Colorado State University

Executing Optimized Irregular Applications Using Task Graphs Within Existing Parallel Models

IA^3 Workshop November 11, 2012

Christopher D. Krieger

Michelle Mills Strout

Colorado State University

Irregular Applications Run Out of Memory Bandwidth

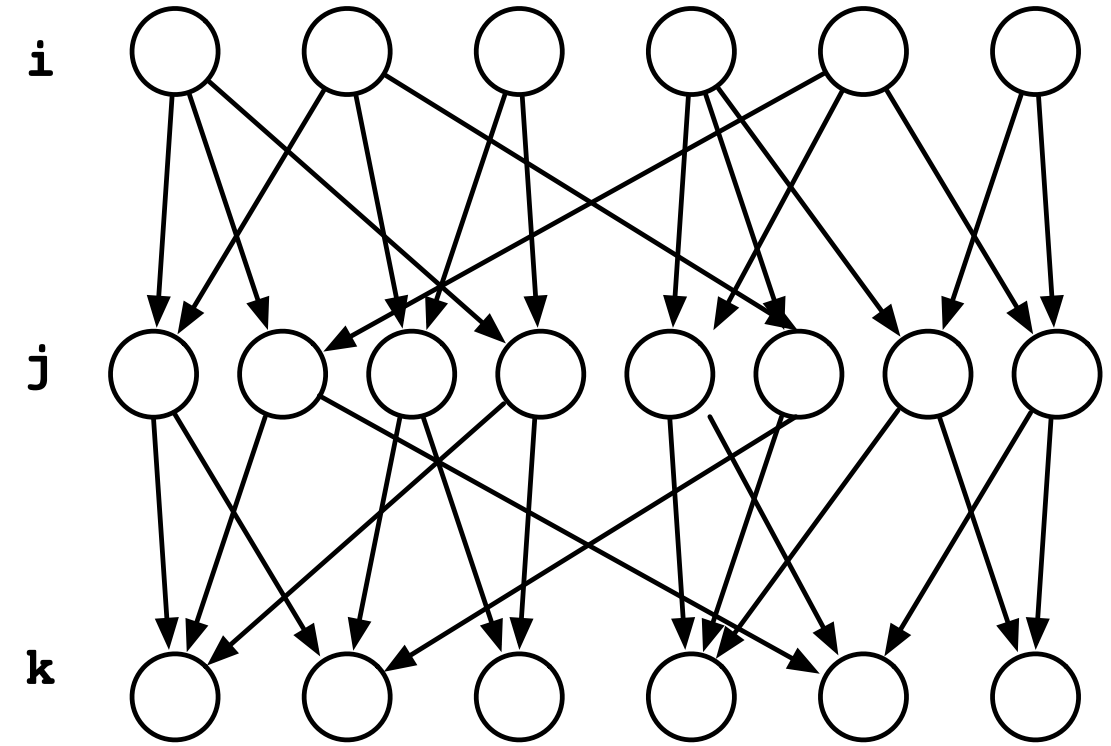
- Even irregular applications often admit straightforward "doall" parallelism
- However, irregular data accesses (e.g. A[B[i]]) hit bandwidth limits
  - Irregular applications challenge prefetchers and cause traffic between caches
  - Scheduling for data locality as well as parallelism is important

Inspector / Executor Scheduling Strategies [Saltz et al 88]

- At runtime, ***inspect*** the data access patterns
  - Reorder data to improve locality
  - Create a new schedule for loops
- Create code (***executor***) that runs the original computation according to modified schedule
  - Execute this schedule repeatedly so as to amortize inspector
- Typical approach for implementing doall distributed memory parallelism

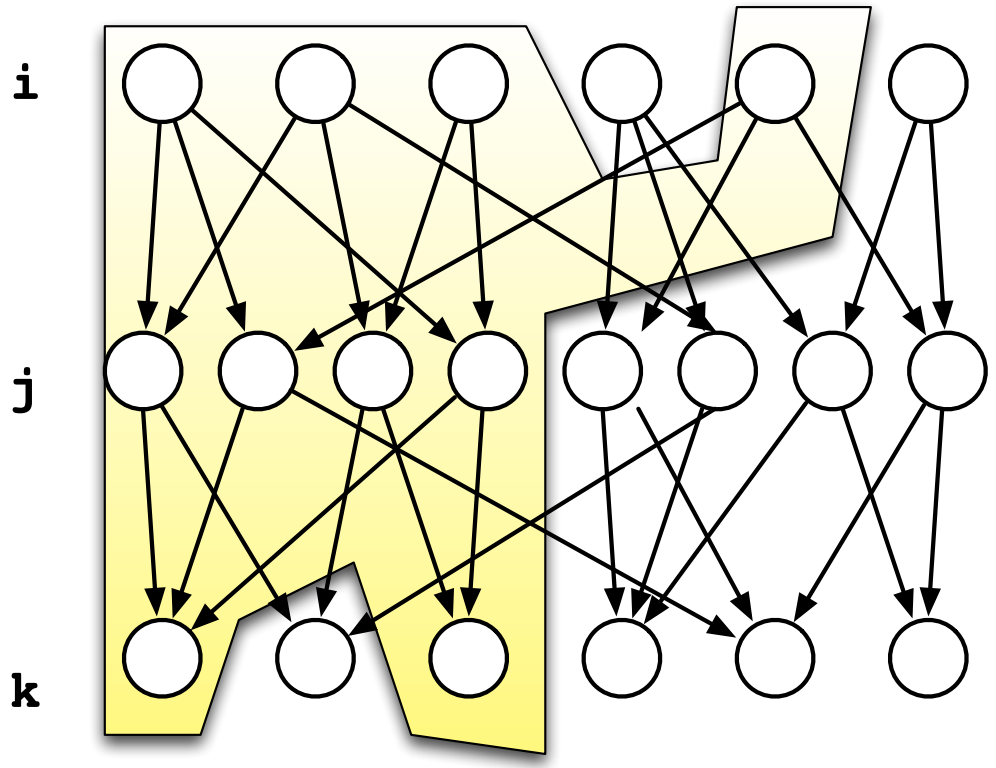
What if we schedule across loops to improve data locality? (Moldyn benchmark example)

```
for s=1,T
  for i=1,n
    ... = ...Z[i]
  endfor

  for j=1,m
    Z[l[j]] = ...
    Z[r[j]] = ...
  endfor

  for k=1,n
    ... += Z[k]
  endfor
endfor
```



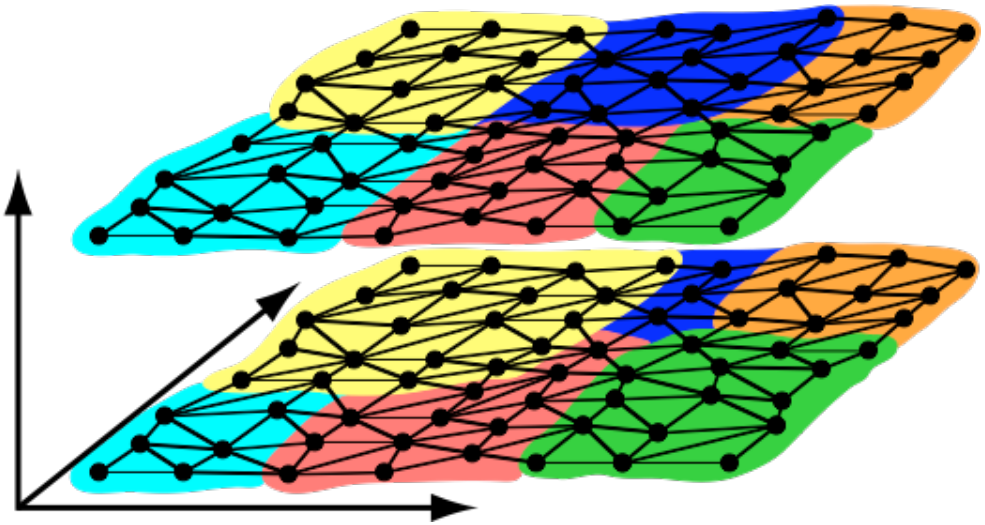
Sparse Tiling Across Loops
(new schedule iterates over tiles)

```
for s=1,T
 for t=0,nt
  for i in sloop0(t)
    ... = ...Z[i]
  endfor

  for j in sloop1(t)
    Z[l''[j]] = ...
    Z[r''[j]] = ...
  endfor

  for k in sloop2(t)
    ... += Z[k]
  endfor
 endfor
endfor
```

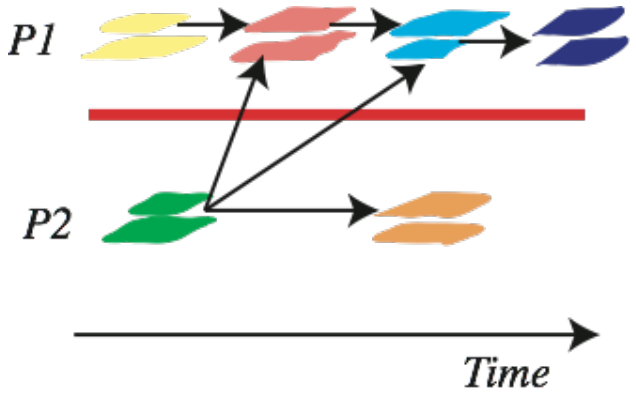


Another Example: Iterative Sparse Computations

Break computation that sweeps over mesh/sparse matrix into chunks/sparse tiles



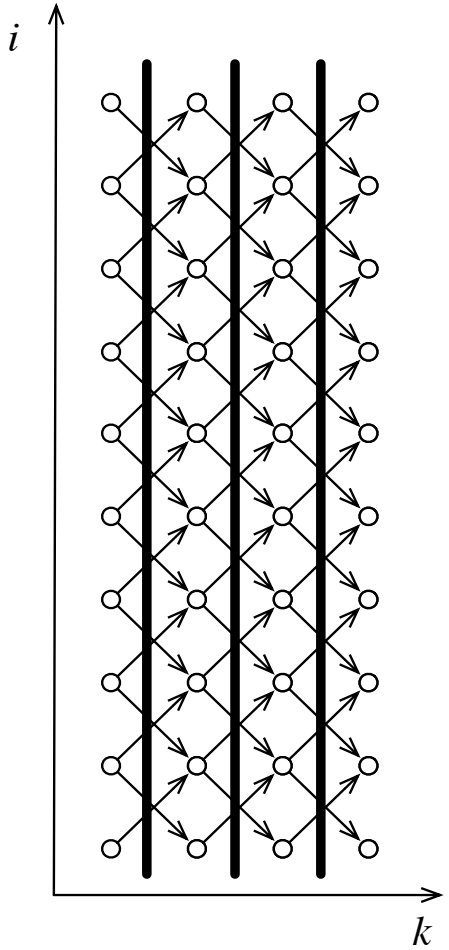
Full Sparse Tiled Iteration Space

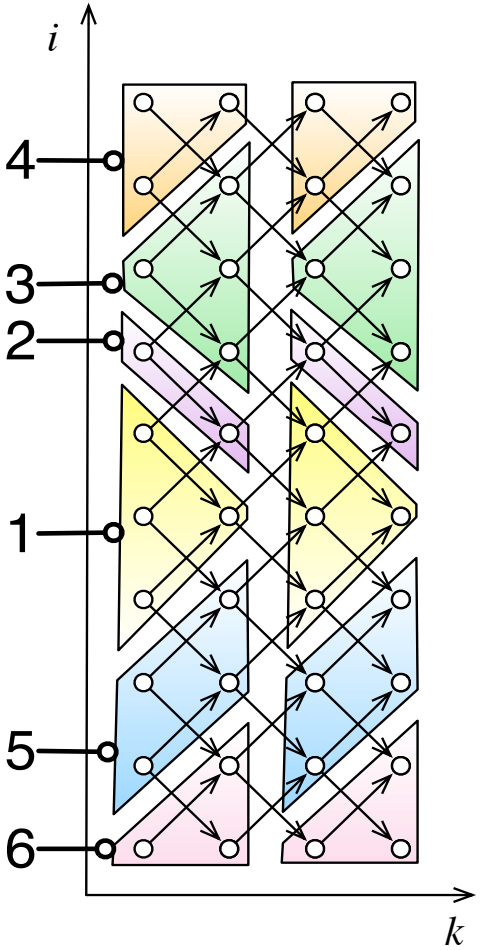


Task Graph

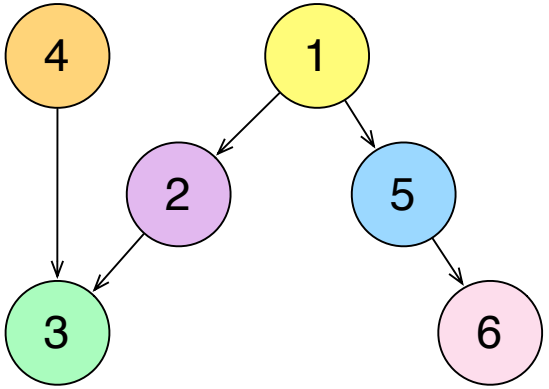
Compare DoAll with Full Sparse Tiling

(Iterative Computation Over Tri-Diagonal Matrix example)



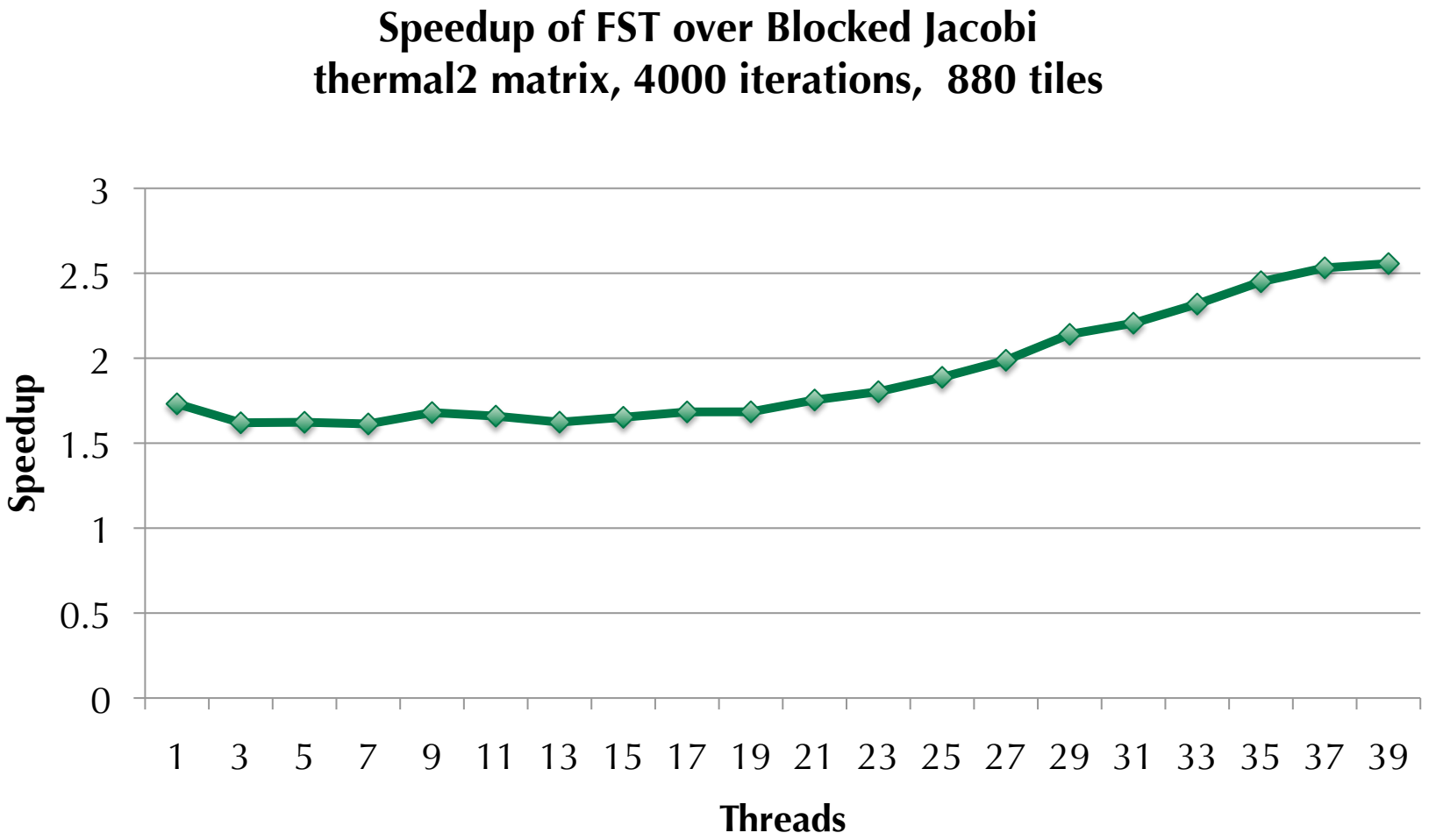


DoAll per outer loop iteration



Full sparse tiling to task graph

Benefit of Full Sparse Tiling



Colorado State University

Sounds Great! What is the problem?

- Implementing sparse tiling inspectors/executors by hand is difficult
- While automating this process, we first investigated ways of expressing the arbitrary task graphs in existing programming models
- Arbitrary task graphs fit easily into TBB task model, but with other programming models the fit was less natural
- When is sparse tiling applicable?

Outline

- Arbitrary Task Graphs for Irregular Applications
- Some Existing Parallel Models
- Performance Evaluation
- Conclusion

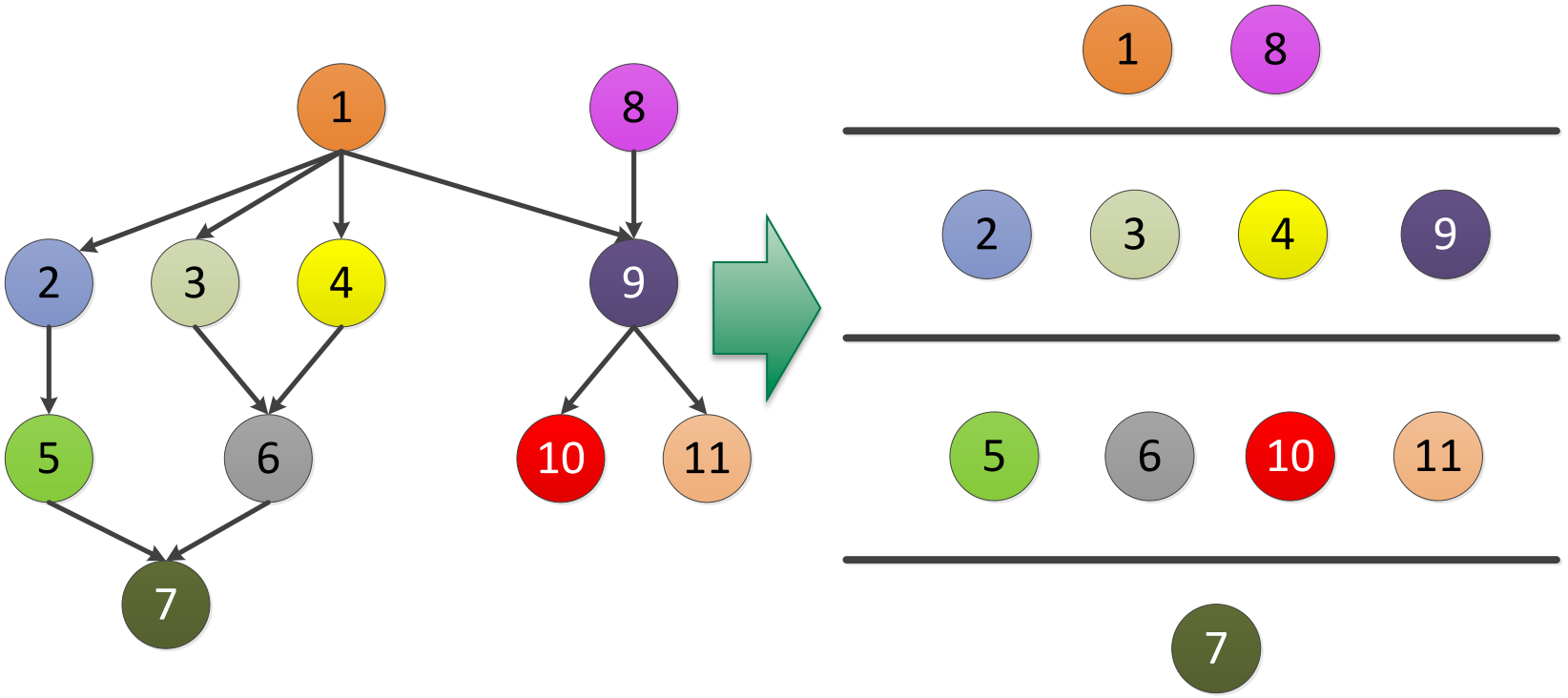
Building on Existing Parallel Models

- Parallel execution engines from common parallel models offer:
  - Efficient task queueing
  - Thread pool management
  - Good dynamic load balancing
  - Mature, documented, standards compliant
- Many to choose from:
  - pThreads, OpenMP, TBB, Cilk Plus

Bridging Between Task Graphs and Existing Models

- Typically focus on:
  - Doall
    - e.g. OpenMP parallel for, TBB parallel_for
  - Fork-join
    - OpenMP task, cilk_spawn
  - TBB now supports task (flow) graphs directly
- Needed to execute a task graph:
  - Way to determine when a task is ready

Task Graph Back to Doall (Frontier)

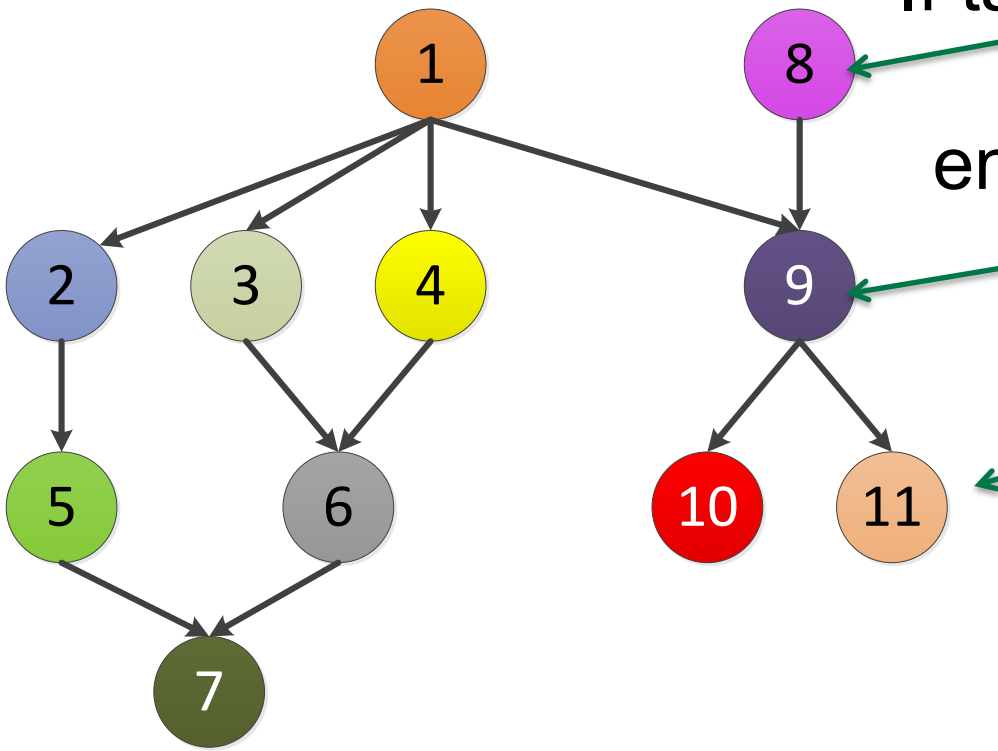


Each level set (or frontier) can be executed using doall parallelism

Task Graphs Using Frontier

- Disadvantages
  - Artificial serialization reduces parallelism
  - Load balancing problems
  - Adds cost of multiple barriers
- Advantages
  - Still enables load balancing within a frontier
  - Very low dynamic overhead

Executing Task Graphs Using Fork-Join



If task is last predecessor of a successor, enqueue that successor

If leaf task completes, atomically decrement leaf counter

When leaf count reaches zero, graph is complete

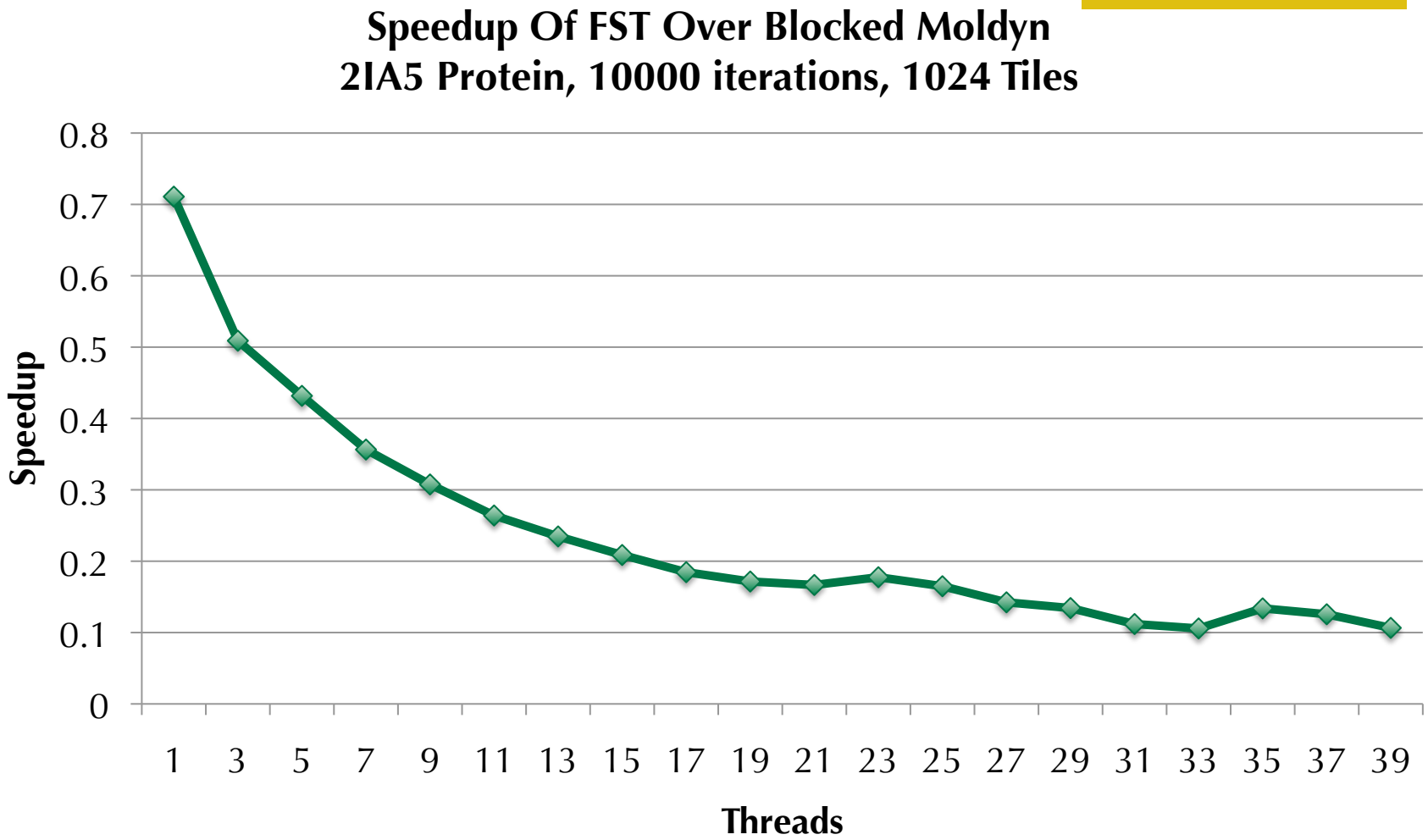
Outline

- Arbitrary Task Graphs for Irregular Applications
- Some Existing Parallel Models
- Performance Evaluation
- Conclusion

Benchmarks

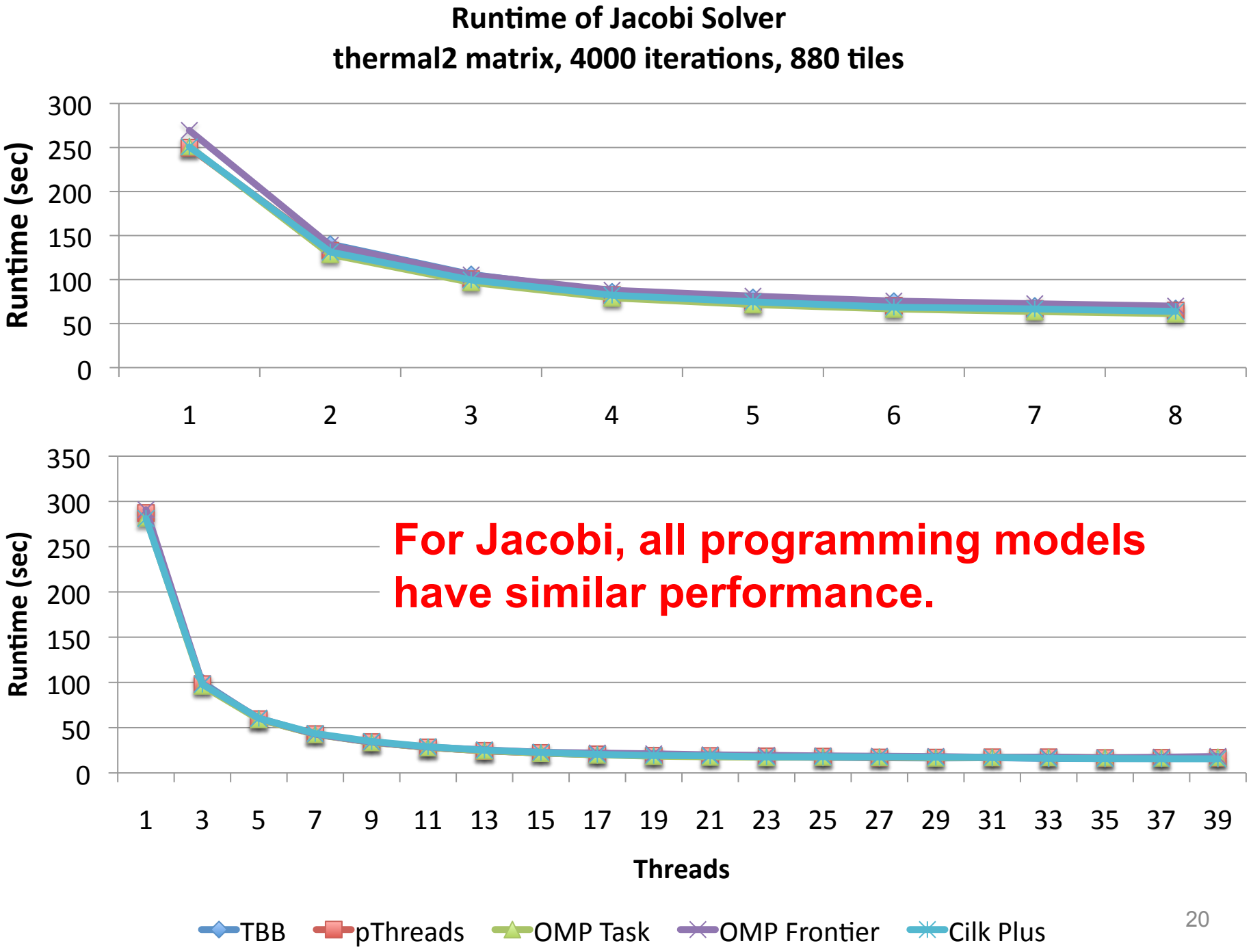
- Sparse Jacobi Solver
  - thermal2 matrix from U of F Matrix Market
    - 1.2 Million rows/cols, 120 MB data footprint
  - 880 tasks in graph, typical task took 250-1000 µs
  - Sparse tiling improves scalability
- Simple molecular dynamics app (Moldyn)
  - 2IA5 protein from Protein Data Bank
    - 28k atoms, 80k interactions, 2.56 MB data footprint
  - 1024 tasks in graph, typical task took 2-8 µs
  - Sparse tiling does NOT improve scalability

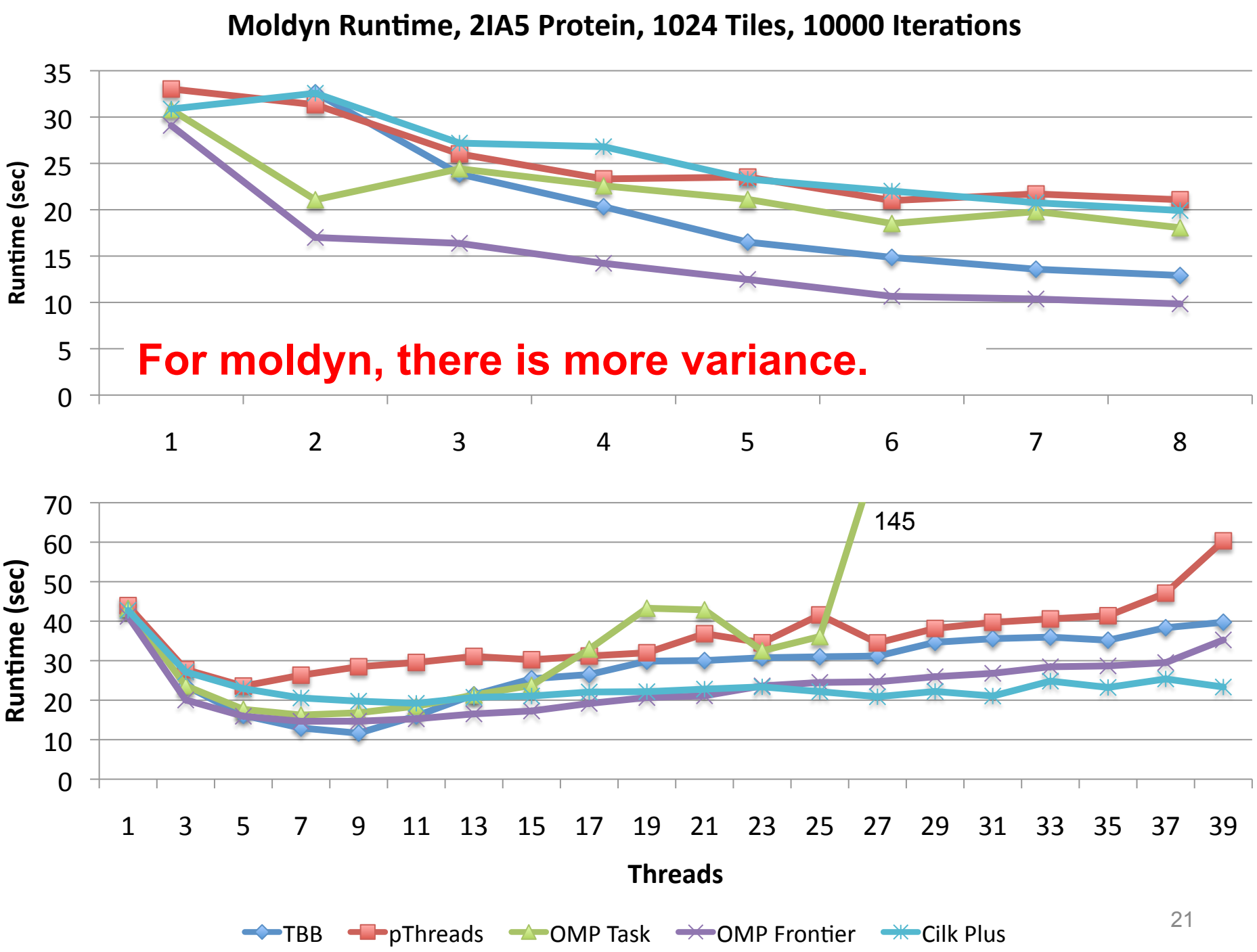
Lack of Benefit using Full Sparse Tiling for Moldyn



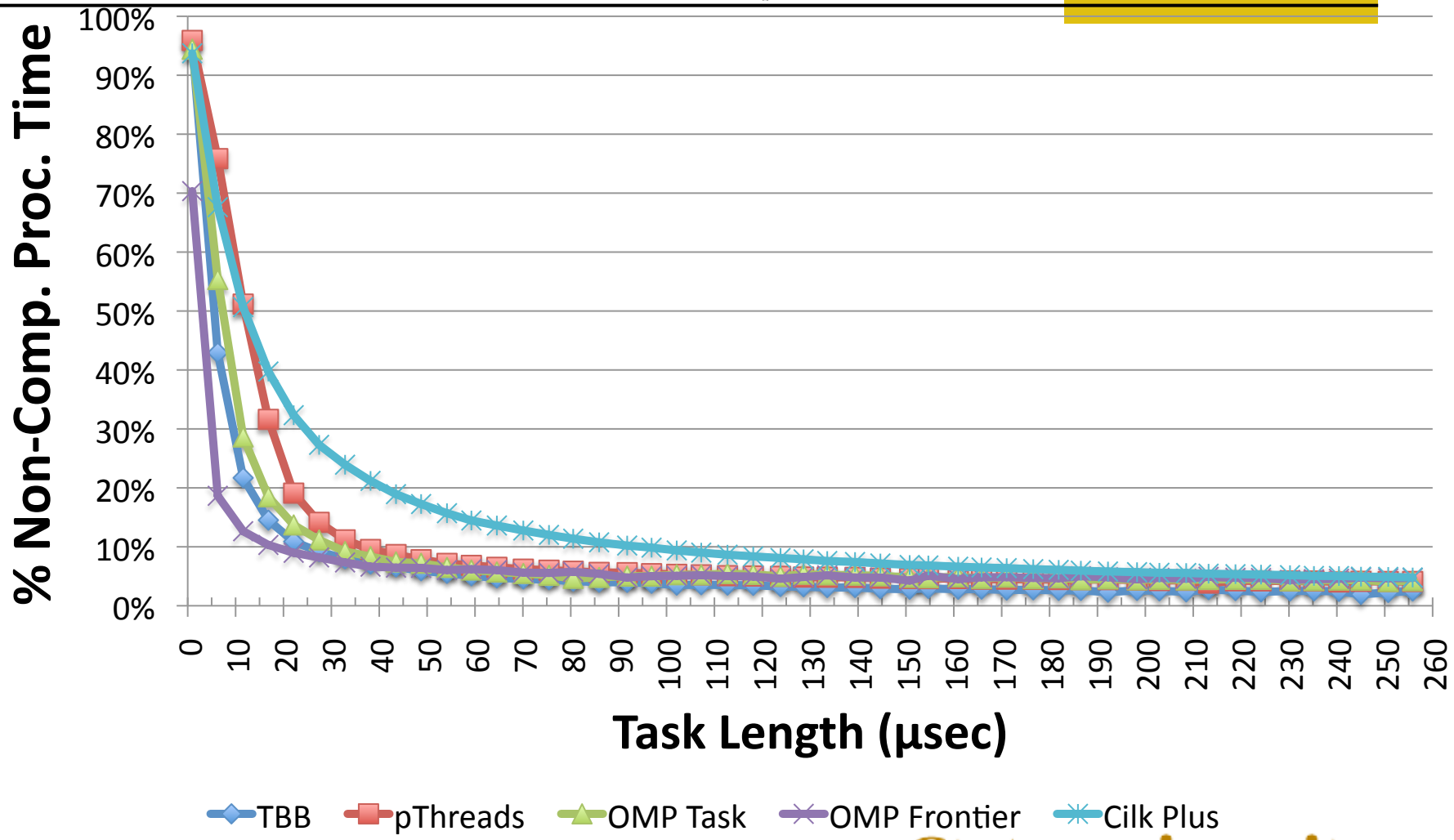
Methodology

- Main question: Does the programming model affect performance?
- Ran on 8 core and 40 core machine
  - 8 Core (2x4 core) Xeon E-5450 Harpertown
  - 40 Core (4x10 core) Xeon E-4860 Westmere EX
    - From Intel's Manycore Testing Lab
- Intel icc compiler used
  - Version 12.1.3 (20120212)
  - icc needed to support Cilk Plus

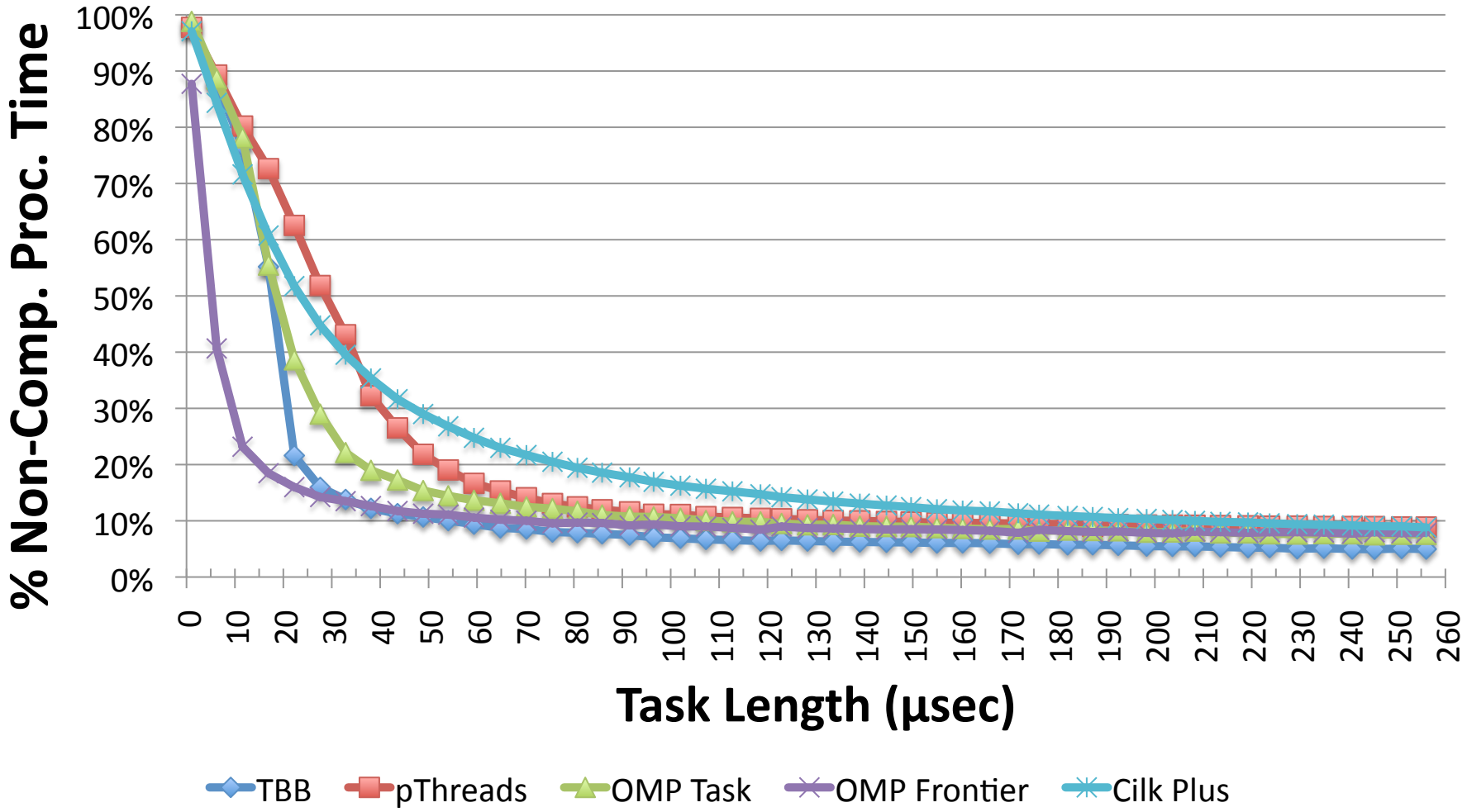

Runtime of Jacobi Solver
thermal2 matrix, 4000 iterations, 880 tiles
Runtime (sec)
300
250
200
150
100
50
0
1
2
3
4
5
6
7
8
Runtime (sec)
350
300
250
200
150
100
50
0
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
Threads
For Jacobi, all programming models have similar performance.
TBB
pThreads
OMP Task
OMP Frontier
Cilk Plus


Moldyn Runtime, 2IA5 Protein, 1024 Tiles, 10000 Iterations
Runtime (sec)
35
30
25
20
15
10
5
0
1
2
3
4
5
6
7
8
For moldyn, there is more variance.
Runtime (sec)
70
60
50
40
30
20
10
0
145
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
Threads
TBB
pThreads
OMP Task
OMP Frontier
Cilk Plus

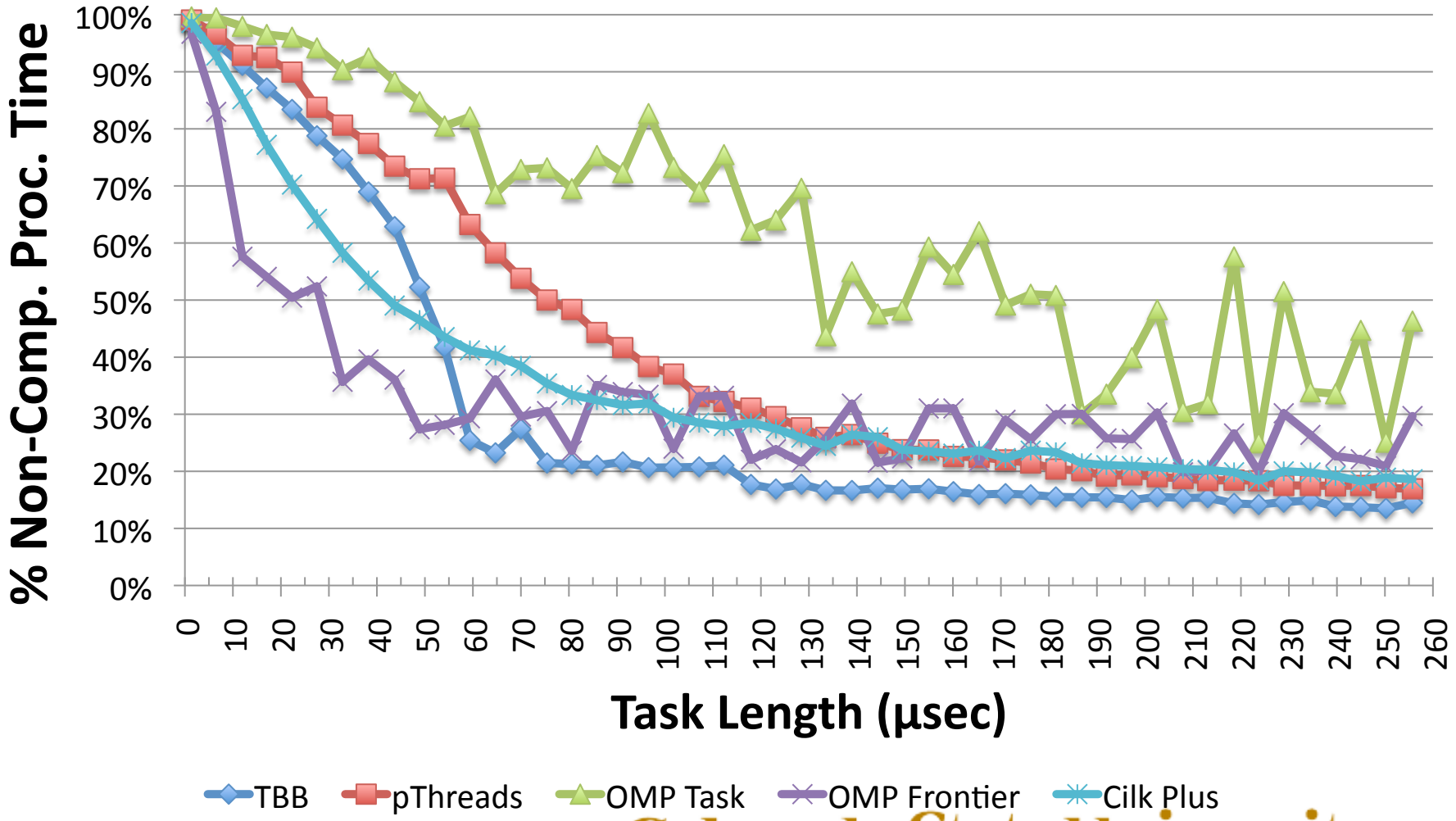
Inefficiency Due to Short Tasks
10 Threads in moldyn



Inefficiency Due to Short Tasks
20 Threads in moldyn



Inefficiency Due to Short Tasks
40 Threads in moldyn



Conclusions & Future Work

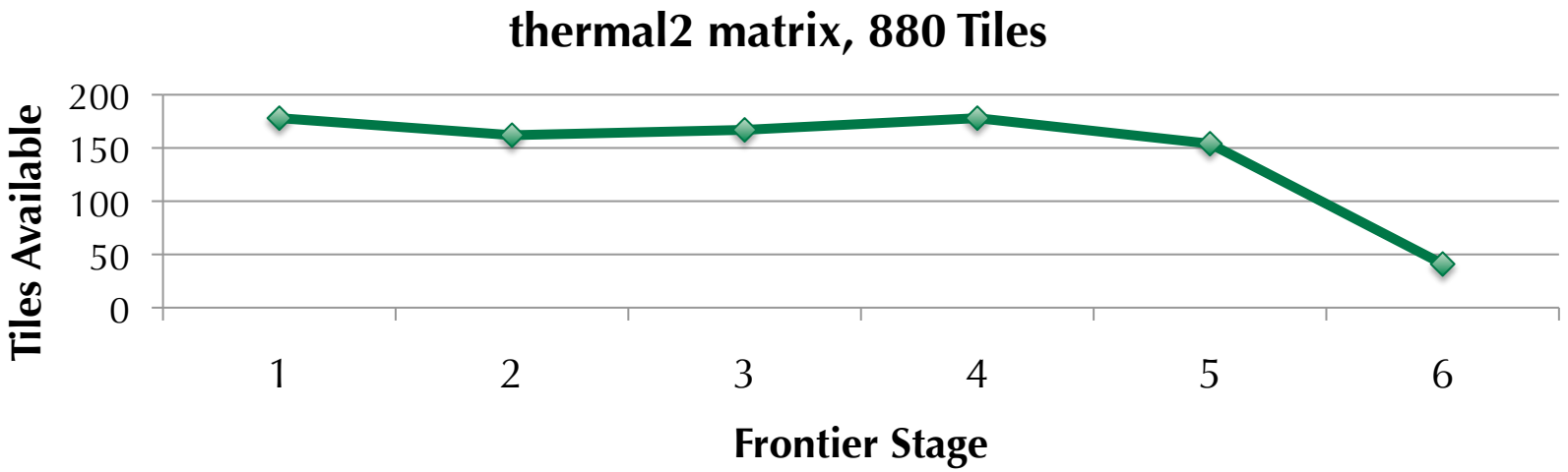
- Sparse tiling parallelization strategies for improved locality lead to task graph parallelism
- Can express and execute these arbitrary graphs on established parallel programming models with TBB providing most natural fit
- Little difference in performance between models
  - Graphs with lightweight tasks perform poorly
  - Frontier models are more tolerant of fine tasks
- Future Work
  - Would like to see natural support for arbitrary task graphs in emerging programming models
  - Develop a model for optimal seed partition size

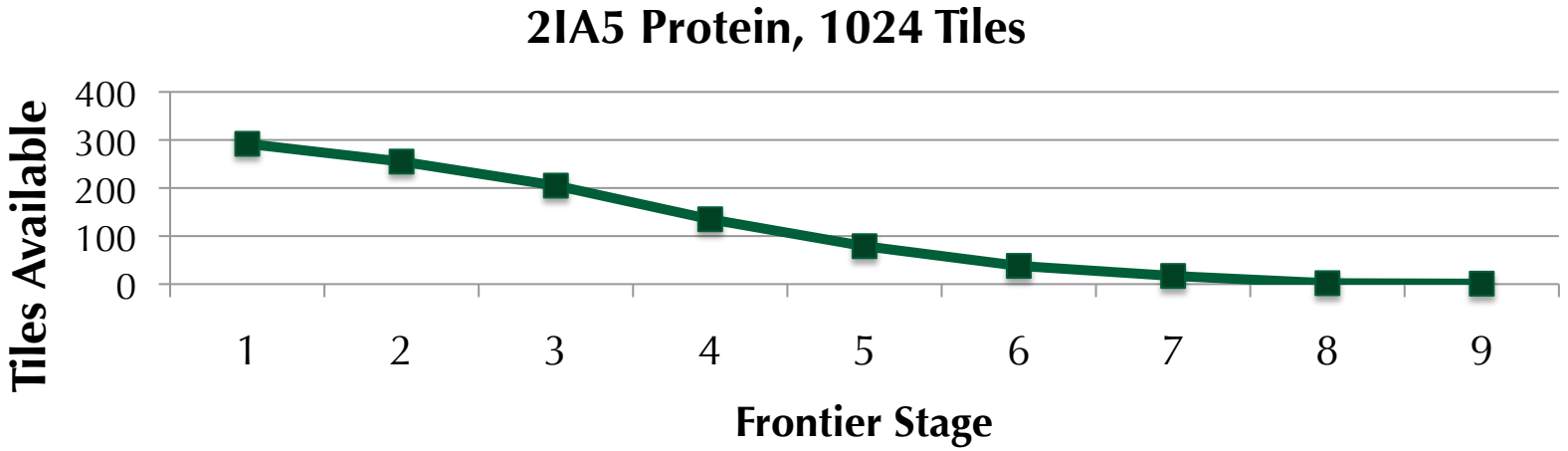
Backup Slides

Limits to Scalability

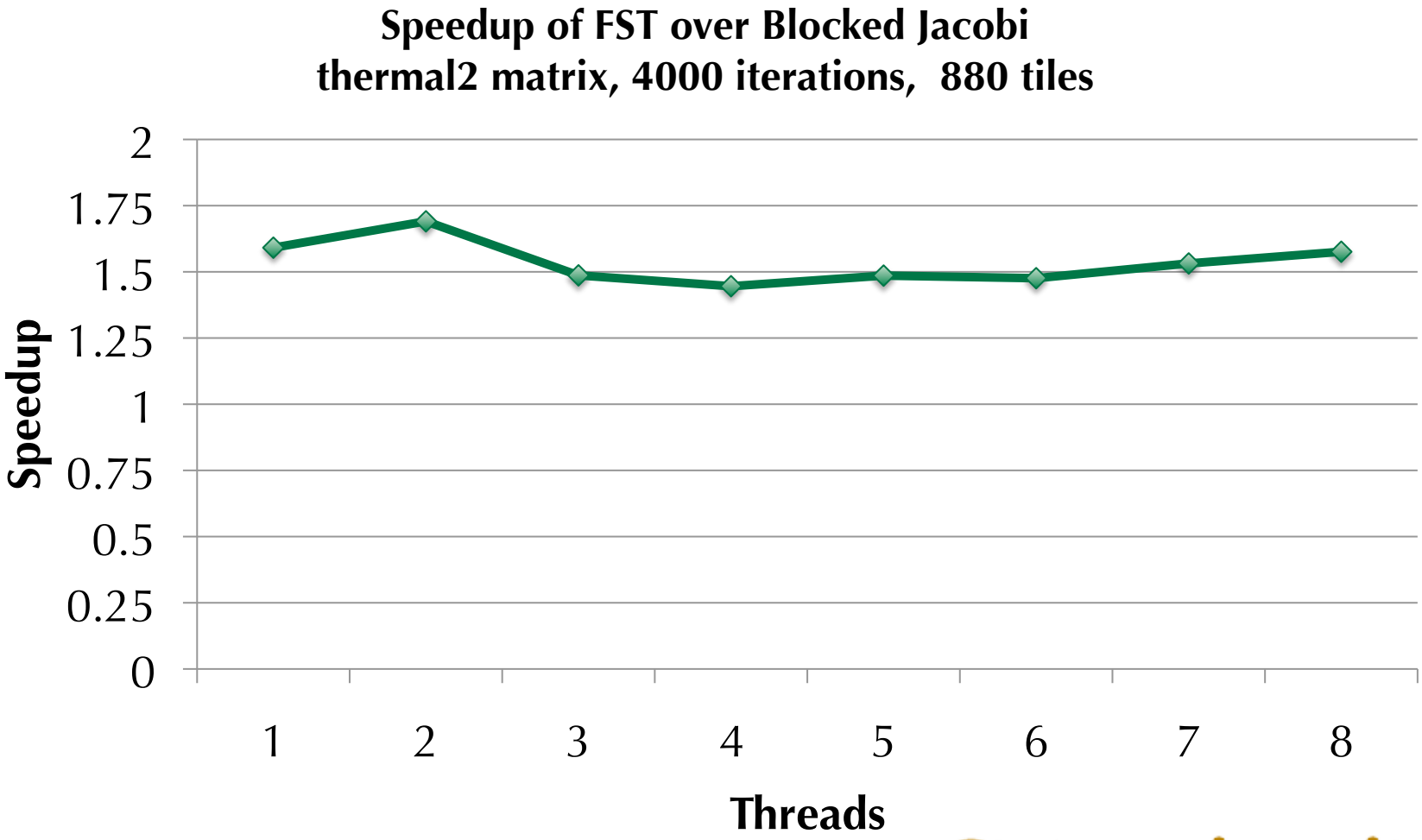
- System Memory Bandwidth
  - Rerunning on 8 cores with no-memory workload shows nearly perfect linear scalability (7.8x @ 8 cores)
  - No memory load on 40 cores shows linear scalability (35x @ 39 cores)

Available Parallelism

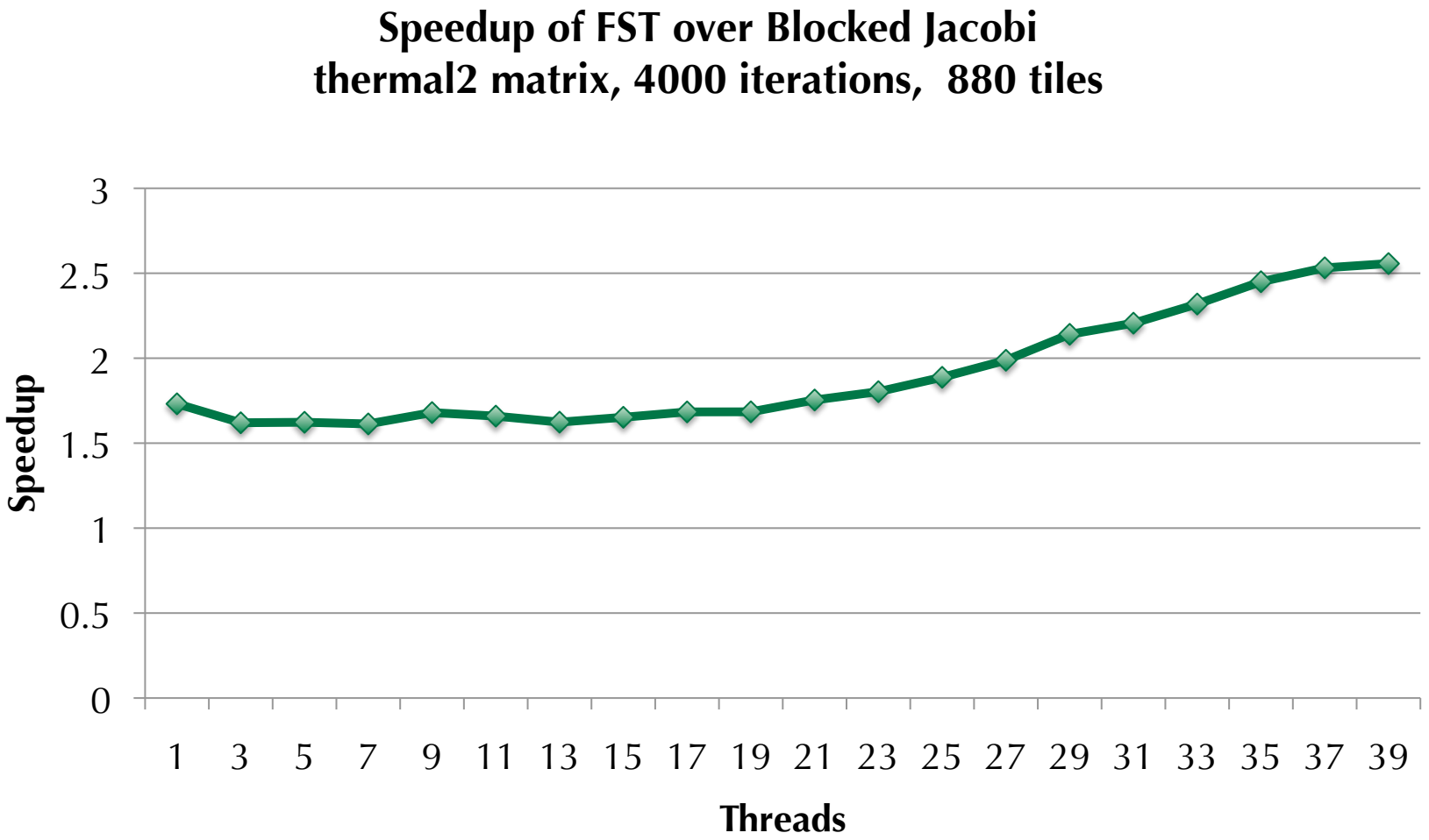




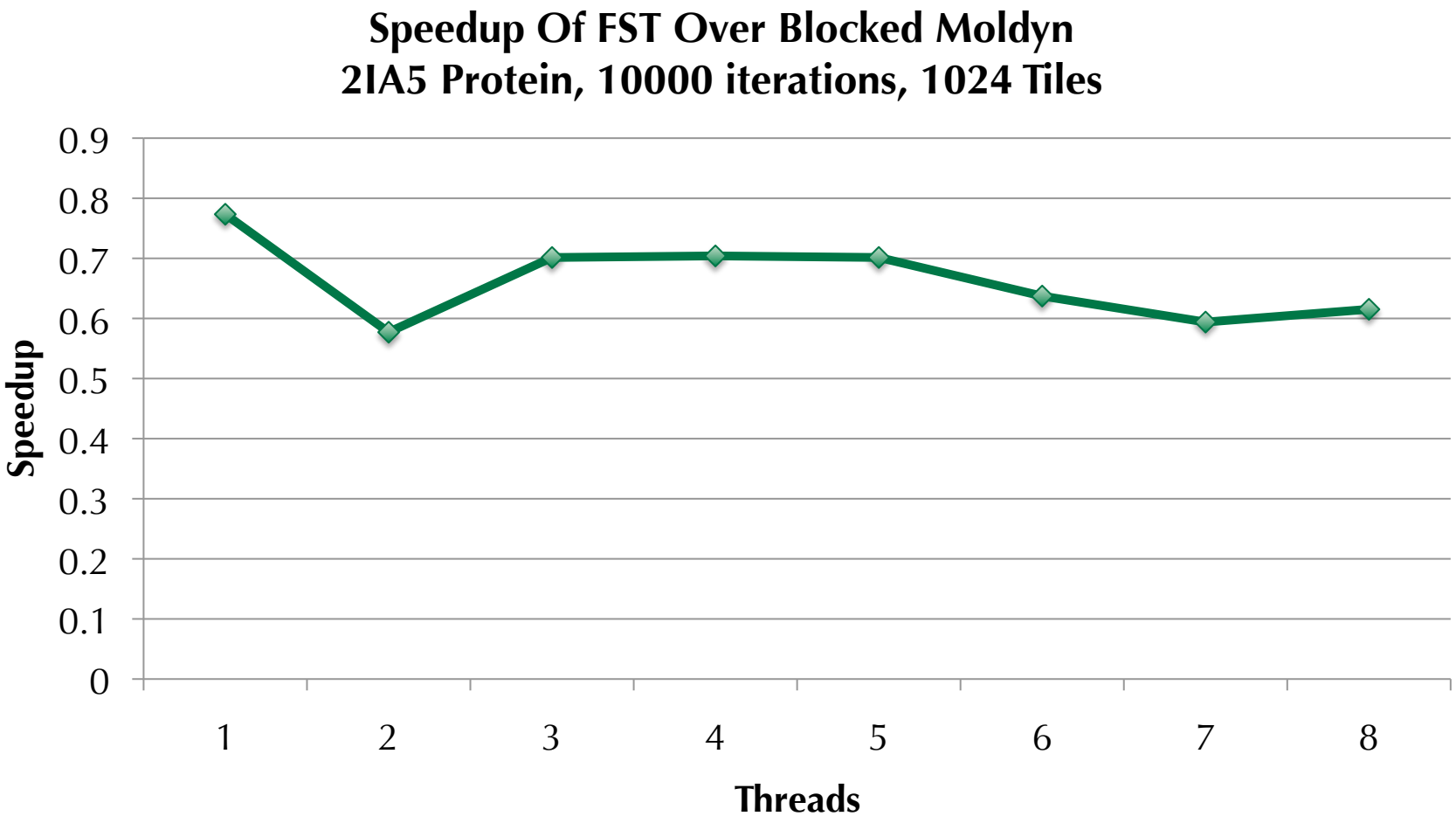
Benefit of Full Sparse Tiling



Benefit of Full Sparse Tiling



Benefit of Full Sparse Tiling



Benefit of Full Sparse Tiling