Will it Blend? Merging Heterogeneous Cores

Ilias Vougioukas
University of Southampton
Arm Research
Cambridge, UK
ilias.vougioukas@arm.com

Andreas Sandberg
Arm Research
Cambridge, UK
andreas.sandberg@arm.com

Stephan Diestelhorst
Arm Research
Cambridge, UK
stephan.diestelhorst@arm.com

Bashir M. Al-Hashimi
University of Southampton
Southampton, UK
bmah@ecs.soton.ac.uk

Geoff V. Merrett
University of Southampton
Southampton, UK
gvm@ecs.soton.ac.uk

***Abstract*—Heterogeneous processors allow different performance/power operation points by pairing high performance Out-of-Order (OoO) cores with small energy efficient In-Order cores. Timely switching between the cores types allows the system to tailor to energy budget and performance constraints. Each core migration incurs a penalty though, which is caused by the necessary transfer of state between cores. Heterogeneous systems are kept from further increasing performance through more frequent migrations, as the benefits are suppressed by these overheads. Sharing some of the components between cores can reduce the transfer overheads and potentially overcome this problem. However, multiplexing core components can have also an adverse effect on performance when done without taking into consideration the extra latency. For this reason, it is critical to strike a balance between component sharing and complexity.**

In our work, we show that sharing the translation mechanism and the level 2 caches capture most of the potential benefits for in-order cores, while the branch predictor state is most critical for Out-of-Order cores. Finally, even under a best case scenario, on average 10% of applications could transfer to an OoO core to gain extra performance, while 6% of execution could migrate to In-Order cores saving significant energy, without incurring any penalty to the system. Under a more relaxed trade-off policy, benefits increase significantly as, on average, for 66% of the time executed a migration to the opposite core to improve the energy delay product.

***Index Terms*—Heterogeneous multiprocessors, HMP, Out-of-Order, In-Order, memory translation, TLB, cache memory, gem5, simulation, branch prediction.**

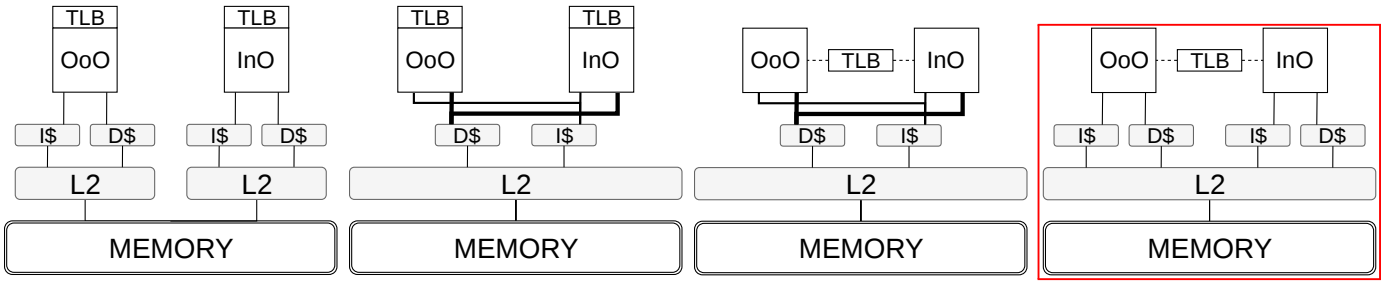
I. Introduction

Modern systems have to balance performance with energy efficiency. To address this, they use heterogeneous cores to allow them to operate at different performance/power efficiency [1], [2]. *Out-of-Order (OoO)* cores are used to deliver high performance, as their wider pipeline, ability to reorder instructions, and larger caches allows them to better exploit *memory level parallelism (MLP)* and *instruction level parallelism (ILP)*. Heterogeneous systems use significantly smaller and simpler *In-Order (InO)* cores that function at a much more energy efficient operation point. By taking advantage of migrations between them, workloads can therefore select the best setting for any given energy/delay goal [3].

This happens as workloads, can be broken down into smaller periods of execution that exhibit some commonality, usually referred to as *phases*. These phases can be characterized as high ILP or MLP for instance, or very memory intensive. In the first two cases it would be beneficial, for example, to execute on a OoO core while in the last case the InO core could conserve energy while waiting for memory responses. Phases are claimed to vary in "size" ranging from large code blocks to just a few thousand instructions [1], [4]. As such, studies performing more frequent switches between the two core types show that this could potentially unlock additional benefits [4]–[7].

However, every migration to a new core unavoidably leads to some slowdown [8]–[10], as the system needs to transfer state from one core to the other. For current systems, the slowdown is mostly attributed to private caches, the branch predictor state, the register file and the address translations. For infrequent switching this slowdown is arguably not noticeable as the system has time to amortize these costs. On the other hand, as the migrations happen more frequently the overheads counterbalance any switching benefits, ending up outweighing them after a certain point. In the past it has been difficult to capture these overheads and provide a more accurate responses to whether it is feasible to improve performance/energy efficiency through frequent migrations and which configuration makes this possible. In this work we:

- Describe a mechanism that quantifies hardware migration overheads.
- Show that sharing level 1 caches is not as beneficial as sharing some of the translation mechanism.
- Show that even under the best case, at most, only 10% of workload execution can improve performance or preserve energy, by switching without having any negative consequence. Under a more relaxed policy like *energy-delay product (EDP)*, we can find 66% of phases where it is beneficial to migrate.



(a) No sharing. (b) Share caches. (c) Share caches and TLB. (d) Proposal: Share L2 and TLB.

Fig. 1: Different types of sharing schemes examined for HMP systems [8].

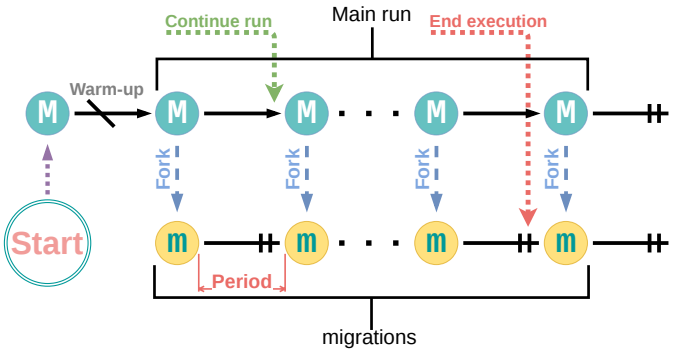


Fig. 2: Our methodology uses a main run (M) that spawns periodic forks which perform a migration (m) [8].

II. Setup and Results

To quantify the overheads we use a modification of the gem5 simulator [11]. Our methodology clones or "forks" the simulation, creating an identical duplicate [8]. In the clone we proceed to replace the core design with a new one without any state. To simulate various levels of sharing, we flush the combinations of components such as the caches and the *translation lookaside buffer (TLB)*, as shown in Figure 1. To get enough data points, we perform this technique across the entire execution of an application, periodically spawning the forking points (Figure 2). To assess the performance degradation of a certain core type, OoO or InO designs, we swap the simulate a migration to a state-clear core of the same type and register the performance difference. Figure 3 shows that for InO sharing the Level 2 caches and the TLB is the best setup, especially when taking into account complexity and performance. The OoO core loses as much as 40% of its performance when migrating at high frequencies. This happens as the incoming core starts with no branch prediction state which heavily penalizes the OoO pipeline (Figure 4).

To assess whether a migration is beneficial in a heterogeneous system we use the same forking methodology. First policy examines phases that improve energy efficiency without trading off performance, by switching from an OoO core to an InO. Similarly, we examine cases where the OoO core performs so much better than the InO design that the system ends up saving energy. Another more lenient metric aims at measuring the cases that minimize the EDP (Figure 5).

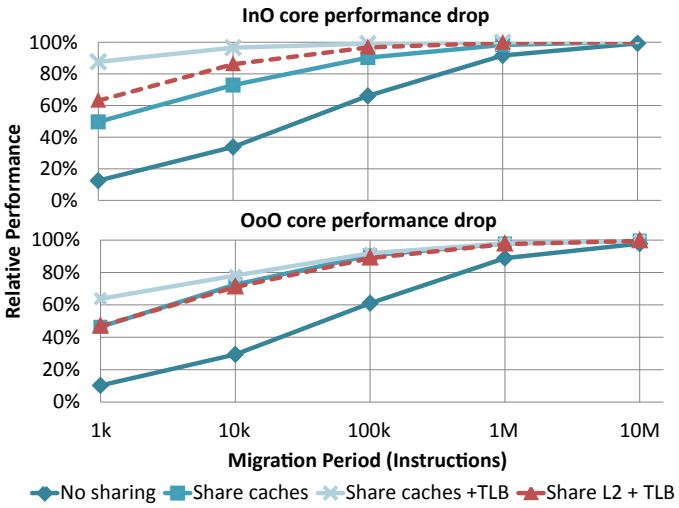


Fig. 3: The average performance degradation for the two core types as a function of migration frequency [8].

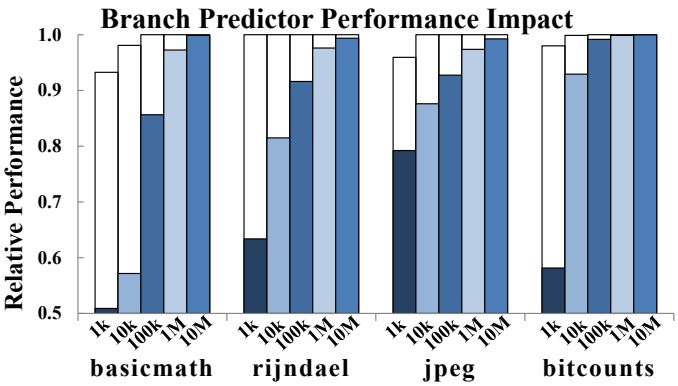
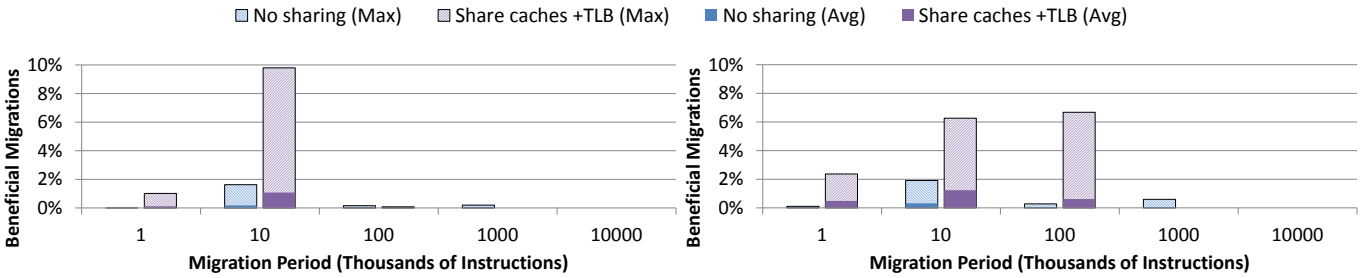


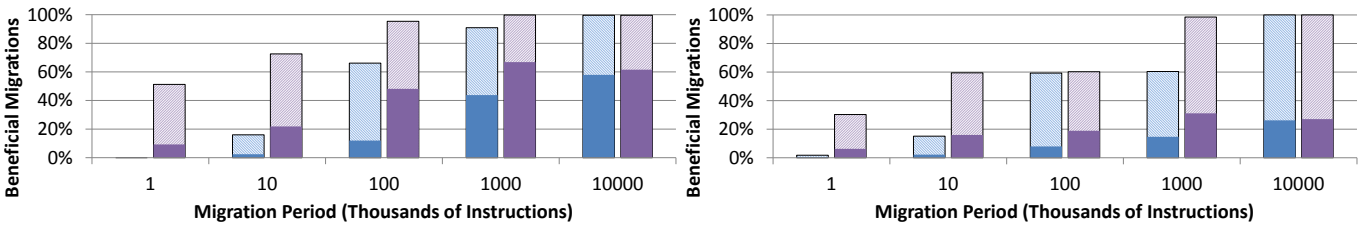
Fig. 4: An OoO core that loses the branch predictor state through migrations (coloured bars), performs as much as 40% worse than a system that preserves it (clear bars).

We observe that minimal gains exists when the policy does not trade performance and energy, no matter the migration period. On the other hand, when using an EDP metric, we show that for migration periods of 100k and 1M instructions, a system that shares the caches and the TLB can improve by about 60% and 30% on average, when switching to the OoO and the InO core respectively [8].



(a) Beneficial migrations from the InO to the OoO core with the No trade-off policy.

(b) Beneficial migrations from the OoO to the InO core with the No trade-off policy.



(c) Beneficial migrations from the InO to the OoO core with equal or better EDP policy.

(d) Beneficial migrations from the OoO to the InO core with equal or better EDP policy.

Fig. 5: Percentage of beneficial migrations when being energy- and EDP-constrained [8].

III. Conclusions

In this work, we have proposed a full system simulation methodology using gem5, to study the effects of migrations in heterogeneous systems. Our methodology accomplishes this by forking the simulation, comparing a system that migrates with one that does not. We use our methodology to focus on identifying how much sharing of internal components (e.g. caches, TLB, branch predictor) is most desirable for migrations as fine as 1k instructions.

Our results show that the two cores have asymmetric warm-up times and behaviour, where the OoO core needs larger instruction windows to amortize the cost of migration than the InO core. Contrary to many academic studies we show that sharing all the caches is not the most advantageous set up and propose an alternative that only shares level 2 caches and the TLB. Our design performs equally as well as the alternatives, but can be physically much simpler to implement [8], [12].

We also measure the potential benefits for heterogeneous systems running a suite of diverse embedded benchmarks. The results show that, even when the overheads are not prohibitive, migrating to save energy without sacrificing performance or vice versa is limited to at most 10% of the execution. Relaxing the policy to find equal or better EDP points is beneficial on average for 66% of the workload execution.

References

[1] 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 - 2012 IEEE/ACM 45th International Symposium on Microarchitecture, MICRO 2012*, 2012.

[2] V. Villebonnet, G. Da Costa, L. Lefevre, J.-M. Pierson, and P. Stolf, "Big, Medium, Little: Reaching Energy Proportionality with Heterogeneous Computing Scheduler," *Parallel Processing Letters*, vol. 25, 9 2015.

[3] S. Padmanabha, A. Lukefahr, R. Das, and S. Mahlke, "DynaMOS," in *Proceedings of the 48th International Symposium on Microarchitecture - MICRO-48*, vol. -, no. -, 2015.

[4] C. Fallin, C. Wilkerson, and O. Mutlu, "The heterogeneous block architecture," in *2014 32nd IEEE International Conference on Computer Design, ICCD 2014*, vol. -, no. -, 2014.

[5] A. Lukefahr, S. Padmanabha, R. Das, R. Dreslinski, T. F. Wenisch, and S. Mahlke, "Heterogeneous microarchitectures trump voltage scaling for low-power cores," *Proceedings of the 23rd international conference on Parallel architectures and compilation - PACT '14*, 2014.

[6] 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, 2016.

[7] S. Navada, N. K. Choudhary, S. V. Wadhavkar, and E. Rotenberg, "A unified view of non-monotonic core selection and application steering in heterogeneous chip multiprocessors," *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*, 2013.

[8] I. Vougioukas, A. Sandberg, S. Diestelhorst, B. Al-Hashimi, and G. Merrett, "Nucleus: Finding the sharing limit of heterogeneous cores," *ACM Transactions on Embedded Computing Systems*, vol. 16, 2017.

[9] 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 *2015 IEEE Hot Chips 27 Symposium, HCS 2015*. IEEE, 8 2016.

[10] E. Rotenberg, B. H. Dwiel, E. Forbes, Z. Zhang, R. Widialaksono, R. Basu Roy Chowdhury, N. Tshibangu, S. Lipa, W. R. Davis, and P. D. Franzon, "Rationale for a 3D heterogeneous multi-core processor," *2013 IEEE 31st International Conference on Computer Design, ICCD*, 2013.

[11] N. Binkert, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, D. A. Wood, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, and T. Krishna, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, 8 2011.

[12] A. Bhattacharjee and M. Martonosi, "Characterizing the TLB behavior of emerging parallelworkloads on chip multiprocessors," in *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*, 2009.