Optimisation of ScaLAPACK on the Cray X1

Adrian P. Tate, *CSAR, University of Manchester*
adrian.tate@man.ac.uk

ABSTRACT: ***Scalability of numerical parallel library routines is naturally inhibited by communications overhead, since the sharing of computation amongst processors becomes outweighed by the cost of communications between those processors. Hence, a fixed problem size will naturally reach saturation point in terms of scalability. In the context of ScaLAPACK, this saturation point is often too low for use in capability codes. The University of Manchester and Cray Inc. have begun a collaboration that will endeavour to address this issue by way of a complete overhaul of the existing communications layer to ScaLAPACK, thus decreasing the time spent in inter-process communications and increasing the scalability of the library routines. Cray's new X1 architecture is designed to make remote memory referencing much quicker, and specific features allow very quick message passing via shmem or Co-array Fortran. With these features in mind, a suite of new communication subroutines is being developed to produce highly scalable parallel numerical library routines for the Cray X1.***

1. Introduction

Cray Inc. will include scalapack in a future release of the X1 scientific library. The university of Manchester will be responsible for the communications optimisation of specific routines within that distribution. The University of Manchester has been actively involved in the optimisation of parallel numerical library routines in the context of the UK academic community; see Tate & Briddon [1]. Several academic users of the national CSAR service have reported scalapack dependence as an inhibiting factor to the production of capability codes, since any speed-up obtained in the application is not always mirrored in the scalapack subroutine, which then becomes a bottleneck. Most Scalapack subroutines perform the majority of the involved computation in a highly tuned blas routine; the lack of scaling is a failure of the Pblas to ensure communication costs are low. Actual communications are carried out using BLACS (Basic linear Algebra Communications Subprograms) which use MPI for message passing. Since the blas are very well tuned on supercomputer systems, and since all the numerically intensive aspects of a given scalapack routine seem to be sensitive to compiler optimisation, these aspects were ignored in the work, which concentrated on reducing communications overhead directly. This was performed in a number of ways

- Replace 2 sided blacs calls with shmem
- Reduce synchronization costs
- Re-order operations and mathematics to allow more efficient communications patterns.

Particular attention was paid to redistribution routines, since these constitute a performance bottleneck due to the block-cyclic distribution. If a matrix A is distributed over a 2-d Blacs grid, then for a numerically intensive area of the code this distribution may be extremely helpful since the array elements separated by a known stride may all be held in the local processors memory (with a sensible choice in blocking factor). In the instances when the contiguous array elements are required however, such as the copying of a subarray of A into a subarray of another 2-d block cyclic array, then consecutive elements are (by implication) spread amongst all other processors and the operation will involve a maximum of inter-process communication. Routine PDGEMR2D, heavily used in QUB's R-matrix Propagator code [2] was replaced with a 1-sided communications version that used shmem and co-array Fortran for data transfer and which minimised communications and synchronization overheads. The resultant routine

operates at 10-60% overhead of the Scalapack original, results for which will be published shortly.
This work exposed the need for a more tuned scalapack library on high-performance systems and offered the opportunity to look more closely at scalapack generally. Though the library is extremely comprehensive, user take-up has been lower than the community expected. Unquestionably, a factor in this is the strict adherence to 2-d block cyclic distribution that a user code must exhibit. The distribution must be in place before a call to any scalapack routine, and usually means that an application is subjected to the same distribution at all other points in the code (i.e. not just when calling scalapack). This discourages new users from experimenting with scalapack, most of its users have a problem that is permanently configured in a scalapack friendly manner. An alternative package, PLAPACK [3] offers a less rigid distribution that can suit the nature of the application as well as some noteable performance increases, but the project is still junior and does not yet offer the level of functionality to allow most users to switch from scalapack. Hence, there is a very real need for Cray to offer a tuned scalapack, and some scope for future improvement and enhancements in areas that will improve the useability and accessibility of the library.

2. Optimisation of Scalapack

Co-array Fortran is an excellent programming paradigm for use on the X1, since the system has been implemented such that remote data will not be cached by a local processor, but will directly enter the vector registers. Further, the X1 compiler will perform pre-fetching on Co-arrays within the program. The reasons that lead to Co-array being used in scalapack optimisation were quite separate; programming difficulties inherent in a block cyclically distributed mode are made simpler using co-array fortran. Consider for example, the copying of one block of a block-cyclically distributed matrix A. Since a block is never contiguous in memory (unless the block length equals the matrix dimension) this involves a series of calls to MPI routines to transfer each column at a time to the remote processor (use of an MPI derived type can help), but in co-array fortran this operation is straightforward:

```
A(li:li+MB,lj:lj+NB) = A(ri:ri+MB,ri:gi+NB) [dest]
```

Where li,lj = local i, local j
Ri,rj = remote I, remote j
MB, NB = block sizes
dest = remote process number

The extremely simple syntax allows the user of a 2-d block cyclic context to vastly decrease the complexity of remote references, and it is for this reason, coupled with the obvious performance advantages, that co-array Fortran will be a valuable tool in the development of a tuned scalapack.

Co-array Fortran will be used in conjunction with shmem, since there is some difference in each's performance for certain data transfer sizes (see results section). Previous work has included both co-array and shmem version of redistribution routines, Cray tuned scalapack will include a switch that selects the most appropriate transfer method depending on block size.

In the short term, all communication procedures will be replaced to use co-array fortran and shmem as a data transfer method only, i.e. arrays within the scalapck and pblas subprograms will not be co-arrays but the internal arrays will be. This can lead to some difficulty; if an actual argument is a standard array then the associated dummy argument within a procedure cannot be either a co-array or a symmetrically allocated array. To avoid expensive copying into a co-array or symmetrically allocated array, two work-around techniques are useful. For co-arrays, a derived type can be allocated internally which has only one member, being a co-array pointer. This pointer can then be assigned to the passed array.

```
subroutine co_pass(A)
type cop
  real,pointer,dimension(:,:) :: co
end type cop
  type (cop)  :: rbuf[*]
REAL,target  :: A(ni,mi)

 RBUF%co => A(1:mi,1:ni)
```

The need to use symmetrically allocated data in shmem programs can be avoided altogether by using a remote memory address and allocating local pointer to the remote memory address of the array in question.

```
Subroutine nonsymtrans(A,m,n,iam,dest)
       Real :: A(m,n), ACOPY(m,n)
       Pointer(cptr,ACOPY)
       Integer*8 :: flag
       Integer   :: iam,dest
       DATA flag /0_8/
       SAVE flag
       If(iam==0)then
```

```
     Call shmem_wait(flag,0)
     cptr = flag
     flag = 0
     call shmem_fence()
     call shmem_get(A,ACOPY,m*n,dest)
  else
     call shmem_put8(flag,loc(A),1,dest)
  endif
  end subroutine
```

This method removes the need for expensive copying, but is also more efficient anyway since symmetric allocation necessarily involves a global barrier which can be expensive.

All BLACS communication procedures will be replaced with more efficient replacements. Since scalapack routines all depend on a core group of communication procedures, this work will have a significant effect on the library as a whole. In the medium term, attention will move from the BLACS library to the Pblas and scalapack libraries. Communications patterns for one-sided calls do not necessarily match those written for 2-sided MPI blacs, so some restructuring at the PBLAS level will be necessary. Restructuring can often allow different operations to be performed with less synchronisation, and hence performance can be increased (see [1]). In the longer term, (subject to continuation of the collaboration) the user interface, blocking strategy and distribution dependence will be tackled.

3. Progress

This collaboration has barely begun, but there has been some initial work that is very promising. Routine PCGETRF, a complex LU factorisation routine was revealed to spend an increasingly worrying proportion of time in blacs communications calls CGESD2D / CGERV2D and CGEBSD2D / CGEBRV2D on higher numbers of processors. Preliminary replacements to CGESD2D and CGERV2D show a significant performance improvement in both co-array and shmem replacements. Figure 1 shows the relative performance of these routines and the MPI-Blacs original. Without access to a production X1 system, and since this project is in such a junior stage, results must be restricted to a synthetic test problem. Blacs replacements for both shmem and Co-array look very promising, and replacements for collective routines PCGEBRVD / PCGEBD2D are expected to give a similar performance gain.

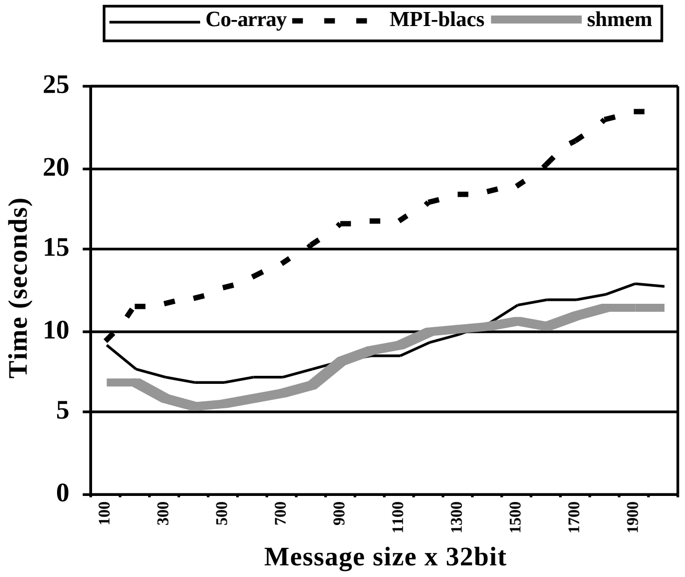

Figure 1 Preliminary BLACS replacement performance on a 2x2 processor grid

The optimisation of the library as a whole is a huge undertaking, and to aid the process, defined functionality is being captured in subroutines to avoid duplication in further work. Thus, operations such as the calculation of a remote process number, or the global location of a remote array element are performed externally in what will become an internal library of library tools.

A suite of high resolution timing routines have been created for this work, along with an interface that allows very simple result analysis, and a GUI that enables graphical representation of results at the click of a mouse. The features will help UoM staff spend more time in the optimisation process than in documentation process.

Around 20% of the work involved in creating new library routines is in developing these routines themselves. The remaining work is with creating a suite of testing and safety routines that can make sure the routine’s parameters are correct, check array bounds etc, and in testing the new library for robustness. Work in this area is likely to constitute a high percentage of the project time in the short and longer term.

Acknowledgements

Mary Beth Hribar, Chao Yang, John Levesque – Cray USA
Steve Jordan, David Tanqueray, Matt Thomason – Cray UK

References

[1] *High Performance Linear Algebra*, A.Tate & P. Briddon, CUG 2002 Proceedings

[2] *2D R-Matrix Propagations*, T. Stitt, S.Scott, P. Scott, P. Burke, Spinger Lecture Notes in Computer Science, 2565, pp354-367

[3] See *Using Plapack*, Robert Van der Geijn, MIT Press, 1997