Chapel Programming

By Parth Patel and Jenis Patel

What is Chapel?

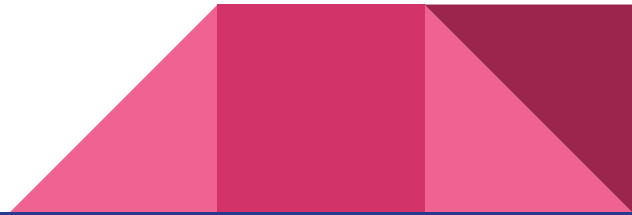
Chapel: A productive parallel programming language

Characteristics:

- portable
- open-source
- a collaborative effort
- a work-in-progress

Goals:

- Support general parallel programming
- “any parallel algorithm on any parallel hardware”
- Make parallel programming far more productive



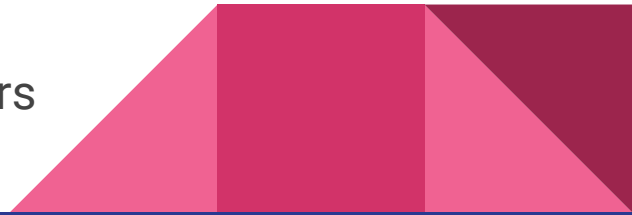
Chapel is Portable

- Chapel is designed to be hardware-independent
- The current implementation requires:
    - a C/C++ compiler
    - a *NIX environment (Linux, OS X, BSD, Cygwin, …)
    - POSIX threads
    - UDP, MPI, or RDMA (for distributed memory execution)
- As a result, Chapel can run on…

  …laptops and workstations

  …commodity clusters

  …the cloud …HPC systems from Cray and other vendors



Chapel is Open-Source

- Chapel's development is hosted at GitHub
- Chapel is licensed as Apache v2.0 software

THE APACHE® SOFTWARE FOUNDATION

GitHub

Chapel is Work-in-Progress

- Based on positive response from the DARPA project, cray undertook a 5 year effort to improve it
- More things have been added and fixed by Cray and the community

Advantages

General Parallelism: Chapel has the goal of supporting any parallel algorithm you can conceive of on any parallel hardware you want to target. In particular, you should never hit a point where you think “Well, that was fun while it lasted, but now that I want to do *x*, I’d better go back to MPI.”

Separation of Parallelism and Locality: Chapel supports distinct concepts for describing parallelism (“These things should run concurrently”) from locality (“This should be placed here; that should be placed over there”). This is in sharp contrast to conventional approaches that either conflate the two concepts or ignore locality altogether.

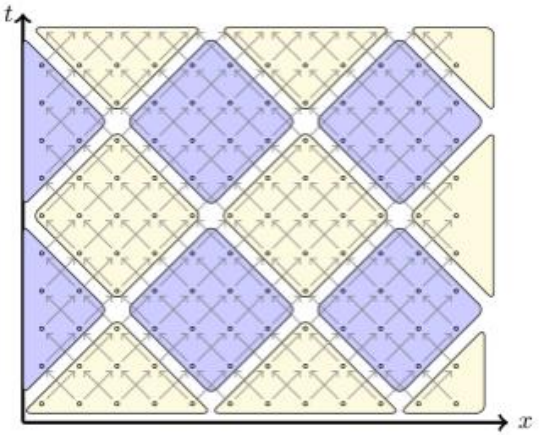
Multiresolution Design: Chapel is designed to support programming at higher or lower levels, as required by the programmer. Moreover, higher-level features—like data distributions or parallel loop schedules—may be specified by advanced programmers within the language.

Productivity Features: In addition to all of its features designed for supercomputers, Chapel also includes a number of sequential language features designed for productive programming. Examples include type inference, iterator functions, object-oriented programming, and a rich set of array types. The result combines productivity features as in Python™, Matlab®, or Java™ software with optimization opportunities as in Fortran or C.

Source: www.cray.com

Example: Diamond Tiling

- Tiling – strategy to reduce mem. bandwidth pressure
  - Improves performance and scalability of benchm
  - Usually must be done by hand
- Traditionally:
  - Difficult to write
  - Difficult to maintain
  - Not especially portable



Source: Bertolacci et. al. “Parameterized Diamond Tiling for Stencil Computations with Chapel parallel iterators.” ACM

Example: Diamond Tiling

```
// Loop over tile wavefronts.
for (kt=ceild(3,tau)-3; kt<=floord(3*T,tau); kt++) {
  // The next two loops iterate within a tile wavefront.
  int k1_lb = ceild(3*Lj+2+(kt-2)*tau,tau*3);
  int k1_ub = floord(3*Uj+(kt+2)*tau,tau*3);
  int k2_lb = floord((2*kt-2)*tau-3*Ui+2,tau*3);
  int k2_ub = floord((2+2*kt)*tau-3*Li-2,tau*3);
  // Loops over tile coordinates within a parallel wavefront of tiles.
  #pragma omp parallel for ...
  for (k1 = k1_lb; k1 <= k1_ub; k1++) {
    for (x = k2_lb; x <= k2_ub; x++) {
      k2 = x - k1;
      // Removing k1 term from k2 upper and lower bounds enables collapse(2).
      // Loop over time within a tile.
      for (t = max(1, floord(kt*tau-1, 3)); t < min(T+1, tau + floord(kt*tau, 3)); t++) {
        write = t & 1;
        // equivalent to t mod 2
        read = 1 - write;
        // Loops over the spatial dimensions within each tile.
        for (i =  max(Li,max((kt-k1-k2)*tau-t, 2*t-(2+k1+k2)*tau+2)); i <= min(Ui,min((1+kt-k1-k2)*tau-t-1,
        2*t-(k1+k2)*tau)); i++) {
          for (j =  max(Lj,max(tau*k1-t, t-i-(1+k2)*tau+1)); j <= min(Uj,min((1+k1)*tau-t-1, t-i-k2*tau));
          j++) {
            A[write][x][y] = (A[read][x-1][y] + A[read][x][y-1] + ... ; } } } } } }
```

Example: Diamond Tiling

```
forall (read, write, x ,y) in DiamondTileIterator(L, U, T, tau) {
  A[write, x, y] = (A[read,x-1,y] + A[read,x,y-1] + A[read,x  ,y] +
        A[read,x,y+1] + A[read,x+1,y]) / 5;
}
```