Optimizing ATLAS software: QT and beyond

CATERINA MARCON – LUND UNIVERSITY

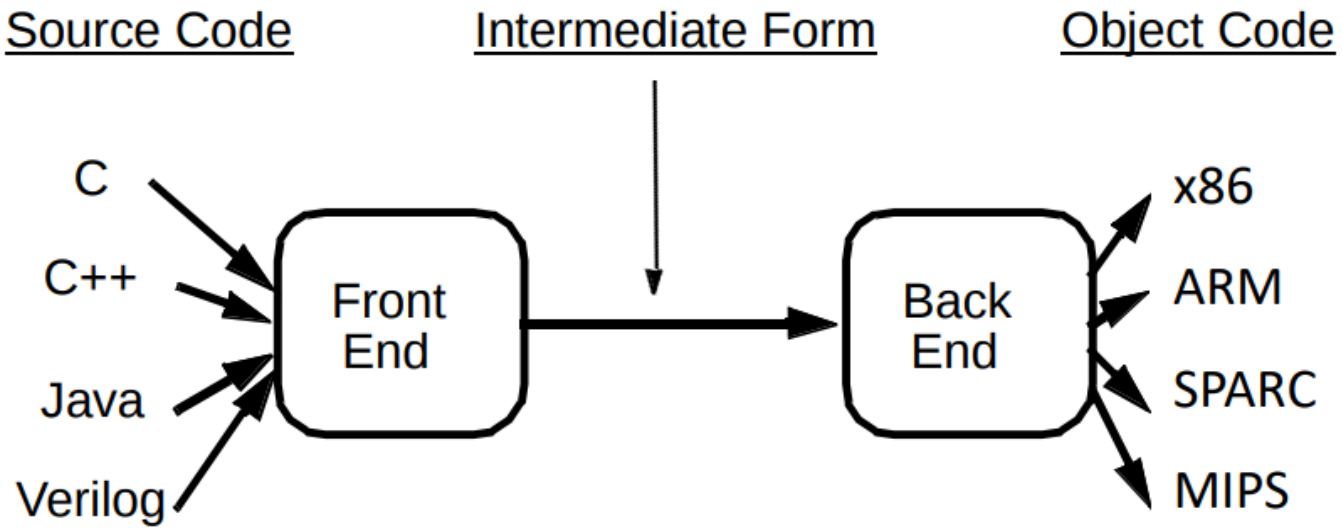
17.XII.2018

Outline

- Optimizing ATLAS software: QT and beyond
  - Optimization principles
  - QT: Investigating possible CPU improvement in simulation job
  - What's next
- Education

The structure of a compiler

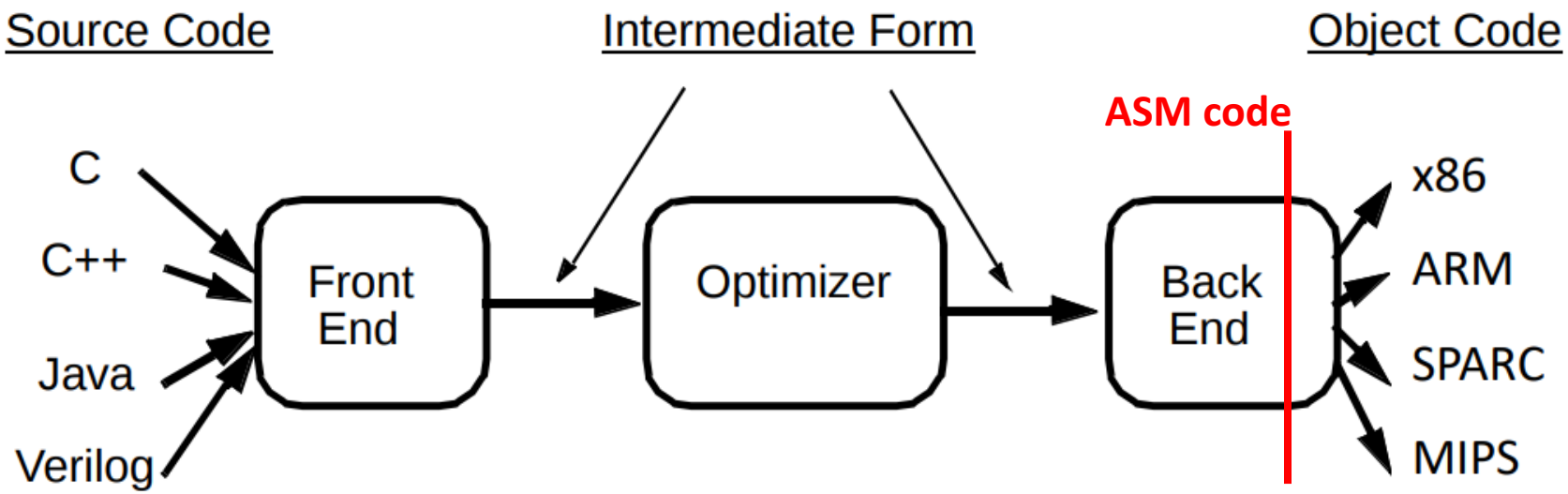
- A compiler is able to map a **source program** into a semantically equivalent **target program**;
- The mapping process involves two steps:
  - The **front end part**: breaks up the source program into different constituent pieces imposing a grammatical structure in order to create an intermediate representation;
  - The **back end** that constructs the desired target program from the intermediate representation.



LUND UNIVERSITY

The structure of a compiler

- Some compilers have a machine-independent optimization phase between the front end and the back end;
- The purpose of this optimization phase is to perform transformations on the intermediate representation, so that the back end can produce a better target program;
- There is a great variation in the amount of code optimization different compilers perform.



LUND UNIVERSITY

Optimization quick guide

option	optimization level	execution time	code size	memory usage	compile time
-O0	optimization for compilation time (default)	+	+	-	-
-O1 or -O	optimization for code size and execution time	-	-	+	+
-O2	optimization more for code size and execution time	--		+	++
-O3	optimization more for code size and execution time	---		+	+++
-Os	optimization for code size		--		++
-Ofast	O3 with fast none accurate math calculations	---		+	+++

+increase ++increase more +++increase even more -reduce --reduce more ---reduce even more

A simple example (C++ front end)

```
void WasteTime() {

  int UselessArray[500][1000];

  for (int i = 0; i < 500; i++)
    for (int j = 0; j < 1000; j++)
      UselessArray[i][j] = 5;

}
```

- This function is completely pointless:
  - Creates a big array
  - Sets all elements to a constant
  - It's void, hence no return

- Optimization is expected to:
  - Foresee the uselessness;
  - Minimize the effects of this waste

LUND UNIVERSITY

A simple example (ASM back end)

`-O0` (i.e. almost no optimization)

```
g++ -O0 -S wastetime.cc -o wastetime_O0.s
```

Loop on `i`
Counting up (1 to 500)

Loop on `j`
Counting up (1 to 1000)

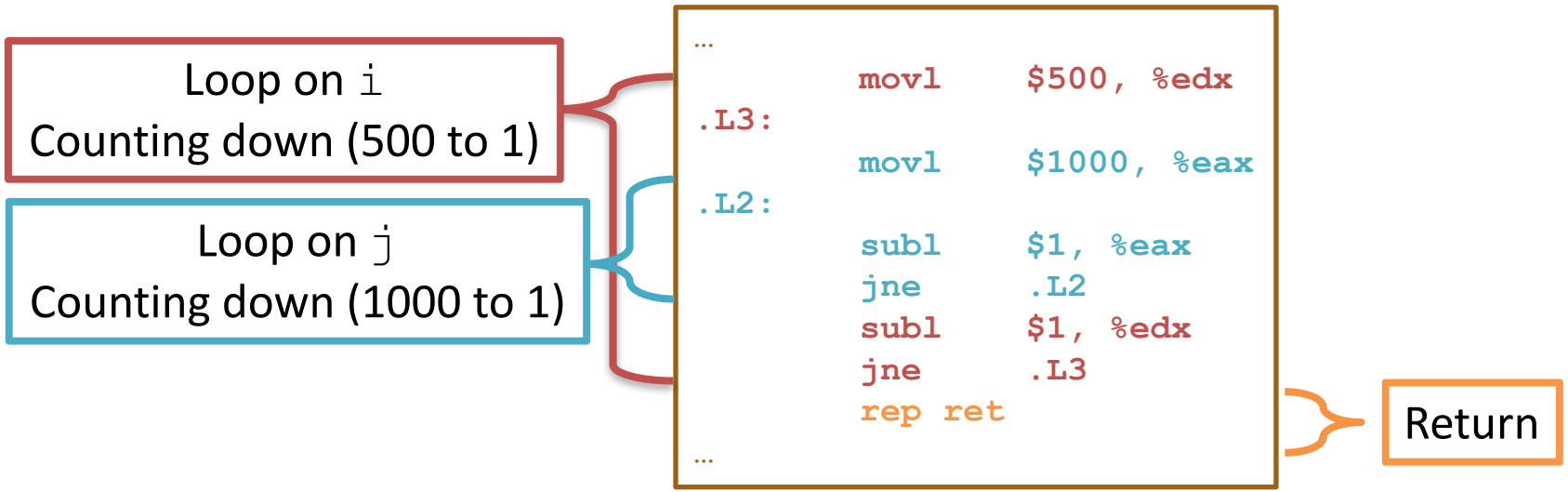
```
        movl    $0, -2000024(%rbp)
.L5:
        cmpl    $499, -2000024(%rbp)
        jg      .L7
        movl    $0, -2000020(%rbp)
.L4:
        cmpl    $999, -2000020(%rbp)
        jg      .L3
…
        movl    $5, -2000016(%rbp,%rax,4)
        addl    $1, -2000020(%rbp)
        jmp     .L4
.L3:
        addl    $1, -2000024(%rbp)
        jmp     .L5
.L7:
```

Setting `ij-th` element to `5`

This is just a straightforward translation from our C++ into ASM

A simple example (ASM back end)

```
-O1
g++ -O1 -S wastetime.cc -o wastetime_O1.s
```



The constant assignment is gone and loops are just countdowns.
Better, but it's still taking time…

A simple example (ASM back end)

```
-O2
g++ -O2 -S wastetime.cc -o wastetime_O2.s
```

```
rep ret
```

Loops are gone as well: just returning.

Success!

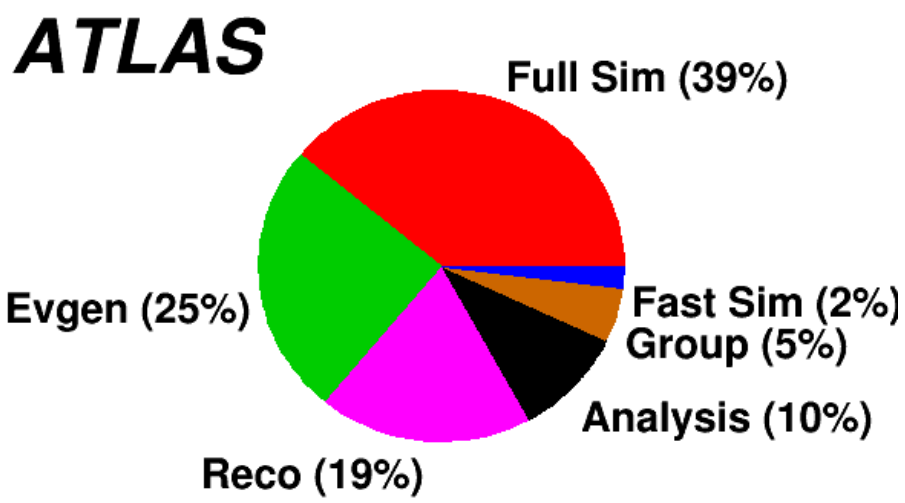
We got rid of the useless function.

What should we take home?

- **More efficient does not necessarily mean shorter**
  - The optimization process actually rewrites code to make it better **for the computer**;
  - Optimized code may look much more verbose than unoptimized.
- **Ther's no such thing as -O∞**
  - In this case, -O3 looks the same as -O2, meaning that optimization can reach saturation;
  - There are cases in which too aggressive optimizations lead to broken code.
- **One size does not fit all**
  - Optimization depends on the actual code: different code with same flags will give different outcomes;
  - It's not straightforward to adopt someone else's successful optimization strategies.

LUND UNIVERSITY

Investigating possible CPU improvement in simulation job



Wall clock time fraction for grid and HPC jobs
July 2015 - July 2016

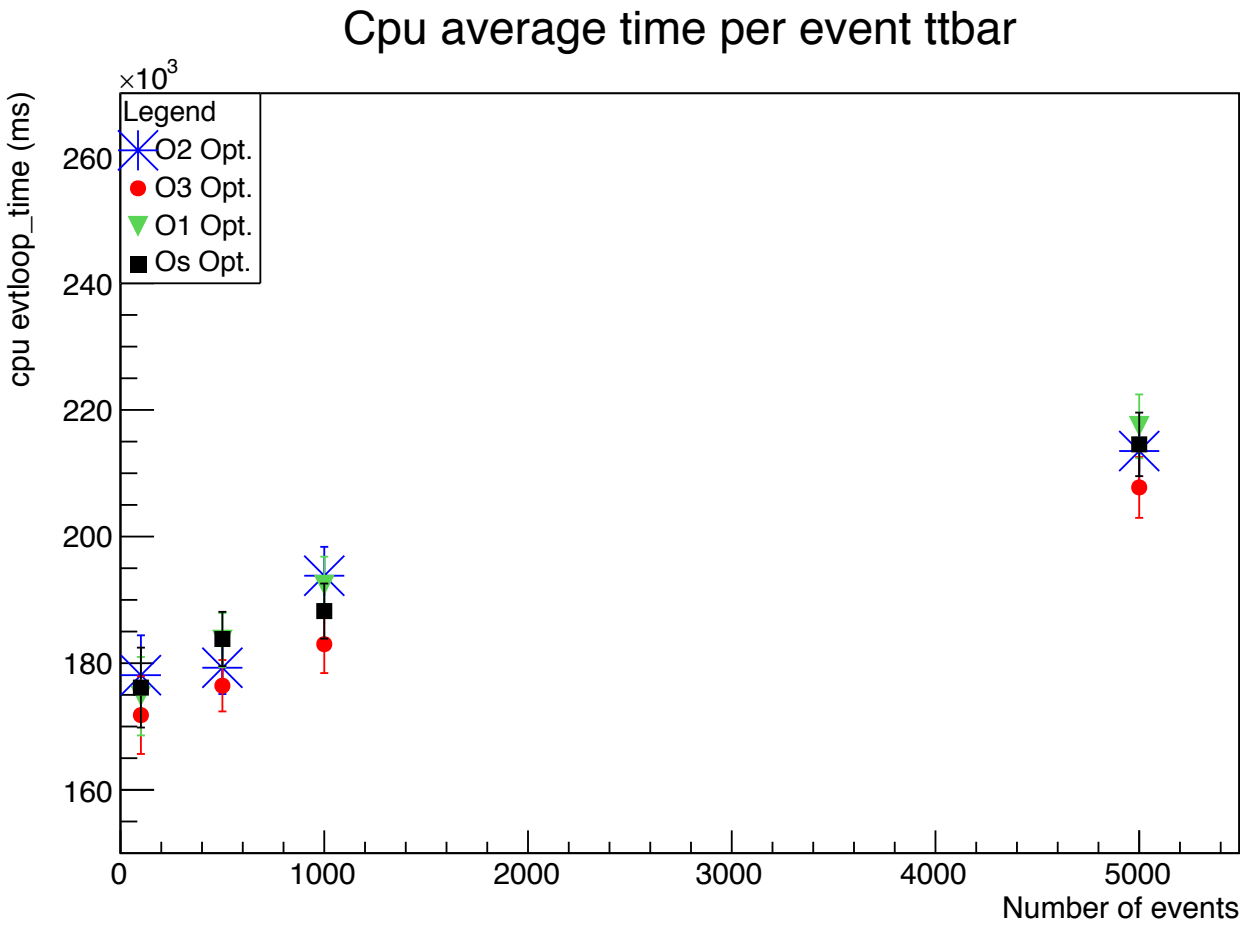
- **Simulations take up to 40% of the total CPU load time**. There is the necessity of an **improvement in speed** (without sacrificing the physics);
- A possible solution for running faster using less CPU resources is changing the **compiler's optimization options**;
- The GNU gcc compiler allows for different optimization settings: **Os, O1, O2 (default one) and O3**.

LUND UNIVERSITY

Preliminary results

- I obtained some **reference results with Os, O1, O2 and O3;**
- **Build the AthSimulation** release on Aurora cluster (Lund):
  - Each node has 64 GB and 20 cores;
  - Default memory request: **3100 MB per core;**
  - Exclusive node access;
  - Multicore execution with **AthenaMP**.
- Consider different samples for the simulation: ttbar, Z -> ee and di-jet;
- Run the jobs **changing the compiler settings** and keeping the **RandomSeed parameter unchanged**;
- Evaluate for each compiler settings the **average CPU time spent per event**.

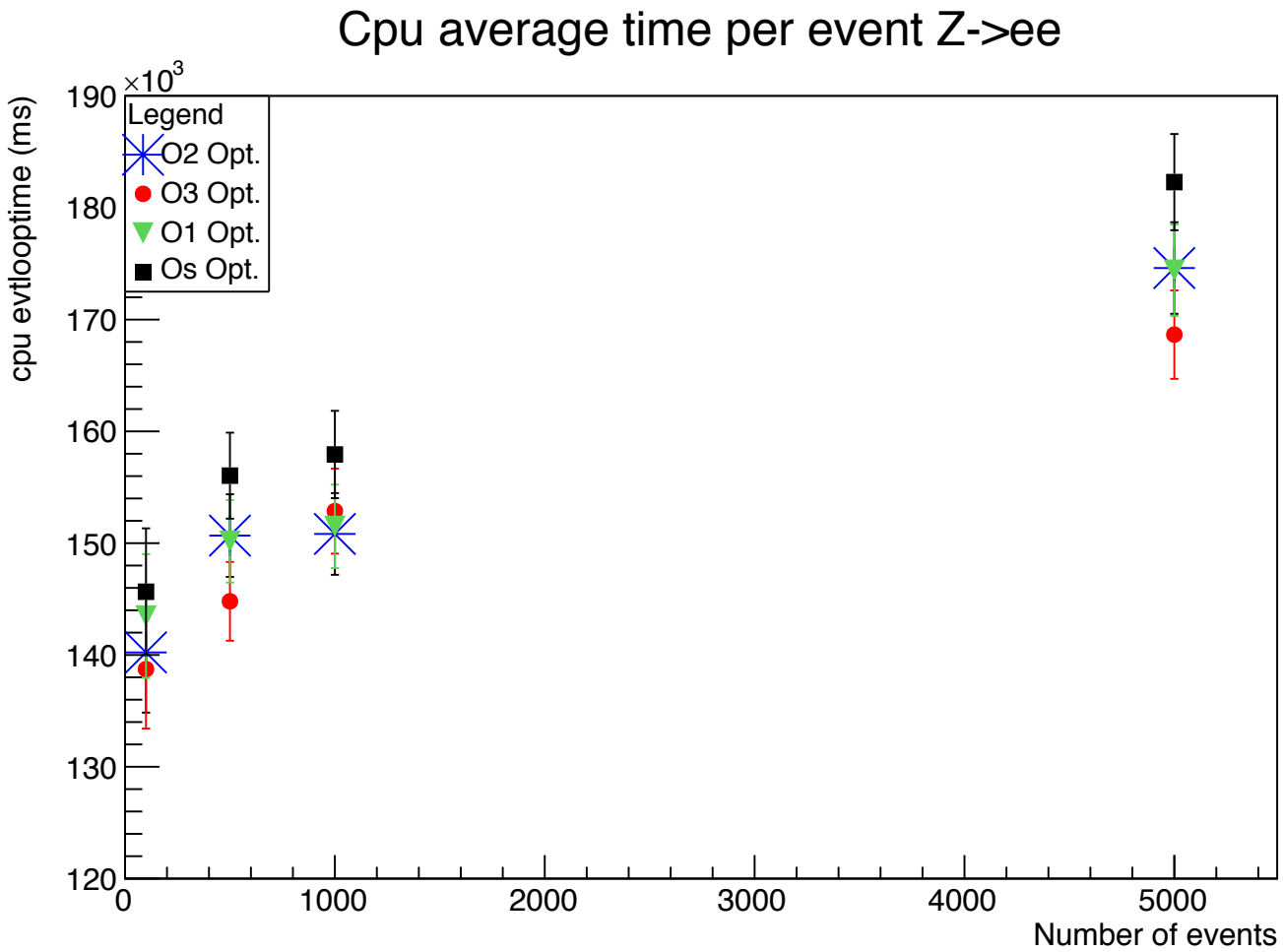
Preliminary results: ttbar sample



- The number of cores used for each set of initial number of events has been chosen in order to have an **error around 2%**;
- In all these cases **O3 is the best optimization**;
- The average improvement of **O3 on O2 is 4.5%**;
- The average improvement of **O3 on Os is 3.75%**.

LUND UNIVERSITY

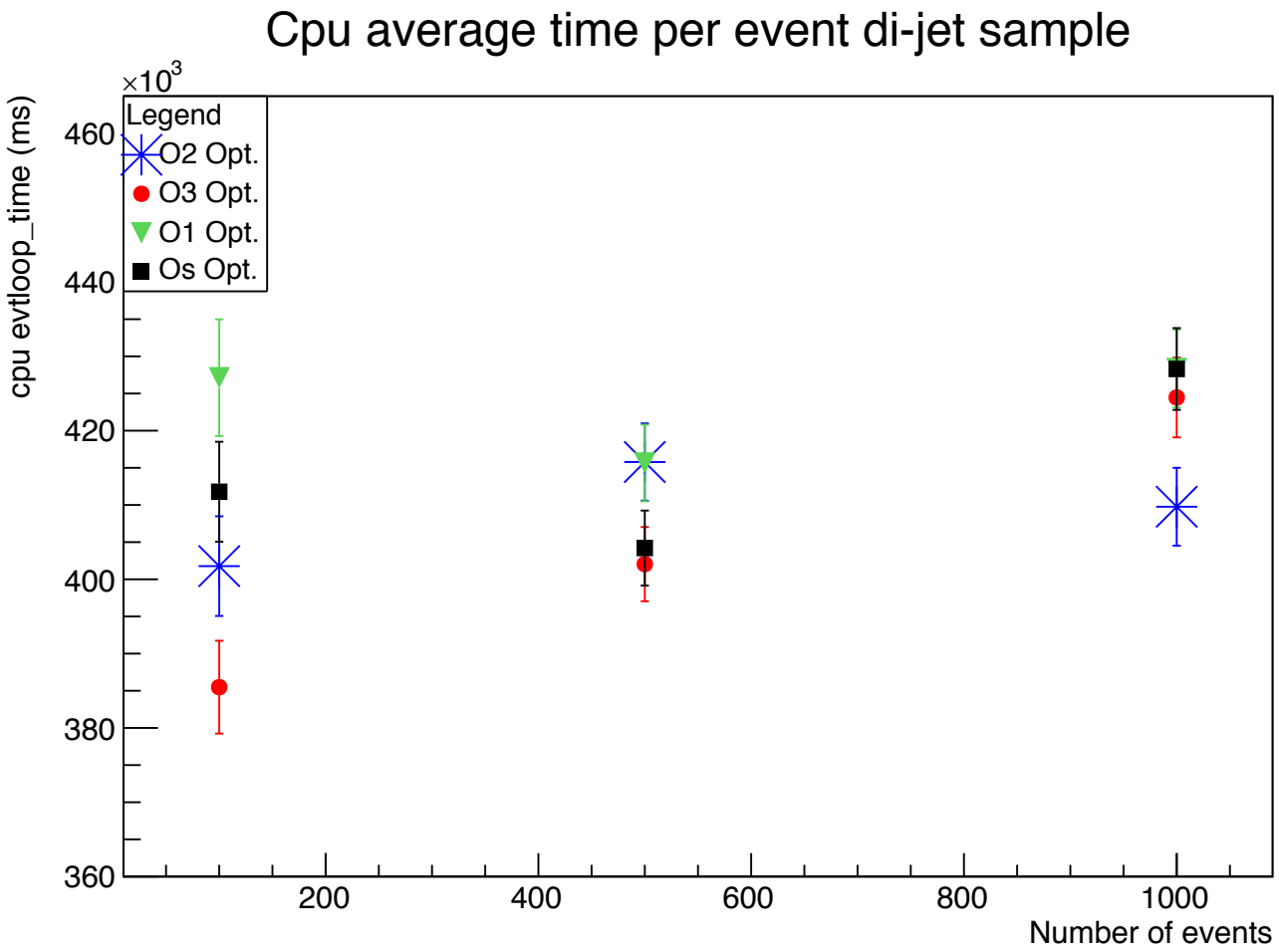
Preliminary results: Zee sample



- The number of cores used for each set of initial number of events has been chosen in order to have an **error around 2%**;
- The average improvement of **O3 on O2 is 2.25%**;
- The average improvement of **O3 on Os is 6.25%**.

LUND UNIVERSITY

Preliminary results: di-jet sample



- The number of cores used for each set of initial number of events has been chosen in order to have an **error around 2%**;
- The average improvement of **O3 on O2 is 2%**;
- The average improvement of **O3 on Os is 3%**.

LUND UNIVERSITY

Conclusions

- **O3 shows a tendency to be better than O2** in all the configurations considered;
- **All the samples examined are affected by the different optimizations**; this makes it reasonable to try to run AthSimulation with other options (such as AutoFDO).
- **The gain with O3 is more pronounced when the number of initial events is low**. This would lead to the conclusion that it would be convenient to split the simulation files in some smaller jobs;

AutoFDO optimization

- TASK: use AutoFDO procedure to generate profiles and recompile ATLAS code and run benchmark jobs. If this study shows at least 5% improvement -> use AutoFDO in nightly build system.
- **AutoFDO** [1] is an optimization technique available in Linux which provides performance gains and is able to collect profile data on production system;
- A statistical analysis of the code flow allows the identification of the most called functions;
- Based on this picture, optimization is applied to the most relevant flows;
- The use of this tool has already been investigated in CMS simulation workflows and has shown a **10% improvement**;

[1] https://gcc.gnu.org/wiki/AutoFDO/Tutorial

Future development

- Start with **a proof of concept** to optimize a specific physics workload:
  - Start with a **simple and well understood G4 simulation in athena**;
  - **Optimize the single simulation** and assess the improvements;
  - Broaden the range of optimizations.
- **Breakdown of the ATLAS software backend library by library** reconstructing how to control and optimize the workload:
  - Instrument the execution flow of the various functions/libraries and investigate through compiler optimizations;
  - Currently, the mixed python/c++ framework cannot be instrumented or manipulated directly using a compiler;
  - Investigate a simpler compilable framework that replicates the execution code flow and use compiler-based optimizations to improve it.

LUND UNIVERSITY

Education

- Introductory work on TLA analysis
- Domain decomposition (pure mathematical approach, not really suited for a *compute* school)
- Particle Physics Phenomenology
- Publication methodology for PhD students
- GEANT4
- Detector school
- Teaching course

LUND UNIVERSITY

LUND
UNIVERSITY