MG4GPU STATUS

FOR CMS–MG JOINT MEETING
24.06.04

CONTENTS

- **Summary of steps in Event Generation**
- **Inspecting Potential Bottlenecks**
- **Discussion**

EVENT GENERATION

From CMS gridpacks

Basic command for evt generation would be:

```shell
./runcmsgrid.sh $NEVT $RANDOMSEED $NB_CORE
```

In this scripts, it does:

1. Set up production environment (i.e. CMSSW)
2. Modify madevent/Cards/me5_configuration.txt (e.g. nb_core...)
3. Determine the no. of evts to be generated in each iteration.
4. Generate events. i.e.

```shell
./bin/gridrun 5000 $SEED $GRAN
```

5. Combine events / Check the no. of evts / Add scale and PDF weights to LHE files.

EVENT GENERATION

From CMS gridpacks

Basic command for evt generation would be

```shell
./runcmsgrid.sh $NEVT $RANDOMSEED $NB_CORE
```

In this scripts, it does:

1. Set up production environment (i.e. CMSSW)
2. Modify madevent/Cards/me5_configuration.txt (e.g. nb_core...)
3. Determine the no. of evts to be generated in each iteration.
4. Generate events. i.e.

```shell
./bin/gridrun 5000 $SEED $GRAN
```

5. Combine events / Check the no. of evts / Add scale and PDF weights to LHE files.

How to parallelize events?
- nb_core setting not working in current version
- Submitting many evts with single thread only utilize <500MB of GPU
- submitting multiple times requires modification in the CMS workflow, especially in the steps afterward / or maybe just parallelize iteration loop?

EVENT GENERATION

From CMS gridpacks

Basic command for evt generation would be

```shell
./runcmsgrid.sh $NEVT $RANDOMSEED $NB_CORE
```

In this scripts, it does:

1. Set up production environment (i.e. CMSSW)
2. Modify madevent/Cards/me5_configuration.txt (e.g. nb_core...)
3. **Determine the no. of evts** to be generated in each iteration.
4. Generate events. i.e.

```shell
./bin/gridrun 5000 $SEED $GRAN
```

5. Combine events / Check the no. of evts / Add scale and PDF weights to LHE files.

- Basic blocks are generating 5000 events in each iteration.
- For each iteration, need to prepare running directories again, e.g. copy & pasting ./madevent executable, ajobs, etc.
- Done by *process/madevent/bin/internal/restore_data*

FIRST BOTTLENECK

restore_data: copy & pasting, untarring each subprocess directories

```
for i in `cat subproc.mg` ; do
    cd $i
    echo $i
    rm -f ftn25 ftn26 >& /dev/null
    if [[ -e $1_results.dat ]]; then
    cp  $1_results.dat results.dat  >& /dev/null
    else
    cp  results.dat $1_results.dat  >& /dev/null
    fi
    for k in G* ; do
    if [[ ! -d $k ]]; then
        continue
    fi
    cd $k
    for j in $1_results.dat ; do
        if [[ -e $j ]] ; then
        cp  $j results.dat
        else
        cp  results.dat $j
        fi
    done
    for j in $1_ftn26.gz ; do
        if [[ -e $j ]]; then
        rm -f ftn26 >& /dev/null
        rm -f $1_ftn26 >& /dev/null
        gunzip $j
        cp $1_ftn26 ftn26
        gzip $1_ftn26
        fi
    done
    cd ../
    done
    cd ../
done
```

```
run.sh 5000 10
Now generating 5000 events with random seed 10 and granularity 1
************************************************************
*                                                          *
*                    W E L C O M E to                      *
*           M A D G R A P H 5 _ a M C @ N L O              *
*                   M A D E V E N T                        *
*                                                          *
*                *                       *                 *
*                  *       * *       *                     *
*                    * * * * 5 * * * *                     *
*                  *       * *       *                     *
*                *                       *                 *
*                                                          *
*         VERSION 3.5.3_lo_vect                            *
*                                                          *
*    The MadGraph5_aMC@NLO Development Team - Find us at   *
*    https://server06.fynu.ucl.ac.be/projects/madgraph     *
*                                                          *
*              Type 'help' for in-line help.               *
*                                                          *
************************************************************
INFO: load configuration from /srv/work/process/madevent/Cards/m
INFO: load configuration from /srv/work/mgbasedir/input/mg5_conf
INFO: load configuration from /srv/work/process/madevent/Cards/m
Using default text editor "vi". Set another one in ./input/mg5_c
No valid eps viewer found. Please set in ./input/mg5_configurati
No valid web browser found. Please set in ./input/mg5_configurat
WRITE GRIDCARD /srv/work/process/madevent
generate 5000 events
P0_gg_epemuux
P0_gg_epemddx
P0_gg_taptamuux
P0_gg_taptamddx
P0_gu_epemgu
P0_gd_epemgd
P0_gux_epemgux
P0_gdx_epemgdx
P0_gu_taptamgu
P0_gd_taptamgd
P0_gux_taptamgux
P0_gdx_taptamgdx
P0_uu_epemuu
P0_uux_epemuux
P0_dd_epemdd
P0_ddx_epemddx
P0_uxux_epemuxux
P0_dxdx_epemdxdx
P0_ud_epemud
P0_uc_epemuc
P0_uux_epemddx
P0_uux_epemccx
P0_udx_epemudx
```

It takes a bit long time to prepare running directories before actual execution of madevent...

e.g. DY+4j has 1455(412560) processes(diagrams)

FIRST BOTTLENECK

- **Simple Test - Parallelizing restore_data**
- Used GNU parallelize command for restoring each subprocesses
- Generating 5k evts for each using condor, compared w/ and w/o parallelized restore_data setup

w/o parallelization	FORTRAN	CPP	CUDA
DY+2j	6m 8s	10m 51s	9m 20s
DY+3j	19m 29s	38m 32s	25m 1s
DY+4j	180m 18s		106m 19s

Not much for CPP:
this is not a major bottleneck

w/ parallelization	FORTRAN	CPP	CUDA
DY+2j	5m 12s	10m 33s	8m 14s
DY+3j	17m 50s	35m 32s	21m 11s
DY+4j	202m 16s	x3 execution speed?🙃	64m 33s

Production time reduced a lot for DY+4j CUDA

EVENT GENERATION

From CMS gridpacks

Basic command for evt generation would be

```shell
./runcmsgrid.sh $NEVT $RANDOMSEED $NB_CORE
```

In this scripts, it does:

1. Set up production environment (i.e. CMSSW)
2. Modify madevent/Cards/me5_configuration.txt (e.g. nb_core...)
3. Determine the no. of evts to be generated in each iteration.
4. Generate events. i.e.

```shell
./bin/gridrun 5000 $SEED $GRAN
```

5. **Combine events / Check the no. of evts / Add scale and PDF weights to LHE files.**
   This part does not take much time

DISCUSSION

- **Running event generation**

- ✓ Most of the bottlenecks coming from I/O bounds → readonly gridpacks could be the option

  No need to restore data,
  No need to change from the options in CMS side

- ✓ Not sure if vectorized_cpu options are I/O bounded or slow itself
  → will try to measure actual timing for step by step

- ✓ How will going to utilize parallelization in event generation tasks?
  Making nb_core option runnable would seamlessly intergrated to CMS workflow (e.g. Hadronization...)
  If not, we might have to change the workflow from the CMS side itself

- ✓ Best choice (and the simplest) would be readonly + parallelize iterations

- **Regarding the physics processes**

- ✓ Most speed-ups are observed for high-multiplicity final states (DY+4j),
  both for the gridpack production and the event generation.

- ✓ Might useful in 2D-binned central samples(e.g. DY with (jet, HT)-binned)

- ✓ For BSM cases with high final-state multiplicity? (e.g. pp > go go > 6j...)

BACK UPS

GRIDPACK PRODUCTION

ENVIRONMENTS

HPCs

- ~~lxplus800~~(GPU): AMD EPYC 7313 16-core processor (AVX2 support), A100 GPU → repeatedly halted
- SNU-server: Intel(R) Xeon(R) CPU E5-2699 v3 (72 cores, AVX2 support), no GPU
  → tested FORTRAN/CPP gridpacks
- lxplus condor: possible to use A100 GPU nodes with 16 AMD cores with isolated environment
  ~~restriction - 100 GB storage(based on AFS area), job halted after 3 days~~
  more than O(100) GB storage can be used in the node
  can access EOS area via xrootd
  still testing on > a week usage

NEW!

Sidenotes

- For testing CPU usage in lxplus condor,
  randomly matches to the nodes with 48/64 cores + AVX2 supports
- There is 4 A100 GPU node but the gridpack production failed if there is multiple GPUs

Might possible to use it for further testing....?

PRODUCTION TIME

Environments

- T
- I
- S
- I
- S

OpenStack project with GPU flavors in pass-through mode

This option is identical to the one described in the Projects section, except that GPU flavors will be assigned to your project. You can then launch instances with GPUs. The available flavors are:

Flavor Name	GPU	RAM	vCPUs	Disk	Ephemeral	Comments
g1.xlarge	V100	16 GB	4	56 GB	96 GB	[^1], deprecated
g1.4xlarge	V100 (4x)	64 GB	16	80 GB	528 GB	[^1]
g2.xlarge	T4	16 GB	4	64 GB	192 GB	[^1], deprecated
g2.5xlarge	T4	168 GB	28	160 GB	1200 GB	[^1]
g3.xlarge	V100S	16 GB	4	64 GB	192 GB	[^1]
g3.4xlarge	V100S (4x)	64 GB	16	128 GB	896 GB	[^1]
g4.p1.40g	A100 (1x)	120 GB	16	600 GB	-	[^1], AMD CPUs
g4.p2.40g	A100 (2x)	240 GB	32	1200 GB	-	[^1], AMD CPUs
g4.p4.40g	A100 (4x)	480 GB	64	2400 GB	-	[^1], AMD CPUs

- S

atedly halted

nment

Least validation

Compatible

	FORTRAN [pb]	CPP [pb]	CUDA [pb]
DY+0j	5704 \pm 10.11	5711 \pm 1.053	5710 \pm 1.484
DY+1j	3335 \pm 7.462	3535 \pm 1.263	3536 \pm 1.442
DY+2j	2228 \pm 3.143	2236 \pm 0.503	2237 \pm 0.4618
DY+3j	1375 \pm 1.265	1387 \pm 0.3515	1385 \pm 0.3288
DY+4j	883.4 \pm 0.3813	845.8 \pm 0.21	job running (> a week)

A bit large errors / different xsecs for FORTRAN?

PRODUCTION TIME

Results (full time)

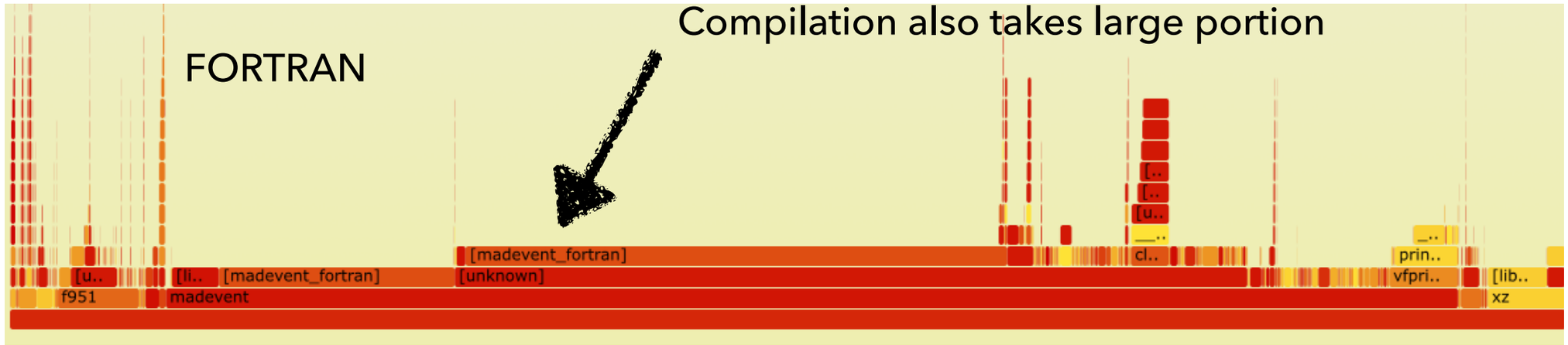
	72 Intel cores	72 Intel cores	batch job 16 AMD cores + 1 A100 GPU
	FORTRAN	CPP	CUDA
DY+0j	11m 31s	6m 32s	8m 1s
DY+1j	9m 28s	11m 7s	17m 20s
DY+2j	17m 15s	39m 33s	71m 25s
DY+3j	185m 35s	316m 58s	274m 44s
DY+4j	19362m 13s 13.5 days...	16242m 59s 11.3 days...	7682m 17s 5.3 days

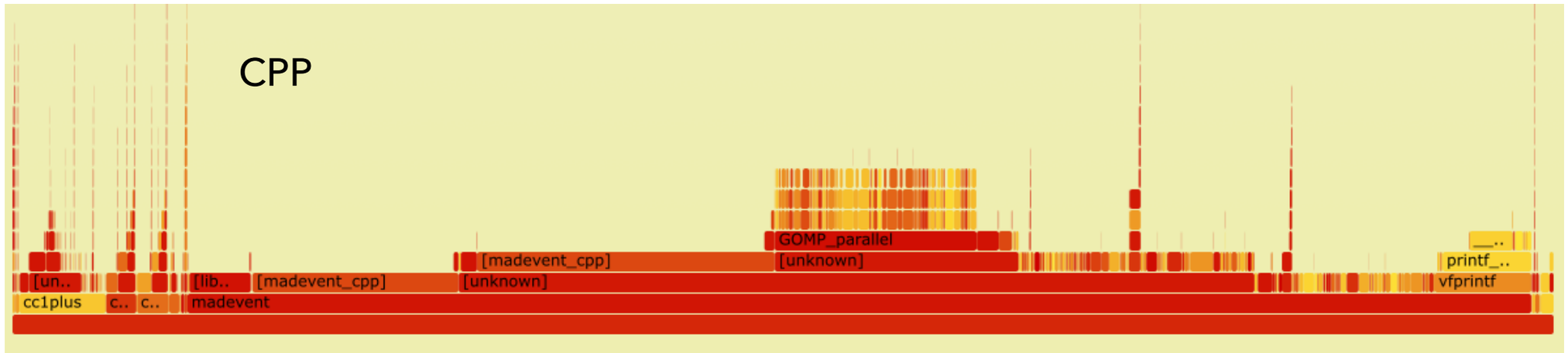
- Used time command to estimate full production time — The only improvement...why?
- Only CUDA environment is isolated - might exist some interruption by other jobs
- Improvement can only be seen in DY+4j...

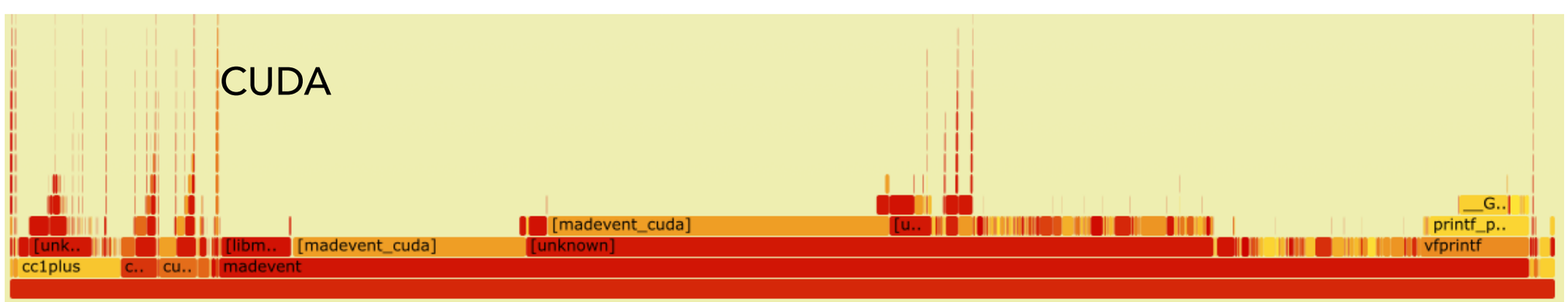
FLAMEGRAPHS

DY+2j

Most of the time consuming part is still madevent...
Compilation also takes large portion







All jobs tested in lxplus800 node

PRODUCTION TIME

Results (full time)

	72 Intel cores	72 Intel cores	batch job 16 AMD cores + 1 A100 GPU
	FORTRAN	CPP	CUDA
DY+0j	11m 31s	6m 32s	8m 1s

```
INFO: Building madevent in madevent_interface.py with 'FORTRAN' matrix elements
INFO:     P0_dxsx_taptamdxsx
INFO: Building madevent in madevent_interface.py with 'FORTRAN' matrix elements
INFO:     P0_uux_epemgg
INFO: Building madevent in madevent_interface.py with 'FORTRAN' matrix elements
INFO:     P0_ddx_epemgg
INFO: Building madevent in madevent_interface.py with 'FORTRAN' matrix elements
INFO:     P0_uux_taptamgg
INFO: Building madevent in madevent_interface.py with 'FORTRAN' matrix elements
INFO:     P0_ddx_taptamgg
INFO: Building madevent in madevent_interface.py with 'FORTRAN' matrix elements
```

Compilation (ME)

```
INFO:  Idle: 1,  Running: 8,  Completed: 281 [ current time: 16h42 ]
INFO:  Idle: 0,  Running: 9,  Completed: 281 [  0.02s  ]
INFO:  Idle: 0,  Running: 6,  Completed: 284 [  3.3s  ]
INFO:  Idle: 0,  Running: 3,  Completed: 287 [  10.5s  ]
INFO:  Idle: 0,  Running: 0,  Completed: 290 [  17.7s  ]
INFO:  Idle: 0,  Running: 0,  Completed: 290 [  17.7s  ]
sum of cpu time of last step: 58m04s
```

Execution (ME)

- Improvement can only be seen in DY+4j...

PRODUCTION TIME

Results (ME calculation - execution)

	72 Intel cores	72 Intel cores (AVX2)	batch job 16 AMD cores + 1 A100 GPU
	FORTRAN	CPP	CUDA
DY+0j	1.1s	24.4s	17.7s
DY+1j	4.9s	48.4s	31.6s
DY+2j	20.3s	4m 44s	2m 29s
DY+3j	1h 59m	3h 19m	33m 34s
DY+4j	315h 38m	247h 45m	108h 45m

- Only CUDA environment is isolated - might exist some interruption by other jobs
- Checked x4(x3) improvement in DY+3j(4j)
- **Compilation also takes big portion of the production**

SUMMARY

- **Comparing timing estimations for FORTRAN/CPP/CUDA**
- Not much, even worse timing improvement compared to FORTRAN
- Major bottleneck is **compilation time** for CUDA
- Both compilation and execution slow in CPP?
- With current usage, expecting highest gain in processes with **small no. of diagrams / >= 6 final states**

PREVIOUS PARTIAL RESULTS

Standalone

Process	x-sec[pb]	error[pb]	diagrams (processes)	timing (FORTRAN)	timing (CPP)	timing (CUDA)
DY+0j	5711	1.054	30 (15)	11m 48s	2m 12s	6m 36s
DY+1j	3535	1.263	180 (45)	14m 3s	2m 58s	9m 50s
DY+2j	2236	0.5005	3120 (285)	34m 12s	8m 18s	41m 31s
DY+3j	1386	0.3747	27600 (435)	230m 38s	31m 24s	125m 25s
DY+4j			412560 (1455)			

PRODUCTION TIME (ALL LXPLUS CONDOR BATCH)

Results

	48 Intel	48 Intel, avx2	16 AMD + 1 A100 GPU
	FORTRAN	CPP	CUDA
DY+0j	7m 59s	8m 38s	8m 1s
DY+1j	9m 27s	21m 3s	17m 20s
DY+2j	21m 24s	85m 6s	71m 25s
DY+3j	293m 38s	698m 41s	274m 44s
DY+4j	job running (> a week)	18509m 11s	7682m 17s

64 Intel, avx2

BACK UP: HOW TO PRODUCE CMS GRIDPACKS

Assuming running the scripts in lxplus (but the only requirement is cvmfs)

- ✓ 1. clone genproduction repo
  git clone https://github.com/choij1589/genproductions.git
  checkout mg4gpu

- ✓ 2. go to /bin/Madgraph5_aMCatNLO
  cd /bin/Madgraph5_aMCatNLO

- ✓ 3. Basic usage of the gridpack_generation script is
  ./gridpack_generation $PROCESSNAME $CARDDIR

- ✓ 4. I have put the GPU cards in cards/13p6TeV/mg4gpu, for DY+0j with CUDA just run
  ./gridpack_generation DY0j_LO_5f_CUDA cards/13p6TeV/mg4gpu/DY0j_LO_5f_CUDA

PROJECT UPDATES

- **Integrating MG4GPU to CMS-genproduction** [genproduction/mg4gpu]
- ✓ Based on the master branch(for RUN3 production) - updated patches for MG352 / mg4gpu
- ✓ Workflow: Environment setup(e.g. CMSSW / CUDA) - **download MG** - apply patches
  - compile processes - ME calc. - systematic calc. - **tarring gridpack**

  Major bottlenecks for large gridpacks
- ✓ Previously used git clone for downloading mg4gpu: large repo, takes ~ 10 min. to clone
  - ⇨ Compressed the repo in EOS area, untar the repo rather than downloading: ~ 4 min.
  - ⇨ No change in tarring gridpack, can be improved by removing unnecessary files / multithreading
- ✓ Two major patches for mg4gpu side

```
diff --git a/madgraph/various/systematics.py b/madgraph/various/systematics.py
index 28eaed0..5f787de 100644
--- a/madgraph/various/systematics.py
+++ b/madgraph/various/systematics.py
@@ -169,7 +169,7 @@ class Systematics(object):
         self.orig_ion_pdf = False
         self.ion_scaling = ion_scaling
         self.only_beam = only_beam
-        if isinstance(self.banner.run_card, banner_mod.RunCardLO):
+        if self.banner.run_card.LO:
             self.is_lo = True
             if not self.banner.run_card['use_syst']:
                 raise SystematicsError('The events have not been generated with use_syst=True. Cannot evaluate systematics error on thes
```

self.banner.run_card does not work with use_syst option

PROJECT UPDATES

- **Integrating MG4GPU to CMS-genproduction** [genproduction/mg4gpu]
- ✓ Based on the master branch(for RUN3 production) - updated patches for MG352 / mg4gpu
- ✓ Workflow: Environment setup(e.g. CMSSW / CUDA) - **download MG** - apply patches
  - compile processes - ME calc. - systematic calc. - **tarring gridpack**

  Major bottlenecks for large gridpacks
- ✓ Previously used git clone for downloading mg4gpu: large repo, takes ~ 10 min. to clone
  - ⇨ Compressed the repo in EOS area, untar the repo rather than downloading: ~ 4 min.
  - ⇨ No change in tarring gridpack, can be improved by removing unnecessary files / multithreading
- ✓ Two major patches for mg4gpu side

```
diff --git a/madgraph/interface/madevent_interface.py b/madgraph/interface/madevent_interface.py
index 8c509e83f..e6e7bd0dc 100755
--- a/madgraph/interface/madevent_interface.py
+++ b/madgraph/interface/madevent_interface.py
@@ -3966,7 +3966,8 @@ Beware that this can be dangerous for local multicore runs.""")
         Pdir = set([os.path.dirname(G) for G in Gdir])
         for P in Pdir:
             allG = misc.glob('G*', path=P)
-            for G in allG:
+            filG = [f for f in allG if not os.path.basename(f).startswith('Gpu')]
+            for G in filG:
                 if pjoin(P, G) not in Gdir:
                     logger.debug('removing %s', pjoin(P,G))
                     shutil.rmtree(pjoin(P,G))
```

some files start with "Gpu*" and erased when clearing some directories like G3*...

PROJECT UPDATES

- **Integrating MG4GPU to CMS-genproduction** [genproduction/mg4gpu]
- ✓ Based on the master branch(for RUN3 production) - updated patches for MG352 / mg4gpu
- ✓ Workflow: Environment setup(e.g. CMSSW / CUDA) - **download MG** - apply patches
  - compile processes - ME calc. - systematic calc. - **tarring gridpack**

  Major bottlenecks for large gridpacks
- ✓ Previously used git clone for downloading mg4gpu: large repo, takes ~ 10 min. to clone
  - ⇨ Compressed the repo in EOS area, untar the repo rather than downloading: ~ 4 min.
  - ⇨ No change in tarring gridpack, can be improved by removing unnecessary files / multithreading
- ✓ Two major patches for mg4gpu side
- ✓ Tested gridpack generation time with DY+0/1/2/3/4j processes [run cards]

```
import model sm-no_b_mass
set nb_core 10

define p = u d c s b u~ d~ c~ s~ b~ g
define j = p
define ell+ = e+ mu+ ta+
define ell- = e- mu- ta-
define nu = ve vm vt
define nubar = ve~ vm~ vt~

generate p p > ell+ ell- j j j j @0

output madevent_gpu DY4j_LO_5f_CUDA -nojpeg
```

DY4j_LO_5f_CUDA_proc_card.dat

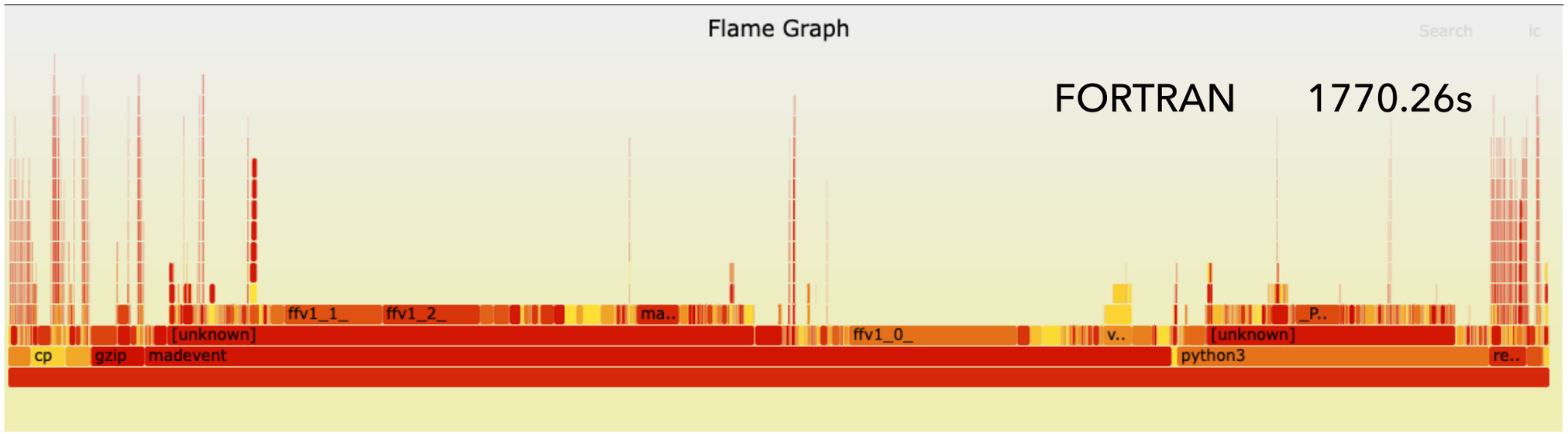
```
set sde_strategy 1
set vector_size 8192
set cudacpp_backend CUDA
```

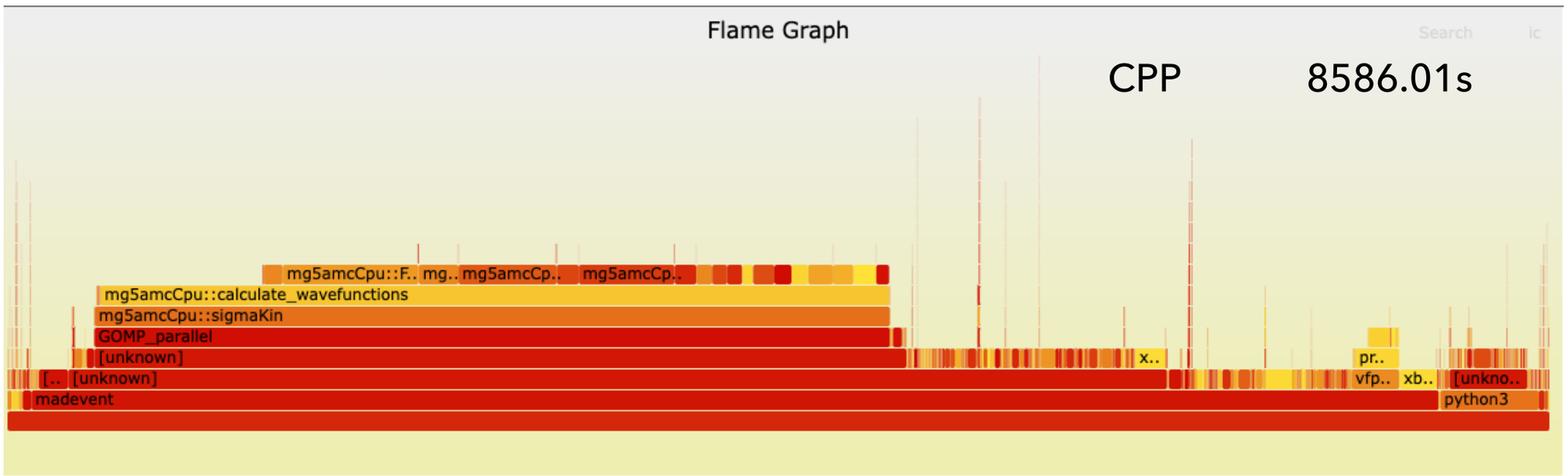
DY4j_LO_5f_CUDA_customizecards.dat

FLAMEGRAPHS

DY+3j (generating 20000 events)

svg files in [lxplus]



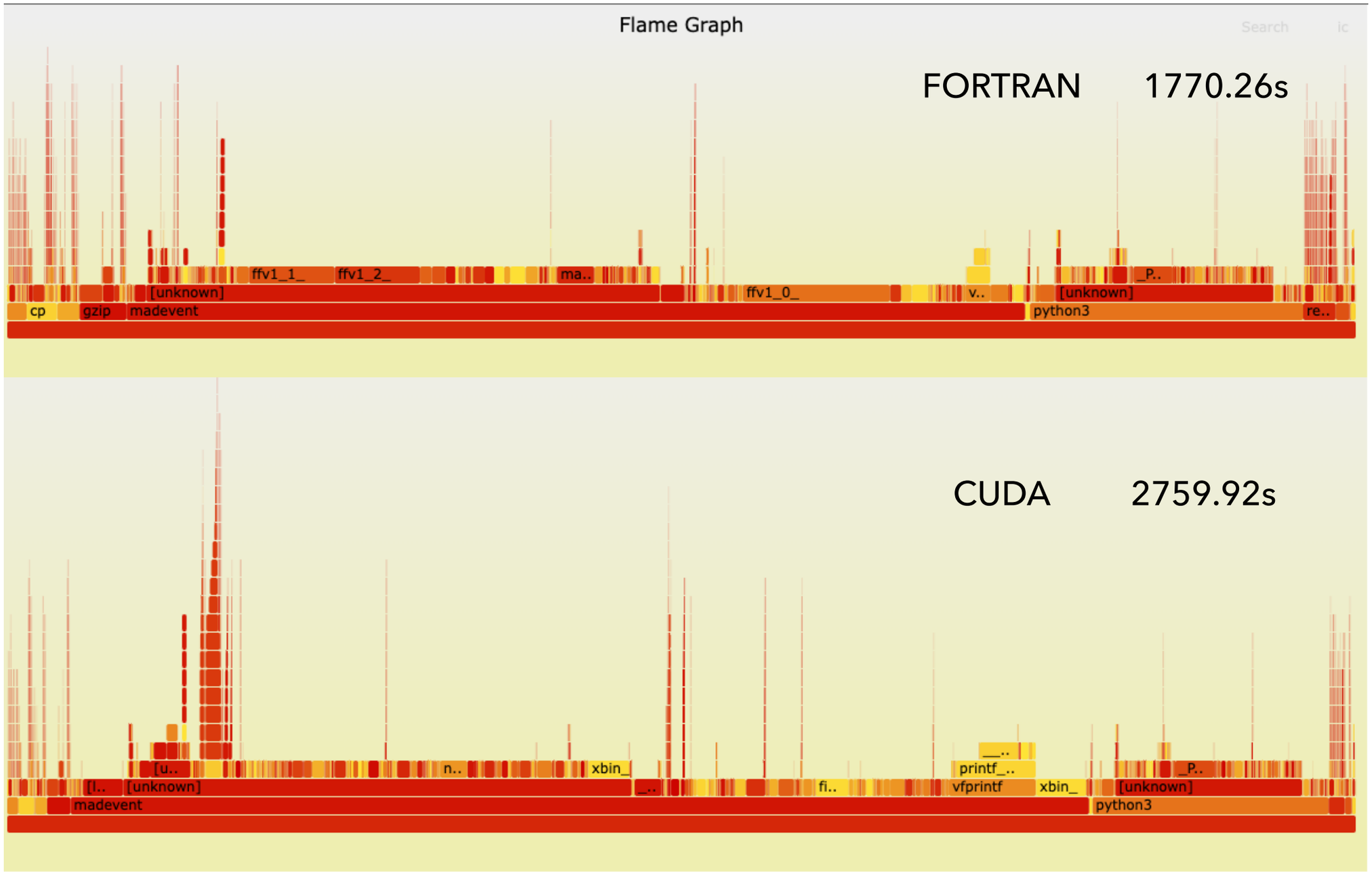


All jobs tested in lxplus8-gpu node

FLAMEGRAPHS

DY+3j (generating 20000 events)

svg files in [lxplus]



All jobs tested in lxplus8-gpu node