https://www.argo-osr.org/handouts/

Memory Management

ANL: Nicolas Denoyelle, Swann Perarnau, Brice Videau
LLNL: Maya Gokhale, Marty McFadden, Roger Pearce, Eric Green, Keita Iwabuchi
UCMerced and LLNL: Kai Wu, Dong Li

Argo Project

The goal of Argo is to augment and optimize existing OS/R components for use in production HPC systems, providing portable, open source, integrated software that improves the performance and scalability of and that offers increased functionality to exascale applications and runtime systems.

For an overview of Argo, please refer to `https://www.argo-osr.org/`.

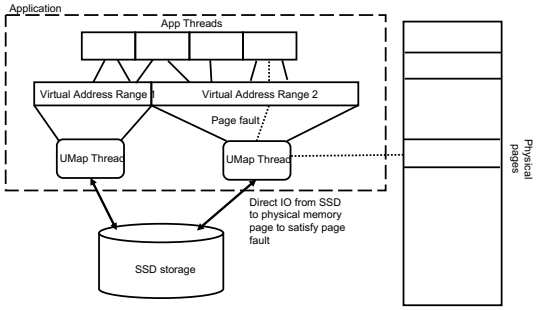
Memory Management Overview

We are developing software techniques to enable applications and runtime systems to make the best use of the capabilities of complex new memory types being incorporated into exascale node designs.

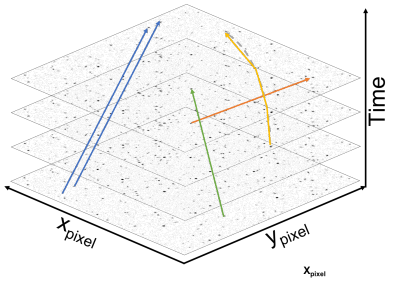
We are currently pursuing two complementary strategies: a *transparent DRAM cache for NVRAM devices* (UMap) and a *managed scratchpad for high-bandwidth memory* (AML).

UMap

User Level Memory Map (UMap) is a memory map approach to incorporating NVRAM into complex memory hierarchy. It replaces `mmap` for out-of-core data. By exploiting large virtual address spaces, data sets can be accessed directly as if in memory; this is particularly well suited to analyzing large observational and simulation data. Thanks to its user-space design, UMap is flexible and easy to use.



Migration of pages between far and near memory is the responsibility of the UMap handler. It uses the low-overhead `userfaultfd` protocol, enabling applications to handle page faults in user space. The UMap handler receives a page fault notification as a message and loads the requested page. Since it runs in the application context, the handler can be specific to an application(-class). The page size (multiple of system page) and the page buffer size are customizable, and the data can be loaded from multiple files. Support for loading from distributed data sets—over the network—is under development.



The figure on the right illustrates an application that uses UMap. The application takes imaging data sets of the sky generated by an optical camera and uses them to detect Near-Earth Asteroids (NEAs). Our transient detection algorithm computes medians along likely NEA trajectories to track asteroid motion in imagery over time. UMap combines many 2-D tiled images to form a 3-D volume for the search algorithm.

Another class of applications we studied is out-of-core graph algorithms that generate an edge or adjacency list representation of graphs from raw data (e.g., Wikimedia Commons edits) and traverse the graphs to compute features of interest for community detection.

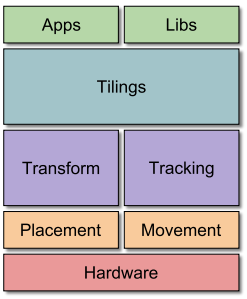
U.S. DEPARTMENT OF ENERGY | Office of Science
National Nuclear Security Administration
Argonne NATIONAL LABORATORY
Lawrence Livermore National Laboratory

AML

AML provides explicit, application-aware memory management for deep memory systems. It offers a collection of building blocks that are *generic*, *customizable*, and *composable*. Applications can specialize the implementation of each offered abstraction and can mix and match the components as needed. AML can be used to create, for example, a software-managed scratchpad for multilevel DRAM hierarchy such as HBM and DDR. Such a scratchpad provides applications with a memory region with a predictable high performance for critical data structures.

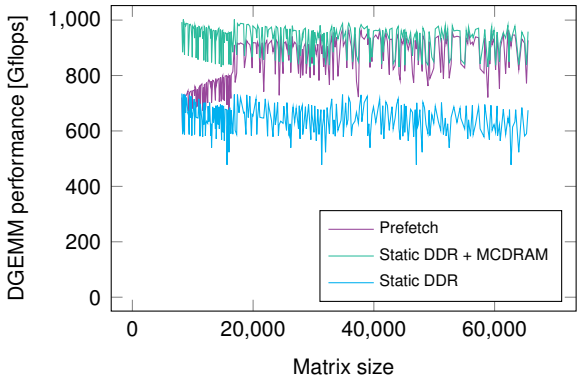
We provide applications and runtimes with a descriptive API for data access, where all data placement decisions are explicit, and so is the movement of data between different memory types. At the same time, the API does abstract the memory topology and other device complexities. We focus on optimizing data locality for current and future hardware generations; applications provide insights for static allocations, and we can also dynamically and asynchronously remap to optimize for a particular data layout or to best take advantage of the memory hierarchy.

The figure on the right depicts the major components of AML, which include the following:



- Topology & hardware management (dependent on NUMA, hwloc, and SICM)
- Data layout descriptions (application-specific)
- Tiling schemes (including ghost areas)
- Data movement facilities (currently primarily `memcpy` or `move_pages`)
- Pipelining helpers (scratchpad, asynchronous requests)

We conducted an experiment on an Intel Knights Landing system running in flat/quad mode to verify the effectiveness of the AML prefetching using a custom DGEMM implementation, which uses pipelining to move data between DDR and MCDRAM, OpenMP tasks, and an inner kernel autotuned for AVX2. We compared prefetching against two static allocation approaches: one where all the data is in the regular DDR memory and one where it is optimally distributed between DDR and MCDRAM. As shown in the figure on the right, the prefetcher achieves performance equal to or better than that of the optimal static allocation on inputs that exceed the capacity of MCDRAM.



Status

Our software is open source. UMap can be found at `https://github.com/LLNL/umap`, with sample application code at `https://github.com/LLNL/umap-apps`; it also runs on LLNL Sierra compute nodes. AML can be found at `https://xgitlab.cels.anl.gov/argo/aml`. We are always looking for additional collaborators with more workloads. If you think that your project could benefit from either UMap or AML, we encourage you to get in touch with us.

Contact

For more information, please contact Maya Gokhale `maya@llnl.gov` or Swann Perarnau `swann@anl.gov`.