Chapel: Project Overview
Outline

- What we do
- Project Status
- Who we are
- Collaboration Sidebar: Chapel on CPU+GPU
Chapel Work

- Chapel Team’s Focus:
  - specify Chapel syntax and semantics
  - implement open-source prototype compiler for Chapel
  - perform code studies of benchmarks, apps, and libraries in Chapel
  - do community outreach to inform and learn from users/researchers
  - support collaborators and users of code releases
  - refine language based on all these activities
In a nutshell:

- Most features work at a functional level
- Many performance optimizations remain

This is a good time to:

- Try out the language
- Give us feedback to improve the language
- Use Chapel for parallel programming education
- Use Chapel for non-performance-critical projects

In evaluating the language:

- Try to judge it by how it should *ultimately* perform rather than how it does today
  - lots of low-hanging fruit remains, as well as some challenges
If I were teaching parallel programming, I’d want to cover:

- data parallelism
- task parallelism
- concurrency
- synchronization
- locality/affinity
- deadlock, livelock, and other pitfalls
- performance tuning
- ...

I don’t think there’s a good language out there...

- for teaching all of these things
- for teaching some of these things well at all

**until now:** I think Chapel has the potential to play a crucial role here
“I Like Chapel, how can I help?”

- Let people know that you like it and why
  - your colleagues
  - your employer/institution
  - Cray leadership (stop by the Cray booth this week)

- Help us evolve it from prototype to production
  - our team’s size is OK for creating and prototyping, but too small to create a product-grade version in a timely manner
    - contribute back to the source base
    - collaborate with us
    - help fund us to grow the team
    - help move us from “How will Cray make Chapel succeed?” to “How can we as a community make Chapel succeed?”
Join Our Team

- **Cray:**
  - Brad Chamberlain
  - Sung-Eun Choi
  - Greg Titus
  - Lee Prokowich
  - Vass Litvinov

- **External Collaborators:**
  - Albert Sidelnik
  - Jonathan Turner
  - Srinivas Sridharan

- **Interns:**
  - Jonathan Claridge
  - Hannah Hemmaplardh
  - Andy Stone
  - Jim Dinan
  - Rob Bocchino
  - Mack Joyner

You?
**Currently:**

- Software Engineer (Compiler Developer)
- Manager

**Upcoming:**

- R&D on Targeting next-generation nodes
  - GPUs, tiled architectures, scratchpad memories, manycore, ...
Select Collaborations

- **Notre Dame/ORNL** (Peter Kogge, Srinivas Sridharan, Jeff Vetter): Asynchronous *software transactional memory* over distributed memory
- **UIUC** (David Padua, Albert Sidelnik, Maria Garzarán): *CPU-GPU computing*
- **BSC/UPC** (Alex Duran): Chapel over Nanos++ *user-level tasking*
- **Argonne** (Rusty Lusk, Rajeev Thakur, Pavan Balaji): Chapel over MPICH
- **Sandia** (Rich Murphy, Kyle Wheeler): Chapel over *Qthreads* user threading
- **UT Austin** (Calvin Lin, Karthik Murthy): Memory consistency models
- **CU Boulder** (Jeremy Siek, Jonathan Turner): Interfaces, concepts, generics
- **U. Oregon/Paratools Inc.** (Sameer Shende): Performance analysis with Tau
- **U. Malaga** (Rafa Asenio, Maria Gonzales, Rafael Larossa): Parallel file I/O
- **PNNL/CASS-MT** (John Feo, Daniel Chavarria): Cray XMT tuning
- *(your name here?)*
Collaboration Ideas (see chapel.cray.com for more details)

- memory management policies/mechanisms
- dynamic load balancing: task throttling and stealing
- parallel I/O and checkpointing
- exceptions; resiliency
- language interoperability
- application studies and performance optimizations
- index/subdomain semantics and optimizations
- targeting different back-ends (LLVM, MS CLR, ...)
- runtime compilation
- library support
- tools: debuggers, performance analysis, IDEs, interpreters, visualizers
- database-style programming
- (your ideas here...)
Select Collaborations

- **Notre Dame/ORNL** (Peter Kogge, Srinivas Sridharan, Jeff Vetter): Asynchronous software transactional memory over distributed memory
- **UIUC** (David Padua, Albert Sidelnik, Maria Garzarán): CPU-GPU computing
- **BSC/UPC** (Alex Duran): Chapel over Nanos++ user-level tasking
- **Argonne** (Rusty Lusk, Rajeev Thakur, Pavan Balaji): Chapel over MPICH
- **Sandia** (Rich Murphy, Kyle Wheeler): Chapel over Qthreads user threading
- **UT Austin** (Calvin Lin, Karthik Murthy): Memory consistency models
- **CU Boulder** (Jeremy Siek, Jonathan Turner): Interfaces, concepts, generics
- **U. Oregon/Paratools Inc.** (Sameer Shende): Performance analysis with Tau
- **U. Malaga** (Rafa Asenio, Maria Gonzales, Rafael Larossa): Parallel file I/O
- **PNNL/CASS-MT** (John Feo, Daniel Chavarria): Cray XMT tuning
- (your name here?)
Targeting GPGPUs With Chapel

Albert Sidelnik, María J. Garzarán, David Padua (UIUC)

Brad Chamberlain (Cray Inc.)
Motivating Example: HPCC Stream Triad

\[ A = \text{scalar} \times B + C; \]
config const m = 1000, tbSizeX = 256;
const alpha = 3.0;
const ProbDist = new dmap(new dist(GPUDist(rank=1, tbSizeX)));
const ProbSpace: domain(1) dmapped ProbDist = [1..m];
var A, B, C: [ProbSpace] real;
forall (a,b,c) in (A,B,C) do
  a = b + alpha * c;

New configuration constant to specify # threads per block
Distribution to target a GPU
Loops and arrays using this domain are implemented on the GPU
No changes required to the computation for other architectures
Case Study: STREAM (current practice)

#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif
static int VectorSize;
static double *a, *b, *c;
int HPCC_StarStream(HPCC_Parms *params) {
    int myRank, commSize;
    int rv, errCount;
    MPI_Comm comm = MPI_COMM_WORLD;
    MPI_Comm_size( comm, &commSize );
    MPI_Comm_rank( comm, &myRank );
    rv = HPCC_Stream(params, 0 == myRank);
    MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM, 0, comm );
    return errCount;
}
int HPCC_Stream(HPCC_Parms *params, int doIO) {
    register int j;
    double scalar;
    VectorSize = HPCC_LocalVectorSize(params, 3, sizeof(double), 0);
    a = HPCC_XMALLOC(double, VectorSize);
    b = HPCC_XMALLOC(double, VectorSize);
    c = HPCC_XMALLOC(double, VectorSize);
    if (!a || !b || !c) {
        if (c) HPCC_free(c);
        if (b) HPCC_free(b);
        if (a) HPCC_free(a);
        if (doIO) {
            fprintf(outFile, "Failed to allocate memory (%d).\n", VectorSize);
            fclose(outFile);
        }
        return 1;
    }
#ifdef _OPENMP
#pragma omp parallel for
#endif
    for (j=0; j<VectorSize; j++) {
        b[j] = 2.0;
        c[j] = 0.0;
    }
    scalar = 3.0;
#ifdef _OPENMP
#pragma omp parallel for
#endif
    for (j=0; j<VectorSize; j++)
        a[j] = b[j]+scalar*c[j];
    HPCC_free(c);
    HPCC_free(b);
    HPCC_free(a);
    return 0;
}

#include <hpcc.h>
#define N 2000000
int main() {
    float *d_a, *d_b, *d_c;
    float scalar;
    cudaMalloc((void**)&d_a, sizeof(float)*N);
    cudaMalloc((void**)&d_b, sizeof(float)*N);
    cudaMalloc((void**)&d_c, sizeof(float)*N);

dim3 dimBlock(128);
dim3 dimGrid(N/dimBlock.x);
if ( N % dimBlock.x != 0 ) dimGrid.x+=1;

set_array<<<dimGrid,dimBlock>>>(d_b, .5f, N);
set_array<<<dimGrid,dimBlock>>>(d_c, .5f, N);
scalar=3.0f;
STREAM_Triad<<<dimGrid,dimBlock>>>(d_b, d_c, d_a, scalar, N);
cudaThreadSynchronize();

cudaFree(d_a);
cudaFree(d_b);
cudaFree(d_c);
}

__global__ void set_array(float *a, float value, int len) {
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    if (idx < len) a[idx] = value;
}

__global__ void STREAM_Triad(float *a, float *b, float *c, float scalar, int len) {
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    if (idx < len) c[idx] = a[idx]+scalar*b[idx];
}
For STREAM, the Chapel and CUDA implementations match performance.
Leveraging Chapel for GPUs
Leveraging the Language

• Chapel has support for user-defined distributions
  – Recipe for mapping data and computation onto the device
  – Implemented in Chapel source code
  – More flexible than HPL and ZPL’s distributions

• Arrays used on the accelerator are declared using the GPU distribution and domain

```chapel
const ProbDist = new dmap(dist(GPUDist(rank=1, tbSizeX)));
const ProbSpace: domain(1) dmapped ProbDist = [1..m];
var A : [ProbSpace] real;
```

• Depend on `forall` as the main support for data-parallelism on a device

```chapel
forall I in ProbSpace do
  A(I) = ...
```
Leveraging the Language (cont.)

- Low-level support for different memory spaces including shared and constant
- Support for both explicit and implicit data transfers between the host and device
  - Implicit data transfers depend on compiler support
  - Based on simple data-flow analysis
- Ongoing work to support whole-array operations
  - E.g. \( A = B + \text{scalar} \times C; \)
- Support for reduction/scan operations executed on the GPU
Another Example

Coulombic Potential (CP) from Parboil Benchmark Suite
const volmemsz_dom = [1..VOLSIZEY,1..VOLSIZEX];
const gdst = new dmap(new GPUDist(rank=2, tbSizeX=BLOCKSIZEX, tbSizeY=BLOCKSIZEY));
const space : domain(2) dmapped gdst = volmemsz_dom;
var energygrid : [space] = 0.0;
const constgdst = new dmap(new GPUConstDist(rank=1));
const atomspace : domain(1) dmapped constgdst = [1..MAXATOMS];

/* Initialize atominfo from input file */
var atominfo : [atomspace] float4 = ...;
forall (xindex,yindex) in space {
  var energyval = 0.0;
  var (coorx,coory) = (gspacing*xindex, gspacing*yindex);
  for atom in atominfo {
    var (dx,dy) = (coorx-atom.x, coory-atom.y);
    var r_1 = 1.0 / sqrt(dx*dx + dy*dy + atom.z);
    energyval += atom.w * r_1;
  }
  energygrid(yindex, xindex) += energyval;
}
Another Example - Coulombic Potential

```plaintext
const volmemsz_dom = [1..VOLSIZEY,1..VOLSIZEX];
const gdst = new dmap(new GPUDist(rank=2,tbSizeX=BLOCKSIZEX, tbSizeY=BLOCKSIZEY));
const space : domain(2) dmapped gdst = volmemsz_dom;
var energygrid : [space] = 0.0;
const constgdst = new dmap(new GPUConstDist(rank=1));
const atomspace : domain(1) dmapped constgdst = [1..MAXATOMS];

/* initialize atominfo from input file */
var atominfo : [atomspace] float4 = ...;
forall (xindex,yindex) in space {
  var energyval = 0.0;
  var (coorx,coory) = (gspacing*xindex, gspacing*yindex);
  for atom in atominfo {
    var (dx,dy) = (coorx-atom.x, coory-atom.y);
    var r_1 = 1.0 / sqrt(dx*dx + dy*dy + atom.z);
    energyval += atom.w * r_1;
  }
  energygrid(yindex, xindex) += energyval;
}
```

Main GPU distribution used
Another Example - Coulombic Potential

```plaintext
const volmemsz_dom = [1..VOLSIZEY,1..VOLSIZEX];
const gdst = new dmap(new GPUDist(rank=2, tbSizeX=BLOCKSIZEX, 
                           tbSizeY=BLOCKSIZEY));

const space : domain(2) dmapped gdst = volmemsz_dom;
var energygrid : [space] = 0.0;
const constgdst = new dmap(new GPUConstDist(rank=1));
const atomspace : domain(1) dmapped constgdst = [1..MAXATOMS];

/* initialize atominfo from input file */
var atominfo : [atomspace] float4 = ...;
forall (xindex,yindex) in space {
  var energyval = 0.0;
  var (coorx,coory) = (gspacing*xindex, gspacing*yindex);
  for atom in atominfo {
    var (dx,dy) = (coorx-atom.x, coory-atom.y);
    var r_1 = 1.0 / sqrt(dx*dx + dy*dy + atom.z);
    energyval += atom.w * r_1;
  }
  energygrid(yindex, xindex) += energyval;
}
```

Domains created using GPU distribution
Another Example - Coulombic Potential

```plaintext
const volmemsz_dom = [1..VOLSIZEY,1..VOLSIZEX];
const gdst = new dmap(new GPUDist(rank=2,tbSizeX=BLOCKSIZEX,
tbSizeY=BLOCKSIZEY));
const space : domain(2) dmapped gdst = volmemsz_dom;
var energygrid : [space] = 0.0;
const constgdst = new dmap(new GPUConstDist(rank=1));
const atomspace : domain(1) dmapped constgdst = [1..MAXATOMS];

/* initialize atominfo from input file */
var atominfo : [atomspace] float4 = ...
forall (xindex,yindex) in space {
    var energyval = 0.0;
    var (coorx,coory) = (gspacing*xindex, gspacing*yindex);
    for atom in atominfo {
        var (dx,dy) = (coorx-atom.x, coory-atom.y);
        var r_1 = 1.0 / sqrt(dx*dx + dy*dy + atom.z);
        energyval += atom.w * r_1;
    }
    energygrid(yindex, xindex) += energyval;
}
```

Allocate GPU memory and initialize to 0.0.
Another Example - Coulombic Potential

```plaintext
const volmemsz_dom = [1..VOLSIZEY,1..VOLSIZEX];
const gdst = new dmap(new GPUDist(rank=2,tbSizeX=BLOCKSZEX,
                           tbSizeY=BLOCKSIZEY));
const space : domain(2) dmapped gdst = volmemsz_dom;
var energygrid : [space] = 0.0;
const constgdst = new dmap(new GPUConstDist(rank=1));
const atomspace : domain(1) dmapped constgdst = [1..MAXATOMS];

/* initialize atominfo from input file */
var atominfo
forall (xindex,yindex)
    var energyval = 0.0;
    var (coorx,coory) = (gspacing*xindex, gspacing*yindex);
    for atom in atominfo {
        var (dx,dy) = (coorx-atom.x, coory-atom.y);
        var r_1 = 1.0 / sqrt(dx*dx + dy*dy + atom.z);
        energyval += atom.w * r_1;
    }
    energygrid(yindex, xindex) += energyval;
```
Another Example - Coulombic Potential

```plaintext
const volmemsz_dom = [1..VOLSIZEY,1..VOLSIZEX];
const gdst = new dmap(new GPUDist(rank=2,tbSizeX=BLOCKSIZEX,tbSizeY=BLOCKSIZEY));
const space : domain(2) dmapped gdst = volmemsz_dom;
var energygrid : [space] = 0.0;
const constgdst = new dmap(new GPUConstDist(rank=1));
const atomspace : domain(1) dmapped constgdst = [1..MAXATOMS];

/* initialize atominfo from input file */
var atominfo : [atomspace] float4 = ...;
forall (xindex,yindex) in space {
    var energyval = 0.0;
    var coorx,coory = (gspacing*xindex, gspacing*yindex);
    for var (dx, dy) = (coorx-atom.x, coory-atom.y);
        var r_1 = 1.0 / sqrt(dx*dx + dy*dy + atom.z);
        energyval += atom.w * r_1;
    }
    energygrid(yindex, xindex) += energyval;
}
```

Allocate GPU Constant memory
Another Example - Coulombic Potential

```c
const volmemsz_dom = [1..VOLSIZEY,1..VOLSIZEX];
const gdst = new dmap(new GPUDist(rank=2,tbSizeX=BLOCKSIZEX,
                             tbSizeY=BLOCKSIZEY));
const space : domain(2) dmapped gdst = volmemsz_dom;
var energygrid : [space] = 0.0;
const constgdst = new dmap(new GPUConstDist(rank=1));
const atomspace : domain(1) dmapped constgdst = [1..MAXATOMS];

/* initialize atominfo from input file */
var atominfo : [atomspace] float4 = ...;
forall (xindex,yindex) in space {
    var energyval = 0.0;
    var (coorx,coory) = (gspacing*xindex, gspacing*yindex);
    for atom in atominfo {
        var (dx,dy) = (coorx-atom.x, coory-atom.y);
        var r_1 = 1.0 / sqrt(dx*dx + dy*dy + atom.z);
        energyval += atom.w * r_1;
    }
    energygrid(yindex, xindex) += energyval;
}
```

The main kernel
Coulombic Potential (CP) (Nvidia GTX 280 GPU)

Execution Time (ms)

- CUDA
- Tuned Chapel w/ Explicit Transfer
- Tuned Chapel w/ Implicit Transfer

Legend:
- □ I/O
- ■ Compute
Chapel Team’s Next Steps

- Expand our set of supported distributions
- Continue to improve performance
- Continue to add missing features
- Expand the set of codes that we are studying
- Expand the set of architectures that we are targeting
- Support the public release
- Continue to support collaborations and seek out new ones
- Continue to expand our team
Questions?

- What we do
- Project Status
- Who we are
- Collaboration Sidebar: Chapel on CPU+GPU