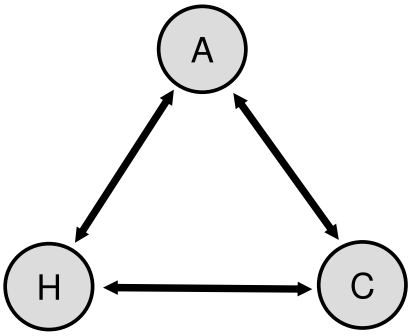
The Relevance of New Data Structures for Dense Linear Algebra in the new Multi-Core / Many Core Environments

Fred Gustavson
IBM T.J. Watson Rearch Center
Yorktown Heights, NY
E-mail: fg2@us.ibm.com

F11: Design and Analysis of Algorithms for HPC Systems
Umea University, Sweden
May 7, 2008

Fundamental "Triangle"



A: Algorithms
H: Hardware
C: Compilers

Algorithm and Architecture

The key to performance is to understand the algorithm and architecture interaction.

A significant improvement in performance can be obtained by matching the algorithm to the architecture or vice-versa.

A cost-effective way of providing a given level of performance.

Multi-core puts more of the burden on the algorithm part of the triangle

Especially hard for the designers of Library Software

Architecture

ƒ Floating point arithmetic is done in the L0 cache

ƒ 2-D Fortran and C arrays do NOT map well into the L1 and L0 caches (this combo is a core)

- The best case happens when the array is contiguous and aligned properly
- Need at least a 3 way set associative L1 cache

ƒ Floating point data must be in the L0 cache for peak performance to occur

- Multiple reuse amortizes the cost of bringing an operand to the L1 / L0 caches or core / FPU
- Multiple reuse only happens well when all operands map well into L1 / L0 or core / FPU

Dense Linear Algebra

ƒ Some scalar a(i,j) algorithms have square submatrix A(I:I+NB-1,J:J+NB-1) algorithms

- LAPACK library
- Golub and Van Loan’s book

ƒ Some square submatrices are both contiguous and fit into a L1 cache or core

ƒ Dense Matrix factorization is a level 3 computation

- Series of submatrix computations
- All submatrix computations are level 3
- In level 3 computations each matrix operand is used multiple times

Basic Algorithm Change

ƒ Map the input Fortran / C 2-D array (matrix A) to a set of contiguous submatrices that each fit into a L1 cache or core

ƒNew Data Structures

ƒ Apply the appropriate submatrix algorithm

•A series of level 3 computations whose operands are contiguous submatrices each fitting into the L1 cache and able to enter L0 or core and FPU in an optimal seamless manner

FMA Instruction

Basic Instruction of Engineering/Scientific Computation

ƒ D = C + A * B

ƒ Basic instruction of Linear Algebra

  ƒ Elementary operations and the concept of equivalence

- Key concept of linear algebra
- Adding a multiple of one row (column) to another row (column) or SIMD vector FMA
- Ax = b if and only if Ux = L^{-1} b
- Above is a series of independent FMAs

Blocking

ƒ TLB Blocking -- minimize TLB misses

ƒ Cache Blocking -- minimize cache misses

ƒ Register Blocking -- minimize load/stores

The general idea of blocking is to get the information to a high-speed storage and use it multiple times so as to amortize the cost of moving the data.

Cache Blocking -- Reduces traffic between memory and cache
Register Blocking -- Reduces traffic between cache and CPU
TLB Blocking – Covers the current working set of a problem

Some Facts on Cache Blocking

ƒ A very important algorithmic technique

ƒ First used by ESSL and the Cedar Project

ƒ Cray 2 was impetus for Level 3 BLAS

ƒ Multi-core may modify the L3 BLAS standard

ƒ The gap between memory speed and many fast cores is too great to allow the current standard to be viable

Standard Fortran and C Matrices

- A has size M rows by N cols with LDA >= M
  - Cols are stride one and rows are LDA
  - This is a one dimensional layout whereas A is 2-D
- A^T has size N rows by M cols with LDAT >= N
  - Rows are stride one and cols are LDAT
  - This is a one dimensional layout whereas A^T is 2-D
- Both A & A^T contain the same information
  - However, two copies are necessary

Standard Fortran and C Matrices

- Can not transpose a sub matrix A(r:s,u:v) in place
  - Image, $A(r:s,u:v)^T$, does not map onto A(r:s,u:v)
- This is why transpose is currently out-of-place
- Can transpose A in place if LDA = M
- We will return to this subject later

Generalization of Standard Format

- Each a(i,j) is now a rectangular or square sub matrix A(I:I+MB-1,J:J+NB-1)
  - All sub matrices are contiguous; LDA = MB
  - Simple and non-simple layouts
- Left over blocks are full; size MB*NB
  - Very important
- Can transpose rectangular or square sub matrices in place

Block Column Major Order

A =

0	5	10	15	20	25	30
1	6	11	16	21	26	31
2	7	12	17	22	27	32
3	8	13	18	23	28	33
4	9	14	19	24	29	34

- A has 500 rows and 700 columns
- Each block i, 0 <= i < 35 has size 100 by 100
- Block i is located at 10000 i

Standard Packed Matrix Arrays

- Used for symmetric and triangular matrices to conserve storage
- N vectors concatenated together
  - Lower and upper formats
  - a11, a21, a31, a22, a32, a33 is lower for n=3
  - a11, a12, a22, a13, a23, a33 is upper for n=3
- Saves storage but is very slow
  - No level 3 packed BLAS

Generalization of Packed Matrix Arrays

- Used for symmetric and triangular matrices to conserve storage
- N vectors of sub matrices concatenated together
  - Lower and upper blocked formats
  - A11, A21, A31, A22, A32, A33 is lower for N=3NB
  - A11, A12, A22, A13, A23, A33 is upper for N=3NB
- Saves storage and is very fast
  - Use level 3 BLAS or better still kernel BLAS

Square Blocked Lower Packed Format

$$A = \begin{array}{|c|c|c|c|c|c|c|c|}
\hline
0 & & & & & & & \\
1 & 8 & & & & & & \\
2 & 9 & 15 & & & & & \\
3 & 10 & 16 & 21 & & & & \\
4 & 11 & 17 & 22 & 26 & & & \\
5 & 12 & 18 & 23 & 27 & 30 & & \\
6 & 13 & 19 & 24 & 28 & 31 & 33 & \\
7 & 14 & 20 & 25 & 29 & 32 & 34 & 35 \\
\hline
\end{array}$$

- A is symmetric and has order 800
- Each block i, 0 <= i < 36 has order 100 by 100
- Block i is located at 10000 i

Blocked Mat-Mult is Optimal

Theorem:

Any algorithm that computes

$a(i, k) * b(k, j)$ for all $0<i, j, k< n+1$

must transfer between memory and an M-word cache $\Omega(n^3 / \sqrt{M})$ words if $M < n^2 / 5$.

$Ax = b$ if and only if $Ux = L^{-1}b$

- Principle of Equivalence in Linear Algebra
- Instead of performing Gaussian Elimination

do the same thing : perform N linear transformations on A to get an equivalent matrix U.

- Conclude: Instead of a collection of

Factorization Algorithms one now has a single procedure of just applying linear transformations.

Matrix Multiplication is Pervasive

- Let R and S be linear transformations
- Let T = S (R) be linear
- Let R and S have basis vectors
- The basis for T, in terms of R and S bases, defines matrix multiplication
- The definition is due to Arthur Cayley the man who invented matrices

Summary of Last Three Slides

- Sketch of a proof that matrix factorization is almost all matrix multiplication

  a) Perform n = N/NB rank NB linear transformations on A to get say U; here PA=LU

  b) Each of these n composed NB linear transformations is matrix multiply by definition

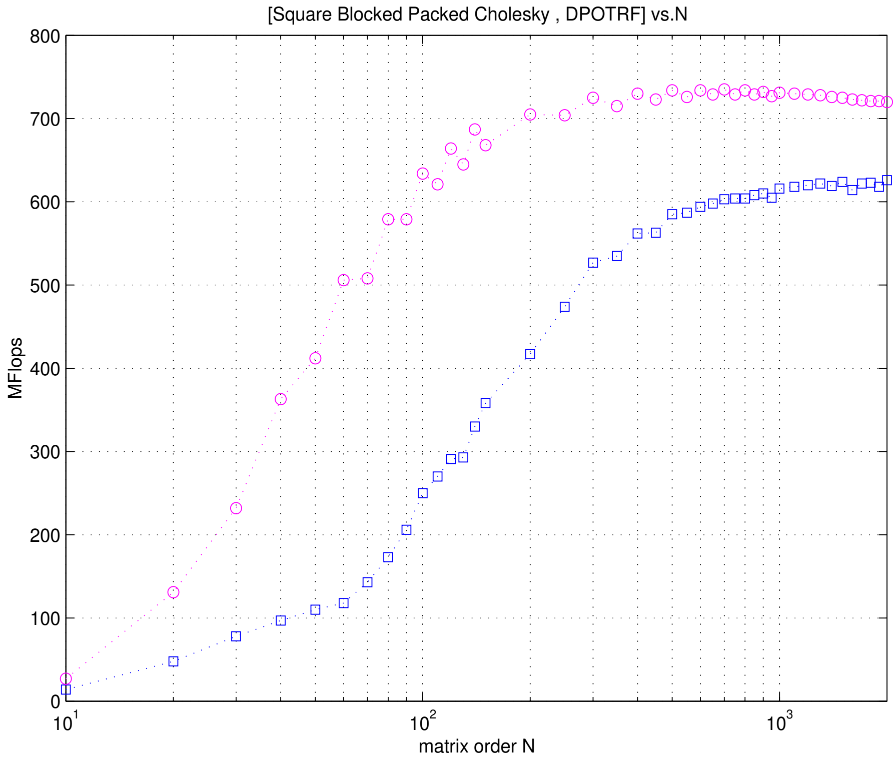
  c) These n transformations preserve the solution properties of Ax = b if and only if Ux = L^{-1}b by the principle of equivalent matrices

Blocked Based Algorithms a la LAPACK

- N coordinate transformations represented as n = N/NB composed rank NB coordinate transformations
- View as a series of kernel algorithms
  - c(i, j)=c(i, j) - a(i, k)*b(k, j) : GEMM, SYRK : C=C-A*B
  - b(i, j)=b(i, j)/a(j, j) : TRSM : B = B*A^{-1}
  - L*U=P*A : Factor Kernel
  - L*L^{T}=A : Cholesky Kernel
  - Q*R=A : QR Kernel
- LAPACK treats factor kernels as a series of NB level two operations
- Factor kernels can be written as level 3 kernels
  - Recursion is helpful
  - Register based programming

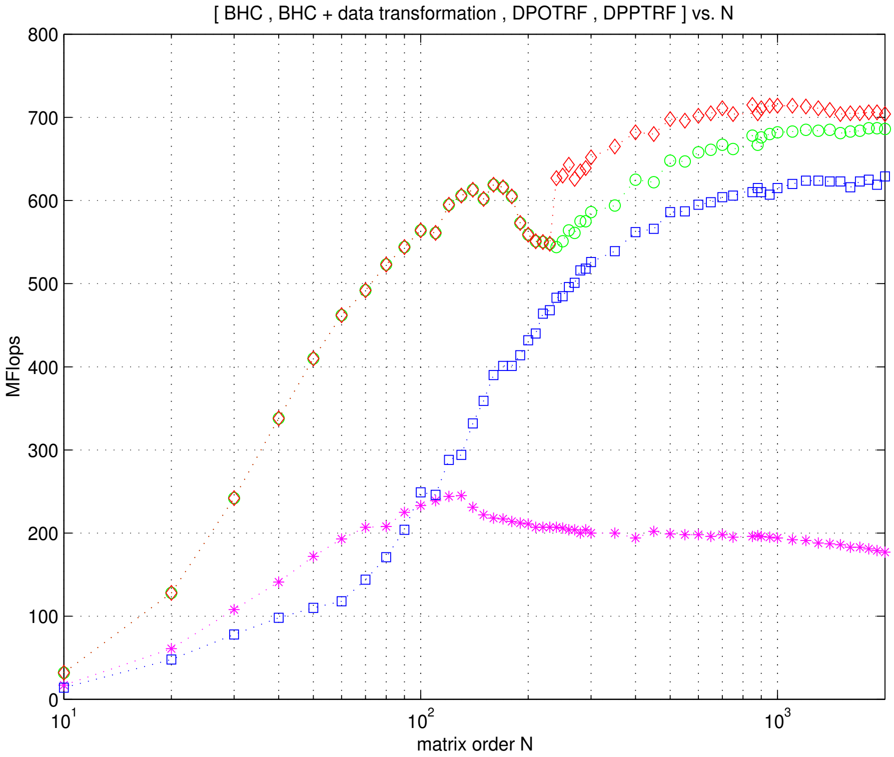
Square Blocked Packed Cholesky vs. DPOTRF

Run on 200 MHz Power3 (Peak 800 Mflops)



Blocked Hybrid Cholesky vs. DPOTRF and DPPTRF

Run on 200 MHz Power3 (Peak 800 Mflops)



Challenge of Machine Independent Design of Dense Linear Algebra Codes via the BLAS

Currently done via the BLAS

- Computer manufacturers supply high performance BLAS
- A dense linear algebra algorithm and its calls to BLAS are related

Examples

- Cholesky; all matrix operands to DTRSM, DSYRK, and hence DGEMM are submatrices of A.
- General Matrix Factor, QR factor,..., : the same is true as for Cholesky.

These examples suggest a general pattern.

Challenge of Machine Independent Design of Dense Linear Algebra Codes via the BLAS

Every Dense Linear Algebra Algorithm calls the BLAS several times. Every one of the multiple BLAS calls has all of its matrix operands equal to the submatrices of the matrices, A, B, ... of the dense linear algebra algorithm.

Can this apparent general truth be exploited?

Can We Exploit This General Relationship?

What do the current BLAS do?

- They try to exploit architecture design while maintaining functionality of the BLAS

Take Level 3 BLAS:

- Factorization algorithms are level 3 algorithms
- Data operands are copied to achieve cache blocking with minimal L1, L2 and TLB misses
- Reason for level 3 BLAS

Repeated calls to BLAS 3 require that multiple data copying be done

- On operands that are related

Can We Exploit This General Relationship?

An answer: change the data structure of the input matrices!

Change must reflect what the BLAS does repetitively.

- Store matrix as aligned contiguous BLOCKS

How are the BLOCKS to be stored?

- BLOCK ROW
- BLOCK COLUMN
- other but still contiguous

Changes

- Dense Linear Algorithm Code Change
  - Changes are minor
  - Current codes are currently blocked based
- BLAS Code Changes
  - No data copy
  - Codes become simpler
  - Higher performance
- Overall performance of Dense Linear Algorithm Codes improve.

Changes are happening now

ƒ Multi-core is forcing this change

- New codes are using two D layouts
- Provably Allows better scaling

ƒ BLAS Code Changes

- No data copy
- Codes are now kernel BLAS
- Can overlap communication and computation

ƒ A programming price is being paid by algorithm designers.

Application of LU=PA on Cell

ƒ Apply the Algorithm and Architecture Approach

- Fast single precision unit
- Use iterative refinement

- Work of Jack Dongarra's team at Univ. Tenn.
  - Linpack Benchmark LU = PA
  - Iterative refinement is O(N^2)
  - Factorization is O(N^3)
  - Use extra storage of a factor of 1.5 times 2
  - Use of BDL was deemed crucial

ƒ Overlapping computation with communication is an architectural feature of the Cell processor

Look ahead Idea for Factorization

ƒ Overlap Schur Complement Update aka matrix multiplication with the previous factor step

ƒ $PA = (L_1U_1)(L_2U_2)\ldots L_n = L_1(U_1L_2)\ldots(U_{n-1}L_n)$

ƒ L part is factor and scale and U part is SC update

- factor step provide the A and B operands of the update GEMM part
- with this use of the associative law the A & B of parts of GEMM is done early aka lookahead
- factorization is almost 100% Update
- makes factorization almost perfectly parallel

Block Data Layout

ƒ Block Data Layout is another name for Square Block Format which we described in this talk

ƒ Design of LU = PA for the Cell processor

ƒ Quotes from Jack Dongarra's et. al. paper

- "most important one is block layout"
- "unlikely that data layout can be hidden within the BLAS"
- "how should block layout be exposed to the user"

Matrices A and A^T in Storage

- Let A be an n x n matrix
- A^T is an n x n matrix
- When A is symmetric only half of A need be stored as $A = A^T$
- Full storage is used as packed storage gives very poor performance in LAPACK
- Half the storage is wasted by LAPACK full symmetric and triangular routine

Triangular Matrices in Storage

Let A be an order N symmetric matrix
Fact: A can be represented by either an an upper or lower triangular matrix

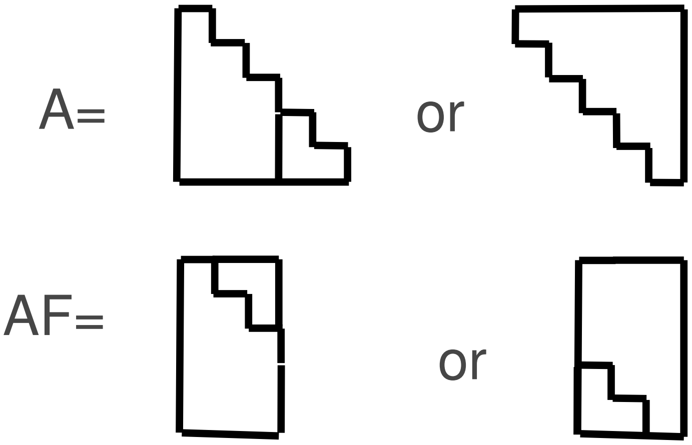
A Triangular Matrix A as a full Rectangular Matrix AF

Let A be an upper or lower triangular matrix of order N

If N = 2*k then A is also a N+1 by k rectangular matrix AF. If N = 2*k+1 then A is also a N by k+1 rectangular matrix AF.

Packed matrices in LAPACK can represent-ed as full matrices. This means that packed LAPACK routines can be Level 3 routines and also use the same minimal storage as packed storage uses.

Representing a Triangular Matrix an order N = 5 as a Rectangular Matrix



Packed or Full LAPACK Algorithms for a Triangular Matrix

- Both these Algorithms can be replaced by a single new simply related algorithm using the AF rectangular array. The new code is obtained from existing Lapack code.
- Any Lapack Algorithm for a Triangular Matrix has two sub-algorithms, 'U' and 'L'
- Conclude: Four algorithms reduce to a single algorithm. There are eight cases

Simply Related Algorithm

$$A = \begin{array}{c} A_{00} \setminus A_{11} \\ A_{10} \end{array}$$

1 Lapack Algorithm on A_{00}

2 A_{10} = BLAS (A_{00},A_{10})

3 A_{10} = BLAS (A_{10},A_{11})

4 Lapack Algorithm on A_{11}

Example Cholesky ('U')

DPPTRF and DPOTRF

Must use DPOTRF

1 DPOTRF (‘L’,A_{00})

2 DTRSM (‘L’,‘L’,‘N’,‘N’, A_{00},A_{01})

3 DSYRK (‘U’,‘T’,A_{01},A_{11})

4 DPOTRF (‘U’,A_{11})

Thank You!

IBM Thomas J. Watson Research Center
Yorktown Heights, New York