Using high performance libraries

- Stefano Cozzini
- CNR-IOM and eXact lab srl

7 Motifs in HPC..

Phil Colella (LBL) identified 7 kernels of which most simulation and data analysis program are composed:

- Dense Linear Algebra
  - Ex: solve Ax=B or Ax=lambdax where A is a dense matrix
- Sparse Linear Algebra
  - Ex: solve Ax=B or Ax=lambdax where A is a sparse matrix (mostly zero)
- Operation on structured Grids:
  - Ex: ANEWj()=4*(A(i,j)-A(i-1,j)-A(i+1,j) -A(i,j-1)-A(i,j+1)
- Operation on unstructured Grids:
  - Ex; similar but list of neighbours varies from entry to entry
- Spectral Methods
  - Ex: Fast Fourier Transform (FFT)
- Particle Methods
  - Ex: Compute electrostatic forces on n-particles
- Monte Carlo
  - Ex: many independent simulation using different inputs

Where should you start optimizing your application ?

Optimization Techniques

- There are basically three different categories:
  - Improve memory performance (the most important)
  - Improve CPU performance
  - Use already highly optimized libraries/subroutines
    - The easiest and more efficient way..

What are Performance libraries ?

- Routines for common (math) functions such as vector and matrix operations, fast Fourier transform etc. written in a specific way to take advantage of capabilities of the CPU.
- Each CPU type normally has its own version of the library specifically written or compiled to maximally exploit that architecture

What are Performance libraries ?

- Routines for common (math) functions such as vector and matrix operations, fast Fourier transform etc. written in a specific way to take advantage of capabilities of the CPU.
- Each CPU type normally has its own version of the library specifically written or compiled to maximally exploit that architecture

Why use performance libraries ?

- Compilers can optimize code only to a certain point. Effective programming needs deep knowledge of the platform
- Performance libraries are designed to use the CPU in the most efficient way, which is not necessarily the most straightforward way.
- It is normally best to use the libraries supplied by or recommended by the CPU vendor
- On modern hardware they are hugely important, as they most efficiently exploit caches, special instructions and parallelism

Any other reason apart from optimization ?

- Usage of libraries
  - Make coding easier. Complicated math operations can be used from existing routines
  - Increase portability of code as standard (and well optimized) libraries exist for ALL computing platforms.
- Lego approach: build your own code using already available bricks..

What is available ?

- Linear Algebra: BLAS/LAPACK/SCALAPACK
- FFT:
  - FFTW
- ODE/PDE
  - PETSC
- Machine Learning:
  - Tensorflow / Caffe etc..

Should I write my own algorithm for L. A. ?

99.99% of time NO

- Tons of libraries out there
- Well tested
- Extremely efficient in 99.99% of the case
- With some "de facto" standard implemented

PORTABILITY IS COMING (almost) FOR FREE

Why Linear algebra ?

3 Basic Linear Algebra Problems in

1. Linear Equations: Solve Ax=b for x

2. Least Squares: Find x that minimizes $||r||_2 \equiv \sqrt{\Sigma\, r_i^2}$ where r=Ax-b
   - Statistics: Fitting data with simple functions

3a. Eigenvalues: Find λ and x where $Ax = \lambda x$
   - Vibration analysis, e.g., earthquakes, circuits

3b. Singular Value Decomposition: $A^TAx=\sigma^2x$
   - Data fitting, Information retrieval

Lots of variations depending on structure of A
   - A symmetric, positive definite, banded, …

Why dense Linear Algebra ?

- Many large matrices are sparse, but …
  - Dense algorithms easier to understand
  - Some applications yields large dense matrices
  - LINPACK Benchmark (www.top500.org)
    - "How fast is your computer?" =
      "How fast can you solve dense Ax=b?"
  - Large sparse matrix algorithms often yield smaller (but still large) dense problems

BLAS: Basic Linear Algebra Subprograms

BLAS history (1/3)

- In the beginning it was libraries like EISPACK (for eigenvalue problems)
- Then the BLAS-1 were invented (1973-1977)
  - Create a standard library of 15 operations (mostly) on vectors
  - “AXPY” ($y = \alpha \cdot x + y$), dot product, scale ($x = \alpha \cdot x$), etc
  - Up to 4 versions of each (S/D/C/Z), 46 routines, 3300 LOC
  - Language: FORTRAN
- Goals
  - Common “pattern” to ease programming, readability
  - Robustness, via careful coding (avoiding over/underflow) --> Accuracy
  - Portability (common interface)
  - Efficiency via machine specific implementations
  - Maintaibility
- Why BLAS-1 ? They do O(n) ops on O(n) data
  - Used in libraries like LINPACK (for linear systems)
  - Source of the name “LINPACK Benchmark” (not the code!)

BLAS history (2/3)

- But the BLAS-1 weren't enough
  - Consider AXPY ($y = \alpha \cdot x + y$): 2n flops on 3n read/writes
  - Computational intensity = (2n)/(3n) = 2/3
  - Too low to run near peak speed (read/write dominates)
- So the BLAS-2 were developed (1984-1986)
  - Standard library of 25 operations (mostly) on matrix/vector pairs
  - "GEMV": $y = \alpha \cdot A \cdot x + \beta \cdot x$, "GER": $A = A + \alpha \cdot x \cdot y^T$, $x = T^{-1} \cdot x$
  - Up to 4 versions of each (S/D/C/Z), 66 routines, 18K LOC
- Why BLAS-2 ?
  - They do $O(n^2)$ ops on $O(n^2)$ data
  - So computational intensity still just $\sim(2n^2)/(n^2) = 2$
  - OK for vector machines, but not for machine with caches

BLAS history (3/3)

- The next step: BLAS-3 (1987-1988)
  - Standard library of 9 operations (mostly) on matrix/matrix pairs
    - "GEMM": $C = \alpha \cdot A \cdot B + \beta \cdot C$, $C = \alpha \cdot A \cdot A^T + \beta \cdot C$, $C = T^{-1} \cdot B$
    - Up to 4 versions of each (S/D/C/Z), 30 routines, 10K LOC
  - Why BLAS 3 ? They do $O(n^3)$ ops on $O(n^2)$ data
  - So computational intensity $(2n^3)/(4n^2) = n/2$ – big at last!
    - Good for machines with caches, other mem. hierarchy levels
  - Performing implementations left to others..

Where are BLAS ?

http://www.netlib.org/blas

- Source: 142 routines, 31K LOC,
- Testing: 28K LOC
- Reference (unoptimized) implementation only !
  - http://www.netlib.org/blas/#_reference_blas_version_3_5_0
  - Ex: 3 nested loops for GEMM

BLAS list

Level 1 BLAS

	dim scalar vector vector scalars 5-element array		prefixes
SUBROUTINE xROTG	(A, B, C, S)	Generate plane rotation	S, D
SUBROUTINE xROTMG	(D1, D2, A, B, PARAM)	Generate modified plane rotation	S, D
SUBROUTINE xROT	(N, X, INCX, Y, INCY, C, S)	Apply plane rotation	S, D
SUBROUTINE xROTM	(N, X, INCX, Y, INCY, PARAM)	Apply modified plane rotation	S, D
SUBROUTINE xSWAP	(N, X, INCX, Y, INCY)	$x \leftrightarrow y$	S, D, C, Z
SUBROUTINE xSCAL	(N, ALPHA, X, INCX)	$x \leftarrow \alpha x$	S, D, C, Z, CS, ZD
SUBROUTINE xCOPY	(N, X, INCX, Y, INCY)	$y \leftarrow x$	S, D, C, Z
SUBROUTINE xAXPY	(N, ALPHA, X, INCX, Y, INCY)	$y \leftarrow \alpha x + y$	S, D, C, Z
FUNCTION xDOT	(N, X, INCX, Y, INCY)	$dot \leftarrow x^T y$	S, D, DS
FUNCTION xDOTU	(N, X, INCX, Y, INCY)	$dot \leftarrow x^T y$	C, Z
FUNCTION xDOTC	(N, X, INCX, Y, INCY)	$dot \leftarrow x^H y$	C, Z
FUNCTION xxDOT	(N, X, INCX, Y, INCY)	$dot \leftarrow \alpha + x^T y$	SDS
FUNCTION xNRM2	(N, X, INCX)	$nrm2 \leftarrow \|\|x\|\|_2$	S, D, SC, DZ
FUNCTION xASUM	(N, X, INCX)	$asum \leftarrow \|\|re(x)\|\|_1 + \|\|im(x)\|\|_1$	S, D, SC, DZ
FUNCTION IxAMAX	(N, X, INCX)	$amax \leftarrow 1^{st} k \ni \|re(x_k)\| + \|im(x_k)\| = max(\|re(x_i)\| + \|im(x_i)\|)$	S, D, C, Z

Level 2 BLAS

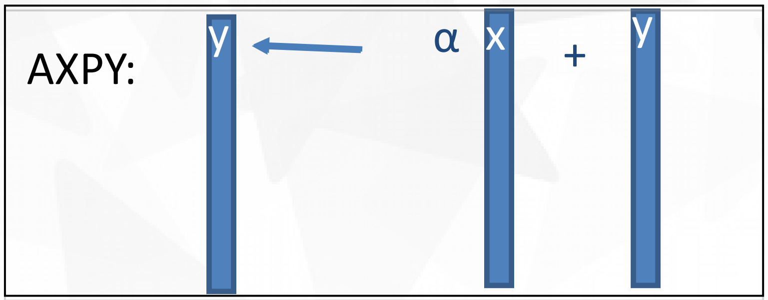
	options dim b-width scalar matrix vector scalar vector		prefixes
xGEMV	(TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
xGBMV	(TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
xHEMV	(UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xHBMV	(UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xHPMV	(UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xSYMV	(UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xSBMV	(UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xSPMV	(UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xTRMV	(UPLO, TRANS, DIAG, N, A, LDA, X, INCX)	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTBMV	(UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTPMV	(UPLO, TRANS, DIAG, N, AP, X, INCX)	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTRSV	(UPLO, TRANS, DIAG, N, A, LDA, X, INCX)	$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
xTBSV	(UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)	$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
xTPSV	(UPLO, TRANS, DIAG, N, AP, X, INCX)	$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
	options dim scalar vector vector matrix		
xGER	(M, N, ALPHA, X, INCX, Y, INCY, A, LDA)	$A \leftarrow \alpha xy^T + A, A - m \times n$	S, D
xGERU	(M, N, ALPHA, X, INCX, Y, INCY, A, LDA)	$A \leftarrow \alpha xy^T + A, A - m \times n$	C, Z
xGERC	(M, N, ALPHA, X, INCX, Y, INCY, A, LDA)	$A \leftarrow \alpha xy^H + A, A - m \times n$	C, Z
xHER	(UPLO, N, ALPHA, X, INCX, A, LDA)	$A \leftarrow \alpha xx^H + A$	C, Z
xHPR	(UPLO, N, ALPHA, X, INCX, AP)	$A \leftarrow \alpha xx^H + A$	C, Z
xHER2	(UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA)	$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
xHPR2	(UPLO, N, ALPHA, X, INCX, Y, INCY, AP)	$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
xSYR	(UPLO, N, ALPHA, X, INCX, A, LDA)	$A \leftarrow \alpha xx^T + A$	S, D
xSPR	(UPLO, N, ALPHA, X, INCX, AP)	$A \leftarrow \alpha xx^T + A$	S, D
xSYR2	(UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA)	$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D
xSPR2	(UPLO, N, ALPHA, X, INCX, Y, INCY, AP)	$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D

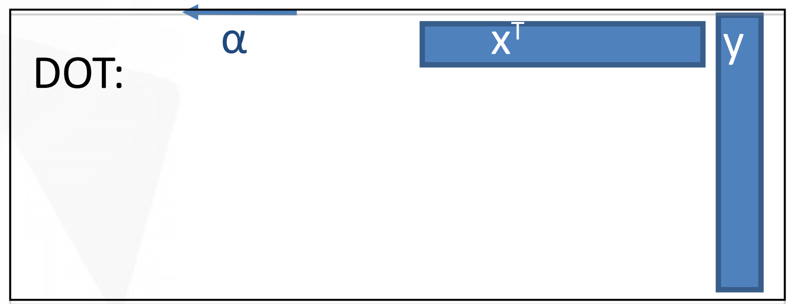
Level 3 BLAS

	options dim scalar matrix matrix scalar matrix		
xGEMM	(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)	$C \leftarrow \alpha op(A)op(B) + \beta C, op(X) = X, X^T, X^H, C - m \times n$	S, D, C, Z
xSYMM	(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^T$	S, D, C, Z
xHEMM	(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^H$	C, Z
xSYRK	(UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)	$C \leftarrow \alpha AA^T + \beta C, C \leftarrow \alpha A^T A + \beta C, C - n \times n$	S, D, C, Z
xHERK	(UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)	$C \leftarrow \alpha AA^H + \beta C, C \leftarrow \alpha A^H A + \beta C, C - n \times n$	C, Z
xSYR2K	(UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)	$C \leftarrow \alpha AB^T + \bar{\alpha} BA^T + \beta C, C \leftarrow \alpha A^T B + \bar{\alpha} B^T A + \beta C, C - n \times n$	S, D, C, Z
xHER2K	(UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)	$C \leftarrow \alpha AB^H + \bar{\alpha} BA^H + \beta C, C \leftarrow \alpha A^H B + \bar{\alpha} B^H A + \beta C, C - n \times n$	C, Z
xTRMM	(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)	$B \leftarrow \alpha op(A)B, B \leftarrow \alpha Bop(A), op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z
xTRSM	(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)	$B \leftarrow \alpha op(A^{-1})B, B \leftarrow \alpha Bop(A^{-1}), op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z

Level 1, 2 and 3 BLAS

Level 1 BLAS Vector-Vector operations

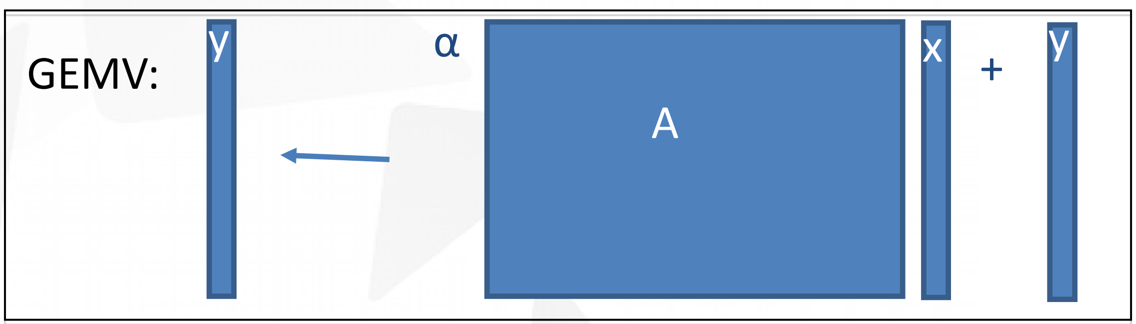




2n FLOP
2n memory reference

RATIO: 1

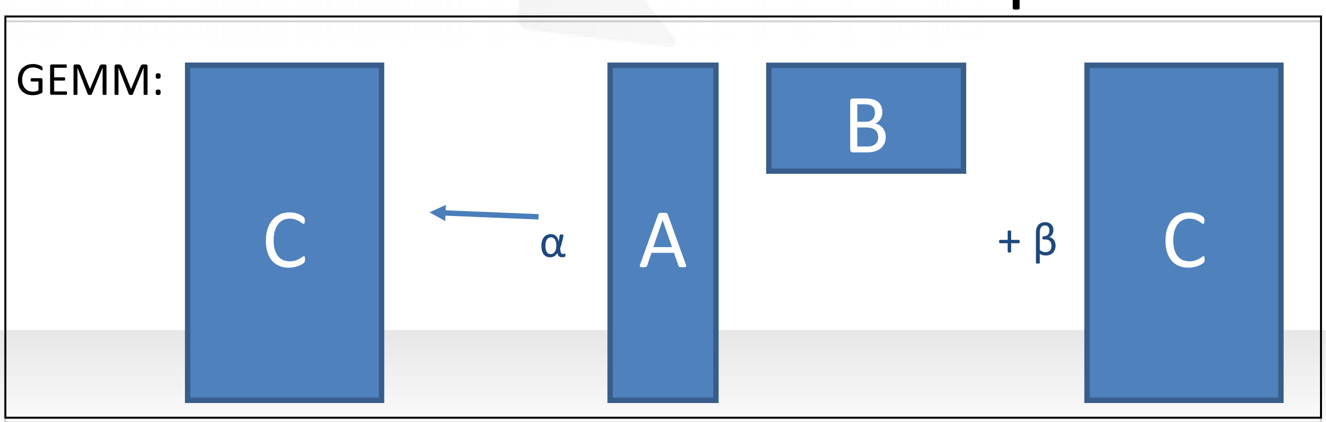
Level 2 BLAS Matrix-Vector operations



$2n^2$ FLOP
n^2 memory references

RATIO: 2

Level 3 BLAS Matrix-Matrix operations



$2n^3$ FLOP
$4n^2$ memory references

RATIO: 2 n

Why BLAS so important?

- Because the BLAS are efficient, portable, parallel, and widely available, they are commonly used in the development of high quality linear algebra software.

- Performance of lot of applications depends a lot on the performance of the underlying BLAS

Standardization (BLAS example)

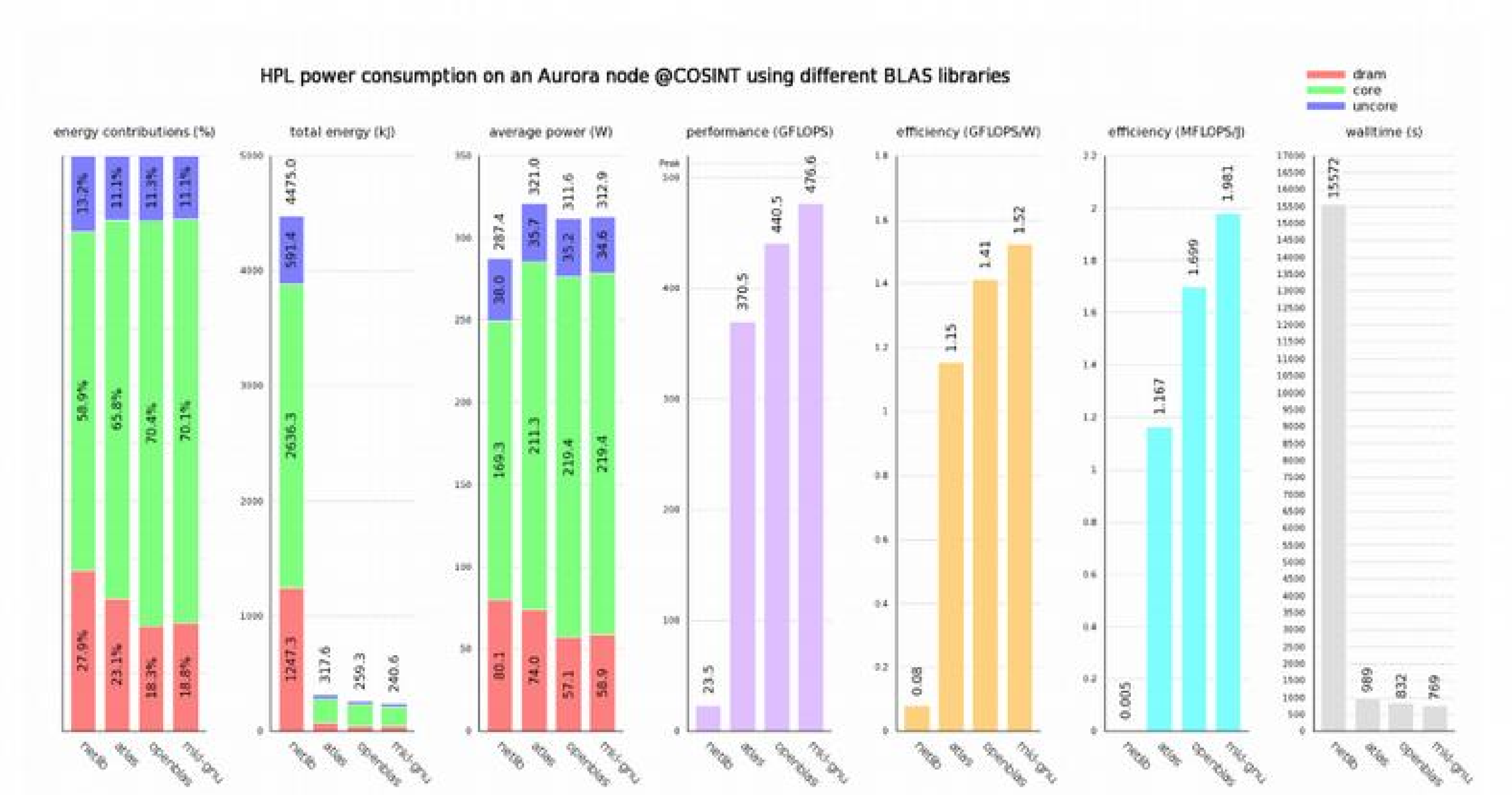
- Each BLAS Subroutines have a standardized layout
- BLAS is documented in the source code
- Man pages exist
- Vendor supplied docs
- Different BLAS implementations have the same calling sequence

```
      SUBROUTINE DGEMM ( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB,
     $                   BETA, C, LDC )
*     .. SCALAR ARGUMENTS ..
      CHARACTER*1        TRANSA, TRANSB
      INTEGER            M, N, K, LDA, LDB, LDC
      DOUBLE PRECISION   ALPHA, BETA
*     .. ARRAY ARGUMENTS ..
      DOUBLE PRECISION   A( LDA, * ), B( LDB, * ), C( LDC, * )
*     ..
*
*  PURPOSE
*  =======
*
*  DGEMM  PERFORMS ONE OF THE MATRIX-MATRIX OPERATIONS
*
*     C := ALPHA*OP( A )*OP( B ) + BETA*C,
*
*  WHERE  OP( X ) IS ONE OF
*
*     OP( X ) = X   OR   OP( X ) = X',
*
*  ALPHA AND BETA ARE SCALARS, AND A, B AND C ARE MATRICES, WITH OP( A )
*  AN M BY K MATRIX,  OP( B )  A  K BY N MATRIX AND  C AN M BY N MATRIX.
*
*  PARAMETERS
*  ==========
*
*  TRANSA - CHARACTER*1.
*           ON ENTRY, TRANSA SPECIFIES THE FORM OF OP( A ) TO BE USED IN
*           THE MATRIX MULTIPLICATION AS FOLLOWS:
*
*              TRANSA = 'N' OR 'N',  OP( A ) = A.
*
*              TRANSA = 'T' OR 'T',  OP( A ) = A'.
*
*              TRANSA = 'C' OR 'C',  OP( A ) = A'.
*
*           UNCHANGED ON EXIT.
*
*  TRANSB - CHARACTER*1.
*           ON ENTRY, TRANSB SPECIFIES THE FORM OF OP( B ) TO BE USED IN
*           THE MATRIX MULTIPLICATION AS FOLLOWS:
*
*              TRANSB = 'N' OR 'N',  OP( B ) = B.
*
*              TRANSB = 'T' OR 'T',  OP( B ) = B'.
```

Vendor/Optimized BLAS libraries

- ACML
  - The AMD Core Math Library, supporting the AMD processors
- ATLAS
  - Automatically Tuned Linear Algebra Software, an open source implementation of BLAS APIs for C and Fortran 77
- Intel MKL
  - The Intel Math Kernel Library, supporting x86 32-bits and 64-bits. Includes optimizations for Intel Pentium, Core and Intel Xeon CPUs and Intel Xeon Phi; support for Linux, Windows and Mac OS X
- cuBLAS
  - Optimized BLAS for NVIDIA based GPU cards
- clBLAS
  - An OpenCL implementation of BLAS
- ESSL
  - IBM's Engineering and Scientific Subroutine Library, supporting the PowerPC architecture under AIX and Linux
- GotoBLAS
  - Kazushige Goto's BSD-licensed implementation of BLAS, tuned in particular for Intel Nehalem/Atom, VIA Nanoprocessor, AMD Opteron
- BLIS
  - BLAS-like Library Instantiation Software framework for rapid instantiation
- OpenBLAS
  - Optimized BLAS based on Goto BLAS hosted at GitHub, supporting Intel platform and other

Blas efficiency: (from Moreno B. MHPC's thesis)



What about my C++/C program ??

- BLAS routines are Fortran-style, when calling them from C-language programs, follow the Fortran-style calling conventions:
  - Pass variables by address, not by value.
  - Store your data in Fortran style, that is, column-major rather than row-major order.
- be aware that because the Fortran language is case-insensitive, the routine names can be both upper-case or lower-case, with or without the trailing underscore. For example, the following names are equivalent:

  dgemm, DGEMM, dgemm_, and DGEMM_

Use CBLAS

- C-style interface to the BLAS routines (http://www.netlib.org/blas/blast-forum/cblas.tgz)
- You can call CBLAS routines using regular C-style calls.
- The header file specifies enumerated values and prototypes of all the functions.
- For details and examples:

https://software.intel.com/en-us/mkl-tutorial-c-multiplying-matrices-using-dgemm

Efficiency: q parameter (aka computational efficiency)

Table 2: Basic Linear Algebra Subroutines (BLAS)

Operation	Definition	Floating point operations	Memory references	q
saxpy	$y_i = \alpha x_i + y_i,\ i = 1, \ldots, n$	$2n$	$3n+1$	$2/3$
Matrix–vector mult	$y_i = \sum_{j=1}^{n} A_{ij} x_j + y_i$	$2n^2$	$n^2 + 3n$	2
Matrix–matrix mult	$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj} + C_{ij}$	$2n^3$	$4n^2$	$n/2$

The parameter q is the ratio of flops to memory references.
Generally:

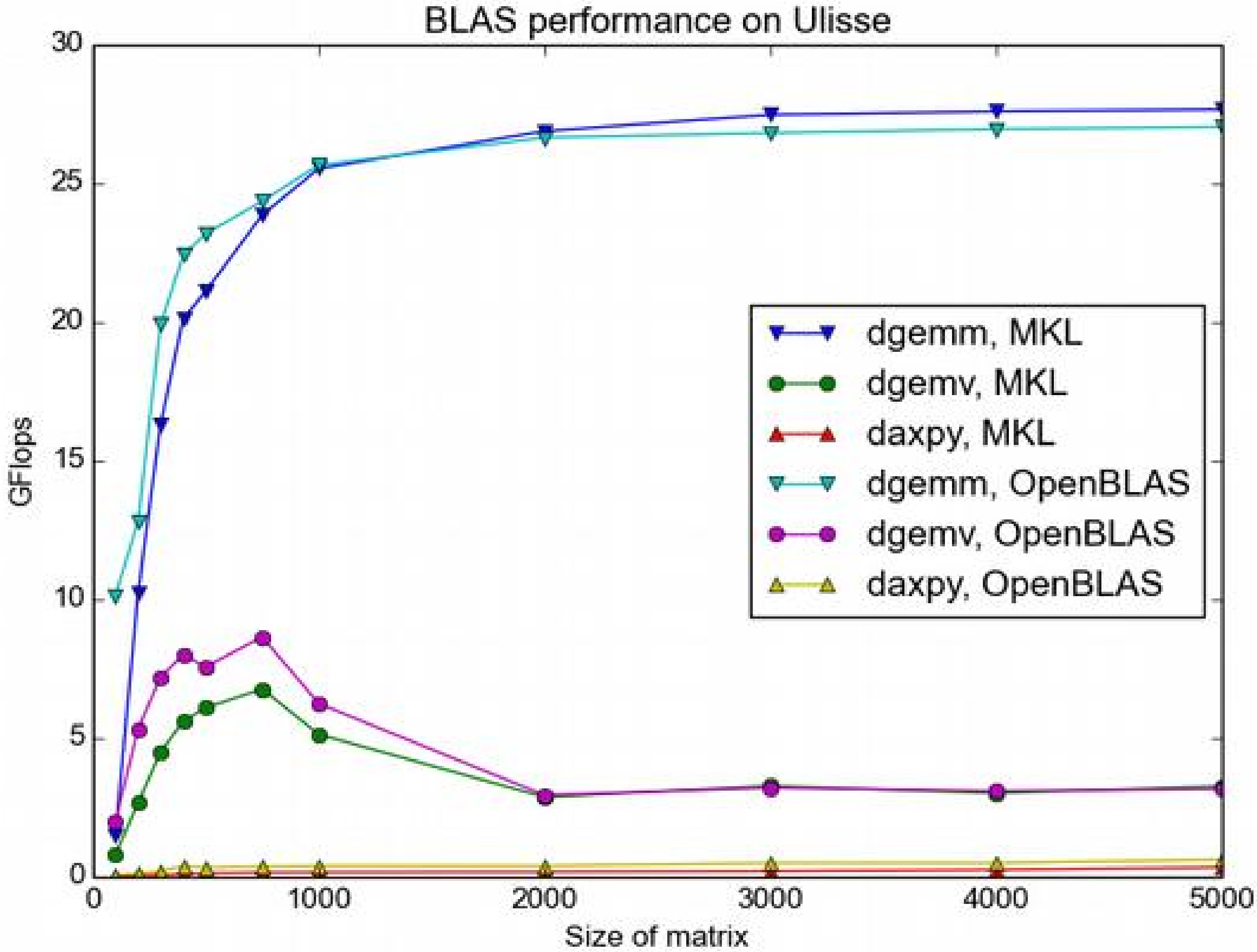
1. Larger values of q maximize useful work to time spent moving data.

2. The higher the level of the BLAS, the larger q.

It follows..

- BLAS1 are memory bounded ! (for each computation a memory transfer is required)
- BLAS2 are not so memory bounded (can have good performance on super-scalar architecture)
- BLAS3 can be very efficient on super-scalar computers because not memory bounded

Blas Performance on ULISSE



Proposed Exercise

- Create the previous graph for the HW/software we are using:
  - Using MKL
  - Using OpenBLAS
  - On a Ulisse old/new nodes
- Steps:
  - Install, if missing needed libraries in your directory
  - Write a small program to call the three routines
  - Write a script to collect all sizes of interest
  - Make nice plots

comment on BLAS

Basic Linear Algebra Subroutines

Name	Description	Examples
Level-1 BLAS	Vector Operations	$C = \sum X_i Y_i$
Level-2 BLAS	Matrix-Vector Operations	$B_i = \sum_k A_{ik} X_k$
Level-3 BLAS	Matrix-Matrix Operations	$C_{ij} = \sum_k A_{ik} B_{kj}$

How to link optimized libraries?

- Reference implementation: order matters !
  - LAPACK uses BLAS
  - => -L/usr/local/lib -llapack -lblas
- OpenBLAS:
  - Automatically includes lapack reference implementation so no need to specify anything else. Please check !
- ATLAS is written C with f77 wrappers:
  - -L/opt/atlas/lib -lf77blas -latlas
- MKL:
  - Generally complex and highly dependent on version and/or HW/SW implementation

    https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor

Exercise 2: Running HPL on our clusters

- Check the README on github account

A few notes:

- Standard input file should be present
- Beware of threads
  - How to control them ?

What about N ?

- N should be large enough to take ~75% of RAM..
  - N = sqrt (0.75 * Number of Nodes * Minimum memory of any node / 8)
- You can compute it via:
  - http://www.advancedclustering.com/act-kb/tune-hpl-dat-file/

HPL benchmark input file HPL.dat

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
50000 Ns
1             # of NBs
768        NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
4 1 2 1        Ps
4 2 2 4        Qs
16.0         threshold
1            # of panel fact
0 1 2        PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
2 8          NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
0 1 2        RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
0 2          BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1 0          DEPTHs (>=0)
1            SWAP (0=bin-exch,1=long,2=mix)
192          swapping threshold
1            L1 in (0=transposed,1=no-transposed) form
1            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
```

Parameters for HPL.dat input file

N	Problem size	**Pmap**	Process mapping
NB	Blocking factor	**threshold**	for matrix validity test
P	Rows in process grid	**Ndiv**	Panels in recursion
Q	Columns in process grid	**Nbmin**	Recursion stopping criteria
Depth	Lookahead depth	**Swap**	Swap algorithm
Bcasts	Panel broadcasting method	**L1, U**	to store triangle of panel
Pfacts	Panel factorization method	**Align**	Memory alignment
Rfacts	Recursive factorization method	**Equilibration**	

Tips to get performance..

- Figure out a good block size (NB) for the matrix multiply routine. The best method is to try a few out. If you happen to know the block size used by the matrix-matrix multiply routine, a small multiple of that block size will do fine. This particular topic is discussed in the FAQs section.
- The process mapping should not matter if the nodes of your platform are single processor computers. If these nodes are multi-processors, a row-major mapping is recommended.
- HPL likes "square" or slightly flat process grids. Unless you are using a very small process grid, stay away from the 1-by-Q and P-by-1 process grids.

What are you supposed to do ?

- Let us read together the readme file.

Thank you ...