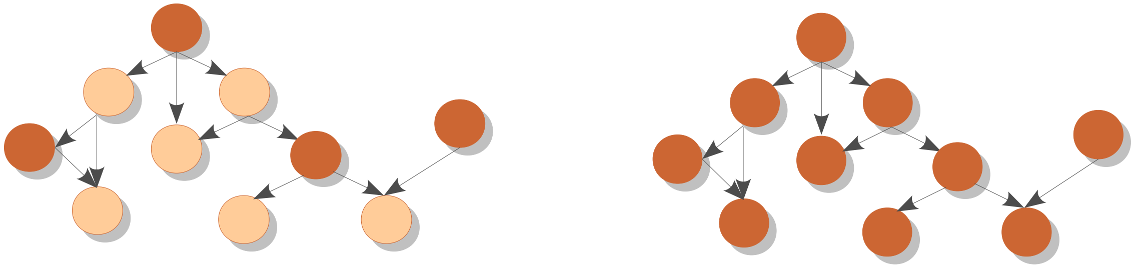
Data-driven versus Topology-driven Irregular Computations on GPUs



Rupesh Nasre
University of Texas
Austin, USA

Martin Burtscher
Texas State University
San Marcos, USA

Keshav Pingali
University of Texas
Austin, USA

IPDPS 2013

What is IrReguLariTy?

- Data-access or control patterns are unpredictable at compile time.

Irregular data-access

```
int a[N], b[N], c[N];
readinput(a);

c[5] = b[a[4]];
```

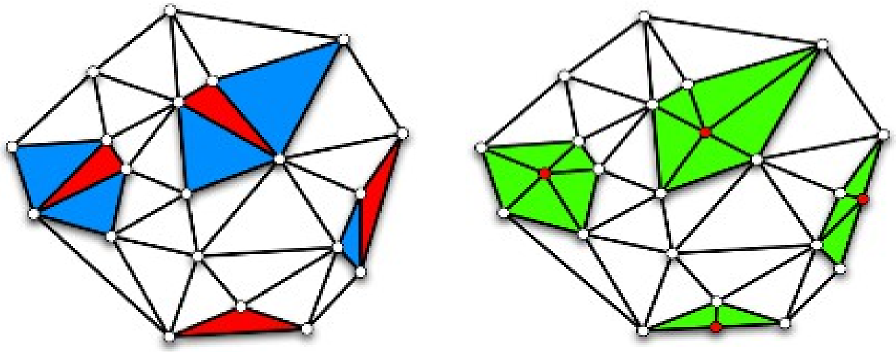
Irregular control-flow

```
int a[N];
readinput(a);

if (a[4] > 30) {
        ...
}
```

Needs dynamic techniques

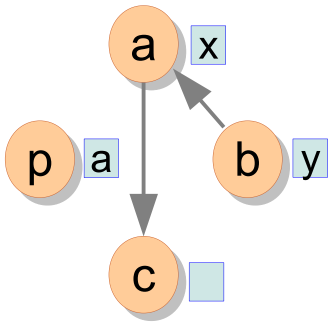
Pointer-based data structures often contribute to irregularity.

Examples of Irregular Programs

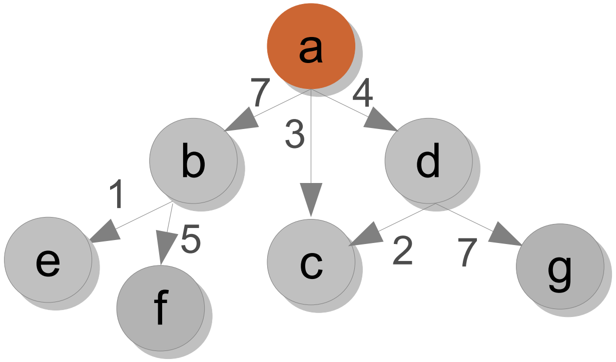


Delaunay Mesh Refinement

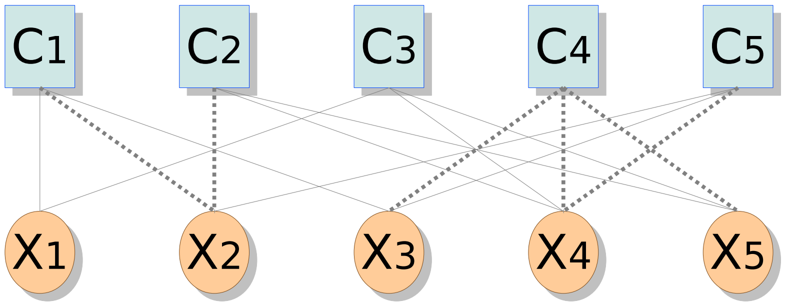
```
a = &x
b = &y
p = &a
*p = b
c = a
```



Points-to Analysis

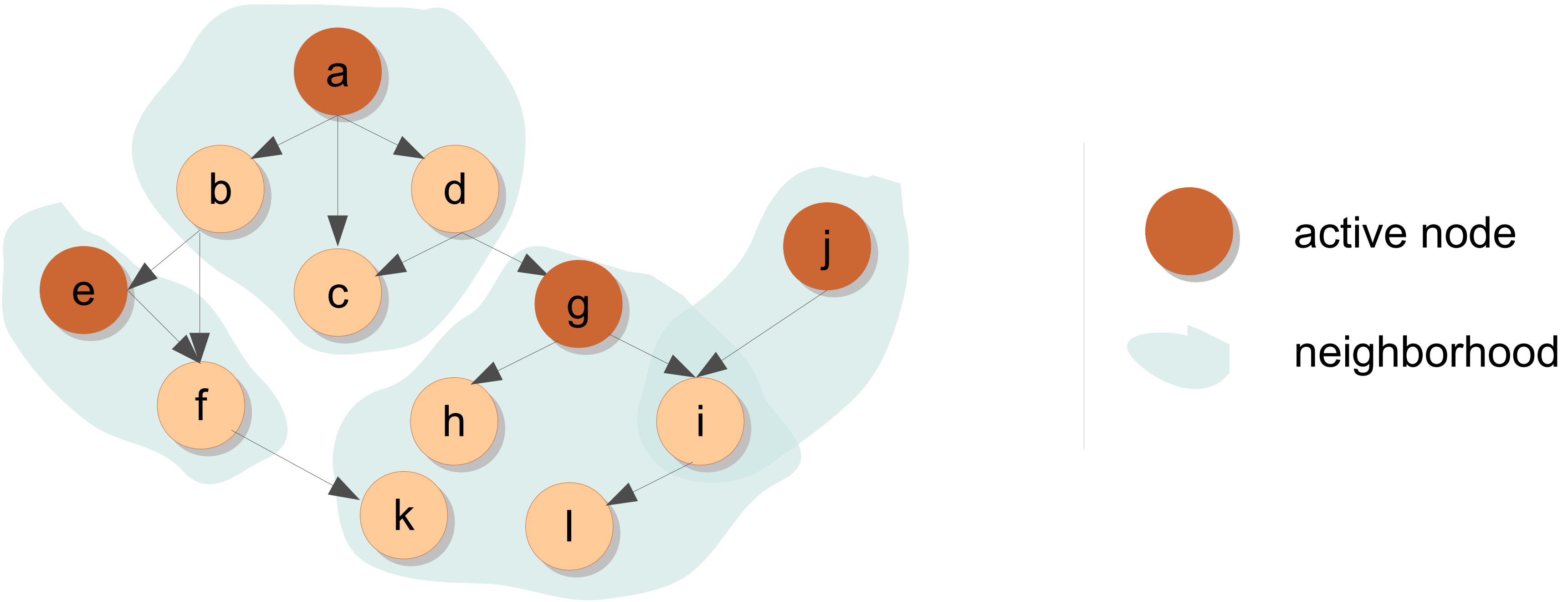


Single-source shortest paths



Survey Propagation

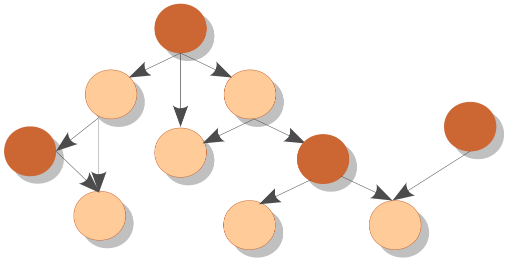
Operator Formulation



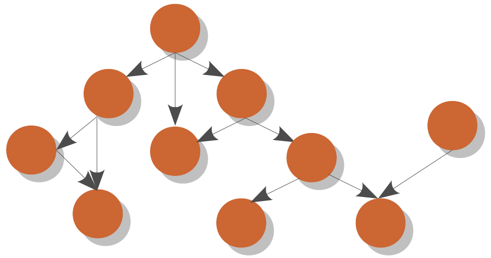
- Computation is modeled as an iterated application of an operator
- **Data-driven processing**: operator is applied only at the nodes where there might be work to be done
- **Topology-driven processing**: operator is applied at all the nodes even if there is no work to do at some nodes

Focus of This Work

- **Study and optimize** topology-driven and data-driven irregular computations on GPUs
    - On GPUs, irregular algorithms are usually implemented in a topology-driven manner
    - On multi-core CPUs, irregular algorithms are usually implemented in a data-driven manner
- **Our findings**
    - Data-driven versions usually perform better
    - Topology-driven versions perform better if additional algorithmic properties can be exploited
    - A hybrid approach can outperform the two



data-driven



topology-driven

Outline

- Motivation

➔ **Data-driven approach**

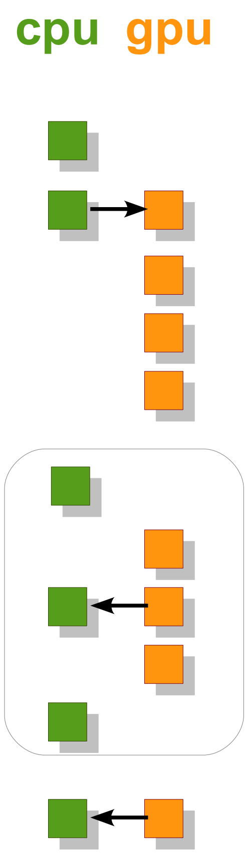
- Topology-driven approach
- Hybrid approach
- Evaluation
- Conclusions

Outline

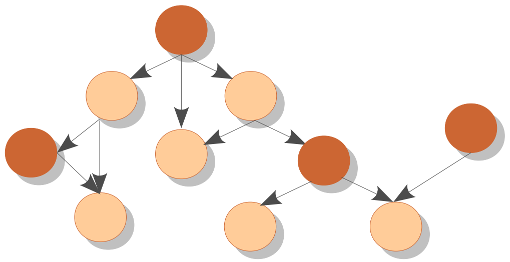
- Motivation
- ➔ **Data-driven approach**
  - dual-worklist
  - hierarchical worklists
  - work-chunking
  - atomic-free worklist update
  - work-donation
  - variable kernel configuration
- Topology-driven approach
- Hybrid approach
- Evaluation
- Conclusions

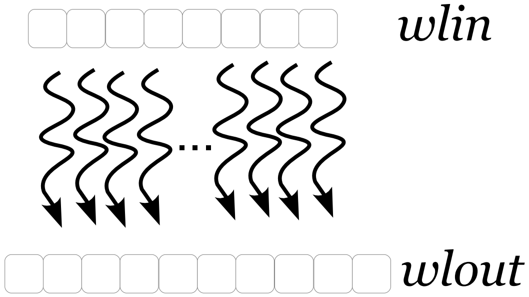
Data-driven: Base Version

```
main {
    read input
    transfer input
    initialize_kernel
    initialize_worklist(wlin)
    clear wlout

    while wlin not empty {
        operator(wlin, wlout, ...)
        transfer wlout size
        clear wlin
        swap(wlin, wlout)
    }
    transfer results
}
```

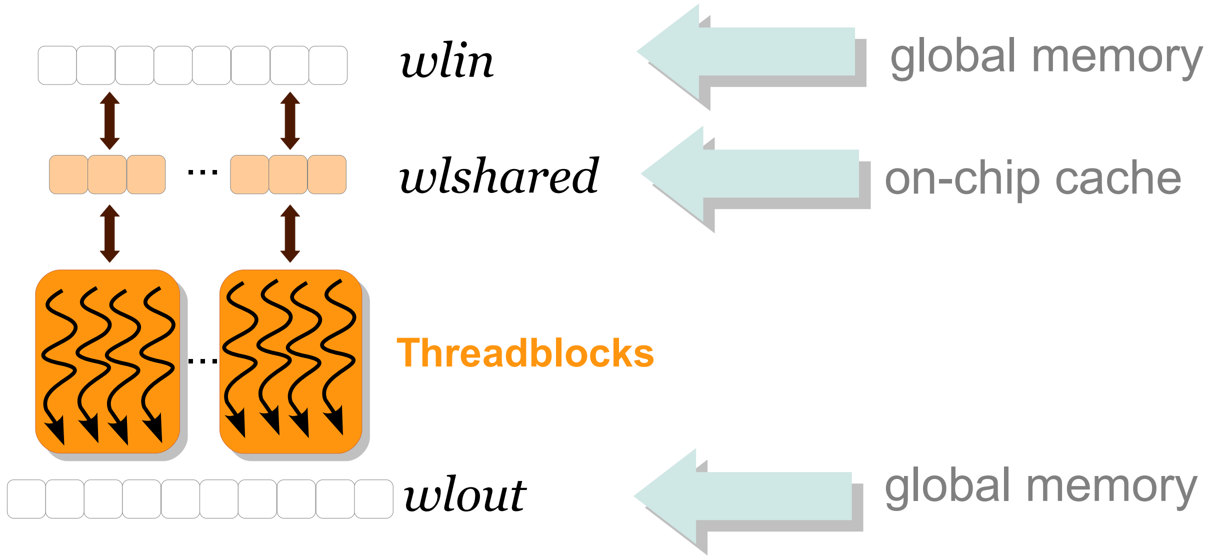


```
sssp_operator(wlin, wlout, ...) {
    src = wlin[...]
    dsrc = distance[src]
    forall edges (src, dst, wt) {
        ddst = distance[dst]
        altdist = dsrc + wt

        if altdist < ddst
            distance[dst] = altdist
            wlout.push(dst)
} }
```



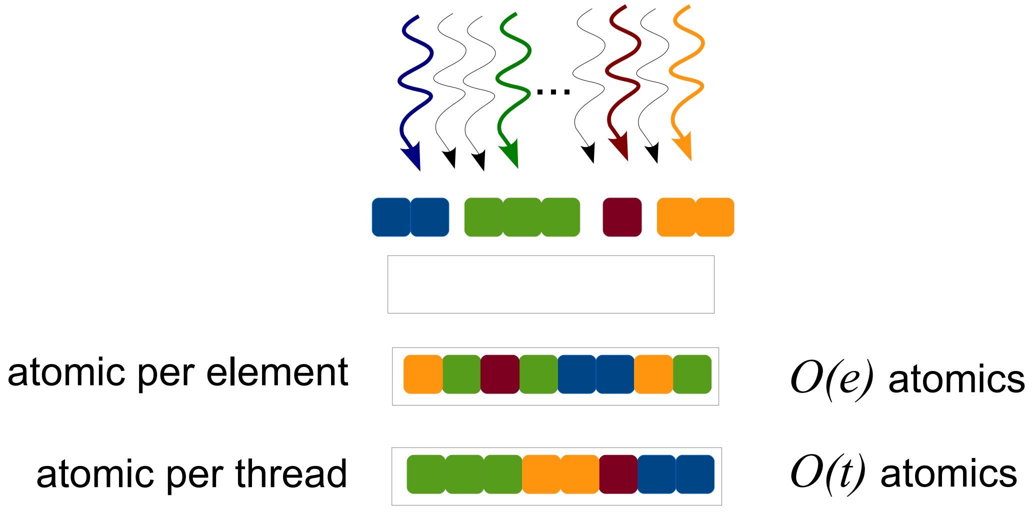


Data-driven: Hierarchical Worklist



- Worklist exploits memory hierarchy
- Makes judicious use of limited on-chip cache

Data-driven: Work Chunking

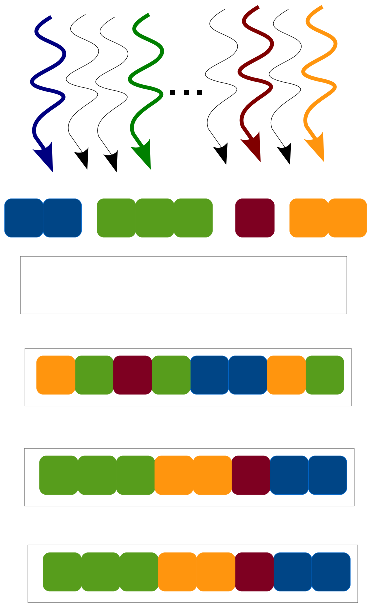


atomic per element $O(e)$ atomics

atomic per thread $O(t)$ atomics

- Reserves space for multiple work-items in a single atomic
- May reduce overall synchronization

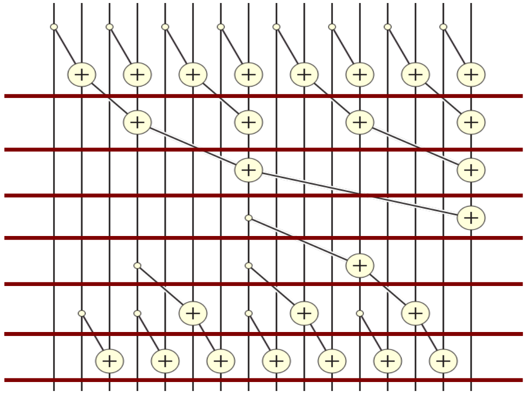
Data-driven: Atomic-free Worklist Update



atomic per element — $O(e)$ atomics

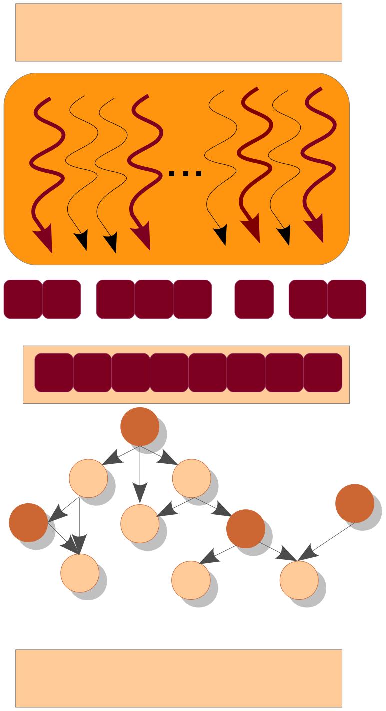
atomic per thread — $O(t)$ atomics

prefix-sum — $O(\log t)$ barriers



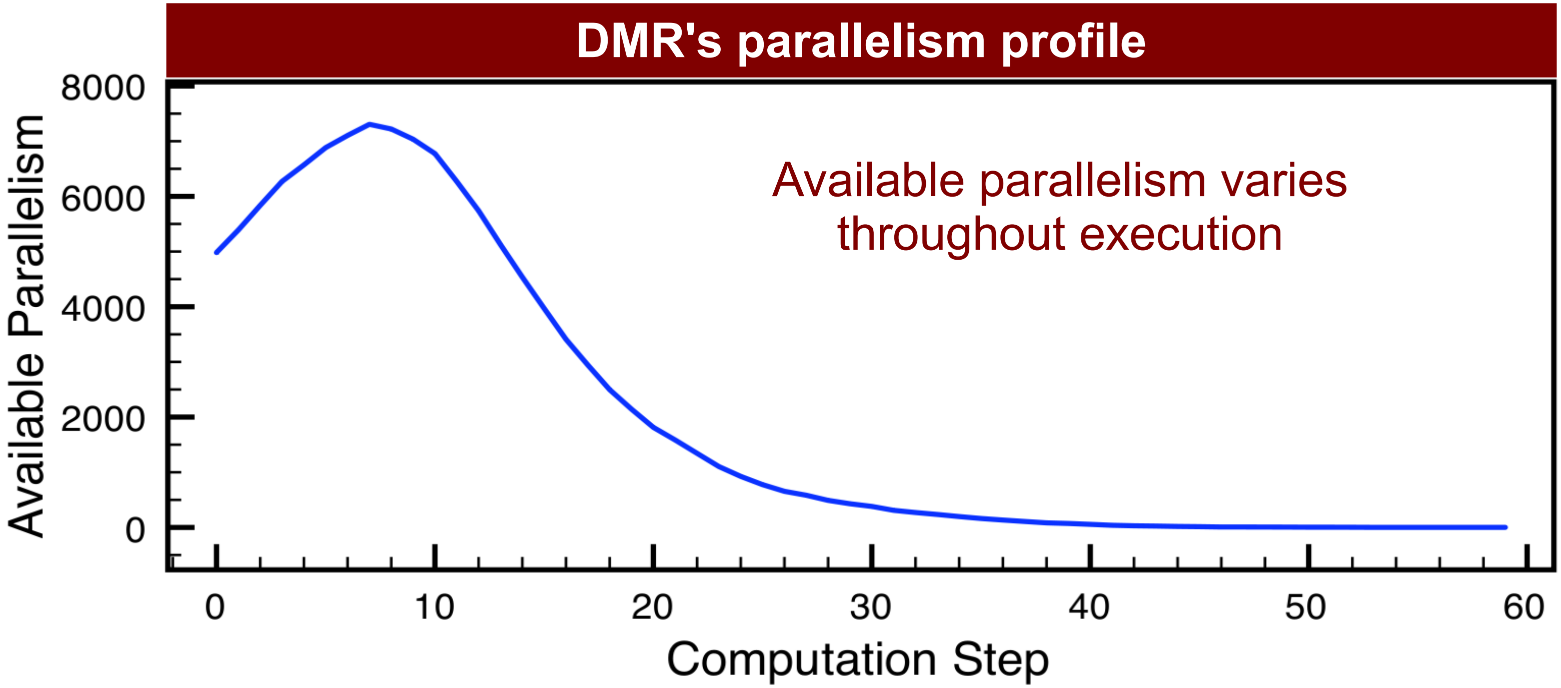
Data-driven: Work Donation

```
donate_kernel {
        shared donationbox[...];


        // determine if I should donate
        --barrier--

        // donate
        --barrier--

        // operator execution


        // empty donation box

}
```



- Work-donation improves load balance

Data-driven: Variable Kernel Configuration



- Varying configuration improves work-efficiency
- It also reduces conflicts and may improve performance

Outline

- Motivation
- Data-driven approach

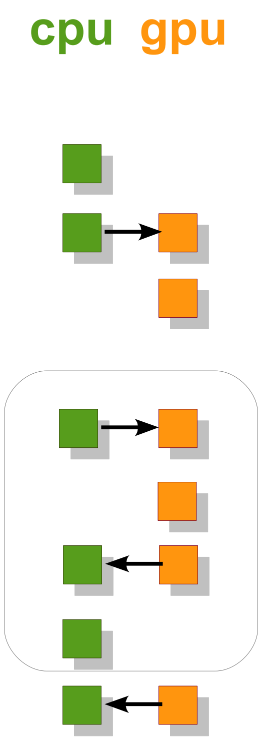
→ **Topology-driven approach**

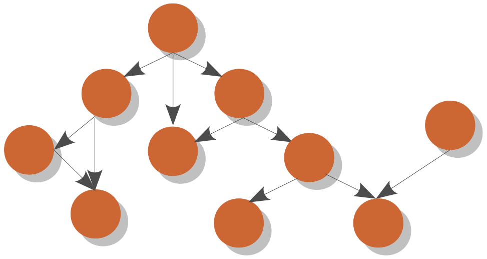
  - no global worklists
  - kernel unrolling
  - exploiting memory hierarchy
  - improved memory layout

- Hybrid approach
- Evaluation
- Conclusions

Topology-driven: Base Version

```
main {
    read input
    transfer input
    initialize_kernel
    do {
        transfer false to changed
        operator(...)
        transfer changed
    } while changed
    transfer results
}
```

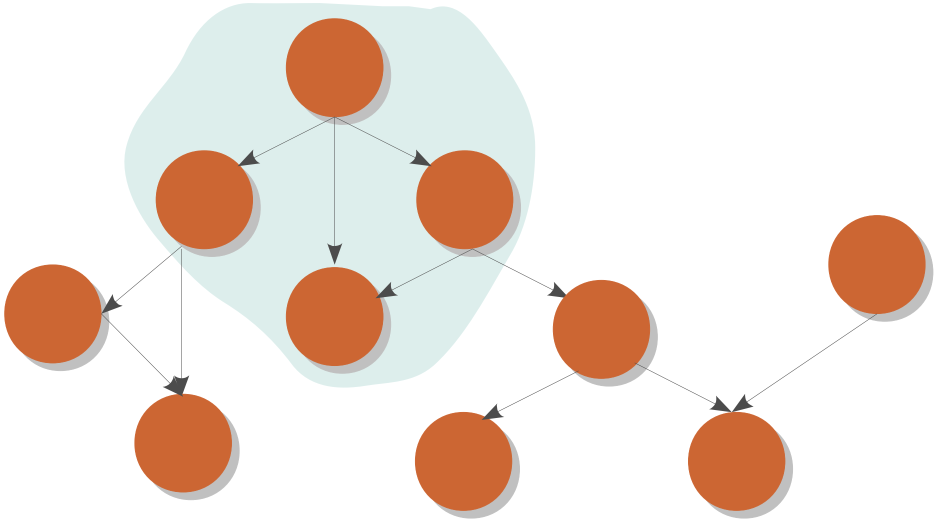




Topology-driven: Kernel Unrolling

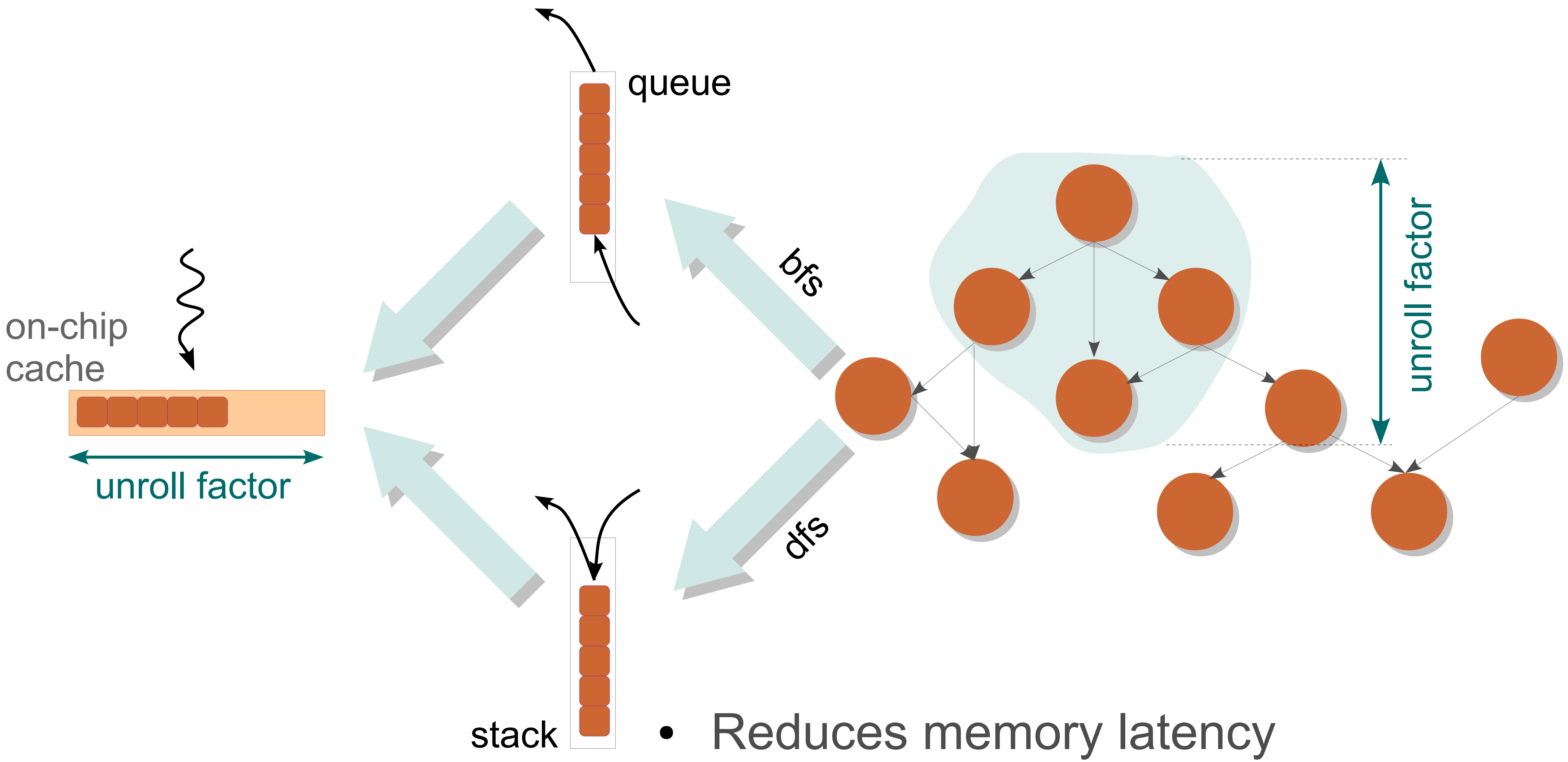
```
sssp_operator(src) {
    dsrc = distance[src]

    forall edges (src, dst, wt) {
        ddst = distance[dst]
        altdist = dsrc + wt

        if altdist < ddst
            distance[dst] = altdist
    }
}
```

Memory-bound kernel



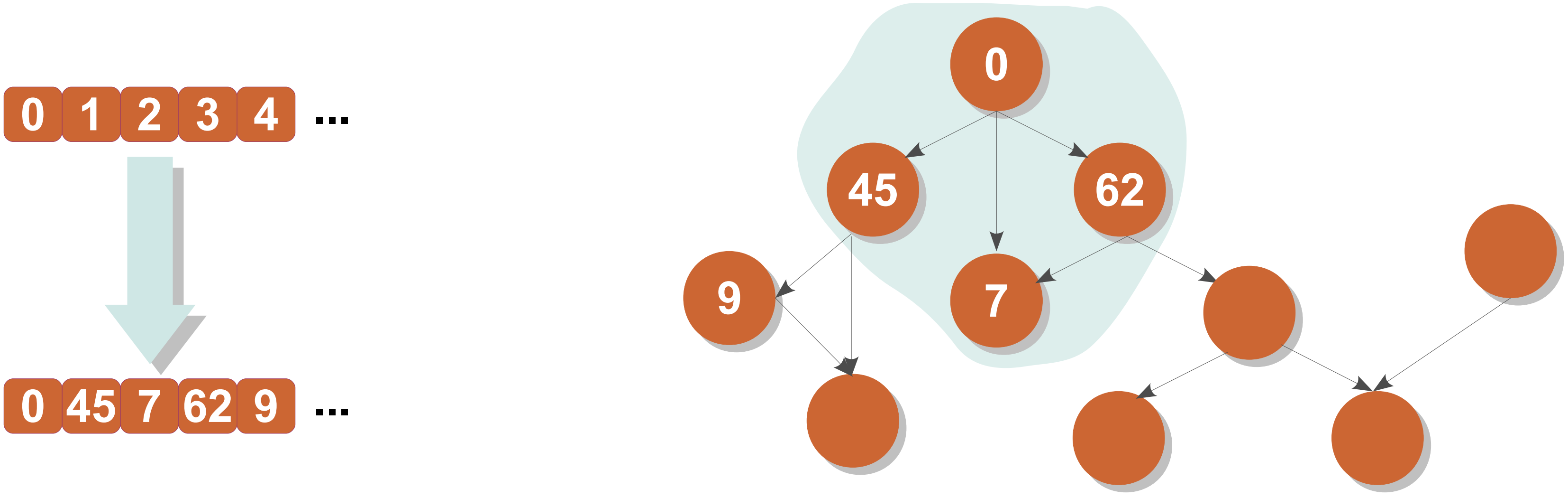
- Improves amount of computation per thread invocation
- Need to ensure absence of races
- Propagates information faster

Topology-driven: Exploiting Memory Hierarchy



- Reduces memory latency
- Requires careful selection of unroll factor

Topology-driven: Improved Memory Layout



- Bring logically close graph nodes also physically close in memory
- Improves spatial locality

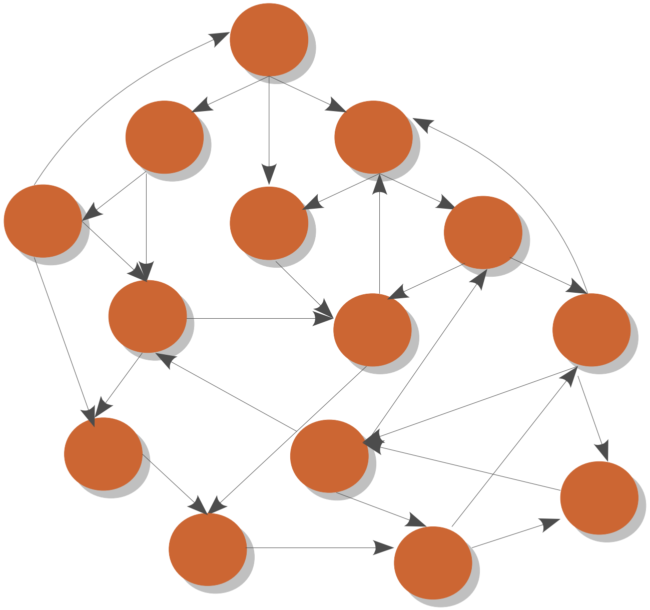
Outline

- Motivation
- Data-driven approach
- Topology-driven approach

➔ **Hybrid approach**

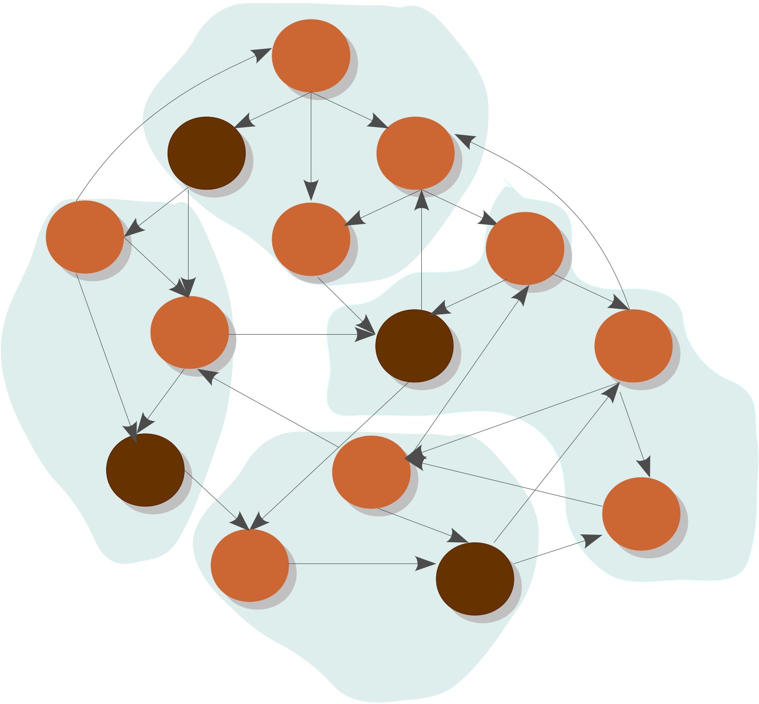
  - spatial
  - temporal
- Evaluation
- Conclusions

Spatial Hybrid

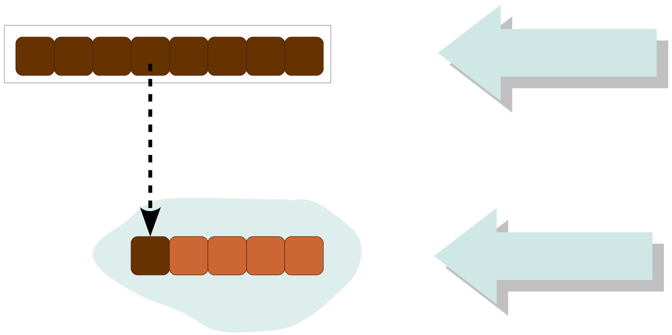


Shortest paths computation

Spatial Hybrid



Shortest paths computation



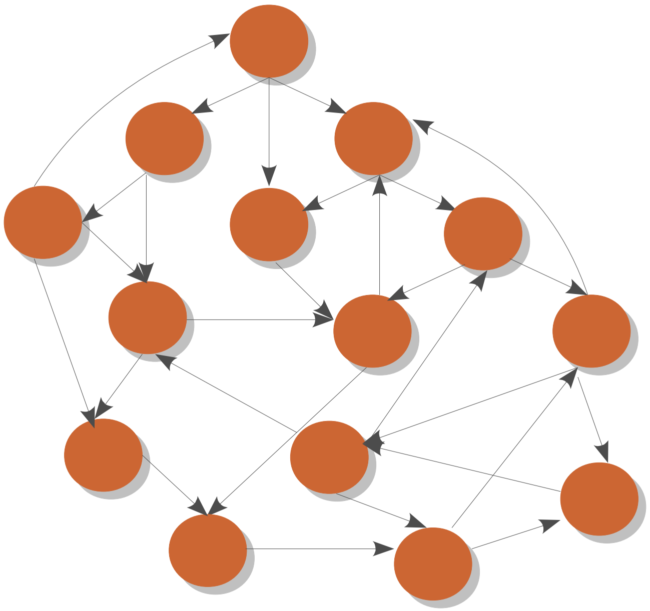
worklist

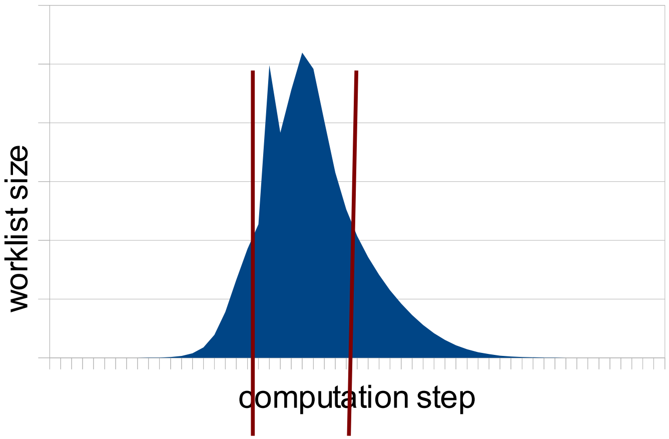
data-driven

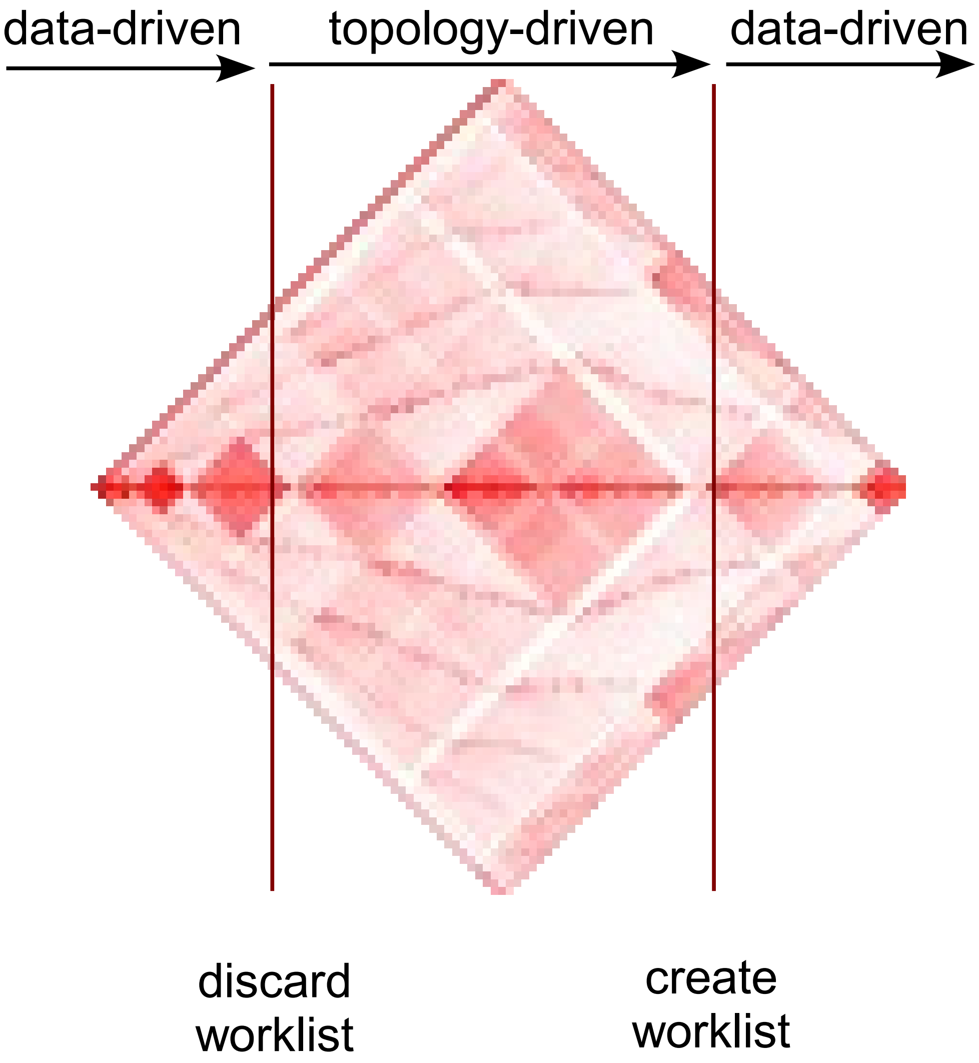
topology-driven

process all nodes in the partition

Temporal Hybrid







data-driven

topology-driven

data-driven

discard worklist

create worklist

Outline

- Motivation
- Data-driven approach
- Topology-driven approach
- Hybrid approach

➔ **Evaluation**

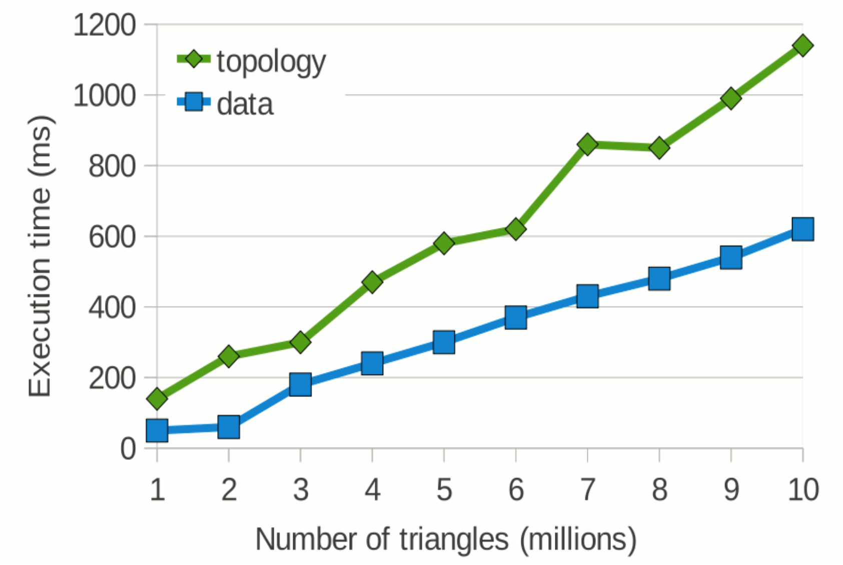
  - Fermi (Quadro 6000, 1.45 GHz, 448 cores over 14 SMs)
  - **LonestarGPU** benchmark suite

    http://iss.ices.utexas.edu/?p=projects/galois/lonestargpu

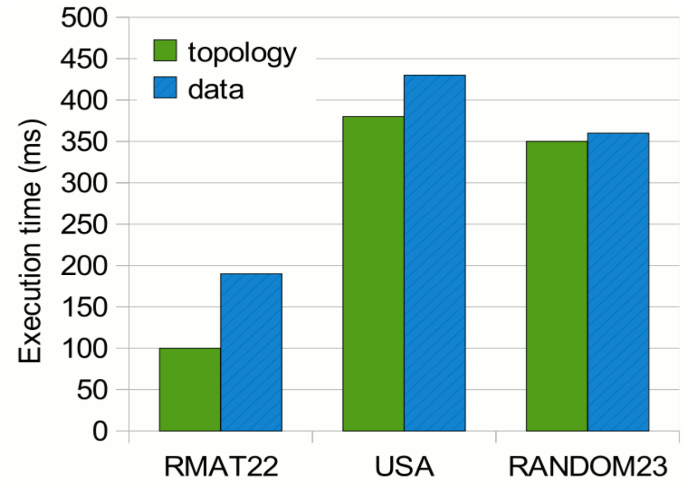
- Conclusions

Data-driven vs. Topology-driven

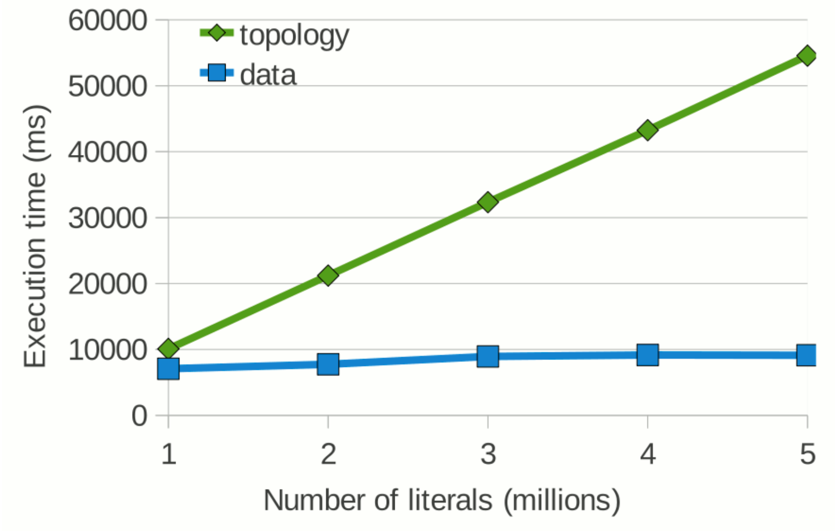
DMR



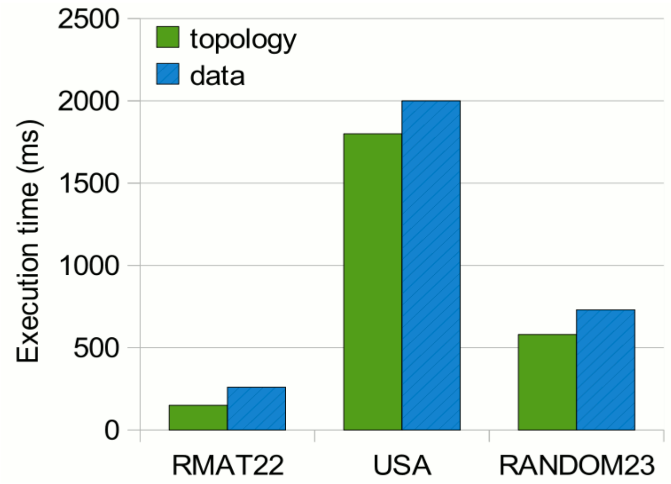
BFS



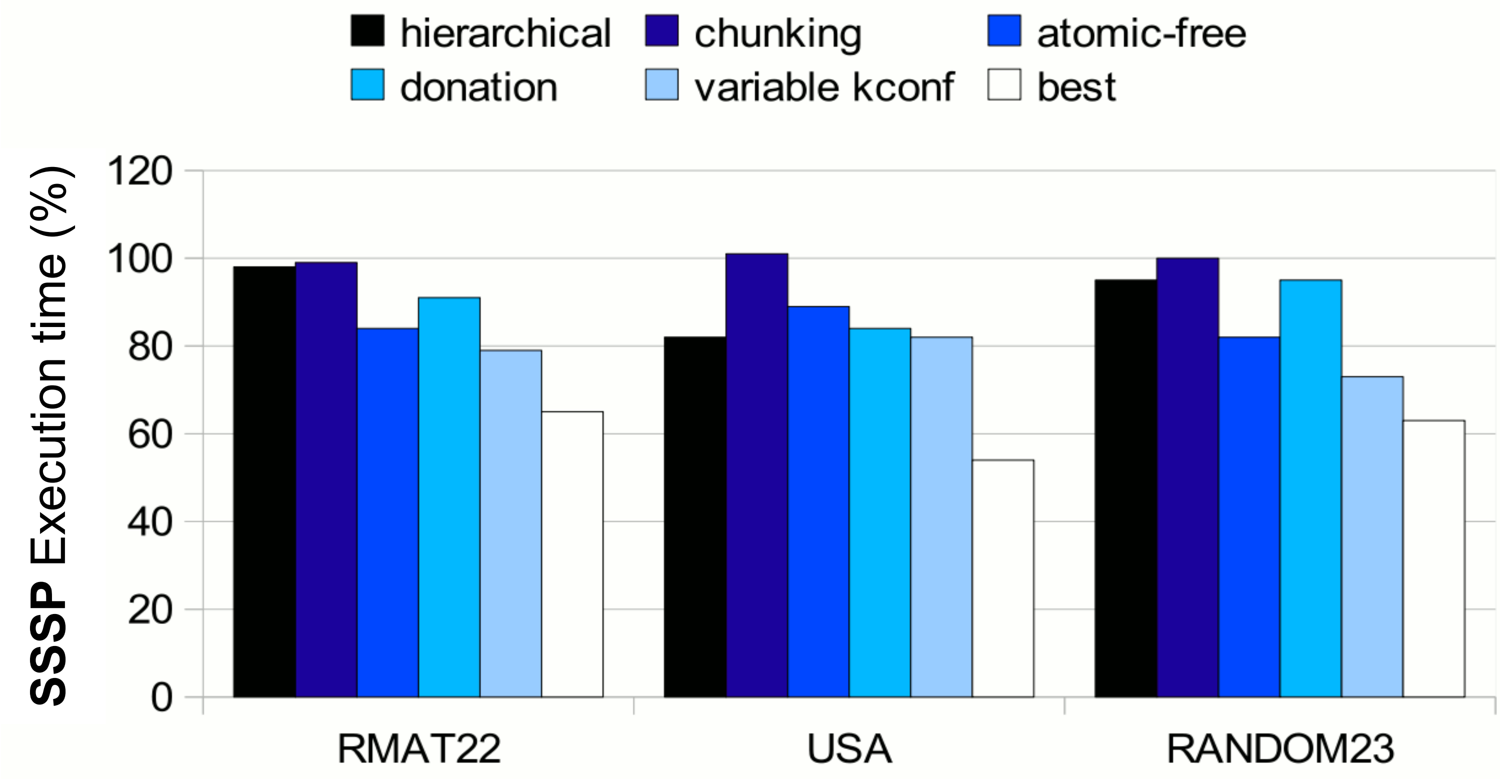
SP



SSSP

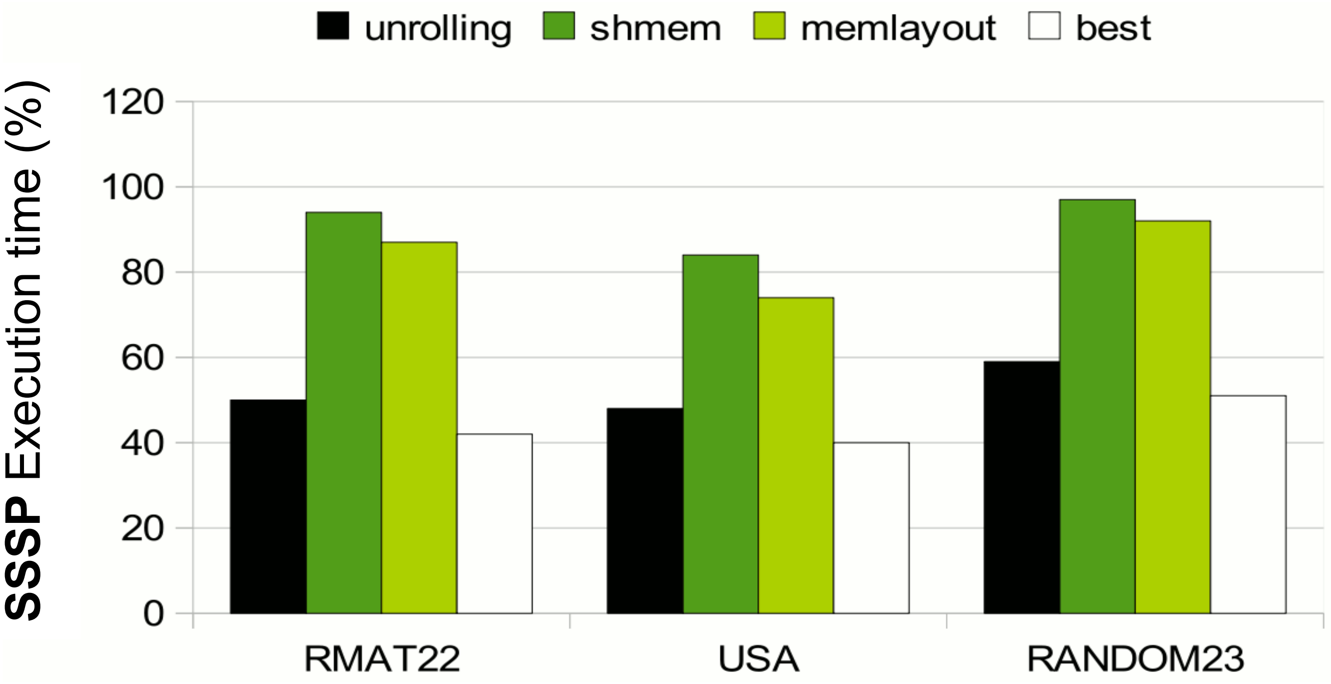


Data-driven Optimizations



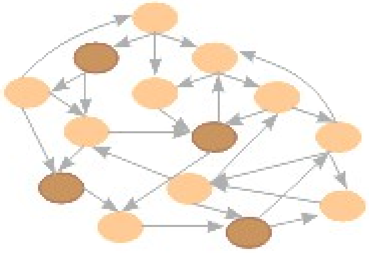
- Hierarchical worklists, atomic-free worklist update and varying configuration help
- Chunking is not effective
- Overall, performance improves by 40-50%

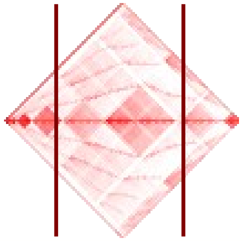
Topology-driven Optimizations

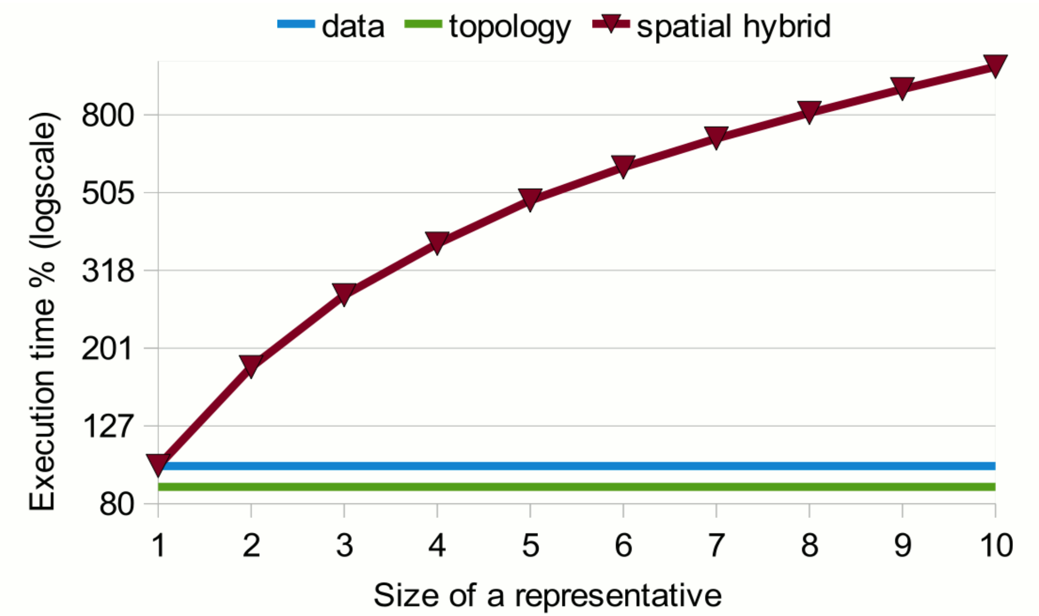


- Effect is more pronounced than the data-driven approach
- Unrolling helps the most
- Overall, performance improves by 50-60%

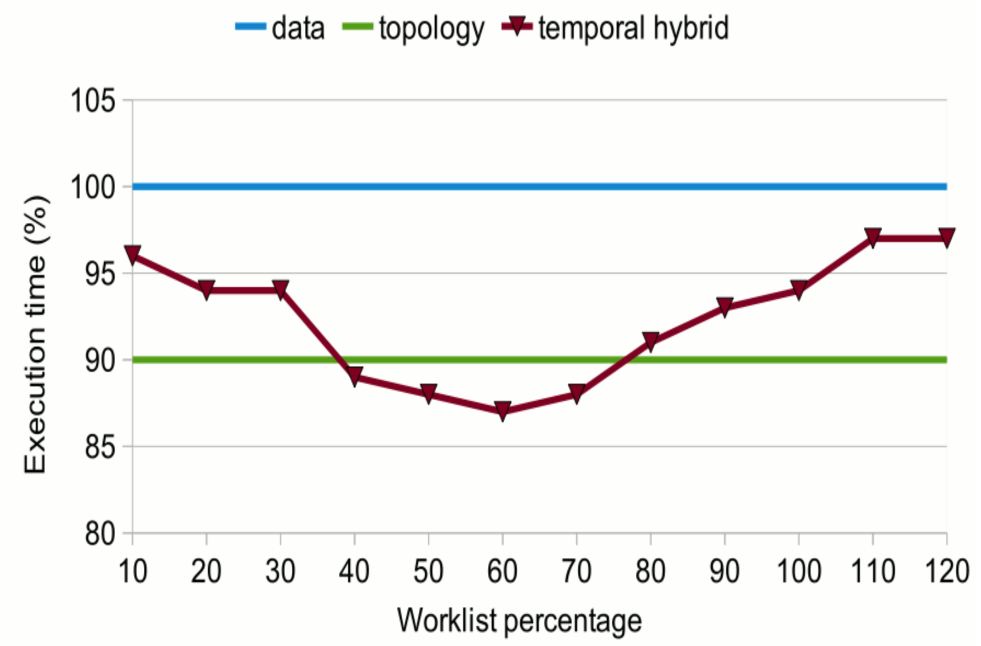
Spatial and Temporal Hybrid







SSSP

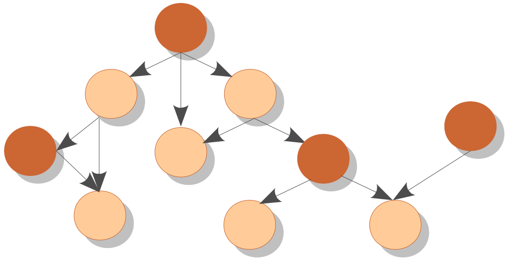


SSSP

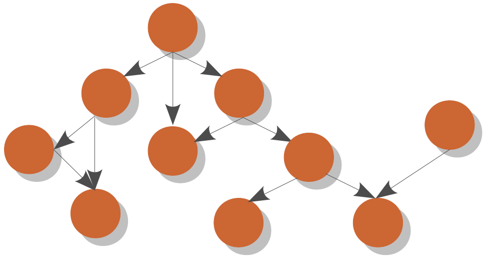
- Spatial hybrid has consistently reduced performance
- Temporal hybrid achieves performance better than the solo approaches

Conclusions

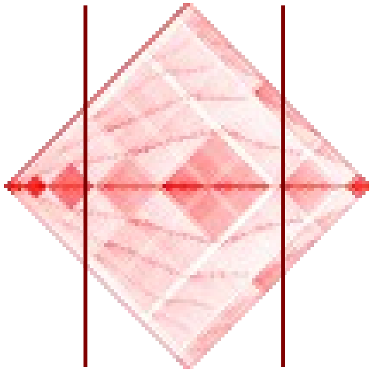
- Data-driven irregular computations usually perform better



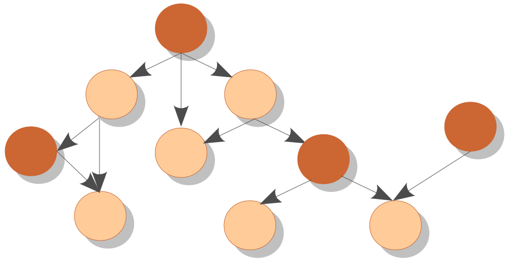
- Topology-driven irregular computations perform better if additional algorithmic properties can be exploited

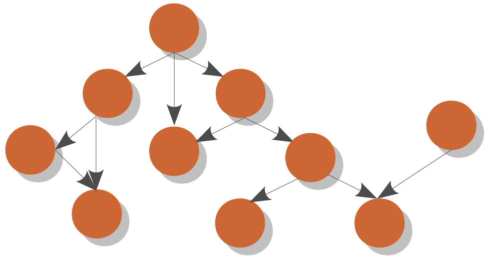


- Hybrid approaches may provide better performance than the individual approaches



Data-driven versus Topology-driven Irregular Computations on GPUs





Rupesh Nasre
University of Texas
Austin, USA

Martin Burtscher
Texas State University
San Marcos, USA

Keshav Pingali
University of Texas
Austin, USA

Acknowledgments: Intel, NSF, NVIDIA, Qualcomm

LonestarGPU

ECL
Efficient Computing Laboratory

Galois

DMR Performance

SP Performance

BFS Performance

SSSP Performance