High-Performance Fortran as a Possible Successor to Global Arrays in the NWChem Parallel Computational Chemistry Code*

Yuhong Wen, D. Bryan Carpenter,
Erol Akarsu, Tomasz Haupt, and David E. Bernholdt†

Northeast Parallel Architectures Center
Syracuse University
111 College Place
Syracuse, NY 13422–4100

June 30, 1997

Abstract

This is the abstract

1 Introduction

When the development of the NWChem parallel computational chemistry package began at the Pacific Northwest National Laboratory (PNNL) in 1993, the question of what parallel programming environment to use received careful attention. It was clear that a straightforward message-passing model would place a significant burden on the programmer. High-Performance Fortran (HPF) provided a higher-level programming model, but suffered from several other problems. In the first place, version 1.0 of the HPF standard addressed only data-parallel operations, while it was clear that both data- and task-parallel capabilities would be required for the algorithms envisioned in NWChem. Second, at that time, the standard had not yet been formally approved, and such HPF implementations as did exist were based on drafts of the standard, were incomplete in their implementation of the language, and were rather immature.

As a result, the NWChem developers decided to implement their own parallel programming model, *Global Arrays*, which would be modeled to a large extent on HPF. The hope was that a future version of HPF would support the capabilities required by NWChem, and

*Prepared for Battelle/Pacific Northwest National Laboratory under contract 315647-A9E.

†Author for correspondence. E-mail address *bernhold@npac.syr.edu*.

that given time, compilers would become more complete and robust, making it possible to transition NWChem from the global array toolkit (GA) to HPF, thereby accruing the benefits of an accepted standard and vendor-supported compilers.

Since the HPF 1.0 standard has now been available for several years, and since version 2.0 of the language standard (currently in a public comment period prior to formal acceptance) includes some task parallelism and other features required by NWChem algorithms, it is appropriate to evaluate the present state of HPF and the prospects for using it to replace the GA toolkit in NWChem and other similar computational chemistry applications.

There are two components to this analysis. The first is an examination of current HPF implementations, and their performance in comparison with GAs. The second is an examination of whether or not the HPF 2.0 standard supports the task parallel and other capabilities required by the algorithms implemented in NWChem.

2 Performance of HPF and GAs

HPF is a fairly new language, the first version of the standard having been approved in 1993, and it has been criticized by some for going too far beyond the realm of what compiler writers currently understand how to do well. Moreover, the array syntax HPF-1 relies on to express many of its parallel functionality comes from Fortran 90, itself a recent standard, with significant additions to its predecessor, Fortran 77.

For HPF to replace GAs in NWChem and other applications, it clearly must be able to provide performance on par with global arrays. It is not immediately obvious how hard this should be to achieve. Global arrays support a wide range of platforms with a single implementation while in most cases, HPF compilers are produced by hardware vendors and each implementation is targeted to a specific platform. On the other hand, GAs are a relatively small package with a more limited range of application so far than most commercial HPF products. This has allowed for a measure of tuning which perhaps has not yet been possible in HPF compilers. HPF compilers have the additional advantage that it is possible to recognize some programming idioms and replace them with library calls or specially optimized code. To the extent that this is done, HPF implementations have some ability to make up for user-written code that may, in its original form, lead to reduced performance. Obviously this is not possible with GAs because they are used directly as a library.

Given that version 1 of HPF clearly does not provide sufficient functionality to replace GAs, it would not be cost-effective to perform an extensive comparison at this time–we presume that by the time HPF is a viable alternative from a functional point of view, the performance will likely have improved as well. Instead we have chosen a simpler comparison intended to provide a feel for how the performance of HPF and GAs (and simple message passing) compare at this point in time.

The test cases chosen are mostly collective communications primitives which appear in many types of applications, including computational chemistry, along with a couple of small computational kernels not specifically tied to computational chemistry. We felt that operations of this sort would be an effective test of the ability of compilers (or libraries) on communication-intensive problems, which is likely to be the weakest point of most current implementations. In Table 1 we compare release 2.1 of the Global Array Toolkit with both PGI's and IBM's HPF compiler offerings (versions 2.1 and 1.1 respectively) on the Cornell Theory Center's IBM SP machine (POWER2 "thin" nodes, version 2 of the High-Performance Switch). All reported timings are for 8-node runs. Timings for the Message Passing Interface (MPI) versions of the benchmarks are also presented. Most of the collective communications primitives are standard MPI library calls.

Some experimentation and tuning was done with the HPF implementations of the algorithms in order to obtain reasonable performance–most especially in the data distributions. As the results indicate, the two HPF compilers tested often give rather different performance for the same code. In these cases it may be possible to specially tune the code for poorly performing compiler, at the expense of the other, but all reported results use the same implementation for both HPF compilers. GA implementations were derived from the HPF implementations and underwent less tuning. In some cases, the particular benchmark might have been more efficiently implemented using message passing, since this is entirely compatible with the GA programming model. In such cases, however, the MPI results can taken as indicative of what can be achieved by resorting to message passing, while the results given for GA are like-to-like comparisons with the HPF algorithms. Appendix A provides a brief description of the benchmarks and their whether they were implemented using library calls or explicitly coded.

If we first compare the two HPF implementations, we see that PGI's product ran faster in nine cases, IBM's in seven cases, and they effectively tied (less than 10% difference) in only three cases. When the relative timings for the two HPF compilers are compared in Table 2, it is clear that when PGI's compiler is the slower of the two, it is significantly slower (a minimum of 3.9 times), while when XLHPF is slower, the performance difference is not nearly so large (in only two of nine cases is XLHPF more than 3.9 times slower). Looking closer, it seems that apart from broadcasts, PGI's basic communications implementations are better, while IBM's matrix operations perform better.

Comparing the two HPF implementations with MPI results is also an interesting exercise. As with the two HPF compilers, the results are highly variable, with the MPI version running fastest in eight of the 19 tests and outperforming at least one of the HPF versions in 14 cases.

The MPI implementations of many tests involve direct library calls, so we might naively expect that in such cases the MPI timings will provide a lower bound for the HPF results. A glance at Table 1 indicates that this is not the case. While both of the HPF compilers do target message passing as their underlying communication method, their run-time environments are clearly not using the same implementations of primitive algorithms that are used in the MPI library.

The wide variability seen in the three implementations considered so far does not make it easy to analyze the GA results. The GA implementation ran faster than both HPF versions in only four cases (better than MPI in three), and faster than at least one HPF implementation in nine instances. In cases where GAs are the worst of the three, the difference in performance is usually substantial. As observed above, the GA implementations of these benchmarks are generally adaptations of the HPF versions. This means that they do not necessarily provide the best performance GAs are capable of for the benchmark operation,

Table 1: Comparison of MPI, HPF (two implementations), and Global Array implementations of some simple communication-intensive algorithms run on 8 nodes of the Cornell Theory Center IBM SP.

	Execution Time (seconds)			
Benchmark	MPI	PGI HPF	IBM XLHPF	GA
All-to-all broadcast	6.02	20.38	2.49	16.07
All-to-all personalized	5.50	2.79	2.81	16.17
All-to-all reduction	2.91	24.00	58.37	6.26
All-to-one maximum	2.28	4.65	13.98	39.35
All-to-one reduction	11.91	3.89	14.26	34.61
One-to-all broadcast	1367.00	3.88	0.34	3.85
One-to-all personalized	1.88	2.00	3.21	1.88
Parallel prefix	1.12	2.33	3.45	13.90
Scatter into matrix	2.75	1.98	4.34	30.16
Gather from matrix	3.50	1.37	0.14	30.19
Matrix transpose	5.00	11.55	2.00	17.00
Matrix-vector mult.	1.00	0.58	0.15	0.20
Matrix multiplication	13.75	235.00	33.45	749.5
Matrix shift	1.11	14.68	14.73	2.54
FFT	0.02	0.98	89.28	0.63
Floyd's algorithm	49.13	19.04	17.78	32.67
Gaussian elimination	5.74	122.00	3.34	9.84
Min. spanning tree	9.75	7.36	66.95	36.57
Laplace solver	22.01	25.00	54.38	32.63

but they facilitate a like-for-like comparison with the analogous HPF implementations.

It seems fairly clear that for these benchmarks, all three approaches (MPI, HPF, GA) can be considered comparable in a very rough sense. Specifically, HPF at this stage of its development, seems like a viable alternative to GAs from the performance point of view. It also seems fairly clear that all three approaches have room for improvement in their performance. This would certainly involve work on the run-time environment (libraries). For the compilers, it may also involve improvements in the "front-end", to improve translation from native HPF into the particular run-time environment chosen (for example Fortran77 + MPI message passing, in the case of PGI HPF). We have not examined the code produced by the two HPF implementations considered to assess the extent to which this contributes to the observed performance differences.

These benchmarks are very limited both in terms of breadth of algorithms tested and in systems and platforms used. Further benchmarking efforts aimed not at the hardware platforms, as is usually the case, but rather at the parallel programming models, would be most useful in refining our understanding of the performance of these approaches, and would perhaps spur improvements in some products. For the purposes of this particular study, it would be most effective to undertake such a benchmarking effort when HPF-2

Table 2: Timings for PGI and IBM HPF systems and GAs relative to fastest HPF result.

	Timings Relative to Best HPF		
Benchmark	PGI HPF	IBM XLHPF	GA
All-to-all broadcast	8.18	1.00	6.45
All-to-all personalized	1.00	1.01	5.75
All-to-all reduction	1.00	2.43	0.28
All-to-one maximum	1.00	3.01	8.46
All-to-one reduction	1.00	3.67	8.90
One-to-all broadcast	11.41	1.00	11.32
One-to-all personalized	1.00	1.61	0.94
Parallel prefix	1.00	1.48	5.97
Scatter into matrix	1.00	2.19	15.23
Gather from matrix	9.79	1.00	215.64
Matrix transpose	5.78	1.00	8.50
Matrix-vector mult.	3.87	1.00	1.33
Matrix multiplication	7.03	1.00	22.41
Matrix shift	1.00	1.00	0.17
FFT	1.00	91.10	0.64
Floyd's algorithm	1.07	1.00	1.84
Gaussian elimination	36.53	1.00	2.95
Min. spanning tree	1.00	9.10	4.97
Laplace solver	1.00	2.18	1.31

compilers begin appearing, so as to include some of the new features in the tests.

3 HPF 2.0 Capabilities Analysis

Version 2.0 of the High-Performance Fortran standard is currently in a public comment period prior to formal acceptance. This version of HPF incorporates some support for task-based parallelism as well as other features similar to those in the GA toolkit. In order to understand the degree to which HPF 2.0 supports the GA programming model, we have looked at the requirements of two algorithms implemented in NWChem which are exemplars of the kind of task parallelism used throughout the code.

We must note that since there are no HPF 2.0 compilers presently available, the proposed implementations we outline are "thought experiments" and subject to all of the uncertainties of both interpretation of the standard itself, and actual implementations.

3.1 NWChem Distributed Data SCF Algorithm

Say something about why HPF 1 can't handle this

Say something about new HPF 2 capabilities that you considered

Say something about what you think is the best approach and why. Sketch code required to implement.

If "standard extensions" are required, please discuss how likely they are to be implemented (say something about utility to other application domains if you know).

Is there anything relevant in the standard which you fell is poen to interpretation and may effect the ability or performance of your proposed implementation? Talk about any modifications or additional features you might propose adding to the language to improve performance, ease of programming, etc. (Anything we should comment on in the public comment period?)

3.2 NWChem MP2 Back-Transformation Algorithm

4 Conclusions and Recommendations

A Benchmark Algorithm Descriptions

We provide brief descriptions of the benchmark algorithms of Section 2, noting whether each was implemented as a library call or explicitly coded.

All-to-all broadcast All processors exchange data. MPI library call. Explicitly implemented for both HPF and GA.

All-to-all personalized Each node sends different data to all other nodes. MPI library call. Explicitly implemented for both HPF and GA.

All-to-all reduction MPI and GA (`GA_DGOP`) library call; explicitly implemented for HPF.

All-to-one maximum All-to-one reduction, but with maximum function. MPI library call; explicitly implemented for both HPF and GA.

All-to-one reduction Reverse of one-to-all broadcast. MPI library call; explicitly implemented for both HPF and GA.

One-to-all broadcast Straightforward broadcast from a single processor. MPI library call; explicitly implemented for both HPF and GA.

One-to-all personalized Single node sends different data to all other nodes. MPI library call; explicitly implemented for both HPF and GA.

Parallel prefix MPI library call; explicitly implemented for both HPF and GA.

Scatter into matrix GA library call; explicitly implemented in MPI and HPF

Gather from matrix GA library call; explicitly implemented in MPI and HPF

Matrix transpose GA and HPF library calls; explicitly implemented in MPI

Matrix-vector mult. Explicitly implemented for all.

Matrix Multiplication GA and HPF library calls; explicitly implemented in MPI

Matrix shift HPF library call; explicitly implemented in MPI and GA

FFT Explicitly implemented for all.

Floyd's Algorithm "All-to-all shortest paths on a graph". Explicitly implemented for all.

Gaussian elimination Explicitly implemented for all.

Min. spanning tree Using Prim's algorithm. Explicitly implemented for all.

Laplace solver SOR solver for Laplace's Equation. Explicitly implemented for all.