DFS: A Simple to Write Yet Difficult to Execute Benchmark

Richard Murphy† Jonathan Berry† William McLendon† Bruce Hendrickson†
Douglas Gregor‡ Andrew Lumsdaine‡

Sandia National Laboratories†[*]
PO Box 5800, MS-1110
Albuquerque, NM 87185-1110

Indiana University‡
CSE Dept, 215 Lindley Hall
Bloomington, IN 47405

{rcmurph,jberry,mcwclen,bahendr}@sandia.gov,
{dgregor,lums}@cs.indiana.edu

1. Introduction and Motivation

Many emerging applications are built upon large, unstructured datasets that exhibit highly irregular (or even nearly random) memory access patterns. Examples include informatics applications, and other problems that are often represented by unstructured graph-based data structures. It is well known that these applications are challenging for conventional architectures to execute (either serially or in parallel). The Depth First Search (DFS) benchmark proposed in this work uses the Boost Graph Library to perform a depth-first search on a large power-law graph, representing "small world" phenomena. The graph in question exhibits a small average distance between any two vertices, a small diameter, and has a few high-degree vertices with a large number of low-degree vertices. Graphs such as this appear in many fields, including networking, biology, social networks, and data mining. Many of these applications are of critical importance to researchers, and the challenge of executing them on conventional machines increases as the graph size grows.

The benchmark proposed in this work is used as the basis for many fundamental algorithms in graph theory, is critical to several emerging applications, is memory intensive, and exhibits poor performance on conventional machines. Section 2 quantitatively demonstrates the memory characteristics of the benchmark in an architecture independent fashion, showing that it is extremely memory intensive. Section 3 describes the execution phases of the benchmark. And Section 4 presents the conclusions.

[*]Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

2. Unique Characteristics of the Benchmark

The von Neumann bottleneck dictates that application performance will be primarily defined by the performance of the memory system. In fact, the increasing relative memory access time, or *memory wall*[5] proves especially challenging for applications that do not exhibit cache-friendly behavior. Typically this behavior can be described by the application's *temporal locality*, or propensity to reuse data from memory over time, and its *spatial locality*, or use of data in memory "near" data that has already been used. This work further characterizes the application's behavior by its *data intensiveness*, or the total amount of unique data consumed by the application. Formally, the measurements presented in this work are defined as:

1. **Temporal Locality**: the miss rate of a fully associative 64-kb (L1 size) cache with 4-byte (machine word size) blocks. This was measured over a 1 billion instruction interval of the program's inner loop.
2. **Spatial Locality**: the average fraction of a 64-byte cache line that is used over a 1,000 instruction window in the same 1 billion instruction interval.
3. **Data Intensiveness**: the total number of unique bytes consumed in the same 1 billion instruction interval.

A further description of the measurements and methodology, as well as a wider range of data points can be found in [3, 2].

Figure 1 shows the memory characteristics described above. The graph depicts temporal vs. spatial locality, with the relative area of the dots representing relative data intensiveness. The search in this work is shown in comparison to several key benchmarks:

1. **LINPACK**: the Supercomputer benchmark used to generate the Top 500 supercomputer list.

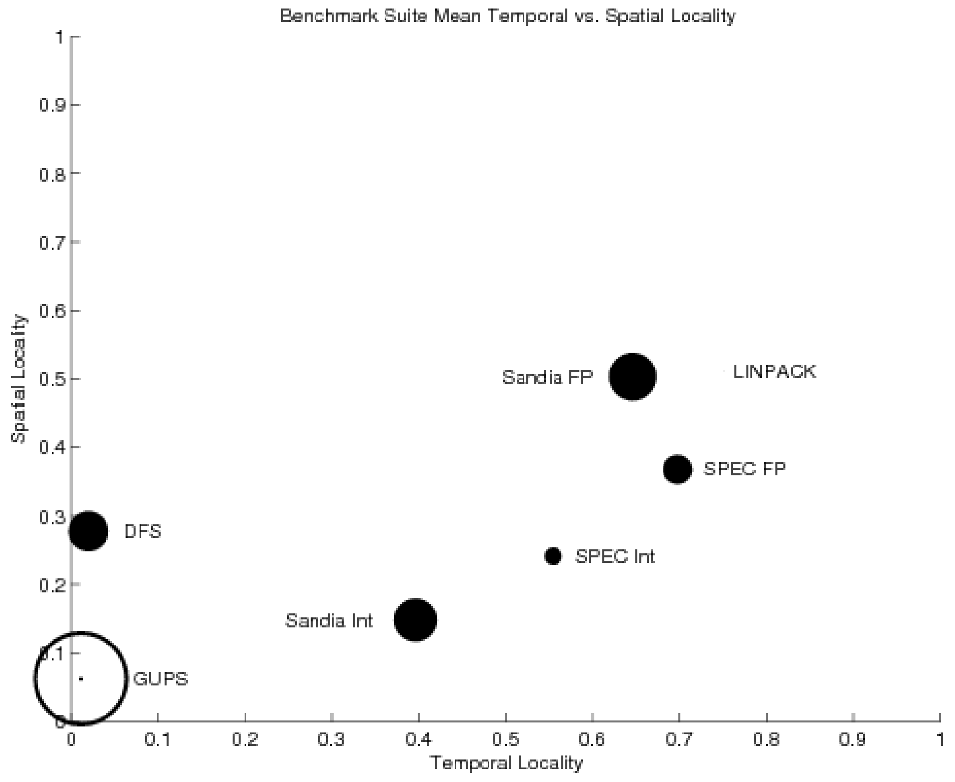

Figure 1. Benchmark Temporal Locality, Spatial Locality, and Data Intensiveness.

2. **SPEC CPU 2000**: represented by **SPEC FP**, the floating point suite describing primarily scientific workloads, and the **SPEC Int** integer suite that is representative of workstation workloads. This has long been the primary suite studied by computer architects[1].
3. **Sandia FP**: a suite of real scientific applications run at supercomputer scale at Sandia National Laboratories[2].
4. **Sandia Int**: a set of emerging integer applications for supercomputer-scale problems, including codes in the graph theory, biology, and discrete math[2].
5. **GUPS**: the Giga-Updates Per Second benchmark (or the RandomAccess HPC Challenge Benchmark) that continuously updates a random location in a large memory array.

The DFS benchmark proposed here exhibits very low spatial and temporal locality, as well as a relatively large data set. In comparison to GUPS, which is artificially constructed to exhibit the lowest spatial and temporal locality, and greatest data intensiveness, DFS exhibits 1.75 times the temporal locality, 4.4 times the spatial locality, and only 19% the data set size. However, the DFS benchmark represents a fundamental application in graph theory, and, unlike GUPS, it can be optimized algorithmically for a particular architecture. DFS exhibits significantly less temporal locality than any other real application. The next closest is the Sandia Integer suite, which has nearly 20 times the temporal locality. Both integer suites have less spatial locality (47% less for the Sandia suite and 13% less for the SPEC). The floating point suites exhibit significantly higher spatial and temporal locality than does DFS.

The most surprising measure, however, is data intensiveness. DFS exhibits only 19% the data intensiveness of GUPS, but 1.86 times that of SPEC FP, and over 5 times that of the SPEC integer suite. The real applications in the Sandia suite are more data intensive than DFS (41% more for the floating point suite and 16% more for the integer).

The most surprising comparison is between DFS and LINPACK, the most prevalent benchmark for supercomputing. LINPACK exhibits nearly 38 times the temporal locality, and over 1.8 times the spatial locality. The data intensiveness, however, is most shocking. DFS consumes nearly $6,900$ times the amount of unique data as compared to LINPACK.

Because the DFS application exhibits a nearly 100% temporal miss rate, and consumes (on average) two unique machine words out of each cache line, cache-based architectures will exhibit low efficiency in the execution of the benchmark.

Classical benchmarks (LINPACK and the SPEC suite) are the most dissimilar to the proposed DFS benchmark. LINPACK's unique data set is so small in comparison to the DFS code that the dot representing it is almost impossible to see. The SPEC suite, particularly SPEC Int, are also very small by comparison. And each of the three exhibits significantly higher spatial and temporal locality. SPEC Int is the closest conventional benchmark to the proposed code, and has nearly twice the temporal locality and nearly three times the spatial locality.

3. DFS Benchmark Execution

The DFS benchmark first constructs a graph of 4 million vertices using the Power Law Out Degree (PLOD) algorithm[4] available in the Boost Graph Library. On a typical workstation, the graph consumes between 250 and 500 MB. While the generation of the graph may prove an interesting benchmark on its own, the core of the DFS benchmark is to perform a Depth First Search on the generated graph. This is repeated several times for timing.

4. Conclusions

The benchmark described in this paper represents an emerging class of applications based on large, unstructured graphs. These codes are unique for the high performance computing community in that they are significantly more data intensive than traditional scientific applications (that are already considered data intensive). This paper has quantitatively demonstrated the difference between the DFS benchmark and several other application suites, including the ubiquitous SPEC, in an architecture independent fashion. The results show that the benchmark is unique compared to other real high performance computing applications, as well as the predominant conventional benchmark suites. Clearly the analysis of large graphs provides a unique challenge to conventional architectures. The benchmark is important because of its emerging use within informatics and uniquely challenging memory properties. It can serve as a real application that fills the critical memory intensive component of a benchmark suite.

References

[1] Daniel Citron, John Hennessy, David Patterson, and Gurindar Sohi. The use and abuse of SPEC: An ISCA panel. *IEEE Mico*, 23(4):73–77, July-August 2003.

[2] Richard C. Murphy. *Traveling Threads: A New Multithreaded Execution Model*. Ph.D. Dissertation, University of Notre Dame, 2006.

[3] Richard C. Murphy, Arun Rodrigues, Peter Kogge, , and Keith Underwood. The implications of working set analysis on supercomputing memory hierarchy design. In *The 2005 International Conference on Supercomputing*.

[4] C. Palmer and J. Steffan. Generating network topologies that obey power laws. In *GLOBECOM*, November, 2000.

[5] Wm. A. Wulf and Sally A. McKee. Hitting the memory wall: Implications of the obvious. *Computer Architecture News*, 23(1):20–24, 1995.