SAND2014-19503PE

Exceptional service in the national interest

Sandia National Laboratories

Manycore Graph Algorithms and the Kokkos Library

Erik Boman, Dagstuhl, Nov. 2014

Joint work with Siva Rajamanickam

U.S. DEPARTMENT OF ENERGY

National Nuclear Security Administration

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.

Outline

Sandia National Laboratories

- Computer architecture challenges
- What is Kokkos?
- Kokkos for Graphs
- Example: Graph Coloring

Motivation

Sandia National Laboratories

- Current trend is greater parallelism on node
- Diversity in architectures and programming models:
  - Distributed memory: MPI, PGAS
  - Shared memory: OpenMP, pthreads, TBB, Cilk+, etc.
  - GPU: CUDA, OpenACC
  - XMT: proprietary directives
- Software must be ported several times to support all platforms
  - Algorithms may also change to optimize performance
- Many choices for X in MPI+X. What to do?

What is Kokkos?

Sandia National Laboratories

- Library and programming model for multithreaded and manycore programming
- Key feature: Performance portable across range of architectures, including
  - Multicore CPU
  - Intel MIC (Xeon Phi)
  - GPU
- Main focus: Scientific computing
  - Will support on-node kernels in Trilinos (huge collection of scientific computing packages)
- Initial version is data parallel
  - Task parallelism is coming

Kokkos Core Features

Sandia National Laboratories

- Kokkos::View is the primary data type
  - Multi-dimensional array
  - Abstraction that hides underlying memory layout
  - Default layout depends on your architecture
  - Also provides “first-touch” optimization
- Data parallel constructs:
  - Parallel_for
  - Parallel_reduce
  - Parallel_scan
- Functor or lambda API
  - User writes code as functor to be invoked by Kokkos
  - Simple operations can be done inline using lambdas (C++11)

Execution and Memory Spaces

Sandia National Laboratories

- The execution space defines *where* and *how* code is executed.
- You may think of this as a ``back end''. Current options:
  - Pthreads, qthreads
  - OpenMP
  - CUDA

- MemorySpace defines where data live
  - Capability (Fast but small), capacity (slow but big), etc.
  - Will be mapped to something appropriate on your hardware
- Each ExecutionSpace has a default MemorySpace and vice versa

Hierarchical Parallelism

Sandia National Laboratories

- Hierarchical parallelism is important on many architectures
  - GPUs have SMs, warps, thread blocks
  - Thread synchronization time can vary significantly!
- Abstraction: Teams and leagues
  - A team is a group of threads working together
  - A league is a group of teams
  - Sports analogy borrowed from OpenMP

Kokkos for Graph Algorithms

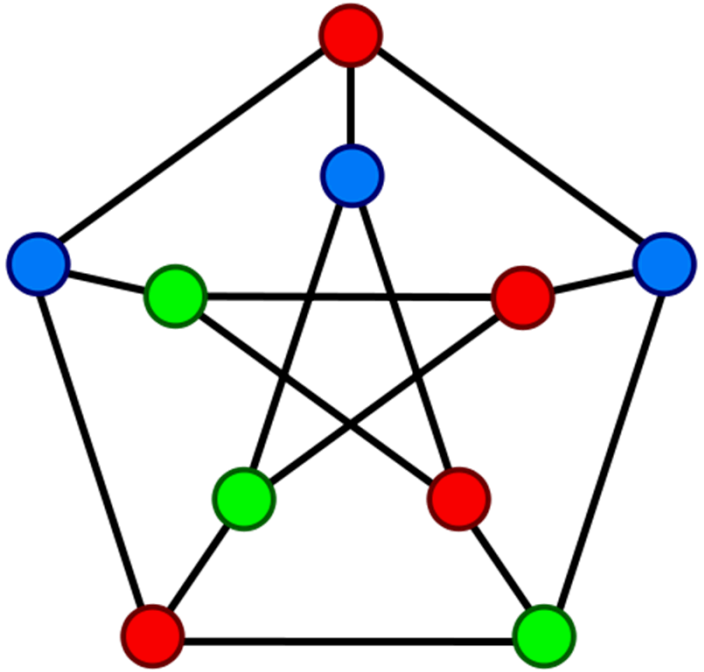
Sandia National Laboratories

- Kokkos was designed for “regular” applications
  - FEM assembly and solve
  - Numerical linear algebra (SpMV)
  - Molycelar dynamics
- Is Kokkos useful for “irregular” graph algorithms?

Case Study: Graph Coloring

Sandia National Laboratories

- Given a graph G(V,E), find a coloring c:V➔N such that no two adjacent vertices have the same color.
- We wish to (approximately) minimize the number of colors.
- Exact solution is NP-hard
- But linear-time greedy methods work well in practice!



Applications: Parallel scheduling, register allocation, sparse matrix ordering, preconditioning, AD

Serial Greedy Algorithm

Sandia National Laboratories

Procedure Greedy(G(V,E))

Allocate forbidden

foreach v in V **do**

forbidden[*] = false

foreach w in adj(v) **do**

forbidden[color[w]] = true

color[v] = min {i>0 | forbidden[i] == false}

End

Coloring depends on the order vertices are visited. Good heuristics are Largest-First (LF) and Smallest-Last (SL). In parallel, we cannot impose any ordering.

Parallel Algorithms

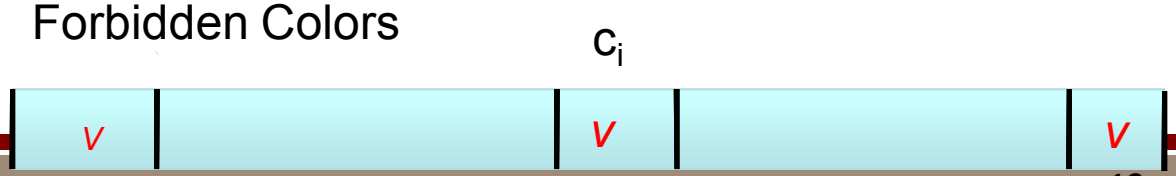
Sandia National Laboratories

- The serial greedy algorithm is inherently sequential, difficult to parallelize.
- Several parallel algorithms have been proposed. The two most popular ones are:
  - Jones-Plassmann (’93): Color sequence of independent sets.
  - Gebremedhin-Manne (‘00): Speculative coloring. Make some mistakes (allow conflicts), go back and fix them later.
- The JP algorithm was popular in the 90’s but now the GM method is generally preferred, due to less synchronization.
  - Zoltan implements the GM speculative method.
  - Extension to distributed-memory by Boman et al. (‘05), Bozdag et al. (‘10)

Multithreaded: Iterative Greedy Algorithm

Sandia National Laboratories

Proc IterativeGreedy $(G = (V, E))$
U is set of vertices to be colored, and R to be recolored
while U is not empty
    1. Speculatively color vertices
    for $v \in U$ **in parallel**
        for each neighbor w of v
            Mark color[w] as forbidden to v
        Assign smallest available value to color[v]
    2. Detect conflicts and create recoloring list
    for $v \in U$ **in parallel**
        for each neighbor w of v
            if color[w] = color[v]
                add higher-numbered vertex to R
    $U = R$
end proc



Algorithmic Issues

Sandia National Laboratories

- Conflict resolution options:
  - Since #conflicts is small, recolor these in serial.
    - GPU: Do on device, or copy over to host?
  - Iterate until all conflicts resolved. Simple in parallel.
  - Another option is to ensure NO conflicts by using atomics, locks, or CAS. This variation is not speculative!

- How to find smallest available color:
  - Typically, a “forbidden” array is used but this requires a lot of memory
    - max-degree is often too pessimistic upper bound, but dynamic allocation is expensive
    - Dynamic reallocation too expensive on Phi and GPU.
  - Can do linear search, which saves memory but increases the work complexity.

Coloring Results for FEM

Sandia National Laboratories

Machine: 16 cores CPU (SandyBridge) + Xeon Phi (KNC)
Graph: audikw1 (944K vertices, 39M edges)
Algorithm: Gebremedhin-Manne with serial conflict resolution
Implementation: Kokkos with OpenMP

	CPU (1)	CPU (4)	CPU (16)	CPU (32)	MIC (57)	MIC(114)	MIC (228)
Total time	0.38	0.18	0.08	0.04	0.17	0.11	0.09
Time speculative	0.23	0.12	0.05	0.03	0.08	0.05	0.04
Time conflict det.	0.15	0.05	0.03	0.01	0.07	0.04	0.03
Time conflict res.	0.00	0.00	0.00	0.00	0.02	0.02	0.03
#colors	54	60	63	63	63	63	63
#conflicts	0	116	540	730	210	416	794

Related Efforts

Sandia National Laboratories

- Several teams are testing Kokkos for graph algorithms
  - MTGL interface to Kokkos (Berry et al.)
  - BFS and SCC using Kokkos (Slota, Rajamanickam, Madduri)
    - Need “Manhattan loop” trick for GPU performance

Conclusions

Sandia National Laboratories

- Manycore programming is hard!
  - Maybe harder than MPI on distributed memory.
  - Choice of algorithm and data structure often tricky.
- Kokkos is useful, but unproven for graph algorithms
  - Works great for scientific computing.
  - Hard to do graph algorithms with just parallel_for, reduce, and atomics. Coloring is a simple problem, so feasible.
  - Task-parallel capabilities in progress; needed for many algorithms.
    - GPU performance?
- No magic bullet to get optimal performance everywhere
  - Must customize (CUDA-style) to get good GPU performance.
  - But this is often slower on CPU/MIC than OpenMP program.
  - Still: Kokkos good option for cross-platform development.

Extra Slides

Sandia National Laboratories

Serial Greedy Algorithm (2)

Sandia National Laboratories

```
Procedure Greedy(G(V,E))
        Allocate and initialize forbidden
        foreach v in V do
                foreach w in adj(v) do
                        forbidden[color[w]] = v
                color[v] = min {i>0 | forbidden[i] != v}
End
```

This optimization saves re-initializing the forbidden array inside the first loop, but requires a sequential order (problematic in parallel).