Porting GASNet to Portals: Partitioned Global Address Space (PGAS) Language Support for the Cray XT

Paul Hargrove
Dan Bonachea, Michael Welcome, Katherine Yelick

What is GASNet?

• GASNet is:
  - A high-performance, one-sided communication layer
  - Portable abstraction layer for the network
    - Can run over portable network interfaces (MPI, UDP)
    - Native ports to a wide variety of low-level network APIs
  - Designed as compilation target for PGAS languages
    - UPC, Co-array Fortran, Titanium, Chapel,...
    - On Cray XT, GASNet is targeted by 7 separate parallel compiler efforts and counting:
      – 3 UPC: Berkeley UPC, GCC UPC, Cray XT UPC
      – 2 CAF: Rice CAF, Cray XT CAF
      – Berkeley Titanium, Cray Chapel
      – Numerous prototyping efforts
PGAS Compiler System Stack

- PGAS Code (UPC, Titanium, CAF, etc)
- PGAS Compiler
- Compiler-generated code (C, asm)
- Language Runtime system
- GASNet Communication System
- Network Hardware

- Platform-independent
- Network-independent
- Compiler-independent
- Language-independent
GASNet Design Overview: System Architecture

- Two-Level architecture is mechanism for portability

GASNet Core API
- Most basic required primitives, narrow and general
- Implemented directly on each network
- Based on Active Messages lightweight RPC paradigm

GASNet Extended API
- Wider interface that includes higher-level operations
  - puts and gets w/ flexible sync, split-phase barriers, collective operations, etc
  - Have reference implementation of the extended API in terms of the core API
  - Directly implement selected subset of interface for performance
    - leverage hardware support for higher-level operations
GASNet Design Progression on XT

• Pure MPI: mpi-conduit
  - Fully portable implementation of GASNet over MPI-1
  - “Runs everywhere, optimally nowhere”
• Portals/MPI Hybrid
  - Replaced Extended API (put/get) with Portals calls
  - Zero-copy RDMA transfers using SeaStar support
• Pure Portals: portals-conduit
  - Native Core API (AM) implementation over Portals
  - Eliminated reliance on MPI
• Firehose integration
  - Reduce memory registration overheads
- Lowest-level software interface to the XT network is Portals
  - All data movement via Put/Get between pre-registered memory regions
  - Provides sophisticated recv-side processing of all incoming messages

- Designed to allow NIC offload of MPI message matching
  - Provides (more than) sufficient generality for our purposes
GASNet Put in Portals-conduit

Node 0’s gasnet_put of A to B becomes:

\[ \text{PortalsPut}(\text{RARSRC}, \text{offset}(A), \text{RARME \mid op_id, offset}(B)) \]

Operation identifier smuggled thru ignored match bits
Node 0’s gasnet_get of B to C becomes:
PortalsGet(TMPMD, 0, RARME | op_id, offset(B))

Dynamically-created MD for large out-of-segment reference
GASNet AM Request in Portals-conduit

Node 0’s gasnet_AMRequestMedium becomes:

\[ \text{PortalsPut(ReqSB\_MD, offset(sendbuffer), Req\_ME | op\_id | <AM metadata>, 0)} \]
Node 1’s gasnet_AMReplyMedium becomes:

\[
\text{PortalsPut}(\text{RplSB\_MD}, \text{offset(sendbuffer)}, \text{Rpl\_ME | op_id | <AM metadata>, request\_offset})
\]
### Portals-conduit Data Structures

**RAR PTE:** covers GASNet segment with 3 MD’s with diff EQs  
**AM PTE:** Active Message buffers  
  - 3 MD’s: Request Send/Reply Recv, Request Recv, and Reply Send  
  - EQ separation for deadlock-free AM  
**TMPMD’s** created dynamically for transfers with out-of-segment local side

<table>
<thead>
<tr>
<th>MD</th>
<th>PTE</th>
<th>Match Bits</th>
<th>Ops Allowed</th>
<th>Offset Mgt.</th>
<th>Event Queue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAR</td>
<td>RAR</td>
<td>0x0</td>
<td>PUT/GET</td>
<td>REMOTE</td>
<td>NONE</td>
<td>Remote segment: dst of Put, src of Get</td>
</tr>
<tr>
<td>RARAM</td>
<td>RAR</td>
<td>0x1</td>
<td>PUT</td>
<td>REMOTE</td>
<td>AM_EQ</td>
<td>Remote segment: dst of RequestLong payload</td>
</tr>
</tbody>
</table>
| RARSRC | RAR   | 0x2        | PUT         | REMOTE      | SAFE_EQ     | Remote segment: dst of ReplyLong payload  
  Local segment: src of Put/Long payload, dst of Get                          |
| ReqRB  | AM    | 0x3        | PUT         | LOCAL       | AM_EQ       | Dest of AM Request Header (double-buffered)                                  |
| ReqSB  | AM    | 0x4        | PUT         | REMOTE      | SAFE_EQ     | Bounce buffers for out-of-segment Put/Long/Get, AM Request Header src, AM Reply Header dst |
| RplSB  | none  | none       | N/A         | N/A         | SAFE_EQ     | Src of AM Reply Header                                                      |
| TMPMD  | none  | none       | N/A         | N/A         | SAFE_EQ     | Large out-of-segment local addressing:  
  Src of Put/AM Long payload, dest of Get                                      |
Performance: Small Put Latency

- All performance results taken on Franklin, quad-core XT4 @ NERSC
- Portals-conduit outperforms GASNet-over-MPI by about 2x
  - Semantically-induced costs of implementing put/get over message passing
  - Leverages Portals-level acknowledgement for remote completion
- Outperforms a raw MPI ping/pong by eliminating software overheads
Performance: Large Put Bandwidth

- Portals-conduit exposes the full zero-copy RDMA bandwidth of the SeaStar
  - Meets or exceeds achievable bandwidth of a raw MPI flood test
  - Mpi-conduit bandwidth suffers due to 2-copy of the payload
Portals-conduit Flow Control

• Most significant challenge in the AM implementation
  - Prevent overflowing recv buffers at the target
  - Prevent overflowing EQ space at either end

• Local-side resources managed using send tokens
  - Request injection acquires EQ and buffer space for send and Reply recv
  - Still need to prevent overflows at remote (target) end

• Initial approach: statically partition recv resources between peers
  - Reserve worst-case space at target for each sender to get full B/W
  - Initiator-managed, per-target credit system
    - Requests consume credits (based on payload sz), Replies return them
  - Downside: Non-scalable buffer memory utilization

• Final approach: Dynamic credit redistribution
  - Reserve space for each receiver to get full B/W
  - Each peer starts with minimal credits, rest banked at the target
  - Target loans additional credits to “chatty” peers
Shows the benefit of implementing AM natively
Portals-conduit AM’s outperform mpi-conduit
  - Less per-message metadata, big advantage under 1 packet
  - Beyond one packet, less software overheads w/o MPI
• **Blocking** put test (no overlap), exaggerates software overheads
• TMPMD pays synchronous MD create/destroy every transfer
  - Incurs a pinning cost linear in the page count (on CNL)
• Firehose exploits spatial/temporal locality to reuse local MDs
  - LRU algorithm with region coalescing – quickly discovers the working set
  - Provides 4% to 8% bandwidth improvement
Conclusions

• Portals-conduit delivers good GASNet performance on Cray XT
  - Outperforms generic GASNet-over-MPI by about 2x
  - Microbenchmark performance competitive with raw MPI
  - Solid comm. foundation for many PGAS compilers

• Future Work
  - Expand Firehose integration to include remote memory

• Funding / Machine acknowledgements:
  - Office of Science DOE Contract DE-AC02-05CH11231.
  - NERSC, DOE Contract DE-AC02-05CH11231.
  - ORNL, DOE Contract DE-AC05-00OR22725.
  - NSF TeraGrid & PSC