Latency-aware Dynamic Voltage and Frequency Scaling on Many-core Architectures for Data-intensive Applications

Zhiquan Lai*, King Tin Lam†, Cho-Li Wang†, Jinshu Su‡*, Youliang Yan§ and Wangbin Zhu§
*National University of Defense Technology, China
†The University of Hong Kong, Hong Kong
‡National Key Laboratory of Parallel and Distributed Processing (PDL), China
§Huawei Technologies Co., Ltd., Shenzhen, China

***Abstract*—Low power is an important design requirement for HPC systems nowadays. Dynamic voltage and frequency scaling (DVFS) has become the commonly used and efficient technology to achieve a trade-off between power consumption and system performance. However, most of the prior work using DVFS did not take into account the latency of voltage/frequency scaling, which is a critical factor in real hardware determining the efficiency of the power management algorithm. This paper investigates the latency aspects of DVFS on a real many-core hardware platform. We propose a latency-aware DVFS algorithm to achieve profile-guided power management to avoid aggressive power state transitions. We evaluate our algorithm on the Intel SCC platform using a data-intensive benchmark, Graph 500. The experimental results not only show impressive potential for energy saving in data-intensive applications (up to 31% energy saving and 60% EDP reduction), but also evaluate the efficiency of our latency-aware DVFS algorithm which achieves 12.0% extra energy saving and 5.0% extra EDP reduction, while increasing the execution performance by 22.4%.**

***Keywords*-power management; DVFS; latency-aware; algorithm; Graph 500; data-intensive**

I. Introduction

Power management is an increasingly important aspect in both research and industry of high performance computing (HPC). As computing systems are approaching a huge scale, power consumption takes a great part in their total costs of ownership. Dynamic voltage and frequency scaling (DVFS) is an efficient technology to achieve a trade-off between performance and power by dynamically and adaptively changing of the clock frequency and voltage [1][2][3][4][5]. However, most of the prior work based on DVFS did not consider the latency of voltage/frequency scaling. As we investigated, the latency of voltage scaling is non-negligible, especially in the many-core architecture with multiple voltage domains, e.g. the Single-chip Cloud Computer (SCC) platform [6]. A power state transition without awareness of the latency would not achieve expected power efficiency, and even worse sometimes, introduces performance lost and more energy dissipation.

The goal of this paper is to explore latency-aware DVFS algorithm for data-intensive applications, which are expected having more potential for energy saving then compute-intensive applications [7]. There are a few of existing work considering the latency overhead of DVFS. Ye et al. [8] proposed reducing the number of power state transitions by including task allocation into learning-based dynamic power management (DPM) for multi-core processors. However, program execution pattern usually change according to the work flow so that the optimal power settings for each phase of program execution are likely to be different. Although task allocation reduces the times of DVFS scaling, it does miss some opportunities for power/energy saving. Ioannou et al. [9], who also used the Intel SCC for evaluation, realized the latency overhead problem, but they just make the voltage transitions more far away with each other using a threshold of the least distance time.

There are also number of DVFS-based DPM scheme proposed for many-core systems [9][10][11][12]. Ma et al. adopted control theory to precisely control the power of the entire many-core chip [10]. Ioannou et al. proposed a hierarchical DVFS controller using phase prediction algorithm for MPI application [9]. David et al. demonstrated a power management algorithm that runs in real time and dynamically adjusts the performance of islands of cores to reduce power consumption while maintaining the same level of performance [13]. However, they all did not consider the latency overhead of DVFS, even though they did the evaluation on the real many-core hardware platforms.

In this paper, we propose a latency-aware DVFS algorithm, which avoids unnecessary aggressive power state transitions. The aggressive here means too short the power state transition is away from last transition. Aggressive transitions will cause frequent voltage/frequency scalings which should introduce more overhead of DVFS. According to our experimental results, the latency-aware algorithm is able to achieve more significant energy and EDP improvements than the baseline power management. The contribution of this paper includes following aspects:

1. First, we study in depth the latency characteristics of voltage/frequency scaling on a real many-core hardware platform, Intel SCC. We find the latency of voltage scale on Intel SCC sometimes can be up to hundreds of milliseconds.
2. Based on the DVFS latency investigation on many-core architecture, we propose a latency-aware DVFS

algorithm for our baseline power management approach, profile-based power management. But the algorithm can be applied to other power management approaches.

3. We evaluate our latency-aware DVFS algorithm on the Intel SCC using the Graph 500 benchmark. The experimental results show that our profile-based DPM scheme obtains impressive energy and EDP savings. Moreover, the latency-aware DVFS algorithm achieves extra improvement in both performance gain and energy saving.

The remainder of this paper is organized as follows. Section II discusses the basic concept of DVFS latency and its investigation on many-core architecture. We describe our new the latency-aware DVFS algorithm in Section III. Section IV presents experiments, and Section V analyzes the results collected. Finally, we conclude the paper in Section VI.

II. Latency of DVFS on Many-core Architecture

Before proposing the latency-aware DVFS algorithm, we investigate the latency behavior of voltage/frequency scaling. In particular, we fucus the study on many-core tiled architecture with multiple voltage domains.

A. Basic concept of DVFS latency

As an important feature for dynamic power management (DPM), many chips now provides multiple power states (different states of voltage and frequency) for the system to adaptively switch betweeen according to different program execution patterns. One basic but important rule during DVFS is that, the voltage must support the frequency all the time, i.e. the current frequency can not exceed the maximal frequency which the current voltage supports. As shown in Fig. 1, we assume there are three different frequency values provided by the hardware, F0, F1, F2, where $F0 < F1 < F2$. For each frequency value, there is a theoretical least voltage value that satisfies this frequency's need. According to this condition, we can draw a line of "safe boundary" for all voltage/frequency states. Thus, all the voltage/frequency states above this boundary are not safe (or dangerous) as they violate the basic condition, which could damage the hardware, whereas all the voltage/frequency states under this boundary are considered safe.

However, to ensure safe execution, we usually apply a higher voltage value than the theoretical least voltage value. As shown in Fig. 1, there is a margin between the least voltage value and the theoretical safe boundary for each frequency. Actually, this margin is somehow necessary in practice. Since the execution performance only depends on frequency, keeping the voltage at the least voltage values should be the most power-efficient states (the green states in Fig. 1). Of course, we can apply much higher voltage than the least voltage for each frequency (the orange states in Fig. 1). Although these states are safe, they unnecessarily consume more power than those least-voltage states with the same frequency.

If we scale the power state (values of voltage and frequency) from (V_s ,F_s) to (V_d ,F_d), assuming they are both safe states,

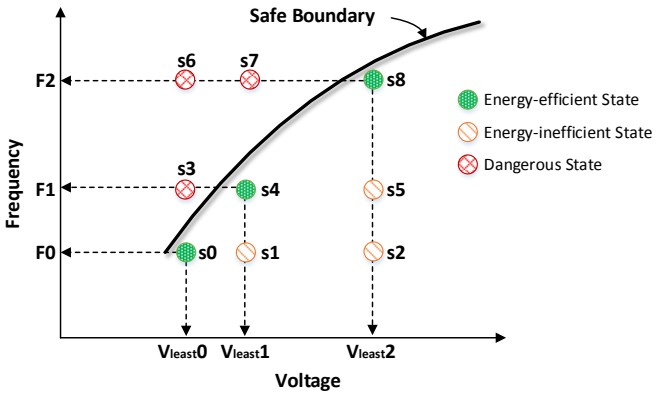

Fig. 1. Relationship between voltage and frequency during dynamic scaling

TABLE I
Latency of DVFS in Different Scaling Cases

Case	Strategy of Voltage/Freuqency Scaling	Latency
$F_s > F_d$ && $V_s > V_d$	1. Scaling down frequency 2. Waiting till frequency scaled 3. Scaling down voltage	$Latency(F_s \rightarrow F_d)$
$F_s < F_d$ && $V_s < V_d$	1. Scaling up voltage fisrt 2. Waiting till voltage scaled 3. Scaling up frequency 4. Waiting till frequency scaled	$Latency(V_s \rightarrow V_d) +$ $Latency(F_s \rightarrow F_d)$

we indeed have to scale the voltage and frequency separately. As we know, there is a delay for both frequency and voltage scaling. However, the latency of voltage scaling is always much larger than that of frequency scaling. Thus we must consider whether the power state will exceed the safe boundary during their scaling. In the case of scaling up voltage and frequency, if we alter the frequency first, then the voltage may not be high enough to support the scaled frequency.

We found that the latency of voltage scaling should be taken into account only when both the frequency and voltage need to be scaled up. In other cases, where min(V_s, V_d) is high enough to support max(F_s, F_d), although latency is needed to scale the voltage from V_s to V_d (also for frequency from F_s to F_d), as current voltage level is high enough to support the frequency, the programs can keep going after scaling the frequency first. Apart from the minuscule latency of frequency scaling, there is no noticeable latency after scaling down the voltage. In the case that $V_s < V_d$ and $F_s < F_d$, after scaling up the voltage (we always scale up the voltage first for reliability in this case), we should wait for a moment to let the voltage reach the level of V_d, which is safe to support the new frequency F_d. If we scale the frequency to F_d when the voltage level is not high enough, the CPU will stop working because the voltage can not support the frequency. This situation is very dangerous and could damage the chip.

In conclusion, we have the strategies for voltage/frequency scaling and the corresponding latency. For low power, we

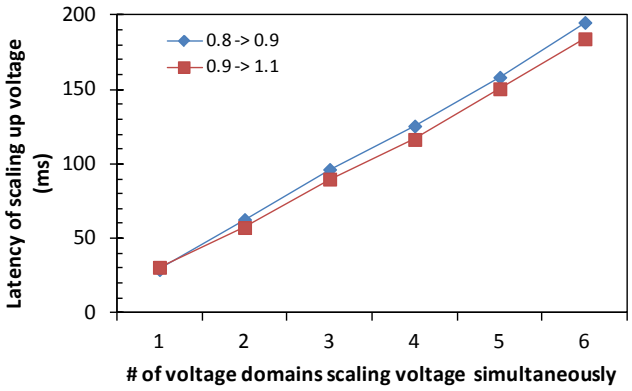

Fig. 2. Latency of voltage scaling on chip with multiple voltage domains

assume the power states switch between power-efficient states. In these cases, it is true that $F_s > F_d$ only if $V_s > V_d$. As shown in TABLE I, in the case of scaling down the power state, we scale down the voltage after scaling down the frequency so that the programs need not wait for voltage scaling to finish. When scaling up the power state, the programs have to suspend and wait until the voltage gets scaled, and then continue on scaling the frequency.

1) Latency of DVFS on many-core architecture: The lack of the model of DVFS latency for many-core architecture with multiple voltage domains is a crucial research gap to fill. We investigate and contribute this model on a real many-core chip, the Intel SCC [6], which is an experimental 48-core CPU for many-core software research consists of 6 voltage domains and 24 frequency domains. On the SCC chip, each 2-core tile forms a frequency domain, while every four tiles form a voltage domain. The frequency of each tile can be scaled by writing the register shared by the two cores of the tile. The voltage of each voltage domain can be scaled by writing the voltage controller register (VRC) shared by all the voltage domains [14].

According to Intel's documentation [15], voltage changes in the SCC can happen in the order of milliseconds whereas frequency changes can happen within 20 CPU cycles. To take the latency of frequency and voltage scaling into account, we conducted experiments to measure the latencies accuately. By our measurement, we found that the latency of frequency scaling is nearly unnoticeable, and we just concentrate on the latency of voltage scaling. In the cases that need to wait for voltage scaling, the latency is introduced by double writing on the VRC register. The second write of the VRC register will return when the voltage reaches the desired value.

Fig. 2 shows the measured latency of voltage scaling for two cases, from 0.8V to 0.9V and from 0.9V to 1.1V. For a single voltage domain, the latency of voltage scaling in the two cases is about 30ms. However, when there are multiple voltage domains scaling the voltage simultaneously, the latency will be much larger and linearly increases with the number of domains. Scaling 6 voltage domains simultaneously from 0.8V to 0.9V needs about 195ms.

III. Latency-aware Power Management

A. Phase-based DVFS

Our basic power management approach is phase-based and implemented into a shared virtual memory (SVM)library. The latency-aware DVFS algorithm that we are going to propose will be evaluated based on, but not limited to, this power management approach. In the SVM programming environment, applications are generally partitioned by barriers or locks. Moreover, the code segments across a barrier or a lock operation are likely to perform different computations and exhibit different memory access patterns. Phases in our implementation are defined as stages partitioned by barriers and locks, including the busy waiting stages in barriers and in locks.

Thus, one of the key problems of phase-based DVFS is how to determine the optimal power level for each phase. We designed a model of power versus performance to predict the power and runtime performance of each phase at different power levels. Then we could choose the optimal one. The power model and performance model are based on two indexes, instructions per cycle (IPC) and bus utilization (ratio of bus cycles), which are derived from performance monitor counters (PMC) provided by the CPU. However, as the power model and performance model are not the main work of the paper, we will skip their details in this paper.

Assuming the goal of power management is to minimize the energy delay product (EDP) [16], which is a commonly used index to represent the power efficiency. We can predict the EDP of each phase at a certain power level using the power and performance model as follows:

$$\begin{aligned} EDP(f) &= Energy(f) \cdot Runtime(f) \\ &= Runtime(f) \cdot Power(f) \cdot Runtime(f) \\ &= Power(f) \cdot Runtime(f)^2 \end{aligned} \tag{1}$$

Then we can determine the optimal power level for each phase to achieve the minimal EDP. However, this method does not consider the latency of voltage/frequency scaling. If the power level before the phase starts is different from the predicted optimal power level for this phase, we have to scale the power level firstly, and could introduce some latency and extra power consumption. Thus, the method which does not take latency into account could make wrong decisions.

B. Latency-aware DVFS

Based on our investigation of DVFS latency in Section II, latency of voltage/frequency scaling is non-negligible and must be taken into account of the optimal power level turning.

Besides the latency of voltage/frequency scaling, issuing power requests can also contribute some portions of the overall latency as it will cause state switches between user space and kernel. We denote the latency of scaling up voltage as Δ_s and the latency of issuing a power request as Δ_i. The optimal power level (assuming the optimization is targeted at the least

EDP) for a certain phase, denoted by f_{optm}, should be the frequency value that minimizes the sum of EDP in running the phase and the EDP consumed in voltage/frequency scaling (from current power level f_c to frequency f), denoted by $EDP_{phaseRun(f)}$ and $EDP_{(f_c \to f)}$ respectively. The minimum sum of EDPs could be denoted by $sumEDP_{min}$ as follows:

$$\begin{aligned} & sumEDP_{min} \\ &= \min_{f_{min} \leqslant f \leqslant f_{max}} (EDP_{phaseRun(f)} + EDP_{f_c \to f}) \\ &= \min_{f_{min} \leqslant f \leqslant f_{max}} (p_f (t_f)^2 + \frac{1}{2}(p_{f_c} + p_f)(\Delta_i + \Delta_{s_{f_c \to f}})^2) \end{aligned} \quad (2)$$

As shown in the above formula, the power during voltage and frequency scaling is estimated to be the average power of the powers before and after the scaling .

Thus, the optimal power level f_{optm} can be denoted by $f_{optm} = f \ s.t. \ sumEDP(f) = sumEDP_{min}$.

The power at current power level (p_{f_c}), power (p_f) at frequency level f and runtime (t_f) at f can be estimated using the performance/power model.

Our current design adopts an offline profile-based approach. The optimal power level for each phase, i.e. a pair of voltage and frequency values minimizing $sumEDP$, can be chosen from TABLE III in the profiling run. Then these optimal power settings will be applied to subsequent production runs.

As we reveal in Chapter II, the largest latency for voltage scaling is about 195ms in our tests. On the other hand, since the latency of frequency scaling is in the order of cycles, we can simply ignore this overhead. Thus, we set Δ_s to be 195ms in the above formula. Although the latency for the local core to issue a power request is in the order of thousands of cycles, we set Δ_i to be 2ms in our experiments to avoid the overhead introduced by the state switches.

IV. EXPERIMENT

The evaluation of our latency-aware DVFS solution is conducted on Intel's SCC using the Graph 500 benchmark.

A. The Graph 500 Benchmark

Graph 500 is a project maintaining a list of the most powerful machines designed for data-intensive applications [17]. Researchers observe that data-intensive supercomputing applications are increasingly important for representing today's HPC workloads, but current benchmarks do not provide useful information for evaluating supercomputing systems for data-intensive applications. In order to guide the design of hardware architectures and software systems to support such applications, they proposed the Graph 500 benchmark.

The Graph 500 is a data-intensive program implementing graph algorithms as its core workload. The main work-flow of this benchmark is described in TABLE II. We port the Graph 500 benchmark to a shared virtual memory (SVM) library tailored to the Barrelfish operating system [18] running on the Intel SCC. The SVM library implements an all-software solution to restore the cache coherence such that programmability at the application level won't be compromised. In our experiment, the execution of Graph 500 is divided into 275 phases by barriers and locks, including two times of BFS searching.

TABLE II
ALGORITHM OF GRAPH 500 BENCHMARK

Algorithm 1: Graph 500 benchmark
Step 1: Generate the edge list.
Step 2: Construct a graph from the edge list.
Step 3: Randomly sample 64 unique search keys with degree at least one, not counting self-loops.
Step 4: For each search key:
Step 4.1: Compute the parent array.
Step 4.2: Validate that the parent array is a correct BFS search tree for the given search tree.
Step 5: Compute and output performance information.

TABLE III
SAFE FREQUENCY AND LEAST VOLTAGE TABLE

Frequency Divider	**Frequency (MHz)**	**Least Voltage (V)**	**Least Voltage Level**
2	800	1.1	4
3	533	0.9	2
<=4	$= 1600/Fdiv$	0.8	1

B. Experimental Settings

All the experiments are conducted on 48 cores of the SCC. The problem size of Graph 500 is set as follows: Scale = 18 (262144 vertices) and Edge factor = 16 (4194304 edges). As the temperatures of the SCC board is maintained at around 40, we ignore the impact of the temperature on the power of SCC chip. The clock frequencies of both the mesh network and memory controllers (MCs) of the SCC are fixed at 800MHz during the experiments.

As discussed in Section II.A , the frequency of a frequency domain could be scaled only if the frequency value is "safe" at the current voltage. In SCC platform, the frequency is scaled by a frequency divider (Fdiv) with value from 2 to 16, and the frequency value will be 1600MHz/Fdiv. According to Intel SCC documentation [14], voltage of 0.8V is enough to support 533MHz. However, in the case of booting Barrelfish on 48 cores of SCC, if the initial voltage is 0.8V while the initial frequency is 533MHz, the booting process will always fail at bootstrap of the 25th core. What's more, we found that the system displayed some weird errors when the voltage was scaled down to 0.7V, especially when we launch programs on a large number of cores (e.g. 48 cores). In order to keep the program run safe, we set the least voltage of 533MHz to be 0.9V, and 0.8V for frequency lower than 400MHz (inclusively). The safe frequency and least voltage (SFLV) table we used for the test bed is shown in TABLE III.

Based on the experimental conditions discussed above, we conducted three experiments with different power manage-

TABLE IV
THE RESULT OF RUNTIME, POWER, ENERGY AND EDP OF GRAPH 500 UNDER DIFFERENT EXPERIMENT SETTINGS

	Static800M	Latency-unaware	Latency-aware
Runtime(s)	28.03	40.86	33.38
Power(W)	62.81	33.76	36.41
Energy(J)	1760.31	1379.24	1215.23
EDP(Js)	110567.83	46560.82	44242.83
Runtime*	1.000	1.458	1.191
Power*	1.000	0.537	0.580
Energy*	1.000	0.784	0.690
EDP*	1.000	0.421	0.400

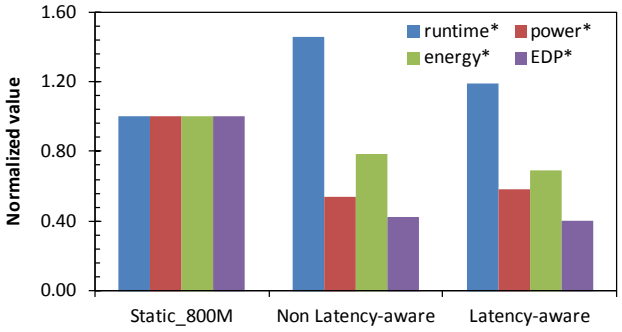

Fig. 3. The result of runtime, power, energy and EDP of Graph 500 under different experiment settings. The values are normalized to the corresponding values in Static 800M

ment policies. They are "Static800M", "Latency-unaware" and "Latency-aware", which are described as follows:

Static800M: This is the baseline experiment using a static power model. All the CPUs' frequencies are set to 800MHz, and the voltages are set to the least value of 1.1V during this experiment. The profile information of Graph 500 benchmark is also derived using this experimental setting.

Latency-unaware: This experiment makes use of our basic methodology of profile-based power management, except the latency-aware DVFS algorithm. Although we do not consider the latency of DVFS in this experiment, we set the latency of issuing a power request (Δ_i discussed in Section III.B) to 2ms to take into account the overhead of state switches.

Latency-aware: Based on the "Latency-unaware" setting, we consider the latency of voltage scaling using the algorithm described in Section III.B. The latency of voltage scaling up is set to be the maximal value of 195ms.

V. RESULTS AND ANALYSIS

A. Results

Under the experimental settings described above, the results of the three experiments are shown in TABLE IV.

In the table, Runtime denotes the entire execution time of Graph 500, including two times of breadth-first search (BFS) for simplicity. Power refers to the chip power of the SCC, including the power of the CPU cores and the network-on-chip (NoC). Energy is the production of power and runtime; and EDP is the product of energy and runtime. We also present the results (the items marked with *) normalized to the corresponding values in "Static800M", referring to . For ease of understanding, we present the normalized values in Fig. 3.

From the experimental results, we can find that both of the two experiments using DVFS achieve great energy and EDP saving compared with the static power model. The basic profile-based power management policy achieves 21.6% energy saving and 57.9% EDP reduction. The policy improved with the latency-aware DVFS algorithm achieve 31.0% energy saving and 60.0% EDP reduction. This implies there is much potential for energy saving in the data-intensive application. Moreover, the system with the latency-aware DVFS algorithm achieves more energy and EDP savings (12.0% and 5.0% respectively) than that without the algorithm, and improves the execution performance by 22.4% as well.

B. Analysis and Discussion

Since our power management approach is profile-based, we present the profile of Graph 500 for dynamic voltage/frequency scaling. The profile includes the optimal power setting for each phase. As it is quite common that the program pattern of the master process (core0) is somehow different from that of the processes running on other cores, we adopt core0's profile and for master process and core1's profile for other cores.

Fig. 4 shows the profile information derived without the latency-aware algorithm. The *x*-asis denotes the phase number, and *y*-asis denotes the optimal frequency (MHz) for corresponding phase. As described in Section IV.B, the least voltage for frequency of 800MHz is 1.1V; for 533MHz it is 0.9V; and for other frequency levels under 533MHz, it is 0.8V. We can find that there are many aggressive DVFS decisions due to the lack of latency awareness. *For example, as pointed by the arrows in the figure, there are many times of frequency scaling among different voltage levels, which lead to voltage scaling with long latency.*

As proposed in Section III, we expect that the latency-aware DVFS algorithm can avoid such aggressive DVFS decisions due to long latency of voltage scaling. Fig. 5 shows the profile of Graph 500 with our latency-aware algorithm enabled. We can see that, there are much fewer DVFS decisions among different voltage levels after applying latency-aware DVFS algorithm. The DVFS decisions are made more "conservatively" in the cases when voltage scaling is needed.

Fig. 6 shows the chip power of the SCC when executing of Graph 500 under different power management policies. Before the 13 seconds of runtime in this figure, the performance and power under different policies are nearly the same. This is because the program is performing compute-intensive edge generation and graph construction in that time range, where opportunity for power saving is lacking (so that high power setting is applied). Beyond this range, the program becomes more data-intensive, so dynamic power management policies

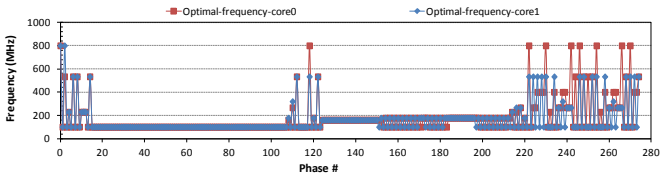

Fig. 4. Profile of Graph 500 WITHOUT latency-aware algorithm. In the figure we can find some aggressive DVFS decisions which will cause power state transitions between different voltage levels

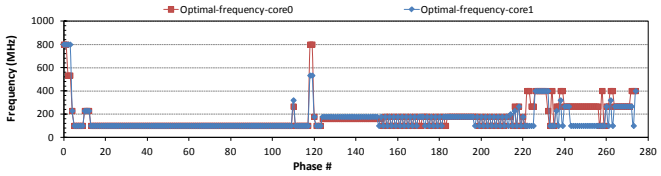

Fig. 5. Profile of Graph 500 WITH latency-aware algorithm

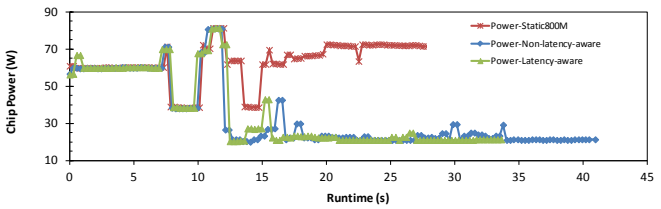

Fig. 6. Compare of chip power during execution under different power management polices

get room to lower the power with little performance lost. Moreover, with the latency-aware DVFS algorithm, dynamic power management avoids much aggressive DVFS scaling which costs long latency. So the latency-aware DVFS algorithm achieves better runtime performance.

VI. Conclusion and Future Work

In this paper, we have investigated the latency characteristics of DVFS, especially on many-core architecture with multiple voltage domains. Based on the study, we proposed a novel latency-aware DVFS algorithm to void aggressive scaling decisions. Experimental evaluations using Graph 500 benchmark were conducted on the Intel SCC platform. The experimental results show that our latency-aware DVFS algorithm achieved 22.4% better performance, 12.0% more energy saving and 5.0% more EDP reduction than a basic profile-based dynamic power management policy.

Currently, the impressive results achieved by our latency-aware DVFS algorithm hinge on an off-line profiling scheme. We plan to apply the concept of latency-awareness to on-line power management policies. We will also present our profile-based power management scheme in detail soon.

Acknowledgement

This research is supported by Hong Kong RGC grant HKU 716712E. Special thanks go to Intel China Center of Parallel Computing (ICCPC) for providing their SCC platform in Wuxi to support this work. The work of this paper also supported by Program for Changjiang Scholars and Innovative Research Team in University (PCSIRT, No. IRT1012) and Aid Program for Science and Technology Innovative Research Team in Higher Educational Institutions of Hunan Province.

References

[1] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced cpu energy," in *Proceeding of the 1st USENIX Conference on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 1994.

[2] K. Govil, E. Chan, and H. Wasserman, "Comparing algorithm for dynamic speed-setting of a low-power cpu," in *Proceedings of the 1st Annual International Conference on Mobile Computing and Networking (MobiCom)*. 215546: ACM, 1995, pp. 13–25.

[3] D. Qingyuan, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini, "Coscale: Coordinating cpu and memory system dvfs in server systems," in *Proceeding of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2012, pp. 143–154.

[4] E. L. Sueur and G. Heiser, "Dynamic voltage and frequency scaling: the laws of diminishing returns," in *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*. 1924921: USENIX Association, 2010, pp. 1–8.

[5] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," in *Proceeding of ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2006, pp. 78–88.

[6] J. Howard, S. Dighe, S. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, and R. V. D. Wijngaart, "A 48-core ia-32 message-passing processor in 45nm cmos using on-die message passing and dvfs for performance and power scaling," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 173–183, 2011.

[7] K. W. Cameron, R. Ge, and X. Feng, "Designing computational clusters for performance and power," *Advances in Computers*, vol. 69, pp. 89–153, 2007.

[8] R. Ye and Q. Xu, "Learning-based power management for multi-core processors via idle period manipulation," in *Proceeding of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2012, pp. 115–120.

[9] N. Ioannou, M. Kauschke, M. Gries, and M. Cintra, "Phase-based application-driven hierarchical power management on the single-chip cloud computer," in *Proceeding of the 20th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2011, work mainly done while N. Ioannou was an intern at Intel Labs.

[10] K. Ma, X. Li, M. Chen, and X. Wang, "Scalable power control for many-core architectures running multi-threaded applications," in *Proceeding of ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2011.

[11] J. Sartori and R. Kumar, "Proactive peak power management for many-core architectures," University of Illinois at Urbana-Champaign, Tech. Rep. CRHC-07-04, 2007.

[12] D. Simone, "Power management in a manycore operating system," Masters Thesis, 2009.

[13] R. David, P. Bogdan, R. Marculescu, and U. Ogras, "Dynamic power management of voltage-frequency island partitioned networks-on-chip using intel's single-chip cloud computer," in *Proceeding of International Symposium on Networks-on-Chip (NOCS)*, 2011, pp. 257–258.

[14] "Scc external architecture specification (eas) (revision 0.94)," Intel Labs, Tech. Rep., 2010.

[15] "The scc programmer's guide (revision 1.0)," Intel Labs, Tech. Rep., 2010.

[16] A. Weissel and F. Bellosa, "Process cruise control: Event-driven clock scaling for dynamic power management," in *Proceeding of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, 2002.

[17] Graph500, "The graph 500 benchmark." [Online]. Available: http://www.graph500.org

[18] A. Baumann, P. Barhamy, P.-E. Dagandz, T. Harrisy, R. Isaacsy, S. Peter, T. Roscoe, A. SchÃijpbach, and A. Singhania, "The multikernel: A new os architecture for scalable multicore systems," in *Proceeding of ACM Symposium on Operating System Principles(SOSP)*, 2009.