Migrating 4M of C++ to run in multithreaded environment

Christos Anastopoulos
Royal Society University Research Fellow
Where these 4M C++ lines are used?

In an experiment like Atlas we reconstruct events.
Broadly speaking, the purpose of this software is to convert the signals in the ATLAS sub-detectors to “particle candidates”. These then form the input for all ATLAS analyses and papers.

Where these 4M C++ lines are used?
Where these 4M C++ lines are used?

“Essential” complexity arises from the actual problem we try to solve. Trying to do non trivial “Physics” with a complex detector

“Accidental” complexity arises from the tools, procedures we use to develop and test our code.

Most of the talk will be more on managing “accidental” complexity.
On the 17th December 2018, ATLAS updated the Athena repository to make it public and open-source.

https://gitlab.cern.ch/atlas/athena
C++ is the main language
Python is next

Programming languages used in this repository
Measured in bytes of code. Excludes generated and vendored code.

- Main compiler for production gcc, but we also build with clang.
- Main platform x86-64, we started exploring aarch64
- Build System CMake
- Static checkers include flake8 (python), gcc plugins, clang-tidy, cppcheck, coverity (in-progress)
- Leak checkers/profilers: Valgrind, Callgrind, Vtune …
- Issue/feature tracking via JIRA
**Time frame**

Stable releases used for analysing data, correspond to a particular branch, only bug fixes allowed

The **master branch** is where we develop new code. We produce “nightly” builds (gcc, clang etc) each day.

The milestone for this talk is the LHC Run-3 data taking period. Start in ~ 3 months ….

All ATLAS papers to be published in the next few years will use the software discussed.
Multi threading

Multi-Process (MP) what we were doing,
Optimal usage of the currently available resources meant we had to migrate to Multi-Threading

Fit more “compute” in given resources.
Plots showing current status
Taken from here.
The problem

Migrate a code base mainly developed with “serial” running in mind
Written mainly pre the C++ “threading” model …

Current estimate is that we touched ~ 1.5 M lines of C++ code …

*While at the same time ensuring that the “physics” related output is correct and even improved.*
Things that helped

Merge request code reviews

Unit tests: Test the output/behaviour of one module on some “mocked” input.

Integration Tests: Run a few actual events through the full reconstruction chain. Check if number of muons, electrons etc change.

All part of automated “pipeline”. Seems trivial but a huge game changer.
Things that helped

On top of the merge request / Continuous Integration (CI) pipeline.

ATLAS Runtime Testing (ART) : 100s tests running for each “nightly” release (24h turnaround)

Example test output for 1 vs 8 threads:

test_receusreco_art_q431_compare1Vs8Threads.sh

Example test distribution that are used for “physics” using high stats sample.
Things that helped

Constant monitoring or cpu/memory metrics ~ 24 h turnaround
Things that helped

On a weekly basis reconstruct a few million of events from different run periods. Catches rare issues on taking unexpected paths:
- Floating point exceptions
- Dangling pointers
- Rare race conditions.

The final step “Physics” validation, collaboration wide effort where data produced by the software is used in realistic analysis scenarios.

It is not “continuous” but represents “sign off” points during the development cycle.
Things that helped

I am pretty convinced that the effort would have failed without this new machinery we started putting in place ~ 2016.

Code reviews, CI pipeline has been an almost “magic” transformation on the way we develop code.

Coupled with additional testing meant that we had a concrete view of where we are, the issues we were facing, the effect of any solution.

What follows is a couple of examples from personal experience.
Irreproducible “Muons”

The issue as described in **Draft MR (closed)** from our software co-ordinator 1 vs 8 threads

In lay terms:

The fitter used for muon trajectories was giving different outputs > 1 threads.

Telltale mark of MT hostile code.
Irreproducible “Muons”

I would probably not have even attempted this without our testing. As I would be effectively touching a critical piece of code “blind”. MR

In less than a week we knew the answer.

Edward Moyse @emoyse · 1 year ago
Wow. This really seems to fix a lot of irreproducibilities - well done Christos!
“Trivial” changes to interfaces

Cases where one needs to touch an interface. Many clients. Example MR

Cl more or less tells if you forgot something.
The rise of the “robots”?

We consider using more clang-tidy in our work flow.
Example of using modernize-replace-auto-ptr to enforce our “style-guide”
Conclusions

The ATLAS collaboration is at the end of long development cycle for its Run-3 software. This included a significant technical component of migrating to Multi-threading.

Personally, not sure I could even contemplate how we would have managed this without rigorous development and testing procedures.

For the Physics you will need to keep an eye for upcoming Run-3 results.
τέλος

OUR CEO SAID HE LIKED YOUR PRESENTATION.

HE MADE ME SHUT UP AND SIT DOWN BEFORE I GOT TO MY FIRST SLIDE.

HE’S NOT A BIG FAN OF CONTENT.
Backup
Figure 2: An example of multithreaded execution in AthenaMT. Four threads are shown, each corresponding to one row. Different events are shown with different colours, and different algorithms are shown with different shapes. The algorithms are executed as soon as their input data are available and a thread is free. (Image: ATLAS Collaboration/CERN)
Event processing in AthenaMT:
  ● Each event is a different color
  ● Each shape is a different algorithm
  ● T1-T4: threads #1-4