An *HPspmd* Programming Model

Bryan Carpenter, Geoffrey Fox and Guansong Zhang

Northeast Parallel Architectures Centre,
Syracuse University,
111 College Place,
Syracuse, New York 13244-410
{dbc,gcf,zgs}@npac.syr.edu

November 12, 1999

Abstract

Building on research carried out in the Parallel Compiler Runtime Consortium (PCRC) project, this article discusses a language model that combines characteristic data-parallel features from the HPF standard with an explicitly SPMD programming style. This model, which we call the *HPspmd* model, is designed to facilitate direct calls to established libraries for parallel programming with distributed data. We describe a Java-based HPspmd language called HPJava.

1 Introduction

Data parallel programming languages have always held a special position in the high-performance computing world. The basic implementation issues related to this paradigm are well understood. However, the choice of high-level programming environment, particularly for modern MIMD architectures, remains uncertain. Six years ago the High Performance Fortran Forum published the first standardized definition of a language for data parallel programming [13, 15]. In the intervening period considerable progress has been made in HPF compiler technology, and the HPF language definition has been extended and revised in response to demands of compiler-writers and end-users [11]. Yet it seems to be the case that most programmers developing parallel applications—or environments for parallel application development—do *not* code in HPF. The slow uptake of HPF can be attributed in part to immaturity in the current generation of compilers. But it seems likely that many programmers are actually more comfortable with the Single Program Multiple Data (SPMD) programming style, perhaps because the effect of executing an SPMD program is more controllable, and the process of tuning for efficiency is more intuitive.

Of course SPMD programming has been very successful. There are countless applications written in the most basic SPMD style, using direct message-passing through MPI [16] or similar low-level packages. Many higher-level parallel programming environments and libraries assume the SPMD style as their basic model. Examples include ScaLAPACK [4], PetSc [2], DAGH [19], Kelp [10], the Global Array Toolkit [17] and NWChem [3]. While there remains a prejudice that HPF is best suited for problems with very regular data structures and regular data access patterns, SPMD frameworks like DAGH and Kelp have been designed to deal directly with irregularly distributed data, and other libraries like CHAOS/PARTI [8] and Global Arrays support unstructured access to distributed arrays.

These successes aside, the library-based SPMD approach to data-parallel programming certainly lacks the uniformity and elegance of HPF. All the environments referred to above have some idea of a distributed array, but they all describe those arrays differently. Compared with HPF, creating distributed arrays and accessing their local and remote elements is clumsy and error-prone. Because the arrays are managed entirely in libraries, the compiler offers little support and no safety net of compile-time checking.

This article discusses a class of programming languages that borrow certain ideas, various run-time technologies, and some compilation techniques from HPF, but relinquish some of its basic tenets. In particular they forgo the principles that the programmer should write in a language with (logically) a single global thread of control, and that the compiler should determine automatically which processor executes individual computations in a program, then automatically insert communications if an individual computation involves accesses is to non-local array elements.

If these assumptions are removed from the HPF model, does anything useful remain? We argue "yes". What will be retained is an explicitly MIMD (SPMD) programming model complemented by syntax for representing distributed arrays, and syntax for expressing that certain computations are localized to certain processors, including syntax for a distributed form of the parallel loop. The claim is that these features are adequate to make calls to various data-parallel libraries, including application-oriented libraries and high-level libraries for communication, about as convenient as, say, making a call to an array transformational intrinsic function in Fortran 90. Besides their advantages as a framework for library usage, the resulting programming languages can conveniently express various practical data-parallel algorithms. The resulting framework may also have better prospects for dealing effectively with *irregular* problems than is the case for HPF.

2 HPspmd language extensions

We aim to provide a flexible hybrid of the data parallel and low-level SPMD paradigms. To this end HPF-like distributed arrays appear as language primitives. But a design decision is made that all access to *non-local* array elements should go through library functions—either calls to a collective communication library, or simply `get` and `put` functions for access to remote blocks of a distributed array. Clearly this decision puts an extra onus on the programmer; but making communication explicit encourages the programmer to write algorithms that exploit locality, and simplifies the task of the compiler writer.

For the newcomer to HPF, one of its advantages lies in the fact that the effect of a particular operation is logically identical to its effect in the corresponding sequential program. Assuming programmers understand conventional Fortran, it is very easy for them to understand the behaviour of a program at the level of what values are held in program variables, and the final results of procedures and programs. Unfortunately, the ease of understanding this "value semantics" of a program is counterbalanced by the difficulty in knowing exactly how the compiler translated the program. Understanding the *performance* of an HPF program may require that the programmer have rather detailed knowledge of how arrays are distributed over processor memories, and what strategy the compiler adopts for distributing computations.

The language model we discuss has a special relationship to the HPF model, but the HPF-style semantic equivalence between the data-parallel program and a sequential program is abandoned in favour of a simple equivalence between the data-parallel program and an MIMD (SPMD) program. Because understanding an SPMD program is presumably more difficult than understanding a sequential program, our language may be slightly harder to learn and use than HPF. But understanding performance of programs should be much easier.

The distributed arrays of an HPspmd language should be kept strictly separate from ordinary arrays. They are a different kind of object, not type-compatible with ordinary arrays. A property of the languages we describe is that if a section of program text *looks like* program text from the unenhanced base language (Fortran 90 or Java, for example), it is translated exactly as for the base language—as local sequential code. Only statements involving the extended syntax are treated specially. This makes preprocessor-based implementation of the new languages straightforward, allows sequential library code to be called directly, and gives programmers good control over the generated code—they can be confident no unexpected overhead have been introduced into code that looked like ordinary Fortran, for example.

We adopt a distributed array model semantically equivalent to to the HPF data model in terms of how elements are stored, the options for distribution and alignment, and facilities for describing *regular sections* of arrays. Distributed arrays may be subscripted with global subscripts, as in HPF. But an array element reference must not imply access to a value held on a different processor.

We sometimes refer to this restriction as the *SPMD constraint*. To simplify the task of the programmer, who must be sure accessed elements are held locally, the languages can add *distributed control* constructs. These play a role something like the `ON HOME` directives of HPF 2.0 and earlier data parallel languages [14]. One special control construct—a distributed parallel loop—facilitates traversal of locally held elements from a group of aligned arrays.

A Java instantiation (HPJava) of this HPspmd language model has been described in [6]. A brief review is given in section 4. In [5] we have outlined possible syntax extensions to Fortran to provide similar semantics to HPJava.

3 Integration of high-level libraries

Libraries are at the heart of our HPspmd model. From one point of view, the language extensions are simply a framework for invoking libraries that operate on distributed arrays. Hence an essential component of the ongoing work is definition of a series of bindings from HPspmd languages to established SPMD libraries and environments. Because the language model is explicitly SPMD, such bindings are a more straightforward proposition than in HPF, where one typically has to pass some extrinsic interface barrier before invoking SPMD-style functions.

We can group the existing SPMD libraries for data parallel programming into three categories. In the first category we have libraries like ScaLAPACK [4] and PetSc [2] where the primary focus is similar to conventional numerical libraries—providing implementations of standard matrix algorithms (say) but operating on elements in regularly distributed arrays. We assume that designing HPspmd interfaces to this kind of package will be relatively straightforward. ScaLAPACK for example, provides linear algebra routines for distributed-memory computers. These routines operate on distributed arrays—specifically, distributed matrices. The distribution formats supported are restricted to two-dimensional block-cyclic distribution for dense matrices and one-dimensional block distribution for narrow-band matrices. Since both these distribution formats are supported by HPspmd, using ScaLAPACK routines from the HPspmd framework should present no fundamental difficulties.

In a second category we place libraries conceived primarily as underlying support for general parallel programs with regular distributed arrays. They emphasize high-level communication primitives for particular styles of programming, rather than specific numerical algorithms. These libraries include compiler runtime libraries like Multiblock Parti [1] and Adlib [21], and application-level libraries like the Global Array toolkit [17]. Adlib is a runtime library that was designed to support HPF translation. It provides communication primitives similar to Multiblock PARTI, plus the Fortran 90 transformational intrinsics for arithmetic on distributed arrays. The Global Array (GA) toolkit, developed at Pacific Northwest National Lab, provides an efficient and portable "sharedmemory" programming interface for distributed-memory computers. Each process in a MIMD parallel program can asynchronously access logical blocks of distributed arrays, without need for explicit cooperation by other processes ("one-sided communication"). Besides providing a more tractable interface for creation of multidimensional distributed arrays, our syntax extensions should provide a more convenient interface to primitives like `ga_get`, which copies a patch of a global array to a local array.

Regular problems (such as the linear algebra examples in section 4) are an important subset of parallel applications, but of course they are far from exclusive. Many important problems involve data structures too irregular to represent purely through HPF-style distributed arrays. Our third category of libraries therefore includes libraries designed to support irregular problems. These include CHAOS [8] and DAGH [19]. We anticipate that irregular problems will still benefit from regular data-parallel language extensions—at some level they usually resort to representations involving regular arrays. But lower level SPMD programming, facilitated by specialized class libraries, is likely to take a more important role. For an HPspmd binding of the CHAOS/PARTI library, for example, the simplest assumption is that the preprocessing phases yield new arrays. Indirection arrays may well be left as HPspmd distributed arrays; data arrays may be reduced to ordinary Java arrays holding local elements. Parallel loops of an executor phase can then be expressed using *overall* constructs. More advanced schemes may incorporate irregular maps into generalized array descriptors [11, 9, 7] and require extensions to the baseline HPspmd language model.

4 HPJava—an HPspmd language

HPJava [6] is an instance of our HPsmpd language model. HPJava extends its base language, Java, by adding some predefined classes and some additional syntax for dealing with distributed arrays.

As explained in the previous section, the underlying distributed array model is equivalent to the HPF array model. Array mapping is described in terms of a slightly different set of basic concepts. *Process group* objects generalize the processor arrangements of HPF. *Distributed range* objects are used instead HPF templates. A distributed range is comparable with a single dimension of an HPF template. These substitutions are a change of parametrization only. Groups and ranges fit better with our distributed control constructs.

Figure 1 is a simple example of an HPJava program. It illustrates creation of distributed arrays, and access to their elements. The class `Procs2` is a standard library class derived from the special base class `Group`. It represents a two-dimensional grid of processes. Similarly the distributed range class `BlockRange` is a library class derived from the special class `Range`; it denotes a range of subscripts distributed with BLOCK distribution format over a specific process

```
Procs2 p = new Procs2(P, P) ;
on(p) {
  Range x = new BlockRange(M, p.dim(0)) ;
  Range y = new BlockRange(N, p.dim(1)) ;

  float [[,]] a = new float [[x, y]], b = new float [[x, y]],
              c = new float [[x, y]] ;

  ... initialize values in 'a', 'b'

  overall(i = x for :)
    overall(j = y for :)
      c [i, j] = a [i, j] + b [i, j] ;
}
```

Figure 1: A parallel matrix addition.

dimension. Process dimensions associated with a grid are returned by the `dim()` inquiry. The `on(p)` construct is a new control construct specifying that the enclosed actions are performed only by processes in group `p`.

The variables `a`, `b` and `c` are all distributed array objects. The type signature of an r-dimensional distributed array involves double brackets surrounding r comma-separated slots. The constructors specify that these all have ranges `x` and `y`—they are all `M` by `N` arrays, block-distributed over `p`.

A second new control construct, *overall*, implements a distributed parallel loop. The constructs here iterate over all locations (selected by the degenerate interval " `:` ") of ranges `x` and `y`. The symbols `i` and `j` scoped by these constructs are *bound locations*. In HPF, a distributed array element is referenced using integer subscripts, like an ordinary array. In HPJava, with a couple of exceptions noted below, the subscripts in element references must be bound locations, and these must be locations in the range associated with the array dimension. This rather drastic restriction is a principal means of ensuring that referenced array elements are held locally.

The general policy is relaxed slightly to simplify coding of stencil updates. A subscript can be a *shifted location*. Usually this is only legal if the subscripted array is declared with suitable *ghost regions* [12]. Figure 2 illustrates the use of the library class `ExtBlockRange` to create arrays with ghost extensions (in this case, extensions of width 1 on either side of the locally held "physical" segment). The communication library routine `Adlib.writeHalo` updates the ghost region. This example also illustrates application of a postfix backquote operator to a bound location. The expression `` i` `` (read "i-primed") yields the integer global loop index.

Distributed arrays can be defined with some sequential dimensions. The

```
Procs2 p = new Procs2(P, P) ;
on(p) {
  Range x = new ExtBlockRange(N, p.dim(0), 1, 1) ;
  Range y = new ExtBlockRange(N, p.dim(1), 1, 1) ;

  float [[,]] u = new float [[x, y]] ;

  ... some code to initialise 'u'

  for(int iter = 0 ; iter < NITER ; iter++) {

    Adlib.writeHalo(u) ;

    overall(i = x for 1 : N - 2)
      overall(j = y for 1 + (i` + iter) % 2 : N - 2 : 2)
        u [i, j] = 0.25 * (u [i - 1, j] + u [i + 1, j] +
                           u [i, j - 1] + u [i, j + 1]) ;
  }
}
```

Figure 2: Red-black iteration.

sequential attribute of an array dimension is flagged by an asterisk in the type signature. As illustrated in Figure 3, element reference subscripts in sequential dimensions can be ordinary integer expressions.

The short examples here have already covered much of the special syntax of HPJava. Other significant extensions allow Fortran-90-like *sections* of distributed arrays. This, in turn, forces us to define certain *subranges* and *subgroups*. Arrays constructed directly using subgroups and subranges can reproduce all the alignment options of HPF. In any case, the language itself is relatively simple. Complexities associated with varied and irregular patterns of communication are dealt with in libraries. These can implement many richer operations than the `writeHalo` and `cshift` functions of the examples.

5 Conclusions

In this article we discussed motivations for introducing an HPspmd programming model: a SPMD framework for using libraries based on distributed arrays. It adopts the model of distributed arrays standardized by the HPF Forum, but relinquishes the high-level single-threaded model of the HPF language. This makes compilers or translators for the HPspmd-extended languages a relatively straightforward proposition. As a concrete example, we described the specific syntax of HPJava.

```
Procs1 p = new Procs1(P) ;
on(p) {
  Range x = new BlockRange(N, p.dim(0)) ;

  float [[,*]] a = new float [[x, N]], c = new float [[x, N]] ;
  float [[*,]] b = new float [[N, x]], tmp = new float [[N, x]] ;

  ... initialize 'a', 'b'

  for(int s = 0 ; s < N ; s++) {

    overall(i = x for :) {

      float sum = 0 ;
      for(int j = 0 ; j < N ; j++)
        sum += a [i, j] * b [j, i] ;

      c [i, (i` + s) % N] = sum ;
    }

    // cyclically shift 'b' (by amount 1 in x dim)...

    Adlib.cshift(tmp, b, 1, 1) ;
    HPspmd.copy(b, tmp) ;
  }
}
```

Figure 3: A pipelined matrix multiplication program.

Two recent languages that have some similarities to our HPspmd languages are F-- and ZPL. F-- [18] is an extended Fortran dialect for SPMD programming. The approach is different to the one proposed here. There is no analogue of global subscripts, or HPF-like distribution formats. In F-- the logical model of communication is built into the language—remote memory access with intrinsics for synchronization—where our basic philosophy is to provide communication through separate libraries. ZPL [20] is a array parallel programming language for scientific computations. It has a construct for performing computations over a *region*, or set of indices, quite similar to our *overall* construct. Communication is more explicit than HPF, but not as explicit as in the language discussed in this article.

At the time of writing the HPJava translator is partially operational. Ongoing work will complete the functionality, and add some optimization for the generated code. The language definition calls for full compile-time or runtime checking of the constraints on locality of reference. The translator will be enhanced to add these. Early benchmarks results will be included in the final version of this paper.

6 Acknowledgements

This work was supported in part by the National Science Foundation Division of Advanced Computational Infrastructure and Research, contract number 9872125.

References

[1] A. Agrawal, A. Sussman, and J. Saltz. An integrated runtime and compile-time approach for parallelizing structured and block structured applications. *IEEE Transactions on Parallel and Distributed Systems*, 6, 1995.

[2] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in object-oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhauser Press, 1997.

[3] D. E. Bernholdt, E. Aprà, H. A. Früchtl, M. F. Guest, R. J. Harrison, R. A. Kendall, R. A. Kutteh, X. Long, J. B. Nicholas, J. A. Nichols, H. L. Taylor, A. T. Wong, G. I. Fann, R. J. Littlefield, and J. Nieplocha. Parallel computational chemistry made easier: The development of NWChem. *Int. J. Quantum Chemistry: Quantum Chem. Symposium*, 29:475–483, 1995.

[4] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK User's Guide*. SIAM, 1997.

[5] Bryan Carpenter, Geoffrey Fox, Donald Leskiw, Xinying Li, Yuhong Wen, and Guansong Zhang. Language bindings for a data-parallel runtime. In *Third International Workshop on High-Level Parallel Programming Models and Supportive Environments*. IEEE Computer Society Press, 1998. http://www.npac.syr.edu/projects/pcrc.

[6] Bryan Carpenter, Guansong Zhang, Geoffrey Fox, Xiaoming Li, Xinying Li, and Yuhong Wen. Towards a Java environment for SPMD programming. In David Pritchard and Jeff Reeve, editors, *4th International Europar Conference*, volume 1470 of *Lecture Notes in Computer Science*. Springer, 1998. http://www.npac.syr.edu/projects/pcrc/HPJava.

[7] B. Chapman, P. Mehrotra, and H. Zima. Programming in Vienna Fortran. *Scientific Programming*, 1(1):1–50, 1992.

[8] R. Das, M. Uysal, J.H. Salz, and Y.-S. Hwang. Communication optimizations for irregular scientific computations on distributed memory architectures. *Journal of Parallel and Distributed Computing*, 22(3):462–479, September 1994.

[9] G. Fox et al. Fortran D language specification. Technical Report CRPC-TR90079, Center for Research on Parallel Computation, Rice University, 1990.

[10] Stephen J. Fink and Scott B. Baden. Run-time data distribution for block-structured applications on distributed memory computers. In *Proceedings of the 7th SIAM Conference on Parallel Processing for Scientific Computing*, February 1995.

[11] High Performance Fortran Forum. High Performance Fortran language specification, version 2.0, January 1997.
http://www.crpc.rice.edu/HPFF/hpf2.

[12] Michael Gerndt. Updating distributed variables in local computations. *Concurrency: Practice and Experience*, 2(3):171–193, 1990.

[13] High Performance Fortran Forum. High Performance Fortran language specification. *Scientific Programming*, special issue, 2, 1993.

[14] C. Koelbel and P. Mehrotra. Compiling global name-space parallel loops for distributed execution. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):440–451, 1991.

[15] C.H. Koelbel, D.B. Loveman, R.S. Schreiber, G.L. Steel, Jr., and M.E. Zosel. *The High Performance Fortran Handbook.* MIT Press, 1994.

[16] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard.* University of Tenessee, Knoxville, TN, June 1995.
http://www.mcs.anl.gov/mpi.

[17] J. Nieplocha, R.J. Harrison, and R.J. Littlefield. The Global Array: Non-uniform-memory-access programming model for high-performance computers. *The Journal of Supercomputing*, 10:197–220, 1996.

[18] R.W. Numrich and J.L. Steidel. F- -: A simple parallel extension to Fortran 90. *SIAM News*, page 30, 1997.

[19] Manish Parashar and J.C. Browne. Systems engineering for high performance computing software: The HDDA/DAGH infrastructure for implementation of parallel structured adaptive mesh. In *Structured Adaptive Mesh Refinement Grid Methods*, IMA Volumes in Mathematics and its Applications. Springer-Verlag.

[20] Lawrence Snyder. A ZPL programming guide. Technical report, University of Washington, May 1997.
http://www.cs.washington.edu/research/projects/zpl/.

[21] Guansong Zhang, Bryan Carpenter, Geoffrey Fox, Xiaoming Li, Xinying Li, and Yuhong Wen. PCRC-based HPF compilation. In Zhiyuan Li et al, editor, *10th International Workshop on Languages and Compilers for Parallel Computing*, volume 1366 of *Lecture Notes in Computer Science*. Springer, 1997. http://www.npac.syr.edu/projects/pcrc.