Optimising Linear Algebra on SGI Origins

Adrian Tate
CSAR Applications Support, University of Manchester

In this article I outline some optimisation work that I have carried out for a user group, and how we intend, with information from the user community, to survey the use of linear algebra libraries with the intention of improving their performance.

ScaLAPACK, BLAS and Scalability

The code that was passed to me by Patrick Briddon from Newcastle University Physics was dependent upon a ScaLAPACK routine that affected scalability to the extent that a complete revision of the techniques was necessary. Scalapack is regarded as the standard distributed linear algebra library, so to deviate from this tradition is perhaps surprising, but if we take a look at the history of Scalapack and its emergence then we can perhaps see some validation in the decision. Many of the Scalapack authors are familiar names that created the BLAS and LAPACK (e.g. Dongarra, Demmel) and the package is infused with those features that are characteristic of BLAS. The PBLAS (Parallel BLAS) were designed to act as parallel version of Levels 1,2 and 3 BLAS whilst the BLACS (Basic Linear Algebra Communication Subprograms) were developed as the communications layer on which the PBLAS would operate.

It seems to me that the field of High Performance Computing has developed hugely since Scalapack was unearthed and that the term *scalable* is seen in a different light. Whilst a numerical analyst may require a code to be parallelised over a number of processors, his main concerns are the numerical stability and error propagation within the resultant code.

In the field of High Performance Computing, as the number of processors of production machines increases and as the availability of resources is more carefully administered, the overriding concerns are to produce truly scalable codes that can be used effectively on *any* number of processors. Hence, the use of ScaLAPACK routines for such purposes could well be described as an over-extension of its original design purpose. Benchmark figures for ScaLAPACK (see [1] ,[2] or [3]) reveal sub-linear scaling to be a generic feature since the ratio of communication/computation is very high. Thus, if we are to improve the scaling potential of Scalapack it is not by improving the computational efficiency but by reducing the time spent in communications.

Routine PBDTRNV

To continue we must imagine a large array (matrix) held on a distributed environment. The distribution in ScaLAPACK is always block-cyclic [4]. Figure 3 shows a 5x5 matrix distributed over a 2x2 BLACS process grid. The benefits of 2-d block cyclic storage lay in the feature of row elements remaining as row elements and column elements remaining as column elements. The code in question uses various ScaLAPACK and PBLAS routine to operate on this distributed process grid.

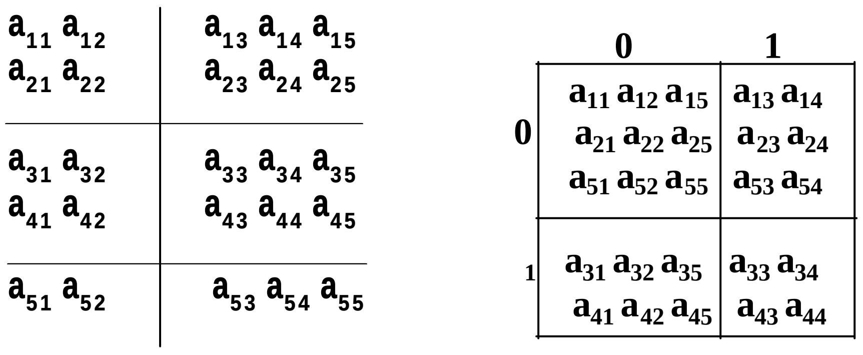
a_{11} a_{12} a_{21} a_{22}
a_{13} a_{14} a_{15} a_{23} a_{24} a_{25}
a_{31} a_{32} a_{41} a_{42}
a_{33} a_{34} a_{35} a_{43} a_{44} a_{45}
a_{51} a_{52}
a_{53} a_{54} a_{55}
0
1
0
a_{11} a_{12} a_{15} a_{21} a_{22} a_{25} a_{51} a_{52} a_{55}
a_{13} a_{14} a_{23} a_{24} a_{53} a_{54}
1
a_{31} a_{32} a_{35} a_{41} a_{42} a_{45}
a_{33} a_{34} a_{43} a_{44}

Figure 1. 2-d block cyclic distribution as used in Scalapack

To be brief, much of the lack of performance in the code was found to be due to Scalapack routine named PBDTRNV. This routine performs a seemingly simple operation upon a vector held in a process grid. Looking at the 3x3 process grid of Figure 4, we can visualize the transposition.

represent a) a natural overhead in terms of latency b) an over complication in terms of the linear algebra involved. In the case in question, the vector was never held within a two-dimensional array, so the extraction element of the routine was unnecessary, additionally the arrays were contiguous and required no condensing/stretching. This vastly simplifies the necessary options, especially if we can use one-sided communications.

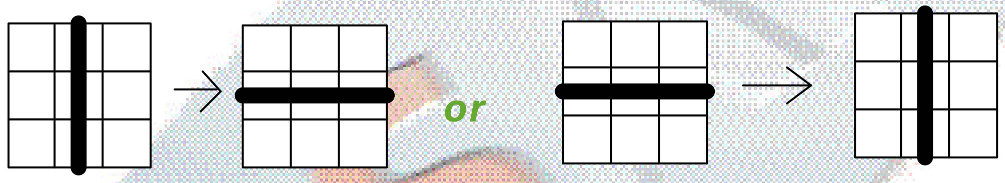

Figure 2. Transpose on a 3x3 process Grid

Though this operation appears simple, routine pbdtrnv is intricate and complex. The corresponding vector segments are extracted from the array within which they are contained, condensed (if not held contiguously), copied to a WORK array, sent to the appropriate processor (using the MPI BLACS), re-stretched and then copied into the appropriate array. Optionally the transposed vector is then copied across all process columns (or rows).

The problem here lays in the use of MPI as the message passing primitive on which the operations are performed. Though a simple operation, the two sided communications

Shmem point-to-point calls are tremendously efficient in comparison to their MPI counterparts. Figure 3 shows the relative performance of BLACS, MPI and shmem point-to-point data transfers. BLACS comms are via MPI so the figures for these are not surprising, but their inclusion in this data is important since they form the communication layer of Scalapack.

Vector length	MPI(SEND)	BLACS(DGERV2D)	SHMEM(GET)
100	0.221	0.249	0.0085
500	0.356	0.385	0.0379
1000	0.576	0.649	0.0602
1500	0.867	0.902	0.0884
2000	1.122	1.175	0.1180

Figure 3. Relative performance of point to point comms (based on 10000 data transfers)

The resultant replacement to pbdtrnv used shmem 1-sided communications to directly extract the necessary blocks of data for each processor. The work is thus reduced to 1) calculating the necessary target processor, and 2) using *shmem_get* calls to directly obtain the vector segment from the processor's memory. There are synchronization issues which will prevent the full performance gain one may expect from the data in figure 3, but with careful coding there is still a notable performance increase. We will present the results for this specific study at the CUG conference in May.

The natural conclusion to draw from this work is that there should be some work going into ScaLAPACK and its use of MPI_BLACS. Cray developed a shmem implementation of the BLACS some years ago, though on the Origins it remains standard to use the MPI version. There is scope to develop an optimised communications layer to existing ScaLAPACK routines specifically on Origins. One option would be, as with the Cray version to entirely use shmem comms. A problem with this is shmem's dependency upon powers of two numbers of processors for its collective routines (stride in collective routines are declared using $\log_2$stride). I'm addressing this issue and hope to create a shmem 'branch' broadcast [5] that can be used on any grid size and which will still outperform MPI_BCAST or the BLACS routine DGEBSD2D. This should improve the code mentioned above significantly for process grids that don't have the desirable grid lengths, whilst otherwise the normal shmem_bcast can be used (which outperforms MPI_BCAST by figures comparable to those in Figure 3.

Further, as we have proved with PBDTRNV, 1-sided comms can improve things at the routinelevel so there is a real possibility of CSAR staff being able to optimize specific library routines in this way. Further work will be to utilise the virtual shared-memory aspect of the Origin. Using careful data placement, remote data may be accessed and copied, thus reducing further the latency in data transfer). So, a library of high-performance replacements to commonly used routines will be developed.

.To aid us in this work it would be very beneficial to survey the CSAR community's relationship and dependency of ScaLAPACK or of any other distributed libraries, if you have got this far through this article you have some interest in this, so please contribute by letting us know if you have any kind of ScaLAPACK dependency in your code, what it is, and if it is causing performance problems. Please send your useful (or critical) comments to adrian.tate@man.ac.uk.

References

[1] NCSA Scalapack Origin 2000 Benchmarks: http://www.ncsa.uiuc.edu/People/sirpa/all.html

[2] UCLA Scalapack benchmarks http://www.ats.ucla.edu/at/beowulf/parallel/parallel_benchmarks.htm#SCALAPACK%20Benchmark

[3]Lemans Scalapack benchmarks http://weblotus.univ-lemans.fr/w3lotus/node15.html

[4] Scalapack users guide http://www.netlib.org/scalapack/slug/node75.html#SECTION04431000000000000000

[5] A branching algorithm applied to a process grid of length n will require $\log_2 n$ shmem_put/get operations, as such it is likely that such a routine will out-perform the MPI_BCAST and BLACS_BROADCAST routines. Further work would be necessary to generalize shmem_barrier calls that are similarly dependent upon $\log_2$ strides.