A High-Speed 2-Opt TSP Solver for Large Problem Sizes

Martin Burtscher
Department of Computer Science
TEXAS STATE UNIVERSITY
The rising STAR of Texas

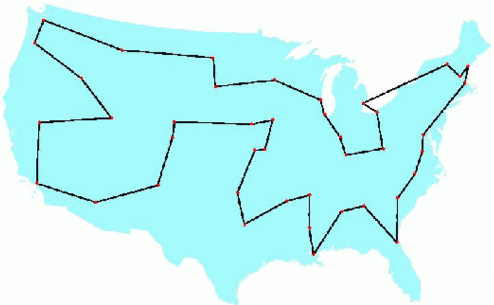
Overview

- CUDA code optimization case study
  - Uses 2-opt improvement heuristic as example
  - Will study 6 different implementations

- Key findings
  - Radically changing the parallelization approach may result in a much better GPU solution
  - Smart usage of global memory can outperform a solution that runs entirely in shared memory

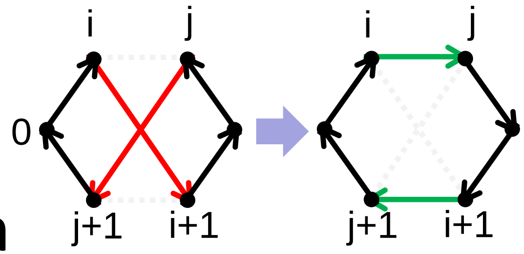
Travelling Salesman Problem (TSP)

- Important combinatorial optimization problem
  - Wire routing, logistics, robot arm movement, etc.
- Given *n* cities, find shortest Hamiltonian tour
  - Must visit all cities exactly once and end in first city
- Usually expressed as a graph problem
  - We use complete, undirected, planar, Euclidean graph
  - Vertices represent cities
  - Edge weights reflect distances



2-opt Improvement Heuristic

- Optimal TSP solution is NP-hard
  - Heuristic algorithms used to approximate solution
- We use 2-opt improvement heuristic
  - Generate *k* random initial tours (city permutations)
  - Iteratively improve tours until local minimum reached
- In each iteration, apply best possible 2-opt *move*
  - Find best *pair* of edges (i,i+1) and (j,j+1) such that replacing them with (i,j) and (i+1,j+1) minimizes tour length



2-opt Pseudo Code (Time Critical Part)

```
// city[i] is ith city (permutation array)
#define dist(a,b) dmat[city[a]][city[b]]
do {
  minchange = 0;
  for (i = 0; i < cities-2; i++) {
    for (j = i+2; j < cities; j++) {
      change = dist(i,j) + dist(i+1,j+1)
        - dist(i,i+1) - dist(j,j+1);
      if (minchange > change) {
        minchange = change;
        mini = i;  minj = j;
  } } }
  // apply mini/minj move
} while (minchange < 0);
```

Distance matrix: $O(n^2)$ time and space

Doubly-nested for loop to visit $O(n^2)$ edge pairs

It suffices to compute change in length: $O(1)$ time

O(n) iterations needed to reach local minimum: overall algorithm is $O(n^3)$

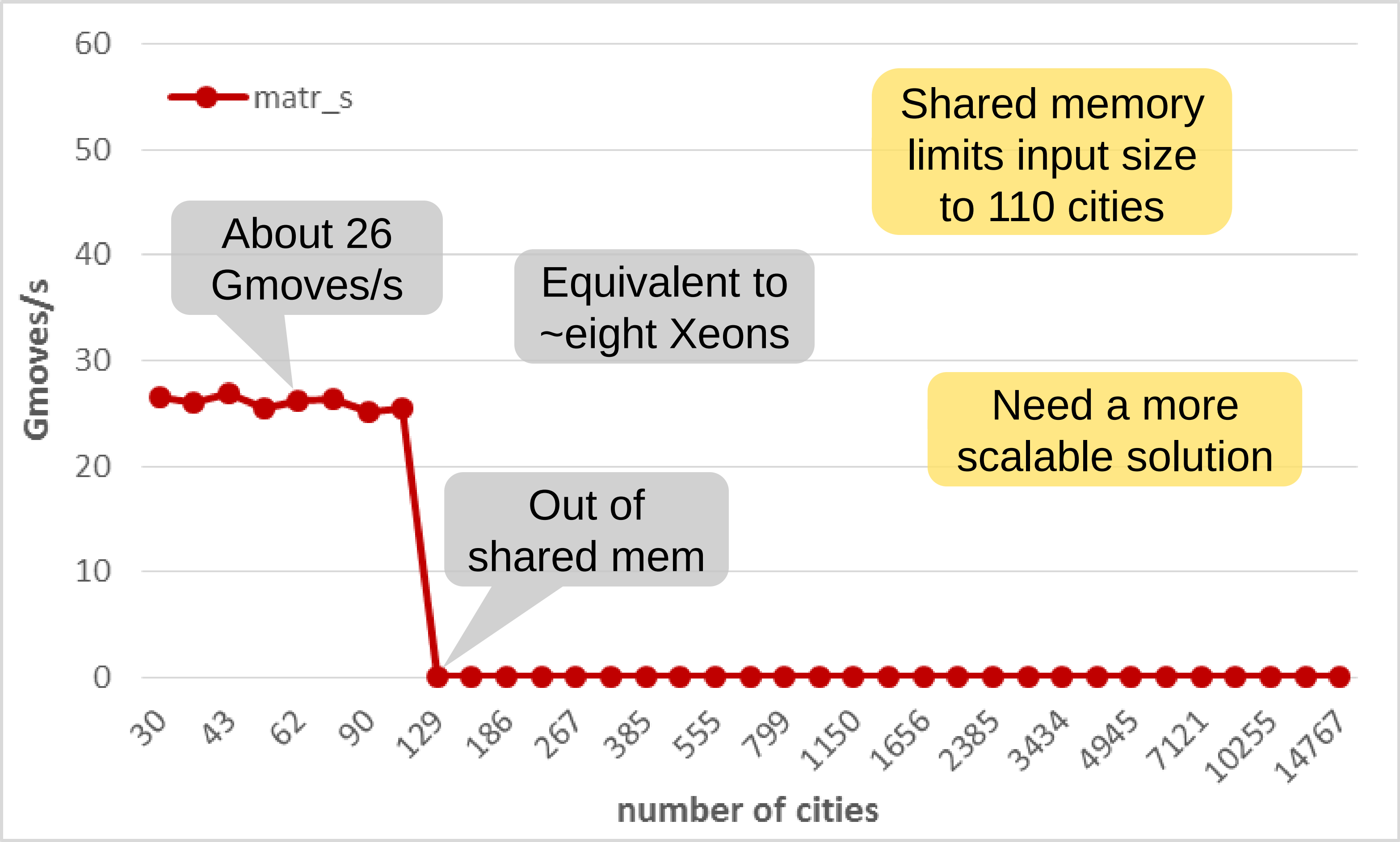
Methodology

- System
  - nvcc v5.5 with "-O3 -arch=sm_35 -use_fast_math"
  - K40 GPU with 15 SMs & 2880 PEs (Kepler based)
- Metric
  - Throughput in billion moves per second (best of 3 runs)
- Inputs
  - First *n* points of "d18512.tsp" from TSPLIB (plus others)
  - *k* random initial tours + 2-opt to find local minimum
  - Select *k* s.t. SMs fully loaded and runtime ≥ 1 second

Distance Matrix in Shared Memory

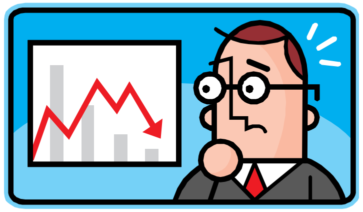
- Algorithm 1 (published in 2011 by M. O'Neil)
  - Each thread applies 2-opt to a different initial tour
  - Need city arrays to record tour (in local memory)
  - Pre-compute distance matrix and store in shared mem
- Benefit
  - Local memory accesses are cached and coalesced
  - Distance lookup accesses go to shared memory
- Drawbacks
  - Limited by shared memory size of 48 kB
  - Random matrix accesses result in bank conflicts

Throughput (Dist Matrix in Shmem)



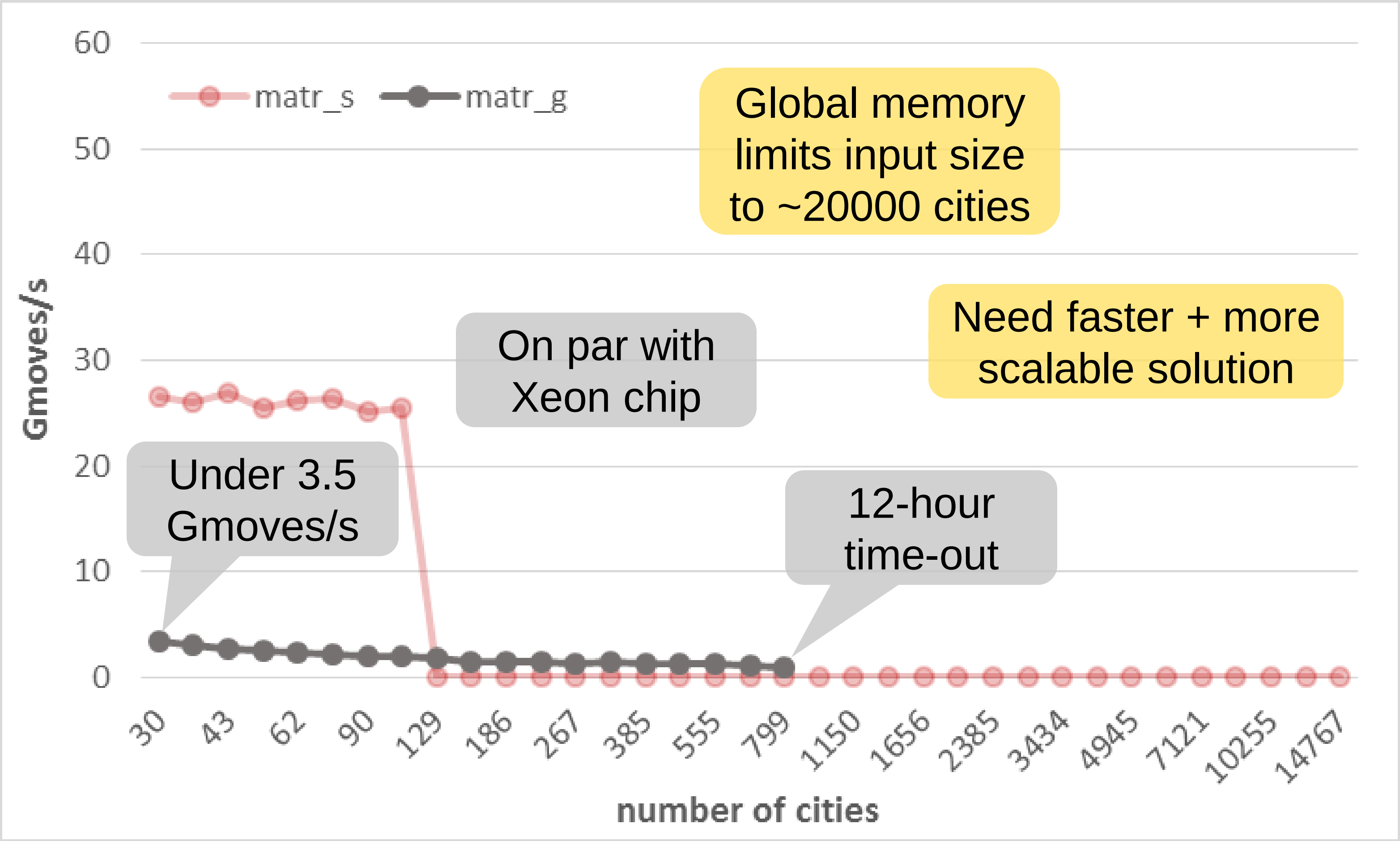
Distance Matrix in Global Memory

- Algorithm 2 (based on Algorithm 1)
  - Pre-compute distance matrix and store in global mem
  - Still need city arrays to record tour (in local memory)
- Benefit
  - Not limited by shared memory capacity



- Drawback
  - Random matrix accesses result in poor cache performance and uncoalesced accesses

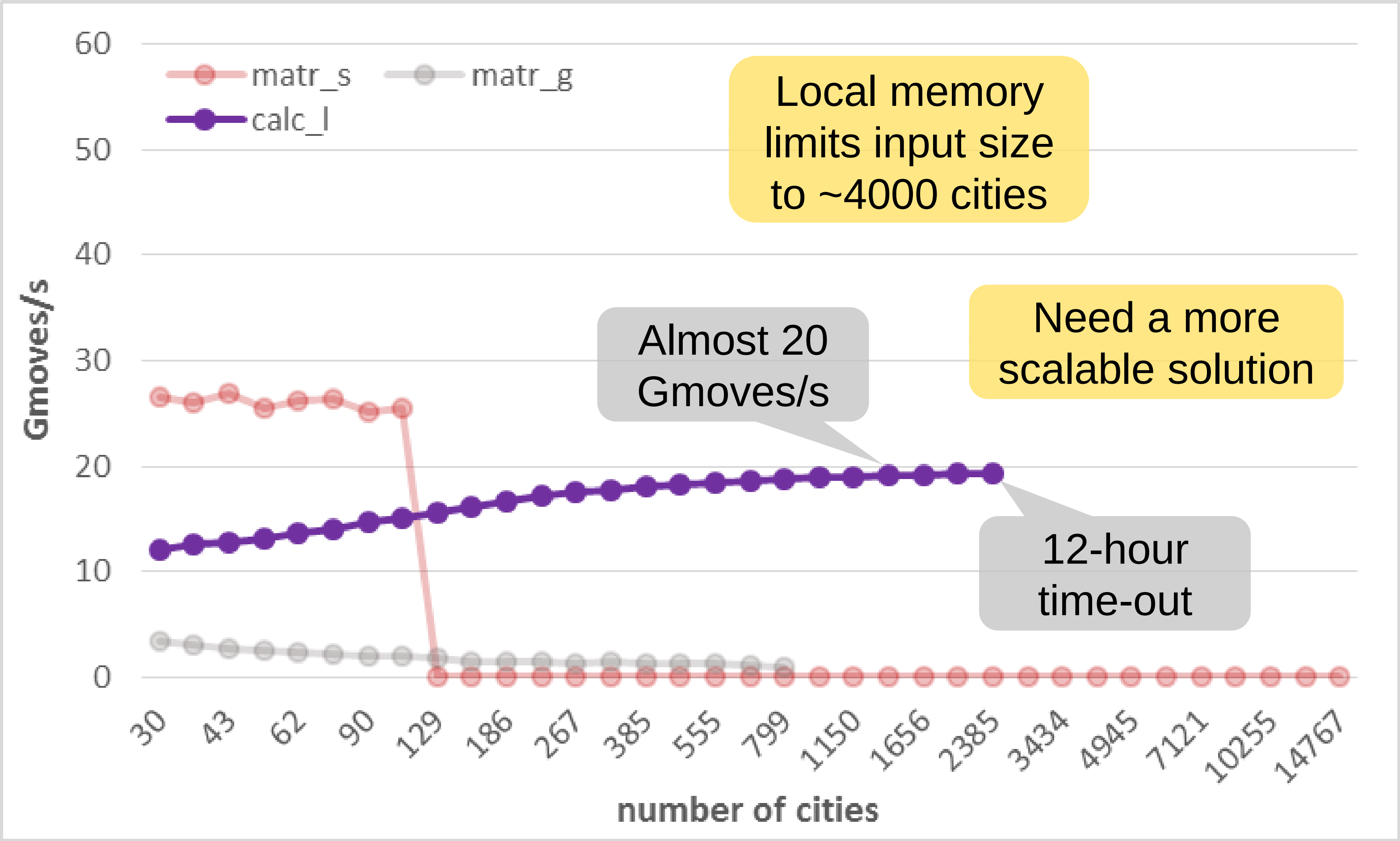
Throughput (Dist Matrix in Glob Mem)



Re-calculating the Distances

- Algorithm 3 (presented at GTC 2012 by K. Rocki)
  - Compute distances rather than looking them up
  - Copy city coordinates into local memory
  - No need for city arrays, permute coords directly
- Benefits
  - O(n) memory usage, coalesced memory accesses
- Drawbacks
  - Limited by local memory size (performance degrades)
  - Large k needed: $k \geq 30720$ to fully utilize K40 GPU

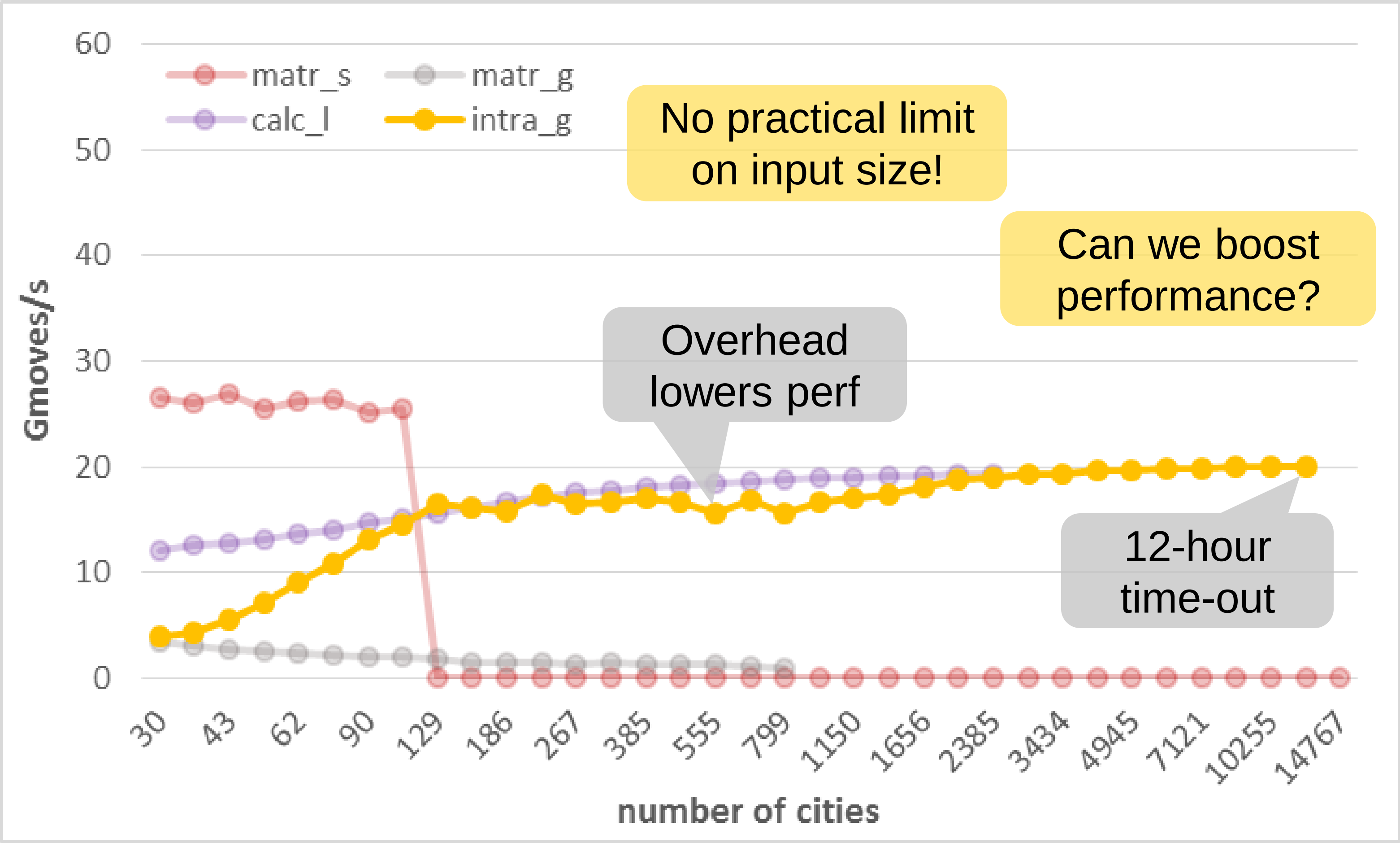
Throughput (Calculating Distances)



Intra Parallelization of 2-opt Step

- Algorithm 4 (Algo 3 + hierarchical parallelization)
  - Assign tours to thread blocks instead of threads
  - Parallelize 2-opt computation across threads
    - Distribute outer *for* loop across threads in each thread block
    - Requires parallel prefix scans, __syncthreads(), etc.
- Benefits
  - Memory usage per block is greatly reduced
  - Latency to find local minimum of a tour is much smaller
- Drawbacks
  - Complexity of implementation, small performance drop

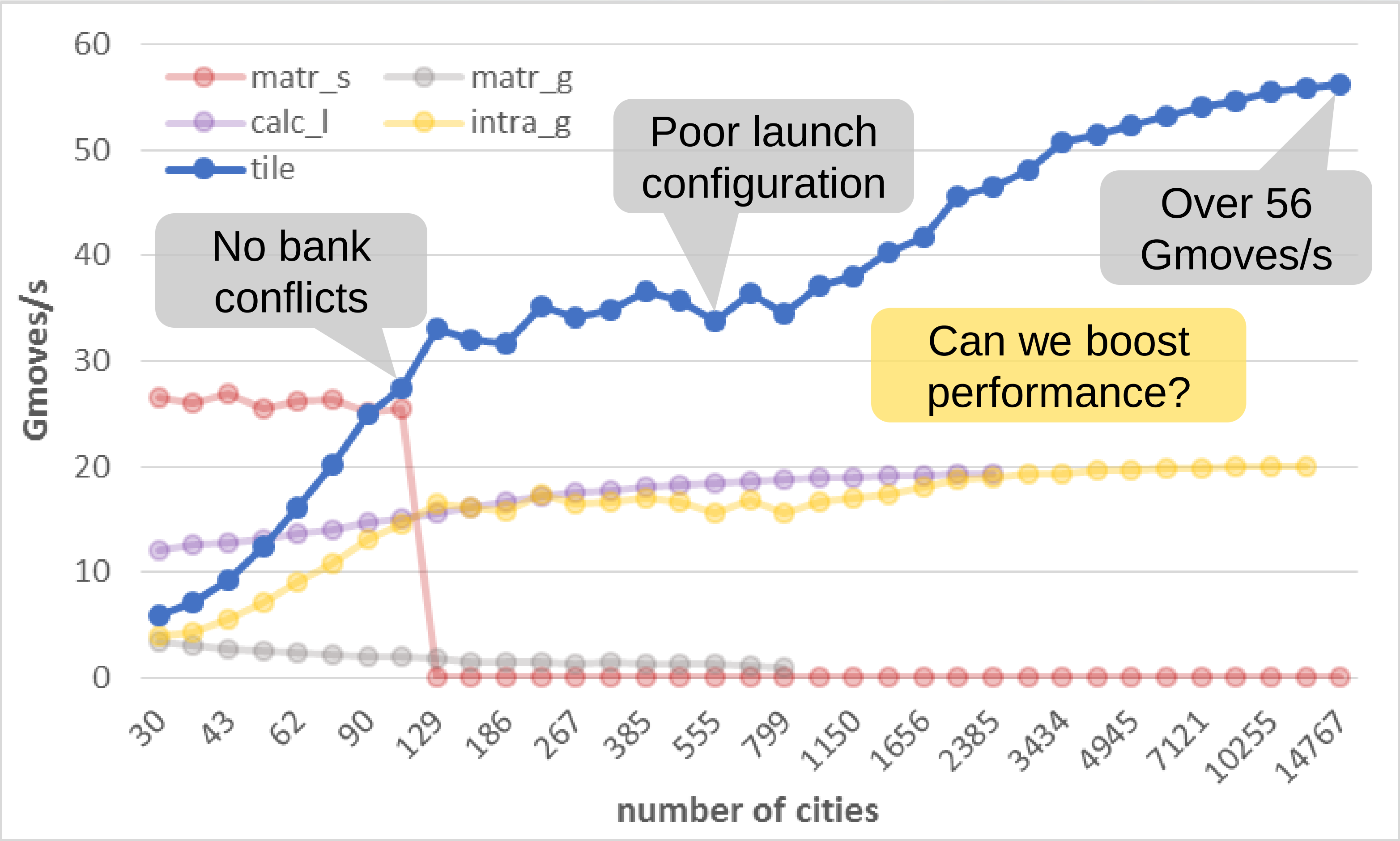
Throughput (Intra-2-opt Parallelization)



Intra-2-opt Parallelization with Tiling

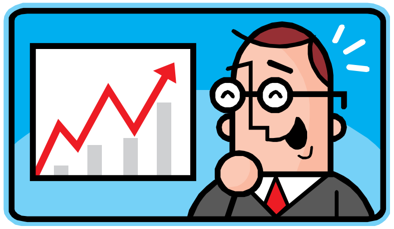
- Algorithm 5 (Algo 4 + tiling in shared memory)
  - Break up computation into chunks such that each chunk's working set fits into shared memory
  - Load data into shared memory before each chunk
    - Works beautifully after reversing inner *for* loop
- Benefits
  - Most accesses go to shared memory due to reuse
  - No bank conflicts, full coalescing
- Drawbacks
  - Complexity of implementation

Throughput (with Tiling)



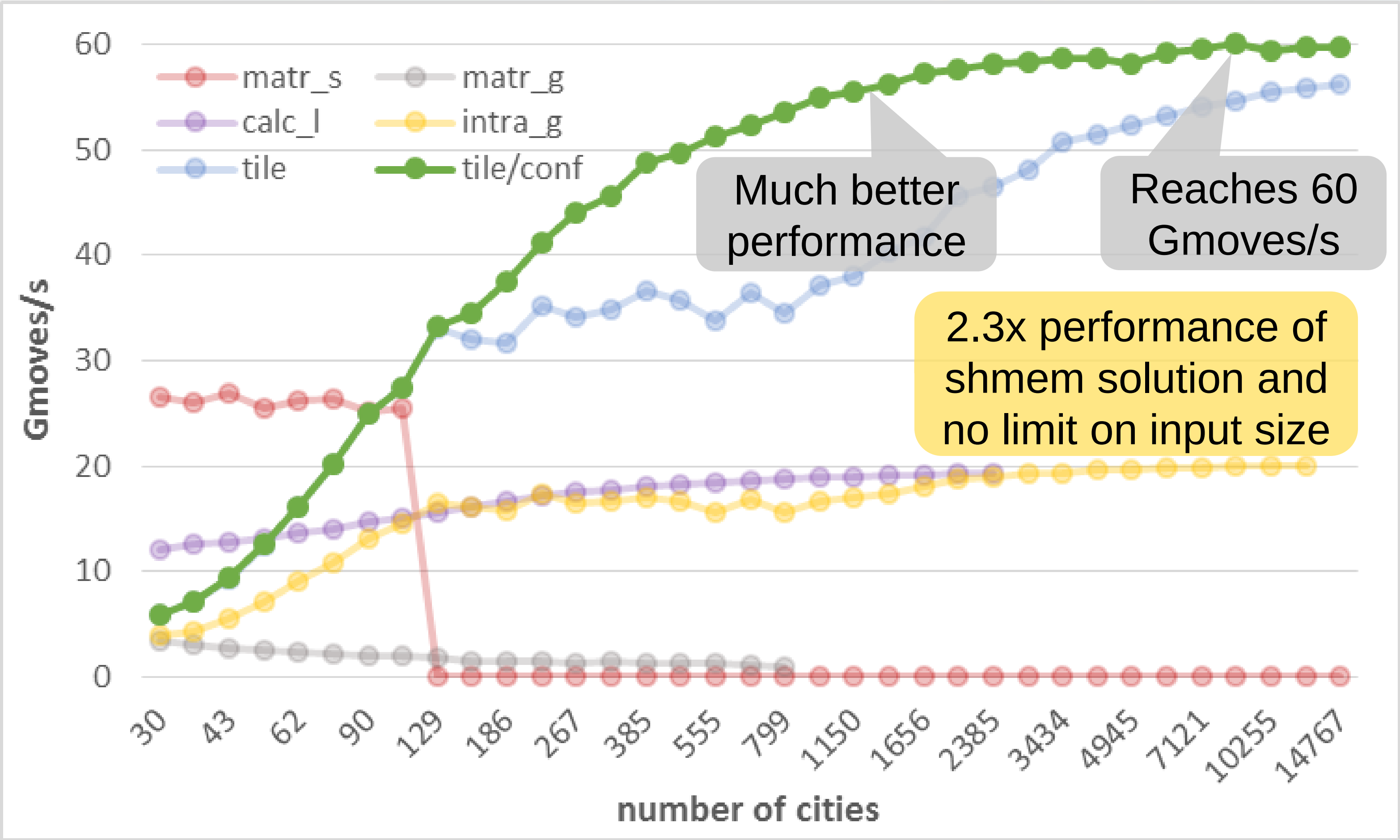
Launch Configuration Optimization

- Algorithm 6 (Algorithm 5 + tuned configuration)
  - Compute optimal thread count per block
    - Based on number of cities, shared memory use, max threads per block, and max blocks per SM (32 registers per thread)
  - Launch kernel with computed thread count per block
- Benefits
  - Maximizes hardware usage



- Drawbacks
  - None (need to write code to evaluate configurations)

Throughput (Tuned Configuration)



Summary and Conclusions

- Fast CUDA implementation of 2-opt TSP solver
  - Over 2x faster than prior solutions, no problem-size limit
  - Interesting optimizations (e.g., compute best launch configuration, reverse loop to enable coalescing & tiling)
- Conclusions
  - Rethinking implementation and parallelization strategy to better exploit GPU hardware may pay off
- CUDA source code is available at
  http://www.cs.txstate.edu/~burtscher/research/TSP_GPU/

Acknowledgments

- Collaborator
  - Kamil Rocki, IBM
- U.S. National Science Foundation
  - DUE-1141022, CNS-1217231, and CNS-1305359
- NVIDIA Corporation
  - Grants and equipment donations
- Texas Advanced Computing Center
  - K40 GPUs in Maverick system