Chapel Background

Brad Chamberlain

Chapel

*Chapel*: a new parallel language being developed by Cray Inc.

**Themes:**

- general parallel programming
  - data-, task-, and nested parallelism
  - express general levels of software parallelism
  - target general levels of hardware parallelism
- global-view abstractions
- multiresolution design
- control of locality
- reduce gap between mainstream & parallel languages
Chapel’s Setting: HPCS

HPCS: High Productivity Computing Systems (DARPA et al.)
- Goal: Raise HEC user productivity by 10x for the year 2010
- Productivity = Performance
  - Programmability
  - Portability
  - Robustness

- Phase II: Cray, IBM, Sun (July 2003 – June 2006)
  - Evaluated the entire system architecture’s impact on productivity…
    - processors, memory, network, I/O, OS, runtime, compilers, tools, …
  - …and new languages:
    Cray: Chapel   IBM: X10   Sun: Fortress

- Phase III: Cray, IBM (July 2006 – 2010)
  - Implement the systems and technologies resulting from phase II
  - (Sun also continues work on Fortress, without HPCS funding)

Chapel and Productivity

Chapel’s Productivity Goals:
- vastly improve programmability over current languages/models
  - writing parallel codes
  - reading, modifying, porting, tuning, maintaining, evolving them
- support performance at least as good as MPI
  - competitive with MPI on generic clusters
  - better than MPI on more capable architectures
- improve portability compared to current languages/models
  - as ubiquitous as MPI, but with fewer architectural assumptions
  - more portable than OpenMP, UPC, CAF, …
- improve code robustness via improved semantics and concepts
  - eliminate common error cases altogether
  - better abstractions to help avoid other errors
Outline

- Chapel's Themes, Context, and Goals
- Programming Model Terminology
  - global-view vs. fragmented programming models
  - multiresolution languages
  - a first taste of Chapel

Parallel Programming Model Taxonomy

**programming model**: the mental model a programmer uses when coding using a language, library, or other notation

**fragmented models**: those in which the programmer writes code from the point-of-view of a single processor/thread

**global-view models**: those in which the programmer can write code that describes the computation as a whole
Global-view vs. Fragmented

**Problem:** “Apply 3-pt stencil to vector”

Global-view

\[
\begin{array}{c}
( \quad ) \\
+ \quad \frac{1}{2}
\end{array}
\]

Fragmented

= 

Global-view vs. Fragmented

**Problem:** “Apply 3-pt stencil to vector”

Global-view

\[
\begin{array}{c}
( \quad ) \\
+ \quad \frac{1}{2}
\end{array}
\]

Fragmented

= 

Brad Chamberlain, Steve Deitz, Samuel Figueroa, David Iten; Cray Inc.
Parallel Programming Model Taxonomy

**programming model:** the mental model a programmer uses when coding using a language, library, or other notation

**fragmented models:** those in which the programmer writes code from the point-of-view of a single processor/thread

**SPMD models:** Single-Program, Multiple Data -- a common fragmented model in which the user writes one program & runs multiple copies of it, parameterized by a unique ID

**global-view models:** those in which the programmer can write code that describes the computation as a whole

Global-view vs. SPMD Code

**Problem:** “Apply 3-pt stencil to vector”

```plaintext

**global-view**

```def main() {
    var n: int = 1000;
    var a, b: [1..n] real;
    forall i in 2..n-1 {
        b(i) = (a(i-1) + a(i+1))/2;
    }
}
```

```plaintext

**SPMD**

```def main() {
    var n: int = 1000;
    var locN: int = n/numProcs;
    var a, b: [0..locN+1] real;
    if (iHaveRightNeighbor) {
        send(right, a(locN));
        recv(right, a(locN+1));
    }
    if (iHaveLeftNeighbor) {
        send(left, a(1));
        recv(left, a(0));
    }
    forall i in 1..locN {
        b(i) = (a(i-1) + a(i+1))/2;
    }
}
```
Global-view vs. SPMD Code

Problem: “Apply 3-pt stencil to vector”

SPMD (pseudocode + MPI)

```plaintext
var n: int = 1000, locN: int = n/numProcs;
var a, b: [0..locN+1] real;
var numProcs, myPE: int;
var retval: int;
var status: MPI_Status;

MPI_Comm_size(MPI_COMM_WORLD, &numProcs);
MPI_Comm_rank(MPI_COMM_WORLD, &myPE);
if (myPE < numProcs) {
    retval = MPI_Send(&a(locN), 1, MPI_FLOAT, myPE+1, 0, MPI_COMM_WORLD);
    if (retval != MPI_SUCCESS) { handleErrror(retval); }
    retval = MPI_Recv(&a(locN+1), 1, MPI_FLOAT, myPE+1, 1, MPI_COMM_WORLD, &status);
    if (retval != MPI_SUCCESS) { handleErrrorWithStatus(retval, &status); }
    else
        innerHi = locN-1;
    if (myPE > 0) {
        retval = MPI_Send(&a(1), 1, MPI_FLOAT, myPE-1, 1, MPI_COMM_WORLD);
        if (retval != MPI_SUCCESS) { handleErrror(retval); }
        retval = MPI_Recv(&a(0), 1, MPI_FLOAT, myPE-1, 0, MPI_COMM_WORLD, &status);
        if (retval != MPI_SUCCESS) { handleErrrorWithStatus(retval, &status); }
        else
            innerLo = 2;
    } else
        innerLo = 1;
for all i in (innerLo..innerHi) {
    b(i) = (a(i-1) + a(i+1))/2;
}
```

Assumes numProcs divides n; a more general version would require additional effort

Global-view vs. SPMD Code

Problem: “Apply 3-pt stencil to vector”

Chapel Background (11)

```plaintext
def main() {
    var n: int = 1000;
    var a, b: [1..n] real;
    forall i in 2..n-1 {
        b(i) = (a(i-1) + a(i+1))/2;
    }
}
```

Chapel Background (12)

MPI SPMD pseudo-code

Problem: “Apply 3-pt stencil to vector”

SPMD (pseudocode + MPI)

```
var n: int = 1000, locN: int = n/numProcs;
var a, b: [1..locN+1] real;
var innerLo: int = 1, innerHi: int = locN;
var numProcs, myPE: int;
var retval: int;
var status: MPI_Status;

MPI_Comm_size(MPI_COMM_WORLD, &numProcs);
MPI_Comm_rank(MPI_COMM_WORLD, &myPE);
if (myPE < numProcs) {
    retval = MPI_Send(&a(locN), 1, MPI_FLOAT, myPE+1, 0, MPI_COMM_WORLD);
    if (retval != MPI_SUCCESS) { handleErrror(retval); }
    retval = MPI_Recv(&a(locN+1), 1, MPI_FLOAT, myPE+1, 1, MPI_COMM_WORLD, &status);
    if (retval != MPI_SUCCESS) { handleErrrorWithStatus(retval, &status); }
    else
        innerHi = locN-1;
    if (myPE > 0) {
        retval = MPI_Send(&a(1), 1, MPI_FLOAT, myPE-1, 1, MPI_COMM_WORLD);
        if (retval != MPI_SUCCESS) { handleErrror(retval); }
        retval = MPI_Recv(&a(0), 1, MPI_FLOAT, myPE-1, 0, MPI_COMM_WORLD, &status);
        if (retval != MPI_SUCCESS) { handleErrrorWithStatus(retval, &status); }
        else
            innerLo = 2;
    } else
        innerLo = 1;
for all i in (innerLo..innerHi) {
    b(i) = (a(i-1) + a(i+1))/2;
}
```

Communication becomes geometrically more complex for higher-dimensional arrays
rprj3 stencil from NAS MG

NAS MG rprj3 stencil in Fortran + MPI
NAS MG *rprj3* stencil in Chapel

```chapel
def rprj3(S, R) {
  const Stencil = [-1..1, -1..1, -1..1],
  w: [0..3] real = (0.5, 0.25, 0.125, 0.0625),
  w3d = [(i,j,k) in Stencil] w((i!=0) + (j!=0) + (k!=0));
  forall ijk in S.domain do
    S(ijk) = + reduce [offset in Stencil]
      (w3d(offset) * R(ijk + offset*R.stride));
}
```

*Our previous work in ZPL showed that compact, global-view codes like these can result in performance that matches or beats hand-coded Fortran+MPI*

Summarizing Fragmented/SPMD Models

- **Advantages:**
  - fairly straightforward model of execution
  - relatively easy to implement
  - reasonable performance on commodity architectures
  - portable/ubiquitous
  - lots of important scientific work has been accomplished with them

- **Disadvantages:**
  - blunt means of expressing parallelism: cooperating executables
  - fails to abstract away architecture / implementing mechanisms
  - obfuscates algorithms with many low-level details
    - error-prone
    - brittle code: difficult to read, maintain, modify, *experiment*
    - “MPI: the assembly language of parallel computing”
Current HPC Programming Notations

- **communication libraries:**
  - MPI, MPI-2
  - SHMEM, ARMCI, GASNet

- **data / control:**
  - fragmented / fragmented/SPMD

- **shared memory models:**
  - OpenMP, pthreads

- **data / control:**
  - global-view / global-view (trivially)

- **PGAS languages:**
  - Co-Array Fortran
  - UPC
  - Titanium

- **PGAS languages:**
  - fragmented / SPMD
  - global-view / SPMD
  - fragmented / SPMD

- **HPCS languages:**
  - Chapel
  - X10 (IBM)
  - Fortress (Sun)

- **HPCS languages:**
  - global-view / global-view
  - global-view / global-view
  - global-view / global-view

Parallel Programming Models: Two Camps

"Why is everything so painful?"

"Why do my hands feel tied?"
Multiresolution Language Design

**Our Approach:** Permit the language to be utilized at multiple levels, as required by the problem/programmer

- provide high-level features and automation for convenience
- provide the ability to drop down to lower, more manual levels
- use appropriate separation of concerns to keep these layers clean