Optimizing the CMSSW Infrastructure for Run 3

A. Valenzuela[1], S. Muzaffar[1], I. Razumov[2]

CERN - CMS Core Software
ACAT 2024 - Stony Book, NY

March 13th, 2024

[1]CERN, [2]Princeton University

CMS Offline Software - CMSSW

- CMSSW contains the **software collection** needed to process event data at CMS.
- It has a large **code base hosted on GitHub**.

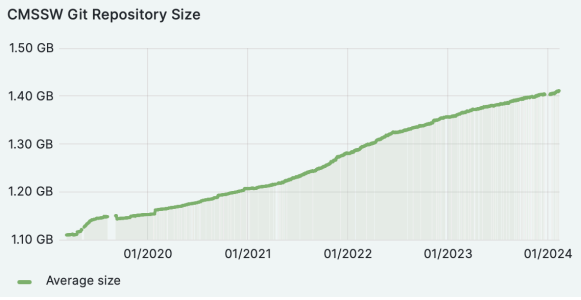

Figure: Growth of the CMSSW GitHub repository since 2019.

CMSSW Code Base

- 100+ Contributors/month
- 500+ Commits/month
- 100+ PRs/week
- 200+ GitHub repositories for CMSSW and externals

- **5.5M lines** of code leading to 3k binary products.
- **550+ external packages** built from source.

CMS Offline & Computing - Core Software

Core Software@CMS

- CMSSW Build Releases
- Integration Builds (IBs)
- Continuous Integration / Delivery (CI/CD) system

- This contribution presents the recent **enhancements to CMSSW**:
  - Software Stack.
  - Infrastructure.

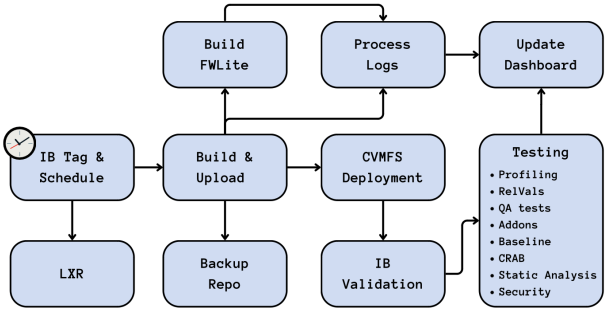

Figure: CMSSW IB Infrastructure schema.

- Those improvements are focused on **obtaining more throughput**, adding a **wider testing coverage** and **reducing manual intervention** on the CMSSW Infrastructure.

Contributions to CMSSW

- Improvements on the CMSSW Software Stack
- Improvements on the CMSSW Infrastructure

Improvements on the CMSSW Software Stack

Why More Throughput?

- We must make **efficient use of computing resources**, including computing capacity.
- Faster code base gives us more flexibility to schedule jobs, to plan sample production, to incorporate a last minute necessary features, etc.

- CMS Simulation, Reconstruction, and HLT code have been used to deliver events for analysis during Run 1 and 2 of the LHC at CERN.
- **Advanced Compiler Options have shown to provide a throughput increase**.
  - Concretely, Link Time (LTO) and Profile Guided (PGO) Optimization techniques showed near *the yearly 10% aspirational performance improvement*.
  - Based on results from *Speeding up CMS simulations, reconstruction and HLT code using advanced compiler options* presented by Danilo Piparo (ACAT 2022).
- Over this last year, we have been working on **incorporating those theoretical improvements to the infrastructure**.

Link Time Optimization - LTO

Link Time Optimization Basics

LTO instruments compilation units with metadata, consulted to optimize when building shared objects. **It provides better runtime performance** through whole-program analysis and cross-module optimization.

- LTO improvements in the event loop time and event throughput showed a **2-3% speedup of Simulation, Reconstruction and HLT** code. ▸ LTO Study
- LTO got the **approval from Physics Validation** in early 2023.
- It was officially integrated around 1 year ago (end of February 2023). ▸ LTO default
- Currently, LTO is enabled the default for CMSSW (as of 13_0_X).

Link Time Optimization - LTO

LTO in external packages

- Apart from CMSSW, we compile some externals such as `geant4`, `vecgeom`, `g4hepem`, `dd4hep`, and `celeritas`, with LTO enabled.

Infrastructure Challenges

When moving LTO to production, we found it **incompatible with some of our IB "flavors"**:

IB Flavor	Discussion	Incompatibility Reason
`CUDA`	cms-sw/cmssw#33667	NVCC cannot handle LTO
`ASAN`	cms-sw/cmssw#43003	GCC optimization bug
`CLANG`	cms-sw/cmsdist#8335	Cannot link against external libraries compiled with GCC
`ppc64le` architecture	cms-sw/cmssw#40177	GCC bug in GCC 11.2.1/11.3.0. Already fixed for GCC11/12 branches[2]

Table: LTO incompatibilities in CMSSW IBs.

[2] We hope to get rid of it in the incoming GCC updates

Profile Guided Optimization - PGO

Profile Guided Optimization Basics

PGO uses profiling to **improve runtime performance**. It implies **one compilation** to build instrumented binaries, **one execution** to produce a profile of the application and **one extra compilation** to re-build from sources with information from the profile.

- PGO performs changes such as inlining, block ordering, register allocation, conditional branch optimization, etc showing **7-8% speedup**. Other findings suggest:
  - Five workflows seem to be enough to produce profiles to optimize all workflows.
  - A few tens of events are enough to produce complete profiles.

  ▸ PGO Study
- PGO does not have the approval from Physics Validation yet.
- **Merging the profiles** is not technically easy since we cannot run full CMSSW at once. Merging profiles give **coverage errors**.

  → As in LTO, we build some externals with PGO, that we profile for a CMSSW run.

Advanced Compiler Techniques - Summary

Compilation Flags

LTO

```
-flto -fipa-icf -flto-odr-type-merging -fno-fat-lto-objects
```

PGO

Generation: `-fprofile-generate -fprofile-dir="/profile"`
Execution: `-fprofile-use -fprofile-partial-training -fprofile-dir="/profile"`

Executive Summary

- LTO has immediate benefits for all CMS applications giving 2-3% speedup. **[DONE]**
- PGO is also effective giving 7-8% speedup. **[ON-GOING]**
- There is no CMS-specific configuration involved in this optimization, so **this analysis could be of interest for other experiments/projects**.

Contributions to CMSSW

- Improvements on the CMSSW Software Stack
- Improvements on the CMSSW Infrastructure

CMSSW Integration Builds

- Integration Builds are crucial to maintain CMSSW.
- All **CMSSW IBs** are automatically deployed via Jenkins CI **every 12 hours** to CernVM-FS.
  - Build for a set of active release cycles, multiple OS/archs/compilers and different IB "flavors".

```
/cvmfs/cms-ib.cern.ch/sw/x86_64/nweek-02823:
bin                 common          el9_amd64_gcc12  SITECONF          slc7_amd64_gcc11
bootstrap.sh        cvmfs           el9_amd64_gcc13  slc6_amd64_gcc472 slc7_amd64_gcc12
bootstraptmp        el8_amd64_gcc10 etc              slc6_amd64_gcc481 slc7_amd64_gcc530
cmsmon              el8_amd64_gcc11 LICENSE          slc6_amd64_gcc530 slc7_amd64_gcc630
cmsset_default.csh  el8_amd64_gcc12 logs             slc6_amd64_gcc630 slc7_amd64_gcc700
cmsset_default.sh   el9_amd64_gcc11 share            slc7_amd64_gcc10
```

  - Once a week, fully build all release cycles.
  - Around **40 IBs** are build and deployed **every day**.

▸ IB page

- The variety of software flavors and architectures, provides different approaches to **identify bugs and legacy code at an early stage**.

Non-amd64 Validation

Architectures `aarch64` and `ppc64le` have been validated (now treated as production).

Variety of IB Flavors

We have integrated new IB flavors in the infrastructure:

- **GPU IBs**: They make use of heterogeneous resources.
  - They were incorporated in CMSSW_11_3 (January 2021).
- **SKYLAKE - AV512 IBs**: As an initial attempt to support multi-vectorization architectures.
  - Including vectorization targets such as `haswell` and `skylake-avx512`.
  - They were incorporated in CMSSW_11_3 (March 2021).
- **MULTIARCHS IBs**: To add support for a standardized set of "micro-architecture levels" in the x86-64 psABI.
  - The psABI levels supported by the processor can be checked from the OS.
  - HLT is planning to move to `MULTIARCHS` this year.
  - They were incorporated in CMSSW_14_1 (January 2024).

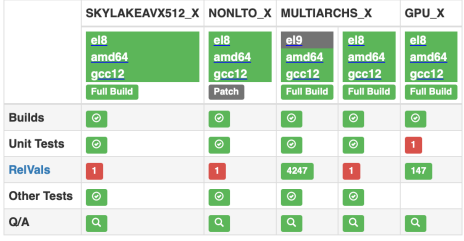

Figure: New IB flavors for CMSSW.

Resource Monitoring & New Scheduler

- The increasing workloads had an impact on the infrastructure.
- IB RelVals were run blindly and killed due to the memory usage.
- We started collecting the **resource usage** of each step of RelVals workflows in OpenSearch.

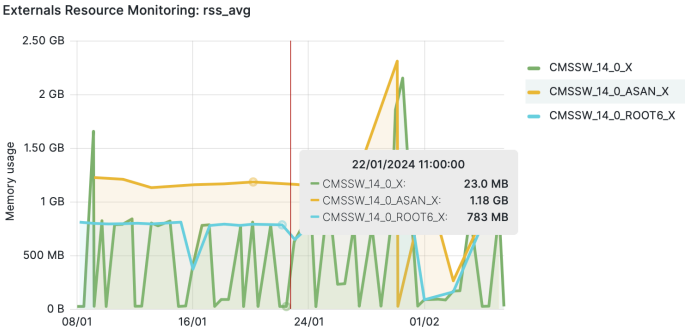

Figure: Resource monitoring dashboard for externals.

- **New workflow scheduler** makes use of the resource information to schedule future runs.
- This setup is also used for monitoring the **resource utilization when building externals**.

Infrastructure Migrations

To efficiently accommodate the increasing workloads, migration of the CernVM File System to a parallel publishing setup was required.

- Deployment jobs experienced delays due to our serial publishing setup, directly impacting on the testing time.

Design Choices

- Multi-release manager setup with **3 publishers** for `amd64` and `ppc64le`, **and a dedicated one** for `aarch64`.
- Artifacts are published in **different paths based on the architecture**.

```
[avalenzu@lxplus728 ~]$ cd /cvmfs/cms-ib.cern.ch
[avalenzu@lxplus728 cms-ib.cern.ch]$ ls sw/
aarch64  ppc64le  x86_64
[avalenzu@lxplus728 cms-ib.cern.ch]$ ls -all sw/x86_64/
total 5
drwxr-xr-x.  5 cvmfs cvmfs 115 Apr  4 09:48 .
drwxr-xr-x.  5 cvmfs cvmfs  20 Oct 31 10:26 ..
lrwxrwxrwx.  1 cvmfs cvmfs  11 Apr  4 09:48 latest -> nweek-02779
drwxr-xr-x. 26 cvmfs cvmfs  17 Apr  1 11:44 nweek-02778
drwxr-xr-x. 25 cvmfs cvmfs  56 Apr  4 06:33 nweek-02779
drwxr-xr-x.  3 cvmfs cvmfs  17 Oct 28 16:56 scramdb
-rw-r--r--.  1 cvmfs cvmfs  19 Apr  4 09:46 stratum0
lrwxrwxrwx.  1 cvmfs cvmfs  43 Apr  4 09:48 week0 -> /cvmfs/cms-ib.cern.ch/sw/x86_64/nweek-02778
lrwxrwxrwx.  1 cvmfs cvmfs  43 Apr  4 09:48 week1 -> /cvmfs/cms-ib.cern.ch/sw/x86_64/nweek-02779
```

Figure: Publishing paths are based on architecture.

CernVM-FS Parallel Publishing Migration

- **Deployment waiting times have reduced** significantly with the parallel setup.
  - Avoid blocking the publishers with non-production architectures.

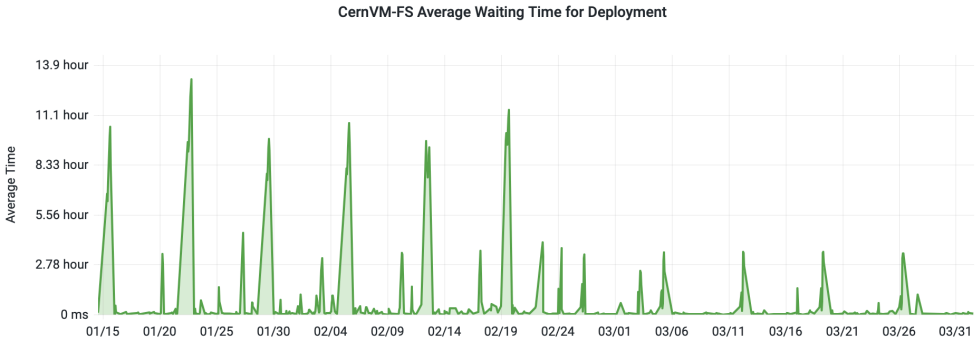

Figure: Waiting time of deployment jobs to `cms-ib.cern.ch`. The parallel setup was put in place the 20th of February 2023.

Conclusion

Overall, these advancements have improved the CMSSW Software Stack in terms of throughput and the CMSSW Infrastructure for Run 3.

- **Advanced compiler techniques** have shown promising results.
  - We will continue working to incorporate PGO to CMSSW, and assess the benefits from the infrastructure side.
- The variety of IBs provides different approaches **to identify bugs and legacy code**.
  - As well as to **incorporate computing techniques** such as multi-vectorization architectures or heterogeneous resources to CMSSW.
- We have seen major improvements with the CernVM-FS Parallel Publishing setup.
  - It has **speed up the IBs deployment**, which has an impact also on the test step.
  - The current setup allows **horizontal scaling**.
- Finally, CI is also one of our major improvement focus in order to adopt efficient methods for monitoring and scheduling, testing, and integrating the Software Stack.

Thanks!

Link Time Optimization (LTO) Workflow

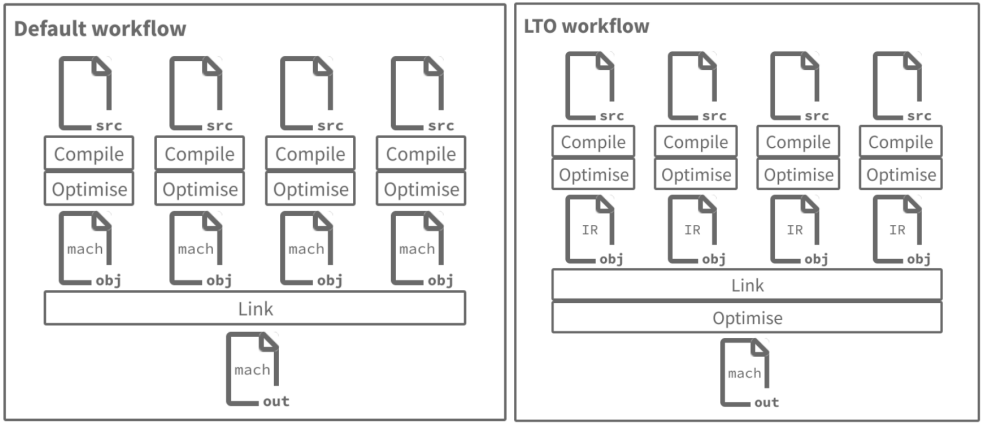

Figure: **Left:** Non-LTO compilation and linking workflow. **Right:** LTO compilation and linking workflow **[source]**.

Parallel Link Time Optimization (LTO) Workflow

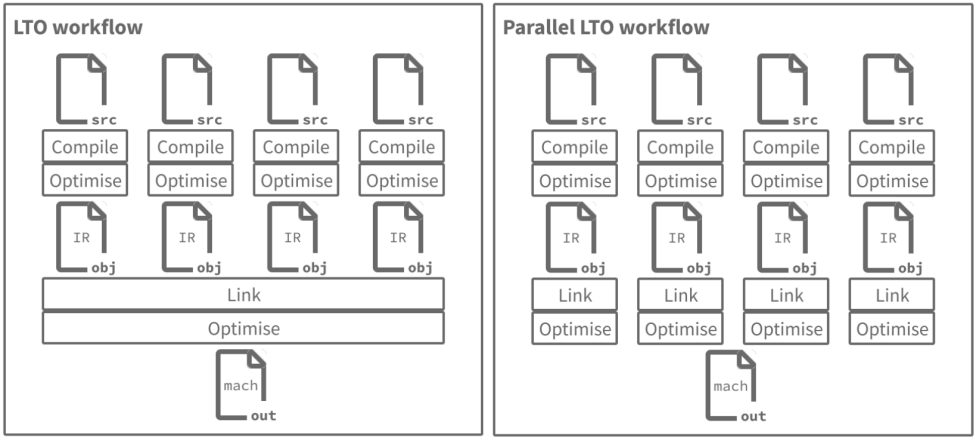

Figure: **Left:** Regular LTO compilation and linking workflow. **Right:** LTO compilation and parallel linking workflow **[source]**.

CMSSW Release Build Infrastructure

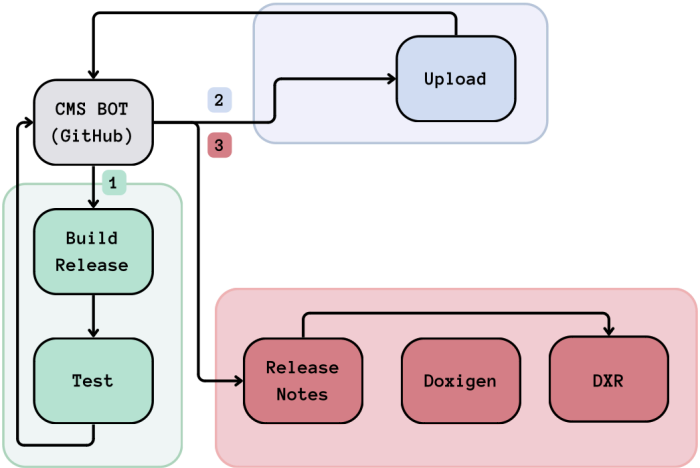

Figure: CMSSW Release Build Infrastructure schema.

CMSSW Improvements on Continuous Integration

Other enhancements to improve the CMSSW CI Infrastructure have been adopted:

1. **On-demand generation of baselines.**
   - Generate baseline only when a Pull Request (PR) needs it.
   - First PR to request it will generate the baseline and other PRs will reuse it.
   - PR tests can make use of latest IB as soon as it is available on `cvmfs`.
2. **Unit test for GPU.**
   - PR tests check for GPU resources to run tests when needed.
3. Integration of the **CRAB Computing Infrastructure** in PR testing (and IBs).
   - Submission of CMSSW jobs to distributed computing resources as part of the testing.