A COMPARISON OF LOOKAHEAD AND ALGORITHMIC BLOCKING TECHNIQUES FOR PARALLEL MATRIX FACTORIZATION

P.E. STRAZDINS,

Department of Computer Science, Australian National University,

Acton ACT 0200, Australia

email: `peter@cs.anu.edu.au`

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. 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.

Results are given on the Fujitsu AP1000 and AP+ multicomputers. The results indicate that both methods are superior to storage blocking, and that the hybrid method is optimal for smaller matrices, due to savings in communication startups. For larger matrices, algorithmic blocking gave the best performance (excepting LLT for the AP+), due to its better load balancing properties. 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.

Key Words: dense linear algebra, matrix factorization, block cyclic decomposition, algorithmic blocking, pipelined communication, 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 provided $\omega \geq \omega_m$, the optimal panel width for cell computation speed.

In this paper, we will consider the $r \times s$ block-cyclic matrix distribution over a $P \times Q$ logical processor grid [1],

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*, where $\omega \approx \omega_m, r = s \approx 1$.

Storage blocking [2, 1] 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 $O(r/N)$ fraction of the overall computation. An imbalance of similar order occurs due to the storage block size in the Level 3 computation [3, 4], so that $\omega < \omega_m$ may be optimal. Algorithmic blocking [5, 6, 4] 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 of whether ω_m is large, and, to a lesser extent, whether the ratio of communication to floating point speed is relatively high [6].

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 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 [7]; 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 [3, 8] can be regarded as a related technique (arguably the ultimate of such techniques). 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. Lookahead was used to enhance a parallel LU factorization algorithm (with a validated performance model) which set world records for the LINPACK MP Benchmark on Intel supercomputers in 1995 [3] and 1997 [8]. Lookahead has been recently implemented to improve load balance for the triangular matrix update routines used in ScaLAPACK [9].

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 (which form the bulk of such overheads [10]) can be effectively parallelized.

The main original contributions of this paper are firstly to propose and develop a mathematical model for computations with lookahead, from which some useful properties of lookahead are established (Section 2). Secondly, to apply lookahead to two new computations, LLT and QR factorization (Section 3). 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 (Section 4), and theoretically by a performance model for LU factorization (Section 5). Such a model covering algorithmic

$\texttt{LU2}(N - i_1, \omega, L^i, p)$	$L^i = A_{i_1..N',i_1..i_2'},\ i_1 = i\omega,\ i_2 = i_1 + \omega$	-(LU.1)
$(W^i \vert U^i) \leftarrow (P(\omega', p_{\omega'}) \ldots P(0, p_0))(W^i \vert U^i)$	$W^i = A_{i_1..i_2',0..i_1'},\ U^i = A_{i_1..i_2',i_2..N'}$	-(LU.2)
$U^i \leftarrow (T^i)^{-1} U^i$	$T^i = L^i_{0..\omega',:}$ (lower tri., unit diag.)	-(LU.3)
$A^i \leftarrow A^i - L^i_{\omega..N'-i_1,:} U^i$	$A^i = A_{i_2..N',i_2..N'}$	-(LU.4)

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

for $j = 0..N'$

find $p_j : j..M'$ s.t. $\vert A_{p_j,j}\vert \geq \vert A_{j:M',j}\vert$		-(LU2.1)
$A \leftarrow P(j, p_j)A$	(swap rows j and p_j)	-(LU2.2)
$l^j \leftarrow l^j / A_{j,j}$	$l^j = A_{j+1..M',j}$	-(LU2.3)
$L^j \leftarrow L^j - l^j u^j$	$L^j = A_{j+1..M',j+1..N'},\ u^j = A_{j,j+1..N'}$	-(LU2.4)

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 $0 \leq i < \frac{N}{\omega}$

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.

1.1 Algorithms

In this section, the block-partitioned matrix factorization algorithms are described, being essentially simplified versions of those used in LAPACK and ScaLAPACK [1].

Fig. 1 gives the indicates the ith partial LU factorization with row pivoting, using a panel width of ω, for an $N \times N$ matrix A. This is repeated for $i = 0, 1, 2, \ldots, \frac{N}{\omega} - 1$ 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 Sections 5.

Figs. 2 and 3 give corresponding algorithms for partial LLT and QR factorizations, respectively. In the parallel implementation of the algorithm in Fig. 7, 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 [1] for the Intel Paragon, we have found this to be the case on the AP+ and AP1000, although only by a narrow margin for $\omega > 32$.

$\texttt{LLT2}(\omega, T^i)$ — $T^i = A_{i_1..i_2',i_1..i_2'}$ (lower tri., non-unit diag.) -(LLT.1)

$L^i \leftarrow L^i(T^i)^{-T}$ — $L^i = A_{i_2..N',i_1..i_2'},\ i_1 = i\omega,\ i_2 = i_1 + \omega$ -(LLT.2)

$A^i \leftarrow A^i - L^i(L^i)^T$ — $A^i = A_{i_2..N',i_2..N'}$ (lower tri.) -(LLT.3)

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

for $j = 0..N'$

$A_{j,j} \leftarrow \sqrt{A}_{j,j};\ \ l^j \leftarrow l^j/A_{j,j}$ — $l^j = A_{j+1..N',j}$ -(LLT2.1)

$L^j \leftarrow L^j - l^j(l^j)^T$ — $L^j = A_{j+1..N',j+1..N'}$ -(LLT2.2)

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

$\texttt{QR2}(N - i_1, \omega, V^i, \tau, V')$ — $V^i = A_{i_1..N',i_1..i_2'},\ i_1 = i\omega,\ i_2 = i_1 + \omega$ -(QR.1)

$\texttt{GR2}(N - i_1, \omega, V^i, V', \tau, T)$ -(QR.2)

$W \leftarrow V'^T A^i$ — $A^i = A_{i_1..N',i_2..N'}$ -(QR.3)

$W \leftarrow TW$ — (T is lower tri., non-unit diag) -(QR.4)

$A^i \leftarrow A^i - V'W$ -(QR.5)

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

for $j = 0..N'$

$(A_{j,j}, \beta, \tau_j) \leftarrow f(A_{j,j}, v^j \cdot v^j)$ — $v^j = A_{j+1..M',j}$ -(QR2.1)

$v^j \leftarrow v^j/\beta$ -(QR2.2)

$V'_{0..j',j} \leftarrow 0, V'_{j,j} \leftarrow 1, V'_{j+1..M',j} \leftarrow v^j$ -(QR2.3)

$w \leftarrow (V'_{j..M',j})^T A_{j..M',j+1..N'}$ -(QR2.4)

$V^j \leftarrow V^j - \tau_j v^j w$ — $V^j = A_{j+1..M',j+1..N'}$ -(QR2.5)

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

for $j = 0..N'$

$t^j \leftarrow -\tau_j (V'_{j..M',j})^T A_{j..M',0..j'}$ — $t^j = T_{j,0..j'}$ -(GR2.1)

$t^j \leftarrow T^j t^j$ — $T^j = T_{0..j',0..j'}$ (lower tri., non-unit diag) -(GR2.2)

$T_{j,j} \leftarrow \tau_j$

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 / AP+

The Fujitsu AP1000 [11] is a SPARC 1-based multiprocessor; it's cells have a theoretical peak speed of 5.5 MFLOPS (double precision). The AP1000 cells have a 128KB direct-mapped copy-back cache, with $\omega_m \approx 64$. 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. $\kappa_{bc} = 1$); such a broadcast will be referred to as a *wormhole broadcast.* Wormhole broadcasts are very easy to implement in hardware: 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. They do *not* require any synchronization, in that the source does not have to wait till the other cells post their corresponding receive calls.

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 [10, 4]. 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 $\gamma_3^{\triangle} = 0.3\mu$s, $\gamma_3(16) = .25\mu$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 ($\omega_m \approx 32$). There is no second-level cache; this reduces level-3 and especially level-2 computation performance. The AP+ uses an identical communication network. On the AP+, benchmark programs have yielded values of $\alpha = 12\mu\mathrm{s}, \beta = 0.64\mu\mathrm{s}, \gamma_2 = 0.12\mu\mathrm{s}, \gamma_3^{\triangle} = 0.083\mu\mathrm{s}, \gamma_3(16) = .035\mu\mathrm{s}$ 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 Lookahead Concepts

This section presents a simplified model of lookahead, to illustrate how it can achieve good load balancing. Consider the following computation on an $m \times N$ 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 $Qs|N$):

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

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 been made available [9].

Consider the ith 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 its 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 $M(L^i, A^i)$. Fig. 4 gives time lines of the Q cells for this computation using pipelined broadcasts, with Mi representing a time unit being spent in $M(L^i, A^i)$ (and similarly for Fi and ci).

From Fig. 4 ($\lambda = 2$), 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 ith 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. For large matrices, this can cause message receive buffer overflow. This situation can be avoided by implementing *handshaking* [3], where in the case of a pipelined 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:

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

$$C^i = T_{q^i}^{i-1} + \rho \quad (2)$$

$$C_q^i = \max(T_q^{i-1}, C^i) \quad (3)$$

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

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:

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

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

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

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.

The following properties together state that implementing handshaking will cause no delay. Property 8 states that broadcasts from cell $Q-1$ will not be stalled by handshaking. Property 9 states that handshake stalls in other cells will not delay the critical path cell $Q-1$. These may be proven using Property 5.

$$C^{i+1} \geq C_q^i + \kappa \quad , \text{if } q^{i+1} = Q-1 \tag{8}$$

$$C_{Q-1}^{i+1} + \kappa \geq C^{i+1} \tag{9}$$

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

$$C_q^i = \begin{cases} C^i & , \text{if } q = q_i \\ \max(T_q^{i-1}, C_{(q-1)\%Q}^i + \kappa) & , \text{if } q \neq q_i \end{cases} \tag{10}$$

and, for handshaking:

$$C_q^i = \max(C_q'^i, C_{(q+1)\%Q}^{i-1} + \kappa) \tag{11}$$

where $C_q'^i$ is the expression on the right hand side of (10).

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 (ie. for $q_i = Q-1$), is given by:

$$C_{Q-1}^i - T_{Q-1}^{i-1} = \max(0, Q\kappa - \min(\lambda^i, \rho+1)) \tag{12}$$

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.

The above equations for T_q^i and C_q^i have been implemented in a simulator (computer program) and all of the above properties have been verified over an exhaustive test over 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 Fig. 4 and is a very useful tool for investigating lookahead. It is available at:

```
http://cs.anu.edu.au/~Peter.Strazdins/
projects/Lookahead
```

2.2 Relationship to Matrix Factorizations with Lookahead

The above model however strictly only applies to a computation such as $A \leftarrow AT^{-1}$ executed on an $1 \times Q$ grid of an ideal machine. For matrix factorizations on a real machine, several differences should now be noted. Firstly, for $P \times Q$ grids, provided the vertical and horizontal communications can occur independently (as it is in our implementation

	LU	LLT	QR
ρ	$\frac{\gamma_2}{2\gamma_3}$	$\frac{\gamma_3^{\triangle}}{\gamma_3}$	$\frac{3\gamma_2}{4\gamma_3}$
(on AP1000:)	(1.0)	(1.5)	(1.5)
(on AP+:)	(2.0)	(2.7)	(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

– see Section 3), the cell columns can be regarded as a unit. Secondly, only when the panel length is large enough, so that effects such as communication startups and software overheads in the formation of L are small, can ρ and κ can be regarded as true constants. Their values, neglecting effects such as the cost of the upper panel formation, are given in Table 1 (the values of the AP+ are reasonably typical for modern multiprocessors).

Thirdly, in matrix factorizations, the global matrix length m decreases by ω every iteration. The effect of this is that when the decrease occurs, events happening in the next iteration occur 'closer' to those in the previous. 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. Hence, the situations where handshaking may be required may be broadened, but the other properties mentioned above should not be significantly affected.

3 Implementation

In our implementation of lookahead, λ, ω, r, s and ω are run-time settable parameters which are passed down to a single routine for each kind of matrix factorization. Handshaking was not implemented, since as explained in the previous section, it was obviated by setting $\rho \leq \lambda < \rho + 1$, which still yielded optimal performance. The DBLAS distributed BLAS library [6, 10] was used to implement these algorithms.

For lookahead, the lower panel had to be broadcast separately, rather than within the DBLAS Level 3 routine; this increases code complexity significantly (including the explicit computation of local array sizes and offsets). 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) (and similarly for LLT and QR).

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 (this also was a source of increased code complexity).

Considerable care had to be taken to ensure the other communications did not interfere with the communication of the lower panel, and hence with lookahead. Indeed, it proved necessary to measure the time spent in the broadcast receive calls for the lower panel in cell (column) $Q - 1$, to check messages had always arrived by the time that the

call was posted (which would indicate that maximal lookahead is being achieved).

For LLT factorization, the fact that A^i is triangular means achieving an effective lookahead of λ'^i block columns of a *rectangular* matrix can be derived as follows. The local size of A^i on a cell on column q is $m^i \times n^i_q$. Noting that the average slope of A^i is Q/P, then $m^i \approx n^i_q Q/P$, 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:

$$\lambda'^i = (2n^i_q \lambda^i)^{\frac{1}{2}}$$

A^i being triangular also means that effectively for the last $2Q$ iterations, no significant lookahead can be applied.

For QR factorization, lookahead must be applied over both Level 3 operations (steps -(QR.3), -(QR.4) and -(QR.5)) for optimal performance. Also the lower panel V and the triangular reflector T must be broadcast at step -(QR.3).

Hybrid lookahead and algorithmic blocking did not require any significant extra coding: it could be invoked by merely setting $r = 1, \omega = s$. Block alignment restrictions in the DBLAS symmetric rank-k update operation disallowed $r \neq s$, so the hybrid method could not be implemented for LLT. For LU, a relatively small amount of extra communication startups are introduced in step -(LU.3). In the case of QR, extra startups are introduced in forming the triangular reflector matrix in step -(GR2.2), but these can be parallelized if the degree of lookahead is sufficient.

4 Results

Figs. 5, 6, and 7 give a comparison of the actual performance of lookahead, hybrid lookahead and algorithmic blocking, algorithmic blocking and storage blocking. 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. On the AP+, the same trends continue up to $N = 8192$, the limit of memory. Note that algorithmic blocking performance is largely insensitive to ω, provided $48 \leq \omega \leq 88$ on the AP1000, and $24 \leq \omega \leq 40$ on the AP+.

For methods using lookahead (denoted '`(la)`' in the Figures), a value of $\lambda = 4$ was experimentally found to be optimal for all situations and used for the results. Indeed, on the AP1000, a value of $\lambda = 2$ was in all cases just as fast, and had to be used for measurements where $N \geq 3072$ and $s \geq 24$ to avoid message receive buffer overflow. On the AP+, $\lambda = 4$ was near-optimal.

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 for 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 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.

5 Performance Analysis

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 [8], at least for large N/P and Q. 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 Fig. 1, with the total time being given by $t = (t_{1,\alpha} + t_{1,\gamma_2}) + t_2 + t_3 + t_L + t_U + t_4$:

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

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.

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

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

Here, note that the upper panel U^i can be broadcast row by row using a pipelined 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, with $E_d(32,1,8) \approx 0.8$ [6].

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 corresponding rows of these columns into the messages of step -(LU.2).

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: $t^{la}_{1,\alpha} = t_{1,\alpha}/Q$ and $t^{la}_{1,\gamma_2} = t_{1,\gamma}/Q$.

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 kth last set of Q iterations, $k = 1, 2, \ldots$, the cell owning the current block to be factored has a local matrix of $m_k \times k$ blocks remaining, where $m_k = k\frac{Q}{P}$. 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:

$$k_{\gamma_2} = \rho + 1 \tag{13}$$

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:

$$k_\alpha(k_\alpha - k_{\gamma_2}) = \delta \qquad \text{, where } \delta = \frac{(\lg_2(P) + 2\kappa'_{bc} + 2)\alpha P}{2\omega^2\gamma_3 Q} \tag{14}$$

k_α can then be computed by taking the positive solution of the quadratic formula to (14).

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

Thus a good approximation for modeling lookahead is by applying t^{la}_{1,γ_2} ($t^{la}_{1,\alpha}$) 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. For the AP1000 and AP+, an empirically obtained value of $\Psi_{\beta',64,8} = 2.3$ was used for $r = 1$.

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 [4]); 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 [10]. For $r = s = 24$ on the AP+, extra TLB misses on the pivoting stage caused some performance degradation. When these factors are taken into account, the model gives very accurate predictions.

Note that on the 8×8 AP1000, at $\omega = s = 16$, we have $k_{\gamma_2} = 2$ and hence $k_\alpha \approx 2.5$ (corresponding to $N \approx 320$). On the 8×8 AP+, at $\omega = s = 24$, we have $k_{\gamma_2} = 3$ 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 Fig. 5.

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

N	512	1024	1536	2560	3584
ω=64, r=s=1	+15	+7	+5	+3	+2
ω=r=s=16, λ=0	+24	+7	+3	+1	+6
ω=r=s=16, λ=4	+20	+6	+1	−1	−2
r=1, ω=s=16, λ=4	+18	+5	+1	0	0
ω=32, r=s=1	+18	+9	+3	+2	+2
ω=r=s=24, λ=0	+11	+0	+11	+3	+5
ω=r=s=24, λ=4	+12	+4	+13	+4	+4
r=1, ω=s=24, λ=4	+13	+13	+9	+5	+4

Table 2: Percentage error between the predicted and the actual LU performance on an 8×8 AP1000 (rows 1–4) and AP+ (rows 5–8)

$N(\times 10^3)$	8	16	32	64	128	235
$\omega=64, r=s=1$	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

given in [8]. A value of $\Psi_{\beta',64,16} = 3.8$ was used here for $r = 1$, and $E_d(64, 1, 286) \approx 0.1, E_d(64, 1, 16) \approx 0.8$. Note that the entry for the third row $N = 235000$ corresponds to the world record set for the LINPACK MP benchmark. It represents a time of 6310 seconds, within 3% of the measured value of 6465 [8]. For $\omega = 64$, we can derive $\rho = 1.8, k_{\gamma_2} = 2.8$ 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 $8000 \le N \le 32000$.

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 avoided and where lookahead can halve the cost of a pipelined broadcast.

Our results and validated performance models 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 wider. 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 potentially reduce the startup overheads from $O(N)$ to $O(N/\omega)$. Unfortunately (14) 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 has the disadvantages of a strong requirement for pipelined communication (not necessarily supported on all platforms), extra implementation complexities and a limited scope for applicability. For example, unavoidable explicit transpose operations in symmetric tridiagonal reduction [4] and in symmetric indefinite matrix factorization preclude lookahead. This may make it slightly less attractive than algorithmic blocking as a single technique to improve load balancing.

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.

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; however, (14) again predicts that the situations where this can achieve significant speedups in the total factorization may be few.

Acknowledgements

The author is greatly indebted to Ken Stanley, for several interesting discussions from which arose this work, and to the Fujitsu Parallel Research Computing Facilities for the use of a 64 node AP+.

References

[1] 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.

[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 J.J. Dongarra, K. Masden, and J. Wasiniewski, editors, *Applied Parallel Computing*, pages 95–106, Berlin, 1995. Springer-Verlag.

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

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

[5] 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.

[6] 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.

[7] 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 Superieure de Lyon, 1996.

[8] B. Greer and G. Henry. High Performance Software on Intel Pentium Pro Processors, or Micro-Ops to TeraFLOPS. In *Supercomputing 97.* ACM SIGARCH - IEEE Computer Society Press, November 1997.

[9] 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.

[10] 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.

[11] 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)*, pages 314–325, May 1993.

Biography

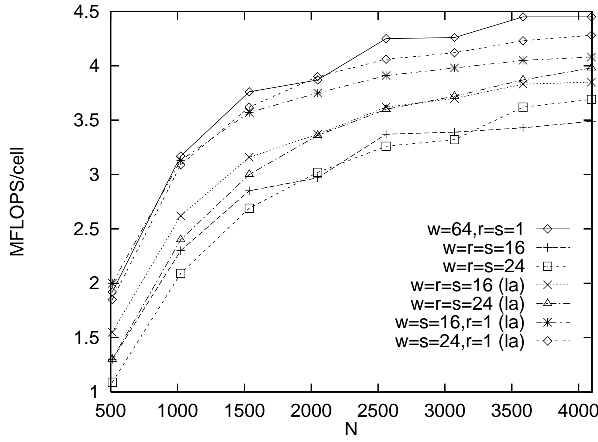
Peter Strazdins received a B.Sc. degree in Physics and a B.Math. degree in Computer Science from the University of Wollongong in 1984, an MSc. in Computation degree from the University of Oxford in 1985, and a PhD degree in Computer Science from the Australian National University in 1990. He is currently a Senior Lecturer in the

Department of Computer Science at the Australian National University, and is Research Leader the ANU-Fujitsu CAP Program in parallel computing. His research interests include computer architecture, operating systems and numerical algorithms for high-performance computing. Dr Strazdins is an Associate Member of IEEE.

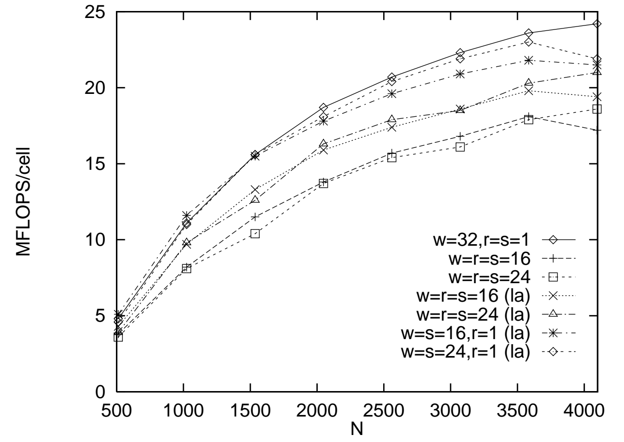
0	F0		
1	c0		
2	M0	c0	
3	M0	M0	c0
4	M0	M0	M0
5		M0	M0
6		M0	M0
7		F1	M0
8		c1	
9		M1	c1
10	c1	M1	M1
11	M1	M1	M1
12	M1		M1
13	M1		M1
14			F2
15			c2
16	c2		M2
17	M2	c2	M2
18	M2	M2	M2
19	M2	M2	
20	F3	M2	
21	c3		
22	M3	c3	
23	M3	M3	c3
24		M3	M3
25		M3	M3
26		F4	M3
27		c4	
28		M4	c4
29	c4	M4	M4
30	M4		M4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

0	F0		
1	c0		
2	M0	c0	
3	M0	M0	c0
4	M0	M0	M0
5		F1	M0
6		c1	M0
7		M0	M0
8		M0	c1
9	c1	M1	M1
10	M1	M1	M1
11	M1	M1	F2
12	M1		c2
13	c2		M1
14	M2	c2	M1
15	F3	M2	M2
16	c3	M2	M2
17	M2	M2	M2
18	M2	c3	
19	M3	M3	c3
20	M3	F4	M3
21		c4	M3
22		M3	M3
23		M3	c4
24	c4	M4	M4
25	M4	M4	F5
26	M4		c5
27	c5		M4
28	M5	c5	M4
29	F6	M5	M5
30	c6	M5	M5
31	M5	c6	
32	M6	M6	c6
33		F7	M6
34		c7	M6
35		M6	c7
36	c7	M7	M7
37	M7		F8
38			c8
39	c8		M7
40	M8	c8	M8
41	F9	M8	
42	c9		
43		c9	
44		M9	c9
45		F10	M9
46		c10	
47			c10
48	c10		M10
49			F11

Figure 4: Lookahead with $\rho = \kappa = 1, Q = 3$ with $\lambda = 0$ AND $\lambda = 2$ for a matrix of 12 block columns

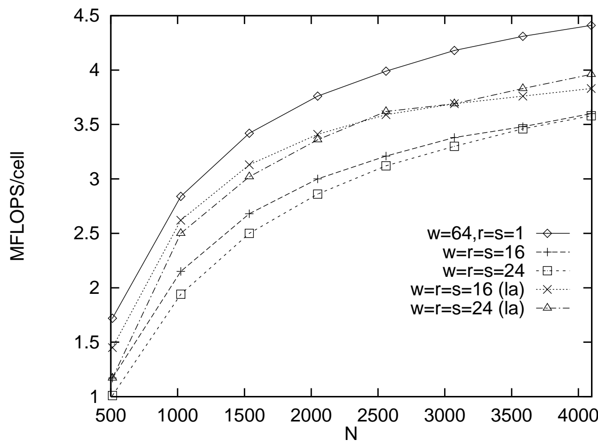


(a) 8×8 AP1000

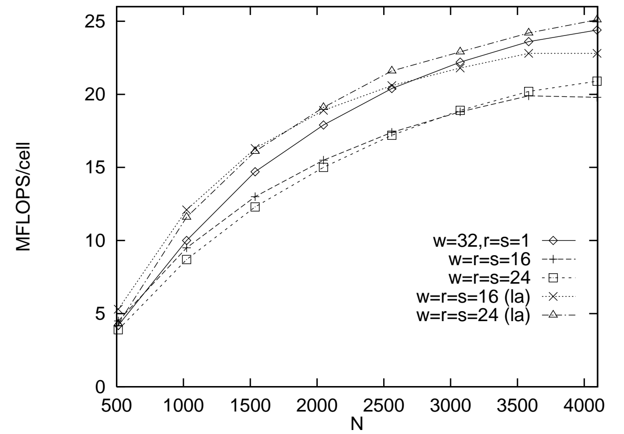


(b) 8×8 AP+

Figure 5: LU factorization speed on $N \times N$ matrices

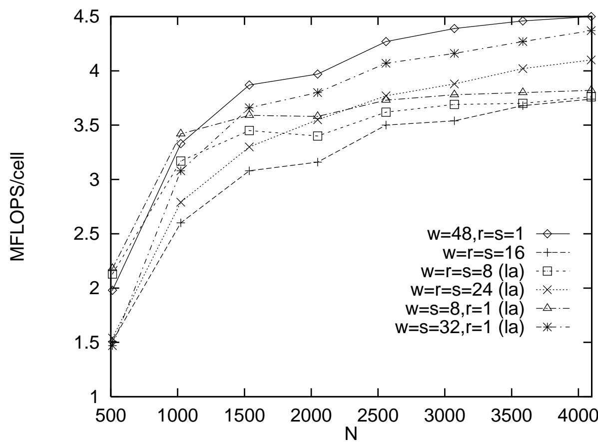


(a) 8×8 AP1000

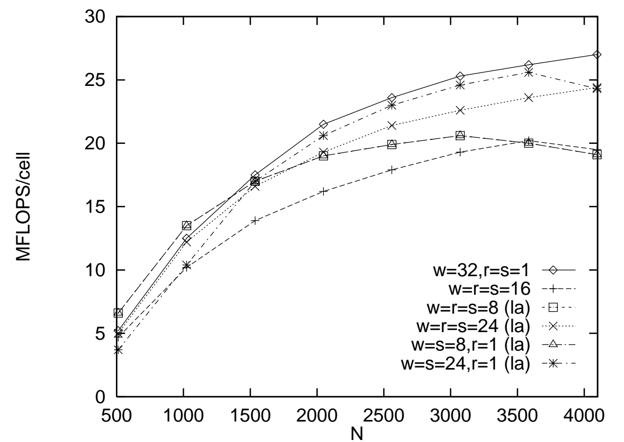


(b) 8×8 AP+

Figure 6: LLT factorization speed on $N \times N$ matrices



(a) 8×8 AP1000



(b) 8×8 AP+

Figure 7: QR factorization speed on $N \times N$ matrices