Mathematical Software Programming (02635)

Module 9 — Fall 2016

Instructor: Martin S. Andersen

Announcement

Assignment 2

- Posted on CampusNet
- Due on November 23, 2016 (hand-in through CampusNet)
- Please post your questions on Piazza

Learning objectives

- call external (third party) programs and libraries (*)
- design, implement, and document a program that solves a mathematical problem
- debug and **test** mathematical software

(*) Only assessed through this assignment (and **not** through final exam)

This week

Topics

- External libraries and basic linear algebra

Learning objectives

- Call external (third party) programs and libraries

Basic Linear Algebra Subroutines (BLAS)

- building blocks for linear algebra (*de facto* standard)
- started as a FORTRAN library (late 1970s)
- linear algebra *engine* in MATLAB, Python, R, Mathematica, ...
- high performance when optimized for a specific system

BLAS

BLAS level 1 routines (1970s)

- vector operations, e.g.,

$$x^T y, \quad \|x\|_2, \quad x \leftarrow \alpha x, \quad y \leftarrow \alpha x + y$$

- use $O(n)$ operations for vectors of length n

BLAS level 2 routines (1980s)

- matrix-vector operations, e.g.,

$$y \leftarrow \alpha Ax + \beta y, \quad A \leftarrow \alpha xx^T + A, \quad x \leftarrow T^{-1}b, \ T \text{ triangular}$$

- use $O(mn)$ operations for matrices of size $m \times n$

BLAS

BLAS level 3 routines (1980s)

- matrix-matrix routines, e.g.,

$$C \leftarrow \alpha\, AB + \beta\, C, \quad X \leftarrow T^{-1}B,\ \ T \text{ triangular}$$

- use $O(n^3)$ operations for matrices of size $n \times n$

What's in a name?

BLAS naming scheme

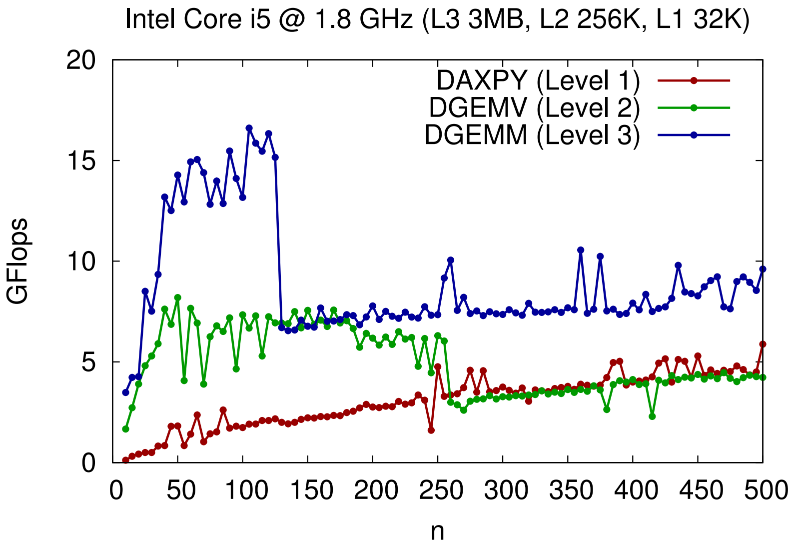
`XYYZZ`

- First character `X` indicates data type (`S`, `D`, `C`, `Z`)
- BLAS level 1: letters `YYZZ` indicate mathematical operation
- BLAS level 2+3: letters `YY` indicate matrix type
- BLAS level 2+3: letters `ZZ` indicate mathematical operation

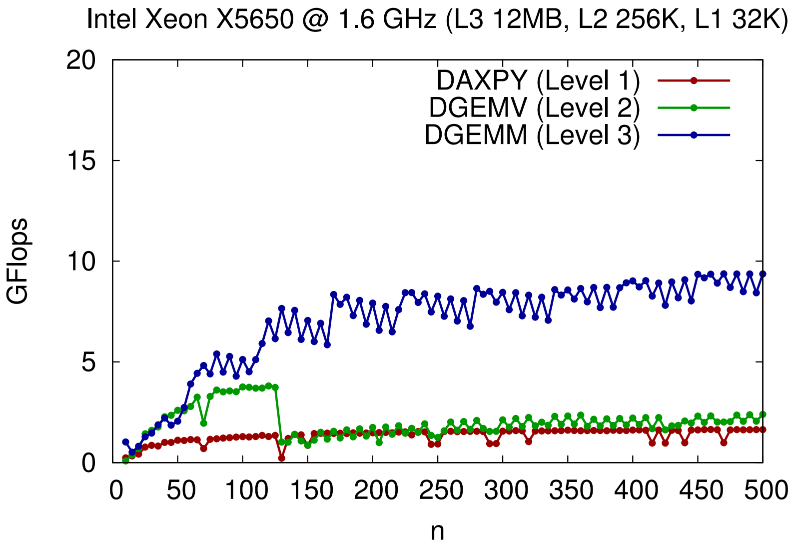
Examples

- `dscal` — *double scale* ($x \leftarrow \alpha x$)
- `saxpy` — *single a x plus y* ($y \leftarrow \alpha x + y$)
- `dgemv` — *double general matrix-vector* ($y \leftarrow \alpha Ax + \beta y$)
- `dtrsv` — *double triangular solve vector* ($x \leftarrow T^{-1}x$)
- `ssymm` — *single symmetric matrix-matrix* ($C \leftarrow \alpha SB + \beta C$)

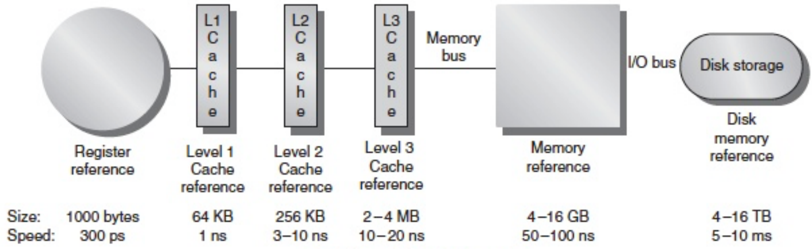
BLAS performance



BLAS performance



Memory hierarchy



Source: `http://goo.gl/yGtSDI`

Possible bottlenecks

- CPU speed is a bottleneck — *CPU bound*
- cache size/speed is a bottleneck — *cache bound*
- memory size/speed is a bottleneck — *memory bound*
- disk or network speed is a bottleneck — *I/O bound*

BLAS memory model

- vectors and matrices are contiguous arrays
- matrices are stored in column-major ordering
- *stride* refers to distance between *consecutive* elements
- *leading dimension* (LDA) refers to distance between columns

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} \\ A_{21} & A_{22} & A_{23} & A_{24} & A_{25} \\ A_{31} & A_{32} & A_{33} & A_{34} & A_{35} \\ A_{41} & A_{42} & A_{43} & A_{44} & A_{45} \end{bmatrix}, \qquad \begin{bmatrix} * & * & * & * & * \\ * & * & A_{23} & A_{24} & A_{25} \\ * & * & A_{33} & A_{34} & A_{35} \\ * & * & * & * & * \end{bmatrix}$$

- ith column of A is a vector of length 4 with stride 1
- ith row of A is a vector of length 5 with stride 4
- $(A_{11}, A_{22}, A_{33}, A_{44})$ is a vector of length 4 with stride 5
- A is a matrix with 4 rows, 5 columns, stride 1, LDA 4
- *slice* (submatrix to the right) has 2 rows, 3 columns, stride 1, LDA 4

Example: BLAS level 1

```c
/* Prototype for BLAS dscal */
void dscal_(
   const int * n,           /* length of array     */
   const double * a,        /* scalar a            */
   double * x,              /* array x             */
   const int * incx         /* array x, stride     */
);

int main(void) {
    int i,incx,n;
    double a, x[5] = {2.0,2.0,2.0,2.0,2.0};

    /* Scale the vector x by 3.0 */
    n = 5; a = 3.0; incx = 1;
    dscal_(&n, &a, x, &incx);

    return 0;
}
```

Makefile (DTU Unix system)

```
CC=gcc
SRCS=example_blas.c
EXECUTABLE=example_blas
INCLUDES=
CFLAGS=-c -Wall
LFLAGS=-L/usr/lib64/atlas   # directory that contains library
LIBS=-lf77blas              # name of BLAS library
OBJS=$(SRCS:.c=.o)

all: $(SRCS) $(EXECUTABLE)

$(EXECUTABLE): $(OBJS)
	$(CC) $(OBJS) $(LFLAGS) $(LIBS) -o $@

%.o: %.c
	$(CC) -c $(CFLAGS) $(INCLUDES) $< -o $@

clean:
	rm *.o $(EXECUTABLE)
```

Example: CBLAS

```c
#include <stdio.h>
#if defined(__MACH__) && defined(__APPLE__)
#include <Accelerate/Accelerate.h>
#else
#include <cblas.h>
#endif

int main(void) {

   int i,incx,n;
   double a, x[5] = {2.0,2.0,2.0,2.0,2.0};

   /* Scale the vector x by 3.0 */
   n = 5; a = 3.0; incx = 1;
   cblas_dscal(n, a, x, incx);

   return 0;
}
```

Makefile (Windows)

```
CC=gcc
SRCS=example_cblas.c
EXECUTABLE=example_cblas.exe
INCLUDES=-I.                # directory that contains cblas.h
CFLAGS=-c -Wall
LFLAGS=-L.                  # directory that contains libraries
LIBS=-lopenblas             # name of BLAS & CBLAS libraries
OBJS=$(SRCS:.c=.o)

all: $(SRCS) $(EXECUTABLE)

$(EXECUTABLE): $(OBJS)
    $(CC) $(OBJS) $(LFLAGS) $(LIBS) -o $@

%.o: %.c
    $(CC) -c $(CFLAGS) $(INCLUDES) $< -o $@

clean:
    rm *.o $(EXECUTABLE)
```