Matrix Factorization using Distributed Panels on the Fujitsu AP1000

Peter Strazdins,

Department of Computer Science,
Australian National University,
Acton, ACT 0200. AUSTRALIA

E-mail: Peter.Strazdins@anu.edu.au

Abstract

Dense linear algebra computations such as matrix factorization require the technique of 'block-partitioned algorithms' for their efficient implementation on memory-hierarchy processors. For scalar-based distributed memory multiprocessors, the register, cache and off-processor memory levels of the memory hierarchy all affect the optimal block-partition size for such algorithms. Most studies on matrix factorization and similar algorithms have assumed that the block-partition size or panel width for the algorithm, ω, to be the same as the matrix distribution block size, r, where a rectangular block-cyclic matrix distribution is being employed. Here the choice of $\omega = r$ is essentially determined by the off-processor memory level of the memory hierarchy, with the value of ω being a tradeoff between communication startup overhead and load balance considerations.

In this paper, we re-examine this assumption in the context of LU and Cholesky factorization of block-cyclic distributed matrices on scalar-based distributed memory multiprocessors, such as the Fujitsu AP1000. Here considerations of the register and cache levels of the hierarchy require a large ω. We find that the choice of ω, given $\omega = r$, leads to a tradeoff between load balance and optimal use of register and cache levels of the hierarchy (rather than communication startup), and that this tradeoff substantially limits performance. We then briefly describe 'distributed panels' versions of these algorithms, where generally $\omega > r$, which effectively diminishes this tradeoff to an $O(\omega/N)$ fraction of the overall computation, where N is the matrix size. Two variants of these versions, one with single rows/columns being communicated, and one with single block rows/columns being communicated, are analyzed for their load balance properties.

The results of the distributed panels versions of the algorithms on the scalar-based distributed memory multiprocessor the Fujitsu AP1000 are given, which give significantly superior performance for distributed panels versions over the $\omega = r$ versions, with optimum performance achieved for $r \approx 1$.

1 Introduction

Dense linear algebra computations such as matrix 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. This is exemplified by LAPACK [6] which is an extensive set of linear algebra routines designed for portability and performance on memory hierarchy processors, vector processors and shared memory parallel processors. Performance and portability is obtained by expressing as much as possible of the computations in terms of basic vector-vector, matrix-vector or matrix-matrix operations, in this case the Level 1, 2 and 3 Basic Linear Algebra Subroutines (BLAS) [8], with preference to the higher levels of the BLAS where possible.

A major issue in porting LAPACK to distributed memory multiprocessors is the distribution of matrices across processors (cells). In practice, an adequate solution to this issue is to use the block-cyclic matrix distribution [13, 4, 5, 3], where a $M \times N$ global matrix A is made up of $r \times s$ blocks which are distributed over a $P \times Q$ processor array. Here, if block $(0,0)$ of A is on processor cell (p_0, q_0), then block (i,j) of A will be on cell $((p_0+i) \bmod P, (q_0+i) \bmod Q)$, in this distribution. The restriction of $r = s$, the square block cyclic

0-7803-2018-2/95/$4.00 © 1995 IEEE

(SBS) distribution, is generally adequate for this purpose.

Where the $(r \times r)$ SBS matrix distribution has been used for such algorithms, the $\omega = r$ assumption has been used [13, 4, 14, 3]. Some exceptions include [1, 10], where for the Fujitsu AP1000 Array Processor, the scattered matrix distribution $(r = 1)$ only was considered; here a (distributed) panel width of $\omega \approx 100$ was considered optimal. However, there have been few systematic studies to determine whether there is a significant advantage for the $r > 1$ distribution, assuming $\omega \geq r$.

For vector-based distributed memory multiprocessors with relatively high communication startup (relative to computation speed), a small block size of $r \approx 5$ is adequate [13, 5]. This is largely because vector-based cells have a very high memory bandwidth, so that a low 'Level 3' factor $\omega = r$ is adequate for near-peak performance. A set of distributed basic matrix-vector and matrix-matrix operations called the Parallel Block BLAS (PB-BLAS) [5], has been developed for this purpose: here at least one of the matrix operands of a PB-BLAS operation must be contained within a single cell row/column. The choice of r then is a tradeoff between obtaining a high 'Level 3' factor, which can reduce message startup costs by a factor of r (and also to get better cell computation performance) and minimizing the load imbalance introduced where only a row or column of cells can be used for a $O(r/N)$ fraction of the overall computation.

However, scalar-based cells have typically a much lower memory bandwidth, compensated for by at least one level of cache. Large caches, however, requires ω to be of the order of the square root of the cache size for near-peak cell performance [1, 12, 10, 2].

The Fujitsu AP1000 [7] is a scalar-based distributed memory multiprocessor with a relatively high communication to computation ratio and a low communication startup overhead. It can be configured into a $P \times Q$ (toroidal) configuration, provided PQ does not exceed the total number of cells. The AP1000 cells are SPARC processors with a 25 MHz clock cycle; they have a peak theoretical speed of 5.5 MFLOPS (double precision) and 8.3 MFLOPS (single precision). The cells have a 16 MB memory with a single-level 128 KB direct-mapped copy-back cache; and each cell has 16 double-precision floating point registers.

The AP1000 also has efficient row and column broadcast operations using wormhole routing on its torus network. These operations can be used for the majority of the communication for the matrix factorization algorithms. These broadcast operations can achieve an overall throughput of 6 MBs^{-1}, which is only marginally lower than the point-to-point message throughput. Communication startup times for these operations are relatively low (approximately $15\mu s$), since they do not require interrupts.

The Cray T3D [9] is also a scalar based distributed memory multiprocessor, with cells capable of 150 MFLOPS (double precision) and having an 8K direct-mapped cache. For point-to-point communication, startup overhead has been measured at 2.3 μs, with a sustained transfer rate of 57 MBs^{-1}. Although row and column broadcast are not supported directly by its hardware, the much faster Cray T3D has thus comparable (ie. within a factor of 5) communication performance relative to floating point speed to the AP1000.

For architectures such as the AP1000, a more efficient approach to matrix factorization algorithms can be made by using *distributed panels*, where $\omega > r$ means that the row/column panels for the algorithm are distributed over $p_{\omega,r} = \min(\lceil \omega/r \rceil, P)$ cell row/columns, if an $r \times r$ SBS distribution is being used. This can potentially improve load balance by a factor of up to $p_{\omega,r}$, while retaining the factor of r reduction in communication startup, and enabling ω to be chosen to give optimum cell computation performance. Furthermore, ω can vary dynamically throughout the computation, if desired.

The basic operation for matrix factorization using distributed panels will then be a fully distributed version of the BLAS kernels for the block-cyclic matrix distribution, ie. one where all matrix operands can be distributed over several cell rows and columns. Such a library, called the DBLAS, has been developed for the Fujitsu AP1000 [11].

The remainder of this paper is organized as follows. Section 2 gives basic notations for block-cyclic distributed matrices. Considering the register and cache levels of the memory hierarchy, Section 3 indicates the optimum values of ω for scalar-based distributed-memory multiprocessors, with a specific value being determined for the AP1000. Section 4 gives a brief description of the implementation and performance of the basic distributed (DBLAS) operations required by matrix factorizations. The block-partitioned matrix factorization algorithms are described in Section 5, with the results for their implementation on the Fujitsu AP1000 being given in Section 6, for both distributed and non-distributed panels. Section

7 gives an analysis of the load balance benefits of using distributed Panels and conclusions are given in Section 8.

2 Notations for Block-Cyclic Distributed Matrices

For the purpose of this paper, a global $M \times N$ global matrix A made up of $r \times s$ blocks is distributed over a $P \times Q$ processor array. Block $(0,0)$ of A can be on processor cell (p_0, q_0) where $0 \leq p_0 < P, 0 \leq q_0 < Q$. Furthermore, block $(0,0)$ can also have block offset (r_0, s_0), where $0 \leq r_0 < r, 0 \leq s_0 < s$. This generality is necessary if A is the sub-matrix at global position of (i,j) of A', where A' has block $(0,0)$ at cell $(0,0)$ with block offset $(0,0)$. For a distributed panels algorithm with $\omega > r$, operations on sub-matrices with non-zero block offsets will occur. In this case the combined block-cell offset is given by:

$$(R_0, S_0) = (rp_0 + r_0, sq_0 + s_0) = (i \bmod rP, j \bmod sQ)$$

As the view of the local matrices and indices can be very complex (consider the case of triangular matrices) for block-scattered storage, it is important to express (and implement) algorithms on these matrices in terms of global indices as much as possible; thus global to local index conversion will be an important operation. For a dimension of a matrix distributed across P cells with block size r and block-cell offset $R_0 = rp_0 + r_0$, the global index i on cell p has the local index given by the conversion function:

$$\gamma^p_{r,P}(i, R_0) = r\lceil \frac{\lfloor i'/r \rfloor P - p'}{P} \rceil + i' \bmod r - r_0 \quad (1)$$

where $i' = i + r_0$ if $p = p_0$ and $i' = i$ otherwise; and $p' = (P + p - p_0) \bmod P$.

3 Optimal Panel Widths for Scalar-based Cells

In this section, we examine the impact of the register/cache memory levels of the memory hierarchy have for choosing the optimal panel width for distributed matrix factorization.

For large matrices, a cell matrix-matrix multiply such as $C \leftarrow C - A^T B$, where A is $\omega \times m$ and B is $\omega \times n$, dominates distributed block-partitioned matrix factorization performance[1] (see Figure 4). Here, typically $\omega \ll m, n$ and, without loss of generality, $m \geq n$ can be assumed.

For scalar-based cells with R registers, an optimal matrix-matrix multiply algorithm [12] has a ratio of floating point operations (FLOPs) to load/store operations of:

$$\rho_{\mathrm{R}} = (1/\omega + 0.5/\mathrm{R}_m + 0.5/\mathrm{R}_n)^{-1}$$

assuming A, B both fit within the cache. Here, the order of the terms on the right-hand side are the contributions of the matrices C, A and B respectively, with $\mathrm{R}_m\mathrm{R}_n + \min(\mathrm{R}_m, \mathrm{R}_n) + 1 \leq \mathrm{R}$. With $\mathrm{R} = 16$ being typical for state-of-the-art RISC scalar processors, $\omega \geq 30$ is required to achieve 90% of the maximum value of ρ_{R}.

For scalar-based cells with a (direct-mapped) cache of size C, an optimal partitioning method [12] for matrix-matrix multiply, assuming $m\omega + n\omega \gg \mathrm{C}$, has a ratio of floating point operations to cache misses of:

$$\rho_{\mathrm{C}} = (1/\omega + 0.5/n + \omega/\mathrm{C})^{-1} \quad (2)$$

which is maximized for $\omega \approx \sqrt{\mathrm{C}}$.

For the Fujitsu AP1000, $\mathrm{R} = 16$ and $\mathrm{C} = 2^{14}$ (double precision). If $\omega \geq 4$, $\mathrm{R}_m = 4$ and $\mathrm{R}_n = 2$ are chosen; otherwise, a rank-1 update based matrix-matrix multiply is optimal. Figure 1 gives the AP1000 cell performance of matrix-matrix multiply for moderate matrix sizes, ie. $m = n = 88$, as a function of ω. Here A, B and, for $\omega < 64$, C are initially in cache. Optimal performance here requires $\omega \geq 32$ (a direct-mapped cache conflict between the C and A matrices slightly reduces performance for $\omega \geq 64$ here). For large matrix sizes, ie. $n = 512$, performance peaks at 4.75 MFLOPS at $\omega = 88$ (cf. Equation 2), with performance not exceeding 4.66 MFLOPS for $32 \leq \omega \leq 64$.

This means that values of ω in the range $32 \leq \omega \leq 88$ will also be required for optimal performance on distributed matrix factorization algorithms. For the AP1000 cells, we choose the optimum panel width for cell performance, based on the register and cache levels of the memory hierarchy, to be:

$$\omega_{\mathrm{cell}} = (\mathrm{C}/2)^{1/2} = 88 \quad (3)$$

[1] $C \leftarrow C - A^T B$ is chosen over $C \leftarrow C - AB$ since in our distributed implementation, it is convenient to have uniform cell storage conventions for both matrix operands, noting that the operands have to be packed for message sending in any case.

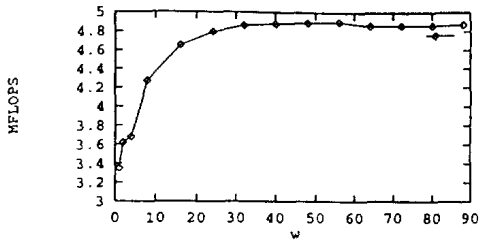


Figure 1: Performance of $C \leftarrow C - A^T B$ where A, B are $\omega \times 88$ on a single cell AP1000 (double precision)

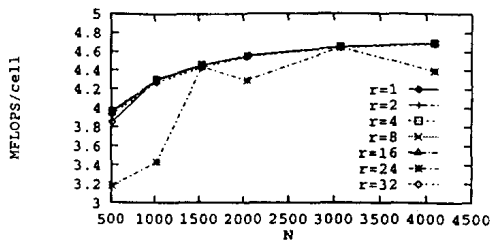


Figure 2: Performance (double precision) of $C \leftarrow C - AB$, where A^T, B are $N \times \omega_{\text{cell}}$ distributed $r \times r$ block cyclic matrices on an 8×8 AP1000

Similar results hold for the cell matrix-matrix multiply where C is a symmetric matrix (where only an upper or lower triangular matrix is computed), the dominant cell computation of Cholesky factorization, and for an $n \times \omega \times n$ multiply (for large n), which occurs in the QR factorization.

4 Implementation of the Basic Distributed Block Operations

For the AP1000, a distributed version of the BLAS Level 2 and 3 library, called the DBLAS [11], has been developed for the block-cyclic matrix distribution. These can fully support all of the distributed matrix operations required for the matrix factorization algorithms, and provide (almost) all of the computation and communication. The algorithms used for the block-cyclic Level 3 DBLAS are generalizations of those described for the scattered matrix distribution in [2, 12]. However, any communication startup reduction afforded by $r, s > 1$ is exploited (where fruitful); furthermore, if any matrix operand of a DBLAS procedure is a (part of) a row or column of blocks, communication across the orthogonal cell dimension is avoided where possible.

It is important that these DBLAS operations are highly optimized, both in communication and computation, so that the performance of the matrix factorization algorithms are not biased by the deficiencies in their implementation. For the same reason, it is also important that these operations' performance do not have a significant bias towards particular values of r (that is avoidable). Assuming that the matrix factorization algorithms will be using an panel width of ω_{cell} (see Equation 3), we will now examine the performance of some of the major component (DBLAS) operations of the matrix factorization algorithms. Where possible, the global matrix sizes are chosen so that each cell holds an equally sized portion of the result matrix of the computation. An 8×8 cell configuration will be used.

The dominant computation in blocked matrix factorization is a matrix-matrix multiply, eg. an $N \times N$ distributed matrix multiply $C \leftarrow C - AB$ (where A is $N \times \omega_{\text{cell}}$). Here, the number of messages broadcast is independent of r. Figure 2 gives the cell performance of this operation as a function of N and r. Low values of r have a minor advantage in terms of load balance, since each cell column (row) packs its $\gamma_{r,8}$ columns (rows) of A (B) in parallel, in preparation for broadcasting. As can be seen, the performance is nearly independent of r; only the case $r = 24$ is significantly worse; this is because $rP \not| N$, so that C is not distributed equally on all cells. The performance is only slightly less on $P \times Q$ configurations provided $2\min(P, Q) \geq \max(P, Q)$.

The next most dominant computation of matrix factorization to update an $\omega \times N$ matrix (panel) by an $\omega \times \omega$ triangular matrix, eg. $U \leftarrow T^{-1}U$. Computationally, this equivalent to a series of ω rank-1 updates (possibly combined with row or column scaling). Figure 3 gives two variants of this Level 3 operation, where the operations' internal panel width ω_1 is set to 1 (this corresponds to forming the $N \times \omega$ panel L in LU decompo-

sition, see Figure 4, where partial pivoting forces the computation to be done as separate rank-1 updates), or is set at an optimal value, in this case $\omega_1 = 12(\approx \sqrt{\omega_{\text{cell}}})$ (this corresponds to forming the $\omega \times N$ panel U in LU factorization). For the latter case, block columns of T are communicated; however, it was found to be more efficient to communicate the rows of U separately, as soon as they were updated.

In either case, load balance decreases with the block size r; in Section 7, we will give an analysis for the $\omega_1 = 1$ case. The balance for the other case is worse, as the internal blocking of the algorithm breaks the computation into $\omega_1 \times N$ and $(\omega - \omega_1) \times N$ pieces; even so, the increased cell performance this affords means that this case is still significantly faster for low values of r, ie. $r = 1$.

5 Description of Matrix Factorization Algorithms

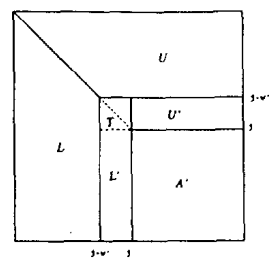
In this section, the block-partitioned matrix factorization algorithms are described. These are essentially simplified versions of those found in LAPACK [6]. The parallelization of these algorithms is entirely within the basic operations such as parallel matrix multiply and triangular matrix updates (see Section 4). Furthermore the algorithm description is independent of the matrix distribution block size r. Hence, the distributed and non-distributed panels versions share the same description: only the relative values of r and ω distinguish them.

The distributed versions of these algorithms exercise a wide range of communication features of a distributed memory multiprocessor; hence collectively they represent distributed dense linear algebra algorithms, in the computational sense.

For the distributed panels versions, $\omega \approx \min(\omega_{\text{cell}}, \sqrt{(N-j)})$ was chosen; however, this value was adjusted so that $r|\omega$ (to reduce communication startup) and, for small r, $rP|\omega$ (to improve load balance), provided that these adjustments did not change ω by more than 25%.

5.1 LU Factorization

Figure 4 gives the indicates LU factorization with partial pivoting and a panel of width ω beginning at position j, $0 \leq j < N$, for an $N \times N$ matrix A. Given the sub-matrix definition in Figure 4, the partial factorization can be expressed by the algorithm:



$l_i = A_{i+1..N',\, i}$
$u_i = A_{i,\, i+1..j}$
$L_i = A_{i+1..N',\, i+1..j}$

$T = A_{j-\omega'..j,\, j-\omega'..j}$
$L' = A_{j+1..N',\, j-\omega'..j}$
$U' = A_{j-\omega'..N,\, j+1..N'}$
$A' = A_{j+1..N',\, j+1..N'}$

$N' = N-1, \omega' = \omega-1$

(T is a lower triangular matrix with a unit diagonal)

Figure 4: Partial LU Factorization of an $N \times N$ matrix A

for $i = j-\omega'..j$ /* form L */
  select $k : i..N'$ s.t. $|A_{k,i}| \geq |A_{l,k}|$ for $l = i, .., N'$
  $A_{i,:} \leftrightarrow A_{k,:}$
  $l_i \leftarrow l_i / A_{i,i}$
  $L_i \leftarrow L_i - l_i u_i$
$U' \leftarrow T^{-1} U'$
$A' \leftarrow A' - L'U'$

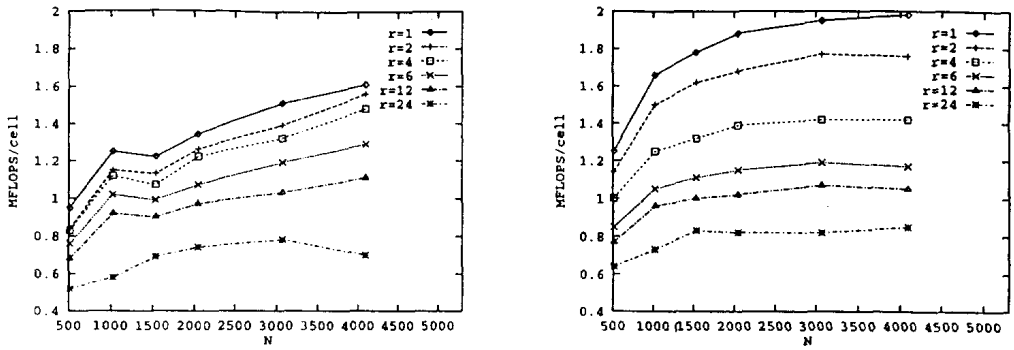
The factorization will be complete with the lower and upper triangular matrices L and U overwriting A, where $LU = PA$ for some permutation matrix P, when $j = N-1$. Here ω can be a function of N and, if desired, j.

If $\omega \leq r$, the panel L' (U') is stored in one cell column (row); otherwise L' and U' are distributed over proper cell sub-arrays. While the formation of L' and U' form only an $O(\omega/N)$ fraction of the overall computation, the communication delay caused by pivoting, the imperfect load balance and fine granularity of the required distributed operations means that this part of the computation can play a significant role in the computation's overall efficiency, even for large matrices.

5.2 Cholesky Factorization

A symmetric positive definite $N \times N$ matrix A can be represented as a lower triangular matrix. It has the factorization $A = LL^T$ where L is lower triangular matrix with a non-unit diagonal. The partial Cholesky factorization of A with a panel of width ω can be visualized by the considering the lower triangular portion only of Figure 4:

for $i = j-\omega'..j$
  $l_i \leftarrow l_i / \sqrt{A_{i,i}}$
  $L_i \leftarrow L_i - l_i l_i^T$



(a) $\omega_1 = 1$ (b) $\omega_1 = 12$

Figure 3: Performance (double precision) of $U \leftarrow T^{-1}U$, where U is an $\omega_{\text{cell}} \times N$ distributed $r \times r$ block-cyclic matrix, T is lower unit triangular, with internal panel width ω_1, on an 8×8 AP1000

$L' \leftarrow L'T^{-T}$
$A' \leftarrow A' - L'L'^{T}$

Here the definitions of the sub-matrices are the same as in Figure 4, except:

$$\begin{aligned} l_i &= A_{i+1..j,\,i} \\ L_i &= A_{i+1..j,\,i+1..j} \end{aligned}$$

noting that now L_i and A' are symmetric and T has a non-unit diagonal.

Computationally, distributed Cholesky factorization has a higher proportion of 'Level 3' operations compared to LU factorization (as pivoting is not required), but affords somewhat slower cell computation, as only the lower triangular portions of the symmetric matrices L_i and A' are updated. Note that explicit transposition of L' can be avoided in the computation of $L'L'^{T}$.

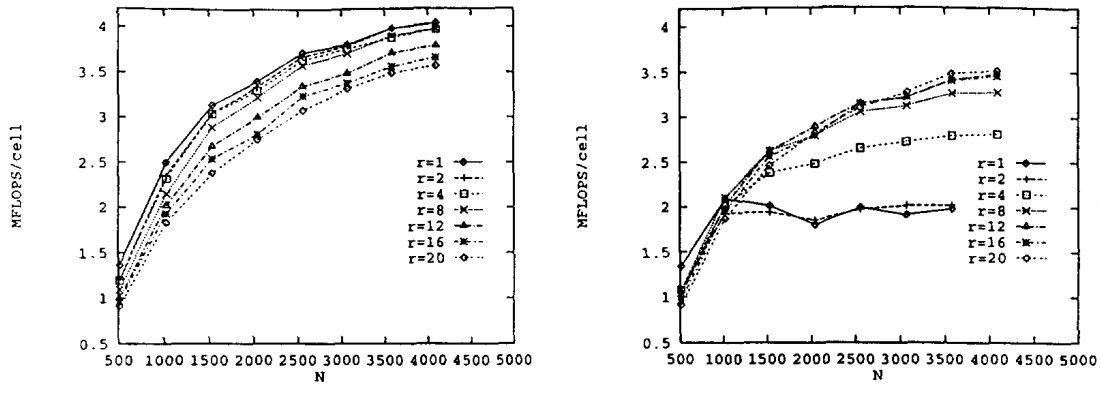
6 Performance of Distributed Matrix Factorization Algorithms

Figure 5(a) gives the performance of LU decomposition with distributed panels ($\omega = \omega_{\text{cell}}$) for the SBS matrix distribution with various values of r. That of the non-distributed panels ($\omega = r$) version is given in Figure 5(b). In Figure 5(a), performance decreases steadily with r; this because on the AP1000 the improvement of load balance for small r is more significant than the reduction of communication for large r. For $N > 4000$, performance steadily improves (to about 4.5 MFLOPS per cell, at the limit of AP1000 cell memory).

For a given r, the distributed panels version is faster than that of the non-distributed panels version, although for large N and r, there is little difference. The non-distributed panels version shows an optimal value of r that increases with N, eg. $r = 12$ being fastest for $N \approx 2000$ and $r = 20$ being fastest for $N \approx 4000$. This is to be expected because an $O(r/N)$ fraction of this computation is poorly load balanced, with r being well below the optimal value for cell computation on the $A' \leftarrow A' - L'U'$ operation. For $r \leq 4$, r is too small to speed up this operation (see Figure 1); hence these performance of these values of r are well below that of larger r.

Figure 6(a) gives the performance of Cholesky Factorization with distributed panels ($\omega = \omega_{\text{cell}}$) for the SBS matrix distribution with various values of r. That of the non-distributed panels ($\omega = r$) version is given in Figure 6(b). The relative performance of distributed and non-distributed panels, and that for various r within each is similar to LU factorization. Overall, Cholesky Factorization is slower, mainly because of number of floating point operations is half that of LU decomposition, while the overall volume of communication is comparable.

For large r, ie. $r \geq 16$, the distributed panels



(a) $\omega \approx \omega_{\text{cell}}$ (b) $\omega = r$

Figure 5: Performance (double precision) of the LU Factorization of an $N \times N$ matrix A on an 8×8 AP1000 for various SBS block sizes r

versions of both algorithms are about the same as that of the non-distributed panels. This is because the load balance advantages of the former diminish here and because the current implementation increases communication volume by a factor of ≈ 2 if $\omega > r$. For example, in Figure 4, each column (row) of the panel L' (U') is communicated in the formation of L' (U'), and later in the update of A'. This is not an intrinsic property of the distributed panels technique; it is merely due to its convenient implementation in terms of independent distributed BLAS operations.

The choice of $\omega = \omega_{\text{cell}} = 88$ was based purely on considerations for cell matrix-matrix multiply (Section 3). However, in the context of block-partitioned algorithms, an opposing consideration (arguing for a smaller value of ω) is minimizing the amount of computation in panel formation (overall $O(N^2\omega)$ FLOPs), as it will be slower than distributed matrix-matrix multiply. For $r = 1$ and $512 \leq N \leq 8192$ on the AP1000, we have found that these two effects turn out to be closely balanced, ie. that performance varied by only a few percent for $32 \leq \omega \leq 88$, with performance being only seriously reduced for $\omega < 24$. For larger r, however, performance degrades quickly as ω is reduced from ω_{cell}, due to loss of load balance over the panel formation.

7 Analysis of Load Balance for Distributed Panels

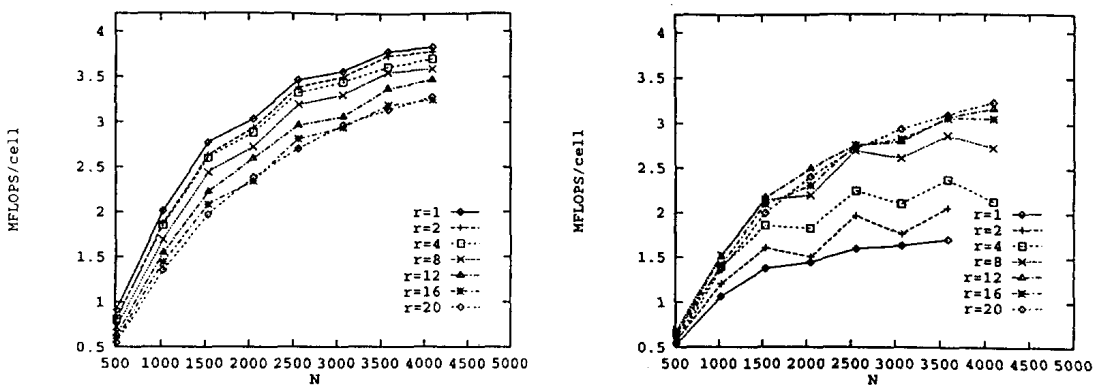
In the previous section, we have seen that distributed panels yields considerable improvement in matrix factorization algorithms over non-distributed panels. For moderate values of the matrix size N, the primary factor is the improvement in load balance afforded by distributed panels, while maintaining $\omega \approx \omega_{\text{cell}}$ for optimal cell computation. The load balance improvement is only over the $O(\omega/N)$ fraction of the overall computation such as forming the upper panel $U \leftarrow T^{-1}U$ in Figure 4; however, the relative slowness of distributed matrix factorization over matrix multiplication (cf. Figures 2 and 5) means that this portion is significant except for very large N.

We will consider this computation where U is an $\omega \times N$ SBS matrix distributed across a $P \times Q$ processor grid. Assuming that $rQ|N$, only the load balance across a column of cells need be considered, as the computation will be balanced across a row. For simplicity, it will be assumed that T has a unit diagonal (this means that no scaling of rows of U are required) and that $r|\omega$. The ('unblocked') algorithm for $U \leftarrow T^{-1}U$ is then, with $\omega' = \omega - 1$:

for $i = \omega'..1$

$$U_{\omega'-i..\omega',:} \leftarrow U_{\omega'-i..\omega',:} - T_{\omega'-i..\omega',\omega-i}U_{\omega-i,:}$$

This computation requires $w = \omega\omega' N$ floating



(a) $\omega \approx \omega_{\text{cell}}$ (b) $\omega = r$

Figure 6: Performance (double precision) of the Cholesky Factorization of an $N \times N$ matrix A on an 8×8 AP1000 for various SBS block sizes r

point operations (FLOPs), with step i requiring $w(i) = 2iN$ FLOPs.

Note that the situation with the formation of L for the LU and Cholesky factorizations is similar, although factors such as column scaling and row pivoting would make the analysis more complex.

The following result will be useful for the analysis:

Result 1 *Given w work units distributed across P cells, such that w_m is the maximum amount of work any cell is given ($w/P \leq w_m \leq w$), the overall efficiency of the computation is given by $E = w/(w_m P)$.*

Proof.

A formal proof can be derived from Amdahl's Law. Informally, the overall computation time will be proportional to w_m, as it will be determined by the cell(s) with w_m work units, by the definition of w_m. An efficiency of unity will occur in the case where $w_m = w/P$, since in this case, the load is perfectly balanced. □

For the non-distributed panels case, ie. $r = \omega$, all work is on a single cell, ie. $w = w_m$ and hence by Result 1 the efficiency is:

$$E_n = 1/P \tag{4}$$

For the distributed panels case, we can assume that U has block offset of zero [2] and, without loss of generality, begins in cell row 0.

Now $U_{\omega'-i..\omega',:}$ begins in cell $p_0(i) = \lfloor(\omega' - i)/r\rfloor \bmod P$ and has block offset $r_0(i) = (\omega' - i) \bmod r$. In the block-cyclic distribution, the cell holding the maximum number of rows of this matrix is either $p_0(i)$ or $p_0(i)+1$. Hence, using Equation 1 and applying Result 1 to the efficiency of the ith step of the update, $E_d(i)$, yields:

$$E_d(i) = r\lceil\frac{i}{rP}\rceil - r'_0(i)$$

where $r'_0(i) = r_0(i)$ if $\lceil i/r\rceil \bmod P = 1$, and $r'_0(i) = 0$ otherwise.

Noting $w(i)/w$ is the fraction of the overall computation in step i, Amdahl's Law then gives the overall efficiency as:

$$\begin{aligned} E_d &= (\Sigma_{i=1}^{\omega'} \frac{w(i)}{wE_d(i)})^{-1} \\ &= \frac{\omega\omega'}{2P} (\Sigma_{i=1}^{\omega'} r\lceil\frac{i}{rP}\rceil - r'_0(i))^{-1} \end{aligned} \tag{5}$$

Equation 5 assumes that on the ith step, each row $U_{\omega-i,:}$ is communicated separately; to reduce communication startup costs, its communication could be delayed till the next block boundary is crossed, at the loss of some load balance. Such a

[2] If ω is always chosen so that $r|\omega$, this will hold (see Section 5).

computation will be easier to implement. We will call this the 'block-distributed' version; its algorithm can be expressed as:

for $i = \omega/r - 1..1$
$\quad U_{j\to r..j',:} \leftarrow T^{-1}_{j\to r..j',\, j\to r..j'} U_{j\to r..j',:}$
$\quad U_{j..\omega',:} \leftarrow U_{j..\omega',:} - T_{j..\omega',\, j\to r..j'} U_{j\to r..j',:}$
$\quad$ where $j = \omega - ir, j' = j - 1$.

Here step i of the computation requires $w_b(i) = w_b^1(i) + w_b^2(i)$ FLOPs, where $w_b^1(i) = r(r-1)N$ is the FLOPs of the first sub-computation (done entirely on one cell row) of the step and $w_b^2(i) = 2ir^2N$ is that of the second.

Using again Result 1, the efficiency of the first sub-computation of step i is $E_b^1(i) = 1/P$, and that of the second is $E_b^2(i) = \frac{ir}{P\lceil ir/P \rceil}$, noting $\lceil ir/P \rceil$ is the largest number of rows of $U_{j..\omega',:}$ to be held in any cell. Thus, the overall efficiency of step i is given by Amdahl's law:

$$E_b(i) = \left(\frac{w_b^1(i)}{w_b(i)E_b^1(i)} + \frac{w_b^2(i)}{w_b(i)E_b^2(i)}\right)^{-1}$$
$$= \frac{1}{2P}\frac{w_b(i)}{N}\left(r(r-1)/2 + r\lceil ir/P \rceil\right)^{-1}$$

A final application of Amdahl's Law then gives the overall efficiency as:

$$E_b = \left(\Sigma_{i=1}^{\omega'/r-1} \frac{w_b(i)}{wE_b(i)}\right)^{-1}$$
$$= \frac{\omega\omega'}{2P}\left(\Sigma_{i=1}^{\omega/r-1} r(r-1)/2 + r\lceil ir/P \rceil\right)^{-1} \quad (6)$$

Qualitatively, the term $r(r-1)/2$ in Equation 6 means that for $r > 1$, E_d will always be somewhat higher than E_b, as one would intuitively expect. Figure 7 gives a quantitative comparison of Equations 5 and 6 for $\omega = \omega_{\text{cell}}$ for various r and number of cell columns, P. Here, one can see that for $r > 1$, E_d is always greater than E_b, especially since the comparison is slightly biased towards E_b for large r, since ω is chosen so that $r|\omega$. Furthermore, the graphs indicate that even for a large value of ω, low values of r yield higher load balance.

As indicated by Figure 7, for large P, efficiency due to load balance must necessarily decrease, assuming ω_{cell} (proportional to the square root of the cache size, see Equation 2) does not increase with P. For $rP \geq \omega = \omega_{\text{cell}}$, Equation 5 reduces to $E_d = \frac{\omega_{\text{cell}}}{2r} \cdot \frac{1}{P}$, indicating that asymptotically for large P, the benefits of distributed panels diminish. However, for moderate to large processor configurations, eg. $16 \leq P \leq 64 < \omega_{\text{cell}}$, low values of r still yield good efficiency, at least significantly more than $\frac{1}{P}$.

Figure 7(a) for $P = 8$ is in close agreement to Figure 3(a) for large N, as for the latter, cell computation speed and communication startup overheads are independent of r.

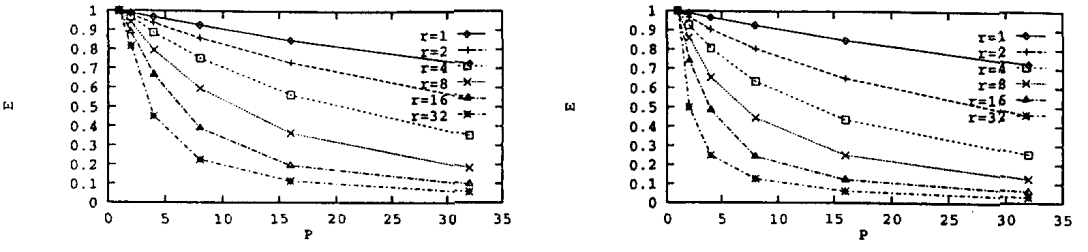
However, it may be argued that the 'block-distributed' version, with its comparable load balance, is adequate; Figure 8 gives an overall comparison of the two versions for $U \leftarrow T^{-1}U$ on the AP1000 for various block sizes r with an internal panel width of $\omega_1 = r$ (which improves cell performance also). It should be noted that in Figure 8(b), for small N, performance is (slightly) adversely affected by DBLAS call overheads[3] since the computation is implemented using $2\omega/r$ DBLAS calls, whereas that of Figure 8(a) is implemented with a single DBLAS call.

For $r \leq 6$, we see that even for such a relatively small computation, load balance is still a more significant factor than communication startup, as performance is consistently higher in Figure 8(a). We find an optimum $r = 4$ in this case, finding a tradeoff between load balance and communication startup (and optimal cell computation, with $\omega_1 = r$). However, a comparison of Figures 8(a) and 3(b) indicates that the (fully) distributed version with an optimally chosen internal panel width $\omega_1 \approx \sqrt{\omega}$, and with small block size $r \leq 2$, gives the overall best performance for this computation.

8 Conclusions

We have found that for scalar-based distributed memory multiprocessors such as the AP1000, due to their relatively low cell memory bandwidth, the panel width ω must be decoupled from the matrix distribution block size r for optimal performance, with $\omega > r$. This is because it enables optimal cell computation and high load balance over panel formation. Even though the panel formation forms only an $O(\omega/N)$ fraction of the overall computation, in practice it still has a significant effect on performance, except for very large N and where $P > \omega$. As indicated in Section 7, a reduction in communication startup by a factor of r can still be afforded, at the loss of some load balance. The value of ω is determined mainly by considerations the register and cache levels of the distributed memory hierarchy, with $32 \leq \omega \leq 88$ being determined as optimal for the Fujitsu AP1000. The performance is significantly better than for

[3] These overheads are from parameter checking and buffer allocation, significant only on very small computations.



(a) distributed version, $\omega = \omega_{\text{cell}}$

(b) 'block-distributed' version, $\omega = r\lfloor \omega_{\text{cell}}/r \rfloor$

Figure 7: Efficiency E due to load balance effects of versions of distributed $U \leftarrow T^{-1}U$ where U has ω rows distributed across P cell rows and T is a unit triangular matrix.

the best non-distributed panels ($\omega = r$) case.

In the formation of these distributed panels, one can still tradeoff some load balance to reduce communication startup overheads by a factor of r, by communicating block rows and columns, rather than individually communicating the required matrix rows and columns. For the AP1000, experiments indicate that the latter gives better performance for moderate to large values of N.

With the restriction of $\omega = r$, the optimal matrix distribution block size r depends on the matrix size N, and must be determined by experimentation. This is not necessary for distributed panels, where a low value of r may be determined from architectural considerations only.

Overall, our results show that distributed panels with $r \approx 1$ is optimal for LU and Cholesky factorizations on the AP1000, justifying previously made assumptions that the scattered ($r = 1$) matrix distribution was optimal for dense linear algebra computations on the AP1000 [12, 1, 10].

For scalar-based distributed memory multiprocessors with faster cell computation performance such as the Cray T3D, considerable benefits can still be expected from the technique of distributed panels. Due to its smaller cache size, a panel width of $20 \leq \omega \leq 32$ with $1 \leq r \leq 4$ is expected to be optimal on the T3D. Note that the relatively higher communication startup time on the T3D should not be a major problem, as overall communication startup time is of a smaller order than that of distributed panel formation, and it can still be reduced by a factor of r for distributed panels in any case. However, such architectures will require an implementation of distributed panels avoiding the redundant communications of the current AP1000 implementation.

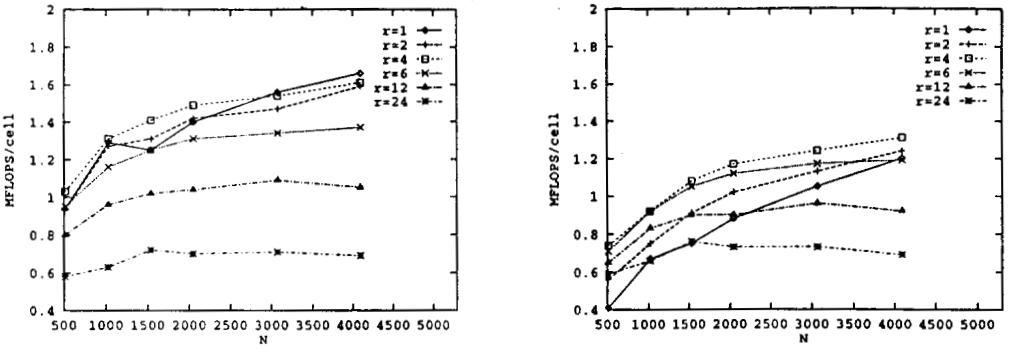
LU and Cholesky factorization cover a range of features of dense distributed linear algebra computations. Future work includes implementing distributed panels for QR factorization, which is rich in triangular matrix update operations and requires efficient distributed vector-matrix multiply operations. Here, $\omega > 1$ requires an $O(N^2\omega)$ extra floating point operations, unlike for the LU and Cholesky factorizations. This overhead may argue for a slightly lower optimal ω, but we would expect that qualitatively, QR factorization with distributed panels would yield comparable results to the LU and Cholesky factorizations.

References

[1] R.P. Brent. The LINPACK Benchmark on the Fujitsu AP 1000. In *Frontiers '92: Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation*, Virginia, October 1992.

[2] R.P. Brent, A. Czezowski, M. Hegland, P.E. Strazdins, and B.B. Zhou. Linear Algebra Research on the AP1000. In *Second Parallel Computing Workshop for the Fujitsu PCRF*, pages P1–L–1 - P1–L–13, Kawasaki, November 1993.

[3] J. Choi, J. J. Dongarra, S. Ostrouchov, A. P. Petinet, D.W. Walker, and R.C. Whaley. The Design and Implementation of the ScaLAPACK LU, QR and Cholesky Factorization Routines. In *Technical Report ORNL/TM-*



(a) distributed version

(b) 'block-distributed' version

Figure 8: Performance (double precision) of distributed $U \leftarrow T^{-1}U$, where U is an $\omega_{\text{cell}} \times N$ distributed $r \times r$ block-cyclic matrix, T is lower unit triangular, with internal panel width $\omega_1 = r$, on an 8×8 AP1000

12470, pages 1–26, Oak Ridge National Laboratories, September 1994.

[4] J. Choi, J.J. Dongarra, R.Pozo, and D.W.Walker. ScaLAPACK: A Scalable Linear Algebra Library for Distributed Memory Concurrent Computers. In *Frontiers '92: Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation*, Virginia, October 1992.

[5] J. Choi, J.J. Dongarra, and D.W. Walker. PB-BLAS: A Set of Parallel Block Basic Linear Algebra Subprograms. In *Proceedings of the Scalable High Performance Computing Conference, 1994*, pages 534–41, Knoxville, Tennessee, May 1994.

[6] C.Anderson et al. *LAPACK User's Guide.* SIAM Press, Philadelphia, 1992.

[7] H. Ishihata, T. Horie, S. Inano, T. Shimizu, and S. Kato. An Architecture of Highly Parallel Computer AP1000. In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 13–16, May 1991.

[8] J.Dongarra, J.J. Du Croz, S.J. Hammarling, and I.S. Duff. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software (16)*, pages 1–17, 1990.

[9] R.W. Numrich, P.L. Springer, and J.C. Peterson. Measurement of Communication Rates on the Cray T3D Interprocessor Network. In *International Conference on High-Performance Computing and Networking*, pages 150–157, Munich, April 1994.

[10] R.P.Brent and P.E. Strazdins. Implementation of BLAS Level 3 and LINPACK Benchmark on the AP1000. *Fujitsu Scientific and Technical Journal*, 29(1):61–70, 1993.

[11] P.E. Strazdins. Prototyping Parallel LAPACK using Block-Cyclic Distributed BLAS. In *Third Parallel Computing Workshop for the Fujitsu PCRF*, pages S–C–1 – S–C–7, Kawasaki, November 1994.

[12] P.E. Strazdins and R.P. Brent. Implementing Blas Level 3 on the AP1000. In *Proceedings of the CAP Workshop '91*, pages H–1 – H–17, Australian National University, Canberra, November 1991.

[13] R.A. van de Geijin. Massively parallel LINPACK benchmark on the Intel Touchstone Delta. In *Technical Report TR-91-92*, University of Texas, August 1991.

[14] B.B. Zhou and R.P. Brent. Parallel Implementation of QRD Algorithms on the Fujitsu AP1000. In *Proceedings of 17th Australian Computer Science Conference*, pages 733–743, Christchurch, January 1994.