Recent Experiences in Using MPI-3 RMA in DART

Workshop on PGAS programming models:
Experiences and Implementations (PGAS-EI'18)

Joseph Schuchart
Roger Kowalewski
Karl Fuehrlinger

January 31, 2018

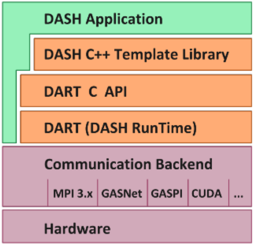
Introduction to the DASH RunTime

DASH:

- C++11/14 PGAS abstraction following STL concepts: iterators + operators
- Static and dynamic distributed containers

Container	Description	Data distribution
Array<T>	1D Array	static, configurable
NArray<T, N>	N-dim. Array	static, configurable
Shared<T>	Shared scalar	fixed, configurable
Directory*<T>	Variable-size, locally indexed Array	manual
CoArray*<T>	Similar to CAF	uniform

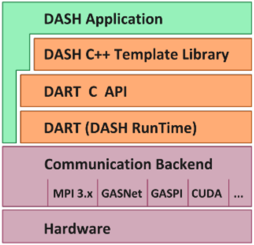
(*) Under construction



Introduction to the DASH RunTime

DASH:

- C++11/14 PGAS abstraction following STL concepts: iterators + operators
- Static and dynamic distributed containers
- Distributed algorithms: `find`, `max_element`, ...
- Local and global view on data
- Any trivially copyable type as elements
- Flexible data distribution patterns



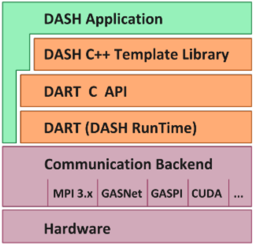
Introduction to the DASH RunTime

DASH:

- C++11/14 PGAS abstraction following STL concepts: iterators + operators
- Static and dynamic distributed containers
- Distributed algorithms: `find`, `max_element`, ...
- Local and global view on data
- Any trivially copyable type as elements
- Flexible data distribution patterns

DART:

- C11 runtime for DASH
- Communication abstraction
- Workhorse implementation: MPI-3 RMA



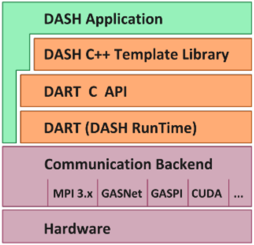
Introduction to the DASH RunTime

DASH:

- C++11/14 PGAS abstraction following STL concepts: iterators + operators
- Static and dynamic distributed containers
- Distributed algorithms: `find`, `max_element`, ...
- Local and global view on data
- Any trivially copyable type as elements
- Flexible data distribution patterns

DART:

- C11 runtime for DASH
- Communication abstraction
- Workhorse implementation: MPI-3 RMA
  - Easy transition from existing parallel codes
  - Available on (nearly) all systems



Example: DASH-DART-MPI

```
dash::Array<int> array(N);
// initialize array
// better: dash::generate()
if (dash::myid() == 0) {
  for (int i = 0; i < N; ++i) {
    array.async[i] = i;
  }
}
array.barrier();


if (dash::myid() == 1)
  std::cout << array[0];
```

DASH

```
dart_team_memalloc_aligned();


dart_put_blocking_local();


dart_flush_all();
dart_barrier();


dart_get_blocking();
```

DART

```
MPI_Win_allocate_shared();
MPI_Win_attach();
MPI_Allgather();


MPI_Rput();
MPI_Wait();

MPI_Win_flush_all();
MPI_Barrier();


MPI_Rget();
MPI_Wait();
```

MPI

MPI-3 Aspects and Features

- Process groups and collectives
- *Thread-safety*
- *Asynchronous Progress*
- *Communication Primitives*
- *Global Memory Allocation*

Thread-safety

DASH/DART functionality generally *thread-safe*
⇝ Usable with common threading abstractions (e.g., OpenMP)

```
void compute(dash::Array<double>& array)
#pragma omp parallel for schedule(dynamic)
  for (int i = 0; i < array.size(); ++i) {
    array.async[i] = f(i);
  }
  array.flush();
}
```

Thread-safety

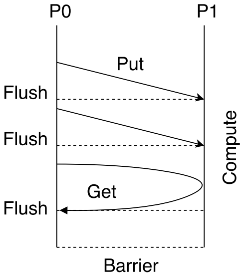
DASH/DART functionality generally *thread-safe*
$\rightsquigarrow$ Usable with common threading abstractions (e.g., OpenMP)

Some limitations apply:

- Unsynchronized data access in global memory
  - Alternative: `dash::Array< dash::Atomic<T> >`
- Collective operation on same team
  - Reductions/synchronization
  - Team management
  - *Global memory allocation*

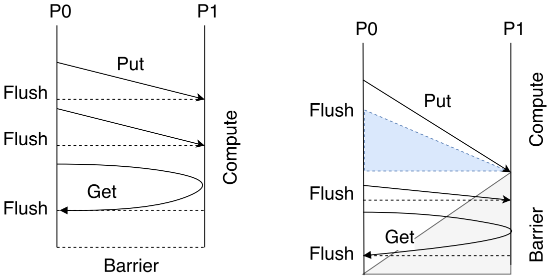
(Asynchronous) Progress

- MPI one-sided can come in two flavors:
  1. Progress happens **without** involvement of the remote process



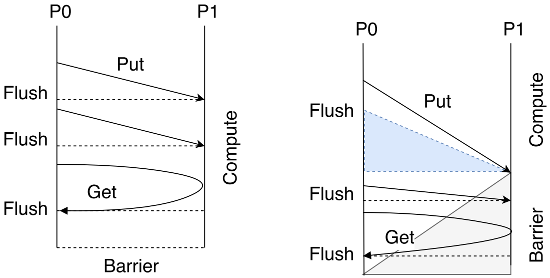
(Asynchronous) Progress

- MPI one-sided can come in two flavors:
  1. Progress happens **without** involvement of the remote process
  2. Progress happens **with** involvement of the remote process



(Asynchronous) Progress

- MPI one-sided can come in two flavors:
  1. Progress happens **without** involvement of the remote process
  2. Progress happens **with** involvement of the remote process



- Does progress happen in the background? We don't know!

(Asynchronous) Progress

- MPI one-sided can come in two flavors:
  1. Progress happens **without** involvement of the remote process
  2. Progress happens **with** involvement of the remote process

- MPI standard is vague:

  > *[...] implementations must guarantee that a process makes progress on all enabled communications it participates in,* ***while blocked on an MPI call****.*

(Asynchronous) Progress

- MPI one-sided can come in two flavors:
  1. Progress happens **without** involvement of the remote process
  2. Progress happens **with** involvement of the remote process

- MPI standard is vague:

  > *[...] implementations must guarantee that a process makes progress on all enabled communications it participates in,* ***while blocked on an MPI call****.*

- Example:
  local polling ≈ blocked?

```
while (!flag) {
  MPI_Get(&flag, mype, win);
  MPI_Flush_local(mype, wine);
}
```

(Asynchronous) Progress

- MPI one-sided can come in two flavors:
  1. Progress happens **without** involvement of the remote process
  2. Progress happens **with** involvement of the remote process

- MPI standard is vague:

  > *[...] implementations must guarantee that a process makes progress on all enabled communications it participates in, **while blocked on an MPI call**.*

- Example:
  local polling $\approx$ *blocked*?

```
while (!flag) {
  MPI_Get(&flag, mype, win);
  MPI_Flush_local(mype, wine);
}
```

- MPI interfaces for triggering progress engine and querying progress semantics?

DART Communication Primitives

- Relies on *passive target mode* (`MPI_Win_lock_all()`)
- Extended Put/Get interface:
  - `dart_get`: non-blocking, requires `dart_flush[_local]`
  - `dart_get_blocking`: remote completion
  - `dart_get_blocking_local`: local completion

DART Communication Primitives

- Relies on *passive target mode* (`MPI_Win_lock_all()`)
- Extended Put/Get interface:
  - `dart_get`: non-blocking, requires `dart_flush[_local]`
  - `dart_get_blocking`: remote completion
  - `dart_get_blocking_local`: local completion
- Strided/indexed access:
  - `dart_create_type_strided` $\rightsquigarrow$ `MPI_Type_vector`
  - `dart_create_type_indexed` $\rightsquigarrow$ `MPI_Type_indexed`

DART Communication Primitives

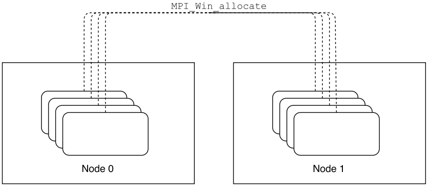
- Relies on *passive target mode* (`MPI_Win_lock_all()`)
- Extended Put/Get interface:
  - `dart_get`: non-blocking, requires `dart_flush[_local]`
  - `dart_get_blocking`: remote completion
  - `dart_get_blocking_local`: local completion
- Strided/indexed access:
  - `dart_create_type_strided` $\rightsquigarrow$ `MPI_Type_vector`
  - `dart_create_type_indexed` $\rightsquigarrow$ `MPI_Type_indexed`
- DART uses `size_t`
  - Transparently chunk up large transfers ($> 2^{31}$ bytes)
  - Preallocate types using `MPI_Type_contiguous`

DART Communication Primitives

- Relies on *passive target mode* (`MPI_Win_lock_all()`)
- Extended Put/Get interface:
  - `dart_get`: non-blocking, requires `dart_flush[_local]`
  - `dart_get_blocking`: remote completion
  - `dart_get_blocking_local`: local completion
- Strided/indexed access:
  - `dart_create_type_strided` $\rightsquigarrow$ `MPI_Type_vector`
  - `dart_create_type_indexed` $\rightsquigarrow$ `MPI_Type_indexed`
- DART uses `size_t`
  - Transparently chunk up large transfers ($> 2^{31}$ bytes)
  - Preallocate types using `MPI_Type_contiguous`
- No implicit ordering guarantees in non-blocking operations

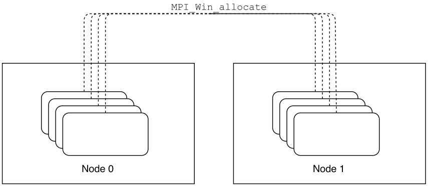
Global Memory Allocation

Win_allocate

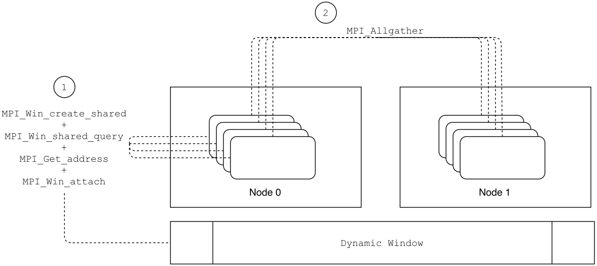


Global Memory Allocation

Win_allocate



Win_dynamic



Notes On Global Memory Allocation

- No control over local memory alignment
  - Natural alignment seems guaranteed

Notes On Global Memory Allocation

- No control over local memory alignment
  - Natural alignment seems guaranteed
- Shared memory system configuration: size of `/dev/shm` and `/tmp`

Notes On Global Memory Allocation

- No control over local memory alignment
  - Natural alignment seems guaranteed
- Shared memory system configuration: size of `/dev/shm` and `/tmp`
- Temporary global allocations used in DASH algorithm
  - Most notably: `accumulate` and `[min,max]_element`
  - Required for custom operators and `ValueType`
  - Cannot always use MPI reduction/collective operations

Notes On Global Memory Allocation

- No control over local memory alignment
  - Natural alignment seems guaranteed
- Shared memory system configuration: size of `/dev/shm` and `/tmp`
- Temporary global allocations used in DASH algorithm
  - Most notably: `accumulate` and `[min,max]_element`
  - Required for custom operators and `ValueType`
  - Cannot always use MPI reduction/collective operations
- What are the performance characteristics?

Global Memory Allocation: Measurements

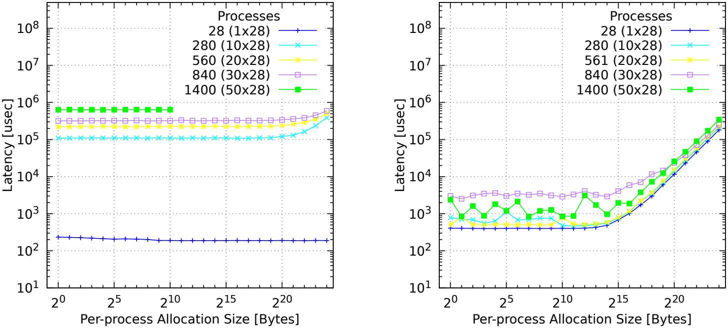
Systems under test:

- Cray XC40 *Hazel Hen*: CCE 8.5.3
- IBM *SuperMUC*: iDataPlex IBM POE 1.4
- IB Linux Cluster: Open MPI 2.0.2

Extension of the OSU benchmark suite

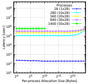
- Allocation latencies
- Communication latencies
- Code available at
  `https://github.com/dash-project/dash-bench`

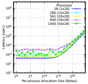
Allocation Latencies: Open MPI

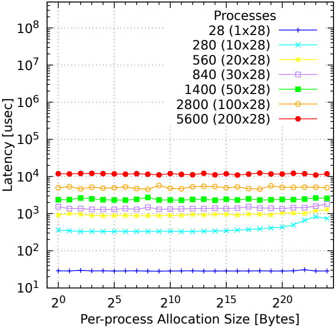


(a) `Win_allocate` (b) `Win_dynamic`

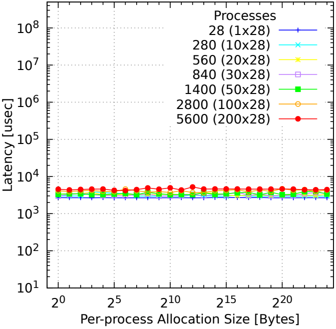
Allocation Latencies: IBM





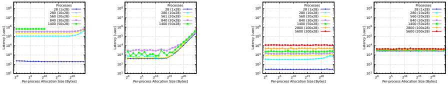


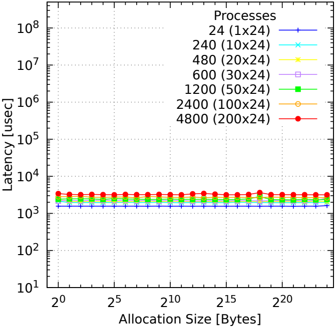
(a) `Win_allocate`



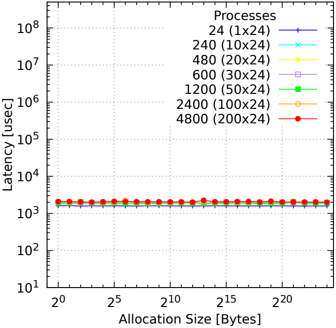
(b) `Win_dynamic`

Allocation Latencies: Cray



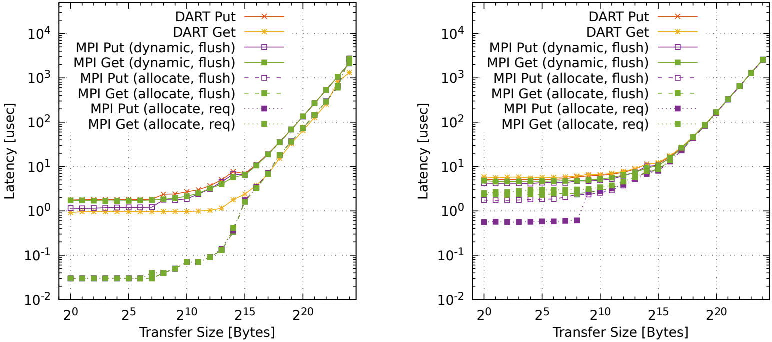


(a) `Win_allocate`



(b) `Win_dynamic`

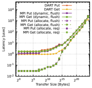
Communication Latencies: Open MPI

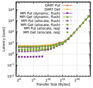


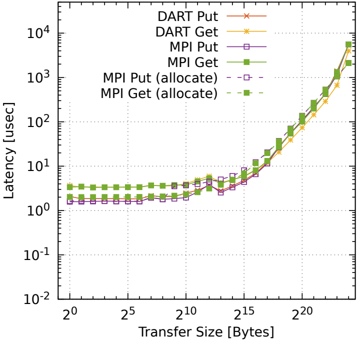
(a) Intra-node

(b) Inter-node

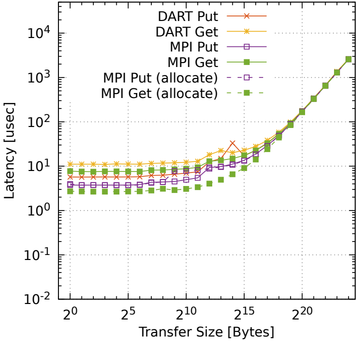
Communication Latencies: IBM





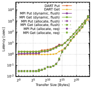


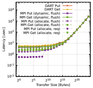
(a) Intra-node

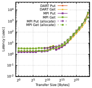


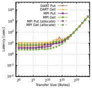
(b) Inter-node

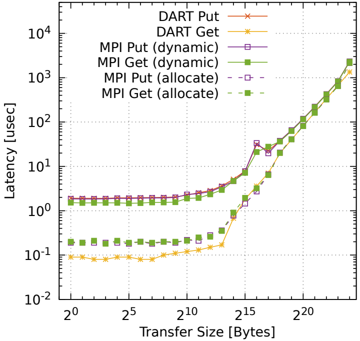
Communication Latencies: Cray



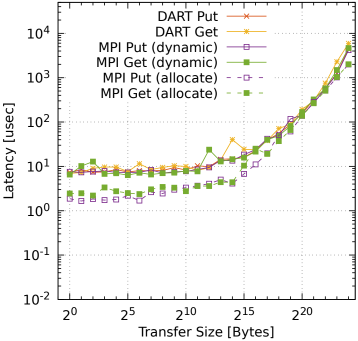






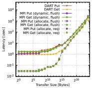
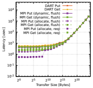
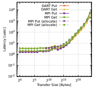
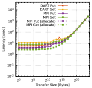
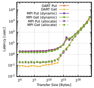
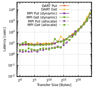


(a) Intra-node



(b) Inter-node

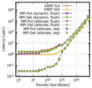
Global Memory Allocation: Summary

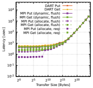







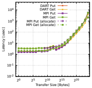
- Heterogeneous latency characteristics
- Generally high allocation latencies: 1 – 600 ms (up to 20 s for 100 GB on IB cluster)
- Shared/Dynamic window allocation potentially faster

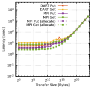
HLRS

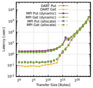
Global Memory Allocation: Summary

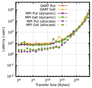






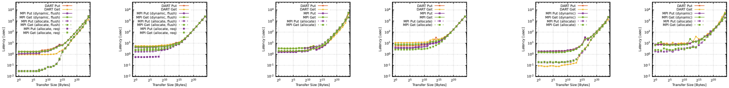






- Heterogeneous latency characteristics
- Generally high allocation latencies: 1 – 600 ms (up to 20 s for 100 GB on IB cluster)
- Shared/Dynamic window allocation potentially faster
- Communication latency higher on dynamic windows
  - Shared memory optimization benefitial in some cases
  - Diminishes in multi-threaded environments

Global Memory Allocation: Summary



- Heterogeneous latency characteristics
- Generally high allocation latencies: 1 – 600 ms (up to 20 s for 100 GB on IB cluster)
- Shared/Dynamic window allocation potentially faster
- Communication latency higher on dynamic windows
  - Shared memory optimization benefitial in some cases
  - Diminishes in multi-threaded environments

$\rightsquigarrow$ Added support for allocated windows (compile-time option)

Conclusions and Future Work

- MPI offers us many important features:
  - team management / collectives
  - global memory allocation
  - communication primitives
- Shared memory optimization vs inter-node communcation latency

Conclusions and Future Work

- MPI offers us many important features:
  - team management / collectives
  - global memory allocation
  - communication primitives
- Shared memory optimization vs inter-node communcation latency
- Important open aspects in MPI:
  - Local alignment of MPI-allocated memory
  - Information on and control of progress
  - Thread-parallel collective operations

Conclusions and Future Work

- MPI offers us many important features:
  - team management / collectives
  - global memory allocation
  - communication primitives
- Shared memory optimization vs inter-node communcation latency
- Important open aspects in MPI:
  - Local alignment of MPI-allocated memory
  - Information on and control of progress
  - Thread-parallel collective operations
- Avoiding temporary global memory allocations (use MPI collectives wherever possible)

There is hope for shared memory...

relax constraints on MPI_WIN_SHARED_QUERY #23

Open jeffhammond opened this issue on Dec 8, 2015 · 3 comments

jeffhammond commented on Dec 8, 2015 Member

Summary

Extend the functionality of `MPI_WIN_SHARED_QUERY` to *all* windows, which will inform the user regarding the MPI shared-memory properties of any window. To what extent this function will return a nontrivial result (i.e. indicate the shared memory has been allocated and is accessible) depends on the implementation. It may be difficult for implementations to use shared memory with `MPI_WIN_CREATE`, although there are multiple existence proofs.

This change *permits* MPI shared-memory accesses on any window, but nothing new is *required*. Implementations will now be allowed to provide more if possible. Previously, if implementations were able to do this, there was no ability for the user to leverage it explicitly.

https://github.com/mpi-forum/mpi-issues/issues/23

Questions?

joseph.schuchart@hlrs.de
github.com/dash-project/
dash-project.org