Reducing Software Overheads in Parallel Linear Algebra Libraries

Peter Strazdins

Department of Computer Science,

Australian National University

e-mail: peter@cs.anu.edu.au

1 Overview

- introduction to dense linear algebra libraries: LAPACK
- issues in parallel versions of these libraries: ScaLAPACK
- the challenge of parallel block-partitioned algorithms
  - example: block LU Factorization
- overview of two parallel LAPACK implementations: ScaLAPACK and DLAPACK
- techniques to reduce software overheads
- conclusions

2 Introduction to Dense Linear Algebra Libraries

- LAPACK is an extensive dense linear algebra library
- uses block-partitioned algorithms:
  - ie. partition the rows and/or columns of a matrix into panels of width $\omega \geq 1$
  - express as much of the computation as possible in terms of matrix-matrix operations using these panels
- these are expressed almost entirely in terms of the BLAS (Basic Linear Algebra Subroutines). Has operations of Levels:
  1. vector-vector: eg. $A_{i,:} \leftrightarrow A_{P[i],:}$ $\quad l_i \leftarrow l_i / A_{i,i}$,
  2. matrix-vector: eg. $L_i \leftarrow L_i - l_i u_i$
  3. matrix-matrix: eg. $U' \leftarrow T^{-1}U'$, $A' \leftarrow A' - L'U'$
     - have $\Theta(n^2\omega)$ operations using only $\Theta(n^2)$ data elements
     - optimal for memory hierarchy processors (enables high data reuse)
- key to performance & portability of LAPACK: use vendor-optimized BLAS
- approach extends well to parallel (distributed memory) versions

3 Issues in Parallel Dense Linear Algebra Libraries

- eg. ScaLAPACK (UTK/ORNL, 1993-), PLAPACK (Uni. of Texas, 1997-)
- ScaLAPACK performance goals include scalability:
  - determined by load balance and communication overhead issues
  - major design decision: use of the block-cyclic matrix distribution to minimize communication startup overheads (let $\omega = r = s$)
- software engineering goals include simplicity, maintainability:
  - strategy: make code as close as possible to LAPACK:
    - matrix structures and indices are global
    - hide communication inside a version of parallel BLAS (eg. the PBLAS)
- another issue, ignored until recently, is software overheads
  - eg. procedure call overheads, indexing calculations, error checking and structure manipulation
  - robustness and generality requirements of libraries $\Rightarrow$ these can be large!
    - eg. a specialized || eigensolver (Hendrickson, Jessup, Smith, 1996) out-performed the ScaLAPACK solver by 30% on *moderate* matrix sizes

3.1 The Block-cyclic Matrix Distribution

- 'standard' for most parallel dense linear algebra applications
- ie. divide global matrix A into $r \times s$ blocks on a $P \times Q$ grid;
  if 0th block is on cell (p_0, q_0), block (i, j) of A is on cell $((i+p_0)\%P, (j+q_0)\%Q)$
- 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}

- high potential for software overheads!

4 The Challenge of Parallel Block-Partitioned Algorithms

- for an $N \times N$ matrix on a $P \times P$ grid, can model || execution time by:

  $$t(N) = c_1 N\alpha + c_2 N^2/P\beta + c_3 N^3/P^2\gamma_3$$

  - β is the communication cost per word; γ_3 is the time produce a new f.p. result
  - α is the communication startup overhead:
    - can be very significant for small-moderate N/P
      - typically: $\frac{\alpha}{\gamma_3} \approx 10^3$, $c_1 = O(\lg_2 P) > c_2 > c_3$

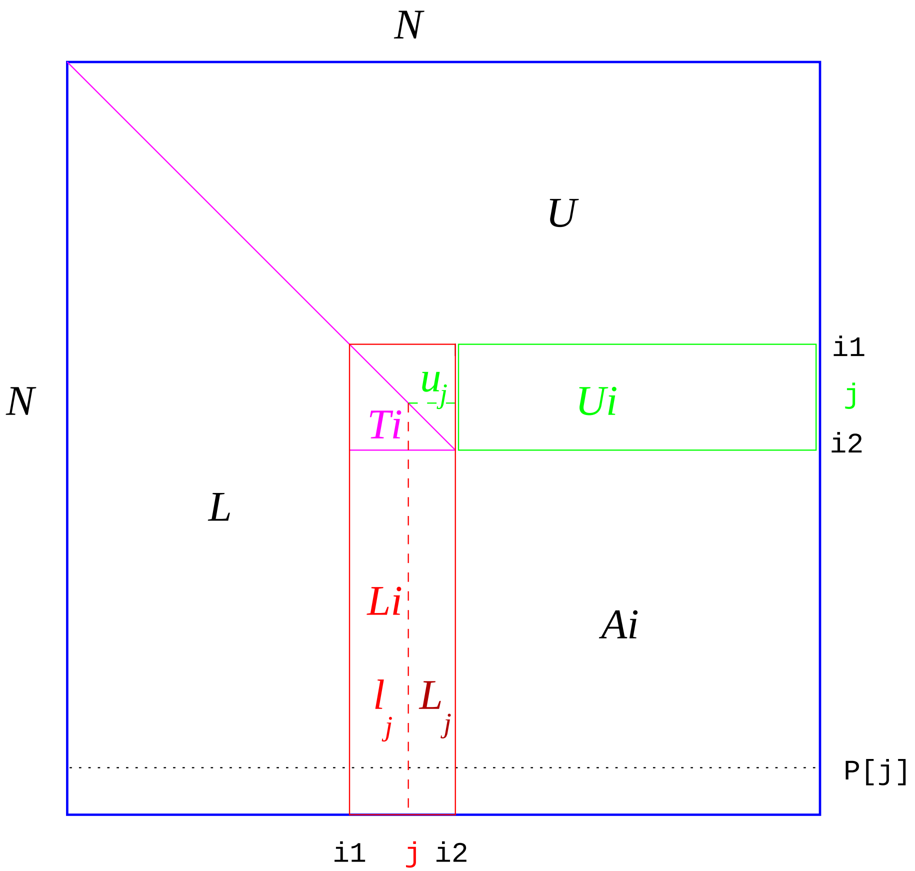
        eg. LU on 8×8 AP1000: $c_1 \approx 11, c_2 = 3, c_3 = \frac{2}{3}$
    - are mainly due to software

      eg. several layers of procedure calls, error checking, message header formation, buffer allocation

- software overheads:
  - are also $\Theta(N)$: require $\Theta(N)$ elementary (BLAS) operations
  - the added complexities of || programming greatly exacerbates their effects
    - eg. consider the sub-matrix operation `A(i, i+1)`

4.1 Example: Blocked LU Factorization

- uses Gaussian elimination with partial pivoting on an $M \times N$ matrix A
- rank-1 updates are blocked by the Level 3 factor ω



$$
\begin{array}{lcll}
l_i & = & \sigma(A, i', i) & : M{-}i' \times 1 \\
u_i & = & \sigma^T(A, i, i') & : 1 \times j_\omega{-}i' \\
L_i & = & \sigma(A, i', i') & : M{-}i' \times j_\omega{-}i' \\
\\
T & = & \sigma(A, j, j) & : \omega \times \omega \\
L' & = & \sigma(A, j_\omega, j) & : M{-}j_\omega \times \omega \\
U' & = & \sigma(A, j, j_\omega) & : \omega \times N{-}j_\omega \\
A' & = & \sigma(A, j_\omega, j_\omega) & : M{-}j_\omega \times N{-}j_\omega
\end{array}
$$

$$(i' = i + 1, j_\omega = j + \omega)$$

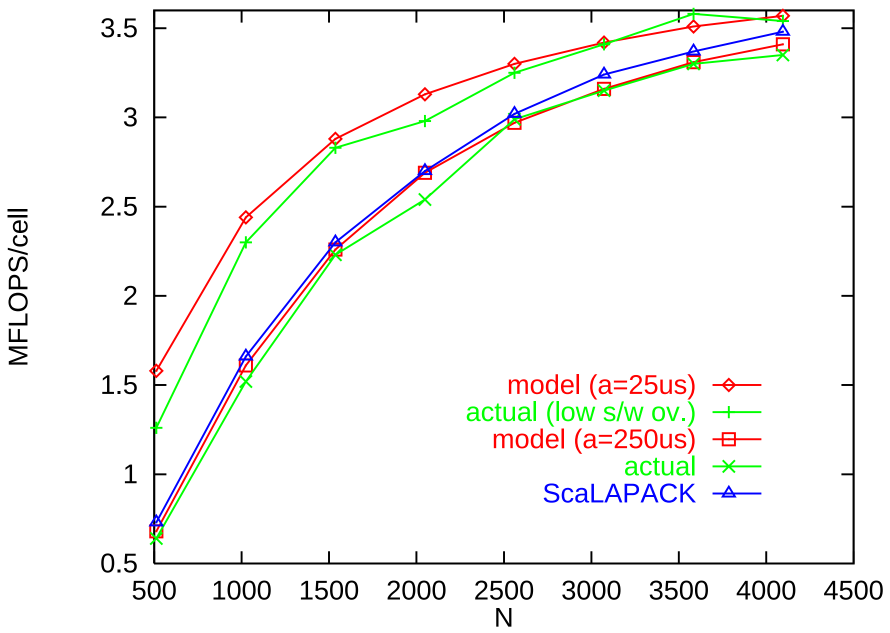
4.2 Example: Blocked LU Factorization - Coding

- DLAPACK (ANU, 1994-): explore alternative approaches to ScaLAPACK
  - DBLAS, the || BLAS, hides distribution info. inside DistMat structures

```
void DDGETF2(int M, int N, DistMat A, int P[])
  int i; double a_max;
  for (i = 0; i < min(M, N); i++)
    int i' = i + 1;   P[i] = i + DDAMAX_1(M−i, σ(A, i, i), &a_max);        /* |L'_{P[i],i}| ≥ |L'_{i:M,i}| */
    DDSWAP_1(N, σ^T(A, i, 0), σ^T(A, P[i], 0));                           /* L'_{i,:} ↔ L'_{P[i],:} */
    DDSCAL_1(M−i', 1.0/a_max, σ(A, i', i));                               /* l_i ← l_i/A_{i,i} */
    DDGER_2(M−i', N−i', −1.0, σ(A, i', i), σ^T(A, i, i'), σ(A, i', i'));  /* L_i ← L_i − l_i u_i */

void DDGETRF(int M, int N, DistMat A, int P[])
  int j = 0;
  while (j < min(M, N))
    int i,   ω = f_ω(M, j, A),   j_ω = j + ω;
    DDGETF2(M−j, ω, σ(A, j, j), &P[j]);
    for (i = j; i < j_ω; i++)
      P[i] += j;   DDSWAP_1(j, σ^T(A, i, 0), σ^T(A, P[i], 0));            /* A_{i,:} ↔ A_{P[i],:} */
      DDSWAP_1(N−j_ω, σ^T(A, i, j_ω), σ^T(A, P[i], j_ω));                 /* (outside L') */
    DDTRSM_3("Lower", "Unit", ω, N−j_ω, 1.0, σ(A, j, j), σ(A, j, j_ω));   /* U' ← T^{-1} U' */
    DDGEMM_3(M−j_ω, N−j_ω, ω, −1.0, σ(A, j_ω, j), σ(A, j, j_ω), 1.0, σ(A, j_ω, j_ω));  /* A' ← A' − L'U' */
    j += ω;
```

4.3 Example: Blocked LU Factorization - Performance



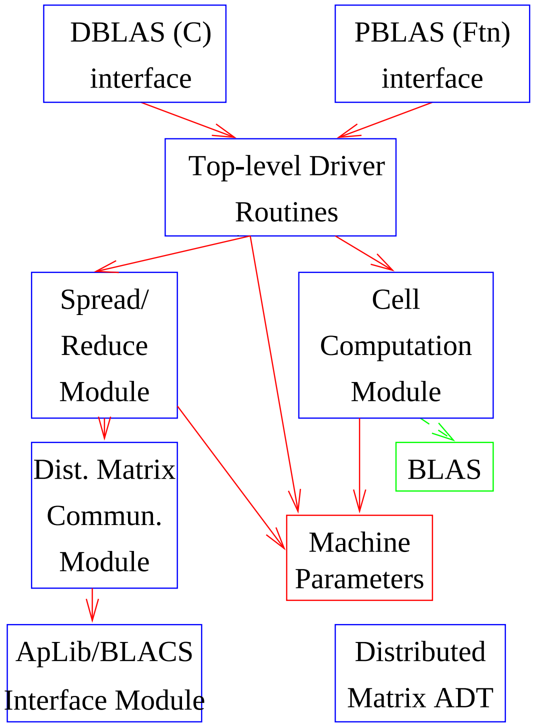
on an 8×8 AP1000 with $\omega = r = s = 16$

the Fujitsu AP1000:

- SPARC 1-based, 5.5 MFLOPS peak
  - 128 KB cache
- physical torus network, wormhole routing:
  - $\surd$ fast row/column broadcast
- $\alpha = 25\mu s, \beta = 1.2\mu s$ and $\gamma_3 = .25\mu s$

- indicates s/w overheads of at least a factor of 10 than that due to α!
  - $5N + N$ (Level 1+2) DBLAS calls, $10N$ sub-matrix $(\sigma(A, i, j))$ opn's,
  - $\geq 7N$ implied global-to-local index conversions $(g(i, D_0))$

5 Overview of two Parallel LAPACK Implementations



DLAPACK:

- 'deep' software architecture, generalized procedures; portable interfaces
- powerful procedural abstraction: only 2 generic communication operations
- powerful data abstraction: the `DistMat` ADT
  $A = (\texttt{bool}\ t, (a_0, I_{0..1}), D_{0..1}),\ D_0 = (r_0, r, p_0, P)$
- $\Rightarrow$ potential worse-case for software overheads!
  - inside DBLAS calls and $\sigma(A, i, j)$ operations

ScaLAPACK similar in its architecture, except:

- (UTK PBLAS) less abstractions used, shallower architecture
- $A = (a_0, i_A, j_A, \texttt{DescA})$ for (i, j)th sub-matrix: $(i_A, j_A) \leftarrow (i_A + i, j_A + j)$
  - $\Rightarrow$ software overheads almost entirely within PBLAS calls

6 Techniques for Reducing Software Overheads

- overall strategy: optimize the existing procedures and data structures
  - 2 categories: general and specific:

    ie. applicable *only* to most 'important' classes of computations
- passing (`DistMat` etc) structures by address:
  - pseudo-functional interface with fast stack allocation; 'packed' structures
  - required extensive code changes; overall reduction: $\approx 0.5\alpha$ per DBLAS call
- optimizing sub-matrix indexing and other arithmetic:
  - formed the bulk of the software overheads
  - integer arithmetic surprisingly expensive on modern RISC processors:

	clk (ns)	*	/, %	f.p. *+
SPARC 1:	40	20–60	50–90	8
SPARC 10:	20	5	22	2
UltraSparc:	6	5–9	45	1

- $\sigma(A,i,j)$ involves computing $g(i,D_0)$ $g(j,D_1)$ & new r_0', p_0', s_0', q_0':
  - express $g(i,(r_0,r,p_0,P))$ using 1 '*' and 2 'combined /%'s;
    - r_0', p_0' computed for free!; reduced cost of $\sigma(A,i,j)$ from 1.6α to 1.0α
  - if P,r powers of 2, store $\lg_2 P$, $\lg_2 r$ inside D_0;
    - using bitwise arithmetic reduced cost to 0.4α
  - exploiting the principle of locality of $g(i,D_0)$ for $i=0,1,2,\ldots$:
    - eg. locate diagonal of Δ matrices using inductive form of $j_i = g(g^{-1}(i,D_0)+1,D_1)$ or $a_i = \sigma(A,i,i)$ reduced cost to $\approx 0.2\alpha$
- PBLAS interface-specific optimizations:
  - requires extensive error checking & conversion to `DistMat` structures
  - caching intra- & inter- call conversions also can exploit the principle of locality
  - reduced overheads by a factor of 4 (in `PDGETRF()`)
- Level 1 and 2-specific optimizations:
  - adequate to optimize the internal DBLAS routines for the vector case:
    - ensure fast $g(1,D)$, conditional evaluations, (especially) buffer allocation
  - reduced latency of spread from 4α to 1.6α, of '`dgemm()`' from 1.6α to 0.5α

7 Conclusions

- software overheads were substantially reduced in the DLAPACK library
  - a potential worst-case for software overheads
  - required the use of specific optimizations
  - even so, still non-negligible eg. the overhead of a DDGER() call is now 2.5α
  - techniques used are highly applicable to ScaLAPACK and PLAPACK
- software overheads (in parallel linear algebra libraries):
  - can be exposed by detailed and well-calibrated performance models
  - are at least as important an issue as communication startup overheads
    - the latter can often be reduced by similar techniques
  - will be of increasing importance:
    - as libraries become more ambitious and complex
    - in functional or object-oriented versions of such libraries
    - for sparse and banded matrices
    - with newer RISC processor technology