Hewlett Packard
Enterprise

Aggregated vs. Non-Aggregated Communication in Distributed Computing Settings

Brad Chamberlain

PNW PLSE 2024
May 7, 2024

Hewlett Packard Enterprise

Here’s something cool!

~~Aggregated vs. Non-Aggregated Communication in Distributed Computing Settings~~

Brad Chamberlain

PNW PLSE 2024
May 7, 2024

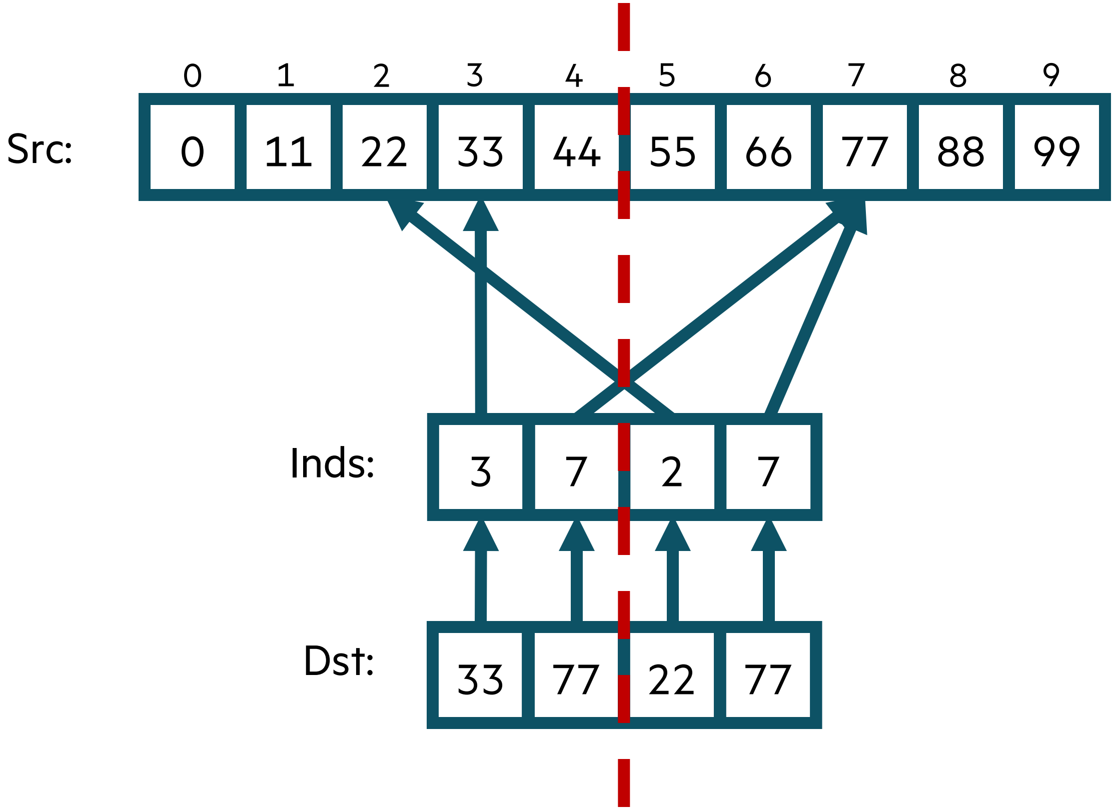
What is Chapel?

Chapel: A modern parallel programming language
- Portable & scalable
- Open-source & collaborative

Goals:
- Support general parallel programming
- Make parallel programming at scale far more productive

One definition of "productive":
- Support code similarly readable/writeable as Python
- While scaling competitively to thousands of compute nodes, millions of cores

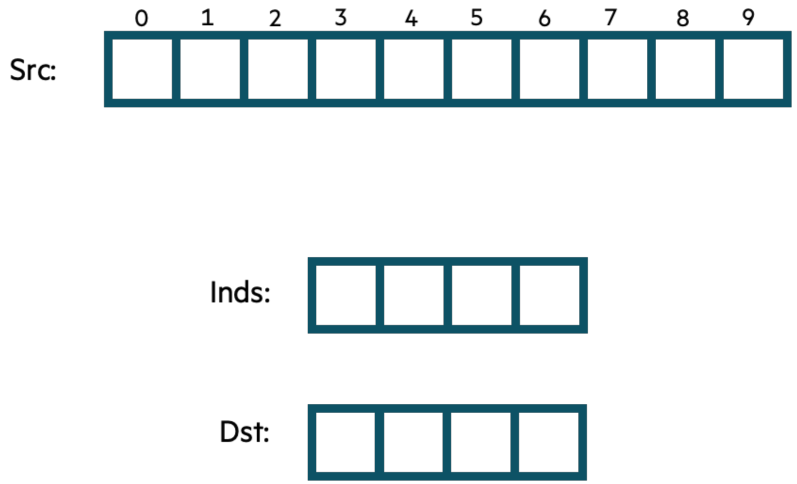
Bale IndexGather (IG): In Pictures



Bale IG in Chapel: Array Declarations

```
config const n = 10,
             m = 4;



var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
```

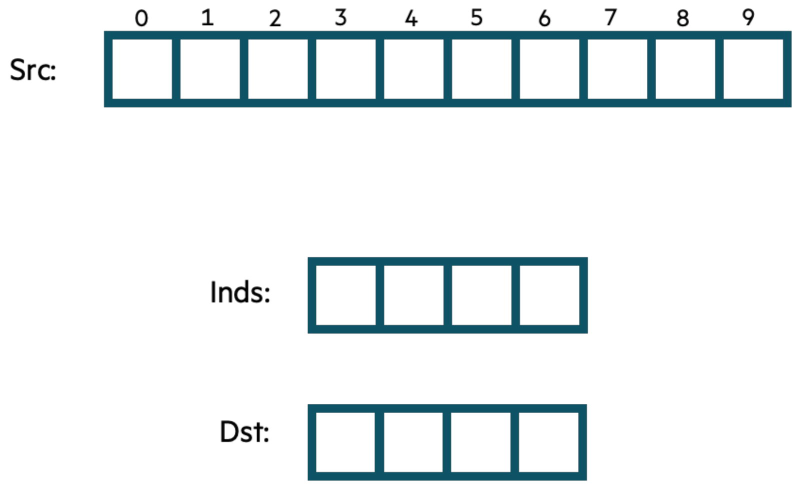
```
$
```



Bale IG in Chapel: Compiling

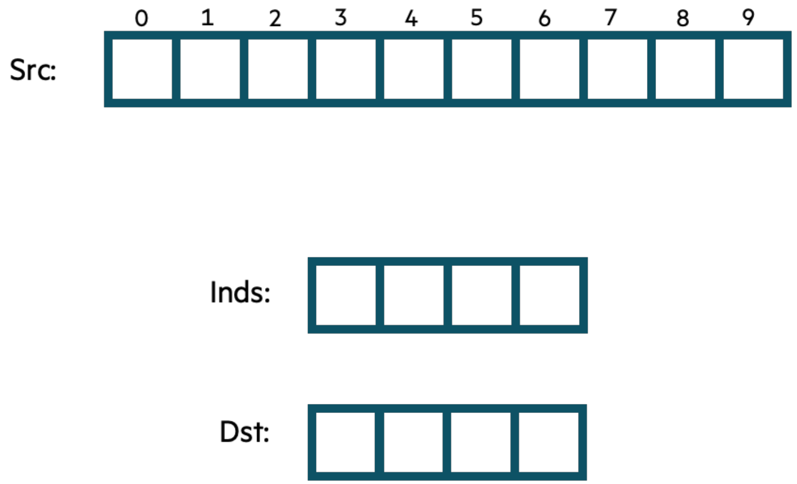
```
config const n = 10,
             m = 4;



var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
```

```
$ chpl bale-ig.chpl
$
```



Bale IG in Chapel: Executing

```
config const n = 10,
             m = 4;



var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
```

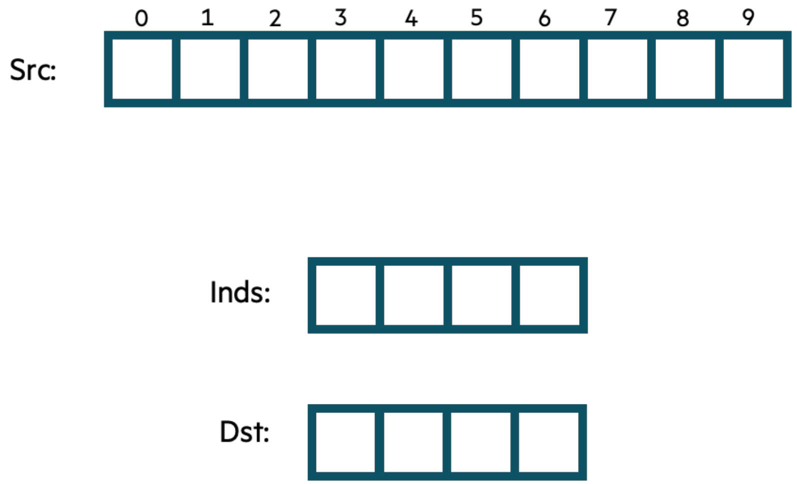



```
$ chpl bale-ig.chpl
$ ./bale-ig
$
```

Bale IG in Chapel: Executing, Overriding Configs

```
config const n = 10,
             m = 4;


var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
```

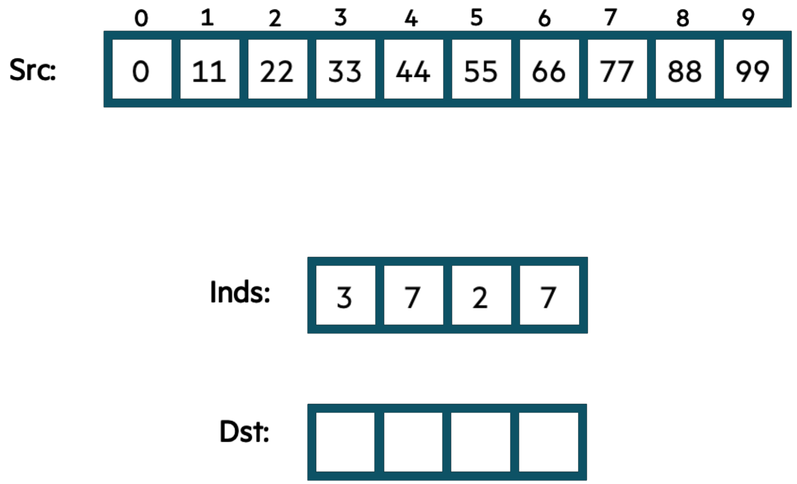
```
$ chpl bale-ig.chpl
$ ./bale-ig --n=1_000_000 --m=1_000_000
$
```



Bale IG in Chapel: Array Initialization

```
use Random;

config const n = 10,
             m = 4;




var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;

Src = [i in 0..<n] i*11;
fillRandom(Inds, min=0, max=n-1);
```

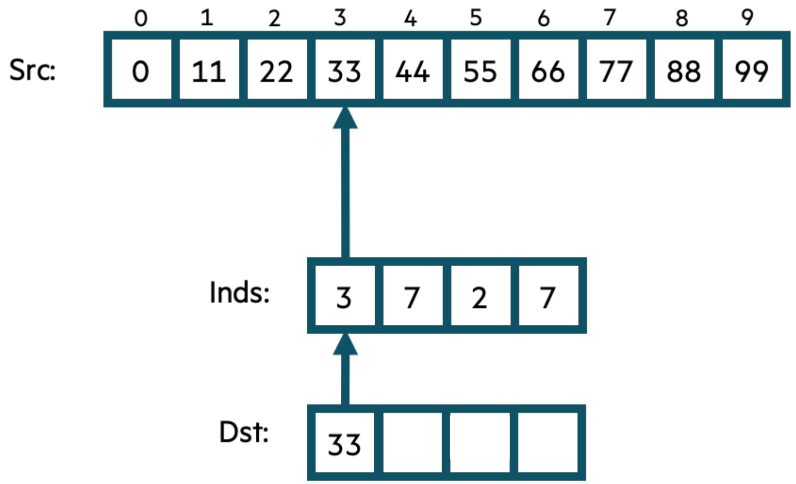
```
$ chpl bale-ig.chpl
$ ./bale-ig --n=1_000_000 --m=1_000_000
$
```



Bale IG in Chapel: Serial Version

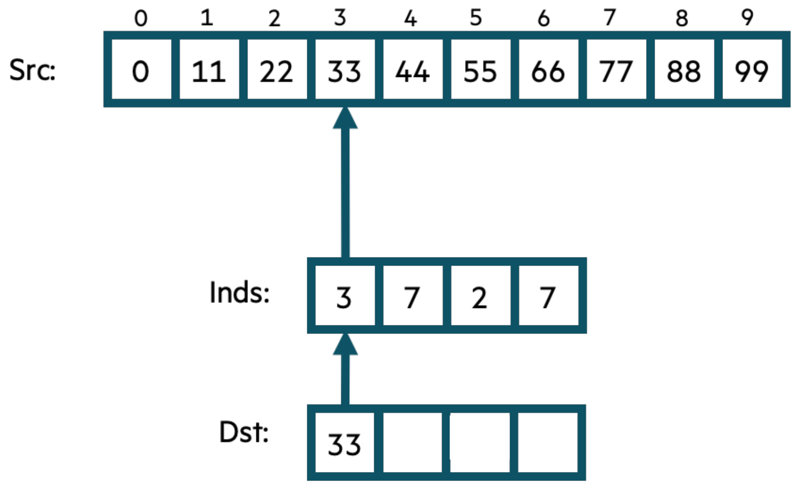
```
config const n = 10,
             m = 4;



var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
…
for i in 0..<m do
  Dst[i] = Src[Inds[i]];
```

```
$ chpl bale-ig.chpl
$ ./bale-ig --n=1_000_000 --m=1_000_000
$
```



Bale IG in Chapel: Serial Version using Zippered Iteration

```
config const n = 10,
             m = 4;



var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
…
for     (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

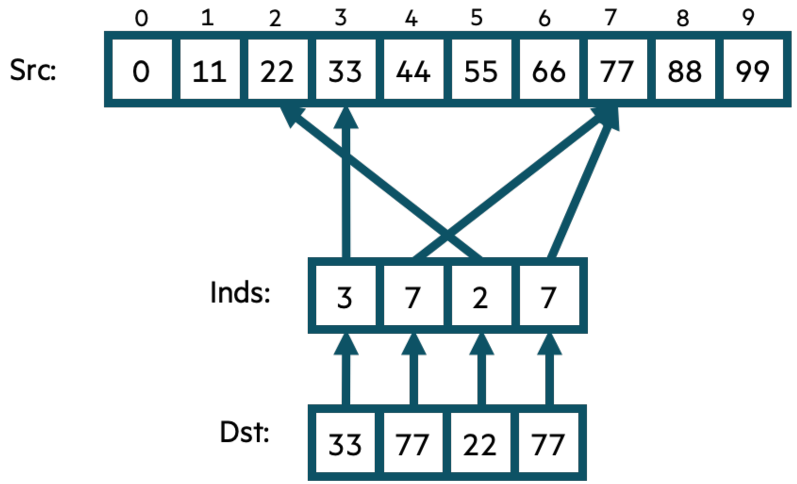



```
$ chpl bale-ig.chpl
$ ./bale-ig --n=1_000_000 --m=1_000_000
$
```

Bale IG in Chapel: Parallel Version (Multicore)

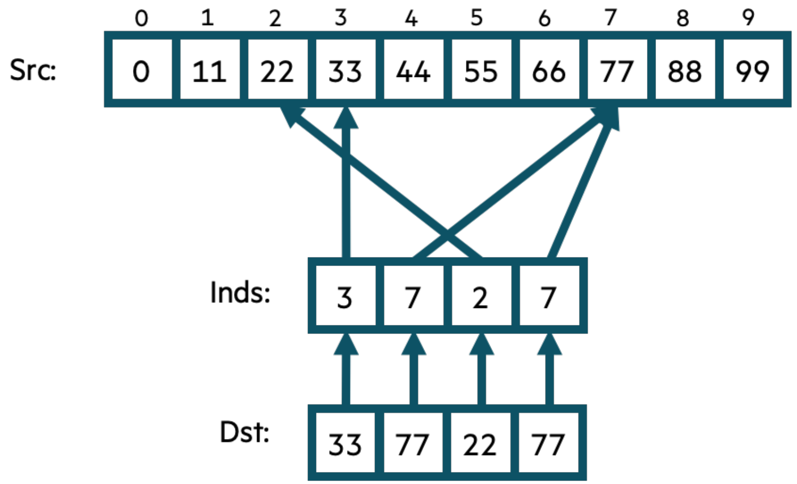
```
config const n = 10,
             m = 4;




var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

```
$ chpl bale-ig.chpl
$ ./bale-ig --n=1_000_000 --m=1_000_000
$
```



Bale IG in Chapel: Parallel Version (GPU)

```
config const n = 10,
             m = 4;



on here.gpus[0] {
  var Src: [0..<n] int,
      Inds, Dst: [0..<m] int;
  …
  forall (d, i) in zip(Dst, Inds) do
    d = Src[i];
}
```

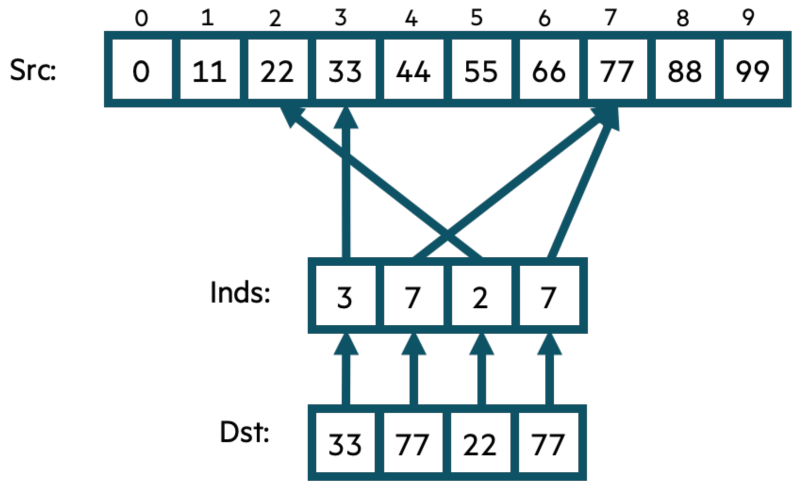



```
$ chpl bale-ig.chpl
$ ./bale-ig --n=1_000_000 --m=1_000_000
$
```

Bale IG in Chapel: Parallel Version (Multicore)

```
config const n = 10,
             m = 4;



var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

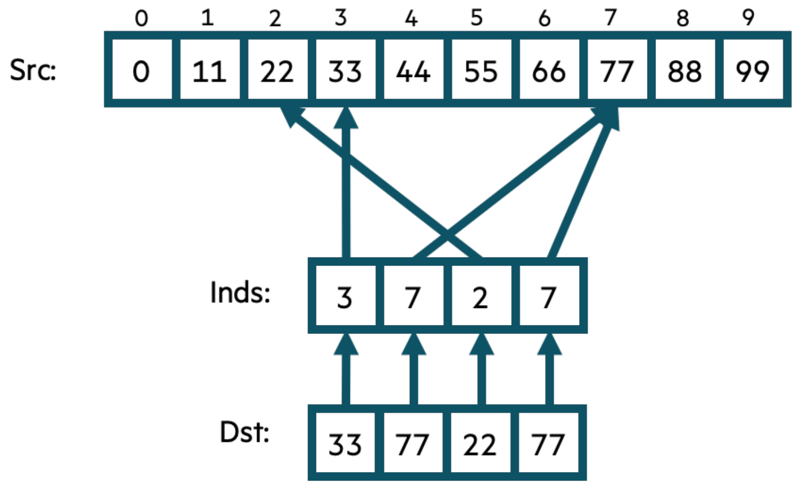
```
$ chpl bale-ig.chpl
$ ./bale-ig --n=1_000_000 --m=1_000_000
$
```



Bale IG in Chapel: Parallel Version (Multicore), with Named Domains

```
config const n = 10,
             m = 4;

const SrcInds = {0..<n},
      DstInds = {0..<m};

var Src: [SrcInds] int,
    Inds, Dst: [DstInds] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

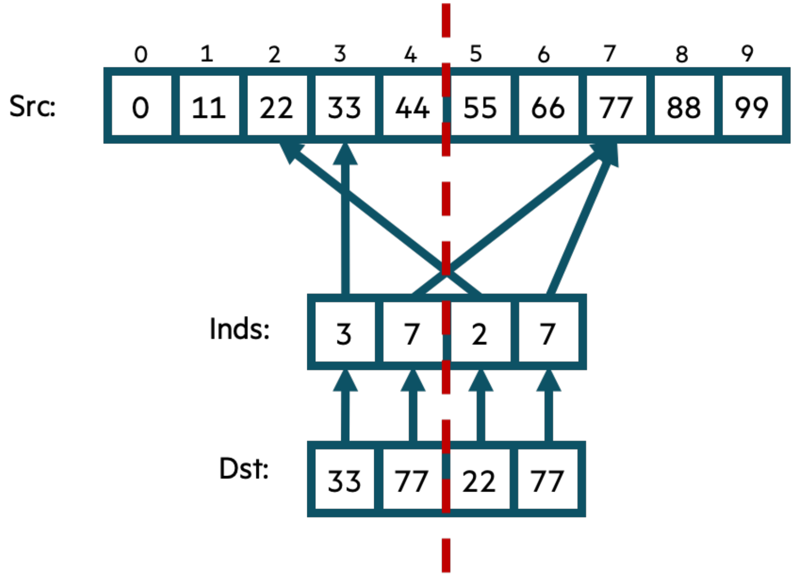
```
$ chpl bale-ig.chpl
$ ./bale-ig --n=1_000_000 --m=1_000_000
$
```



Bale IG in Chapel: Distributed, Simple Version

```
use BlockDist;

config const n = 10,
             m = 4;

const SrcInds = blockDist.createDomain(0..<n),
      DstInds = blockDist.createDomain(0..<m);

var Src: [SrcInds] int,
    Inds, Dst: [DstInds] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

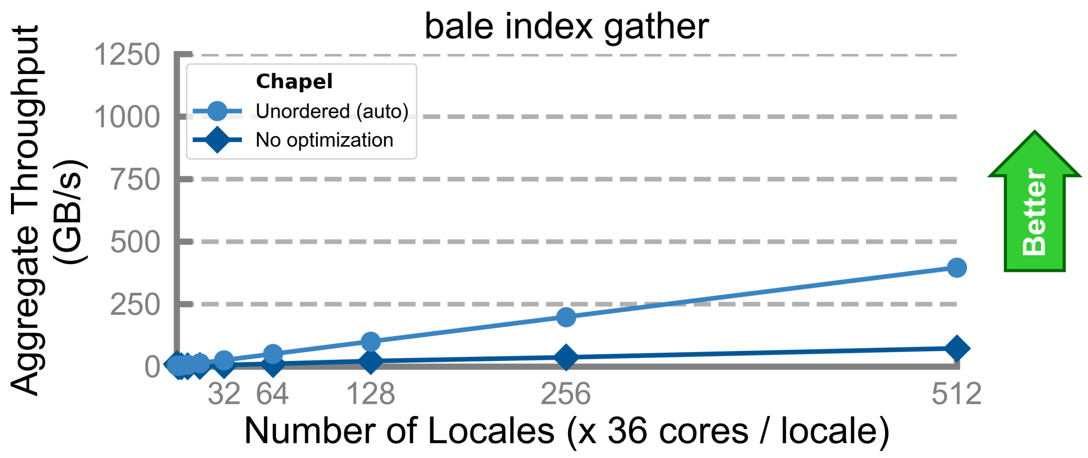
```
$ chpl bale-ig.chpl
$ ./bale-ig --n=1_000_000 --m=1_000_000 -nl 512
$
```



Bale IG in Chapel: Distributed, Simple Version

```
use BlockDist;

config const n = 10,
             m = 4;

const SrcInds = blockDist.createDomain(0..<n),
      DstInds = blockDist.createDomain(0..<m);

var Src: [SrcInds] int,
    Inds, Dst: [DstInds] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

```
$ chpl bale-ig.chpl
$ ./bale-ig --n=1_000_000 --m=1_000_000 -nl 512
$
```



Cray XC (Aries)

Bale IG in Chapel: Distributed, Simple Version

```
use BlockDist;

config const n = 10,
             m = 4;

const SrcInds = blockDist.createDomain(0..<n),
      DstInds = blockDist.createDomain(0..<m);

var Src: [SrcInds] int,
    Inds, Dst: [DstInds] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

Gets lowered roughly to…

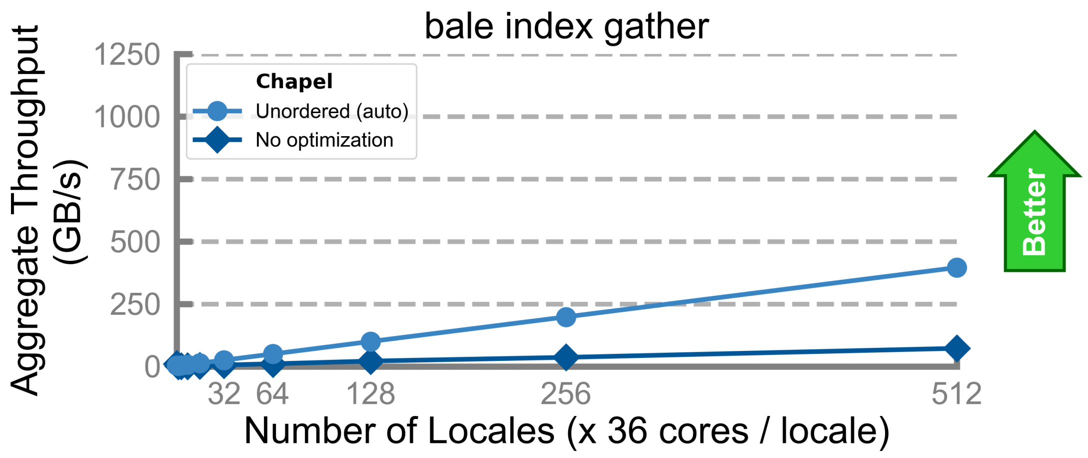
```
$ chp
$ ./b
$
```

```
coforall loc in Dst.targetLocales do on loc do
  coforall tid in 0..<here.maxTaskPar do
    for idx in myInds(loc, tid, …) do
      Dst[idx] = Src[Inds[idx]];
```

Create a task per compute node

Create a task per core on that node

Compute that task's gathers serially



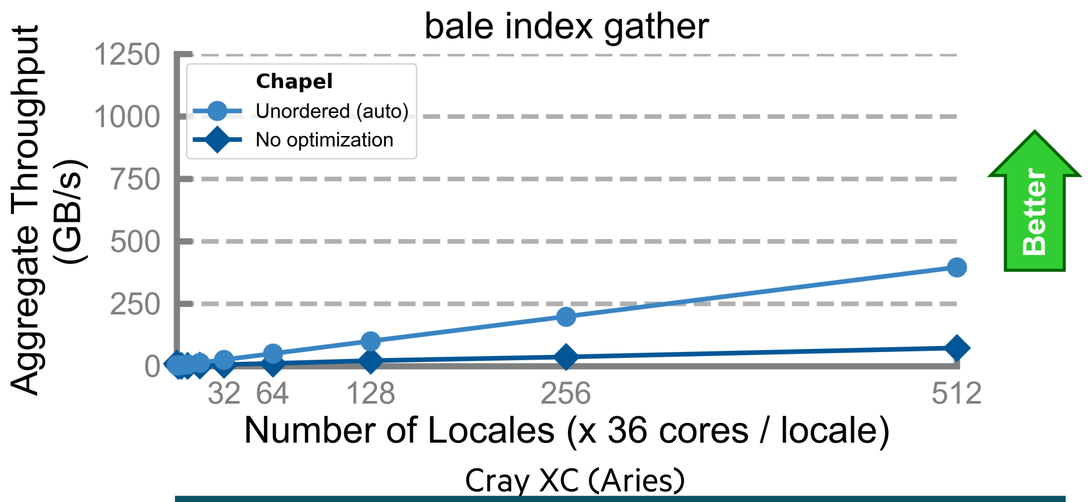
Cray XC (Aries)

Bale IG in Chapel: Distributed, Simple Version

```
use BlockDist;

config const n = 10,
             m = 4;

const SrcInds = blockDist.createDomain(0..<n),
      DstInds = blockDist.createDomain(0..<m);

var Src: [SrcInds] int,
    Inds, Dst: [DstInds] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

```
$ chp
$ ./b
$
```

```
coforall loc in Dst.targetLocales do on loc do
  coforall tid in 0..<here.maxTaskPar do
    for idx in myInds(loc, tid, …) do
      Dst[idx] = Src[Inds[idx]];
```



The user told us this loop was parallel, so why perform these high-latency ops serially?

So, our compiler rewrites the inner loop to perform them asynchronously

```
for idx in myInds(loc, tid, …) do
  asyncCopy(Dst[idx], Src[Inds[idx]]);
asyncCopyTaskFence();
```

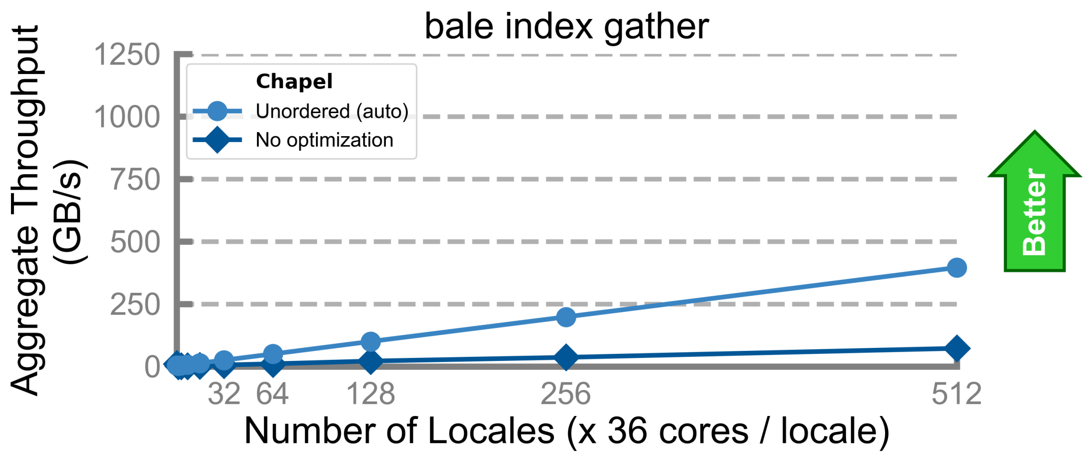
Bale IG in Chapel: Distributed, Simple Version

```
use BlockDist;

config const n = 10,
             m = 4;

const SrcInds = blockDist.createDomain(0..<n),
      DstInds = blockDist.createDomain(0..<m);

var Src: [SrcInds] int,
    Inds, Dst: [DstInds] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

```
$ chp
$ ./b
$
```

```
coforall loc in Dst.targetLocales do on loc do
  coforall tid in 0..<here.maxTaskPar do
    for idx in myInds(loc, tid, …) do
      Dst[idx] = Src[Inds[idx]];
```

```
for idx in myInds(loc, tid, …) do
  asyncCopy(Dst[idx], Src[Inds[idx]]);
asyncCopyTaskFence();
```

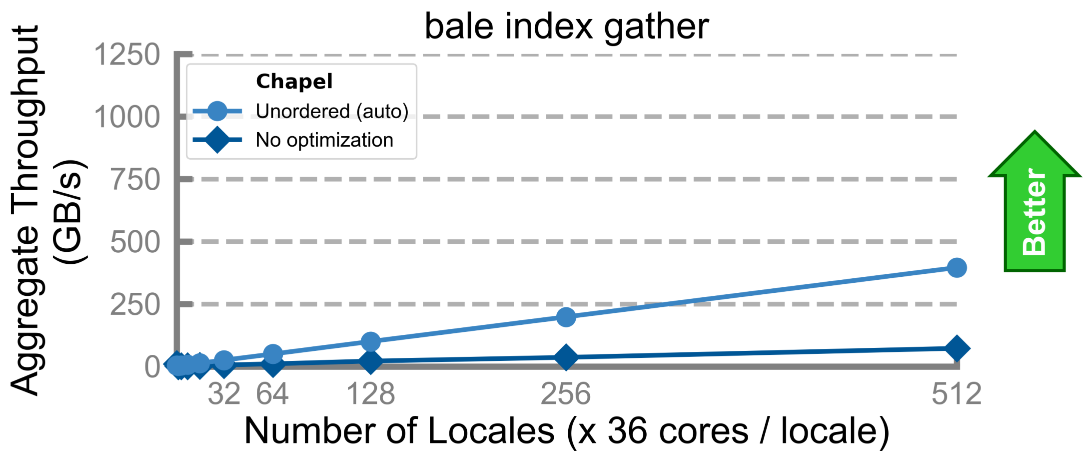


Cray XC (Aries)

Bale IG in Chapel: Distributed, Simple Version

```
use BlockDist;

config const n = 10,
             m = 4;

const SrcInds = blockDist.createDomain(0..<n),
      DstInds = blockDist.createDomain(0..<m);

var Src: [SrcInds] int,
    Inds, Dst: [DstInds] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

```
$ chpl bale-ig.chpl
$ ./bale-ig --n=1_000_000 --m=1_000_000 -nl 512
$
```

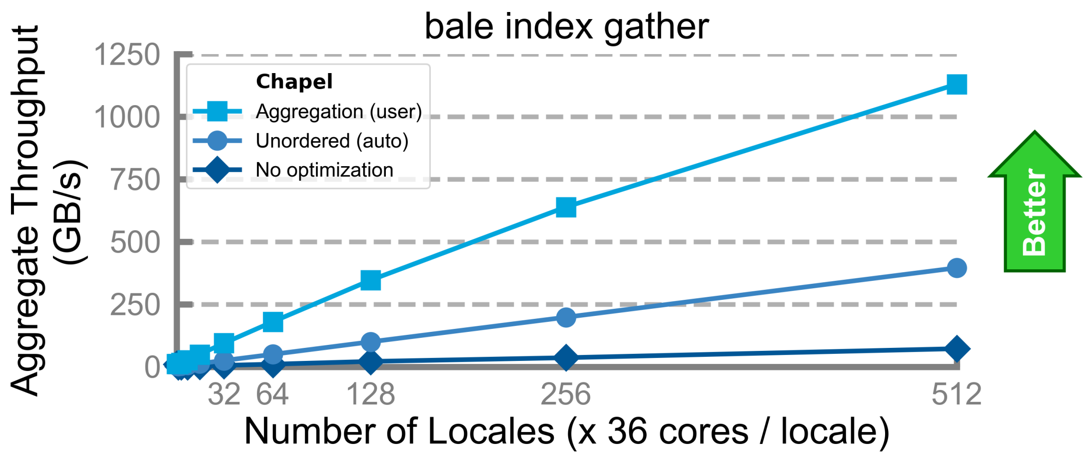


Cray XC (Aries)

Bale IG in Chapel: Distributed, Explicitly Aggregated Version

```
use BlockDist, CopyAggregation;

config const n = 10,
             m = 4;

const SrcInds = blockDist.createDomain(0..<n),
      DstInds = blockDist.createDomain(0..<m);

var Src: [SrcInds] int,
    Inds, Dst: [DstInds] int;
…
forall (d, i) in zip(Dst, Inds) with
 (var agg = new SrcAggregator(int)) do
  agg.copy(d, Src[i]);
```

```
$ chpl bale-ig.chpl
$ ./bale-ig --n=1_000_000 --m=1_000_000 -nl 512
$
```

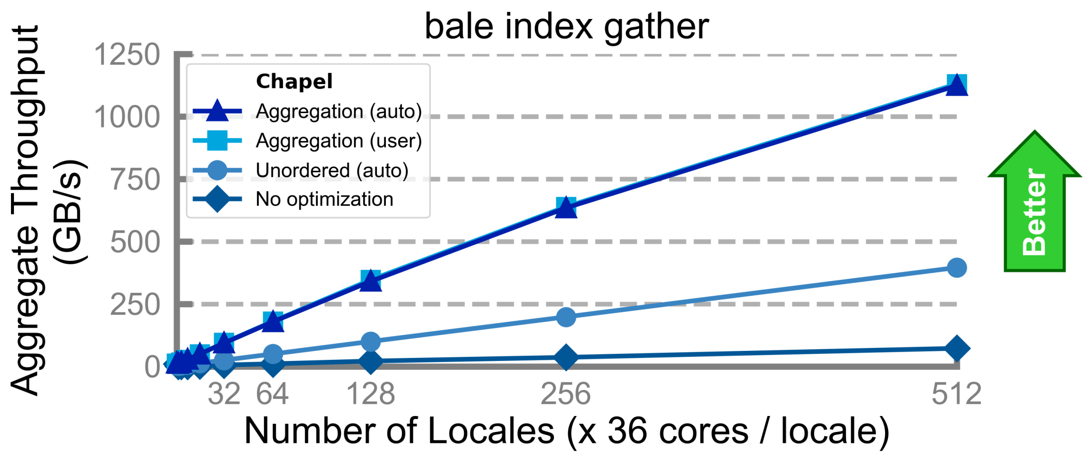


Cray XC (Aries)

Bale IG in Chapel: Distributed, Auto-Aggregated Version

```
use BlockDist;

config const n = 10,
             m = 4;

const SrcInds = blockDist.createDomain(0..<n),
      DstInds = blockDist.createDomain(0..<m);

var Src: [SrcInds] int,
    Inds, Dst: [DstInds] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

```
$ chpl bale-ig.chpl --auto-aggregation
$ ./bale-ig --n=1_000_000 --m=1_000_000 -nl 512
$
```



Cray XC (Aries)

Bale IG in Chapel vs. SHMEM on Cray XC

Chapel (Simple / Auto-Aggregated version)

```
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

Chapel (Explicitly Aggregated version)

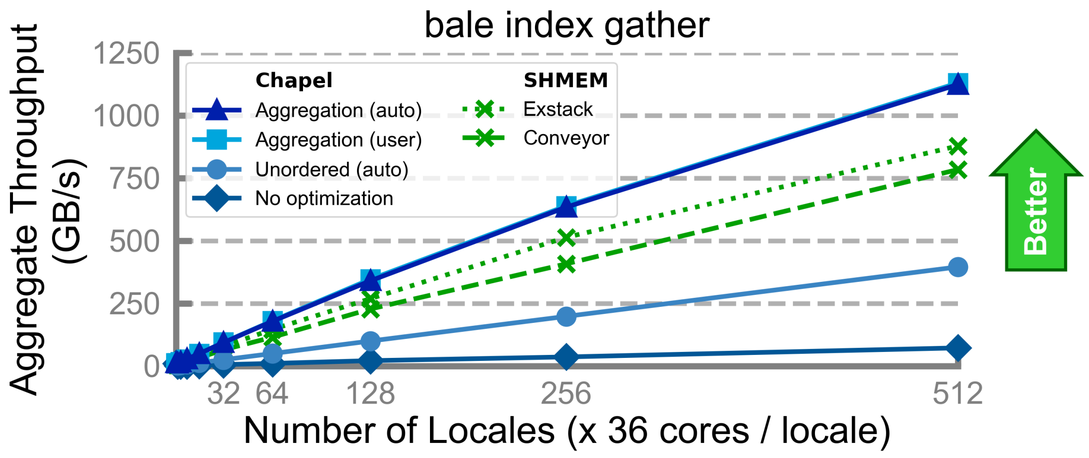
```
forall (d, i) in zip(Dst, Inds) with
 (var agg = new SrcAggregator(int)) do
  agg.copy(d, Src[i]);
```

SHMEM (Exstack version)

```
i=0;
while( exstack_proceed(ex, (i==l_num_req)) ) {
  i0 = i;
  while(i < l_num_req) {
    l_indx = pckindx[i] >> 16;
    pe  = pckindx[i] & 0xffff;
    if(!exstack_push(ex, &l_indx, pe))
      break;
    i++;
  }

  exstack_exchange(ex);

  while(exstack_pop(ex, &idx , &fromth)) {
    idx  = ltable[idx];
    exstack_push(ex, &idx, fromth);
  }
  lgp_barrier();
  exstack_exchange(ex);

  for(j=i0; j<i; j++) {
    fromth = pckindx[j] & 0xffff;
    exstack_pop_thread(ex, &idx, (uint64_t)fromth);
    tgt[j] = idx;
  }
  lgp_barrier();
}
```

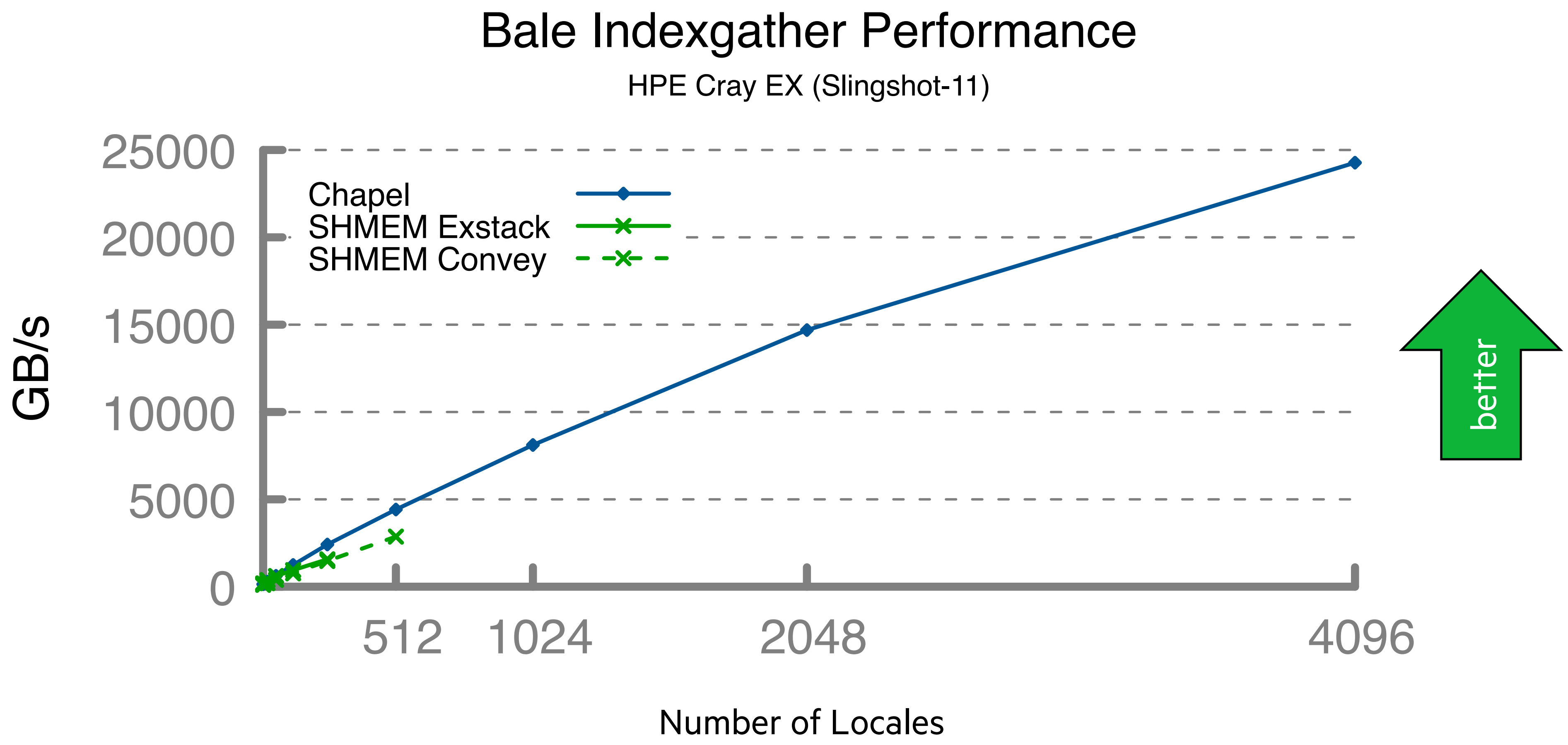
SHMEM (Conveyors version)

```
i = 0;
while (more = convey_advance(requests, (i == l_num_req)),
       more | convey_advance(replies, !more)) {

  for (; i < l_num_req; i++) {
    pkg.idx = i;
    pkg.val = pckindx[i] >> 16;
    pe = pckindx[i] & 0xffff;
    if (! convey_push(requests, &pkg, pe))
      break;
  }

  while (convey_pull(requests, ptr, &from) == convey_OK) {
    pkg.idx = ptr->idx;
    pkg.val = ltable[ptr->val];
    if (! convey_push(replies, &pkg, from)) {
      convey_unpull(requests);
      break;
    }
  }

  while (convey_pull(replies, ptr, NULL) == convey_OK)
    tgt[ptr->idx] = ptr->val;
}
```

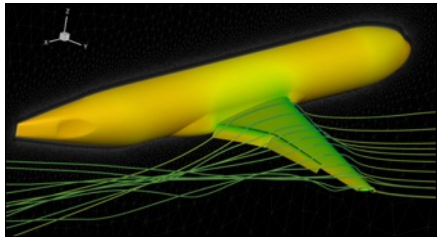


Cray XC (Aries)

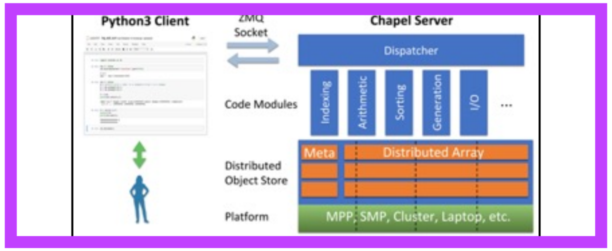
Bale IG in Chapel vs. SHMEM on HPE Cray EX



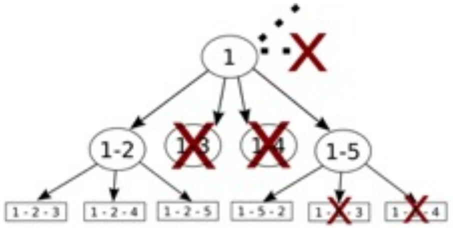
Applications of Chapel



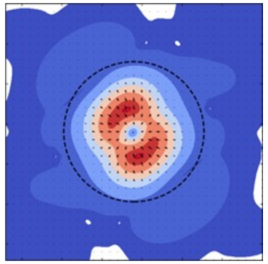
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



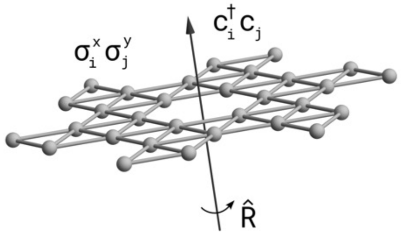
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



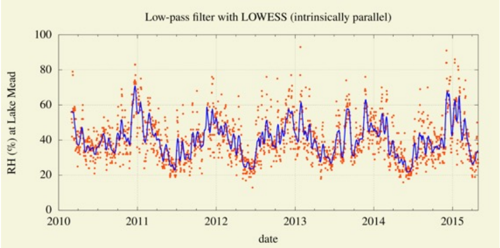
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



ChplUltra: Simulating Ultralight Dark Matter
Nikhil Padmanabhan, J. Luna Zagorac, et al.
Yale University et al.

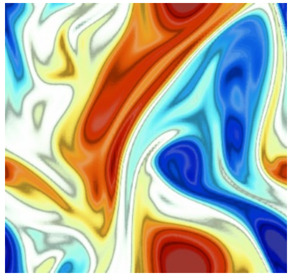


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

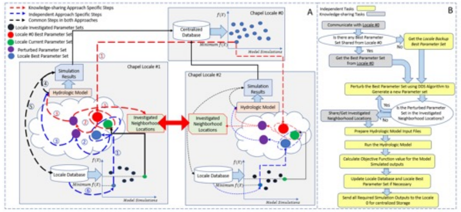


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

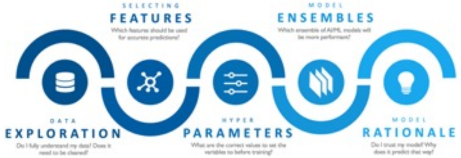
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



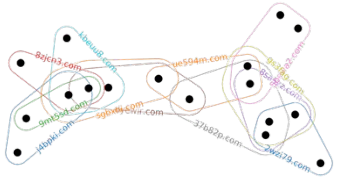
ChapQG: Layered Quasigeostrophic CFD
Ian Grooms and Scott Bachman
University of Colorado, Boulder et al.



Chapel-based Hydrological Model Calibration
Marjan Asgari et al.
University of Guelph



CrayAI HyperParameter Optimization (HPO)
Ben Albrecht et al.
Cray Inc. / HPE



CHGL: Chapel Hypergraph Library
Louis Jenkins, Cliff Joslyn, Jesun Firoz, et al.
PNNL

?

Your Application Here?

(images provided by their respective teams and used with permission)

Arkouda Argsort Performance Milestones

HPE Apollo (May 2021)

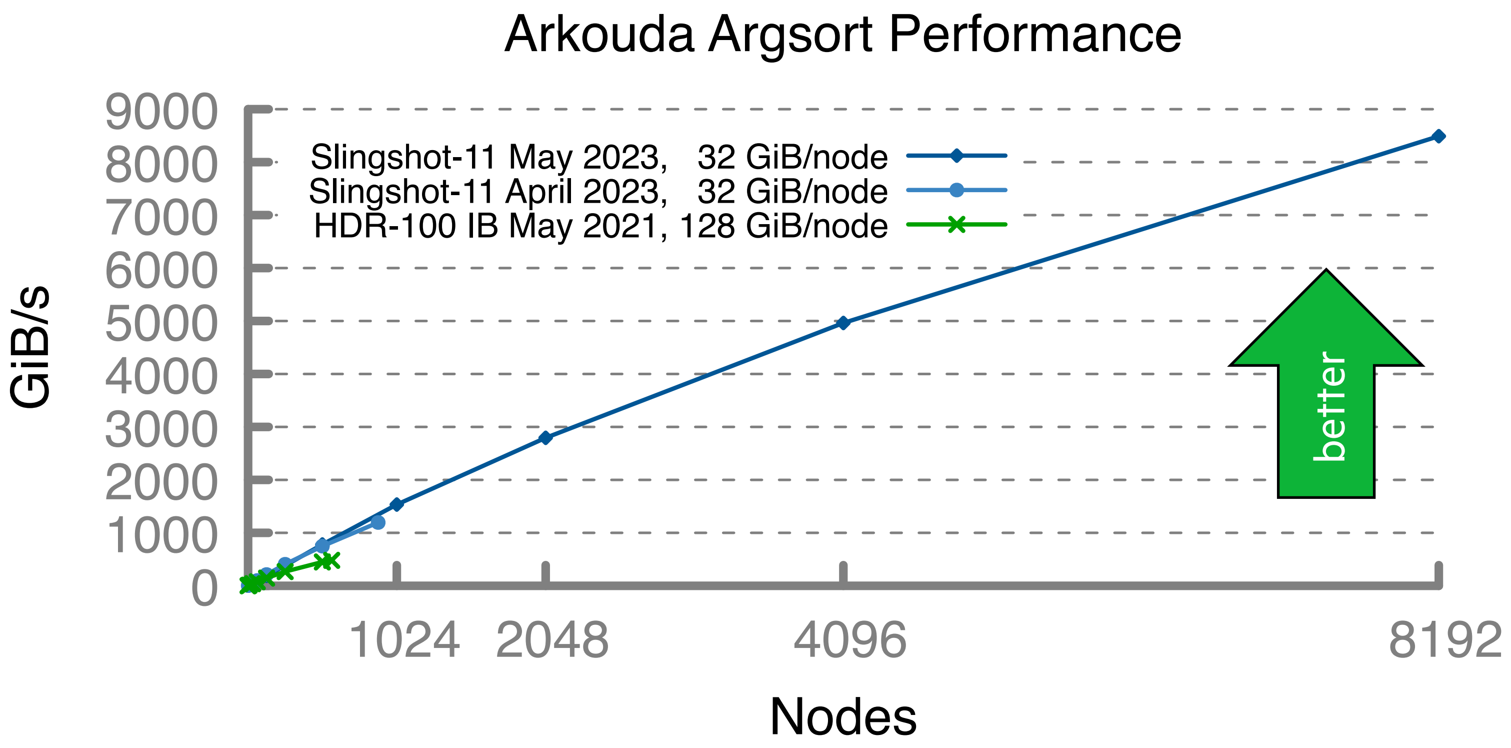
- HDR-100 Infiniband network (100 Gb/s)
- 576 compute nodes
- 72 TiB of 8-byte values
- ~480 GiB/s (~150 seconds)

HPE Cray EX (April 2023)

- Slingshot-11 network (200 Gb/s)
- 896 compute nodes
- 28 TiB of 8-byte values
- ~1200 GiB/s (~24 seconds)

HPE Cray EX (May 2023)

- Slingshot-11 network (200 Gb/s)
- 8192 compute nodes
- 256 TiB of 8-byte values
- ~8500 GiB/s (~31 seconds)



This performance is enabled by aggregators, as in the Bale IG example

Summary: What this example illustrates

For scalable parallel computing, good language design can…

…**provide built-in abstractions** to simplify the expression of parallel operations

– e.g., global namespace, parallel loops and iterators

…**more clearly represent parallel computations** compared to standard approaches

– e.g., MPI, SHMEM, CUDA, HIP, SYCL, OpenMP, OpenCL, OpenACC, Kokkos, RAJA, …

…permit users to **create new abstractions** supporting performance and/or clean code

– e.g., per-task aggregators

…**enable new optimization opportunities** by expressing parallelism and locality clearly

– e.g., asynchronous operations, auto-aggregation of communication

…**support excellent performance and scalability**

– e.g., to thousands of nodes and hundreds of thousands of cores

Hewlett Packard Enterprise

The Value of Languages in Parallel Computing: Aggregated vs. Non-Aggregated Communication in Distributed Computing Settings

Brad Chamberlain

PNW PLSE 2024
May 7, 2024

Other Chapel News

- Chapel 2.0 was released in March 2024

- We're currently seeking new users and partners
  - We're happy to speak at schools and companies who'd like to hear more

- ChapelCon'24 is coming up June 5–7
  - tutorials, coding sessions, community talks
  - online and free
  - https://chapel-lang.org/ChapelCon24.html

Chapel Language Blog

About Chapel Website Featured Series Tags Authors All Posts

Chapel 2.0: Scalable and Productive Computing for All

Posted on March 21, 2024.

Tags: Chapel 2.0 Release Announcements

By: Daniel Fedorin

Table of Contents

Today, the Chapel team is excited to announce the release of Chapel version 2.0. After years of hard work and continuous improvements, Chapel shines as an enjoyable and productive programming language for distributed and parallel computing. People with diverse application goals are leveraging Chapel to quickly develop fast and scalable software, including physical simulations, massive data and graph analytics, portions of machine learning pipelines, and more. The 2.0 release brings stability guarantees to Chapel's battle-tested features, making it possible to write performant and elegant code for laptops, GPU workstations, and supercomputers with confidence and convenience.

In addition to numerous usability and performance improvements — including many over the previous release candidate — the 2.0 release of Chapel is **stable**: the core language and library features are designed to be backwards-compatible from here on. As Chapel continues to grow and evolve, additions or changes to the language should not require adjusting any existing code.

https://chapel-lang.org/blog/posts/announcing-chapel-2.0/

The Chapel Team at HPE

Chapel Resources

Chapel homepage: https://chapel-lang.org

- (points to all other resources)

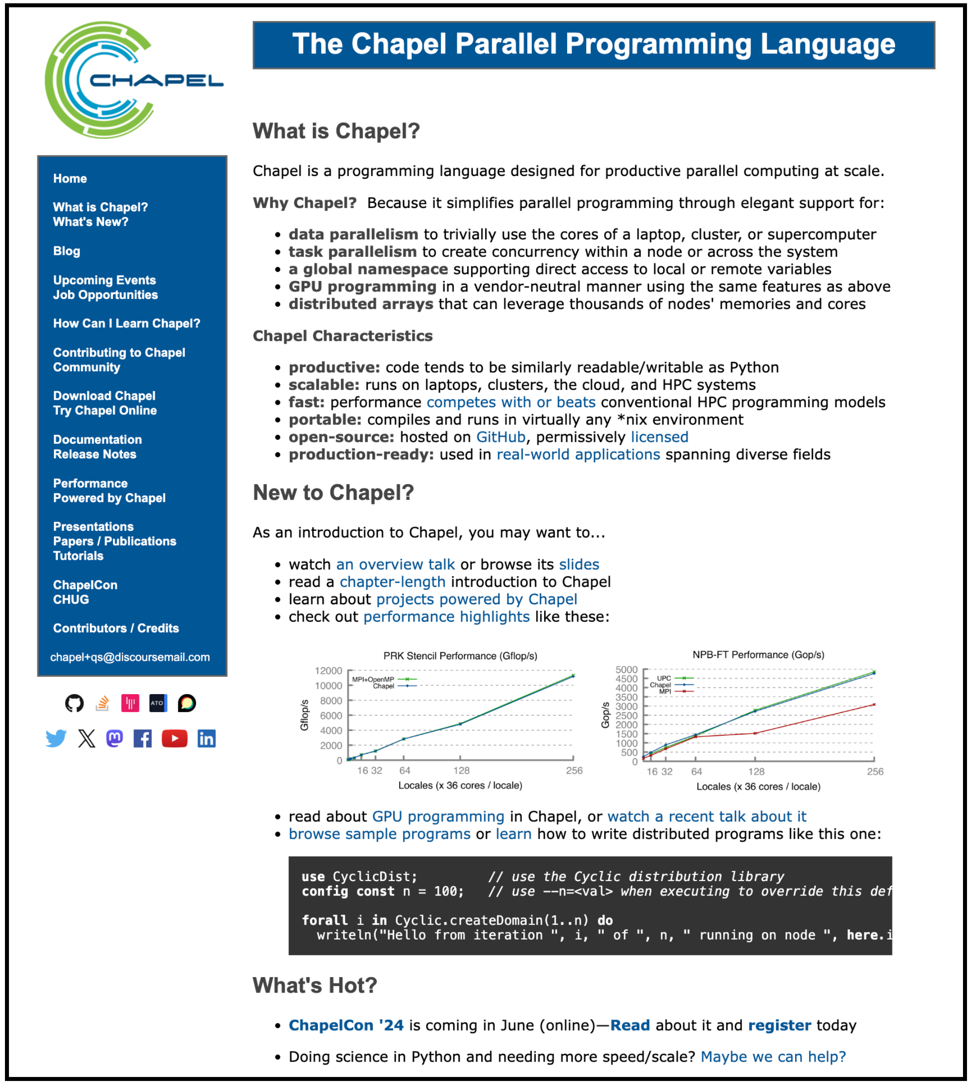
Blog: https://chapel-lang.org/blog/

Social Media:

- Facebook: @ChapelLanguage
- LinkedIn: https://www.linkedin.com/company/chapel-programming-language/
- Mastadon: @ChapelProgrammingLanguage
- X / Twitter: @ChapelLanguage
- YouTube: @ChapelLanguage

Community Discussion / Support:

- Discourse: https://chapel.discourse.group/
- Gitter: https://gitter.im/chapel-lang/chapel
- Stack Overflow: https://stackoverflow.com/questions/tagged/chapel
- GitHub Issues: https://github.com/chapel-lang/chapel/issues



Where can I use Chapel?

Online:

- GitHub Codespaces
- Attempt This Online (ATO)

Laptops/Desktops:

- Linux/UNIX
- Mac OS X
- Windows (w/ WSL)

Systems:

- Commodity clusters
- HPE/Cray supercomputers, such as:
  - Frontier
  - Perlmutter
  - Piz Daint
  - Polaris
  - …
- Other vendors' supercomputers

Cloud:

- AWS
- Microsoft Azure
- Google Cloud(?)

CPUs:

- Intel
- AMD
- Arm (M1/M2, Graviton, A64FX, Raspberry Pi, …)

GPUs:

- NVIDIA
- AMD

Networks:

- Slingshot
- Aries/Gemini
- InfiniBand
- EFA
- Ethernet

How can I get Chapel?

- Source releases
- HPE modules
- Homebrew
- Docker
- Spack (WIP)
- apt/rpm (WIP)
- AMIs (WIP)

How is Chapel supported?

- GitHub issues
- Discourse
- Gitter
- Stack Overflow
- Email
- pair-programming sessions

Thank you

https://chapel-lang.org
@ChapelLanguage