Global HPCC Benchmarks in Chapel: STREAM Triad, Random Access, and FFT

HPC Challenge BOF, SC06
Class 2 Submission
November 14, 2006

Brad Chamberlain, Steve Deitz,
Mary Beth Hribar, Wayne Wong
Chapel Team, Cray Inc.
Overview

- **Chapel**: Cray’s HPCS language

- Our approach to the HPC Challenge codes:
  - performance-minded
  - clear, intuitive, readable
  - general across…
    - types
    - problem parameters
    - modular boundaries
Code Size Summary

STREAM
Triad

Random
Access

FFT

Reference Version
- Framework
- Computation

Chapel Version
- Prob. Size (common)
- Results and output
- Verification
- Initialization
- Kernel declarations
- Kernel computation
Chapel Code Size Summary

<table>
<thead>
<tr>
<th>Category</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>STREAM Triad</td>
<td>86</td>
</tr>
<tr>
<td>Random Access</td>
<td>124</td>
</tr>
<tr>
<td>FFT</td>
<td>156</td>
</tr>
</tbody>
</table>

- **Problem Size (common)**: Blue
- **Results and output**: Light gray
- **Verification**: Cyan
- **Initialization**: Red
- **Kernel declarations**: Green
- **Kernel computation**: Yellow
Chapel Code Size Summary

- **Problem Size (common)**
- **Results and output**
- **Verification**
- **Initialization**
- **Kernel declarations**
- **Kernel computation**

**STREAM Triad**
- Problem Size: 593
- Results and output
- Verification
- Initialization
- Kernel declarations
- Kernel computation

**Random Access**
- Problem Size: 863
- Results and output
- Verification
- Initialization
- Kernel declarations
- Kernel computation

**FFT**
- Problem Size: 1299
- Results and output
- Verification
- Initialization
- Kernel declarations
- Kernel computation
STREAM Triad Overview

```
const ProblemSpace: domain(1) distributed(Block) = [1..m];
var A, B, C: [ProblemSpace] elemType;

A = B + alpha * C;
```
STREAM Triad Overview

```
const ProblemSpace: domain(1) distributed(Block) = [1..m];
var A, B, C: [ProblemSpace] elemType;
A = B + alpha * C;
```

- Declare a 1D arithmetic *domain* (first-class index set)
- Specify its distribution
- Use domain to declare distributed arrays
- Express computation using *promoted* scalar operators and whole-array references ⇒ parallel computation
Random Access Overview

\[ [i \text{ in TableSpace}] \ T(i) = i; \]

\[ \text{forall block in subBlocks(updateSpace) do} \]
\[ \text{for } r \text{ in RAStream(block.numIndices, block.low) do} \]
\[ T(r \& \text{indexMask}) ^= r; \]
Random Access Overview

[i in TableSpace] T(i) = i;

forall block in subBlocks(updateSpace) do
  for r in RAStream(block.numIndices, block.low) do
    T(r & indexMask) ^= r;

Express table updates using forall- and for-loops

Random stream expressed modularly using an iterator

iterator RAStream(numvals, start:randType = 0): randType {
  var val = getNthRandom(start);
  for i in 1..numvals {
    getNextRandom(val);
    yield val;
  }
}
FFT Overview (radix 4)

for i in [2..log2(numElements)) by 2 {
    const m = span*radix, m2 = 2*m;

    forall (k,k1) in (Adom by m2, 0..) {
        var wk2 = ..., wk1 = ..., wk3 = ...;

        forall j in [k..k+span) do
            butterfly(wk1, wk2, wk3, A[j..j+3*span by span]);

        wk1 = ...; wk3 = ...; wk2 *= 1.0i;

        forall j in [k+m..k+m+span) do
            butterfly(wk1, wk2, wk3, A[j..j+3*span by span]);
    }

    span *= radix;
}

def butterfly(wk1, wk2, wk3, inout A:[1..radix]) { ... }
FFT Overview (radix 4)

for i in [2..\log_2(\text{numElements})] by 2 {
  const m = \text{span} \times \text{radix}, m_2 = 2 \times m;

  forall (k,k1) in (\text{Adom by } m_2, 0..) {
    var wk2 = \ldots, wk1 = \ldots, wk3 = \ldots;

    forall j in [k..k+\text{span}) do
      butterfly(wk1, wk2, wk3, A[j..j+3\times\text{span by } \text{span}]);

    wk1 = \ldots; wk3 = \ldots; wk2 *= 1.0i;

    forall j in [k+m..k+m+\text{span}) do
      butterfly(wk1, wk2, wk3, A[j..j+3\times\text{span by } \text{span}]);
  }

  \text{span} *= \text{radix};
}

def butterfly(wk1, wk2, wk3, inout A:[1..\text{radix}]) { \ldots }
Chapel Compiler Status

- All codes compile and run with our current Chapel compiler
  - focus to date has been on…
    - prototyping Chapel, not performance
    - targeting a single locale
  - platforms: Linux, Cygwin (Windows), Mac OS X, SunOS, …

- No meaningful performance results yet
  - written report contains performance discussions for our codes

- Upcoming milestones
  - **December 2006**: limited release to HPLS team
  - **2007**: work on distributed-memory execution and optimizations
  - **SC07**: intend to have publishable performance results for HPCC`07
Summary

- Have expressed HPCC codes attractively
  - clear, concise, general
  - express parallelism, compile and execute correctly on one locale
  - benefit from Chapel’s global-view parallelism
  - utilize generic programming and modern SW Engineering principles

- Our written report contains:
  - complete source listings
  - detailed walkthroughs of our solutions as Chapel tutorial
  - performance notes for our implementations

- Report and presentation available at our website:
  http://chapel.cs.washington.edu

- We’re interested in your feedback:
  chapel_info@cray.com
Backup Slides
Compact High-Level Code...

- **EP**
  - Language: Zeplin (ZPL), Fortran + MPI (F+MPI)
  - Lines of Code:
    - Communication: F+MPI 249, ZPL 204
    - Declarations: F+MPI 332, ZPL 249
    - Computation: F+MPI 128, ZPL 135

- **CG**
  - Language: Zeplin (ZPL), Fortran + MPI (F+MPI)
  - Lines of Code:
    - Communication: F+MPI 54, ZPL 17
    - Declarations: F+MPI 36, ZPL 25
    - Computation: F+MPI 79, ZPL 89

- **FT**
  - Language: Zeplin (ZPL), Fortran + MPI (F+MPI)
  - Lines of Code:
    - Communication: F+MPI 152, ZPL 72
    - Declarations: F+MPI 22, ZPL 152
    - Computation: F+MPI 80, ZPL 111

- **MG**
  - Language: Zeplin (ZPL), Fortran + MPI (F+MPI)
  - Lines of Code:
    - Communication: F+MPI 566, ZPL 202
    - Declarations: F+MPI 242, ZPL 87
    - Computation: F+MPI 87, ZPL 79

- **IS**
  - Language: Zeplin (ZPL), C + MPI (C+MPI)
  - Lines of Code:
    - Communication: C+MPI 152, ZPL 111
    - Declarations: C+MPI 79, ZPL 72
    - Computation: C+MPI 80, ZPL 80
...need not perform poorly

See also Rice University’s recent D-HPF work…

HPCC BOF, SC06