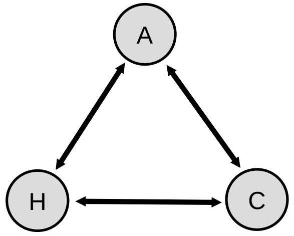
Blocking and New Generalized Data Structures Lead to Very High Performance Dense Linear Algebra Algorithms and on Cell for the Linpack Benchmark

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

Multi-cores / Many Cores 2007
San Francisco California
March 2, 2007

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.

Architecture

- Floating point arithmetic is done in the L0 cache
- 2-D Fortran and C arrays do NOT map well into the L1 and L0 caches
  - The best case happens when the array is contiguous and aligned properly
  - Need at least a 3 way set associative L1 cache
- 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 and L0 caches
  - Multiple reuse only happens well when all operands map well into the L1 and L0 caches

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 L1 cache

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 L1 cache

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 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)
  - 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

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

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

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

Standard Full Format (Column Major Order)

A is M by N, LDA >= M

Example: M=11, N=10, LDA=12

```
 1  13  25  37  49  61  73  85   97 109
 2  14  26  38  50  62  74  86   98 110
 3  15  27  39  51  63  75  87   99 111
 4  16  28  40  52  64  76  88  100 112
 5  17  29  41  53  65  77  89  101 113
 6  18  30  42  54  66  78  90  102 114
 7  19  31  43  55  67  79  91  103 115
 8  20  32  44  56  68  80  92  104 116
 9  21  33  45  57  69  81  93  105 117
10  22  34  46  58  70  82  94  106 118
11  23  35  47  59  71  83  95  107 119
 *   *   *   *   *   *   *   *    *   *
```

Block Hybrid Full Format (Row Major Order)

A is M by N, LDA >= M

Example: M=11, N=10, LDA=12

```
 1  2 | 25  26  27  28 | 73  74   75  76
 3  4 | 29  30  31  32 | 77  78   79  80
 5  6 | 33  34  35  36 | 81  82   83  84
 7  8 | 37  38  39  40 | 85  86   87  88
 9 10 | 41  42  43  44 | 89  90   91  92
11 12 | 45  46  47  48 | 93  94   95  96
13 14 | 49  50  51  52 | 97  98   99 100
15 16 | 53  54  55  56 | 101 102 103 104
17 18 | 57  58  59  60 | 105 106 107 108
19 20 | 61  62  63  64 | 105 110 111 112
21 22 | 65  66  67  68 | 113 114 115 116
 *  * |  *   *   *   * |  *   *   *   *
```

Square Blocked Lower Packed Order

A =

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

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

Square Blocked Lower Packed Format

Example NB=4, TRANS='T'

```
  1   *   *   *
  5   6   *   *
  9  10  11   *
 13  14  15  16
 17  18  19  20  49   *   *   *
 21  22  23  24  53  54   *   *
 25  26  27  28  57  58  59   *
 29  30  31  32  61  62  63  64
 33  34  35  36  65  66  67  68  81   *   *   *
 37  38  39  40  69  70  71  72  85  86   *   *
  *   *   *   *   *   *   *   *   *   *   *   *
  *   *   *   *   *   *   *   *   *   *   *   *
```

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 $W(n^3/\sqrt{M})$ 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

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 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

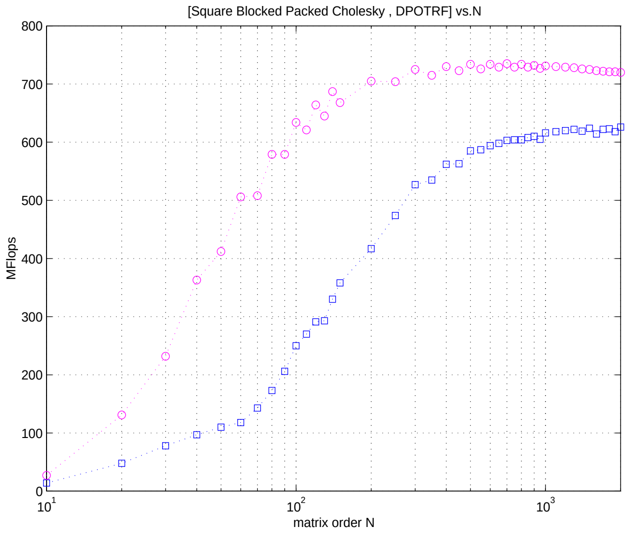
- N coordinate transformations represented as n = N/NB rank NB coordinate transformations
- View as a series of kernel algorithms

c(i, j)=c(i, j) - a(i, k)*b(k, j)	:	GEMM, SYRK
b(i, j)=b(i, j)/a(j, j)	:	TRSM
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 usually be written as level 3 kernels
  - Recursion is helpful
  - Register based programming

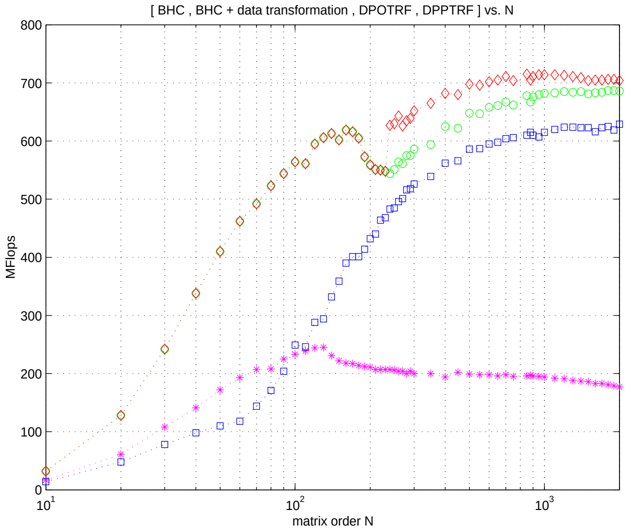
Square Blocked Packed Cholesky vs. DPOTRF

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 BLOCKS

How are the BLOCKS to be stored?

- BLOCK ROW
- BLOCK COLUMN

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.

Application of LU=PA on Cell

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

Look ahead Idea for Factorization

- Overlap Schur Complement Update with the previous factor step
- $PA = (L_1U_1)(L_2U_2)\ldots L_n = L_1(U_1L_2)\ldots(U_{n-1}L_n)$
- Schur Complement is the Gemm part of LU=PA
  - factor step provide the A and B operands of GEMM
  - with look ahead A & B is done beforehand
  - factorization is almost 100% Update
  - makes factorization almost perfectly parallel

Block Data Layout

- Block Data Layout is another name for Square Block Format which we described in this talk
- Design of LU = PA for the Cell processor
  - "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

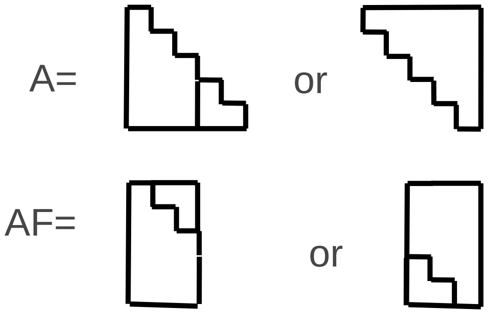
A Triangular Matrix A as a full Rectangular Matrix AF

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 represented 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

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

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})

New LAPACK Type Routine

Example: Symmetric Indefinite Factorization

Subroutine DBSTRF (uplo, n, ap, ipiv, nsinfo)

nsinfo >= n (n+1)/2

ap is in standard packed format

nsinfo is the size of ap

Algorithm:

- Use ap (1:ns) to map ap to abpo in place
- Do level 3 Bunch Kaufman factor of abpo

Note: If nso > nsinfo, return -nso in nsinfo and don't change ap.

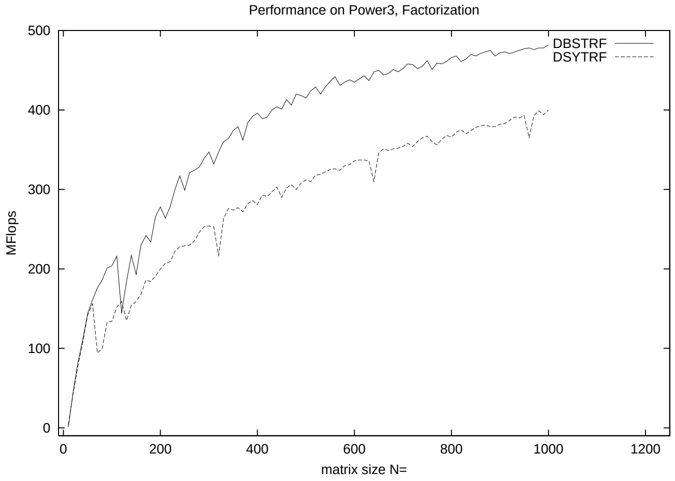
Blocked Lower Packed Overlapped Format

Example: NB=4, LDA=12, M=10

```
 1   *   *   *   *
 2  14   *   *   *
 3  15  27   *   *
 4  16  28  40   *
 5  17  29  41  53   *   *   *   *
 6  18  30  42  54  62   *   *   *
 7  19  31  43  55  63  71   *   *
 8  20  32  44  56  64  72  80   *
 9  21  33  45  57  65  73  81  89   *
10  22  34  46  58  66  74  82  90  94
 *   *   *   *   *   *   *   *   *   *
 *   *   *   *   *   *   *   *   *   *
```

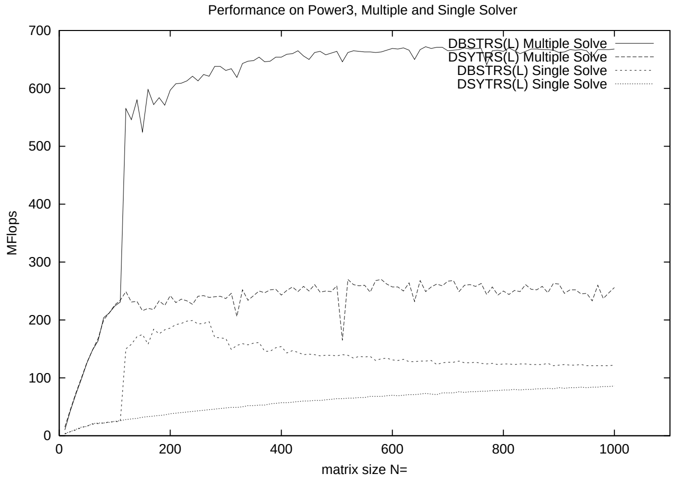
DBSTRF vs. DSYTRF

Run on 200 MHz Power3 (Peak 800 Mflops)



DBSTRS vs. DSYTRS

Run on 200 MHz Power3 (Peak 800 Mflops)



Thank You!

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