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

(Revision 1.1 — HPCC BOF, SC06)

Bradford L. Chamberlain
Steven J. Deitz
Mary Beth Hribar
Wayne A. Wong

The first two lines define a domain and array used to represent the distributed table, T, that is randomly accessed

values are required in certain language contexts, such as when specifying a scalar type's bit-width or an array's `computeProblemSize()` defined in the *HPCCProblemSize* module. This routine takes as its arguments the element type being stored and the number of arrays to be allocated and returns the problem size that will fill the appropriate fraction of system memory as an

The syntax for specifying a domain's definition is as follows:

```
domain [ ( <domain-args> ) ] [
```

a case this simple, even a modest compiler ought be able to eliminate the dynamic bounds check without any trouble.

```
    }
  }
```

The body of `printConfiguration()` is guarded by a conditional statement whose test checks the value of the `config const` *printParams*. Conditionals in Chapel are similar to those in other modern languages. The first statement of the conditional calls a helper routine defined in the *HPCCProblemSizes* module which prints out some information about the problem size, taking in

This routine starts by constructing an instance of the *RandomStream* class which is defined in Chapel's standard *Random*

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

This loop is an example of expressing parallelism in a simple, architecturally-neutral way, as alluded to in Section 3. Our goal is to create parallel work such that each locale has an appropriate number of threads, each performing a fraction of the locale's random updates. To this end, we create a parallel outer loop to describe the parallelism across the machine resources and a serial inner loop to express the updates owned by that resource.

that it is an array (due to the square brackets) but with no element type specified. This requires that the argument

An Explicit Module Declaration In all of our previous examples, we have defined modules that are not used by any other modules, and therefore have relied on the convenience of having the compiler generate a module name from the filename. For the *RARandomStream* module, we want to `use` it from other modules and would also like the option of creating several implementations in different files, so we scope the code and name the module as follows in to be a *randType*

The final new concept is the use of a type cast to coerce a *tuple*

Locale Types and Variables

updates to the global table one element at a time, requiring a lot of fine-grain communication. For architectures that

At one end of the spectrum, even on a parallel machine with a flat shared memory like the Cray MTA, the clean expression of the computation as given here is unlikely to perform optimally. In Appendix C.2, we provide an alternate Chapel implementation of FFT that is based on last year's HPCC submission for the MTA2 [16]. In that implementation, several specialized versions of the inner loops were created in order to maximize outer-loop parallelism, take advantage of twiddle values with zero components and tune for the MTA's characteristics. While we might hope that our more straightforward implementation of the benchmark would pone close to achieving the

In conclusion, it is worth noting that while these benchmarks have demonstrated many of Chapel's productivity features for global-view programming and software engineering, they remain rather restricted in terms of the parallel

[16] Petr Konecny, Simon Kahan, and John Feo. SC05 HPCChallenge class 2 award—Cray MTA2. November 2005.

[17] John Mellor-Crummey. Personal communication.

[18] Robert W. Numrich and John Reid. Co-array Fortran for parallel programming. *SIGPLAN Fortran Forum*,

Walk(w)1mbe4039(,)-7llenge,Dongo-arng.

A STREAM Triad Chapel Code

```
use Time;
use Types;
use Random;

use HPCCProblemSize;


param numVectors = 3;
type elemType = real(64);

```

B Random Access Chapel Code

B.1 Random Access Computation

```
use Time;

use HPCCProblemSize;
use RARandomStream;
```

B.2 Random Access Random Stream Generator

```
module RARandomStream {
  param randWidth = 64;
  type randType = uint(64);

  const bitDom = [0..randWidth),
        m2: [bitDom] randType = computeM2Vals(randWidth);


  iterator RAStream(numvals, start:randType = 0): randType {

             9uint
      RsaindWidth);
      3constconstuint

      RsSintD[(9)]TJ/F12 769715f 19.13inD[(uint)]TJ/F10 7.4.35f 19.13n)mStream {
      9          paramconst
                 m2juintandType 0x2Width);
        9uint

    m2juintandType 0Width);
     9uintTD[(3)]TJ/F12 7.97 Tf 33.47do {
         9uintm2>> j)mS600(&mS600(1)m TD[(9)]TJ/F12 781.27f 19.13then)[(type)]TJ/F10 7.97 Tf 23.91am(bitDoˆe)-600(m2flјD[((64)


         RsintTD[(9)]TJ/F12 766 Tf  19.13inouD[(const)]TJ/F10 7.97 Tf 52.58x)mS600(e)-600({)]TJ/F1 4.134.26 -45.7 -21.378 TD[(
9uint
idth);constuint
```

Ccom[TwiddleDom]00(commType,)-TJ/F1 4.98 Tf -38.64 -18.93[(C)-9.134J/F12 7.97 Tf 9.912 528C

C

C FFT Chapel CodeC 1D radix-4 FFT Chapel CodeCCC

2CC

3CCCCCCCC = 4;CCC = 2;

11CC =CC128);CC coistC = computeProblemSize-elemType, numVectors, returnLog2 =CCCCC coistC coistCC

```
  fillRandom(z, seed);

  if (printArrays) {
    writeln("After initialization, Twiddles is: ", Twiddles, "\n");
    writeln("z is: ", z, "\n");
  }
}
```

7.mz, "\n"y3def9.46 TD[(66)]TJ/F10 1.97 Tf 1compute00(",)-6€",)-6printArrays) {

65

```
    }
    span *= radix;
```

C.2 Chapel Version of HPCC'05 1D radix-4 FFT for the MTA

In this section, we provide an alternate Chapel implementation of FFT, based on last year's HPC Challenge implementation for the Cray MTA [16]. This version of the code is provided to suggest some of the transformations that an aggressive performance-minded programmer might utilize to best take advantage of architectural characteristics.

```
    if (i <= n/2) then
      cftmd1(span, Z, Twiddles);
    else
      cftmd2(span, Z, Twiddles);
    span *= radix;
  }

  if (radix*span == Z.numElements) then
    forall j in [0..span) do
      butterfly(1.0, 1.0, 1.0, Z[j..j+3*span by span]);
  else
    forall j in [0..span) {
      const a = Z(j),
            b = Z(j+span);
      Z(j)      = a + b;
      Z(j+span) = a - b;
    }
}


def printConfiguration() {
  if (printParams) then printProblemSize(elemType, numVectors, m);
}


def initVectors(Twiddles, z) {
  computeTwiddles(Twiddles);
  bitReverseShuffle(Twiddles);

  fillRandom(z, seed);

  if (printArrays) {
    writeln("After initialization, Twiddles is: ", Twiddles, "\n");
    writeln("z is: ", z, "\n");
  }
}
```

96D[(84)]TJ/F10 7.97 Tf 19.52 0 TD[(computeTwiddles(Twiddles600(-);)]TJ/F1 4.98 Tf 97seed);
JD[T1dcompute T73TD[(*)].78 10 7.97 Tf 28.69 048.217.97 Tf 19.5846 TD[(70)]TJ/F1248.217f -9.96 lt64 0 TD[(const2.))

```
        valReverse = bitRotLeft(valReverse64, revBits);
  return valReverse: valType;
}


def verifyResults(z, Z, Twiddles) {
  if (printArrays) then writeln("After FFT, Z is: ", Z, "\n");

  Z = conjg(Z) / m;
  bitReverseShuffle(Z);
  dfft(Z, Twiddles);

  if (printArrays) then writeln("After inverse FFT, Z is: ", Z, "\n");

  var
```

```
x0 = x1 + x3*1.0i;
A(5) = wk1r * (x0.re - x0.im, x0.re + x0.im):complex;
x0 = (x3.im + x1.re, x3.re - x1.im):complex;
A(7) = wk1r * (x0.im - x0.re, x0.im + x0.re):complex;
forall (j,k1)
```

```
    return;
  g

  forall j in [0..span) {
    forall (k,k1) in ([m2..numElems) by m2, 1..) {
      const wk2 = W(k1),
            wk1 = W(k1 + k1),
            wk3 = interpIm(wk1, wk2);
      butterfly(wk1, wk2, wk3, A[j+k..j+k+3*span by span]);
    g

    forall (k,k1) in ([m2..numElems) by m2, 1..) {
      const wk2 = W(k1),
            wk1 = W(2*k1 + 1),
            wk3 = interpRe(wk1, wk2);
      wk2 = wk2*1.0i;

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


def cftmd21(span, A, W) {
  const m = radix*span,
        m2 = 2*m;

  for (k,k1) in ([m2..A.numElements) by m2, 1..) {
    var wk2 = W(k1),
        wk1 = W(2*k1),
        wk3 = interpIm(wk1, wk2);

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

    wk1 = W(2*k1 + 1);
    wk3 = interpRe(wk1, wk2);
    wk2 = wk2*1.0i;

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


def interpIm(a, b)
  return (a.re - 2*b.im*a.im, 2*b.im*a.re - a.im):complex;


def interpRe(a, b)
  return (a.re - 2*b.re*a.im, 2*b.re*a.re - a.im):complex;
```

D HPCC Problem Size Computation Code

1