CS698 Project Report

Rohith Mukku (14402)

November 2017

1 Introduction

The aim of this project is to study the performance of data center workloads on the cache hierarchy. To represent data center workloads we have selected applications from the GAP benchmark suite[1].

2 GAP Benchmark Suite

Graph Algorithm Platform (GAP) Benchmark Suite is a benchmark suite for graph algorithms and their applications. Graph algorithms and their application are currently gaining renewed interest, partly because of social networking and its analysis[2]. There hasn't been any standard for these applications, so Graph500 was developed for this. The biggest shortcomings of Graph 500 are its focus on one kernel (breadth-first search) and only one synthetic input graph topology, hence, University of California Berkeley introduced GAP to help standardize graph processing evaluations for easier comparisons of research results and improvements[1]. They selected six kernels based on how commonly they are used. These kernels are:

- **Breadth-First Search (BFS):** BFS is a traversal order starting from a source vertex. BFS traverses all vertices at the current depth (distance from the source vertex) before moving onto the next depth.
- **Single-Source Shortest Paths (SSSP):** SSSP computes the distances of the shortest paths from given source vertex to every other reachable vertex.
- **PageRank (PR):** PR computes the PageRank score for all vertices in the graph.
- **Connected Components (CC):** CC labels all vertices by their connected component and each connected component is assigned its own unique label.
- **Betweenness Centrality (BC):** BC approximates the betweenness centrality score for all vertices in the graph by only computing the shortest paths from a subset of the vertices.
- **Triangle Counting (TC):** TC computes the total number of triangles (triangle to be three vertices that are directly connected to each other) in a graph.

The input graphs for the benchmark:

- **Twitter** is an example of a social network topology. It is a directed graph with number of vertices, |V|=61.6M, and edges, |E|=1,468.4M.
- **Web** is a web-crawl of the .sk domain (Internet country code top-level domain (ccTLD) for Slovakia). It is a directed graph with number of vertices, |V|=61.6M, and edges, |E|=1,468.4M.
- **Road** is the distances of all of the roads in the USA. It is a directed graph with number of vertices, |V|=23.9M, and edges, |E|=58.3M.

- **Kron** uses the Kronecker synthetic graph generator[3]. It is an undirected graph with number of vertices, |V|=134.2M, and edges, |E|=2,111.6M.
- **Urand** is synthetically generated by the Erdős–Réyni model (Uniform Random). It is an undirected graph with number of vertices, |V|=134.2M, and edges, |E|=2,147.4M.

2.1 Motivation

As there is a surge of interest in graph algorithms and their applications in social networks, importance of their analysis is growing[4][5]. Also, as the present cache replacement policies (LRU, Hawkeye, SHiP) are not giving a good performance (mainly because of random accesses)[6], there is a need for understanding the accesses made by the graph applications.

2.2 Resources & Methodologies

GAP Benchmark Suite has been implemented and maintained in a GitHub repository (`https://github.com/sbeamer/gapbs`). The graph inputs can be loaded in various formats as given in the mentioned link. We looked at serialized pre-built graphs (unweighted and weighted), or to put it in a better way:

- BC, BFS, CC, PR kernels were run with unweighted directed input graphs Twitter, Web.
- SSSP kernel was run with weighted directed input graphs Twitter, Web.
- TC kernel was run with unweighted undirected input graphs Kron, Urand.

Our analysis on traces of the above mentioned cases were done with the help of ChampSim (`https://github.com/ChampSim/ChampSim`). The traces were generated with the help of PIN tool tracer.

Trace Generation

The in-built tracer generates a different trace structure format compared to that used by ChampSim. Hence, we had to modify the PIN tool tracer to generate the required trace structure format. The modified tracer generates 4 threads of a graph application with an input graph (24 such combinations as mentioned above).

Parameters/Assumptions

The ChampSim simulated the generate traces with:

- Address space IDs: asid[0] = 4, asid[1] = 16
- Number of warm-up instructions: 250M
- Number of simulation instructions: 250M
- Cache Replacement Policies: LRU, Hawkeye, SHiP

Statistics

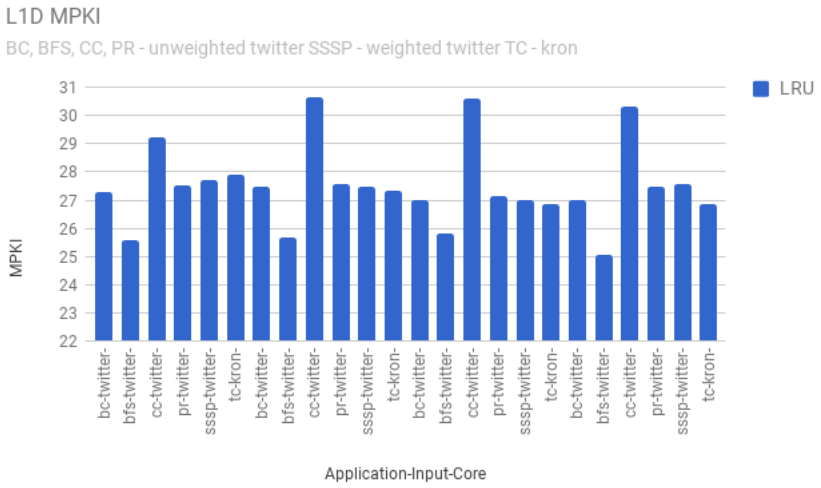
Python scripts were used to collect the statistics produced by the simulator.

2.3 Analysis

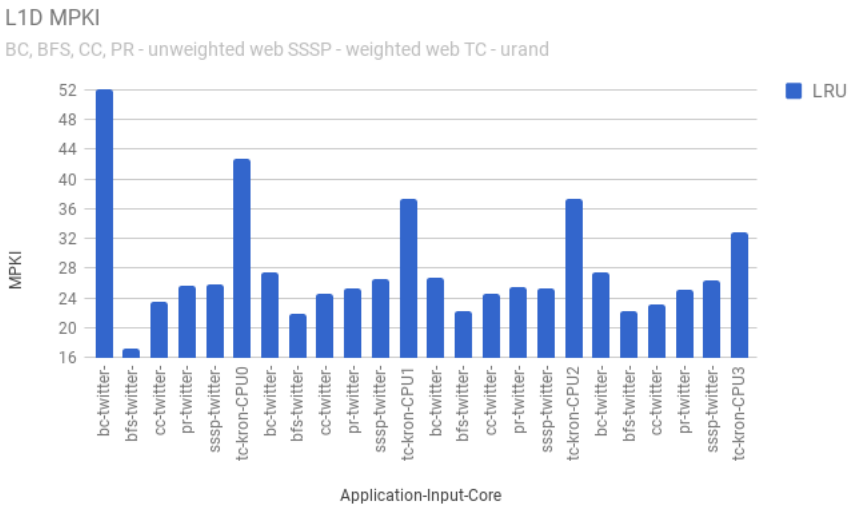
Below are the obtained results when ChampSim was simulated on the 24 combinations. As there are many combinations, we divide them into 2 sets:

- **First Set:** BC, BFS, CC, PR, SSSP with Twitter and TC with Kron
- **Second Set:** BC, BFS, CC, PR, SSSP with Web and TC with Urand

For L1I cache, the results were not that insightful compared to the other caches. For the lower level caches (L1D and L2C), as the replacement policies are same, we neglected the results obtained from simulation of other replacement policies as they were for L3C.

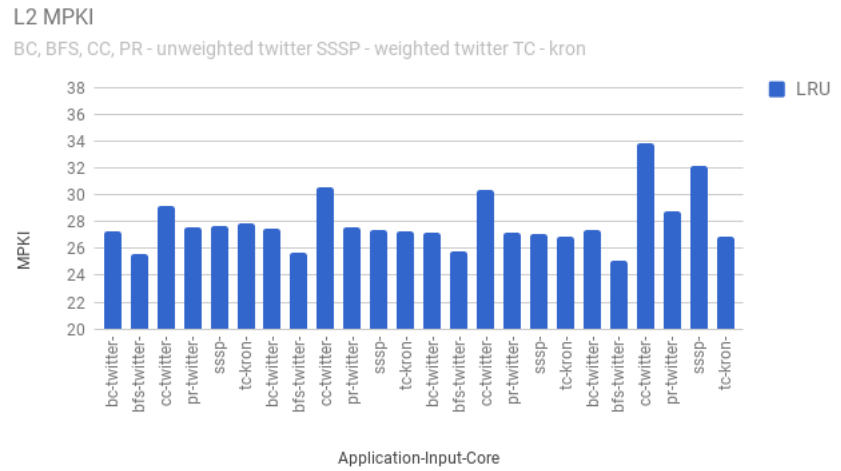


(a) First Set

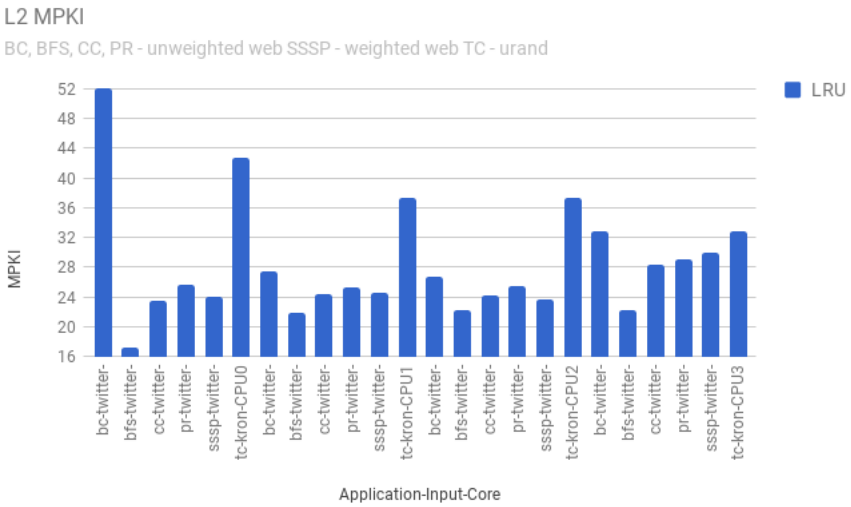


(b) Second Set

Figure 1: L1D MPKI

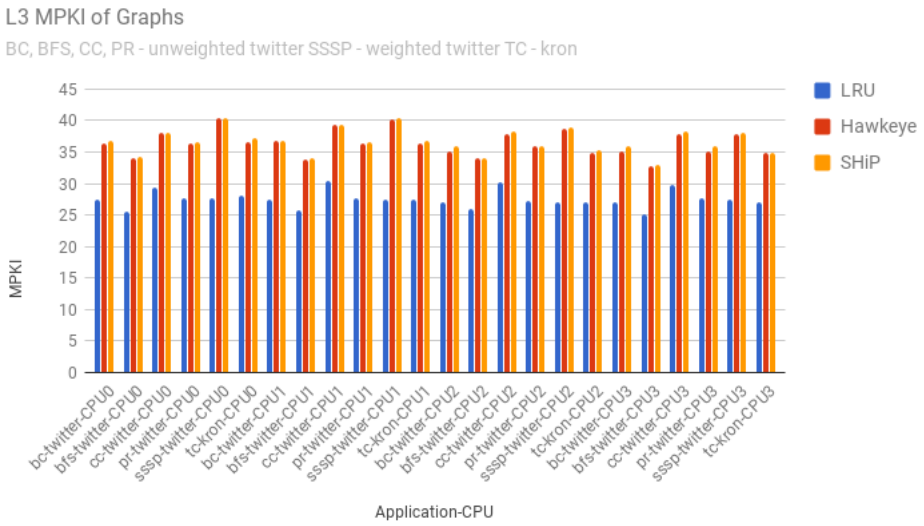


(a) First Set

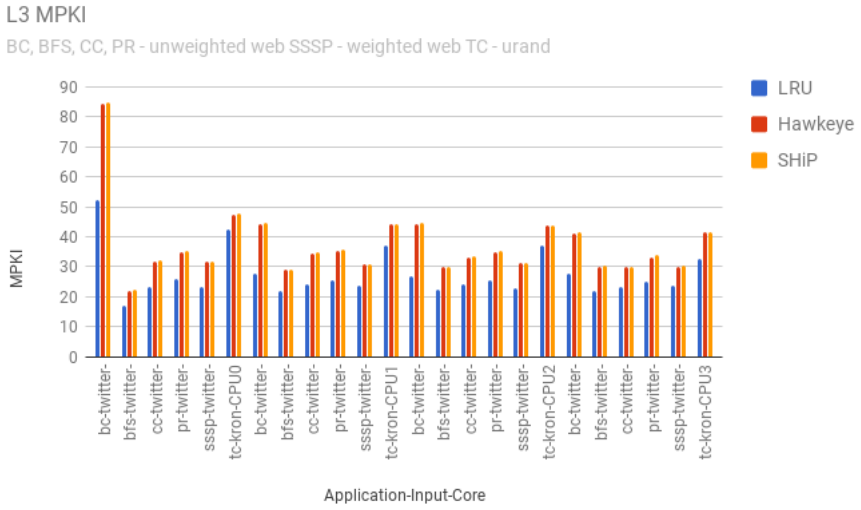


(b) Second Set

Figure 2: L2C MPKI



(a) First Set



(b) Second Set

Figure 3: LLC MPKI

2.4 Future Work

We could check the traces with opt and compare them with the obtained results. Moreover, this report does work only on the above 24 combinations. This can be extended to other combinations, thus, can use the results in studying the cache replacement policies in graph applications.

References

[1] S. Beamer, K. Asanovic, and D. A. Patterson, "The GAP benchmark suite," *CoRR*, vol. abs/1508.03619, 2015.

[2] T. Wang, Y. Chen, Z. Zhang, T. Xu, L. Jin, P. Hui, B. Deng, and X. Li, "Understanding graph sampling algorithms for social network analysis.," in *ICDCS Workshops*, pp. 123–128, IEEE Computer Society, 2011.

[3] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *J. Mach. Learn. Res.*, vol. 11, pp. 985–1042, Mar. 2010.

[4] W. Aiello, F. Chung, and L. Lu, "A random graph model for power law graphs," *Experimental Math*, vol. 10, pp. 53–66, 2000.

[5] S. Beamer, *Understanding and Improving Graph Algorithm Performance*. PhD thesis, EECS Department, University of California, Berkeley, Sep 2016.

[6] A. Mukkara, N. Beckmann, and D. Sanchez, "Cache-Guided Scheduling: Exploiting Caches to Maximize Locality in Graph Processing," in *1st International Workshop on Architectures for Graph Processing (AGP 2017), held in conjuntion with ISCA 2017*, June 2017.