PRODUCTIVE PARALLEL PROGRAMMING USING CHAPEL

Nordic-RSE Seminar Series
November 30, 2022
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
Imagine having a programming language for parallel computing that was as...

...**programmable** as Python

...yet also as...

...**fast** as Fortran

...**scalable** as MPI

...**GPU-ready** as CUDA/OpenMP/OpenCL/OpenACC/...

...**portable** as C

...**fun** as [your favorite programming language]

**This is our motivation for Chapel**
OUTLINE

• What is Chapel, and Why?
• Chapel Characteristics
• Chapel Benchmarks & Apps
• Chapel Features
• Wrap-up
CHAPEL CHARACTERISTICS
WHAT DO CHAPEL PROGRAMS LOOK LIKE?

**helloTaskPar.chpl**: print a message from each core in the system

```chapel
coforall loc in Locales {
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      printf("Hello from task %n of %n on %s\n", tid, numTasks, here.name);
  }
}
```

**fillArray.chpl**: declare and parallel-initialize a distributed array

```chapel
use CyclicDist;
config const n = 1000;
const D = {1..n, 1..n}
dmapped Cyclic(startIdx = (1,1));
var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```bash
> chpl helloTaskPar.chpl
> ./helloTaskPar --numLocales=4
Hello from task 1 of 4 on n1032
Hello from task 4 of 4 on n1032
Hello from task 1 of 4 on n1034
Hello from task 2 of 4 on n1032
Hello from task 1 of 4 on n1033
Hello from task 3 of 4 on n1034
...
```

```bash
> chpl fillArray.chpl
> ./fillArray --n=5 --numLocales=4
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```
FIVE KEY CHARACTERISTICS OF CHAPEL

1. **compiled:** to generate the best performance possible
2. **statically typed:** to avoid simple errors after hours of execution
3. **interoperable:** with C, Fortran, Python, …
4. **portable:** runs on laptops, clusters, the cloud, supercomputers
5. **open-source:** to reduce barriers to adoption and leverage community contributions
Q: What is provided in a Chapel release?

A: Chapel releases contain...

...the Chapel compiler (`chpl`): translates Chapel source code into optimized executables
...runtime libraries: maps Chapel programs to a system’s capabilities (e.g., processors, network, memory, ...)
...library modules: provide standard algorithms, data types, capabilities, ...
...documentation (also available online at: https://chapel-lang.org/docs/)
...sample programs: primers, benchmarks, etc.

Q: How often is Chapel released? And in what formats?

A: Chapel is released quarterly (March, June, Sept, Dec) in a variety formats:

- open-source tarballs on GitHub
- as a homebrew formula and bottle for Mac and Linux
- as a Docker image
- as a module on HPE Cray systems
Our core team consists of:

- 16 developers + 1 starting early 2023
- 1 visiting scholar
- 1 manager
- 1 tech lead
- 1 project lead (technical manager)
- 1/n director

see: https://chapel-lang.org/contributors.html
CHAPEL BENCHMARKS AND APPLICATIONS
FOR DESKTOP BENCHMARKS, CHAPEL IS COMPACT AND FAST

[plot generated by summarizing data from https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html as of May 10, 2022]
FOR DESKTOP BENCHMARKS, CHAPEL IS COMPACT AND FAST (ZOOMED)

[plot generated by summarizing data from https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html as of May 10, 2022]
FOR HPC BENCHMARKS, CHAPEL TENDS TO BE CONCISE, CLEAR, AND COMPETITIVE

STREAM TRIAD: C + MPI + OPENMP

use BlockDist;
config const m = 1000, alpha = 3.0;
const Dom = {1..m} dmapped ...
var A, B, C: [Dom] real;
B = 2.0;
C = 1.0;
A = B + alpha * C;

HPCC RA: MPI KERNEL

forall (_, r) in zip(Updates, RAStream()) do
t[r & indexMask].xor(r);

STREAM Performance (GB/s)

RA Performance (GUPS)
TWO FLAGSHIP CHAPEL APPLICATIONS

CHAMPS: 3D Unstructured Computational Fluid Dynamics (CFD)

Arkouda: Interactive Data Analytics at Supercomputing Scale
What is it?

- 3D unstructured CFD framework for airplane simulation
- ~85k lines of Chapel written from scratch in ~3 years

Who wrote it?

- Professor Éric Laurendeau’s students + postdocs at Polytechnique Montreal
- Not open-source, but available on request

Why Chapel?

- performance and scalability competitive with MPI + C++
- students found it far more productive to use
HPC Lessons From 30 Years of Practice in CFD Towards Aircraft Design and Analysis (June 4, 2021)

“To show you what Chapel did in our lab... [our previous framework] ended up 120k lines. And my students said, ‘We can't handle it anymore. It’s too complex, we lost track of everything.’ And today, they went from 120k lines to 48k lines, so 3x less.

But the code is not 2D, it’s 3D. And it’s not structured, it’s unstructured, which is way more complex. And it’s multi-physics... So, I've got industrial-type code in 48k lines.”

 “[Chapel] promotes the programming efficiency ... We ask students at the master’s degree to do stuff that would take 2 years and they do it in 3 months. So, if you want to take a summer internship and you say, ‘program a new turbulence model,’ well they manage. And before, it was impossible to do.”

“So, for me, this is like the proof of the benefit of Chapel, plus the smiles I have on my students everyday in the lab because they love Chapel as well. So that’s the key, that’s the takeaway.”

• Talk available online: https://youtu.be/wD-a_KyB8al?t=1904 (hyperlink jumps to the section quoted here)

(images provided by the CHAMPS team and used with permission)
CHAMPS HIGHLIGHTS

Community Activities:

- Team participated in the 7th AIAA High-lift Prediction Workshop and 1st AIAA Ice Prediction Workshop
  - Generating comparable results to high-profile sites: Boeing, Lockheed Martin, NASA, JAXA, Georgia Tech, ...
- Five papers published this past summer at 2022 AIAA Aviation
- While on sabbatical, Éric has presented CHAMPS and Chapel at ONERA, DLR, Université de Strasbourg, ...
- Student presentations at CASI/IASC Aero 21 Conference and to CFD Society of Canada (CFDSC)

Application - Fourth AIAA High Lift Prediction Workshop

Fourth AIAA High Lift Prediction Workshop

- Case 1b: Grid refinement study for a constant angle of attack of 7.06°.
- Results are in line with state of the art RANS solver.

Application - First AIAA Ice Prediction Workshop

First AIAA Ice Prediction Workshop

- Case 241 (left): Time-ice prediction on small NACA23012 airfoil (2D, low temp.).
- Case 363 (right): Case ice prediction on NACA6012 swept wing (2D, warmer temp.).

Numerical Verification

Fifth Drag Prediction Workshop (DPW)

- The pressure drag convergence of CHAMPS is similar to the workshop results.

(slide images taken from Éric Laurendeau’s SIAM PP22 talk, A Case Study on the Impact of Chapel within an Academic Computational Aerodynamic Laboratory, with permission)
TWO FLAGSHIP CHAPEL APPLICATIONS

**CHAMPS**: 3D Unstructured Computational Fluid Dynamics (CFD)

**Arkouda**: Interactive Data Analytics at Supercomputing Scale
DATA SCIENCE IN PYTHON AT SCALE?

Motivation: Say you’ve got...

...HPC-scale data science problems to solve

...a bunch of Python programmers

...access to HPC systems

How will you leverage your Python programmers to get your work done?
ARKOUDA’S HIGH-LEVEL APPROACH

Arkouda Client
(written in Python)

Arkouda Server
(written in Chapel)

User writes Python code in Jupyter, making familiar NumPy/Pandas calls
ARKOUDA SUMMARY

What is it?
- A Python library supporting a key subset of NumPy and Pandas for Data Science
  - Uses a Python-client/Chapel-server model to get scalability and performance
  - Computes massive-scale results (multi-TB-scale arrays) within the human thought loop (seconds to a few minutes)
- ~25k lines of Chapel, written since 2019

Who wrote it?
- Mike Merrill, Bill Reus, et al., US DoD
- Open-source: https://github.com/Bears-R-Us/arkouda

Why Chapel?
- high-level language with performance and scalability
- close to Pythonic
  - enabled writing Arkouda rapidly
  - doesn’t repel Python users who look under the hood
- ports from laptop to supercomputer
ARKOUDA PERFORMANCE COMPARED TO NUMPY

<table>
<thead>
<tr>
<th>benchmark</th>
<th>NumPy 0.75 GB</th>
<th>Arkouda (serial) 0.75 GB 1 core, 1 node</th>
<th>Arkouda (parallel) 0.75 GB 36 cores x 1 node</th>
<th>Arkouda (distributed) 384 GB 36 cores x 512 nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>argsort</td>
<td>0.03 GiB/s</td>
<td>0.05 GiB/s</td>
<td>0.50 GiB/s</td>
<td>55.12 GiB/s</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>1.66x</td>
<td>16.7x</td>
<td>1837.3x</td>
</tr>
<tr>
<td>coargsort</td>
<td>0.03 GiB/s</td>
<td>0.07 GiB/s</td>
<td>0.50 GiB/s</td>
<td>29.54 GiB/s</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>2.3x</td>
<td>16.7x</td>
<td>984.7x</td>
</tr>
<tr>
<td>gather</td>
<td>1.15 GiB/s</td>
<td>0.45 GiB/s</td>
<td>13.45 GiB/s</td>
<td>539.52 GiB/s</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>0.4x</td>
<td>11.7x</td>
<td>469.1x</td>
</tr>
<tr>
<td>reduce</td>
<td>9.90 GiB/s</td>
<td>11.66 GiB/s</td>
<td>118.57 GiB/s</td>
<td>43683.00 GiB/s</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>1.2x</td>
<td>12.0x</td>
<td>4412.4x</td>
</tr>
<tr>
<td>scan</td>
<td>2.78 GiB/s</td>
<td>2.12 GiB/s</td>
<td>8.90 GiB/s</td>
<td>741.14 GiB/s</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>0.8x</td>
<td>3.2x</td>
<td>266.6x</td>
</tr>
<tr>
<td>scatter</td>
<td>1.17 GiB/s</td>
<td>1.12 GiB/s</td>
<td>13.77 GiB/s</td>
<td>914.67 GiB/s</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>1.0x</td>
<td>11.8x</td>
<td>781.8x</td>
</tr>
<tr>
<td>stream</td>
<td>3.94 GiB/s</td>
<td>2.92 GiB/s</td>
<td>24.58 GiB/s</td>
<td>6266.22 GiB/s</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>0.7x</td>
<td>6.2x</td>
<td>1590.4x</td>
</tr>
</tbody>
</table>
ARKOUDA ARGSORT AT MASSIVE SCALE

- Ran on a large Apollo system, summer 2021
  - 73,728 cores of AMD Rome
  - 72 TiB of 8-byte values
  - 480 GiB/s (2.5 minutes elapsed time)
  - ~100 lines of Chapel code

Close to world-record performance—quite likely a record for performance/SLOC
OVERVIEW OF CHAPEL FEATURES
CHAPEL FEATURE AREAS

Chapel language concepts

- Domain Maps
- Data Parallelism
- Task Parallelism
- Base Language
- Locality Control
- Target System
BASE LANGUAGE

Chapel language concepts

- Domain Maps
- Data Parallelism
- Task Parallelism
- Base Language
- Locality Control
- Target System

“Lower-level” Chapel
A TOY COMPUTATION: THE FIBONACCI SEQUENCE

• Our first program shows a stylized way of computing $n$ values of the Fibonacci sequence in Chapel...
  • This is admittedly a very artificial example, but it’s short and illustrative

• The Fibonacci Sequence:
  • Starts with: 0, 1
  • Successive terms obtained by adding the previous two terms: 1, 2, 3, 5, 8, ...
FIBONACCI ITERATION

```chapl
config const n = 10;

for f in fib(n) do
  writeln(f);

iter fib(x) {
  var current = 0,
      next = 1;

  for i in 1..x {
    yield current;
    current += next;
    current <=> next;
  }
}
```

`prompt> chpl fib.chpl`

`prompt> ./fib`

```
0
1
1
2
3
5
8
13
21
34
```

```
fib.chpl

```chpl
config const n = 10;

for f in fib(n) do writeln(f);

iter fib(x) {
    var current = 0,
    next = 1;

    for i in 1..x {
        yield current;
        current += next;
        current <=> next;
    }
}
```

Drive this loop by invoking fib(n)

```bash
prompt> chpl fib.chpl
prompt> ./fib
```

29
FIBONACCI ITERATION

```chapl
config const n = 10;

for f in fib(n) do
  writeln(f);

iter fib(x) {
  var current = 0,
     next = 1;

  for i in 1..x {
    yield current;
    current += next;
    current <=> next;
  }
}
```

Execute the loop’s body for that value

'yield' this expression back to the loop’s index variable

prompt> chpl fib.chpl
prompt> ./fib
0

3
1
1
2
3
5
8
13
21
34

30
FIBONACCI ITERATION

```chpl
config const n = 10;
for f in fib(n) do writeln(f);
iter fib(x) {
  var current = 0,
  next = 1;
  for i in 1..x {
    yield current;
    current += next;
    current <=> next;
  }
}
```

- Execute the loop’s body for that value
- Then continue the iterator from where it left off
- Repeating until we fall out of it (or return)

```
prompt> chpl fib.chpl
prompt> ./fib
0
1
1
2
3
5
8
13
21
34
```
FIBONACCI ITERATION

config const n = 10;

for f in fib(n) do
  writeln(f);

iter fib(x) {
  var current = 0,
  next = 1;

  for i in 1..x {
    yield current;
    current += next;
    current <=> next;
  }
}
FIBONACCI ITERATION

```chapl
fn    config const n = 10;
for f in fib(n) do
  writeln(f);
iter fib(x) {
  var current = 0,
  next = 1;
  for i in 1..x {
    yield current;
    current += next;
    current <=> next;
  }
}
```

Prompt>
```
chpl fib.chpl
prompt> ./fib --n=1000
```

0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
...

Static type inference for:
- constants / variables
- arguments
- return types

Explicit typing also supported
config const n: int = 10;

for f in fib(n) do
    writeln(f);

iter fib(x: int): int {
    var current: int = 0,
        next: int = 1;

    for i in 1..x {
        yield current;
        current += next;
        current <=> next;
    }
}
FIBONACCI ITERATION

```chpl
config const n = 10;

for (i,f) in zip(0..<n, fib(n)) do
    writeln("fib ", i, " is ", f);

iter fib(x) {
    var current = 0,
        next = 1;

    for i in 1..x {
        yield current;
        current += next;
        current <=> next;
    }
}
```

---

Prompt> chpl fib.chpl
Prompt> ./fib --n=1000
fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
fib #7 is 13
fib #8 is 21
fib #9 is 34
fib #10 is 55
fib #11 is 89
fib #12 is 144
fib #13 is 233
fib #14 is 377
...
FIBONACCI ITERATION

```chpl
config const n = 10;

for (i, f) in zip(0..<n, fib(n)) do
  writeln("fib ", i, " is ", f);

iter fib(x) {
  var current = 0,
  next = 1;

  for i in 1..x {
    yield current;
    current += next;
    current <=> next;
  }
}
```

prompt> chpl fib.chpl
prompt> ./fib --n=1000
fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
fib #7 is 13
fib #8 is 21
fib #9 is 34
fib #10 is 55
fib #11 is 89
fib #12 is 144
fib #13 is 233
fib #14 is 377
...
OTHER BASE LANGUAGE FEATURES

• Various basic types: bool, int(w), uint(w), real(w), imag(w), complex(w), enums, tuples
• Error-handling
• Compile-time meta-programming
• Object-oriented programming
  • Value-based records (like C structs supporting methods, generic fields, etc.)
  • Reference-based classes (somewhat like Java classes or C++ pointers-to-classes)
    – Nilable vs. non-nilable variants
    – Memory-management strategies (shared, owned, borrowed, unmanaged)
    – Lifetime checking
• Generic programming / polymorphism
• Procedure overloading / filtering
• Arguments: default values, intents, name-based matching, type queries
• Modules (supporting namespaces)
• and more...
TASK PARALLELISM AND LOCALITY CONTROL

Chapel language concepts

- Domain Maps
- Data Parallelism
- Task Parallelism
- Base Language
- Locality Control
- Target System

“Lower-level” Chapel
THE LOCALE: CHAPEL’S KEY FEATURE FOR LOCALITY

- **locale**: a unit of the target architecture that can run tasks and store variables
  - Think “compute node” on a typical HPC system

```
prompt> ./myChapelProgram --numLocales=4 # or ‘–nl 4’
```

 Locales array:

Locale 0  Locale 1  Locale 2  Locale 3

User’s program starts running as a single task on locale 0
const numTasks = here.maxTaskPar;
coforall tid in 1..numTasks do
  writef("Hello from task %n of %n on %s\n", tid, numTasks, here.name);
```chapl
const numTasks = here.maxTaskPar;
coforall tid in 1..numTasks do
    printf("Hello from task %n of %n on %s\n", tid, numTasks, here.name);
```

- 'here' refers to the locale on which this code is currently running.
- How many parallel tasks can my locale run at once?
- What's my locale's name?
TASK-PARALLEL "HELLO WORLD"

```chpl
const numTasks = here.maxTaskPar;
coforall tid in 1..numTasks do
  printf("Hello from task \%n of \%n on \%s\n", tid, numTasks, here.name);
```

A `coforall` loop executes each iteration as an independent task.
TASK-PARALLEL "HELLO WORLD"

```chapl
const numTasks = here.maxTaskPar;
coforall tid in 1..numTasks do
    printf("Hello from task %d of %d on %s\n", tid, numTasks, here.name);
```

So far, this is a shared-memory program

Nothing refers to remote locales, explicitly or implicitly
const numTasks = here.maxTaskPar;
coforall tid in 1..numTasks do
  writef("Hello from task %n of %n on %s\n", tid, numTasks, here.name);
cooparallel "HELLO WORLD" (DISTRIBUTED VERSION)

```chpl
helloTaskPar.chpl

cooparallel loc in Locales {
    on loc {
        const numTasks = here.maxTaskPar;
        cooparallel tid in 1..numTasks do
            printf("Hello from task %n of %n on %s\n", tid, numTasks, here.name);
    }
}
```
**TASK-PARALLEL “HELLO WORLD” (DISTRIBUTED VERSION)**

```chpl
coforall loc in Locales {
    on loc {
        const numTasks = here.maxTaskPar;
        coforall tid in 1..numTasks do
            printf("Hello from task %n of %n on %s\n", tid, numTasks, here.name);
    }
}
```

Locales array:

- Locale 0
- Locale 1
- Locale 2
- Locale 3

The array of locales we’re running on (introduced a few slides back)
**TASK-PARALLEL “HELLO WORLD” (DISTRIBUTED VERSION)**

```chapl
coforall loc in Locales {
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      printf("Hello from task %n of %n on %s\n", tid, numTasks, here.name);
  }
}
```

create a task per locale on which the program is running

have each task run ‘on’ its locale

then print a message per core, as before

```
prompt> chpl helloTaskPar.chpl
prompt> ./helloTaskPar -numLocales=4
Hello from task 1 of 4 on n1032
Hello from task 4 of 4 on n1032
Hello from task 1 of 4 on n1034
Hello from task 2 of 4 on n1032
Hello from task 1 of 4 on n1033
Hello from task 3 of 4 on n1034
Hello from task 1 of 4 on n1035
...```
**TASK-PARALLEL “HELLO WORLD” (DISTRIBUTED VERSION)**

```chpl
helloTaskPar.chpl

coforall loc in Locales {
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      printff("Hello from task %n of %n on %s\n", tid, numTasks, here.name);
  }
}
```

```
prompt> chpl helloTaskPar.chpl
prompt> ./helloTaskPar -numLocales=4
Hello from task 1 of 4 on n1032
Hello from task 4 of 4 on n1032
Hello from task 1 of 4 on n1034
Hello from task 2 of 4 on n1032
Hello from task 1 of 4 on n1033
Hello from task 3 of 4 on n1034
Hello from task 1 of 4 on n1035
...```
CHAPEL FEATURE AREAS

Chapel language concepts

- Domain Maps
- Data Parallelism
- Task Parallelism
- Base Language
- Locality Control
- Target System
DATA PARALLELISM AND DOMAIN MAPS

Chapel language concepts

<table>
<thead>
<tr>
<th>Domain Maps</th>
<th>Data Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Parallelism</td>
<td>Base Language</td>
</tr>
<tr>
<td>Locality Control</td>
<td></td>
</tr>
<tr>
<td>Target System</td>
<td></td>
</tr>
</tbody>
</table>

Higher-level Chapel
config const n = 1000;

const D = {1..n, 1..n};

var A: [D] real;

forall (i, j) in D do
  A[i, j] = i + (j - 0.5)/n;

writeln(A);
DATA-PARALLEL ARRAY FILL

```chpl
config const n = 1000;

const D = {1..n, 1..n};

var A: [D] real;

forall (i, j) in D do
    A[i, j] = i + (j - 0.5)/n;

writeln(A);
```

- declare a domain, a first-class index set
- declare an array over that domain
DATA-PARALLEL ARRAY FILL

fillArray.chpl

config const n = 1000;

const D = {1..n, 1..n};

var A: [D] real;

forall (i, j) in D do
  A[i, j] = i + (j - 0.5)/n;

writeln(A);

declare a domain, a first-class index set

declare an array over that domain

iterate over the domain’s indices in parallel, assigning to the corresponding array elements
DATA-PARALLEL ARRAY FILL

fillArray.chpl

```chapel
config const n = 1000;

const D = {1..n, 1..n};

var A: [D] real;

forall (i, j) in D do
    A[i, j] = i + (j - 0.5)/n;

writeln(A);
```

prompts>
```
chpl dataParallel.chpl
.
```
```
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```

So far, this is a shared-memory program

Nothing refers to remote locales, explicitly or implicitly
DATA-PARALLEL ARRAY FILL

```chpl
config const n = 1000;

const D = {1..n, 1..n};

var A: [D] real;

forall (i, j) in D do
    A[i, j] = i + (j - 0.5)/n;

writeln(A);
```

So far, this is a shared-memory program

Nothing refers to remote locales, explicitly or implicitly
config const n = 1000;
const D = \{1..n, 1..n\};
var A: [D] real;

forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;

writeln(A);
**DATA-PARALLEL ARRAY FILL (DISTRIBUTED VERSION)**

```chapel
define fillArray.chpl
    use CyclicDist;

    config const n = 1000;

    const D = {1..n, 1..n}
        dmapped Cyclic(startIdx = (1,1));

    var A: [D] real;

    forall (i,j) in D do
        A[i,j] = i + (j - 0.5)/n;

    writeln(A);
```

```plaintext
1.1 1.3 1.5 1.5 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```
DATA-PARALLEL ARRAY FILL (DISTRIBUTED VERSION)

```
fillArray.chpl

use CyclicDist;

config const n = 1000;

const D = {1..n, 1..n}
         dmapped Cyclic(startIdx = (1,1));

var A: [D] real;

forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;

writeln(A);
```

apply a domain map, specifying how to implement...
...the domain's indices,
...the array's elements,
...the loop's iterations,
...on the program's locales

Locales array:
DATA-PARALLEL ARRAY FILL (DISTRIBUTED VERSION)

```chapel
use CyclicDist;

cfg const n = 1000;

cfg D = {1..n, 1..n}
    dmapped Cyclic(startIdx = (1,1));

cfg A: [D] real;

forall (i,j) in D do
    A[i,j] = i + (j - 0.5)/n;

writeln(A);
```

Because this computation is independent of the locales, changing the number of locales or distribution doesn't affect the output.

```
locale 0
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```

Prompt: `chpl dataParallel.chpl`
Prompt: `./dataParallel --n=5 --numLocales=1`

```
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```
DATA-PARALLEL ARRAY FILL (DISTRIBUTED VERSION)

```chapel
fillArray.chpl

use CyclicDist;

config const n = 1000;

const D = {1..n, 1..n}
    dmapped Cyclic(startIdx = (1,1));

var A: [D] real;

forall (i, j) in D do
    A[i, j] = i + (j - 0.5)/n;

writeln(A);
```

Because this computation is independent of the locales, changing the number of locales or distribution doesn’t affect the output.
DATA-PARALLEL ARRAY FILL (DISTRIBUTED VERSION)

```chapel
fillArray.chpl

use CyclicDist;

config const n = 1000;

const D = {1..n, 1..n}
  dmapped Cyclic(startIdx = (1,1));

var A: [D] real;

forall (i, j) in D do
  A[i,j] = i*10 + j + (here.id+1)/10.0;

writeln(A);

```

If we make it sensitive to the locales, the output varies with the distribution details.

prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5 --numLocales=1

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11.1</td>
<td>12.1</td>
<td>13.1</td>
<td>14.1</td>
<td>15.1</td>
</tr>
<tr>
<td>21.1</td>
<td>22.1</td>
<td>23.1</td>
<td>24.1</td>
<td>25.1</td>
</tr>
<tr>
<td>31.1</td>
<td>32.1</td>
<td>33.1</td>
<td>34.1</td>
<td>35.1</td>
</tr>
<tr>
<td>41.1</td>
<td>42.1</td>
<td>43.1</td>
<td>44.1</td>
<td>45.1</td>
</tr>
<tr>
<td>51.1</td>
<td>52.1</td>
<td>53.1</td>
<td>54.1</td>
<td>55.1</td>
</tr>
</tbody>
</table>
**DATA-PARALLEL ARRAY FILL (DISTRIBUTED VERSION)**

```chpl
use CyclicDist;

cfg const n = 1000;

const D = {1..n, 1..n}
dmapped Cyclic(startIdx = (1,1));
var A: [D] real;

forall (i,j) in D do
    A[i,j] = i*10 + j + (here.id+1)/10.0;

writeln(A);
```

If we make it sensitive to the locales, the output varies with the distribution details.
use CyclicDist;

config const n = 1000;

const D = {1..n, 1..n}
  dmapped Cyclic(startIdx = (1,1));
var A: [D] real;

forall (i, j) in D do
  A[i, j] = i*10 + j + (here.id+1)/10.0;

writeln(A);
WRAP-UP
Chapel is unique among programming languages

- built-in features for scalable parallel computing make it HPC-ready
- supports clean, concise code relative to conventional approaches
- ports and scales from laptops to supercomputers

Chapel is being used for productive parallel computing at scale

- users are reaping its benefits in practical, cutting-edge applications
- applicable to domains as diverse as physical simulations and data science

If you or your users are interested in taking Chapel for a spin, let us know!

- we’re happy to work with users and user groups to help ease the learning curve
CHAPEL RESOURCES

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

Social Media:
• Twitter: @ChapelLanguage
• Facebook: @ChapelLanguage
• YouTube: http://www.youtube.com/c/ChapelParallelProgrammingLanguage

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
THANK YOU

https://chapel-lang.org
@ChapelLanguage