Dynamic Memory Management on Mome DSM

Yvon Jégou
IRISA/INRIA, FRANCE

- Introduction
- Goals
- Basic implementation
- Current work

Why ?

- Few DSM implementations provide a global shared memory management
- Must be provided by applications

- Problem:
  - portability of sequential codes (libraries)
  - needed for OpenMP

OpenMP on clusters

- Everything is implicitly shared
- Stacks are shared
- Dynamically allocated memory is potentially shared

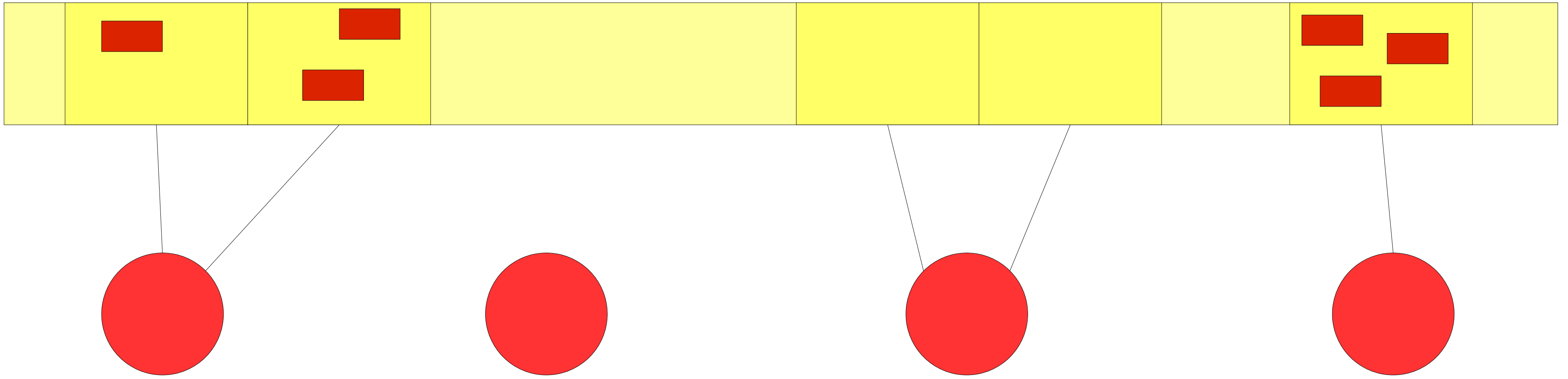
- Load balancing through worker migration: private data need to be shared

Key goals

- No penalty for private (local) memory management
- Symmetry
- Balanced / Unbalanced loads
- Scalability (hundreds of nodes)
- Efficiency

Basic implementation

- Two levels
  - Top level: shared management of large blocks
  - Low level: local management of small blocks



Top level

- Global management protected by global mutex lock
- Top-level metadata in a shared DSM segment
- Current 32 bit implementation: blocks >4Mbytes handled at top level

Low level

- Arena: a list of top-level blocks (heaps)
- Memory allocation inside arenas
  - *glibc malloc/free in our implementation*
- Each arena managed by a single node
  - *can have multiple arenas/node to reduce contention inside the node*

- All consistency models

Arena/heap creation

- Initialization: empty arena list
- First malloc:
  - *request heap from top level*
  - *create first arena in heap*
- On malloc, if free space exhausted in arena
  - *request extra heaps from top level and extend arena*
- Contention on access to arena (SMP)
  - *switch to another arena (if possible), or*
  - *create a new arena and switch allocations to this arena*

Symmetry: free

- free can be requested from all nodes
- **free (addr)**
  - **addr** belongs to a local arena: handled locally
  - **addr** is not local: send **addr** to its manager node

- We need efficient handling of block ownership

Lock-free implementation of ownership

- Heap: fixed size 2^h
- Heap addresses: (**ha**) aligned on 2^h boundary
- Heap Id: **ha>>h.** All addresses from same heap have same Id.

- Ownership vector
  - `owner[Id]==valid node number` node management
  - `owner[Id]==GLOBAL` global management
  - `owner vector` located in DSM shared space
- updated during heap allocation (and deallocation): atomic
- read during `free` request: atomic

Efficiency considerations

- Need for global lock: reduced to big block and heap management
- Symmetrical management
- Efficient private memory management

- Efficiency measure: #page faults during memory management

Page faults

- Top level (32 bits implementation):
  - 256 big blocks (max): top-level metadata in one single DSM page
  - **`owner`** vector: one DSM page
  - big block alloc/free: one page-fault (max)
  - heap alloc/free: two page-faults (max), more expensive
- Low-level
  - heap allocation (not frequent)
  - `free` operation: ownership test (few page faults)
  - false-sharing between glibc metadata and user data (frequent on small blocks)

Global performance

- Highly dependent on metadata/data false sharing
- OpenMP on HPC numerical codes: performance is OK
- High stress (frequent small malloc/free + data sharing): performance limited by false-sharing.

- False-sharing reduction
  - highly dependent on the DSM
  - current work
  - separate metadata from data ?

Current work

- On the DSM
  - move to full 64 bits support
  - support for hundreds of nodes (hierarchical)
  - improve support for
    - multiple memory consistency models
    - multiple views of shared space

Current work on the memory allocator

- Use multiple views of the shared space
  - metadata and data in different views of the shared space
- Consistency of metadata view: (very) weak
  - modifying metadata does not invalidate the page on other nodes
  - modifying user data does not invalidate metadata view

Conclusion

- Still a lot of work...