DiPOSH: a modular testbed and OpenSHMEM implementation

Camille Coti

LIPN, CNRS UMR 7030, Université Paris 13, Sorbonne Paris Cité, France

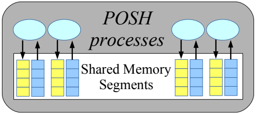
University of Oregon

SC 2019, OpenSHMEM BoF
Denver, Colorado
November 19th, 2019

Roadmap

Implementing OpenSHMEM in POSH

POSH runs multiple OpenSHMEM processes



Shared heap is **symmetric**

- POSH creates a shared memory segment for each process
- Just locate objects at the **same offset** in the shared segments

Communication routines are datatype-specific

- `shmem_int_put`, `shmem_char_put`, `shmem_float_put`..

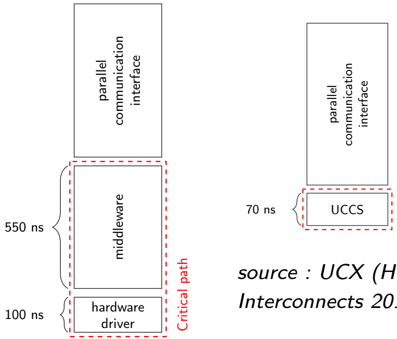
$->$ Use **C++ templates**

- Implement `shmem_<T>_put` and let the compiler do the job

Global static data is symmetric

- In practice :
  - In the BSS segment of the executable if not initialized at compile-time
  - Int the data segment if they are
- Workaround : parse the code and **replace them** by SHMEM allocations just after the initialization of the library

Thin layers



source : UCX (HOT Interconnects 2017)

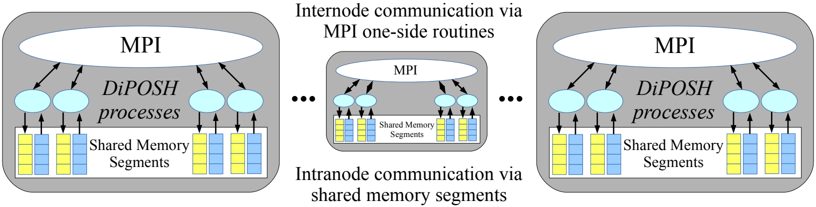
Software overhead

- Achieve low network latency
- Waste no time going through the software stack !

Take advantage of the **simple OpenSHMEM interface**

- Implement data movements in a few instructions
- Avoid additional copies, branches
- ... while being portable

DiPOSH Architecture



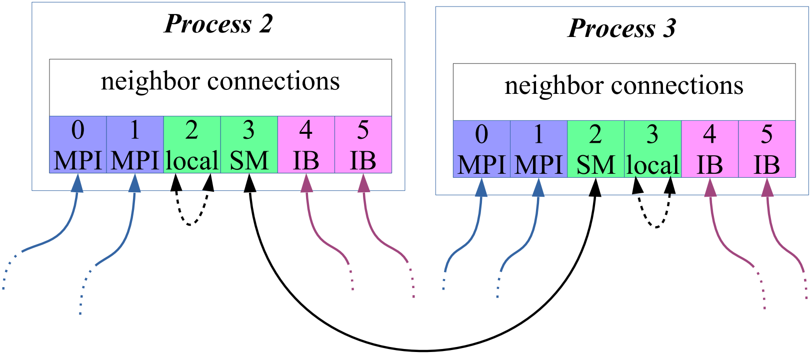
Shared heaps : cornerstone

- One shared heap per process
- Processes on the same node communicate through this heap
  - Segment of shared memory
  - Copy into/from the segment
- Inter-node communications : network
  - Buffers read from/written into this shared memory segment

Run-time environment

- In charge with starting the OpenSHMEM processes, sharing their communication information...
- Any distributed overlay network (currently supported : MPI and PadicoTM)

Network Portability



Currently supported :

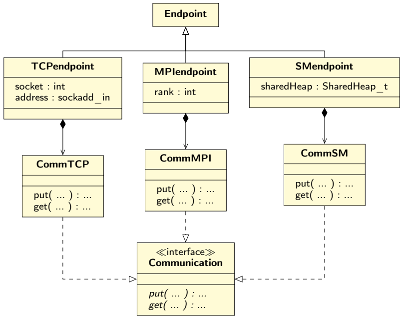
- MPI, TCP, shared memory, KNEM
  - Under testing : Knem and NewMadeleine
- Want to see yours in this list ? Contact us !

Software composition for network portability

The API calls network-specific routines

- Each network driver must implement an **interface**
- ... plus some network-specific methods

→ composition over inheritance



At start-up time, processes discovers how they can communicate with the other ones

- "*Plug*" the right object into the neighbor's local communication gate
  - `Endpoint` (polymorphism)
- Calling `neighbor[rank]->put(....)` will call the appropriate low-level communication routine
  - `Communication` interface

Multi-library programming

DiPOSH lets you **use other parallel programming interfaces**

- For example, MPI
- Possible with DiSPOH's MPI run-time environment

→ Take advantage of **both programming models**

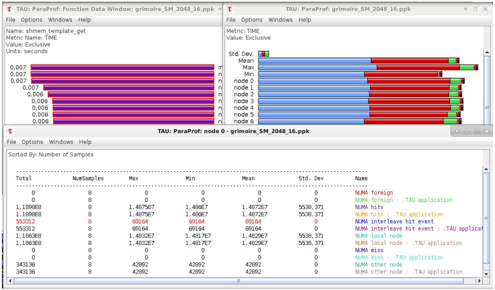
- *e.g.*, mix MPI's two-sided semantics and OpenSHMEM's blocking one-sided semantics

```
start_pes( 0 );
rank = shmem_my_pe();
value = (int*)shmalloc( sizeof( int ) );
/* do stuff */
if( 0 == rank )
    shmem_int_put( value, &result, 1, 1 );
MPI_Barrier( MPI_COMM_WORLD );
/* do stuff */
if( 0 == rank )
    MPI_Send( &number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD );
if( 1 == rank )
    MPI_Recv( &number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &stat );
```

Low-level profiling

TAU already supports OpenSHMEM

- Low-level profiling information
- Tune application using communication optimization



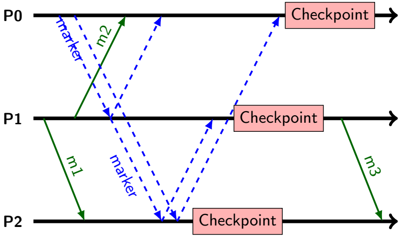
Low-level profiling information

- Usual function calls
  - `MPI_Init()` in red
  - `shmem_*_get` in purple
- Information about the NUMA communications.

Fault tolerance : Chandy & Lamport's algorithm (1985)

Idea : circulate a marker

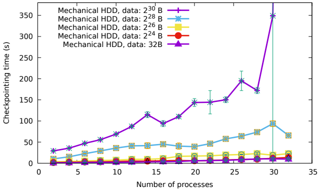
- Initiate the checkpoint wave by **sending a first marker**
- Once a process receives the marker :
  - **Flush** the communication channels
  - Take a **local snapshot**
  - **Send the maker** to all the other processes
- Checkpoint wave done (locally) after reception of all the other processes' markers.



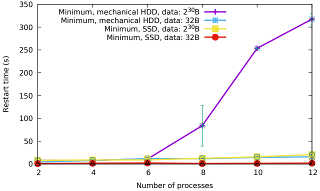
Adaptation : `get()` might cross the marker !

Fault tolerance performance

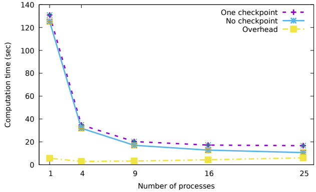
- Checkpointing time scalability on a mechanical HDD with various memory footprints.



- Restart time : SSD and mechanical HDD.



- Scalability of 30 multiplications of two square matrices of size 2048×2048, with and without a checkpoint during the computation. Mechanical HDD.



Checkpoint storage is *critical*

Open perspectives

Network portability (cf [Coti & Malony, PPAM 2019])

- Support other networks
- At various levels : UCX, NewMadeleine... vs InfiniBand
- Support other run-time environments

Use as a **testbed** for distributed algorithms over **one-sided communications**

- Fault tolerance (cf [Butelle & Coti, HPCS 2018])
- Collective communications

OS support

- Shared heap : binding ? Bound to the process it belongs to ? Moving with communications ?

Fault tolerance

- Scalability, other algorithms

... still under development !