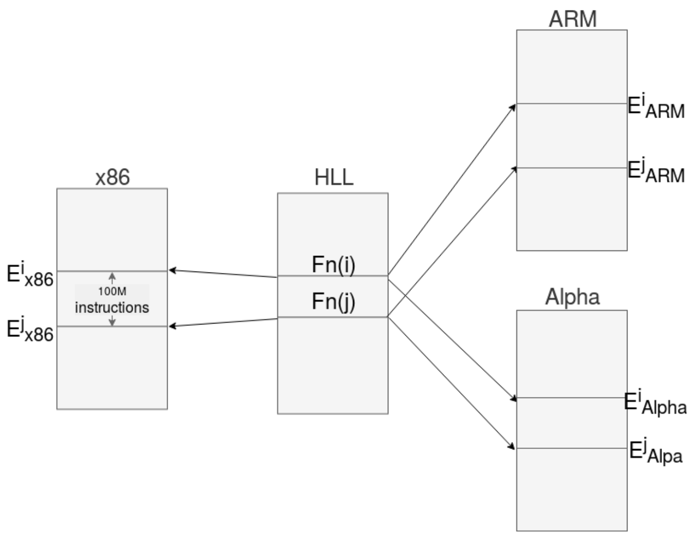

Fig. 3. Equivalence points are chosen at function call sites in the compiler IR. They are typically around 100M instructions apart

This division of the program into phases poses a significant challenge for the scheduler. Ideally, we would like to use the first few (10-20M) instructions of the phase to predict which ISA it is suited to, much like [6]. Unfortunately, this is infeasible since migration can only occur at the equivalence points. Consequently, all scheduling decisions must be made before the phase begins to execute.

We extract the parameters mentioned in Section 3.1 for only the final 10M instructions of a phase and feed it into our regression-based performance model which predicts the performance on the target ISAs. Using the final 10M instructions leads to accurate predictions because it is a small yet significant number of instructions closest to the next phase and variations will be few, but meaningful predictions can be made.

Once we have the predictions for execution time for all the target ISAs, we simply employ greedy scheduling, i.e., at the next equivalence point, the scheduler migrates the program to the ISA with least predicted execution time. Naturally, for all the target ISAs we include the migration overhead into the estimation.

5 Evaluation

In this section, we describe the results from our evaluation, which answer 2 primary questions: 1) Does the regression-based performance model predict performance accurately across ISAs and 2) Does the scheduling algorithm correctly migrate the program to deliver overall speedups in single-threaded performance?

5.1 Methodology

As mentioned previously, we consider 3 ISAs in this work - x86, ARM and Alpha. Since the focus is on inter-ISA heterogeneity, we keep the configuration for all 3 ISAs identical (Number of GPRs is a property of the ISA). The configuration for all the cores is shown in Table 1. The clock speed for all three cores is 2GHz. We use the gem5 simulator [1] to simulate performance and SPEC CPU2006 [4] benchmark suite as our target programs.

Table 1. Core configurations

Design Parameter	ARM	Alpha	x86
Architectural Registers	32 GPR	64GPR	16 GPR
Cache line size(bytes)	64	64	64
LSQ size (bytes)	32	32	32
Fetch width	4	4	4
Instruction Queue entries	64	64	64
ROB entries	192	192	192
DCache,ICache size	32KB	32KB	32KB
L2 Cache size	256KB	256KB	256KB

We use the results from [15] to determine the migration overhead. As a simple over-approximation, we consider the maximum migration overhead they report for the SPEC CPU2006 benchmark suite.

5.2 Results

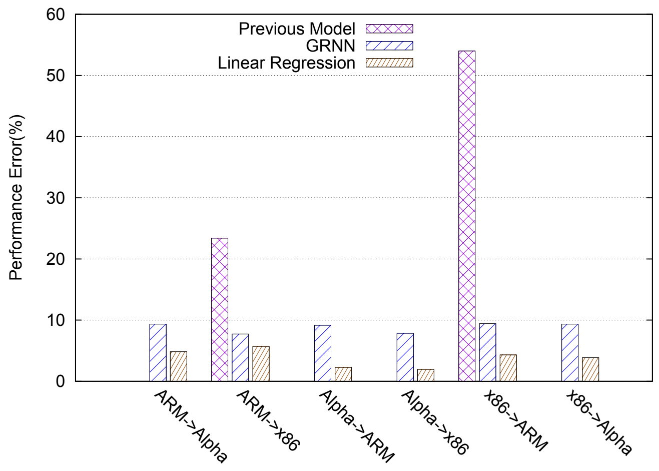

Fig. 4. Root mean squared error of performance modelling of Previous Model [2], GRNN and Linear Regression. ISA_1 —> ISA_2 denotes estimation of performance for the core with ISA_2 while running the program on the core with ISA_1

Accuracy of performance modelling technique To evaluate the prediction accuracy of our regression-based performance model, we compare the Root Mean Squared (RMS) prediction error across several schemes for each of the 6 models that we build. We compare our linear regression-based model against two schemes - 1) A GRNN model from prior work [2](Previous Model) and 2) A GRNN model using all 13 parameters - including the ones accounting for inter-ISA heterogeneity (GRNN). For (1) we only use values wherever provided. Comparing against (1) illustrates the joint effect of both using linear regression and including parameters that capture the inter-ISA heterogeneity. Comparing against (2) enables us to quantify the improvements due to using linear regression since the parameter set is identical for both schemes.

Figure 4 compares the RMS prediction error for all three schemes across predictions for all phases of all the benchmarks in the SPEC CPU2006 benchmark suite. The phase length taken for performance modelling is 10M.

Two clear trends emerge:

– Irrespective of the model, linear regression always outperforms the GRNN. The linear regression based model leads to errors ranging from 1.7% to 5.7%, while the GRNN leads to errors ranging from 7.5% to 9.4%.

– Both schemes outperform prior work considerably. This is due to two factors - 1) The introduction of additional parameters that capture inter-ISA heterogeneity such as the dynamic instruction count and the instruction mix and 2) Replacing the two-phase prediction which compounds prediction errors with a simple one-phase prediction scheme.

We also compare the standard deviation of the RMS error for both the GRNN and Linear regression-based models. Linear regression has a 26% lower standard deviation than the GRNN model making it lesser uncertain and more reliable.

Dynamic scheduling To evaluate whether our scheduling algorithm migrates the program correctly, we compare the speedup obtained in the HeIMC architecture as opposed to the base case of a single ISA architecture. Figure 5 illustrates the result. Our regression based scheduler shows a 29.6% increase in mean performance when compared to the x86 baseline and a 19.5% increase in performance on the best performing architecture. Additionally, the regression-based predictor is only 12.2% off the oracle. Please note that the *oracle is a hypothetical case in which each phase runs on the core it is most suited to. Hence the Oracle case represents the maximum possible speedup*. The phase length for dynamic scheduling is taken as 100M. The models were trained using data from all of the SPEC CPU2006 benchmarks (70% training data, 30% testing data).

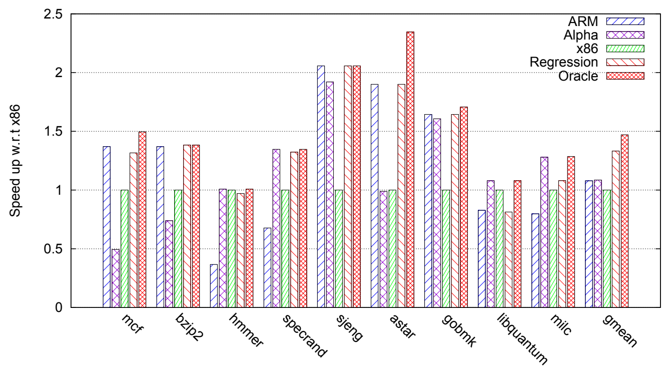

Fig. 5. Average speedup of different benchmarks when entire program is scheduled on ARM, Alpha or x86 and compared with HeIMC architecture with regression and oracle based scheduling

In the previous experiment, the model was only evaluated on programs it had already been exposed to. When deployed, however, our framework must run accurately even for programs it has not been trained on. To evaluate this generality and resilience of our migration framework, we trained the model using only a subset of the SPEC CPU2006 benchmark suite and tested it on others. Then we measured how accurately the scheduler migrates the program and the speedups achieved on these unseen programs. Figure 6 illustrates the results.

We see that on average, our system works even for programs it has not seen before, producing an average speedup of 24.4% over x86 and only 16% less than the oracle. Additionally, our system migrates the program to the core it is most suited 82.94% of the time. Given that the SPEC CPU2006 benchmark suite consists of programs that are inherently very different from one another, these results show that our model is resilient and can be deployed without having to be re-trained frequently.

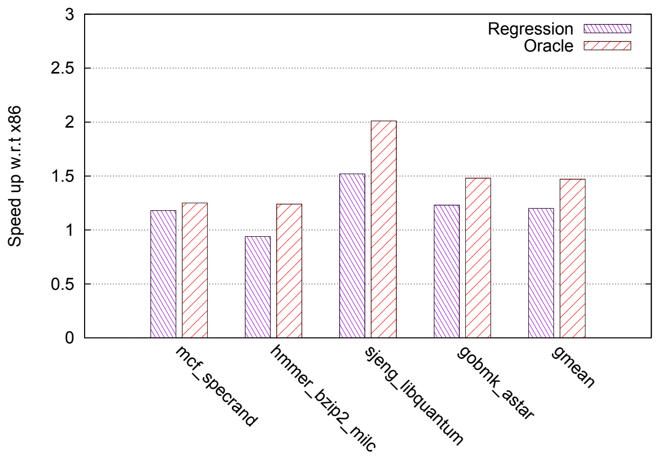

Fig. 6. Speedup w.r.t x86 of SPEC CPU2006 benchmarks when our system faces programs it has not been trained on. Here the x-axis labels represents benchmarks used only for testing while remaining benchmarks were used for training.

Energy efficiency We also ran an experiment to see whether the HeIMC architecture would increase the energy efficiency of the system. For this, we used the McPAT [8] simulator to compare the energy consumed. We have calculated the energy consumed while the whole benchmark is run on ARM, alpha, x86 and these energies are compared with energy consumed for the HeIMC architecture which uses our performance model and scheduling algorithm. In our experiments, we find that there is very little change in energy consumption. This is likely because all three cores in our system had an identical configuration the ISA which displays the maximum performance also consumes the maximum power. Additionally, our scheduler is only designed for optimizing performance. In future work, we plan to incorporate the energy consumption also into the decision making process.

Hardware Overhead The proposed modelling method and scheduling algorithm have minimal hardware overhead. Since we are doing the training offline in software, we only require 14 registers to store the weights for performance

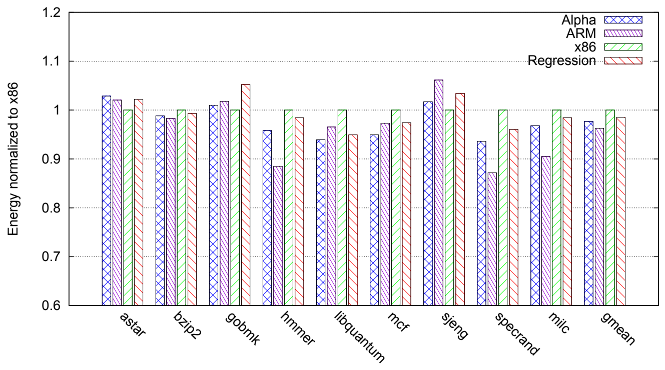

Fig. 7. Energy comparison

modelling using the linear regression method. The compute scheduler requires thirty-nine 8-bit multipliers for applying weights to the input parameters along with three 32-bit adders. Please note that a GRNN based model has more hardware overhead when compared to a linear regression based one.

6 Conclusion and Future Work

In this work, we have shown that it is feasible to extract performance benefits from HeIMC architectures by utilizing lightweight and practical techniques for performance modelling and dynamic scheduling. Our framework combines a regression-based performance model with a greedy scheduling algorithm. Our performance model estimates the performance of a program across ISAs within 6% error-limit and our scheduler migrates the program to the core it is most suited to 83% of the time for program types it has never seen before. Together, these techniques achieve an average increase of 29.6% in single-threaded performance on the SPEC CPU2006 benchmark suite.

We see several interesting avenues for future work. First and foremost, we plan to introduce energy efficiency into the scheduling decisions which will allow HeIMC architectures to save considerable energy as opposed to HeMC ones. For instance, introducing a specifically energy-efficient ISA such as thumb and only migrating when the ILP of the program is very low can lead to an energy efficient system with minimal degradation in performance. In such energy efficient systems, we would also like to focus on the idea proposed by Lee et al. [7] which shows that reducing the instructions in the ARM ISA has shown to greatly reduce the logical complexity of hardware, such ISA diversity enables further energy efficiency.

Another avenue is modifying the scheduling algorithm for when multiple programs exist and want to migrate at different times. We expect this co-located scenario to be quite challenging. Additionally, apart from heterogeneous ISA scheduling, performance modelling can have many other applications which may be helpful for other researchers. For instance, accurate performance modelling can provide information w.r.t to which workloads must be co-located together to avoid destructive interference in performance.

References

1. Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., et al.: The gem5 simulator. ACM SIGARCH Computer Architecture News (2011)
2. Boran, N.K., Meghwal, R.P., Sharma, K., Kumar, B., Singh, V.: Performance modelling of heterogeneous ISA multicore architectures. In: East-West Design & Test Symposium (EWDTS), 2016 IEEE. pp. 1–4. IEEE (2016)
3. DeVuyst, M., Venkat, A., Tullsen, D.M.: Execution migration in a heterogeneous-isa chip multiprocessor **40**(1), 261–272 (2012)
4. Henning, J.L.: SPEC CPU2006 benchmark descriptions. SIGARCH Comput. Archit. News (Sep 2006). https://doi.org/10.1145/1186736.1186737
5. Ipek, E., Kirman, M., Kirman, N., Martinez, J.F.: Core fusion: accommodating software diversity in chip multiprocessors. ACM SIGARCH Computer Architecture News **35**(2), 186–197 (2007)
6. Kumar, R., Farkas, K.I., Jouppi, N.P., Ranganathan, P., Tullsen, D.M.: Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In: Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on. pp. 81–92. IEEE (2003)
7. Lee, W., Sunwoo, D., Emmons, C.D., Gerstlauer, A., John, L.: Exploring opportunities for heterogeneous-isa core architectures in high-performance mobile socs. Tech. rep., Technical Report UT-CERC-17-01, The University of Texas At Austin (2017)
8. Li, S., Ahn, J.H., Strong, R.D., Brockman, J.B., Tullsen, D.M., Jouppi, N.P.: Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In: Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on. pp. 469–480. IEEE (2009)
9. Lukefahr, A., Padmanabha, S., Das, R., Sleiman, F.M., Dreslinski, R., Wenisch, T.F., Mahlke, S.: Composite cores: Pushing heterogeneity into a core. MICRO-45, IEEE Computer Society, Washington, DC, USA (2012). https://doi.org/10.1109/MICRO.2012.37
10. Patterson, D.: The trouble with multicore. Spectrum, IEEE **47**, 28 – 32, 53 (08 2010). https://doi.org/10.1109/MSPEC.2010.5491011
11. Pricopi, M., Muthukaruppan, T.S., Venkataramani, V., Mitra, T., Vishin, S.: Power-performance modeling on asymmetric multi-cores. In: Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2013 International Conference on. pp. 1–10. IEEE (2013)
12. Sprunt, B.: The basics of performance-monitoring hardware. IEEE Micro **22**(4), 64–71 (2002)
13. Taylor, M.B., Lee, W., Miller, J.E., Wentzlaff, D., Bratt, I., Greenwald, B., Hoffmann, H., Johnson, P., Kim, J.S., Psota, J., Saraf, A., Shnidman, N., Strumpen, V., Frank, M.I., Amarasinghe, S.P., Agarwal, A.: Evaluation of the raw microprocessor: An exposed-wire-delay architecture for ILP and streams. In: 31st International Symposium on Computer Architecture (ISCA 2004), 19-23 June 2004, Munich, Germany (2004)

14. Van Craeynest, K., Jaleel, A., Eeckhout, L., Narvaez, P., Emer, J.: Scheduling heterogeneous multi-cores through performance impact estimation (pie) **40**(3), 213–224 (2012)
15. Venkat, A., Tullsen, D.M.: Harnessing ISA diversity: Design of a heterogeneous-isa chip multiprocessor. In: Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on. pp. 121–132. IEEE (2014)