



THE AUSTRALIAN NATIONAL UNIVERSITY

TR-CS-98-07

**A Comparison of Lookahead and
Algorithmic Blocking Techniques for
Parallel Matrix Factorization**

Peter Strazdins

July 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

This technical report series is published jointly by the Department of Computer Science, Faculty of Engineering and Information Technology, and the Computer Sciences Laboratory, Research School of Information Sciences and Engineering, The Australian National University.

Please direct correspondence regarding this series to:

Technical Reports
Department of Computer Science
Faculty of Engineering and Information Technology
The Australian National University
Canberra ACT 0200
Australia

or send email to:

`Technical.Reports@cs.anu.edu.au`

A list of technical reports, including some abstracts and copies of some full reports may be found at:

<http://cs.anu.edu.au/techreports/>

Recent reports in this series:

- TR-CS-98-06 M. Manzur Murshed. *Optimal computation of the contour of maximal elements on mesh-connected computers*. July 1998.
- TR-CS-98-05 M. Manzur Murshed. *Optimal computation of the contour of maximal elements on constrained reconfigurable meshes*. May 1998.
- TR-CS-98-04 M. Wilson and K. Yap. *ACSys/RDN experiences with Telstra's experimental broadband network, second progress report*. April 1998.
- TR-CS-98-03 M. D. Wilson, S. R. Taylor, M. Rezny, M. Buchhorn, and A. L. Wendelborn. *ACSys/RDN experiences with Telstra's experimental broadband network, first progress report*. April 1998.
- TR-CS-98-02 M. Manzur Murshed and Richard P. Brent. *Adaptive AT^2 optimal algorithms on reconfigurable meshes*. March 1998.
- TR-CS-98-01 Scott Milton. *Thread migration in distributed memory multicomputers*. February 1998.

A Comparison of Lookahead and Algorithmic Blocking Techniques for Parallel Matrix Factorization

Peter Strazdins

peter@cs.anu.edu.au

Department of Computer Science,
Australian National University

Abstract

In this paper, we analyse and compare the techniques of *algorithmic blocking* and (*storage blocking* with) *lookahead* for distributed memory LU, LLT and QR factorizations. Concepts and some useful properties of a simplified model of lookahead are explored, including the minimal degree of lookahead required for optimal performance. Issues in the implementation of lookahead are discussed, which are more involved for the cases of LLT and QR factorizations. It is also explained how hybrid algorithmic blocking and lookahead techniques can be implemented. Implications for parallel linear algebra library design are also discussed.

Results are given on the Fujitsu AP1000 and AP+ multicomputers, which have relatively high communication to computation to speeds. The results indicate that both methods are superior to storage blocking (without lookahead). They also indicate that for such machines, the hybrid method is optimal for smaller matrices, due to savings in communication startups. For larger matrices, algorithmic blocking gave the best performance, due to its better load balancing properties. An exception was LLT for the AP+, where lookahead alone gave comparable or better performance for larger matrix sizes as well. Performance models, predicting the minimum matrix size where lookahead becomes effective, indicate this trend can be expected for machines with lower communication to computation speeds, but that the range for where lookahead is superior is extended.

Keywords: dense linear algebra, block cyclic decomposition, storage blocking, algorithmic blocking, pipelining, lookahead.

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 [3], 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* [7, 12, 9, 11], where $\omega \approx \omega_m, r = s \approx 1$.

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 [1, 11]. For small to moderate N , these imbalances can be significant, so that a value of $\omega < \omega_m$ may be optimal. However, storage blocking minimizes communication startup and volume overhead, although by only a constant factor (excepting the case of LLT).

Algorithmic blocking (also known as ‘distributed panels’ [12]) achieves better load balancing properties by distributing the panel across all processors, at the expense of increased communication overhead, both in startup and volume. Whether a multiprocessor favors algorithmic blocking over storage blocking depends on whether ω_m is large, and, to a lesser extent, whether the ratio of communication to floating point speed is relatively high [12].

However, the load balance in panel formation is not perfect, being about 90% for $\omega = 64, r = 1$ and $P = 8$, but diminishing for larger P/ω ratios [12].

With storage blocking, or more precisely for $\omega = s$, the broadcast of the ‘lower’ panel can be *pipelined* for these matrix factorizations, since all horizontal communication flows from left to right. This has the benefit that κ_{bc} , the ratio of the cost of these broadcasts relative to a point-to-point send, can be reduced from $\lg_2(Q)$ to 2 [2].

With the pipelined communication of the lower panel, it is well known that it is possible to *overlap communication with computation*, that is to reorganize the computation so that other cells can perform useful work while the lower panel is being formed. A survey of various such techniques for the case of LU factorization is given in [4]; this paper also presents the new technique of performing the row swaps on other cells at the same time as the lower panel is formed. This improved LU performance by up to 20% for small matrices, decreasing to 5% for large matrices ($N = 8000$) on a 64 node Paragon.

Lookahead can be regarded as a related technique. In *lookahead*, the formation (computation and communication) of the lower panel is overlapped with the other (mainly Level 3 computation) operations of the previous iteration. It could be called the ultimate such technique in that it offers the best chance that the overlapping is 100%. Lookahead was used to enhance a parallel LU factorization algorithm which set world records for the LINPACK MP Benchmark on the Intel ASCI supercomputer in 1995 and 1997 [1, 6]. A validated performance model for LU with lookahead was given here.

Lookahead has been recently implemented to improve load balance for the triangular matrix update routines used in ScaLAPACK [10]. Lookahead has also been discussed in [14], with a comparison of lookahead with algorithmic blocking being given for ‘inverse’ LU and LLT factorization on the Fujitsu AP1000¹. The results indicated that lookahead gave better performance for small matrices, but algorithmic blocking was faster for moderate to large matrices.

For lookahead, the cell column holding the lower panel has deferred sufficient of its Level 3 computation from the previous iteration, so that it is ready to broadcast the lower panel as soon as the other cells are ready to receive it. This means that the load imbalance for forming the lower panel can be (to a large extent) eliminated, and also that the communication startup and software overheads in forming the panel can be effectively parallelized. It has been noted that for the latter, the bulk of such overheads for these computations arise from the formation of the lower panel [15].

The main original contributions of this paper are firstly to propose and develop a model for computations with lookahead, from which some useful properties of lookahead are discovered. Secondly, to apply lookahead to two new computations, LLT and QR factorization. Thirdly, to provide the first comparison of algorithmic blocking and lookahead techniques, to address the question of which is the superior technique for dense linear algebra. This comparison is made both experimentally on two distributed memory platforms, and theoretically by a performance model for LU factorization. Such a model covering algorithmic blocking has not been published previously; for lookahead, the model is extended from previous work to cover the small N case. Finally, to propose and evaluate hybrid lookahead and algorithmic blocking methods, which in some circumstances will perform better than either method alone.

This paper is organized as follows. Section 1.1 briefly describes the block-partitioned matrix factorizations, to provide a basis for discussion in the rest of the paper. The Fujitsu AP1000 and AP+ multicomputers, capable of *wormhole broadcasts*, are described in Section 1.2. Key ideas in lookahead, with and without wormhole broadcasts, are discussed in Section 2, together with the development of some useful properties. The implementation of lookahead for the matrix factorizations is described in Section 3. The performance on the AP1000 and AP+ of the implementations of *lookahead*, algorithmic

¹By ‘inverse’ LLT factorization, we mean recovering the original matrix A from L , where $A = LL^T$. This computation proceeds like the LLT factorization but in reverse; thus, the panel formation computations can be naturally overlapped with the matrix-matrix updates. With $\omega = r = s$, it forms an equivalent situation as with LLT with lookahead.

$$\begin{aligned}
& \text{LU2}(N - i_1, \omega, L^i, p) & L^i = A_{i_1..N', i_1..i'_2}, \quad i_1 = i\omega, \quad i_2 = i_1 + \omega & \text{-(LU.1)} \\
& (W^i | U^i) \leftarrow (P(\omega', p_{\omega'}) \dots P(0, p_0))(W^i | U^i) & W^i = A_{i_1..i'_2, 0..i'_1}, \quad U^i = A_{i_1..i'_2, i_2..N'} & \text{-(LU.2)} \\
& U^i \leftarrow (T^i)^{-1} U^i & T^i = L_{0..\omega', :}^i \text{ (lower tri., unit diag.)} & \text{-(LU.3)} \\
& A^i \leftarrow A^i - L_{\omega..N'-i_1, :}^i U^i & A^i = A_{i_2..N', i_2..N'} & \text{-(LU.4)}
\end{aligned}$$

where $\text{LU2}(M, N, A, p)$ is defined as:

$$\begin{aligned}
& \text{for } j = 0..N' & & \\
& \quad \text{find } p_j : j..M' \text{ s.t. } |A_{p_j, j}| \geq |A_{j:M', j}| & \text{-(LU2.1)} \\
& \quad A \leftarrow P(j, p_j)A & \text{(swap rows } j \text{ and } p_j) & \text{-(LU2.2)} \\
& \quad l^j \leftarrow l^j / A_{j,j} & l^j = A_{j+1..M', j} & \text{-(LU2.3)} \\
& \quad L^j \leftarrow L^j - l^j u^j & L^j = A_{j+1..M', j+1..N'}, \quad u^j = A_{j, j+1..N'} & \text{-(LU2.4)}
\end{aligned}$$

and $P(j, k)$ is the identity matrix with rows j and k permuted.

Figure 1: Partial LU factorization of an $N \times N$ matrix A , with $0 \leq i < \frac{N}{\omega}$

$$\begin{aligned}
& \text{LLT2}(\omega, T^i) & T^i = A_{i_1..i'_2, i_1..i'_2} \text{ (lower tri., non-unit diag.)} & \text{-(LLT.1)} \\
& L^i \leftarrow L^i (T^i)^{-T} & L^i = A_{i_2..N', i_1..i'_2}, \quad i_1 = i\omega, \quad i_2 = i_1 + \omega & \text{-(LLT.2)} \\
& A^i \leftarrow A^i - L^i (L^i)^T & A^i = A_{i_2..N', i_2..N'} \text{ (lower tri.)} & \text{-(LLT.3)}
\end{aligned}$$

where $\text{LLT2}(N, A)$ is defined as:

$$\begin{aligned}
& \text{for } j = 0..N' & & \\
& \quad A_{j,j} \leftarrow \sqrt{A_{j,j}}; \quad l^j \leftarrow l^j / A_{j,j} & l^j = A_{j+1..N', j} & \text{-(LLT2.1)} \\
& \quad L^j \leftarrow L^j - l^j (l^j)^T & L^j = A_{j+1..N', j+1..N'} & \text{-(LLT2.2)}
\end{aligned}$$

Figure 2: Partial LLT factorization of a lower triangular $N \times N$ matrix A , with $0 \leq i < \frac{N}{\omega}$

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 is used to extend the comparisons to other machines. Conclusions are given in Section 6.

1.1 Matrix Factorization Algorithms

In this section, the block-partitioned matrix factorization algorithms are described, being essentially simplified versions of those used in LAPACK [5] and ScaLAPACK [3].

Figure 1 gives the indicates the i th partial LU factorization with row pivoting, using a panel width of ω , for an $N \times N$ matrix A . This is repeated for $i = 0, 1, 2, \dots, \frac{N}{\omega} - 1$ to give a full factorization. Matrix indices begin from 0, with the notation x' being used as a shorthand for $x - 1$. More details on the parallel implementation of this algorithm can be found in Section 5.

Figures 2 and 3 give corresponding algorithms for partial LLT and QR factorizations, respectively. In the parallel implementation of the algorithm in Figure 6, for iteration i , the temporary matrices V' and T may be regarded as having their top-left corners aligned with the top-left corner of V^i , and the temporary matrix W may be thought of being aligned with A^i .

Note that steps -(QR.3) and -(QR.5) introduce (further) redundant floating point operations with the above-diagonal part of the rectangular V' being padded by zeroes (cf. step -(QR2.3)). This is efficient provided the Level 3 matrix multiply speed is at least twice that of the (also Level 3) triangular matrix update speed (eg. as for step -(QR.4)) over operands of width ω . As in [3] for the Intel Paragon, we have found this to be the case on the AP+ and AP1000, although only by a narrow margin for $\omega > 32$.

$$\begin{array}{ll}
\text{QR2}(N - i_1, \omega, V^i, \tau, V') & V^i = A_{i_1..N', i_1..i_2'}, i_1 = i\omega, i_2 = i_1 + \omega & \text{-(QR.1)} \\
\text{GR2}(N - i_1, \omega, V^i, V', \tau, T) & & \text{-(QR.2)} \\
W \leftarrow V'^T A^i & A^i = A_{i_1..N', i_2..N'} & \text{-(QR.3)} \\
W \leftarrow TW & (T \text{ is lower tri., non-unit diag}) & \text{-(QR.4)} \\
A^i \leftarrow A^i - V'W & & \text{-(QR.5)}
\end{array}$$

where $\text{QR2}(M, N, A, \tau, V')$ is defined as:

$$\begin{array}{ll}
\text{for } j = 0..N' & \\
(A_{j,j}, \beta, \tau_j) \leftarrow f(A_{j,j}, v^j \cdot v^j) & v^j = A_{j+1..M', j} & \text{-(QR2.1)} \\
v^j \leftarrow v^j / \beta & & \text{-(QR2.2)} \\
V'_{0..j', j} \leftarrow 0, V'_{j,j} \leftarrow 1, V'_{j+1..M', j} \leftarrow v^j & & \text{-(QR2.3)} \\
w \leftarrow (V'_{j..M', j})^T A_{j..M', j+1..N'} & & \text{-(QR2.4)} \\
V^j \leftarrow V^j - \tau_j v^j w & V^j = A_{j+1..M', j+1..N'} & \text{-(QR2.5)}
\end{array}$$

and $\text{GR2}(M, N, A, V', \tau, T)$ is defined as:

$$\begin{array}{ll}
\text{for } j = 0..N' & \\
t^j \leftarrow -\tau_j (V'_{j..M', j})^T A_{j..M', 0..j'} & t^j = T_{j, 0..j'} & \text{-(GR2.1)} \\
t^j \leftarrow T^j t^j & T^j = T_{0..j', 0..j'} \text{ (lower tri., non-unit diag)} & \text{-(GR2.2)} \\
T_{j,j} \leftarrow \tau_j & &
\end{array}$$

and $f()$ is a 3-valued function of 2 scalar variables.

Figure 3: Partial QR factorization of an $N \times N$ matrix A , with $0 \leq i < \frac{N}{\omega}$

1.2 The Fujitsu AP1000 and AP+

The Fujitsu AP1000 [8] is a SPARC 1-based multiprocessor; it's cells have a theoretical peak speed of 5.5 MFLOPS (double precision). The AP1000 uses a physical torus communication network using wormhole routing, with each link being capable of 25 MBs⁻¹. 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 [2]); such a broadcast will be referred to as a *wormhole broadcast*. Wormhole broadcasts do *not* require any synchronization, in that the sender does not have to wait till the other cells post the corresponding receive calls.

Wormhole 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 AP1000 cells have a 128KB direct-mapped copy-back cache (shared instruction and data). This means that a logical blocking factor $\omega_m \approx 64$ is optimal for this architecture.

On the AP1000, benchmark programs have yielded the following parameters that will be used in Section 5 for the Distributed Linear Algebra Model (DLAM) of [2] for LU, with some extensions [15, 11]. The communication startup cost $\alpha = 25\mu s$, the communication transmission cost per word $\beta = 1.2\mu s$, the level-2 computation cost per floating point operation $\gamma_2 = 0.4\mu s$, the computation speed for a triangular matrix update operation (eg. LT^{-1} or $T^{-1}U$) is $\gamma_3^\Delta = 0.3\mu s$ ², $\gamma_3(16) = .25\mu s$ and $\gamma_3(64) = .20\mu s$, where $\gamma_3(\omega)$ is the cost per floating point operation in a local matrix multiply having input operands of width ω .

The AP+ is similar to the AP1000, except that its cells are based on a 50 MHz Viking SuperSparc chip, having 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 $\omega_m \approx 32$.

The AP+ uses an identical communication network³. On the AP+, benchmark programs have yielded

²This depends somewhat on the panel width ω ; the value above represents the cost at an intermediate width of $\omega = 32$.

³Because a memory copy is part of a communication on the AP1000/AP+, the AP+ has a lower value of β due to a faster memory bus.

values of $\alpha = 12\mu s$, $\beta = 0.64\mu s$, $\gamma_2 = 0.12\mu s$, $\gamma_3^\Delta = 0.083\mu s$, $\gamma_3(16) = .035\mu s$ and $\gamma_3(32) = .030\mu s$.

On both the AP and AP+, the mode of message transmission that will be used is that a when message arrives at a cell, it is stored at first in a *ring buffer* (of maximum size 512 KB). The storage consumed by the message may only be reclaimed when the receiving cell has posted a corresponding receive call, and finished copying that message into its specified destination. Overflow of this ring buffer is a problem on these machines that is potentially exacerbated by lookahead.

2 Concepts and Properties of Lookahead

This section presents a simplified model of lookahead, to illustrate how it can achieve better load balancing with and without wormhole broadcasting. We will concentrate mainly on the situation with wormhole broadcasting, since that is simpler and is used for the results of Section 4; however, most essential properties for this case apply to both.

Consider the following computation on a $m \times N$ matrix A distributed in a block-cyclic fashion over a $1 \times Q$ processor grid, with $\omega = s$ (assume, for the sake of simplicity, that $Qs|N$):

$$\begin{array}{lll} \text{for } i = 0..N/\omega - 1 & & \\ L^i \leftarrow F(L^i) & L^i = A_{:,i_1..i_2'}^i, \ i_1 = i\omega, \ i_2 = i_1 + \omega & (m\omega^2 \text{ flops, on cell } q^i) \\ \text{broadcast } L^i & & (\text{volume is } m\omega) \\ A^i \leftarrow M(L^i, A^i) & A^i = A_{:,i_2..N'}^i & (2m\omega(N - i_2) \text{ flops}) \end{array}$$

Note that the upper triangular matrix update $A \leftarrow AT^{-1}$ can be modelled in this way, with $F(L^i) = L^i T_{i_1..i_2', i_1..i_2'}$ and $M(L^i, A^i) = A^i - L^i T_{i_1..i_2', i_2..N'}$. Portable source codes for such a computation have recently been made available [10].

Consider the i th iteration of the above algorithm. Let n_q^i be the local number of block columns of A^i on cell q , and $q^i = i\%Q$. Lookahead can be applied to this algorithm by letting cell q^{i+1} defer the update of the last λ^i block columns of A^i until after the broadcast stage of the next iteration, where $0 \leq \lambda^i \leq n_q^i - 1$. In this way, the broadcast can be received by the other cells as soon as they are ready, provided λ^i is sufficiently large.

Assume that the cost of updating a block column in $M(L^i, A^i)$ is unity. Let ρ (κ) be the ratio of time spent in $f(L^i)$ (broadcast L^i) to that spent in updating a single block column in $M(L^i, A^i)$. To a good approximation, ρ and κ will be constant over this computation, and can be regarded as parameters of it. Figure 7 gives time lines of the Q cells for this computation, with Mi representing a time unit being spent in $M(L^i, A^i)$ (and similarly for Fi and ci).

From Figure 7(b)–(c), the upper bound on the degree of lookahead, given by $n_q^i - 1$, decreases with i , so that in the last Q iterations, no lookahead at all is possible. The initial degree of lookahead is denoted λ , with a lookahead of $\lambda^i = \min(\lambda, n_q^i - 1)$ being applied on the i th iteration. In previous work, the algorithms described correspond to the maximal lookahead of $\lambda = n_0^0 - 1$ [1, 6, 10], but Figure 7(b)–(c) indicates that this is not necessary for optimal performance.

Figure 7(c) indicates lookahead with a *ring broadcast*, in which each intermediate cell receives the message, and then passes it on to its neighbor on the right. The cost relative to a point-to-point message is $\kappa_{bc} \approx 2$ in this case, since the pipelining introduces a ‘bubble’ of size $Q\kappa$ every Q iterations (cf. time steps 24–25 in cell $Q - 1$ for Figure 7(c), noting that the size of the bubble can be reduced by $\rho + 1$ with a sufficient degree of lookahead).

It can be seen that cell $Q - 1$, owning the last block column, is on the critical path for this computation, having to perform more ‘M’ operations than any of the other cells. Generally, cell $Q - 1$ is the last to complete any iteration.

As lookahead reduces the degree of synchronization of such computations, a potential problem is the number of (unread) (L^i) panels that must be buffered on any cell. This can be at most 2 [1]. For example with wormhole broadcasting with $\rho = 2, \kappa = 1, \lambda = 3, n_0^0 = 4$ and $Q = 4$, cell 3 is begins reading L^8 at the same time as cell 2 begins its broadcast of L^9 . For large matrices, this can cause ring buffer overflow on the AP machines and similar problems on the ASCI supercomputer [1]. This situation can be avoided by implementing *handshaking* [1], where in the case of a ring broadcast, a cell does not pass on L^i to its neighbor until the neighbor has posted an ‘acknowledge’ message, indicating that the receipt of L^{i-1} is complete. With wormhole broadcasting, handshaking is more complex: the sender must wait for such an acknowledgment from *all* other cells before sending.

2.1 Properties

Consider the above computation with wormhole broadcasting for iterations i such that $\lambda \leq n_q^i - 1$. The time T_q^i in which cell $q, 0 \leq q < Q$, completes iteration i is given by:

$$T_q^i = C_q^i + \kappa + \delta_{q,q^{i-1}}\lambda + (n_q^i - \delta_{q,q^i}\lambda) \quad (1)$$

$$C_q^i = \max(T_q^{i-1}, C^i) \quad (2)$$

$$C^i = T_{q^i}^{i-1} + \rho \quad (3)$$

$$T_q^{-1} = 0 \quad (4)$$

where $\delta_{i,j}$ is the Kronecker delta. Here C_q^i represents the time cell q begins the send (or receipt) of L^i .

We believe that the following properties hold for this computation:

$$T_q^i \leq T_{Q-1}^i + \delta_{q^i,Q-2}\lambda + \rho - 1 \quad (5)$$

$$C_{Q-1}^i = T_{Q-1}^{i-1} + \delta_{q^i,Q-1}\rho, \text{ if } \lambda \geq \rho \quad (6)$$

$$C^{i+1} \geq C_q^i + \kappa, \text{ if } \lambda \leq \rho + 1 \quad (7)$$

Property 5 identifies that cell $Q - 1$ is on the critical path for the computation, by giving a limit for how much other cells can exceed reaching the corresponding point in the computation. Property 6 gives the condition for an optimal degree of lookahead; it is intuitively evident: a lookahead of no more than the equivalent amount of work in forming L^i is required for optimal performance. Property 7 gives the maximal lookahead required before handshaking is needed.

If handshaking is implemented, C^i must be redefined as:

$$C^i = \max(T_{q^i}^{i-1} + \rho, \max_{q=0}^{Q-1} (C_q^{i-1} + \kappa)) \quad (8)$$

The following properties together state that implementing handshaking will cause no delay. Property 9 states that broadcasts from cell $Q - 1$ will not be stalled by handshaking. Property 10 states that handshake stalls in other cells will not delay the critical path cell $Q - 1$. These may be proven using Property 5.

$$C^{i+1} \geq C_q^i + \kappa, \text{ if } q^{i+1} = Q - 1 \quad (9)$$

$$C_{Q-1}^{i+1} + \kappa \geq C^{i+1} \quad (10)$$

In the case of ring broadcasts, Equation 2 must be modified to:

$$C_q^i = \begin{cases} C^i & , \text{ if } q = q_i \\ \max(T_q^{i-1}, C_{(q-1)\%Q}^i + \kappa) & , \text{ if } q \neq q_i \end{cases} \quad (11)$$

and, for handshaking:

$$C_q^i = \max(C_q^i, C_{(q+1)\%Q}^{i-1} + \kappa) \quad (12)$$

where C_q^i is the expression on the right hand side of Equation 11.

Similar properties hold for this case: cell $Q - 1$ is on the critical path in at least as strong a sense, $\lambda \geq \rho$ gives optimal lookahead, and handshaking is not required for any λ . A final interesting result is that the size of the ‘bubble’ due to pipelined communication, which occurs at cell $Q - 1$ every Q iterations, is given by:

$$C_{Q-1}^i - T_{Q-1}^{i-1} = \max(0, Q\kappa - \min(\lambda^i, \rho + 1)) \quad , \text{ if } q_i = Q - 1 \quad (13)$$

This means that provided $\lambda \geq \rho + 1 \geq Q\kappa$, $\kappa_{bc} = 1$, that is, lookahead can improve the efficiency of the pipelined broadcast.

While formal proofs of these properties have not yet been developed, the above equations for T_q^i and C_q^i have been implemented in a computer program and the above properties have been checked over an exhaustive test for various values of Q, κ, ρ and λ , for the cases with and without wormhole broadcasting, and with and without handshaking. The program was used to generate Figure 7 and is a very useful tool for investigating lookahead. It is available at:

<http://cs.anu.edu.au/people/Peter.Strazdins/projects/Lookahead>

	LU	LLT	QR
dominant part	LU2.4	LLT2.2	QR2.4–5, GR2.1
ρ (on AP1000:) (on AP+:)	$\frac{\gamma_2}{2\gamma_3}$ (1.0) (2.0)	$\frac{\gamma_3^\Delta}{\gamma_3}$ (1.5) (2.7)	$\frac{3\gamma_2}{4\gamma_3}$ (1.5) (3.0)
κ	$\frac{\beta}{2\omega\gamma_3}$	$\frac{\beta}{\omega\gamma_3}$	$\frac{\beta}{4\omega\gamma_3}$

Table 1: Large-matrix values for ρ and κ

2.2 Relationship to Matrix Factorizations with Lookahead

The above model strictly only applies to a computation such as $A \leftarrow AT^{-1}$ executed on an ideal machine. In this section, how well the above properties can be applied to lookahead in the three matrix factorizations on a real machine will be discussed.

Consider the case where the panel length is large enough so that effects such as communication startups and software overheads in the formation of L can be ignored, and that the computation speed can be approximated by the ‘dominant part’ of the panel formation. Then, the parameters ρ and κ can be regarded as constants; using the algorithms of Section 1.1, values for ρ and κ are derived in Table 1.

Typically for modern multiprocessors, $2 \leq \frac{\gamma_2}{\gamma_3} \leq 4$, $1.5 \leq \frac{\gamma_3^\Delta}{\gamma_3} \leq 2$, $8 \leq \omega \leq 64$ and $10 \leq \frac{\beta}{\gamma_3} \leq 80$. With Property 13, one can see that for ω towards the upper end of this range, and for small–moderate Q , lookahead can gain an advantage in communication speed by reducing the size of the pipeline bubble.

For matrix factorizations, we must consider a $P \times Q$ array, but our attention can be concentrated on the processor row owning the last block row of the matrix, being on the critical path (in the same way that the cell column holding the last block column is on the critical path).

The first major difference with matrix factorizations and the model above is that, for the case $\omega = r = s$, the panel length decreases by a block every P iterations. Effectively, this means the amount of work per iteration decreases faster with i for a matrix factorization (for the case $r < s$, this decrease would at be smoother, but at the same overage rate). The effect of this is that when the decrease occurs, events happening in the next iteration occur ‘closer’ to those in the previous. Hence, the situation where handshaking may be required may be broadened, but Properties 5, 6, 9, 10 and 13 should not be significantly affected.

Secondly, for the case of LU, with $\omega = r = s$, there is an increased amount of work in performing step -(LU.3) every P iterations, when that processor row owns U^i (this situation similarly exists for QR). This could potentially affect all the above Properties, but only by a small extent if panel length is still large, as the increase would be relatively small. Indeed on a real processor, there will be fluctuations in the amount of work per iteration not only between iterations but between cells in a given iteration, due to unpredictable effects such as cache misses.

Thirdly, as the panel length becomes small enough for communication startup and software overheads to affect overall performance, ρ and κ may tend to increase with i . However, it is likely that at this point that the width of the remaining matrix will be too small to permit full lookahead in any case.

In summary, it may be prudent to implement handshaking for matrix factorizations even where the results of the previous section predict it would have had no effect. However, the other Properties should still be reasonably good predictors for matrix factorizations on a real multiprocessor.

3 Implementation of Lookahead

In our implementation of lookahead, λ is a run-time settable parameter, which was passed down into the respective matrix factorization routine. Handshaking was not implemented, since for the range of matrix sizes and blocking factors of interest, setting λ to the smallest value for maximal lookahead, as predicted by Property 7 and using the Table 1, sufficed to both to avoid AP1000/AP+ ring buffer overflow, and to yield optimal performance.

The same codes were used to implement algorithmic blocking, and storage blocking with and without lookahead. The run-time choice of the parameters ω, r, s and λ determined which method was used. The DBLAS distributed BLAS library [13] 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.

For lookahead, the lower panel (eg. L^i for LU) was broadcast separately using a ‘spread’ operation; a column-replicated matrix descriptor was then constructed and passed to any DBLAS procedure using it (eg. step -(LU.4)), where a descriptor for the non-replicated L^i would have been used for pure storage blocking. For algorithmic blocking, redundant communications were avoided. In the case of LU, the rows of U^i , broadcast within step -(LU.3), were ‘cached’ and passed to the DBLAS operation corresponding to step -(LU.4); a similar scheme was used for saving broadcasts within steps -(LLT.2) and -(QR.2).

For LU factorization, the lookahead was implemented only over step -(LU.4), (although it could just as well have included -(LU.3) also). This meant that on iteration i , cell q_i kept any buffers not only for L^i but U^i also, to be used and released upon the next iteration. No alignment problems with $r \neq s$ arise in the LU computation for the DBLAS implementation, enabling hybrid lookahead and algorithmic blocking (ie. lookahead on the horizontal processor dimension, and algorithmic blocking on the vertical) to be implemented, by merely setting $r = 1, \omega = s$.

For LLT factorization, the fact that A^i is triangular means achieving an effective lookahead of λ^i block columns of a *rectangular* matrix was a little more complicated. Thus can be achieved by looking ahead λ^i block columns of the triangular A^i , where λ^i can be derived as follows. The local size of A^i on a cell on column q is $m^i \times n_q^i$. Noting that the average slope of A^i is Q/P , then $m^i \approx n_q^i Q/P$, and the desired condition is achieved when $\frac{1}{2} \lambda^i \times (\lambda^i Q/P) = m^i \lambda^i$. λ^i can then be computed by:

$$\lambda^i = (2n_q^i)^{\frac{1}{2}}$$

A^i being triangular also means that effectively for the last $2Q$ iterations, no significant lookahead can be applied. With $r \neq s$, alignment problems occur in step -(LLT.3), where block alignment is required for the efficient selection of rows of the column-replicated L^i that will contribute to $(L^i)^T$ for the current processor column. For this reason, hybrid lookahead and algorithmic blocking was not implemented for LLT.

For QR factorization, lookahead was at first only performed over step -(QR.5); as this led to sub-optimal performance for all but rather large matrices, lookahead was then performed over steps -(QR.3) and -(QR.4) also. Here, it should be noted that both V' and T effectively comprise the lower panel, and both must be broadcast before step -(QR.3) to achieve proper lookahead. Step -(GR2.2) has potential alignment problems if $r \neq s$: t^j , which is initially within one cell of the column, must be aligned with the both the rows and columns of T^j for the efficient implementation of $t^j \leftarrow T^j t^j$. One solution, sufficient for this case, was to regard T^j and t^j as having $r \times r$ blocks distributed over a $P \times 1$ grid⁴; this is valid as they are both contained in a single cell column, for which the value of s is irrelevant.

For hybrid lookahead and algorithmic blocking, a relatively small amount of extra communication startups are introduced for LU factorization in step -(LU.3). In the case of QR, extra startups are introduced in step -(GR2.2), to transpose and then reduce t_j . However, the latter are not so serious in that they are from the formation of the lower panel, and so can be parallelized if the degree of lookahead is sufficient.

3.1 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 [2, 13, 15]. As previously mentioned, lookahead requires the broadcast of the lower panel to be performed separately, which considerably complicates the implementation, and furthermore requires the explicit computation of local array sizes and offsets.

Another source of complexity with lookahead is the need to free buffers allocated in previous iterations of the computation. To detect errors such as memory leaks or trying to free from invalid pointers, which

⁴This was achieved by ‘forging’ the DBLAS matrix descriptors for T^j and t^j in such a way.

can easily arise in such a situation, we have found a ‘defensive programming’ front-end to the dynamic memory allocation and free calls to be very useful.

It was also necessary to measure the time spent in the broadcast receive calls for the lower panel, to check whether in cell (column) $Q - 1$ the messages had always arrived by the time that the call was posted (which indicates that maximal lookahead is being achieved). This revealed some subtle performance bugs in the implementation of lookahead in LU and QR factorization.

As lookahead requires pipelining, it requires the send operation for the lower panel to complete on the sending cell regardless of the state of the receiver, even when the lower panel becomes very large. This is not necessarily true of all implementations of the underlying message libraries on all multicomputers. In particular, the MPI Standard does not require such a property in MPI implementations.

Finally, in terms of applicability, lookahead can be applied to the improved load balance on the lower panel in matrix factorizations, but not on the upper panel. Furthermore, for some dense linear algebra algorithms, such as symmetric tridiagonal reduction, lookahead cannot be used since there are transposition operations and the summing of vectors across both grid dimensions, which prohibit pipelining [11]. In these circumstances, algorithmic blocking may be required to improve load balance.

4 Results

Figures 4, 5, and 6 give a comparison of the actual performance of lookahead, hybrid lookahead and algorithmic blocking, algorithmic blocking and storage blocking on the Fujitsu AP1000 and AP+.

The DBLAS implementation of these algorithms allows local matrix storage to be either row or column major [13, 15]. For these experiments, the best of these was chosen, ie. row major for LU and QR on the AP1000, and column major for all other situations.

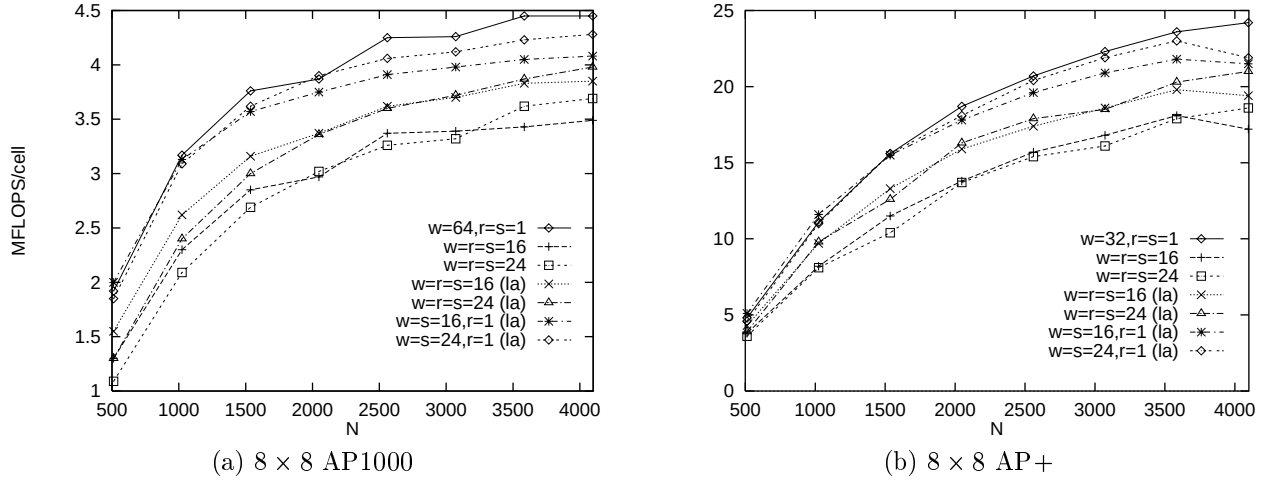


Figure 4: Speed in MFLOPs/cell of algorithmic blocking and storage blocking with and without lookahead (1a) for LU factorization on $N \times N$ matrices

Square processor configurations were chosen, as these have been found to be optimal for these machines [12]; a value of $P = 8$ was the largest available for both machines.

For such grid sizes, the range $512 \leq N \leq 4096$ was chosen. The lower bound of $N = 512$ is the smallest 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 [15]. For both the AP1000 and AP+ machines used for these experiments, $N = 8192$ represents the limit of available memory. Time constraints, especially on the AP1000, have rather caused us to set the upper bound to be $N = 4096$; however, the trends are fairly predictable from the range $3072 \leq N \leq 4096$, except that larger values of s (eg. $s = 32$) may begin to perform slightly better here. Section 5 reports some LU results for larger matrix sizes.

For the methods relying on the storage block size for their performance, the block sizes performing best at either end of the range of N were chosen. This means that an intermediate block size generally performed slightly better at the crossover points. Algorithmic blocking, on the other hand, is largely insensitive to the value of ω , provided $48 \leq \omega \leq 88$ on the AP1000, and $24 \leq \omega \leq 40$ on the AP+, even for small matrix sizes. The best blocking factor ω has been chosen for the results.

For methods using lookahead (denoted ‘(1a)’ in the Figures), a value of $\lambda = 4$ was experimentally found to be optimal for all situations and used for the results. On the AP1000, indeed a value of $\lambda = 2$ was in all cases just as fast, as predicted by Table 1 and Property 7, and had to be used for experiments where $N \geq 3072$ and $s \geq 24$ to avoid ring buffer overflow. On the AP+, $\lambda = 4$ was near-optimal.

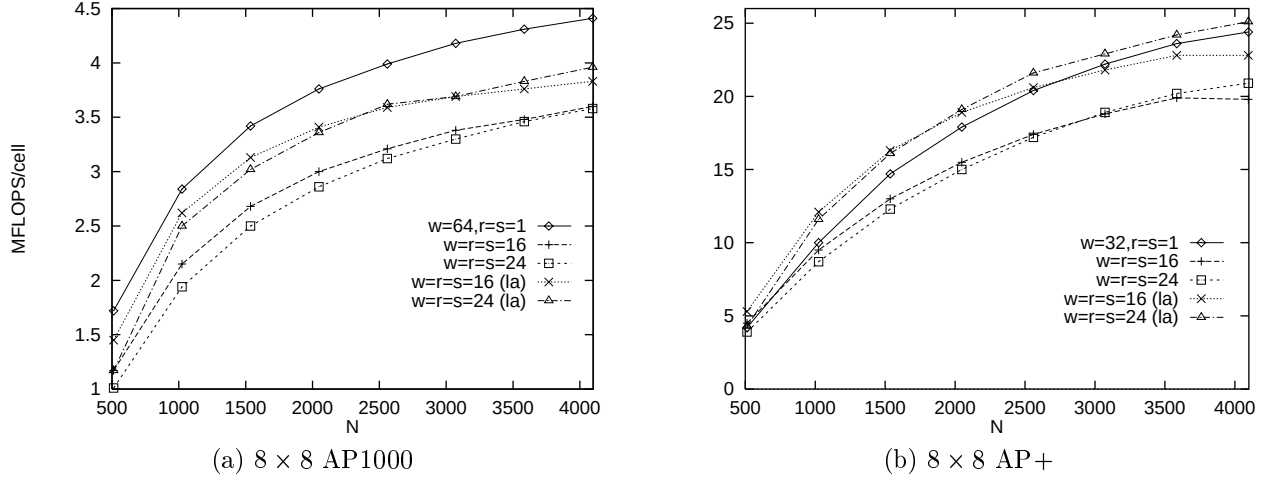


Figure 5: Speed in MFLOPs/cell of algorithmic blocking and storage blocking with and without lookahead (1a) for LLT factorization on $N \times N$ matrices

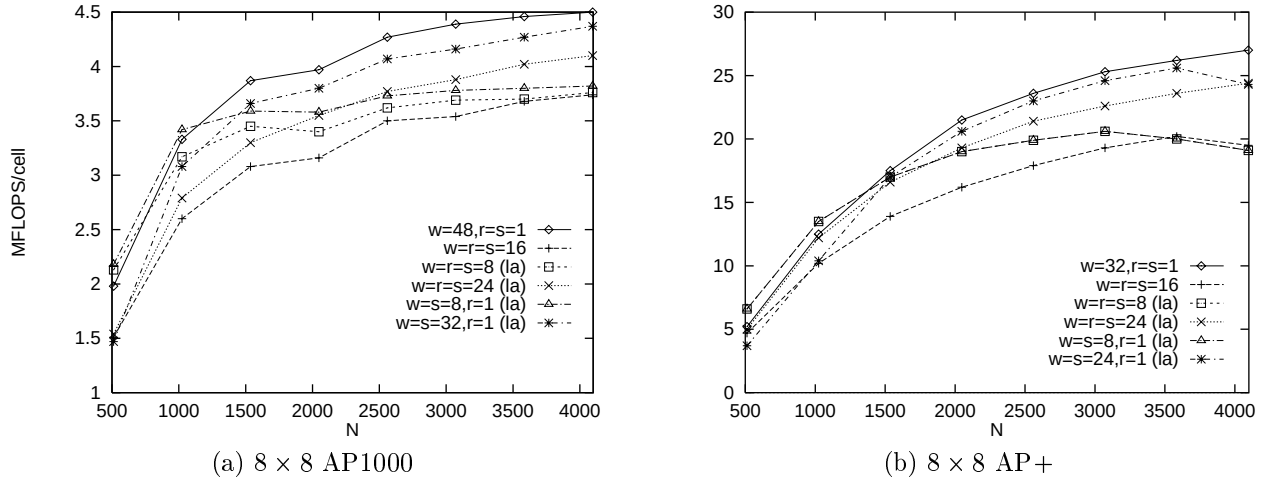


Figure 6: Speed in MFLOPs/cell of algorithmic blocking and storage blocking with and without lookahead (1a) for QR factorization on $N \times N$ matrices

For QR factorization, the $O(\omega/N)$ redundant floating point operations introduced by block-partitioned algorithms favors small blocking factors, at least for small N . For this reason, smaller values of s and ω was found to be optimal for QR than for LU or LLT.

The overall trends are as follows: algorithmic blocking out-performed pure storage blocking by about 25–30% for moderate to large N . For small N , the difference can be slightly larger, except that it is eroded for the case of the AP+ (higher $\frac{\alpha}{\gamma_3}$), and for LLT (a larger proportional amount of communication startups are introduced by algorithmic blocking), and to a lesser extent for QR (redundant computations arguing for a smaller ω).

Lookahead with $r = s$ generally made up half of this difference for moderate to large N , except for LLT on the AP1000 and QR on the AP+, where it made up 2/3 of the difference, and LLT on the AP+ where it actually out-performed algorithmic blocking. For small N , lookahead was competitive with algorithmic blocking.

Hybrid lookahead with algorithmic blocking (with $r = 1$), was the fastest for small N , and still very competitive with algorithmic blocking for larger N . The range at which it was faster is rather small because of the ratio of communication to computation speeds of these machines is unusually high; this point we will revisit in a more general context in the next section.

5 Analysis of Lookahead and Algorithmic Blocking Techniques

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. It will be validated on the AP1000 and AP+, and then used to predict how the methods should compare on a machine of lower communication to computation speed ratios.

In the case of lookahead, the model is essentially equivalent to that of [6], at least for large N/P and Q . However, notationally, the model is an extension of the DLAM model [2], 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 $Q - 1 \approx Q$. As previously mentioned, κ_{bc} is the cost of a broadcast where pipelining can be used; κ'_{bc} will denote the cost of a broadcast across P cells where pipelining cannot be used; thus $\kappa'_{bc} = 1$ if wormhole broadcasts are available, and $\kappa'_{bc} = \lg_2(P)$ otherwise.

The model will be developed first for the case of storage blocking ($\omega = r = s, \lambda = 0$); the subscripts for the t variables reflect the steps in Figure 1, with the total time being given by $t = (t_{1,\alpha} + t_{1,\gamma_2}) + t_2 + t_3 + t_L + t_U + t_4$:

$$\begin{aligned} t_{1,\alpha} &= (\lg_2(P) + 2\kappa'_{bc} + 2)N\alpha, & t_{1,\gamma_2} &= \frac{N^2\omega}{2P}\gamma_2, & t_2 &= 4N\alpha + \frac{N^2}{Q}\beta', & t_3 &= \frac{N^2\omega}{2Q}\gamma_3^\Delta \\ t_L &= \kappa_{bc}\frac{N^2}{2P}\beta, & t_U &= \kappa'_{bc}\frac{N^2}{2Q}\beta, & t_4 &= \frac{2N^3}{3PQ}\gamma_3 + \frac{N^2}{2}\left(\frac{r}{Q} + \frac{s}{P}\right)\gamma_3 \end{aligned}$$

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_4 is the load imbalance term due to the block-cyclic distribution block sizes. Note also that the communication startup term in t_2 can be reduced by a factor of 2 if the required rows of U^i and W^i are concatenated before sending [1].

If a block size of $r = 1$ is used, the following terms need modification:

$$t_2^{r=1} = t_2/\Psi_{\beta',\omega,P}, \quad t_3^{r=1} = t_3/(PE_d(\omega, 1, P)) + \kappa_{bc}N\alpha, \quad t_U^{r=1} = \kappa_{bc}\frac{N^2}{2Q}\beta$$

Here, note that the upper panel U^i can be broadcast row by row using a ring broadcast in step -(LU.3), and that $\Psi_{\beta',\omega,P} \geq 1$ is the degree of parallelization possible with ω row swaps with the row indices equally distributed over P processors, taking into account that the effective communication cost β' is generally several times that due only to the hardware. $E_d(\omega, r, P) \in [P^{-1}, 1]$ is the load balance efficiency factor of a triangular update of width ω with a block size r distributed over P cells [12].

If a block size of $s = 1$ is used with algorithmic blocking, the following terms are changed:

$$t_{1,\gamma_2}^{s=1} = t_{1,\gamma_2}/(QE_d(\omega, 1, Q)), \quad t_L^{s=1} = (\kappa_{bc} + 1)N\alpha + t_L$$

The first term in $t_L^{s=1}$ is due to the broadcast of the pivot p_j followed by that of the column l^j (in steps -(LU.2.2), -(LU.2.4)). As these originate from the same cell column, the pipeline ‘bubble’ from p_j is hidden by that of l^j for the case of $\kappa_{bc} = 2$. Subsequent pivots in L^i must also be applied to previously buffered columns l^j , but this can be achieved without extra overhead by coalescing the columns into the messages of step -(LU.2).

N	512	1024	1536	2560	3584	5120	7168
AP, $\omega = 64, r = s = 1$	+15	+7	+5	+3	+2		
AP, $\omega = r = s = 16, \lambda = 0$	+24	+7	+3	+1	+6		
AP, $\omega = r = s = 16, \lambda = 4$	+20	+6	+1	-1	-2		
AP, $r = 1, \omega = s = 16, \lambda = 4$	+18	+5	+1	0	0		
AP+, $\omega = 32, r = s = 1$	+18	+9	+3	+2	+2	+0(26.4)	-1(28.2)
AP+, $\omega = r = s = 24, \lambda = 0$	+11	+0	+11	+3	+5	+6(21.5)	+4(23.8)
AP+, $\omega = r = s = 24, \lambda = 4$	+12	+4	+13	+4	+4	+6(23.8)	+4(25.6)
AP+, $r = 1, \omega = s = 24, \lambda = 4$	+13	+13	+9	+5	+4	+1(25.8)	-1(27.3)

Table 2: Percentage error between the performance model and the actual LU computation on on 8×8 AP1000 and AP+

If lookahead is used, only the term for the lower panel need be modified, if the overlap is complete, ie. if $N/Q \gg \omega$, then we have: $t_{1,\alpha}^{la} = t_{1,\alpha}/Q$ and $t_{1,\gamma_2}^{la} = t_{1,\gamma}/Q$. As previously mentioned, no overlap at all is possible for $N \leq Q\omega$. We will now determine at which point these modified terms should take effect.

Consider the case of $\lambda \geq \rho, r = s$. On the k th last set of Q iterations, $k = 1, 2, \dots$, the cell owning the current block to be factored has a local matrix of $m_k \times k$ blocks remaining, wherer $m_k = k \frac{Q}{P}$. The computations in the panel formation in this cell can be (completely) overlapped by the lookahead on the matrix multiply on the cell to the right, which has $k - 1$ block columns available, providing $m_k \omega^3 \gamma_2 \leq 2m_k(k - 1)\omega^3 \gamma_3$, which solves to $k \geq k_{\gamma_2}$, where:

$$k_{\gamma_2} = \rho + 1 \quad (14)$$

and ρ is defined according to Table 1.

The communication startups in the panel formation can also be (completely) overlapped with the remaining $(k - k_{\gamma_2})$ block columns available for lookahead on the matrix multiply provided $(\lg_2(P) + 2\kappa'_{bc} + 2)\omega\alpha \leq 2m_k(k - k_{\gamma_2})\omega^3 \gamma_3$, which solves to $k \geq k_\alpha$, where:

$$k_\alpha(k_\alpha - k_{\gamma_2}) = \delta \quad , \text{ where } \delta = \frac{(\lg_2(P) + 2\kappa'_{bc} + 2)\alpha P}{2\omega^2 \gamma_3 Q} \quad (15)$$

k_α can then be computed by taking the positive solution of the quadratic formula to Equation 15.

For the case, $r \leq s$, we can adjust this calculation by using an average panel height in the k th iteration, ie. $m_k = (k - 0.5) \frac{Q}{P}$.

Thus a good approximation for modeling lookahead is by applying t_{1,γ_2}^{la} ($t_{1,\alpha}^{la}$) only for the fraction of the panel computations where $k \geq k_{\gamma_2}$ ($k \geq k_\alpha$). This may be somewhat pessimistic as it does not take into account partial overlapping, but as $\frac{N}{\omega Q}$ exceeds k_{γ_2} (k_α) one can see that this fraction very quickly becomes close to unity.

The above performance model (without most of the simplifying assumptions used above, and with some correction for partial lookahead for small N) has been implemented in a computer program; the % difference between the model and the actual is given in Table 2. Here, a selection of values of N likely to have smaller cache miss effects was chosen. For the last 2 columns for the AP+, the speed in MFLOPs/cell is given for the model in parentheses. For the AP1000, a value of $\Psi_{\beta', 64, 8} = 2.3$ was used for $r = 1$; for the AP+, the same value was also used.

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 [11]); these are least for $r, s = 1$ and most for $r, s = 24$, due to the degree of optimization possible for each case in the block-cyclic indexing calculations, identified to be a major part of this overhead [15]. With this in taken into account, the model gives very accurate predictions, the error being well under 10% except for the last row for $1536 \leq N \leq 2560$.

Note that on the 8×8 AP1000, at $\omega = s = 16$, we have $k_{\gamma_2} = 2$, $\delta = 1.4$ and hence $k_\alpha \approx 2.5$ (corresponding to $N \approx 320$). On the 8×8 AP+, at $\omega = s = 24$, we have $k_{\gamma_2} = 3$, $\delta = 2.5$ and hence $k_\alpha \approx 3.5$ (this corresponds to $N \approx 700$). As this predicts, the benefits of lookahead are small for $N = 512$ on Figure 4(b).

This phenomenon becomes more marked on machines of higher $\frac{\alpha}{\gamma_3}$, such as the 16×286 ASCI supercomputer. Table 3 gives the predictions of the above performance model on this machine, using the

N	8000	16000	32000	64000	128000	235000
$\omega = 64, r = s = 1$	13	44	118	215	280	307
$\omega = r = s = 64, \lambda = 0$	11	33	79	145	212	257
$\omega = r = s = 64, \lambda = 8$	11	33	91	204	272	300
$r = 1, \omega = s = 64, \lambda = 8$	12	35	94	212	280	305
$\omega = r = s = 32, \lambda = 0$	13	39	89	148	193	217
$\omega = r = s = 32, \lambda = 4$	13	43	124	190	221	233
$r = 1, \omega = s = 32, \lambda = 4$	14	45	127	198	226	236

Table 3: Predicted speed in MFLOPs/cell on the Intel ASCI supercomputer

model parameters given in [6]. A value of $\Psi_{\beta', 64, 16} = 3.8$ was used here for $r = 1$. Note that the entry for the third row $N = 235000$ corresponds to the world record setting for the LINPACK MP benchmark, it corresponds to a time of 6310 seconds, within 3% of the measured value of 6465 [6]. For $\omega = 64$, we can derive from these parameters $\rho = 1.8, k_{\gamma_2} = 2.8, \delta = 1.0$ and $k_{\alpha} = 3.1$ (corresponding to $N = 57000$).

For large N , lookahead alone yields most of the performance improvement over storage blocking. For $\omega = 32$, we have estimated $0.75\gamma_3(32) = \gamma_3(64)$. Under this assumption, the Table indicates that this is a better block size to use for $N < 64000$, with the hybrid lookahead and algorithmic blocking giving the best performance for $8000 \leq N \leq 32000$.

6 Conclusions

Understanding the subtleties of lookahead can be greatly aided by a simulator program, such as the one produced by this work. This was also used to check some useful properties of lookahead, such as the minimal degree of lookahead for optimum performance, and the conditions where handshaking can be useful. These also predict that in certain circumstances, lookahead can be used to reduce the size of the bubble that forms from pipelining, effectively increasing the bandwidth for communicating the lower panel.

Our results and validated performance models for lookahead for matrix factorization indicate that hybrid lookahead and algorithmic blocking can give best performance for smaller N , with pure algorithmic blocking yielding the best performance for larger N . An exception was LLT, which favors lookahead more strongly than does LU or QR. The crossover point depends on several factors, but most importantly the communication to computation speed ratio of the machine. However, if this ratio is high, the range for where hybrid lookahead is superior becomes very wide, and that most of these benefits are achieved by lookahead alone. In any case, both lookahead and algorithmic blocking are effective load balancing techniques, and promise in most situations to be significantly faster than storage blocking. Lookahead can also overlap the communication startups and similarly software overheads in the formation of the lower panel, where the majority of such overheads occur for matrix factorizations. This is especially the case for QR, where this can reduce the startup overheads from $O(N)$ to $O(N/\omega)$. Unfortunately Equation 15 predicts this cannot occur for small $\frac{N}{Q\omega}$, where this would be most useful.

In terms of portable dense linear algebra library design, lookahead with its strong requirements for pipelined communication, extra implementation difficulties and limited scope for applicability may make it slightly less attractive than algorithmic blocking as a single technique to improve load balancing.

Future work includes investigation into lookahead into the various matrix reduction algorithms, and using lookahead to accelerate the lower panel formation when algorithmic blocking with $s < \omega$ is used. The latter is a similar situation to lookahead in triangular matrix updates (eg. step -(LU.3)); however, Equation 15 again predicts that the situations where this can achieve significant speedups in the total factorization may be few.

In general, lookahead can be applied to any parallel computation that occurs in regular stages, where at each stage, data required by all processors must be formed and broadcast by a predetermined processor (group). Provided that processor can defer enough of its computation from the previous stage, it can have its data ready for broadcast as soon as the other processors are ready to receive it, thus obtaining an improvement in load balance. An interesting area for further research is to see what applications outside

dense linear algebra can be made to fall in this category.

Acknowledgements

The author is greatly indebted to Ken Stanley for several interesting discussions on pipelining with lookahead and algorithmic blocking, from which arose this work. The author would also like to thank the Fujitsu Parallel Research Computing Facilities for the use of the 64 node AP+.

References

- [1] 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.
- [2] 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.
- [3] 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.
- [4] F. Desperez, S. Domas, and B. Tourancheau. Optimization of the ScaLAPACK LU Factorization Routine using Communication/Computation Overlap. Technical Report 96-17, Laboratoire de l'Informatique du Parallelisme, Ecole Normale Supérieure de Lyon, 1996.
- [5] C. Anderson et al. *LAPACK User's Guide*. SIAM Press, Philadelphia, 1992.
- [6] B. Greer and G. Henry. High Performance Software on Intel Pentium Pro Processors, or Micro-Ops to TeraFLOPS. In *Supercomputing 97*, November 1997.
- [7] 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.
- [8] 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.
- [9] A. Petitet. *Algorithmic Redistribution Methods for Block Cyclic Decompositions*. PhD thesis, University of Tennessee, Knoxville, December 1996. xv+193p.
- [10] A. Petitet. PBLAS (version 2.0 ALPHA) – alignment restriction free PBLAS using logical algorithmic blocking techniques. source codes, available from <http://www.netlib.org/scalapack/prototype/pblasV2alpha.tar.gz>, November 1997.
- [11] K. Stanley. *Execution Time of Symmetric Eigensolvers*. PhD thesis, University of California, Berkeley, 1997. viii+184p.
- [12] 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.
- [13] 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.

- [14] 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.
- [15] 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.

0	F0				0	F0				0	F0			
1	c0	c0	c0	c0	1	c0	c0	c0	c0	1	c0			
2	M0	M0	M0	M0	2	M0	M0	M0	M0	2	M0	c0		
3	M0	M0	M0	M0	3	M0	M0	M0	M0	3	M0	M0	c0	
4	M0	M0	M0	M0	4	M0	M0	M0	M0	4	M0	M0	M0	c0
5		M0	M0	M0	5		F1	M0	M0	5		F1	M0	M0
6		F1			6		c1	c1	c1	6		c1	M0	M0
7	c1	c1	c1	c1	7	M1	M0	M1	M1	7		M0	M0	M0
8	M1	M1	M1	M1	8	M1	M1	M1	M1	8		M0	c1	M0
9	M1	M1	M1	M1	9	M1	M1	M1	M1	9		M1	M1	c1
10	M1	M1	M1	M1	10		M1	F2	M1	10	c1	M1	M1	M1
11			M1	M1	11		c2	c2	c2	11	M1	M1	F2	M1
12			F2		12	M2	M2	M1	M2	12	M1		c2	M1
13	c2	c2	c2	c2	13	M2	M2	M2	M2	13		M1	M1	M1
14	M2	M2	M2	M2	14	M2	M2	M2	M2	14			M1	c2
15	M2	M2	M2	M2	15			M2	F3	15	c2	M2	M2	M2
16	M2	M2	M2	M2	16	c3	c3	c3	c3	16	M2		M2	M2
17				M2	17	M3	M3	M3	M2	17	M2	c2	M2	F3
18				F3	18	M3	M3	M3	M3	18	M2	M2		c3
19	c3	c3	c3	c3	19	F4	M3	M3	M3	19	c3	M2		M2
20	M3	M3	M3	M3	20	c4	c4	c4	M3	20	M3	c3		M2
21	M3	M3	M3	M3	21	M3	M4	M4	c4	21	F4	M3	c3	M3
22	M3	M3	M3	M3	22	M4	M4	M4	M4	22	c4	M3	M3	M3
23	F4				23	M4	F5	M4	M4	23	M3	M3	M3	M3
24	c4	c4	c4	c4	24	c5	c5	c5	M4	24	M3	c4	M3	
25	M4	M4	M4	M4	25	M5	M4	M5	c5	25	M4	M4	c4	c4
26	M4	M4	M4	M4	26	M5	M5	M5	M5	26	M4	F5	M4	M4
27		M4	M4	M4	27		M5	F6	M5	27		c5	M4	M4
28		F5			28	c6	c6	c6	M5	28		M4	M4	M4
29	c5	c5	c5	c5	29	M6	M6	M5	c6	29		M4	c5	M4
30	M5	M5	M5	M5	30	M6	M6	M6	M6	30		M5	M5	c5
31	M5	M5	M5	M5	31			M6		31	c5	M5	F6	M5
32			M5	M5	32				F7	32	M5		c6	M5
33			F6		33	c7	c7	c7	c7	33	M5		M5	M5
34	c6	c6	c6	c6	34	M7	M7	M7	M6	34		c6	M6	c6
35	M6	M6	M6	M6	35	F8	M7	M7	M7	35		M6		M6
36	M6	M6	M6	M6	36	c8	c8	c8	M7	36	M6	c6	M6	F7
37				M6	37	M7	M8	M8	c8	37	M6	M6		c7
38				F7	38	M8	F9	M8	M8	38	c7	M6		M6
39	c7	c7	c7	c7	39	c9	c9	c9	M8	39	M7	c7		M6
40	M7	M7	M7	M7	40	M9	M8	M9	c9	40	F8	M7	c7	M7
41	M7	M7	M7	M7	41		M9	F10	M9	41	c8	M7	M7	M7
42	F8				42	c10	c10	c10	M9	42	M7	c8	M7	
43	c8	c8	c8	c8	43	M10	M10	M9	c10	43	M8	M8	c8	
44	M8	M8	M8	M8	44			M10	M10	44		F9	M8	c8
45		M8	M8	M8	45					45		c9	M8	M8
46		F9			46	c11	c11	c11	c11	46		M8	c9	M8
47	c9	c9	c9	c9	47	M11	M11	M11	M10	47		M9	M9	c9
48	M9	M9	M9	M9	48	F12			M11	48	c9		F10	M9
49			M9	M9	49	c12	c12	c12	c12	49	M9		c10	M9
50			F10		50		M12	M12	M12	50			M9	c10
51	c10	c10	c10	c10	51					51	c10		M10	M10
52	M10	M10	M10	M10	52	c13	c13	c13	c13	52	M10	c10	F11	F11
53				M10	53			M13	M13	53		M10		c11
54				F11	54			F14		54	c11			M10
55	c11	c11	c11	c11	55	c14	c14	c14	c14	55	M11	c11	c11	M11
56	M11	M11	M11	M11	56			M14		56	F12	M11		
57	F12				57			F15		57	c12			
58	c12	c12	c12	c12						58		c12		
59		M12	M12	M12						59		M12	c12	c12
60		F13								60		F13	M12	M12
61	c13	c13	c13	c13						61		c13		c13
62			M13	M13						62			M13	M13
63			F14							63			c14	
64	c14	c14	c14	c14						64				c14
65				M14						65				M14
66				F15						66				F15

(a) $\lambda = 0$, wormhole b/c

(b) $\lambda = 1$, wormhole b/c

(c) $\lambda = 2$, ring b/c

Figure 7: Visualization of lookahead with $\rho = \kappa = 1$ on a 1×4 grid, for a matrix of 16 block columns