Chapel
An Emerging Parallel Programming Language

If you wouldn’t recommend your HPC programming model to a non-HPC friend, why aren’t you demanding something better?

HPC has no languages that are as modern, productive, and enjoyable as Python / Matlab / Java / (your favorite language here). This is due less to technical challenges than to lack of community will, resources, effort, and patience.

Let’s change that!

Chapel Characteristics

- **General Parallelism**
  - Any parallel algorithm
  - Any parallel hardware
- **Separate Parallelism from Locality from Mechanism**
  - What should run concurrently?
  - Where should tasks / data be placed?
- **Core features are functional, ready to be used**
- **Hosted and developed at SourceForge**
- **v1.8: 19 contributors from 8 organizations/5 countries**

Chapel’s Implementation

- **Open Source**
- **BSD License**
- **Hosted and developed at SourceForge**
- **research**
- **outreach**
- **features designed to compete with C/Fortran + MPI**
- **scalable, portable design**
  - From laptops to clusters, supercomputers, & the cloud
  - Requirements: C/C++, POSIX tools and threads
- **status**
  - Version 1.8 released October 17, 2013
  - Core features are functional, ready to be used
  - Additional performance optimizations needed
- **Community**
  - v1.8: 19 contributors from 8 organizations/5 countries

Data Parallelism

Role: drive parallelism via domains/arrays

```chapel
const ElemSpace = (.,.,Elems);  // exact and distributed domain
Nodes = ElemSpace dapped Short(ElemSpace);
```

Computations from the Chapel version of LULESH

- **roles**: map global-view domains/arrays to locales
- **declarations from the Chapel version of LULESH**

Domain Maps

Goal: Support multiple tiers of features

- higher levels for programmability, productivity
- lower levels for greater degrees of control

Chapel language concepts

- build the higher levels in terms of the lower
- permit the user to intermix layers arbitrarily

Task Parallelism

Role: express parallelism via concurrent tasks

```chapel
cobegin // producer() consumer()
proclw (int i) cy
var k
for k = 0 to #buffsize - 1
do
buff[i] = buff[i] + consumer()
end
```

Bounded buffer producer-consumer in Chapel

Locality Control

Role: permit tuning for locality and affinity

```chapel
var : int,                @Locals[
var : int,
] = int(1);
```

Sample mapping of tasks and their variables to locales

Next Steps

- **performance optimizations**
  - **accelerator support (GPUs, MIC)**
  - **improved interoperaibility**
  - **feature improvements**
  - **research efforts**
  - **outreach**

A Team Effort — Join Us!