THE AUSTRALIAN NATIONAL UNIVERSITY

TR-CS-98-10

Optimal Load Balancing Techniques for Block-Cyclic Decompositions for Matrix Factorization

Peter Strazdins

September 1998

Joint Computer Science Technical Report Series

Department of Computer Science
Faculty of Engineering and Information Technology

Computer Sciences Laboratory
Research School of Information Sciences and Engineering

Optimal Load Balancing Techniques for Block-Cyclic Decompositions for Matrix Factorization

Peter Strazdins
peter@cs.anu.edu.au

Department of Computer Science,
Australian National University,

September 15, 1998

Abstract

In this paper, we present a new load balancing technique, called panel scattering, which is generally applicable for parallel block-partitioned dense linear algebra algorithms, such as matrix factorization. Here, the panels formed in such computation are divided across their length, and evenly (re-)distributed among all processors. It is shown how this technique can be efficiently implemented for the general block-cyclic matrix distribution, requiring only the collective communication primitives that required for block-cyclic parallel BLAS. In most situations, panel scattering yields optimal load balance and cell computation speed across all stages of the computation. It has also advantages in naturally yielding good memory access patterns.

Compared with traditional methods which minimize communication costs at the expense of load balance, it has a small (in some situations negative) increase in communication volume costs. It however incurs extra communication startup costs, but only by a factor not exceeding 2. To maximize load balance and minimize the cost of panel re-distribution, storage block sizes should be kept small; furthermore, in many situations of interest, there will be no significant communication startup penalty for doing so.

Results will be given on the Fujitsu AP+ parallel computer, which will compare the performance of panel scattering with previously established methods, for LU, LLT and QR factorization. These are consistent with a detailed performance model for LU factorization for each method that is developed here.

Keywords: dense linear algebra, block cyclic decomposition, storage blocking, algorithmic blocking, physically based matrix distribution.

1 Introduction

Dense linear algebra computations such as LU, LLT (Cholesky) and QR factorization require the technique of 'block-partitioned algorithms' for their efficient implementation on memory-hierarchy processors. Here, the rows and/or columns of a matrix are partitioned into *panels*, ie. block row/columns of width $\omega \geq 1$, and by performing matrix-vector or matrix-matrix operations on these panels. Once these panels are formed, the remainder of the computation typically involves 'Level 3' or matrix-matrix operations, which can run at optimal speed.

In the distributed memory multiprocessor context, most, if not all, communication occurs within the panel formation stage. Once the optimal panel width $\omega = \omega_m$ is achieved for the matrix-matrix operations, the most scope for the acceleration of the overall computation is in the optimization of the panel formation stages.

In this paper, we will consider the $r \times s$ block-cyclic matrix distribution over a $P \times Q$ logical processor grid (see Figure 1) [5], where, for an $N \times N$ global matrix A, block (i, j) of A will be on processor $(i \bmod P, j \bmod Q)$. We will now review two established techniques for parallel panel formation, known as *storage blocking*, where $\omega = r = s$, and *algorithmic blocking* [6, 13, 9, 10], where $\omega \approx \omega_m, r \approx s \approx 1$.

Storage blocking suffers from load imbalance on the panel formation stage, in that only one row or column of processors of the grid will be involved in this stage, which is an $O(\omega/N)$ fraction of the overall computation. Furthermore, there is an $O(N^2(r/Q + s/P))$ load imbalance on the Level 3 computation, since the cell owning the last block row and column will have more overall work to do than the others in these stages [3, 10]. For small to moderate N, these imbalances can be significant, so that a value of $\omega < \omega_m$ may be optimal. It is widely believed that storage blocking minimizes communication startup and volume overheads.

Algorithmic blocking (also known as 'distributed panels' [13]) achieves better load balancing properties by distributing the panel across all processors, at the expense of some increased communication overhead, both in startup and volume. It generally achieves very high load balance. Whether a multiprocessor favors algorithmic blocking over storage blocking depends of whether ω_m is large, and, to a lesser extent, whether the ratio of communication to floating point speed is relatively high [13].

However, the load balance in panel formation is not perfect, being about 81% for $\omega = 32, r = 1$ and $P = 8$, but diminishing for larger P/ω ratios [13]. Furthermore, since algorithmic blocking involves distributing a panel across its *narrowest* dimension, this can degrade cell computation speed. This is because each processor's portion of the panel has a small local width, which can be thought of as reducing a Level 3 (or Level 2) sub-computation into a 'Level 2.5' (or 'Level 1.5') sub-computation.

Algorithmic blocking has been shown to have a 15–30% performance gain over storage blocking on LU and LLT factorizations on the Fujitsu AP1000 and AP+ multicomputers, even using implementations which have redundant communications, which can increase the communication volume associated with panel formation by as much as 100% [13, 17]. If one is prepared to pay such a cost in extra communication volume, some of the load balance and cell computation speed deficiencies of algorithmic blocking can be overcome.

Consider again the storage blocking scheme with $\omega = r = s$ for the formation of a vertical panel L of size $M \times \omega$, with $M \gg \omega$. L is a sub-matrix of the matrix being factorized; it is initially contained in processor column $q, 0 \leq q < Q$. In each cell in column q, L could be sliced (vertically) into Q pieces along the shortest dimension: this would create a similar situation as for algorithmic blocking. However, instead L could be sliced (horizontally) into Q pieces along the longest dimension, and scattered to each of the Q cells across that row. The panel formation can then proceed with near-perfect load balance. Once L is formed, it is replicated in all processor columns, and the original part of the matrix corresponding to L is updated on cell column q.

This technique we call *panel scattering*, although $\omega = r = s$ is not required. Unlike algorithmic blocking, it obeys the general parallel computing principle of making the local cell portion of the (scattered) panel as near as possible to square, thereby gaining a greater volume to surface area (which in turn, tends to reduce memory hierarchy traffic).

The extra communication overhead occurs on the *scatter* of the panel, which, because it involves only point-to-point sends, is at least as efficient as a broadcast operation. However, pipelined communication is lost, and one-dimensional reduction and broadcasting operations are replaced by two-dimensional operations, which may well be slower.

This paper is organized as follows. Section 1.1 describes recent related work, based on the Physically Based Matrix Distribution [18]. The Fujitsu AP+ multicomputer is described in Section 1.2. Key ideas in panel scattering for block-cyclic decompositions are developed in Section 2. The implementation of panel scattering for the matrix factorizations is described in Section 3. The performance on the AP+ of panel scattering, algorithmic blocking and storage blocking is given in Section 4. Section 5 gives an analysis of the various methods for LU factorization, resulting in a performance model which verifies that our implementations behave much as expected for these methods. Conclusions are given in Section 6.

1.1 Related Work

In [15], the idea of *panel scattering* was introduced as an alternate load balancing technique of parallel matrix factorization, with a qualitative comparison of its computational and communication performance against algorithmic and storage blocking being given. As no implementation or results for panel scattering were given there, this paper is a follow-up for that work.

Very recently, a similar technique has been proposed in [1] for matrix factorizations using the Physically-Based Matrix Distribution (PBMD), which forms the basis for the PLAPACK project [18]. In PBMD, a vector is typically decomposed into blocks of size r which, on a $P \times Q$ grid, are cyclically distributed over PQ cells in column-major order, ie. by assigning block i to cell (p,q) where $i \equiv (p+qP)$ modulo PQ. A matrix distribution is then *induced* by making its rows *and* columns 'conformal' to this vector [18]. This distribution amounts to a special case of the block-cyclic distribution where:

$$s = Pr \tag{1}$$

Thus, to implement the PBMD equivalent of panel scattering, the lower panel L, initially distributed over a column of cells, is transformed into a *multivector* L' distributed over all cells via a scatter operation. Once this panel is factored, it is gathered back again into the original matrix, before replicating it for subsequent (parallel BLAS) operations [1].

In [1], PLAPACK algorithms for LLT, LU and QR factorizations are given. Some performance comparisons with the corresponding ScaLAPACK routines [2], which use storage blocking, were given on a 4×4 Cray T3E, for which the version of cell BLAS used at that time had $\omega_m = 128$. Results were only given for $\omega = \omega_m = 128, r = \omega/2$, which was presumably the best combination for the implementation of PLAPACK at that time. However there was no detailed performance analysis given to compare the two methods.

One claim of [1] is that the high level of abstraction within PLAPACK was important in developing a more complex but (arguably) more efficient algorithm than storage blocking. We believe that this claim this applies equally our development of panel scattering for the more general block-cyclic distribution, especially with our aim to investigate both small and large block sizes in order to seek maximal load balance.

1.2 The Fujitsu AP+

The Fujitsu AP+ [7] is a SPARC 10-based multiprocessor; it's cells have a theoretical peak speed of 50 MFLOPs (double precision). The AP+ uses a physical torus communication network using wormhole routing, with each link being capable of 25 MBs^{-1} in either direction. A useful feature of this network is the ability to perform a row or column broadcast for the cost of a normal point-to-point message (ie. $\kappa_{bc} = 1$ in the notation of [4]); such a broadcast will be referred to as a *unit-cost broadcast*. On the AP+, these do *not* require any synchronization, in that the sender does not have to wait till the other cells post the corresponding receive calls.

Unit-cost broadcasts are very easy to implement in hardware: with wormhole routing, they only require the communication routers to be able to forward the packets of a broadcast message to both the next node in the network, and to the adjacent processor. Given that row and column broadcasts are so widely used in dense linear algebra (and other applications), it is surprising that other vendors have not emulated this capability.

The AP+ cells have a set-associative write-through 16 KB data cache, and a set-associative 20 KB instruction cache. There is no second-level cache; this reduces level-3 and especially level-2 computation performance. The optimal logical blocking factor is $\omega_m \approx 32$. Virtual memory is implemented on the AP+ cells; thus attention has to be given to memory access patterns when optimizing program performance.

On the AP+, benchmark programs have yielded the following parameters that will be used in Section 5 for (a slightly extended version [16, 10] of) the Distributed Linear Algebra Model (DLAM) of [4]. The communication startup cost is $\alpha = 12\mu s$, the communication transmission cost per word is $\beta = 0.64\mu$s (corresponding to 12.5MBs), the level-2 computation cost per floating point operation is $\gamma_2 = 0.12\mu$s and the computation speed for a triangular matrix update operation (eg. LT^{-1} or $T^{-1}U$) is $\gamma_3^{\triangle} = 0.083\mu$s. $\gamma_3(16) = .035\mu$s and $\gamma_3(64) = .030\mu$s (this corresponds to 33 MFLOPs), where $\gamma_3(\omega)$ is the cost per floating point operation in a local matrix multiply having input operands of width ω.

The AP+ has also two mechanisms for global broadcasts, which are required in the lower panel formation of LU and QR factorization. The first uses a special *broadcast network*, which can attain a speed of 50 MBs, but with a latency much higher than the row broadcast, which makes it too slow for small messages. The second is by the same mechanism as the row and column broadcasts. While in principle this should also be of unit-cost, which would make it ideal for panel scattering, in practice the latency is $\approx 2.0\alpha$. This is due to an implicit synchronization, which is apparently required to ensure the safety of this operation on the AP+.

$l_{0,0}$ $l_{3,0}$ $\dots$ $l_{24,0}$	$l_{0,1}$ $l_{3,1}$ $\dots$ $l_{24,1}$		
$l_{1,0}$ $l_{4,0}$ $\dots$ $l_{22,0}$	$l_{1,1}$ $l_{4,1}$ $\dots$ $l_{22,1}$		
$l_{2,0}$ $l_{5,0}$ $\dots$ $l_{23,0}$	$l_{2,1}$ $l_{5,1}$ $\dots$ $l_{23,1}$		

(a) before scatter

$l_{0,0}$ $l_{0,1}$ $l_{12,0}$ $l_{12,1}$ $l_{24,0}$ $l_{24,1}$	$l_{3,0}$ $l_{3,1}$ $l_{15,0}$ $l_{15,1}$	$l_{6,0}$ $l_{6,1}$ $l_{18,0}$ $l_{18,1}$	$l_{9,0}$ $l_{9,1}$ $l_{21,0}$ $l_{21,1}$
$l_{1,0}$ $l_{1,1}$ $l_{13,0}$ $l_{13,1}$	$l_{4,0}$ $l_{4,1}$ $l_{16,0}$ $l_{16,1}$	$l_{7,0}$ $l_{7,1}$ $l_{19,0}$ $l_{19,1}$	$l_{10,0}$ $l_{10,1}$ $l_{22,0}$ $l_{22,1}$
$l_{2,0}$ $l_{2,1}$ $l_{14,0}$ $l_{14,1}$	$l_{5,0}$ $l_{5,1}$ $l_{17,0}$ $l_{17,1}$	$l_{8,0}$ $l_{8,1}$ $l_{20,0}$ $l_{20,1}$	$l_{11,0}$ $l_{11,1}$ $l_{23,0}$ $l_{23,1}$

(b) after scatter

Figure 1: Scattering of the blocks of a $25r \times 2s$ block-cyclic panel L, whose leading block is in cell $(0, 0)$, on a 3×4 grid

2 Panel Scattering on Block-Cyclic Decompositions

In this section, it will be shown how communication operations that are required in any case for a block-cyclic parallel BLAS implementation, namely transpose and *spread* (a form of multicast), can be used to implement panel scattering. It will also be shown that the resulting scattered panel obeys a block-cyclic distribution on a linear virtual grid (or *communication context*). Both of these allow an easy and elegant implementation of panel scattering using an existing block-cyclic parallel BLAS library.

Consider again the $M \times \omega$ vertical panel L. Figure 1 illustrates the scattering of such a panel. The scattered panel L' in Figure 1(b) can be regarded as an $M \times \omega$ block-cyclic matrix over the $PQ \times 1$ grid formed by assigning processor ids in a column-major order.

Now consider L^p and L'^p, the portions of L and L' respectively across grid row $p, 0 \leq p < P$. Denoting m^p as the length of L^p, we can also regard L^p as an $m^p \times \omega$, and L'^p as an $\omega \times m^p$, matrix distributed across a $1 \times Q$ sub-grid (imagine that the local portions of L'^p are stored in a transposed layout from that shown in Figure 1(b)). From this observation, one can derive the central result for panel scattering:

Result 1 *L' is an $r \times s$ block-cyclic distributed matrix over a $PQ \times 1$ grid, where L' is formed from the concatenation of L'^p, $0 \leq p < P$, and $L'^p = (L^p)^T$.*

Proof:

Assume, without loss of generality, that the leading block of L is on cell $(0, 0)$. Then global block (i, j) of L, where $i = i'P + p$ and $0 \leq p < P$, forms part of L^p. After the row-wise transposition $(L^p)^T$, this block ends up on cell $(p, i'\%Q)$, by the defn. of transposition on a $1 \times Q$ sub-grid. This is cell $(p + (i'\%Q)P, 1)$ of the grid of L'.

By induction on i', it can be easily proved that $p + (i'\%Q)P = i\%(PQ)$, and hence that block (i, j) of L' resides on cell $(i\%(PQ), 1)$. Hence L' obeys a block-cyclic distribution over the $PQ \times 1$ grid beginning at cell $(0, 0)$. $\Box$

Once the panel L' is formed, and then appropriately updated for a matrix factorization, it needs to be replicated row-wise across the original $P \times Q$ grid. Denoting $\text{spread}_\text{r}(L)$ $(\text{spread}_\text{c}(L))$ to be the row-wise (column-wise) replication of L [14], the following result applies to the combined 'gather-replication' of L':

Result 2 $(\text{spread}_\text{r}(L))^p = \text{spread}_\text{r}((L'^p)^T)$.

Proof:

This follows immediately by noting that for a purely row-wise operation, $(\text{spread}_\text{r}(L))^p = (\text{spread}_\text{r}(L^p))$, and using $L^p = (L'^p)^T$ from Result 1. $\Box$

14	2	6	10
15	3	7	11
12	0	4	8
13	1	5	9

(a) vertical panel L ($PQ \times 1$ grid)

11	8	9	10
15	12	13	14
3	0	1	2
7	4	5	6

(b) horizontal panel U ($1 \times PQ$ grid)

Figure 2: Processor ids in the communication contexts for scattered panels having leading block in cell (2,1) of a 4×4 grid.

However, the main significance of this result is that the combined *transposed-spread*, $\text{spread}_\text{r}((L'^p)^T)$, which occurs over a $1 \times Q$ grid, can be performed for the cost of an ordinary spread operation. Indeed the same communication operation which forms $\text{spread}_\text{r}((L'^p)^T)$ from $\text{spread}_\text{c}(L'^p)$ in a symmetric rank-k update $A \leftarrow A - (L'^p)^T L'^p$ (as is used in LLT factorization), can be used here, by simply setting the vertical grid size to unity, for which $L'^p = \text{spread}_\text{c}(L'^p)$.

The corresponding results apply to a horizontal $\omega \times N$ panel U with a row-major $1 \times PQ$ grid. The above results also apply for L (or U) having their leading block at an arbitrary cell (p, q) in the original $P \times Q$ grid, since we can always define a virtual $P \times Q$ grid with an origin at this cell. However, practicalities arise for $p, q > 0$ that an implementation of panel scattering must take into account.

Figure 2 shows such a situation for $(p, q) = (2, 1)$. For the lower panel, the first 4 cells in the grid of L' remain in the original column q. Cell $(p', 0)$ in the grid for L' maps to cell $((p' + p)\%P, p'\%Q)$ in that of L. This leads to the following result, which is sufficient to implement the communication context for L':

Result 3 *The co-ordinates of relative cell* $(\delta, 0), 0 \leq \delta < PQ$, *in the grid of* L' *is that of the relative cell* $(\delta\%P, \frac{\delta+p}{P})$ *in the original* $P \times Q$ *grid.*

This result may be proved by simple arithmetic. It also implies that p must be 'remembered' when the context for L' is formed.

3 Implementation of Panel Scattering

The DBLAS block-cyclic distributed BLAS library [14, 16] was used to implement these algorithms. This implementation requires block alignment in all but one of the common matrix dimensions in the operands of a DBLAS call. This enables non-square block sizes to be handled in many situations.

In a DBLAS call, distributed matrix operands are represented by `DistMat` data objects. These contain information about the block sizes and also the processor grid size for that operand. If an underlying $P \times Q$ grid is being used, input operands can be specified to be on $1 \times Q$, $P \times 1$, or 1×1 sub-grids, corresponding to row-replicated, column-replicated and row-and-column replicated matrices.

In terms of functionality, a DBLAS call has the same effect on a global matrix as its ordinary BLAS counterpart; note however that the BLAS side and transposition specifiers, as well as local matrix storage information, are hidden inside `DistMat` data objects. Local matrix storage may be either column-major, or row-major.

This high level of abstraction enables one DBLAS-based coding of an algorithm to transparently handle a variety of block sizes and storage schemes, which can potentially affect performance.

To support panel scattering, the DBLAS only had to be extended in the following ways:

- implementing the contexts for the scattered panels (eg. L', U'). Result 3 implies these must be created dynamically, as the context depends on which cell contains the leading block of the panel (eg. L, U).

  While the DBLAS has a BLACS interface [14], to avoid the typical overhead of dynamic context creation in the BLACS, these contexts were implemented directly in the DBLAS communication interface module.

```
for($j = 0; j < \min(M,N); j$+=$\omega$)
    $L'$= PanelScatter(ROW, $M-j, \omega,\ A_{j,j}$);                     // $L' \leftarrow A_{j..M',j..j'_\omega}$ (on a $PQ \times 1$ grid)
    $D$GETF2($M-j, \omega, L', p$);                                        // level-2 LU fact'n of $L'$, with pivot vector $p_{0..\omega'}$
    $\tilde{L}$= ScatterSpread(ROW, $M-j, \omega,\ A_{j,j},\ L'$);         // $\tilde{L} \leftarrow L'$ (column-replicated, on a $P \times Q$ grid)
    $D$LASWP("Fwd", $j, \omega,\ A^T_{j,0},\ p$);                          // $A_{j..j'_\omega,\ 0..j'} \leftarrow P(\omega', p_{\omega'}) \ldots P(0,p_0) A_{j..j'_\omega,\ 0..j'}$
    $D$LASWP("Fwd", $j, \omega,\ A^T_{j,j_\omega},\ p$);                   // $A_{j..j'_\omega,\ j_\omega..N'} \leftarrow P(\omega', p_{\omega'}) \ldots P(0,p_0) A_{j..j'_\omega,\ ;j_\omega..N'}$
    $\tilde{\tilde{T}}$= SpreadCopy(COL, $\omega, \omega,\ \tilde{L}$);     // $\tilde{\tilde{T}} \leftarrow \tilde{L}_{0..\omega',:}$ (replicated on all cells)
    $U'$= PanelScatter(COL, $\omega, N-j_\omega,\ A_{j,j_\omega}$);        // $U' \leftarrow A_{j..j'_\omega,j_\omega..N'}$, (on a $1 \times PQ$ grid
    $D$TRSM("Lower","Unit", $\omega, N-j_\omega, 1.0,\ \tilde{\tilde{T}},\ U'$);   // $U' \leftarrow \tilde{\tilde{T}}^{-1} U'$, $\tilde{\tilde{T}}$ is lower tri., unit diag.
    $\tilde{U}$= ScatterSpread(COL, $\omega, N-j_\omega,\ A_{j,j_\omega},\ U'$);   // $\tilde{U} \leftarrow U'$ (row-replicated, on a $P \times Q$ grid)
    $D$GEMM($M-j_\omega, N-j_\omega, \omega,\ -1.0,\ \tilde{L}_{\omega,0},\ \tilde{U},\ 1.0,\ A_{j_\omega,j_\omega}$);   // $A_{j_\omega..M',\ j_\omega..N'}$ -= $\tilde{L}_{\omega,0}\tilde{U}$
    $D$COPY($M-j, \omega,\ \tilde{L},\ A_{j,j}$);   $D$COPY($\omega, N-j_\omega,\ \tilde{U},\ A_{j,j_\omega}$);   // $A_{j..M',j..j'_\omega} \leftarrow \tilde{L}$, $A_{j..j'_\omega,j'_\omega.N'} \leftarrow \tilde{U}$
    <dispose $L', \tilde{L}, \tilde{\tilde{T}}, U', \tilde{U}$>;
```

Figure 3: DBLAS-based LU Factorization algorithm of an $M \times N$ matrix A ($P(i,j)$ is the idenity matrix with rows i and j permuted)

- implementing the panel scattering routine `PanelScatter()`. This required 'forging' $1 \times Q$ sub-grids in the data objects representing L and L', and a call to the internal DBLAS parallel transpose routine. Here, each cell packs together all blocks to be sent to another cell, sends this as a contiguous message, and the receives and unpacks any messages from other cells. The block-cyclic packing/unpacking routines are optimized for both small and large block sizes.

  A further issue is the local storage of L', which can be either unit stride along the length of the panel (column-major for L'), which is likely to yield the best memory access patterns, or along its width, which can avoid packing and unpacking overheads when the rows of L' are communicated. Since both are easy to implement using the DBLAS, both options will be tried.

- implementing the combined 'gather-replication' of L' routine, `ScatterSpread()`. This similarly required the forging of $1 \times Q$ sub-grids in the data objects representing L' and $\tilde{L}$, followed by a call the internal DBLAS spread routine.

The following algorithms are presented in DBLAS-based pseudocode. Matrices (eg. A) are represented by `DistMat` descriptors. To simplify the presentation, sub-matrix references $A_{i,j}$ represents the function call `SubMat(A, i, j)`, and A^T represents the function call `Trans(A)`. The latter does *not* perform any data movement; it merely toggles a bit in the `DistMat` descriptor to signify that the operand is globally transposed from its 'normal' sense. For integer expressions, the shorthand i' denotes $i-1$. j_ω denotes $j+\omega$. The prefix D in front of a BLAS or LAPACK procedure name signifies the DBLAS distributed equivalent, on the appropriate precision (eg. double precision). Apart from these, and some other mainly syntactic simplifications, the pseudocode corresponds closely to the actual DBLAS code.

Figure 3 gives the algorithm for panel scattered LU factorization. The actual code keeps the pivot vector for all of A replicated on all cells; this was omitted to simplify the presentation. Note that the level-2 factorization code DGETF2() did not require any modification from that developed for algorithmic blocking (see [16]). This was also the case for LLT and QR.

In general, for each panel, there are 4 extra calls to be made: to scatter the panel, to 'gather-replicate' the panel, to copy the replicated panel back into the original matrix and to dispose of the panels. Also, note that triangular factors (eg. $\tilde{\tilde{T}}$) are replicated across *all* cells before being applied to a scattered panel.

The panel-scattered LLT algorithm is given in Figure 4. The portion of the matrix for level-2 factorization, $A_{j..j'_\omega,j..j'_\omega}$, is small; for this reason, it is not scattered before factorization, as this will achieve no load balance advantage.

For LLT factorization, with $r \neq s$, alignment problems occur in the call to D`SYRK()`, where block alignment is required for the efficient selection of the rows of $\tilde{L}$ that will contribute to $(\tilde{L})^T$ for the current processor column. For this reason, mixed block sizes could not be implemented for LLT.

The panel-scattered QR algorithm is given in Figure 5. As is done for the ScaLAPACK QR routine [5], the calls to DGEMM() introduce redundant floating point operations with the above-diagonal part

```
for($j = 0; j < N; j$+=$\omega$)
    $D$POTF2($\omega, A_{j,j}$);                                  // level-2 LLT fact'n of $A_{j..j'_\omega\ ,j..j'_\omega}$
    $\tilde{T}$= SpreadCopy(COL, $\omega, \omega,\ A_{j,j}$);              // $\tilde{T} \leftarrow A_{j..j'_\omega\ ,j..j'_\omega}$ (row-replicated)
    $\tilde{\tilde{T}}$= SpreadCopy(ROW, $\omega, \omega,\ \tilde{T}$);              // $\tilde{\tilde{T}} \leftarrow \tilde{T}$ (replicated on all cells)
    $L'$= PanelScatter(ROW, $N - j_\omega, \omega,\ A_{j_\omega,j}$);        // $L' \leftarrow A_{j_\omega..N'\ ,j..j'_\omega}$ (on a $PQ \times 1$ grid)
    $D$TRSM("Lower","NonUnit", $\omega, N - j_\omega, 1.0,\ \tilde{\tilde{T}},\ (L')^T$);  // $L' \leftarrow L'\tilde{\tilde{T}}^{-T}$, $\tilde{\tilde{T}}$ is lower tri., non-unit diag.
    $\tilde{L}$= ScatterSpread(ROW, $N - j_\omega, \omega,\ A_{j|\omega,j},\ L'$);   // $\tilde{L} \leftarrow L'$ (column-replicated, on a $P \times Q$ grid)
    $D$SYRK(Lower, $N - j_\omega, \omega,\ -1.0,\ \tilde{L},\ 1.0,\ A_{j_\omega,j_\omega}$);  // $A_{j_\omega..N',\ j_\omega..N'}$ -= $\tilde{L}\tilde{L}^T$ (only lower tri. part of $A$ updated)
    $D$COPY($N - j_\omega, \omega,\ \tilde{L},\ A_{j_\omega,j}$);                 // $A_{j_\omega..N'\ ,j..j'_\omega} \leftarrow \tilde{L}$
    <dispose $L', \tilde{L}, \tilde{T}, \tilde{\tilde{T}}$>;
```

Figure 4: DBLAS-based LLT Factorization algorithm of an $N \times N$ matrix A

```
for($j = 0; j < \min(M, N); j$+=$\omega$)
    $V'$= PanelScatter(ROW, $M - j, \omega,\ A_{j,j}$);            // $V' \leftarrow A_{j..M'\ ,j..j'_\omega}$ (on a $PQ \times 1$ grid)
    $D$GEQR2($M - j, \omega, V', T'$);                            // level-2 QR fact'n of $V'$, with tri. reflector $T'$ st.
                                                           // $T'_{i,0:i'} = -\tau_j (V'_{i..M',i})^T (V'_{i..M',0..i'}), T'_{i,i} = \tau_i$
    $\tilde{V}$= ScatterSpread(ROW, $M - j, \omega,\ A_{j,j},\ V'$);       // $\tilde{V} \leftarrow V'$ (column-replicated, on a $P \times Q$ grid)
    $D$COPY($M - j, \omega,\ \tilde{V},\ A_{j,j}$)                        // $A_{j..M',\ j..j'_\omega} \leftarrow \tilde{V}$
    $\tilde{T}$= ScatterSpread(ROW, $\omega, \omega,\ A_{j,j},\ T'$);          // $\tilde{T} \leftarrow T'$ (column-replicated, on a $P \times Q$ grid)
    FormTriRefl($\omega, \tilde{T}$);                                // $\tilde{T}'_{i,0:i'} \leftarrow \tilde{T}^i \tilde{T}'_{i,0:i'}$, $\tilde{T}^i = \tilde{T}'_{0..i',\ 0:i'}$ is lower tri.
    $\tilde{\tilde{T}}$= SpreadCopy(ROW, $\omega, \omega,\ \tilde{T}$);              // $\tilde{\tilde{T}} \leftarrow \tilde{T}$ (replicated on all cells)
    ClearUpperTri($\omega, \tilde{V}$);                              // $\tilde{V}_{i,i} \leftarrow 1, \tilde{V}_{i,i+1..\omega} \leftarrow 0$
    $W = D$NewMat($\omega, N - j_\omega, A_{j,j_w}$, NULL);             // create new $\omega \times N - j_\omega$ matrix aligned with $A_{j,j_w}$
    $D$GEMM($\omega, N - j_\omega, M - j,\ 1.0, (\tilde{V})^T, A_{j,j_w}, 0.0, W$);   // $W \leftarrow \tilde{V'}^T A_{j..M',\ j_\omega..N'}$
    $W'$ = PanelScatter(COL, $\omega, N - j_\omega, W$);               // $W' \leftarrow W$ (on a $1 \times PQ$ grid)
    $D$TRMM("Lower","NonUnit", $\omega, N - j_\omega, 1.0,\ \tilde{\tilde{T}},\ W'$);   // $W' \leftarrow \tilde{\tilde{T}}W'$, $\tilde{\tilde{T}}$ is lower tri., non-unit diag.
    $\tilde{W}$= ScatterSpread(COL, $\omega, N - j_\omega,\ A_{j,j_\omega},\ W'$);    // $\tilde{W} \leftarrow W'$ (row-replicated, on a $P \times Q$ grid)
    $D$GEMM($M - j, N - j_\omega, \omega,\ -1.0,\ \tilde{V},\ \tilde{W},\ 1.0,\ A_{j,j_\omega}$);  // $A_{j..M',\ j_\omega..N'}$ -= $\tilde{V}\tilde{U}$
    <dispose $V', T', \tilde{V}, W, W', \tilde{W}, \tilde{T}, \tilde{\tilde{T}}$>;
```

Figure 5: DBLAS-based QR Factorization algorithm of an $M \times N$ matrix A

of the V' being padded by zeroes. The alternative would be use the triangular matrices $(V'_{0..\omega',\ :})^T$ and $V'_{0..\omega',\ :}$ to update $A_{j..j'_\omega,\ j_\omega..N'}$, as is done in LAPACK. The former is computationally superior for $\omega = 32$ on the AP+, as $\gamma_3(32) \leq 2\gamma_3^{\triangle}$ (see Section 1.2). Furthermore, it greatly simplifies the algorithm, and reduces software and communication overheads. However, this decision should be re-evaluated on other machines, especially those with large ω.

A new optimization for parallel QR in the formation of the triangular reflector T' is also made here. The vector-matrix multiply to form $T'_{i,0..i'}$ is merged with the vector-matrix multiply with $V'_{i,i..M-j}$ required in the level 2 factorization. This reduces communication startup and software overheads substantially, and also is likely to yield better computation speed.

The above algorithms are developed for the general block-cyclic distribution. If the assumption of PBMD (Equation 1) is made, the only significant implementation difference, potentially affecting performance, is that one can form $\tilde{L}^T$ directly from L' for the same cost as for $\tilde{L}$. In this case, whether L' is oriented row-wise or column-wise is irrelevant[1]. Note that all cells in a column will hold part of L' that will be broadcast to form $\tilde{L}^T$. This is useful in the symmetric rank-k update in LLT [1]. Our implementation instead, internal to the DSYRK() procedure, forms $\tilde{L}^T$ from $\tilde{L}$, with only GCD(P, Q)

[1]Eg. in a PBMD QR implementation, W' could have the same orientation as V' without any extra communication cost, whereas in our implementation, it must have the opposite orientation.

cells holding part of $\tilde{L}$ to be re-broadcast. On machines with unit-cost broadcasting, such as the AP+, there will be no difference between the two ways of forming $\tilde{L}^T$. On other machines, the way exploiting PBMD will have half the communication volume cost for large L' when $\mathrm{GCD}(P, Q) = 1$ [8].

3.1 Reducing Communication Startup Costs for the Triangular Factors

For small block sizes, ie. $r, s < \omega$, $O(N)$ extra communication startup overheads can be introduced in the formation of the triangular factors, eg. D`POTF2`$(A_{j,j})$ and `FormTriRefl`$(\omega, \tilde{T})$. Stanley [10] has shown in that in corresponding parts of the symmetric tridiagonal reduction algorithm, it is possible to reduce communication overheads by performing redundant computations, and has suggested that a similar scheme should be possible for LLT [11].

One way that this could be done is by replicating the factor across all cells, and (redundantly) forming the factor on each, eg. D`POTF2`$(\omega, \tilde{T})$ (or even D`POTF2`$(\omega, \tilde{\tilde{T}})$) and `FormTriRefl`$(\omega, \tilde{\tilde{T}})$. In the case of LLT, $\tilde{T}$ (or $\tilde{\tilde{T}}$) must be copied back into $A_{j..j'_\omega,\, j..j'_\omega}$ after this step.

As the amount of computation in forming these factors is small ($O(\omega^3)$), this will be an optimization on most multicomputers. This means that provided ω is reasonably large compared with P, there need be no significant communication startup overhead penalty in choosing small r, s.

3.2 Implications for Library Design

Matrix factorizations using storage or algorithmic blocking methods can be efficiently coded in terms of parallel BLAS operations, with little or no explicit references to local array, block or grid sizes [4, 14, 16].

It can be seen that this also applies to panel scattering (see also [1]). As compared with algorithmic or storage blocking counterparts, our implementation of panel scattering required only 4–6 extra procedure calls (all very simple) per panel, which makes it relatively easy to implement[2]. As an example, the parallel QR algorithm was coded, tested and debugged, with the results generated, all within a single day.

As panel scattering does not use pipelined communication, it can be more easily implemented on all message passing multicomputers.

Finally, in terms of applicability, panel scattering can be applied in block-partitioned dense linear algebra algorithms (including bi- and tri-diagonal reductions) to improved load balance in the panel formation stage, as can algorithmic blocking.

4 Results

The DBLAS distributed BLAS library has been very highly tuned for the Fujitsu AP+ machines, in terms of computation and communication speed, memory copy overheads and software overheads [14, 16]. Even for storage blocking, it compares favorably with ScaLAPACK codes running on the AP+, where the ScaLAPACK and DBLAS codes use the same BLAS and BLACS (computation and communication) libraries [12]. As the algorithms described in the previous section should be optimal with respect to communication performance, we believe the results of this section represent the full potential of the panel scattering technique on the AP+.

In this section, the performance of crucial components of the `PanelScatter()` routine will be examined, followed by a comparison of panel scattering, algorithmic blocking and storage blocking. In both cases, the effect of varying the block size on performance will be examined.

4.1 Performance of Component Computations

Table 1 indicates the computational speeds of triangular update corresponding to the formation of the upper panel in LU and QR factorizations. The 8×8 grid case corresponds to algorithmic blocking, which runs at approximately half the speed of that of the 1×64 grid, which corresponds to panel scattering. Part of this is due to imperfect load balance by a factor $E_d(32, 1, 8) \approx 0.8$ [13], the rest is due to degraded computation speed due to the local length of U being reduced by a factor of P (ie. reducing it to a 'Level

[2]One can 'recover' the algorithmic or storage blocking versions of these codes simply by removing these calls and replacing the references to $\tilde{\tilde{T}}, \tilde{L}$ etc with the corresponding sub-matrix of A.

$P \times Q$	8×8	1×64
$U \leftarrow \tilde{T}^{-1}U$	5.1 (6.4)	11.7
$U \leftarrow \tilde{T}U$	6.4 (8.0)	11.5

Table 1: Computational speeds in MFLOPs/cell of triangular update operations on a 32×4096 matrix U with $r = s = 1$ on a 64-node AP+

r :	1	2	4	8	16	32
speed (MBs):	36	34	36	29	22	13

Table 2: Speeds in MBs of the panel scattering operation (matrix transpose) on a 512×32 matrix with $r \times r$ blocks on a 1×8 AP+

2.5' rather than a 'Level 3' computation). The resulting effective cell computation speed for algorithmic blocking, denoted $\gamma_{2.5}^{\triangle}$, is indicated in parentheses in Table 1.

The overall speed of the panel scattering operation, essentially a block-cyclic matrix transpose over a row or column of cells, is given in Table 2. This shows a strong dependence on block size. For $r = 32$, the matrix is initially only on one cell, and the transposition bandwidth R_r^T is limited by the memory copy speed R^c (in the packing stage) and the message sending speed R^s in this cell. For $r \leq \frac{Q}{\omega}$, the panel is initially distributed equally across all cells, and thus the bandwidth is limited by the packing and unpacking speeds on an $\frac{1}{Q}$ portion of the matrix, and the saturation bandwidth of the $1 \times Q$ network R^h, ie.:

$$R_r^T = \begin{cases} (\frac{1}{R^c} + \frac{1}{R^s})^{-1} & \text{, if } r \geq \omega \\ (\frac{1}{R^c Q} + \frac{1}{R^h} + \frac{1}{R^c Q})^{-1} & \text{, if } r \leq \frac{\omega}{Q} \end{cases} \qquad (2)$$

Note that the communication startup cost is $(Q-1)\alpha$, independent of r.

On the AP+, R^c = 27MBs, R^s = 25MBs, and as the network is a bi-directional torus, $R^h = 2 \cdot 2 \cdot 25$MBs. At $Q = 8, \omega = 32$, Equation 2 yields R_{32}^T = 13MBs and R_1^T = 52MBs, in reasonable agreement with Table 2. R_1^T is considerably faster than the bandwidth for the replication of the panel, which is $8/\beta$ = 12.5MBs.

4.2 Performance of the Matrix Factorizations

Figures 6, 7, and 8 give a comparison of the actual performance of panel scattering, algorithmic blocking and storage blocking on the Fujitsu AP+. Part (a) of these Figures give the optimal combination of parameters and matrix-storage for each method (with '`r-m`' ('`c-m`') signifying row- (column) major storage). Part(b) gives variations of panel scattering, examining the effect of block sizes in particular.

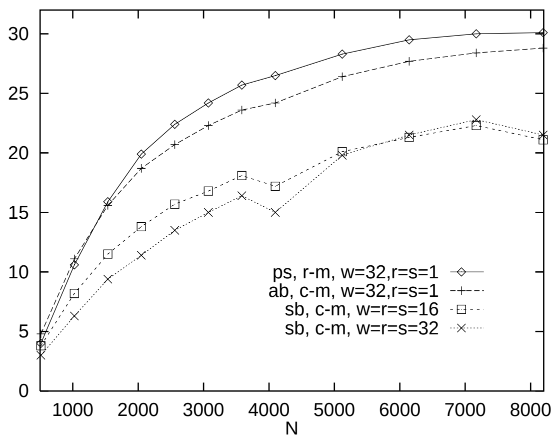
Square processor configurations were chosen, as these have been found to be optimal for such machines; a value of $P = 8$ was the largest available. For such grid sizes, the range $512 \leq N \leq 8192$ was chosen. The lower bound of $N = 512$ is the largest size where software overheads, in a distributed BLAS implementation specifically optimized for this purpose, should have an effect of less than 25% on overall performance [16]. For the AP+ machine used for these experiments, $N = 8192$ represents the limit of available memory.

For all 3 factorizations, the local storage for the scattered panels, eg. L', was unit stride along the panel length: this was slightly faster for $N \geq 3072$, and used for the above plots. However, the difference in performance of the opposite storage was never large enough to have made a discernible difference on the above plots; this is because in either case the panels are packed, which ensures reasonably good memory access performance.

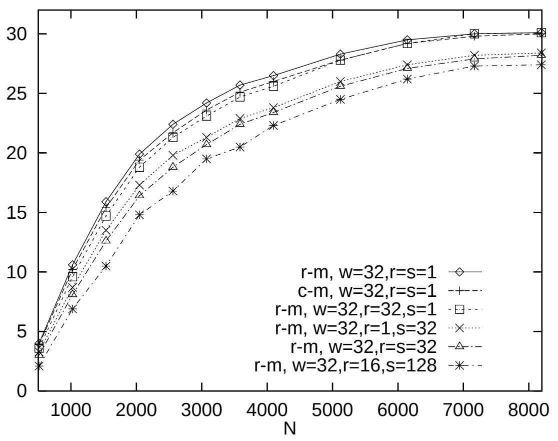
Unlike algorithmic and storage blocking, which strongly favor column-major storage for the main matrix in all cases, this packed storage meant that the performance of panel scattering was relatively independent of the local storage of A. An exception was LU, where row-major storage yields a significant advantage in row swapping, as row packing and unpacking (with a large memory stride) is not required.

In Figure 6(a), this enabled panel scattering to out-perform algorithmic blocking by as much as 10%, in the middle of the range. Elsewhere, in the upper half of the range, panel scattering was slightly faster than algorithmic blocking, partly due to its advantage in panel formation speed, partly due to its better memory access patterns. In the lower half of the range, algorithmic blocking was somewhat faster, especially for LU and QR, due to several reasons. Firstly, there are $O(N)$ communications along the panel, which have double cost for panel scattering. Secondly, for $N < 2048 = \omega PQ$, the local panel length of L' becomes less than its local width ($= \omega$), which effectively means greater loop startup overheads in the computational routines. Also, for $N < rPQ$, load imbalance will occur in the formation of L'. Thirdly, panel scattering has extra memory copying, communication and software overheads, which degraded performance for small N.

Both panel scattering and algorithmic blocking out-performed storage blocking by about 25–30% for moderate to large N.



(a) compared with algorithmic (ab) and storage blocking (sb)



(b) variations of panel scattering

Figure 6: Speed in MFLOPs/cell of panel scattering for LU factorization of an $N \times N$ matrix A on an 8×8 AP+

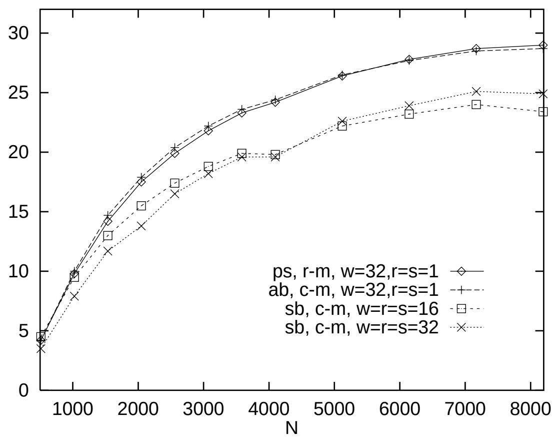
A comparison of the row-major plots in Figure 6(b) gives an indication of effect of parallelization of row-swapping for $r = 1$ and the load imbalance due to block size in the calls to DDGEMM(). For the latter, the mixed block sizes $\{r, s\} = \{1, 32\}$ should be the same as each other, and half that of $r = s = 32$. From this, we can see that the parallelization of row-swapping has a considerably larger impact on performance.

For LLT and QR, load imbalance due to block size takes on a more noticeable effect. For LLT, a processor on the lower left corner of the grid will have an extra row, column and diagonal of blocks than one on the upper right corner, on each call to DSYRK(), which has only half the operations of a corresponding DGEMM(). Hence we would expect this effect to have approximately 3 times the impact as it does for LU. Note that for $r = s = 16$ in Figure 7(b), the load imbalance should be half that for $r = s = 32$.

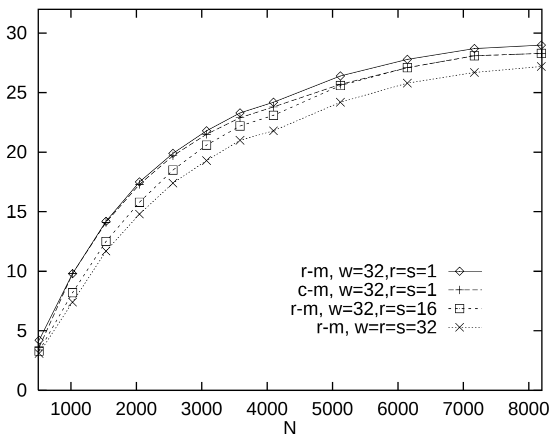
For QR, this load imbalance has also a relatively greater impact than in LU. This is because, for $r \geq \omega$, the processor row 'owning' W normally has an extra block row to multiply, and the subsequent reduction on the first call to DGEMM() means that all cells must wait at that point for this to complete.

The plots for LU and QR with $r = 16, s = 128 = Pr$ correspond to PBMD with $r = \omega/2$, ie. correspond to the choice of block sizes used in [1]. These indicate a considerably larger degree of imbalance overheads than for the $r = s = 32$ plots.

For $r = s = 1$ on LLT, it was found to be slightly faster *not* to replicate the triangular factor before performing the level-2 factorization. This is because of the relatively low overheads of the row and column broadcasts on the AP+. However, for QR, it was faster to fully replicate $\tilde{T}$ before calling FormTriRefl(), by as much as 20% at $N = 512$. This is because the communications in FormTriRefl($\tilde{T}$), being a column transpose and summation, are more expensive on the AP+.



(a) compared with algorithmic (ab) and storage blocking (sb)



(b) variations of panel scattering

Figure 7: Speed in MFLOPs/cell of panel scattering for LLT factorization of an $N \times N$ matrix A on an 8×8 AP+

5 Analysis of the Techniques for LU Factorization

In this section, a detailed performance model will be developed for LU factorization, which can be applied to all of the methods used in the preceding section. This model can also be applied to other machines too. The model will then be validated on AP+.

The model used is an extension of the DLAM model [4], using parameters introduced in Section 1.2. For simplicity of presentation, it will be assumed that $N \gg \omega \gg 1$ and $P-1 \approx P$, and $Q-1 \approx Q$. It will be assumed that ω exceeds P, Q by a factor of 2 or more, so that the $(\frac{3(Q-1)}{\omega} + \frac{2(P-1)}{\omega})N$ communication startups introduced by panel scattering (algorithmic blocking introduces similar overheads) can be neglected when compared with the total startup costs.

As previously mentioned, κ_{bc} is the cost of a broadcast where pipelining can be used; $\kappa'_{bc}(P)$ will denote the cost of a broadcast across P cells where pipelining cannot be used; thus $\kappa'_{bc}(P) = 1$ if unit-cost broadcasts are available (but a value of $\kappa'_{bc}(PQ) \approx 2$ must be used for the AP+), and $\kappa'_{bc}(P) = \lg_2(P)$ otherwise.

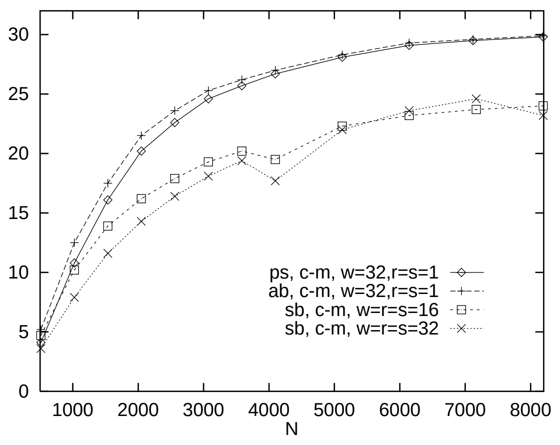
The model will be developed first for the case of storage blocking ($\omega = r = s$); the subscripts for the t variables reflect the steps in Figure 3, with the total time for storage blocking being given by $t^{\mathrm{sb}} = (t_{\mathrm{f2},\alpha}(P) + t_{\mathrm{f2},\gamma_2}(P)) + t_{\mathrm{sw},1} + t_{\triangle}(Q) + t_L + t_U + t_{\mathrm{mm}}$, where:

$$t_{\mathrm{f2},\alpha}(P) = (\lg_2(P) + 2\kappa'_{bc}(P) + 2)N\alpha, \quad t_{\mathrm{f2},\gamma_2}(P) = \tfrac{N\omega}{2} r \lceil \tfrac{N}{Pr} \rceil \gamma_2, \quad t_{\mathrm{sw},1} = 4N\alpha + \tfrac{N^2}{Q}\beta'$$
$$t_{\triangle}(Q) = \tfrac{N\omega}{2} s \lceil \tfrac{N}{Qs} \rceil \gamma_3^{\triangle}, \quad t_L = \kappa_{bc}\tfrac{N^2}{2P}\beta, \quad t_U = \kappa'_{bc}(Q)\tfrac{N^2}{2Q}\beta, \quad t_{\mathrm{mm}} = \tfrac{2N^3}{3PQ}\gamma_3 + \tfrac{N^2}{2}(\tfrac{r}{Q} + \tfrac{s}{P})\gamma_3$$

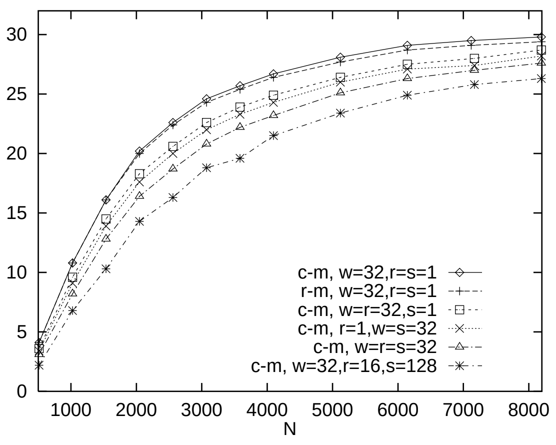
Note that β' is β modified to take into account 2 extra memory copies if column major storage is used, and that the last term in t_{mm} is the load imbalance term due to the block-cyclic distribution block sizes. The terms $t_{\mathrm{f2},\gamma_2}(P)$ and $t_{\triangle}(Q)$ partially take into account load imbalances when the number of blocks along a panel is of comparable size (or less) than the number of processors along that dimension. The coefficient $\kappa'_{bc}(Q)$ for t_U assumes that a tree broadcast is used, as does [4, 3]; it is a little pessimistic for large N, as by using other methods, it could be reduced to 2 [8].

For algorithmic blocking, the following terms are changed:

$$t^{\mathrm{ab}}_{\mathrm{f2},\gamma_2} = t_{\mathrm{f2},\gamma_2}(P)/(E_d(\omega, s, Q)Q), \quad t^{\mathrm{ab}}_L = (\kappa_{bc}+1)N\alpha + t_L + \tfrac{N^2}{2P}\beta$$
$$t^{\mathrm{ab}}_{\triangle} = \tfrac{N\omega}{2} s \lceil \tfrac{N}{Qs} \rceil \gamma_{2.5}^{\triangle}/(E_d(\omega, r, P)P) + \kappa_{bc}N\alpha, \quad t^{\mathrm{ab}}_U = \kappa_{bc}\tfrac{N^2}{2Q}\beta$$



(a) compared with algorithmic (ab) and storage blocking (sb)



(b) variations of panel scattering

Figure 8: Speed in MFLOPs/cell of panel scattering for QR factorization of an $N \times N$ matrix A on an 8×8 AP+

Note that the upper panel U can be broadcast row by row using a ring broadcast in the call to DTRSM(), and $E_d(\omega, r, P) \in [0, 1]$ is the load balance efficiency factor of a triangular update of width ω with a block size r distributed over P cells [13].

The first term in t_L^{ab} is due to the broadcast of the pivot p_j followed by that of the column $A_{j+1..M',\,j}$ (in DGETF2()). As these originate from the same cell column, the pipeline 'bubble' from p_j is hidden by that of $A_{j+1..M',\,j}$ for the case of $\kappa_{bc} = 2$. The third term is due to L having to be re-broadcast in DGEMM(); provided that $\omega \geq rQ$ this can be achieved most efficiently by $Q - 1$ ring shift operations of messages of local size $\frac{N}{P}\lceil\frac{\omega}{Q}\rceil$ [8].

Thus, the execution time for algorithmic blocking can be modelled by: $t^{\mathrm{ab}} = (t_{\mathrm{f2},\alpha}(P) + t^{ab}_{\mathrm{f2},\gamma_2}(P)) + t_{\mathrm{sw},\omega/\mathrm{r}} + t^{\mathrm{ab}}_{\triangle} + t^{\mathrm{ab}}_L + t^{\mathrm{ab}}_U + t_{\mathrm{mm}}$, where the modified row swap term for $r < \omega$ is:

$$t_{\mathrm{sw},\omega/r} = t_{\mathrm{sw},1}/\Psi_{\beta',\omega,r,P}$$

Here, note that $\Psi_{\beta',\omega,r,P} \geq 1$ is the degree of parallelization possible with ω row swaps with the row indices equally distributed over $\min(\omega/r, P)$ processors, taking into account that the effective communication cost β' is generally several times that due only to the hardware. The estimation of $\Psi_{\beta',\omega/r,P}$ is complicated, but is based on similar arguments to that for R_r^T given previously. For the AP+, the value of $\Psi_{\beta',32,1,8} = 2.3$ is consistent with experiments.

If panel scattering is used, the terms for the communication of L and U must be modified:

$$t_L^{\mathrm{ps}} = \frac{N^2}{2P}\beta(\frac{1}{R_r^T} + 1); \quad t_U^{\mathrm{ps}} = \frac{N^2}{2Q}\beta(\frac{1}{R_s^T} + 1)$$

noting again that the ring-shifts can be used to broadcast the panels here also. Hence the total time for panel scattering can be given by $t^{\mathrm{ps}} = (t_{\mathrm{f2},\alpha}(PQ) + t_{\mathrm{f2},\gamma_2}(PQ)) + t_{\mathrm{sw},\omega/r} + t_{\triangle}(PQ) + t_L^{\mathrm{ps}} + t_U^{\mathrm{ps}} + t_{\mathrm{mm}}$.

The above performance model (without most of the simplifying assumptions used above, has been implemented in a computer program; the % difference between the model and the actual is given in Table 3. Here, a selection of values of N likely to have smaller cache miss effects was chosen.

For $N \leq 1024$, the bulk of the error is due to software overheads, which are not incorporated in this model (although it is possible to do so [10]). For $N > 2048$, the model accurately, ie. within 5%, reflects the actual performance for all but storage blocking. This is because the model does not take fully into account the cache and TLB misses in calls to DLASWP() and to a lesser extent DTRSM(), for which column-major storage with storage blocking presents the worst case situation.

The model is less accurate for panel scattering for $N < 2048$, mainly due to increased software (eg. loop startup) overheads, as mentioned in Section 4. For large r, performance is degraded by load imbalance

N	512	1024	1536	2560	3584	5120	7168
`ab`, `c-m`, $\omega = 32, r = s = 1$	+25	+9	+3	+2	+1	+0	−1
`sb`, `c-m`, $\omega = r = s = 32$	+23	+22	+15	+13	+12	+7	+5
`ps`, `c-m`, $\omega = 32, r = s = 1$	+28	+15	+9	+3	+1	+0	−2
`ps`, `r-m`, $\omega = 32, r = s = 1$	+35	+17	+10	+3	+1	−1	−1
`ps`, `r-m`, $\omega = 32, r = s = 32$	+30	+19	+4	+1	+0	−3	−2

Table 3: Percentage error between the performance model and the actual LU computation on an 8×8 AP+

	LU		LLT		QR	
	AP+	$\kappa'_{bc} = 6$	AP+	$\kappa'_{bc} = 6$	AP+	$\kappa'_{bc} = 6$
s-b, $\omega = r = s$	$11N$	$15N$	0	0	$8N$	$12N$
a-b,r=s=1	$12N$	$18N$	N	$2N$	$10N$	$15N$
p-s,r=s=1	$13N$	$22N$	0	0	$16N$	$24N$

Table 4: Dominant communication startup terms for LU, LLT and QR factorizations for storage blocking, algorithmic blocking and panel scattering, for $P = Q = 8$ and assuming $\omega \gg \max(P, Q)$ and $\Psi(\omega, 1, P) = 2$.

in the panel formation. For example, at $N = 1024, r = 32$, only half the processors can hold a block of the scattered panel.

At $N = 512$, the model predicts that communication startup costs account for 21% (.074s), 37% (.084s) and 43% (.11s) of the total execution time for storage blocking, algorithmic blocking, and panel scattering, respectively, on the Fujitsu AP+. For multiprocessors with higher α/γ_3, the differences in communication startup costs for the three methods will have a larger impact on total performance, especially since for most of these, $\kappa'_{bc} > 1$.

Table 4 gives the dominant startup coefficients for these methods, applying the above model for LU, and a using similar analysis for LLT and QR (for QR, the startups are due to reductions/broadcasts when forming V; using the optimization described in Section 3, there need only be two of these). For $r = s = 1$, it is assumed that the triangular factors for LLT and QR are fully replicated, as explained in Section 3.1, which minimizes this term. Algorithmic blocking introduces 'across-panel' communications, which can be pipelined to reduce this term. Panel-scattering, on the other hand, tends to amplify the 'along-panel' communications, by a factor of approximately 2 for either the AP+ or other machines with $\kappa'_{bc}(P) = \lg_2 P$. For LU and QR, there are more 'along-panel' communications, which means that panel-scattering will have slightly higher overheads.

6 Conclusions

Panel scattering is general technique for dense parallel linear algebra algorithms that can achieve maximal load balance and cell computation speed in most situations. We have shown that it can be easily implemented using the existing building blocks of a parallel BLAS library for the block-cyclic distribution, with a reasonably concise and elegant expression of the matrix factorization algorithms possible. However, panel scattering has extra communication startup and software overheads which somewhat reduce its performance for small N.

While the scattering of the panels appears to incur extra communication volume costs, this can be relatively small compared with the panel broadcasts. For sufficiently large N, this cost can be regained (on machines without unit-cost row and column broadcasts) from a faster panel broadcast, since here, as opposed to storage blocking, all cells hold part of the panel to be broadcast (this applies in most situations to algorithmic blocking also).

However, a rather surprising performance advantage of panel scattering lies in its natural optimization of memory access patterns, due to the packing of the panels. While such effects are hard to estimate, they are likely to be significant on most modern memory-hierarchy multiprocessors, which also have virtual memory. This means that there was little or negative cost in using row-major storage for LLT and QR factorization, which runs against conventional wisdom. Furthermore, for LU, panel scattering's ability to use row-major storage gave it a decisive advantage in row swapping speed over algorithmic and storage blocking.

To achieve the best load balance, a minimal block size of $r, s \approx 1$ should be chosen. This also achieves the best communication bandwidth for the panel scattering operation. Furthermore, we have shown that with full replication of the triangular factors in LLT and QR, the minimal block size introduces no significant communication startup costs. Finally, $r = 1$ enables a limited parallelization of row swapping in LU, which will in most situations reduce its communication startup (and volume) overheads by a factor of at least 2. Choosing a minimal r is thus especially important for PBMD, where the block width is constrained by $s = Pr$.

Compared with algorithmic blocking, which also can achieve high load balance, panel scattering will normally have greater startup overheads, except in the case of LLT. In the event, this was true even on the AP+, which has unit-cost for row and column broadcasts, but not for global broadcasts. However, this would not apply to a machine which could perform a global broadcast for the same cost as a row or column broadcast. For $\omega > 2\max(P, Q)$, both have sufficiently good load balance and their performance will be very similar, and likely to be significantly better than that of storage blocking. For $\omega \leq 2\max(P, Q)$, panel scattering should have an advantage in load balance.

Acknowledgements

The author would like to thank the Fujitsu Parallel Research Computing Facilities for the use of the 64 node AP+, which was used to generate the results of this paper.

References

[1] G. Baker, J. Gummels, G. Morrow, B. Riviere, and R. van de Geijin. PLAPACK: High Performance through High Level Abstraction. To appear in ICPP98, available from http://www.cs.utexas.edu/users/plapack/new/pubs.html, February 1998.

[2] L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, J. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. *ScaLAPACK User's Guide*. SIAM Press, Philadelphia, 1997.

[3] J. Bolen, A. Davis, W.Dazey, S.Gupta, G. Henry, D. Robboy, G Sciffler, D. Scott, M. Stallcup, A. Taragi, S. Wheat, L. Fisk, G.Istrail, C.Jong, R. Riesen, and L. Shuler. Massively Parallel Distributed Computing: World's First 281 Gigaflop Supercomputer. In *Proceedings of the Intel Supercomputer Users Group*, 1995.

[4] J. Choi, J. Demmel, J. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performance. In *Applied Parallel Computing*, pages 95–106, Berlin, 1995. Springer-Verlag.

[5] J. Choi, J. J. Dongarra, S. Ostrouchov, A. P. Petitet, D.W. Walker, and R.C. Whaley. The Design and Implementation of the ScaLAPACK LU, QR and Cholesky Factorization Routines. *Scientific Programming*, 5:173–184, 1996.

[6] B. A. Hendrickson and D. E. Womble. The Torus-Wrap Mapping for Dense Matrix Calculations on Massively Parallel Computers. *SIAM J. Sci. Stat. Comput.*, 15(5):1201–1226, 1994.

[7] T. Horie, K. Hayashi, T. Shimizu, and H. Ishihata. Improving AP1000 Parallel Computer Performance with Message Communication. In *International Symposium of Computer Architecture (ISCA '93)*, 1993.

[8] P. Mitra, D. Payne, R. van de Geijn L. Shuler, and J. Watts. Fast Collective Communication Libraries, Please. In *Proceedings of the Intel Supercomputing Users' Group*, 1995.

[9] A. Petitet. *Algorithmic Redistribution Methods for Block Cyclic Decompositions.* PhD thesis, University of Tennessee, Knoxville, December 1996. xv+193p.

[10] K. Stanley. *Execution Time of Symmetric Eigensolvers.* PhD thesis, University of California, Berkeley, 1997. viii+184p.

[11] K. Stanley. Private comunications, February 1998.

[12] P. Strazdins. Software Overhead and Blocking Issues in Parallel BLAS. Presented to the ScaLAPACK Conference, available from http://cs.anu.edu.au/ Peter.Strazdins/projects/DBLAS/PBSWOv.ps.gz, March 1998.

[13] P.E. Strazdins. Matrix Factorization using Distributed Panels on the Fujitsu AP1000. In *IEEE First International Conference on Algorithms And Architectures for Parallel Processing (ICA3PP-95)*, pages 263–73, Brisbane, April 1995.

[14] P.E. Strazdins. A High Performance, Portable Distributed BLAS Implementation. In *Sixth Parallel Computing Workshop*, pages P2–K–1 – P2–K–10, Kawasaki, November 1996. Fujitsu Parallel Computing Research Center.

[15] P.E. Strazdins. Load Balance and Communication Tradeoffs in Parallel Matrix Factorization. In *Seventh International Parallel Computing Workshop*, pages P1–Q–1 – P1–Q–5, Canberra, September 1997. Australian National University.

[16] P.E. Strazdins. Reducing Software Overheads in Parallel Linear Algebra Libraries. In *The 4th Annual Australasian Conference on Parallel And Real-Time Systems*, pages 73–84, Newcastle Australia, September 1997. Springer.

[17] P.E. Strazdins and H. Koesmarno. A High Performance Version of Parallel LAPACK: Preliminary Report. In *Sixth Parallel Computing Workshop*, pages P2–J–1 – P2–J–8, Kawasaki, November 1996. Fujitsu Parallel Computing Research Center.

[18] R. van de Geijin. *Using PLAPACK: Parallel Linear Algebra Package.* MIT Press, Boston, 1997.