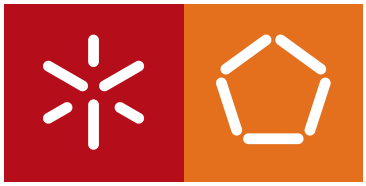
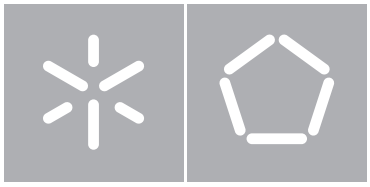


Universidade do Minho
Escola de Engenharia

André Martins Pereira

Efficient processing of ATLAS events analysis in platforms with accelerator devices

Fevereiro de 2013



Universidade do Minho
Escola de Engenharia
Departamento de Informática

André Martins Pereira

Efficient processing of ATLAS events analysis in platforms with accelerator devices

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de
Professor Alberto Proença
Professor António Onofre

Fevereiro de 2013

Abstract

Contents

Glossary

Event head-on collision between two particles at the LHC

LHC Large Hardron Collider particle accelerator

ATLAS project Experiment being conducted at the LHC with an associated particle detector

LIP Laboratório de Instrumentação e Física Experimental de Partículas, Portuguese research group working in the ATLAS project

CERN European Organization for Nuclear Research, which results from a collaboration from many countries to test HEP theories

HEP High Energy Physics

Analysis Application developed to process the data gathered by the ATLAS detector and test a specific HEP theory

Accelerator device Specialized processing unit connected to the system by a PCI-Express interface

CPU Central Processing Unit, which may contain one or more cores (multicore)

GPU Graphics Processing Unit

GPGPU General Purpose Graphics Processing Unit, recent designation to scientific computing oriented GPUs

DSP Digital Signal Processor

MIC Many Integrated Core, accelerator device architecture developed by Intel, also known as Xeon Phi

QPI Quickpath Interconnect, point-to-point interconnection developed by Intel

HT HyperTransport, point-to-point interconnection developed by the HyperTransport Consortium

NUMA Non-Uniform Memory Access, memory design where the access time depends on the location of the memory relative to a processor

ISE Instruction Set Extensions, extensions to the CPU instruction set, usually SIMD

Homogeneous system Classic computer system, which contain one or more similar multicore CPUs

Heterogeneous system Computer system, which contains a multicore CPU and one or more accelerator devices

SIMD Single Instruction Multiple Data, describes a parallel processing architecture where a single instruction is applied to a large set of data simultaneously

SIMT Single Instruction Multiple Threads, describes the processing architecture that NVidia uses, very similar to SIMD, where a thread is responsable for a subset of the data to process

SM/SMX Streaming Multiprocessor, SIMT/SIMD processing unit available in NVidia GPUs

Kernel Parallel portion of an application code designed to run on a CUDA capable GPU

Host CPU in a heterogeneous system, using the CUDA designation

CUDA Compute Unified Device Architecture, a parallel computing platform for GPUs

OpenMP Open Multi-Processing, an API for shared memory multiprocessing

OpenACC Open Accelerator, an API to offload code from a host CPU to an attached accelerator

GAMA GPU and Multicore Aware, an API for shared memory multiprocessing in platforms with a host CPU and an attached CUDA enabled accelerators

List of Figures

1. Introduction

1.1. Motivation

2. Parallelization Approaches

The section of code which takes more time is the `ttDilepKinFit` function. The main objective is to run as many kinematical reconstructions per event, with a slight variation to the particle characteristics, and as they increase the `ttDilepKinFit` execution time also increases. So, since it is the critical section of the application, the efforts on performance optimizations will be focused on this portion of the code.

The `ttDilepKinFit` workflow can be divided in three main stages. Each event can have an arbitrary number of jets and leptons associated, requiring a minimum of two of each to perform the kinematical reconstruction of the `ttbar` system. Events that not fulfill this requirement are discarded in the previous cuts. If there is more than the minimum number of jets and leptons it is necessary to combine them, in pairs of two jets with two leptons, and perform the kinematical reconstruction for every combination possible. Note that their order on the combination is not relevant, reducing the total amount of possible combinations. Then, it is possible to apply a variation to the jets and leptons characteristics, motivated by the reasons explained in section 1.1. The variation has a magnitude equivalent to the experimental resolution of the ATLAS detector (which is 5%) and it is applied to the three momentums and energy of the particles, and causing the need to re-compute other auxiliary parameters to the rest of `ttDilepKinFit`. The number of variations to apply to each jet/lepton combination is arbitrary and defined by the user.

`ttDilepKinFit` has a main loop, for each jet/lepton combination and for each variation of each combination, where the most intensive computation occurs, which will be explained next, and a final section of code that iterates through all the reconstructions and picks only the best, discarding all the others computed.

Inside the main loop of `ttDilepKinFit` is possible to identify three distinct stages of the reconstruction. The first is the variation of the jets and leptons momentums, as explained before. The second stage is the kinematical reconstruction of the `ttbar` system, using the varied combinations. It attempts to reconstruct the `ttbar` system, presented in section 1.1 and produces a result (the Top Quarks), which has an computed probability associated to its accuracy. This probability determines the quality of the reconstruction. The third stage is the reconstruction of the Higgs boson, using the results of the kinematical reconstruction. This reconstruction also has a probability associated, and the final quality of the overall reconstruction of the event is given by the multiplication of this probability with the one of the kinematical reconstruction of the `ttbar` system. Figure 2.1 presents the explained workflow of `ttDilepKinFit`.

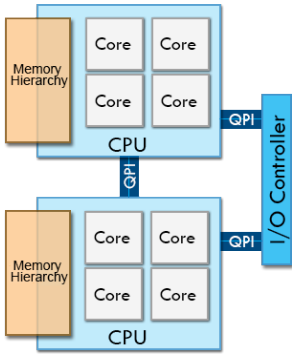


Figure 2.1.: Schematic representation of the `ttDilepKinFit` workflow.

Note that there is data dependencies between loop iterations, since that for chosing the next jet/lepton combination it is necessary to know which combinations were already picked, and between the three stages within the loop. The kinematical reconstruction needs the combination already varied, and the Higgs boson reconstruction needs the result from the kinematical reconstruction in order to chose different

particles (since one particle cannot belong to the ttbar system and the Higgs boson decay). The current workflow is not suitable for parallelization.

2.1. Shared Memory Parallelization

The current workflow of ttDilepKinFit needs to be changed to expose potentially parallel tasks. One challenge to the implementation is that most of the variables are global to the application, and there is no data structure holding the information needed during ttDilepKinFit, such as the combinations. The computation of the jet/lepton combinations must be kept sequential, since choosing one combination is always dependent on all the previous combinations made and, therefore, cannot be parallelized. The final iteration through all the results, in which the best solution is chose and is "uploaded" to global variables, cannot also be parallelized in the current implementation (there is, however, a way to overcome part of this problem that will be explained later). During this chapter a concurrent task is considered to be the subset of a parallel region. The aggregation of all these tasks is the whole parallel region. Figure 2.2 presents the workflow used for this implementation, and is explained next.

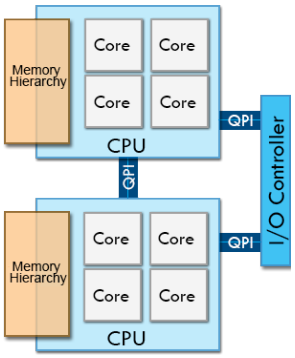


Figure 2.2.: Schematic representation of the ttDilepKinFit workflow.

The first step of the new workflow is to create a data structure holding all the combinations and other information associated with them. Each task will pick the respective combination and perform a loop over the subset of the total number of variations. The grain size of the parallel work of each task will be dependent on the total number of combinations times the number of variations per combination (the higher this value the coarser the grains size) and the number of parallel tasks (the higher the number of tasks the thinner the grain size).

The second step, the kinematical reconstruction of the ttbar system, will be performed within the same task, meaning that there is no synchronization between this and the previous steps among different tasks. The same happens between this and the third step, the Higgs boson reconstruction. By aggregating these three steps the grain size of the tasks is increased, compared to the generalist parallelization model presented before, which benefits their execution on CPU (a lesser number of coarser tasks means that the CPU spends more time performing computations relevant to the application and less time switching context, which is very slow).

As explained before, after these three stages, on the original workflow, there is an iteration through all the solutions and the best is chosen. Instead of saving all the solutions, after each iteration of the loop of the current workflow the solution is compared to the previous and only the best of the two is save. When all tasks finish all their respective iterations, each will have the best solution for the subset of combinations and variations that they processed. A reduction can be made so that the best solution from all the tasks is found. However, another data structure must be created, since the best solution is a set of variables and TLorentzVectorWFlags (from the LipMiniAnalysis library) class instances. The final "uploading" of the best solution to global variables is only made by the task with the best solution.

This workflow will have two limitations to the performance scalling, the creation of the data structure holding the combinations and the best solutions and the creation and work sharing between the tasks. Its time increases with the number of combinations and variations and the number of parallel tasks to use.

Appendix A. Test Environment

This appendix focuses on fully characterizing the hardware and software used in all performance measurements of the application for the different implementations developed.

For the shared memory implementation testing was used three dual-socket multicore systems. The first has two Intel Xeon E5-2650 (Sandy Bridhe architecture) **??**, using the Quick Path Interconnect (QPI) interface between CPUs, in a Non Unified Memory Access model (NUMA), meaning that the latency of a CPU acessing its own memory bank is lower than accessing the other CPU memory bank. The QPI interface can perform up to 8 GT/s (giga transfers per second) of 2 bytes packets, in each of the two unidirectional links, with a total bandwidth of 32 GB/s. Figure Appendix A.1 illustrates the architectural model of this system. The system features 64 GB of DDR3 RAM with a speed of 1333 MHz, for a maximum bandwidth of ZZ GB/s, measured with the STREAM benchmark **??**.

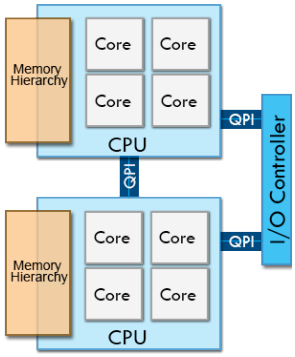


Figure Appendix A.1.: Schematic representation of a NUMA system with QPI interface.

The second system has the same amount of RAM at the same speed, with a maximum bandwidth of ZZ GB/s. The two CPUs are Intel Xeon X5650 (Nehalem architecture). The difference of memory bandwidth is due to the different memory controllers, while the one in Nehalem has 3 memory channels the one in Sandy Bridge has 4. The two CPUs are interconnect by a QPI interface, but with a different speed than the Sandy Bridge, performing 6 GT/s in each of the two unidirectional channels, for a total bandwidth of 24 GB/s.

The third system features two AMDOpteron 6174, being the system with more physical cores. It has 64 GB of DDR3 RAM at 1333 MHz, with a maximum measured bandwidth of ZZ GB/s. AMDuses HyperTransport (HT) 3.0 technology, a point-to-point interconnection similar to QPI capable of transmitting 4 byte packets through two links, for an aggregate bandwidth of 51.2 GB/s. The characteristics of the CPUs on the three systems are presented in table Appendix A.1.

The Roofline model **??** was used characterize the system in terms of attainable peak performance. This model uses two metrics for the performance calculation: the peak CPU performance and the memory bandwidth. With the peak values of these two metrics a roofline is drawn, being the theoretical limit for the performance on the system. Then, other ceilings can be added, which further limit the maximum attainable performance. The classic Roofline uses float point computation as the peak CPU performance metric. It may be a good metric for heavy computational algorithms, such as matrix multiplication, but the type operations on the critical region (`ttDilepKinFit` function) are much more varied, as shown in the instruction mix presented in section **??**. Instead, the computational intensity was used for measuring the CPU peak performance, as it considers all types of instructions.

Figure **??** illustrates the Roofline model for the three systems.

CPU	Intel Xeon E5-2650	Intel Xeon X5650	AMDOpteron 6174
Architecture	Sandy Bridge	Nehalem	MagnyCours
Clock Freq.	2.0 GHz	2.66 GHz	2.2 GHz
# of Cores	8	6	12
# of Threads	16	12	12
L1 Cache	32 KB I. + 32 KB D. per Core	32 KB I. + 32 KB D. per Core	64 KB I. + 64 KB D. per Core
L2 Cache	256 KB per Core	256 KB per Core	512 KB per Core
L3 Cache	20 MB shared	12 MB shared	
CPU Interconnection	QPI @4.0 GHz	QPI @3.2 GHz	HT @3.2 GHz
ISE	AVX	SSE 4.2	SSE 4a

Table Appendix A.1.: Characterization of the CPUs featured in the three test systems.

The compiler used was the GNU compiler version 4.8, using the -O3 optimizations and the AVX/SSE 4.2/SSE 4a (depending on the CPU architecture) instruction set on the code regions that the compiler sees fit. The compiler features the OpenMP version 3.2 used in the shared memory implementation. For the GPU implementation was used the CUDA 5 SDK, in conjunction with the GNU compiler version 4.6.3 for the code to run on the CPU (any later versions are not supported by the NVidia NVCC compiler). The ROOT **??** version used was the 5.34/05. All libraries/frameworks used were compiled with compliance to the C++ 11 specifications to ensure, among other things, thread safety on memory allocations. Was used the Performance API version 5.0 for measuring the hardware counters of the different CPUs for the characterization of `ttDilepKinFit`.