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 m tions can be used from existing routines
  - Increase portability of code as stan well optimized) libraries exist for ALL computing pl
- Lego approach: build your own code ady available bricks..

we dont want just to optimize the code, we want to make the code easy, save time dont reinvent the wheel. Increase portability, by changing library

knowing library allows to use lego approach

What is available ?

- Linear Algebra: BLAS/LAPACK/SCALAPACK
- FFT:
  - FFTW
- ODE/PDE
  - PETSC
- Machine L
  - Tensorfl e etc.

FFT of the West

Ordinary Differential Equations/ partial differential equations Integrate heavy PDE

DL2

thousands of frameworks a lot of stuff:
tensorflow
caffe,torch
pitorch
framework where to build a machine learining library
Orwood
parallelizing tensorflow

a lot of libraries available

scalable pack(paralllel approach)
li er algebra ckage: almost linear algrbra eartion

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

Allows you to be portable, adapt the code

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 = λ 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 ?

majority of
algorithms are
well defined in
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

large sparse
matrix,
operations
sparpack sparse

dense problem
problem solved
by dense
matrices

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

need a standard library to create a

  - "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

single,
double,complex,z

no dinamic allocation
no pointers

- Goals
  - Common "pattern" to ease p[illegible]adability
  - 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!)

not important

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

poor intensity

  - Too low to run near peak speed (read/write dominates)

benchmark to read/write for memory

- So the BLAS-2 were developed (1984-1986)
  - Standard library of 25 operations (mostly) on matrix/vector pairs
  - "G[illegible] $= \alpha\cdot A\cdot x + \beta\cdot x$, "GER": $A = A + \alpha\cdot x\cdot y^T$, $x = T^{-1}\cdot x$

general matrix vector operations

  - 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

computational intensity is the number of opearations im

BLAS history (3/3)

- The next step: BLAS-3 (1987-1988)
  - Standard library of 9 operations (mostly) on matrix/matrix pairs
    - "GEMM": $C = \alpha \cdot A \cdot B + \beta \cdot C$, $C = \alpha \cdot A \cdot A^T + \beta \cdot C$, $C = T^{-1} \cdot B$

      general matrix matrix operation
    - 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. hi[illegible]vels
  - Performing implementations left to others..

here i am bounded by the cpu activity, doing cpu operations and not only memory access

Vendor has to choose the specific routine

Where are BLAS ?

http://www.netlib.org/blas

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

reference implementations, not so much performance

three nested loops you can write

BLAS list

quick look

Level 1 BLAS

Routine	dim scalar vector vector scalars 5-element array	Operation	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)	$nrm2 \leftarrow \lVert x \rVert_2$	S, D, SC, DZ
FUNCTION xASUM	(N, X, INCX)	$asum \leftarrow \lVert re(x) \rVert_1 + \lVert im(x) \rVert_1$	S, D, SC, DZ
FUNCTION IxAMAX	(N, X, INCX)	$amax \leftarrow 1^{st} k \ni \lvert re(x_k)\rvert + \lvert im(x_k)\rvert = max(\lvert re(x_i)\rvert + \lvert im(x_i)\rvert)$	S, D, C, Z

Level 2 BLAS

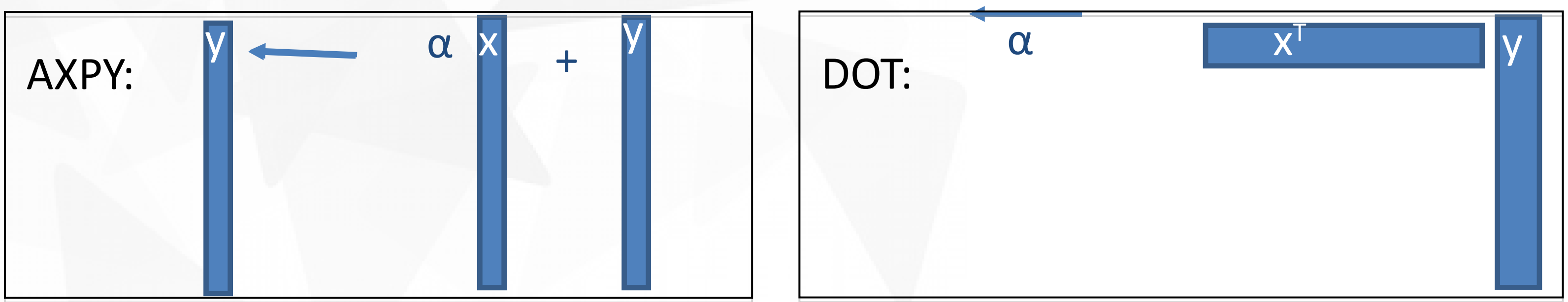
Routine	options dim b-width scalar matrix vector scalar vector	Operation	prefixes
xGEMV	(TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
xGBMV	(TRANS, M, N, KL, 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

Routine	options dim scalar matrix matrix scalar matrix	Operation	prefixes
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

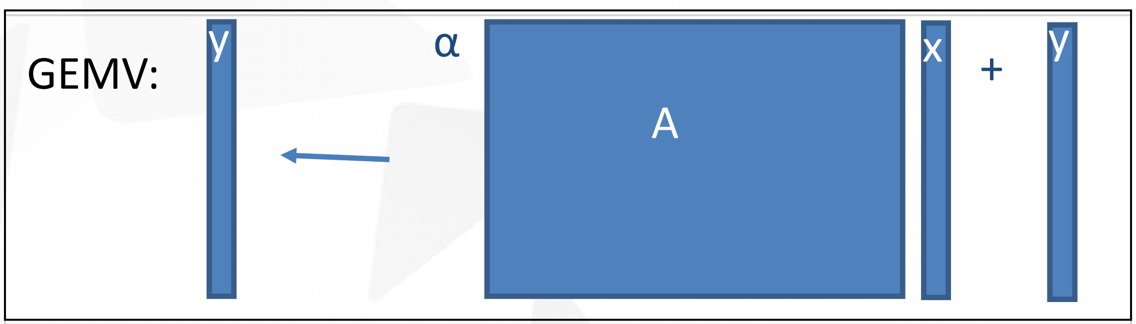


2n FLOP
2n memory reference

RATIO: 1

Level 2 BLAS Matrix-Vector operations

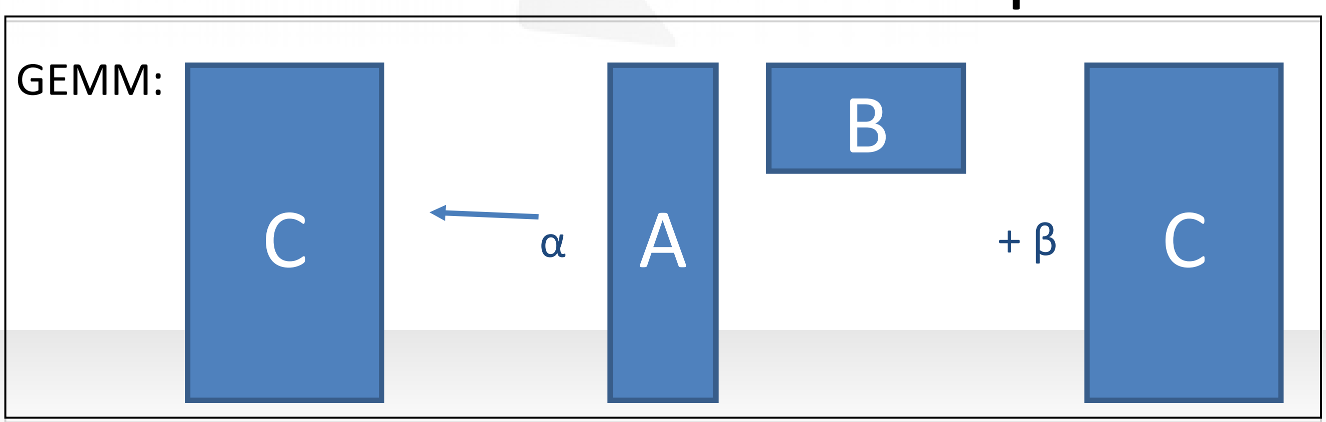
quick look



$2n^2$ 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, po rallel, and widely available, they are commor the development of high quality linear oftware.

- Performance of lot of applications a lot on the performance of the underlying BL

numpy library
for numerical
operations on
array, used for
satellite
computation,
time-constraint
problem, on
their cluster

600 seconds
numpy
implementation
was the
reference
implementation
of blas
reinstalled
numpy.
at the end it was
250 seconds
used OpenBLAS
install OpenBlas
most efficeint
library for this
architecture

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'.
```

man pages
blas given by
netlib use
openBLAS

Vendor/Optimized BLAS libraries

- ACML
  - The AMD Core Math Library, supporting the AMD processors
- ATLAS

the start of the century: it tests he hardware and auto tunes itself performs well

  - Autom Linear Algebra Softwa ource implementation of BLAS A Fortran 77
- Intel MK

the best one has BLAS and LAPACK and a lot of things

  - The Intel M brary, supporting x86 32-bits ncludes optimizations for Intel Pentium, Core and Intel Xeon CPUs and Intel Xeon Phi; support for Linux, Windows and Mac OS X
- cuBLAS

for a large matrix matrix multiplication on the GPU

  - Optimize IDIA based GPU cards
- clBLAS
  - An OpenCL implementation of BLAS
- ESSL

power 9 machines with 4 indivia cards

use ESSL supports the BLAS libraries and more

  - IBM' g and Scientific Subr y, supporting the Pow ture under AIX and Linux
- GotoBl
  - Kazu BSD-licensed implementation of BLAS, tuned in particula ehalem/Atom, VIA Nanopr Opteron

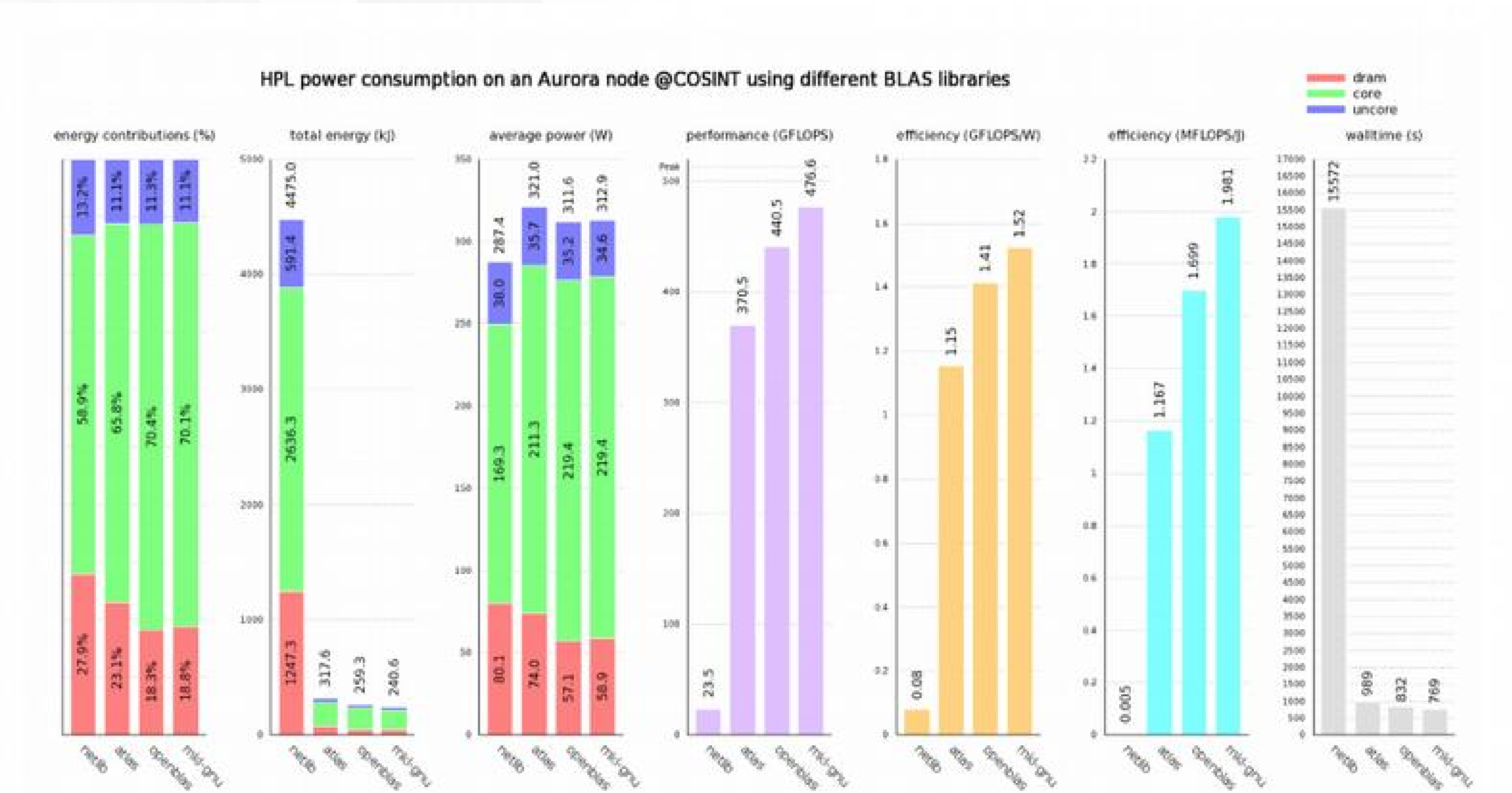
goto BLAS from a japanese guy at Texas supercomputer advanced center

Highly optimized verision of blas routines

find GOTO BALS faster than openBLAS

- BLIS
  - BLAS-lik antiation Software framew instantiation
- OpenBLAS
  - Optimiz ed on Goto BLAS hosted a pporting Intel platform and other

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



standard implementations
the hicher the best
mkl is the best

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.

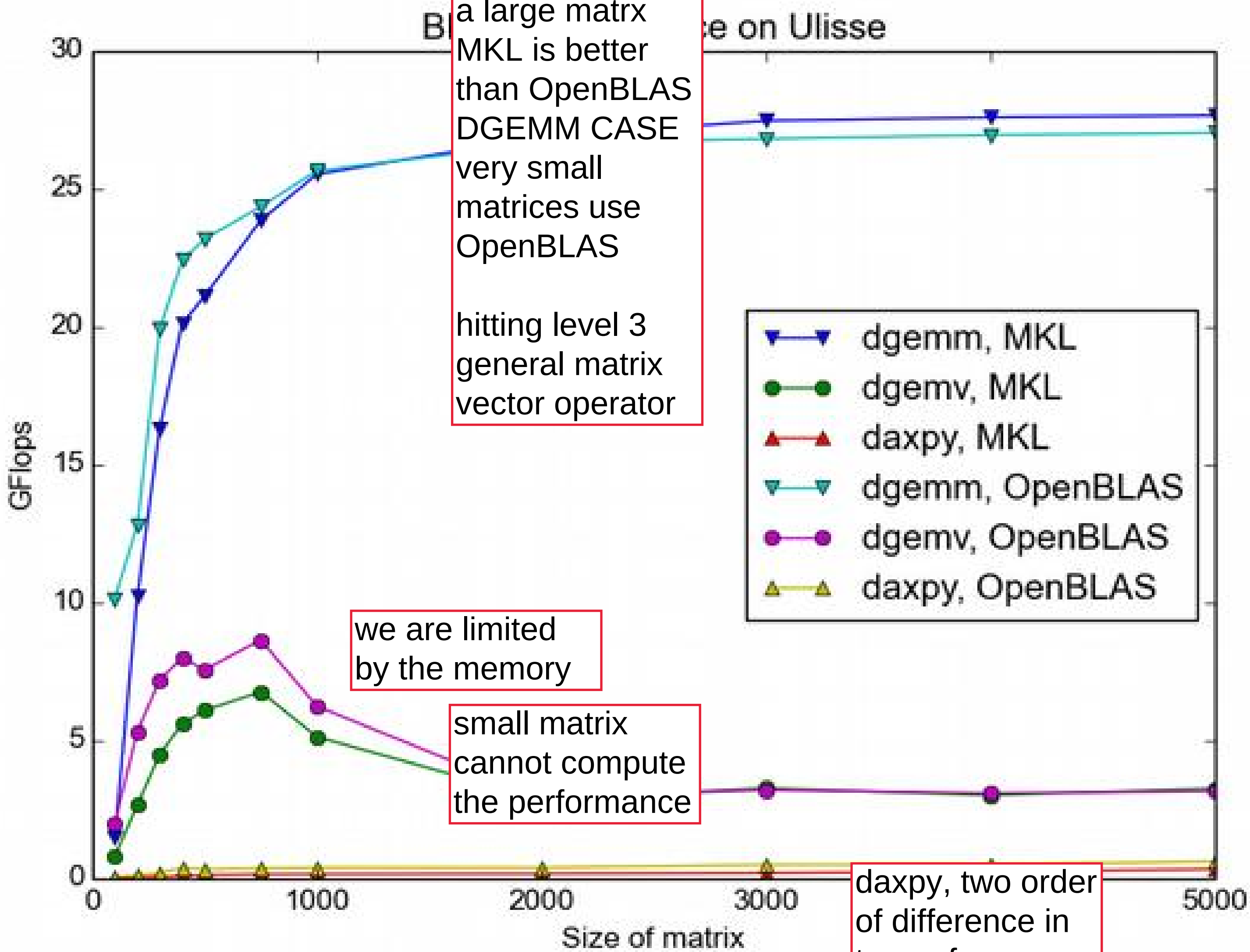
It follows..

- BLAS1 are memory bounded ! (for each computation a memory transfer is required)

im bounded to memory too many memory access

- BLAS2 are not so memo d (can have good performance on super-scalar architecture)
- BLAS3 can be very efficient on super-scalar computers because not memory bounded

Blas Performance o E



peak performance of the core
look at the difference, with a large matrx MKL is better than OpenBLAS
DGEMM CASE
very small matrices use OpenBLAS

hitting level 3 general matrix vector operator

we are limited by the memory

small matrix cannot compute the performance

daxpy, two order of difference in term of performance

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 ?

need an input file
if you use mkl library it is
multithreaded
if parallel aplicaion uses
multithreaded library
chech number of each thread

mpirun -np 24 hpl_command

hpl has been linked against hpl
we want to use HPL in a
multithreaded approach check how
many threads i have, i have 24
threads for each processor24 mpi
processes 24 threads spawn by
each

control by exporting the numbers of
threads that i want fix OMP_NUM
THREAD

it is best to have 24 MPI and 1
thread

or 1 mpi process and 24 thread

TWO COMBINATIon
on HPL or a standard DGEM
routine

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
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=Ri
1            # of recursive stopping crite
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=Ri
1            # of broadcast
0 2          BCASTs (0=1rg,1=1rM,2=2rg,3=2                 nM)
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)
```

number of problem sizes

block size

number of processors we have

this the input files 4 bloocksize 2 processor grid 8 cpu 8 cores processes run 32 experiments

just 1 instance of everything

mpirun

Parameters for HPL.dat input file

very large matrix is spawned in

6 times 4 and 4 times 6

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.

6 times 4 instead of 4 times 6
THE REcommendations

- 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 ...