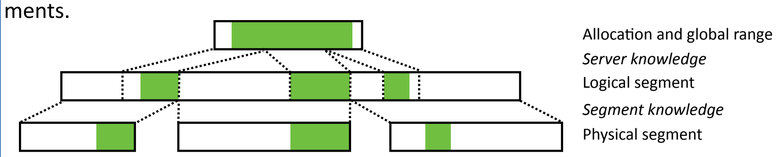
Directory/Cache API for Sharing Data in Distributed Memory Systems

Tiberiu Rotaru[1], Bernd Lörwald[1], Mirko Rahn[1], Nick Brown[2], Vicenç Beltran Querol[3], Olivier Aumage[4], Xavier Teruel[3]

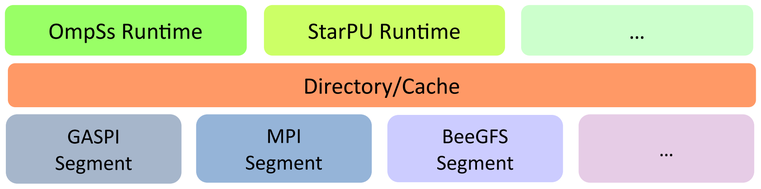
Abstract Data Representation

The end users only work with data abstractions in terms of local and global ranges, safely assuming that the data with which they interact is contiguous and linearly addressable. Local ranges are contiguous regions of memory allocated in local caches. Global ranges are linear representations of global data distributed in segments.



Directory/Cache API

The directory/cache allows task-based runtime systems to efficiently run distributed applications, while being able to consistently manage data stored in distributed memory and in local caches. The directory/cache API allows runtimes to be completely independent from the physical representation of data and from the type of storage used, facilitating the access through the same interface to an extendable list of memory segment implementations. Moreover, applications may also use the directory/cache API directly.



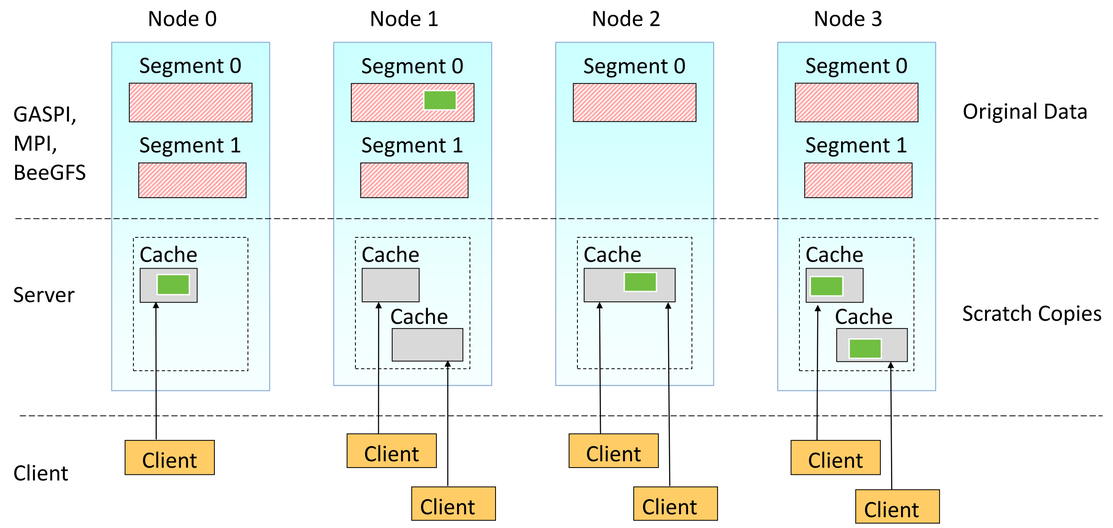
The API provides methods for *creating* and *deleting* segments, for *creating* and *deleting* local caches and for *allocating* or *deallocating* memory for global data in segments. The API allows performing operations with local and global ranges, such as: *allocating* memory in caches, *releasing* data stored in caches, *retrieving* read-only or modifiable copies of data from the global memory into a local cache and *transferring* data from a local cache into the global memory. The API also provides other operations with enriched semantics, such as *tagged* data transfers and operations that atomically combine data transfers with cache releases. The operations with ranges are asynchronous and facilitate keeping the cache coherence and the global consistency of data, performing in background additional bookkeeping operations related to managing data copies and their associated reference counters. For optimization purposes, the trusted clients may be granted direct access to the segment memory when this is possible. The API provides *transfer costs* associated with a list of operations and *data locality* information.

www.intertwine-project.eu, @intertwine_eu. Project funded by the European Commission. Grant agreement number: 671602.

[1]Fraunhofer ITWM, [2]EPCC, [3]Barcelona Supercomputing Center, [4]INRIA

See us at booth #J-640

Directory/Cache Architecture



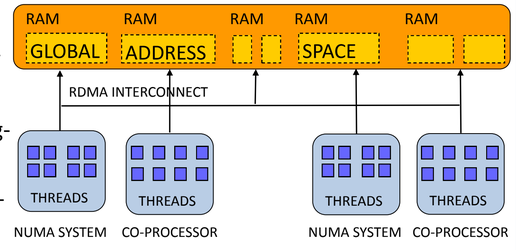
The directory/cache relies on a client-server architecture. The original data is stored in one or more segments across several nodes and copies of global memory regions are stored in local caches. A local server may create and manage multiple local caches. Multiple clients may connect to a local server and each client may attach to specific local caches that may be shared between numerous clients. The clients can be either started within the same process as the corresponding local server or in a distinct process. The second approach offers the advantage of supporting the coupling of external programs with an already running directory/cache service. Moreover, running clients in different processes to the server makes the directory/cache tolerant to client failures. The local servers coordinate with each other for carrying out operations with memory ranges in a consistent way. This architecture is flexible and extensible, allowing adding in a straightforward manner new segment implementations and specific extensions to the client interface. Primary clients for the directory/cache are considered the task-based runtime systems, but applications may also directly use it.

GASPI and MPI Segments

A segment is a continuous space of globally accessible memory which can be addressed in terms of offset of data.

Global **A**ddress **S**pace **P**rogramming **I**nterface (**GASPI**) is a Partitioned Global Address Space API. **GPI-2** is an implementation of the **GASPI** standard. **MPI** is a standardized message-passing system for distributed-memory applications used in parallel computing.

Global address space Programming Interface (GPI-2)



The directory/cache is designed to work with an extendable list of segment types (GASPI, MPI, etc.) that can be used simultaneously. The API provides a segment interface that all segment types should implement: low-level get and put methods for transferring data from/into segments using the corresponding communication library (GPI-2, MPI library, etc), methods for getting the list of transfer costs, the list of home nodes of a list of memory ranges and optionally, for getting direct access to the segment memory.

OmpSs

OmpSs is a task-based programming model that aims to provide portability and flexibility for sequential codes while the performance is achieved by the dynamic exploitation of the parallelism at task level. OmpSs targets the programming of heterogeneous and multi-core architectures and offers asynchronous parallelism in the execution of tasks. The runtime is able to schedule and run these tasks, taking care of the required data transfers and synchronizations. OmpSs is a promising programming model for future exascale systems, with the potential to exploit unprecedented amounts of parallelism while coping with memory latency, network latency and load imbalance.

StarPU

The StarPU Runtime System provides a framework for task scheduling on heterogeneous platforms. It is capable to drive task-based applications over heterogeneous cluster nodes equipped with nVidia CUDA devices, Intel Xeon Phi devices and/or OpenCL accelerators. StarPU keeps track of data replicates on discrete accelerators and disposes of a data prefetching engine to overlap transfers with computation. StarPU uses self-tuned per-task performance models for improving the scheduling quality of parallel linear algebra algorithms. Instead of rewriting the entire code, programmers encapsulate existing kernels within codelets objects. Codelet may be enriched with additional kernel implementations for each supported architecture. StarPU schedules the codelets as efficiently as possible over the entire machine, dynamically selecting adequate implementations as execution unfolds. In order to relieve programmers from the burden of explicit data transfers, a high-level data management library enforces memory coherency over the machine.