Intel® Math Kernel Library

Matrix Multiplication

Andrés Sicard-Ramírez

Ciclo de Conferencias
Centro de Computación Científica Apolo
Universidad EAFIT
2018-03-21

Motivation/Conclusion

Recall that if $A_{m\times p}$ and $B_{p\times n}$ are two matrices, their product is a matrix $(AB)_{m\times n}$.

Test: Matrix multiplication using different libraries.

Case: $m = 20000$, $p = 2000$ and $n = 10000$

Library	Time	Speed up
Naive (no library)	54.5 min	1X
GSL 2.1 (proper CBLAS)	17.3 min	3.1X
GSL 2.1 + CBLAS via ATLAS 3.10.2	2.5 min	21.8X
OpenBLAS 0.2.20	20.5 sec	159.5X
MKL 2018.1.163 (proper CBLAS)	16.3 sec	200.6X

Introduction

- Tools
  - Compilers
  - Debuggers
  - Profilers
  - ...
  - Libraries
    - Math Kernel Library
    - ...

Matrix Multiplication

Definition

Let $A = (a_{ij})_{m \times p}$ and $B = (b_{ij})_{p \times n}$ two matrices. Their product is matrix $C = (c_{ij})_{m \times n}$ where the ij-entry is given by

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}.$$

Matrix Multiplication

Running example

The matrices A are B have double-precision floating point values and them have the following shape:*

$$A = \begin{pmatrix} 1.0 & 2.0 & \dots & 1000.0 \\ 1001.0 & 1002.0 & \dots & 2000.0 \\ 2001.0 & 2002.0 & \dots & 3000.0 \\ \vdots & \vdots & \ddots & \vdots \\ 990001.0 & 990002.0 & \dots & 1000000.0 \end{pmatrix},$$

$$B = \begin{pmatrix} -1.0 & -2.0 & \dots & -1000.0 \\ -1001.0 & -1002.0 & \dots & -2000.0 \\ -2001.0 & -2002.0 & \dots & -3000.0 \\ \vdots & \vdots & \ddots & \vdots \\ -990001.0 & -990002.0 & \dots & -1000000.0 \end{pmatrix}.$$

*Example from https://software.intel.com/en-us/mkl/documentation/code-samples .

Naive Solution

Source code

```
#define m 2000
#define p 200
#define n 1000

double A[m][p], B[p][n], C[m][n];

int i, j, k;

for (i = 0; i < m; i++)
  for (j = 0; j < n; j++)
    for (k = 0; k < p; k++)
      C[i][j] += A[i][k] * B[k][j];
```

Naive Solution

Compiling, linking and running

```
$ gcc -Wall -c naive-mm.c
$ ./a.out
```

Remark

Always use `-Wall`.

BLAS (Basic Linear Algebra Subprograms)

Description

From
https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms:

> *"BLAS is a specification that prescribes a set of low-level routines for performing common linear algebra operations... They are the de facto standard low-level routines for linear algebra libraries; the routines have bindings for both C and Fortran ... using them can bring substantial performance benefits."*

BLAS

The dgemm function

Given compatible matrices A, B and C, and scalars α and β, the dgemm function calculates

$$C \leftarrow \alpha(AB) + \beta C.$$

ATLAS (Automatically Tuned Linear Algebra Software)

Description

From http://math-atlas.sourceforge.net/:

> *"The ATLAS project is an ongoing research effort focusing on applying empirical techniques in order to provide portable performance. At present, it provides C and Fortran77 interfaces to a portably efficient BLAS implementation,"*

GNU Advice

From
https://www.gnu.org/software/gsl/doc/html/usage.html#compiling-and-linking:

> *"The ATLAS package. . . should be installed for any work requiring fast vector and matrix operations."*

GSL (GNU Scientific Library) Solution

Source code

```
#include <gsl/gsl_blas.h>

int m, n, p;
double *A, *B, *C;

m = 2000, p = 200, n = 1000;

gsl_matrix_view AA = gsl_matrix_view_array (A, m, p);
gsl_matrix_view BB = gsl_matrix_view_array (B, p, n);
gsl_matrix_view CC = gsl_matrix_view_array (C, m, n);

gsl_blas_dgemm (CblasNoTrans, CblasNoTrans,
                alpha, &AA.matrix, &BB.matrix,
                beta, &CC.matrix);
```

GSL Solution

Compiling

```
$ gcc -Wall -c gsl-mm.c
```

Linking and running using the proper CBLAS

```
$ gcc gsl-mm.o -lgsl -lgslcblas -lm
$ ./a.out
```

Linking and running using CBLAS via ATLAS

```
$ gcc -Wall -c gsl-mm.c
$ gcc gsl-mm.o -lgsl -lcblas -latlas -lm
$ ./a.out
```

MKL (Math Kernel Library)

Some features

- Support Linux, macOs and Windows.
- C, C++, and Fortran development (native support). Python through Intel®.

MKL Solution

Source code

```
#include "mkl.h"

int m, n, p;
double *A, *B, *C;

m = 2000, p = 200, n = 1000;

cblas_dgemm (CblasRowMajor, CblasNoTrans, CblasNoTrans,
             m, n, p, alpha, A, p, B, n, beta, C, n);
```

Arguments for the `cblas_dgemm` function

See `readme.html` from MKL matrix multiplication example.

MKL Solution

Compiling, linking and running

```
$ export CPRO_PATH = ...
$ export MKLROOT = ${CPRO_PATH}/mkl
$ export OPTIONS_LINK = ...

$ gcc -I${MKLROOT}/include -Wall -c mkl-mm.c
$ gcc mkl-mm.o ${OPTIONS_LINK}
./a.out
```

Matrix Multiplication Using Different Libraries

Case 1: $m = 2000$, $p = 200$ and $n = 1000$

Library	Time	Speed up
Naive (no library)	1.46 sec	1X
GSL 2.1 (proper CBLAS)	1.06 sec	1.4X
GSL 2.1 + CBLAS via ATLAS 3.10.2	0.17 sec	8.6X
OpenBLAS 0.2.20	0.042 sec	34.8X
MKL 2018.1.163 (proper CBLAS)	0.039 seg	37.4X

Matrix Multiplication Using Different Libraries

Case 2: $m = 20000$, $p = 2000$ and $n = 10000$

Library	Time	Speed up
Naive (no library)	54.5 min	1X
GSL 2.1 (proper CBLAS)	17.3 min	3.1X
GSL 2.1 + CBLAS via ATLAS 3.10.2	2.5 min	21.8X
OpenBLAS 0.2.20	20.5 sec	159.5X
MKL 2018.1.163 (proper CBLAS)	16.3 sec	200.6X

Thanks!