PGAS, Global Arrays, and MPI-3
How do they fit together?

Brad Chamberlain
Chapel Team, Cray Inc.

Global Arrays Technical Meeting

May 7, 2010

CRAY
THE SUPERCOMPUTER COMPANY

Disclaimer

This talk's contents should be considered my personal opinions (or at least one facet of them) and not necessarily those of Cray Inc. nor my funding sources

Outline

- PGAS Programming Models
- PGAS and communication libraries like ARMCI, MPI-3
- PGAS and Global Arrays

PGAS Definition

PGAS: Partitioned Global Address Space
(Or perhaps better: partitioned global namespace)

Concept:

- support a strong sense of ownership and locality
  - each variable is stored in a particular memory segment
  - local variables are cheaper to access than remote ones
  - details vary between languages
- support a shared namespace
  - permit tasks to access any lexically visible variable, local or remote

```
var x = 10;
…
{
  … x …          // x is lexically visible
}
```

PGAS Languages

charter members: UPC, Co-Array Fortran, Titanium

- extensions to C, Fortran, and Java, respectively
- details vary, but potential for:
  - arrays that are decomposed across nodes
  - pointers that refer to remote objects

honorary retroactive members: HPF, ZPL, Global Arrays, …

the next generation: Chapel, X10, …

PGAS: What's in a Name?

		memory model	*programming model*	*execution model*	*data structures*	*communication*
	MPI	distributed memory	cooperating executables (often SPMD in practice)		manually fragmented	APIs
	OpenMP	shared memory	global-view parallelism	shared memory multithreaded	shared memory arrays	N/A
PGAS Languages	CAF	PGAS	Single Program, Multiple Data (SPMD)		co-arrays	co-array refs
	UPC				1D block-cyc arrays/ distributed pointers	implicit
	Titanium				class-based arrays/ distributed pointers	method-based
	Chapel	PGAS	global-view parallelism	distributed memory multithreaded	global-view distributed arrays	implicit

PGAS: What's in a Name?

		memory model	*programming model*	*execution model*	*data structures*	*communication*
	X10	PGAS	global-view parallelism	distributed memory multithreaded	global-view distributed arrays	visible in syntax
	Global Arrays	PGAS	Single Program, Multiple Data (SPMD)		global-view distributed arrays	implicit via APIs
PGAS Languages	CAF	PGAS	Single Program, Multiple Data (SPMD)		co-arrays	co-array refs
	UPC				1D block-cyc arrays/ distributed pointers	implicit
	Titanium				class-based arrays/ distributed pointers	method-based
	Chapel	PGAS	global-view parallelism	distributed memory multithreaded	global-view distributed arrays	implicit

Evaluation: Traditional PGAS Languages

Strengths:

+ Clean expression of communication through variable namespace
+ Able to reason about locality/affinity to support scalable performance
+ Separation of data transfer from synchronization to reduce overheads
+ Elegant, reasonably minimalist extensions to established languages

Weaknesses:

– Limited to a single-threaded SPMD programming/execution model
  - unless you mix in some other model (*if* you are able to)
– Conflation of parallelism and locality
  - MPI has this problem as well: a PE is the unit for both concepts
  - fails to support Harrison's "non-process-centric computing"
– Arrays more restricted than ideal
  - no native support for sparse, associative, unstructured arrays
  - CAF: no global-view, supports distributed arrays of local arrays
  - UPC: as in C, 1D arrays only

A Design Principle HPC should revisit

"Support the general case, optimize for the common case"

Claim: a lot of suffering in HPC is due to programming models that focus too much on common cases:

- *e.g.*, only supporting a single mode of parallelism
- *e.g.*, exposing too much about target architecture and implementation

Impacts:

- hybrid models needed to target all modes of parallelism (HW & SW)
- challenges arise when architectures change (e.g., multicore, GPUs)
- presents challenges to adoption ("linguistic dead ends")

That said, this approach is also pragmatic

- particularly given community size, (relatively) limited resources
- and frankly, we've achieved a lot of great science when things fit

But we shouldn't stop striving for more general approaches

How do HPCS PGAS languages differ?

- **focus on general parallel programming**
  - support for task- and data-parallelism, concurrency, nestings of these
  - support multiple granularities of software and hardware parallelism
- **distinct concepts for parallelism vs. architectural locality**
  - *tasks* are units of parallel work
  - *locales*/*places* are units of architectural locality
  - each locale/place can execute multiple tasks
- **post-SPMD (APGAS) programming/execution models**
  - user is not forced to code in an SPMD style (but may)
  - ability to run multiple threads within each process/node
- **rich support for arrays**
  - multidimensional, sparse, associative, unstructured
  - user-defined distributions and memory layouts

Implementing PGAS

	Single-sided communication	Remote Task Creation	Thread Safety	Portability
Traditional PGAS	✓	--	--	✓
HPCS PGAS	✓	✓	✓	✓

	Single-sided communication	Remote Task Creation	Thread Safety	Portability
MPI-2	✘	✘	~	✓
SHMEM	✓	✘	✘	✘
ARMCI	✓	✓	~	✓
GASNet	✓	✓	✓	✓
MPI-3	?	?	?	?

MPI-3 for PGAS

Two key working groups:

- *Remote Memory Access* (Single-sided Communication)
  - **Goal:** Do 1-sided communication right (fix MPI-2's support)
  - **Status:** active
  - **References:**
    - *Investigating High Performance RMA Interfaces for the MPI-3 Standard*, Tipparaju et al., ICPP 2009
    - https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/RmaWikiPage
  - **Challenges:** memory consistency model (how strict or weak?)
  - **Concern:** are ARMCI and GASNet teams sufficiently involved?
- *Active Messages*
  - **Goal:** add support for active messages to MPI
  - **Status:** inactive
  - **References:** http://meetings.mpi-forum.org/mpi3.0_am.php
  - **Note:** Active Messages are a nice fallback for implementing RMA in cases that the hardware doesn't, so this shouldn't get left behind

MPI-3 for PGAS: Great Potential

- Would provide a long-overdue standard for 1-sided comm.
- Would result in multiple implementations of the same API
- Would result in vendor-optimized implementations
- Would support standardized single-sided benchmarks
- Could drive hardware designs in favor of global address spaces, better support for single-sided communication
- Would greatly help with interoperability between distinct programming models
- Would permit the ARMCI/GASNet teams to claim victory and work on things other than keeping up with each new network

Failing this, a merge of ARMCI and GASNet would be the next best thing for users like us, helping with many of the above

What about GA and HPCS languages?

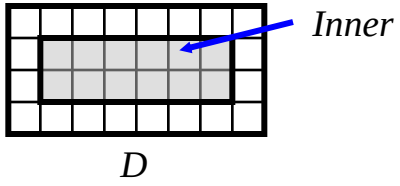
- First, let me touch on a few relevant Chapel features…

Chapel Domains and Arrays

Domains: first-class index sets

- describe size/shape of arrays (potentially distributed)
- describe iteration spaces
- describe index sets for high-level operations

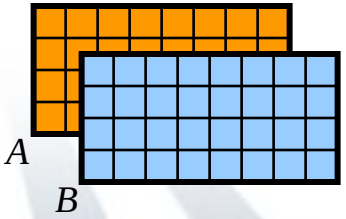
```
var D: domain(2) = [1..4, 1..8];
var Inner: subdomain(D) = [2..3, 2..7];
```



Arrays: mappings from domain indices to variables

- *multidimensional:* arbitrary dimensions and element types
- *associative:* map from arbitrary values to elements (hash table-like)
- *unstructured:* support irregular (pointer-based) data structures
- *sparse:* support sparse subsets of any of the above

```
var A, B: [D] real;
```



Chapel Distributions and Layouts (*Domain Maps*)

Goal: separate algorithm from implementation for parallel, distributed data structures

Approach:

- permit users to author their own distributions & layouts, written in Chapel
- rather than a simple mapping, use a method-based approach
  - authors create classes that implement domains and arrays:
    - how they're mapped between locales
    - how they're stored in memory
    - how to access them, iterate over them, slice them, etc.
- implement standard domain maps using the identical framework
  - to avoid performance cliffs between user cases and "built-in"
- a work-in-progress: operational today, but still evolving and being optimized
- For a general overview, see upcoming HotPAR paper:
  User-Defined Distributions and Layouts in Chapel: Philosophy and Framework

Interoperability in Chapel

- A crucial goal for the adoption of Chapel (or any new language) is general interoperability with existing languages
  - to preserve legacy application code
  - to re-use existing libraries
  - to permit existing kernels to be rewritten using new technologies
  - to support differing tastes and diverse needs within a single app.
- Our initial focus is primarily on supporting calls from Chapel to C to reuse key libraries
- We believe user-defined distributions will play a key role in interoperating with existing distributed data structures *in situ*
  - *e.g.,* "I want to view my distributed MPI data structures as a global array within Chapel"

Global Arrays and (HPCS) PGAS languages

At first blush, there are both synergies and mismatches:

- **Similarities:**
  - global-view data structures
  - permit users to think in logical terms rather than physical
- **Differences:**
  - SPMD vs. post-SPMD programming/execution models
  - moving data only vs. moving data and computation
  - varying degrees of richness in data structures
    - limitations on rank, element type, flavor, distributions
  - different levels of maturity
    - GA is here, in-use, and successful
    - HPCS languages are still emerging and proving themselves
- In spite of differences, let me propose four possible interaction models

1) Support GA-based distributions

Concept: Use Chapel's interoperability features to create a distribution that wraps Global Arrays

- benefit from effort invested in GA and preserve it
- provide language-based support for GA-style computations

```
var D: domain(2) dmapped GA(…) = [1..4, 1..8];
```

2) Support GA-HPCS Interoperability

Concept: Investigate what it would take to support Chapel/X10 and GA within a single program

- in terms of mechanics like runtime software layers
- in terms of language semantics
  - at what granularity could languages execute within an app.?
  - could they share data structures?
  - what types of control flow exchanges could occur?

Note: Tom Epperly and Jim McGraw from LLNL (Babel team) are working on organizing an interoperability workshop this summer/fall to look at questions like this across today's major parallel models (MPI, OpenMP, PGAS, HPCS, etc.) Please contact them if interested in participating.

3) Implement Global Arrays over HPCS Language

- Fast-forward ten years into the future and imagine that Chapel *and* X10 are unmitigated successes (!)
- Yet, legacy GA code still exists

Concept: Implement GA in terms of Chapel/X10

- removes need to duplicate lower-level layers of software
- preserves familiar interface for existing codes and users

A General Challenge: How do we reduce our need to maintain every programming model going forward?

- What is the right way to retire a programming model?
- As indicated before, I don't think the answer is "cease to make new ones"…

4) Exchange of Information

- Many of GA's emerging directions mesh with themes being pursued in the HPCS languages
  - task queuing for dynamic load balancing/work stealing
  - sparsity
  - user-defined data structures
  - next-generation applications
  - petascale and exascale architectures

Concept: Exchange information in these common areas

Better: architect code so that it can be shared

Also: Perform cross-language comparisons of flexibility, readability, maintainability in application codes and idioms

Summary

GA and PGAS languages have similar needs

- single-sided communication
- active messages (in GA, for future idioms and architectures)
- in my opinion, support for both these things in MPI-3 would represent a huge step forward for HPC

Yet, there are also differences

- GA is more evolutionary: what works well and what can we add next?
- HPCS strives for revolution: what might an ideal parallel language be like?

We have many common challenges to work on

- more advanced computational idioms
- dynamic load balancing
- resiliency
- exascale's increasingly heterogeneous/hierarchical nodes
  - DOE workshops have identified desire for evolutionary and revolutionary, yet existing, programming models

For more information on HPCS languages

Chapel:

chapel_info@cray.com
http://chapel.cray.com
https://sourceforge.net/projects/chapel/

X10:

http://x10.codehaus.org/
http://www.research.ibm.com/x10/