Google Summer of Code 2013 Project Proposal

**SIMD Abstraction for Boost.uBLAS**

To be mentored by David Ballot of Boost

Atluri Aditya Avinash

May 1st 2013

1. **Introduction:**

 BLAS are used to perform basic linear algebra operations such as vector and matrix multiplication. BLAS have been extensively used in the field of High Performance Computing. Numerical linear algebra, particularly the solution of linear systems of equations, linear least squares problems, Eigen value problems and singular value problems, is fundamental to most calculations in scientific computing, and is often the computationally intense part of such calculations. The BLAS functionality is divided into 3 categories.

**Category 1:** This category contains vector operations of the form:

$$y\leftarrow α\*x+y$$

As well as scalar dot products and vector norms , among other things.

**Category 2:** This category contains matrix-vector operations of the form:

$$y\leftarrow αAx+βy$$

As well as solving $Tx=y$ for $x$ with $T$ being triangular, among other things.

**Category 3:** This category contains matrix-matrix operations of the form:

$$C\leftarrow αAB+βC$$

As well as solving $B\leftarrow αT^{-1}B$for triangular matrices$T$, among other things. This level contains the widely used General Matrix Multiply operation.

 Designers of computer programs involving linear algebraic operations have frequently chosen to implement low level operations, such as dot product or the matrix vector product, as separate subprograms. This may be observed both in many published codes and in codes written for specific applications.

1. **Existing Solutions:**
2. **ATLAS:**

Automatically Tuned Linear Algebra Software (ATLAS) is [software library](http://en.wikipedia.org/wiki/Library_%28computer_science%29) for [linear algebra](http://en.wikipedia.org/wiki/Linear_algebra). It provides a mature [open source](http://en.wikipedia.org/wiki/Open_source) implementation of [BLAS](http://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms) [APIs](http://en.wikipedia.org/wiki/Application_programming_interface) for [C](http://en.wikipedia.org/wiki/C_programming_language) and [Fortran77](http://en.wikipedia.org/wiki/Fortran). ATLAS is often recommended as a way to automatically generate an [optimized](http://en.wikipedia.org/wiki/Optimization_%28computer_science%29) BLAS library. While its performance often trails that of specialized libraries written for one specific [hardware platform](http://en.wikipedia.org/wiki/Platform_%28computing%29), it is often the first or even only optimized BLAS implementation available on new systems and is a large improvement over the generic BLAS available at [Netlib](http://en.wikipedia.org/wiki/Netlib). For this reason, ATLAS is sometimes used as a performance baseline for comparison with other products. ATLAS has good SIMD support.

1. **Intel MKL:**

The [Intel](http://en.wikipedia.org/wiki/Intel) [Math Kernel Library](http://en.wikipedia.org/wiki/Math_Kernel_Library), supporting the old Intel [Pentium](http://en.wikipedia.org/wiki/Pentium_%28brand%29) (although there are some doubts about future support to the Pentiumarchitecture), [Core](http://en.wikipedia.org/wiki/Intel_Core) and [Itanium](http://en.wikipedia.org/wiki/Itanium) CPUs under [Linux](http://en.wikipedia.org/wiki/Linux), [Windows](http://en.wikipedia.org/wiki/Microsoft_Windows) and [Mac OS X](http://en.wikipedia.org/wiki/Mac_OS_X).

1. **cuBLAS:**

Optimized BLAS for NVIDIA based GPU cards. The NVIDIA CUDA Basic Linear Algebra Subroutines (cuBLAS) library is a GPU-accelerated version of the complete standard BLAS library that delivers 6x to 17x faster performance than the latest MKL BLAS. Building on the GPU-accelerated BLAS routines in the cuBLAS library, heterogeneous LAPACK implementations such as CULA Tools and MAGMA are also available.

1. **Goals of SIMD Abstraction of Boost.uBLAS:**

 Our aim with SIMD (Single Instruction Multiple Data) Abstraction is to create a fast, modern, well designed BLAS library that can run on Intel, AMD and ARM processors and can use their native parallelism. In [computing](http://en.wikipedia.org/wiki/Computing), **Streaming SIMD Extensions** (**SSE**) is an [SIMD](http://en.wikipedia.org/wiki/SIMD) [instruction set](http://en.wikipedia.org/wiki/Instruction_set) extension to the [x86](http://en.wikipedia.org/wiki/X86) architecture. SSE contains 70 new instructions, most of which work on [single precision](http://en.wikipedia.org/wiki/Single_precision) [floating point](http://en.wikipedia.org/wiki/Floating_point) data. SSE originally added eight new 128-bit registers known as XMM0 through XMM7. SIMD instructions can greatly increase performance when exactly the same operations are to be performed on multiple data objects. Typical applications are [digital signal processing](http://en.wikipedia.org/wiki/Digital_signal_processing) and [graphics processing](http://en.wikipedia.org/wiki/Graphics_processing). SSE was subsequently expanded by Intel to [SSE2](http://en.wikipedia.org/wiki/SSE2), [SSE3](http://en.wikipedia.org/wiki/SSE3), [SSSE3](http://en.wikipedia.org/wiki/SSSE3), and [SSE4](http://en.wikipedia.org/wiki/SSE4). The processors contain inbuilt hardware that can operate on 128-bit data (256 for AVX).

**For AMD Processors:**

 Initially, the SIMD abstraction for AMD processors will be developed on AMD K10 (supports 3DNow!, SSE, SSE2, SSE3, SSE4a) and Bulldozer Architectures (supports 3DNow!, SSE, SSE2, SSE3, SSE4.1, SSE4.2, CULMUL, AVX).

**For Intel Processors:**

 The SIMD abstraction for Intel processors will be developed on Core 2 Duo [aka Penryn] (supports upto 4.1, AVX) and same generation architectures. Later extended to Nehalem and current generation architectures (supports upto SSE 4.2, AVX).

 **Examples of SIMD Abstraction:**

**SSE:** The below code shows the basic matrix addition using SSE deployed on Intel processors.

int lcount = loopCount;

 int iBufferHeight = MAT\_HEIGHT;

 for (int i = 0; i < lcount; i++){

 float\* pImage1 = InputBuffer1;

 float\* pImage2 = InputBuffer2;

 float\* pOutImage = OutputBuffer;

 for(int iY = 0; iY < iBufferHeight; iY++){

 \_\_m128 Xmm\_A = \_mm\_load\_ps(pImage1);

 \_\_m128 Xmm\_B = \_mm\_load\_ps(pImage2);

 \_\_m128 Xmm\_C = \_mm\_add\_ps (Xmm\_A, Xmm\_B);

 \_mm\_store\_ps(pOutImage, Xmm\_C);

 pOutImage+=4;

 pImage1+=4;

 pImage2+=4;

 }

 }

**AVX:** The below code shows the basic matrix addition using AVX deployed on Intel Processors.

int lcount = loopCount;

 int iBufferHeight = MAT\_HEIGHT;

 for (int i = 0; i < lcount; i++)

 {

 float\* pImage1 = InputBuffer1;

 float\* pImage2 = InputBuffer2;

 float\* pOutImage = OutputBuffer;

 for(int iY = 0; iY < iBufferHeight; iY+= 2)

 {

 \_\_m256 Ymm\_A = \_mm256\_load\_ps(pImage1);

 \_\_m256 Ymm\_B = \_mm256\_load\_ps(pImage2);

 \_\_m256 Ymm\_C = \_mm256\_add\_ps (Ymm\_A, Ymm\_B);

 \_mm256\_store\_ps(pOutImage, Ymm\_C);

 pOutImage+=8;

 pImage1+=8;

 pImage2+=8;

 }

 }

1. **Current State of Development:**

Currently. Boost.uBLAS do have a good SIMD extensions on CUDA and OpenCL which have a greater performance through put for massively parallel data sets and algorithms. But, uBLAS doesn’t have good SIMD extensions on Intel, AMD CPUs. These extensions can handle small data sets where there is a significant requirement of parallel and serial implementation.

1. **Goals of GSOC Project:**

The following are the goals of the GSOC project during the development of SIMD extensions (abstraction) for Boost.uBLAS.

* + - 1. Checking the uBLAS library and find the areas where there can be significant improvements in performance of the algorithms.
			2. Designing algorithms appropriately so that it can be implemented on SSE and AVX.
			3. Development of backend using SSE and AVX on current generation architectures.
			4. Development of frontend for the SIMD abstraction using Object Oriented Programming concepts in C++.
			5. Checking for errors and Error Handling of the frontend and backend.
			6. Extending the backend to SSE2, SSE3, SSE4.1, SSE4.2 using their newly added instruction sets.
			7. Adding the feature of CULMUL to AMD processors.
			8. Development of frontend for added SSEx backend to exploit their features.
			9. Testing of the library on different Intel and AMD architectures.
			10. Development of uBLAS on ARM Neon.

The point 10 is too concise. Its description is, development of uBLAS on ARM Neon which goes through the same process as in development of SIMD abstraction on Intel and AMD CPUs.

1. **Schedule:**

**1st Month:** Points 1 to 4.

**2nd Month:** Points 4 to 7.

**3rd Month:** Points 8 to 10.

1. **Background Information:**

**7.1 Personal Details:**

**Name:** Atluri Aditya Avinash

**University:** Indian School of Mines, Dhanbad

**Major:** Electrical Engineering

**Degree Program:** Bachelor of Technology

**Email:** adityaavinash143@gmail.com

**Availability:** 3-4 days a week + occasional weekends; starting from June to September. Will work when there is a top priority situation to deal with.

**7.2 Educational Background:**

 My major in college is Electrical Engineering. I have did a project for power generation firm Reliance Power Limited during summer of 2011 on MATLAB. I have coded the algorithms in my courses and improved them using programming concepts. I have secured 100% in both maths and science in my school. I passed out of my high school with aggregate of 90% having scored 95%+ in both science and math.

**7.3 Programming Background:**

 I program in C/C++, Python, GO, shell, MATLAB. I have experience in using OpenMP, Intel AVX and SSE, CUDA, OpenCL, Boost libraries. I have been programming since 2005. My first code is in html for online exams. I have experience in using Windows and Ubuntu(extensively). I am one of the Kernel Developer for Ubuntu. I have submitted few bug fixes and patches for Ubuntu but, the same patches were submitted by other developers. I have presented a paper at a national conference. The paper is related to the field of bio-informatics where an algorithm is accelerated using CUDA. I have worked on software projects with my friends. I have developed most the algorithms in Electrical Engineering to CUDA. “Think Parallel!!”, this is what I am good at. I write CUDA code for fun, knowing all the boundaries of performance bottlenecks. I participate in coding social networking sites like stackoverflow, some communities on #IRC and github.com.

**7.4 Self Assessment:**

**C++:** 4

**C++ STL:** 3

**Boost Libraries:** 3

 Familiar with odeint and uBLAS.

**SSE and AVX:** 3

**Linear Algebra:** 4