**SEVENTH FRAMEWORK PROGRAMME**
**Research Infrastructure**

**FP7-INFRASTRUCTURES-2010-2 – INFRA-2010-1.2.3:**
**Virtual Research Communities**

**Combination of Collaborative Project and Coordination and Support Actions (CP-CSA)**


**LinkSCEEM-2**
# Linking Scientific Computing in Europe and the Eastern Mediterranean – Phase 2

**Grant Agreement Number: RI-261600**

# D12.1
# Report on identification, installation and testing of libraries.

***Final***

Version: 0.5
Author(s): Dimitrios Karkoulis, IERS
Date: 30/08/2011

## Project and Deliverable Information Sheet

| | | |
|---|---|---|
| **LinkSCEEM Project** | **Project Ref. №: RI-261600** | |
| | **Project Title: LinkSCEEM-2** | |
| | **Project Web Site:** http://www.linksceem.eu | |
| | **Deliverable ID:** < **D12.1**> | |
| | **Deliverable Nature:** <DOC_TYPE: Report> | |
| | **Deliverable Level:** PU | **Contractual Date of Delivery:** 31 / 08 / 2011 |
| | | **Actual Date of Delivery:** 31 / 08 / 2011 |
| | **EC Project Officer: Leonardo Flores Anover** | |

* - The dissemination level are indicated as follows: **PU** – Public, **PP** – Restricted to other participants (including the Commission Services), **RE** – Restricted to a group specified by the consortium (including the Commission Services). **CO** – Confidential, only for members of the consortium (including the Commission Services).

## Document Control Sheet

| | | |
|---|---|---|
| **Document** | **Title: Report on identification, installation and testing of libraries** | |
| | **ID:** <**D12.1**> | |
| | **Version:** <0.5> | **Status:** Draft |
| | **Available at:** http://www.eniac.cyi.ac | |
| | **Software Tool:** Microsoft Word 2007 | |
| | **File(s):** LinkSCEEM-2-D12.1.docx | |
| **Authorship** | **Written by:** | Dimitrios Karkoulis, IERS |
| | **Contributors:** | WP Leaders and Participants |
| | **Reviewed by:** | A. Mirone, A. Goetz, R. Dimper, J. Kieffer, IERS<br>D. Saparilla, A. Strelchenko, CaSToRC,<br>S. Matalgah, SESAME |
| | **Approved by:** | PMO |

## Document Status Sheet

| Version | Date | Status | Comments |
|---|---|---|---|
| 0.1 | 12/07/2011 | Draft | First version |
| 0.2 | 12/07/2011 | Draft | Revision by D. Saparilla |
| 0.3 | 24/08/2011 | Draft | Revision by D. Saparilla and A. Strelchenko. Addition of the benchmark results. |
| 0.4 | 29/08/2011 | Draft | Revision by A. Strelchenko. |
| 0.5 | 30/08/2011 | Draft | Revision by J. Kieffer |

## Document Keywords

| | |
|---|---|
| **Keywords:** | LinkSCEEM-2, Computational Science, HPC, GPGPU, OpenCL, BLAS, FFT |

# Table of Contents



# References and Applicable Documents

[1] http://www.linksceem.eu
[2] http://www.prace-project.eu
[3] LinkSCEEM-2 Project Execution Plan

# List of Acronyms and Abbreviations

| | |
|---|---|
| ACF | Advanced Computing Facility |
| API | Application Programming Interface |
| CaSToRC | Computation-based Science and Technology Research Centre of the CyI |
| CPU | Central Processing Unit |
| CUDA | Compute Unified Device Architecture (NVIDIA) |
| CyI | The Cyprus Institute |
| CyNet | The Cyprus NREN |
| DEISA | Distributed European Infrastructure for Supercomputing Applications. EU project by leading national HPC centres. |
| EC | European Community |
| EoI | Expression of Interest |
| ESFRI | European Strategy Forum on Research Infrastructures; created roadmap for pan-European Research Infrastructure. |
| FP | Floating-Point |
| FPU | Floating-Point Unit |
| FZJ | Forschungszentrum Jülich (Germany) |
| GB | Giga (= $2^{30}$ ~ $10^9$) Bytes (= 8 bits), also GByte |
| Gb/s | Giga (= $10^9$) bits per second, also Gbit/s |
| GB/s | Giga (= $10^9$) Bytes (= 8 bits) per second, also GByte/s |
| GÉANT | Collaboration between National Research and Education Networks to build a multi-gigabit pan-European network, managed by DANTE. GÉANT2 is the follow-up as of 2004. |
| GFlop/s | Giga (= $10^9$) Floating point operations (usually in 64-bit, i.e. DP) per second, also GF/s |
| GHz | Giga (= $10^9$) Hertz, frequency =$10^9$ periods or clock cycles per second |
| GigE | Gigabit Ethernet, also GbE |
| GNU | GNU's not Unix, a free OS |
| GPGPU | General Purpose GPU |
| GPU | Graphic Processing Unit |
| HDD | Hard Disk Drive |
| HE | High Efficiency |
| HET | High Performance Computing in Europe Taskforce. Taskforce by representatives from European HPC community to shape the European HPC Research Infrastructure. Produced the scientific case and valuable groundwork for the PRACE project. |
| HPC | High Performance Computing; Computing at a high performance level at any given time; often used synonym with Supercomputing |
| HPCC | HPC Challenge benchmark, http://icl.cs.utk.edu/hpcc/ |
| HPL | High Performance LINPACK |
| HWA | HardWare accelerator |
| IB | InfiniBand |
| IBA | IB Architecture |
| IBM | Formerly known as International Business Machines |
| IEEE | Institute of Electrical and Electronic Engineers |
| I/O | Input/Output |
| ISC | International Supercomputing Conference; European equivalent to the US based SC0x conference. Held annually in Germany. |
| JSC | Jülich Supercomputing Centre (FZJ, Germany) |
| KB | Kilo (= $2^{10}$ ~$10^3$) Bytes (= 8 bits), also KByte |
| LQCD | Lattice QCD |
| LinkSCEEM | Linking Scientific Computing in Europe and the Eastern Mediterranean |
| LinkSCEEM-2 | Linking Scientific Computing in Europe and the Eastern Mediterranean – Phase 2 |
| LS | Local Store memory (in a Cell processor) |
| MB | Mega (= $2^{20}$ ~ $10^6$) Bytes (= 8 bits), also MByte |
| MB/s | Mega (= $10^6$) Bytes (= 8 bits) per second, also MByte/s |
| MFlop/s | Mega (= $10^6$) Floating point operations (usually in 64-bit, i.e. DP) per second, also MF/s |
| MHz | Mega (= $10^6$) Hertz, frequency =$10^6$ periods or clock cycles per second |

| | |
|---|---|
| MIPS | Originally Microprocessor without Interlocked Pipeline Stages; a RISC processor architecture developed by MIPS Technology |
| Mop/s | Mega (= $10^6$) operations per second (usually integer or logic operations) |
| MoU | Memorandum of Understanding. |
| MPI | Message Passing Interface |
| MPP | Massively Parallel Processing (or Processor) |
| NDA | Non-Disclosure Agreement. Typically signed between vendors and customers working together on products prior to their general availability or announcement. |
| NoC | Network-on-a-Chip |
| NFS | Network File System |
| NIC | Network Interface Controller |
| OpenCL | Open Computing Language |
| OpenGL | Open Graphic Library |
| Open MP | Open Multi-Processing |
| OS | Operating System |
| pNFS | Parallel Network File System |
| POSIX | Portable OS Interface for Unix |
| PRACE | Partnership for Advanced Computing in Europe; Project Acronym |
| PRACE-1P | Partnership for Advanced Computing in Europe – First Implementation Phase |
| PRACE | Partnership for Advanced Computing in Europe – Research Infrastructure |
| RAM | Random Access Memory |
| SDK | Software Development Kit |
| SSD | Solid State Disk or Drive |
| TB | Tera (= $2^{40}$ ~ $10^{12}$) Bytes (= 8 bits), also TByte |
| TCO | Total Cost of Ownership. Includes the costs (personnel, power, cooling, ...) in addition to the purchase cost of a system. |
| TFlop/s | Tera (= $10^{12}$) Floating-point operations (usually in 64-bit, i.e. DP) per second, also TF/s |
| Tier-0 | Denotes the apex of a conceptual pyramid of HPC systems. In this context the PRACE Supercomputing Research Infrastructure would host the Tier-0 systems; national or topical HPC centres would constitute Tier-1 |
| UNICORE | Uniform Interface to Computing Resources. Software for seamless access to distributed resources. |
| VO | Virtual Organization |
| VRC | Virtual Research Community |

# Executive Summary

Workpackage 12, "Synchrotron high performance data analysis and modelling", aims at decreasing the latency of data processing in synchrotron research that will allow for faster off-line data analysis and simulation. Additionally it will make their use on-line feasible despite the increasing rate of data acquisition. As a consequence, the quality and efficiency of experiments will be improved.

Deliverable D12.1 suggests that general purpose algorithms based on OpenCL are discovered, installed and tested. A common problem in software development, especially in the scientific sector, is spending time developing and maintaining algorithms when identical ones have already been developed, tested and made publicly available or are superior. Indeed, at least for CPU processors, there are abundant mathematical and imaging libraries available which can be used for specific tasks and save precious development time and resources.

This deliverable has also, indirectly, the purpose of evaluating the OpenCL API for GPGPU programming, which will be used as the basis for GPU development on the following deliverables. Even though OpenCL will be discussed in detail later on, it is important to highlight here some key elements of OpenCL:

- It is open source, without redistribution-licensing limitations and loyalty.
- It can take advantage of different kind of devices by various vendors, not only GPU and a specific vendor as is the case with propriety APIs.
- It is a very new API that does not yet enjoy the richness in features and 3rd party libraries that other APIs do.

We focused our research on two very common types of general purpose libraries used at synchrotron and the sciences. First, libraries that contain set of mathematical algorithms for Linear Algebra typically referred to as BLAS. Second, libraries that contain algorithms for Fast Fourier transforms, typically abbreviated as FFT.

Benchmarks are provided for each library as well as performance comparison tables.

To evaluate OpenCL we proceeded directly to work on the deliverable D12.4. As part of the development involved, we had the chance to evaluate the advantages and disadvantages of this API.

# 1 Basic Linear Algebra Subprograms

## 1.1 Summary

A Basic Linear Algebra Subprograms library is a defined set of functions for Linear Algebra operations, organised in levels depending on the kind of the operation:

- Level 1, vector – vector
- Level 2, matrix – vector
- Level 3, matrix – matrix

Each operation has a standard name regardless the library. I.e. a matrix – matrix multiplication with double precision is part of Level 3 BLAS with name DGEMM. This operation will be called DGEMM on all popular BLAS libraries.

## 1.2 Identified libraries

In contrast to CPU BLAS implementations, we were able to identify only a few libraries for OpenCL:

- AMD Accelerated Parallel Processing Math Libraries (APPML)
- ViennaCL
- OpenCL BLAS

APPML is a propriety package of BLAS and FFT libraries by AMD that are based on OpenCL. The same company also offers two more such packages, one for parallel processing on CPU and one for GPU processing, but based on its own propriety GPGPU API, Stream. The latest version for Linux is 1.4.182 for Linux. However, this version was released after the benchmark results for the previous version were acquired. As such we will present results for versions 1.2.144 and 1.4.18.

ViennaCL is an open source BLAS library developed at the Institute for Microelectronics, TU Wien. The latest version is 1.1.2 for Windows and Linux alike.

OpenCL BLAS is an open source project which has been inactive for two year and never made it past the early development stage. It does not offer any documentation either. For these reasons it has been discarded from the list of candidate libraries for testing.

Installation and testing were performed on a HPC node with two Intel XEON E5640 (2.67Ghz) CPUs and two NVIDIA M2050 GPUs with ECC enabled. The node is running the 64-bit Linux distribution CentOS 5 with OpenCL APIs by NVIDIA (Toolkit version 4.0) and AMD (APP version 2.4) installed. The system setup is presented in detail in the Benchmark section.

## 1.3 AMD Accelerated Parallel Processing Math Libraries (APPML) – BLAS

### 1.3.1 Features

APPML BLAS 1.2.144 does not provide Level 1 and Level 2 functionality, but is limited to some matrix – matrix operations of Level 3. In specific only *GEMM, *TRMM and *TRSM are available, where the asterisk denotes the precision (float or double) and the space (real or complex). *GEMM is a function for matrix – matrix multiplication, *TRMM is for the triangular matrix – general matrix product and *TRSM is the solver to a non-singular triangular system of equations.

APPML BLAS 1.4.182 provides extended functionality in contrast to the older version as well as performance optimisation. Some Level 2 functionality is added, the *GEMV and *SYMV matrix-vector multiplication functions. Level 3 is extended with the addition of *SYRK and *SYR2K rank update for symmetric matrices functions. *GEMV, *SYMV, *SYRK and *SYR2K do not support complex numbers.

Since APPML is based on OpenCL it can be executed on CPU for multi-core implementation or debugging and on GPU. APPML officially supports devices that are supported by AMD's OpenCL API. However, APPML may be used with other devices as long as the corresponding OpenCL driver is installed.

### 1.3.2 Installation

Since APPML depends on APP SDK 2.4, the latter must first be deployed.

**APP SDK 2.4 installation**

APP SDK enables OpenCL support for devices manufactured by AMD. The corresponding APP SDK files for our system (*AMD-APP-SDK-v2.4-lnx64.tgz*) can be found at *http://developer.amd.com*. The Linux files for APP SDK are merely compressed archives and upon their extraction some additional manual steps are required to complete the set-up, most notably the "icd registration" to ensure that the devices will be visible and accessible.

**APPML installation**

APPML BLAS 1.2.144 is not available by AMD, due to the release of the new version. However, the library can be provided upon request. The file corresponding to the BLAS libraries for our system is "*clAmdBlas-1.2.144-Linux.tar.gz*". Extraction of that archive reveals an installation script "*install-clAmdBlas-1.2.144.sh*" which we need to execute and then refresh our library system paths to ensure APPML BLAS is reachable.

APPML BLAS 1.4.182 is available freely at *http://developer.amd.com*. The installation process is similar to the previous version.

### 1.3.3 Testing

Prior to performing any tests for an OpenCL based library, the OpenCL installation needs to be checked. The APP SDK provides a simple executable that displays the available OpenCL devices called clInfo. If OpenCL is installed and the environment set-up properly we should get a list of the supported devices present on the machine.

The next step is to check the APPML installation itself. APPML BLAS provides example executables, one for each of the BLAS operations available that can be used to test if the library is working correctly.

We executed all the available examples (*example_sgemm, example_trsm, example_trrm*) and they all completed successfully on the CPU (AMD OpenCL), but with some minor annoyances on the GPU (NVIDIA OpenCL). NVIDIA GPUs are not officially supported by APPML however and it is sufficient to clarify that the problems were caused by few minor incompatibilities between the OpenCL 1.0 specification used by NVIDIA and OpenCL 1.1 used by AMD.

The tuning application provided by APPML BLAS could not complete successfully.

## 1.4 ViennaCL

### 1.4.1 Features

ViennaCL offers a complete set of BLAS operations, since all the three levels of BLAS are implemented, without support for complex numbers however. Additionally it supports some extended functionality:

- Sparse Matrices: Condensed or Coordinated, for all datatypes.
- Iterative solvers: Conjugate Gradient, Stabilised Bi-Conjugate Gradient, Generalised Minimum Residual.
- LU decomposition
- LU, Jacobi and row-normalization preconditioners.

ViennaCL is interoperable with the uBLAS package of the popular C++ Boost library, as it allows use of uBLAS objects by ViennaCL and vice-versa. Moreover, it provides an interface with MATLAB, EIGEN and MTL 4, allowing for ViennaCL functionality directly through these packages.

ViennaCL is designed in such way that it can hide the OpenCL API from the developer. Custom kernels, memory copies between devices and device selection can be invoked directly through ViennaCL without the need to use the OpenCL API directly.

All OpenCL devices are supported as long as the appropriate OpenCL driver is installed. However, ViennaCL uses the default device dictated by OpenCL unless the user explicitly implements custom functionality to choose a specific device of interest by using the OpenCL API and then pass it to ViennaCL. In other words, ViennaCL is generally unable to handle OpenCL platforms and devices properly. If the desired device is not the first one of the first platform, the OpenCL context and queue must be created manually and passed to ViennaCL.

**Installation**

As any OpenCL based library, primarily to the installation of ViennaCL, the OpenCL drivers corresponding to the devices we need to use must be installed. Since the installation of the AMD OpenCL driver was described in the previous subsection, we will just show the installation of the NVIDIA OpenCL driver here. There are also OpenCL drivers by IBM for and CELL/BE processor, by VIA and by Intel, but since we do not possess the corresponding devices or some APIs are not in production quality stage, we will not take them into account.

The OpenCL implementation of NVIDIA is provided by the CUDA toolkit whose current version is 4. All CUDA toolkits of version 3.2 and up provide OpenCL functionality.

For some of the ViennaCL tests, the Boost library is required; however it is not a prerequisite.

**CUDA Toolkit Installation**

The CUDA Toolkit is available for Linux, Windows and Mac OSX. For Linux, numerous packages are available, covering all the popular distributions. The CUDA toolkit can be downloaded freely. It is comprised by a single installation script which performs all the necessary set-up steps automatically.

**ViennaCL Installation**

ViennaCL is freely available for Windows and Linux and can be downloaded from *http://viennacl.sourceforge.net/*. For the Linux version, the package is provided in the form of source code. ViennaCL is a header-based library, which means that the library is not linked dynamically or statically, but is compiled with the application directly by including the needed headers.

### 1.4.2 Testing

First we check that the NVIDIA OpenCL library is correctly installed. Since no such utility is available in the CUDA Toolkit we need to install the CUDA SDK and execute oclDeviceQuery.

Next step is to perform the ViennaCL provided tests which succeeded. However we had to do tricks to force the ViennaCL tests to use the desired device. ViennaCL always uses the default device, offering no built-in way of choosing a device. That was a problem in our case since we have two OpenCL drivers installed, one for the CPU (AMD OpenCL) and one for the GPU (NVIDIA OpenCL). ViennaCL by default would use only the CPU.

# 2 Fast Fourier Transform

## 2.1 Summary

Fast Fourier Transform algorithms (FFT) are very common in synchrotron applications, due to their importance in signal analysis. Fast Fourier transform is an efficient Discrete Fourier transform algorithm with complexity *O(NlogN)* instead of $O(N^2)$.

## 2.2 Identification

The available OpenCL based FFT libraries are, as the case of BLAS, limited. We were able to identify the following library only:

- AMD Accelerated Parallel Processing Math Libraries (APPML)

APPML has already been summarised in section 1, where we focused on the BLAS part. In the following subsection we will describe the FFT part, skipping any OpenCL related installation or test that has already been described.

Installation and testing were performed on a HPC node with two Intel XEON E5640 (2.67Ghz) CPUs and two NVIDIA M2050 GPUs with ECC enabled. The node is running the 64-bit Linux distribution CentOS 5 with OpenCL APIs by NVIDIA (Toolkit version 4.0) and AMD (APP version 2.4) installed.

## 2.3 AMD Accelerated Parallel Processing Math Libraries (APPML) - FFT

### 2.3.1 Features

In contrast to the APPML BLAS, the FFT part provides many features. Version 1.2.144 is considered limited. The most important features of APPML FFT 1.2 are:

- Up to three-dimensional vectors and arrays
- Single precision FFT, direct and inverse. Double precision is not supported
- Power of 2 problem sizes

APPML FFT 1.4 extends the functionality of the library with the following:

- Double precision FFT
- Power of 3 and 5 problem sizes

### 2.3.2 Installation

In section 2 the installation of the prerequisite APP SDK 2.4 is described. The FFT part of the APPML 1.2, must like the BLAS part, is no longer provided by AMD. However it can be provided upon request. The file corresponding to the FFT libraries for our system is "*clAmdFft-1.2.144-Linux.tar.gz*" from which we get an installation script. After its execution, update of the system library paths is required.

APPML 1.4.182 is available freely at *http://developer.amd.com*. The installation process is similar to the previous version.

### 2.3.3 Testing

Testing of the AMD OpenCL library was covered in section 1. To test APPML FFT we execute the available samples. These samples are provided as source code which needs to be compiled with the cmake tool. The samples failed to compile due to their dependency on the Boost library.

We test the library by creating a data-set, transforming it to the frequency domain with FFT, and then transforming back to real domain with Inverse FFT.

# 3 Benchmarks

## 3.1 Benchmark setup and methodology

The popular uBLAS and FFTw3 libraries are used as performance reference. uBLAS was chosen because it is part of the C++ Boost extension, while FFTw3 is well established in scientific software. Both are free, open-source and single-threaded, however a multi-threaded version of FFTw3 exists. For BLAS, Intel Math Kernel BLAS is known to be one of the best in terms of performance, however it is not free.

uBLAS and ViennaCL have the DNDEBUG flag active which ensures that both are running their "release" version of GEMM and not the debug version which performs additional checks.

APPML FFT is set to use the fast precision (*CLFFT_SINGLE_FAST*, *CLFFT_DOUBLE_FAST*) which ensures that sines and cosines needed for the FFT are calculated on the GPU instead of the CPU.

For the benchmark the following were used:

- G++ 4.4.4 64-bit compiler
- Linux CentOS 5 64-bit Operating System with kernel 2.6.18-194.el5
- 48Gb DDR3-1066 ECC Registered memory
- A Quad-core Intel XEON E5640 CPU clocked at 2.67Ghz with hyper-threading
- A NVIDIA Tesla M2050 GPU with ECC enabled and driver version 260.19.36 64-bit (development driver)
- An AMD Cayman HD6990 GPU (only one of the two available processors on the PCB board was used)
- CUDA Toolkit 4.0 (OpenCL 1.0, CUFFT, CUBLAS)
- AMD APP SDK 2.4 (OpenCL 1.1)
- APPML 1.2.142, 1.4.188
- Boost 1.46.1 (uBLAS)
- ViennaCL 1.1.2

A template program was created that implements data and device initialisation, profiling and benchmarking functions.

The input data arrays, whose size is given as a command line argument, are initialised via a random number generator. For OpenCL, device initialisation consists of selecting a device (GPU or CPU) based on a command line argument and initiating an active context to it which is passed to the studied library. CUDA and CPU libraries do not require explicit device selection due to the fact that these libraries can use only one specific device. The profiler is using the Linux (sys/time.h) or Windows (windows.h) high precision timers. APPML allows the use of OpenCL events, the clEvents, which are also used to time library calls. All APPML results presented here are based on clEvents.

The benchmarking function initialises and executes the library in question while profiling each step. Library functions that do not belong to the initialisation step are called five times and the elapsed time returned is averaged. In the case of BLAS libraries, these functions are called once and timed and then called five times in order to calculate part of the initialisation lag. Each benchmark executable is executed twice in order to be able to cross-check the results.

Example of the above from a ViennaCL benchmark output:

```
#1st Run (includes initialization)
196.600000 (ms) 0.196600 (s)

# 5 Runs. Time averaged
7.2482 (ms) 0.0072482 (s)
```

The main and benchmarking functions were customised for each library, precision, FFT "placeness" and library tuning. All the steps, from initialisation to execution are profiled. Non-crucial results, such as memory copies and data initialisation, are not presented here. Results for out-of-place FFT transforms are excluded from the report since the performance is similar to in-place transform.

The initialisation time for each library is excluded from the tables but will be discussed briefly.

The results are gathered in tab-delimited columns by special scripts, one for FFT and one for BLAS.

All benchmarks were executed for *N in {128,256,512,1024,2048}* with problem size *NxN*. In the 2D case, *N* is the size of each dimension of the matrix. In the 1D case NxN is size of the 1D array.

All the presented time values are in milliseconds (ms).

Important remarks:

- APPML FFT of both versions failed on the NVIDIA M2050 GPU
- ViennaCL does not support complex numbers
- APPML FFT 1.2.144 does not support double precision
- ViennaCL failed when set to use double precision on the AMD Cayman GPU
- APPML FFT plan creation implicitly does Just-In-Time compilation of the needed kernels which explains the increased time seen in the results.
- CUBLAS and CUFFT and CUDA in general were seen to experience significant initialisation delay of at least four seconds which cannot be seen on the results. This behaviour was replicated on a similar setup at CaSToRC with C2070 GPUs, but could not be replicated on a workstation with a GTX 580 GPU. It appears to be common on systems that do not have a Xorg server running. OpenCL did not exhibit this behaviour.

## 3.2 BLAS results

### 3.2.1 uBLAS

| uBLAS | | | | |
|---|---|---|---|---|
| | Intel XEON E5640 (1 thread) | | | |
| NxN | Float | Double | Complex Float | Complex Double |
| 16384 | 5,336 | 6,578 | 27,901 | 23,576 |
| 65536 | 34,872 | 42,114 | 243,235 | 245,520 |
| 262144 | 326,126 | 360,768 | 1964,700 | 2254,010 |
| 1048576 | 9788,15 | 9822,77 | 17977,30 | 24291,40 |
| 4194304 | 80357,7 | 86439,5 | 203710,0 | 245240,0 |

uBLAS is not build with performance as its primary goal. It is designed as a generic and reliable easy to use library.

### 3.2.2 CUBLAS

| CUBLAS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | NVIDIA M2050 | | | | | | | |
| | Column-Major (default for CUBLAS) | | | | Row-Major (Transposed) | | | |
| NxN | Float | Double | Complex Float | Complex Double | Float | Double | Complex Float | Complex Double |
| 16384 | 0,0420 | 0,0646 | 0,0704 | 0,133 | 0,0418 | 0,0642 | 0,0704 | 0,135 |
| 65536 | 0,123 | 0,250 | 0,422 | 0,577 | 0,124 | 0,247 | 0,423 | 0,579 |
| 262144 | 0,718 | 1,303 | 1,576 | 3,909 | 0,754 | 1,293 | 1,554 | 3,912 |
| 1048576 | 4,681 | 7,533 | 11,598 | 29,657 | 4,685 | 7,576 | 11,450 | 29,688 |
| 4194304 | 29,032 | 57,773 | 89,638 | 232,896 | 29,010 | 58,084 | 88,566 | 233,250 |

CUBLAS is one of the top BLAS libraries in terms of performance. Column-major and row-major data orientations are compared, because CUBLAS by default uses column-major data orientation in order to maximise compatibility with the FORTRAN BLAS library. However, in C the default data orientation is row-major and therefore the matrices need to be transposed in CUBLAS. The performance between the two orientations is virtually identical.

For the comparative results only the data of the transposed matrices will be used.

### 3.2.3 APPML BLAS

**APPML BLAS 1.2.144**

| | Intel XEON E5640 (4 cores) | | | | NVIDIA M2050 | | | |
|---|---|---|---|---|---|---|---|---|
| ***NxN*** | Float | Double | Complex Float | Complex Double | Float | Double | Complex Float | Complex Double |
| 16384 | 0,910 | 1,767 | 1,847 | 4,053 | 0,554 | 0,527 | 0,822 | 0,837 |
| 65536 | 5,023 | 9,617 | 11,179 | 26,449 | 0,867 | 0,504 | 2,263 | 1,547 |
| 262144 | 36,757 | 42,278 | 72,912 | 209,344 | 4,796 | 3,178 | 19,601 | 11,813 |
| 1048576 | 281,061 | 351,881 | 639,517 | 1742,192 | 34,571 | 24,003 | 150,867 | 95,034 |
| 4194304 | 2584,450 | 2725,534 | 4655,952 | 13297,718 | 276,790 | 193,380 | 1213,913 | 780,406 |

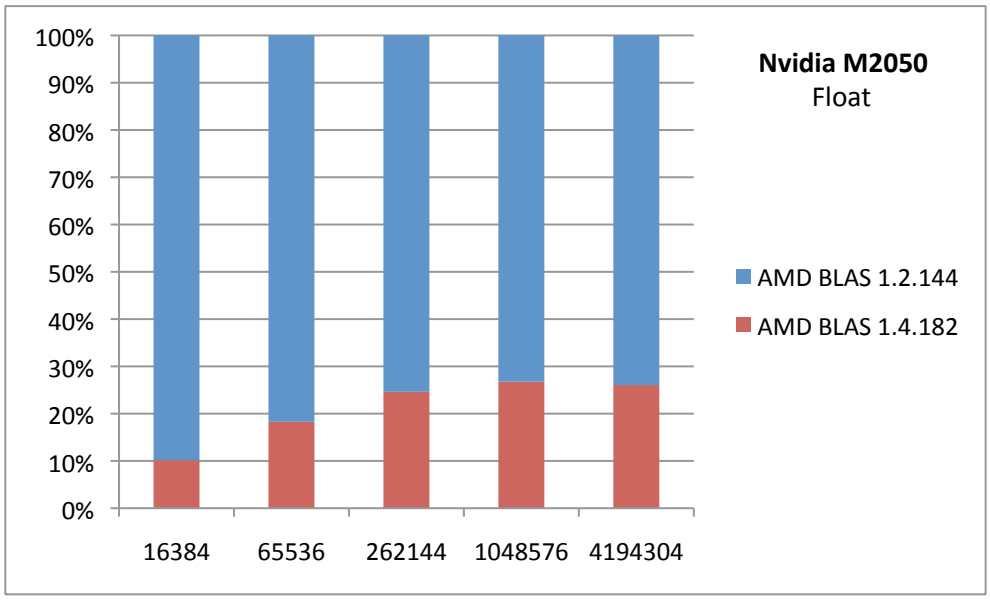
**APPML BLAS 1.4.182**

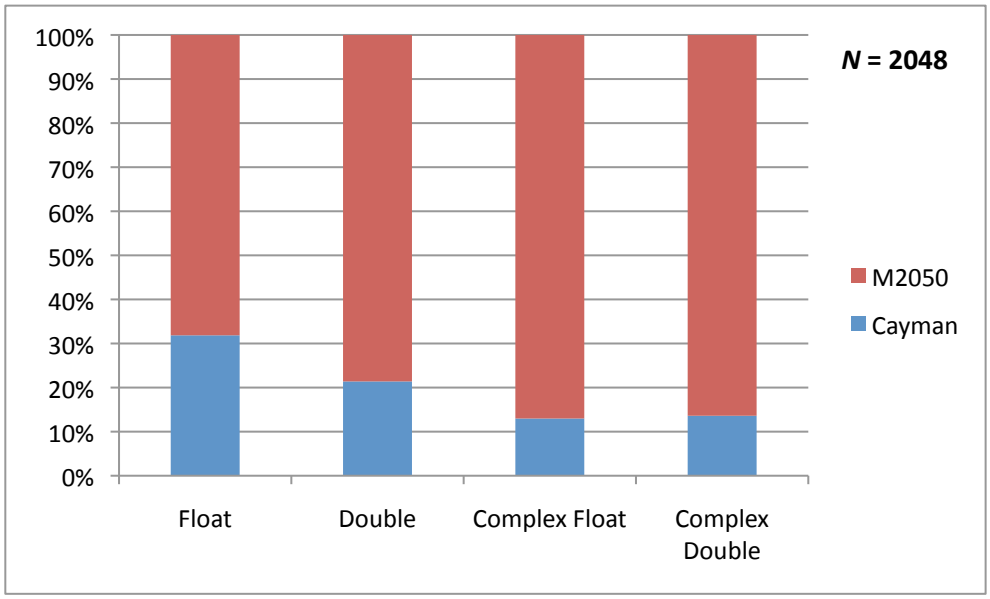
| | Intel XEON E5640 (4 cores) | | | | NVIDIA M2050 | | | |
|---|---|---|---|---|---|---|---|---|
| ***NxN*** | Float | Double | Complex Float | Complex Double | Float | Double | Complex Float | Complex Double |
| 16384 | 0,281 | 0,708 | 1,248 | 3,229 | 0,056 | 0,166 | 0,328 | 0,435 |
| 65536 | 1,489 | 4,061 | 9,948 | 12,117 | 0,159 | 0,458 | 0,990 | 2,872 |
| 262144 | 12,933 | 22,641 | 78,682 | 109,660 | 1,181 | 3,528 | 7,207 | 22,163 |
| 1048576 | 145,262 | 262,859 | 781,505 | 950,143 | 9,255 | 28,285 | 57,760 | 194,653 |
| 4194304 | 1592,512 | 2641,496 | 6582,075 | 9860,412 | 72,108 | 240,640 | 504,624 | 1611,922 |

| | AMD Cayman HD6990 | | | |
|---|---|---|---|---|
| ***NxN*** | Float | Double | Complex Float | Complex Double |
| 4194304 | 22,972 | 51,491 | 65,517 | 219,382 |

APPML is a rather new library with plenty of room for improvement. The new version of APPML BLAS shows a remarkable performance gain in some cases and direct comparison of the two OpenCL-based libraries is of interest. On the GPU side, the library seems to handle double-precision worse on the M2050 GPU, but there is up to ten times improvement in single precision. APPML BLAS does not officially support Intel processors and Nvidia GPUs nor is expected to be optimised for them. An AMD GPU, the Cayman HD6990, was added to the tests to study how the library performs on the devices it was designed for.

Following are two comparison charts. For 100% being the execution time of the slowest library or device, they show the relative execution time of the fastest. The first compares the SGEMM execution time on the M2050 GPU between the two library versions. The latter compares SGEMM, DGEMM, CGEMM and ZGEMM of APPML 1.4 between the Nvidia M2050 and the AMD Cayman for square matrices of $N = 2048$ (matrices with $4194304$ elements).



### 3.2.4 ViennaCL

**ViennaCL**

| | Intel XEON E5640 (8 hyperthreads) | | | | NVIDIA M2050 (ECC) | | | |
|---|---|---|---|---|---|---|---|---|
| | Default | | Tuned | | Default | | Tuned | |
| ***NxN*** | Float | Double | Float | Double | Float | Double | Float | Double |
| 16384 | 7,248 | 7,445 | 7,264 | 7,186 | 0,246 | 0,288 | 0,246 | 0,296 |
| 65536 | 48,670 | 51,450 | 48,618 | 49,325 | 0,859 | 1,281 | 0,884 | 1,329 |
| 262144 | 350,612 | 366,655 | 350,286 | 385,188 | 5,158 | 8,331 | 5,157 | 8,324 |
| 1048576 | 2699,100 | 2730,310 | 2671,650 | 2755,070 | 37,654 | 61,032 | 37,646 | 61,011 |
| 4194304 | 21000,200 | 21319,600 | 20949,100 | 21349,500 | 290,526 | 471,793 | 290,616 | 471,775 |

| | AMD Cayman (HD6990) | |
|---|---|---|
| | Default | |
| ***NxN*** | Float | Double |
| 4194304 | 1735,18 | ***Failed*** |

ViennaCL is built to be easy to use and be interoperable with other BLAS libraries such as uBLAS and eigen. Performance issues were noticed on the AMD Cayman GPU as well as failure of the library when using double precision on the latter.

The library provides tuning capability. Tuning does not seem to affect GEMM performance and is discarded in the comparative results. The only difference noticed by the tuning was the initialisation delay of ViennaCL was moved from the ViennaCL matrix allocation to the parameter loader.

### 3.2.5 GEMM Comparative results

The following tables present the performance grain of each library against uBLAS ($t_{ublas}/t_{other}$). A value of 1 represents identical performance and of less than 1 worse performance (marked as red). On each table the results marked as green denote the best performance for a given kind of device (CPU, GPU).

**SGEMM Acceleration factors (Single precision**

| | E5640 | | | M2050 | | | | HD6990 | |
|---|---|---|---|---|---|---|---|---|---|
| ***NxN*** | AMD 1.2 | AMD 1.4 | ViennaCL | AMD 1.2 | AMD 1.4 | ViennaCL | CUBLAS | AMD 1.4 | ViennaCL |
| 16384 | 5,863 | 19,015 | 0,736 | 9,628 | 94,823 | 21,689 | 127,646 | | |
| 65536 | 6,943 | 23,420 | 0,717 | 40,203 | 220,011 | 40,587 | 282,138 | | |
| 262144 | 8,872 | 25,216 | 0,930 | 68,006 | 276,233 | 63,222 | 432,528 | | |
| 1048576 | 34,826 | 67,383 | 3,626 | 283,132 | 1057,558 | 259,951 | 2089,075 | | |
| 4194304 | 31,093 | 50,460 | 3,827 | 290,320 | 1114,408 | 276,594 | 2770,019 | 3498,061 | 46,311 |

**DGEMM Acceleration factors (Double precision)**

| | E5640 | | | M2050 | | | | HD6990 |
|---|---|---|---|---|---|---|---|---|
| ***NxN*** | AMD 1.2 | AMD 1.4 | ViennaCL | AMD 1.2 | AMD 1.4 | ViennaCL | CUBLAS | AMD 1.4 |
| 16384 | 3,722 | 9,290 | 0,884 | 12,476 | 39,646 | 22,839 | 102,455 | |
| 65536 | 4,379 | 10,371 | 0,819 | 83,506 | 92,018 | 32,886 | 170,501 | |
| 262144 | 8,533 | 15,934 | 0,984 | 113,505 | 102,267 | 43,303 | 279,016 | |
| 1048576 | 27,915 | 37,369 | 3,598 | 409,228 | 347,278 | 160,946 | 1296,530 | |
| 4194304 | 31,715 | 32,724 | 4,054 | 446,993 | 359,206 | 183,215 | 1488,186 | 1678,746 |

**CGEMM Acceleration factors (Single precision Complex)**

| | E5640 | | M2050 | | | HD6990 |
|---|---|---|---|---|---|---|
| ***NxN*** | AMD 1.2 | AMD 1.4 | AMD 1.2 | AMD 1.4 | CUBLAS | AMD 1.4 |
| 16384 | 15,106 | 22,357 | 33,933 | 85,090 | 396,324 | |
| 65536 | 21,758 | 24,450 | 107,504 | 245,715 | 575,568 | |
| 262144 | 26,946 | 24,970 | 100,236 | 272,607 | 1264,286 | |
| 1048576 | 28,111 | 23,003 | 119,160 | 311,242 | 1570,125 | |
| 4194304 | 43,753 | 30,949 | 167,813 | 403,687 | 2300,103 | 3109,293 |

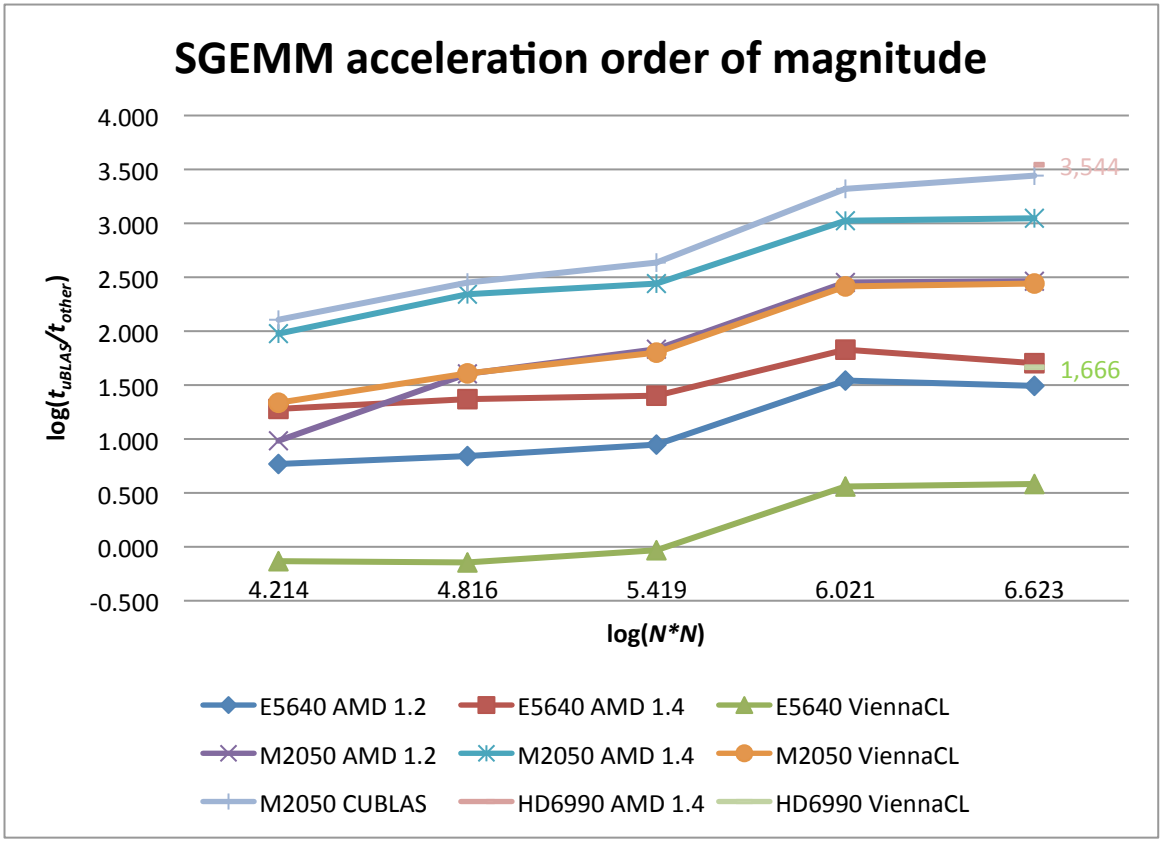
**ZGEMM Acceleration factors (Double precision Complex)**

| | E5640 | | M2050 | | | HD6990 |
|---|---|---|---|---|---|---|
| ***NxN*** | AMD 1.2 | AMD 1.4 | AMD 1.2 | AMD 1.4 | CUBLAS | AMD 1.4 |
| 16384 | 5,817 | 7,301 | 28,159 | 54,257 | 174,123 | |
| 65536 | 9,283 | 20,262 | 158,703 | 85,480 | 424,335 | |
| 262144 | 10,767 | 20,555 | 190,814 | 101,703 | 576,178 | |
| 1048576 | 13,943 | 25,566 | 255,606 | 124,793 | 818,217 | |
| 4194304 | 18,442 | 24,871 | 314,246 | 152,141 | 1051,404 | 1117,866 |

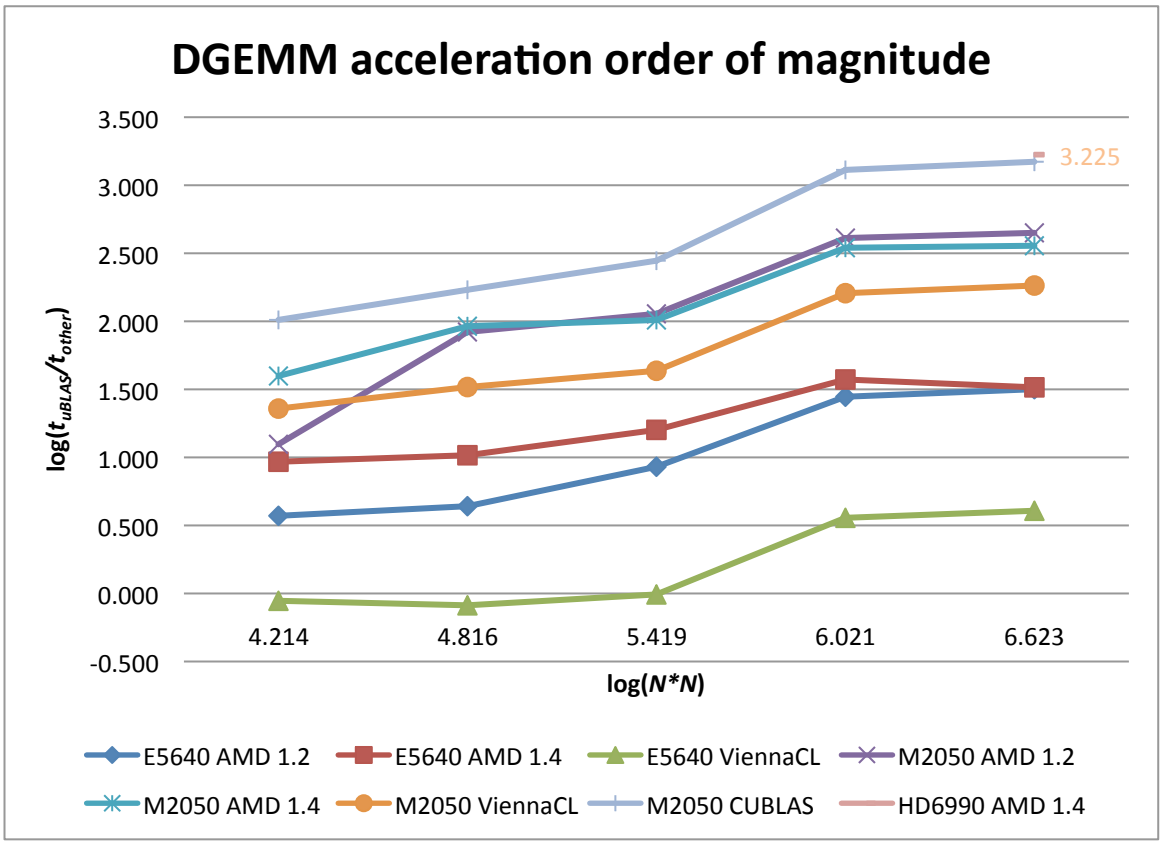
ViennaCL in general, on the Intel processor, even though it is using four CPU cores instead of one as uBLAS does, performs worse than uBLAS, while for big matrices it roughly scales to

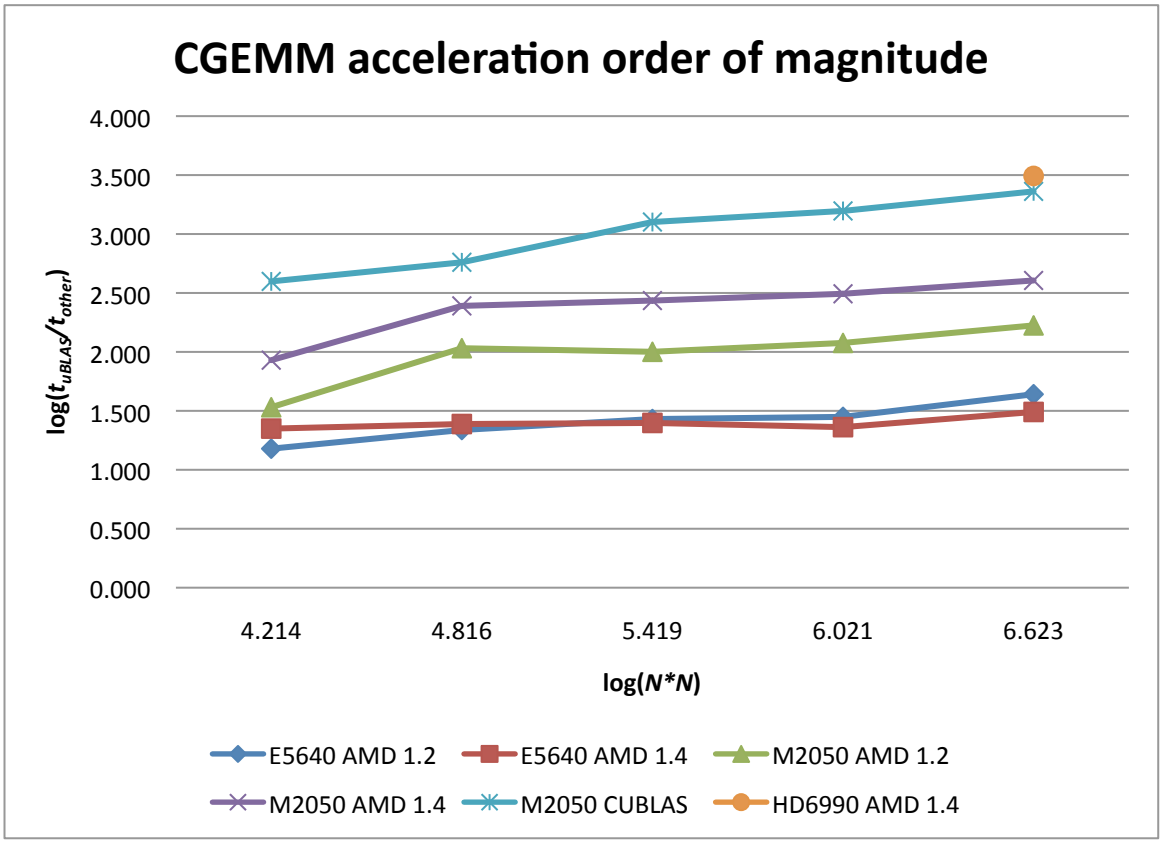
the CPU core count. On the Nvidia M2050 GPU the library performs similarly with APPML 1.2. As far as the Cayman GPU is concerned, the library clearly has issues with this device and performs poorly or fails (double precision).

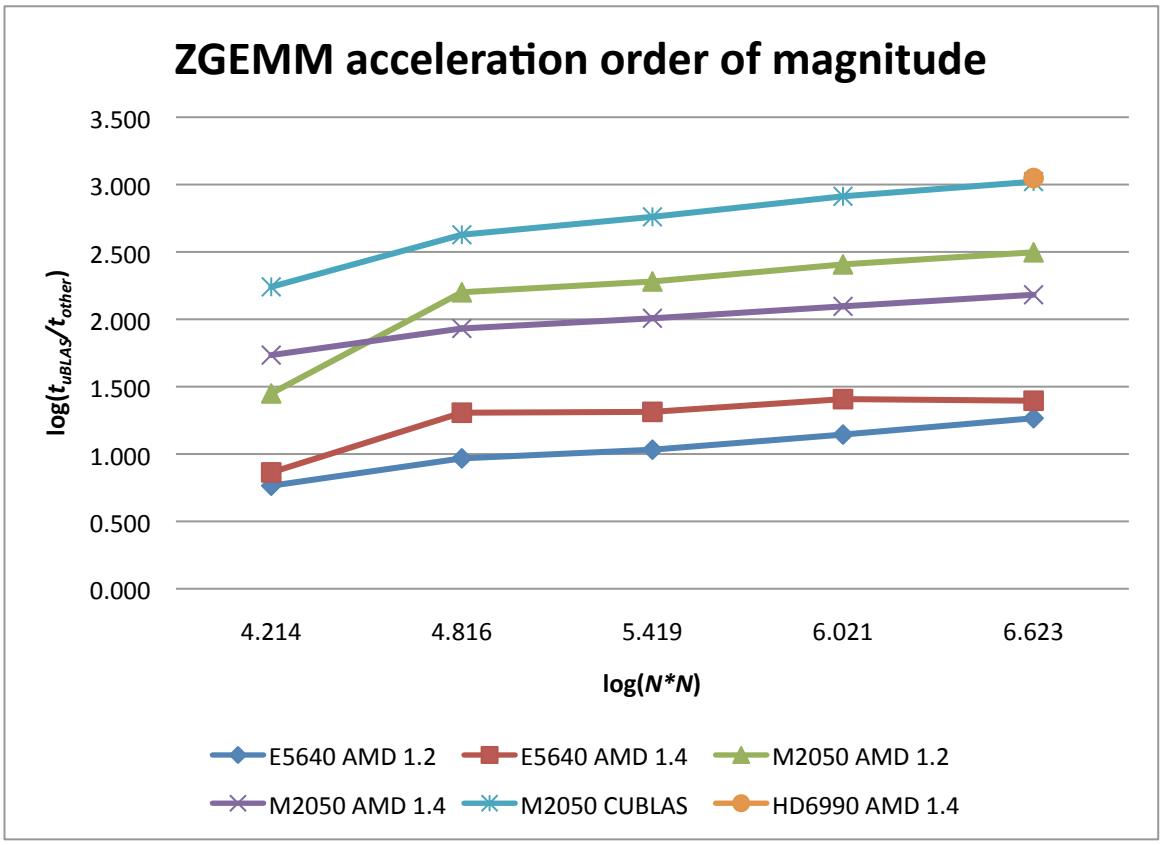
APPML 1.4 performed very well on the AMD Cayman GPU. However it is important to note that a single Cayman GPU processor has a theoretical peak of ~2,5 GFlops in single precision, while M2050 is rated at ~1 GFlop (ECC functionality does not seem to affect performance here). The theoretical peak performance in double precision is similar between the two GPUs. Some results for CUBLAS SGEMM and DGEMM with ECC off and on that show the effect of ECC are available by NVIDIA[1].




[1] http://www.microway.com/pdfs/TeslaC2050-Fermi-Performance.pdf


DGEMM acceleration order of magnitude
3.500
3.000
2.500
2.000
1.500
1.000
0.500
0.000
-0.500
3.225
4.214
4.816
5.419
6.021
6.623
log(N*N)
log(tuBLAS/tother)
E5640 AMD 1.2
E5640 AMD 1.4
E5640 ViennaCL
M2050 AMD 1.2
M2050 AMD 1.4
M2050 ViennaCL
M2050 CUBLAS
HD6990 AMD 1.4


CGEMM acceleration order of magnitude
4.000
3.500
3.000
2.500
2.000
1.500
1.000
0.500
0.000
4.214
4.816
5.419
6.021
6.623
log(N*N)
log(tuBLAS/tother)
E5640 AMD 1.2
E5640 AMD 1.4
M2050 AMD 1.2
M2050 AMD 1.4
M2050 CUBLAS
HD6990 AMD 1.4




As far as initialisation lag is concerned, uBLAS has none. OpenCL libraries are expected to have some latency due to the JIT compilation. In general initialisation of GPU devices also adds latency. As discussed before, the M2050 GPU had significant lag when CUDA was used (CUBLAS).

**Approximate Initilisation delay (*ms*)**

| | XEON | M2050 / C2070 | Cayman | GTX580 |
|---|---|---|---|---|
| **APPML 1.2** | 230 | 630 | *N/A* | *N/A* |
| **APPML 1.4** | 90 | 210 | 200 | *N/A* |
| **VIENNACL** | 400 | 1000 | 1000 | *N/A* |
| **CUBLAS** | *N/A* | 4000-5000 | *N/A* | 170 |

## 3.3 FFT results

### 3.3.1 FFTw3

| | FFTw3 – Intel XEON E5640 (1 thread) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | ***1D*** In-place Complex - float | | | | | ***2D*** In-place Complex - float | | | | |
| **NxN** | Forward | Inverse | Total | Plan | Total | Forward | Inverse | Total | Plan | Total |
| 16384 | 0,465 | 0,461 | 0,927 | 0,905 | 1,832 | 0,496 | 0,495 | 0,991 | 2,229 | 3,220 |
| 65536 | 2,424 | 2,390 | 4,814 | 1,120 | 5,934 | 2,524 | 2,514 | 5,038 | 1,722 | 6,760 |
| 262144 | 11,650 | 11,637 | 23,287 | 3,462 | 26,749 | 10,956 | 10,916 | 21,873 | 2,008 | 23,881 |
| 1048576 | 55,824 | 57,664 | 113,488 | 6,657 | 120,145 | 64,002 | 62,582 | 126,584 | 1,724 | 128,308 |
| 4194304 | 251,564 | 252,257 | 503,821 | 6,783 | 510,604 | 310,132 | 310,206 | 620,338 | 1,563 | 621,901 |

| | FFTw3 – Intel XEON E5640 (1 thread) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | ***1D*** In-place Complex - double | | | | | ***2D*** In-place Complex - double | | | | |
| **NxN** | Forward | Inverse | Total | Plan | Total | Forward | Inverse | Total | Plan | Total |
| 16384 | 0,523 | 0,509 | 1,032 | 1,031 | 2,063 | 0,602 | 0,603 | 1,205 | 2,339 | 3,544 |
| 65536 | 2,242 | 2,976 | 5,217 | 3,129 | 8,346 | 2,948 | 2,713 | 5,661 | 1,958 | 7,619 |
| 262144 | 14,291 | 14,284 | 28,575 | 8,709 | 37,284 | 13,830 | 13,790 | 27,619 | 1,903 | 29,522 |
| 1048576 | 58,603 | 58,402 | 117,005 | 6,908 | 123,913 | 69,053 | 69,333 | 138,386 | 1,571 | 139,957 |
| 4194304 | 275,666 | 273,954 | 549,620 | 6,629 | 556,249 | 339,774 | 339,284 | 679,058 | 1,558 | 680,616 |

The "default" single threaded FFTw3 version is used for the tests. FFTw3 provides some options when creating a plan which affect the performance of the FFT calculations, but can increase the FFT calculation time significantly. In our tests the fast plan creation is used (*FFTW_ESTIMATE*).

### 3.3.2 CUFFT

| | CUFFT - M2050 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | ***1D*** In-place Complex - float | | | | | ***2D*** In-place Complex - float | | | | |
| **NxN** | Forward | Inverse | Total | Plan | Total | Forward | Inverse | Total | Plan | Total |
| 16384 | 0,051 | 0,041 | 0,093 | 0,158 | 0,251 | 0,040 | 0,040 | 0,080 | 0,111 | 0,191 |
| 65536 | 0,070 | 0,060 | 0,130 | 0,157 | 0,287 | 0,061 | 0,060 | 0,120 | 0,188 | 0,308 |
| 262144 | 0,170 | 0,160 | 0,330 | 0,253 | 0,583 | 0,160 | 0,158 | 0,319 | 0,192 | 0,511 |
| 1048576 | 0,627 | 0,615 | 1,242 | 0,313 | 1,555 | 0,614 | 0,615 | 1,229 | 0,221 | 1,450 |
| 4194304 | 2,400 | 2,382 | 4,782 | 0,449 | 5,231 | 2,386 | 2,381 | 4,767 | 0,322 | 5,089 |

| | CUFFT - M2050 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | ***1D*** In-place Complex - double | | | | | ***2D*** In-place Complex - double | | | | |
| **NxN** | Forward | Inverse | Total | Plan | Total | Forward | Inverse | Total | Plan | Total |
| 16384 | 0,048 | 0,040 | 0,088 | 0,184 | 0,272 | 0,040 | 0,039 | 0,079 | 0,108 | 0,187 |
| 65536 | 0,118 | 0,108 | 0,225 | 0,306 | 0,531 | 0,108 | 0,106 | 0,215 | 0,233 | 0,448 |
| 262144 | 0,309 | 0,297 | 0,606 | 0,285 | 0,891 | 0,301 | 0,297 | 0,598 | 0,206 | 0,804 |
| 1048576 | 1,169 | 1,158 | 2,327 | 0,391 | 2,718 | 1,156 | 1,157 | 2,313 | 0,256 | 2,569 |
| 4194304 | 5,419 | 5,392 | 10,811 | 0,606 | 11,417 | 5,411 | 5,410 | 10,821 | 0,470 | 11,291 |

### 3.3.3 APPML FFT

| | APPML FFT 1.2.144 - XEON E5640 (4 cores) | | | | |
|---|---|---|---|---|---|
| | *1D* In-place Complex - float | | | | |
| **NxN** | Forward | Inverse | Total | Plan | Total |
| 16384 | 0,768 | 0,582 | 1,351 | 530,307 | 531,658 |
| 65536 | 2,626 | 1,966 | 4,592 | 1146,186 | 1150,778 |
| 262144 | 8,240 | 6,386 | 14,626 | 766,210 | 780,836 |
| 1048576 | 37,657 | 38,117 | 75,774 | 1772,579 | 1848,353 |
| 4194304 | 194,267 | 194,123 | 388,390 | 1938,636 | 2327,026 |
| | | | | | |
| | *2D* In-place Complex - float | | | | |
| **NxN** | Forward | Inverse | Total | Plan | Total |
| 16384 | 0,437 | 0,370 | 0,807 | 922,921 | 923,728 |
| 65536 | 1,684 | 1,579 | 3,263 | 1016,779 | 1020,042 |
| 262144 | 5,071 | 4,932 | 10,003 | 1113,055 | 1123,058 |
| 1048576 | 23,960 | 21,995 | 45,956 | 1228,455 | 1274,411 |
| 4194304 | 139,515 | 142,788 | 282,303 | 1612,308 | 1894,611 |

| | APPML FFT 1.4.182 – Intel XEON E5640 (4 cores) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *1D* In-place Complex - float | | | | | *2D* In-place Complex - float | | | | |
| **NxN** | Forward | Inverse | Total | Plan | Total | Forward | Inverse | Total | Plan | Total |
| 16384 | 0,695 | 0,508 | 1,203 | 950,111 | 951,314 | 0,460 | 0,334 | 0,794 | 941,30 | 942,09 |
| 65536 | 1,909 | 1,504 | 3,412 | 2390,10 | 2393,51 | 1,337 | 1,180 | 2,516 | 1006,06 | 1008,57 |
| 262144 | 5,339 | 5,126 | 10,465 | 2768,05 | 2778,52 | 5,094 | 4,617 | 9,710 | 1122,73 | 1132,44 |
| 1048576 | 28,149 | 28,432 | 56,581 | 2933,22 | 2989,81 | 23,438 | 22,000 | 45,438 | 1251,17 | 1296,61 |
| 4194304 | 200,27 | 191,93 | 392,21 | 1939,32 | 2331,53 | 263,76 | 264,60 | 528,35 | 846,34 | 1374,69 |

| | APPML FFT 1.4.182 – Intel XEON E5640 (4 cores) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *1D* In-place Complex - double | | | | | *2D* In-place Complex - double | | | | |
| **NxN** | Forward | Inverse | Total | Plan | Total | Forward | Inverse | Total | Plan | Total |
| 16384 | 0,585 | 0,511 | 1,096 | 948,82 | 949,92 | 0,480 | 0,340 | 0,820 | 942,34 | 943,16 |
| 65536 | 1,926 | 1,568 | 3,494 | 2388,74 | 2392,23 | 1,479 | 1,618 | 3,097 | 1010,66 | 1013,75 |
| 262144 | 5,282 | 5,442 | 10,724 | 2771,77 | 2782,49 | 5,951 | 4,463 | 10,414 | 1138,60 | 1149,02 |
| 1048576 | 27,841 | 27,268 | 55,109 | 2941,08 | 2996,19 | 24,137 | 22,214 | 46,351 | 1250,21 | 1296,56 |
| 4194304 | 195,17 | 199,68 | 394,85 | 1942,28 | 2337,13 | 272,28 | 266,36 | 538,64 | 863,34 | 1401,98 |

| | APPML FFT 1.4.182 - Cayman HD6990 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *1D* In-place Complex - float | | | | | *2D* In-place Complex - float | | | | |
| **NxN** | Forward | Inverse | Total | Plan | Total | Forward | Inverse | Total | Plan | Total |
| 4194304 | 45,248 | 44,584 | 89,832 | 5988,29 | 6078,12 | 160,93 | 148,42 | 309,35 | 2706,53 | 3015,89 |

### 3.3.4 FFT Comparative results

The following tables present the performance gain of each library against FFTw3 ($t_{FFTw3}/t_{other}$). A value of 1 represents identical performance and of less than 1 worse performance (marked as red). Only the in-place forward an inverse Fast-Fourier transforms are compared.

We avoid comparing FFT plan creation elapsed times due to their ambiguous nature. FFTw3 offers various ways to create a plan which require different time to complete, but can enhance the performance of the FFT. APPML compiles the OpenCL FFT kernels upon creation of the plan and it is not possible to distinguish between the actual time spent on the plan creation and on the JIT compilation. CUFFT plan creation is rapid in comparison to FFTw3, but is in the same scale of the actual time needed for the FFT transform on the GPU.

Both APPML FFT and CUFFT offer "batched transform" capability. In many cases FFTs that need to be performed are small in size. With the batched transform feature, the FFTs can be gathered and performed in such way that the device resources are efficiently utilised. Performing sequential small FFT transforms on a GPU is inefficient due to device underutilisation and the higher latency of small memory transfers that can accumulate. High performance in the case of the FFT plans typically found in synchrotron image processing can be achieved by batching multiple plans and executing them at once.

**Acceleration Factors over FFTw3**

***1D In-place Complex-float***

| | AMD 1.2 - E5640 | | AMD 1.4 - E5640 | | CUFFT - M2050 | | AMD 1.4 - HD6990 | |
|---|---|---|---|---|---|---|---|---|
| ***NxN*** | Forward | Inverse | Forward | Inverse | Forward | Inverse | Forward | Inverse |
| 16384 | 0,606 | 0,792 | 0,670 | 0,907 | 9,054 | 11,194 | | |
| 65536 | 0,923 | 1,215 | 1,270 | 1,590 | 34,530 | 39,967 | | |
| 262144 | 1,414 | 1,822 | 2,182 | 2,270 | 68,449 | 72,729 | | |
| 1048576 | 1,482 | 1,513 | 1,983 | 2,028 | 89,063 | 93,732 | | |
| 4194304 | 1,295 | 1,299 | 1,256 | 1,314 | 104,827 | 105,884 | 5,560 | 5,658 |

***2D In-place Complex-float***

| | AMD 1.2 - E5640 | | AMD 1.4 - E5640 | | CUFFT - M2050 | | AMD 1.4 - HD6990 | |
|---|---|---|---|---|---|---|---|---|
| ***NxN*** | Forward | Inverse | Forward | Inverse | Forward | Inverse | Forward | Inverse |
| 16384 | 1,134 | 1,339 | 1,080 | 1,481 | 12,467 | 12,380 | | |
| 65536 | 1,498 | 1,592 | 1,888 | 2,132 | 41,644 | 42,047 | | |
| 262144 | 2,160 | 2,213 | 2,151 | 2,365 | 68,392 | 68,917 | | |
| 1048576 | 2,671 | 2,845 | 2,731 | 2,845 | 104,272 | 101,693 | | |
| 4194304 | 2,223 | 2,172 | 1,176 | 1,172 | 129,980 | 130,262 | 1,927 | 2,090 |

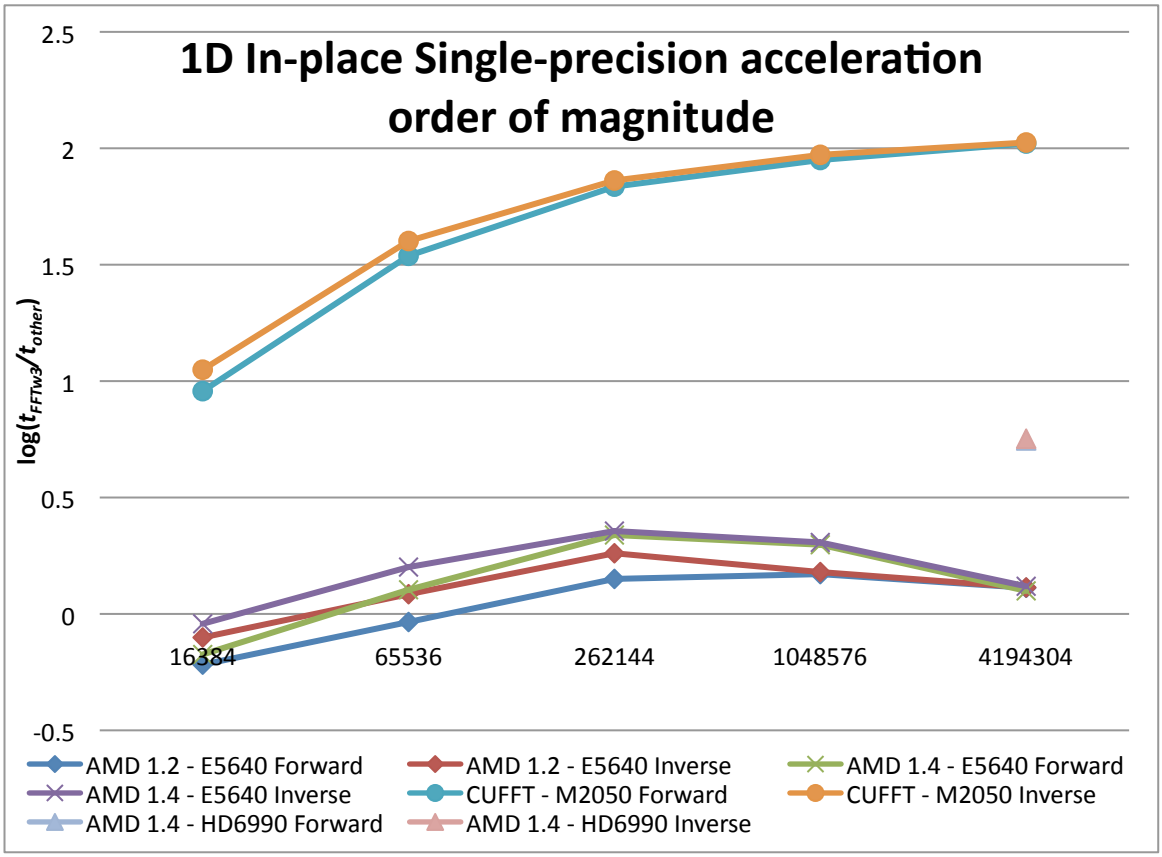
***1D In-place Complex-double***

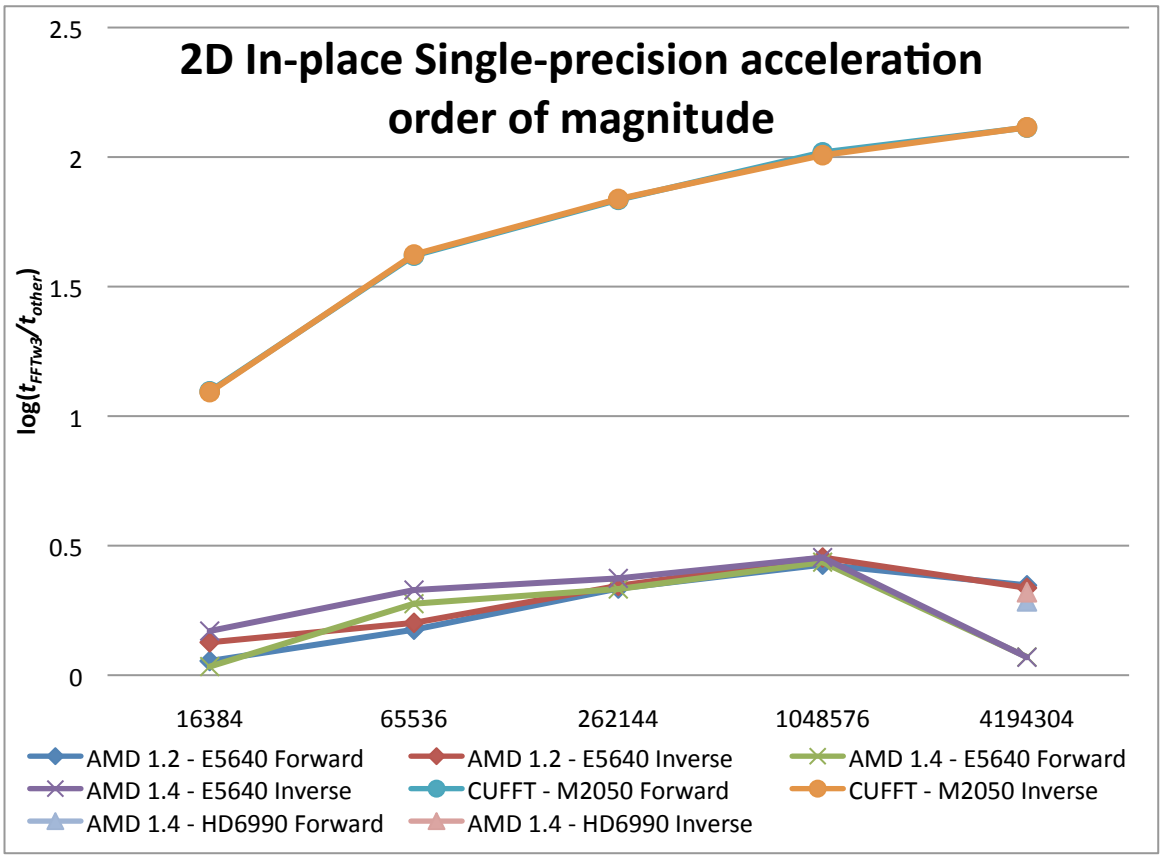
| | AMD 1.4 - E5640 | | CUFFT - M2050 | |
|---|---|---|---|---|
| ***NxN*** | Forward | Inverse | Forward | Inverse |
| 16384 | 0,895 | 0,995 | 10,904 | 12,848 |
| 65536 | 1,164 | 1,898 | 19,061 | 27,605 |
| 262144 | 2,706 | 2,625 | 46,250 | 48,061 |
| 1048576 | 2,105 | 2,142 | 50,131 | 50,451 |
| 4194304 | 1,412 | 1,372 | 50,874 | 50,806 |

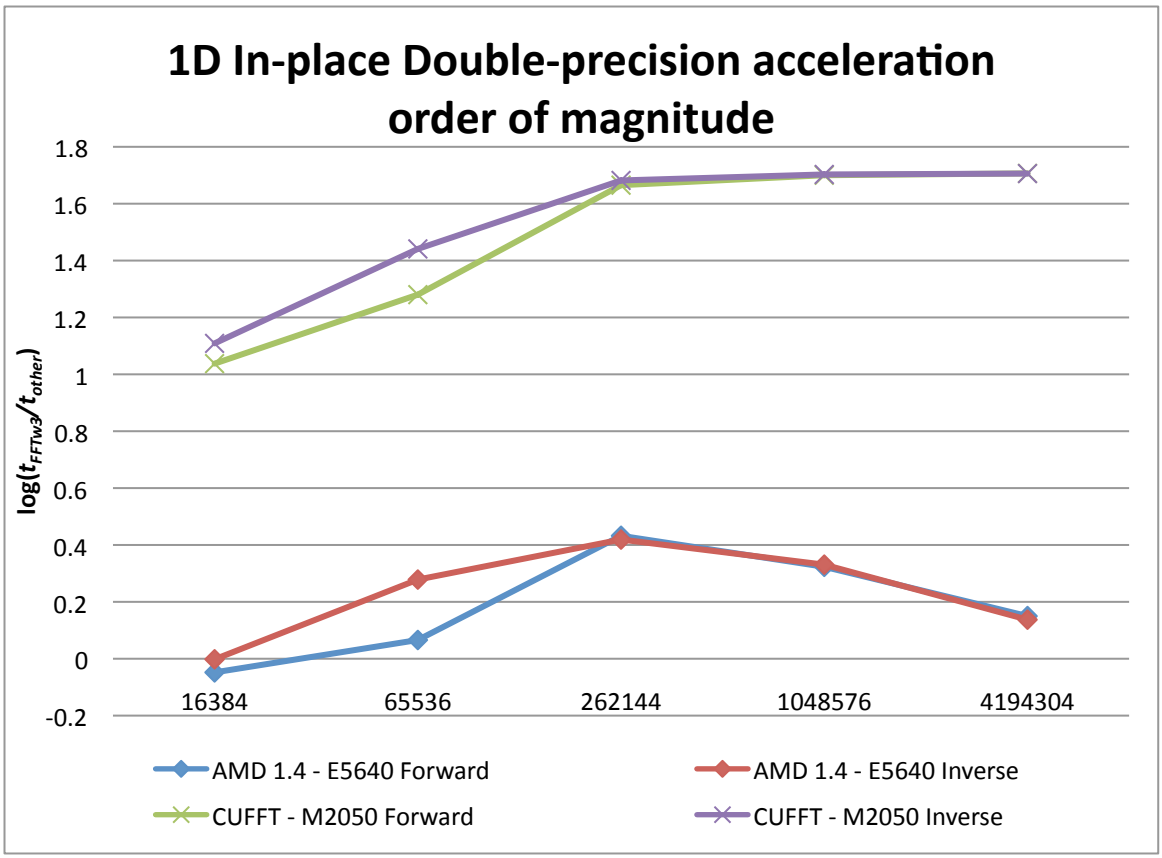
*2D In-place Complex-double*

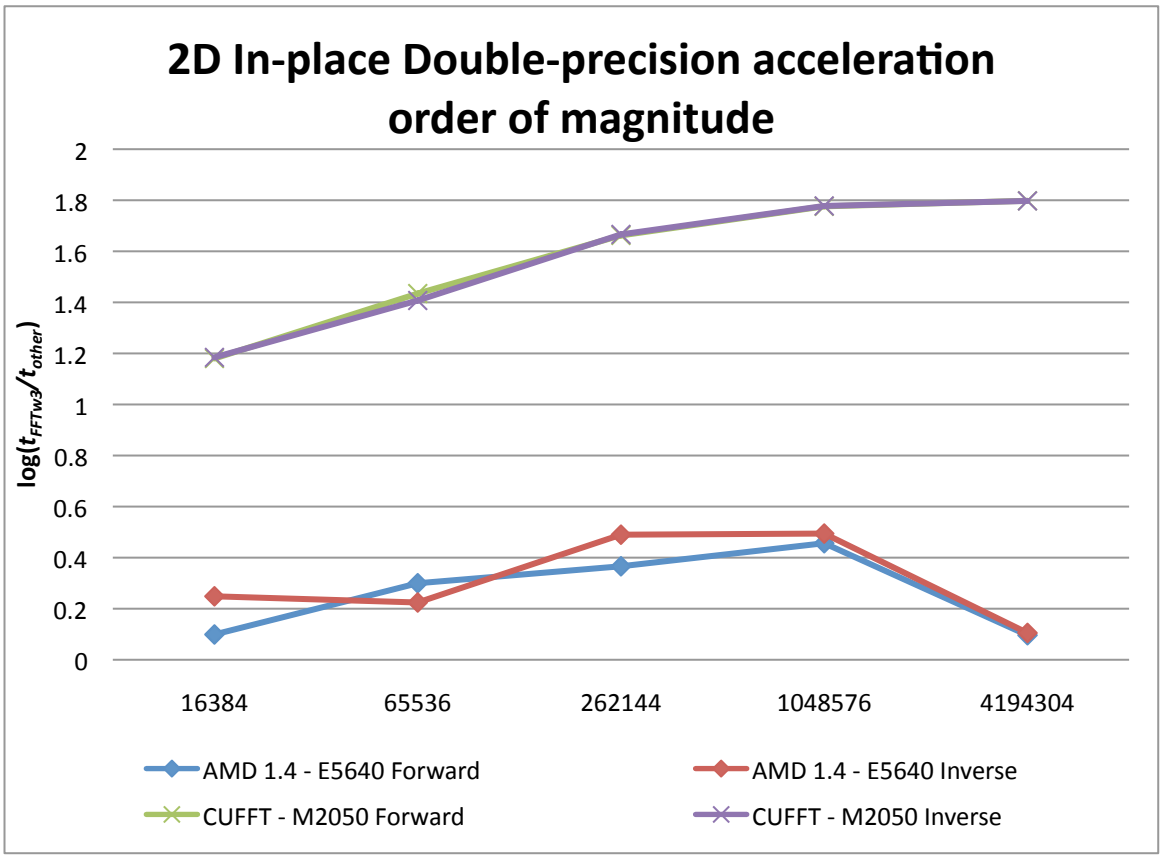
| | AMD 1.4 - E5640 | | CUFFT - M2050 | |
|---|---|---|---|---|
| ***NxN*** | Forward | Inverse | Forward | Inverse |
| 16384 | 1,256 | 1,772 | 15,136 | 15,294 |
| 65536 | 1,994 | 1,676 | 27,197 | 25,544 |
| 262144 | 2,324 | 3,090 | 45,946 | 46,367 |
| 1048576 | 2,861 | 3,121 | 59,724 | 59,946 |
| 4194304 | 1,248 | 1,274 | 62,796 | 62,710 |

APPML is in general faster than FFTw3 in performing FFT transforms, except for small FFT sizes. However, FFTw3 is single-threaded as opposed to APPML that is using all the four cores of the XEON E5640 CPU. There is a performance issue with the library and the HD6990 GPU and unfortunately there was no other AMD GPU model available to repeat the tests.




2D In-place Single-precision acceleration order of magnitude
log(tFFTw3/tother)
2.5
2
1.5
1
0.5
0
16384
65536
262144
1048576
4194304
AMD 1.2 - E5640 Forward
AMD 1.2 - E5640 Inverse
AMD 1.4 - E5640 Forward
AMD 1.4 - E5640 Inverse
CUFFT - M2050 Forward
CUFFT - M2050 Inverse
AMD 1.4 - HD6990 Forward
AMD 1.4 - HD6990 Inverse


1D In-place Double-precision acceleration order of magnitude
log(tFFTw3/tother)
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
16384
65536
262144
1048576
4194304
AMD 1.4 - E5640 Forward
AMD 1.4 - E5640 Inverse
CUFFT - M2050 Forward
CUFFT - M2050 Inverse




CUFFT exhibits high initilisation lag on the test system similar to the one identified in CUBLAS, due to known limitations of CUDA when Xorg is not running. APPML library initialisation lag is hidden in the plan creation.

# 4 OpenCL Evaluation

## 4.1 Introduction

The OpenCL API was initially developed by Apple Inc., in order to unite the different APIs for GPGPU in its operating system. It was later refines into collaboration with technical teams from NVIDIA, AMD, Intel and IBM, which led to the formation of the Khronos Compute Working Group and the first OpenCL specification, 1.0. OpenCL 1.0 was made publicly available in December 2008. Even though OpenCL was initially developed by Apple Inc., it currently supports all major operating systems, Windows, Linux and OS X.

Prior to OpenCL, the only APIs that could take advantage of GPUs regardless of vendors were the Sh and BrookGPU. They worked by using OpenGL or Direct X directly and they were soon abandoned and replaced by the offerings of the GPU vendors. Still, however, there was no single unified API and programming language to access heterogeneous systems of different compositions by different vendors. NVIDIA uses CUDA, AMD uses Stream and IBM provides support for the CELL/BE via its compilers.

The current specification of OpenCL is 1.1. OpenCL 1.2 has not been ratified, but is expected by the end of 2011.

## 4.2 Features

OpenCL is a parallel-compute platform designed to allow a host program to discover OpenCL supported devices and perform operations on them through its API. Such operations are functions called kernels. These kernels can be executed on any device, whether it is a CPU, GPU or CELL. That is a great advantage, since if we consider a host program using CUDA running a CUDA kernel on an NVIDIA GPU, if we need to execute such a kernel on an AMD GPU or a CELL processor we would have to rewrite completely the host program and the kernel. OpenCL incorporates Just-In-Time (JIT) compilation, meaning that a kernel gets compiled for a selected device on runtime, but can also store and load precompiled binaries.

OpenCL is a low-level API and is very similar to the CUDA Driver API by NVIDIA, enabling advanced control of a system. In example, different types of devices, such as a CPU and a GPU can be switched on runtime and used concurrently. However, some libraries such as *ViennaCL*, *PyOpenCL*, *JOCL* (Java-OpenCL), *CLyther*, *OpenCL Studio* and *OpenCL.NET* have such frameworks already developed and can provide high-level access to OpenCL functionality.

JIT, apart from the multi-device support, offers another clear advantage, kernels can be dynamically altered during runtime based on the circumstances. The ability of reusing stored precompiled binaries must be avoided, unless it is known in advance that the device used will be the same as the one the binary was compiled for. The reason is that there is no standard way to identify devices. In example, Intel's OpenCL driver assigned a different "Device Id" number than the AMD OpenCL driver for the processor of the system used for the testing (Intel XEON E5640).

OpenCL API, especially version 1.1, is rich in features when it comes to manipulating regions of memory and sharing workload between difference devices. The same kernel can be

executed by many different devices, operating on a part of the dataset automatically by assigning build-in work-group offsets and memory region segmentation.

OpenCL, due to the fact that it is relatively new, is lacking the abundance of tools and libraries found in other GPGPU APIs, but there is high momentum in porting CUDA or CAL libraries and tools to OpenCL. It is clearly lacking uniform BLAS, FFT and LAPACK libraries yet. Moreover, we found that OpenCL is lacking built-in complex numbers support which is commonly used in scientific computing. Some compatibility problems were also identified between the Windows and Linux version of OpenCL.

OpenCL allows the support of vendor-specific features. The advantage is that OpenCL can use all the capabilities of the device, but each vendor may introduce his own extensions which, if used, will cause kernels to fail on unsupported devices. Such an example is the extensions available for 64-bit precision support. Two different extensions exist, the *khr64* and *amd64*. Nvidia GPUs support the khr64 extension while the AMD CPUs support the amd64. Trying to compile a kernel that enables the khr64 extension on an AMD CPU will cause the compilation to fail.

Debuggers and profiling utilities are available for the major OpenCL implementations.

### 4.3 Conclusion

OpenCL is still in its infancy, but offers support for a wide range of heterogeneous devices, as well as for all major Operating Systems (OS). It is a viable solution to unite the numerous GPGPU APIs and the Just-In-Time compilation makes it easy to deploy programs in different systems. The performance of the API when it comes to GPUs is on par with the propriety implementations as long as the kernel is optimised for the specific device.

# 5 OpenCL support by manufacturer

## 5.1 Intel

Intel uses OpenCL 1.1 supporting all its modern CPUs that support the SSE4 instruction set.

## 5.2 NVIDIA

NVIDIA implements OpenCL 1.0 supporting all its modern GPUs. However there is no support for its ARM system-on-chip (SoC), such as Tegra and Tegra 2. The latest NVIDIA GPU display driver, 180.13, offers OpenCL 1.1 support for the Fermi-class GPUs (GTX4xx, GTX5xx, Tesla 20xx).

## 5.3 AMD

AMD is using OpenCL 1.1 supporting all its modern CPUs, GPUs and APUs.

## 5.4 VIA

VIA supports the integrated GPU (IGP) ChromotionHD 2.0 of the VN1000 chipset through OpenCL 1.0.

## 5.5 IBM

IBM is using OpenCL 1.1 and supports the CELL broadband-engine (BE) processor. This processor is an alternative to heterogeneous high performance computing and is already used by the scientific community for algorithm acceleration.

## 5.6 ARM

Currently ARM supports one GPU device via OpenCL 1.1, the Mali T604.