Betatron tune measurement in the LHC damper using GPU

Frédéric Dubouchet

January 18, 2013

Abstract

This paper study a possible futur implementation of the betatron tune measurement in the Large Hadron Collider (LHC) at the European organization for nuclear research (CERN) using General Purpose Graphic Processing Unit (GPGPU) to analyse data from the damper acquisition. It start by describing the present hardware and the future possible implementations using the Accelerating Damper Transverse (ADT) acquisitions and describe the Graphic Processing Unit (GPU) computing. The ADT data have to be processed to be able to extract the betatron tune. To get the tune the method used is to move the signal from temporal domain to frequency domain using Fast Fourier Transform (FFT) on GPUs. We show that it is possible to achieve one order of magnitude faster FFTs on GPU than what is done using Central Processing Unit (CPU). This makes per bunch FFT computation and betatron tune measurement possible.

Acknowledgements

I wish to thank CERN and Haute école du paysage, d'ingénieurie et d'architecture de Genève (Hepia) to have made this master thesis possible. I also wish to thank Dr. Andy Butterworth and Dr. Ing. Erk Jensen who supported me on the choice to make this master thesis, Dr. Wolfgang Höfle for suggesting that I use GPU in this particular field and supervising. Dr. Valuch for the assistance on the damper and for all the ideas. Dr. Rama Calaga for the help on the mathematics of Singular Value Decomposition (SVD) and hints. Finaly, I wish to thank Pr. Paul Albuquerque who was my professor and supervisor during the whole thesis and his assistant Pierre Kunzli.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 State of knowledge

In a particle accelerator, the charged particles circulate around the ring and oscillate due to the magnets and the accelerating structures. The accelerating structures, in the LHC supra-conducting cavities apply a strong electrical field that oscillates at the RF frequency (f_0) to particles in order to collect and accelerate particles in bunches inside a frequency bucket.

The particles inside a bucket are oscillating longitudinally along the ring and transversally in the vertical and horizontal plane. The longitudinal oscillations are damped by the beam control system. But the transversal oscillations must be damped by a separate system : the ADT[12, 18].

One of the key parameters of the accelerator is the betatron tune. The betatron tune, Q, is the quotient of the betatron oscillation and the particle frequencies.

$$f_\beta = Q * f_0$$

This value allow us to check if the particle beam is stable and don't reach any dangerous instabilities.

1.2 Tune measurement in the LHC

In order to measure the betatron tune in an accelerator we use Beam Position Monitors (BPMs). These monitors are able to measure the position of the beam in the vacuum chamber.

In the present setup Beam Instrumentation (BI) group is using their diode-based base-band-tune (BBQ) [4] system to acquire the tune over a certain number of machine turn (256 to 128'000). This can work as a passive instrument or as an on demand system by exciting 12 bunches in the beam with the tune kicker (MKQA). ADT has also been used for tune measurement excitation[11].

In normal operation, as the ADT is active, it is difficult to have a good picture of the excited bunches and make a fine tune measurement : the oscillations created by the MKQA are damped by the ADT. There have been studies to disable the ADT for a certain number of bunches in order to get a better tune measurement[13], but this may not be sufficient.

1.3 Proposed system

The ADT also have BPMs and these can have per bunch measurement[17]. This could allow a much precise measurement. But due to the high among of data to be processed (estimated to 640 mega bytes per seconds for each BPM) dedicated hardware is needed to compute the correct tune[14].

In order to be able to apply direct correction to beam oscilliation the betatron tune has to be measured at a high frequency, this has been estimated by BI to be between 5 and 10[Hz], once every 100 to 200[ms].

During the 2012 normal operation of the LHC, data has be acquired using the ADT acquisition system and data processing techniques has been tried to asses the modification that will be needed in order to make a reliable betatron tune measurement at a reasonable rate[14].

The current VMEbus implementation has some serious issues in particular the bus is quite slow the data rate of the bus is around 40 megabits per seconds. The data need to either be processed on the acquisition board or to be off-loaded to another computer using the serial link available on the board[3].

1.3.1 DSP on VME board

Digital Signal Processors (DSPs) are able to compute FFTs at high rate and these are used already in the machine at different places to make high speed feedback loops. The question is : is it fast enough to compute all the FFTs needed, DSPs are two orders of magnitude slower than GPUs. We also would have to develop a completely new system in order to be able to use them, in fact we don't have DSPs in the present ADT. The cost of development and the complexity of the deployment should also be studied.

1.3.2 FPGA pre-processing on VME board

Like in the approach using DSPs on VME boards, the question of computing power is still unsolved. We already have in house experience and we already have a lot of Field-Programmable Gate Array (FPGA) installed in the ADT. But if we want to do it we will have to create a new card able to replace the existing one and to make the computation. This mean create a potential problem in the existing setup. The cost is also to be studied we have to develop a new card, test it and install it in the LHC. Also the number of card and FPGA is not well understood, it could be anywhere between 4 and 3000.

The biggest issue for FPGA computing been the fact that you need something like 100M bytes of memory to store the temporary values for computation of only one beam plane.

> 96'265'920 Bytes from 2024 points times 2 for complex times 2880 the number of bunches times 4 size of float times 2 because you need to double buffer.

This is far too much memory to be access simultaneously by a single FPGA. In case we want to make these computations on FPGA it would cost a lot of money in hardware and development.

1.3.3 GPU off-board computing

This solution can be integrated easily in the present setup. The present acquisition cards already have a digital output and could be used to transfer the data in another crate that could do the computations. The GPUs are cheap (compare to the price of developing a new VMEbus card) and easily scalable. The GPU should have largely enough computing power to be able to make the FFTs. Another interesting aspect of this solution is the ability to test it using CPU.

1.4 Problem definition

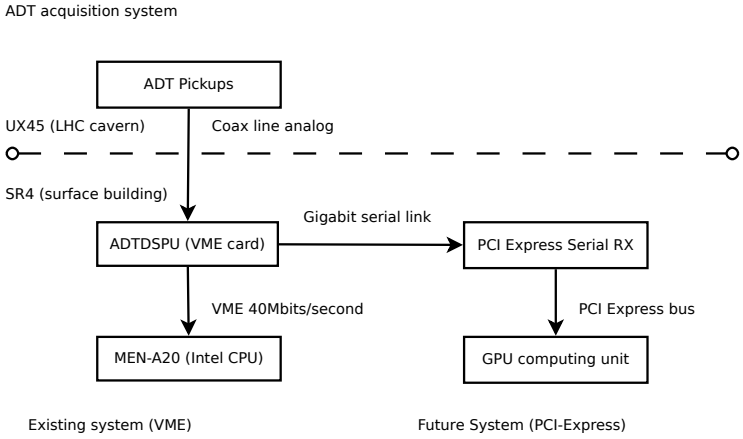
Show how to implement a GPU based system that can deliver a tune value for each beam and planes at a frequency that allow the system to be responsive enougth to allow tune correction to be applied automatically.

1.4.1 Algorithm

We need the tune frequency and we have the tune position per bunch, we have to calculate the FFT to move from time to frequency domain. Then we need to identify the tune in the transformation.

1.4.2 Hardware

Figure 1.1: ADT acquisition hardware



Per bunch position mesearment has to be avalable to the system for each beam and each plane. This should be provided from the ADTDSPU card and has to be transfered through a serial link to the CPU/GPU crate for computation.

We need a card in the CPU/GPU crate to unserialize the data and transfer them to the GPU memory. It may be possible to copy from the acquisition card directly to the GPU memory.

And finally fast enougth GPU to process the data. The number and the type of card should be looked at. The possibilty for expantion should be kept as the possibility to implement other algorithms.

1.4.3 Timing

Acording to BI we have to provide the tune measurement betweeen 5 Hz and 10 Hz. This mean that the transfer and the computation has to be made in less than 200 ms.

At a higher frequency because of the acquisition frequency (11 kHz) the precision may be unsufficient (NyquistShannon sampling theorem).

Chapter 2

Graphic Processing Unit

2.1 Introduction

General purpose programming on graphical processors is a new field with a growing community of practitioners. Until recently only proprietary interfaces existed to harness the power of these chips. With the arrival of the Open Computing Language (OpenCL) a new open interface has appeared, and with it a hope for a unified, simple and portable framework for general purpose computing on heterogeneous hardware.

2.2 OpenCL

The OpenCL is the open standard for GPGPU programming. It is maintained by the Khronos group, and was initially proposed by Apple. Many companies of the industry are members of the OpenCL Working group: Altera, AMD, Apple, ARM, Broadcom, Codeplay, DMP, EA, Ericsson, Fixstars, Freescale, Hi corp, IBM, Intel, Imagination Technologies, Kestrel Institute, Kishonti, Los Alamos, Motorola, Movidius, Multicoreware, Nokia, NVIDIA, OpenEye, Presagis, Qualcomm, Rightware, Samsung, ST, Symbio, Texas Instruments, The University of West Australia, Vivante and Xilinx.

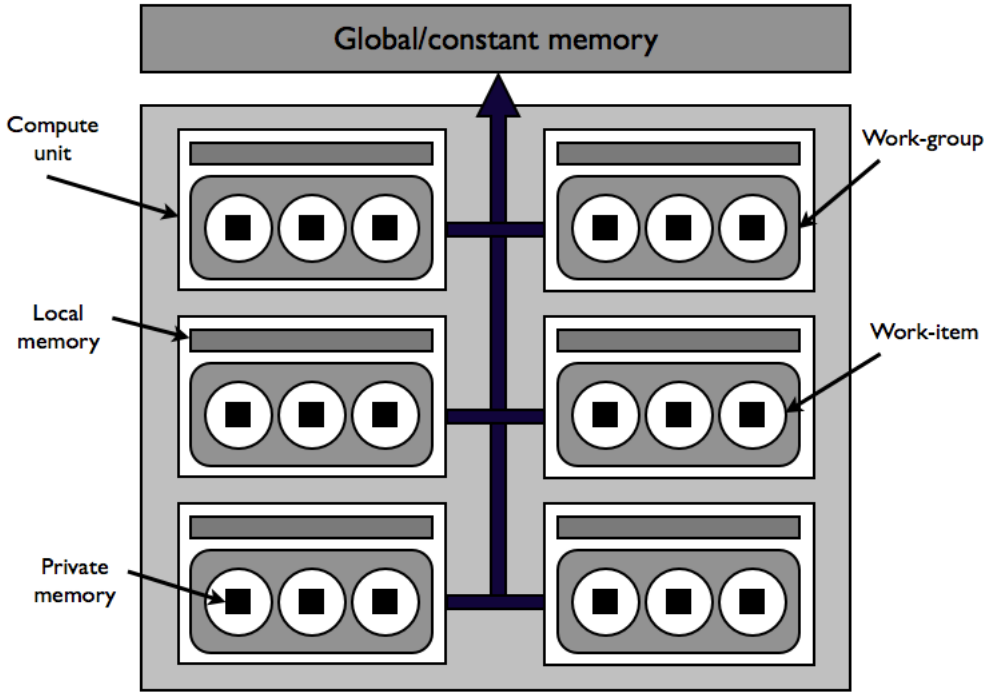
> OpenCL™is the first open, royalty-free standard for cross-platform, parallel programming of modern processors found in personal computers, servers and hand-held / embedded devices. OpenCL greatly improves speed and responsiveness for a wide spectrum of applications in numerous market categories from gaming and entertainment to scientific and medical software.
>
> The Khronos Group is a not for profit industry consortium creating open standards for the authoring and acceleration of parallel computing, graphics, dynamic media, computer vision and sensor processing on a wide variety of platforms and devices. All Khronos members are able to contribute to the development of Khronos Application Programing Interface (API) specifications, are empowered to vote at various stages before public deployment, and are able to

accelerate the delivery of their cutting-edge 3D platforms and applications through early access to specification drafts and conformance tests.

2.2.1 Architecture

OpenCL is not really a language but an API and a kernel language derived from C99 that is compiled for, and executed on the device itself. It defines an abstract hardware architecture onto which the real hardware is mapped. Every piece of hardware (CPU, GPU, others) is called a device. This device is divided into work groups, each of which is further subdivided into work items.

Figure 2.1: OpenCL Device Model[16]



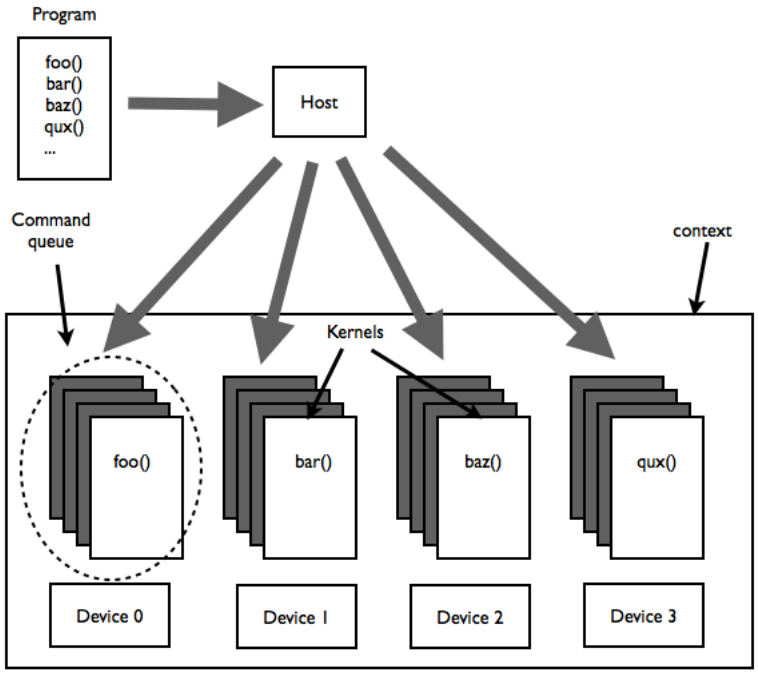
2.2.2 OpenCL API

The OpenCL API is composed of two parts: The host code that runs on the CPU which runs kernel handling event synchronization and manages the memory buffer, and the device code, which is a kernel language based on the C99 specification.

The host API is written in C but there is a C++ interface provided by Khronos, which is the interface used in this paper. Several other wrappers for other languages are also available, such as Java, C#, Python, and even javascript with something called WebCL (also specified by the Khronos group).

The device code the OpenCL kernel code is based on is C99, adding some extensions like vectorization, image access, and linear algebra functions.

Figure 2.2: Kernel Distribution among OpenCL-compliant devices[16]



The workload is itself specified in terms of context, program and command queues. The context contains the parameters of the current problem and manages the devices available for computation, as well as the programs and the command queues. The program consists of the set of kernels which are constructed from OpenCL code. The context is responsible for dispatching these kernels into command queues. Each command queue is a set of commands that can be executed in any order, or simultaneously. A command queue is limited to a single device.

Events fired by devices are also available for triggering specific actions in case a finer synchronization mechanism is needed. Once the device has finished, the host receives and processes the output data via the buffer API.

2.3 Other GPGPU languages

For the purpose of completeness we present also the other general purpose computing languages used on GPU, shader languages will also be briefly mentioned they are not a real general purpose computing language but are still in use.

2.3.1 CUDA

Compute Unified Device Architecture (CUDA) is the most used interface for general purpose GPU programming. It was developed by NVIDIA and has been used for many years and is still very much in use. This only works on NVIDIA cards, and will not run on CPUs. Since it is a proprietary language,

NVIDIA is in total control of the API and the development kit. It is interesting to note that NVIDIA is talking about releasing the interface of the compiler to academics.

The abstraction level in CUDA is less strong, that is, the language is closer to the architecture. This means it could potentially offer more optimization possibilities to the programmer. We will see that this is not a key element as the compiler is able to reach this level of optimization without having to compromise code readability.

CUDA has its kernel and GPU code directly embedded inside the C/C++ code. In OpenCL the code is separated from the C/C++ host code. This can lead to incompatibilities with C++ as valid C++ code will be flagged as invalid by the CUDA compiler. This problem does not exist for OpenCL as the compiler is a separate entity and the code is fed to it via the API. This design code from the fact that OpenCL can have many target that won't produce a different compiled code. This means that compiled OpenCL code is not portable across platforms and you should always keep the code non compiled in the executable.

CUDA offers a full suite of libraries and tools that are presently not available in OpenCL. Among these are performance tuning tools and libraries for matrices and FFTs. Recently many implementations of common algorithms have started to appear for OpenCL, and unlike with CUDA these are directly available to any computing device.

2.3.2 DirectCompute

This is the Microsoft interface, highly tied to the Windows platform. DirectCompute is part of the DirectX API, the game development tools for Windows. It works with DirectX 10 and 11 under Windows Vista and Windows 7.

DirectCompute shares a lot of concepts with both CUDA and OpenCL, and is presently working with both AMD and NVIDIA graphic cards.

2.3.3 Shader Languages

Shaders are the steps that the graphic card has to make before rendering to the screen. There is mainly two steps the vertex shader, also called vertex code and the pixel shader, also called fragment code, vertex shader is manipulating the data as point in space, as vector and the pixel shader the final rendering as pixels on screen.

There are many different shader languages. The main ones are: Cg, the NVIDIA neutral shading language one of the first GPGPU language widely used. High Level Shader Language (HLSL), the Microsoft version tightly coupled with DirectX, and therefore easier to use on a Microsoft platform (XNA). OpenGL Shader Language (GLSL), the Open Graphic Language (OpenGL) version, that is supported widely but not implemented completely on all platforms.

Before the general purpose graphic processing languages appeared this was the only way to access the power of the graphic card, and this is still much in use today. As it was developed over many years, this is a very stable technology and there are many examples and libraries using theses languages.

Shader languages are oriented toward graphics programming and are therefore not well-adapted to some of the algorithms we could want to implement. They are made to handle vectors of four components: position, color, normal.

This can be quite inconvenient. This is also their strength as this is what the GPU was originally built for. Communication between host and GPU tend to be messy at best, especially with non-graphical structures.

2.4 Conclusion

The system is going to run on dedicated CERN computers that are under Scientific Linux CERN (SLC) so the fact that it has to run under Linux is a must. The ability to be able to test the whole system on CPU and to compare the processing speed is also a key factor. The system should be able to be expanded and not be tight to any hardware provider. OpenCL is then the only API that can offer all theses .

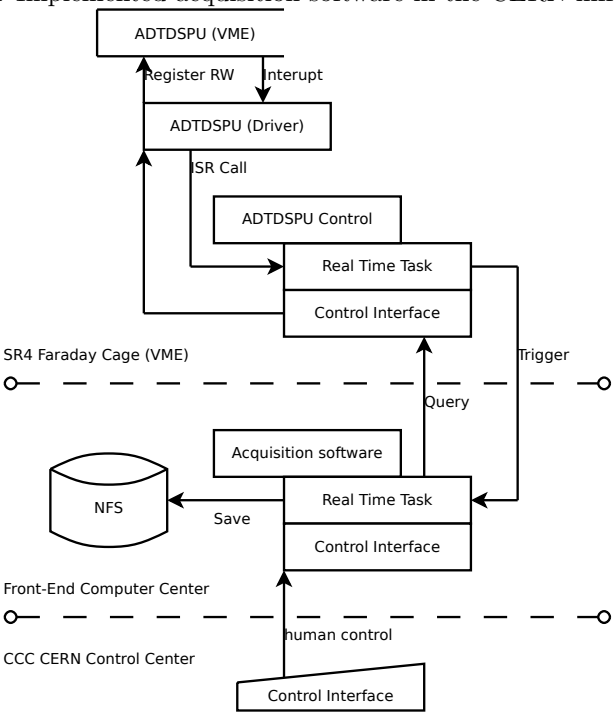
Chapter 3

Results

3.1 Implemented System

The implemented system consist of tree pieces of software. The software controlling the acquisition card in the machine. An acquisition software that run on a normal front-end Linux machine that is taking data during the Machine Developments (MDs). And an analyzing software. The analyzing software is in fact modular and has a version that has to run on a GPU enable machine to use the GPU to compute the FFTs.

Figure 3.1: Implemented acquisition software in the CERN infrastructure



In the final version the software will be merged in single executable that

should run on the GPU-enabled machine. This solution has been put into place because the present hardware is still in development and there is no way of acquiring the full 2880 bunches of the machine, the hardware is receiving the acquisition data but the VMEbus is not fast enough to transfer it to the CPU.

3.1.1 ADTDSPU control software

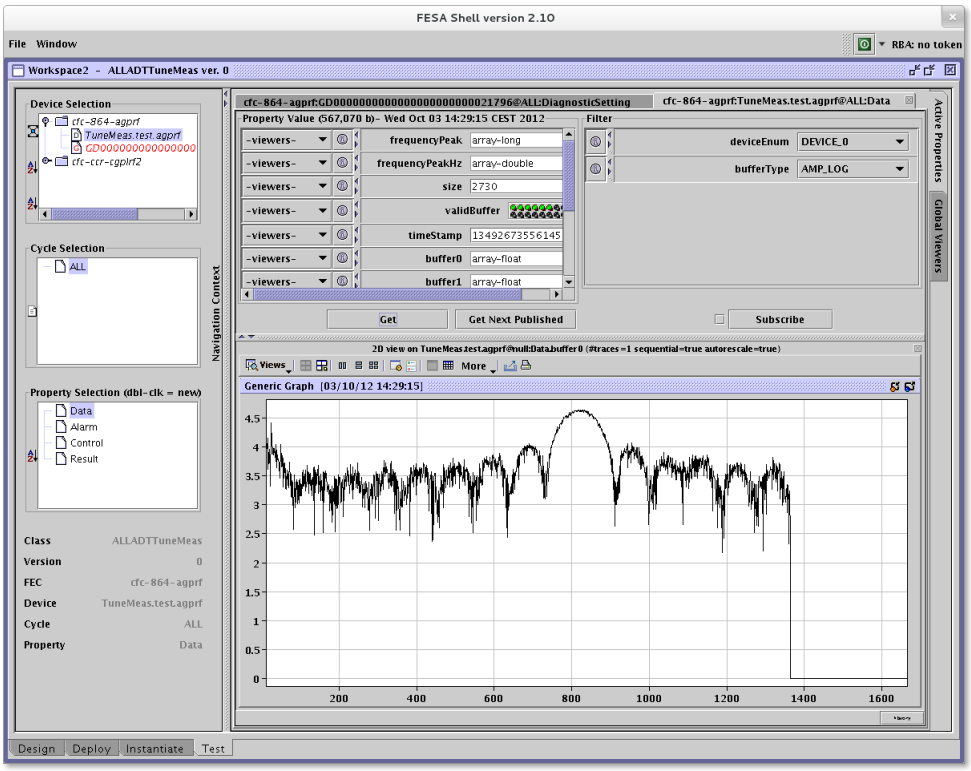
The first layer that was needed is a driver that can control the VMEbus card and forward the interrupts. This is using standard driver framework from the Control (CO) group at CERN.

Then the normal FESA environment is used to develop a higher level software to control the card. This particular card need real time task to react to interrupt coming from the hardware to inform when new acquisition is ready to be read.

3.1.2 Acquisition software

The acquisition software was used to check that the idea of getting the tune out of the DSPU card was doable and being able to log the acquisition to file to be checked and processed separately.

Figure 3.2: Tune acquisition software interface



It uses Control Middleware (CMW), a library used at CERN to communicate between different layers of the accelerators control software, to connect to the

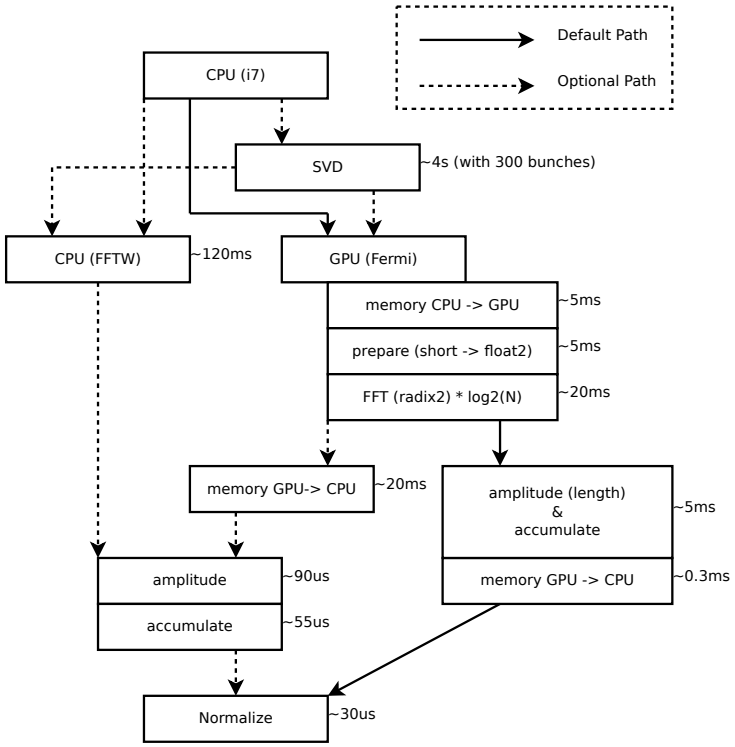
the ADTDPSU control software and get the data when they are published (at interrupt time).

It is then able to compute the acquisition FFT using FFTW and display it to the operators in the CERN Control Center (CCC) where all the eight accelerators of CERN are controlled. On the interface you can decide witch type of graph you want to see and also enable saving the data into files to be processed by the data analysis software.

3.1.3 Data analysis software

The data analysis software is a set of modules that can be enabled or bypassed to test the the usefulness of an algorithm. As we have precise time allocated to make the full computation this allow to test the different modules.

Figure 3.3: Time flow with different implementations and with 3000 bunches of 2048 points each.



The data is first loaded from the files that were written by the acquisition software. Then it is filtered by a notch filter as described in the notch section 3.2. And finally to the different algorithms as shown in the figure 3.3.

First optional step is SVD this as described in Rama Calaga PHD thesis[5] and Wolfgang Höfle[14] should reduce noise in the signal result are seen in section3.5.

Then come the the FFT either using the GPU or using the CPU, actually you could use OpenCL on the CPU and test the whole OpenCL path. The

different path are due to memory copying, in the case of computation made on the GPU you have to move the memory data from the CPU central memory to the GPU memory. this will described in section 3.3.

To have a clear image and to combine the real and imaginary part of the FFT we use the amplitude. It has been validated in the acquisition software as been the best metric, this could be changed as will in the final version. The amplitude is described in section 3.4.

As a final value is wanted we do the accumulation of all the bunches, it will give an average spectrum of the whole machine.

The normalization step is just present for displaying the spectrogram (see section 3.7) and will not be needed for the final version.

3.2 Notch filter

A notch filter is used to cut the low frequencies, this doesn't change anything on the high frequencies, it won't be used in the final version. For every sample it takes the present sample and subtract the next element.

$$y_{[n]} = x_{[n]} - x_{[n+1]}$$

This is a differential filter, as we are only interested into the higher frequency this has no incidence on our results and allow for a better visualization of the spectrograms.

3.3 FFT

The Fourier transform is a mathematical operation that moves a function from a temporal domain to a frequency domain. In our case we are talking about Discreet Fourier Transform (DFT), and in particular, the FFT.

3.3.1 Definition

The DFT is a discrete transform. It transform a function into another, which is called frequency domain, or simply DFT, of the original function.

$$x_0, ..., x_{N-1} \in \mathbb{C}$$

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi k \frac{n}{N}}$$

In order to compute the DFT one must compute N number of values N times. The complexity is thus

$$\mathcal{O}(N^2)$$

The most commonly used FFT algorithms are based on a divide-and-conquer approach similar to the algorithm of Cooley and Turkey[6]. The computation of a DFT of length N is done by splitting the input sequence into a fixed small number of subsequences, compute their DFT, and assemble the outputs to build

the final sequence. If the split and assembly are linear in time the complexity becomes

$$\mathcal{O}(N \log(N))$$

3.3.2 FFTW

FFTW is an implementation of the DFT that adapt to the hardware in order to maximize performance[9]. It is widely regarded as one of the fastest implementation of FFT on CPU.

It was selected as a reference for our the implementation, as OpenCL can be run on directly on a CPU is it possible to compare the time performances between the OpenCL code running on CPU and the FFTW version see section3.6.

It is under GPL license, can be purchased from the Massachusetts Institute of Technology (MIT) for commercial purposes and is used in many commercial and scientific package and software. As an example it is used in the MATLAB.

3.3.3 FFT with OpenCL on GPU

The FFT version used in the software is derived from Eric Bainville's version. This is the reference implementation of FFT on OpenCL and is now the version distributed by Apple[2]. For the sake of simplicity we just used the radix2 version of the implementation but it is probably possible to gain even more time by using a higher radix.

On GPU the OpenCL result is the same than the one using FFTW with the advantage to be faster on recent GPU. The kernel is called a number of time equal to the size of the vector and then 11 times in the case of 2024 points ($\log 2^{11} = 11$). A second dimension is used to pass the multiple vector.

3.3.4 FFT with OpenCL on CPU

As OpenCL is also able to run on CPU we used the same code than the one used on GPU on CPU and got the same results. It is interesting to note also that the time is very similar to the FFTW version.

3.4 Amplitude

The amplitude is the length of the complex vector composing the FFT, it is equal to the euclidean norm.

$$\mid x + iy \mid \equiv \sqrt{x^2 + y^2}$$

There is different ways to compute the norm of a 2-dimensional vector in OpenCL and all of them seams to work equally on our test.

3.5 SVD

It was suggested by Rama Calaga[5] to use SVD in order to diminish the noise and improve the visibility of the tune in the frequency domain. The idea is to

take the raw data and use the multiple bunch as a second dimension of our SVD then apply the algorithm and suppress the values in the Σ matrix that are off a certain scale, finally recompose the M matrix and hopefully have less noise in it.

$$M = U\Sigma V^T$$

This was first tested on generated data by Wolfgang Höfle[11] unfortunately we have only 6 bunches in our dataset per acquisition so it is not possible to make this work with the present set up. The Σ matrix is only 6 by 6 values so it is not easy to suppress values in notable fashion.

Taking more than one acquisition can at least give some idea on the speed of processing. So SVD was implemented using the GNU Scientific Library (only double precision float for SVD). The problem is that correlation between beam of the same acquisition make the process a lot faster than uncorrelated bunches.

Table 3.1: SVD with vary heavily with correlation of bunches

Bunches	Acquisitions	Time
5	20	0.15 s
4	25	0.30 s
2	50	2.04 s
1	100	16.9 s

So it is very difficult to estimate the time the computing with the real data would take, but we have to keep in mind that these computation where done using double precision as single would probably be enough, these computation were only done using a single thread (SVD is quite difficult to parallelize) and only done on CPU.

There is a way to make these computation on GPU[15] and the estimation are around 5 times better than on CPU. As soon as we have better data set (witch will require hardware upgrade) we can further investigate on this.

3.6 Performances

Computation were made by accumulation to simulate the number of bunches that could be present in the final version (2880). As 6 bunches were acquired we used 500 successive acquisition to make it close to the 2880 and have a safe margin.

Different strategy were used to try to improve performances as shown in figure3.3. But also use of pipelining (not used on the figure as it is difficult to estimate time when using pipelining) and test on different hardware.

The hardware used was mainly the Tesla M2090 based on a Fermi chip this chips has more cores than a CPU and we can see an improvement in respect of computing speed as shown on the table 3.4. The table 3.2 show the characteristic of the Tesla Fermi card family.

Table 3.2: NVIDIA Fermi hardware available on the market[7]

Features	Tesla M2090	Tesla M2075
Peak double performance	665 Gflops	512 Gflops
Peak single performance	1331 Gflops	1030 Gflops
Memory bandwidth (ECC off)	177 GB/sec	150 GB/sec
Memory size (GDDR5)	6 GB	6 GB
CUDA cores	512	448

But if we have a look on the different card that are available today on the market we see that the new generation should have even better result and are available today. These card show around 5 times the number of CUDA core and around 10 times the number of floating-point operations per second (flops) as shown on table 3.3.

Table 3.3: NVIDIA Kepler hardware available on the market[7]

Features	Tesla K20X	Tesla K20	Tesla K10
Peak double performance	1.31 Tflops	1.17 Tflops	190 Gflops
Peak single performance	3.95 Tflops	3.52 Tflops	4577 Gflops
Memory bandwidth (ECC off)	250 GB/sec	208 GB/sec	320 GB/sec
Memory size (GDDR5)	6 GB	5 GB	8GB
CUDA cores	2688	2496	3072

3.6.1 Pipelining

To improve the performances one of the options to try is to remove all waiting time between the different operations on the GPU, as shown on the figure 3.3 the different operations done on the GPU that are done sequentially could be pipelined.

Pipelining mean that you won't wait for all the sub-operations to be finished before starting the next one. The first operation is copying the memory from the CPU to the GPU this takes a certain time but as soon as some of the data is copied the computing could start.

In OpenCL the different commands are queued and this command queue can be flushed. to make the pipelining work the only thing to do is to only flush at the end when all the computing has been queued.

In our case this mean that the copying of the data from CPU to GPU the preparation of the data the FFT itself the amplitude computation, the accumulation and getting back the values to the CPU is queued together in one go.

That allow us to have a 10% improvement on performances, but it is difficult then to estimate the computing time of individual steps so the values shown on figure 3.3 are without pipelining.

3.6.2 Memory

Copying memory from and to the GPU can be expansive time wise, as shown on figure 3.3 copying 3000 times 2024 values in short (2 bytes) take around 10 ms.

This is the reason why it is important to make the accumulation on the GPU because after the FFT computation there is 3000 times 2024 complex values in float (8 bytes). It would have cost at least 40 ms to copy these values back to the CPU. The 20 ms shown on figure 3.3 correspond to half the values because we can cut half of the result as the FFT computed it in real only so the result is mirrored.

3.6.3 Time

Table 3.4: Speed for 3000 batch of 2048 points

Device	Type	Threads	Speed [GHz]	pipeline	Time [ms]
Xeon X5650	FFTW	12	2.67	N/A	291
Xeon X5650	OpenCL	12	2.67	enable	284
Xeon X5650	OpenCL	12	2.67	disable	288
i7-3720QM	FFTW	8	2.6	N/A	310
i7-3720QM	OpenCL	8	2.6	enable	272
i7-3720QM	OpenCL	8	2.6	disable	273
Tesla M2090	OpenCL	512	1.3	enable	35
Tesla M2090	OpenCL	512	1.3	disable	37
GeForce 650M	OpenCL	384	0.9	enable	355
GeForce 650M	OpenCL	384	0.9	disable	365

Time performances were computed using the timing library from boost [1] on different hardware, GPUs and CPUs, with and without pipeline enable as shown on table 3.4.

The time performances between FFTW and OpenCL on CPU is very close, so the radix 2 implementation used must be good and we can hope the performances on GPU could then be fairly compared with FFTW.

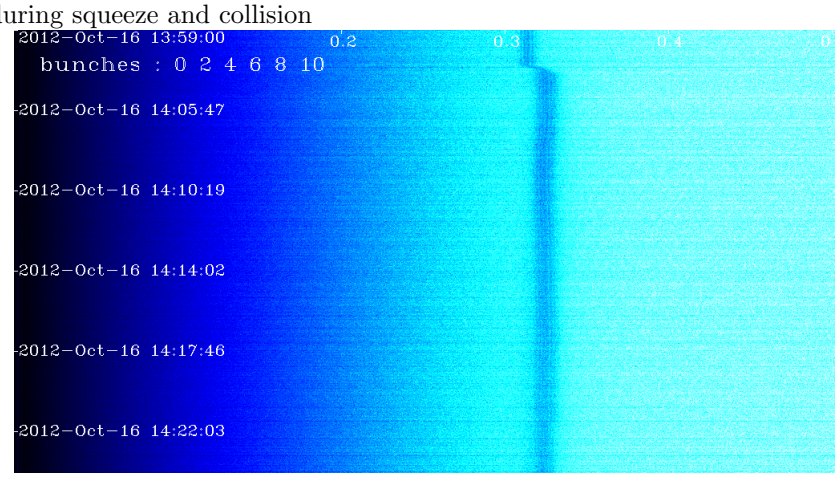
On a dedicated GPU like the Tesla M2090 the performances are around 10 times better than on a modern CPU. This is very encouraging and means that on last generation hardware we should be able to achieve even better performances.

3.7 Spectrogram

A Spectrogram is a time-varying spectral representation of a signal. the signal is transformed via FFT from time domain to spectral domain, each transformation makes a line in this case and is tagged with the time of the acquisition. the amplitude is used to have a single representation of both real and imaginary part of the result.

As we do a normalization by acquisition at the end of the computing we have a representation of the highest value with highest color (white) and if the amplitude is respectively weaker we have a darker representation of the color (black). To have a finer grain in representation intermediate color was chosen in this case blue as you can see in figure 3.4 4.3 and 4.1.

Figure 3.4: Spectrogram with ADT off on the 16 October 2012 on vertical beam 1 during squeeze and collision



The Spectrogram allow us to clearly see a mark in the signal that correspond to a region where the tune is suppose to be.

Chapter 4

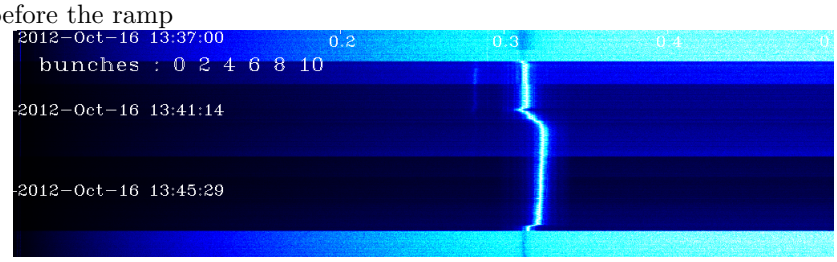
Discussion

4.1 Observation

4.1.1 Without damper

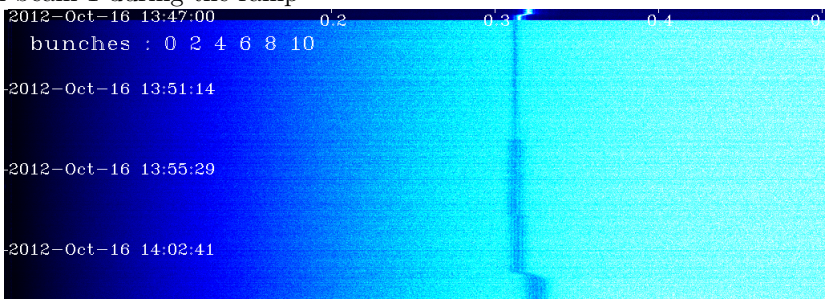
Clear view of the tune

Figure 4.1: Spectrogram with ADT off on the 16 October 2012 on vertical beam 1 before the ramp



4.1.2 With damper

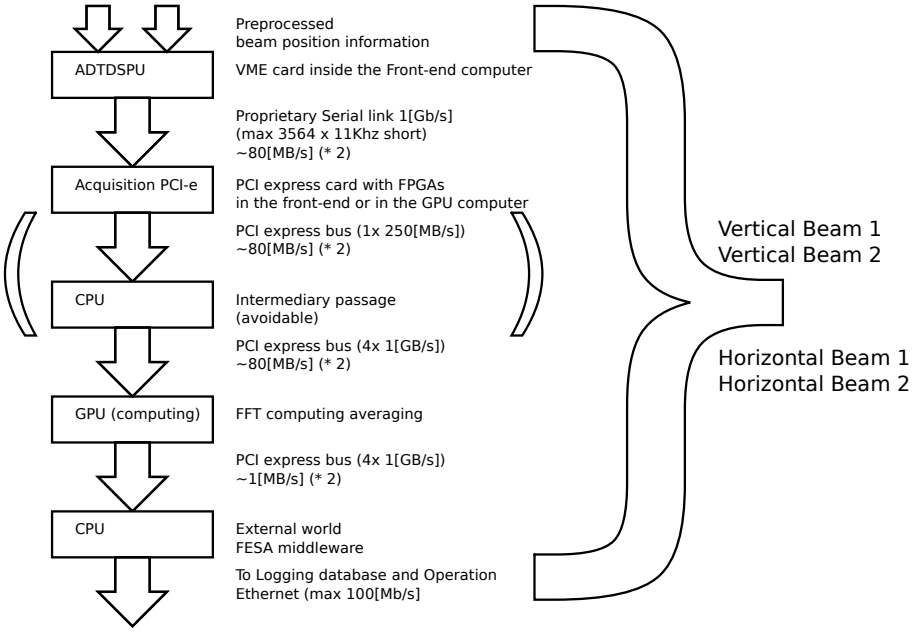
Figure 4.2: Spectrogram with damper working on the 16 Octover 2012 on vertical beam 1 during the ramp



The tune is inverted speak about cite the hofle paper mathematic morphology more processing needed but should be quite low.

4.2 Data flow

Figure 4.3: Acquisition data flow



4.3 Hardware

4.3.1 ADT Aquisition boards

4.3.2 Serial link interface

4.3.3 GPUs

4.4 Software

4.4.1 Drivers

4.4.2 OpenCL

4.4.3 Front-end

4.5 Estimated Cost

Chapter 5

Conclusion

This project looks like a nice place to try using GPUs in accelerators. The possibilities are promising and the gain for the stability of the LHC could allow more physic time. GPUs could prove to be useful and be used in other places in accelerators where computing power is needed.

Chapter 6

Annex

6.1 Estimation of the amount of data

Presently the LHC is working with an interval of 50ns between bunches this correspond to a bunch every 10 buckets. But the Operation (OP) is planning to move to 25ns bunches spacing this would mean 5 buckets between bunches. With the RF frequency (f_0) we can compute the number of acquisitions per seconds.

$$for\ 50\ ns\ : \frac{400.789M}{20} = 20'039'450 \leq 2^{25}$$

$$for\ 25\ ns\ : \frac{400.789M}{10} = 40'078'900 \leq 2^{26}$$

This represent the amount of data for one pickup (BPM), in the case of ADT we have two of them per beam and per plane so as the LHC has two rings and for each ring there are two transversal plane and there are two pickups per plane. This means we still have to multiply this value by eight.

$$for\ 50\ ns\ : 2^{25} * 8 = 2^{28}$$

$$for\ 25\ ns\ : 2^{26} * 8 = 2^{29}$$

As FFTs on GPUs start to be be faster than CPUs around 2^{15} acquisitions it seems interesting to study this kind of system to compute the betatron tune.

6.2 Measurement with the ADT

In order to check the feasibility of the system and to have a good prototype the first test will be to excite some of the bunches and acquire the betatron tune using the ADT during the end of 2012 run[8].

A piece of software has been developed that will acquire the bunch by bunch acquisition and compute various algorithm on the data using the CPU and the FFTW library in the CERN infrastructure using CO group control system and the OP group infrastructure.

6.3 Experimental Set-up

6.3.1 Hardware

The experimental set up is not presently able to acquire more than a certain number of bunches due to memory limitation 16k and interupt frequency so during the MDs only 6 bunches were acquired by bunch by planes.

6.3.2 Software

6.4 FFT

Used the algorithm described here[10].

6.5 SVD

Used the GNU scientific library.

6.6 Machine development sessions

Using the ADT BPMs we acquired data in the machine during 3 independant MDs. Most of the data taking was done in parallel to other normal LHC operation or during ADT dedicated MD time.

6.6.1 First session

Night session of the 11 october 2012.

6.6.2 Second session

Parasitic session of the 16 october 2012

6.6.3 Third session

Ramp acquisition of the 14 november 2012

Glossary

betatron tune the betatron tune is the frequency of the oscillations of the bunches divided by the RF frequency (f_0). 1, 7, 26

bucket at every Radio Frequency (RF) period in the RF frequency (f_0) there is a bucket, in each of these bucket a particles bunch can potentially be stored in the ring. 6, 26

bunch particules trapped inside an RF bucket circulating in the machine. 6, 26, 28

cavity RF structure made to accelerate the particles, it uses a high power radio frequency into a resonating structure to increase the energy of the particles. 6, 28

damper machine in an accelerator that damp the transverse oscillation of the beam by applying a transverse electric field. 1, 29

FESA FESA is the Front-End Software Architecture a standard framwork used at CERN and provided by the CO group.. 15

FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST). 16, 18, 21, 26

kicker machine in an accelerator that can kick the beam transversally, used to kick the beam in or out (injection or extraction kicker) of the beam pipe but also in our case excite the beam transversally. 6, 30

RF frequency (f_0) the base frequency of the RF in the cavities in the case of the LHC this frequency is 400.789MHz, this frequency dictate the number of bucket that the machine can have. 6, 26, 28

VMEbus a computer bus standard widespread at CERN, in the case of the LHC RF the bus has a larger board and some of the pins are used to route custom signals between cards. 7, 15

Acronyms

ADT Accelerating Damper Transverse. 1, 6, 7, 26, 27

API Application Programing Interface. 9–13

BBQ diode-based base-band-tune. 6

BI Beam Instrumentation. 6–8

BPM Beam Position Monitor. 6, 7, 26, 27

CCC CERN Control Center. 16

CERN European organization for nuclear research. 1, 13, 15, 16, 26, 28

CMW Control Middleware. 15

CO Control. 15, 26, 28

CPU Central Processing Unit. 7, 10, 11, 15–21, 26

CUDA Compute Unified Device Architecture. 11, 12, 20

DFT Discreet Fourier Transform. 17, 18

DSP Digital Signal Processor. 7

FFT Fast Fourier Transform. 1, 7, 12, 14, 16–18, 20, 21, 26

flops floating-point operations per second. 20

FPGA Field-Programmable Gate Array. 7

GLSL OpenGL Shader Language. 12

GPGPU General Purpose Graphic Processing Unit. 1, 9, 12

GPU Graphic Processing Unit. 1, 7, 10–13, 15–21, 25, 26

Hepia Haute école du paysage, d'ingénieurie et d'architecture de Gemève. 1

HLSL High Level Shader Language. 12

LHC Large Hadron Collider. 1, 6, 7, 25, 26

MD Machine Development. 14, 27

MIT Massachusetts Institute of Technology. 18

MKQA tune kicker. 6

OP Operation. 26

OpenCL Open Computing Language. 9–13, 16, 18, 20, 21

OpenGL Open Graphic Language. 12

RF Radio Frequency. 28

SLC Scientific Linux CERN. 13

SVD Singular Value Decomposition. 1, 16, 18

Bibliography

[1] Boost c++ libraries. http://www.boost.org/.

[2] Eric Bainville, mars 2011. http://www.bealto.com/.

[3] P Baudrenghien, Wolfgang Höfle, G Kotzian, and V Rossi. Digital signal processing for the multi-bunch lhc transverse feedback system. oai:cds.cern.ch:1124094. *LHC-PROJECT-Report*, 1151:4, Sep 2008.

[4] A Boccardi, M Gasior, R Jones, P Karlsson, and RJ Steinhagen. First results from the lhc bbq tune and chromaticity systems. Technical Report LHC-Performance-Note-007. CERN-LHC-Performance-Note-007, CERN, Geneva, Jan 2009.

[5] Rama Calaga. *Linear Beam Dynamics and Ampere Class Superconducting RF Vavities at RHIC*. PhD thesis, Stony Brook University, 2006.

[6] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19:297–301, 1965.

[7] NVIDIA corp., 2012. http://www.nvidia.com/.

[8] F Dubouchet, W Höfle, G Kotzian, and D Valuch. what you get transverse damper system (adt). In *Evian 2012 proceedings*, 2012. https://indico.cern.ch/.

[9] M. Frigo and S.G. Johnson. The design and implementation of fftw3. *Proceedings of the IEEE*, 93(2):216–231, feb. 2005.

[10] Naga K. Govindaraju and Dinesh Manocha. Cache-efficient numerical algorithms using graphics hardware, 2007.

[11] Wolfgang Höfle. Lhc transverse damper observations versus expectations. In *Evian 2010 proceedings*, 2010. https://espace.cern.ch/acc-tec-sector/Evian/Papers-Dec2010/3.3_WH.pdf.

[12] Wolfgang Höfle. Lhc transverse damper from commissioning to routine. *Beams Department newsletter*, 1:6–7, 2011. https://espace.cern.ch/be-dep/BEDepartmentalDocuments/BE/BE_Newsletter/BE_Newsletter_001.pdf.

[13] Wolfgang Höfle. Transverse feedback : high intensity operation, abort gap cleaning, injection gap cleaning and lessons for 2012. In *Evian 2011 proceedings*, 2011. https://indico.cern.ch/.

[14] Wolfgang Höfle. Transverse damper. In *Chamonix 2012 proceedings*, 2012. https://indico.cern.ch/.

[15] S. Lahabar and P.J. Narayanan. Singular value decomposition on gpu using cuda. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1 –10, may 2009.

[16] Matthew Scarpino. *OpenCL in Action*. Manning Publications, 2011.

[17] Daniel Valuch. Beam phase measurement and transverse position measurement module for the lhc, 2007. Poster from the Low Level RF workshop 2007. Available "https://edms.cern.ch/document/929563/1.

[18] V M Zhabitsky, E V Gorbachev, N I Lebedev, A A Makarov, N V Pilyar, S V Rabtsun, R A Smolkov, P Baudrenghien, Wolfgang Höfle, F Killing, I Kojevnikov, G Kotzian, R Louwerse, E Montesinos, V Rossi, M Schokker, E Thepenier, and D Valuch. Lhc transverse feedback system: First results of commissionning. oai:cds.cern.ch:1141925. Technical Report LHC-PROJECT-Report-1165. CERN-LHC-PROJECT-Report-1165, CERN, Geneva, Sep 2008.