The Future of GASNet

Paul H. Hargrove

PHHargrove@lbl.gov

https://sites.google.com/a/lbl.gov/gasnet-ex-collaboration/

Overview

- High-level introduction
  - –GASNet's role in DEGAS
- Mid-level introduction
  - –A survey of the current GASNet API
- The Future
  - –A survey of the GASNet-EX plans

HIGH-LEVEL INTRODUCTION

GASNet Background

- NOT an API for applications authors
  - Library/runtime authors
  - Machine-generated code
- Rich set of one-sided Put/Get interfaces
  - Good mapping to capabilities of modern network H/W
- Active Messages
  - "Function Shipping"
  - "Remote Procedure Call"
- MPI-interoperable
  - (most of the time)

GASNet

IN A NUTSHELL

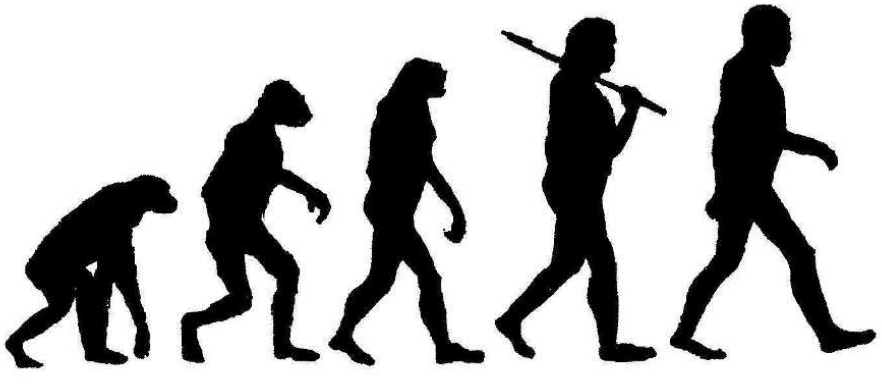
Project Role for Communications

To serve…

- "Communication is an artifact"
  - We don't communicate for its own sake
- Enable efficient implementation of high-level language ideas
- Support communication needs of the other project components (BLCR, IPM, etc.)
- Will need requirements gathering
  - Already have feedback from
    - Yili – re: Echelon/Sequioa & re: implementing collectives
    - Rice – re: CAF runtime
    - Cray – re: Chapel runtime (they've not complained yet ☺)

Evolutionary Work

- Better support for asynchronous runtimes
  - –Don’t assume ever library entry is a “yield”
  - –Finer-grained buffer management/ownership
- Better support for Active Messages clients
  - –More flexible “work flows”
  - –Better buffer management approach(es)



Revolutionary Work

- Support resilience and migration efforts
  - "Consistent" checkpointing of GASNet jobs
  - Enable migration (platform independent manner)
- Introspection and instrumentation
  - For IPM, adaptation and autotuning
- Dynamic job membership
- Multi-client support (hybrid applications)
- More thread-centric (vs. process-centric)

SURVEY OF THE CURRENT GASNET API

GASNet Core API

- GASNet Core API
  - Job Control
    - Init, Attach and Exit
  - Active Messages
    - Categories: Short, Medium and Long
    - Request and Reply
  - Atomicity Control
    - Handler-Safe Locks and No-Interrupt Sections

GASNet Core: Job Control

- `gasnet_init()`
  - Analogous to MPI_Init()
  - Might spawn processes on some platforms
  - Call exactly once per process ("node")
- `gasnet_attach()`
  - Roughly analogous to MPI_Win_create()
  - Allocates the GASNet segment
  - Call exactly once per process
- `ganet_exit()`
  - Roughly like MPI_Finalize() with a timeout

GASNet Core: Active Messages I

- An Active Message (AM) is a remote procedure call
  - Specify node on which to run
  - Specify function by index (established at attach)
  - Some number of 32-bit integer arguments
  - Optional payload determined by “category”
- Three “Categories” of AM
  - Short: no payload
  - Medium: payload in GASNet-managed buffer
  - Long: payload in caller-specified location
    - Location must be “in-segment”

GASNet Core: Active Messages II

- Initiating an AM Request (where *M* is an integer):
  - `gasnet_AMRequestShort`*M*`()`
  - `gasnet_AMRequestMedium`*M*`()`
  - `gasnet_AMRequestLong`*M*`()`
  - `gasnet_AMRequestLongAsync`*M*`()`
- Initiator specifies target node, args and payload
- Medium and Long block until payload is reusable
- LongAsync may return before payload is reusable
  - A Reply is required to release the payload

GASNet Core: Active Messages III

- An Active Message "handler"
  - –Client-provided code runs on the target node
  - –May run "synchronously"
    - Client should occasionally call `gasnet_AMPoll()`
  - –GASNet may run handlers asynchronously
    - True even if client is single-threaded
- Client provide the handler code matching template prototype, which includes:
  - –An opaque "token"
  - –Payload address and length (Medium and Long)
  - –The 32-bit handler arguments

GASNet Core: Active Messages IV

- A Request Handler (the code run remotely)
  - –May use Handler-Safe Locks
    - More on this later
  - –May reply at most once to the initiator
    - Reply functions have a token arg in place of node
  - –May make a limited set of other GASNet calls
    - NOT permitted to make AM Requests
    - NOT permitted to make Extended API calls
- Issuing Replies (where *M* is an integer)
  - –`gasnet_AMReplyShortM()`
  - –`gasnet_AMReplyMediumM()`
  - –`gasnet_AMReplyLongM()`

GASNet Core: Atomicity Control

- Handler-Safe Locks (aka HSLs)
  - Like pthread mutexes with usage restrictions
  - AM handler may acquire an HSL, but must release before return
  - While holding an HSL a client must *not*
    - Make GASNet communication calls
    - Make calls to `gasnet_AMPoll()`
  - May not be acquired recursively
  - Must be released in reverse order of acquisition
- No-Interrupt Sections
  - Suspends interrupt-driven handler execution
  - Similar to blocking POSIX signals

GASNet Survey: Extended API

- The Extended API
  - –Put and Get
    - Memory-to-memory and Register-based
    - Blocking, Explicit-handle NB, Implicit-handle NB
    - Bulk and non-bulk
  - –Barrier
  - –Unofficial additions

- A “reference implementation” implements the entire Extended API in terms of the Core
  - –Network/platform specific code can individually replace portions with optimized versions

Extended API: memory and registers

- Memory-to-memory transfers:
  - Destination of a Put must be in-segment
  - Source of a Get must be in-segment
  - Local address is unconstrained

- Register-to-memory and memory-to-register:
  - Can Put values passed by-value
  - Can Get values as function return value
  - Remote address must be in-segment
  - Limited to 1, 2, 4 or 8-byte quantities

Extended API: blocking and non-blocking

- Three variants of most Put and Get calls
  - Blocking
    - Calls return when data movement is complete
  - Explicit-handle non-blocking ("nb")
    - Calls return a handle used to block/poll for the completion of data movement
    - Can try or wait single, "some" or "all" handles
  - Implicit-handle non-blocking ("nbi")
    - Calls have `void` return type
    - Synchronize (wait or try) for outstanding nbi operations (Put, Gets or All)
    - Can use "access regions" to convert a series of nbi operation into a single explicit handle

Extended API: bulk and non-bulk

- Two "flavors" of Put and Get call
  - Independent of blocking, nb and nbi
- Bulk
  - No requirement on alignment of address or size
  - For non-blocking Put, the source buffer is not safe to reuse until the operation is completed
- Non-bulk
  - Address and size must be "aligned"
  - Non-blocking Puts don't return until the source buffer is safe to reuse

Extended API: Barrier

- GASNet's barrier is modeled after UPC's
- Barrier is "split-phased"
  - Step 1: Notify
    - Imagine incrementing an arrival counter
  - Step 2: Wait or Try
    - Imagine blocking or polling the counter
  - Client can do work between these steps
- Barrier is optionally "named"
  - Each node may independently specify an integer value or the "anonymous" flag
  - If more than one distinct value is passed then an error code is returned from the wait or try call

Extended API: Unofficial Extras

- VIS: Vector, Indexed and Strided
  - Calls to Put or Get non-contiguous data
  - Vector: array of (addr,len) pairs
  - Indexed: array of indices and a single length
  - Strided: slices of multi-dimensional arrays
- Collectives
  - Based on UPC data-movement collectives
    - Broadcast, scatter, gather, gather-all, exchange
  - Non-blocking and blocking
  - Specialized interfaces for threaded clients
  - "Teams" support almost complete

SURVEY OF FUTURE WORK (GASNET-EX)

Future: High-Level I

- Multi-client support
  - No longer limited to single Init and Attach
  - Can have multiple segments (memory regions)
  - Can have multiple AM handler tables
- Resilience and migration support
  - Implementation-level work to “run-through”
  - Mechanisms to expose errors to client
    - Return codes and error callbacks
    - Sparse naming of nodes (processes)
- Dynamic job membership
  - Can add and remove compute nodes

Future: High-Level II

- Unofficial features become official
  - –Document the VIS extensions
  - –Complete Collectives with simpler interface
- Remove unused/unimplemented features
  - –No-interrupt sections
    - No client uses them correctly anyway!
    - Never had an interrupt-driven platform
  - –PARSYNC (like MPI_THREAD_SERIALIZED)
    - Not aware of any client for this mode
    - Never implemented better than PAR

Future: High-Level III

- Progress Functions
  - Client-provided code which GASNet runs when blocked
    - Non-communicating work for ANY context
    - Communicating work for non-handler context

Future: Active Messages I

- Issue: fixed-argument Request and Reply calls
  - Makes for messy client code when passing pointer or `size_t` arguments as either 1 or 2 32-bit arguments
- Solution: add varargs Request and Reply calls
  - The "`M`" becomes an argument instead of part of the function name

Future: Active Messages II

- Issue: multiple copies in constructing AM payload
  - The "user" code passes args to some runtime
  - The runtime copies user's data to a buffer to marshal it together with its own data
  - This buffer is passed to AMReqestMedium
  - GASNet copies the buffer *again*
    - To expedite return of control to caller
    - Possibly to pre-pinned memory
    - To marshal with its own header
- Solution 1: add a MediumAsync request
- Solution 2: add call to allocate buffer from GASNet

Future: Active Messages III

- Issue: LongAsync requires a Reply
  - –Use of any other synchronization disallowed
  - –At least one current platform truly requires this
- Solution: drop this requirement from the spec
  - –Replace with rule that source buffer is safe to reuse as soon as handler begins execution
    - Reply is one option
    - Handler might set a flag that another thread uses to signal (via AM, Put, barrier, etc.)
  - –Implementation will be responsible for the additional work to ensure this works
- NOTE: will apply to MediumAsync if such is added

Future: Active Messages IV

- Issue: Reply-at-most-once rule is limiting
  - Request Handler cannot send to a third party
  - Reply Handler cannot communicate at all
- Solution: Multiple independent virtual networks
  - Each Attach may instantiate another network
  - The reply-at-most-once still applies *per-network*
  - Handlers may Request on "higher" networks
  - Implementation still needs only finite resources per-network to ensure deadlock freedom

Future: Active Messages V

- Issue: largest Medium may under utilize network
  - Typical implementation has a fixed-sized buffer for assembly of AM Medium (header+payload)
  - The max size of a Medium is often determined by reserving space for the max number of args
  - Mediums with less than the max arguments may therefore waste up to 10% of the buffer space
  - An issue in fragmentation/reassembly scenarios
- Solution: variable-length AM Medium
  - Implementation sends as much as it can fit and returns the count of bytes sent
  - Work very much like short-writes to sockets

Future: Put/Get I

- Issue: even non-blocking calls might block
  - Will spin-pool to progress the lower-level API if there are insufficient resources available
  - While polling it may or may not be possible to run AM Reply handlers, but little else
- Solution 1: "now-or-never" flag
  - Caller can request that *instead* of spin polling, the call return a failure code
  - Caller may reissue call later or use some alternative that doesn't require this communication
- Solution 2: progress functions (described earlier)

Future: Put/Get II

- Issue: "trysync" of an NB handle runs progress engine
  - Client wants to call `gasnet_AMPoll()` once in its own progress loop
  - Client then has many handles (not marshaled in an array for a "trysome" call) to test
  - Client want to amortize the GASNet progress costs over all the handles it must test
- Solution: add "test" calls that don't try to progress
  - Already implemented as undocumented "`try_*_nopoll`" calls in current release

Future: Put/Get III

- Issue: “bulk” conflates alignment with the buffer lifetime/ownership of Puts (but not of Gets)
- Solution: separate these two concepts
  - “Bulk” will assert only alignment
  - Use a flag to Puts to determine when to return
  - Most implementations don’t care about the alignment anyway

Future: Put/Get IV

- Issue: client needs a “fence” between ops
  - –Blocking for first op is undesirable
  - –Tracking of handles is burdensome OR not possible due to use of nbi operations.
- Solution: add “dependent” operations
  - –Completion of an operation will initiate any dependent operation(s)
  - –Can map to lower-level API in some cases

Future: Handles I

- Issue: nb handles are thread-specific
  - Prevents client-level progress threads
  - Complicates reference implementation of barrier and collectives
- Solution: remove the thread-specific restriction
  - Current implementations don't have any true thread-specific nature to the handles
  - This rule does have the advantage of ensuring no locking required to sync (try or wait), and implementation will need to address the loss of this assurance

Future: Handles II

- Issue: spec only allows 65536 outstanding ops per thread
  - At most this many nb handles outstanding on any thread
  - At most this many nbi operations outstanding on any thread
  - Not sure current clients are aware of the nbi restriction
- Solution: remove the limit for nbi (keep for nb)
  - All modern architectures can support this with zero overhead relative to the current code

Future: Barrier

- Issue: UPC semantics are very heavy weight
  - Can't use h/w barrier on any current system
- Solution 1: Introduce UNNAMED barrier flag
  - Must be passed by all callers or by none
  - Turns off name matching entirely
  - Sufficient to use many h/w barriers (e.g. BG/Q)
- Solution 2: Introduce single-phase barrier
  - Can be more efficient than split-phase
  - May enable use of additional h/w support (FCA)

I will be at the poster session with these slides as my poster.

I am very open to questions, comments and discussion.

THANK YOU