Language-bindings for a Data-Parallel Runtime

Bryan Carpenter, Geoffrey Fox, Xinying Li,
Yuhong Wen, Guansong Zhang

Northeast Parallel Architectures Centre,
Syracuse University,
Syracuse, New York

{dbc,gcf,xli,wen,zgs}@npac.syr.edu

Introduction

- Widely accepted that data-parallel programming has a vital role in HPC.
- "Production" parallel computing still dominated by low-level SPMD programming. So far HPF has had limited impact.
- Will suggest a middle road between HPF and SPMD programming, combining some data parallel language features of HPF with free use of established SPMD libraries for distributed arrays—**HPspmd**.

Libraries

PCRC

Parallel Compiler Runtime Consortium

- DARPA supported.
- Aimed to develop a public domain infrastructure for runtime support of high-level parallel languages.
- Targetted languages include parallel versions Fortran and C++.
- Common library should be shared by different languages and programming models.
- Partners: NPAC, Rice, Maryland, Austin, Indiana, CSC, Rochester, Florida.

PCRC-related libraries

Data-parallel run-time libraries developed by PCRC collaborators include

- Multiblock PARTI
- CHAOS
- Meta-Chaos
- DAGH
- Adlib

Multiblock PARTI

- Developed at University of Maryland
- Library to support applications involving structured meshes—eg nested meshes for multigrid codes or irregularly coupled, multiblock codes.
- Basic collective communication primitives are `Regular_Section_Copy` and `Overlap_Cell_fill`.
- Was integrated with the Syracuse Fortran 90D compiler.

CHAOS

- Subsumes PARTI, with emphasis on irregular problems.
- Arrays preprocessed (data arrays repartitioned to improve locality, indirection arrays reduced to local subscripts, etc) and communication schedule computed in **inspector** phase.
- Arithmetic performed in series of **executor** loops.

Meta-Chaos

- Developed at Maryland during PCRC.
- Library for transferring data between distributed data structures owned by different data-parallel runtime libraries.
- Currently has interfaces for HPF, the PARTI/CHAOS libraries, and Tulip.

DAGH

- Developed at Texas, Austin.
- Distributed Adaptive Grid Hierarchy.
- Application-oriented library for integrating time-dependent PDEs, using adaptive grids. Exploits space (and time) locality.
- Directly manages memory and exchange of sub-mesh boundaries for a hierarchy of grids of differing refinement.

Adlib

- Early development in **shpf** project at Southampton, UK. Reimplemented at Syracuse in PCRC.
- Designed specifically as runtime support for HPF translation.
- Kernel is C++ class library, using MPI. Fortran, C++ and Java interfaces available or under development.
- Communication primitives similar to Multiblock PARTI, plus all F90-style transformational intrinsic for array arithmetic (and gather/scatter for irregular).
- Array descriptor and communication library supports full HPF 1.0 distributed array model.

Distributed Array Descriptor

Designing an effective DAD for HPF is non-trivial. Must describe

- Many distribution formats: simple block, simple cyclic, block-cyclic, collapsed, (general block, ...)
- Various alignment options: alignments to arbitrary regular sections of templates including embedded alignments, collapsed and replicated alignment.
- Regular array sections constructed with arbitrary combinations of scalar and triplet subscripts.

Adlib DAD

DAD includes

- Type and rank information.
- Description of process subgroup to which array is mapped
- For each dimension, distribution format and alignment information.
- For each dimension, local memory stride.

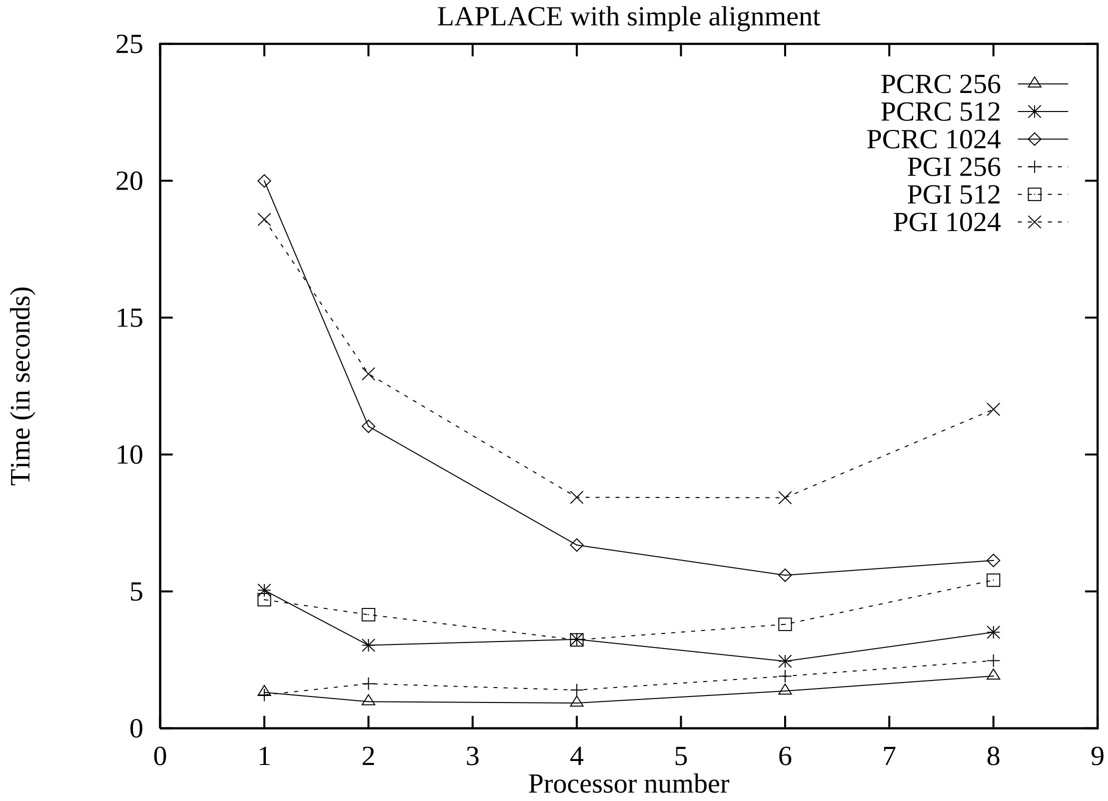
Early Benchmarks

Compared the new NPAC compiler with the PGI HPF compiler, version 2.0. The programs were run on the IBM SP2 at NPAC.

The Laplace benchmark performs Jacobi relaxation on 256×256, 512×512 and 1024×1024 arrays, distributed blockwise.

Compilers achieve about the same performance on a single node, but generally our compiler exhibits better speedup on multiple processors, presumably due to more effective handling of communication.

Laplace Update



HPspmd

Advantages of HPF

- Easy to understand semantics: equivalent to conventional Fortran program. Directives only change performance.
- Parallelism can be expressed explicitly through whole-array operations of Fortran 90/95.
- Performance can be tuned by changing a few directives to alter the distribution format of arrays.

Problems with HPF

- Works best for algorithms that have essentially SIMD parallelism—identical updates across whole arrays. Many algorithms—even data parallel ones—easier to express with looser coupling between updates of the individual array elements.
- Many algorithms have completely irregular patterns of access to the elements of (nevertheless regular) arrays. May be expressible using (eg) FORALL, but hard for compiler to generate good code.
- Other applications simply don't have such regular arrays as major data structures—more general meshes needed.

HPspmd Goals

Aim for a flexible hybrid of the data parallel and SPMD approaches, bringing together some of the established technologies for parallel processing.

Incorporates two complementary sectors:

- Syntax extensions for description of, and access to elements of, distributed arrays.
- Class library bindings for various communication paradigms.

Features

- **Data parallel**: Generalized, HPF-like distributed arrays as language primitives.
- **SPMD**: processors have freedom to independently execute complex procedures on local data.
- **SPMD**: All access to non-local array elements goes through library functions.
- New **distributed control** constructs facilitate access to local elements of arrays, and simplify parallel array traversals.

Language Model

- Distributed array model semantically equivalent to to the HPF array model.
- Add **distributed control** constructs. HPF 2.0 and precursors have similar mechanisms (`ON HOME`) as **optional** directives. HPspmd generally **requires** specification of execution location.
- Switch between data-parallel and low-level SPMD programming is a change in point of view—no **extrinsic interface** barrier.

Communication Libraries

Need at least:

- Regular collective (Adlib-like)
- Irregular collective (CHAOS-like)
- One-sided access to array sections (GA-like)
- Low-level SPMD (MPI)
- Bindings to higher-level application-oriented libraries (DAGH, ScaLAPACK, . . .)

Language Bindings

Prototyped, in progress, or suggested:

- C++: ad++
- Java: HPJava
- Fortran: FAST Fortran

ad++

- Native user-level interface to Adlib.
- Prototype of our language model. It extends C++ only by class libraries and macros.
- Distributed arrays implemented as container classes.
- Introduces the three distributed control constructs, **on**, **at** and **overall**, implemented as macros.
- The current ad++ is inefficient (and concrete syntax is quite clumsy) compared with what could be achieved with a purpose-built compiler or preprocessor.

ad++ Fragment

```
Procs2 p(2, 2) ;

ON(p) {
  BlockRange x(100, p.dim(0)), y(10, p.dim(1)) ;

  Array2<float> a(x, y) ;

  Index i(x), j(y) ;
  OVERALL(i) {
    OVERALL(j) {
      a(i, j) = i + j ;
    } ALLOVER(j) ;
  } ALLOVER(i) ;

} NO(p) ;
```

HPJava

- Fix the efficiency problems of ad++ by using a preprocessor to optimize the parallel loops.
- Modelled semantically on ad++, but with a more readable syntax.
- Implementation by a simple translator from HPJava to Java (in progress).

Distributed Arrays

- Distributed arrays flagged by double brackets. Orthogonal to ordinary Java arrays.
- Distribution expressed in terms of `Group` (subclass `Procs`) and `Range`.
- Example:

```
Procs2 p = new Procs2(3, 2) ;

Range x = new BlockRange(100, p.dim(0)) ;
Range y = new BlockRange(200, p.dim(1)) ;

float [[,]] a = new float [[x, y]] on p ;
```

The *on* construct and the APG

- In SPMD programming, often necessary to restrict execution of a block of code to processors in a particular group, `p`.
- HPJava provides a special construct for this:

```
on(p) {
  ...
}
```

- Language incorporates a formal idea of an **active process group** (APG). Inside `on(p)`, value of the APG changes to `p`.

Locations and the *at* construct

- Distributed array access such as

```
a [17, 23] = 13 ;
```

  is legal, but **only** if the local process holds the element in question.

- Introduce idea of **location** and **at** construct to alleviate inconvience of this restriction:

```
Location i = x [17], j = y [23] ;
at(i)
  at(j)
    a [i, j] = 13 ;
```

Distributed loops

- Last and most important distributed control construct is called **overall**.
- Implements a distributed parallel loop.
- Parametrized by member of iterator class `Index`. Example:

```
float [[,]] a = new float [[x, y]], b = new float [[x, y]] ;
Index i, j ;
overall(i = x | :)
  overall(j = y | :)
    a [i, j] = 2 * b [i, j] ;
```

Examples

Two examples follow:

- Cholesky decomposition
- Potts update

Choleski decomposition

In pseudocode, sequential algorithm is

$$
\begin{array}{l}
For\ k = 1\ to\ n-1 \\
\quad l_{kk} = a_{kk}^{1/2} \\
\quad For\ s = k+1\ to\ n \\
\quad\quad l_{sk} = a_{sk}/l_{kk} \\
\quad For\ j = k+1\ to\ n \\
\quad\quad For\ i = j\ to\ n \\
\quad\quad\quad a_{ij} = a_{ij} - l_{ik} l_{jk} \\
l_{nn} = a_{nn}^{1/2}
\end{array}
$$

Note HPJava code should be enclosed by a suitable `on(p)`.

```
Range x = new CyclicRange(N, p.dim(0));

float [[*,]] a = new float [[N, x]] ;
float [[*]]  b = new float [[N]] ;  // buffer

Location l ;
Index m ;

for(int k = 0 ; k < N - 1 ; k++) {

  at(l = x [k]) {
    float d =  Math.sqrt(a [k, l]) ;
    a [k, l] = d ;
    for(int s = k + 1 ; s < N ; s++)
      a [s, l] /= d ;
  }

  Adlib.remap(b [[k + 1 : ]], a [[k + 1 : , k]]);

  overall(m = x | k + 1 : )
    for(int i = x.idx(m) ; i < N ; i++)
      a [i, m] -= b [i] * b [x.idx(m)] ;
}

at(l = x [N - 1]) a [N - 1, l] = Math.sqrt(a [N - 1, l]) ;
```

Potts update pseudocode

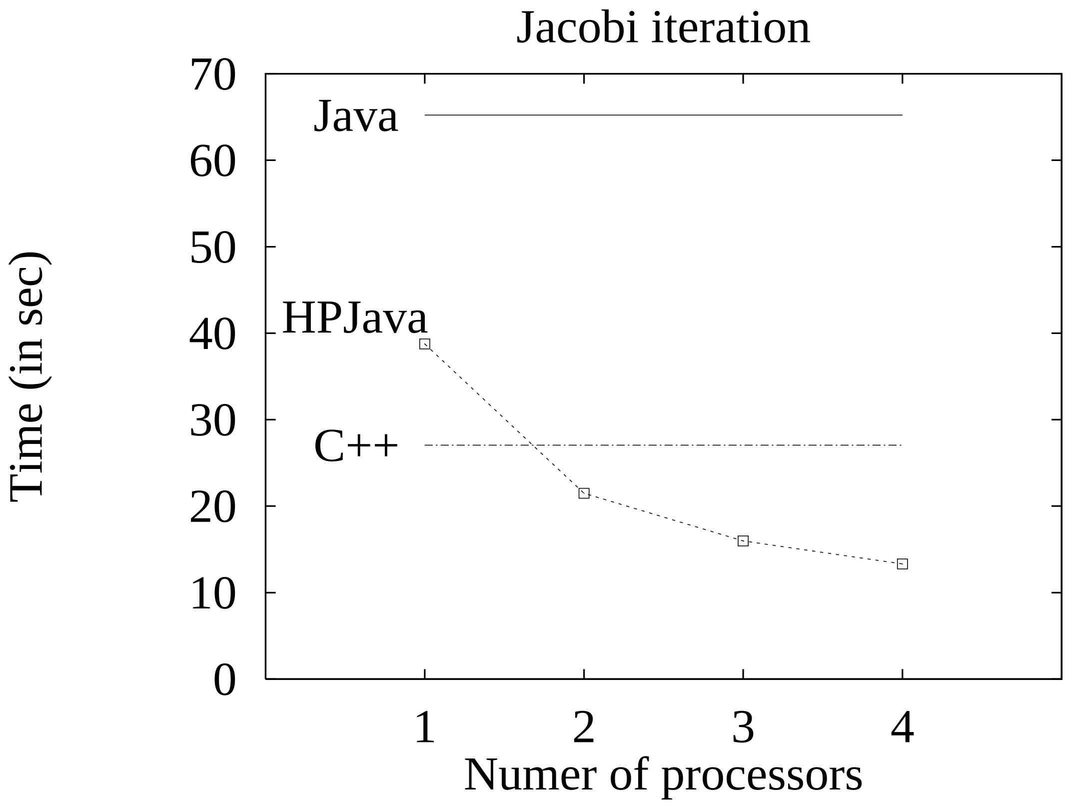
```
do parity = 0, 1

  forall i, j such that i + j mod 2 = parity
    old  = latt [i, j]

    cand = (old + (int) (rand() * q - 1) + 1) % q

    e_old = (old == latt [i + 1, j]) + (old == latt [i - 1, j]) +
            (old == latt [i, j + 1]) + (old == latt [i, j - 1])

    e_new = (cand == latt [i + 1, j]) + (cand == latt [i - 1, j]) +
            (cand == latt [i, j + 1]) + (cand == latt [i, j - 1])

    if(rand() < exp((e_old - e_new) * cup))
      latt [i, j] = cand
  end forall

end do
```

```
Index i, j ;
for (int parity = 0; parity < 2; parity ++) {

  Adlib.writeHalo(latt, w, w, mode) ;

  overall(i = x | :)
    overall(j = y | (parity + x) % 2 : : 2) {
      int old  = latt [i, j] ;

      int cand = (old + (int) (Math.erand48(xsubi) * q - 1) + 1) % q ;

      float e_old =
          (old == latt [i + 1, j]) + (old == latt [i - 1, j]) +
          (old == latt [i, j + 1]) + (old == latt [i, j - 1]) ;

      float e_new =
          (cand == latt [i + 1, j]) + (cand == latt [i - 1, j]) +
          (cand == latt [i, j + 1]) + (cand == latt [i, j - 1]) ;

      if (Math.erand48(xsubi) < Math.exp((e_old - e_new) * cup))
        latt [i, j] = cand ;
    }
```

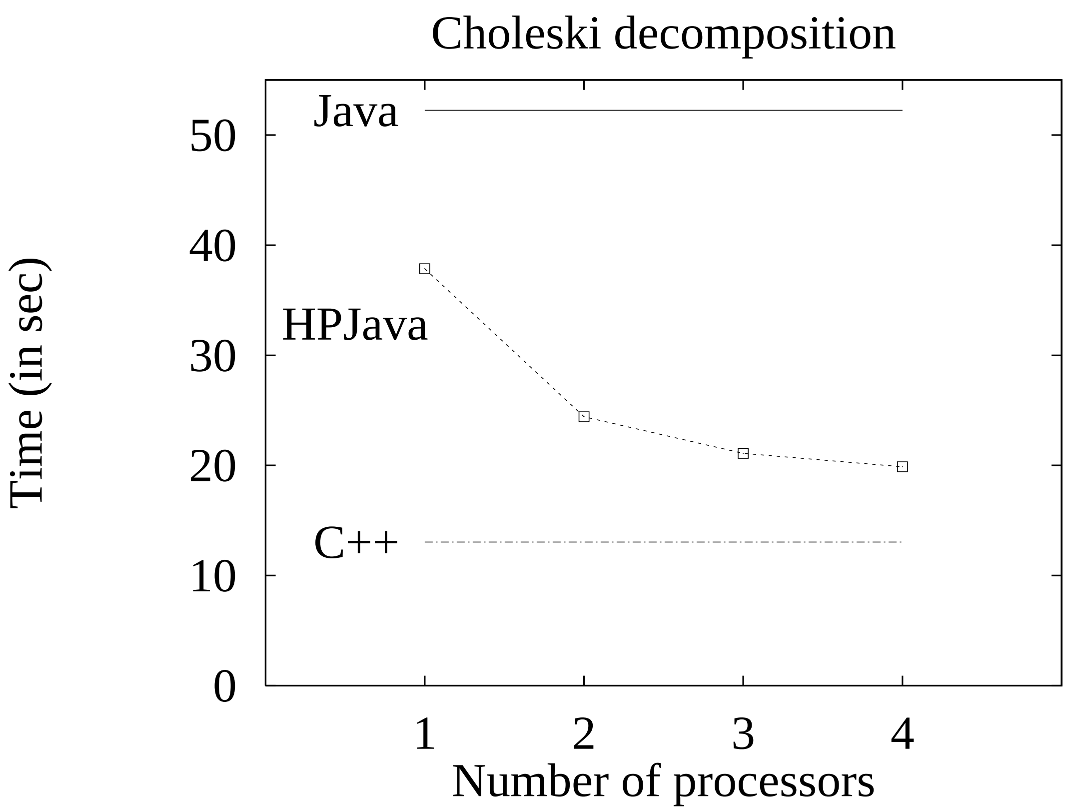
Preliminary Performance Data

- Examples are Jacobi update for 2d Laplace and Cholesky decomposition.
- HPJava code was hand-translated to Java, applying only optimizations that will be incorporated in the initial translator.
- Parallel Java on 4 Ultrasparc processors, using JDK 1.2 JIT compiler Generated code uses a JNI interface to Adlib, with MPICH for underlying communications.
- Comparisions are with simple Java and C++ versions.
- Overhead in parallel case attributable to MPI—SunHPC may reduce this by an order of magnitude.

Jacobi Update



Choleski


Choleski decomposition
Java
HPJava
C++
Time (in sec)
0
10
20
30
40
50
1
2
3
4
Number of processors

Conclusions

- HPspmd language extensions provide much of the expressive power of HPF, but in a strictly SPMD enviroment.
- Allows programs to combine data parallel code and SPMD library calls directly.
- Initial full binding will be to Java—originally regarded as a prototype, but rapidly advancing Java performance may make this a competitive production environment.
- Similar Fortran extensions have been suggested (FAST proposal). May be implemented by Peking collaborators. . .