Parallel Graph Algorithms:
Architectural Demands of Pathological Applications

Bruce Hendrickson
Jonathan Berry
Keith Underwood
Sandia National Labs

Richard Murphy
Notre Dame University

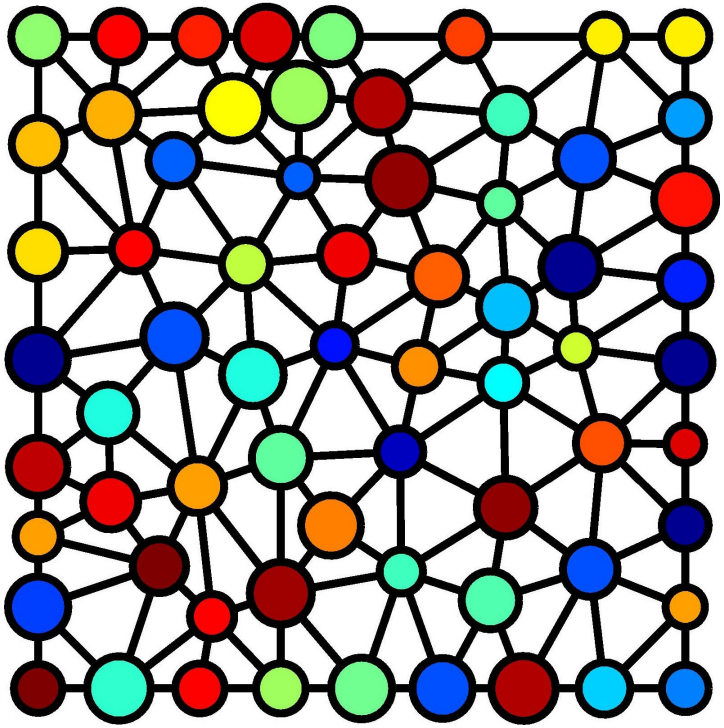
Confessions of a Message-Passing Snob

Bruce Hendrickson

Sandia National Labs

Why Graphs?

- **Exemplar of memory-intensive application**
- **Widely applicable and can be very large scale**
  - » **Scientific computing**
    - – **sparse direct solvers**
    - – **preconditioning**
    - – **radiation transport**
    - – **mesh generation**
    - – **computational biology, etc.**
  - » **Informatics**
    - – **data-centric computing**
    - – **encode entities & relationships**
    - – **look for patterns or subgraphs**

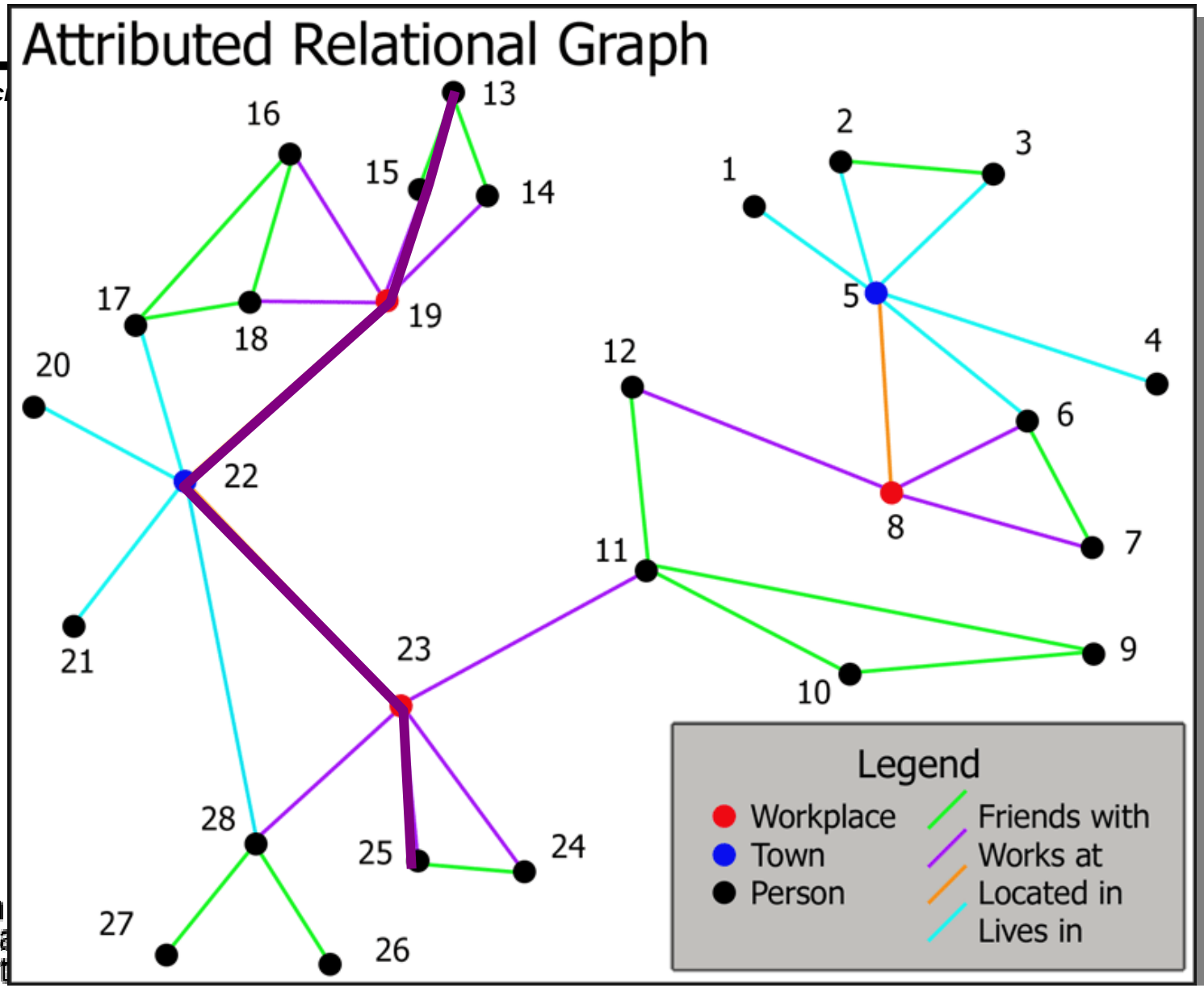


Characteristics

- **Data patterns**
  - » **Moderately structured for scientific applications**
    - – **Even unstructured grids make "nice" graphs**
    - – **Good partitions, lots of locality on multiple scales**
  - » **Highly unstructured for informatics**
    - – **Similar to random, power-law networks**
    - – **Can't be effectively partitioned**

- **Algorithm characteristics**
  - » **Typically, follow links of edges**
    - – **Maybe many at once – high level of concurrency**
    - – **Highly memory intensive**
      - **Random accesses to global memory – small fetches**
      - **Next access depends on current one**
      - **Minimal computation**

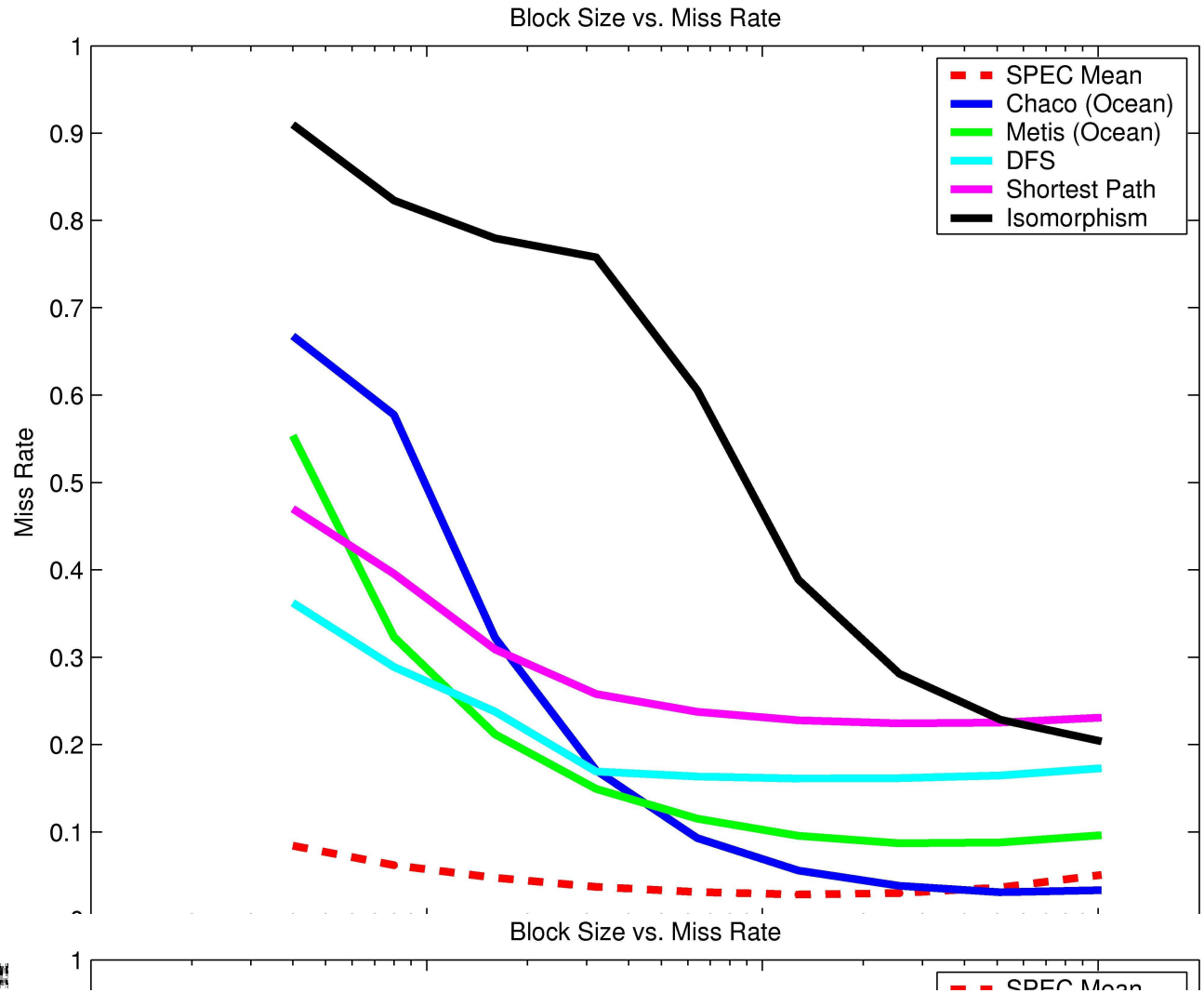
Shortest Path Illustration



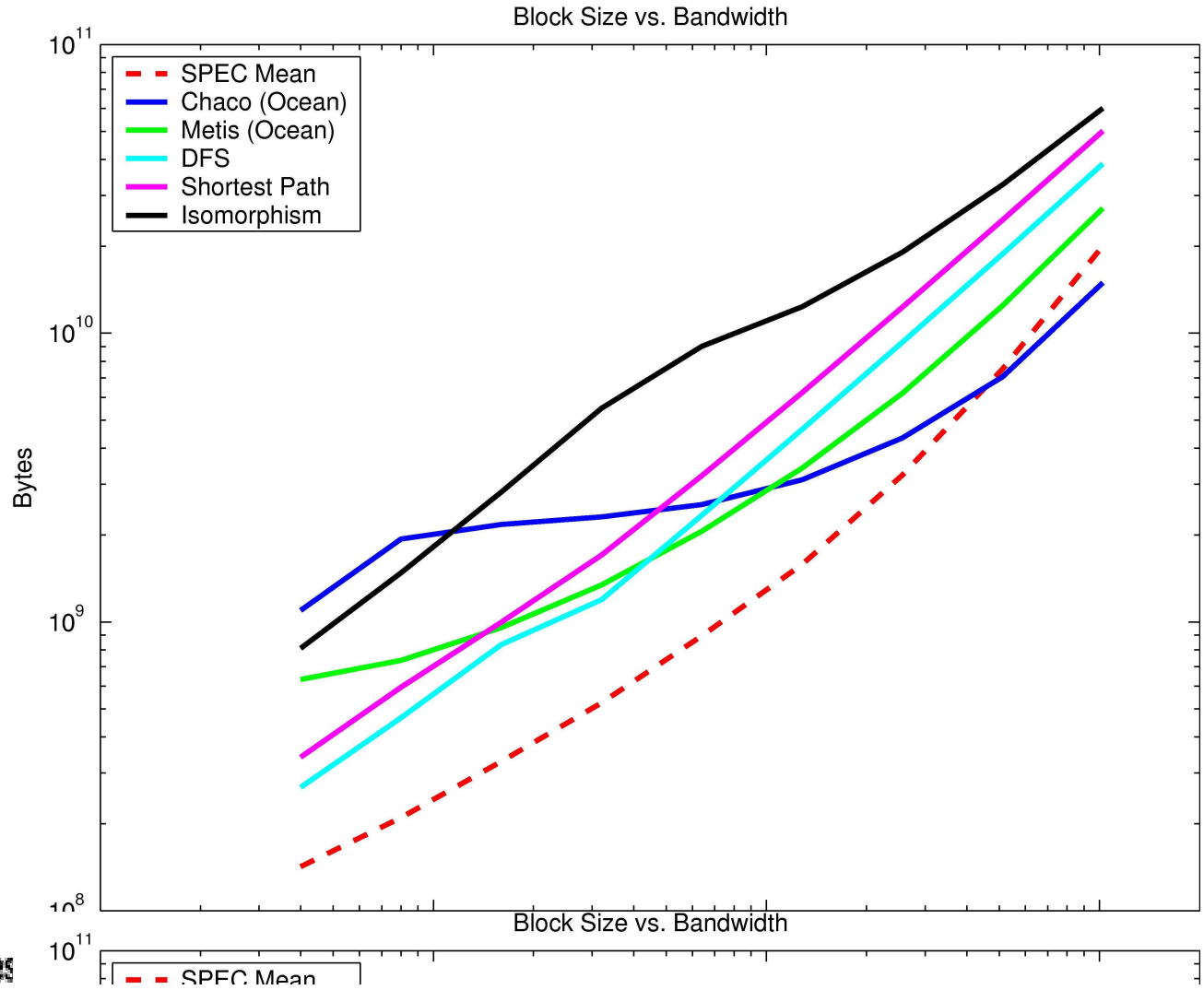
Architectural Challenges

- Runtime is dominated by latency
- Essential no computation to hide memory costs
- Access pattern is data dependent
  - » Prefetching unlikely to help
  - » Often only want small part of cache line
- Potentially abysmal locality at all levels of memory hierarchy

Caching Futility



Larger Blocks are Expensive



Properties Needed for Good Graph Performance

- Low latency / high bandwidth
  - » For small messages!
- Latency tolerant
- Light-weight synchronization mechanisms
- Global address space
  - » No graph partitioning required
  - » Avoid memory-consuming profusion of ghost-nodes

- These describe Burton Smith's MTA!

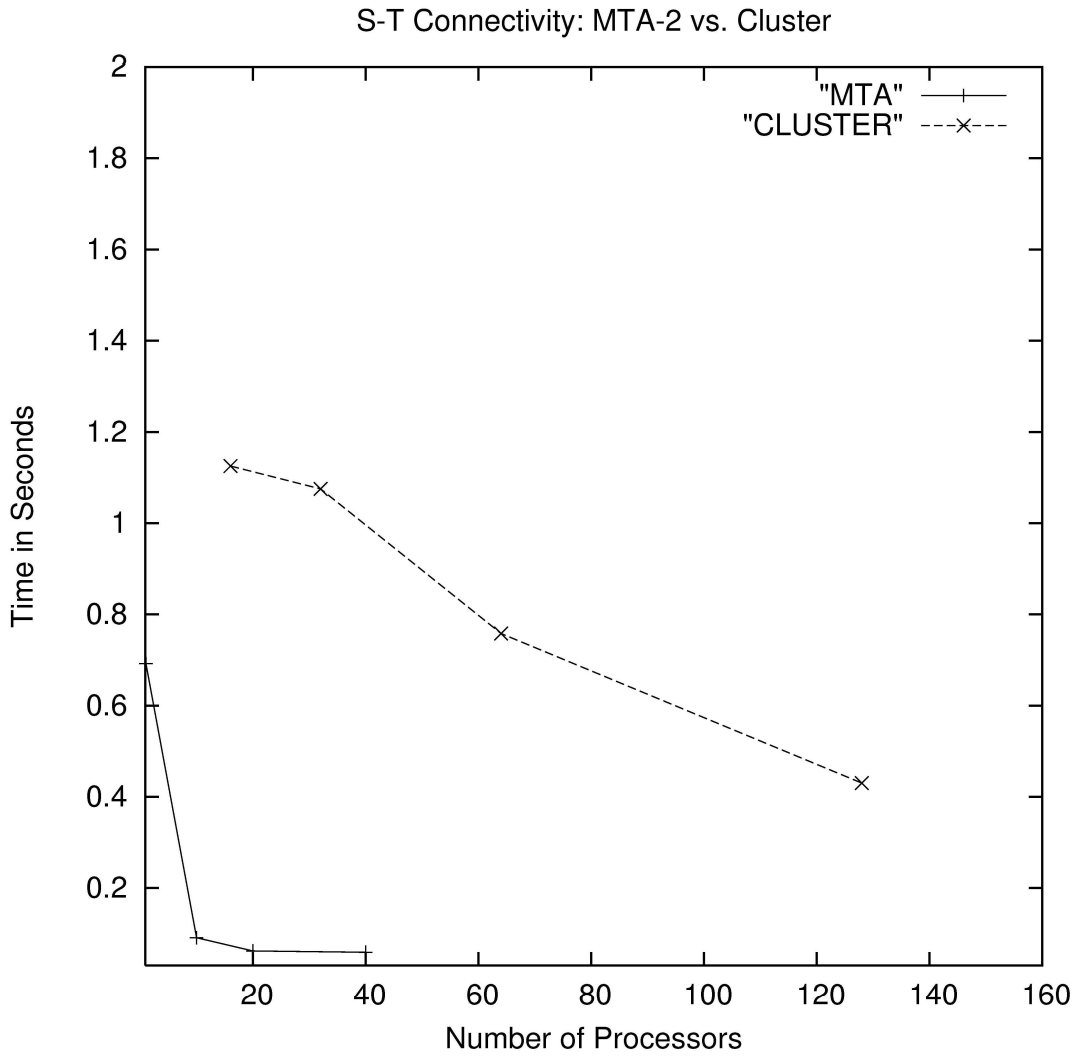
MTA Introduction

- **Latency tolerance via massive multi-threading**
  - » Each processor has hardware support for 128 threads
  - » Context switch in a single tick
  - » Global address space, hashed to reduce hot-spots
  - » No cache. Context switch on memory request.
  - » Multiple outstanding loads
- **Good match for applications which:**
  - » Exhibit complex memory access patterns
  - » Aren't computationally intensive (slow clock)
  - » Have lots of fine-grained parallelism
- **Programming model**
  - » Serial code with parallelization directives
  - » Code is cleaner than MPI, but quite subtle
  - » Support for "future" based parallelism

Case Study – Shortest Path

- Compare codes optimized for different architectures
- Option 1: Distributed Memory CompNets
  - » Run on Linux cluster: 3GHz Xenons, Myrinet network
  - » LLNL/SNL collaboration – just for short path finding
  - » Finalist for Gordon Bell Prize on BlueGene/L
  - » About 1100 lines of C code
- Option 2: MTA parallelization
  - » Part of general-purpose graph infrastructure
  - » About 400 lines of C++ code

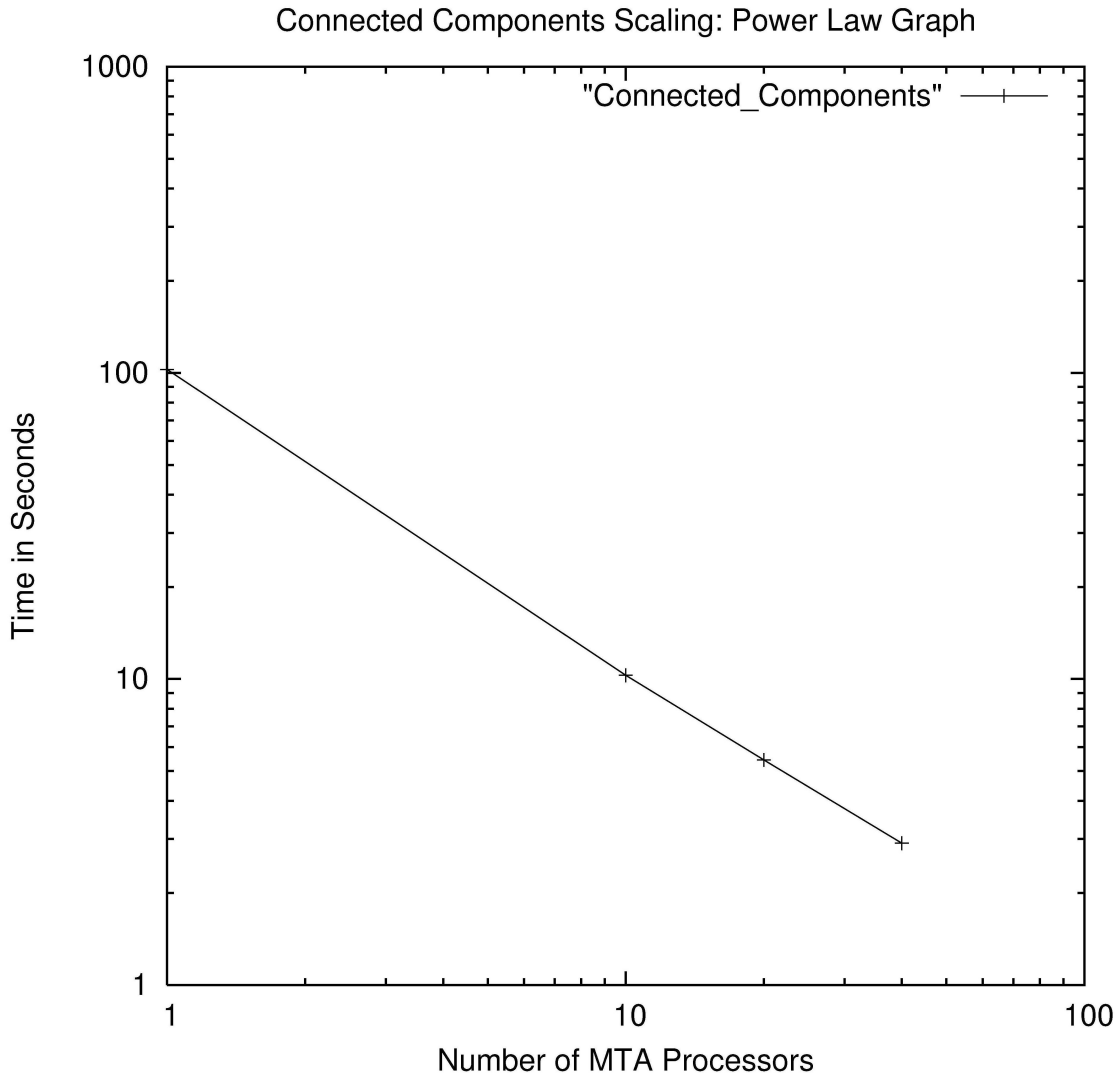
Short Paths on Erdos-Renyi Random Graphs (V=32M, E=128M)



Sandia National Laboratories

Connected Components on MTA-2 Power-Law Graph V=34M, E=235M

procs	time
1	102.7
10	10.29
20	5.44
40	2.91



Sandia National Laboratories

Remarks

- **Single processor MTA competitive with current micros, despite 10x clock difference**
- **Excellent parallel scalability for MTA on range of graph problems**
  - » **Identical to single processor code**
- **Eldorado is coming next year**
  - » **Hybrid of MTA & Red Storm**
  - » **Less well balanced, but affordable**

Broader Lessons

- **Space of important apps is broader than PDE solvers**
  - » **Data-centric applications may be quite different from traditional scientific simulations**
- **Architectural diversity is important**
  - » **No single architecture can do everything well**
- **As memory wall gets steeper, latency tolerance will be essential for more and more applications**
- **High level of concurrency requires**
  - » **Latency tolerance**
  - » **Fine-grained synchronization**