Domain Map Improvements

Chapel Team, Cray Inc.

Chapel version 1.14

October 6, 2016

Safe Harbor Statement

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

Outline

- Stencil Distribution
- Sparse Block Distribution
- Bulk Add for Sparse Domains
- Locality Queries for Domains and Distributions
- Associative Domain Improvements
- Other Domain Map Improvements

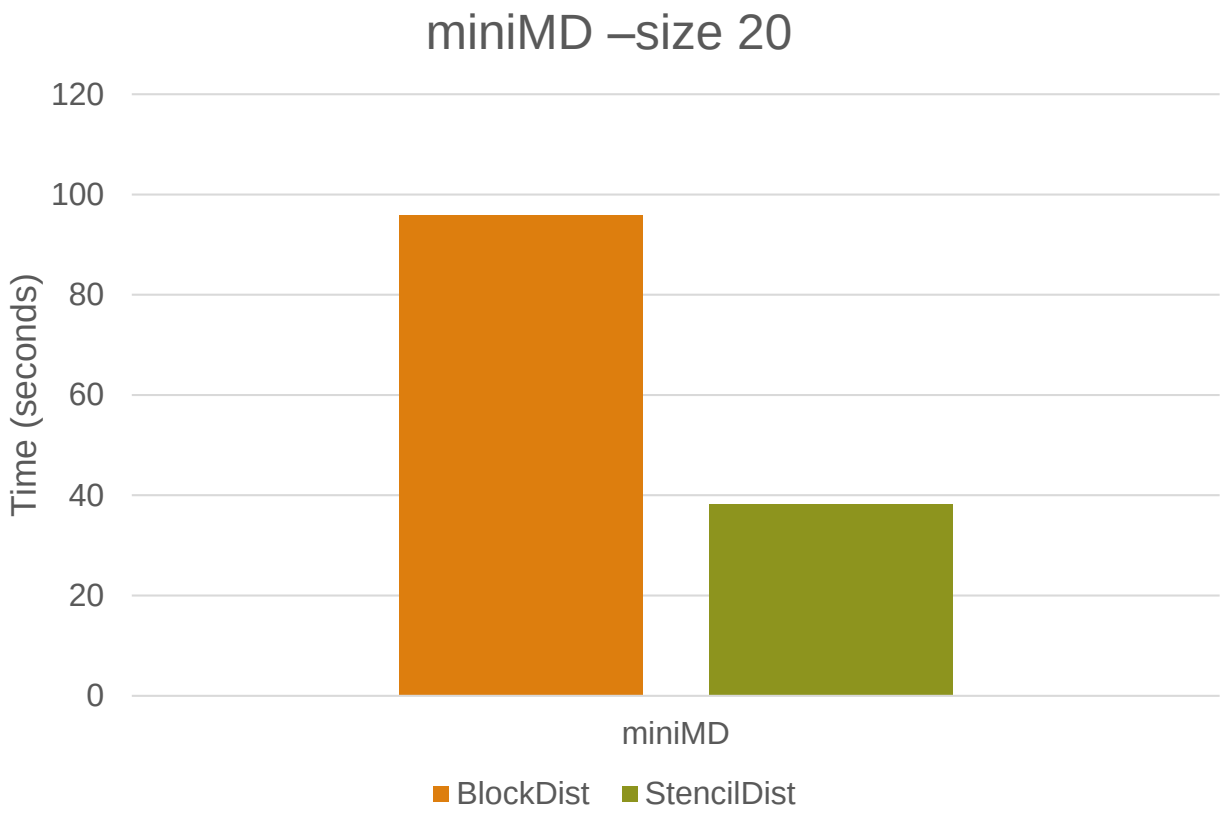
Stencil Distribution

StencilDist: Background

- **Block-like distribution that caches remote halo elements**
  - For a user-provided number of indices outside the domain
- **Useful because stencils access neighboring elements**
- **Initially developed during miniMD intern project**
  - Functional for miniMD, but buggy for other use cases
  - Lived under miniMD's source tree in previous releases
- **Occasional interest from external users**
  - Typically replied with "Cool! But be aware of bugs"
  - Having to copy the distribution from miniMD's directory was odd

StencilDist: Background

- **Much better miniMD performance compared to BlockDist**
  - 16 BW nodes on XC30



StencilDist: This Effort

- **Promoted StencilDist to $CHPL_HOME/modules/dists**
  - Makes it simple to "`use`" in a program
- **Improved testing**
  - Confirmed and exposed some embarrassing bugs
- **Bug fixes**
  - Slicing (used to drop cached elements)
  - Strided domains (had been completely broken)
  - Block-like view (now its own method)
    - Supports a view with no caching, easier to reason about reads/writes

StencilDist: Status and Next Steps

Status:

- Now a standard distribution
- No known correctness bugs

Next Steps:

- Improve performance
  - Look at some known locality issues
- Investigate new features
  - 6-face halo exchange vs. 27-point
  - Only cache on one side of an axis (PRK motivated)
- Possibly merge with BlockDist to create a single powerful distribution
- Move towards having compiler automatically insert communication

Sparse Block Distribution

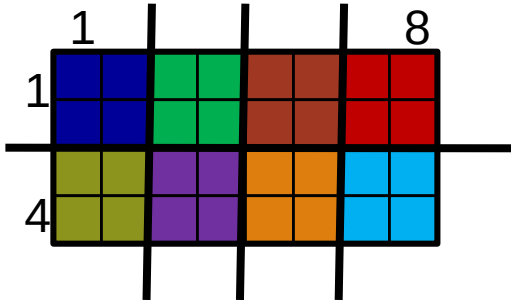
Sparse Block: Background

- **Sparse Block originally prototyped in fall 2012**
- **Supported distributed sparse arrays and domains**
  … but prototype fell by the wayside

- **Sparse Block implemented with the Block distribution**
  - the data distribution is the same
  - but store sparse local domains and arrays rather than dense

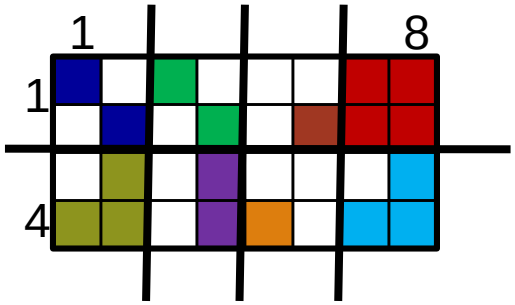
Sparse Block: Example

```
var Dom = {1..4, 1..8} dmapped Block( {1..4, 1..8} );
```



distributed to


```
var SparseDom : sparse subdomain(Dom);
var Indices = [(1,1), (2,2), (3,1), …]
SparseDom += Indices;
```



distributed to

Sparse Block: This Effort

- **Cleaned up the prototype from 2012 and get it working**
  - fixed compiler bugs
  - added missing features
- **Supports users of distributed sparse data structures**

Sparse Block: Status and Next Steps

Status:

- Basic functionality available
  - There may be missing features – not broadly tested yet
- bulkAdd or using += in bulk is critical to performance
- Available in 1.14 release

Impact: Enables sparse / distributed graph algorithms

Next Steps:

- Build more with sparse block
- Consider other ways of declaring sparse arrays and domains
- Further optimize communication for sparse block
- Investigate related compiler optimizations
- Explore distributions other than Block (e.g., recursive bisection)

Bulk Add for Sparse Domains

Bulk Add

Background: Often, += is used to add elements to a domain

```
var DenseDomain = {1..10};
var D: sparse subdomain(DenseDomain);
D += 1; D += 2; D += 3; D += 5; D += 7;
```

- but this approach has significant performance overhead
  - per-element locking, reallocating, sorting, communicating, resizing arrays

This Effort: sparse domains now support bulk addition

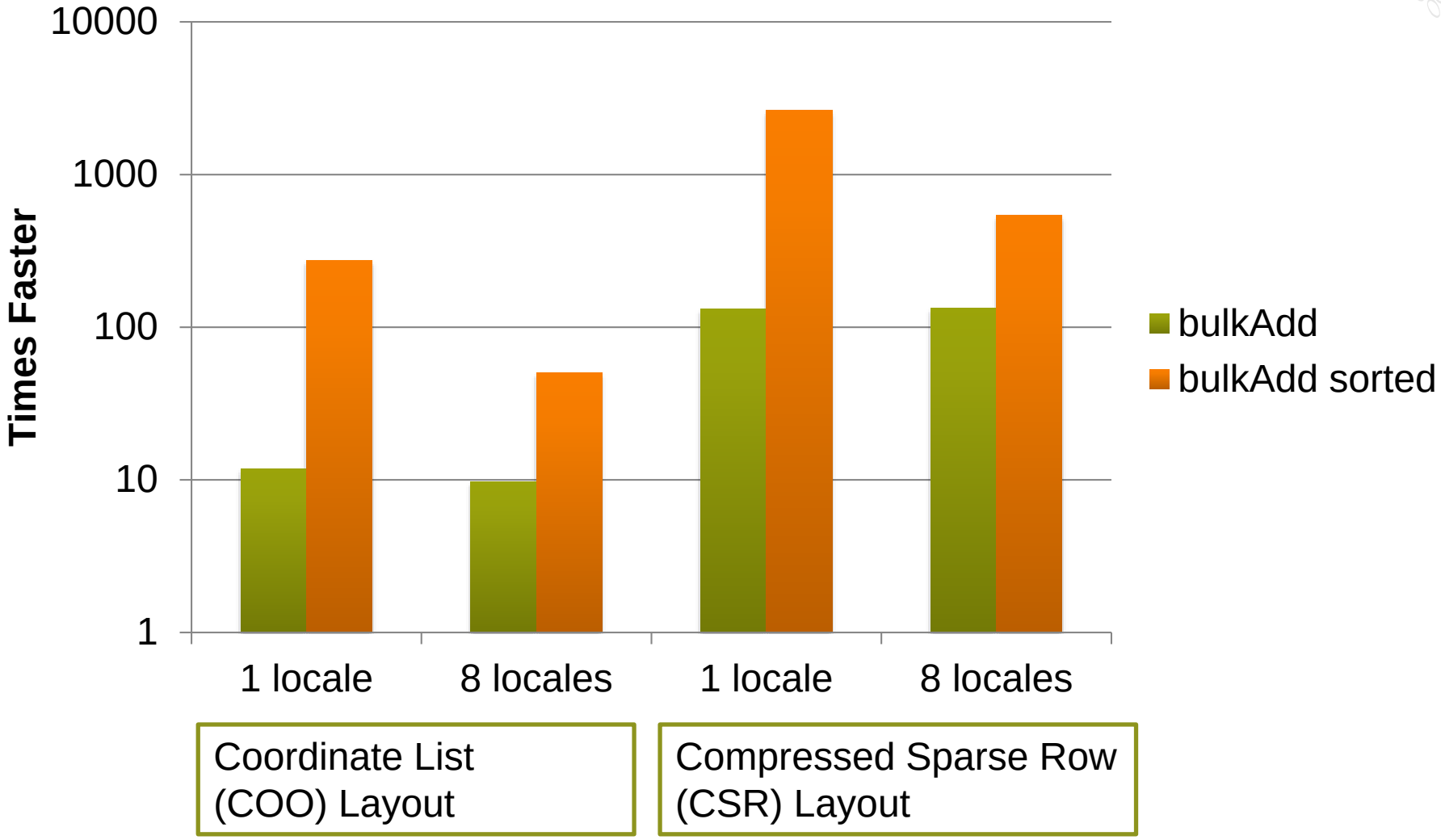
```
D += [1,2,3,5,7];
// or, for more control:
D.bulkAdd([1,2,3,5,7], dataSorted=true);
```

Impact: Greatly reduces overhead

Next Steps: Support bulkAdd efficiently on all irregular domains

- Associative and Opaque

Speedup from bulkAdd, adding 100000 indices



Locality Queries for Domains and Distributions

Locality Queries for Domains and Distributions

Background: Chapel supports locality-based queries on arrays

```
A.targetLocales()                 // returns the locales grid
A.getLocalSubdomain()             // returns the single local portion, if possible
A.getLocalSubdomains()            // iterates over all local portions
A.hasSingleLocalSubdomain()       //  check if getLocalSubdomain can be called
```

- these were only supported on arrays

This Effort: Added these operations to domains and distributions

Impact: Arrays no longer required to make these queries

	distribution	domain	array
.targetLocales()	✓	✓	✓
.getLocalSubdomain() .getLocalSubdomains()		✓	✓

Associative Domain Improvements

Associative Domain Sizes

Background: Associative domains implemented via a hash table

- implemented with quadratic probing
  - uses a table of prime numbers
- these primes previously stopped at about 2 billion elements,
  - this limited the size of associative domains

This Effort: Improve the primes used by associative domains

- primes now go up to 2^{60}
- primes selected to maximize usage of jemalloc allocation blocks

Impact: Larger associative domains are possible

Next Steps: Consider supporting different hash table algorithms

requestCapacity on Associative Domains

Background: requestCapacity existed but wasn't documented

- useful when adding many elements to an associative domain
- can be used to minimize the number of reallocations

```
var D: domain(int);
D.requestCapacity(n); // indicate n elements will be added
for i in 1..n {
  D += computeElement(i);
}
```

This Effort: Adds documentation for requestCapacity

Impact: requestCapacity now available to Chapel users

Next Steps: Support requestCapacity on other irregular domains

Distributed Associative Domains/Arrays (WIP)

Background: Distributed associative support has been missing

- associative domains require distinct distributions
  - e.g., Block doesn't make sense for a domain(string)
- a prototype was developed in 2009 but was only partially functional
  - concept: user provides a value→locale mapper object

This Effort: Clean up distributed associative implementation

```
proc Map.indexToLocaleIndex(ind, targetLocs: [] locale): int
var D: domain(string) dmapped
      new UserMapAssoc(idxType=string, mapper=new Map());
```

Impact: Supported multi-locale label propagation study

Next Steps:

- Promote to modules/distributions
- Improve performance

Other Domain Map Improvements

Other Domain Map Improvements

- **Block distributions support strided bounding box args**
  - previously, bounding boxes had to be non-strided

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

CRAY

THE SUPERCOMPUTER COMPANY