Thread Motion in Private L1 Chip Multiprocessors

Athula Balachandran, Lavanya Subramanian

***Abstract*—Low power computing has become a topic of great interest in the recent past. But computation limited by power budget can lead to suboptimal performance. However studies have shown that applications show very high variability in their performance requirement and this could be exploited. Dynamic Voltage Frequency Scaling (DVFS) is a traditional technique that is used to exploit run time variability and conserve power with minimal performance degradation. However due to practical limitations, DVFS cannot exploit the very fine grained variability shown by various applications. In this context, thread motion was proposed as a mechanism that helps applications operate at frequencies and voltages that are appropriate for their performance levels, by migrating at very low granualarity. However there is no work that looks at the performance degradation imposed by performing thread motion in case of Chip MultiProcessors (CMP) with private L1 caches. In this project, we show the performance degradation that is incurred in a generic CMP with private L1 caches and also present some interesting results on the energy trends, for different migration intervals. We also evaluate the use of a moving average history IPC predictor over a last value IPC predictor and show that the performance degradation is reduced by around 5%.**

***Index Terms*—Parallell architecture, Distributed Architecture, Performance, Low power computing**

I. Introduction

There has been a lot of emphasis on low power computing in the near past. Low power computing is very essential in the case of battery-operated computing systems with limited power capabilities. It is also a key requirement in high end machines and data centers where cooling has become one of the major concerns. Even in the multi-core chips power consumption seems be a very crucial design consideration. This is because high power consumption can lead to side effects that affect the reliabilty of the chip, performance and packaging cost.

In order to stay within the power budget, there is a need to carefully constrain application performance leading to suboptimal performance. However studies have shown that applications show very high variability in their performance requirement. Hence this can be used to optimize the performance requirement while staying within the given power budget. Also die process variation can cause the individual cores in a Chip Multi Processor (CMP) to significantly differ from the others. These can lead to intelligently scheduling processes based on their requirement based on the variation of processors within the CMP can lead to significant performance improvement.

In the context of power savings, Dynamic Voltage Frequency Scaling (DVFS) is a traditional technique that is used to exploit run time variability and conserve power with minimum performance degradation. Intel's XScale and Transmeta's Crusoe are examples of the many microprocessors that support DVFS functionality. Typically DVFS is employed at the OS scheduler intervals. However recent research shows that applications' variability behaviour is more fine-grained. Hence this cannot be exploited efficiently by performing DVFS at OS scheduler intervals. The OS scheduler samplings are of the order of milliseconds whereas the application variations occur at nanoseconds time scale. But employing per-core DVFS at fine grained intervals imposes a huge delay overhead for the regular voltage level transistions and hence is practically impossible with off-chip regulators. However the fact that multicores run processes with different computation needs can be utilized to efficiently switch cores instead of performing per-core DVFS. Hence by having different cores assigned different voltage/performance levels and employing an intelligent scheduling mechanism we can achieve good performance in the multi core system. Our project looks at the challenges and bottlenecks in applying this to a generic chip multiprocessor.

This is the basic idea that has been put forth in [1]. The authors evaluate it in the context of an architecture where cores are clustered and the L1 cache is shared by the processors in that cluster. However, generic CMP architectures are not built this way. The processors typically have private L1 caches and share a physically partitioned L2 cache. The main motivation for this project has been to evaluate how well thread motion works for this sort of a more generic architecture.

A. Related Work

Reducing power density and power-aware designs has been the focus of much work in the past. DVFS is a well known traditional technique for performing power management in many of the contemporary microprocessors. The power consumption can be reduced when the

cores are idle by appropriately setting the voltage and frequency levels of the individual cores. In the case of single core microprocessors, it can be used to boost the performance by remapping the VF settings accordingly. The application variability is tracked and is used to arrive at the appropriate voltage/frequency setting for the core. [2] looks at DVFS in the context of Chip Multi Processors and exploits this application variabilily. [3] looks at variation aware DVFS, where the DVFS algorithms also exploit the variability information of the processors.

In the case of a multi-core set up, it is difficult to maintain a per-core DVFS control based on the application requirement running on that core. Generally the individual needs of each of the cores are taken into consideration to arrive at a VF setting for the entire core. This can lead to suboptimal performance and power efficiency. The other option is to have control over per core VF setting. However employing DVFS at fine-grained intervals imposes a huge delay overhead for the regulator voltage level transitions and is practically impossible, with off-chip regulators. The scalability and cost issues related to this are huge and hence this is not a viable solution.

Hence DVFS algorithms are implemented by the operating system and the scheduling and application variability monitoring occur at millisecond time interval of the OS scheduler. To exploit fine-grained application variability, DVFS has to be performed at finer grained intervals. Hence there is a need to monitor application phase monitoring at finer time scales. These require taking decisions on DVFS decisions at intervals of the order of hundreds of microseconds. This cannot however be supported by even the state of the art power management schemes due to hardware limitations. These systems incur a huge VF transistion delay before reaching the target power mode. Voltage transistion times are highly limited by off-chip voltage regulators that limit how quickly the voltage can be changed. Similarly, PLL relock times limit how fast the frequency can be changed.

Because of the limitations that are mentioned above, it is almost impossible to evaluate application behavior and remap VF at finer intervals. Previous work does do migration either at OS intervals, [4] for process variation-aware application mapping combined with DVFS and [5] during thermal hotspots/emergencies. However performing migration of processes based on their variability and mapping processes to the right processor with the apt VF setting can achieve the advantage offered by a finer-grained DVFS scheme without hardware restrictions. In [1], the authors employ an architecture similar to the Sun ROCK processor to perform thread motion. This architecture groups processors into clusters and they share an L1 cache. The migration algorithm attempts to map applications ranked in the order of IPC to cores ranked in the order of frequency. A last value predictor is used for the IPC. Migrations that are performed within a cluster do not suffer the impact of missing L1 cache data. However, in most Chip Multiprocessor Systems, each processor has a private L1 cache. So, our project explores the effectiveness of the ”Thread motion” scheme in this scenario. Specifically, we try to quantify the performance degradation, that would result from the L1 misses, when migration is performed. We observe that the concept of intra and inter clusters does not apply in our scenario.

B. Contributions

The following are our contributions in the project.

- We quantify the performance degradation when the migration overhead is not accounted for. This is the baseline and clearly indicates the upper limit for thread motion.
- We quantify and compare the performance degradation versus power savings for different migration intervals, when the migration overhead is accounted for. We compare this against the baseline.
- We look at moving average IPC predictors, that use different amounts of history to predict the IPC for the next interval. Specifically, we attempt to quantify how much history helps in better IPC prediction. This is effectively quantified by the resultant reduction in performance degradation and increase in power saving.

II. Design Details

We briefly describe the algorithm we use to carry out our analyses. The next section details the kind of architecture we use and the migration effects we model. This, we think, is the major part of our project, as the very idea is to look at thread motion from the perspective of a typical CMP architecture, when the effect of migration is modeled.

A. Algorithm

The thread motion algorithm that we employ in the one proposed in the [1] for the first set of experiments. A last value predictor is used to predict the IPC of each application for the next migration interval.

For the latter half of the evaluations, we employ a moving average IPC predictor. The impact of the choice of α (the history parameter) on the resultant performance degradation and power savings are studied in detail.

III. Implementation Details

This section talks in detail about the architecture and modeling framework that we build to use in all our evaluations. The choice of architecture and the migration impact we model, are critical to our evaluation.

A. Simulator

We use an inhouse x86 simulator called the BLESS simulator. This is CMP simulator which models the cores and caches at a coarse granularity and the interconnect network in finer detail. This would suit our purposes, since quantifying the impact of migration is largely about modeling the on-chip communication between the cores and the various physically apart L2 banks.

B. Power Macromodeling

BLESS is a performance simulator. The power models in this simulator are built out of static profiling of SPEC2000 benchmarks in Wattch/simplescalar.

C. Migration Infrastructure

We built migration infrastructure into the BLESS simulator. The processors are ranked according to their frequencies statically. We implement a scheduler/controller that collects the IPCs of different applications every migration interval. The IPCs are either the last values from the previous intervals or the values from the moving average history predictor, depending on the evaluation. We map the applications onto the processors such that teh highest IPC application gets onto the highest frequency processor. The key component of the migration infrastructure is the migration overhead. This consists of the following:

- I Cache cold start effect
- D Cache cold start effect
- Register/Architectural state transfer latency

We model 2 and 3. However, we do not model 1, as the simulator does not take in the instruction addresses/Program counters. However, 2 and 3 are the major components and we believe our results would be reasonably accurate, despite not accounting for 1. We model a CMP with 16 cores arranged in a mesh network (4x4). Each core has a private L1 and the L2 is in the form of physically separate banks, that are shared among all cores.

IV. Experimental Setup

The key objective behind our evaluations is to quantify the performance degradation due to the migrations cost. We use $MIPS$, $TotalEnergy$ and $Energy/(Throughput)^2$ (as a proxy for ED2) normalized to to the 100% power budget case to quantify performance degradation and power reduction. We evaluate these across different migration intervals of 500, 1000, 5000 and 10000.

All evaluations have been carried out with SPEC2006 benchmarks. We use a 4x4 CMP network for all our evaluations. All results are averaged across 10 different random application mixes. We ran each simulation/evaluation for a span of 5 million instructions. We used 2 different voltage/frequency levels f and f/2, (where f is 3.0 GHz), to derive voltage/frequency configurations for the different power budgets.

V. Evaluation

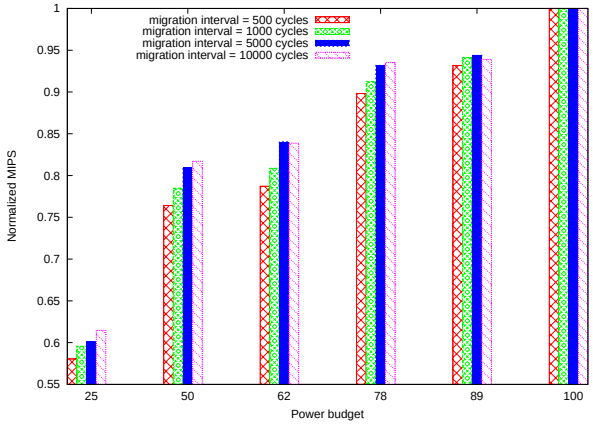

Fig. 1. Normalized $MIPS$ at different power budget for various migration intervals assuming no overhead during migration

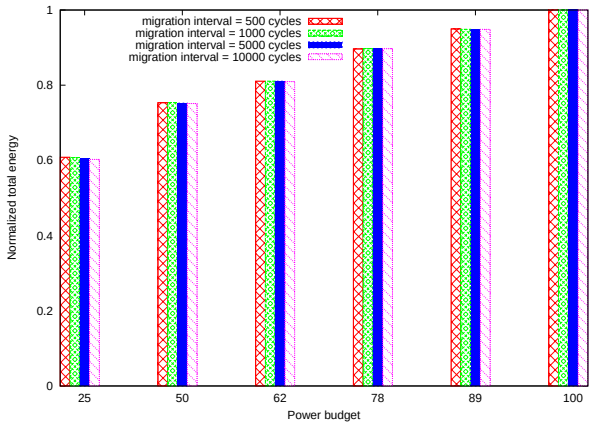

Fig. 2. Normalized $TotalEnergy$ at different power budgets for various migration intervals assuming no overhead during migration

As a baseline, we evaluate the performance degradation and the power savings obtained, when there is no overhead associated with migration. This represents the theoretical upper limit on what thread motion can deliver.

Fig. 1 shows the performance degradation for this baseline case. The performance degradation suffered here is minimal and is solely due to some cores operating at lower frequencies. Fig. 2 shows the energy savings in this ideal scenario. There is not too much variation due to the migration interval. This is interesting

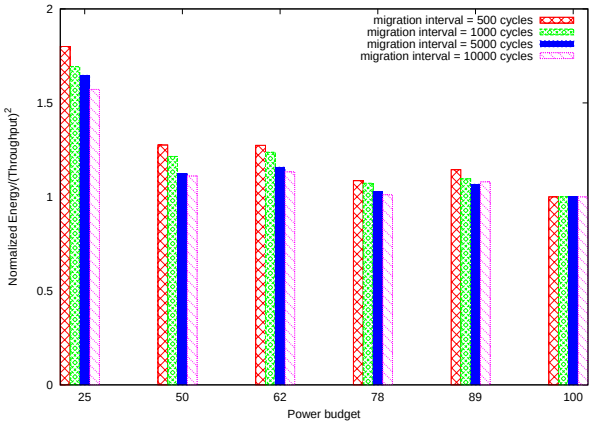

Fig. 3. Normalized $Energy/(Throughput)^2$ at different power budgets for various migration intervals assuming no overhead during migration

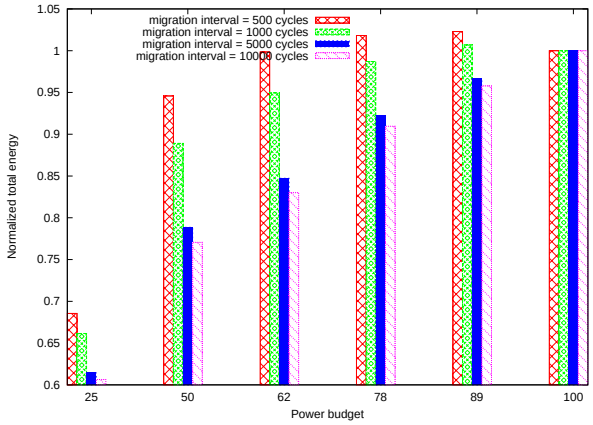

Fig. 5. Normalized $TotalEnergy$ at different power budgets for various migration intervals

because [1] clearly states that the motivation for fine grained thread motion is that it exploits application variability at fine grained intervals, providing better scope for power savings. [1] presents the variability of different applications in terms of their IPC and makes a case for using slower processors, with instantaneous power savings in mind. However, it doesnt show how much total energy savings this translates into. Fig. 3 shows the overall energy/performance impact in terms of $Energy/(Throughput)^2$ increase.

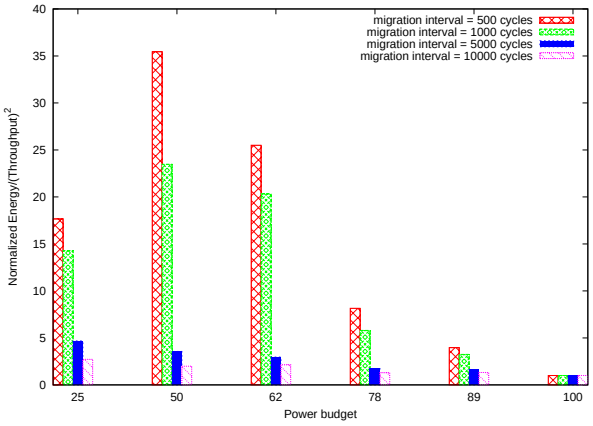

Fig. 6. Normalized $Energy/(Throughput)^2$ at different power budgets for various migration intervals

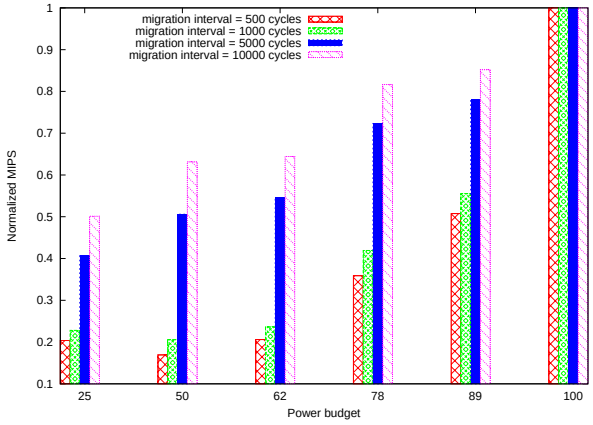

Fig. 4. Normalized $MIPS$ at different power budget for various migration intervals

Fig. 4, 5 and 6 show the normalized $MIPS$, $TotalEnergy$ and $Energy/(Throughput)^2$ for different migration intervals respectively. The $MIPS$ clearly shows a very high degradation as the power budget decreases, specifically as the migration interval decreases. The length of the migration interval determines the frequency of migrations. So, this trend is to be expected. However, Fig. 5 shows an interesting trend. For different power budgets, the total energy consumed shows a large increase (ranging from around 5% to 20% across power budgets) as the migration interval is varied from 500 cycles to 10000 cycles. This is clearly due to the migration cost, as this was not present in the baseline. This shows that the cost of migration in terms of energy is also huge when it is done at finer grained intervals and hence does not really translate into an energy aware design choice. Fig. 6 shows the large increase in $Energy/(Throughput)^2$ when the migration interval is made very small.

All the results above are for a last value predictor. Fig. 7, 8 and 9 show the normalized $MIPS$, $TotalEnergy$ and $Energy/(Throughput)^2$ for a power budget of 78% but for different values of α. The predicted IPC is assumed to be a moving average as follows $PredictedAvgIPC = \alpha \times PreviousIPC + (1-\alpha) \times PredictedAvgIPC$. $PredictedAvgIPC$ is used to determine the core to which migration is to be done. We use different values for α and look at the

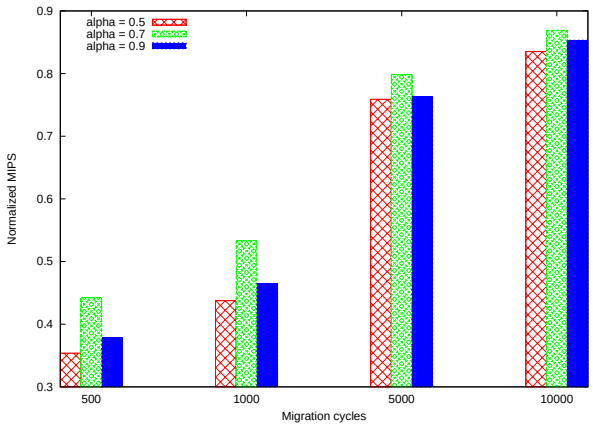

Fig. 7. Normalized $MIPS$ at a power budget of 78% for different α values and migration intervals

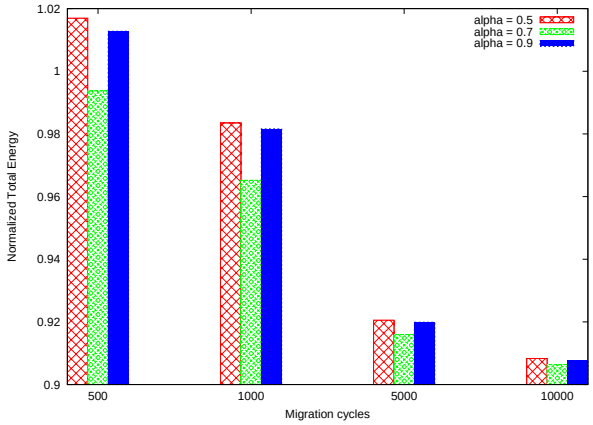

Fig. 8. Normalized $TotalEnergy$ at a power budget of 78% for different α values and migration intervals

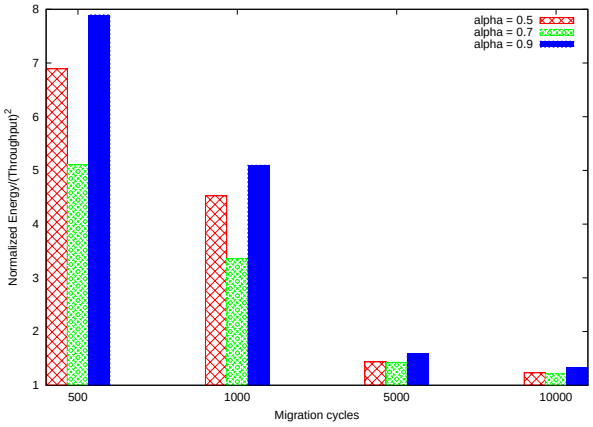

Fig. 9. Normalized $Energy/(Throughput)^2$ at a power budget of 78% for different α values and migration intervals

performance degradation and power savings. We present the results only for a specific power budget, however this trend is followed across all power budgets. α value of 0.7 yields the least performance degradation (around 5% less than than for 0.9 and 0.5), as shown in Fig. 7 and also marginally improves the power savings as shown in Fig. 8. α of 0.9 is very similar to a last value predictor. So, these results show that using a moving average predictor with a balanced mix of recent and past history (with an α of around 0.7), definitely reduces the performance hit and also marginally improves power savings.

VI. Surprises and Lessons learned

In terms of tools setup and what to expect in terms of tool capabilities, there weren't any major surprises. This was because we were reasonably aware of the capabilities and limitations of the simulator, when we started on the project. However, the results were surprisingly interesting.

- First of all, though we started with the knowledge that there would be a larger performance degradation in a private L1 architecture, the kind of degradation numbers we got with decreasing migration intervals were much larger than we expected.
- Further, we expected to see a larger energy saving with decreasing migration intervals. This again did not hold true and basically kills the motivation to perform thread motion at such fine granularities, if the energy costs incurred are as huge.

VII. Conclusions and Future Work

In Conclusion, thread motion makes for an energy-aware and performance effective design choice, only if

- the major hurdle of migration cost is studied and mitigated.
- the number of migrations themselves is reduced

Using a moving average history predictor with a balanced mix of recent and past history (with an α of around 0.7) would definitely reduce the performance hit to some extent, by leveraging the second point discussed above.

We believe there is a huge research potential in reducing the migration cost. One possible solution to mitigate the migration cost is migrating within a limited geographical distance. If this were to be done, then the different voltage/frequency cores should be placed strategically, to support this. Another option is to predict where an application would migrate next and transfer the L1 D cache contents to the L2 closest to it.

VIII. Acknowledgements

We would like to thank Prof. Mowry for giving us the opportunity to do the project. We also thank him for his valuable feedback. We are also indebted to Dr.Siddharth Garg for his valuable inputs and feedback, through the course of the project.

REFERENCES

[1] K. K. Rangan, G.-Y. Wei, and D. Brooks, "Thread motion: fine-grained power management for multi-core systems," in *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*. New York, NY, USA: ACM, 2009, pp. 302–313.

[2] S. Herbert and D. Marculescu, "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors," in *ISLPED '07: Proceedings of the 2007 international symposium on Low power electronics and design*. New York, NY, USA: ACM, 2007, pp. 38–43.

[3] ——, "Variation-aware dynamic voltage/frequency scaling," in *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, Feb. 2009, pp. 301–312.

[4] R. Teodorescu and J. Torrellas, "Variation-aware application scheduling and power management for chip multiprocessors," in *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 363–374.

[5] A. K. Coskun, R. Strong, D. M. Tullsen, and T. Simunic Rosing, "Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors," in *SIGMETRICS '09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM, 2009, pp. 169–180.