Modeling and Tuning Parallel Performance in Dense Linear Algebra

Initial Experiences with the Tile QR Factorization on a Multi-Core System

CScADS Workshop on Automatic Tuning for Petascale Systems
Snowbird, Utah, 7/9/2008
presented by Jakub Kurzak
University of Tennessee
Proposition:

Today's autotuning work does not address the challenges of petascale (in the context of dense linear algebra).

What challenges need to be addressed?

Deemphasize weak scaling (isoscaling)
Emphasize strong scaling
Tuning Tile QR Factorization

Topics:
- Challenging existing performance models
- The tile QR algorithm
  - Tile algorithm versus block algorithm
- Modeling the main kernel
  - Tuning performance of the _SSRFB kernel
- Modeling the tile QR factorization
  - Tuning parallel granularity of the tile QR factorization
ScaLAPACK Performance Model

\[ T(N, P) = \frac{C_f N^3}{P} t_f + \frac{C_v N^2}{\sqrt{P}} t_v + \frac{C_m N}{NB} t_m \]

**Assumptions:**
- Flop/s rate is constant
- Load is balanced
Challenging ScaLAPACK Model

**Hypothesis:**

- Flop/s rate is not constant
- Load imbalance matters

- Coarse granularity gives high Flop/s rate per core
- Fine granularity produces high level of parallelism (good load balance)

- Use granularity to trade one for the other
Block QR Factorization (LAPACK)

- Panel factorization

- Trailing submatrix update
Tile QR Factorization (PLASMA)

- Panel factorization
- Trailing Submatrix Update

\[ \text{BB} = \frac{N}{NB} \]
In general

- big number of tasks
- of almost identical size
Performance Critical Kernel (_SSRFB)

- Matrix-Multiply-like kernel

- Level 2 BLAS
- Memory-bound
- No extra FLOPs

- Level 3 BLAS
- Compute-bound
- 25% extra FLOPs
Let's not measure noise

Rely on:

- Block Data Layout
- Warm caches / TLBs
- Huge TLB pages
- Best performance over multiple runs
  - (although small variation due to the bullets above)
- Static (pipelined) schedule
  - For this (initial) experiment only
  - Dynamic scheduling to be used ever after
Warm Cache / Cold Cache

![Graph showing performance metrics over time for warm and cold cache. The graph includes three lines representing different data sets, each with a unique color. The x-axis represents time in increments of 50 units, starting from 0 up to 250. The y-axis represents performance metrics ranging from 0 to 100. The lines show trends and comparisons between warm and cold cache performances.]
DSSRFB Kernel Performance

IB – constant
NB – the larger, the better
DSSRFB Kernel Performance

Intel Xeon 2.4 GHz -- Single Core

- NB – constant
- IB – ???

Modeling & Tuning
DSSRFB Kernel Performance

\[
\frac{(NB^3)}{(NB^3) + (IB \times NB^2) + \left(\frac{NB^3}{IB}\right) + (NB^2)}
\]

Parameters: a, b, c, d
x = NB
y = IB

Basic FLOPs \quad Extra FLOPs \quad Basic mem \quad Extra mem
a = 9.59
b = 0.24
c = 3.77
d = 21.40

peek = 9.6 [Gflop/s]

error = 0.18 %
(of peak estimate)
DSSRFB Performance Model

- < 1% error – 70% samples
- < 2% error – 90% samples
Tile QR Parallel Performance

Tile QR -- Intel Xeon 2.4 GHz
one socket quad-core (4 cores)

Gflop/s

Matrix Size

IB:
60
60
40
20

NB:
240
180
120
60
Tile QR Parallel Performance

Tile QR -- Intel Xeon 2.4 GHz
dual-socket quad-core (8 cores)

IB:
60
60
40
20

NB:
240
180
120
60
Tile QR Parallel Performance

Tile QR -- Intel Xeon 2.4 GHz
quad-socket quad-core (16 cores)

Gflop/s vs. Matrix Size

- NB: 240, 180, 120, 60
- IB: 60, 60, 40, 20
Tile QR Parallel Performance

Tile QR -- Intel Xeon 2.4 GHz
quad-socket quad-core (16 cores)

Gflop/s vs Matrix Size

IB:
- 60
- 60
- 40
- 20

NB:
- 240
- 180
- 120
- 60
Tile QR Parallel Performance

- No “one size fits all”
- “Blind” choice can give catastrophic effects
For smaller problems finer granularity allows to exploit higher degree of parallelism at the price of small per-core performance drop.
Tile QR Performance Tuning

\[
NB = f \text{ (problem_size, #cores)}
\]

\[
IB = f \text{ (NB)}
\]
Modeling Load Imbalance

\[ P \times BB \times NB^3 = \frac{P \times N}{NB} \times NB^3 = P \times N \times NB^2 \]

\[ P^2 \times NB^3 \]
Tile QR Parallel Performance Model

\[
\frac{(NB^3)}{(BB^3 NB^2) + (P \times BB^2 NB^2) + (P \times N \times NB^2)}
\]

**No data affinity:**
- Each tile operation causes a P2P communication
- Each panel operation causes a broadcast to P destinations

\[
P \times \text{dssrfb}(NB, IB) \quad \frac{1 + \frac{a}{x} + \frac{b \times P}{x}}{1 + \frac{b \times P}{y} + c \times P \times \left(\frac{x}{y}\right)^{2.5}}
\]

Parameters: a, b, c, d
x = NB
y = N
Block QR Factorization

4 cores
Block QR Factorization

16 cores
Block QR Factorization

Parallel Tile QR Model Accuracy

Distribution of Relative Error

Fraction of Samples [%]

Error Range

- < 5%
- < 10%
- < 15%
- < 20%
- < 25%
- < 30%
- > 30%
Conclusions

- Performance of new, tile, algorithms is easier to model
  - Many adverse effects minimized (e.g. cache misses)

- Performance of the parallel algorithm can be derived from the performance of the most critical kernel

- Strong scaling matters
  - Load imbalance needs to be taken into account

- Sequential performance can be traded for parallelism
void spe_tile_sssrfb(float *V2, float *T, float *C1, float *C2) {
  int i, j, m, n, k;
  float W[4*64];

  for (j = 0; j < 64; j+=4) {
    for (i = 0; i < 4*64; i++)
      W[i] = C1[j*64+i];

    for (m = 0; m < 4; m++)
      for (n = 0; n < 64; n++)
        for (k = 0; k < 64; k++)
          W[m*64+n] += (V2[j+k*64+m] * C2[k*64+n]);

    for (m = 3; m >= 0; m--)
      for (n = 0; n < 64; n++)
        for (k = 0; k <= m; k++)
          temp += T[j*64+j + k*64+m] * W[k*64+n];

    W[m*64+n] = temp;

  }

  for (m = 0; m < 64; m++)
    for (n = 0; n < 64; n++)
      C2[m*64+n] -= (V2[j+m*64] * W[m*64+n]);

  for (m = 0; m < 4; m++)
    for (n = 0; n < 64; n++)
      C1[m*64+j*64+n] -= W[m*64+n];
}

◆ 1,600 LOC in C
◆ 2,200 LOC in ASM
◆ 1 – 3 person / months

Rethink SIMD'zation:
◆ Why not use supercomputer power?
  (I really don't mind compiling it for a week on a rack of Blue Gene)