CASK, Cray Adaptive Sparse Kernels

Keita Teranishi
Cray, Inc.

CScADS 2008, Summer Workshop 2008
Questions we are going to answer

- What architecture/platform should we target?
  - Cray’s systems!

- Will self-tuned libraries always outperform compiler-generated code?
  - Yes, because we search the best combination of compiler’s tuning flags and auto-generated code.

- Will simple performance models obviate the need for empirical search?
  - Simple and good performance models would reduce the search space a lot (since we do just for Cray).
Cray’s role in libraries (in the past)

- Early Cray machines were important in the evolution of Math Software – e.g. BLAS 3

- Extensive tuning of BLAS and FFT in assembly language
  - Tuned for Cray’s hardware (usually vector)

- Provide custom sparse solvers and special data structures

- Things were simple then
  - Same or similar ISA
  - Hardware complexity was at register level
  - Single processing cores and bandwidth galore
  - API’s were sufficient for good performance
  - Custom solvers were attractive

Nothing was adaptive because it was not necessary
Things change...

3 main things have changed that motivate auto-tuning
Reason why we do auto-tuning

1. Hardware

2. Evolution of Linear System Solvers

3. Application Dependent Performance
#1 Hardware

- Now Cray sell mostly x86 based supercomputers
  - AMD Opteron currently
  - Intel Xeon in near future

- No longer control the ISA – it keeps changing on us!
  - Single core (AMD64)
  - Dual core (faster memory, includes SSE3 instructions)
  - Quad-core (new memory hierarchy + new instructions)
  - Very short development cycle (compared to old vector supercomputers)
  - Cray sells machines for all 3 concurrently

- Now have Multi-core chips, existing methods and API’s may not allow us to overcome the memory bandwidth wall.
#2 Evolution of Linear System Solvers

- ¼ PetaFlop and 31,000 cores motivates a deep emphasis on sparse iterative solvers
  - Even more cores in our future machines!

- Maturity of numerical methods and software infrastructure
  - Iterative methods and preconditioning
    - Templates, SciDAC TOPs Projects, etc.
  - De facto standard software packages
    - PETSc, Trilinos, hypre

- Cray supports these software packages and tune them for our systems
  - Like how we support dense linear algebra packages
  - No change in API
  - No change in functionality
#3 Application Dependent Performance

- SpMV performance depends wildly on sparsity pattern / character

- Density (number of non-zeroes per row) and sparsity pattern govern how well the base CSR kernel performs
  - Unlike dense linear algebra

- General purpose tuned codes cannot be written for these types of problem
Generic CSR SpMV code (the Bedrock)

```c
    do q = 1, n_rhs
        next_row_begin = row_start (1)
        do i = 1, n_rows
            row_begin = next_row_begin
            next_row_begin = row_start (i +1)
            ip = 0.0
            do k = row_begin, next_row_begin - 1
                ip = ip + values (k) * x (col_index (k), q)
            end do
            y (i, q) = ip
        end do
    end do
```

Irregular memory access on values
Irregular access on col_index
Very little re-use of col_index and values
Code is memory bandwidth bound

Vendors have not been ‘doing their part’ here.
Performance of 2 tuned SpMV kernels relative to BASE case

Opt 1: prefetch 4 cachelines, 4 iterations ahead of time, no unrolling

Opt 2: K loop unrolled by 2, no prefetching

Matrix Dimension

Relative Performance

opt1 - low density
opt1 - high density
opt2 - low density
opt2 - high density

July 08 Copyright 2008, Cray Inc. Slide 10
Can’t just throw a great optimization at SpMV

Optimization: Unroll k loop by 2, prefetch 2 cachelines, 4 iterations ahead of time

% Improvement over base case

Matrix dimension

July 08
Copyright 2008, Cray Inc.
Same problem with aggressive tuning by compiler (PGI using -fast –fastsse)

Relative Performance against Base Case (-O3)

Matrix ID#

Relative Performance

Base -fast -fastsse

July 08

Copyright 2008, Cray Inc.
SpMV summary

- Linear solve results in many serial SpMV operations
- These are the most expensive part of a linear solve
- Code is memory bandwidth bound
- Performance of SpMV changes with respect to matrix characteristics
- Applicability of an optimization applies only to a certain problem or set of problems

There is no “General Purpose” SpMV Code
Cray Adaptive Sparse Kernels

- Large-scale application
  - Highly portable
  - User controlled

- PETSc / Trilinos
  - Highly portable
  - User controlled

- CASK
  - XT specific / tuned
  - User not aware
Consider this reasonable optimization space

- **Unroll k loop from 1, up to 8**
  - 8 variants → 8 implementations

- **Unroll q loop, from 1 to 4**
  - 4 variants → 32 implementations

- **Prefetch 1 to 5 cache lines before loop**
  - 5 variants → 160 implementations

- **Prefetch before loop 1 to 10 rows in advance**
  - 10 variants → 1600 implementations

- **Prefetch 1 cacheline with inner loop, 1 to 30 iterations ahead of time**
  - 30 variants → 48000 implementations

= work for 250 summer internships
CASK design / philosophy

- Using limited analysis and without the involvement of the user
  
1. Analyze matrix at minimal cost
2. Categorize matrix against internal classes
3. Based on offline experience, find best CASK code for particular matrix class
4. Previously assign “best” compiler flags to CASK code
5. Assign best CASK kernel and perform Ax
Auto-tuning Framework (Continued)

- Implemented with Ruby and XML
  - Ruby orchestrates the tuning process
  - XML defines the target range of tuning parameters and test sparse matrix

- Code generator

- Tester program contains a random matrix generator
  - Takes 10+ parameters
  - Can generate blocked and banded sparse matrices
  - Also imports HB format sparse matrix files

- Performance analysis tool integrated with CrayPat
  - Allows off-line tuning as well as establishing a scheme for on-line tuning with a negligible runtime overhead.
CASK and PETSc, single core
CASK and PETSc – quad core

Matrix ID#

Single core

Parallel on 4 cores

bone010

Idoor

audikw_1

thermal2

sme3Dc

pre2

Copyright 2008, Cray Inc.

July 08
FBSR and CSR Comparison

Performance of CASK CSR and FBSR Kernels
N= 40,000, BW=6000, BS=4, QC (4 core)
Auto-tuning Experience

- Good performance model is required to enable good auto-tuning
  - We incrementally refined our performance model!
  - Often the auto-tuning results pointed out what we were missing.

- Good matrix generator is required
  - Even for random matrices, it is possible to reproduce real-application-like matrices in terms of performance behavior
  - Care must be taken when using matrices in the public repositories (NIST, U of Florida)
Auto-tuning Experience (Continued)

- Ruby and XML provides a great flexibility of tool development
  - Easy to add a new tuning feature
  - XML is used to control the tuning, parameter search space, and log-keeping, etc.

- The size of the search space for parameters seems to be big at the beginning, but it can be reduced.
  - Supervised search using our knowledge in compiler flags and sparse matrix computation
  - Every auto-tuning results tell us what to trim
  - How to automate this trimming process?

- Works as a good regression test tool!
Further Auto-tuning Work in LibSci

- CASK 1.0 released in August 2008
  - Runs with PETSc
  - CSR and FBSR SpMV
  - Version 2.0 will support Trilinos, VBR and Transpose SpMV, Triangular Solution and more.

- CRAFFT Version 1.0 will be released in July 2008
  - Very simple API
  - FFTW, ACML support
  - Less runtime tuning overhead

- Working with Spiral team for CRAFFT Version 2

- Parallel FFTs

- Auto-tuning in Dense Linear Algebra? (GEMM, eigensolvers, ScaLAPACK)
Special Thanks

- CASK Team
  - Adrian Tate
  - John Lewis