Impact of memory allocation on the performance of a delegation synchronization algorithm

Presented by **SID-LAKHDAR Riyane**
(M2 MoSIG: ENSIMAG / UJF)
Supervised by **ROPARS Thomas**
(LIG, team ERODS)

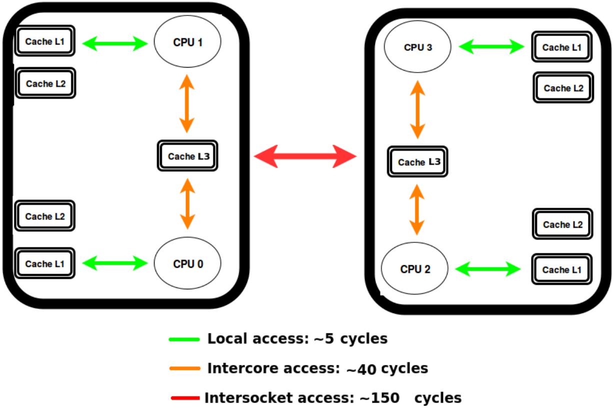
Motivation: the multithreading limitation

- Theoretical objective of multithreading:

$$\textit{Time}(\text{multithreaded task}) = \frac{\textit{Time}(\text{Sequential task})}{\text{Number of thread}}$$

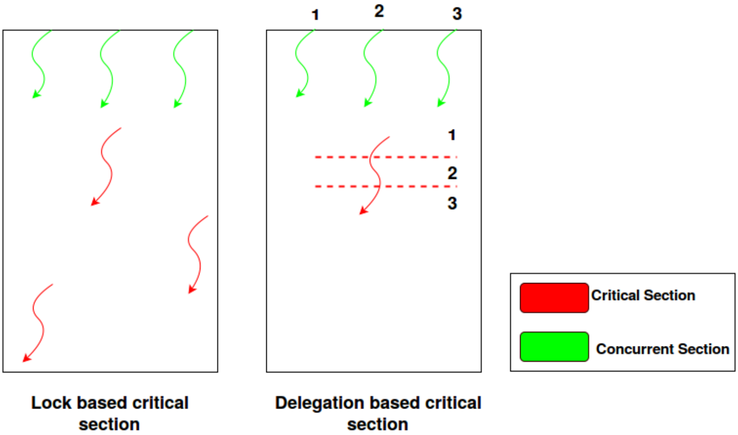
- Objective is still way of the mark

Motivation: Complex processor → complex allocator



- **Memory allocator** is an important cause of this limitation

MOTIVATION: DELEGATION APPROACH



Our contribution

Prove the impact of dynamic memory allocation on

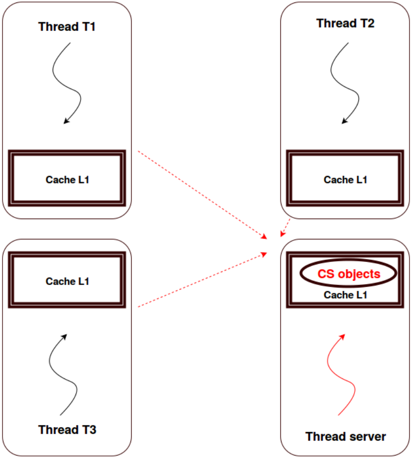
- The absolute performances of a custom multithreaded queue
- The relative performance of two custom delegation algorithms

THE DELEGATION APPROACH FOR MUTUAL EXCLUSION

MEMORY ALLOCATION: CHALLENGES AND PROPOSED SOLUTIONS

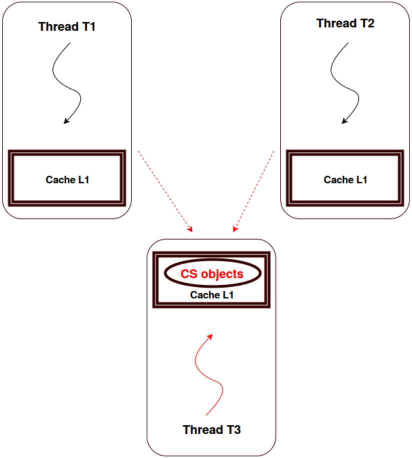
EXPERIMENTAL EVALUATIONS

The delegation approach

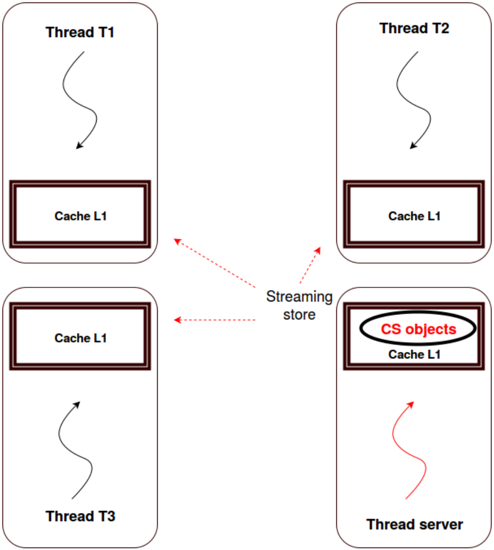


- **Improve cache locality**: memory accesses of the CS are mostly locals

THE COMBINER ALGORITHM



THE EXPECTED OPTIMIZATION: STREAMING STORE



Limit of the evaluation: the allocator

Used allocator:

- Large thread-local buffer used for all the allocations
- No returned memory

Unrealistic allocator:

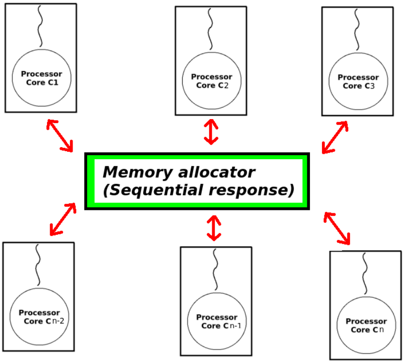
- Outperforms the general purpose allocators
- Cannot scale (memory footprint)

The delegation approach for mutual exclusion

Memory allocation: challenges and proposed solutions

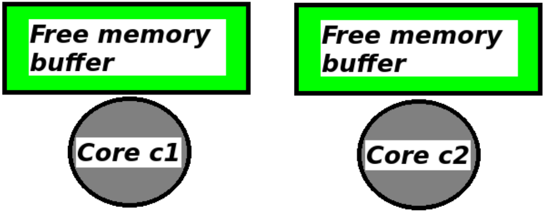
Experimental evaluations

MEMORY ALLOCATOR: A BOTTLENECK



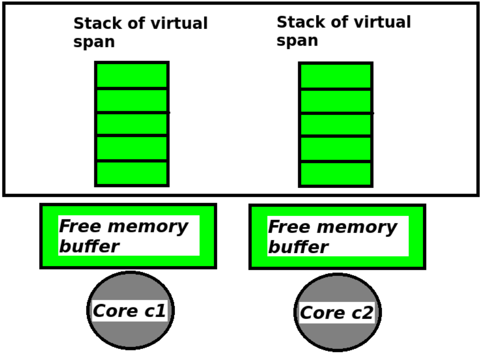
- Sequential memory allocation
- Inefficient due to contention

Core local buffer



- **Improved allocation time**
- **Buffer made of contiguous addresses** → low page-fault probability

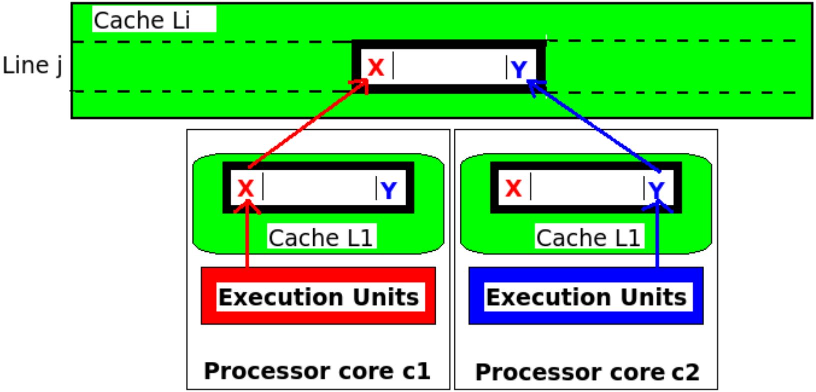
Further contention lightening



One stack per core:

- No synchronization between cores
- Lock free concurrent data structure

Software issues: False sharing



THE SPAN STRUCTURE

- **Span size is a multiple of a cache line size:** no false sharing
- **Simplified allocation algorithm**
- **Extra memory aligned to page size:** swapped out of the RAM

THE DELEGATION APPROACH FOR MUTUAL EXCLUSION

MEMORY ALLOCATION: CHALLENGES AND PROPOSED SOLUTIONS

EXPERIMENTAL EVALUATIONS

Designed test environment

Two main rules: Avoid overhead linked to **scheduling**

- **No more threads than cores**
- **Pin threads on independent cores**

Environments

- A 20 cores **Intel Xeon E5-2620** with a **Debian** kernel 3.16.7
- A 24 cores **AMD Opteron 6164** with a **Debian** kernel 3.16.7

Testbed

Tested algorithms:

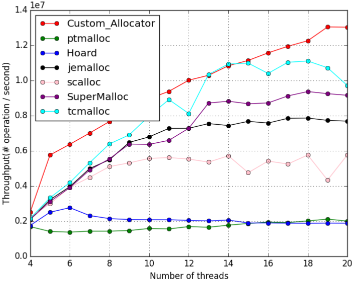
- Compare contention on a **Michael-Scott** queue
- Compare the **combiner** and the **server based** algorithms.

Considered memory allocators:

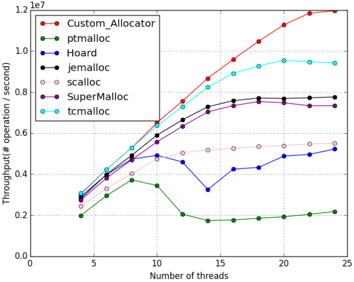
- **Custom allocator** (used in previous delegation evaluation)
- **ptmalloc** (C standard library)
- **hoard**
- **jemalloc**
- **scalloc**
- **supermalloc**
- **tcmalloc**

Allocator comparison (with the combiner algorithm)

Compare the throughput of the **allocators**: The higher the better



(f) Intel Xeon E5-2620

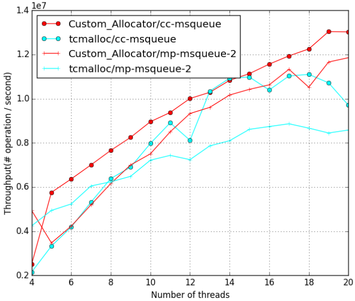


(g) AMD Opteron 6164

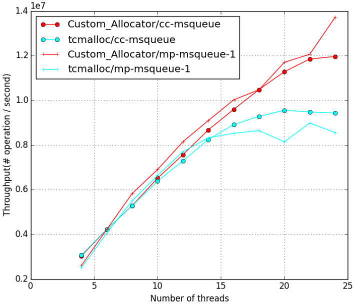
- tcmalloc outperforms the other general purpose allocators

Delegation algorithms comparison

Compare the throughput of the **delegation algorithms**: The higher the better



(h) Intel Xeon E5-2620



(i) AMD Opteron 6164

- With TCmalloc, the relative performance of the algorithms change on AMD

RECAP

During this study, we have:

- Highlighted the impact of memory allocation on multithreaded algorithms
- Infirmed the performance property of two delegation algorithms

Open issues: understand the reasons of the noticed behavior

- The allocation design of tcmalloc
- The processor allocation policy

Impact of memory allocation on the performance of a delegation synchronization algorithm

Presented by **SID-LAKHDAR Riyane**
(M2 MoSIG: ENSIMAG / UJF)
Directed by **ROPARS Thomas**
(LIG, team ERODS)

June 21, 2016