Proposal of Interface for Runtime Memory Manipulation of Applications via PGAS-based Communication Library

Takeshi Nanri (RIIT Kyushu University)

at PGAS-EI Workshop
31 Jan, 2018
Background

- PGAS Libraries has become practical candidates as parallel programming tools
  - PGAS-based library
    - OpenSHMEM
    - GASPI
    - Global Arrays
    - ACP
    - etc.
  - Message-Passing based library
    - MPI RMA
  - Fundamental communication library
    - UCX
    - Portals4
    - etc.
- One of their most important advantages is asynchronousness
Proposal of an API for Runtime Memory Manipulation

• Enable asynchronous access to any exposed memory region in an application from outside.

• Empowered by PGAS-based communication library.
Outline

• Advantages of Libraries with PGAS Interfaces

• ACP (Advanced Communication Primitives): an Example of PGAS-based Communication Library

• Proposal of OpAS (Open Address Space) Interface

• Sample Implementation of OpAS on ACP
PGAS vs Message Passing

• PGAS (One-sided communication)
  = Data transfer
  • Lower overhead
  • Lower memory consumption
  • Better overlapping

  ... assuming RDMA is available

• Message Passing
  = Data transfer
  + Synchronization
  • Usually, easier and more efficient to describe data dependency among processes
PGAS Language vs PGAS Library

- **PGAS languages**: UPC, CAF, Xcalable MP, etc.
  - Provides high abstraction of process parallelism
  - Various communication optimizations in compilers are expected
    - Sometimes, they are difficult to apply

- **PGAS libraries**:
  - Enable detailed control of data transfer and synchronization
  - May show better performance for complicated patterns of communication
ACP
(Advanced Communication Primitives)

• An example of "PGAS-based" communication library

• Basic Layer
  • Provides PGAS interfaces.
    • memory management
    • copy
    • atomic
    • barrier
  • Runs on:
    • IB, Ethernet (UDP), Tofu/Tofu2

• Middle Layer
  • Optional interfaces built on top of Basic Layer:
    • Message Passing
    • Data Structures (List, Vector, Map, etc.)
  • Memory consumptions, such as buffers, are explicit

Use this layer for OpAS
Memory Model of ACP

- Local memory:
  - Ordinal address space of each process, managed by OS.

- Global memory:
  - Memory space virtually shared among processes.
    - Any local memory space of any process can be mapped to the global memory via registration.

![Memory Model Diagram]

- Global memory
- Local memory
- CPU
Fundamental Interfaces of ACP

- **Infrastructure**
  - initialization / finalization / synchronization / ranks

- **Global Memory Management**
  - registration / de-registration / query on Global Memory

- **Global Memory Access**
  - copy / atomic on Global Memory
Global Memory Management

- **Registration of Local Memory:**
  - Creates an address-translation key (atkey).

- **Global address:**
  - Queried for the pair of atkey and local address.

- These are "local" operations.
Global Memory Access

- Copy operation
  - Not Get / Put
  - Why? Because this is global memory

- Atomic operation

How to exchange "global addresses"?
Starter Memory

- A special memory region of each process that is registered at the initialization.
  - Global address of starter of any process can be queried directly.
  - Mainly used for exchanging global addresses.
Details of Global Memory Access

- Non-blocking:
  - Wait with handle.

- 'Order' argument:
  - Handle of a previous operation to wait.
  - Used for describing algorithms of patterned communications.

```c
acp_handle_t acp_copy (acp_ga_t dst, acp_ga_t src, size_t size, acp_handle_t order)
```

- In-order completion:
  - Completion of one operation guarantees completions of all preceding operations.
Evaluation of Memory Consumption

- Estimated memory consumption per process of ACP with 1M procs:

<table>
<thead>
<tr>
<th>InfiniBand</th>
<th>Tofu</th>
<th>UDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>369MB / process</td>
<td>67MB / process</td>
<td>34MB / process</td>
</tr>
</tbody>
</table>

- may require 10GB for message passing with 10KB buffer / proc
Fundamental Performance of ACP

Local to Remote (Put)

- UDP/GbE
- IB QDR
- Tofu
- Tofu2

Remote to Local (Get)

- UDP/GbE
- IB QDR
- Tofu
- Tofu2

Remote to Remote
Effect of Overlap with ACP

- Code: 2D stencil with 1D decomposition
  - One-sided version uses "ready flag" for consistency
Communication Time of Stencil

- MPI: Message Passing (isnd) and RMA (actv/pasv) of Open MPI 2.0.0rc3 (om) and MVAPICH2 2.2rc1 (mv)
- ACP: Non-overlap (novlp) and Overlap (ovlp)
ACP + MPI

• ACP can be used to connect multiple MPI codes => enables memory-efficient MPI with smaller communicator
Memory Consumption of ACP + MPI

• Sample code:
  • Hierarchical master-worker
    • Main Master / Sub Master / Worker

• Compare
  • ACP + MPI
  • MPI-Spawn
    • with MPI_Comm_spawn()
  • MPI-Split
    • with MPI_Comm_split()

• Largest memory consumption:
  MPI-Split > MPI-Spawn > ACP
Short Summary of ACP

- Memory registration is **flexible and local**
  - Any region can be registered at any time without synchronization

- Global addresses can be exchanged via **starter memories**

- **copy and atomic** operations within global memory

- **orders** can be used to express dependencies

- **ACP + MPI**
Introduction of OpAS (Open Address Space) Interface

• Leverage "asynchronous", "flexible" and "ACP+MPI" features of ACP.
  • Enable asynchronous access to any exposed memory region in an application from outside.

• Possible usage:
  • In-situ visualization, Runtime manipulation, Debug, etc.
Similar Approach

• Libsim of VisIt
  • Library for In-situ visualization
  • Instructs simulations to communicate with the visualizer (VisIt)
    • Wait for connections
    • Provide information about internal data structure
    • Handle commands from VisIt
    • Notify VisIt that the time step changed etc.
OpAS vs Libsim

- OpAS is more primitive
  - Able to (or Need to) write detailed interactions between the application and the external process.
  - Provide opportunity for other style of interactions
    - Completely independent access from the external process.
      = Less overhead and modification on applications.
    - Ex) Expose once, and do nothing afterwards.

![Diagram of OpAS vs Libsim](image-url)
Structure of OpAS

- OpAS Target
  = Application
- OpAS Server
  = Connector between an application and an external process
- OpAS Client
  = External process interacts with the application
OpAS on ACP

- Invoke OpAS Target and OpAS Server by ACP+MPI
- OpAS Client connects to the server, at anytime.
  - Maybe via TCP/UDP
- Discuss about implementations of Target and Server in this talk.
Example of flows between Target and Server

OpAS Target

- opast_init()
- opast_expose_area("name", addr, size)
- opast_sync()
- opast_lock_area(area, LOCKED)

OpAS Server

- opass_init()
- opass_progress_tgt()
- register "name", rank, addr
- opass_sync()
- opass_open("name", rank, &addr)
- opass_get(addr, offset, size)
- change status to locked
Implementation

• Request Queue on Server

• Exposing Area
  • Target Side
  • Server Side

• Access to the Area

• Lock / Unlock Area
Request Queue on Server

- Used for accepting requests from targets
  - Exposing, Locking and Unlocking Areas

- Prepared at opas{s|t}_init()
  - Malloc, ACP Register
  - Distribute GA via Starter Memory

- Pushed from Target
  - ACP Atomic Add to Tail
  - ACP Copy request to Tail % Qlen

- Handled at Server in opass_progress()
  - Handle request at Head % Qlen
  - Head++
Exposing Area: Target Side

```c
opast_area_t opast_expose_area(
    char *name, void *addr, size_t size, int stat)
```

- ACP Register area

- Push exposure request to the request queue in OpAS Server
  - Rank
  - Global Address
  - Size
  - State
    - ReadOnly or ReadWrite
  - Name
    - ACP Copied separately via Starter Memory because length is unlimited.

- Wait for completion
  - Check a Flag to be updated by the server
  - Also get the index of the Area Information Table in the Server via Starter Memory
Exposing Area: Server Side

- In response to the exposure request from Target

- Store Area Information
  - Via hash table of rank and name

- Send Ack to Target
  - Modify the Flag
  - Also notify index of the area in the table via Starter Memory
Access to the Area from Server

- **Open Area**
  
  ```c
  opass_area_t opass_open_area(int rank, char *name)
  ```
  
  - Return Information of the Area of the Rank and Name

- **Make access**
  
  Ex) Get

  ```c
  int opass_get(opass_area_t area, size_t offset,
      size_t size, void *addr)
  ```

  - Check state
    - Return if not permitted
  - Check offset
  - ACP Copy from Target
    - Pipelined with local buffers
Lock / Unlock Area

• Target
  • Request Lock / Unlock with the index of the area information table to the Server
  • Wait for Ack from the Server

• Server
  • Handle the request in opass_progress() function
  • Change the state of the area to Locked / Unlocked
  • Send Ack to the Target
Evaluation

• Environment
  • Hardware: ITO System in Kyushu University, Japan
    • Intel Xeon Gold 3.0 GHz cluster
    • InfiniBand EDR
  • Software:
    • Open MPI 3.0.0 for Targets
    • ACP 3.0.0 for Targets-Server
  • Target: 4 processes (2 procs / node)

• Measurement
  • Exposure (at Target)
  • Lock (at Target)
  • Get (at Server)
Performance

Latency of Exposure

<table>
<thead>
<tr>
<th>Processes</th>
<th>Latency (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>75</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>127</td>
</tr>
</tbody>
</table>

Latency of Lock

<table>
<thead>
<tr>
<th>Processes</th>
<th>Latency (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>60</td>
</tr>
<tr>
<td>2</td>
<td>76</td>
</tr>
<tr>
<td>3</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>148</td>
</tr>
</tbody>
</table>

Latency and Bandwidth of Get

<table>
<thead>
<tr>
<th>Size</th>
<th>Latency (us)</th>
<th>Bandwidth (MB/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4B</td>
<td>5</td>
<td>0.2</td>
</tr>
<tr>
<td>4KB</td>
<td>67</td>
<td>150</td>
</tr>
<tr>
<td>4MB</td>
<td>734</td>
<td>1,428</td>
</tr>
</tbody>
</table>
Conclusion

• Proposed OpAS (Open Address Space), an interface for runtime access to the memory of applications

• Now we are working for:
  - in-situ visualization
  - runtime manipulation
  - debugging

• Github repository will be prepared, soon
  • ACP is available from
    https://github.com/project-ace/ACP
Acknowledgement

• ACP Library was developed in ACE Project supported by JST CREST
  • Members:
    • Kyushu Univ.
      T. Nanri, H. Honda, R. Susukita, T. Kobayashi and Y. Morie
    • Fujitsu Ltd.
      S. Sumimoto, Y. Ajima, K. Saga, T. Nose and N. Shida
    • ISIT Kyushu
      H. Shibamura and T. Soga
    • Kyoto Univ.
      K. Fukazawa
    • Oita Univ.
      T. Takami