High Performance Cholesky and FFT Algorithms in HPF

Harald J. Ehold
VCPC, European Centre for Parallel Computing at Vienna
ehold@vcpc.tuwien.ac.at

Wilfried N. Gansterer
Institute for Applied and Numerical Mathematics
Vienna University of Technology
ganst@aurora.tuwien.ac.at

Herbert Karner
Institute for Applied and Numerical Mathematics
Vienna University of Technology
karner@aurora.tuwien.ac.at

Dieter F. Kvasnicka
Institute for Physical and Theoretical Chemistry
Vienna University of Technology
kvasnicka@tuwien.ac.at

Christoph W. Ueberhuber
Institute for Applied and Numerical Mathematics
Vienna University of Technology
christof@uranus.tuwien.ac.at

July 1999

AURORA TR1999-10

The work described in this report was supported by the Special Research Program SFB F011 "AURORA" of the Austrian Science Fund.

Abstract

Numerical algorithms written in HPF often have to pay performance penalties, when they are compared with optimized numerical libraries on single processors. Obviously, this is a problem for achieving high *parallel* performance, too. In order to overcome these efficiency problems a method to utilize existing numerical single processor software for computationally expensive kernels within HPF is proposed. This technique enables an even lower parallelization overhead than numerical libraries parallelized using MPI. In addition to performance benefits, the utilization of suitable building blocks from existing numerical libraries saves coding and debugging effort, in particular for large mathematical applications.

The methods suggested utilize HPF's EXTRINSIC mechanism and are independent of implementation details of HPF compilers.

Chapter 1

Introduction

The integration of numerical libraries (for example [1, 5, 9, 10, 13, 15]) into HPF is crucial for several reasons:

- In many scientific applications a lot of programming effort can be saved by using existing library routines as building blocks instead of (re-)coding them in HPF directly.
- A considerable amount of expertise has been incorporated into high quality software packages like the ones mentioned above. It is hardly possible to achieve comparable floating-point performance when coding in HPF directly. Even if a problem is well suited for an HPF implementation, the success of such an attempt heavily relies on the maturity of HPF compilers, which cannot be taken for granted at present (see Ehold et al. [12]).
- When standard numerical operations are coded in HPF the resulting code often suffers from poor local performance. Thus, one of the key issues is to optimize local performance in order to improve overall parallel performance.
- Highly optimized BLAS implementations are available for most target systems. On systems without vendor optimized BLAS routines optimized BLAS libraries can be built using ATLAS (Bilmes et al. [3]) or PHIPAC (Whaley, Dongarra [19]). Therefore, the use of BLAS routines for local computations ensures *performance portability*.
- Usually the main motivation for parallelization is performance improvement. It is essential to optimize the *sequential* code first in order to be able to measure the benefits of parallelization in a clean way. The sequential optimization typically involves restructuring the code, for

example, by increasing the fraction of Level 3 (matrix-matrix) operations and by using appropriate Level 3 BLAS routines (or other high performance library routines) wherever possible. Thus, when the parallelization is done by using HPF, the necessity of combining the BLAS (and other numerical packages and libraries) with HPF arises quite naturally.

For all of these reasons the goal of this report is to investigate ways to utilize high performance numerical libraries in an HPF context. This report concentrates on solving high-level algorithms.

The basic facility provided by HPF for integrating procedures from other programming languages or models is the EXTRINSIC mechanism (HPF Forum [14]). This report describes methods for calling existing sequential and parallel library routines from HPF using this mechanism. The techniques described are *portable* in the sense that they only rely on features from the HPF standard [14] and, additionally, on very few elements of the HPF 2.0 Approved Extensions. In particular, the required HPF features are:

- an advanced form of the ALIGN directive, namely
  `ALIGN A(j,*) WITH B(j,*)`
  (replication of `A` along one dimension of `B`);
- the INHERIT directive;
- EXTRINSIC(HPF_LOCAL) and EXTRINSIC(F77_LOCAL) subroutines;
- the LOCAL_TO_GLOBAL subroutine of the HPF_LOCAL_LIBRARY.

The main ideas and basic concepts presented in this report are applicable to many numerical algorithms, but their implementation will differ in various technical details. For the purpose of illustration two algorithms, which arise very frequently at the core of scientific applications, serve as prototypes: The Cholesky factorization and the two-dimensional FFT.

Chapter 2

Calling Sequential Routines from HPF

The topic of this section is the integration of extrinsic library routines into an HPF program for *local* computations only. All of the communication in the multiprocessor environment is organized by the HPF compiler. Even if the HPF compiler does a good job with organizing data distribution and communication between processors, the performance achieved can be disappointingly low due to bad node performance.

2.1 Cholesky Factorization.

Experiments have been performed with several HPF implementations of algorithms for computing the Cholesky factorization $A = LL^\top$ of a real symmetric positive definite matrix A.

The principle factorization of the right-looking blocked version of the Cholesky algorithm is the following:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} L_{11}^\top & L_{21}^\top \\ 0 & L_{22}^\top \end{pmatrix} = \begin{pmatrix} L_{11}L_{11}^\top & L_{11}L_{21}^\top \\ L_{21}L_{11}^\top & L_{21}L_{21}^\top + L_{22}L_{22}^\top \end{pmatrix}.$$

This factorization involves the following three tasks: (i) factorize $A_{11} = L_{11}L_{11}^\top$ using the routine `LAPACK/dpotrf`, (ii) solve the linear system $L_{21}L_{11}^\top = A_{21}$ for L_{21} using `BLAS/dtrsm`. (iii) multiply $L_{21}L_{21}^\top$ using the symmetric matrix update routine `BLAS/dsyrk` and continue recursively with $A_{22} - L_{21}L_{21}^\top = L_{22}L_{22}^\top$.

In the HPF + BLAS version the first task (`LAPACK/dpotrf`)—which is computationally negligible—is done sequentially on one processor, but the subsequent steps (`BLAS/dtrsm` and `BLAS/dsyrk`) are parallelized. The choice of the order b of sub-matrix A_{11} implies a (CYCLIC(kb), CYCLIC(lb)) distribution of the symmetric matrix A, with $k, l \in \mathbb{N}$.

ScaLapack, native HPF, and the HPF+Blas routine use this algorithm. It should be noted, however, that (i) the HPF + Blas code is very short and easy to understand, and (ii) the parallelization overhead is very small.

It turned out that the sequential Blas-based Cholesky factorization (the routine `LAPACK/dpotrf`) is more than 10 times faster than the best native HPF code[1] on one processor and still 1.5 times faster than this code on 8 processors, as shown in Table 2.1. The newly developed parallel HPF code utilizing optimized Blas routines is faster than the corresponding ScaLapack routine and shows a similar speed-up as this highly tuned routine. For 8 processors the amount of data to be transmitted turns out to be the limiting factor for both ScaLapack and HPF+Blas.

Table 2.1: Cholesky factorization of a matrix of order 3000 on a Beowulf cluster (five dual Pentium II PCs with 350 MHz connected via Fast Ethernet), using (i) Lapack, (ii) a native HPF routine, (iii) ScaLapack, and (iv) a routine using HPF and the Blas.

p	1	2	4	8
`LAPACK/dpotrf`	**43.5** s			
Native HPF	472.7 s	214.1 s	105.9 s	63.8 s
`SCALAPACK/pdpotrf`	83.1 s	47.9 s	27.6 s	19.2 s
HPF + Blas	48.6 s	34.2 s	25.3 s	18.4 s

2.2 2-D Fast Fourier Transform.

The fast Fourier transform (FFT) is one of the principal algorithmic tools in the field of scientific computing. Though the invention of the FFT algorithm can be traced back to Gauss, its rediscovery by Cooley and Tukey in their 1965 paper [8] is responsible for the algorithm's widespread use.

A 2-D FFT applies 1-D FFTs along each column of a two-dimensional array and then along each row. In the used *parallel* 2-D FFT algorithms

[1] HPF routines which do not use the EXTRINSIC mechanism will be called *native* HPF routines.

the two-dimensional array is distributed columnwise. This distribution allows the 1-D FFTs along the columns to proceed without communication. The two-dimensional array is then transposed before performing the 1-D FFTs along the rows (again without communication). The transpose intrinsic `A=transpose(A)` involves all-to-all communication leading to considerable communication overhead. Nevertheless, this approach is more efficient than an algorithm based on a static decomposition and parallel 1-D FFTs to perform the FFTs along the rows.

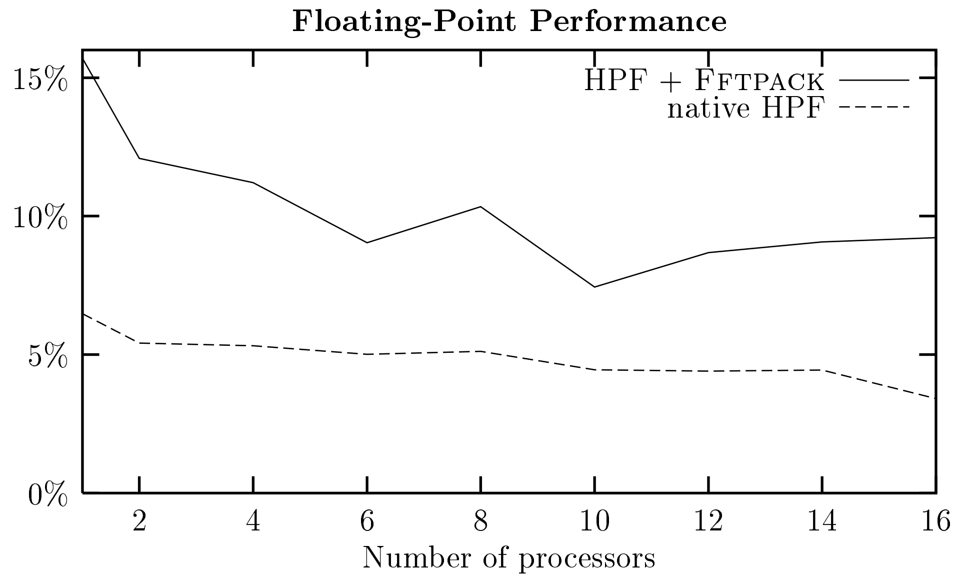

Figure 2.1: Relative floating-point performance of two different 2-D FFT codes, given in terms of the theoretical peak performance, running on an IBM SP-2. One of the HPF routines calls a one-dimensional Fortran 77 routine of the highly optimized sequential FFTPACK, and the other one is written entirely in HPF.

Fig. 2.1 shows a comparison of a native HPF implementation with an HPF code based on a 1-D FFTPACK[2]) routine. In the native HPF version a Cooley-Tukey radix-2 FFT algorithm (Van Loan [18]) is used for the 1-D FFTs.

[2]Despite its age (the first version dates back to 1973) FFTPACK is still one of the most efficient FFT packages (see Auer et al. [2]).

Chapter 3

Calling Parallel Libraries from HPF

As an alternative to the approach described in the previous section, extrinsic *parallel* library routines such as ScaLapack [5] and PLapack [13] can be invoked from HPF for distributed computations. In this case, the extrinsic library procedure itself performs communication as well as computation. HPF is used as a framework for conveniently distributing data and for organizing high-level parallelism. Calling distributed library routines from HPF mainly involves the issue of passing distribution information from HPF to the library routines.

Blackford et al. [6] have developed SLHPF, an interface from HPF to the ScaLapack [5]. Lorenzo et al. [16] developed another interface from HPF to the ScaLapack. The public domain HPF compilation system Adaptor (Brandes and Greco [7]) also contains a ScaLapack interface.

If the data distribution of the HPF code is not compatible with ScaLapack, the matrices have to be redistributed. If this is not necessary, the performance of this approach is very similar to ScaLapack itself.

It should be noted that these interfaces from HPF to parallel libraries should also be useful for converting existing *sequential* numerical libraries into HPF versions. In many cases it should suffice to replace calls to sequential Blas or Lapack routines in an existing Fortran 77 code by calls to PBlas or ScaLapack routines in the HPF version.

Chapter 4

Conclusions and Outlook

It has been shown that subroutines of existing numerical high performance libraries can be integrated into HPF code in a portable and efficient way, using HPF language features only. In particular, it has been demonstrated how to utilize sequential BLAS and FFT routines in HPF programs efficiently. This guarantees high local performance in HPF programs and yields significant performance improvements compared to native HPF implementations which do not take advantage of existing software packages and libraries.

Due to the encouraging performance of the new approach applied to Cholesky factorization and two-dimensional Fourier transformation the authors will continue to develop HPF routines which are based on highly tuned numerical libraries.

Bibliography

[1] 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 *User's Guide*, 2nd ed. SIAM Press, Philadelphia 1995.

[2] M. Auer, R. Benedik, F. Franchetti, H. Karner, P. Kristöfel, R. Schachinger, A. Slateff, C. W. Ueberhuber, *Performance Evaluation of FFT Routines—Machine Independent Serial Programs*, AURORA Tech. Report TR1999-05, Institute for Applied and Numerical Mathematics, Technical University of Vienna, 1999.

[3] J. Bilmes, K. Asanovic, C.-W. Chin, J. Demmel, *Optimizing Matrix Multiply using* PHIPAC*: a Portable, High-Performance, ANSI C Coding Methodology*, Proceedings of the International Conference on Supercomputing, ACM, Vienna, Austria, 1997, pp. 340–347.

[4] J. Bilmes, K. Asanovic, J. Demmel, D. Lam, C.-W. Chin, *Optimizing Matrix Multiply using* PHIPAC*: a Portable, High-Performance, ANSI C Coding Methodology*, Technical report, LAPACK Working Note 111, 1996.

[5] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley: SCALAPACK *Users' Guide*, SIAM Press, Philadelphia 1997.

[6] L. S. Blackford, J. J. Dongarra, C. A. Papadopoulos, and R. C. Whaley: *Installation Guide and Design of the HPF 1.1 Interface to* SCALAPACK*, SLHPF*, LAPACK Working Note 137, University of Tennessee 1998.

[7] T. Brandes and D. Greco: *Realization of an HPF Interface to* SCALAPACK *with Redistributions*. High-Performance Computing and Networking. International Conference and Exhibition, Springer-Verlag, Berlin Heidelberg New York Tokyo 1996, pp. 834–839.

[8] J. W. Cooley, J. W. Tukey, *An Algorithm for the Machine Calculation of Complex Fourier Series*, Math. Comp. 19 (1965), pp. 297–301.

[9] J. J. Dongarra, J. Du Croz, S. Hammarling, R. J. Hanson: *An Extended Set of* BLAS, ACM Trans. Math. Software 14 (1988), pp. 18–32.

[10] J. J. Dongarra, J. Du Croz, S. Hammarling, I. Duff: *A Set of Level 3* BLAS, ACM Trans. Math. Software 16 (1990), pp. 1–17.

[11] H. J. Ehold, W. N. Gansterer, D. F. Kvasnicka, C. W. Ueberhuber, *HPF and numerical libraries*, Parallel Computation, 4th International ACPC Conference, (P. Zinterhof, M. Vajtersic, A. Uhl, Eds.), Vol. 1557 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin Heidelberg, 1999, pp. 140–152.

[12] H. J. Ehold, W. N. Gansterer, and C. W. Ueberhuber: *HPF—State of the Art*, Technical Report AURORA TR1998-01, European Centre for Parallel Computing at Vienna, Vienna 1998.

[13] R. van de Geijn: *Using* PLAPACK*: Parallel Linear Algebra Package*, MIT Press 1997.

[14] High Performance Fortran Forum: *High Performance Fortran Language Specification Version 2.0*, 1997. URL: `www.crpc.rice.edu/HPFF/hpf2/` or
`www.vcpc.unvie.ac.at/information/mirror/HPFF/hpf2/`.

[15] C. L. Lawson, R. J. Hanson, D. Kincaid, F. T. Krogh: BLAS *for Fortran Usage*, ACM Trans. Math. Software 5 (1979), pp. 63–74.

[16] P. A. R. Lorenzo, A. Müller, Y. Murakami, and B. J. N. Wylie: *HPF Interfacing to* SCALAPACK, Third International Workshop PARA '96, Springer-Verlag, Berlin Heidelberg New York Tokyo 1996, pp. 457–466.

[17] C. W. Ueberhuber: *Numerical Computation*, Springer-Verlag, Berlin Heidelberg New York Tokyo 1997.

[18] C. F. Van Loan, *Computational Frameworks for the Fast Fourier Transform*, SIAM Press, Philadelphia, 1992.

[19] R. C. Whaley, J. J. Dongarra, *Automatically Tuned Linear Algebra Software*, Technical Report, LAPACK Working Note 131, 1997.