Foundations of High Performance Computing

Using High Performance Libraries

Agenda

- Intro:
  - What are we using HPC for ?
  - Where should we start optimizing ?
- High Performance Libraries
- Linear Algebra libraries
- Using HP libraries: some examples.

The seven dwarfs of 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
  - solve Ax=B or Ax=lambdax where A is a sparse matrix (mostly zero)
- Operation on structured Grids:
  - 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:
  - similar but list of neighbours varies from entry to entry
- Spectral Methods
  - Fast Fourier Transform (FFT)
- Particle Methods
  - Compute electrostatic forces on n-particles
- Monte Carlo
  - many independent simulation using different inputs

Is this the real picture in 2022 ?

We are missing ALL data ML/DL working load..

But…

A lot of them relies on highly optimized libraries.

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 High Performance Libraries ?

- Routines for common (math) functions written in a specific way to take advantage of all 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 using High 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
- Parallelism (at least on single node) comes for free..

Any other reason apart from performance ?

- Usage of libraries makes 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 dense Linear Algebra ?

- Many large matrices are sparse, but …
- Dense algorithms easier to understand
- Some applications yields large dense matrices
- Large sparse matrix algorithms often yield smaller (but still large) dense problems
- LINPACK Benchmark (www.top500.org)

“ How fast is your computer?”
=
“How fast can you solve dense Ax=b?”

BLAS

Basic Linear Algebra Subprograms

BLAS summary

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	$C_{ij} = \sum_k A_{ik} B_{kj}$

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		
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

Where can I get BLAS?

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_11_0
- Ex: 3 nested loops for GEMM

Why BLAS are 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
- Lot of applications include ML/DL stuff as well….
  - https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/

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 , 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; suppor for Linux, Windows and Mac OS X
- cuBLAS
  - Optimized BLAS for NVIDIA based GPU cards
- 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, VIA Nanoprocessor, AMD Opteron
- OpenBLAS
  - Optimized BLAS based on Goto BLAS hosted at GitHub, supporting Intel platform and other
- And many others…

What about 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

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.
- Details and examples here:

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

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	$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj} + C_{ij}$	$2n^3$	$4n^2$	$n/2$

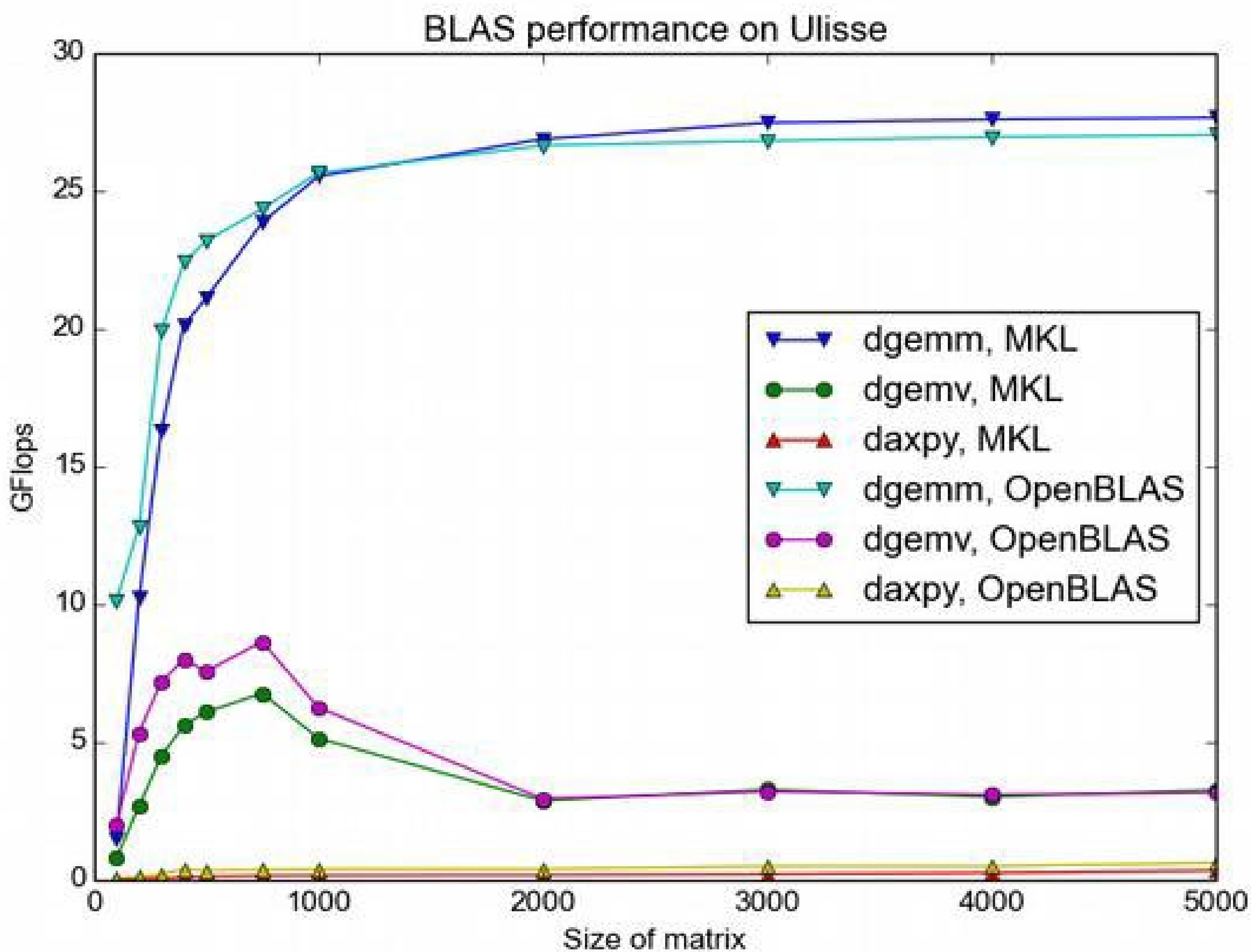
The parameter q is the ratio of flops to memory references.
Generally:

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 SandyBridge (1core)



Proposed exercise/tutorial

- Create the same graph for ORFEO cores using MKL and OpenBLAS
- STEPS:
  - Install OpenBLAS libraries in your directory
  - Write a small program to call the three routines
    - dgemm/dgemv/daxypi
  - Write a script to collect all sizes of interest
  - Make nice plots

Linking optimized libraries…

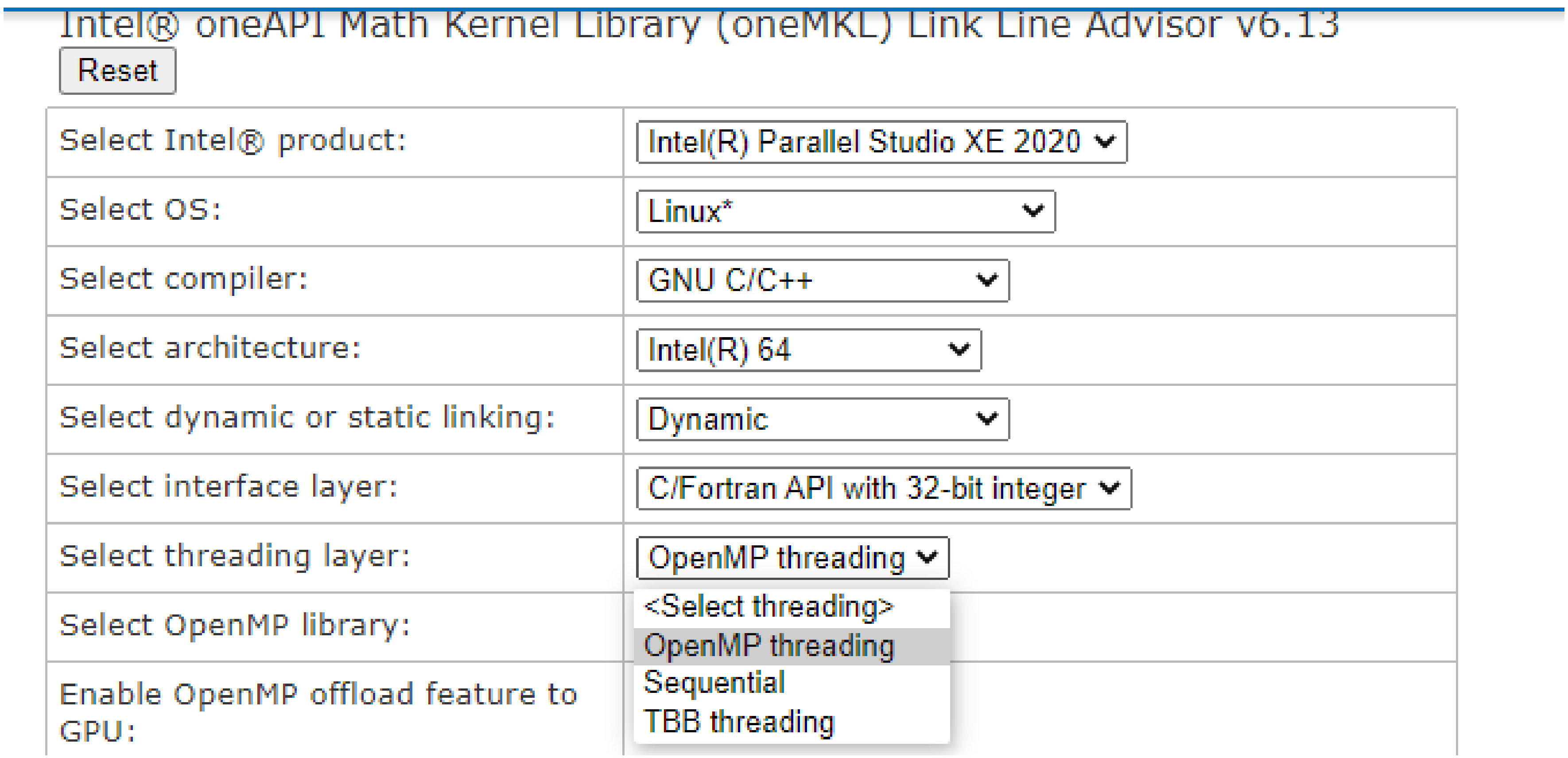
- OpenBLAS:
  - In comes with cblas bundled in so no problem with C/C++
  - Automatically includes lapack reference implementation
  - Compilation is straightforward:

```
gcc -o test test.c -I
/your_path/OpenBLAS/include/
   -L/your_path/OpenBLAS/lib -lopenblas
```

- 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

Are this libraries multithreaded ?

- MKL
  - Both sequential and multithreaded version available



MKL: how to control number of threads?

- OpenMP threading ? → OMP_NUM_THREADS
- Other threading ? → MKL_NUM_THREDS
- Define yourself the number of threads:
  - Place mkl_set_num_thread(N) routine in your code.
- All MKL routines call takes precedence over any environment variables. and MKL environment Variables will take precedence over the OpenMP* environments.

More details here:

https://software.intel.com/content/www/us/en/develop/articles/recommended-settings-for-calling-intel- mkl-routines-from-multi-threaded-applications.html

OpenBLAS:

- By default : multithreaded version, maximum number of threads established by cores available on the machine when compilation is performed;
- An example (ORFEO BLAS):

```
OpenBLAS build complete. (BLAS CBLAS LAPACK LAPACKE)

  OS               ... Linux
  Architecture     ... x86_64
  BINARY           ... 64bit
  C compiler       ... GCC  (cmd & version : cc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-39))
  Fortran compiler ... GFORTRAN  (cmd & version : GNU Fortran (GCC) 9.3.0)
  Library Name     ... libopenblas_haswellp-r0.3.13.a (Multi-threading; Max num-threads is 48)

To install the library, you can run "make PREFIX=/path/to/your/installation install".
```

OpenBLAS: caveat

- If your application is already multi-threaded, it will conflict with OpenBLAS multi-threading. Thus, you must set OpenBLAS to use single thread as following.
  - export OPENBLAS_NUM_THREADS=1 in the environment variables.
  - call openblas_set_num_threads(1) in the application on runtime.

- You can compile the library itself in sequential mode:
  - make USE_THREAD=0 USE_LOCKING=1*(see comment below)
  - If your application is parallelized by OpenMP, please build OpenBLAS with USE_OPENMP=1

Next lectures

- Monday 5:
  - Tutorial on STREAM
- Tuesday 6:
  - Tutorial on HPL
- Wednesday 7:
  - Tutorial on IOR/iozone

Ruggero Lot with the help of Niccolo’ Tosato.