Invertastic: Large-scale Dense Matrix Inversion

Alan Gray, EPCC, The University of Edinburgh

June 15, 2016

Abstract

Linear algebraic techniques are widely used in scientific computing, often requiring large-scale parallel resources such as those provided by the ARCHER service. Libraries exist to facilitate the development of appropriate parallel software, but use of these involves intricacies in decomposition of the problem, managing parallel input and output, passing messages and the execution of the linear algebra operations themselves. In this paper a relatively simple application, Invertastic, is presented. This is designed to perform a real operation: the inversion of a dense symmetric positive definite matrix using multiple processors in parallel. The inversion of arbitrarily large matrices is demonstrated, with the only constraint being the size of compute resource available. Inversion cost is known to have $O(N^3)$ complexity, which the results confirm allowing for some parallel communication overhead. Inversion of a 2,097,152 x 2,097,152 matrix (of size 32TB) took 6.4 hours on 2,048 compute nodes (49,152 cores). The Invertastic software is freely available online and installed as a central package on ARCHER. It can be used directly (e.g. for genomic studies where the matrix represents the genetic relationships between multiple individuals), or instead as a reference or template for the development of more complex algorithms.

1. Introduction

Many computational problems involve linear algebraic techniques, and libraries which facilitate such operations have been available for many years. However, when operating on large-scale parallel computers, it is often difficult for the user to quickly develop the required software because of the complexities involved with driving these libraries correctly. This paper, and the associated software, is aimed at providing a relatively simple test-case for large-scale linear algebra, in particular the inversion of a large-scale dense symmetric positive definite (SPD) matrix, that can be used directly or as a reference.

One area where such operations can be directly useful is that of genomics. A construct often employed in regression analysis is the Genomic Relationship Matrix (GRM). For a study with N individuals, the GRM is a square SPD NxN matrix which, for each pair of individuals, provides a measure of genomic correlation. This can then be used together with phenotype information for regression, with many techniques requiring that the inverse of the GRM is calculated (see e.g. [1] and [2]). Traditionally, the GRM would be constructed using known pedigree information, and would hence have a relatively sparse structure. However, with the recent revolution in sequencing, dense GRMs can be constructed where nonzero genomic relationships exist even between distant individuals. The number of individuals continues to dramatically increase as sequencing costs decrease. For sparse matrices, there exist inversion techniques that take advantage of the sparsity to minimise memory usage and compute time; however for dense matrices, as N becomes large it is necessary to distribute the memory and compute requirements across parallel compute clusters or supercomputers.

In this paper *Invertastic*, a utility for inverting large-scale dense SPD matrices using parallel computing techniques and libraries, is introduced. Invertastic is a freely-available utility that, if not used directly, may be a good starting point for those wishing to implement different or more complex algorithms. The source code is freely available under the Apache 2 open source licence [3]. Invertastic is also installed as a centrally available ready-to-use package on the ARCHER UK national supercomputing facility.

Section 2 describes the structure of Invertastic, and the computational performance is presented in Section 3.

2. Background and Methods

Parallel computers comprise multiple interconnected nodes, each (similar to a personal workstation) containing multiple cores (perhaps across multiple chips), where each core can directly access memory local to the node, but not that of remote nodes. Parallel computing algorithms exploit multiple nodes to solve a single problem by distributing work and data across the nodes, where parallel communications exchange data between nodes as necessary.

Parallel libraries aide this process: MPI [4] and BLACS for message passing, where the latter is a lightweight layer on the former, required by PBLAS and ScaLAPACK which is used for linear algebra [5][6]. MPI-IO is used for parallel I/O operations. The base language used is C.

Algorithm 1 The steps performed by Invertastic to perform a parallel inversion of an SPD matrix. The prefix ^{P} indicates the use of global parallel communications: otherwise the step is performed independently on each local sub-matrix.

```
 1: $^P$ Init comms, MPI-IO and local memory
 2: **if** “–input filename” option specified **then**
 3:     $^P\ A \leftarrow_{\text{MPI-IO}}$ matrix on disk
 4: **else**
 5:     $A \leftarrow$ random init
 6:     $^P\ A \leftarrow_{\text{pgeadd}} A + A^T$ (make symmetric)
 7:     increase diagonal of $A$ to make positive definite
 8: **if** “–check” option specified **then**
 9:     $\hat{A} \leftarrow_{\text{copy}} A$ (for later verification)
10: $^P\ LL^T \leftarrow_{\text{pdpotrf}} A$
11: $^P\ A^{-1} \leftarrow_{\text{pdpotri}} LL^T$
12: **if** “–output filename” option specified **then**
13:     $^P$ matrix on disk $\leftarrow_{\text{MPI-IO}} A^{-1}$
14: **if** “–check” option specified **then**
15:     $^P\ B \leftarrow_{\text{pdsymm}} A^{-1}\hat{A}$
16:     Local element-wise comparison of $B$ with $I$
17: $^P$ Finalise comms and memory
```

Algorithm 1 shows the steps performed by the utility. The main inversion is performed using the ScaLAPACK library in steps 10 and 11. Cholesky factorisation (`pdpotrf` routine) is used to decompose the matrix to the form $A = LL^T$ where L is a lower triangular matrix. This is then used to then determine the inverse A^{-1} (with the `pdpotri` routine). Before these steps, the matrix must exist in distributed form across the parallel computer. For regular usage, the `--input <filename>` option will be specified by the user, and the file is read in parallel using MPI-IO operations (line 3). A block cyclic distribution (supported by both MPI-IO and ScaLAPACK) is used, with block size 128 in each of the two dimensions. Similarly, after the inversion the resulting inverse is written to file (line 13). When no input file is specified, to allow performance benchmarking and testing the code will create an SPD matrix on-the-fly using random numbers (lines 5-7). If the `--check` option is specified, the code will multiply the original matrix by the resulting inverse, compare this product (element-wise) with the expected identity matrix, and print the maximum deviation.

3. Results and Discussion

The UK national supercomputing facility, ARCHER, is a Cray XC30 architecture comprising 4,920 compute nodes, each featuring two Intel 12-core E5-2697 v2 (Ivy Bridge) series processors (i.e. a total of 118,080 compute cores). Nodes are tightly coupled via the Cray Aries interconnect. Each matrix is represented in Invertastic using NxN double precision (8 byte) values, and the software allocates 3 separate matrix storage areas, so $24N^2$ bytes is required. On ARCHER, each node has access to 64GB of memory, so the largest possible submatrix per node is 46,000×46,000. As N increases, the the number of nodes is increased accordingly to acquire enough memory. The Cray MPI library is used, and the linear algebra libraries are provided through the Intel Math Kernel Library. A separate MPI task is assigned to each core, such that there are 24 MPI tasks per node.

Figure 1 shows the cost of computing the matrix inverse with increasing N. The smallest matrix has N=32,768 on a single node (requiring 24GB). For the second smallest matrix, N is doubled to $65,536$ and use 2 nodes, with 48GB per node. N is then doubled

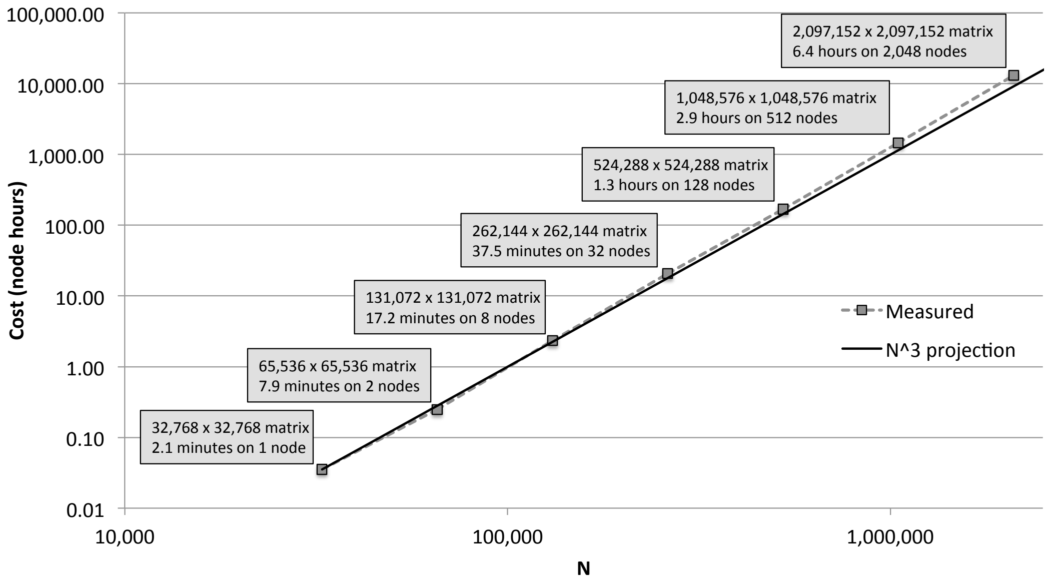

Figure 1: The cost of Invertastic for dense SPD matrix inversion on ARCHER. The total measured time (in node hours) is plotted for a range of N, where the square matrix is of size NxN. The number of nodes (each containing two 12-core Intel Ivy-bridge CPUs) increases with the matrix size. Also plotted is a projection of the cost using only the smallest measured result as a baseline and assuming $O(N^3)$ complexity.

several further times, each time increasing the node count by a factor of 4 (retaining 48GB per node), up to the largest matrix with N=2,097,152 on 2,048 nodes using a total of 32TB of memory. On each node all 24 cores are utilised, i.e. the largest result uses 49,152 cores.

The cost is given in node hours (the number of nodes multiplied by the number of hours runtime). This cost can be directly translated to a monetary figure, where the exchange rate varies from around £0.48 to £2.40 per node hour, depending on the mode of access to ARCHER.

The computational complexity of matrix inversion is $O(N^3)$. Therefore, the cost measured on a single node can be used to project a predicted cost for larger N via the multiplicative scaling factor $(N/N_{1-node})^3$, where N_{1-node} is the size used for the single-node problem. This projection, which is plotted for comparison, assumes perfect parallel scalability, and the deviation of our measured results gives an indication of communication overhead caused by the distribution across nodes.

For our four largest problems, in order of increasing problem size, this overhead is 15%, 17%, 27% and 43%: significant but far from debilitating. The largest 2,097,152 X 2,097,152 matrix was inverted in 6.4 hours using 2,048 nodes: this would be reduced to 4.5 hours with perfect scalability. Note that, while power-of-2 sizes are chosen for performance analysis, all sizes are supported and there is no discernible performance advantage from this choice.

Since this procedure involves reading/writing large matrices from/to disk, it is imperative to not only use parallel I/O within the application, but also to ensure appropriate settings regarding how each file is handled by the parallel filesystem: otherwise the available bandwidth can be throttled by orders of magnitude [7]. Here, with use of MPI-IO combined with properly configured file striping, the I/O overhead is recorded to be relatively small at around the 10% level.

For validation, the resulting inverted matrix is multiplied by the original matrix with the result compare the result with the expected identity matrix. This check has been performed up to N=524,288, where the maximum element-wise difference (due to rounding) is 1.6×10^{-14}. This is seen to increase only sub-linearly with N, giving confidence that it remains many orders of magnitude less than the typical error due to noise in the data itself, even for large matrix sizes.

For even larger matrix inversions, more nodes can simply be employed to provide the required memory. The parallel communication overhead will continue to grow as the number of nodes increases, meaning further deviation from the ideal N^3 scaling, but in terms of feasibility the only limiting factor is the size of the compute resource available.

References

[1] VanRaden, P. M. (2008). Efficient methods to compute genomic predictions. *Journal of dairy science*, 91(11), 4414-4423.

[2] Meyer, K., Tier, B., Graser, H. U. (2013). Technical note: updating the inverse of the genomic relationship matrix. *Journal of animal science*, 91(6), 2583-2586.

[3] Alan Gray, (2016) Invertastic GitHub Repository, `github.com/agray3/invertastic`.

[4] Gropp, W., Lusk, E., Doss, N., Skjellum, A. (1996). A high-performance, portable implementation of the MPI message passing interface standard, *Parallel computing*, 22(6), 789-828.

[5] Dongarra, J. *et al.* (1990) A set of Level 3 Basic Linear Algebra Subprograms, *ACM Trans. Math. Soft., 16*, pp. 1–17.

[6] L. Blackford *et al.* (1997), ScaLAPACK Users' Guide, *Society for Industrial and Applied Mathematics*, 0-89871-397-8 (paperback),

[7] David Henty, Adrian Jackson, Charles Moulinec, Vendel Szeremi (2014) Performance of Parallel IO on ARCHER, *ARCHER White paper* 1402.