Thread and Memory Placement on NUMA Systems: Asymmetry Matters

Baptiste Lepers, Alexandra Fedorova
Simon Fraser University

Vivien Quéma
Grenoble University
Machines are NUMA
Machines are NUMA
This talk

Interconnects.
Let’s execute an application...

Node 0

Node 1

Node 2

Node 3

Execution time: 149s
Let’s execute the same application again...

Node 0

Node 1

Node 2

Node 3

Memory

Memory

Memory

Memory

Execution time: 277s!
Why?!
Interconnects have different bandwidths

Some interconnects are fast, some are slow!
Modern machines are even more complex
No direct link between node 0 and 7, 0 will do “2-hops” to access 7
Fast (6GB/s) and slow (3GB/s)
Fast in only one direction (read 4GB/s, write 3GB/s)
Unidirectional links

Node 3

Node 7
Streamcluster running on 2 nodes
Streamcluster running on 2 nodes

Some 2-hops configurations are faster than some 1-hop configurations
Bandwidth is more important than latency
Current optimizations

• Avoid 2-hops (Linux, ...)

• Place I/O threads close to I/O nodes
Our solution: AsymSched

Asymmetry aware scheduling
Tries to maximize bandwidth between communicating threads
Overview

- Thread migration
  - Place threads on well interconnected nodes

- Memory placement
  - Dynamic memory migration for small working sets
  - Fast bulk memory migration otherwise

- Continuous profiling in background
- Takes decisions every second
Step one: cluster threads
Limitations

• Hardware counters work at the scale of a node
  • E.g.: Node 0 did an access to node 7

• So we cluster **per node**.

• We only cluster threads that have the same pid.
Step two: migrate threads

- Migration is done on a node basis
  - We move all threads running on a node to another node.
Challenges

- Find the best placement
  - I.e., the placement that maximizes bandwidth between threads.

- The number of placements is huge
  - Up to factorial(#nodes)
  - We skip “obviously bad” configurations
    - Skip placements that use the “slowest” links
  - We only do computations on non-equivalent configurations.
    - Hash function placement -> generic placement.
Step three: migrate memory

T1 and T2 might continue accessing memory located on the previous node of T1
Implementation

- We use IBS to detect accessed pages

- It is not precise, and might not be sufficient
  - Do full memory migration in that case

- Problem: Linux system call takes 5.1s to migrate 1GB!
  - Our workloads use up to 30GB of RAM.
Fast memory migration

- Implementation:
  - Freeze the application (SIGSTOP)
  - Compute a list of all pages to migrate
  - Modify PTEs directly

- No lock
- Only limited by interconnect bandwidth
- Migrate memory from multiple nodes in parallel

- Migrates 1GB from 1 node in 0.3s (17x faster than Linux)
- Migrates 2GB from 2 nodes in 0.3s (34x)
Evaluation (1/4)
Evaluation (2/4)

Performance improvement / average placement (%)

-100%  -50%  0%  50%  100%  150%  200%  250%

-100% -50%  0%  50%  100%

streamcluster  pca  facerec

-\text{Worst thread placement}
-\text{Best thread placement}
-\text{Dynamic memory migration only}
-\text{Asymsched}
Evaluation (3/4)

![Performance improvement diagram](image)

- **Worst thread placement**
- **Best thread placement**
- **Dynamic memory migration only**
- **Asymsched**
Evaluation - Multiapp (4/4)

Performance improvement / average placement (%)

Worst thread placement
Best thread placement
Dynamic memory migration only
Asymsched
Conclusion

• Systems should maximize bandwidth between threads

• Asymsched
  • Up to 200% faster than average placement
  • Up to 91% faster than dynamic memory migration alone
Questions?