HPCC STREAM and RA in Chapel Performance and Potential

Steve Deitz
Cray Inc.
What is Chapel?

- A new parallel language
  - Under development at Cray Inc.
  - Supported through the DARPA HPCS program
- Goals
  - **Improve programmer productivity**
  - Improve the programmability of parallel computers
  - Match or improve performance of MPI/UPC/CAF
  - Provide better portability than MPI/UPC/CAF
  - Improve robustness of parallel codes
  - Support multi-core and multi-node systems
Outline

• What is Chapel?
• Chapel’s Parallel Programming Model
• HPCC STREAM Triad in Chapel
• HPCC RA in Chapel
• Summary and Future Work
Fragmented vs. Global-View: Definitions

- Programming model
  
  *The mental model of a programmer*

- Fragmented models
  
  *Programmers take point-of-view of a single processor/thread*

- SPMD models (Single Program, Multiple Data)
  
  *Fragmented models with multiple copies of one program*

- Global-view models
  
  *Programmers write code to describe computation as a whole*
### 3-Point Stencil Example (n=6)

<table>
<thead>
<tr>
<th>Initial state</th>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Iteration 3</th>
<th>Steady state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0 0.0 0.0 0.0 0.0 12.0</td>
<td>2.0 1.0 0.0 0.0 6.0 12.0</td>
<td>2.0 1.0 0.5 3.0 6.0 12.0</td>
<td>2.0 1.25 2.0 3.25 7.5 12.0</td>
<td>2.0 4.0 6.0 8.0 10.0 12.0</td>
</tr>
</tbody>
</table>
Global-View vs. Fragmented Computation

**Global-View**

\[
\frac{(\text{[image]} + \text{[image]})}{2}
\]

\[
\frac{(\text{[image]} + \text{[image]})}{2}
\]

\[
\text{[image]}
\]

**Fragmented**

\[
\frac{(\text{[image]} + \text{[image]})}{2}
\]

\[
\frac{(\text{[image]} + \text{[image]})}{2}
\]

\[
\text{[image]}
\]
3-Point Stencil Example: Code

Global-View vs. Fragmented Code

**Global-View**

```chapel
def main() {
    var n = 1000;
    var A, B: [1..n] real;

    forall i in 2..n-1 do
        B(i) = (A(i-1)+A(i+1))/2;
}
```

**Fragmented**

```chapel
def main() {
    var n = 1000;
    var me = commID(), p = commProcs(), myN = n/p, myLo = 1, myHi = myN;
    var A, B: [0..myN+1] real;

    if me < p {
        send(me+1, A(myN));
        recv(me+1, A(myN+1));
    } else myHi = myN-1;
    if me > 1 {
        send(me-1, A(0));
        recv(me-1, A(1));
    } else myLo = 2;

    for i in myLo..myHi do
        B(i) = (A(i-1)+A(i+1))/2;
}
```

Assumes $p$ divides $n$
NAS MG Stencil in Fortran + MPI, 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 inds in S.domain do
        ind = [i,j,k] in S.domain |
            S(inds) = + reduce [offset in Stencil] (W3D(offset) * R(inds + offset*R.stride));
}
```
Outline

- What is Chapel?
- Chapel’s Parallel Programming Model
- HPCC STREAM Triad in Chapel
- HPCC RA in Chapel
- Summary and Future Work
Given: $m$-element vectors $A, B, C$

Compute: \[
\text{forall } i \text{ in } 1..m \text{ do } \\
A(i) = B(i) + \alpha \times C(i);
\]
Given:  

\[ m \]-element vectors \( A, B, C \)

Compute:  

\[
\text{forall } i \text{ in } 1..m \text{ do } \\
A(i) = B(i) + \alpha \times C(i);
\]
Given: \( m \)-element vectors \( A, B, C \)

Compute: \( \text{forall } i \text{ in } 1..m \text{ do} \)
\[
A(i) = B(i) + \alpha \times C(i);
\]
STREAM Triad in Chapel: Single Locale

Given: \( m \)-element vectors \( A, B, C \)

Compute: \( \text{forall } i \text{ in } 1..m \text{ do} \)
\[
A(i) = B(i) + \alpha \times C(i);
\]

\[
\text{config const } m: \text{int}(64) = \ldots;
\]
\[
\text{const alpha: real } = 3.0;
\]
\[
\text{const ProblemSpace: domain}(1, \text{int}(64)) = [1..m];
\]
\[
\text{var A, B, C: [ProblemSpace] real;}
\]

\( \text{forall } i \text{ in } \text{ProblemSpace do} \)
\[
A(i) = B(i) + \text{alpha} \times C(i);
\]
Given:  \( m \)-element vectors \( A, B, C \)

Compute:  \[
\text{forall } i \text{ in } 1..m \text{ do } \\
A(i) = B(i) + \alpha \times C(i);
\]

```chapel
config const m: int(64) = ...;
const alpha: real = 3.0;
const ProblemSpace: domain(1,int(64)) = [1..m];
var A, B, C: [ProblemSpace] real;

A = B + alpha \times C;
```

More concise variation using whole array operations
STREAM Triad in Chapel: Single Locale

Given: \( m \)-element vectors \( A, B, C \)

Compute: \( \text{forall } i \text{ in } 1..m \text{ do } \)
\[
A(i) = B(i) + \alpha \times C(i);
\]

\[
\text{config const } m: \text{ int}(64) = ...;
\]
\[
\text{const alpha: real } = 3.0;
\]
\[
\text{const ProblemSpace: domain}(1, \text{int}(64)) = [1..m];
\]
\[
\text{var A, B, C: [ProblemSpace] real;}
\]

\[
\text{forall (a,b,c) in (A,B,C) do }
\]
\[
a = b + \alpha \times c;
\]

Variation that iterates directly over the arrays
STREAM Triad in Chapel: Multi-Locale

Given:  \( m \)-element vectors \( A, B, C \)

Compute:  \( \text{forall } i \text{ in } 1..m \text{ do} \)
\[
A(i) = B(i) + \alpha \times C(i);
\]

```chapel
config const m: int(64) = ..., tpl = ...;
const alpha: real = 3.0;
const BlockDist = new Block(1,int(64),[1..m],tpl);
const ProblemSpace: domain(1, int(64))
    distributed BlockDist = [1..m];
var A, B, C: [ProblemSpace] real;

forall (a,b,c) in (A,B,C) do
    a = b + alpha * c;
```
What is a Distribution?

A “recipe” for distributed arrays that...
Instructs the compiler how to Map the global view...
...to a fragmented, per-processor implementation
Given: \( m \)-element vectors \( A, B, C \)

Compute: \[
\text{forall } i \text{ in } 1..m \text{ do } \\
\quad A(i) = B(i) + \alpha \ast C(i);
\]

\[
\begin{align*}
\text{config const } m & : \text{int(64)} = \ldots, \text{tpl} = \ldots; \\
\text{const alpha: real} & = 3.0; \\
\text{const BlockDist} & = \text{new Block}(1, \text{int(64)}, [1..m], \text{tpl}); \\
\text{const ProblemSpace: domain}(1, \text{int(64)}) & \\
& \quad \text{distributed BlockDist} = [1..m]; \\
\text{var A, B, C: [ProblemSpace] real};
\end{align*}
\]

\[
\text{forall \ ((a,b,c) in (A,B,C) do } \\
\quad a = b + \text{alpha} \ast c;
\]
HPCC STREAM Performance

Performance of STREAM in Chapel

Number of Locales

GB/s

1 TPL
2 TPL
3 TPL
4 TPL
5 TPL
HPCC STREAM Efficiency

Efficiency of STREAM in Chapel

<table>
<thead>
<tr>
<th>Number of Locales</th>
<th>1 TPL</th>
<th>2 TPL</th>
<th>3 TPL</th>
<th>4 TPL</th>
<th>5 TPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>2</td>
<td>98%</td>
<td>98%</td>
<td>98%</td>
<td>98%</td>
<td>98%</td>
</tr>
<tr>
<td>4</td>
<td>96%</td>
<td>96%</td>
<td>96%</td>
<td>96%</td>
<td>96%</td>
</tr>
<tr>
<td>8</td>
<td>94%</td>
<td>94%</td>
<td>94%</td>
<td>94%</td>
<td>94%</td>
</tr>
<tr>
<td>16</td>
<td>92%</td>
<td>92%</td>
<td>92%</td>
<td>92%</td>
<td>92%</td>
</tr>
<tr>
<td>32</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
</tr>
<tr>
<td>64</td>
<td>88%</td>
<td>88%</td>
<td>88%</td>
<td>88%</td>
<td>88%</td>
</tr>
<tr>
<td>128</td>
<td>86%</td>
<td>86%</td>
<td>86%</td>
<td>86%</td>
<td>86%</td>
</tr>
<tr>
<td>256</td>
<td>84%</td>
<td>84%</td>
<td>84%</td>
<td>84%</td>
<td>84%</td>
</tr>
<tr>
<td>512</td>
<td>82%</td>
<td>82%</td>
<td>82%</td>
<td>82%</td>
<td>82%</td>
</tr>
<tr>
<td>1024</td>
<td>80%</td>
<td>80%</td>
<td>80%</td>
<td>80%</td>
<td>80%</td>
</tr>
</tbody>
</table>
Simple example

```chapel
var Dist: Block(1,int(64));
var Dom: domain(1,int(64))
    distributed Dist;
var Arr: [Dom] int;
```

Reference to local data requires communication

```chapel
on Locales(1) {
    Arr(5) = 0;
}
```
Outline

• What is Chapel?
• Chapel’s Parallel Programming Model
• HPCC STREAM Triad in Chapel
• HPCC RA in Chapel
• Summary and Future Work
Given: $m$-element table $T$ (where $m = 2^n$)

Compute: \(\text{forall } r \text{ in RandomUpdates do}
T(r \& (m-1)) ^= r;\)
Given: \( m \)-element table \( T \) (where \( m = 2^n \))

Compute: \( \text{forall } r \text{ in RandomUpdates do } T(r \& (m-1)) ^= r; \)
RA in Chapel: Single Locale

Given: \( m \)-element table \( T \) (where \( m = 2^n \))

Compute: \textbf{forall} \( r \) in \textbf{RandomUpdates} \textbf{do}\\
\( T(r \& (m-1)) ^= r; \)

\textbf{config} \textbf{const} \( m = \ldots, \ N_U = \ldots; \)
\textbf{const} \( \text{TableSpace}: \text{domain}(1,\text{uint}(64)) = [0..m-1], \)
\( \text{Updates}: \text{domain}(1,\text{uint}(64)) = [0..N_U-1]; \)
\textbf{var} \( T: [\text{TableSpace}] \text{uint}(64); \)

\textbf{forall} \( (i,r) \) in (\text{Updates,RAStream()}) \textbf{do}\\
\( T(r \& (m-1)) ^= r; \)
RA in Chapel: Multi-Locale

Given: \( m \)-element table \( T \) (where \( m = 2^n \))

Compute: \( \forall r \in \text{RandomUpdates} \) do
\[
T(r \& (m-1)) \ ^= r;
\]

config const \( m = \ldots, \) N_U = \ldots, tpl = \ldots;
const TableDist = new Block(1,uint(64),[0..m-1],tpl),
UpdateDist = new Block(1,uint(64),[0..N_U-1],tpl),
TableSpace: domain(1,uint(64))
    distributed TableDist = [0..m-1],
Updates: domain(1,uint(64))
    distributed UpdateDist = [0..N_U-1];
var T: [TableSpace] uint(64);

\( \forall (i,r) \in \text{(Updates,RASStream())} \) do
    on \( T(r \& (m-1)) \) do
        \( T(r \& (m-1)) \ ^= r; \)
Given: \( m \)-element table \( T \) (where \( m = 2^n \))

Compute: \( \forall r \in \text{RandomUpdates do} \)
\[
T(r \& (m-1)) ^= r;
\]

`config const m = ..., N_U = ..., tpl = ...;
const TableDist = new Block(1,uint(64),[0..m-1],tpl),
UpdateDist = new Block(1,uint(64),[0..N_U-1],tpl),
TableSpace: domain(1,uint(64))
  distributed TableDist = [0..m-1],
Updates: domain(1,uint(64))
  distributed UpdateDist = [0..N_U-1];`

`var T: [TableSpace] uint(64);`

`forall (i,r) in (Updates,RAStream()) do
  on T.domain.dist.ind2loc(r & (m-1)) do
    T(r & (m-1)) ^= r;`

Call ind2loc method directly
HPCC RA Performance

Performance of RA in Chapel

Number of Locales (or local for optimized on 1 locale)

GUP/s

1 TPL
2 TPL
3 TPL
4 TPL
5 TPL
Efficiency of RA in Chapel

% Efficiency of scaled best 1-locale GUP/s vs Number of Locales (or local for optimized on 1 locale)

- 1 TPL
- 2 TPL
- 3 TPL
- 4 TPL
- 5 TPL
Efficiency of RA in Chapel

Number of Locales

% Efficiency of scaled best 2-locale GUP/s

1 TPL
2 TPL
3 TPL
4 TPL
5 TPL
Simple example

```chapel
var Arr: [Dom] int;
var r: int;
on Locales(1) {
  Arr(r) ^= r;
}
```
Outline

- What is Chapel?
- Chapel’s Parallel Programming Model
- HPCC STREAM Triad in Chapel
- HPCC RA in Chapel
- Summary and Future Work
The global-view programming model is easy to use.

- Shorter, more concise code
- Separation of concerns (partitioning)
- Easy to change data distributions

Distributions implement the global-view model.

- Flexible mechanism for experimentation
- Implementation of distributions is in Chapel
Future Work

- Optimizations
  - Within the compiler
  - Within the runtime
  - Within the distributions

- Complete implementation of Block distribution

- Implement new distributions
  - Cyclic, BlockCyclic, RecursiveBisection

- Experiment with variations of STREAM and RA
Questions?