BLAS and Vectorization extensions.

Carlos Pachajoa

December 5, 2012

Contents

BLAS

Stands for *Basic Linear Algebra Subprograms*

Is an **interface** for linear algebra operations. BLAS itself is a specification for Fortran. The equivalent interface in `C` is CBLAS.

Use the local implementation with

```
#include <cblas.h>
```

BLAS levels

The operations are divided in three levels:

- Level 1: Vector operations like
$$\mathbf{y} \leftarrow \alpha\mathbf{x} + \mathbf{y},\ \mathbf{x}, \mathbf{y} \in \mathbb{Z}^N$$
and also dot products and vector norms.
- Level 2: Matrix-vector operations like
$$\mathbf{y} \leftarrow \alpha\mathbf{A}\mathbf{x} + \beta\mathbf{y},\ \mathbf{x}, \mathbf{y} \in \mathbb{Z}^N,\ \mathbf{A} \in \mathbb{Z}^{M \times N}$$
and solutions for triangular systems, among others.
- Level 3: Matrix-matrix operations like
$$\mathbf{C} \leftarrow \alpha\mathbf{A}\mathbf{B} + \beta\mathbf{C},\ \mathbf{C} \in \mathbb{Z}^{M \times N},\ \mathbf{A} \in \mathbb{Z}^{M \times P},\ \mathbf{B} \in \mathbb{Z}^{P \times N}$$

Different calls for different precisions and whether real or complex numbers.

BLAS function naming conventions

The first letter specifies precision:

- **S** for real, single precision.
- **D** for real, double precision.
- **C** for complex, single precision.
- **Z** for complex, double precision.

The first letter is followed by a function, for example `xAXPY` is $y \leftarrow \alpha x + y$ from level one. Here, x represents a space and a precision.
Therefore, `SAXPY` will perform the operation using single precision floating point numbers.

CBLAS data representation

CBLAS receives data as contiguous positions in memory and a size. Both matrices and vectors are stored in this way. To specify a matrix, one additionally has to provide a stride and define whether it is row- or column- major.

$\{1,2,3,4,5,6,7,8,9\}$ will be

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \text{ in R.M. and } \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix} \text{ in C.M.}$$

The ordering is given using this enumeration:

```
enum CBLAS_ORDER {CblasRowMajor=101, CblasColMajor=102};
```

A function signature

$$\mathbf{y} \leftarrow \alpha \mathbf{A}\mathbf{x} + \beta \mathbf{y}, \quad \mathbf{y} \leftarrow \alpha \mathbf{A}^{\mathbf{T}}\mathbf{x} + \beta \mathbf{y}$$

```
void cblas_sgemv (
    const enum CBLAS_ORDER Order,
    const enum CBLAS_TRANSPOSE TransA,
    const int M,
    const int N,
    const float alpha,
    const float *A,
    const int lda,
    const float *X,
    const int incX,
    const float beta,
    float *Y,
    const int incY
);
```

Enumeration types

```
enum CBLAS_ORDER {
     CblasRowMajor=101,   /* row-major arrays */
     CblasColMajor=102};    /* column-major arrays */
enum CBLAS_TRANSPOSE {  // Whether to work with transpose
     CblasNoTrans=111,      /* trans='N' */
     CblasTrans=112,        /* trans='T' */
     CblasConjTrans=113};   /* trans='C' */
enum CBLAS_UPLO {       // The matrix is upper or lower
     CblasUpper=121,        /* uplo ='U' */
     CblasLower=122};       /* uplo ='L' */
enum CBLAS_DIAG {       // The matrix is unitriangular
     CblasNonUnit=131,      /* diag ='N' */
     CblasUnit=132};        /* diag ='U' */
enum CBLAS_SIDE {       // Order of matrix multiplication
     CblasLeft=141,         /* side ='L' */
     CblasRight=142};       /* side ='R' */
```

Some CBLAS implementations

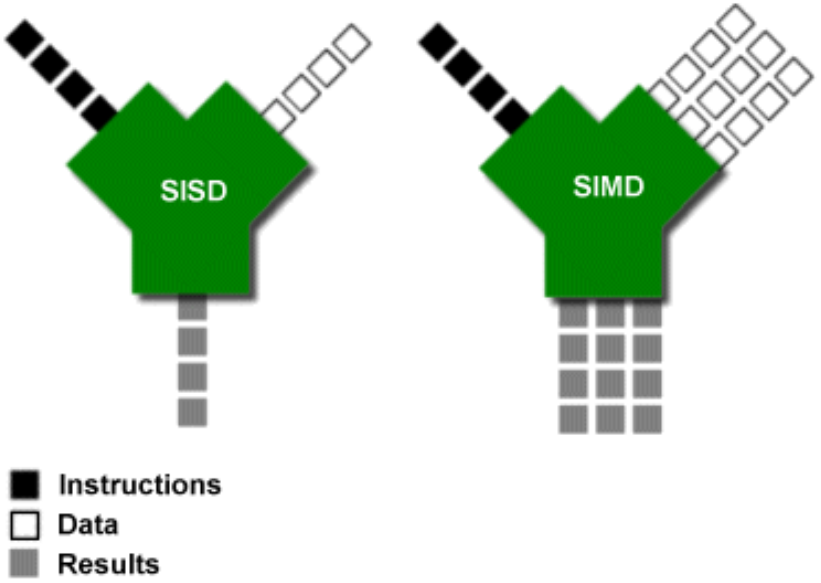
- ATLAS (Automatically Tuned Linear Algebra Software)
- MKL (Math Kernel Library)
- CUBLAS

Documents with CBLAS routines

- `http://math-atlas.sourceforge.net/psdoc/cblasqref.ps`
- `https://developer.apple.com/library/mac/documentation/Accelerate/Reference/BLAS_Ref/Reference/reference.html`

SIMD

Single Instruction, Multiple Data



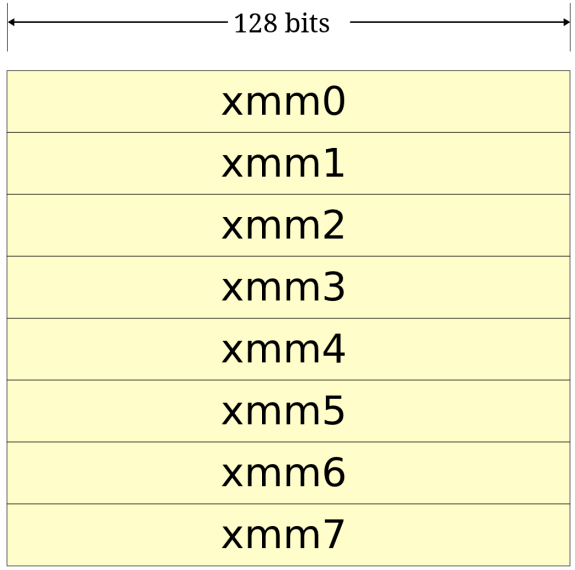
Taken from

`http://archive.arstechnica.com/cpu/1q00/simd/figure6.gif`

SSE

Streaming SIMD Extensions.

Additional registers in the processor and operations in the architecture.



`http://en.wikipedia.org/wiki/File:XMM_registers.svg`

8 registers with 128 bits each. 4 single precision floating point numbers in each register.

Some instructions

```
; All instructions end up in S
; penultimate letter denotes scalar or vector
; S stands for scalar, P for vector.
; Operands are XMM registers

; Adds all elements of arrays op1 and op2 into op1
ADDPS op1, op2

; Adds the first element of op1 and op2 into
; the first position of op2
ADDSS op1, op2
```

Some SSE instructions

```
vec_res.x = v1.x + v2.x;
vec_res.y = v1.y + v2.y;
vec_res.z = v1.z + v2.z;
vec_res.w = v1.w + v2.w;
```

```
;xmm0 = v1.w | v1.z | v1.y | v1.x
movaps xmm0, [v1]

;xmm0 = v1.w+v2.w | v1.z+v2.z | v1.y+v2.y | v1.x+v2.x
addps xmm0, [v2]

movaps [vec_res], xmm0
```

http://en.wikipedia.org/wiki/Streaming_SIMD_Extensions

SSE upgrades

SSE2 Allows to represent multiple types of data fitting on the vectors, including integers and characters, and perform the corresponding operations.
AMD's implementation also doubled the number of XMM arrays.

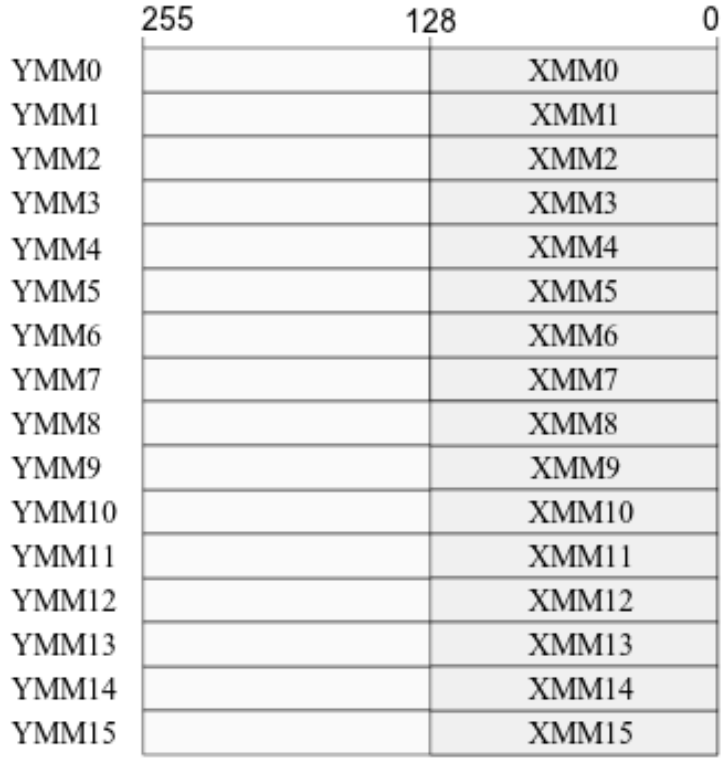
SSE3 Addition of horizontal operations within the XMM arrays, such as data reduction.

SSSE3 Additional instructions for SSE3.

SSE4 Introduction of Dword multiply, allowing to multiply two pairs of 32-bit integers to produce 2 64-bit numbers. Vector dot products.

AVX

Intel's extension to SSE for the Sandy Bridge microarchitecture, introduced in 2011. Also available in AMD's Bulldozer.



`http://upload.wikimedia.org/wikipedia/commons/f/f5/AVX_registers.svg`

Automatic vectorization

The Intel compiler can, under certain conditions, vectorize loops in the code.

This can be activated by using the `-vec` option.

```
for(i=0; i<SIZE; i++)A[i] = B[i] + C[i]
```

Obstacles to vectorization

- **Non-contiguous memory access**

```
for(i=0; i < SIZE; i+=stride) A[i] = B[i] + C[i];
```

- **Data dependencies**

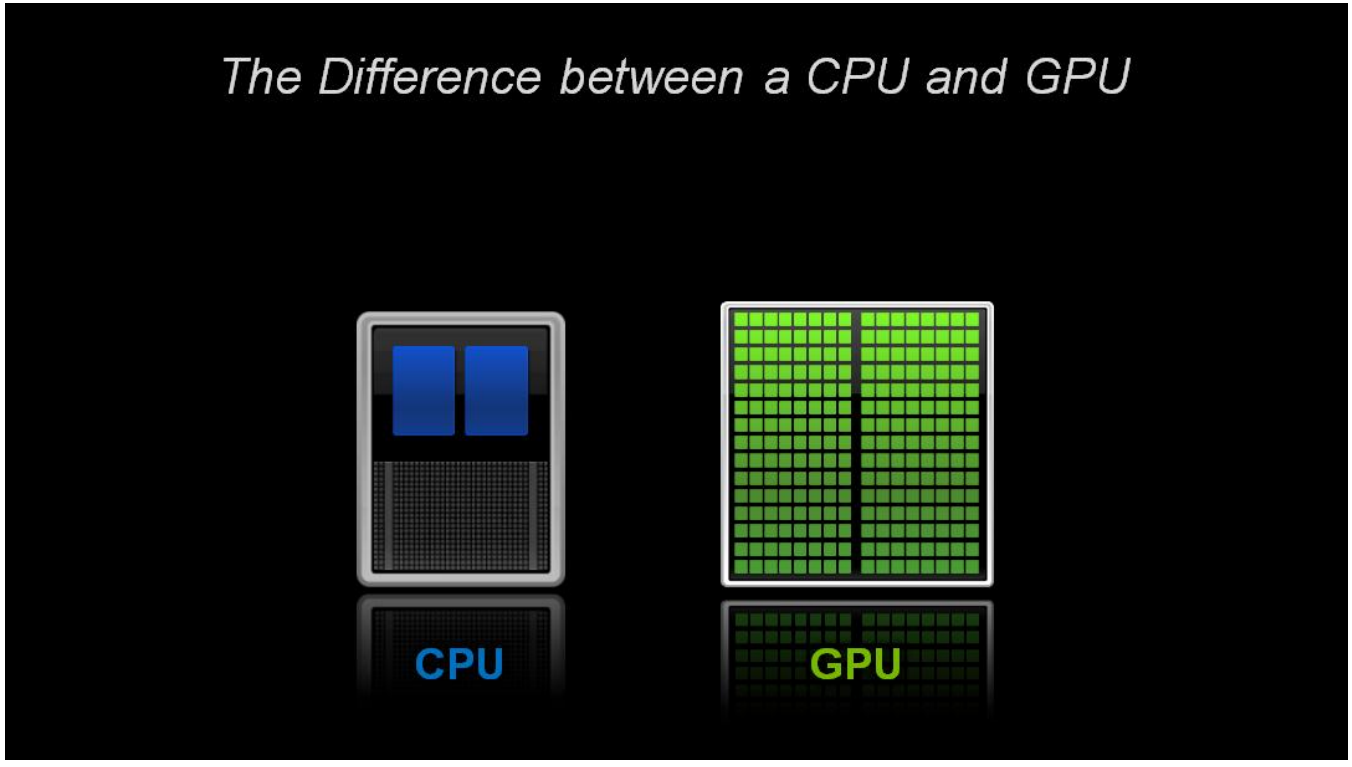
```
for(i=0; i < SIZE; i++) A[i] = A[i-1] + B[i];
```

Guiding ICC vectorization

- **Pragmas** For example, `#pragma ivdep`, among others to control when to vectorize a loop.
- **Keywords**, such as *restrict*.
- **Switches** passed to the compiler as optimization levels.

Look at the ICC automatic vectorization documentation[11].

GPGPU



http://blogs.nvidia.com/2009/12/whats-the-difference-between-a-cpu-and-a-gpu/

GPGPU and CPU

CPU

- General purpose
- Pipelines
- Few threads
- Lots of cache (Correlation)

GPGPU

- Specialized for local vector operations
- Many cores and threads
- Little cache
- Smaller consumption relative to a CPU

CUDA

Stands for *Compute Unified Device Architecture*.

Effectively, a programming model to access and control GPUs using a virtual instruction set, in a simmilar manner as a CPU.

Only supported on NVIDIA cards.

Uses the NVIDIA compiler, and can be programmed using *CUDA C/C++*, languages based on C/C++.

OpenCL

Stands for *Open Computing Language*

It also provides access to the GPU.

- It's an open standard, supported by NVIDIA and AMD, among others.
- Provides a language based on C99.
- Functionality provided by a driver.
- Compilation handled by linking to the correct library.

References


http://en.wikipedia.org/wiki/Streaming_SIMD_Extensions


http://www.netlib.org/blas/


http://www.stanford.edu/class/me200c/tutorial_77/18.1_blas.html


http://math-atlas.sourceforge.net/faq.html


http://software.intel.com/sites/products/documentation/hpc/mkl/mklman/GUID-2BCA8900-BD2F-4A15-9044-0AA23D07D0D2.htm


https://developer.nvidia.com/cublas


http://www.godevtool.com/TestbugHelp/XMMfpins.htm


software.intel.com/en-us/avx


http://www.khronos.org/opencl/


http://developer.download.nvidia.com/CUDA/training/GTC_Express_Sarah_Tariq_June2011.pdf


http://software.intel.com/sites/products/documentation/hpc/composerxe/en-us/2011Update/cpp/lin/optaps/common/optaps_vec_use.htm