Locality Aware Scheduling of Sparse Computations for Energy and Performance Efficiencies

Michael Frasca

Kamesh Madduri

Padma Raghavan

Department of Computer Science & Engineering

The Pennsylvania State University

SIAM Conference on Computational Science and Engineering

February 25-March 1, 2013
The Westin Boston Waterfront
Boston, Massachusetts, USA

March 1, 2013

ABSTRACT

Exploiting parallelism for large-scale irregular applications requires efficient use of a complex memory hierarchy.

We develop dynamic workload strategies that map the demands of parallel graph algorithms onto shared compute and memory resources, while achieving improved power and performance efficiencies

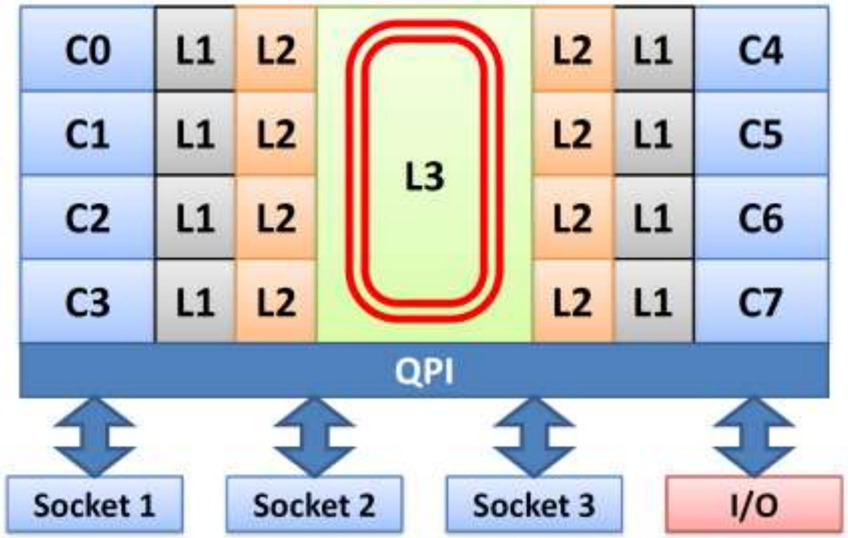
Outline

- Background
  - Non-Uniform Memory Access (NUMA)
  - Betweenness Centrality (BC)
- NUMA-Aware Dynamic Strategies
  - Adaptive Data Layout (ADL)
  - NUMA-Aware Workload Queues (NWQ)
- Power & Performance Results
  - 20 large graph inputs
  - 1-32 cores/threads
  - Detailed working-set analysis
- Conclusions

Background | NUMA Cache Hierarchy

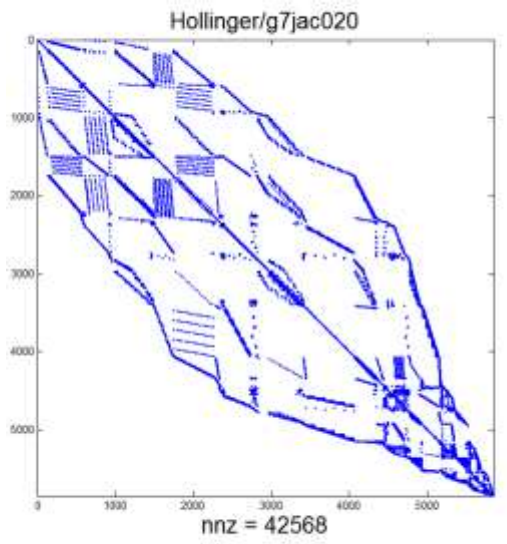
- Scalable cache design
  - Local caches with low-latency
  - Shared caches with higher-capacity
  - Shared cache latency depends on distance to core
  - Multi-socket systems connected via point-to-point interconnect

- Coherence policy
  - Multiple copies of shared data
  - Writing to data invalidates non-owned copies
  - Applies to caches across multiple sockets



Background | Graph Mining

- Graph analysis
  - Inherently unstructured data
  - Irregular access patterns, unknown until runtime
  - Data partitioning is hard



- Parallelism
  - Shared memory, dynamic partitioning
  - Light-weight threading
  - Efficient synchronization

- Betweenness Centrality
  - Measure of a node's importance in a graph

Background | Betweenness Centrality

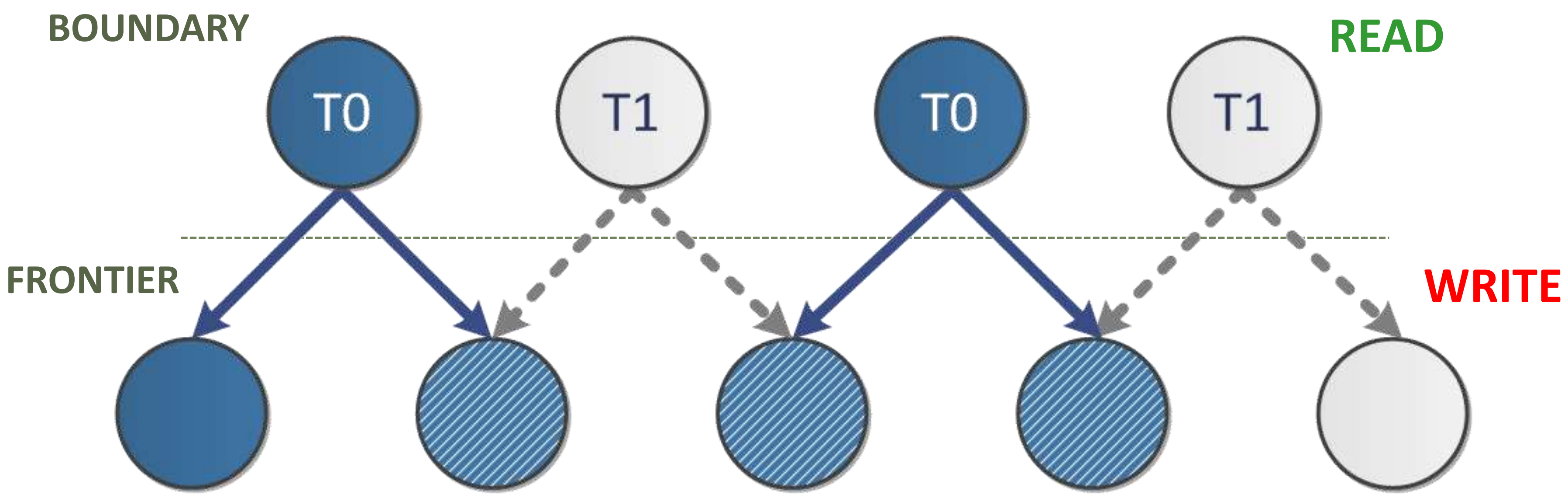
- Count of all shortest paths $\mathbf{s} \dashrightarrow \mathbf{t}$
- Count of all shortest paths $\mathbf{s} \dashrightarrow \mathbf{t}$ that contain $\mathbf{v}$

$$BC(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

- All-pairs shortest path
  - One BFS per node + updates to per-node metadata

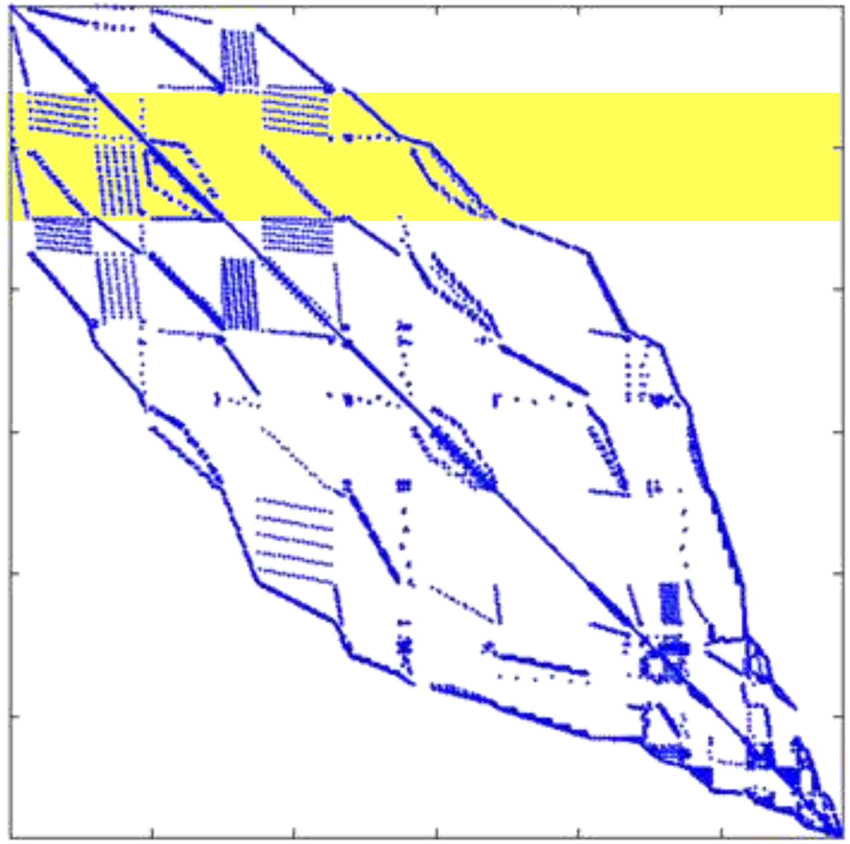
- Lock-free implementation
  - Level-synchronous design
  - Dynamic workload balancing
  - Madduri et al. [IPDPS 2009]

Parallel BFS | Memory Behavior

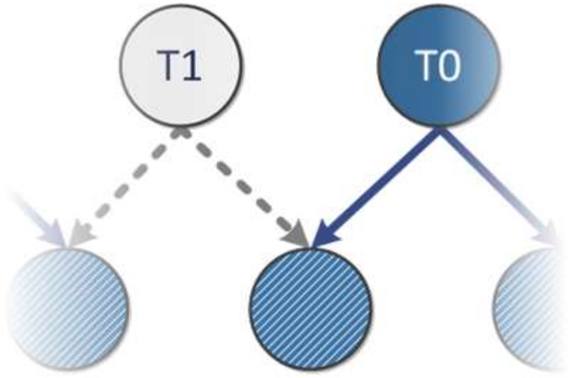


- Dynamic thread-vertex assignment at each level
  - Outgoing edge => update to frontier node
  - Dictates reuse across tree depths
  - Can generate unnecessary sharing on the frontier

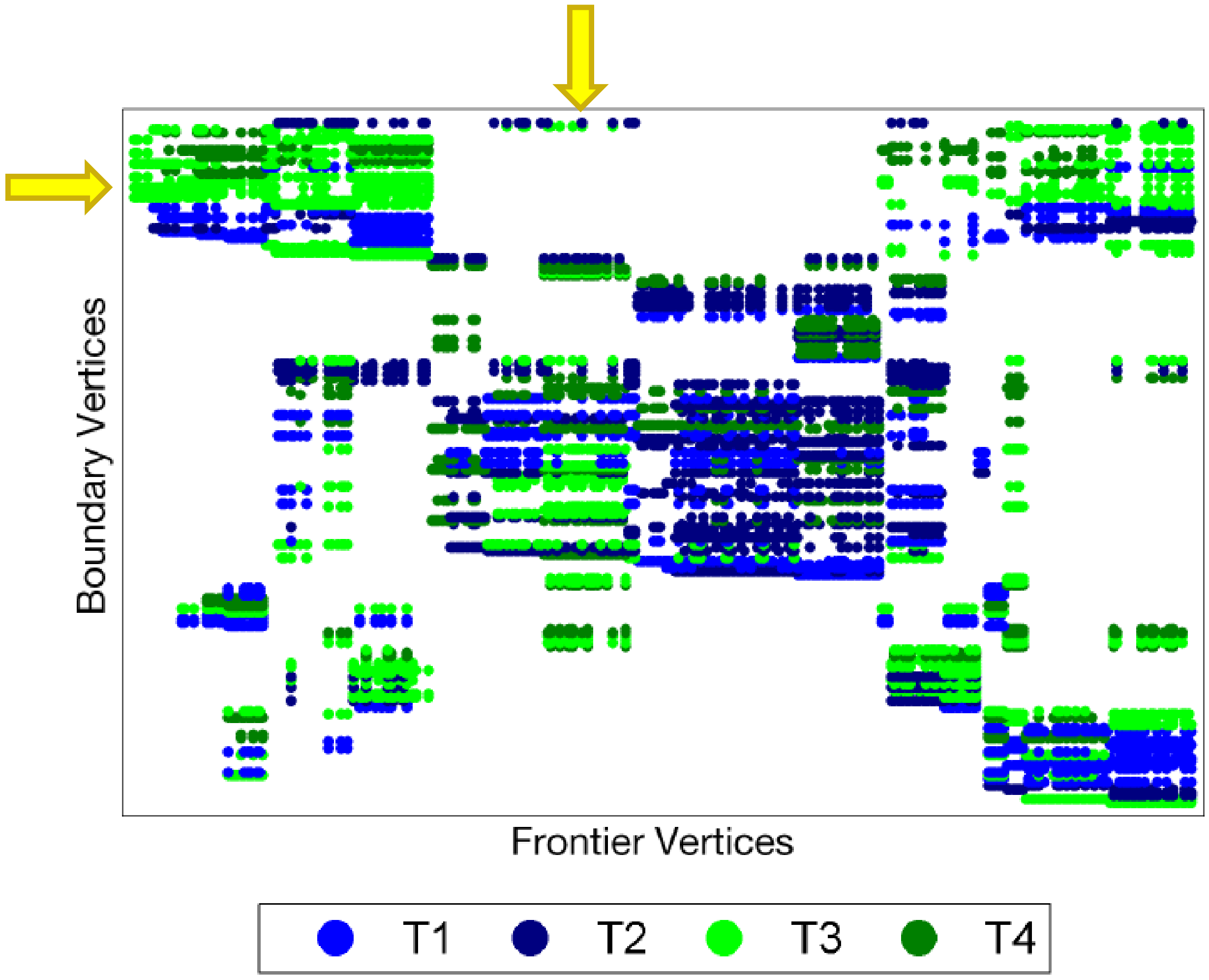
Parallel BFS | Memory Behavior

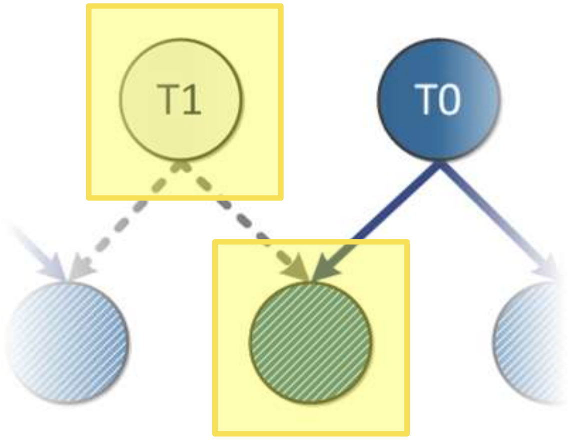


Adjacency Matrix Representation



Parallel BFS | Memory Behavior





Workload distribution of a front

NUMA-Aware Techniques

NUMA-Aware | Adaptive Data Layout

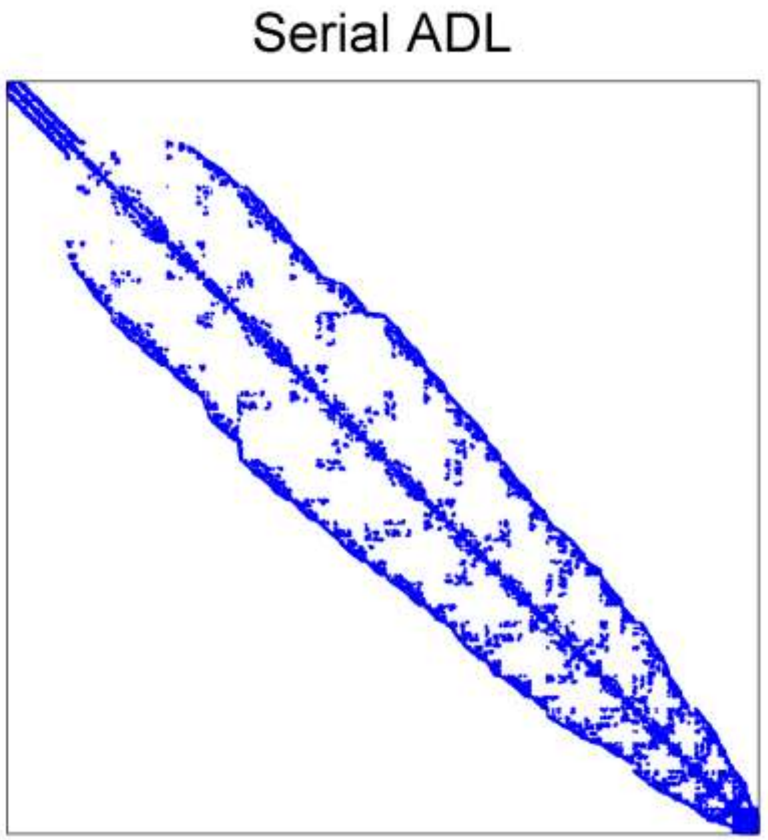
- We assume graphs have a random ordering
  - Poor spatial data locality
  - High amount of false sharing

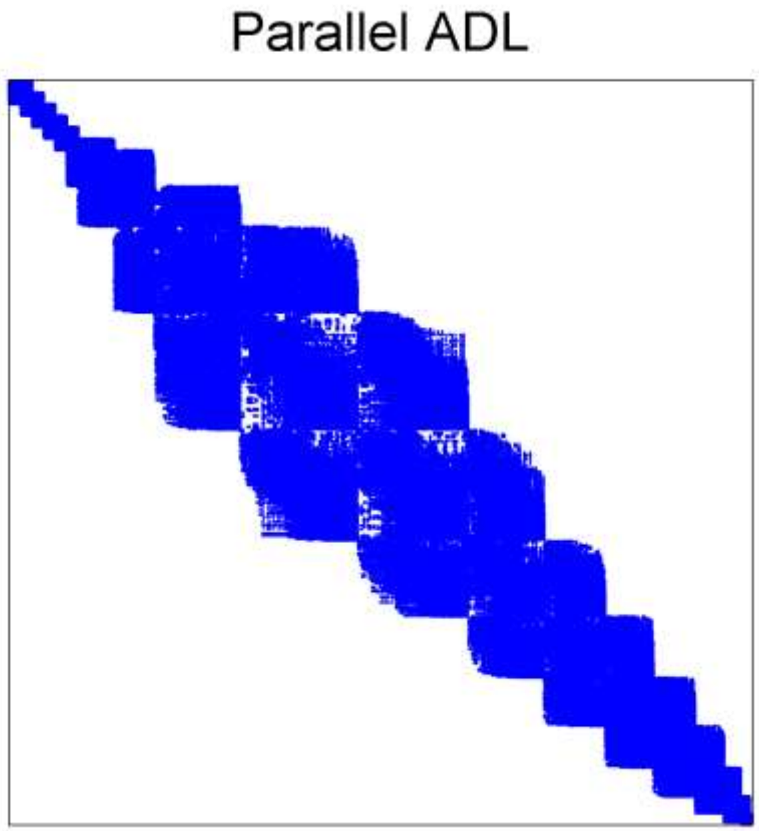
- Dynamic graph reordering
  - The first BFS traverses the random graph
  - Order of discovery is used to permute the graph
  - Improved locality for remaining BFS traversals

Permutation Sort

$$\langle\ depth,\ \ time\ discovered\ \rangle$$

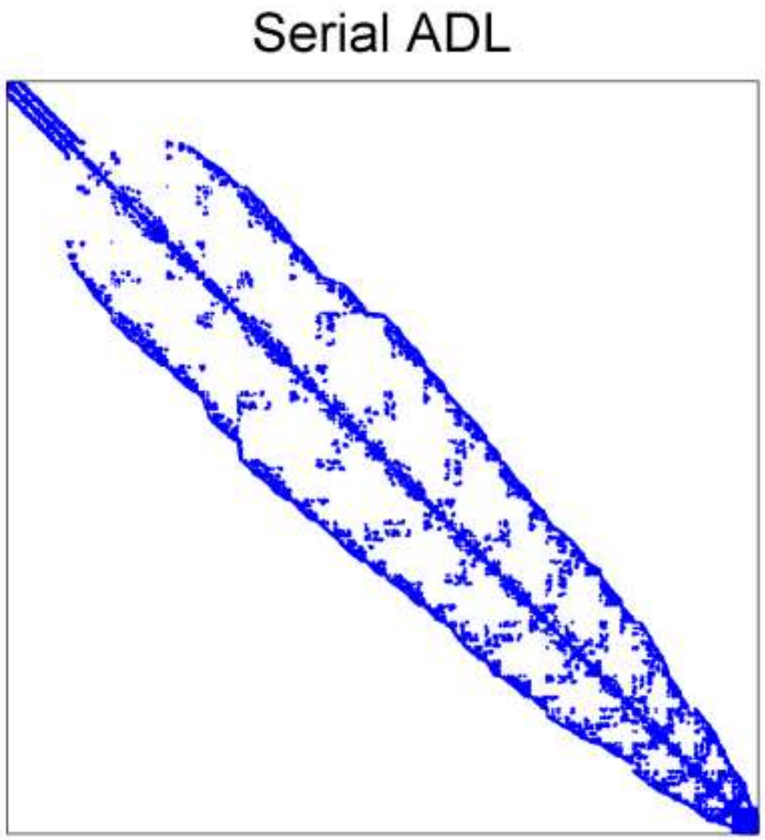
NUMA-Aware | Adaptive Data Layout

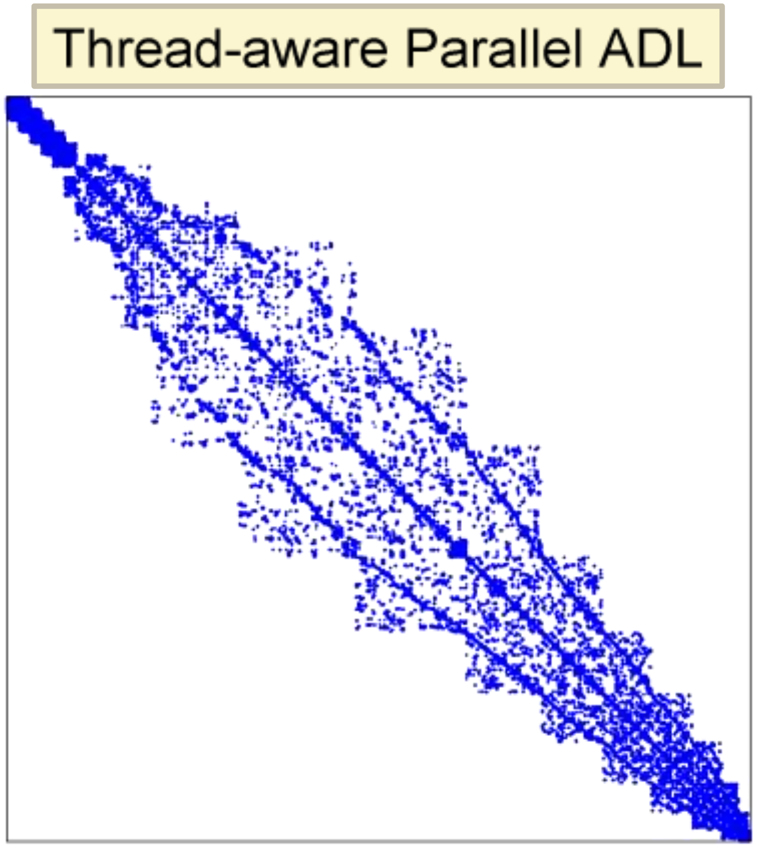




$$\langle\ depth,\ \ time\ discovered\ \rangle$$

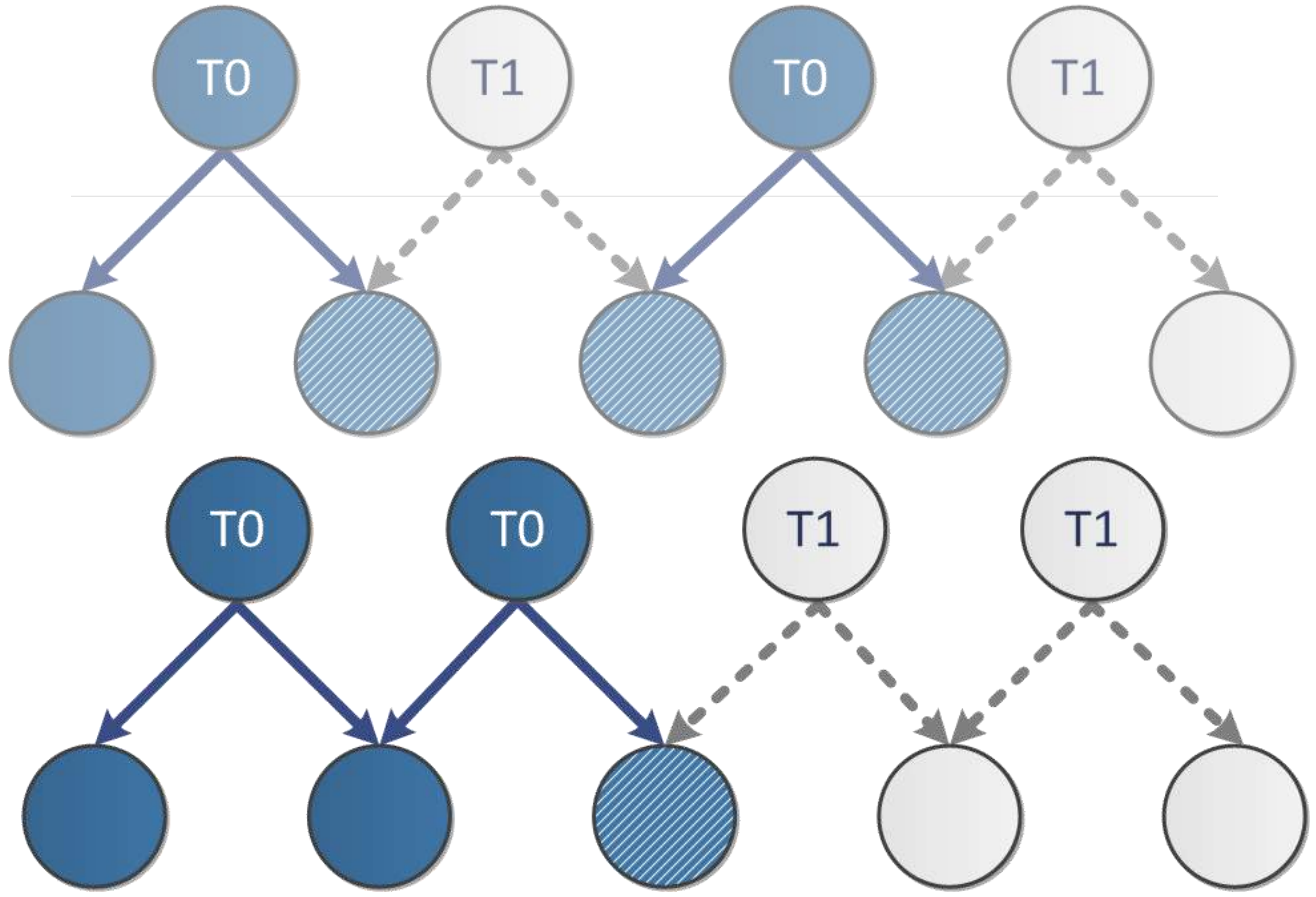
NUMA-Aware | Adaptive Data Layout



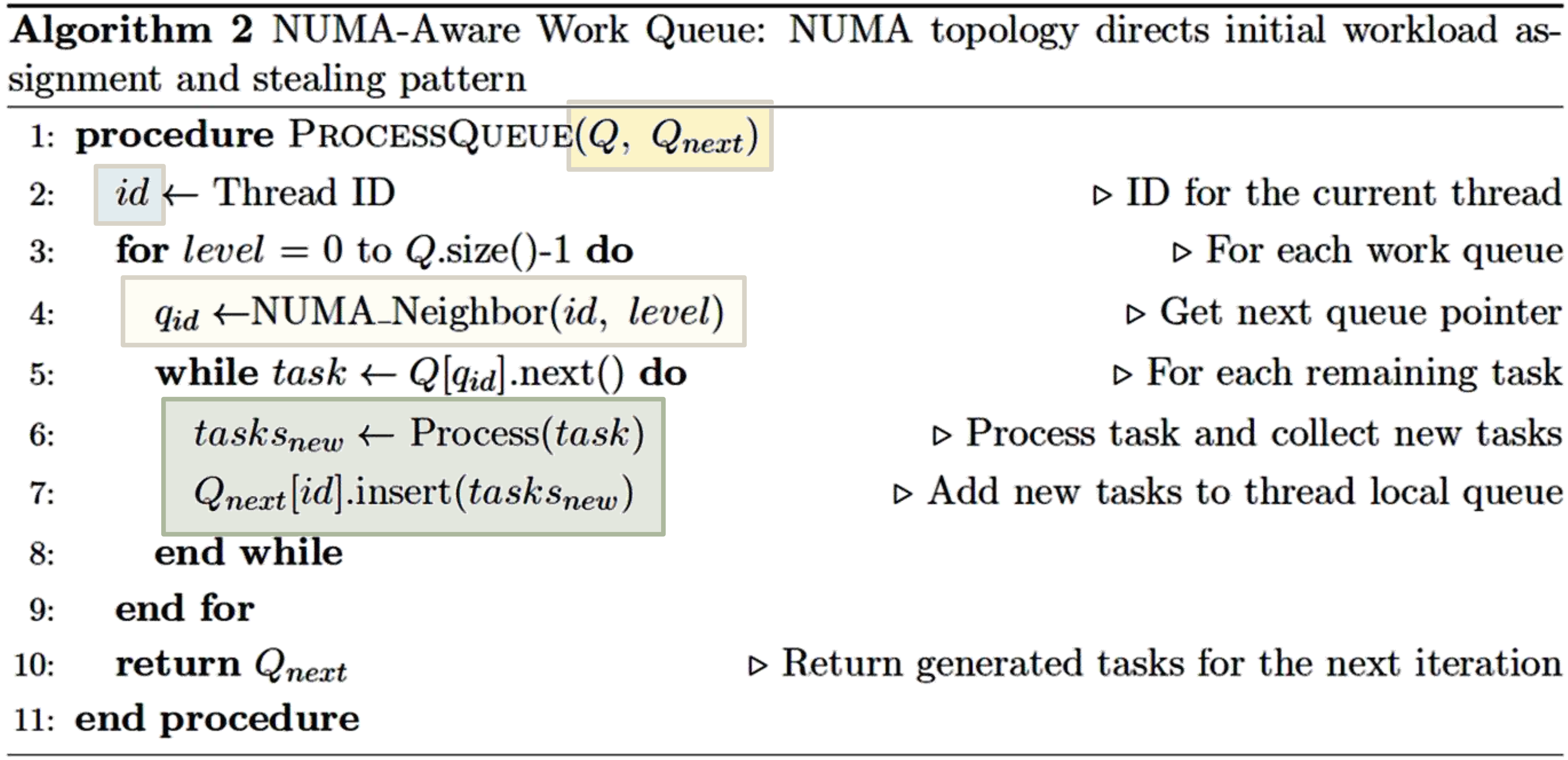


$$\langle\ depth,\ \boxed{thread\ owner,}\ time\ discovered\ \rangle$$

NUMA-Aware | Dynamic Work Queues



NUMA-Aware | Dynamic Work Queues

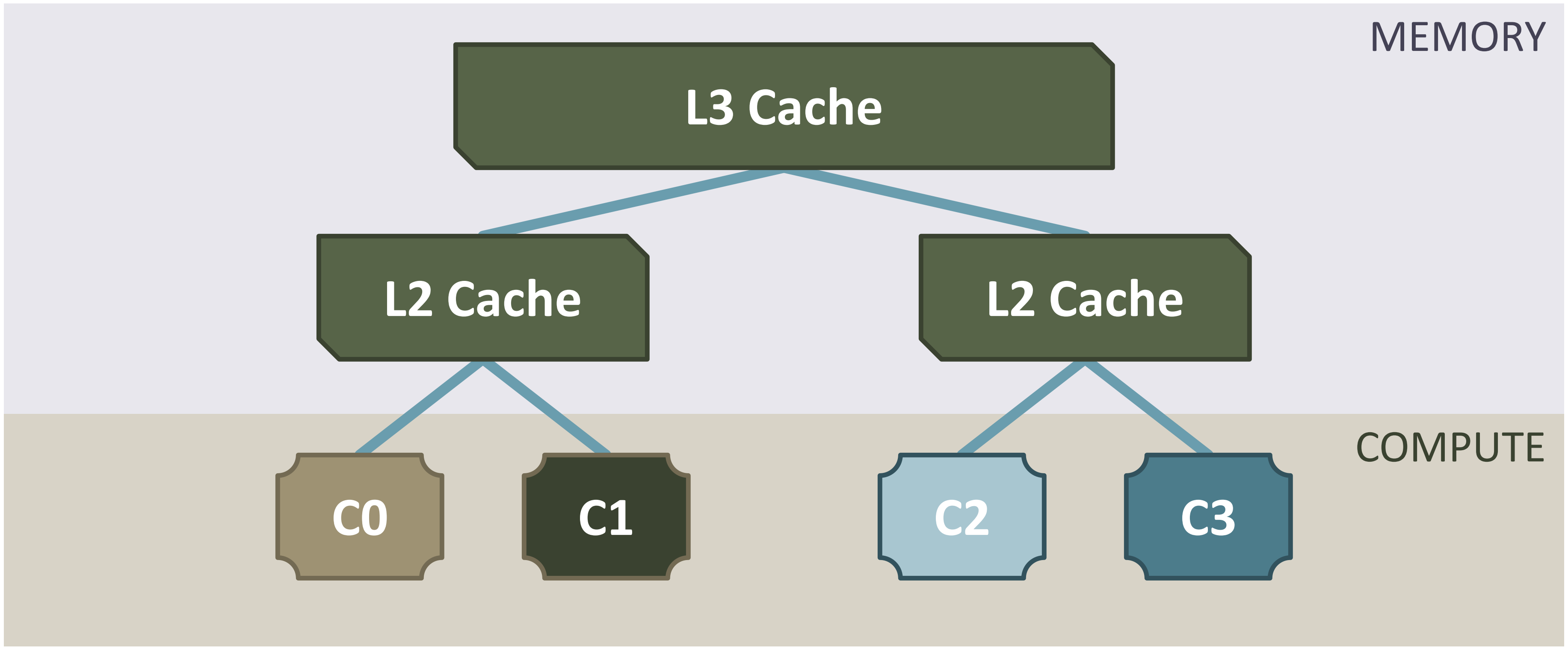


Algorithm 2 NUMA-Aware Work Queue: NUMA topology directs initial workload assignment and stealing pattern

```
procedure ProcessQueue($Q$, $Q_{next}$)
  $id \leftarrow$ Thread ID                              ▷ ID for the current thread
  for $level = 0$ to $Q$.size()-1 do                      ▷ For each work queue
    $q_{id} \leftarrow$NUMA_Neighbor($id$, $level$)          ▷ Get next queue pointer
    while $task \leftarrow Q[q_{id}]$.next() do               ▷ For each remaining task
      $tasks_{new} \leftarrow$ Process($task$)              ▷ Process task and collect new tasks
      $Q_{next}[id]$.insert($tasks_{new}$)                  ▷ Add new tasks to thread local queue
    end while
  end for
  return $Q_{next}$                                     ▷ Return generated tasks for the next iteration
end procedure
```

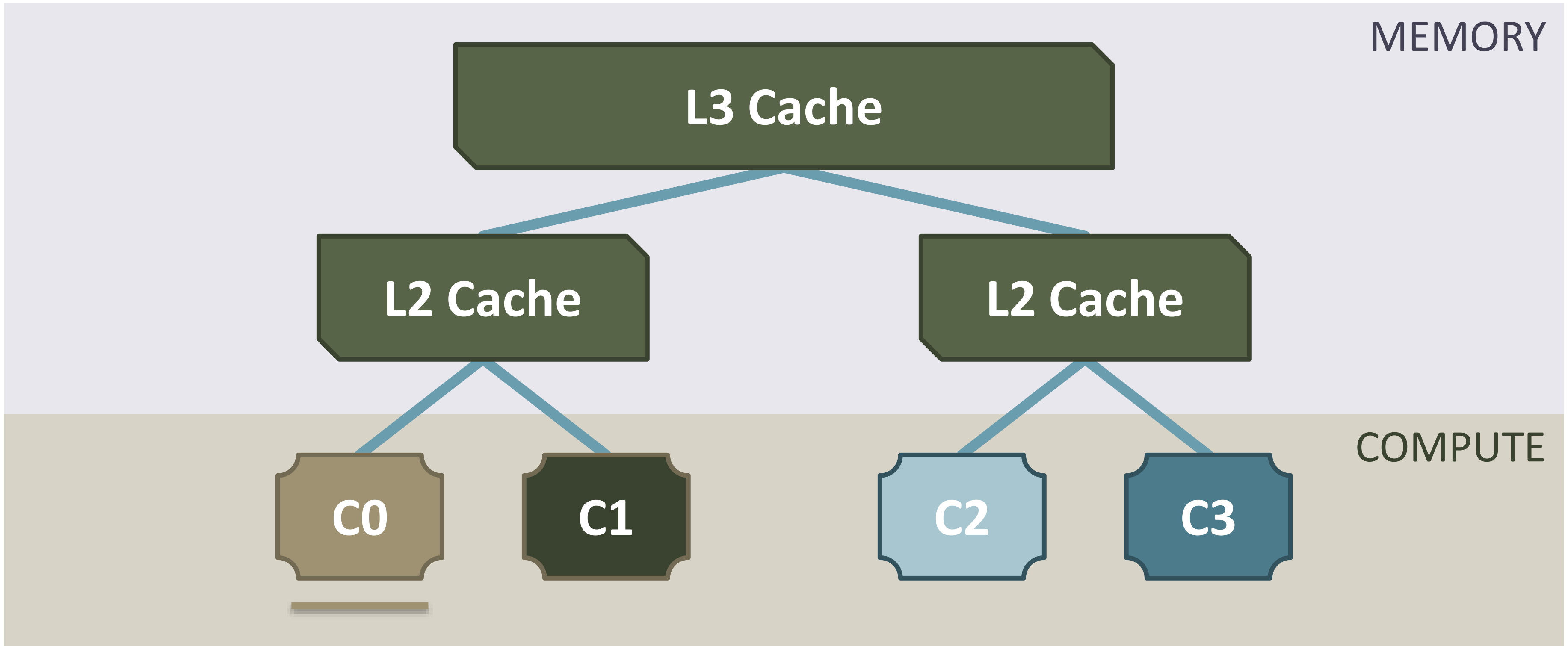
- Applicable to iterative algorithms that dynamically generate work
- BC: Boundary/Frontier represented as per-thread work queues

NUMA-Aware | Dynamic Work Queues



- Each Core/Thread has its own work queue
- When a thread completes, it aids other threads
- Work queues traversed in order of NUMA-distance

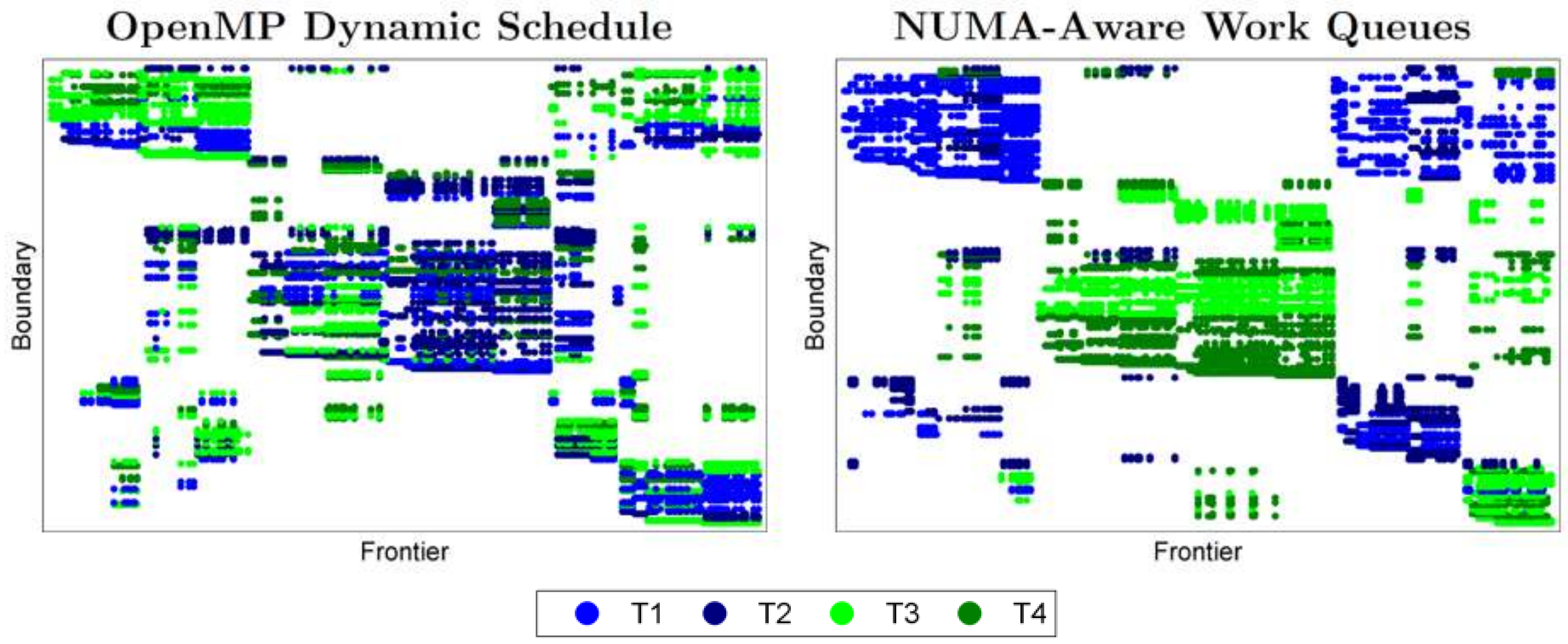
NUMA-Aware | Dynamic Work Queues



Work Queue Traversal

- T0: { C0, C1, C2, C3 }
- T1: { C1, C0, C3, C2 }
- T2: { C2, C3, C0, C1 }
- T3: { C3, C2, C1, C0 }

NUMA-Aware | Dynamic Work Queues

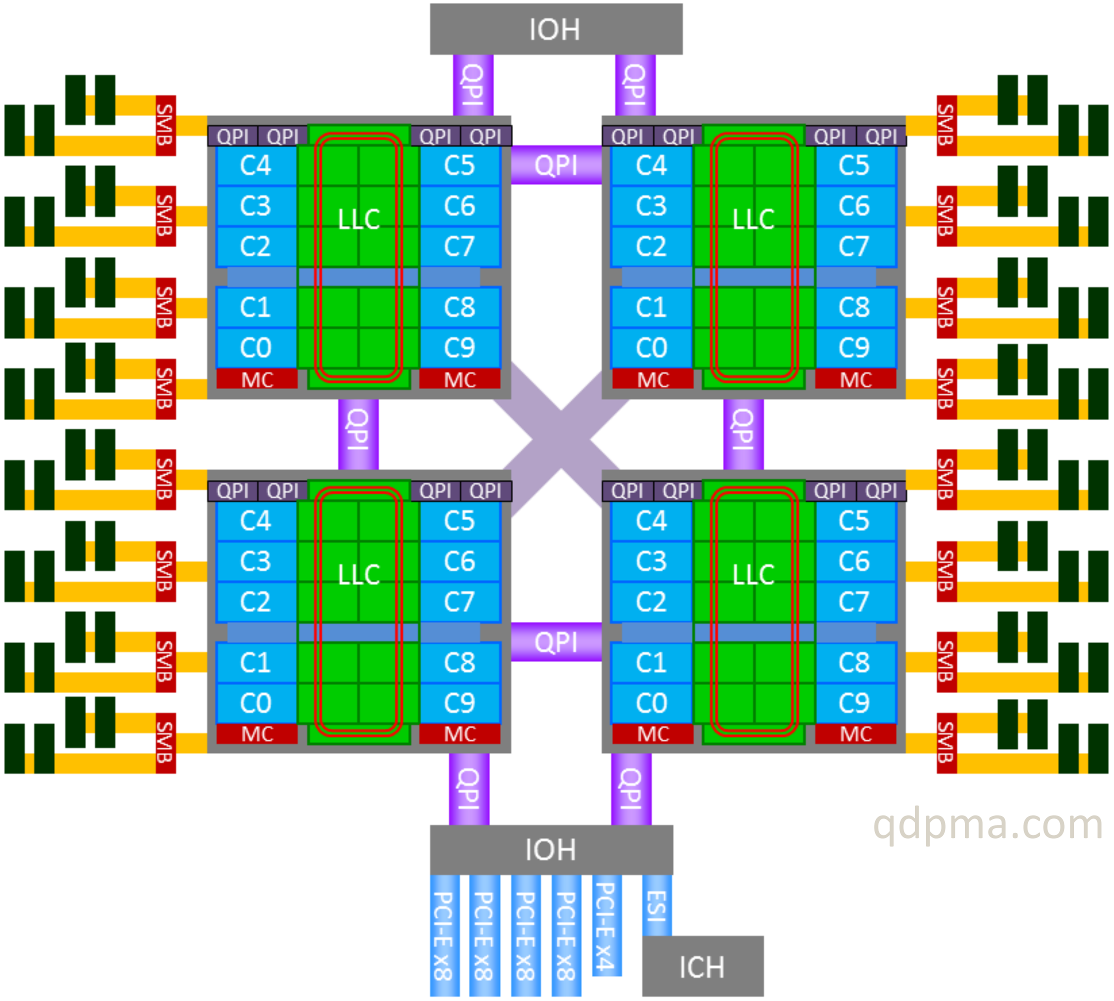


- Improved per-thread reuse at frontier
- Reduced frontier sharing
- Shared frontier likely between neighboring threads

Experimental Analysis

Experimental | Cache Architecture

- 4 x Intel Xeon E7-8837 (Westmere)
- 8 Cores per socket, 32 cores total
- Direct QPI Inter-Processor Communication
- 4 memory channels per socket

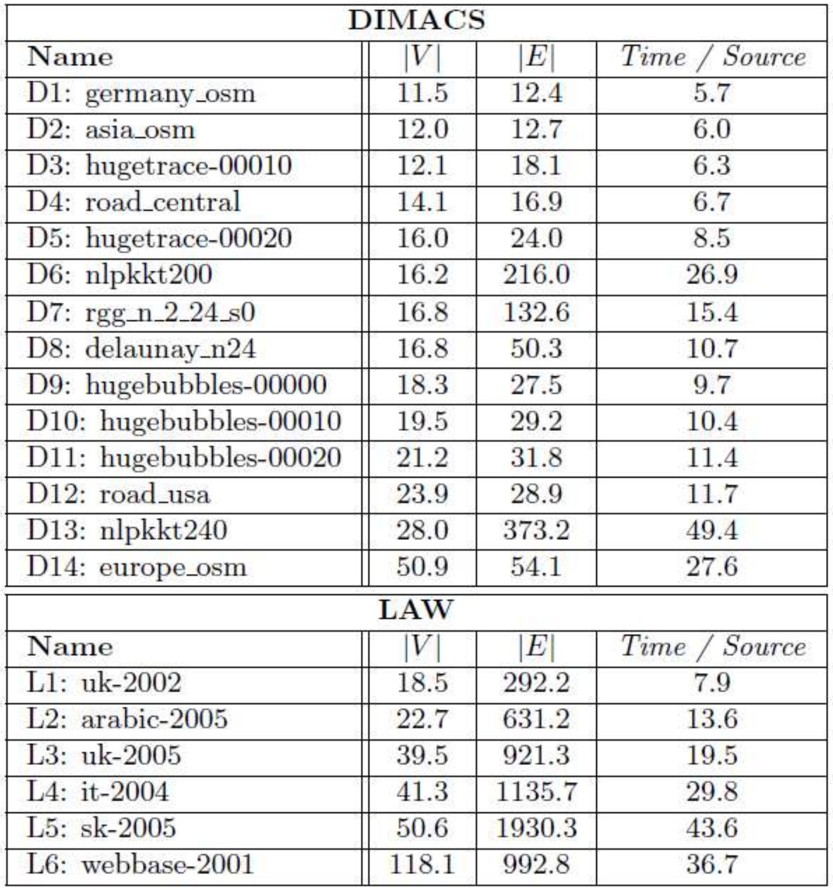


Experimental | Inputs

- 20 large sparse graphs
  - Road networks
  - Finite element meshes
  - Web crawls

- $|V| \in [\ 11.5,\ 118.1\ ]$ million
  $|E| \in [\ 12.4,\ 1930.3\ ]$ million

- Scaling from 1 to 32 cores
  - Measured time, power, cache

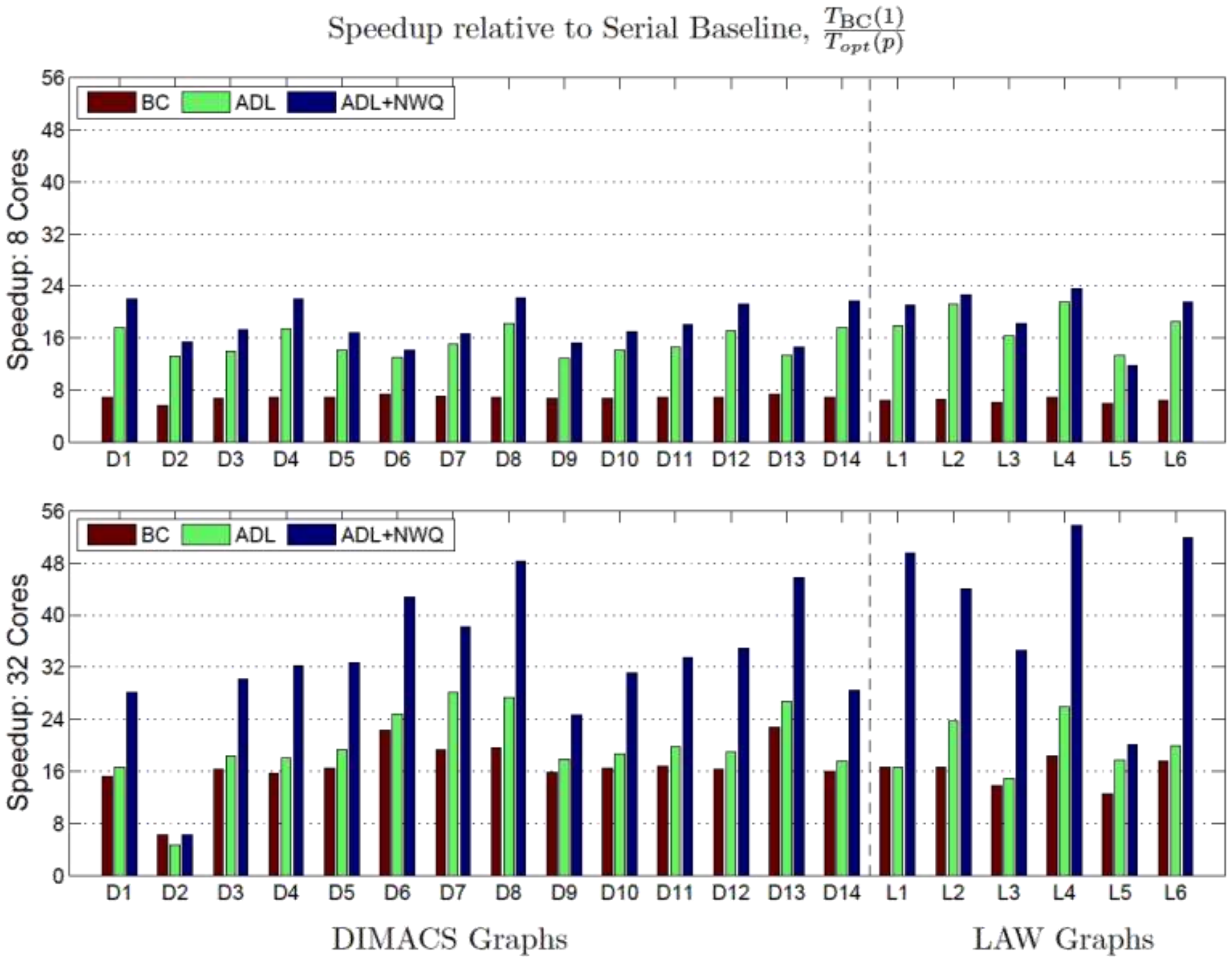


DIMACS			
Name	$\lvert V\rvert$	$\lvert E\rvert$	*Time / Source*
D1: germany_osm	11.5	12.4	5.7
D2: asia_osm	12.0	12.7	6.0
D3: hugetrace-00010	12.1	18.1	6.3
D4: road_central	14.1	16.9	6.7
D5: hugetrace-00020	16.0	24.0	8.5
D6: nlpkkt200	16.2	216.0	26.9
D7: rgg_n_2_24_s0	16.8	132.6	15.4
D8: delaunay_n24	16.8	50.3	10.7
D9: hugebubbles-00000	18.3	27.5	9.7
D10: hugebubbles-00010	19.5	29.2	10.4
D11: hugebubbles-00020	21.2	31.8	11.4
D12: road_usa	23.9	28.9	11.7
D13: nlpkkt240	28.0	373.2	49.4
D14: europe_osm	50.9	54.1	27.6

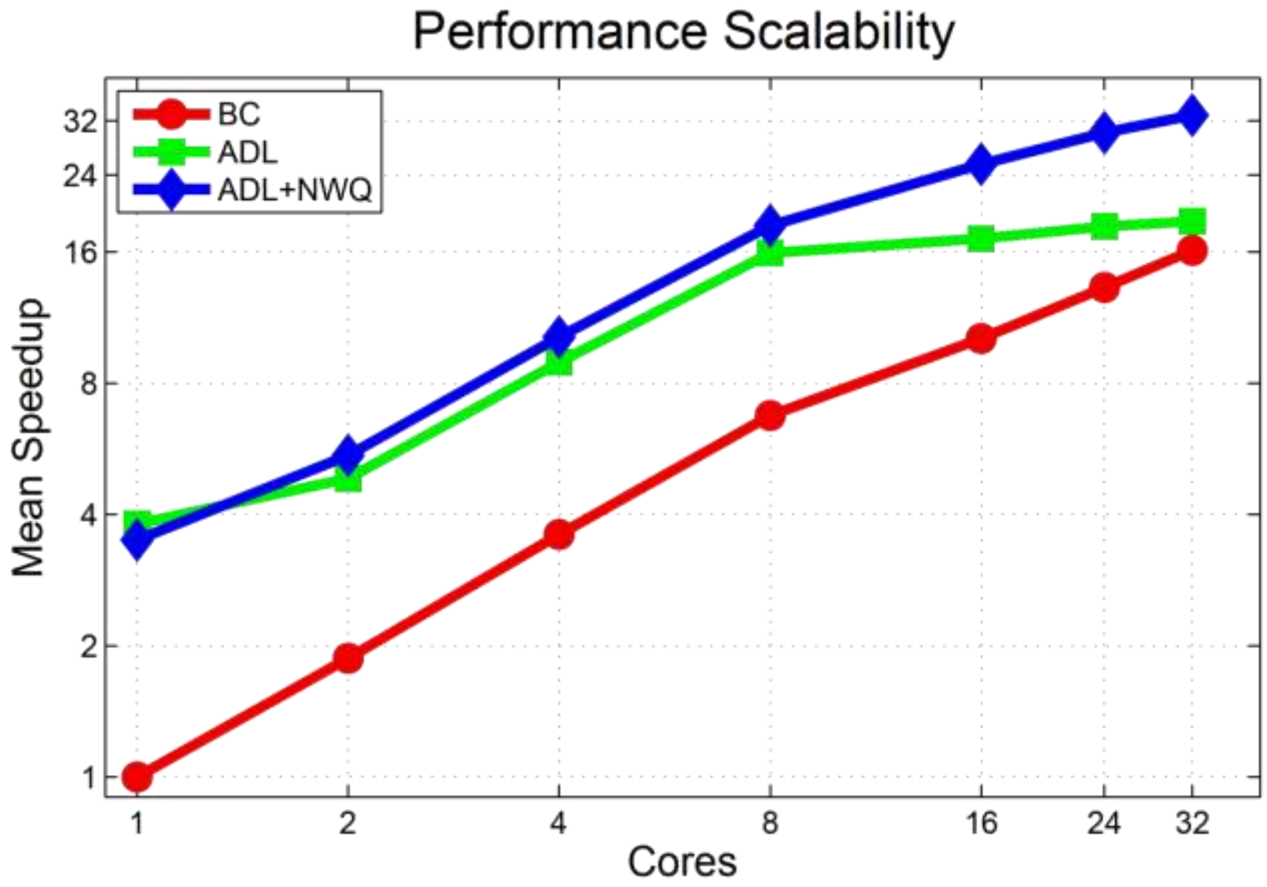
LAW			
Name	$\lvert V\rvert$	$\lvert E\rvert$	*Time / Source*
L1: uk-2002	18.5	292.2	7.9
L2: arabic-2005	22.7	631.2	13.6
L3: uk-2005	39.5	921.3	19.5
L4: it-2004	41.3	1135.7	29.8
L5: sk-2005	50.6	1930.3	43.6
L6: webbase-2001	118.1	992.8	36.7

$$\textbf{Speedup}(opt,\ p) = \frac{T_{\mathrm{BC}}(1)}{T_{opt}(p)} \qquad \textbf{Energy Reduction}(opt,\ p) = \frac{E_{\mathrm{BC}}(p) - E_{opt}(p)}{E_{\mathrm{BC}}(p)}$$

Performance Results | Per Input Speedup

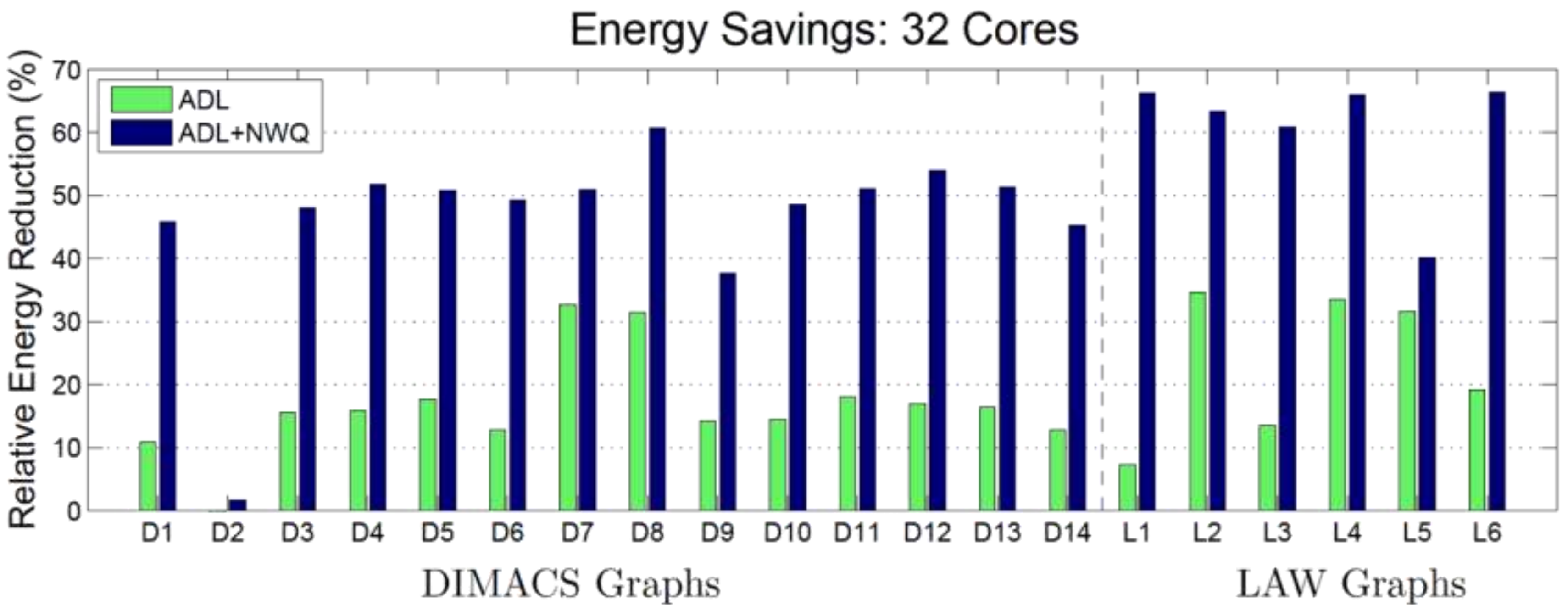


Performance Results | Average Workload Speedup

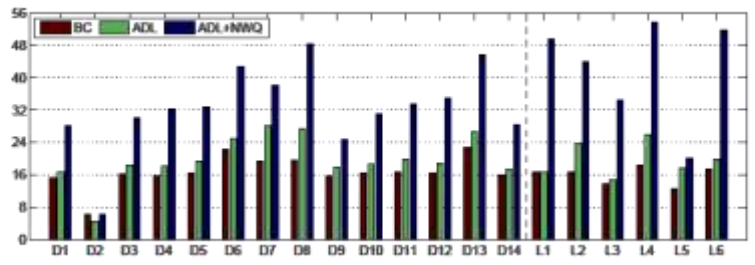


Speedup	BC	ADL	ADL+NWQ
8 Threads	6.8x	16.0x	18.4x
32 Threads	16.9x	20.2x	32.9x

Energy Results | Per Input Savings



- Mean energy reduction savings
  - ADL: 17.9%
  - ADL+NWQ: 52.4%
- Reasonably correlated with speedup

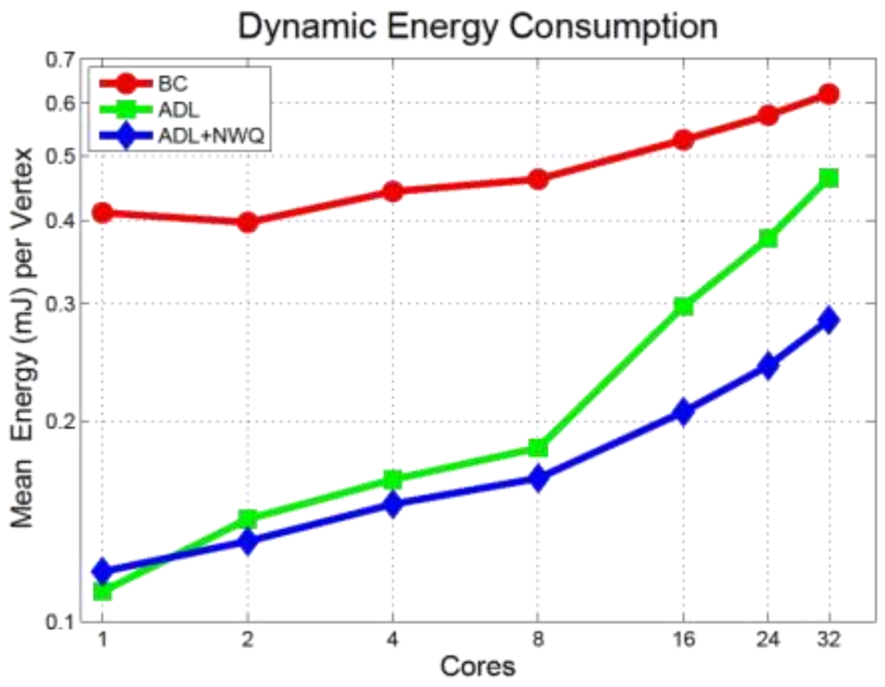


speedup

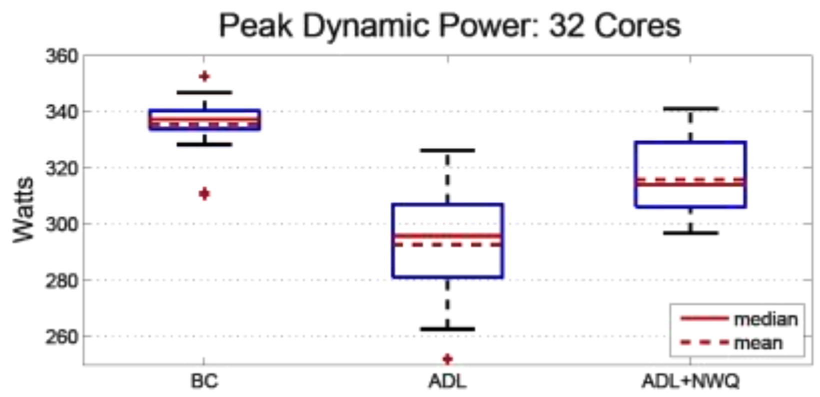
Energy Results | Understating Consumption

- Static Energy
  - Power consumed by system at idle
- Dynamic Energy
  - Increased power consumed during utilization
  - Arithmetic, logic, branch units, cache and DRAM
- Efficient code uses less of both
  - Reduced runtime => less static power consumed
  - Fewer cache misses, branch miss-predictions, pipeline stalls => less dynamic power consumed

Energy Results | Scaling Trends

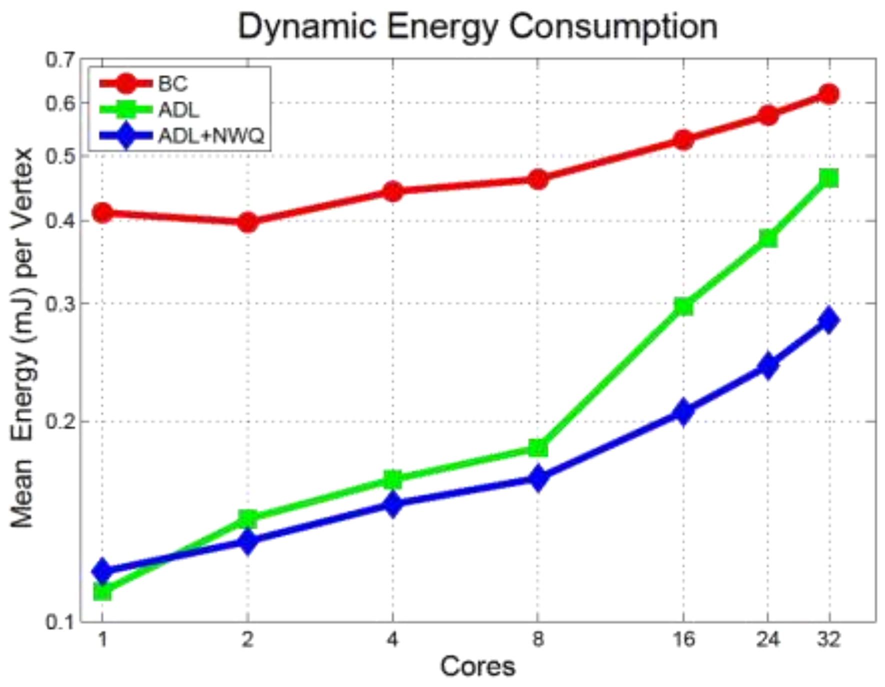


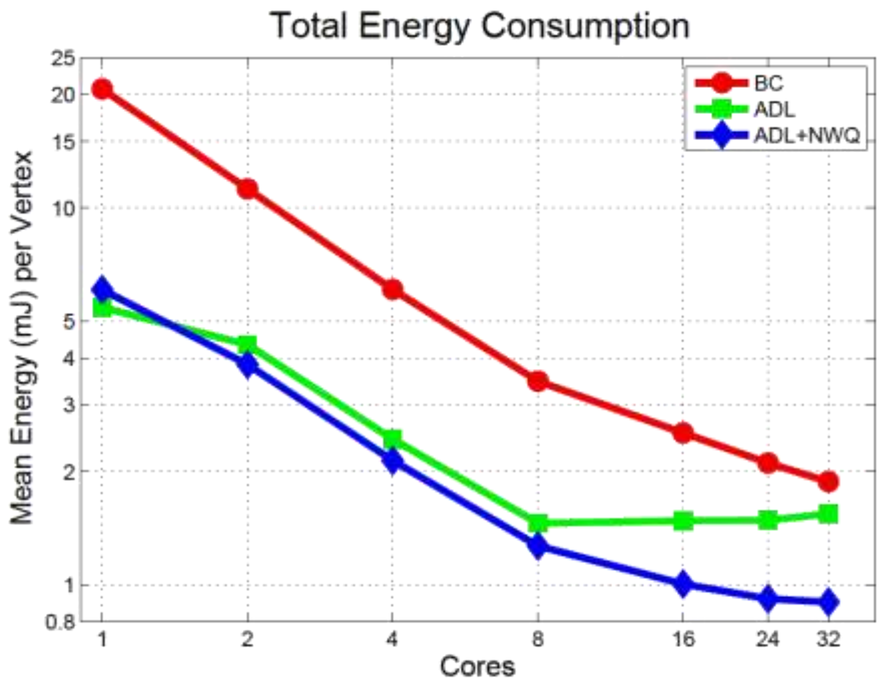
Dynamic energy increases due to parallel overheads

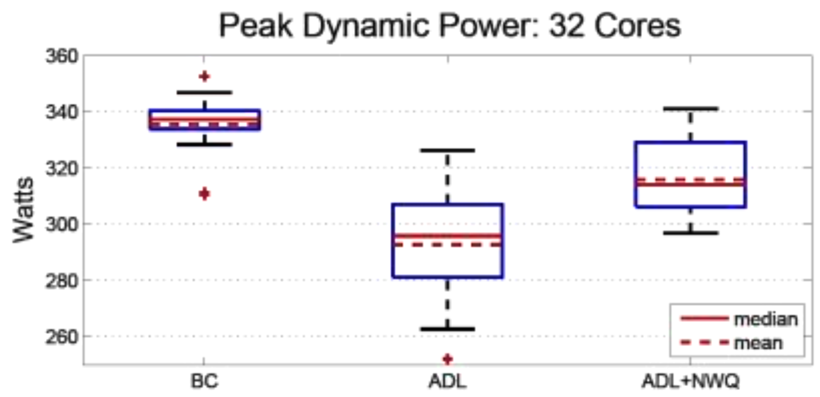


- Improved dynamic energy requirements

Energy Results | Scaling Trends

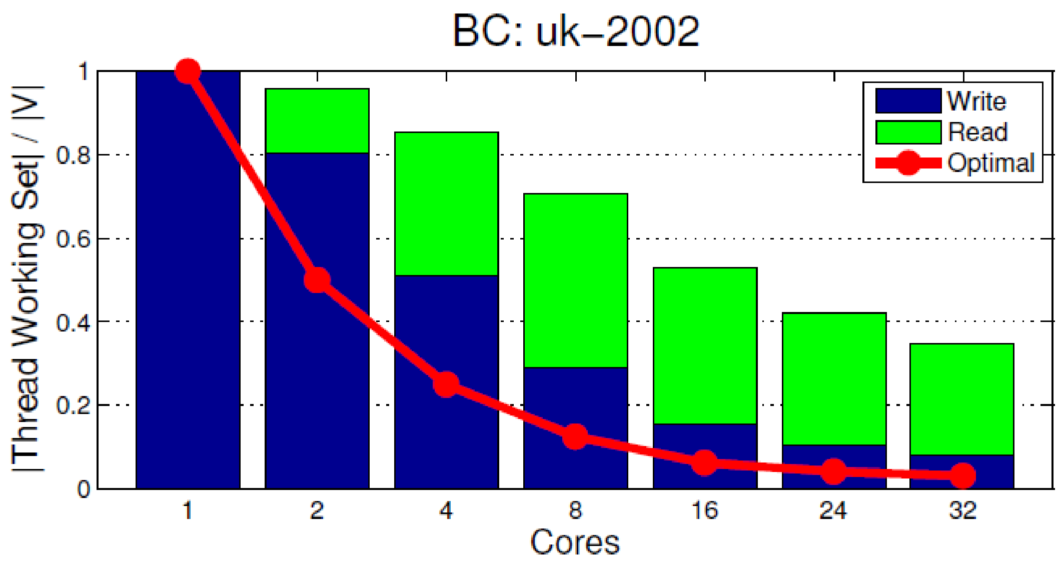




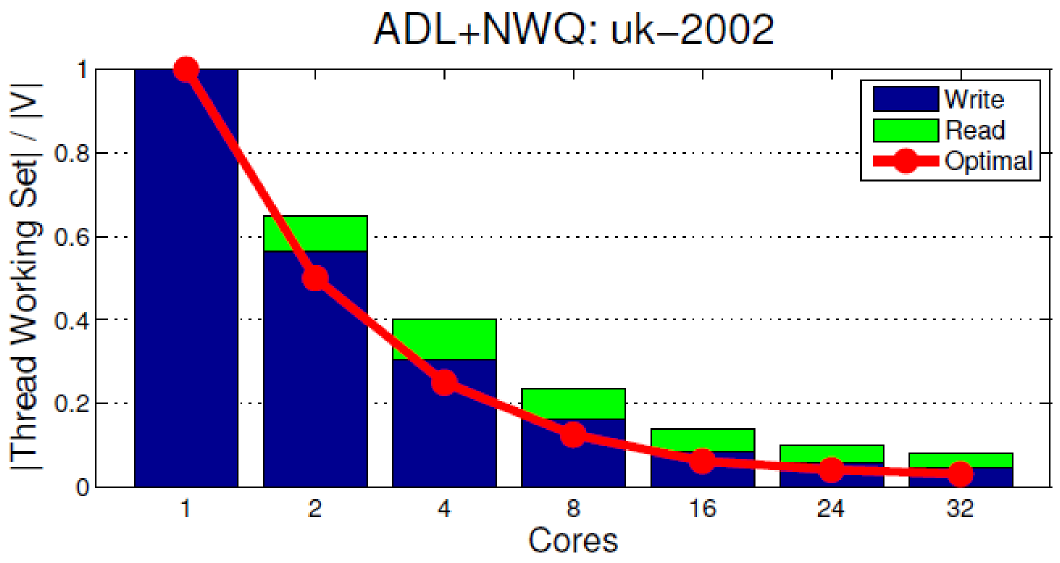


- Improved dynamic energy requirements
- Better performance creates energy savings at scale

Working Set Analysis

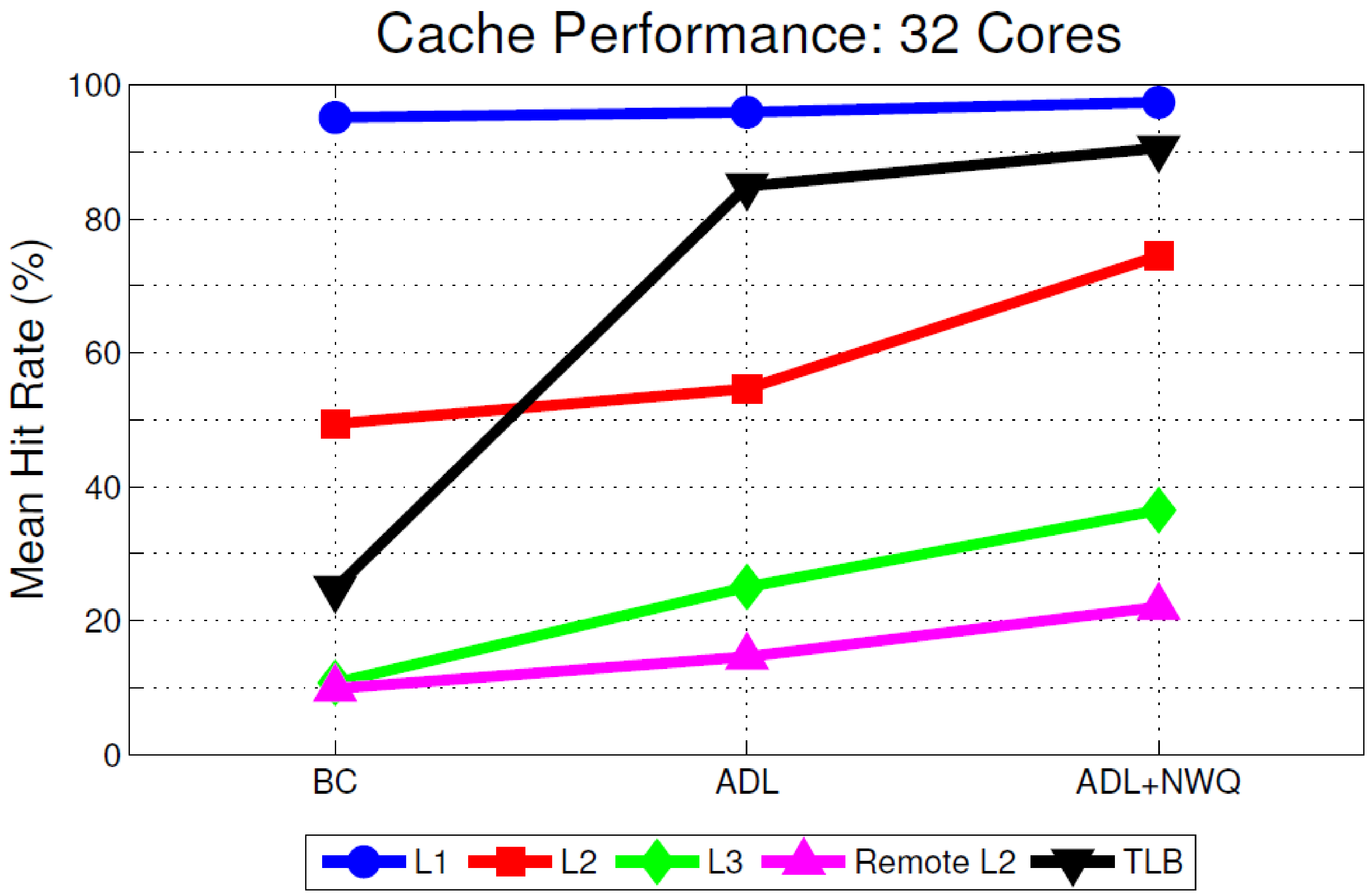


- Considerable per-thread overlap
- Inefficient use of memory bandwidth across sockets
- High costs associated with coherence traffic



- NUMA-Aware scheduling reduces redundancy
- More efficient use of cache space
- Reduced cache invalidations

Working Set Analysis | Cache Impact



Conclusions

- Memory issues have great impact as we scale algorithms and architectures

- We believe dynamic runtime environments are key in exploiting workload specific variability

Conclusions | Future Directions

NUMA-Aware Workload Scheduling

- Adapting scheduler for other irregular algorithms
- Incorporating other forms of system heterogeneity
- Detailed analysis of cache behavior via simulation
  - Location of shared frontier sets within the NUMA hierarchy
  - Impact on load at functional units (e.g. reordering, branches)
- NUMA-Aware graph data structures
- Appropriate for distributed memory?

thank you