Final Report
Center for Programming Models for Scalable Parallel Computing: Co-Array Fortran
Grant Number DE-FC02-01ER25505 M004 *

Robert W. Numrich

April 22, 2008

Summary

The major accomplishment of this project is the production of CafLib, an "object-oriented" parallel numerical library written in Co-Array Fortran. CafLib contains distributed objects such as block vectors and block matrices along with procedures, attached to each object, that perform basic linear algebra operations such as matrix multiplication, matrix transpose and LU decomposition. It also contains constructors and destructors for each object that hide the details of data decomposition from the programmer, and it contains collective operations that allow the programmer to calculate global reductions, such as global sums, global minima and global maxima, as well as vector and matrix norms of several kinds. CafLib is designed to be extensible in such a way that programmers can define distributed grid and field objects, based on vector and matrix objects from the library, for finite difference algorithms to solve partial differential equations.

A very important extra benefit that resulted from the project is the inclusion of the co-array programming model in the next Fortran standard called Fortran 2008. It is the first parallel programming model ever included as a standard part of the language. Co-arrays will be a supported feature in all Fortran compilers, and the portability provided by standardization will encourage a large number of programmers to adopt it for new parallel application development. The combination of object-oriented programming in Fortran 2003 with co-arrays in Fortran 2008 provides a very powerful programming model for high-performance scientific computing.

Additional benefits from the project, beyond the original goal, include a program to provide access to the co-array model through access to the Cray compiler as a resource for teaching and research. Several academics, for the first time, included the co-array model as a topic in their courses on parallel computing. A separate collaborative project with LANL and PNNL showed how to extend the co-array model to other languages in a small experimental version of Co-array Python. Another collaborative project defined a Fortran 95 interface to ARMCI to encourage Fortran programmers to use the one-sided communication model in anticipation of their conversion to the co-array model

*rwn/papers/progressReports/pmodelsFinal.tex later. A collaborative project with the Earth Sciences community at NASA Goddard and GFDL experimented with the co-array model within computational kernels related to their climate models, first using CafLib and then extending the co-array model to use design patterns. Future work will build on the design-pattern idea with a redesign of CafLib as a true object-oriented library using Fortran 2003 and as a parallel numerical library using Fortran 2008.

1 CafLib: A parallel numerical library for Co-Array Fortran

The development of CafLib, a parallel numerical library to support Co-Array Fortran [15, 16], was the project's main goal. A copy of the CafLib Users' Manual [17] is included with this report along with a compressed archive file containing the source code. The library consists of about 75,000 lines of code, fully tested and documented.

CafLib is written in Fortran 95 using object-oriented design principles [1, 7, 8, 11, 13]. Since Fortran 95 s not a true object-oriented language, objects are emulated as Fortran derived types for three different data types and two working precisions for each type: 4-byte and 8-byte real, 8-byte and 16-byte complex, and 4-byte and 8-byte integer. These objects represent vectors and matrices for each of the six options along with distributed vectors and distributed matrices of each kind through an emulation of inheritance. Distribution across processors is defined by Maps attached to each distributed object, and communication between objects takes place with co-array syntax using information in the maps. Each object has a constructor that knows how to build distributed objects based on input parameters supplied by the programmer. Destructors for each object clean up allocated memory to avoid memory leaks.

1.1 Object Maps

The design of every parallel application starts with problem decomposition. Problem decomposition in CafLib is represented by maps. A map contains all the information required to describe a set of objects, defined globally, and their distribution to sets of local objects distributed across processors locally. In Caflib, these maps are called ObjectMaps as shown in Figure 1.

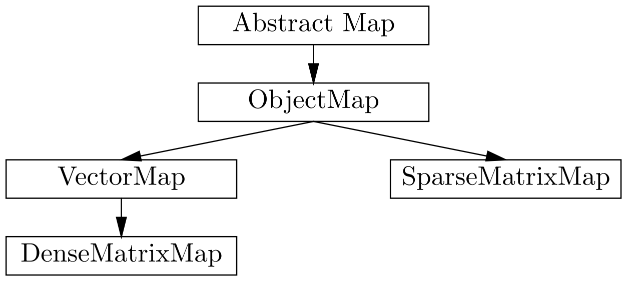

Figure 1: Problem decomposition represented as extensions of an Abstract Map.

These maps are loosely based on the Composite design pattern [5, 10]. As implemented in CafLib, an Object Map is a composite function,

$$L^p = \Lambda^p(\Pi(G)) \ . \tag{1}$$

A global set of n objects $G = \{G_1, \ldots, G_n\}$ is first permuted,

$$\pi = \Pi(G) \ , \tag{2}$$

such that

$$\pi_j = \Pi^i_j(G_i) \qquad i, j = 1, \ldots, n \ . \tag{3}$$

Subsets of these permuted objects are then projected to n_p local objects $L^p = \{L^p_1, \ldots, L^p_{n_p}\}$ on specific processors,

$$L^p = \Lambda^p(\pi) \ , \tag{4}$$

such that

$$L^p_k = (\Lambda^p)^j_k(\pi_j) \qquad k = 1, \ldots, n_p \ , \qquad p = 1, \ldots, \text{num\_images}() \ . \tag{5}$$

Figure 2 shows an example of an Object Map. A commonly used permutation in scientific applications, especially in linear algebra, is a cyclic permutation. CafLib uses this default permutation unless the programmer specifies a different permutation. These programmer-provided permutations are completely general and may be chosen to represent particular aspects of a particular application. In the same way, the projection onto specific processors is also general with the default chosen as an equal number of objects on each processor. But to balance work load, the programmer may specify a projection specific to an application, including the zero projection for some processors. These permutations and projections can change dynamically by switching from one map to another during execution and by moving data accordingly using procedures provided by CafLib.

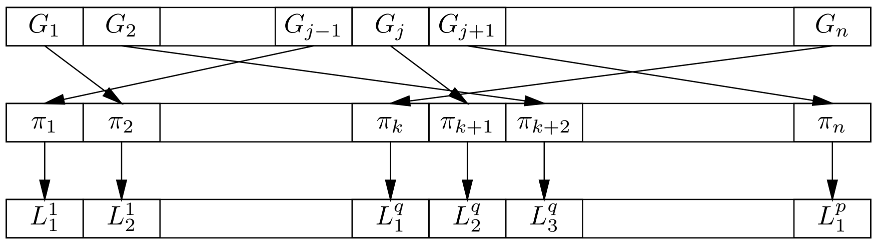

Figure 2: A composite Object Map.

Had Fortran 2003, a true object-oriented language, been available for this project, these maps would have been represented as concrete instantiations of an Abstract Map as shown in Figure 1. Fortran 95 not being a true object-oriented language, these maps were implemented by emulating inheritance as well as possible starting with the Object Map. A Vector Map extends the Object Map to a map for specific objects that correspond to blocks of a vector. A Dense Matrix Map extends the Vector Map by assigning two Vector Maps, one for rows of the matrix and one for columns. A Sparse Matrix Map extends the Object Map by defining the objects to be irregular pieces of the matrix specific to a chosen compressed storage scheme.

The composite function (1) represented by an Object Map has an inverse so that each processor knows how its set of local objects is related to the set of global objects. Figure 3, for example, shows how processor q locates its neighbor to the left for its second local object L^q_2 as the first local object L^1_1 on processor 1 and its neighbor to the right as local object L^p_1 on processor p. These relationships are encapsulated in procedures associated with each object that perform, for example, halo exchanges between distributed matrix objects.

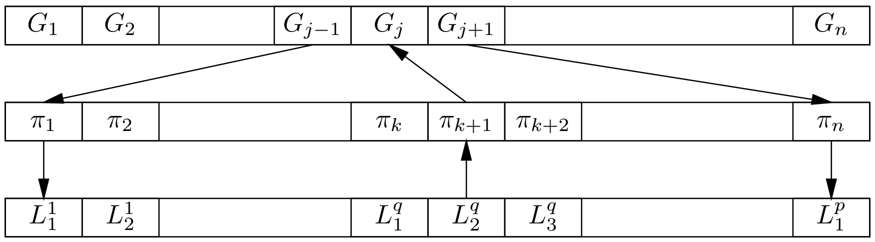

Figure 3: Inverse map to nearest neighbors.

1.2 Library Design

The Object Maps described in Section 1.1 are key to the design of CafLib. As Figure 4 shows, they occupy a central position in the library's design in the form of concrete Vector Maps used both for distributed Block Vector objects and for rows and columns of distributed Block Matrix objects.

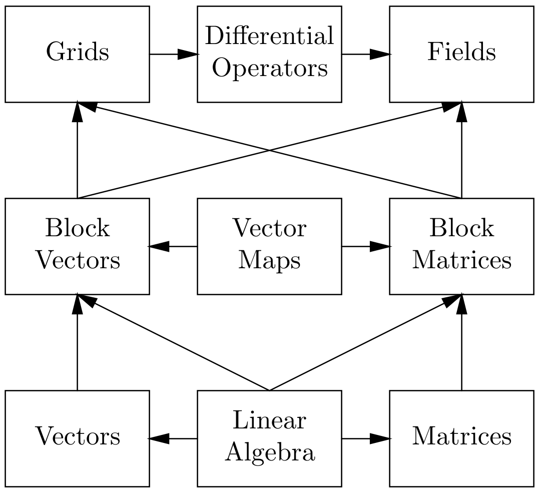

Figure 4: Design of CafLib.

A Block Matrix object, for example, has the structure,

```
type BlockXXMatrix
private
  type(VectorMap)             :: rowMap
  type(VectorMap)             :: colMap
  type(XXMatrix),allocatable :: block(:,:)
  !-other components-!
end type BlockXXMatrix
```

where the symbol XX can be any of six different precisions, XX = {R4,R8,C4,C8,I4,I8}. Each `block(:,:)` of the structure is itself a structure of `type(XXMatrix)` that holds the data in a normal two-dimensional array,

```
type XXMatrix
  real(kind=XX),allocatable :: matrix(:,:)
  !-other components-!
end type XXMatrix
```

These distributed block matrices are created by calling a generic constructor,

```
type(BlockXXMatrix) :: a
call newBlockMatrix(a,m,n,k,l,p,q,w1,w2)
```

where m and n are the global size of the matrix, k and l are block sizes, p and q are co-dimensions describing the processor decomposition, and $w1$ and $w2$ are optional halo widths. Based on the arguments presented to the constructor, it builds vector maps for rows and columns of the matrix and allocates the appropriate number of blocks for each processor and the appropriate amount of memory for each block. An alternative form of the constructor,

```
call newBlockMatirx(a,rowMap,colMap)
```

allows the programmer to build vector maps outside of the constructor and pass them as arguments. There is also a generic destructor for each distributed object,

```
call deleteBlockMatirx(a)
```

that cleans up the allocated memory and prevents memory leaks.

The library contains basic linear algebra procedures for the distributed block vector and block matrix objects. It contains vector and matrix norms, vector and matrix reductions, matrix-vector multiplication, matrix transpose, matrix-matrix multiplication, LU decomposition, and linear solution procedures, all coded as parallel procedures using co-array syntax for data communication. At the heart of each algorithm are calls to LAPACK procedures [3] to obtain high performance on the local blocks of data on each processor. In addition, the programmer can call LAPACK procedures directly for the non-distributed data structures at the bottom of Figure 4 by passing the data component of the object directly to the procedure.

2 Scaling Results

The high level of abstraction in CafLib does not result in a degradation of performance. A measure of performance is the comparison of the scalability of the LU decomposition procedure from CafLib with the same procedure from ScaLAPACK [9, 16, 26]. Figure 5 shows such a comparison where both procedures use the same block-cyclic distribution and the same internal block size.

The left side of Figure 5 shows execution time as a function of the number of processors. The lower the curve, the better the results. As can be seen, the CafLib time is lower than the ScaLAPACK time even for a single processor. This experiment is a strong-scaling experiment where the problem size remains fixed at $n = 1000$ for all processor counts. Eventually the amount of computational work done by each processor goes to zero, and all that is left is the communication overhead. As the figure shows, for large numbers of processors, the communication overhead for the CafLib procedure is about half the overhead for the ScaLAPACK procedure. This difference is caused by the high startup cost inherent to the underlying MPI implementation of the ScaLAPACK procedure. The co-array implementation in CafLib has very low startup cost because the compiler is able to generate instructions that take advantage of the global address space of the underlying Cray-T3E hardware.

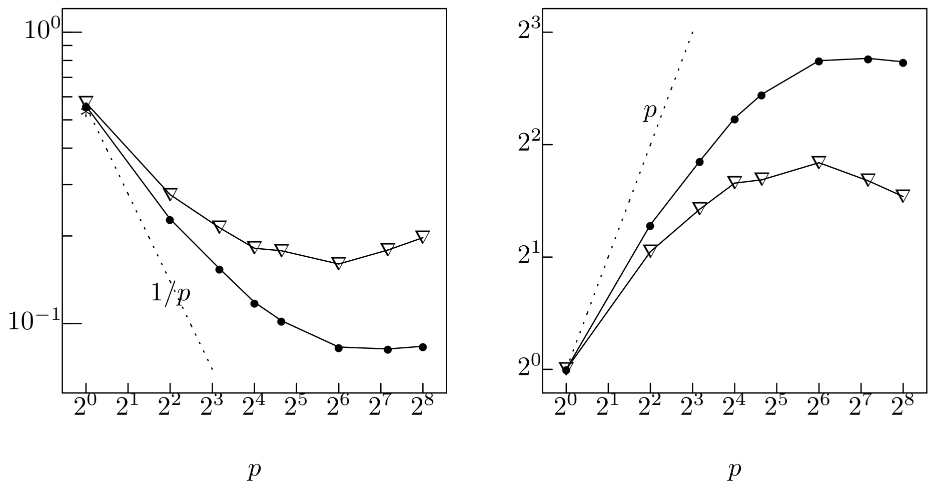

Figure 5: Scaling results for LU Decomposition. The left side shows time plotted as a function of the number of processors, $p = q \times r$. Time is expressed in dimensionless giga-clock-ticks, $\nu t \times 10^{-9}$, as measured on a CRAY-T3E with frequency $\nu = 300$ MHz. The global matrix size is 1000×1000 with local blocks of size 48×48 on each image. The curve marked with bullets ($\bullet$) is CafLib code [16]; the one marked with triangles ($\triangledown$) is ScaLAPACK code [26]. The right side shows speedup as a function of the number of processors. The dotted line on each side represents perfect scaling.

Although weak-scaling experiments, where the problem size grows with the number of processors, show better scaling behavior than strong-scaling experiments, it is still important to examine how well programming models behave for fixed-size problems. Not every problem can be artificially increased in size to make it scale to large numbers of processors. When latency effects become important for a fixed-size problem, it is important that performance declines gracefully to the asymptotic limit represented by the the residual communication overhead. As Figure 5 shows, the MPI implementation does not decline gracefully. The communication overhead starts to grow, perhaps linearly with processor count, at around 64 processors. The CafLib implementation, on the other hand, remains relatively constant in the asymptotic limit.

3 Grids and Fields

The purpose of the CafLib design was not to reproduce the ScaLAPACK Library. The real purpose was to provide an underlying foundation of vector and matrix structures that can be used as a basis for defining physical fields on physical grids to support software for solving partial differential equations using finite difference schemes. As an example, we implemented A finite difference scheme applied to the one-dimensional shallow water equations [15].

The partial differential equations, defined by Cahn [2] and by Arakawa and Lamb [4], for the surface height $h(x,t)$ and the two velocity components $u(x,t)$ and $v(x,t)$, as functions of the space variable x and the time variable t, are the equations [4, eqs. 23-25, p. 183]

$$\begin{aligned} \frac{\partial u}{\partial t} - Fv + G\frac{\partial h}{\partial x} &= 0 \\ \frac{\partial v}{\partial t} + Fu &= 0 \\ \frac{\partial h}{\partial t} + H\frac{\partial u}{\partial x} &= 0\ . \end{aligned}$$

In these equations, F is the Coriolis frequency, G is the acceleration of gravity, and H is the mean height of the surface, which is assumed to be small relative to the width of the space interval.

CafLib supports distributed Field Objects, at the top of Figure 4, based on block-vector and block-matrix objects at lower levels. The fields `u,v,h` are represented as block vectors declared as co-arrays:

```
    type(BlockR8Vector),dimension[*] :: h,u,v
    call newBlockVector(h,n,k,p,w)
    call newBlockVector(u,getVectorMap(h))
    call newBlockVector(v,getVectorMap(h))
    call setBlockVector(h,h0)
```

The first call to the constructor creates the field `h` with `n` grid points cut into blocks of size `k` distributed over `p` images. The halo width `w` equals one, wide enough for a two-point stencil for the first-order difference operator. Calls to an alternative form of the constructor create the fields `u` and `v` with a predefined vector map, returned by the procedure `getVectorMap(h)` as a second argument. Use of the second form of the constructor avoids the overhead of building the map more than once and guarantees that all three fields have the same distribution. The procedure `setBlockVector(h,h0)` sets the field `h` from the input array `h0(:)`, which contains its initial values.

Having created field objects, the programmer decides to let each image perform work on the local blocks that it owns. For each of its local blocks, an image obtains the length of the block and a pointer into the block, with or without halos depending on how it is used in the difference formula. Each image performs the appropriate finite difference operation independently of the others. Synchronization among images occurs within the halo exchange operation, which uses co-array syntax internally to update overlapping halo regions. With a loop over some predetermined number of time steps, `tMax`, the code might look like the following:

```
do t=1,tMax
  do b=1,getNumLocalBlocks(u)
    m = getLocalBlockLength(u,b)
    hPtr => pointerToLocalBlock(h,b)
    uPtr => pointerToLocalBlockwithHalo(u,b)
    hPtr(1:m) = hPtr(1:m) - 0.5*H*(dt/dx)*(uPtr(2:m+1)-uPtr(0:m))
  enddo
  call cyclicHaloExchange(h)
  do b=1,getNumLocalBlocks(u)
    m = getLocalBlockLength(u,b)
    hPtr => pointerToLocalBlockwithHalo(h,b)
    uPtr => pointerToLocalBlock(u,b)
    vPtr => pointerToLocalBlock(v,b)
    uPtr(1:m) = uPtr(1:m) + F*dt*vPtr(1:m) &
                          - 0.5*G*(dt/dx)*(hPtr(2:m+1)-hPtr(0:m))
    vPtr(1:m) = vPtr(1:m) - F*dt*uPtr(1:m)
  enddo
  call cyclicHaloExchange(u)
  call cyclicHaloExchange(v)
enddo
```

The library hides from the programmer all the details of data distribution and all the details of how to exchange data between objects. The block vector objects themselves contain all the necessary information, and the procedures associated with them know how to perform the required operations.

4 Co-arrays in the Fortran 2008 Standard

One of the most important outcomes of the Pmodels Project is the decision of the Fortran Standards Committee to include co-arrays as an official feature of the language. Throughout this project, I have worked closely with the committee to define the co-array standard. The Forran 2008 standard is in its final stage of adoption with a projected release in 2010. The proposed co-array extension is now open for public comment, and I will be engaged with the committee to follow developments through this last stage.

Details of the co-array model have changed from our original proposal in light of comments from committee members and especially in light of new features of Fortran 2003 that arose after our original design. We now allow pointer components of co-array derived types to be assigned to local data. This is a very powerful way for programmers to use co-array syntax for communication among irregular objects allocated asymmetrically on different images. We also allow derived types to have co-array components. It is illegal for a co-array derived type to also have a co-array component. But we allow a derived type to be a co-array with no co-array components. We also allow a derived type, which is not a co-array, to have co-array components. Relaxing this restriction allows the programmer to emulate a form of inheritance that was very difficult to do with the original restriction on co-array components.

We have also redefined some of the synchronization intrinsic procedures for Co-Array Fortran . We originally allowed only a small amount of asynchronous behavior through the `sync_team()` procedure with an optional wait list. We had many requests for a `notify()/wait()` protocol that allows true asynchronous behavior among images. We have added these intrinsic procedures to the new specification, and we have redefined the `sync_team()` procedures in terms of the `notify()/wait()` protocol.

John Reid and I contributed technical reports [29] and full published papers [25, 19, 18, 28] to describe these changes as they evolved. We are now in the process of writing *The Co-array Book* to describe the official Fortran 2008 co-array extension.

Co-Array Fortran is finally on its way to a standard, portable parallel programming model. When that happens, it will be a very satisfying result of about fifteen years of my effort and a fitting highlight to the support given to Co-Array Fortran through the Pmodels grant over a five year period. Without support from that grant, Co-Array Fortran most likely would have withered on the vine.

5 Public Access to Co-Array Fortran for Education and Research

One of the major hurdles in front of people wanting to try Co-Array Fortran has been the lack of access to a Co-Array Fortran compiler. To attack this problem, I convinced Paul Muzio, at the Army High Performance Computing Research Center (AHPCRC) in Minneapolis, to make a Cray-X1 available as a public resource for teaching and research. The AHPCRC had a small prototype Cray-X1 for several years, and after upgrading to a larger machine, they were able to free the smaller one from its production load. For about a year, until the AHPCRC left Minneapolis, educators were able to request accounts on the AHPCRC machine so they could include Co-Array Fortran in their parallel programming courses.

Our first experiment along these lines, in collaboration with the DARPA HPCS Project, took place at the University of California, San Diego where Alan Snavely included Co-Array Fortran in his graduate-level course in parallel programming. I presented a Co-Array Fortran tutorial to his class with a live, interactive demonstration of how to use Co-Array Fortran on the Cray-X1 back at the AHPCRC in Minneapolis. The students were assigned a Sharks and Fishes problem using both Co-Array Fortran and MPI. Vic Basili's students from the University of Maryland measured their programming effort to compare their productivity using the two programming models. The students reported very favorable impressions of their experiences with Co-Array Fortran compared with their experiences with MPI. We performed the same experiment with John Gilbert's class at the University of California, Santa Barbara with similar results.

This work has resulted in a new metric space for productivity in software development [23]. This new metric space, which is an extension of my earlier work on computational action metrics [14], will change the way we think of productivity in a fundamental way. This work is also related to two companion papers [20, 22] that shows how to define a similar metric space based on computational action for programs as they execute.

6 Co-Array Python

The parallel programming model underlying the three language extensions, Co-Array Fortran, UPC, and Titanium, is independent of language and independent of architecture. The problem in demonstrating that point has been the lack of access to proprietary compilers, especially for Fortran, so that independent researchers could demonstrate the utility of these models. To further emphasize the point, we have implemented the Co-Array Fortran model using the Python language and have called it Co-Array Python [27]. Because Python is an open source language, it was a simple matter to insert co-array syntax into the language and to program a simple Laplace solver in the new language. The project was not intended to show high performance, which no one expects from an interpreted language, but rather to show the ease of implementation and the increase in productivity for programmers writing code using this programming model.

7 Fortran 90 Interface to ARMCI

In the absence of standard Co-Array Fortran compilers, we have designed a Fortran 95 interface for the ARMCI Library [12]. This library is a portable communication library written in C that provides one-sided message-passing capability, which is more efficient than MPI code, and it is available on more platforms than the popular Shmem Library. In the absence of a Co-Array Fortran compiler, ARMCI is the next best approach to the underlying programming model based on one-sided communication. This new interface will allow programmers to write code that switches between co-array syntax and ARMCI library calls depending on the availability of a Co-Array Fortran compiler.

Since Fortran 95 has no standard interface to the C language, it has been a challenge to design an interface that works for all compilers. Fortran 2003 now defines a standard interface to the C language so this design should be looked at again.

8 Collaboration with Earth Sciences

In collaboration with NASA Goddard Space Flight Center and the Geophysical Fluid Dynamics Laboratory, I started a project to investigate the use of Design Patterns for the development of large-scale application codes for the Earth Sciences. We have identified six design patterns for grid-codes, codes that apply finite difference operators to fields defined on underlying physical grids[5]. We implemented a prototype shallow water example written in Java using the Driver/Kernel model [6] emulating the CafLib design. We used the Strategy Pattern to define the kind of architecture involved. A Builder Pattern takes the strategy as input and returns a Composite Pattern, which represents the domain decomposition of the problem for a particular architecture. We used the Mediator Pattern to pick the best communication method for a given architecture, and we used the Observer Pattern to implement asynchronous communication among domains transparently to the programmer. The Iterator Pattern allows the programmer to traverse the domain decomposition either locally or globally independent of architecture. By picking different strategies, we can use either MPI or Shmem or ARMCI or Co-Array Fortran for communication. The applications programmer, writing numerical kernels, never knows which one is used in the driver part of the code.

9 Publications Resulting from the Project

9.1 Peer-Reviewed Publications

Robert W. Numrich, A Metric Space for Computer Programs and The Principle of Computational Least Action, *The Journal of Supercomputing*, 43(3):281-298, 2008.

Robert W. Numrich, The computational energy spectrum of a program as it executes, *The Journal of Supercomputing*, Under review, 2008.

John Reid and Robert W. Numrich, Co-arrays in the next Fortran Standard, *Scientific Programming*, 15(1): 9-26, 2007.

Robert W. Numrich, A note on scaling the Linpack benchmark, *Journal of Parallel and Distributed Computing*, 67(4): 491-498, 2007.

Ricky A. Kendall, Masha Sosonkina, William D. Gropp, Robert W. Numrich, Thomas Sterling, Parallel Programming Models Applicable to Cluster Computing and Beyond, in *Numerical Solution of Partial Differential Equations on Parallel Computers*, Are Magnus Bruaset and Aslak Tveito, editors, Lecture Notes in Computational Science and Engineering, 51, 3-54, Springer 2006.

Robert W. Numrich, A Parallel Numerical Library for Co-Array Fortran, *Parallel Processing and Applied Mathematics: Proceedings of the Sixth International Conference on Parallel Processing and Applied Mathematics (PPAM05)*, Springer Lecture Notes in Computer Science, LNCS 3911, 960-969, Poznan, Poland, September 11-14, 2005.

Robert W. Numrich, Parallel numerical algorithms based on tensor notation and Co-Array Fortran syntax, *Parallel Computing*, 31, 588-607, 2005.

Robert W. Numrich and John Reid, Co-arrays in the next Fortran Standard, *ACM Fortran Forum*, 24(2): 2-24, 2005.

J. Nieplocha, D. Baxter, V. Tipparaju, C. Rasmussen, and Robert W. Numrich, Symmetric Data Objects and Remote Memory Access Communication for Fortran 95 Applications, *Euro-Par 2005 Parallel Processing: 11th International Euro-Par Conference*, August 30-September 2, Lisbon, Portugal, Lecture Notes in Computer Science Number 3648, Springer-Verlag GmbH, 720-729, 2005.

V. Balaji and Robert W.Numrich, A Uniform Memory Model for Distributed Data Objects on Parallel Architectures, *Use of High-Performance Computing in Meteorology*, Walter Zwieflhofer and George Mozdzynski editors, World Scientific Publishing Co., 272-294, 2005.

Robert W. Numrich, Lorin Hochstein, Victor Basili, A Metric Space for Productivity Measurement in Software Development, *Proceedings SE-HPCS'05, Second International Workshop on Software Engineering for High Performance Computing System Applications*, St. Louis, Missouri, May 15, 2005.

Craig E. Rasmussen, Matthew J. Sottile, Jarek Nieplocha, Robert W. Numrich, Eric Jones, Co-Array Python: A Parallel Extension to the Python Language, *Euro-Par 2004 Parallel Processing: 10th International Euro-Par Conference*, Pisa, Italy, August 31-September 3, Lecture Notes in Computer Science Number 3149, Springer-Verlag GmbH, 632-637, 2004.

9.2 Conferences, Workshops, Tutorials, Technical Reports

Robert W. Numrich, Combining Object-oriented Techniques with Co-arrays in Fortran 2008, *13th SIAM Conference on Parallel Processing for Scientific Computing (PP08)*, Atlanta, GA, March, 2008.

Robert W. Numrich, Computational Forces in the Linpack Benchmark, *13th SIAM Conference on Parallel Processing for Scientific Computing (PP08)*, Atlanta, GA, March 12, 2008.

Robert W. Numrich, A Parallel Numerical Library for Co-Array Fortran, Louisiana State University, CCT Seminar, January 17, 2007.

Robert W. Numrich, A New Scaling Formula for the Linpack Benchmark, *SIAM Conference on Computational Science and Engineering*, Costa Mesa, CA, February 19-23, 2007.

Robert W. Numrich, Tutorial on Co-Array Fortran, Albert Einstein Institute, Potsdam, Germany, September 14, 2007.

Robert W. Numrich, Tutorial on Co-Array Fortran, University of Southern California, Los Angeles, graduate course in parallel programming, January 12, 2006

Robert W. Numrich, Ragged allocatable/pointer co-arrays, Technical Report J3/06-135, (http://www.j3-fortran.org), February, 2006.

Robert W. Numrich, Simplification of the co-array proposal, Technical Report J3/06-134, (http://www.j3-fortran.org), February, 2006.

Robert W. Numrich, Tutorial on Co-Array Fortran, *The 20th ACM International Conference on Supercomputing*, Cairns, Queensland, Australia, June 28-July 1, 2006.

Robert W. Numrich, Co-Array Fortran, *Workshop on Programming Languages for High Performance Computing (HPC WPL)*, Sandia National Laboratory, Albuquerque, NM, December 13, 2006.

Robert W. Numrich, Tutorial on Co-Array Fortran, University of California, San Diego, graduate course in parallel programming, January 11, 2005.

Robert W. Numrich, Tutorial on Co-Array Fortran, University of California, Santa Barbara, graduate course in parallel programming, April 13, 2005.

V. Balaji, Thomas L. Clune, Robert W. Numrich and Brice T. Womack, An Architectural Design Pattern for Problem Decomposition, *Workshop on Patterns in High Performance Computing*, Champaign-Urbana, IL, May 4-6, 2005.

Robert W. Numrich, Tutorial on Co-Array Fortran, *SIAM Conference on Parallel Processing for Scientific Computing*, San Francisco, CA, February 25-27, 2004.

Robert W. Numrich, Tutorial on Co-Array Fortran, *New Methods for Developing Peta-Scale Codes*, Pittsburgh Supercomputing Center, May 3-4, 2004.

Robert W. Numrich, Observations on Parallel Languages, *High Productivity Programming Languages and Models*, Santa Monica, CA, May 17-20 2004.

Robert W. Numrich, Tutorial on Co-Array Fortran, NASA Goddard Space Flight Center, summer intern program, July 27, 2004.

Robert W. Numrich, Tutorial on Co-Array Fortran, University of Maryland graduate seminar, September 28, 2004.

Robert W. Numrich, Co-Array Fortran in HPCS, *HPC User Forum*, Tucson, AZ, September 20-22, 2004.

Robert W. Numrich, Tutorial on Co-Array Fortran, *17th International Parallel and Distributed Processing Symposium (IPDPS)*, Nice, France, April 22-26, 2003.

Robert W. Numrich, An overview of Co-Array Fortran with Some Recent Results, SCICOMP8, Minneapolis, MN, August 5-8, 2003.

Robert W. Numrich, Co-Array Fortran: What is it? Why should you put it on BlueGene/L?, *Blue Gene/L Workshop*, Reno, NV, October 14, 2003.

Robert W. Numrich, Tutorial on Co-Array Fortran, *Supercomputing 2003*, Phoenix, AZ, November 15-21, 2003.

10 Future Work

Now that Fortran 2003 is a true object-oriented language, CafLib should be rewritten with a new design based on inheritance. Emulating object-oriented design using Fortran 95 is awkward and difficult [1, 7, 8, 11, 13]. Without inheritance, many pieces of code are replicated over and over again. Without true constructors and destructors, dealing with memory leaks is very difficult. These object-oriented features are now part of Fortran 2003, and they can be used to advantage to improve productivity for programming high-performance, parallel scientific codes [21].

The new design would be based on Abstract Maps as indicated already in Figure 1. In Fortran 2003, these maps would look something like the following:

```
module AbstractMap
type,abstract :: Map
  integer,private :: numberOfObjects = 0
contains
  procedure(GtoL),deferred,pass :: getGlobalToLocal
  procedure(LtoG),deferred,pass :: getLocalToGlobal
  procedure                     :: getNumberOfObjects
end type Map
abstract interface
  integer function GtoL(a,iGlobal) result(iLocal)
    import Map
    class(Map),intent(in) :: a
    integer,intent(in)    :: iGlobal
  end function GtoL
end interface
contains
  integer function getNumberOfObjects(a) result(n)
    class(Map),intent(in) :: a
    n = a\%numberOfObjects
  end function getNumberOfObjects
end module AbstractMap
```

Every concrete extension to the abstract map inherits the private component variable `numberOfObjects` and a public function `getNumberOfObjects()` that returns its value. Every concrete extension must provide a function `getGlobalToLocal()` that maps global objects to local objects and a function `getLocalToGlobal()` that maps local objects to global objects. These are the composite functions represented in Figures 2 and 3.

An ObjectMap is an extension of the Abstract Map, for example,

```
module ObjectMap
use AbstractMap
type,extends(Map)               :: ObjectMap
  integer,private,allocatable   :: local(:)
contains
  procedure,private,nopass      :: newObjectMap
  generic,public                :: ObjectMap => newObjectMap
  procedure                     :: getGlobalToLocal
  procedure                     :: getLocalToGlobal
  final                         :: delete
end type ObjectMap
contains
  integer function getGlobalToLocal(a,global) result(local)
    class(ObjectMap),intent(in) :: a
    integer,intent(in)          :: global
    local=a%local(global)
  end function getGlobalToLocal
end module ObjectMap
```

It adds an allocatable component array `local(:)` that holds the local object number for each global object. It also adds a generic constructor `ObjectMap`, not shown, that allocates the memory for the array and fills in the correct values based on some mapping function particular to each specific kind of map. It also implements a destructor `final()`, not shown, that deallocates memory whenever the map goes out of scope in a program. Finally, it implements the functions `getGlobalToLocal(g)`, shown, and `getLocalToGlobal`, not shown.

In the new CafLib, I would add sparse matrix objects along with the dense matrix objects already included in the old CafLib. I decided about half way through the project that it did not make sense to include these new objects without true inheritance, which is just now becoming available from compiler vendors. I would also include grids and fields as objects in their own right as extensions of vector and matrix objects rather than leaving it up to the programmer to define them as shown in the shallow water example.

I intend to continue work with the Fortran Standards Committee as it takes Co-Array Fortran through the long standardization process. I will continue to work with John Reid writing the official specification of the language and to help him with the official edits to the standard document. We also intend to write a book on Co-Array Fortran and to include it as a chapter in the next edition of Fortran 2003/2008 Explained.

I am actively encouraging compiler vendors to provide early implementations of the co-array extension in Fortran 2008. With all new hardware being based on multicore chips, a first implementation for a single multicore chip with shared memory is very easy to produce. In fact, the first implementation of co-arrays took place on an SGI Origin 2000 with globally addressable, distributed shared memory [24]. It only required a simple partitioning of the memory. I hope the vendors will take this route as a quick way to implement co-arrays well in advance of Fortran 2008 becoming the official standard.

References

[1] J. E. Akin. *Object-oriented programming via Fortran 90/95*. Cambridge University Press, 2003.

[2] Albert Cahn, Jr. An investigation of the free oscillations of a simple current system. *Journal of Meteorology*, 2(2):113–119, June 1945.

[3] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, second edition, 1995.

[4] Akio Arakawa and Vivian R. Lamb. Computational design of the basic dynamical processes of the UCLA general circulation model. *Methods in Computational Physics*, 17:173–265, 1977.

[5] V. Balaji, Thomas L. Clune, Robert W. Numrich, and Brice T. Womack. An architectural design pattern for problem decomposition. In *Workshop on Patterns in High Performance Computing*, Champaign-Urbana, IL, May 4-6, 2005.

[6] V. Balaji and Robert W. Numrich. A uniform programming model for complex distributed data objects in distributed and shared memory. In *Proceedings ECMWF High Performance Computing Workshop*. World Scientific, 2004.

[7] V. K. Decyk, C. D. Norton, and B. K. Szymanski. How to express C++ concepts in Fortran 90. *Scientific Programming*, 6(4):363, 1998.

[8] Viktor K. Decyk, Charles D. Norton, and Boleslaw K. Szymanski. How to support inheritance and run-time polymorphism in Fortran 90. *Computer Physics Communications*, 115:9–17, 1998.

[9] Jack J. Dongarra and David W. Walker. Software libraries for linear algebra computations on high performance computers. *SIAM Review*, 37(2):151–180, June 1995.

[10] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Publishing Company, 1995.

[11] Mark G. Gray and Randy M. Roberts. Object-based programming in Fortran 90. *Computers in Physics*, 11(4):355–361, July-August 1997.

[12] J. Nieplocha, D. Baxter, V. Tipparaju, C. Rasmussen, and Robert W. Numrich. Symmetric data objects and remote memory access communication for fortran 95 applications. In *Euro-Par 2005 Parallel Processing: 11th International Euro-Par Conference, August 30-September 2, Lisbon, Portugal*, pages 720–729. Lecture Notes in Computer Science Number 3648, Springer-Verlag GmbH, 2005.

[13] C.D. Norton, V.K. Decyk, and B.K. Szymanski. High performance object-oriented scientific programming in Fortran 90. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*. SIAM Activity Group on Supercomputing, Society for Industrial and Applied Mathematics, March 1997. CD ROM format.

[14] Robert W. Numrich. Performance metrics based on computational action. *International Journal of High Performance Computing Applications*, 18(4):449–458, 2004.

[15] Robert W. Numrich. A Parallel Numerical Library for Co-Array Fortran. In *Parallel Processing and Applied Mathematics: Proceedings of the Sixth International Conference on Parallel Processing and Applied Mathematics (PPAM05)*, pages 960–969, Poznan, Poland, September 11-14 2005. Springer Lecture Notes in Computer Science, LNCS 3911.

[16] Robert W. Numrich. Parallel numerical algorithms based on tensor notation and Co-Array Fortran syntax. *Parallel Computing*, 31:588–607, 2005.

[17] Robert W. Numrich. CafLib Users' Manual: Release 1.2. Available from the author, 2006.

[18] Robert W. Numrich. Ragged allocatable/pointer co-arrays. Technical Report J3/06-135, (http://www.j3-fortran.org), February 2006.

[19] Robert W. Numrich. Simplification of the co-array proposal. Technical Report J3/06-134, (http://www.j3-fortran.org), February 2006.

[20] Robert W. Numrich. A metric space for computer programs and the principle of computational least action. *The Journal of Supercomputing*, 43(3):281–298, March 2008.

[21] Robert W. Numrich. Combining Object-oriented Techniques with Coarrays in Fortran 2008. 13th SIAM Conference on Parallel Processing for Scientific Computing (PP08), Atlanta, GA, March, 2008.

[22] Robert W. Numrich. The computational energy spectrum of a program as it executes. The Journal of Supercomputing, Under review, 2008.

[23] Robert W. Numrich, Lorin Hochstein, and Victor Basili. A metric space for productivity measurement in software development. In *Proceedings SE-HPCS'05, Second International Workshop on Software Engineering for High Performance Computing System Applications*, St. Louis, Missouri, May 15, 2005.

[24] Robert W. Numrich, John Reid, and Kieun Kim. Writing a multigrid solver using Co-Array Fortran. In Bo Kågström, Jack Dongarra, Erik Elmroth, and Jerzy Waśniewski, editors, *Applied Parallel Computing: Large Scale Scientific and Industrial Problems*, pages 390–399. 4th International Workshop, PARA98, Umeå, Sweden, June 1998, Springer, 1998. Lecture Notes in Computer Science 1541.

[25] Robert W. Numrich and John K. Reid. Co-arrays in the next Fortran Standard. *ACM Fortran Forum*, 24(2):2–24, 2005.

[26] The ScaLAPACK Project. www.netlib.org/scalapack.

[27] Craig E. Rasmussen, Matthew J. Sottile, Jarek Nieplocha, Robert W. Numrich, and Eric Jones. Co-Array Python: A parallel extension to the python language. In *Euro-Par 2004 Parallel Processing: 10th International Euro-Par Conference,*, pages 632–637. Lecture Notes in Computer Science Number 3149, Springer-Verlag GmbH, Pisa, Italy, August 31- September 3, 2004.

[28] John Reid and Robert W. Numrich. Co-arrays in the next Fortran Standard. *Scientific Programming*, 15(1):9–26, 2007.

[29] John K. Reid and Robert W. Numrich. Co-Array Fortran for parallel programming. Technical Report JTC1/SC22/WG5 N1592 (http://www.nag.co.uk/sc22wg5/ftp.html), ISO/IEC, Geneva, May, 2004.