The MultiThreaded Graph Library (MTGL)

Jonathan Berry (Sandia Labs)

March 13, 2008

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.

Outline

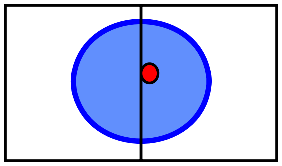
- Graph Software Overview
- Motivating Challenges
- MultiThreaded architectures
- The MTGL
- Integration
- Results
- Paths forward

Graph Software Overview

- LEDA (Mehlhorn, Naher)
- Stanford Graph Base (Knuth, 1993)
- LINK (DIMACS, 1996)
- Boost Graph Library (Siek,Lee,Lumsdaine, 2001)
- Parallel Boost Graph Library (Gregor, Lumsdaine, 2005)
- MultiThreaded Graph Library (2008)

Motivating Challenges

- Many graph algorithms are latency-limited
- Many real-world graph instances exhibit power laws – partitioning harder



- Many real-world graph instances are large enough to utilize HPC

How should we write graph algorithms for HPC that are scalable in both running time and memory?

HPC Options [Programming Model]

- **Distributed Memory [MPI, UPC]**
  - Parallel Boost Graph Library [ghost nodes]
  - LLNL Blue Gene/Light work [no ghost nodes]
- **SMP [OpenMP, UPC, MPI]**
  - SIMPLE, SNAP
- **SMT (e.g. Niagara) [pthreads, LWT]**
- **Massive MultiThreading (MTA/XMT) [CRAYpe]**

HPC Options [Programming Model]

- **Distributed Memory [MPI, UPC]**
  - Parallel Boost Graph Library [ghost nodes]
  - LLNL Blue Gene/Light work [no ghost nodes]
- **SMP [OpenMP, UPC, MPI]**
  - SIMPLE, SNAP
- **SMT (e.g. Niagara) [pthreads, LWP]**
- **Massive MultiThreading (MTA/XMT) [CRAYpe]**

MTGL

Sandia National Laboratories

The MTGL

```
template <class graph>
void print(graph& g)
{
      typedef typename graph_traits<graph>::vertex_descriptor      vertex_descriptor_t;
      typedef typename graph_traits<graph>::adj_vertex_iterator      adj_iterator_t;
      int i, j;
      int n = g.get_order();
      adj_iterator_t begin_v, end_v;
      vertex_id_map<graph> vid_map = get(_vertex_id_map,g);
      #pragma mta assert parallel
      for (i=0; i<n; i++) {
            vertex_descriptor_t v = g.get_vertex(i);
            tie(begin_v,end_v) = adj_vertices(v,g);
            int deg = degree(v,g);
            int vid = get(vid_map, v);
            printf("%d: ", vid);
            #pragma mta assert parallel
            for (j=0; j<deg; j++) {
                  begin_v.set_position(j);
                  vertex_descriptor_t neighbor = *begin_v;
                  int nid = get(vid_map, neighbor);
                  printf("%d ", nid);
            }
            printf("\n");
      }
}
```

Boost-like generic programming

Berkeley Open-Source License pending

Anatomy of a “Hello World” MTGL Program

```
template <class graph>
void print(graph& g)
{
    typedef typename graph_traits<graph>::vertex_descriptor
                                        vertex_descriptor_t;
    typedef typename graph_traits<graph>::adj_vertex_iterator
                                        adj_iterator_t;
```

Generic programs retrieve artifacts of the (hidden) Underlying graph representation via “graph traits”

Adapter Methods and Iterators

```
int n = g.get_order();
adj_iterator_t begin_v, end_v;
vertex_id_map<graph> vid_map = get(_vertex_id_map,g);
#pragma mta assert parallel
for (i=0; i<n; i++) {
        vertex_descriptor_t v = g.get_vertex(i);
        tie(begin_v,end_v) = adj_vertices(v,g);
        int deg = degree(v,g);
        int vid = get(vid_map, v);
        printf("%d: ", vid);
```

MTGL iterators (random access)

Boost-like maps for attributes

Graph adapter method call

MTGL generic functions

Iteration via Iterators

```
#pragma mta assert parallel
  for (j=0; j<deg; j++) {
        begin_v.set_position(j);
        vertex_descriptor_t neighbor = *begin_v;
        int nid = get(vid_map, neighbor);
        printf("%d ", nid);
  }
```

Synchronization

- **MTA/XMT offer word-level synchronization; MTGL exports that interface**
  - mt_incr(a, i): int_fetch_add
  - mt_readfe(w): read full/empty
  - mt_write(w,v): write empty/full
- **On the MTA/XMT these reduce to the underlying calls**
- **MTGL is being integrated with Sandia's "Qthreads" framework, which handles the implementation on SMP/SMT.**

Performance

Rough results below are for graphs with power-law or near power-law degree distributions ("rough" since code and data have been in flux)

- **MTGL connected components algorithms 200M+ edges**
  - 3GHz workstation ~5 min
  - 40 MTA-2 processors ~2-5s
  - Fastest algorithm in practice uses MTGL primitives to beat Shiloach-Vishkin
- **MTGL single-source shortest paths (Meyer & Sanders delta stepping): 200M+**
  - 3 GHz workstation, pure C code (K. Madduri): ~200+s
  - 3 GHz workstation, MTGL version: ~400+s
  - 40 MTA-2 processors: ~3s C, ~6s MTGL
  - Compiler inlining may eventually explain discrepancy
- **S-T Connectivity:**
  - 3 GHz workstation, pure C (K. Madduri), small input ~0.07s
  - 3 GHz workstation, MTGL, small input ~0.13s
  - Counting argument: 5-10 MTA-2 processors compete with 32k BG/L proc.

MultiThreading Encourages Thinking Differently

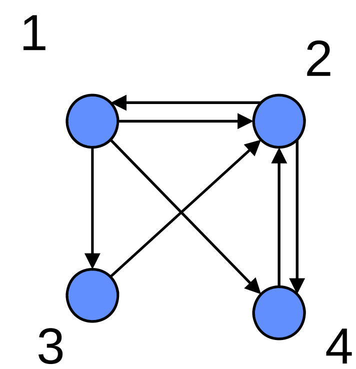
The following sparse matrix-vector multiplication examples show how basic problems might be approached differently in a multithreaded context

The MTGL is going to encapsulate several multithreaded idioms such as the ones that follow

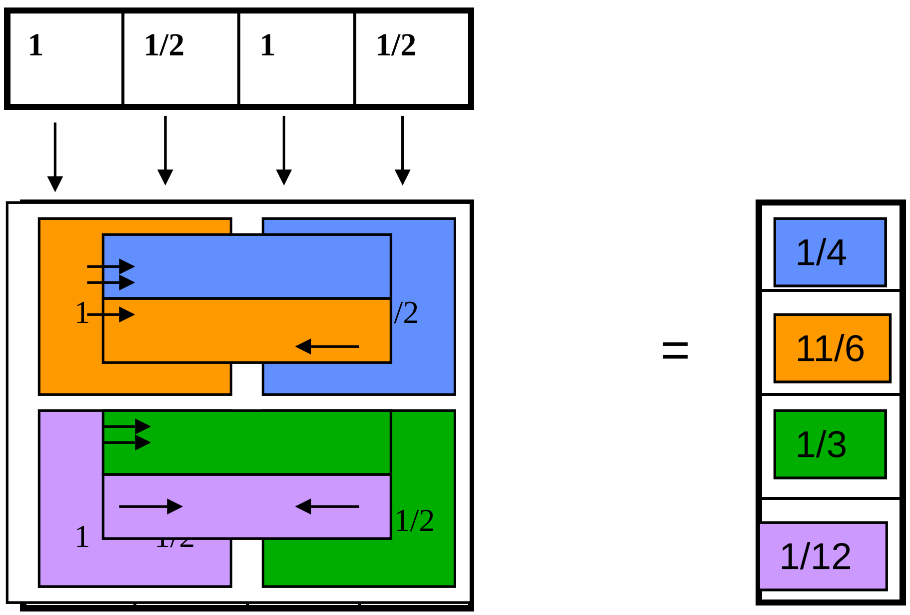
Multithreading Case Study: MatVec

$$A^T$$

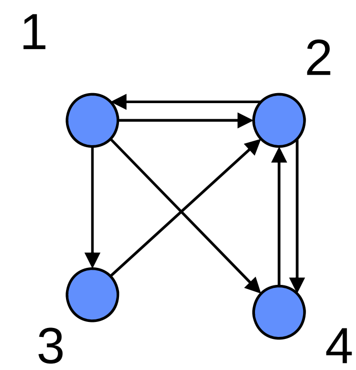
$$\begin{bmatrix} & 1/2 & & \\ 1/3 & & 1 & 1 \\ 1/3 & & & \\ 1/3 & 1/2 & & \end{bmatrix} \begin{bmatrix} 1 \\ 1/2 \\ 1 \\ 1/2 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 7/3 \\ 1/3 \\ 5/6 \end{bmatrix}$$



MatVec in Distributed Memory

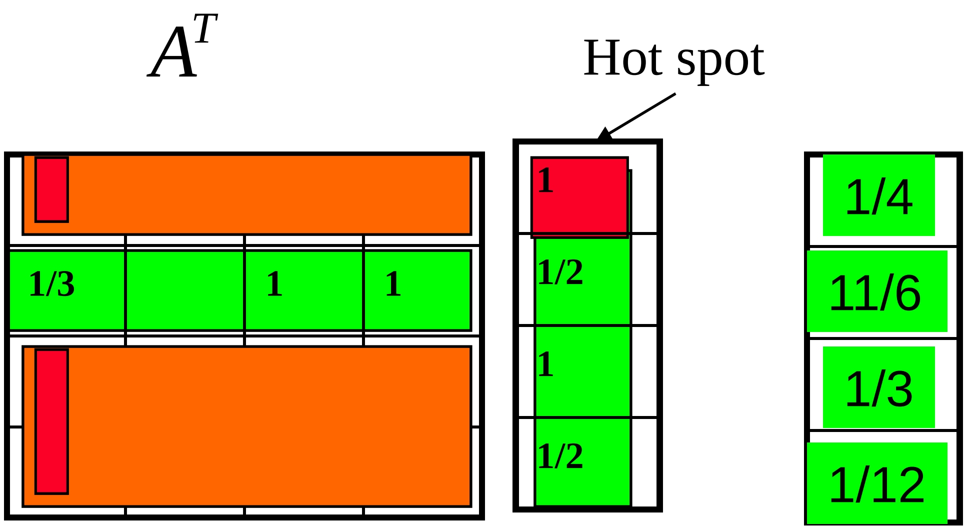


A^T



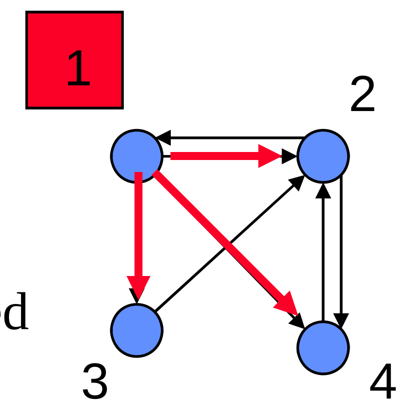
Multithreading Case Study: MatVec

Attempt #1



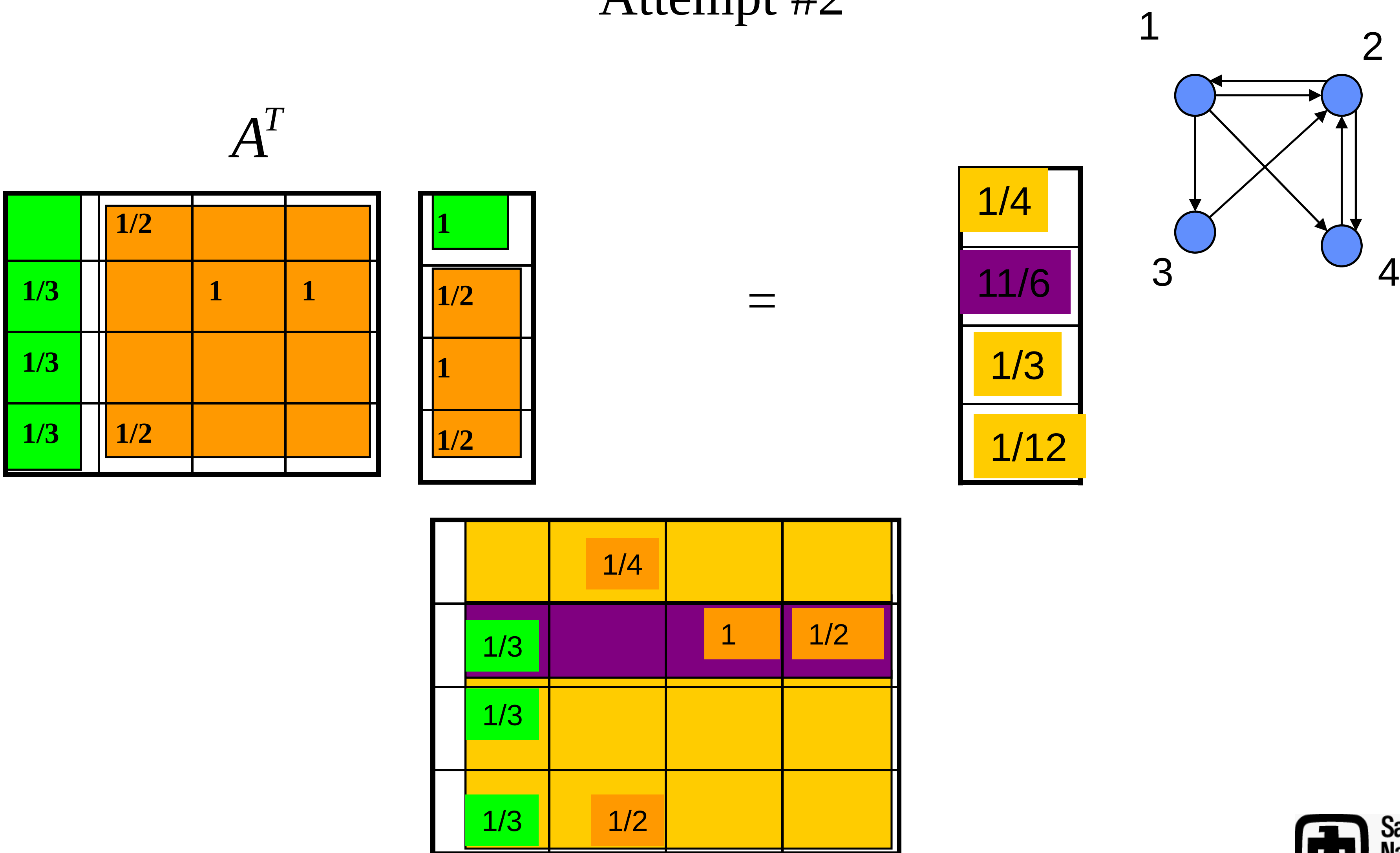
No hot spot;

Compiler handled



Multithreading Case Study: MatVec

Attempt #2



MTGL Features

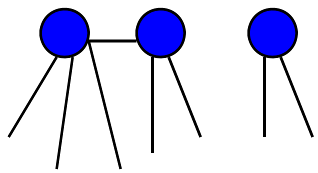
- **Generic programming model means that algorithms can be applied to a wide variety of contexts**
  - vtkGraph
  - Memory-mapped file
  - Matrix Market sparse matrix
  - etc.
- **Recent integration with Sandia's "Qthreads" thread virtualization framework**
  - Scalable generation of R-MAT graphs on Sun Niagara
  - Preliminary scalable execution of PageRank on Sun Niagara
- **Techniques for avoiding hot spots and load imbalances incorporated into primitives**
- **Generic algorithms tend to run within a factor of two of optimal C**
- **Critical representation-specific code can run at optimal C speed:** **f(ga.get_graph())**

Four Modes of MTGL Graph Exploration

visit_adj

for v in V:

visit v’s neighbors

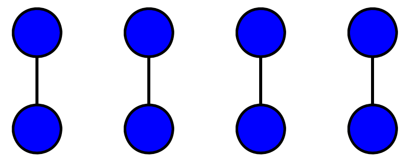


~10 memref/edge

visit_edges

for e in E:

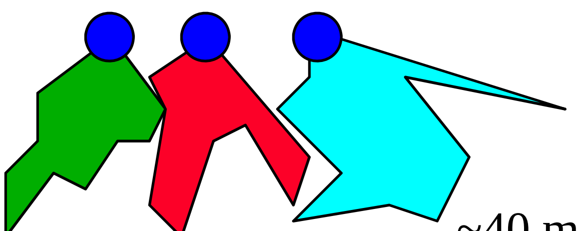
visit e’s endpoints



~10 memref/edge

psearch

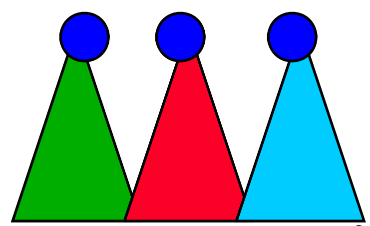
recursive parallel search



~40 memref/edge

breadth-first search

bfs



~20-40 memref/edge

Sandia National Laboratories

Current MTGL Algorithms

- Connected components (psearch, visit_edges, visit_adj)
- Strongly-connected components (psearch)
- Maximal independent set (visit_edges)
- Typed subgraph isomorphism (psearch, visit_edges)
- S-t connectivity (bfs)
- Single-source shortest paths (psearch)
- Betweenness centrality (bfs-like)
- Community detection (all kernels)
- Connection subgraphs (bfs, sparse matrix, mt-quicksort)
- Find triangles (psearch)
- Find assortativity (psearch)
- Find modularity (psearch)
- PageRank (matvec)

Under development:

- Motif detection
- more

Acknowledgements

- Bruce Hendrickson

- Cray, Inc.   Simon Kahan, Petr Konecny, Mike Ringenburg, Josette Huang

- Vitus Leung (Sandia): metrics
- Brad Mancke (Sandia): MTGL kernels
- William McLendon (Sandia): SSSP, S-T connectivity, connection subgraphs
- Bob Heaphy   (original matrix-vector kernel)
- David Bader, Kamesh Madduri (Ga. Tech) (s-t connectivity)
- Cindy Phillips (help with algorithms)
- Andrew Lumsdaine (Indiana U.) (Parallel Boost Graph Library)
- Douglas Gregor (Indiana U.) (Parallel Boost Graph Library)
- Others!

Sandia National Laboratories