Parallel Matrix Computations: from Systolic Arrays to Supercomputers

Peter Strazdins,
Department of Computer Science,
The Australian National University

Symposium on Mathematics and Computer Science (RP60),
Australian National University, 20 April 2006

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

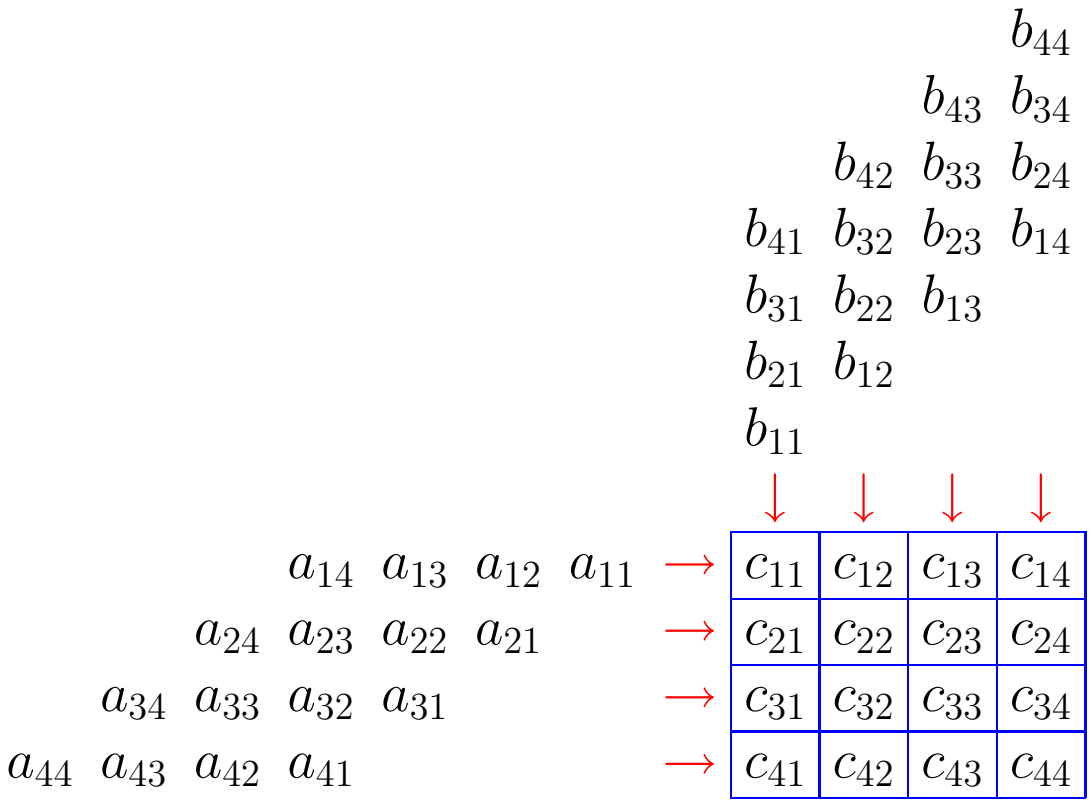
THE AUSTRALIAN NATIONAL UNIVERSITY

1 Overview

- parallel matrix computations on systolic arrays
- the emergence of large-scale general purpose parallel computers
- influence of systolic algorithms:
  - matrix multiply, pipelined communications, singular value decomposition
- the decompositional approach to matrix computations; matrix storage issues
- blocking and algorithm design issues (LU, LLT (Cholesky) & QR factorization)
  - storage and algorithmic blocking
  - the lookahead technique & other load balancing methods
  - developments and performance comparisons
- parallel LDLT factorization
- current status & conclusions

2 Systolic Arrays: Concepts

- pump data through arrays of simple processing elements (on a chip)
  - multiply-add unit, accumulator, communication channels
- example: matrix multiply $C \leftarrow C + AB$ on a 4×4 systolic array



3 Parallel Matrix Computations on Systolic Arrays

- emerged in the 70's for digital signal processing applications
- VLSI fabrication enabled widespread deployment over the 80's
- development of a wide range of (matrix) computations: [RPB's publications]
  - matrix multiply, polynomial GCD [77,82], filtering [103,104], triangular systems solution [41], matrix factorization (LU, Cholesky [74], Givens QR [65], Toeplitz [78,88]), singular values[80] & symmetric eigenvalues [84,86], . . .
  - highly esoteric; algorithms requiring pivoting problematic
- mid 80's: issue of programmability emerges
  - the instruction systolic array: pump instructions with the data (mid-80's H.W. Lang, Heiko Schroder & the Kiel group)
  - programming models developed, reduction of bandwidth requirements: SISA [Lang '87]; program compression [PES '90]
- found niches in special-purpose applications;
  potential for general-purpose parallel processing?

4 Large-scale General Purpose Parallel Computers

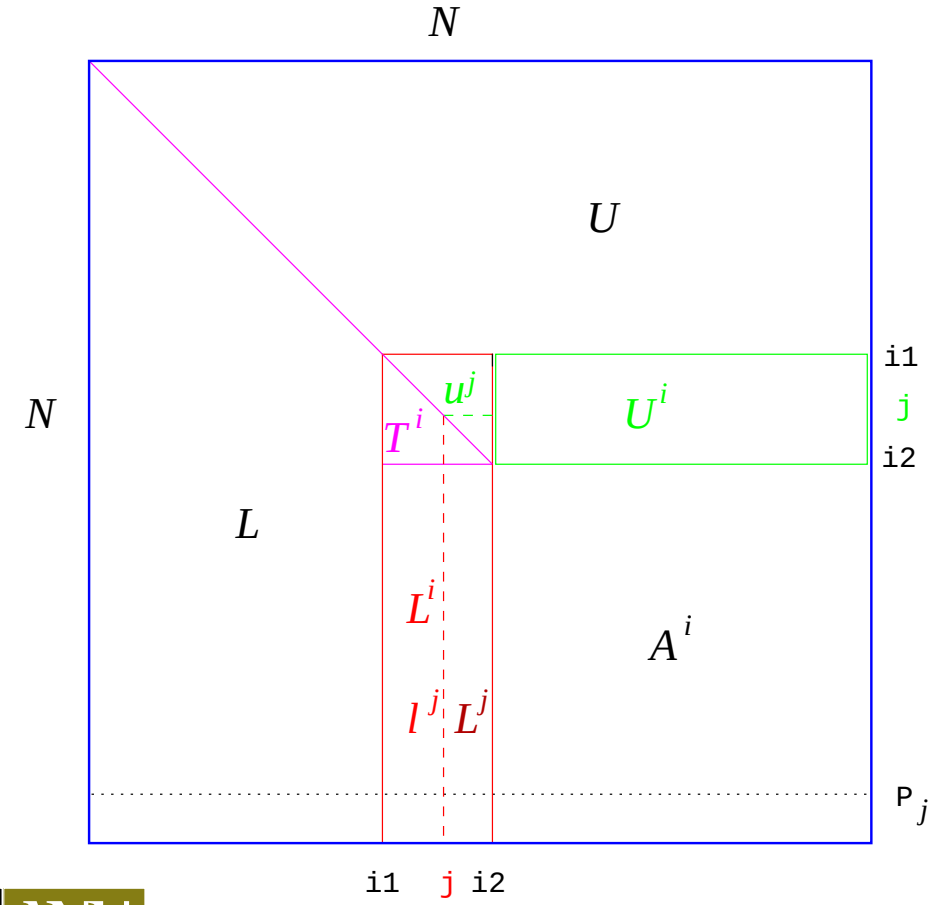
- a brief history:
  - mid 80's: scalable arrays of microprocessors connected by communication network developed
    - motivated by more general applications, e.g. rendering
    - e.g. the Cellular Array Processor ($\Rightarrow$ the ANU-Fujitsu CAP Program '89-'02)
  - early – mid 90's: displaced the vector-parallel processors as supercomputers
    - due to advent of the "killer micros"
  - late 90's – present: off-the-shelf solutions (clusters) gradually displace vendor solutions
- for matrix computations: a more flexible programming model
  - assign matrix (sub-) blocks to each processing element
    - amortize communication startup and (sometimes) volume costs
    - pass sub-blocks using messaging libraries

5 Influence of Systolic Array Algorithms on Parallel Computers

- e.g. systolic matrix multiply: easily adopted
  - problem: had to 'skew' matrices ($A \leftarrow$; $B \uparrow$) first [121]
  - generally displaced by broadcast-based algorithms
    - broadcast more costly than normal messages (but not the CAP! [121,136])
    - but inspired the pipelined broadcast: $2\times$ slower [ScaLAPACK group '93]
      - splitting sub-blocks on matrix multiply: $\approx 1\times$ slower; [Choi '95]
    - more general multi-b/c by rotation algorithm (DBLAS library [PES '98])
- Hestene's singular value decomposition algorithm
  - Brent-Luk systolic algorithm used a $1 \times \frac{N}{2}$ processor array: orthogonalize & exchange column pairs (NS steps) [80,84]:
  - also easily adopted using blocking techniques [Czezowski & PES '92,'94]
    - standard optimizations: software pipelining & cache blocking
    - 2-dimensional array to improve cache & communication re-use
    - 4-column simult. orthogonalization (performance-convergence trade-off)

6 The Decompositional Approach to Matrix Computations

- rated in the Top 10 “Algorithms” of the 20th Century CS&E, Jan 2000)
- e.g. blocked LU factorization with partial pivoting on an $N \times N$ matrix A (on step i, $i_1 = i\omega, i_2 = i_1 + \omega$)



1. factor the lower panel L^i:
   for j = $i_1..i_2$
   find $P[j]$ s.t. $|{L^i}_{P[j],j}| \geq |{L^i}_{j:M,j}|$
   ${L^i}_{j,:} \leftrightarrow {L^i}_{P[j],:}$
   $l_j \leftarrow l_j / L^i_{j,j}$
   $L_j \leftarrow L_j - l_j u_j$
2. do swaps for $P[i_1..i_2]$ outside L^i
3. row broadcast of T^i, L^i
4. $U^i \leftarrow (T^i)^{-1} U^i$
5. column broadcast of U^i
6. $A^i \leftarrow A^i - L^i U^i$

THE AUSTRALIAN NATIONAL UNIVERSITY

7 Matrix Storage on Parallel Computers

- the block-cyclic distribution is 'standard' for the decompositional approach (load balance on triangular & sub- matrices)
- divide the global matrix A into $r \times s$ blocks on a $P \times Q$ array;
  if block $(0,0)$ is on cell (p_0, q_0), block (i,j) is on cell $((i+p_0)\%P, (j+q_0)\%Q)$
- e.g. $p_0 = q_0 = 0, r = 3, s = 2$ on a 2×3 array, a 10×10 matrix A:

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}

- PBMD (Physically Based Matrix Distribution) is a special case, with $s = Pr$ (PLAPACK)
  - induced from a vector distributed over PQ cells
  - $\surd$ enables special algorithms
  - $\times$ s is large

8 Blocking Issues and Parallel Algorithm Design

- storage blocking: $(\omega = r = s)$ [van de Geijin '91], [Dongarra '94], [ScaLAPACK '95]
  - $\surd$ simplest to implement, minimizes number of messages
  - $\times$ suffers from load imbalance on panel formation:
    i.e. one processor column (row) holds L^i (U^i); also in $A^i \leftarrow A^i - L^i U^i$
- lookahead: [Henry '96] (MP Linpack record on ASCII Red)
  - $\surd$ eliminate imbalance in forming L^i by computing it in advance
  - $\times$ hard to implement; only applicable to some computations
- algorithmic blocking: uses optimal ω, $r = s \approx 1$ [130] '91
  - $\surd$ greatly reduces these imbalances
  - $\times$ introduces $4N$ extra messages; local panel width is small ($\approx \omega/Q$)
- panel scattering: redistribute L^i, U^i along logical $1 \times PQ$ array [PLAPACK '97]
  - $\surd$ gets perfect load balance in panel formation, local panel width is ω
  - $\times$ moderately complex; appears to require more communication

9 Algorithmic Blocking: Developments

- Fujitsu AP1000 (CAP) LU implementation ($r = s = 1$) [130,136]
  - analysis (+LLT,QR $r, s \geq 1$) and comparison with storage blocking [PES '95]
- implementation in ScaLAPACK symmetric eigenvalue [Stanley '97]
- outperformed ScaLAPACK by 40–14% (LU), -15–3% (LLT) & 29–10% (QR) on 64-node Paragon ($r = s = 4$) [PES '98]
- panel scattering slightly faster for LU & QR but *only* for $r = s \approx 1$ [PES '98]
  - PLAPACK used in conjunction with PBMD (for $P = 8$, $r = 16, s = Pr = 64 \Rightarrow$ slower)
- analysis of lookahead ($\Rightarrow$ faster pipelined broadcast) & comparison [PES '98]
- lookahead instead adopted for High Performance Linpack ('99)
- algorithmic blocking outperformed storage blocking on fast ethernet-based cluster (Bunyip) ($r = s = 4$); also HPL by 50–20% [PES '01]
  - key reason: extensive use of pipelined broadcasts

10 LDLT Factorization: the one ScaLAPACK missed

- solving dense indefinite symmetric linear systems arises in many applications, e.g. Electromagnetic Compatibility Analysis (moment method)
- which $\frac{N^3}{3} + O(N^2)$ FLOPS algorithm to parallelize?
  - tridiagonalization methods hard to parallize, whereas LDLT methods require much less (expensive) row/column pivoting
  - LAPACK version of Bunch-Kaufman LDLT algorithm was very fast!
- || alg. needed many optimizations; storage blocking slightly faster [PES '99]
  - further speedups from reduced pivoting (stability tradeoffs) [PES '00]
- solver also integrated into ANU Data Mining ToolBox [Christen et al '01]
- with better UltraSPARC BLAS, 3× faster EMC analysis on AP3000
  - ||ized $O(N^2)$ iterative solver (but direct solver still needed!) [PES & Fujitsu team '01,'02]
- stability problems with the B-K algorithm discovered! [Ashcraft, Grimes & Lewis '98]
  - ||ization of stable LDLT algorithms [PES & Lewis '01]; out-of-core [PES '03]

11 Current Status and Conclusions

- ScaLAPACK (mostly) uses storage blocking
  - √ stable, robust, reasonably wide functionality
    - sometimes sub-optimal – usually not a real issue
  - × user must understand block-cyclic matrix distribution
      - set up matrix descriptors; carefully choose storage block size
- PLAPACK uses potentially better algorithm; not widely used
- HPL is highly configurable & widely used
    - can be out-performed on some systems, especially those with relatively low communication bandwidth
- (AFAIK) open-source parallel LDLT algorithm still only exists within ANU Data Mining Toolbox
- field has come a long way from systolic algs.; some influence remains
- has been exciting to have participated!
    - some excellent opportunities for industrial collaboration