Software Overhead and Blocking Issues in Parallel BLAS

Peter Strazdins

Australian National University

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

March 18, 1998

1 Introduction

- comparison of the performance of the following implementations of the Parallel BLAS (PBLAS):
  - UTK PBLAS 1.2 (May 1996)
  - ANU PBLAS (May 1997)
    - a 'front-end' to the DBLAS
    - URL: `http://cs.anu.edu.au/people/Peter.Strazdins/projects/DBLAS/index.html`
  - UTK PBLAS 2.0 Alpha (Nov 1997)
  - UTK PBLAS 2.0+ Alpha [1] (Feb 1998)
- using LU decomposition with partial pivoting and LLT decomposition:
  - to illustrate the issues of algorithmic blocking
    - the panel width $\omega >$ the block-cyclic storage block size r

    and software overheads

  on the Fujitsu AP1000 and AP+ parallel computers

[1] A pre-release version, with some performance enhancements. Results will only be given here for tests yielding a significant improvement.

2 The Fujitsu AP1000

- a SPARC 1-based multiprocessor URL: `http://cap.anu.edu.au/`
  - theoretical peak speed of 5.5 MFLOPS / cell
- cells are connected in a torus network using wormhole routing
  - row/column broadcasts have the cost of a normal message, ie $\kappa_{bc} = 1$
- cells have a 128 KB direct-mapped copy-back cache (shared instruction & data)
  - $\omega \approx 64$ optimal for cell computation
  - can minimize the effect of the placement of code & data in memory (for small computations) by:
    - 'packing' the active code used in a timing into a contiguous segment
    - varying the displacement between the heap and code segments
- the AP1000 is benchmarked to yield $\alpha = 25\mu s$, $\beta = 1.2\mu s$, $\gamma_3(16) = .25\mu s$ and $\gamma_3(64) = .20\mu s$
- low software overhead AP BLAS & BLACS have been implemented

3 The PDGER() Benchmark

- PDGER() is the most critical PBLAS routine in LU wrt. software overheads
- when used to perform $C = C - AB$ on a 4×4 AP, A, B, C are also 4×4, & $r = 1$:

routine used	uses BLAS &BLACS	s/w overhead / call ($\times\alpha = 25\mu s$)	active code size
ANU PDGER()	no	3.2 (1.2)	90 KB
ANU PDGER()	yes	4.1 (1.2)	112 KB
UTK 1.2 PDGER()	yes	12.9 (6.3)	76 KB
UTK 2.0 PDGER()	yes	11.0 (4.6)	176 KB

  - the matrix descriptor checking & indexing component is in ()'s:
    - for UTK 2.0, $\approx$ half of this is in error checking & half is in Minfog2l()
- improvement in UTK 2.0 over 1.2 largely due to more efficient indexing arithmetic
- the cost of portability (BLAS (mainly) & BLACS interfaces) is 0.9α
- the use of descriptor caching & other techniques $\Rightarrow$ ANU PDGER() overheads are smaller by a factor of 3.4

3.1 The `PDGER()` Benchmark on the AP+

- the Fujitsu AP+:
  - similar to the AP1000, except SuperSparc-based cells, with 16 KB 4-way (20 KB 5-way) set associative write-through data (instrn.) caches
    - $\omega \approx 32$ optimal for cell computation, max 50 MFLOPS / cell
    - no level-2 cache $\Rightarrow$ higher (esp. instrn.) cache miss effects & higher γ_2/γ_3
  - has $\alpha = 12\mu s$, $\beta = 0.64\mu s$, $\gamma_3(16) = .035\mu s$ & $\gamma_3(32) = .030\mu s$
- the `PDGER()` benchmark (on a 2×2 AP+):

routine used	uses BLAS & BLACS	s/w overhead / call ($\times\alpha = 12\mu s$)
ANU `PDGER()`	no	3.9 (1.1)
ANU `PDGER()`	yes	5.5 (1.1)
UTK 1.2 `PDGER()`	yes	12.1 (3.5)
UTK 2.0 `PDGER()`	yes	12.0 (2.3)

  - role of indexing arithmetic is less, but o.w. similar results to the AP1000

4 The LU Benchmark

- a LU decomposition of an $N \times N$ matrix using modified ScaLAPACK 1.2 codes:
  - `PDGETRF()` and `PDGETF2()` were extended for algorithmic blocking
  - URL: `http://cs.anu.edu.au/people/Peter.Strazdins/projects/ExtendScaLAPACK/index.html`
- for '`Al`', $\omega = 64$ and $r = 1$, for '`St`', $\omega = r = 16$;
  '`P`' ('`N`') signifies a portable (non-portable) interface
- speed in MFLOPS/cell on an 8×8 AP1000:

N	512	1024	1536	2048	2560	(active code size)
`ANU-Al-N`	1.66	2.87	3.53	3.87	4.09	(128 KB)
`ANU-St-N`	1.22	2.20	2.75	3.08	3.31	
`ANU-Al-P`	1.51	2.74	3.43	3.80	4.05	(144 KB)
`ANU-St-P`	1.14	2.09	2.68	3.03	3.29	
`UTK1.2-St-P`	0.83	1.80	2.43	2.84	3.11	(128 KB*)
`UTK2.0-Al-P`	0.66	1.33	1.84	2.23	2.55	(250 KB*)
`UTK2.0-St-P`	0.89	1.87	2.51	2.89	3.17	
`UTK2.0+-Al-P`	0.94	2.10	2.88	3.37	3.70	(240 KB*)

5 The LLT Benchmark

- a lower triangular LLT decomposition benchmark is similar to LU except:
  - alg. blocking required a `PDSYR()`-based version of `PDPOTF2()` (denoted by `*-pdsyr`)
  - this increased software overheads & communication startups more than for LU
    - $\Rightarrow$ a smaller improvement for low N is expected
  - however an $O(\frac{N^2 r}{P})$ load imbalance due to tri. matrices favors low r
- speed in MFLOPS/cell on an 8×8 AP1000:

N	512	1024	1536	2048	2560	3072	(active code size)
ANU-Al-P-pdsyr	1.23	2.39	3.05	3.48	3.79	3.96	(168 KB)
ANU-St-P-pdsyr	1.00	1.93	2.49	2.83	3.06	3.23	
ANU-St-P	1.05	2.00	2.55	2.87	3.10	3.26	
UTK1.2-St-P-pdsyr	0.57	1.35	1.96	2.38	2.68	2.92	(120 KB)
UTK1.2-St-P	0.83	1.66	2.22	2.57	2.83	3.03	
UTK2.0-Al-P-pdsyr	0.51	1.19	1.71	2.12	2.42	2.63	(240 KB)
UTK2.0-St-P-pdsyr	0.69	1.55	2.18	2.58	2.87	3.09	
UTK2.0-St-P	0.97	1.90	2.44	2.79	3.03	3.22	

6 LU and LLT Benchmarks on the AP+

- here, $\omega = 32$ was used for 'Al'; $r = 16$ was generally optimal for 'St'
- speed in MFLOPS/cell on an 8×8 AP+:

	N	512	1024	1536	2048	2560	3072	3584
LU	ANU-Al-P	3.5	8.6	12.8	15.8	18.1	19.9	21.3
	ANU-St-P	2.9	6.8	10.0	12.1	14.1	15.6	16.8
	UTK1.2-St-P	2.3	6.0	9.2	11.5	13.6	15.2	16.4
	UTK2.0-St-P	2.3	6.0	9.2	11.6	13.8	15.4	16.7
	UTK2.0+-Al-P	2.4	6.4	10.1	13.1	15.3	17.0	18.5
LLT	ANU-Al-P-pdsyr	2.7	7.1	11.1	14.4	16.9	18.9	20.5
	ANU-St-P-pdsyr	2.8	7.0	10.5	13.2	15.4	17.1	18.4
	ANU-St-P	4.2	8.7	12.2	14.6	16.6	18.1	19.2
	UTK1.2-St-P-pdsyr	1.5	4.3	7.1	9.3	11.2	12.8	14.0
	UTK1.2-St-P	3.1	6.7	9.6	11.5	13.1	14.4	15.4
	UTK2.0-St-P-pdsyr	1.9	5.3	8.5	10.5	12.3	13.7	14.8
	UTK2.0-St-P	3.5	7.3	10.0	11.9	13.4	14.6	15.6

7 Discussion

1. algorithmic blocking had the most impact ($\approx$ the same for LU & LLT, for AP1000 & AP+):
   - despite its extra communication overheads:
     - eg. $3N$ extra startups, and $2\times$ the volume for the L & U panels in LU
     - however, for LLT, software & startup overheads increase from $O(N/r)$ to $O(N)$, degrading small N performance
   - this improvement is sustained for larger N (eg. for LU on the AP+, up to $N = 8192$)
2. the software overhead from the BLAS & BLACS was fairly small ($\approx 10\%$ for $N = 512$):
   (a) the BLAS must perform $y \leftarrow \alpha x$ in 2 steps ($\alpha \neq 1$)
   (b) the BLAS does not provide a low-latency but limited version of `dgemm()`
   (c) the AP BLAS adds an extra level of procedure calls
   (d) the BLACS requires the index to be broadcast separately in `PDAMAX()`
3. (other) software overheads (cf. `*-St-P`) had a significant impact:
   - UTK 2.0 shows an improvement over UTK 1.2

- ANU PBLAS has the least, $\Rightarrow$ a 25% improvement for $N = 512$ for LU
- relative software overheads of `PDSYR()` are similar to that of `PDGER()`
- for LLT, overheads in `PBLAS()`-based level-2 codes degraded performance:
  - further magnified by extra instrn. cache misses on the AP+

5. the overall differences between the portable '1997' LU tests `ANU-A1-P` & `UTK*-St-P` are, on the AP1000 (AP+):
   - factors of 1.70 (1.52) for $N = 512$ and of 1.25 (1.31) for $N = 2560$
   - this difference is $\approx$ halved for $N = 2560$ by the recent `UTK2.0+ PBLAS`
6. that between the LLT tests `ANU-*-P*` & `UTK*-St-P` are smaller for low N:
   - on the AP1000: ≈ 1.25 across $512 \leq N \leq 3072$
   - on the AP+: 20% at $N = 512$ increasing to 30% for $N = 3584$
7. future work:
   - expect similar improvements for QR;
     also for UltraSparc-based AP3000 (at least for moderate–large N)
   - would like to extend work to the T3E, SP-2, Berkeley NOW & Paragon!