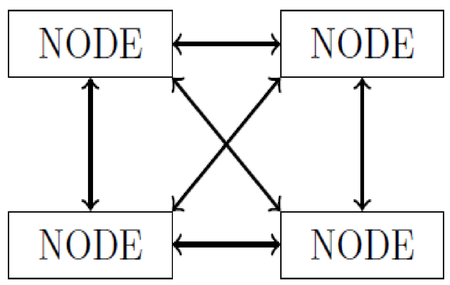
NUMA Aware Garbage Collection

Maria Patrou, Panagiotis Patros, Kenneth B. Kent, Gerhard Dueck
University of New Brunswick, IBM Canada
Faculty of Computer Science
{Maria.Patrou, Patros.Panos, ken, gdueck}@unb.ca

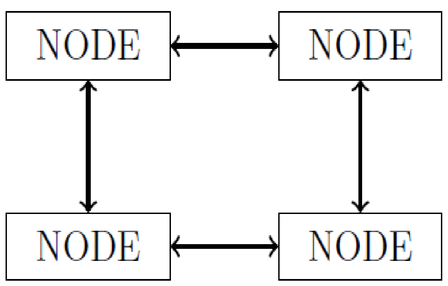
Background

Non Uniform Memory Access (NUMA) is a memory design used in multiprocessing. In NUMA architectures, nodes are composed of associated physical processors (CPUs). Each node has its own memory and connection bus. There are different kinds of NUMA node connections:

Double NUMA

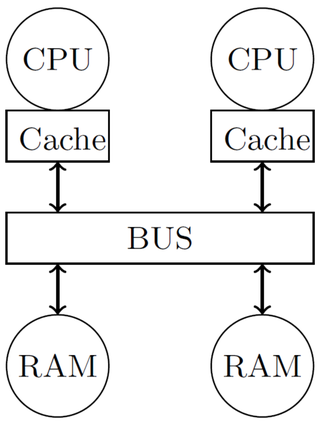


Quad NUMA

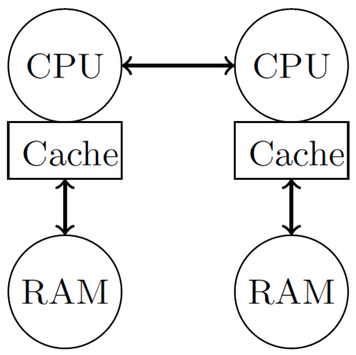


Square NUMA

The main benefit of NUMA over traditional UMA is scalability, which is performed by increasing the number of nodes.



Uniform Memory Access
All memory accesses have the same cost

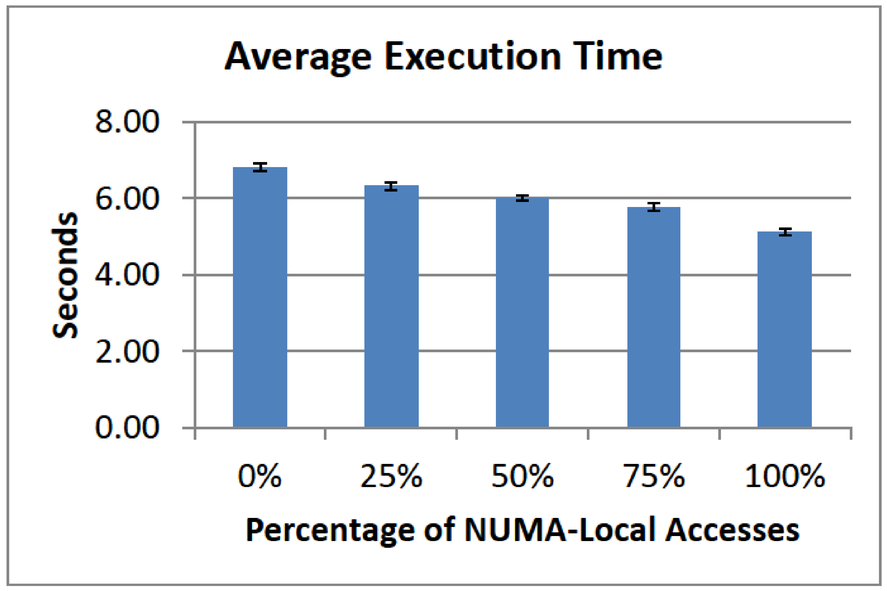


Non Uniform Memory Access
Memory accesses have varying cost

Cache coherence is important in the performance of NUMA. Storing data in the cache improves the retrieval speed of objects. However, different processors can locally access and change objects, causing reading and writing from/to memory. This is done to keep remote copies in sync, which slows down the program's performance.

Motivation

Accesses within node memory are faster than inter-node ones. So the number of remote accesses should be minimized.
Also, there are many cache issues, such as cache misses, that increase memory and cache interaction. False sharing is also an issue, since we do not want untouched and updated objects to be in the same cache lines, causing the unnecessary updates of the whole line.
Lastly, there are problems with Garbage Collection (GC) in NUMA systems. Grouping threads and objects on nodes can improve some applications, but this is not general enough to be beneficial for all.



Testing on a Double NUMA node computer with 8/16 cores suggests that no NUMA-local vs all NUMA-local accesses are slower by 33%.

Hypothesis

We aim to test if different heuristics would improve NUMA performance:

- Minimization of remote accesses
- Workload balance between nodes
- Cache coherence and garbage collection

We will investigate regrouping objects and threads between nodes, based on volatile objects, cache issues, performing GC per node at different times etc.

Experimental Evaluation

1. NUMA simulator; to retrieve useful statistics about the state-of-the-art algorithms.
2. Multithreaded application that will perform random and synchronized memory accesses across multiple NUMA nodes.
3. Capturing real memory accesses from Java applications and cache simulator.
4. Combining these, we expect to come up with heuristics or a model that will improve applications' performance.