Aggregated vs. Non-Aggregated Communication in Distributed Computing Settings

Brad Chamberlain
PNW PLSE 2024
May 7, 2024
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

Src:

| 0 | 11 | 22 | 33 | 44 | 55 | 66 | 77 | 88 | 99 |

Inds:

| 3 | 7 | 2 | 7 |

Dst:

| 33 | 77 | 22 | 77 |
config const n = 10,
m = 4;

var Src: [0..<n] int,
    Inds, Dst: [0..<m] int;
Bale IG in Chapel: Compiling

```chapel
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

```chapel
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

```chapel
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);
```

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

```chapel
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
$ 
```
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
$
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)

```chapel
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

```chapel
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**

```chapel
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];
```

```bash
$ chpl bale-ig.chpl
$ ./bale-ig --n=1_000_000 --m=1_000_000 -nl 512
$ 
```
Bale IG in Chapel: Distributed, Simple Version

```chapel
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

```chapel
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];
```

Create a task per compute node
Create a task per core on that node
Compute that task’s gathers serially

Gets lowered roughly to...

```chapel
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]];
```

Cray XC (Aries)
Bale IG in Chapel: Distributed, Simple Version

```chapel
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];
```

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.

```chapel
for idx in myInds(loc, tid, ...) do
    Dst[idx] = Src[Inds[idx]];
```
Bale IG in Chapel: Distributed, Simple Version

```chapel
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];

coforall loc in Dst.targetLocales do on loc do
cforall tid in 0..<here.maxTaskPar do
    forall 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
$ ```
Bale IG in Chapel: Distributed, Explicitly Aggregated Version

```chapel
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

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

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

SHMEM (Exstack version)

```c
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, (unsigned int)l_indx, pe))
      break;
    i++;
  }
  exstack_exchange(ex);
  while (exstack_popexes(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_threadexit(ex, (uint64_t)fromth);
    tgt[j] = idx;
  }
  lgp_barrier();
}
```

SHMEM (Conveyors version)

```c
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;
  }
```
Bale IG in Chapel vs. SHMEM on HPE Cray EX

Bale Indexgather Performance
HPE Cray EX (Slingshot-11)

Number of Locales

GB/s

Chapel
SHMEM Exstack
SHMEM Convey

better
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
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

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

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

Summary: What this example illustrates
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](https://chapel-lang.org/ChapelCon24.html)
The Chapel Team at HPE
Chapel Resources

**Chapel homepage:** [https://chapel-lang.org](https://chapel-lang.org)
- (points to all other resources)

**Blog:** [https://chapel-lang.org/blog/](https://chapel-lang.org/blog/)

**Social Media:**
- Facebook: [@ChapelLanguage](https://www.facebook.com/ChapelLanguage)
- LinkedIn: [https://www.linkedin.com/company/chapel-programming-language/](https://www.linkedin.com/company/chapel-programming-language/)
- Mastadon: [@ChapelProgrammingLanguage](https://mastodon.social/@ChapelProgrammingLanguage)
- X / Twitter: [@ChapelLanguage](https://twitter.com/ChapelLanguage)
- YouTube: [@ChapelLanguage](https://youtube.com/ChapelLanguage)

**Community Discussion / Support:**
- Discourse: [https://chapel.discourse.group/](https://chapel.discourse.group/)
- Gitter: [https://gitter.im/chapel-lang/chapel](https://gitter.im/chapel-lang/chapel)
- Stack Overflow: [https://stackoverflow.com/questions/tagged/chapel](https://stackoverflow.com/questions/tagged/chapel)
- GitHub Issues: [https://github.com/chapel-lang/chapel/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