SAND2008-5011C

The PageRank Derby

Karen Devine, Jonathan Berry and Steve Plimpton
Scalable Algorithms Department 1416, Sandia National Laboratories

Introduction: Massively multithreaded parallel architectures such as Cray's MTA and XMT are proving to be highly effective for graph analysis algorithms. By providing uniform memory access times for data with much irregularity and little locality, these machines have demonstrated excellent scalability for a wide range of graph-based algorithms. However, no apples-to-apples comparisons between these architectures and our traditional distributed memory architectures had been performed using realistic input data.

We make the first such comparisons using Google's PageRank method as the algorithmic kernel and synthetic datasets with power-law vertex degree distributions. One might expect PageRank to favor traditional distributed-memory architectures, as PageRank reduces to matrix-vector multiplication with floating point arithmetic. However, the choice of data clearly favors the multithreaded architectures. By comparing PageRank performance using both distributed memory and massively multithreaded paradigms, we seek to assess the effect of algorithm and data on architecture choice.

PageRank: PageRank [5] computes the importance of web pages based on the importance of pages that link to them. The importance of page s increases if s is pointed to by other important pages. The share of importance that s receives from page t is inversely proportional to the number of pages that t links to.

PageRank models the web as a directed graph $G(V, E)$, with each vertex $v \in V$ representing a web page and each edge $e_{ij} \in E$ representing a hyperlink from v_i to v_j. The probability of moving from v_i to another vertex v_j is $\alpha/d_{out}(v_i) + (1 - \alpha)/|V|$, where α is a user-defined parameter (usually 0.8-0.9), $d_{out}(v)$ is the outdegree of vertex v, and $|V|$ is the cardinality of V. The first term represents the probability of following a given link on page v_i; the second represents the probability of moving to a random page. For pages with no outlinks, the first term is $\alpha/|V|$, indicating equal likelihood to move to any other page.

MultiThreaded Graph Library Implementation: In the MTGL [1] implementation of PageRank, rank propagation is accomplished through adjacency list traversal. Our results were obtained with an underlying compressed sparse row data structure, but the same code would run on other graph representations. A key requirement for scaling is that the code must be written so that a single thread spawns the loop that processes all in-neighbors of a given vertex. This enables the compiler to generate hotspot-free code.

Distributed-Memory Implementation: The distributed-memory implementations of PageRank represent the graph as a matrix A [4], with matrix entries $A_{ij} = \alpha/d_{out}(v_i)$ if vertex v_i links to v_j. The PageRank algorithm, then, is simply a power-method iteration in which the dominating computation is matrix-vector multiplication $Ax = y$, where x is the PageRank vector from the previous iteration. Terms representing random links could conceptually be included in A, but they are more efficiently handled as adjustments to y. Rows or non-zeros of A are uniquely assigned to processors, along with the associated entries of the PageRank vector x. Interprocessor communication is needed to gather x values for matrix-vector multiplication and to sum partial products into the y vector. Most communication is point-to-point communication, but some global communication is needed for computing residuals and norms of x and y.

Experimental Data: Our experimental data are R-MAT graphs [2]. R-MAT graphs are recursively generated graphs with power-law degree distributions. They are commonly used to represent web and social networks. They are generated using only four parameters a, b, c, and d, which represent the probability of an edge being generated in one of four recursive quadrants of a matrix representing the graph. We used two different R-MAT data sets, each with average vertex degree of eight and 25 R-MAT levels (i.e., 2^{25} vertices). The "nice" data set uses R-MAT parameters $a = 0.45$, $b = 0.15$, $c = 0.15$ and $d = 0.25$; the resulting maximum vertex degree is 1108. The "nasty" data set uses R-MAT parameters $a = 0.57$, $b = 0.19$, $c = 0.19$, and $d = 0.05$; the resulting maximum vertex degree is $230,207$.

Results: We ran our experiments on Cray's XMT and MTA as well as Sandia's RedStorm and a small cluster called Odin. Cray MTA and XMT both support a global address space, hiding memory latency by using 128 instruction streams per processor. The MTA has 220 MHz processors and a modified Cayley network; the XMT has 500 MHz processors and a 3D-Torus network. RedStorm is a distributed memory parallel supercomputer with two Dual-Core AMD 64-bit 2.4 GHz Opteron processors per node, 2GB memory per node, and 9.6 GB/s link bandwidth. Odin has two AMD Opteron 2.2GHz processors and 4 GB of RAM

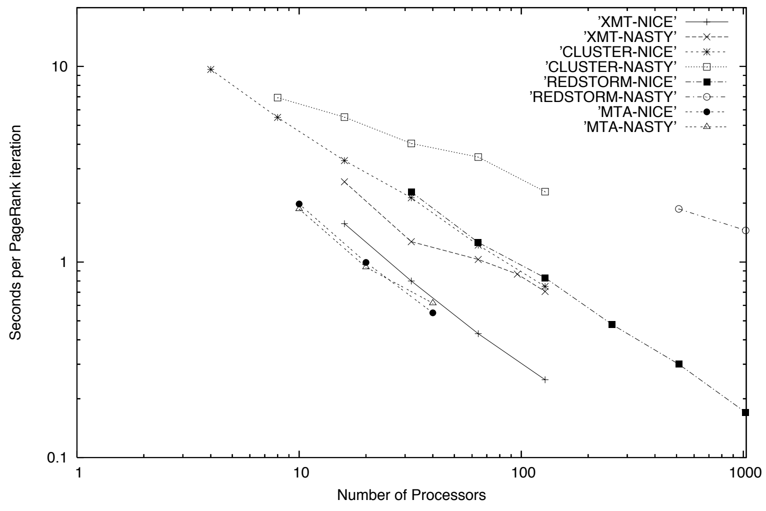

Figure 1: Preliminary PageRank Derby results.

per node; nodes are connected with a Myrinet network.

For each data set, we computed page rank on each architecture; we present the time for one pagerank iteration in Figure 1. For the nice data set, all implementations scaled well, with the XMT and MTA taking less time per iteration than the distributed memory machines. With the nasty data set, load imbalance limits the scalability of the distributed memory implementations; the maximum number of non-zeros of A assigned to a processor is an order of magnitude greater than the average. Load-balancing techniques like those in Zoltan [3] have the potential to improve the distributed memory performance. The XMT also struggles somewhat with the nasty data set. Previous experiments on the MTA-2 led us to expect little or no change in the scaling plots. The hitch in the XMT's nasty data plot was an unwelcome surprise that has yet to be explained. However, we note that the XMT's strong scaling on these data from 64 to 128 processors resumes a near-optimal slope. Parallelizing compiler artifacts may eventually explain the hitch.

Conclusions: Our results are preliminary. We have yet to run on exactly the same instances, and there are artifacts of each implementation that need further exploration. However, several themes are emerging.

- **Distributed memory clusters *can* process large, unstructured datasets in certain contexts.** Our simple algorithm applied to nasty data found strong scalability through at least 1000 processors.
- **Massively multithreaded architectures *can* outperform microprocessor-based clusters, even with floating point-intensive algorithms on datasets that are only mildly unbalanced**. The MTA/XMT machines have been spoken of only as boutique, pointer chasing vehicles. Before this study, there was little reason to expect 500 MHz processors without floating point units to compete with modern microprocessors in any floating point context. We explore sparse matrix multiplication on mildly unbalanced data, however, and see this surprising result in our nice data plots.
- **Realistic informatics data require programmer intervention for distributed memory architectures, but not for massively multithreaded architectures** The slopes of the strong scaling plots suggest that without load balancing, 10,000 Red Storm processors might not match 32 XMT processors on realistic informatics data. The effectiveness of load balancing remains to be measured.

[1] J. W. Berry, B. Hendrickson, S. Kahan, and P. Konecny. Software and algorithms for graph queries on multithreaded architectures. In *Proceedings of the 21st International Parallel and Distributed Processing Symposium*, March 2007.

[2] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-mat: A recursive model for graph mining. In *Fourth SIAM International Conference on Data Mining*, April 2004.

[3] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, 4(2):90–97, 2002.

[4] A. N. Langville and C. D. Meyer. A survey of eigenvector methods for web information retrieval. *The SIAM Review*, 47(1):135–161, 2005.

[5] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.