A Cross-Core Performance Model for Heterogeneous Many-Core Architectures

Rui Pinheiro, Nuno Roma, and Pedro Tomás *

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa

Abstract. An accurate performance predictor to identify the most suitable core-architecture to execute each thread/workload in a heterogeneous many-core structure is proposed. The devised predictor is based on a linear regression model that considers several different parameters of the many-core processor architectures, including the cache size, issue-width, re-order buffer size, load/store queues size, etc.. The devised predictor is easily integrated in most system schedulers, providing the ability to periodically determine whether a certain thread is running in the most efficient core-architecture. The obtained experimental results show that the devised model is able to identify the correct core-architecture in a large majority of the cases, leading to average performance differences as low as 7% when compared with an oracle scheduling solution.

1 Introduction

Advances in processor design have recently pushed for the development of heterogeneous processors, in order to tackle the power and memory walls. In particular, by relying on appropriate and different core architectures, it is possible to efficiently leverage Memory-Level Parallelism (MLP) and Instruction-Level Parallelism (ILP) [4, 5] such as to minimize power and energy consumption with a reduced performance loss. However, exploiting heterogeneity often requires the development of efficient scheduling mechanisms, in order to anticipate the performance gains due to the migration of an application from one core to another, or to the morphing of a given core, which can be achieved by means of clock/power gating or by relying on reconfigurable technologies.

In particular, driven by the introduction of the ARM big.LITTLE heterogeneous processor [1] (although not exclusively), intensive research has recently been put forth in the exploitation of heterogeneous processor systems composed of multiple in-order and out-of-order cores, by developing methodologies to manage the allocation of tasks to cores. For example, Patsilaras et al. [6] described a Chip Multi-Processor (CMP) with two core architectures, where one is tuned for exploiting MLP and the other for ILP. To manage the application allocation, the authors make use of on-line sampling techniques performed on both core architectures, as well as a heuristic algorithm based on the detection of clustered Last-Level Cache (LLC) misses.

* This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT), under project UID/CEC/50021/2013.

A similar methodology was employed by Kumar et al. [5], although relying on a two-stage approach. During the first stage (*sampling*), applications are permuted over all core architectures in order to obtain a set of per-core statistics, retrieved from Hardware Counters (HCs). In a second stage, the gathered statistics are used to predict which core is better suited to each application. Although the authors consider the possibility of using more than two different core types,they still require periodic on-line sampling of all application-core permutations, a slow process during which the system is running sub-optimally.

Various attempts have been made to avoid this slow sampling process. Shelepov et al. [10] described a computational system where the scheduler is supplied with application signatures, obtained through off-line analysis. However, this requires all applications to be re-compiled specifically for such a system, which is not always feasible. Saez et al. [8] use the count of LLC misses to grossly estimate the speedup factor without having to sample all application-core permutations. However, other parameters (e.g., core width), which cause many different interactions affecting application performance, cannot be properly described by just analyzing cache miss rates. Craeynest et al. [12] took a similar approach, by deriving a HC based simplified model to estimate performance differences between *small* in-order and *big* out-of-order cores. Based on this model, the authors developed a system scheduler to regularly estimate the performance of running applications on the alternate core and decide whether a core switch is worthwhile. However, this approach is constrained to two core types and only takes into account changes in Re-order Buffer size and issue width. Pricopi et al. [7] took inspiration from these two approaches and developed a prediction model specifically for the ARM big.LITTLE processor able to take into account more architectural parameters, using a mixture of HC statistics and offline analysis. However, in addition to the requirement of an offline analysis, it still only considers two possible core variations at once.

In accordance, this paper addresses the identified issues and limitations, by proposing a new low-overhead architecture-independent method to derive adaptable models that estimate the attainable performance over a large range of varying micro-architectural parameters, and that can be used both at a hardware-level, or as a software module integrated into the OS scheduler. The considered approach uses a Linear Regression Model based on several commonly available HCs. In order to fully illustrate the proposed method, an example model based on out-of-order cores with different cache hierarchies, Re-order Buffer (ROB), Load Queue (LQ) and Store Queue (SQ) sizes was derived. The resulting model was then cross-validated with a set of 81 different core types using the PARSEC benchmark suite [2] and the micro-architectural simulator Sniper [3]. The proposed model is shown to be highly accurate for all considered core types.

2 Performance Modeling

Most current processors are equipped with multiple HCs that can be configured to measure various runtime statistics (e.g., cycle counts, retired instruc-

tions, cache misses), which can then be used to infer the application performance [8, 10, 12]. Such information allows for the development of intelligent software and/or hardware modules, capable of scheduling running applications to the most appropriate core architectures and/or adapting the characteristics of each core according to the scheduled application's computational requirements. Hence, it is herein considered that, during program execution, a set of HCs are measured at a *source core*, in order to characterize the current application phase. Based on such information, the devised system is able to predict the attainable performance on a *target core*, in order to support a decision on whether to move the thread to a different core or to apply any core morphing techniques. Like previous cross-core performance models, the proposed methodology assumes that any cross-thread interaction effects (i.e., cache sharing or synchronization) are core-independent, such that they manifest on all target cores similarly to the source core, reducing the modeling difficulty considerably.

Hence, this manuscript leverages the correlation between HC statistics and application performance in order to derive cross-core performance models. To attain such a goal, a Linear Regression Model (LRM) is adopted, which allows accurate performance predictions across hypothetical changes on several micro-architectural parameters, given an initial representative training set. Moreover, considering that the *retired instruction count* is an easy-to-measure and core-independent runtime statistic, it is used to normalize all runtime statistics into an application-independent scale that is easier to work with. As a result, Cycles Per Instruction (CPI) becomes an obvious choice for performance metric and is therefore used as LRM dependent variable, since it can also be easily measured and is already normalized by the instruction count.

Furthermore, in order to improve the quality of the model, a logarithm link function is used. This is a natural approach, not only because the CPI metric is always positive, but also because experimental evaluation has shown that the original model's residual distribution is log-normal. Accordingly, since normally-distributed residuals are preferable in order to ensure that the least-squares estimator matches the maximum-likelihood estimator (as the latter has better statistical properties [9]), the proposed model is built based on the regression

$$\log(\hat{\mathrm{CPI}}_{tgt}) = \beta_0 + \beta_1 \log(CPI_{src}) + \sum_{i=1}^{N} \beta_{i+1} x_i \ , \tag{1}$$

where $\hat{\mathrm{CPI}}_{tgt}$ represents the estimated CPI at the target core, β_i are model coefficients (in particular, β_0 represents the constant or intercept term), $\log(CPI_{src})$ represents the logarithm of the CPI measured in the source core, and $x_1, \cdots, x_N$ represent the set of N regression terms obtained by coupling the statistics gathered by using HCs with the micro-architectural parameter variations. Each regression term x_i is herein considered to express the product of the variation Δp of a given micro-architectural parameter p between the source (p_{src}) and target (p_{tgt}) cores ($\Delta p = p_{tgt} - p_{src}$), with a runtime statistic S_i, normalized by the retired instruction count I, $x_i = \frac{S_i}{I} \Delta p_i$.

Concerning the selection of regression terms, it is important to note that, although the model accuracy increases with the introduction of more regression

Table 1: Description of the considered set of core parameters, together with their dominant effects concerning the attained performance.

Architecture Parameter	Description	Dominant Effects
$L\{1,2,3\}size$	Total size of caches L1, L2 and L3	Impacts the cache hit rate, significantly impacting the memory access latency.
$LQsize$	Load Queue size	When full, generates structural hazards for new load instructions, causing pipeline stalls at the issue stage.
$SQsize$	Store Queue size	When full, generates structural hazards for new store instructions, causing pipeline stalls at the issue stage.
ROB	Re-order Buffer size	When full, generates structural hazards, leading to stalls at instruction issue.
W	Core issue, dispatch and commit Width	Affects the peak instruction throughput at issue, dispatch and commit stages.

terms, this leads to an increase in model complexity and possibly to over-fitting, reducing its effectiveness with unobserved applications. It is therefore important to pick the minimum number of terms that allow an effective modeling of the dominant effects of all architectural parameters of interest. This procedure can be automated using statistical methods for automatic regressor choice, for example Lasso [11] or Elastic Net [13], which provide the means for an automatic search over the regressor space in order to retrieve the most adequate architectural parameters and runtime statistics.

In order to obtain a generic model that covers a representative set of parameters, and simultaneously shows the flexibility of using a LRM to predict performance differences between different cores, a highly heterogeneous many-core CMP is herein considered as an example proof of concept, including many different out-of-order architectures of varying cache sizes (although limited to equal sized L1 instruction and data caches), issue widths, ROB sizes, as well as different load and store queue sizes (modeled as two separate queues). The set of considered parameters and their dominant effects are summarized in Table 1.

Accordingly, it is necessary to choose runtime application-dependent statistics that are most correlated with the dominant effects of each micro-architectural parameter being varied. In order to choose between different runtime statistics that explain similar effects, their effect on the model prediction quality was evaluated by relying on the Sniper Multi-Core Simulator [3] to provide accurate simulations of several x86 micro-architectures. To analyze the results, the t-statistic (i.e., significance) was used, as well as the coefficient of determination R^2 of the resulting model. Nonetheless, the ease of measuring the various possible statistics in real hardware was also taken into account. The result of this analysis is presented in Table 2. As can be seen, all the chosen statistics correlate with at least one of the dominant effects mentioned in Table 1. To better illustrate the considered statistics, the maximum t-statistic value for a corresponding 3-coefficient model ($N = 1$) is also presented, measured under the same experimental methodology as the results that will be presented in Section 3. For comparison purposes, some statistics that were left out from the proposed

Table 2: Runtime statistics subset (most-relevant) for each processor parameter, the corresponding effect, and the maximum observed absolute t-Statistic value. Boldfaced t-Statistic values represent the variables used in the final model.

Architecture parameter	Hardware counter	Correlation	t-Stat.
Core Width (W)	I: Instruction Count	Peak performance	**11.36**
	$Hdep$: Data Hazards at dispatch	Instruction interdependency	**10.12**
ROB Size (ROB)	$Hrob$: Hazards due to full ROB	ROB occupancy	**6.62**
	$Hdep$: Data Hazards at dispatch	Instruction interdependency	**6.75**
Load Queue Size ($LQsize$)	LD: Load Uops Count	Load queue usage rate	**26.81**
	Hlq: Hazards due to full LQ	Load queue usage rate	18.11
Store Queue Size ($SQsize$)	ST: Store Uops Count	Store queue usage rate	**19.51**
	Hsq: Hazards due to full SQ	Store queue usage rate	16.71
Cache Size ($L\{1,2,3\}size$)	$L\{1,2,3\}miss$: Cache miss Count	Memory access latency	**7.80**
	LD: Load Uops Count	Cache access rate	3.81
	ST: Store Uops Count	Cache access rate	4.12

model are also shown. As can be seen, their corresponding t-statistic values are considerably lower than that of the selected HC based statistics (in boldface).

Conversely, it was observed that some of the variables present a non-linear correlation with the corresponding architecture parameter. To model such cases, a second-order Taylor series expansion was used, in order to allow an effective modeling of the architectural effect, while preserving model simplicity. Hence, the following simple (but still highly representative) 14-term LRM was obtained:

$$\begin{aligned}\log(\hat{CPI}_{tgt}) = &\beta_0 \ + \beta_1 \log(CPI_{src}) + \beta_2 \ L1miss_n \ \Delta L1size + \\ &\beta_3 \ L2miss_n \ \Delta L2size + \beta_4 \ L3miss_n \ \Delta L3size + \\ &LD_n \ (\beta_5 \ \Delta LQsize + \beta_6 \ \Delta LQsize^2) + \\ &ST_n \ (\beta_7 \ \Delta SQsize + \beta_8 \ \Delta SQsize^2) + \\ &Hdep_n(\beta_9 \ \Delta ROB + \beta_{10} \ \Delta W) + \\ &\beta_{11} \ Hrob_n \ \Delta ROB + \beta_{12} \ \Delta W + \beta_{13} \ \Delta W^2 \ .\end{aligned} \tag{2}$$

Finally, the coefficients β_i can now be calculated by training the above LRM with observations obtained by running a representative set of benchmarks on all core variations of interest, resulting in a linear number of models, one per core.

3 Experimental Results

In order to properly evaluate the developed cross-core performance model, the Sniper Multi-Core Simulator [3] was used, to provide accurate simulations of several x86 micro-architectures. Hence, a vast set of core variations was described in this simulation framework, by varying several micro-architecture and cache organization parameters, as depicted in Tables 3 and 4. In accordance, a total of 81 different core variations were simulated, allowing an effective modeling of the interaction between several parameters.

Table 3: Considered cache hierarchy variations (associativity, set count, and total size in KB); The block size was set fixed and equal to 64 Bytes.

	L1-D; L1-I			L2			L3		
Name	**Assoc.**	**# Sets**	**Total**	**Assoc.**	**# Sets**	**Total**	**Assoc.**	**# Sets**	**Total**
Small	2	8	1	4	32	8	8	1024	512
Medium	2	16	2	8	64	32	16	2048	2048
Large	4	32	8	8	256	128	16	8192	8192

Table 4: Considered architecture variations

Parameter	**Values**
Core Width	1; 4; 8
L/S Queue Size	1; 5; 10
ROB Size	32; 64; 128

To ensure the representativeness of the devised model when considering multiple types of workloads, the PARSEC [2] benchmark suite was chosen for its training and validation procedures. For such purpose, simulator-specific *magic* instructions were added to each of the eleven PARSEC benchmarks, in order to define the appropriate simulation Region of Interest (ROI) for each benchmark and exclude the initialization and shutdown phases, since these are uninteresting from an architectural point-of-view. These benchmarks were then executed to completion using the predefined "small" input set on each of the 81 different processors, with the pre- and post-ROI sections simulated in fast-forward mode in order to reduce the processing time. All runtime statistics required by the model were measured during the execution and stored for later processing.

3.1 Model validation

Since the model assumes the representativeness of the training set for all possible applications and cores, it makes sense to use as much information as possible during its validation. Therefore, a leave-one-out cross-validation approach was adopted, such that one random observation was removed from the training set in each iteration, and subsequently used for model validation. In order to further illustrate the quality of the model, multiple goodness-of-fit measures were calculated for each of the 81 individual source core models.

Figure 1 presents the CPI normalized prediction over all considered architecture variations, represented as a Tukey box-plot for each benchmark. As can be observed, the model provides accurate predictions over a wide range of application characteristics for all considered core parameters. On the other hand, it can also be observed that the largest prediction error occurs for the *canneal* and *streamcluster* applications, which is explained by the fact that these benchmarks comprehend a larger inter-phase variation of the observed CPI. Such a variation could be explained (in future work) by evaluating the error across application phases, instead of evaluating across the whole application execution.

An F-test of overall significance [9] was also performed on all models, in order to evaluate whether a simple intercept-only fit would be statistically indistinguishable from the proposed models. The obtained results showed a p-Value of 0 for all cases, which fulfills this basic quality requirement.

Lastly, a scheduler-specific validation test was performed, which evaluates whether the proposed model could effectively predict the most efficient core for each application. Hence, for each iteration of the test, a permutation of one

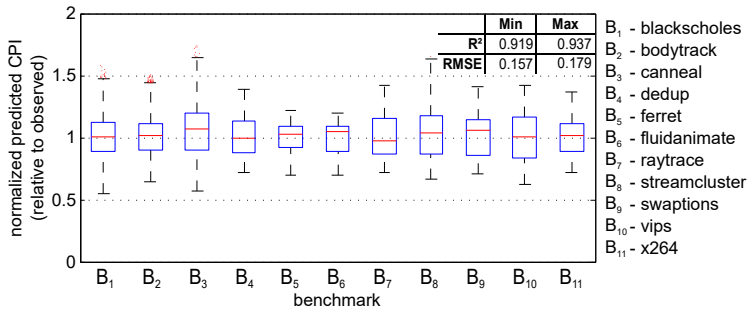

Fig. 1: Predicted CPI (with cross-validation) for all considered architecture variations, with the minimum and maximum values of the coefficient of determination (R^2) and of the Root Mean Square Error ($RMSE$) obtained for all models.

Table 5: Scheduler validation test results

# of Cores N	2	3	6	11
Random Scheduler CPI	1.67	1.67	1.67	1.67
Best/Oracle Scheduler CPI	1.38	1.22	1.07	1.03
Proposed Model Scheduler CPI	1.41	1.28	1.15	1.10
Relative Error (Proposed vs. Oracle)	2.17%	4.92%	7.48%	6.80%

source core and $N-1$ alternative target cores was picked at random. The model was then used to predict the best core (minimum CPI) for each application, out of the N possible choices. The observed CPI in the chosen core was then compared with the observed CPI of a scheduler using either a *random* or an *oracle* policy. A total of 891 000 iterations of this validation mechanism were executed using different values of N. The results, presented in Table 5, show that the model manages to estimate the correct core in a large majority of the cases. Furthermore, when the proposed model performs an incorrect guess, only a reduced performance loss is observed when compared to the *oracle* case.

4 Conclusions

An accurate performance predictor based on a Linear Regression Model is herein proposed to identify, within a heterogeneous many-core processor, the most suitable core-architecture to execute each thread/workload. Hence, it considers the co-existence of multiple cores, characterized by several different parameters, including the cache size, issue-width, ROB size, load/store queues size, etc.

The devised predictor is easily integrated in most system schedulers, providing the ability to periodically determine whether a certain thread is running under the most efficient core-architecture. Conversely, it can also be used in morphable or dynamically reconfigurable structures, not only to determine when the processing architecture should be reconfigured, but also to determine the corresponding set of parameters.

The experimental evaluation showed that the devised model is able to identify the correct core-architecture in a large majority of the cases, leading to average performance differences as low as 7% when compared with the *oracle* solution.

The offered flexibility makes the devised model easily adaptable to other optimization metrics besides the considered CPI. As an example, an energy estimation model can be easily implemented, in order to obtain energy/power-aware scheduling schemes.

References

[1] big.LITTLE Technology: The Future of Mobile. Tech. rep., ARM (2011), available at `https://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf`

[2] Bienia, C.: Benchmarking Modern Multiprocessors. Ph.D. thesis, Princeton University, Princeton, NJ, USA (2011)

[3] Carlson, T.E., Heirman, W., Eyerman, S., Hur, I., Eeckhout, L.: An evaluation of high-level mechanistic core models. ACM Transactions on Architecture and Code Optimization (TACO) (2014)

[4] Kumar, R., Farkas, K.I., et al.: Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In: 36th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 81–92. MICRO 36, IEEE Computer Society (2003)

[5] Kumar, R., Tullsen, D.M., et al.: Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. SIGARCH Comput. Archit. News 32(2), 64–75 (2004)

[6] Patsilaras, G., Choudhary, N.K., Tuck, J.: Efficiently exploiting memory level parallelism on asymmetric coupled cores in the dark silicon era. ACM Transactions on Architecture and Code Optimization (TACO) 8(4), 28:1–28:21 (2012)

[7] Pricopi, M., Muthukaruppan, T.S., et al.: Power-performance modeling on asymmetric multi-cores. In: 2013 Int. Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES). pp. 1–10 (2013)

[8] Saez, J.C., Prieto, M., et al.: A comprehensive scheduler for asymmetric multicore systems. In: 5th European Conference on Computer Systems. pp. 139–152. EuroSys '10, ACM (2010)

[9] Seber, G.A.F., Lee, A.J.: Linear Regression Analysis. John Wiley & Sons (2003)

[10] Shelepov, D., Saez Alcaide, J.C., Jeffery, S., Fedorova, A., Perez, N., Huang, Z.F., Blagodurov, S., Kumar, V.: HASS: A scheduler for heterogeneous multicore systems. SIGOPS Oper. Syst. Rev. 43(2), 66–75 (2009)

[11] Tibshirani, R.: Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological) pp. 267–288 (1996)

[12] Van Craeynest, K., Jaleel, A., et al.: Scheduling heterogeneous multi-cores through performance impact estimation (PIE). In: 39th International Symposium on Computer Architecture. pp. 213–224. ISCA '12, IEEE Computer Society (2012)

[13] Zou, H., Hastie, T.: Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 67(2), 301–320 (2005)