Snug: Architectural Support for Relaxed Concurrent Priority Queueing in Chip Multiprocessors

Azin Heidarshenas†*, Tanmay Gangwani†*, Serif Yesil†, Adam Morrison‡, and Josep Torrellas†

University of Illinois Urbana-Champaign†
Tel-Aviv University‡

International Conference on Supercomputing (ICS), June 2020

*Both authors contributed equally to this research.
Priority-based Task Scheduling

Tasks are executed based on their *priority order*
If $T_1$ has higher priority than $T_2$ it gets executed before $T_2$

Example: Dijkstra’s Single-Source Shortest Path (SSSP)
  - Source vertex A

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>
Concurrent Priority Queue (PQ)

Parallel threads **dequeue** (pop) tasks and **enqueue** (push) new ones

Head ptr → highest priority

Core 1

Core 2

Core 3

Enqueue

Dequeue

Dequeue

Not scalable due to synchronization bottleneck

1 2 4 7

Compare&Swap(&head_ptr, …)
Relaxed PQ

Relaxed Priority $\Rightarrow$ alleviate synchronization at the dequeues

Core 1
Core 2
Core 3

Dequeue
Dequeue

Head ptr $\Rightarrow$ highest priority

In many applications, relaxed priorities don’t harm correctness
Graph applications, discrete event simulation, ...
Relaxed PQ: Synch. vs Wasted Work

Relaxed PQ doesn’t enforce ordered execution of tasks
Lower priority task can be overwritten by a higher priority one

Example: Single-Source Shortest Path (SSSP)

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>∞</td>
</tr>
<tr>
<td>B</td>
<td>∞ 2</td>
</tr>
<tr>
<td>C</td>
<td>∞</td>
</tr>
</tbody>
</table>

![Diagram of a graph with vertices A, B, and C, and edges A-B with weight 3, A-C with weight 1, and C-B with weight 1.]

A, 0  C, 1  B, 2
Overview of Snug

Wasted work vs. synchronization trade-off

Local enqueues, relaxed global dequeues

Core 1

Core 2

Core 3

Enqueue

Enqueue

Enqueue

Work Registers

Pointer

Priority

0x...

0x...

0x...

Software

Hardware
Snug Programming Model

Programmer

Application

Snug Library

New Instructions

AllocHeads
UpdateHead
FetchHead
PickHead

Snug Architecture

0x...
0x...
0x...

Work Registers

One logical queue

Multiple physical queues, distributed
Snug Architecture

Each core has access to a set of work registers and a PickHead module.
Gather ptrs to queue heads + priorities

Sorting heads of physical queues
Return one priority at random
Then, SW performs a CAS

PM
0x...10
PM
0x...3
PM
0x...5
PM
0x...15
PM
0x...24
PM
0x...12
PM
0x...1
PM
0x...8
PM
0x...20
PM
0x...17
PM
0x...9
PM
0x...2
PM
0x...7
PM
0x...6
PM
0x...21
PM
0x...4

PickHead Module

Snapshot Memory
ptr, 1
ptr, 2
ptr, 3
ptr, 4
ptr, 5
...
ptr, 24

Relaxation Count (R)
Other Modes of PickHead Instruction

**Global Access:** gathers the heads of all queues

**No Network Access:** reuses the snapshot before it gets stale

**Local Access:** uses the head of the local queue
Other Modes of PickHead Instruction

- Global Access
- No Network Access
- Local Access

PickHead Module

- ptr, 1
- ptr, 2
- ptr, 3
- ptr, 4
- ptr, 5
- ... (Repeat U times)
- ptr, 24

Relaxation Count (R)

No Network Access

Local Access

... (Repeat L times)

Local Access

0x...

...
Adaptivity of Snug to CAS Failures

SW performs a CAS on the chosen queue. It can fail.

If failure: retry the same queue

Snug adapts to contention
  - If frequent CAS failures → increase R.
  - If rare CAS failures → decrease R.

Relaxation Count (R)
PickHead Module

- Selected Head
  - Tag
  - Snapshot Memory
  - Sorter & Selector
  - Tag Counter
  - Work Registers

- Decision
  - PickHead Instruction
  - Reuse Snapshot
  - Local/Global Dequeue
  - Request Generator
  - DEMA
  - R Reg
Evaluation Setup

64-core simulations in Gem5, \( U = 4 \) and \( L = 4 \)
Applications: SSSP, BFS, SIMUL, A*

PQs evaluated:
- SW-SK: Concurrent skiplist\(^1\) implementation (baseline)
  - Always dequeues the highest priority task
- SW-SP: Concurrent spraylist\(^2\)
  - Dequeues are sprayed over a range of high priority tasks
- SW-D: Distributed concurrent skiplist with work-stealing
  - Local enqueues, local dequeues
- HW-C: Centralized version of Snug. No PickHead
- HW-D: Snug
  - Local enqueues, global dequeues

Push and pop contention in SW-SK and SW-SP. Wasted work in SW-D

Snug achieves 1.4x, 2.4x, and 3.6x speedup over SW-SK, SW-SP and SW-D
Breakdown of Tasks

SW-D suffers from wasted work due to local dequeues
Adaptation of the Relaxation Count

R is able to adapt to the CAS contentions

**SIMUL_M2**
More in the Paper

- Analysis of Network Traffic
- Snug Scalability
- Sensitivity Analysis of Snug Parameters
- Characterizing R Adaptivity
- Power and Area Analysis
- ...
Take-away

- Relaxed PQ alleviates synchronization overhead but is prone to wasted work.

- Snug distributes PQ in hardware and minimizes both wasted work and synchronization overhead simultaneously.

- Snug’s relaxed PQ adapts to the rate of synchronization failures over time.

- For 64 cores: Snug achieves avg. speedup of 1.4x, 2.4x, and 3.6x speedup over skip PQ, spray PQ, and distributed concurrent skiplist.
Snug: Architectural Support for Relaxed Concurrent Priority Queueing in Chip Multiprocessors

Azin Heidarshenas†*, Tanmay Gangwani†*, Serif Yesil†, Adam Morrison‡, and Josep Torrellas†

University of Illinois Urbana-Champaign†
Tel-Aviv University‡

International Conference on Supercomputing (ICS), June 2020

*Both authors contributed equally to this research.