Recent ATLAS development
Minimizing Startup Costs for Performance
Critical Threading

R. Clint Whaley
whaley@cs.utsa.edu
www.cs.utsa.edu/~whaley

Anthony M. Castaldo
castaldo@cs.utsa.edu

University of Texas at San Antonio
Department of Computer Science
Recent ATLAS trends

In ATLAS 3.9.12

1. Help others using ATLAS tuning framework:
   - `atlas_[mv,r1,mm]parse.h`: accesses standard I/O files
   - `atlas_[mv,r1,mm]testtime.h`: tester/timer calls for search

2. Tune mem-dominated kernels for in-L1, in-L2, out-of-cache (GER)

3. New threading mechanism (this talk) → Tony Castaldo (PhD)
   - Can install ATLAS to use pthreads, Winthreads, or OpenMP (Ryan Moon, BS)

Coming soon (i mean soon)

1. GEMM generator using gcc SSE intrinsics → Chad Zalkin (MS)
2. Improved QR/QL/LQ/RQ → Siju Samuel (MS)
3. GER/GEMV generators (me, maybe not so soon)
Reducing Thread Overhead
Minimizing Startup Costs for Performance Critical Threading

Extract of paper presented at IPDPS09


Motivation

We found that rank-60/120 update was getting less than 4-fold speedup on 8 processor machine!

- Suspects: thread migration, bus traffic, startup costs (don’t be absurd)
  - I told student problem obviously thread migration, go use affinity
    - Made almost no difference
  - Culprit was startup costs
Overhead rises with $N$ rather than $p$!

**LJ, LJA Overhead**
Many times early threads are started on same core as master
Each thread creation is system call which causes context switch
Computational kernels hog up entire context switch
  → startup costs rise with problem size rather than $p$!
⇒ Solution: use affinity to start threads on master’s core last
  • Too intrusive to tie user process to core
  • Easiest fix is to spawn a thread-master to core 0, immediately join
    → $(1 + \log(p))$ or $(1 + p)$ steps
PMO after applying master-last LJ, LJA, LJAML, PWAML on 2.5Ghz Core2 Xeon

LJ, LJA vs. LJAML, PWAML

Problem Size

PMO (uSec)

Figure 1 (b) Master-last thread assignment

C2

LJ
LJA
LJAML
PWAML
Roughly 60- (250-) fold speedup for large problems:
LAPACK speedups
Master-last makes huge difference for higher algorithms

Performance Improvement from new threading implementation on 8-processor Core2

- LLt
- LU
- QR
LU Speedup on 4-core Windows
Master-last makes huge difference for higher algorithms

LU: Windows threads (wt aff) / pthreads (w/o aff)
Mixing threaded libraries dangerous

gcc’s OpenMP can’t be safely mixed with anything

Mixing thread paradigms kills performance

- LJ/LJ (no affect)
- LJ/PW: both libs hugely slower
- LJA/PW: PW gets slower (assume cond var)
- PWA/PWA: should be OK if both use cond vars

If all libs don’t use affinity and cond vars/LJ they strongly interfere:

- ATLAS presently uses LJAML → Will have PWAML option in future
- Most libs use PW, some w/o cond var (eg., GotoBLAS)
- gcc’s OpenMP uses PW w/o affinity, so even LJ messes it up!

⇒ Libs should use LJ or PW wt cond vars and affinity