The Dangers of Heterogeneous Network Computing: Heterogeneous Networks Considered Harmful

Jim Demmel[*] Jack Dongarra[†] Sven Hammarling[‡] Susan Ostrouchov[§]
Ken Stanley[¶]

Abstract

This report addresses the issue of writing reliable numerical software for networks of heterogeneous computers. Much software has been written for distributed memory parallel computers and in principal such software could readily be ported to networks of machines, such as a collection of workstations connected by Ethernet, but if such a network is not homogeneous there are special challenges that need to be addressed. The symptoms can range from erroneous results returned without warning to deadlock.

Some of the problems are straightforward to solve, but for others the solutions are not so obvious and indeed in some cases, such as the method of bisection which we shall discuss in the report, we have not yet decided upon a satisfactory solution that does not incur an unacceptable overhead. Making software robust on heterogeneous systems often requires additional communication.

In this report we describe and illustrate the problems and, where possible, suggest solutions so that others may be aware of the potential pitfalls and either avoid them or, if that is not possible, ensure that their software is not used on heterogeneous networks.

1 Introduction

There are special challenges associated with writing reliable numerical software on networks containing heterogeneous processors, that is processors which may do floating point arithmetic differently. This includes not just machines with completely different floating point formats and semantics (e.g. Cray machines running 'Cray arithmetic' versus workstations running IEEE standard floating point arithmetic), but even supposedly identical machines running with different compilers, or even just different compiler options or runtime environments.

[*]The University of California at Berkeley
[†]The University of Tennessee at Knoxville and Oak Ridge National Laboratory
[‡]The University of Tennessee at Knoxville and The Numerical Algorithms Group Ltd, Oxford
[§]The University of Tennessee at Knoxville
[¶]The University of California at Berkeley

The basic problem occurs when making $d$ata dependent branches on different processors. The flow of an algorithm is usually data dependent and so slight variations in the data may lead to different processors executing completely different sections of code.

This report represents the experience of two groups developing numerical software for distributed memory message-passing systems, both of whom became aware, at about the same time, that the software being developed may not perform correctly on heterogeneous systems. We briefly describe the work of these two groups in Section 2.

In Sections 3, 4 and 6 we look at three areas that require attention in developing software for heterogeneous networks: machine parameters, where we discuss what the values of machine parameters, such as machine precision should be; checking global arguments and communicating floating point values; and algorithmic integrity, that is, how can we ensure that algorithms perform correctly in a heterogeneous setting. The particular case of communicating floating point values on IEEE machines is briefly discussed in Section 5.

2 Motivation and Background

The challenges of heterogeneous computing discussed in this report came to light during the development of ScaLAPACK (Choi, Demmel, Dhillon, Dongarra, Ostrouchov, Petitet, Stanley, Walker and Whaley, 1995) and the NAG Numerical PVM Library (NAG, 1995).

ScaLAPACK is a library of high performance linear algebra routines for distributed memory MIMD machines. It is a continuation of the LAPACK project, which has designed and produced an efficient linear algebra library for workstations, vector supercomputers and shared memory parallel computers (Anderson, Bai, Bischof, Demmel, Dongarra, Du Croz, Greenbaum, Hammarling, McKenney, Ostrouchov and Sorensen, 1995). Both libraries contain routines for the solution of systems of linear equations, linear least squares problems and eigenvalue problems. The goals of the LAPACK project, which continue into the ScaLAPACK project, are efficiency so that the computationally intensive routines execute as fast as possible; scalability as the problem size and number of processors grow; reliability, including the return of error bounds; portability across machines; flexibility so that users may construct new routines from well designed components; and ease of use. Towards this last goal the ScaLAPACK software has been designed to look as much like the LAPACK software as possible.

Many of these goals have been attained by developing and promoting standards, especially specifications for basic computational and communication routines. Thus LAPACK relies on the BLAS (Lawson, Hanson, Kincaid and Krogh, 1979; Dongarra, Du Croz, Hammarling and Hanson, 1988; Dongarra, Du Croz, Duff and Hammarling, 1990), particularly the Level 2 and 3 BLAS for computational efficiency, and ScaLAPACK relies upon the BLACS (Dongarra and Whaley, 1995) for efficiency of communication and uses a set of parallel BLAS, the PBLAS (Choi, Dongarra, Ostrouchov, Petitet, Walker and Whaley, 1995), which themselves call the BLAS and the BLACS. LAPACK and ScaLAPACK will run on any machines for which the BLAS and the BLACS are available. A PVM ((Geist, Beguelin, Dongarra, Jiang, Manchek and Sunderam, 1994)) version of the BLACS has been available for some time and the portability of the BLACS has recently been further increased by the development of a version that uses MPI (Message Passing Interface Forum, 1994; Gropp, Lusk and Skjellum, 1994; Snir, Otto, Huss-Lederman, Walker and Dongarra, 1996).

The NAG Numerical PVM Library is a small library of numerical routines also for distributed memory MIMD machines that contains routines for dense and sparse linear algebra, including some ScaLAPACK routines, quadrature, optimization, random number generation and various utility routines for operations such as data distribution and error handling. This Library uses much the same model for distributed memory computing as ScaLAPACK and was developed with the same goals in mind (Hammarling, 1994).

Both ScaLAPACK and the NAG Numerical PVM Library were developed with heterogeneous environments in mind, as well as standard homogeneous machines. But, in both cases during development is was realized that we could not guarantee the safe behaviour of all our routines in a heterogeneous environment and so, for the time being, both libraries are only fully supported on homogeneous machines, although they are tested on networks of IEEE machines and are believed to work correctly in such environments. It is, though, intended to be able to fully support other heterogeneous environments in the near future.

3 Machine Parameters

A simple example of where an algorithm might not work correctly is an iteration where the stopping criterion depends on the value of the machine precision. If the precision varies from processor to processor, different processors may have significantly different stopping criteria. In particular, the stopping criterion used by the most accurate processor may never be satisfied if it depends on data computed less accurately by other processors.

Many such problems can be eliminated by using the *l*argest machine precision among all participating processors. In LAPACK routine `DLAMCH` returns the (double precision) machine precision (as well as other machine parameters). In ScaLAPACK this is replaced by `PDLAMCH` which returns the largest value over all the processors, replacing the uniprocessor value returned by `DLAMCH`. Similarly, one should use the smallest overflow threshold and largest underflow threshold over the processors being used. In a non-homogeneous environment the ScaLAPACK routine `PDLAMCH` runs the LAPACK routine `DLAMCH` on each processor and computes the relevant maximum or minimum value. We refer to these machine parameters as the *m*ultiprocessor machine parameters.

It should be noted that if the code contains communication between processors within an iteration, it will not complete if one processor converges before the others. In a heterogeneous environment, the only way to guarantee termination is to have one processor make the convergence decision and broadcast that decision. This is a strategy we shall see again in later sections.

The NAG Numerical PVM Library will use an equivalent mechanism.

4 Global Arguments and Floating Point Values

The high level routines in both ScaLAPACK and the NAG Numerical PVM Library check arguments supplied by users for their validity in order to aid users and provide as much reliability as possibility. In particular, global arguments are checked. When these are floating point values they may of course, for the reasons already discussed, have different values on different processors.

This raises the question of how, and even whether, such arguments should be checked, and what action should be taken when a failure occurs. If we compare the values, they may not be the same on different processors, so we need to allow a tolerance based upon the multiprocessor machine precision. Alternatively, we can check a global argument on just one processor and then, if the value is valid, broadcast that value to all the other processors. Of course this alternative approach has extra overhead, but it may be the most reliable solution, since the library routine has algorithmic control, and puts slightly less burden on the user.

Similar issues occur whenever we communicate a floating point value from one processor to another. Unless we have special knowledge, and one such case will be discussed in the next section, we should not assume that the target processor will have exactly the same value as the sending processor and we must write the code accordingly.

5 Communicating Floating Point Values on IEEE Machines

The IEEE standard for binary floating point arithmetic (IEEE, 1985) specifies how machines conforming to the standard should represent floating point values. We refer to machines conforming to this standard as *I*EEE machines[1]. Thus, when we communicate floating point numbers between IEEE machines we might hope that each processor has the same value. This is a reasonable hope and will often be realized.

For example, XDR (External Data Representation, SunSoft (1993)) uses the IEEE representation for floating point numbers and so a message passing system that uses XDR will communicate floating point numbers without change[2]. PVM is an example of a system that uses XDR. MPI suggests the use of XDR, but does not mandate its use (Snir et al., 1996, Section 2.3.3), so presumably we cannot assume that floating point numbers will be communicated without change on IEEE machines when using MPI unless we have additional information about the implementation.

[1] It should be noted that there is also a radix independent standard (IEEE, 1987).

[2] It should be noted that it is not clear whether or not this can be assumed for denormalized numbers.

6 Algorithmic Integrity

The suggestions we have made so far certainly do not solve all the problems. We are still left with many of those problems associated with the major concern of varying floating point representations and arithmetic operations between different processors, different compilers and different compiler options. We illustrate the difficulties with just two examples from ScaLAPACK, the second example giving rather more severe difficulties than the first.

Consider the LAPACK routine DLARFG which computes an elementary reflector (Householder transformation matrix) H such that

$$Hx = \beta e_1,$$

where β is a scalar, x is an n element vector and e_1 is the first column of the unit matrix. H is represented in the form

$$H = I - \tau v v^T,$$

where τ is a scalar and v is an n element vector. Since H is orthogonal we see that

$$|\beta| = \|x\|_2.$$

If $|\beta|$ is very small (sub-normal or close to being sub-normal), DLARFG scales x and recomputes $\|x\|_2$. This computation is at the heart of the LAPACK QR, and other, factorizations (see for example (Golub and Van Loan, 1989)).

In the case of the equivalent ScaLAPACK routine PDLARFG, x will usually be distributed over several processors, each of which participates in the computation of $\|x\|_2$ and, if necessary, scales its portion of the vector x and recomputes $\|x\|_2$. From the previous discussion we can see that we clearly need to take care here, or else, in close cases, some processors may attempt to recompute $\|x\|_2$, while others do not, leading to completely erroneous results, or even deadlock.

There are many other routines in the LAPACK and ScaLAPACK libraries where scaling takes place, either to avoid problems associated with overflow and underflow, or to improve numerical stability, as in the equilibration routines for linear equations.

One way to ensure correct computation is to put one processor in control of whether or not scaling should take place, and for that processor to request the other processors all either to scale, or not to scale. Having a $c$ontrolling processor is a common way to solve such problems on heterogeneous networks.

As a somewhat harder problem consider the method of bisection for finding the eigenvalues of symmetric matrices performed by the ScaLAPACK routine PDSYEVX. In this algorithm, the real axis is broken into disjoint intervals to be searched by different processors for the eigenvalues contained in each interval. Disjoint intervals are searched in parallel. The algorithm depends on a function, say `count(a,b)`, that counts the number of eigenvalues in the half open interval [`a`, `b`). Using `count`, intervals can be subdivided into smaller intervals containing eigenvalues until the intervals are narrow enough to declare the eigenvalues they contain as being found. The problem here is that two processors may not agree on the boundary between their intervals. This could result in multiple copies of eigenvalues if intervals overlap, or missing eigenvalues if there are gaps between intervals. Furthermore, the `count` function may count differently on different processors, so an interval [`a`, `b`) may be considered to contain 1 eigenvalue by processor A, but 0 eigenvalues by processor B, which has been given the interval by processor A during load balancing. This can happen even if processors A and B are identical in hardware terms, but if the compilers on each one generate slightly different code sequences for `count`. In this example we have not yet decided what to do about all these problems, so we currently only guarantee correctness of `PDSYEVX` for networks of processors with identical floating point formats (slightly different floating point operations turn out to be acceptable). See (Demmel, Dhillon and Ren, 1995) for further discussion. Assigning the work by index rather than by range and sorting all the eigenvalues at the end may give the desired result with modest overhead. Of course, if floating point formats differ across processors, sorting is a problem in itself. This requires further investigation.

Similar problems, although not usually with quite such potentially severe consequences, could occur in the adaptive quadrature routines of the NAG Numerical PVM Library, where again different processors are assigned different intervals on which to perform numerical integration.

Redundant work on different processors which is intended to result in identical results, may not do so in a heterogeneous environment. One approach for parallelizing the symmetric eigenproblem is to perform a tridiagonal QR iteration redundantly on all processors and accumulate the resulting Givens rotations in parallel. This results in $O(n^2)$ redundant work, $O(n^3)$ parallel work, and requires no communication. Since QR is not forward stable, slight differences in the underlying arithmetic can lead to completely different rotations and completely incorrect results. This can be solved by having one processor compute all the reflectors and broadcast them to the other processors, but the communication cost is substantial: $O(n^2)$.

7 Closing Remarks

We have tried to illustrate the potential difficulties concerned with floating point computations on heterogeneous networks. Some of these difficulties are straightforward to address, while others require considerably more thought. All of them require some additional level of defensive programming to ensure the usual standards of reliability that users have come to expect from packages such as LAPACK and libraries such as the NAG Library.

We have suggested reasonably straightforward solutions to the problems associated with floating point machine parameters and global values, and have suggested the use of a controlling processor to solve some of the difficulties of algorithmic integrity. This can probably be used to solve most of these problems, but in some cases at the expense of considerable additional overhead.

A topic that we have not discussed is that of the additional testing necessary to give confidence in heterogeneous environments. The testing strategies that are needed are similar to those already employed in reputable software packages such as LAPACK and the NAG Library, but it may be very hard to produce actual test examples that would detect incorrect implementations of the algorithms because, as we have seen, the failures are likely to be very sensitive to the computing environment, and in addition may be non-deterministic.

8 Acknowledgments

We thank Inderjit Dhillon and Huan Ren of the University of California at Berkeley for valuable input to this paper, and our other ScaLAPACK and NAG colleagues for a number of useful discussions on heterogeneous computing.

References

Anderson, E., Bai, Z., Bischof, C. H., Demmel, J., Dongarra, J. J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S. and Sorensen, D. C. (1995). *LAPACK Users' Guide*, 2nd edn, SIAM, Philadelphia, PA, USA.

Choi, J., Demmel, J., Dhillon, I., Dongarra, J. J., Ostrouchov, S., Petitet, A. P., Stanley, K., Walker, D. W. and Whaley, R. C. (1995). ScaLAPACK: a portable linear algebra library for distributed memory computers – design issues and performance. LAPACK Working Note No.95, *Technical Report CS-95-283*, Department of Computer Science, University of Tennessee, 107 Ayres Hall, Knoxville, TN 37996-1301, USA.

Choi, J., Dongarra, J. J., Ostrouchov, S., Petitet, A. P., Walker, D. W. and Whaley, R. C. (1995). A proposal for a set of Parallel Basic Linear Algebra Subprograms. LAPACK Working Note No.100, *Technical Report CS-95-292*, Department of Computer Science, University of Tennessee, 107 Ayres Hall, Knoxville, TN 37996-1301, USA.

Demmel, J., Dhillon, I. and Ren, H. (1995). On the correctness of parallel bisection in floating point, *ETNA* **3**: 116–149.

Dongarra, J. J., Du Croz, J., Duff, I. S. and Hammarling, S. (1990). A set of Level 3 Basic Linear Algebra Subprograms, *ACM Trans. Math. Software* **16**: 1–28. (Algorithm 679).

Dongarra, J. J., Du Croz, J., Hammarling, S. and Hanson, R. J. (1988). An extended set of FORTRAN Basic Linear Algebra Subprograms, *ACM Trans. Math. Software* **14**: 1–32. (Algorithm 656).

Dongarra, J. J. and Whaley, R. C. (1995). A users' guide to the BLACS v1.0. LAPACK Working Note No.94, *Technical Report CS-95-281*, Department of Computer Science, University of Tennessee, 107 Ayres Hall, Knoxville, TN 37996-1301, USA.

Geist, A., Beguelin, A., Dongarra, J. J., Jiang, W., Manchek, R. and Sunderam, V. (1994). *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, MA, USA.

Golub, G. H. and Van Loan, C. F. (1989). *Matrix Computations*, 2nd edn, The Johns Hopkins University Press, Sudbury, MA, USA.

Gropp, W., Lusk, E. and Skjellum, A. (1994). *Using MPI: Portable Programming with the Message-Passing Interface*, MIT Press, Cambridge, MA, USA.

Hammarling, S. (1994). Parallel library work at NAG, *in* J. J. Dongarra and B. Tourancheau (eds), *Environments and Tools for Parallel Scientific Computing*, SIAM, Philadelphia, PA, USA, pp. 172–182. (Proceedings of the Second Workshop, Townsend, TN, USA).

IEEE (1985). *ANSI/IEEE Standard for Binary Floating Point Arithmetic: Std 754-1985*, IEEE Press, New York, NY, USA.

IEEE (1987). *ANSI/IEEE Standard for Radix Independent Floating Point Arithmetic: Std 854-1987*, IEEE Press, New York, NY, USA.

Lawson, C. L., Hanson, R. J., Kincaid, D. and Krogh, F. T. (1979). Basic Linear Algebra Subprograms for FORTRAN usage, *ACM Trans. Math. Software* **5**: 308–323. (Algorithm 539).

Message Passing Interface Forum (1994). MPI: A Message-Passing Interface standard, *Int. J. Supercomputer Applics.* **8**(3/4).

NAG (1995). *NAG Numerical PVM Library Manual, Release 1*, The Numerical Algorithms Group Ltd, Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, UK.

Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W. and Dongarra, J. J. (1996). *MPI: The Complete Reference*, MIT Press, Cambridge, MA, USA.

SunSoft (1993). *The XDR Protocol Specification. Appendix A of "Network Interfaces Programmer's Guide"*, SunSoft.