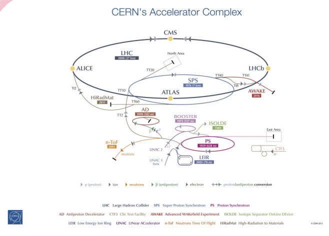
ATLAS HLT and FTF

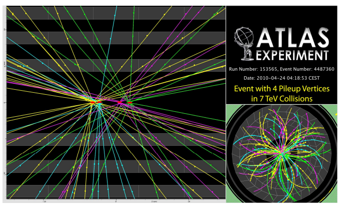
Rosie Hasan



https://cds.cern.ch/images/OPEN-PHO-ACCEL-2013-056-1

Introduction

ATLAS is one of 4 experiments at the Large Hadron Collider (LHC) at CERN. The Rutherford Appleton Laboratory's (RAL) Particle Physics department work on many aspects of this experiment. The LHC collides bunches of protons, within the 4 experiments. Each of these bunch crossings is known as an event. Within each event there are several collisions between the protons in the bunch, which produce various particles. Each event produces a lot data, of how the particles produced interact with the different layers of the detector. This data is then used to reconstruct the tracks of the particles and where energy was deposited, similar to the picture on the right. This data is then analysed for new physics .


ATLAS EXPERIMENT
Run Number: 153565, Event Number: 4487360
Date: 2010-04-24 04:18:53 CEST
Event with 4 Pileup Vertices in 7 TeV Collisions

https://twiki.cern.ch/twiki/bin/view/AtlasPublic/EventDisplayStandAlone

Trigger Project

Working within the trigger group, I learnt about the software upgrade for run 3 of the LHC, how this is being developed and optimised.
The aim is to improve the throughput, (number of events processed per second) and reduce the amount of memory being used as there is a limit to the amount available. I looked at 2 methods used for this: memory allocators, threads and forks.

Method

I ran the Atlas HLT system on a sample of 100 events varying the number of threads and forks and the memory allocators. I then studied the PSS (a measure of memory usage) and throughput (events processed per second) from the output of each run. I set the number of events processed by each thread (slots) to 1 and then varied the number of forks and threads up to 8 events being processed simultaneously. I looked at 4 memory allocators : stdcmalloc, tcmalloc (currently used), jemalloc and mimalloc.

Threads and forks

Splitting up the tasks to be completed. Reduce overall time and memory

- Forks: new process that runs independently, they run their own threads
- Threads: subtasks, can share memory

Triggering

The trigger system selects data that is more relevant as there is not enough storage for everything produced at ATLAS

Level 1 (L1): Custom Hardware, keeps 1 in 400 events

↓

High Level Trigger (HLT): software, many algorithms working to find event signatures, keeps 1 in 60 events

↓

Mass storage Offline reconstruction

Memory Allocators

Memory allocators optimize the assigning and deleting of memory required by processes. Each uses different algorithms to make these decisions.

Program: Starts new process which requires memory

↓

Memory Allocator (hidden): Receives requests and then assigns memory. It can combine several different requests

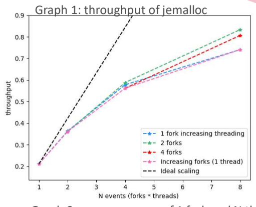
↓

Operating system: Provides memory for process

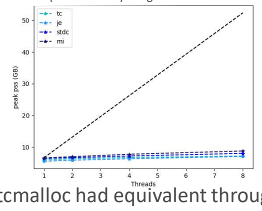
Results

Threads and forks: Overall I found that using thread and forks improved the throughput, as shown in graph 1, with the best throughput from 2 forks and 4 threads. Graph 2 shows how the maximum amount of memory being used was much less than if N computers were used (the black scaling line) so this technique is very efficient in terms of memory usage. This overall improvement was expected and threads and forks are already a technique being used in the trigger software.

Graph 1: throughput of jemalloc

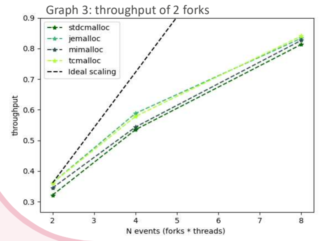


Graph 2: memory usage of 1 fork and N threads

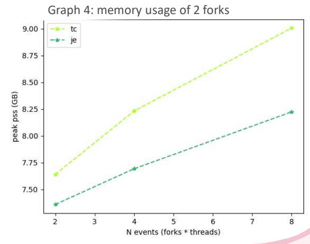


Memory allocators: I found jemalloc and tcmalloc had equivalent throughput (graph 3) but jemalloc used 8.7% less memory for the maximum throughput, as shown in graph 4. This could be a good reason to use it instead of tcmalloc, which is currently used as for the same throughput less memory is being used. Jemalloc is going be to tested in the next technical run in September

Graph 3: throughput of 2 forks



Graph 4: memory usage of 2 forks



Tracking Project

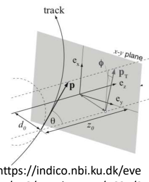
Before offline reconstruction the trigger system does partial reconstruction of tracks to aid in the decision process of whether to keep an event. An issue with the efficiency of these algorithms was found in previous studies, recreating the problem to see if I can find a cause

Method

1554 events were simulated, including what the tracks would look like (truth), then offline reconstruction and FTF (HLT and LRT) run on these events. I compared the tracks from all these methods and matched tracks with $\Delta\phi<0.1$ and $\Delta\eta<0.1$. I produced histograms of these matched tracks and the total number from each method, then took the ratio of these, which is equal to the efficiency. I measured the overall efficiency and how it varied in terms of the 5 parameters.

Tracking

As a charge particle passes through the inner detector it leaves 'hits' in the different layers it interacts with. These hits can be joined to form tracks, though there are millions of different combinations. The fast track finder (FTF) reduces the time it takes by only looking at hits in small sections of the detectors. These regions of interest (ROI) are identified by the L1 trigger. The FTF normally looks for tracks starting near the collision sight (HLT), though this means tracks from secondary vertices can be lost. The large radius tracker (LRT) uses algorithms optimised to find tracks starting at a larger radius. The tracks are described by the perigee (closest approach) parameters as shown in the diagram.



https://indico.nbi.ku.dk/event/758/attachments/1684/2344/trackingnote.pdf

Results

Same issue as found in previous studies maybe some histograms of unmatched and one of the efficiency compared to different parameters. Artificially high efficiency from the FTF matching to fake tracks found by the offline

Comparison	Efficiency
HLT vs Offline	78.44%
LRT vs Offline	63.85 %
HLT vs Truth	42.51 %
LRT vs Truth	25.37 %
Offline vs Truth	77.43%

Thank you to my supervisor Dr Stewart Martin-Haugh and RAL Particle Physics department