Understanding Query Complexity and its Implications for Energy-Efficient Web Search

Emily Bragg,[1] Marisabel Guevara,[2] and Benjamin C. Lee[2]
[1]Georgia Institute of Technology, [2]Duke University
ebragg09@gatech.edu, mg@cs.duke.edu, benjamin.c.lee@duke.edu

Today's largest datacenters dissipate megawatts of power. Efficiency is rapidly becoming the primary determinant of datacenter capability. To understand microarchitectural factors that affect efficiency, we must study datacenter workloads.

Most studies treat the workload as a large, monolithic piece of software. But a workload is often comprised of many, diverse software tasks. For example, a web search engine executes many individual queries. There is a vast difference between the complexity of searching for a single term and that of searching for a collection of related terms interspersed with Boolean and wildcard operators, which are increasingly common in search engines [1,6].

Methodology. We use Nutch to crawl Wikipedia and gather a set of 50,065 documents across a wide range of topics. We then use Solr and Lucene to search and index text.

We implement a query generator that produces queries with a specified number of terms and Boolean operators. The algorithm starts with an initial word and finds other words that commonly occur in its proximity to add a term to the query. Recursively invoking this algorithm produces a query of the desired length. Finally, the algorithm combines multiple query terms using various Boolean operators.

We deploy web search in cycle-accurate simulation for a detailed analysis of microarchitectural activity. In contrast, a recent study of search query complexity considers Bing on physical Xeons and Atoms [4]. This study of Bing anonymizes query types whereas we provide transparent insight.

We use MARSSx86, a cycle-accurate processor simulator that models x86-64 architectures [3]. McPAT provides power estimates for the processor cores [2]. We link the simulator to DRAMSIM2, which models memory performance and power [5]. We focus on the differences between a variety of in-order (IO) and out-of-order (OOO) cores.

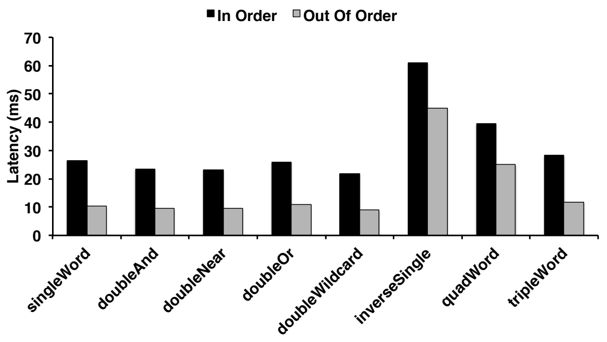

Figure 1. Latency Measurements: OOO cores are most beneficial for simpler queries (e.g., singleWord). In comparison, benefits are modest for more complex queries (e.g., inverseSingle).

Average Latency. As shown in Figure 1, latency clearly depends on query complexity. Single word queries ('singleWord') complete quickly and latencies increase with query length. The query that finds all documents not matching a specified term ('inverseSingle') is most expensive as it must search and return the largest number of relevant pages.

Remarkably, OOO benefits are most prominent for the simplest queries. Latency for singleWord falls by more than 50%. In contrast, latency reductions for inverseSingle are far more modest. Complex queries are slow regardless of hardware choice. If OOO cores are scarce, they should be allocated to simple queries to maximize system throughput.

Latency Variance. Dynamic instruction scheduling accounts for much of the latency reduction. Further increasing superscalar width and increasing cache sizes have negligible effects. Hence, many OOO cores exhibit similar performance. In contrast, IO cores exhibit higher latency variance since other microarchitectural parameters play a larger role in determining performance. Thus, the risk of designing a poor OOO core is far lower than that of designing a poor IO one.

Latency Cut-offs. Suppose we aim to complete all queries in less than 30ms. Only a few, power-intensive cores (>23W) can deliver this performance for complex inverseSingle queries. At the same time, low-power cores (2.5W) are suitable for simple, singleWord queries.

By steering complex queries to OOO cores and simple queries to IO ones, a system might meet the 30ms target. Without the ability to differentiate query types, a system would need to rely solely on OOO cores to guarantee latency for rare, complex queries, a highly inefficient approach. A system would rather use IO cores. More of them can fit within a given power budget, thereby increasing system throughput.

Conclusion. Microarchitectural simulations show that performance and power trade-offs differ significantly depending on query complexity. While most, common queries can execute on small cores and meet service-level-agreements, complex queries on such cores are likely to violate performance targets. If complexity could be determined when a query first enters the system, steering rules could be devised to ensure performance. Our methodological contributions give a starting-point for designing heterogeneous hardware and managing software-to-hardware mapping.

Acknowledgements. This work is supported by NSF grant CCF-1149252 and by STARnet, a Semiconductor Research Corporation program, sponsored by MARCO and DARPA.

REFERENCES

[1] B. Jansen and A. Spink, "How are we searching the World Wide Web? A comparison of nine search engine transaction logs". Info. Processing & Management, Vol. 42, Issue 1, 2006.
[2] S. Li et al., "McPAT," MICRO, 2009.
[3] A. Patel et al., "MARSSx86," DAC, 2011.
[4] V. Reddi et al., "Web Search Using Mobile Cores: Quantifiying and Mitigating the Price of Efficiency," ISCA, 2010.
[5] P. Rosenfeld et al., "DRAMSim2," CAL, Vol. 10, No. 1, 2011.
[6] C. Silverstein et al. "Analysis of a very large web search engine query log," ACM SIGIR Forum, 1999.