Performance evaluation of Graph500 considering CPU-DRAM power shifting

Yuta Kakibuka
Graduate School of Information Science and Electrical Engineering, Kyushu University
yuta.kakibuka@cpc.ait.kyushu-u.ac.jp

Yuichiro Yasui
Institute of Mathematics for Industry, Kyushu University
y-yasui@imi.kyushu-u.ac.jp

Takatsugu Ono
Faculty of Information Science and Electrical Engineering, Kyushu University
takatsugu.ono@cpc.ait.kyushu-u.ac.jp

Katsuki Fujisawa
Institute of Mathematics for Industry, Kyushu University
fujisawa@imi.kyushu-u.ac.jp

Koji Inoue
Faculty of Information Science and Electrical Engineering, Kyushu University
inoue@ait.kyushu-u.ac.jp

1 MOTIVATION

Power wall is one of the most serious issues for post peta-scale high-performance computing. A promising solution to tackle this problem is to effectively manage power resources based on the characteristics of workloads. A representative example is an over-provisioned system, in which the designers are allowed to install hardware components that total peak power overs a given limitation, and a dedicated software ensures that the effective power does not go over the power constraint. In such power constrained computing, the key is to translate the limited power budget into sustained performance effectively. To achieve this goal, assigning the appropriate amount of power budget to each hardware component, or power shifting, is a critical challenge.

In this work, we focus on large-scale graph processing. Graph analysis algorithms are increasing its importance with growing demands of big data applications. Yasui et al. analyze the performance per power consumption of Graph500 [5]. They show an implementation which adjusts direction optimized BFS [1] to NUMA architecture has high energy efficiency. However, as far as we know, no paper has focused on the impact of power shifting between CPU and DRAM on graph performance.

To deeply analyze power and performance, we execute Graph 500 on single node system. Graph500 is a benchmark, which reflects data-intensive workload and published in 2010[2]. Graph500 executes Breadth-First Search (BFS) and Single Source Shortest Path (SSSP) for an undirected graph the number of nodes and edges of which is 2^{scale} and $edgefactor \times 2^{scale}$. In this paper, we evaluate power and performance of a version 1.2 of Graph500 benchmark which only executes BFS and its preprocessing. The metric of the performance is TEPS (traversed edge per second). TEPS is calculated by the following method. Let G' be the component of G, which includes source vertex of BFS, and let t be time spend on BFS. Then, TEPS = $\frac{|E(G')|}{t}$. The performance of Graph500 largely depends on the input size, i.e., the value of scale, and the strategy of power allocation for CPU and DRAM, i.e., how much power resources should be assigned to CPU and DRAM under a given constraint. We reveal the relationship between the power consumption and performance of Graph500 program under power constraints.

Table 1: The specifications of the experimental environment.

CPU	Intel Xeon E5-2620 (6 cores) × 2, TDP = 95[W]
Memory	16GB × 8 (128GB)
OS	CentOS 6.4 64bit (kernel 2.6.32)
Compiler	Intel Compiler Version 14.0.0

2 IMPACT OF POWER SHIFTING ON GRAPH PERFORMANCE

Table 1 summarizes the single node system which we use in experiments. We use Running Average Power Limit (RAPL) [3], to measure and manage power consumption. We use energy efficient Graph500 implementation [4] in this evaluation. This is an optimized implementation for small world graph and NUMA (non-uniformed memory access) architecture.

We evaluate the impact of CPU-DRAM power shifting on the performance of Graph500 under power constraints. In this evaluation, the total power limitations are either 100[W] or 120[W]. The candidate combination of power budgets for CPU and DRAM are set within this limitation. Then, we measure the performance of Graph500 for each candidate under the power constraints. The results are shown in Figure1. The y-axis is the performance of graph500, and the x-axis shows input size. We describe the power allocation as cpu:x[W]-dram:y[W] which means the power x[W] and y[W] are allocated to CPU and DRAM, respectively.

In addition, to consider the impact of power shifting, we also evaluate the impacts of input size on the power consumption of Graph500. Figure 2 shows the power consumption of BFS kernel with changing the number of *scale* from 16 to 28. The x-axis shows input size and the y-axis represents power consumption of CPU, DRAM, and total(CPU+DRAM). The power consumption of CPU increase in the *scale* range from 16 to 23. This caused by the increase of last level cache misses per seconds which is brought by the increase of an input graph size. Among *scale* = 16 to *scale* = 18 and *scale* 24 to *scale* = 28, the DRAM power consumption is the almost same and increasing between *scale* = 18 to *scale* = 24.

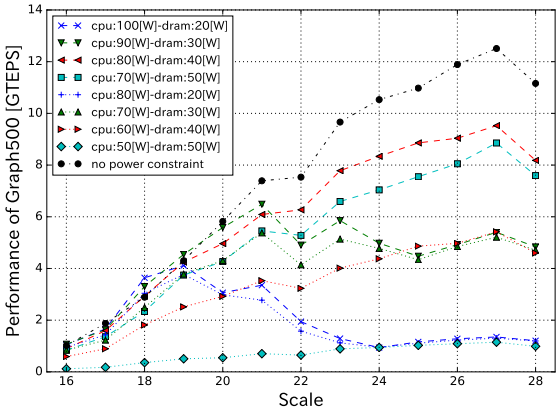

Figure 1: Problem size v.s. performance at 100,120[W] power budget.

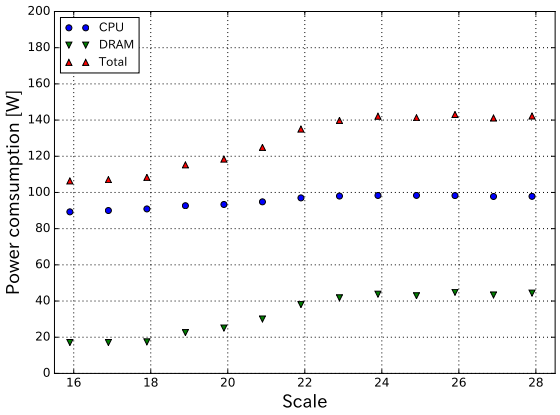

Figure 2: Problem size v.s. power consumption.

We investigate the variation of the power consumption while Grap500 is running. Figure 3 shows the power consumption in each 10 ms at scale = 24. We can find there are several phases of the power consumption. Graph500 generates graphs until 18.2 s and constructs graphs until 60.3 s. The processes of the graph construction are following:

(1) from 18.2[s] to 21.6[s] : counts out-degree of each vertex.
(2) from 21.6[s] to 25.0[s] : sorts by out-degree.
(3) from 25.0[s] to 29.4[s] : makes CSR index.
(4) from 29.4[s] to 36.2[s] : constructs NUMA-aware graph expression.
(5) from 36.2[s] to 47.5[s] : finds multiple edge.
(6) from 47.5[s] to 60.3[s] : sots the edges.

Also, the program executes BFS from 60[s] to the end of the execution.

We can confirm that the graph generation and the graph search have one phase, on the other hand, the graph construction has multiple phases. The power consumption at the graph construction corresponds to the processes. The power consumption from

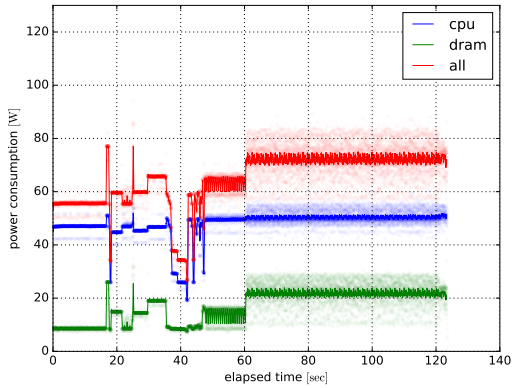

Figure 3: Dynamic behavior of power consumption.

36.2[s] to 47.5[s] can be divided into two phases. In fact, the first phase counts degree of vertexes and the second phase sorts edges.

The performance tends to increase with the increase power budget of CPU in $16 \leq scale \leq 18$. In this range, Figure 2 indicates that the power consumption of CPU and DRAM are about 90[W] and 20[W] under no power constraints. The power consumption of DRAM is less than the fewest power budget for DRAM in this range, so power shiftings to DRAM are enough. As a result, it is the same as there are only power constraints for CPU. Therefore, shifting more power to CPU is a good strategy.

On the other hand, we can achieve high performance by shifting much power budget for DRAM in range $25 \leq scale \leq 28$ except for 50[W] to DRAM. Figure 2 indicates the power consumptions of CPU and DRAM are 100[W] and 40[W], respectively. This means that the power consumptions of CPU and DRAM are capped simultaneously except the power consumption of DRAM is capped with 50[W]. It is important to shift enough power budget to DRAM for memory-intensive application such as Graph500. The performance of cpu:50[W]-dram:50[W] is lower than other results. We investigate the power consumption of CPU and DRAM at cpu:50[W]-dram:50[W], CPU and DRAM consumes 50[W] and 17.9[W], respectively. The reason is that the power constraint for CPU forced to decrease the frequency of CPU, hence the number of memory access per time is also decreased.

ACKNOWLEDGMENT

This work was partially supported by the Japan Science and Technology Agency under Core Research for Evolutional Science and Technology.

REFERENCES

[1] Scott Beamer, Krste Asanović, and David Patterson. 2012. Direction-optimizing Breadth-first Search. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 12:1–12:10.
[2] Graph 500 Steering Committee. [n. d.]. Graph 500 Benchmark 1 ("Search"). ([n. d.]). www.graph500.org/?page_id=12
[3] Efraim Rotem, Alon Naveh, Avinash Ananthakrishnan, Eliezer Weissmann, and Doron Rajwan. 2012. Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge. *IEEE Micro* 32, 2 (March 2012), 20–27.
[4] Yuichiro Yasui, Katsuki Fujisawa, Eng Lim Goh, John Baron, Atsushi Sugiura, and Takashi Uchiyama. 2016. NUMA-aware Scalable Graph Traversal on SGI

UV Systems. In *Proceedings of the ACM Workshop on High Performance Graph Processing*. 19–26.

[5] Yuichiro Yasui, Katsuki Fujisawa, and Yukinori Sato. 2014. Fast and Energy-efficient Breadth-First Search on a Single NUMA System. In *Proceedings of the 29th International Conference on Supercomputing*. 365–381.