3-D Graph Processor

William S. Song, Jeremy Kepner, Huy T. Nguyen, Joshua I. Kramer, Vitaliy Gleyzer, James R. Mann, Albert H. Horst, Larry L. Retherford, Robert A. Bond, Nadya T. Bliss, Eric I. Robinson, Sanjeev Mohindra, Julie Mullen
Lincoln Laboratory, Massachusetts Institute Technology, Lexington, MA 02420
{song, kepner, hnguyen, joshua.kramer, vgleyzer, jmann, ahorst, retherford, rbond, nt, erobinson, smohindra, jsm}
@ll.mit.edu

Introduction

Graph algorithms are used for numerous database applications such as analysis of financial transactions, social networking patterns, and internet data. While graph algorithms can work well with moderate size databases, processors often have difficulty providing sufficient throughput when the databases are large. This is because the processor architectures are poorly matched to the graph computational flow. For example, most modern processors utilize cache based memory in order to take advantage of highly localized memory access patterns. However, memory access patterns associated with graph processing are often random in nature and can result in high cache miss rates. In addition, graph algorithms require significant overhead computation for dealing with indices of vertices and edges. Figure 1 shows example computational throughput differences between conventional processing and graph processing. Shown in blue is a matrix multiply kernel running on the PowerPC and Intel Zeon processors. In contrast, shown in red is a graph edge traversal kernel running on the identical processors. The graph computation throughput is approximately 1,000 times lower, which is consistent with typical application codes.

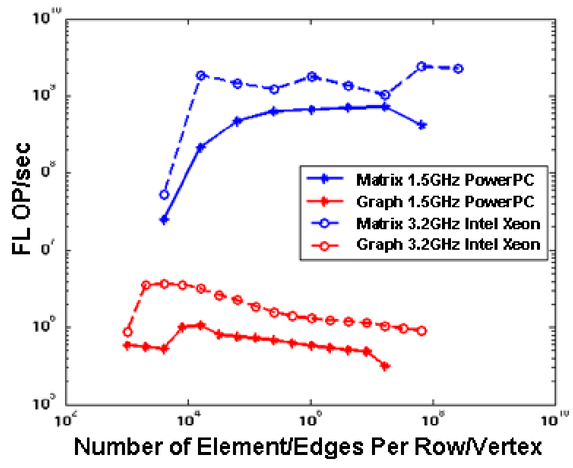

Figure 1: Computational Throughput Differences between Conventional and Graph Processing.

The multi-core processors can help the graph algorithms run faster by providing higher computational throughput. However, because commercial multi-core processors tend to rely on cache based memory architecture, the performance gain tends to be limited. Networked parallel processors can also accelerate graph processing by distributing computation over multiple processors. However, the speed up factor quickly levels off with a small number of processors due to immense inter-processor communication requirements generated by non-localized database structures.

This work was sponsored by DARPA under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

3-D Graph Processor Architecture

In order to achieve significantly higher graph computation performance, MIT Lincoln Laboratory has developed an advanced multiprocessor architecture that is optimized for graph algorithms analyzing large databases. The processor instruction set is based on sparse matrix algebra operations. The graph operations are first converted into sparse matrix operations before they are run on the processor [1]. Although the computations are equivalent in both approaches, the sparse matrix approach simplifies both the instruction set and the multiprocessor architecture design.

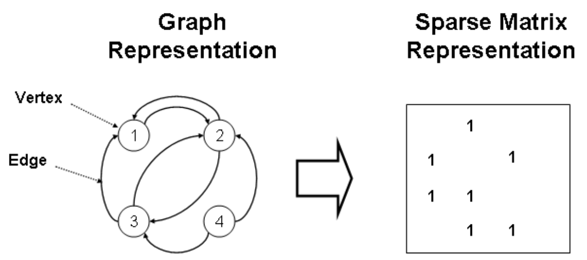

Figure 2: Sparse Matrix Representation of Graph.

Graphs can be represented as sparse matrices as shown in Figure 2. The graph *G(V, E)* with vertices *V* and edges *E* can be represented with the sparse matrix *A* where the matrix element A_{ij} represents the edge between the vertex *i* and vertex *j*. In this example, A_{ij} is set to 1 when there is an edge between the vertices *i* and *j*. If there is no edge between the vertices *i* and *j*, then A_{ij} would be zero and thus would have no entry in the sparse matrix. Once the graphs have been converted to the sparse matrix format, the sparse matrix operations can be used to implement the graph algorithms. Important kernels include sparse matrix multiply, addition, subtraction, and division operations. Individual element level operators within these matrix operations, such as multiply and accumulate operators in the matrix multiply operation, may need be replaced with other arithmetic or logical operators such as maximum, minimum, AND, OR, XOR, etc. in order to implement general graph algorithms. Numerous graph algorithms have already been converted to sparse matrix algorithms [1].

The new graph processor architecture is a parallel processor interconnected in 3-D toroidal configuration using very high bandwidth links [2] as shown in Figure 3. The 3-D toroid provides much higher communication performance than 2-D toroid due to higher bisection bandwidth. In order to minimize communication link lengths, the 2-D toroidal cluster is placed on a circuit board and multiple circuit boards are stacked on top of each other to form the 3-D toroid. The links between the circuit boards are enabled by an array of electromagnetic coupling connectors [3] that can communicate at high data rates without requiring physical conductor connections. Custom designed high-speed I/O circuitries provide high-bit-rate low-power communication for 2-D links within the board and 3-D links between the boards. Each node processor is designed to be capable of over 1 trillion bits per second communication rate to keep up with the communication demands of graph algorithms.

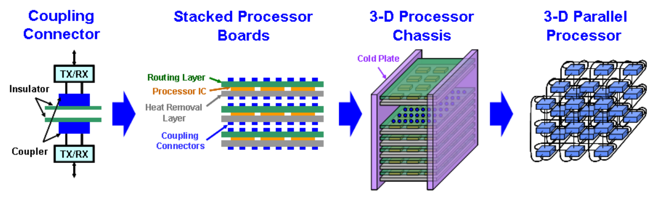

Figure 3: 3-D Graph Processor with Electromagnetic Coupling Communications between Processor Boards.

The communication network is a packet routing network optimized to support small packet sizes that are as small as a single sparse matrix element. The network scheduling and protocol are designed so that successive communication packets from a node would have randomized destinations in order to minimized network congestions [4]. This is a great contrast to typical multi-processor message routing schemes that are based on much larger message sizes and globally arbitrated routing. According to the simulations, the randomized destination packet switching network can provide up to six times higher communication data rates for graph algorithms than conventional parallel processor networks with identical link bandwidths.

The individual node processor architecture is also a great departure from cache-based von Neumann machines. It utilizes specialized modules connected through a high bandwidth network as shown in Figure 3 [2]. There is no cache, since the cache miss rates tend to be high in graph processing. Most of the sparse matrix computation is done by the specialized modules that are designed to optimally perform the given tasks. The matrix reader and writer take care of all the overhead memory accesses required in dealing with the sparse matrix indices. The sorter module is used for finding matching element indices for matrix operations and sorting results for storage. This module is critical since over 95% of computational throughput can be associated sorting of indices. The systolic k-way merge sorter architecture [5] was developed to provide up to an order of magnitude higher throughput than the conventional merge sorter. The custom systolic sorter module can provide up to two orders of magnitude higher sorter throughput than prevailing microprocessor based sorting. Advanced process mapping algorithms and sparse matrix compiler are also being developed to optimize computational load balancing and to enable simplified user interface.

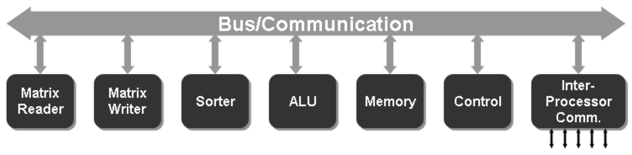

Figure 4: Node Processor Architecture.

Simulation and Performance Projection

Detailed simulation of the architecture was performed to verify the design and to estimate the performance. The bit-level accurate simulation models were used to simulate the entire 1024-node processor running the graph algorithm kernels. The performance projection was achieved by extrapolating the existing computation circuits to the target fabrication process at 60nm. The new custom communication circuitry was developed to provide 3-D interconnect based on coupling connectors. Figure 5 shows the computational throughput projections versus number of processor nodes assuming that the database size scales with the number of processors. It is projected that the processor would provide several orders of magnitude higher graph computational throughput compared to the commercial alternatives. The power efficiency was also projected to be several orders of magnitude higher.

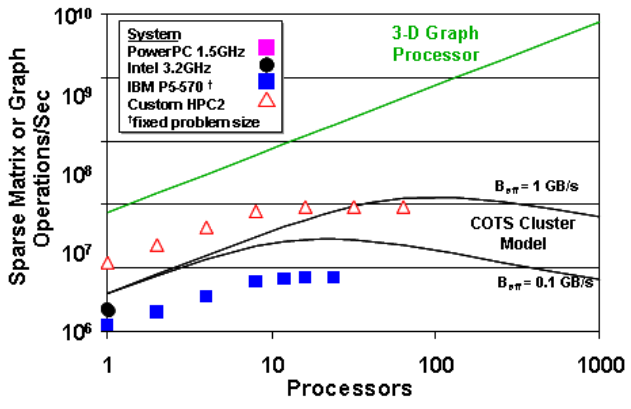

Figure 5: Performance Projection via Simulation.

Acknowledgement

We would like to acknowledge late Dennis Healy at DARPA MTO, who made this work possible.

References

[1] Jeremy Kepner and John Gilbert, "Graph Algorithms in the Language of Linear Algebra," *SIAM Press,* 2010. (To be published)

[2] William S. Song, "High-Performance Processor for Large Graph Algorithm and Sparse Matrix Computations," *MIT Invention Disclosure,* March 2010.

[3] William S. Song, "Electromagnetic Coupling Connector for Three-Dimensional Electronic Circuits," *US Patent No. 6,891,447,* May 10, 2005.

[4] William S. Song, "Multiprocessor Communication Networks," *US Patent Pending No. 12,703,938,* February 11, 2010.

[5] William S. Song, "Systolic Merge Sorter," *US Patent Pending No. 12,403,903,* March 13, 2009.