Getting Rid of Coherency Overhead for Memory-Hungry Applications

Héctor Montaner
Federico Silla
Holger Fröning
Jose Duato

September 21, 2010
Index

Introduction

A new shared-memory architecture

Implementation

Results

Conclusions & future work
Introduction

High-Performance Computing frequently relies on x86-based clusters

CLUSTER → Nodes with private memory → Message Passing Paradigm

Shared memory applications are confined to a single node

x86 machines provide up to 64 cores and a few hundreds of GB of memory
How can we release these shared-memory applications so that they could use resources outside a node?
Introduction

- Many commercial attempts to aggregate nodes...
  - ScaleMP, 3Leaf, Numascale, etc
- ... but paying a penalty for sharing resources:
  - COHERENCY and/or SOFTWARE OVERHEAD
- **Decouple memory from cores aggregation**
  - Many shared-memory parallel applications do not scale beyond a few tens of cores...
  - ...but may benefit from large amounts of memory
    - In-memory databases, datamining, VM, scientific applications, etc
Introduction

• In order to provide more memory, Violin memories is able to provide up to 504 GB (with high cost and latency)

• Memory in the nodes of current clusters is overscaled in order to fit the requirements of “any” application

• It remains unused most of the time

• Unleash your memory-constrained application by using the memory in the rest of nodes

• Without software layers

• Without useless coherency overhead
Index

Introduction

A new shared-memory architecture

Implementation

Results

Conclusions & future work
A new shared-memory architecture

We propose to share memory across the cluster but not to share cores neither caches.

The overhead of the coherency protocol among nodes is avoided.
A new shared-memory architecture

**RESTRICTION**
A given memory region can only be accessed by processors in a single node so...
...an application can be parallelized up to the number of cores present in a node

**ADVANTAGE**
Our system perfectly scales even with huge amounts of memory (Terabytes)

**ADVANTAGE**
Applications can directly use this system without modifications or recompilations or any kind of library or run-time
A new shared-memory architecture

Example:

- Each node has 5 memory controllers: 1 in each socket plus 1 RMC (remote memory controller)
- Each CPU (and RMC) knows where to forward a memory request regarding to the system memory map
A new shared-memory architecture

- **KEY FACTOR**: there are more bits in the *physical address* than needed → use some to insert information in the address itself
A new shared-memory architecture

• Moreover...

• We have a shared address space

• Every processor can address any memory location, without the need of translation tables

• Actually, every processor can address the same memory location: this leaves the door open to non-coherent shared memory across the cluster
Index

Introduction

A new shared-memory architecture

Implementation

Results

Conclusions & future work
Implementation
Index

Introduction

A new shared-memory architecture

Implementation

Results

Conclusions & future work
Results

• Analyzing the system bottleneck: **local RMC**

• Analyzing the system bottleneck: **remote RMC**
Results

• Data retrieval operation (b-tree)

• Remote swap: \[ T_{remote\_swap} = A_{total} \times L_{local} + \frac{A_{total}}{A_{page}} \times L_{swap} \] (1)

• Our proposal: \[ T_{remote\_memory} = A_{total} \times L_{remote} \] (2)

Studying the optimum number of children:
Results

• Remote swap vs our proposal
Results

• Testing the prototype with PARSEC
Index

Introduction

A new shared-memory architecture

Implementation

Results

Conclusions & future work
(A lot of) Future work

• Design prefetch techniques
• Analyze real applications (databases)
• Study memory allocation policies
• Study network topologies
• Explore the inter-process inter-node communication possibilities
Conclusions

• We presented a low-cost approach to memory aggregation in cluster
  • Low latency
  • High scalability

• A given application can only be executed in the cores present in a single motherboard but can use the entire memory in the cluster

• Execute memory–hungry applications where it was not possible...
  • ...without undergoing the “useless” coherency overhead

• No need for memory overscaling $\rightarrow$ energy saving
Thank you.

September 21, 2010