Scheduling Parallel Programs by Work Stealing with Private Deques

Umut Acar
Carnegie Mellon University

Arthur Charguéraud
INRIA

Mike Rainey
Max Planck Institute for Software Systems

PPoPP
25.2.2013
Scheduling parallel tasks
Scheduling parallel tasks

set of cores
Scheduling parallel tasks

pool of tasks

Friday, July 5, 13
Scheduling parallel tasks

- Goal: dynamic load balancing
- A centralized approach: does not scale up
- Popular approach: work stealing
- Our work: study implementations of work stealing
Work stealing
Work stealing

deque
Work stealing
Work stealing

pop push

pop push

pop push

pop push
Work stealing
Work stealing

steal
Work stealing
Concurrent deques

- Deques are shared.
- Two sources of race:
  - between thieves
  - between owner and thief
- Chase-Lev data structure resolves these races using atomic compare&swap and memory fences.
Concurrent deques

- **Well studied:** shown to perform well both in theory and in practice ...

however, researchers identified two main limitations

- **Runtime overhead:** In a relaxed memory model, `pop` must use a memory fence.

- **Lack of flexibility:** Simple extensions (e.g., steal half) involve major challenges.
Previous studies of private deques

<table>
<thead>
<tr>
<th>Author</th>
<th>Year</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feeley</td>
<td>1992</td>
<td>Multilisp</td>
</tr>
<tr>
<td>Hendler &amp; Shavit</td>
<td>2002</td>
<td>C</td>
</tr>
<tr>
<td>Umatani</td>
<td>2003</td>
<td>Java</td>
</tr>
<tr>
<td>Hirashi et al.</td>
<td>2009</td>
<td>C</td>
</tr>
<tr>
<td>Sanchez et al.</td>
<td>2010</td>
<td>C</td>
</tr>
<tr>
<td>Fluet et al.</td>
<td>2011</td>
<td>Parallel ML</td>
</tr>
</tbody>
</table>
Private deques

- Each core has exclusive access to its own deque.
- An idle core obtains a task by making a **steal request**.
- A busy core regularly checks for incoming requests.
Private deques

Addresses the main limitations of concurrent deques:

• no need for memory fence
• flexible deques (any data structure can be used)

but

• new cost associated with regular polling
• additional delay associated with steals
Unknowns of private deques

• What is the best way to implement work stealing with private deques?

• How does it compare on state of art benchmarks with concurrent deques?

• Can establish tight bounds on the runtime?
Unkowns of private deques

• What is the best way to implement work stealing with private deques?

We give a receiver- and a sender-initiated algorithm.

• How does it compare on state of art benchmarks with concurrent deques?

We evaluate on a collection of benchmarks.

• Can establish tight bounds on the runtime?

We prove a theorem w.r.t. delay and polling overhead.
Receiver initiated

1

-1

-1

-1

-1

1

2

3

4
Receiver initiated
Receiver initiated

1

-1

CAS

Friday, July 5, 13
Receiver initiated

CAS

1

2

-1

-1

-1

1

2

3

4
Receiver initiated

2

-1

-1

-1

1

2

3

4
Receiver initiated
Receiver initiated
Receiver initiated
From receiver to sender initiated

- Receiver initiated: each idle core targets one busy core at random
- Sender initiated: each busy core targets one core at random
- Sender initiated idea is adapted from distributed computing.
- Sender initiated is simpler to implement.
Sender initiated

... ... ... ...

1 2 3 4
Sender initiated

... 0 ...

1 2 3 4
Sender initiated

CAS
Sender initiated

... → CAS

1 2 3 4
Sender initiated
Sender initiated
Sender initiated
Performance study

• We implemented in our own C++ library:
  • our receiver-initiated algorithm
  • our sender-initiated algorithm
  • our Chase-Lev implementation

• We compare all of those implementations against Cilk Plus.
Benchmarks

• Classic Cilk benchmarks and Problem Based Benchmark Suite (Blelloch et al 2012)

• Problem areas: merge sort, sample sort, maximal independent set, maximal matching, convex hull, fibonacci, and dense matrix multiply.
Performance results

Intel Xeon, 30 cores
polling period = 30µsec

Normalized run time

- matmul
- cilk sort (expintseq)
- cilk sort (randintseq)
- fib
- matching (eggrid2d)
- matching (egrfg)
- matching (egrmat)
- MIS (grid2d)
- MIS (rg)
- MIS (rmat)
- hull (plummer2d)
- hull (uniform2d)

Concurrent deques
Sender init
Receiver init
Cilk Plus

Normalized execution time

Friday, July 5, 13
Analytical model

\( P \) number of cores
\( T_I \) serial run time
\( T_\infty \) minimal run time with infinite cores
\( T_P \) parallel run time with \( P \) cores
\( \delta \) polling interval
\( F \) maximal number of forks in a path
Our main analytical result

Bound for greedy schedulers:

\[ T_P \leq \frac{T_1}{P} + \frac{P-1}{P} T_\infty \]

Bound for concurrent deques (ignoring cost of fences):

\[ \mathbb{E}[T_P] \leq \frac{T_1}{P} + \frac{P-1}{P} T_\infty + O(F) \]

Bound for our two algorithms:

\[ \mathbb{E}[T_P] \leq \left( \frac{T_1}{P} + \frac{P-1}{P} T_\infty + O(\delta F) \right) \cdot \left( 1 + \frac{O(1)}{\delta} \right) \]
Our main analytical result

Bound for greedy schedulers:

\[ T_P \leq \frac{T_1}{P} + \frac{P-1}{P} T_\infty \]

Bound for concurrent deques (ignoring cost of fences):

\[ \mathbb{E} [T_P] \leq \frac{T_1}{P} + \frac{P-1}{P} T_\infty + O(F) \]

Bound for our two algorithms:

\[ \mathbb{E} [T_P] \leq \left( \frac{T_1}{P} + \frac{P-1}{P} T_\infty + O(\delta F) \right) \cdot \left( 1 + \frac{O(1)}{\delta} \right) \]
Our main analytical result

Bound for greedy schedulers:

\[ T_P \leq \frac{T_1}{P} + \frac{P-1}{P} T_\infty \]

Bound for concurrent deques (ignoring cost of fences):

\[ \mathbb{E} [T_P] \leq \frac{T_1}{P} + \frac{P-1}{P} T_\infty + O(F) \]

Bound for our two algorithms:

\[ \mathbb{E} [T_P] \leq \left( \frac{T_1}{P} + \frac{P-1}{P} T_\infty + O(\delta F) \right) \cdot \left( 1 + \frac{O(1)}{\delta} \right) \]
Conclusion

• We presented two new private-deques algorithms, evaluated them, and proved analytical results.

• In the paper, we demonstrated the flexibility of private deques by implementing the steal half policy.