Cray Math Software: current and future developments

Adrian Tate
Tech Lead of Math Software
Cray Inc.
Contents

- Roadmap
- XT4 Opteron libraries
- ‘Baker’ Libraries
- ‘Cascade’ Libraries
- Future emphasis
Contributors

- John Lewis
- Jonathan Bentz
- John Levesque
- Other Libraries people
  - Chao Yang
  - Keita Teranishi
  - Neal Gaarder
The Cray Roadmap

Phase I: Rainier Program
Multiple Processor Types with Integrated Infrastructure and User Environment

Phase II: Cascade Program
Adaptive Hybrid System

Phase 0: Individually Architected Machines
Unique Products Serving Individual Market Needs
Cray XT4 Supercomputer
Purpose-Built for MPP Applications

- Next-generation Massively Parallel Processing (MPP) supercomputer from Cray
  - Follow-on to Cray XT3 & Cray XD1 systems
  - Based on industry-leading AMD Opteron processors
  - Maintains strong system balance:
    - 2X injection bandwidth with SeaStar2
    - 2X memory bandwidth with DDR2
    - Dual-core today, quad-core in 4Q07
  - Support for Linux compute nodes (2H07)
  - Support for FPGA nodes (1H08)
- Results in application performance and highly reliable operation at massive scale
- Introduces the Cray XT infrastructure

“Scalable Computing At Work”
Product evolution with demonstrated support for applications requiring hundreds or thousands of processors working simultaneously on the same problem
XT4 BLAS & LAPACK

- Cray is currently in the process of migrating LA products
  - From ACML to libGoto + Cray LibSci

- Eventual package will be a piecemeal collection of best routines and hand tuned cases for certain problem sizes

- Will explore ATLAS to help fill in the gaps
XT4 libraries - GotoBLAS vs ACML BLAS

% improvement Goto vs ACML

matrix

May 16, 2007
Copyright 2007 – Cray Inc.
XT4 libs – Sparse Iterative Solvers

- With peta-scaling machines, we see an increased need for highly tuned sparse iterative solvers
- Cray will not develop an iterative solver package for scalar systems
  - Leverage PETSc and Trilinos
  - Tune for Cray processor / interconnect
- Add Cray custom value in 3 areas
  - Cray Sparse BLAS
  - Parallel performance of solvers
  - Custom preconditioners
Iterative solvers, serial/parallel breakdown

- For large systems, sparse kernels are the key
- For small problems, need to redesign solver in a way that hides more latency
  - Easy to do on ‘Baker’ system, hard to do on XT systems
Cray sparse BLAS overview

- CSR matrix vector product
  - Generic
    - Unrolled over rhs, columns
    - Prefetching of matrix values and column index
    - Various compilers / compiler switches
    - Support for 0 and 1 base indexing
    - Various orders of loops

- FBR and VBR implementations
- Jagged diagonals and Segmented Scan implementations
- Level-based solves
- Comprehensive test infrastructure

At least for immediate future, emphasis is on generic CSR
Generic CSR MV code optimization opportunities

**Unroll q loop**

\[
\text{do } q = 1, \ n_{\text{rhs}} \\
\text{next\_row\_begin} = \text{row\_start}(1) \\
\text{do } i = 1, \ n_{\text{rows}} \\
\text{row\_begin} = \text{next\_row\_begin} \\
\text{next\_row\_begin} = \text{row\_start}(i + 1) \\
\text{ip} = 0.0 \\
\text{do } k = \text{row\_begin}, \text{next\_row\_begin} - 1 \\
\text{ip} = \text{ip} + \text{values}(k) \ast x(\text{col\_index}(k), q) \\
\text{end do} \\
\text{y}(i, q) = \text{ip} \\
\text{end do} \\
\text{end do}
\]

**Prefetch directives**

(Prefetch X cachelines, Y iterations ahead)

**Unroll k loop**

**Interchange all loops**

choices of compilers zero / one indexing
Aggressive optimizations can be dangerous

 pref etch value against local rows in MV
 low density systems

 Number of local rows

 Prefetch value added (%)
Sparse BLAS software engineering challenge

- Experientially, cannot make strong predictions about applicability of specific optimizations
- Also cannot predict the relationship between optimizations
- Needed a harness to analyze performance of every implementation
- E.g. testing
  - unroll Q loop [1:10] times
  - unroll K loop [1:10] times
  - base [0, 1]

  -> 32,000 different sparse MV implementations

  256 summer internships each implementing 125 routines !

This is ignoring loop interchanges and compiler flags etc....
Cray Sparse BLAS generator

- Ruby code generator reads a sparse BLAS template file
  - Modified Fortran code
  - Directives for unrolling and expansion, prefetching etc.
  - Allows readable end product
- Ruby test harness generates wrappers for each routine
  - allows a set of compilers/compiler flags to be defined and tested against
- Parallel test environment allows testing of each implementation for a specific class of sparse matrices
- Ruby end product re-organizes data and imports into a ‘viewable’ format
## C versus Ruby code generation experiences

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>Ruby</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development time</td>
<td>~40 hours</td>
<td>~40 hours</td>
</tr>
<tr>
<td>Generality</td>
<td>Completely specific</td>
<td>General and extensible</td>
</tr>
<tr>
<td>Aesthetics (generator)</td>
<td>UGLY</td>
<td>Simple and attractive</td>
</tr>
<tr>
<td>Aesthetics (generated code)</td>
<td>UGLY</td>
<td>attractive</td>
</tr>
<tr>
<td>Performance (6000 routines)</td>
<td>~7 minutes</td>
<td>~45 minutes</td>
</tr>
</tbody>
</table>
Iterative solver tuning philosophy

- Whilst we cannot make predictions about individual kernel performance, we can use empirical data to make statements about best implementation for certain matrix classes.

- Cannot afford sparsity analysis stage
  - If data is not in block format, it is not treated as such.
  - Same if data is not explicitly represented at symmetric.

- We know, or can find out enough about a sparse matrix to categorize it
  - Local number of rows
  - Number of RHS
  - Density
  - block size if any
  - Symmetry
  - Standard deviation of density

- Appropriate tuning recipe can then be applied accordingly.
E.g. Most important case (1 RHS & unstructured) PETSc library, using BCGS solver and bjacobi preconditioner. Matrix is low density, order 2M.
Longer term strategy

- Like OSKI, we see that block structure can be and should be exploited
  - “A large majority of sparse solver users have block structure property in their problems, but very few of them take advantage of that structure in any explicit way.” Mike Heroux

- FBR/VBR kernel tuning is another great advantage of the code generator
  - Explicitly move users over where possible

- Where not possible (majority?) we may require integration of the distinct products
Future iterative solver support vision

- **Solves systems**: PETSc & Trilinos
- **Machine independent**: PETSc & Trilinos
- **Some machine affinity**: OSKI
- **Tuned for Cray machines**: OSKI

- **Compiles block Structure and best kernel match**: OSKI
- **Compiler sensitivities**: Directed optimizations
  - Highly tuned for FBR/VBR

May 16, 2007
Multiple core challenges

- All computer vendors are facing this question
  *How do we exploit parallelism within a socket*

- For supercomputing vendors the issue is somewhat different
  *How do we exploit additional parallelism within a socket*

- Mixed-mode parallel libraries are hot again
  On how many cores can we stretch this approach to fit?
E.g. Mixed mode ScaLAPACK

- Cray supports ScaLAPACK working in mixed mode
  - MPI across sockets (1 BLACS process per socket)
  - Threaded BLAS within sockets
  - Persistent requirement

- How many cores-per-socket can this model scale to?
  - Studied the degradation of local BLAS3 dimensions as a function of block size
  - How much parallelism can we exploit in the common local BLAS operations that are called via scalapack?

E.g. *
E.g. pdgetrf

NB=64, process=0, process grid=8x8

Matrix size vs. BLAS3 count for different matrix sizes and processes.
E.g. pdpotrf
E.g. Divide-and-Conquer

dsyevd: dgemm on proc0, blocksize=64, matrix=25k

Tridiagonal Reduction
Find eigenvalues
Eigenvectors

size

M
N
K

dgemm count

May 16, 2007
Multiple core support

- Our results show that the mixed model may be falling over as early as quad core
  - 8 cores as 2-way x 4 cores NUMA will certainly be a problem

- One MPI process per core model is not a viable option

- Can recursive algorithms do better?
  - Cray are looking at this

- Customers do not want to re-visit dense linear algebra
  - performance vs. inconvenience
Cray Iterative Refinement Toolkit

- Instigated by the work at UTK
- Exploits longer SSE vector lengths by performing factorizations in single precision and using iterative refinement (mixed precision)
- Includes serial and parallel (real and complex) versions of
  - LU, Cholesky, QR
- Includes advanced interface allowing
  - Minimization of forward error
  - Control over iterative refinement process
  - Advanced convergence scheme
IRT on XT4
(Condition vs. performance)

Measuring speed-up for various condition numbers, Matrix dimension = 3000, irt_lu_real_serial used

IRT works well
IRT may help
IRT will not help
‘Baker’ System (simplified)

High Bandwidth Gemini interconnect
Configurable

Baker Opteron Compute Nodes

Local memory

Globally addressable memory
Support for PGAS
Baker libraries

- Parallel optimization centered around reducing communication bottlenecks by exploiting Baker interconnect and memory system
  - Remote load and stores (assembly, CAF, UPC)
- E.g. ScaLAPACK on X1e
  - ScaLAPACK collectives replaced with lightweight CAF versions (specific)
- Advantage of 1-sided collectives
  - Better algorithms
  - Potential for latency hiding
  - Inline code
Library optimization through PGAS

- Support of PGAS languages is high on Cray’s agenda
  - Do not expect many application to be written entirely in PGAS

- Want to allow users to switch in and out of PGAS languages, and to allow their usage internally within libraries

- Normally, need symmetrically allocated memory to use PGAS

- Need a mechanism to allow remotely accessibility of PGAS objects without having to allocate from the symmetric heap
CAF/UPC as a bottleneck solution

- Support pointer addressing to allow use of these models without re-allocation from the symmetric heap
  - Use a Co-array of derived type, which has a one member – a pointer to a local array.

Parallel library optimization or benchmarking optimization using PGAS is entirely dependent on its implementation
Cascade libraries

- DARPA phase III awarded to Cray to build next generation machine

CRAY SIGNS $250 MILLION AGREEMENT WITH DARPA TO DEVELOP BREAKTHROUGH ADAPTIVE SUPERCOMPUTER

SEATTLE, WA, November 21, 2006 -- Global supercomputer leader Cray Inc. announced today that it has been awarded a $250 million agreement from the U.S. Defense Advanced Research Projects Agency (DARPA).

Under this agreement, Cray will develop a revolutionary new supercomputer based on the company's Adaptive Supercomputing vision, a phased approach to hybrid computing that integrates a range of processing technologies into a single scalable platform.

[...]
• Globally addressable memory with unified addressing architecture
• Configurable network, memory, processing and I/O
• Heterogeneous processing across node types, and within MVP nodes
• Can adapt at configuration time, compile time, run time
General Cascade library issues

- Heterogeneity
  - Which processor?
  - Separate library implementations on both the Opteron and the Scorpio compute nodes must agree…
  - How do we support multiple libraries simply?
  - How much decision making is at compile time, and how much at runtime?

- Ease of use
  - Hiding machine complexity from users is central, libraries are a big part

- Petascaling
re-visiting CSR MV

\[
\begin{align*}
    \text{do } q &= 1, n_{\text{rhs}} \\
    \text{next\_row\_begin} &= \text{row\_start} (1) \\
    \text{do } i &= 1, n_{\text{rows}} \\
    \text{row\_begin} &= \text{next\_row\_begin} \\
    \text{next\_row\_begin} &= \text{row\_start} (i + 1) \\
    ip &= 0.0 \\
    \text{do } k &= \text{row\_begin}, \text{next\_row\_begin} - 1 \\
    \text{ip} &= \text{ip} + \text{values} (k) \times \text{col\_index} (k, q) \\
    \text{end do} \\
    y (i, q) &= \text{ip} \\
    \text{end do} \\
\end{align*}
\]

Multi-thread opportunity?

Vectorize opportunity?

Scalar opportunity?
Future emphasis

- Heterogeneous compute nodes = more runtime awareness in libraries
- More complex analysis and more build complication = More automation
  - OO languages generating low-level codes
- Bigger systems and bigger scaling applications = More emphasis on sparse solvers
- More layers of software support = better integration with the community
Thank You!

Q&A