Runtime Optimization through Explicit Memory Management

Nicolas Denoyelle
ndenoyelle@anl.gov

Argonne National Laboratory

April 17, 2019

Node level memory and locality management.

How do you manage memory/locality in your application/library?

- malloc() / free() / jemalloc?
- mmap() / munmap() ?
- memkind / libnuma / hwloc
- Do you bind memory and threads with a policy / openmp pragmas?
- Do you move data manually? How?
- Do you move data across devices?

Does any of these have portable performance?

AML Memory Library

For people interested in managing memory.

Operating System

Runtime

AML Memory Management Library

Abstractions for managing shared memory:

- Data (access) representation
- Data movement

Scientific Library

Application

AML Memory Library

Operating System

Runtime

(R) runtime optimizations with and inside AML building blocks.

AML Memory Management Library

Abstractions for managing shared memory:

- Data (access) representation
- Data movement

Scientific Library

(S), Scientific library highly optimized with AML.

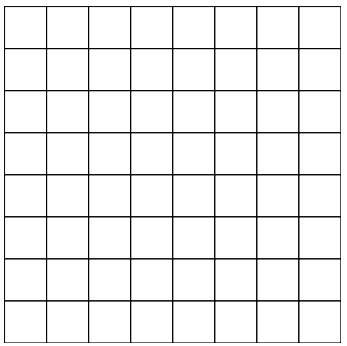
Application

(A) Efficient applications with a limited impact on the code.

User Data and Micro-Architecture Optimizations

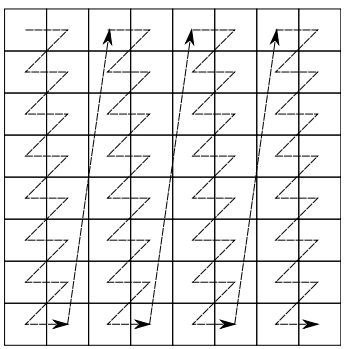
User objects

- Representation of user data (*e.g.* a matrices).
- Set of operations on user data (*e.g.* dgemm, dpotrf, dtrsm *etc*...).

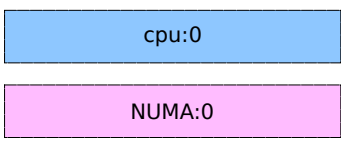


Inner (*e.g.* blas) Library Objects

Micro-Architecture optimized representation and operations: coalescing, vectorization, cache blocking



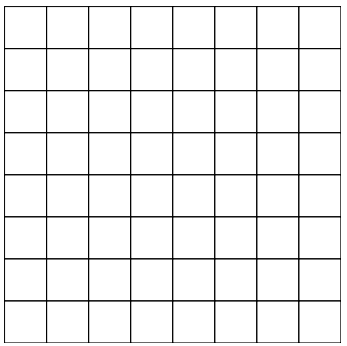
Performance portability $\Longrightarrow$ abstraction of data representation.



User Data and Locality Optimizations

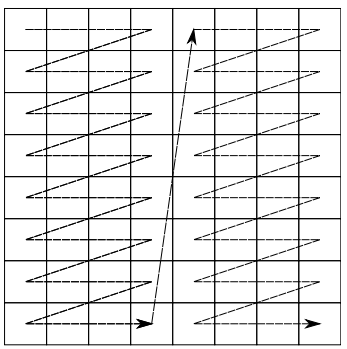
User objects

- Representation of user data (*e.g.* a matrices).
- Set of operations on user data (*e.g.* dgemm, dpotrf, dtrsm *etc*...).

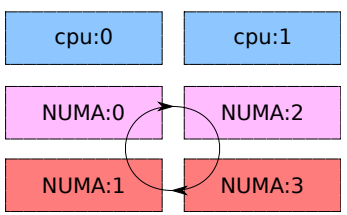


Inner (*e.g.* blas) Library Objects

Locality optimizations: NUMA locality/migrations, prefetching



Performance portability $\Longrightarrow$ abstraction of data movements.



AML Library for Memory Management

Portable optimized code requires:

- Translation between user and hardware data representation.
- Data migrations between memories/devices.

AML is a toolbox for easing memory management. AML does:

- Simplify the burden of indexing data, *i.e.* data representation.
- Abstract memory locations, allocators, and migrations,
- Implement on the fly translation between data representations when moving data.

Through the following abstractions:

- Iterator abstraction (dense structures, trees, indexes, composition *etc*...),
- Specialized Allocators (NUMA, HBM, NVRAM, OpenCL devices, Cuda devices),
- Migration engine and scractchpad,

Walkthrough with AML Building Blocks

Allocators and Memory Locations

1. (A) creates data on host processor.
5. (S) or AML migrations engine, allocates destination buffer when moving data.

Iterators for Data Organization

2. (A) describes its data layout, *e.g.* block cyclic matrix, and calls optimized library.
3. (S) describes hardware optimized layout and implements user operations on such a layout.

Reshaping on the Fly Through Migrations

4. When optimizing locality, (S) uses AML blocks for automatically performing transforms.
6. Iterators embed next access intents $\Longrightarrow$ (R) possibility to pack, prefetch *etc*. . . data.

State of AML

Library Quality

- Autotools build and testing,
- Full continuous integration pipeline,
- Spack packaging (Ongoing),
- man, html, pdf documentation (Ongoing).

Proofs of concept

- Software prefetch for matrix multiplication.
- Qualitative memory allocation for latency sensitive applications (Ongoing).

To be officially released soon.
Currently available under BSD-3 license on ANL gitlab:
https://xgitlab.cels.anl.gov/argo/aml,