Bigger and Faster Data-graph Computations for Physical Simulations

Predrag Gruevski, William Hasenplaugh, and James J. Thomas

Massachusetts Institute of Technology
Cambridge, MA 02139, USA
{predrag, whasenpl, lesani, jjthomas}@mit.edu
`http://toc.csail.mit.edu/`

Abstract. We investigate the problem of implementing the physical simulations specified in the domain-specific language Simit as a ***data-graph computation***. *Data-graph computations* consist of a graph $G = (V, E)$, where each vertex has data associated with it, and an update function which is applied to each vertex, taking as inputs the neighboring vertices. PRISM is a framework for executing data-graph computations in shared memory using a scheduling technique called *chromatic scheduling*, where a coloring of the input graph is used to parcel out batches of independent work, sets of vertices with a common color, while preserving determinism. An alternative scheduling approach is *priority-dag scheduling* where a priority function ρ mapping each vertex $v \in V$ to a real number is used to orient the edges from low to high priority and and thus generate a dag. We propose to extend PRISM in two primary ways. First, we will extend it to use distributed memory to enable problem sizes many orders of magnitude larger than the current implementation using a graph partitioning approach which minimizes the number of edges that cross distributed memory nodes. Second, we will replace the chromatic scheduler in PRISM with a priority-dag scheduler and a priority function which generates a cache-efficient traversal of the vertices when the input graph is locally connected and embeddable in a low-dimensional space. This subset of graphs is important for the physical simulations generated by the language Simit.

1 Progress

Contributions

We anticipate contributing a new technique for and a well-engineered implementation of a cache-efficient data-graph computation using priority-dag scheduling. In addition, our implementation will support extremely large simulations on distributed-memory clusters. Both of these contributions are high-impact features for the customers of Simit, for which PRISM will be the new backend for multi-cores and clusters of multi-cores. We have some initial exploratory code of priority functions (e.g. BFS depth, color).

Task Breakdown

James and Predrag will extend PRISM to support data reshuffling based on a priority per vertex. Initially, this will be a standalone utility that reads a graph file, reorders the vertices and writes out the shuffled file.

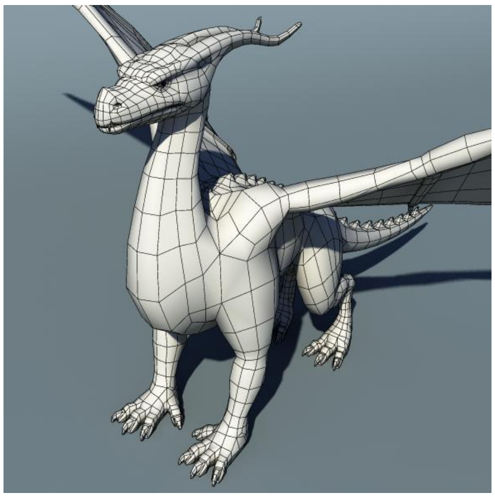

Fig. 1: A mesh graph where lines correspond to edges and intersections of lines correspond to vertices.

Will and Mohsen will engineer the MPI support for distributed memory, including support for distributed data loading.

Issues

We descoped our project to focus on being a good back-end for Simit, which works on graphs of physical simulations. This means ditching the investigation of software prefetching and static assignment of vertices to processors. However, our new focus has allowed us to do something more elegant, essentially a cache-oblivious scheduling algorithm for data-graph computations on graphs of physical simulations.

2 Introduction

This section introduces the problem of deterministically scheduling data-graph computations while preserving good cache efficiency on the special subset of graphs that are locally connected and embeddable in a low-dimensional space.

2.1 Cache Efficiency Perils

This section will demonstrate the cache efficiency issues inherent in chromatic scheduling and dag-scheduling with a poor priority function.

2.2 Distributed Memory

In this section, we will discuss the challenges in extending the implementation of PRISM to support distributed memory.

3 Space-filling Curves

In this section, we will describe the rationale behind using the Hilbert space-filling curve as a way of mapping an N-dimensional space onto the real line, specifically to use this mapping as a priority function for the priority-dag scheduler in Prism. We will present event counters (e.g. cache misses, TLB misses etc.), measured performance and span for three serial experiments on the same set of graphs: chromatic scheduling, priority-dag scheduling with a random priority function and priority-dag scheduling with a Hilbert curve priority function.

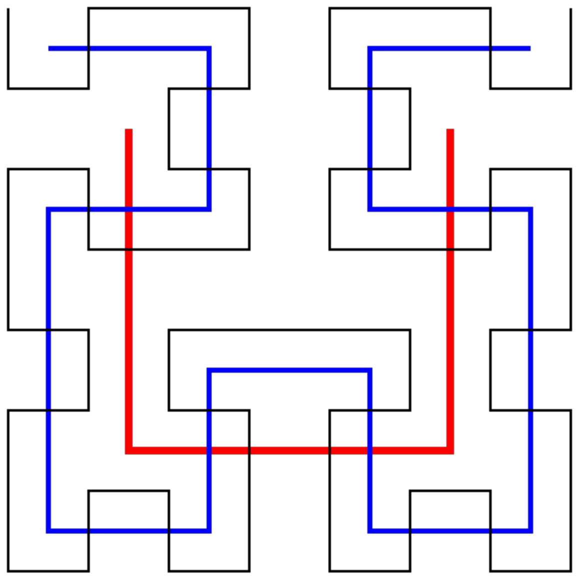

Fig. 2

4 Graph Partitioning for Distributed Memory

In this section, we will describe how the Hilbert curve is also a convenient mechanism for paritioning locally connected graphs that are embeddable in a low-dimensional space. That is, it generates a partition with small edge cuts. We will discuss how the priority-dag scheduling approach enables us to decompose the problem into two phases. The first phase is to extend PRISM to support a reshuffling operator, given a priority value from each vertex, which re-organizes the graph data structure in linear order according to the priority function. The second phase is to partition the n vertices by merely assigning n/p-sized compact subintervals of vertices to each of the p multi-core nodes. Finally, we will describe the software architecture that integrates MPI commands communicating over edges spanning partitions with the priority-dag scheduled computations on each multi-core node. We will test the performance of our implementation by measuring strong-scaling performance on a small set of test graphs.