ArgoDSM Enhancements for Task-Based Systems

UPPSALA UNIVERSITET

per.ekemark@it.uu.se

Per Ekemark, Sven Lundgren, David Klaftenegger, Konstantinos Sagonas, Stefanos Kaxiras

1 Introduction

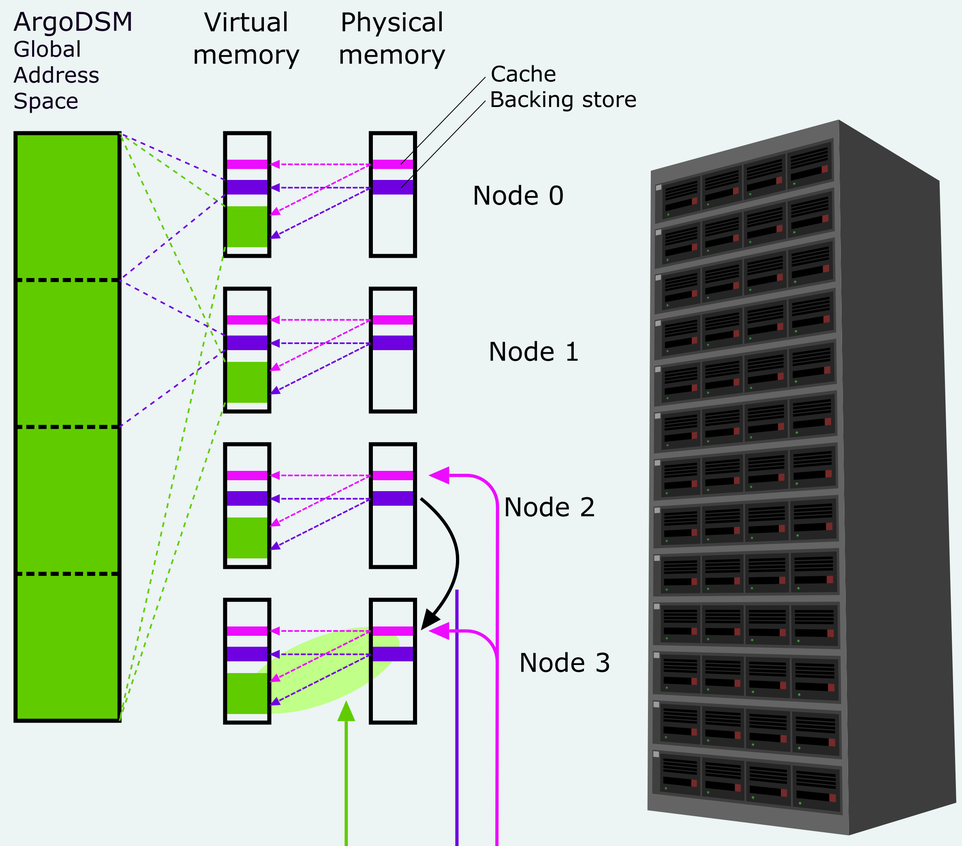
ArgoDSM is a distributed shared memory system that implements a global address space (GAS) through caching and a coherence protocol. Programs running on several nodes in a distributed system can allocate memory in ArgoDSM's global address space and use pointers into the allocated memory transparently from any node in the system. ArgoDSM handles transfers and caching between nodes. The cache coherency implements a form of release consistency but supports sequential consistency for data-race-free programs.

OmpSs-2 (developed at BSC) is a task-based programming model using data-flow annotations to schedule tasks. Recent versions of the model's runtime system, Nanos6, can schedule tasks on different nodes in a distributed system. ArgoDSM is being integrated as a memory backend to facilitate seamless use of global pointers between distributed nodes.

2 ArgoDSM for Task-Based Systems

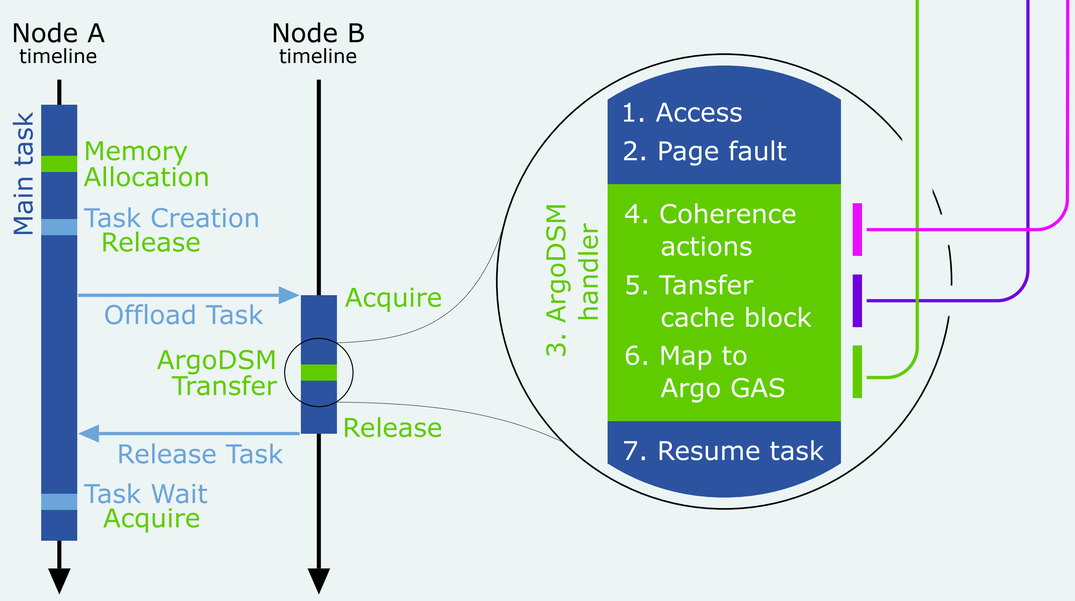
Memory layout

- Programs reference locations in the ArgoDSM GAS, allocated at the same virtual address range on every node.
- Each node provides a backing store for a part of the GAS.
- Each node has a cache to store data belonging to other nodes.
- Memory in use is mapped from the cache or backing store into the ArgoDSM GAS.



Task-based runtime

- The main task of a task-based program will run on a single node.
- Memory should be allocated with ArgoDSM to make sure it is available anywhere in the distributed system.
- On task creation, changes in memory are published by the runtime system using ArgoDSM's release operation.
- Some tasks are offloaded to other nodes. To make sure the latest values are used, an acquire operation is performed before the task starts.
- ArgoDSM handles any remote accesses.
- When an offloaded task finishes, a release operation publishes changes in memory.
- After a parent task has waited for children to finish, an acquire operation is used to make sure only updated data is used in the future.



Cache misses

The first time a location in the ArgoDSM GAS is (1) accessed, a (2) page fault will occur, triggering the (3) ArgoDSM page fault handler.

The handler performs three main actions: (4) updates the cache meta-data according to the VIPS coherence protocol on the requesting node and the requested location's home node, (5) transfers the requested cache block between nodes, and (6) maps the block to the ArgoDSM GAS.

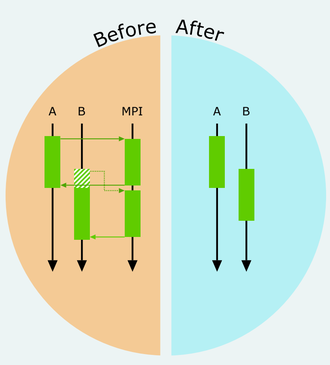
The program can now access the data at the requested location and (7) resume execution.

3 ArgoDSM Enhancements

Serialisation elimination: Communication thread

Before: Dedicated communication (MPI) thread causes stalls when ArgoDSM operations overlaps in different program threads.

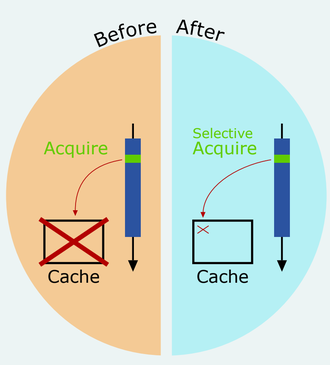
After: Program threads handle communication directly.



Selective coherence operations

Before: Acquire operations clears the entire cache.

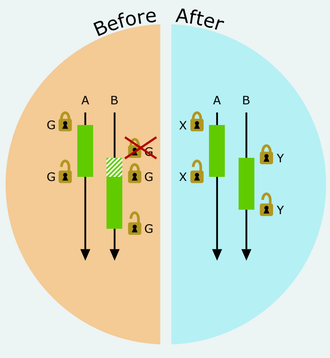
After: With information from task directives, only relevant parts of the cache are cleared by selective acquire.



Serialisation elimination: Global lock [WIP]

Before: Cache operations protected by a single global lock, prevents multiple simultaneous ArgoDSM operations.

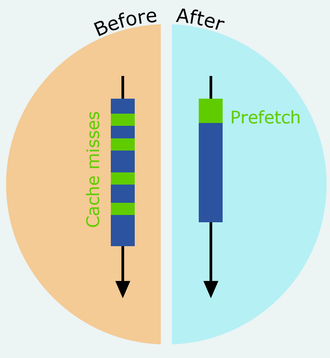
After: Different cache regions are protected by different locks. Enables parallelism between unrelated accesses.



Prefetching [Future]

Before: Miss on many occasions, requiring multiple handler calls.

After: Using hints from task directives, or from other sources, data can be fetched early, eliminating some miss overhead.



4 Enhancement Impact

Serialisation elimination

- Less contention between threads
- Up to 90 % of time spent waiting
- **Early results: 1.5x - 2x speedup**

Selective coherence

- Less interference between threads
- Relies on task information
- **Early results: 1.5x speedup**

Prefetching

- Avoid page fault overhead
- Relies on additional information
- Policy examples
  - Demanded by task
  - Invalidated by acquire
  - Profiled likely use

The EPEEC project has received funding from the European Union's Horizon 2020 research and innovation programme under the grant agreement Nº 801051.