HIGH PERFORMANCE DENSE LINEAR SYSTEMS SOLUTION ON A BEOWULF CLUSTER

Peter Strazdins

Department of Computer Science,
Australian National University

`http://cs.anu.edu.au/~Peter.Strazdins` (`/seminars`)

5th Int'l Conference on High-Performance Computing in the Asia-Pacific Region
26 September 2001

1 Talk Outline

- introduction to parallel dense linear system solution
- background concepts:
  - pipelined communication
  - the block-cyclic matrix distribution
- block-partitioned LU factorization
  - basic algorithm
  - ||ization via algorithmic and storage blocking
  - optimizations for algorithmic blocking
- algorithm design for algorithmic blocking for LLT & QR factorization, and back-solve stage
- performance characteristics of optimized algorithmic and storage blocking
- performance results on the ANU Beowulf cluster
- conclusions

2 Introduction

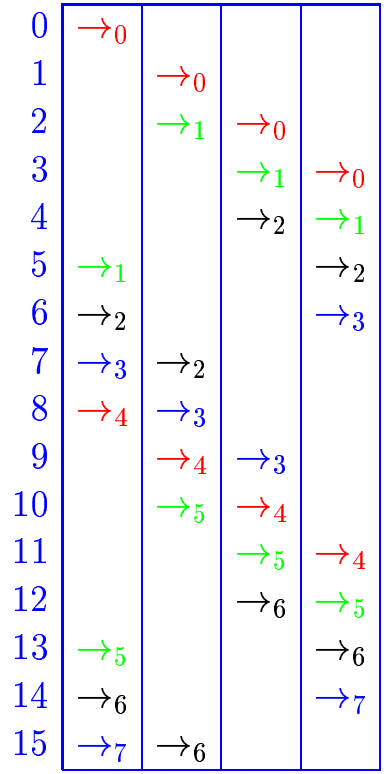
- dense linear system solution of $Ax = b$ via matrix factorization
  - LU: $A \rightarrow LU$, L is lower triangular, U is upper triangular with unit diagonal
  - LLT: $A \rightarrow LL^T$, A is symmetric +ve definite, L is lower triangular
  - QR: $A \rightarrow QR$, Q is orthonormal, R is as for U
- used in many apps.; LINPACK Benchmark is the #1 scientific benchmark
- cluster computers: an increasingly popular supercomputing platform
  - $\surd$ use COTS components; highly cost-effective
  - $\times$ communication networks are relatively slow (high latencies, low bandwidth)
    - pipelined communication can reduce this effect
- what is the best technique for dense linear systems solution on clusters?
- what is the best advanced technique for parallel dense linear algebra libraries?

  - storage blocking: used in ScaLAPACK (UTK, 1995–8)
  - lookahead: LINPACK MP Benchmark (1997), portable HPL (UTK, 2000)
  - algorithmic blocking: used in DLAPACK (ANU, 1995–2000)

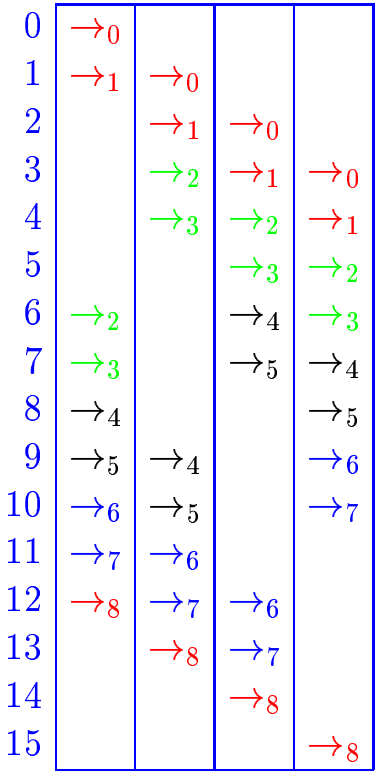
  issues: development effort & reliability, applicability and (portable) performance

3 Background: Pipelined Communication

- multiple row broadcasts (of equal size) across 4 processors:



(a) alternating sources, cost per b/c is $1 + 1$



(b) paired sources, cost per b/c is $1 + \frac{1}{2}$

- effectively the pattern of horizontal communication in matrix factorization

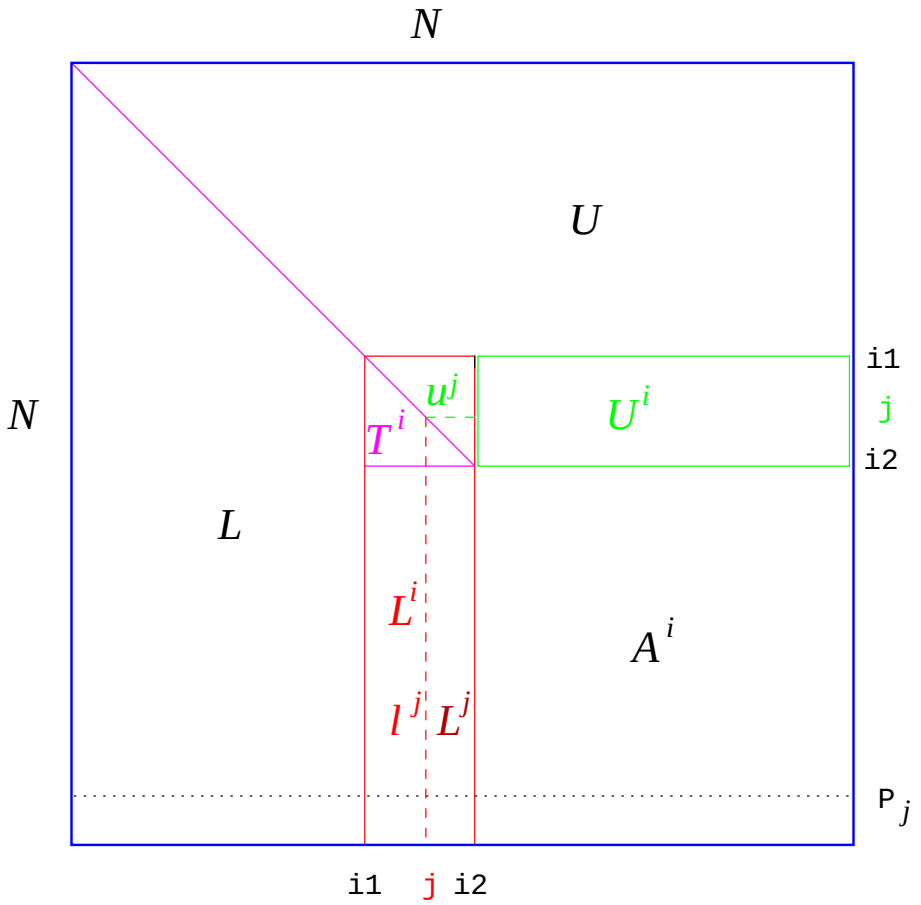
Background: the Block-cyclic Matrix Distribution

- 'standard' for most parallel dense linear algebra applications
  - good load balance for tri. & sub- matrices; r affects performance
- ie. divide global matrix A into $r \times s$ blocks on a $P \times Q$ processor grid;
  if 0th block is on proc. (p_0, q_0), block (i, j) of A is on proc. $((i+p_0)\%P, (j+q_0)\%Q)$
- eg. if $p_0 = q_0 = 0, r = 3, s = 2$ on a 2×3 grid, a 10×10 matrix A is distributed:

a_{00}	a_{01}	a_{06}	a_{07}	a_{02}	a_{03}	a_{08}	a_{09}	a_{04}	a_{05}
a_{10}	a_{11}	a_{16}	a_{17}	a_{12}	a_{13}	a_{18}	a_{19}	a_{14}	a_{15}
a_{20}	a_{21}	a_{26}	a_{27}	a_{22}	a_{23}	a_{28}	a_{29}	a_{24}	a_{25}
a_{60}	a_{61}	a_{66}	a_{67}	a_{62}	a_{63}	a_{68}	a_{69}	a_{64}	a_{65}
a_{70}	a_{71}	a_{76}	a_{77}	a_{72}	a_{73}	a_{78}	a_{79}	a_{74}	a_{75}
a_{80}	a_{81}	a_{86}	a_{87}	a_{82}	a_{83}	a_{88}	a_{89}	a_{84}	a_{85}
a_{30}	a_{31}	a_{36}	a_{37}	a_{32}	a_{33}	a_{38}	a_{39}	a_{34}	a_{35}
a_{40}	a_{41}	a_{46}	a_{47}	a_{42}	a_{43}	a_{48}	a_{49}	a_{44}	a_{45}
a_{50}	a_{51}	a_{56}	a_{57}	a_{52}	a_{53}	a_{58}	a_{59}	a_{54}	a_{55}
a_{90}	a_{91}	a_{96}	a_{97}	a_{92}	a_{93}	a_{98}	a_{99}	a_{94}	a_{95}

5 Blocked LU Factorization: algorithm

- uses Gaussian elimination with partial pivoting on an $N \times N$ matrix A



$i_1 = i\omega, i_2 = i_1 + \omega$

```
for (j=i1; i<i2; j++)
```

find $P[j]$ s.t. $|L^i{}_{P[j],j}| \geq |L^i{}_{j:M,j}|$

$L^i{}_{j,:} \leftrightarrow L^i{}_{P[j],:}$

$l_j \leftarrow l_j / L^i_{j,j}$

$L_j \leftarrow L_j - l_j u_j$

```
for (j=i1; j<i2; j++)
```

$A_{j,:} \leftrightarrow A_{P[j],:}$ (outside L^i)

(row broadcast of T^i, L^i)

$U^i \leftarrow (T^i)^{-1} U^i$

(column broadcast of U^i)

$A^i \leftarrow A^i - L^i U^i$

- LLT (Cholesky) factorization similar except $U^i = (L^i)^T$ and A^i is symmetric (lower triangular form)

6 LU Factorization: parallelization

- LU has $(\lg_2 P + 2)N$ commun. startups in forming L^i; (also $4N$ others)
  - cf. $2/3N^3$ FLOPs, $(2/Q + \frac{\lg_2 P}{2}/Q + 1/P)N^2$ commun. volume
- storage blocking: $\omega = r = s$:
  - ie. one processor column (row) holds L^i (U^i)
  - × suffers from load imbalance:
    - in the panel formation: of L^i: $O(N^2 s/P)$ (largest); of U^i: $O(N^2 r/Q)$
    - due to block size in $A^i \leftarrow A^i - L^i U^i$: $O(N^2(r/Q + s/P))$ FLOPs
- algorithmic blocking: uses optimal ω, $r = s \approx 1$
  - √ greatly reduces these imbalances
    - faster overall on some vendor-built multiprocessors of the mid 90's
  - × extra communication is now required to form L^i, U^i
    - a very unlikely candidate for clusters?
- LLT has fewer startups $(4N/r)$
- LLT & QR have higher block size load imbalances

LU Factorization: optimizations for algorithmic blocking

- use a pipelined broadcast for l_j and P_j: only an extra $(2+\frac{1}{r})N$ startups
  - do *not* try to 'block' r consecutive broadcasts (also too complex)
  - also cache these broadcasts (need not re-broadcast L^i and T^i)
  - now communication volume for L^i is $(1+\frac{1}{r})\frac{N^2}{2}$ (cf. storage blocking: $2 \cdot \frac{N^2}{2}$)
  - can be similarly applied to lower panel formation in LLT and QR
- similarly, use pipelined broadcasts and caching for rows of U^i in $U^i \leftarrow (T^i)^{-1}U^i$
  - extra $(1+\frac{1}{r})N$ startups but volume is now $(1+\frac{1}{r})\frac{N^2}{2}$ (cf. storage blocking: $\lg P \cdot \frac{N^2}{2}$)
- for the multiple row swaps $A_{j,:} \leftrightarrow A_{P[j],:}$
  - must also be applied to the cached l_j columns
  - minimize startups: pack all row segments into buffers before swaps
    - for column-major storage, can now optimize cache and TLB usage
    - results in significantly improved serial performance (now used in LAPACK)
  - for $\omega \geq rP$, multiple swaps will naturally ||ize!
    - by a factor of $\frac{P}{2}$ for contention-free networks (eg. switch-based clusters)

8 Algorithms for LLT & QR Factorization and Backsolve

- LLT & QR require formation of $\omega \times \omega$ tri. matrices (T^i)
  - can row- or fully-replicate T^i to minimize extra startups
  - some redundant computation, but this is small
- for QR, can re-arrange lower panel formation (of L^i):
  merge 2 vector-matrix multiplies & a dot product into 1 vector-matrix multiply
  - better computational speed (also for storage blocking and serial execution)
  - reduces associated startups from $6\lg_2(P)N$ to $2\lg_2(P)N$ (also for storage blocking)
  - algorithmic blocking now achieves perfect load balance
    - also does so in the upper panel formation
- for the backsolve, ie. $x \leftarrow b;\ \ x \leftarrow A^{-1}x$:
  - apply the factorization on the augmented matrix $(A|x)$
    - now need only perform $x \leftarrow U^{-1}x$ separately
  - can block communications by r here; thus requires only $O(\frac{N}{r})$ startups

Performance Characteristics: algorithmic vs. storage blocking

- performance factors for storage : algorithmic blocking ($r = 4$) for cluster with $P = Q = 8$ (assumes large $N, \omega \geq 4P$)

	LU	LLT	QR
startups ($\times N$)	5 : 8.5	0.0 : 2.8	6.0 : 9.8
volume ($\times N^2/P$)	3 : 1.5	2.5 : 2.1	4.0 : 2.1
panel load balance	0.1 : 0.7	0.1 : 0.7	0.1 : 0.9
panel FLOPs ($\times \omega N^2$)	1	0.5	3
block size imbalance ($\times r N^2$)	1	1	2
overall FLOPs ($\times N^3$)	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{4}{3}$

- here, algorithmic blocking:
  - need only introduce a relatively small number of extra startups
  - has significantly (except for LLT) lower communication volume
  - has better load balance (as expected)

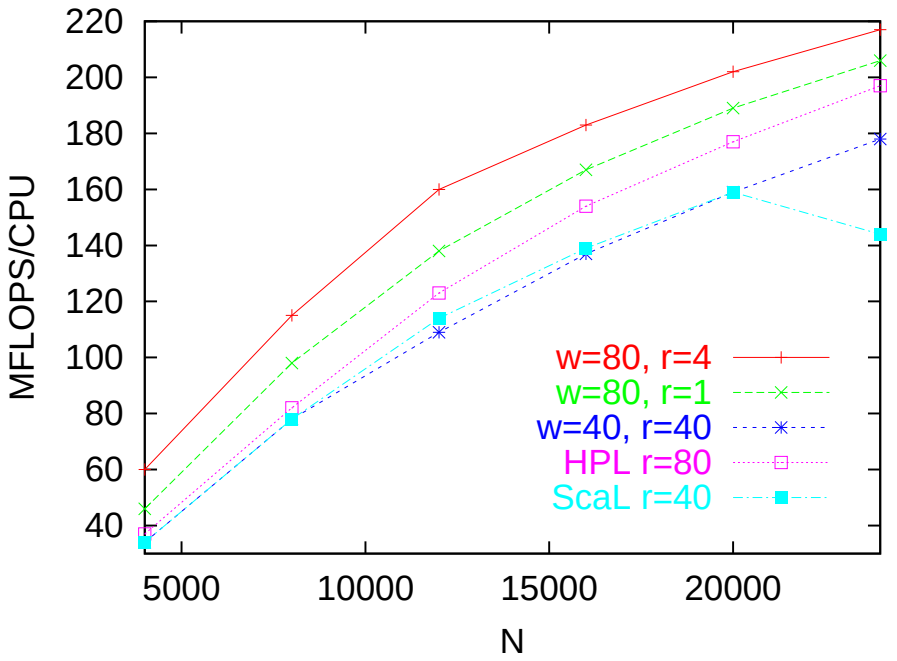
The ANU Beowulf Cluster and Serial Performance

- Bunyip has 4×24 550 MHz dual Pentium III nodes:
  - tetrahedral topology using four 48-port switches, 100 Mb/s max.
  - from LAM MPI: startup cost of 44μs, bandwidth of 6.5 MB/s
  - won 2000 Gordon-Bell Price/Performance Award (US 92c/MFLOPS)
- ATLAS 3.0 BLAS used for computation:
  - large matrix multiply: 377 MFLOPS ($\omega = 40$), 400 MFLOPS ($\omega = 80$, optimal)
    - large optimal ω due to size of the 256 KB direct-mapped L2 cache
  - rank-1 update: 100 MFLOPS; matrix-vector multiply: 175 MFLOPS
  - at $N = 3000, \omega = 80$:

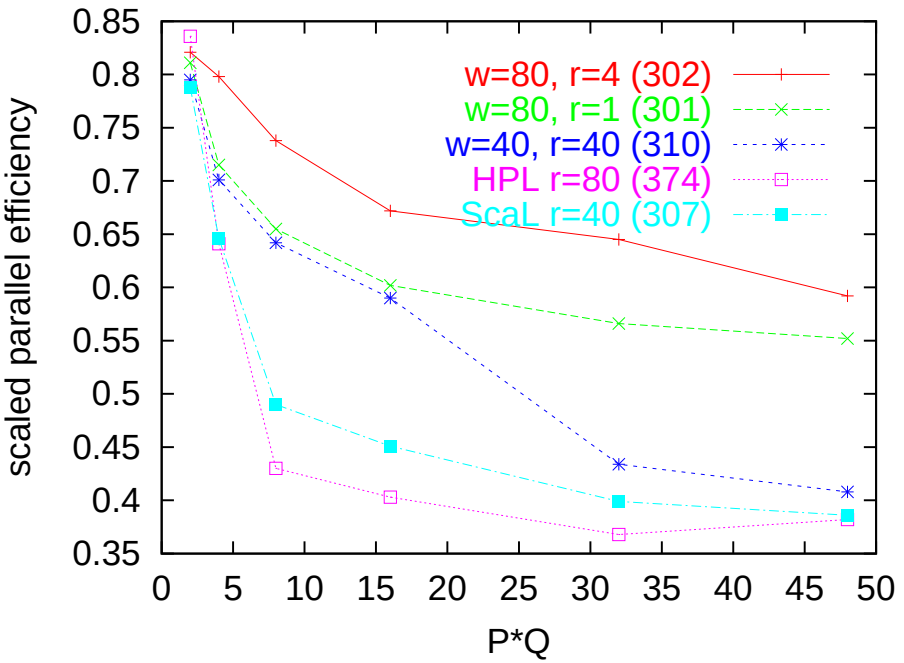
	LU	LLT	QR
MFLOPS	320	380	330

- for $||$ execution, run an MPI process on each CPU

Parallel Performance Comparisons



(a) LU Speed/CPU on a 6×8 grid



(b) LU $||$ efficiency at $\frac{N}{\sqrt{PQ}} = 2000$

- HPL uses recursive methods to form L^i: very fast computational speeds
- HPL also has aggressive communication optimizations
- apart from this, our storage blocking implementation seems optimal
  - algorithmic blocking: 76% faster for small N, 22% faster for large N
- optimal ω chosen for each case

- LLT: difference was less for small N (as expected)
  - for large N still about 20% faster (surprisingly); 200 MFLOPS/CPU here
- QR: trends similar to LU; 240 MFLOPS/CPU at large N
- similar performance benefits for algorithmic blocking on a 64-node Fujitsu AP+ and a 64-node Intel Paragon

12 Conclusions

- even with a slow communication network, high performance is possible for dense linear systems solve on a cluster
- algorithmic blocking was not an obvious candidate for a Beowulf:
  - with pipelined broadcasts with caching & ||ized row swaps, has less communication volume
    - minimized with a small storage block size of $r = 4$
    - this is the most decisive factor on the Bunyip
  - the number of extra communication startups can be kept small

  but achieved significant performance gains, especially for small N
- load balance effects in panel formation increase with increasing CPU speed
- algorithmic blocking has now been shown an effective technique on a large variety of platforms and linear algebra computations
- optimizations for LU and QR have also modestly improved serial performance
- future work: speed up lower panel formation in LU (to match HPL)