Setup

- Get Chapel on Peano, then try a program

```
<ssh into peano>
$ module load git singularity gnu8
$ git clone git@github.com:ghbrown/chapel-introduction.git
$ source /tmp/chapel/get_chpl.sh
```

```
$ cd chapel-introduction/src
$ chpl hello.chpl
$ ./hello
```

- **or**

```
<on your local machine>
$ browser  https://github.com/ghbrown/chapel-introduction/tree/main
$ browser  https://shorturl.at/ezIV7
```

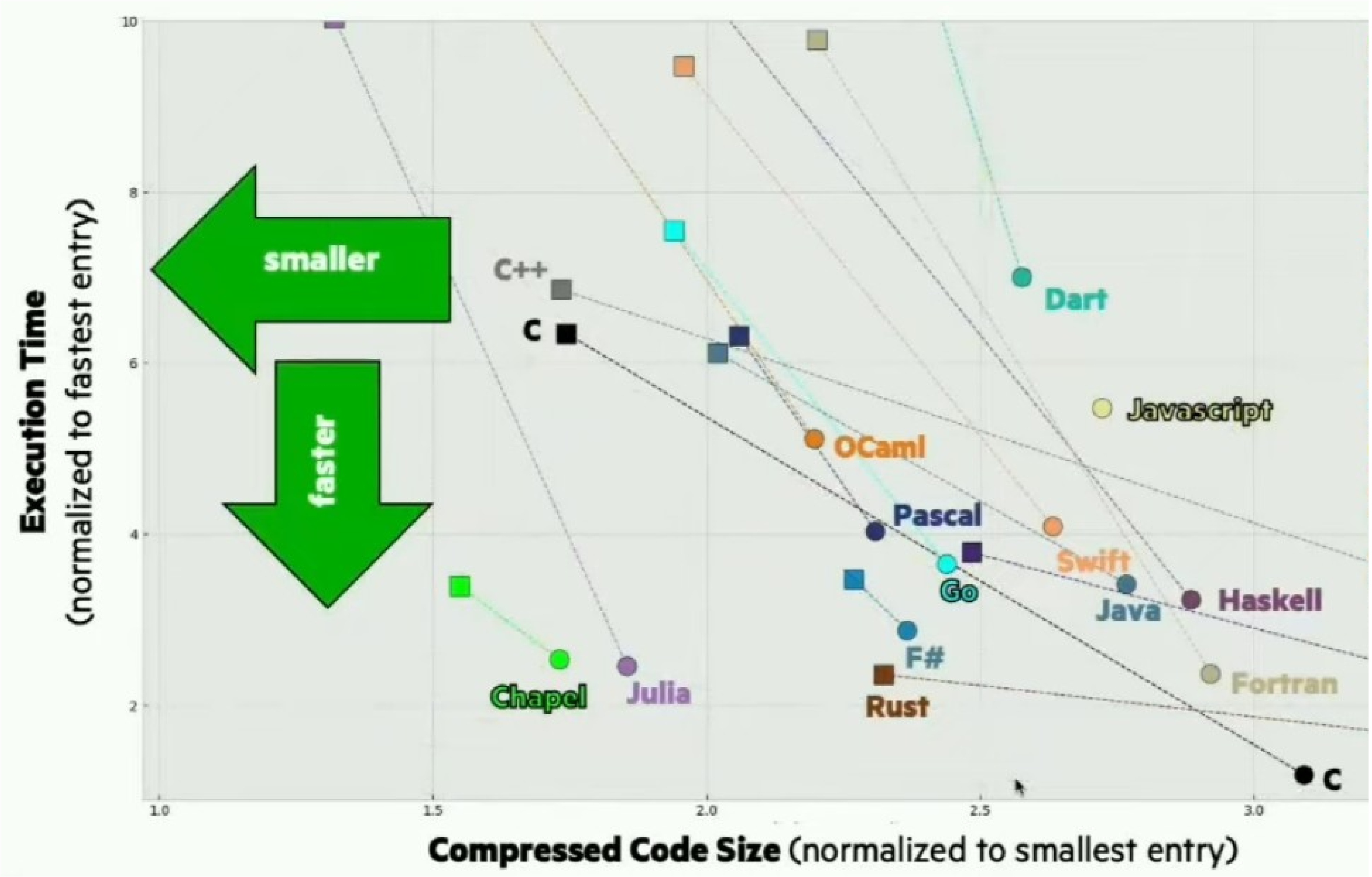
Chapel Parallel Programming Language:

a **user** introduction

History

- Cray’s entry into DARPA’s High Productivity Computing Systems project
  - “there exists a critical need for improved software tools, standards, and methodologies for effective utilization of multiprocessor computers” – Lusk and Yelick, 2007
- A language for high performance and productivity parallel programming
- Development now continued at HPE

Motivation: language benchmarks



source

Chapel: serial language features

- static typing (with type inference)

```
var s : string = 'foo';

var a : real = 1.0;
var b : real = 3.0;
var c : real = a + b;
```

```
var s = 'foo';

var a = 1.0;
var b = 3.0;
var c = a + b;
```

Chapel: serial language features

- generics
  - functions can work for multiple input types
  - not present in C or Fortran (without interfaces)

```
proc square(v) {
  return v*v;
}

var a : real(32) = 1.0;
var b : real(64) = 2.0;

writeln(square(a));
writeln(square(b));
```

Chapel: serial language features

- first class arrays (with fancy indexing)

```
proc frobenius_norm(A) {
    return sqrt(+ reduce A**2);  // sqrt(sum of a_ij^2 for all i,j)
}


// --size=<N> to specify different value on command line
config const size : int = 10;

// two arrays of same size with different indexing
var B : [1..size, 1..size]   real = 1.0;
var C : [0..<size, 0..<size] real = 1.0;

// call function directly on arrays
writeln(frobenius_norm(B));
writeln(frobenius_norm(C));
```

Chapel: serial language features

- iterators and zipping

```
use IO.FormattedIO;

var ordinals = ('first','second','third');

for (ord,i) in zip(ordinals,1..) {
  writeln('ordinal "%s" corresponds to number "%i"'.format(ord,i));
}
```

Chapel: serial language features

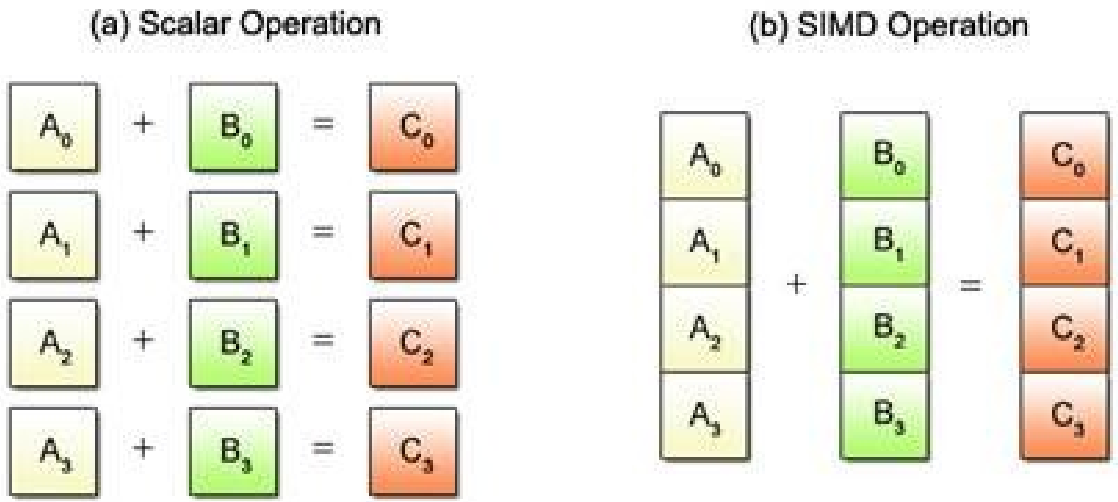
- Also:
  - multiple outputs
  - default arguments
  - objects

```
proc first_four_powers(a) {
  var p1 = a;
  var p2 = p1*a;
  var p3 = p2*a;
  var p4 = p3*a;
  return (p1, p2, p3, p4);
}

var a : int = 2;
writeln('The first four powers of ',a,' are:');
for p in first_four_powers(a) {
  writeln('  ',p);
}
```

Parallel programming models: SIMD

- single instruction multiple data
  - gives up flexibility for speed
  - CPU instructions like addpd
  - contributes to performance of Numpy* and GPUs



source

Parallel programming models: SPMD

- single program multiple data
  - trade flexibility for program complexity
  - same program runs on every node, doing possibly different things simultaneously
  - MPI is dominant standard in HPC, designed for distributed memory

```
# send vector in chunks to other nodes
if (node == 0):
    for i in [0,…,n_nodes]:
        MPI_SEND(a[8*i:8*(i+1)])
        MPI_SEND(b[8*i:8*(i+1)])

# all nodes receive chunks
MPI_RECEIVE(a_local)
MPI_RECEIVE(b_local)
c_local = a_local + b_local

if (node == 12):
    print(“I’m node 12.”)
...
```

Parallel programming models: directives

- Serial code annotated with directives for compiler
  - OpenMP and OpenACC both work for CPUs and GPUs
  - preserves portabilty (AMD vs Intel, Nvidia vs AMD)
  - often combined with MPI

```
#pragma omp parallel for
for(int i=0; i<ARRAY_SIZE; i++)
    c[i] = a[i] + b[i];
}
```

Note: “directives” summarizes a way of interacting with source code, but the directives themselves can do many things, including SIMD on CPUs or GPU kernels

Parallel programming models: PGAS

- Partitioned Global Address Space
  - global address space: threads can read/write remote data
  - partitioned: data designated as local versus global
  - examples: Unified Parallel C, CoArray Fortran, **Chapel**

MPI: isolated processes with isolated memories exchange messages

OpenMP: multiple threads can read and write a shared memory

PGAS

- virtually global address space realized across distributed hardware with non-uniform memory access times
- compiler uses program as specification to map processes to hardware

source 1

source 2

Chapel: locales

- Resources with compute and memory
  - most naturally a (many core) compute node
- Hierarchical:
  - may contain sublocales (like GPUs)

```
for loc in Locales {
  writeln("Started on locale #", here.id);
  on loc do
    writeln("Hello from locale #", here.id);
  writeln("Back on    locale #", here.id, "\n");
}
```

***this is not a parallel program**

Chapel: task parallelism

- begin:
  - fork one non-blocking process
- cobegin
  - fork one process per statement, blocking
- coforall
  - fork one process per iteration, blocking

```
writeln('Hello 1 (always executes first)');
cobegin {
    writeln('Hello 2');
    writeln('Hello 3');
    writeln('Hello 4');
}
writeln('Hello 5 (always executes last)');
```

```
writeln('Hello 1 (always executes first)');
coforall i in 2..4 {
    writeln('Hello ',i);
}
writeln('Hello 5 (always executes last)');
```

Chapel: data parallelism

- forall:
  - behavior dictated by supplied iterators, which determine degree of concurrency and location of execution
  - requires order-independence of loop body

```
// this is a data race
forall i in 2..n-1 do
  A[i] = A[i-1] + A[i+1];
```

Chapel: orthogonality of locality and parallelism

```
// serial
for loc in Locales do
  on loc do
    writeln("Hello from locale ", loc.id, " of ", numLocales);

// parallel
coforall loc in Locales do
  on loc do
    writeln("Hello from locale ", loc.id, " of ", numLocales);
```

Let’s run some code!

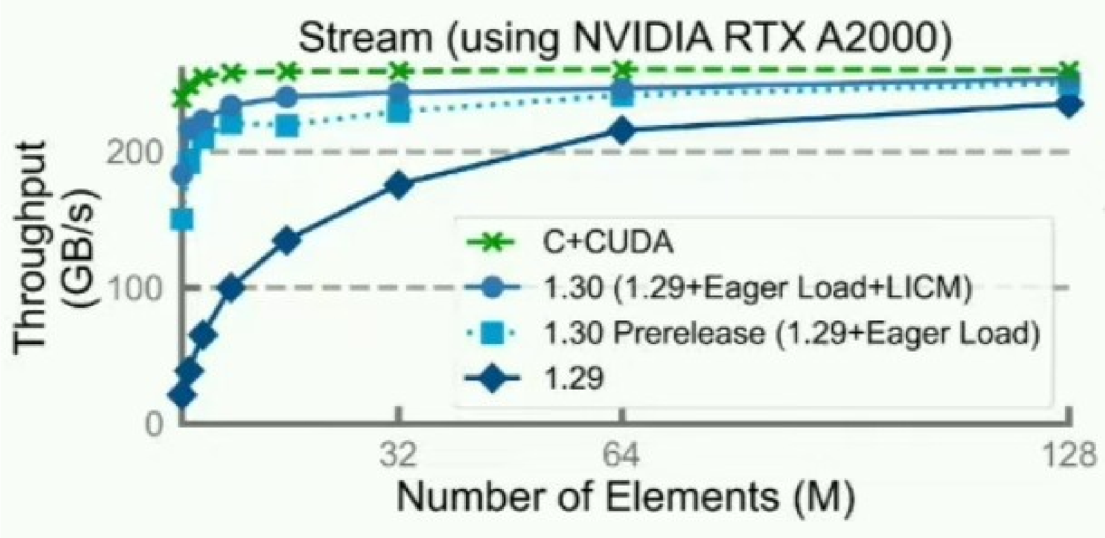
- Code in chapel-introduction/src/parallel/triad
- Compile a Chapel source file with

```
$ chpl --fast -o build/<name> <name>.chpl
```

- Compile all Fortran source files with make
- Setup times: 0.58 s (Fortran), 0.41 s (Chapel)

Chapel is ready for prime time, but...

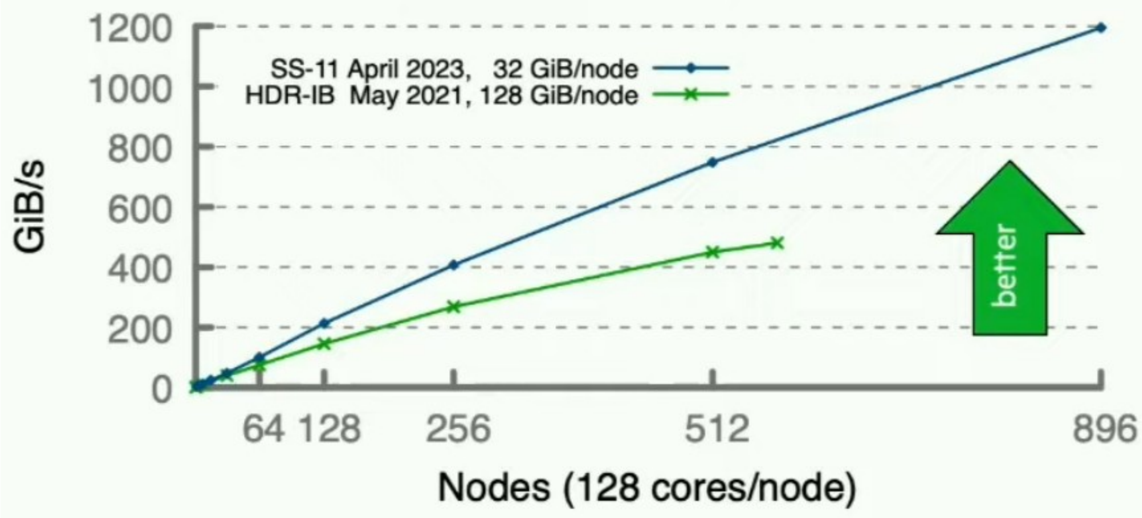
- compile times
- tooling
- hardware*
- packaging
- availability on HPC systems



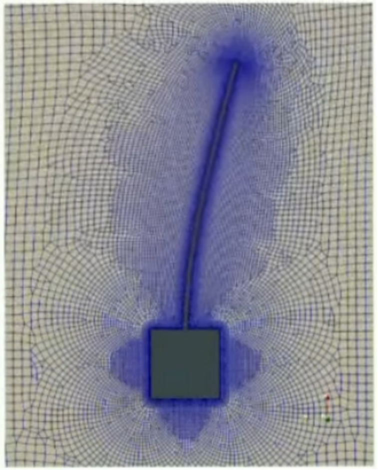
source

Success stories

- Sorting:
  - (near) world record performance in 100 lines of Chapel



- CHAMPS
  - 3D unstructured CFD, airplane simulation
  - “students take 3 months to do what used to take 2 years”



source

Wrapping up

- Could not cover:
  - language interoperability: C, Fortran, Python
  - iterators, ranges, (sparse) domains, distributions, layouts, atomics, synchonization
  - GPUs
- Resources:
  - Learn Chapel in Y minutes
  - Official Cray/HPE talk on Chapel
  - Online compiler