High Performance Graph Analytics on Manycore Processors

George M. Slota[1,2] Sivasankaran Rajamanickam[2]
Kamesh Madduri[1]

[1]Penn State University, [2]Sandia National Laboratories
gslota@psu.edu, madduri@cse.psu.edu, srajama@sandia.gov

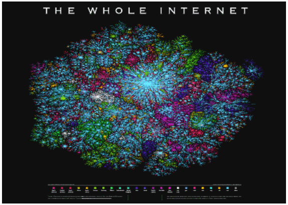
IPDPS 2015, Hyderabad, India
May 26, 2015

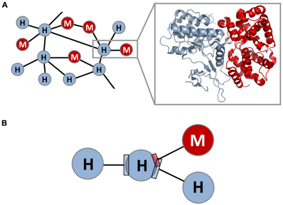
Graphs are...

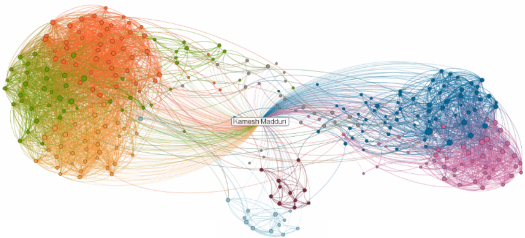
- **Everywhere**

Graphs are...

- **Everywhere**
  - Internet
  - Social, communication networks
  - Computational biology and chemistry
  - Scientific computing, meshing, interactions







- Figure sources: Franzosa et al. 2012, `http://www.unc.edu/ unclng/Internet_History.htm`

Graphs are...

- **Complex**

Graphs are...

- **Complex**
  - Graph analytics is listed as one of DARPA's 23 toughest mathematical challenges
  - Highly diverse – graph structure and problems vary from application to application
  - Real-world graph characteristics makes computational analysis challenging

Graphs are...

- **Complex**
  - Graph analytics is listed as one of DARPA's 23 toughest mathematical challenges
  - Highly diverse – graph structure and problems vary from application to application
  - Real-world graph characteristics makes computational analysis challenging
    - Skewed degree distributions
    - 'Small-world' nature
    - Dynamic

Accelerators (GPUs, Xeon Phi) are also ...

- **Everywhere**
  - Most of the top supercomputers and academic clusters use GPUs and Intel Xeon Phi co-processors
  - Manycore processors might replace multicore in future

Accelerators (GPUs, Xeon Phi) are also ...

- **Everywhere**
  - Most of the top supercomputers and academic clusters use GPUs and Intel Xeon Phi co-processors
  - Manycore processors might replace multicore in future
- **Complex**
  - Multilevel memory, processing hierarchy
  - Explicit communication and data handling
  - Require programming for wide parallelism

- Figure sources: NCSA, NVIDIA

Motivating questions for this work

- **Q**: What are some **common abstractions** that we can use to develop parallel graph algorithms for manycores?
- **Q**: What **key optimization strategies** can we identify to design new parallel graph algorithms for manycores?
- **Q**: Is it possible to develop **performance-portable implementations** of graph algorithms using advanced **libraries and frameworks** using the above optimizations and abstractions?

Our contributions

- **Q**: **Common abstractions** for manycores?
  - We use array-based data structures, express computation in the form of nested loops.
- **Q**: **Key optimization strategies**
  - We improve load balance by manual loop collapse.
- **Q**: **Performance-portable implementations** of graph algorithms using advanced **libraries and frameworks**?
  - We use Kokkos (Edwards et al., JPDC 2014).
- We compare high-level implementations using new framework to hand-optimized code + vary graph computations + vary graph inputs + vary manycore platform.

Talk Overview

- Manycores and the Kokkos programming model
- Abstracting graph algorithms
- Optimizing for manycore processing
- Algorithms
- Results

Background

GPU and Xeon Phi microarchitecture

- **GPU**
  - Multiprocessors (up to about 15/GPU)
  - Multiple groups of stream processors per MP (12×16)
  - *Warps* of threads all execute SIMT on single group of stream processors (32 threads/warp, two cycles per instruction)
  - Irregular computation (high degree verts, if/else, etc.) can result in most threads in warp doing NOOPs
- **Xeon Phi (MIC)**
  - Many simple (Pentium 4) cores, 57-61
  - 4 threads per core, need at least 2 threads/core for OPs on each cycle
  - Highly vectorized (512 bit width) - difficult for irregular computations to exploit

Background
Kokkos and GPU microarchitecture

- Kokkos
  - Developed as back-end for portable scientific computing
  - Polymorphic multi-dimensional arrays for varying access patterns
  - Thread parallel execution for fine-grained parallelism
- Kokkos model - performance portable programming to multi/manycores
  - Thread team - multiple warps on same multiprocessor, but all still SIMT for GPU
  - Thread league - multiple thread teams, over all teams all work is performed
  - Work statically partitioned to teams before parallel code is called

Abstracting graph algorithms

for large sparse graph analysis

- **Observation**: most (synchronous) graph algorithms follow a tri-nested loop structure
  - Optimize for this general algorithmic template
  - Transform structure for more parallelism

Initialize temp/result arrays $A_t[1..n]$, $1 \le t \le l$. ▷ $l = O(1)$
Initialize $S_1[1..n]$.
for $i = 1$ **to** $niter$ **do** ▷ $niter = O(\log n)$
    Initialize $S_{i+1}[1..n]$. ▷ $\sum_i |S_i| = O(m)$
    for $j = 1$ **to** $|S_i|$ **do** ▷ $|S_i| = O(n)$
        $u \leftarrow S_i[j]$
        Read/update $A_t[u]$, $1 \le t \le l$.
        for $k = 1$ **to** $|E[u]|$ **do** ▷ $|E[u]| = O(n)$
            $v \leftarrow E[u][k]$
            Read/update $A_t[v]$.
            Read/update S_{i+1}.
        Read/update $A_t[u]$.

Abstracting graph algorithms

for large sparse graph analysis

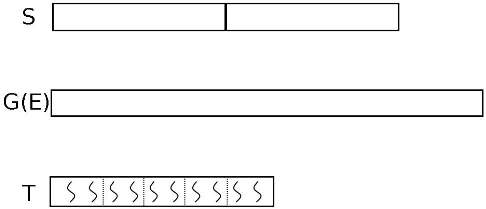
- **Observation**: most (synchronous) graph algorithms follow a tri-nested loop structure
  - Optimize for this general algorithmic template
  - Transform structure for more parallelism

Initialize temp/result arrays $A_t[1..n]$, $1 \le t \le l$. $\triangleright\ l = O(1)$
Initialize $S_1[1..n]$.
for $i = 1$ **to** $niter$ **do** $\triangleright\ niter = O(\log n)$
    Initialize $S_{i+1}[1..n]$. $\triangleright\ \sum_i |S_i| = O(m)$
    for $j = 1$ **to** $|S_i|$ **do** $\triangleright\ |S_i| = O(n)$
        $u \leftarrow S_i[j]$
        Read/update $A_t[u]$, $1 \le t \le l$.
        for $k = 1$ **to** $|E[u]|$ **do** $\triangleright\ |E[u]| = O(n)$
            $v \leftarrow E[u][k]$
            Read/update $A_t[v]$.
            Read/update S_{i+1}.
        Read/update $A_t[u]$.

Optimizations for Manycore Processors

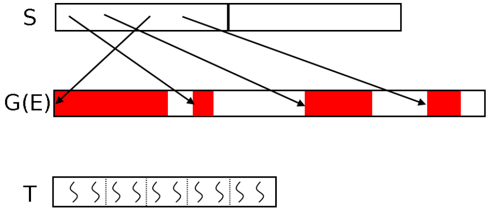
Parallelization strategies

- Baseline parallelization



Optimizations for Manycore Processors
Parallelization strategies

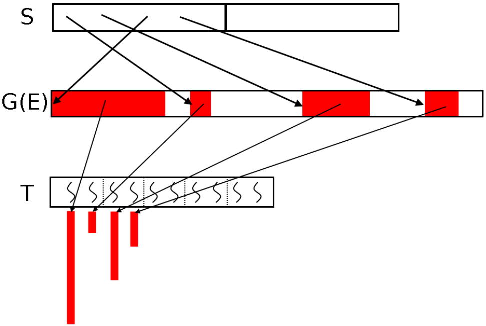
- Baseline parallelization



Optimizations for Manycore Processors

Parallelization strategies

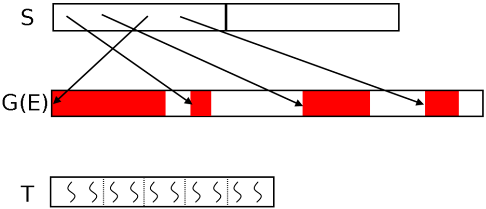
- Baseline parallelization



Optimizations for Manycore Processors

Parallelization strategies

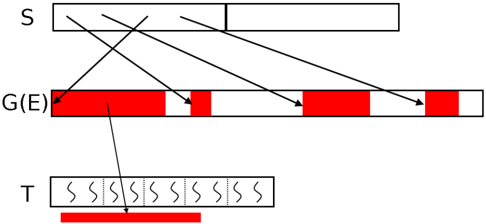
- Baseline parallelization
- Hierarchical expansion (e.g., Hong et al., PPoPP 2011)



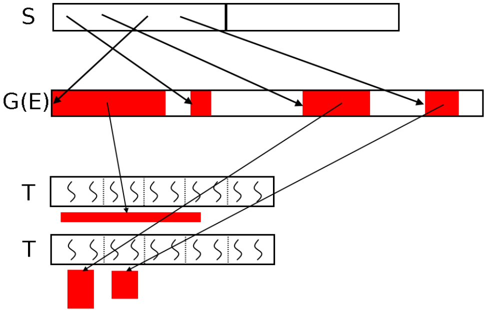
Optimizations for Manycore Processors

Parallelization strategies

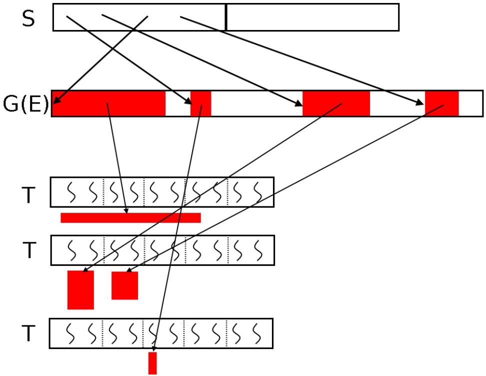
- Baseline parallelization
- Hierarchical expansion (e.g., Hong et al., PPoPP 2011)



- Baseline parallelization
- Hierarchical expansion (e.g., Hong et al., PPoPP 2011)



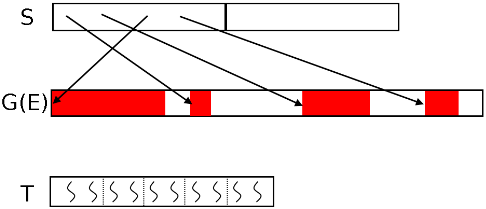
- Baseline parallelization
- Hierarchical expansion (e.g., Hong et al., PPoPP 2011)



Optimizations for Manycore Processors

Parallelization strategies

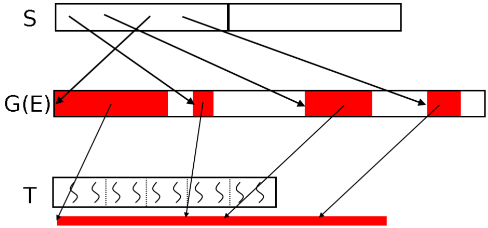
- Baseline parallelization
- Hierarchical expansion (e.g., Hong et al., PPoPP 2011)
- ‘Manhattan collapse - local’ (e.g.m Merrill et al., PPoPP 2012)



Optimizations for Manycore Processors

Parallelization strategies

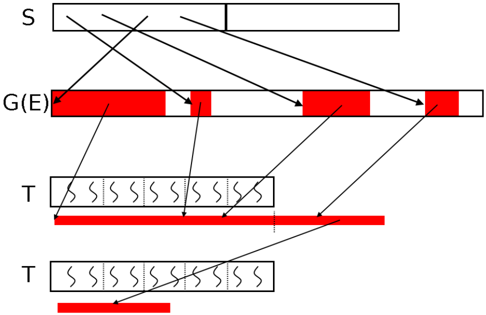
- Baseline parallelization
- Hierarchical expansion (e.g., Hong et al., PPoPP 2011)
- 'Manhattan collapse - local' (e.g.m Merrill et al., PPoPP 2012)



Optimizations for Manycore Processors

Parallelization strategies

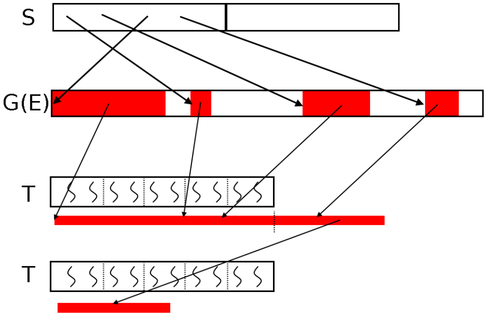
- Baseline parallelization
- Hierarchical expansion (e.g., Hong et al., PPoPP 2011)
- 'Manhattan collapse - local' (e.g.m Merrill et al., PPoPP 2012)



Optimizations for Manycore Processors

Parallelization strategies

- Baseline parallelization
- Hierarchical expansion (e.g., Hong et al., PPoPP 2011)
- 'Manhattan collapse - local' (e.g.m Merrill et al., PPoPP 2012)
- 'Manhattan collapse - global' (e.g., Davidson et al., IPDPS 2014)



- Memory access
  - Explicit shared memory utilization on GPU
  - Coalescing memory access (locality)
  - Minimize access to global/higher-level memory
- Collective operations
  - Warp and team-based operations (team scan, team reduce)
  - Minimize global atomics (team-based atomics)

Graph computations

Implemented algorithms

- Breadth-first search
- Color propagation
- Trimming
- The Multistep algorithm (Slota et al., IPDPS 2014) for Strongly Connected Components (SCC) decomposition

Graph computations
Breadth-first search

- Useful subroutine in other graph computations

$A_1[1..n] \leftarrow -1$
$S_1[1] \leftarrow root$
$level \leftarrow 0$
while $|S_i| \neq \emptyset$ **do**
    Initialize S_{i+1}
    for $j = 1$ **to** $|S_i|$ **do**
        $u \leftarrow S_i[j]$
        for $k = 1$ **to** $|E[u]|$ **do**
            $v \leftarrow E[u][k]$
            if $A_1[v] < 0$ **then**
                $A_1[v] \leftarrow level$
                $S_{i+1} \leftarrow v$
    $level \leftarrow level + 1$

Graph computations
Breadth-first search

- Useful subroutine in other graph computations

$A_1[1..n] \leftarrow -1$
$S_1[1] \leftarrow root$
$level \leftarrow 0$
$\boxed{\textbf{while } |S_i| \neq \emptyset \textbf{ do}}$
    Initialize S_{i+1}
    $\boxed{\textbf{for } j = 1 \textbf{ to } |S_i| \textbf{ do}}$
        $u \leftarrow S_i[j]$
        $\boxed{\textbf{for } k = 1 \textbf{ to } |E[u]| \textbf{ do}}$
            $v \leftarrow E[u][k]$
            if $A_1[v] < 0$ **then**
                $A_1[v] \leftarrow level$
                $S_{i+1} \leftarrow v$
    $level \leftarrow level + 1$

Graph computations

Color propagation

- Basic algorithm for connectivity
- General approach applies to other algorithms (e.g., label propagation)

$A_1[1..n] \leftarrow [1..n]$
$S_1[1..n] \leftarrow [1..n]$
while $|S_i| \neq \varnothing$ **do**
    Initialize S_{i+1}
    for $j = 1$ **to** $|S_i|$ **do**
        $u \leftarrow S_i[j]$
        for $k = 1$ **to** $|E[u]|$ **do**
            $v \leftarrow E[u][k]$
            if $A_1[u] > A_1[v]$ **then**
                $A_1[v] \leftarrow A_1[u]$
                $S_{i+1} \leftarrow v$

Graph computations

Color propagation

- Basic algorithm for connectivity
- General approach applies to other algorithms (e.g., label propagation)

```
A_1[1..n] ← [1..n]
S_1[1..n] ← [1..n]
while |S_i| ≠ ∅ do
    Initialize S_{i+1}
    for j = 1 to |S_i| do
        u ← S_i[j]
        for k = 1 to |E[u]| do
            v ← E[u][k]
            if A_1[u] > A_1[v] then
                A_1[v] ← A_1[u]
                S_{i+1} ← v
```

Graph computations
Trimming

- Routine for accelerating connectivity decomposition
- Iteratively trim 0-degree vertices

$A_1[1..n] \leftarrow 1$
$S_1[1..n] \leftarrow [1..n]$
while $|S_i| \neq \varnothing$ **do**
    Initialize S_{i+1}
    for $j = 1$ **to** $|S_i|$ **do**
        $u \leftarrow S_i[j]$
        $trim \leftarrow$ **true**
        for $k = 1$ **to** $|E[u]|$ **do**
            $v \leftarrow E[u][k]$
            if $A_1[v] = 1$ **then**
                $trim \leftarrow$ **false**
        if $trim =$ **true then**
            $A_1[u] \leftarrow 0$
            $S_{i+1} \leftarrow E[u]$

Graph computations

Trimming

- Routine for accelerating connectivity decomposition
- Iteratively trim 0-degree vertices

```
A_1[1..n] ← 1
S_1[1..n] ← [1..n]
while |S_i| ≠ ∅ do
    Initialize S_{i+1}
    for j = 1 to |S_i| do
        u ← S_i[j]
        trim ← true
        for k = 1 to |E[u]| do
            v ← E[u][k]
            if A_1[v] = 1 then
                trim ← false
        if trim = true then
            A_1[u] ← 0
            S_{i+1} ← E[u]
```

Graph computations

Multistep SCC decomposition (Slota et al., IPDPS 2014)

- Combination of trimming, BFS, and color propagation

$T \leftarrow \mathsf{Trim}(G)$
$V \leftarrow V \setminus T$
Select $v \in V$ for which $d_{in}(v) * d_{out}(v)$ is maximal
$D \leftarrow \mathsf{BFS}(G(V, E(V)), v)$
$S \leftarrow D \cap \mathsf{BFS}(G(D, E'(D)), v)$
$V \leftarrow V \setminus S$
while $\mathsf{NumVerts}(V) > 0$ **do**
    $C \leftarrow \mathsf{ColorProp}(G(V, E(V)))$
    $V \leftarrow V \setminus C$

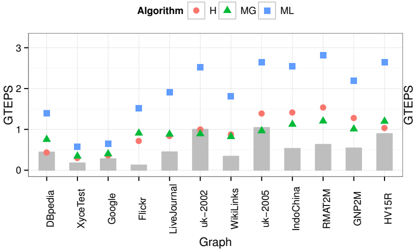
Experimental Setup

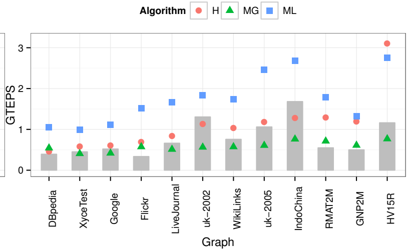
- Test systems: One node of *Shannon* and *Compton* at Sandia, Blue Waters at NCSA
  - Intel Xeon E5-2670 (Sandy Bridge), dual-socket, 16 cores, 64-128 GB memory
  - NVIDIA Tesla K40M GPU, 2880 cores, 12 GB memory
  - NVIDIA Tesla K20X GPU, 2688 cores, 6 GB memory
  - Intel Xeon Phi (KNC, ∼3120A), 228 cores, 6 GB memory
- Test graphs:
  - Various real and synthetic small-world graphs, 5.1 M to 936 M edges
  - Social networks, circuit, mesh, RDF graph, web crawls, R-MAT and $G(n, p)$, Wikipedia article links

Results

BFS and Coloring versus loop strategies

- Performance in GTEPS (10^9 trav. edges per second) for BFS (left) and color propagation (right) on Tesla K40M.
- H: Hierarchical, ML: Local collapse, MG: Global collapse, gray bar: Baseline

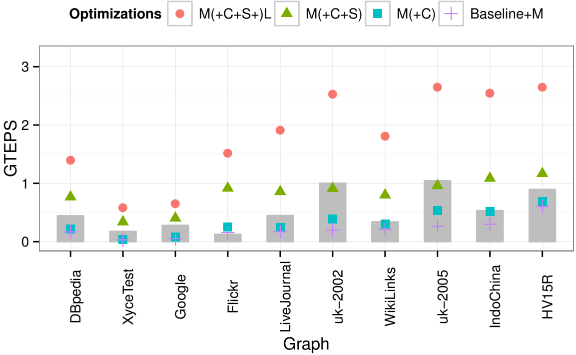




Results

BFS performance and cumulative impact of optimizations, Tesla K40M

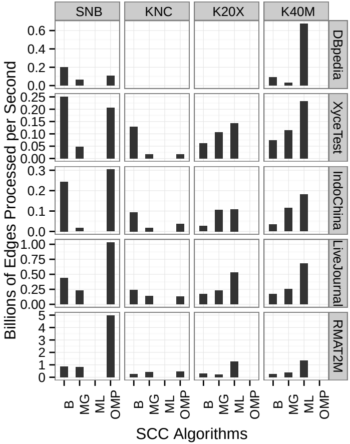
- M: local collapse, C: coalescing memory access, S: shared memory use, L: local team-based primitives



Results

SCC cross-platform performance comparison

- B: Baseline, MG: Manhattan Global, ML: Manhattan Local, OMP: Optimized OpenMP code



Conclusions

- We express several graph computations in the **Kokkos** programming model using an **algorithm design abstraction** that allows portability across both multicore platforms and **accelerators**.
- The **SCC** code on GPUs (using the **Local Manhattan Collapse** strategy) demonstrates up to a $3.25\times$ speedup relative to a state-of-the-art parallel CPU implementation running on a dual-socket compute node.
- Future work: Expressing other computations using this framework; Heterogeneous CPU-GPU processing; Newer architectures.

This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070, ACI-1238993, and ACI-1444747) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. This work is also supported by NSF grants ACI-1253881, CCF-1439057, and the DOE Office of Science through the FASTMath SciDAC Institute. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.