Efficient HPF Programs

Harald J. Ehold[1]
Wilfried N. Gansterer[2]
Dieter F. Kvasnicka[3]
Christoph W. Ueberhuber[2]

[1] VCPC, European Centre for Parallel Computing at Vienna
E-Mail: ehold@vcpc.univie.ac.at

[2] Institute for Applied and Numerical Mathematics
Vienna University of Technology
E-Mail: ganst@aurora.tuwien.ac.at, christof@uranus.tuwien.ac.at

[3] Institute for Physical and Theoretical Chemistry
Vienna University of Technology
E-Mail: kvasnicka@tuwien.ac.at

June 1999

AURORA TR1999-09

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

Abstract

HPF was originally created to simplify high-level programming of parallel computers. The inventors of HPF strove for an easy-to-use language which was intended to enable portability *and* efficiency. However, up until now the desired efficiency has not been reached. On the contrary, HPF programs are notorious for their poor performance.

This paper provides a rehabilitation of HPF. It is demonstrated how currently available HPF constructs can be utilized to solve sizeable numerical problems in a highly efficient manner. Using the techniques published in this paper, the empirical efficiency, i.e. the ratio between the empirical floating-point performance and the theoretical peak performance, can be driven up to 60 % and more. Even on message passing machines with slow communication networks, such as PC clusters (Beowulf clusters) using 100 Mbit/s Ethernet interconnection, empirical efficiency results at a very satisfactory level can be obtained.

1 Introduction

High Performance Fortran (HPF [6]) is a programming language which provides very high-level support for the development of parallel programs. It has been designed primarily for regular, data-parallel applications. One of the central goals of HPF is to combine *high performance* with *portability* across a wide range of (distributed memory) parallel computers.

HPF is a conceptually very elegant and attractive approach. In particular, it provides very convenient ways for specifying data distributions and for expressing data parallelism. Development of parallel code using HPF is much easier and requires less effort than message passing programming, for example, using MPI. Nevertheless, so far HPF has not become widely accepted by the users for several reasons (see also Ehold et al. [5]).

- For a very long time there was no full language support by commercial compilers. Only recently mature compilers which support the full standard have become available. Unfortunately, several companies have suspended their activities of HPF compiler development because the market for HPF compilers has been assessed too small.
- The few HPF compilers which are available were not able to deliver acceptable performance. In fact, in many cases the performance of HPF codes was so inferior to explicit message-passing programming, that HPF could not be considered a competitive alternative despite its obvious advantages in terms of code development, debugging, flexibility, maintenance, and portability.

In this paper a concept for achieving high performance with HPF is introduced which opens up new perspectives. It is based on utilizing existing numerical software in HPF programs.

Previous work has concentrated on interfacing HPF to *parallel* software packages. Blackford et al. [2] have developed an interface called SLHPF from HPF to the ScaLapack package. Lorenzo et al. [7] developed another interface from HPF to ScaLapack. The public domain HPF compilation system Adaptor (Brandes and Greco [3]) also contains an interface to ScaLapack.

Nevertheless, in many cases it turns out to be preferable to utilize *sequential* library routines on each processor, as illustrated in this paper.

2 HPF and Numerical Libraries

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.

Much effort has been spent on developing highly efficient implementations of the Blas and on numerical software packages for dense linear algebra which use the Blas as building blocks. Important examples are Lapack, which is the standard sequential package for dense or banded linear algebra methods, and parallel packages such as ScaLapack or PLapack.

Highly optimized Blas implementations are available for most target systems. If no efficient Blas implementation is provided for a certain computer system it can be generated by the user. This is made possible by code generation tools which have been developed recently (see Bilmes et al. [1], Whaley, Dongarra [9]). These tools automatically find the best choice of hardware dependent parameters for an efficient implementation of the BLAS.

Clearly, HPF users should be able to benefit from such resources. Thus, the existing sequential software has to be utilized in order to optimize local performance of parallel programs which in turn can improve overall parallel performance provided communication is fast enough. Moreover, the use of widely available optimized routines like the BLAS for local computations ensures *performance portability.*

This paper describes how routines from sequential numerical packages (*extrinsic* routines in HPF terminology) can be integrated into an HPF program for *local* computations. The HPF compiler remains responsible for handling and organizing the parallelism supported by the directives of the user.

The main ideas and basic concepts presented in this paper are applicable to many data-parallel mathematical operations (mostly, but not exclusively from linear algebra).

HPF Features Used. The basic facility provided by HPF for integrating procedures from other programming languages or models is the EXTRINSIC mechanism.

The concepts which have been developed are based on this mechanism. In addition, they rely on features from the HPF standard [6] and on two specific features which are part of the HPF 2.0 Approved Extensions. In particular, the required HPF features are:

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

The extrinsic kind HPF_LOCAL refers to procedures implemented in the HPF language, but in the *local* programming model. Each processor executes this code on its local data. Specifically, it provides a means to call the sequential BLAS routines for local computations.

3 The Problem

To demonstrate the usefulness of our approach we implemented a matrix-matrix multiplication, a Cholesky factorization, and a 2D FFT (see also Ehold et al. [4]). In order to save space only results pertaining to matrix-matrix multiplication are presented in this paper.

4 The Algorithm

An HPF routine called `par_dgemm` (*parallel general matrix-matrix multiplication*) has been developed, which computes the product $C \in \mathbb{R}^{m \times n}$ of two matrices $A \in \mathbb{R}^{m \times l}$, $B \in \mathbb{R}^{l \times n}$. All matrices involved can be distributed arbitrarily. Internally, this operation is split up into local operations on subblocks, each of which is performed by calling the *sequential* general matrix-matrix multiplication routine `BLAS/dgemm`.

In the general case, the multiplication of two distributed matrices involves *non-local* computations. However, choosing the proper loop order and doing some clever replication of small submatrices allows to localize *all* of the computation involved.

For performance reasons, a *blocked* version of matrix-matrix multiplication (Ueberhuber [8]) was implemented. Two levels of wrapper routines are involved. The programmer calls the HPF routine `par_dgemm`, which takes the matrices A, B as input and returns the product matrix C.

Inside this routine the HPF_LOCAL routine `local_dgemm` is called in a loop. In addition to the blocks involved `local_dgemm` also takes their size (the block size of the algorithm) as an argument and performs the local outer products of a block column of A and a block row of B by calling the routine `BLAS/dgemm`.

In `par_dgemm`, work arrays for a block column of A and for a block row of B are aligned properly with C (*partial replication*), and then the corresponding subarrays of A and B are copied there. This copying operation in the outermost loop adjusts the distribution of the currently required parts of A and B to that of C. It is the only place where inter-processor communication occurs. The main advantage is that the routine is made fully independent of the prior distributions of A and B and at the same time all the communication is restricted to the outermost loop.

5 The Machines

Numerical experiments to verify the usefulness of the new techniques have been carried out on several multiprocessor machines (Meiko, IBM, etc.). The results reported in the following sections were obtained on an IBM SP2 and on a PC cluster (Beowulf cluster).

IBM SP2. The SP2 used in the experiments is a system equipped with 37 processors (POWER2 architecture RS/6000), of which we were able to use 16 at a time for our experiments. Communication is provided via the high performance switch. The theoretical peak performance using p processors is $p \times 267$ Mflop/s, $p = 1, 2, \ldots, 37$. HPF support is provided via Portland Group's HPF compiler *pghpf* version 2.4-4. The local BLAS routines used were part of the highly optimized ESSL.

Beowulf Cluster. The PC cluster used in the experiments is a computer system consisting of one front-end PC and five dual processor PCs equipped with 350 MHz Pentium II processors. Communication is provided via fast Ethernet (100 Mbit/s). The theoretical peak performance using p processors of the cluster is $p \times 350$ Mflop/s, $p = 1, 2, \ldots, 10$. HPF support is provided via Portland Group's HPF compiler *pghpf* version 3.0-3. The local BLAS routines used have been developed for the ASCI Red project (see `http://www.cs.utk.edu/~ghenry/distrib/`).

6 The Results

IBM SP2. Fig. 1 illustrates the floating-point performance on 16 processors of an IBM SP2 of the routines `par_dgemm` and `PBLAS/pdgemm` called from an HPF program using the SLHPF interface [2] for two different data distributions: "cyclic(1)" in both dimensions ("*d1*"), "block" in the first and "cyclic(1)" in the second dimension ("*d2*"). The intrinsic function MATMUL is not shown because of its unacceptably low performance.

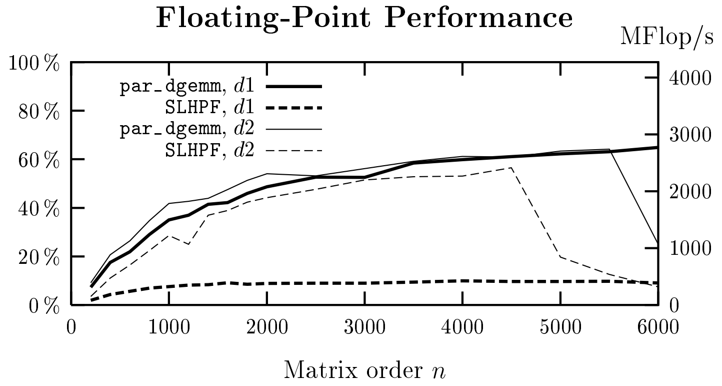

Figure 1: Matrix multiplication on 16 processors of an IBM SP2.

Fig. 2 gives the efficiency on 2 to 16 processors of (i) the newly developed routine `par_dgemm`, (ii) `PBLAS/pdgemm` called using the SLHPF interface, and (iii) MATMUL on an IBM SP2. MATMUL shows a very poor performance. As a matter of fact, MATMUL running on 16 processors of an IBM SP2 achieves a lower floating-point performance than `ESSL/dgemm` running on one processor.

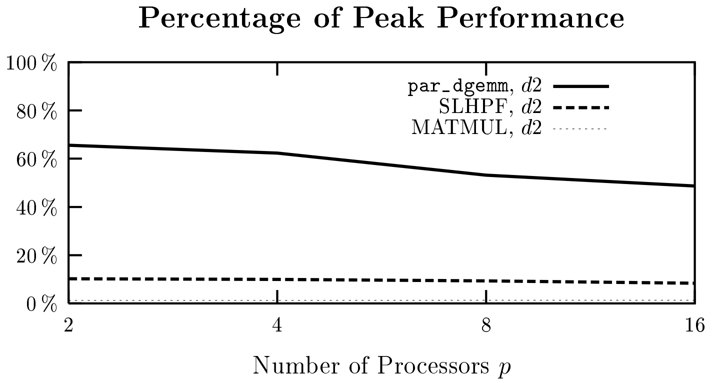

Figure 2: Matrix multiplication for $n = 2000$ on p processors of an IBM SP2.

Beowulf Cluster. Fig. 3 compares the routines `par_dgemm` and `PBLAS/pdgemm`. A "cyclic(200)" data distribution ("*d3*") in both dimensions was used for both routines. This distribution results in best performance with `PBLAS/dgemm` and is also a very favorable distribution for `par_dgemm`. Fig. 3 shows that HPF programs can be as efficient as subroutines from highly optimized parallel numerical libraries.

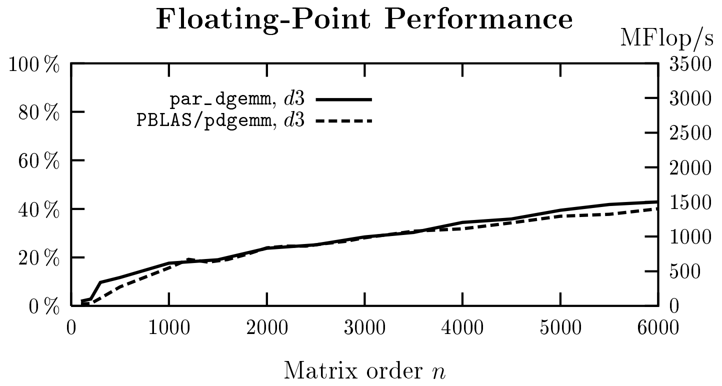

Figure 3: Matrix multiplication on 10 processors of a Beowulf Cluster. The HPF routine is as efficient as the highly optimized PBLAS (*Parallel* BLAS) routine `pdgemm`.

7 Conclusion

In this paper, HPF is shown to support construction of codes portable across important parallel architectures, while achieving a very satisfactory percentage of theoretical peak performance.

The price/performance ratio of 11 US$ per Mflop/s, achieved on a small PC cluster, is better than the price/performance achievement which was awarded the Gordon Bell Prize in 1998.

8 Acknowledgements

We want to express our gratitude to John Merlin for many helpful discussions and to the Austrian Science Fund (FWF) for financial support.

References

[1] J. Bilmes, K. Asanovic, C. W. Chin, J. Demmel: *Optimizing Matrix Multiply using* PHIPAC*: a Portable, High-Performance, ANSI C Coding Methodology.* In *Proceedings of the International Conference on Supercomputing*, ACM, Vienna, Austria, 1997, pp. 340–347. Also LAPACK Working Note 111.

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

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

[4] H. J. Ehold, W. N. Gansterer, D. F. Kvasnicka, C. W. Ueberhuber: *HPF and Numerical Libraries.* In *Parallel Computation. Proceedings of the 4th International ACPC Conference* (P. Zinterhof, M. Vajteršic, A. Uhl, eds.), Springer-Verlag, Berlin Heidelberg New York Tokyo, Salzburg, Austria, 1999, vol. 1557 of *Lecture Notes in Computer Science*, pp. 140–152.

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

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

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

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

[9] R. C. Whaley, J. J. Dongarra: *Automatically Tuned Linear Algebra Software*. LAPACK Working Note 131, University of Tennessee, Knoxville TN, 1997.