MG4GPU STATUS

FOR CMS-MG JOINT MEETING
24.07.02

CONTENTS

- **Updating MG4GPU repo / Renewing Gridpacks**
- **Partial Results**
- **For CHEP**

UPDATING REPO

Renewing gridpacks with latest MG4GPU github

- Ends with floating-point exception with CMS run_cards.dat (CUDA)

```
 getting user params
Enter number of events and max and min iterations:
 Number of events and iterations        1000          5          3
Enter desired fractional accuracy:
 Desired fractional accuracy:   0.10000000000000001
Enter 0 for fixed, 2 for adjustable grid:
Suppress amplitude (0 no, 1 yes)?
 Using suppressed amplitude.
Exact helicity sum (0 yes, n = number/event)?
 Explicitly summing over helicities

Program received signal SIGFPE: Floating-point exception - erroneous arithmetic operation.

Backtrace for this error:
#0  0x7f7da61175af in ???
#1  0x4d9d88 in ???
#2  0x47a6b4 in ???
#3  0x46eec0 in ???
#4  0x46ef41 in ???
#5  0x46c7b6 in ???
#6  0x438444 in ???
#7  0x439628 in ???
#8  0x439b2d in ???
#9  0x44bfe8 in ???
#10  0x4374e3 in ???
#11  0x437924 in ???
#12  0x7f7da61037e4 in ???
#13  0x41b03d in ???
#14  0xffffffffffffffff in ???
```

- Also compilation failure with CPP (see BackUp)
- Original CMS run_cards.dat uses ptj / ptl = 0 cut → Can be fixed using ptl = 10 cut for DY+0/1j, ptl = 10 & ptj = 10 for DY+2j (not tested for 3/4j)
- Preliminary tests done using renewed run_cards.dat

PARTIAL RESULTS

- **Parallelized restore_data**
- Tested the production time of gridpack generation with renewed restore_data
- Partial results for DY+<4j

lxplus condor node
Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz / 16 Threads

lxplus8-gpu node
AMD EPYC 7313 16-Core Processor / 16 Threads + A100 GPU

w/o parallelization	FORTRAN	CPP	CUDA
DY+0j	7m 59s	x	16m 16s
DY+1j	9m 27s	x	23m 8s
DY+2j	21m 24s	x	79m 14s
DY+3j	293m 38s	x	x

w/ parallelization	FORTRAN	CPP	CUDA
DY+0j	8m 2s	x	16m 1s
DY+1j	11m 1s	x	23m 34s
DY+2j	16m 28s	x	79m 31s
DY+3j	287m 17s	x	x

Not much difference - Still Compilation is the major bottleneck

PARTIAL RESULTS

But Expecting improvements in Event Generation

- **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
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

Not much for CPP:
this is not a major bottleneck

Production time reduced a lot for DY+4j CUDA

PARTIAL RESULTS

Compilation

- nb_core option does not work widely
- Can see using whole threads while compiling single process

```
survey  pilotrun --accuracy=0.01 --points=2000 --iterations=8 --gridpack=.true.
INFO: compile directory
INFO: Using LHAPDF v6.4.0 interface for PDFs
compile Source Directory
Using random number seed offset = 21
INFO: Running Survey
Creating Jobs
Working on SubProcesses
INFO:     P0_gu_epemu
INFO: Building madevent in madevent_interface.py with 'cuda' matrix elements
INFO:     P0_gd_epemd
INFO: Building madevent in madevent_interface.py with 'cuda' matrix elements
INFO:     P0_gux_epemux
INFO: Building madevent in madevent_interface.py with 'cuda' matrix elements
INFO:     P0_gdx_epemdx
INFO: Building madevent in madevent_interface.py with 'cuda' matrix elements
INFO:     P0_gu_taptamu
INFO: Building madevent in madevent_interface.py with 'cuda' matrix elements
INFO:     P0_gd_taptamd
```

htop (ssh)

```
  0[||||||||||||||||78.4%]  4[||||||||||||||   64.1%]   8[|||||||||||||    58.3%] 12[|||||||||||||    56.2%]
  1[||||||||||||||||74.4%]  5[||||||||||||||   64.1%]   9[||||||||||||||||85.1%] 13[||||||||||||||||87.0%]
  2[||||||||||||||||91.0%]  6[||||||||||||||||85.1%]  10[|||||||||||||    56.2%] 14[||||||||||||||||78.0%]
  3[||||||||||||||||74.8%]  7[||||||||||||||| 69.0%]  11[||||||||||||||| 68.7%] 15[||||||||||||||||84.8%]
Mem[|||||||||||||||||||||||||          6.57G/115G] Tasks: 46, 1 thr, 0 kthr; 12 running
Swp[||                                 1.14M/50.0G] Load average: 1.92 1.65 2.37
                                                   Uptime: 3 days, 15:27:25
```

Main | I/O

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
2567884	choij	20	0	0	0	0	R	94.6	0.0	0:01.38	/cvmfs/cms.cern.ch/el8_amd64_gcc11/external/gc
2568094	choij	20	0	214M	175M	15292	R	45.0	0.1	0:00.60	cicc --c++17 --gnu_version=110401 --display_er
2568097	choij	20	0	214M	177M	15268	R	42.8	0.2	0:00.57	cicc --c++17 --gnu_version=110401 --display_er
2568098	choij	20	0	210M	174M	15428	R	39.8	0.1	0:00.53	cicc --c++17 --gnu_version=110401 --display_er
2568100	choij	20	0	179M	142M	15228	R	31.5	0.1	0:00.42	cicc --c++17 --gnu_version=110401 --display_er
2568101	choij	20	0	184M	147M	15368	R	31.5	0.1	0:00.42	cicc --c++17 --gnu_version=110401 --display_er
2568074	choij	20	0	63184	21540	11120	R	28.5	0.0	0:00.38	/cvmfs/cms.cern.ch/el8_amd64_gcc11/external/gc
2568102	choij	20	0	144M	107M	15316	R	20.3	0.1	0:00.27	cicc --c++17 --gnu_version=110401 --display_er
2568103	choij	20	0	132M	98040	15312	R	18.0	0.1	0:00.24	cicc --c++17 --gnu_version=110401 --display_er
2568104	choij	20	0	116M	81256	15204	R	15.0	0.1	0:00.20	cicc --c++17 --gnu_version=110401 --display_er
2568105	choij	20	0	74316	36848	15284	R	4.5	0.0	0:00.06	cicc --c++17 --gnu_version=110401 --display_er
2568106	choij	20	0	0	0	0	R	2.3	0.0	0:00.03	ptxas
2567674	choij	20	0	28912	4216	3172	R	0.8	0.0	0:00.04	htop
2410612	choij	20	0	89904	9648	7964	S	0.0	0.0	0:01.20	/usr/lib/systemd/systemd --user
2410613	choij	20	0	314M	6124	24	S	0.0	0.0	0:00.00	(sd-pam)
2410638	choij	20	0	161M	6468	5096	S	0.0	0.0	0:00.66	sshd: choij@pts/1
2410640	choij	20	0	72292	9396	4584	S	0.0	0.0	0:04.78	-zsh
2432635	choij	20	0	84816	5536	5048	S	0.0	0.0	0:00.03	/usr/bin/dbus-daemon --session --address=syste
2432645	choij	20	0	84816	5536	5048	S	0.0	0.0	0:00.00	/usr/bin/dbus-daemon --session --address=syste
2448965	choij	20	0	161M	5600	4236	S	0.0	0.0	0:00.56	sshd: choij@pts/2
2448968	choij	20	0	72304	9456	4632	S	0.0	0.0	0:03.65	-zsh
2558529	choij	20	0	161M	5660	4296	S	0.0	0.0	0:00.33	sshd: choij@pts/3
2558530	choij	20	0	72160	9168	4492	S	0.0	0.0	0:00.73	-zsh
2558991	choij	20	0	10408	3132	2324	S	0.0	0.0	0:00.00	/bin/bash ./gridpack_generation.sh DY2j_LO_5f_
2558996	choij	20	0	11220	3436	1728	S	0.0	0.0	0:00.05	/bin/bash ./gridpack_generation.sh DY2j_LO_5f_
2558997	choij	20	0	4476	940	864	S	0.0	0.0	0:00.28	tee /eos/user/c/choij/MG4GPU/GPUtestV7_1/genpr

PARTIAL RESULTS

- **Compilation**
- ✓ Done by misc.compile
- ✓ Can see using whole threads while compiling single process → Is it the best working point?

```
511  #===============================================================================
512  def compile(arg=[], cwd=None, mode='fortran', job_specs = True, nb_core=1 ,**opt):
513      """compile a given directory"""
514
515      if 'nocompile' in opt:
516          if opt['nocompile'] == True:
517              if not arg:
518                  return
519              if cwd:
520                  executable = pjoin(cwd, arg[0])
521              else:
522                  executable = arg[0]
523              if os.path.exists(executable):
524                  return
525          del opt['nocompile']
526
527      cmd = ['make']
528      try:
529          if nb_core > 1:
530              cmd.append('-j%s' % nb_core)
531          cmd += arg
532          p = subprocess.Popen(cmd, stdout=subprocess.PIPE,
533                               stderr=subprocess.STDOUT, cwd=cwd, **opt)
534          (out, err) = p.communicate()
535      except OSError as error:
536          if cwd and not os.path.exists(cwd):
537              raise OSError('Directory %s doesn\'t exists. Impossible to run make' % cwd)
538          else:
539              error_text = "Impossible to compile %s directory\n" % cwd
540              error_text += "Trying to launch make command returns:\n"
541              error_text += "    " + str(error) + "\n"
542              error_text += "In general this means that your computer is not able to compile."
543              if sys.platform == "darwin":
544                  error_text += "Note that MacOSX doesn\'t have gmake/gfortan install by default.\n"
545                  error_text += "Xcode contains gmake. For gfortran we advise: http://hpc.sourceforge.net/"
546              raise MadGraph5Error(error_text)
```

From [link]

Some ideas...

1. Make compilation fast itself
2. Assume compilation done in HPCs
- Split jobs (e.g. make -j4 with 4 parallel processors)

FOR CHEP

- **Highlight from my side will be**

- The bottlenecks in CMS production environments.
  Compilation time & restoring data time for both gridpack generation & event generation

- Final Improvement (should be approved by CMS)
  → What would be the reference point?
  1. FORTRAN production
  2. Original CUDA production without parallelized compile & restore_data
  3. Both?

- Let's see how the final numbers would be.

- Visualization? - Flamegraphs are hard to wee these bottlenecks

BACK UPS

UPDATING REPO

Renewing gridpacks with latest MG4GPU github

Compilation failed with CPP

```
Creating Jobs
Working on SubProcesses
INFO:     P0_uux_epem
INFO: Building madevent in madevent_interface.py with 'cpp' matrix elements
Start waiting for update. (more info in debug mode)
^[[1;31mError detected in "generate_events pilotrun"
write debug file /srv/work/genproductions/bin/MadGraph5_aMCatNLO/DY0j_LO_5f_CPP/DY0j_LO_5f_CPP_gridpack/work/processtmp/pilotrun_tag_1_debug.log
If you need help with this issue please contact us on https://answers.launchpad.net/mg5amcnlo
str : [Fail 5 times]
     A compilation Error occurs when trying to compile /srv/work/genproductions/bin/MadGraph5_aMCatNLO/DY0j_LO_5f_CPP/DY0j_LO_5f_CPP_gridpack/work/processtmp/SubProcesses/P0_uux_epem.
    The compilation fails with the following output message:
        make BACKEND=cppauto ./madevent_cpp
        make[1]: Entering directory '/srv/work/genproductions/bin/MadGraph5_aMCatNLO/DY0j_LO_5f_CPP/DY0j_LO_5f_CPP_gridpack/work/processtmp/SubProcesses/P0_uux_epem'
        make -f cudacpp.mk
        make[2]: Entering directory '/srv/work/genproductions/bin/MadGraph5_aMCatNLO/DY0j_LO_5f_CPP/DY0j_LO_5f_CPP_gridpack/work/processtmp/SubProcesses/P0_uux_epem'
        BACKEND=cppavx2 (was cppauto)
        OMPFLAGS=-fopenmp
        FPTYPE='d'
        HELINL='0'
        HRDCOD='0'
        HASCURAND=hasCurand
        HASHIPRAND=hasNoHiprand
        Building in BUILDDIR=. for tag=avx2_d_inl0_hrd0_hasCurand_hasNoHiprand (USEBUILDDIR != 1)
        g++  -I. -I../../src -O3  -std=c++17 -Wall -Wshadow -Wextra -ffast-math  -fopenmp  -march=haswell  -DMGONGPU_FPTYPE_DOUBLE -DMGONGPU_FPTYPE2_DOUBLE -fPIC -DUSE_NVTX -I/cvmfs/cms.cern.ch/el8_amd64_gcc11/cms/cmssw/CMSSW_13_2_9/external/el8_amd64_gcc11/include/ -DMGONGPU_HAS_NO_HIPRAND -c check_sa.cc -o check_sa_cpp.o
        g++  -I. -I../../src -O3  -std=c++17 -Wall -Wshadow -Wextra -ffast-math  -fopenmp  -march=haswell  -DMGONGPU_FPTYPE_DOUBLE -DMGONGPU_FPTYPE2_DOUBLE -fPIC -DMGONGPU_HAS_NO_HIPRAND -I/cvmfs/cms.cern.ch/el8_amd64_gcc11/cms/cmssw/CMSSW_13_2_9/external/el8_amd64_gcc11/include/ -c CurandRandomNumberKernel.cc -o CurandRandomNumberKernel_cpp.o
        CurandRandomNumberKernel.cc:14:10: fatal error: curand.h: No such file or directory
           14 | #include "curand.h"
              |          ^~~~~~~~~~
        compilation terminated.
        make[2]: *** [cudacpp.mk:700: CurandRandomNumberKernel_cpp.o] Error 1
        make[2]: *** Waiting for unfinished jobs....
        In file included from timermap.h:17,
                         from check_sa.cc:26:
        nvtx.h:22:10: fatal error: nvtx3/nvToolsExt.h: No such file or directory
           22 | #include "nvtx3/nvToolsExt.h"
              |          ^~~~~~~~~~~~~~~~~~~
        compilation terminated.
        make[2]: *** [cudacpp.mk:700: check_sa_cpp.o] Error 1
        make[2]: Leaving directory '/srv/work/genproductions/bin/MadGraph5_aMCatNLO/DY0j_LO_5f_CPP/DY0j_LO_5f_CPP_gridpack/work/processtmp/SubProcesses/P0_uux_epem'
        make[1]: *** [makefile:142: .cudacpplibs] Error 2
        make[1]: Leaving directory '/srv/work/genproductions/bin/MadGraph5_aMCatNLO/DY0j_LO_5f_CPP/DY0j_LO_5f_CPP_gridpack/work/processtmp/SubProcesses/P0_uux_epem'
        make: *** [makefile:173: madevent_cpp_link] Error 2
```

EVENT GENERATION

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)

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...)

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

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

atedly halted

nment

XSECS

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)

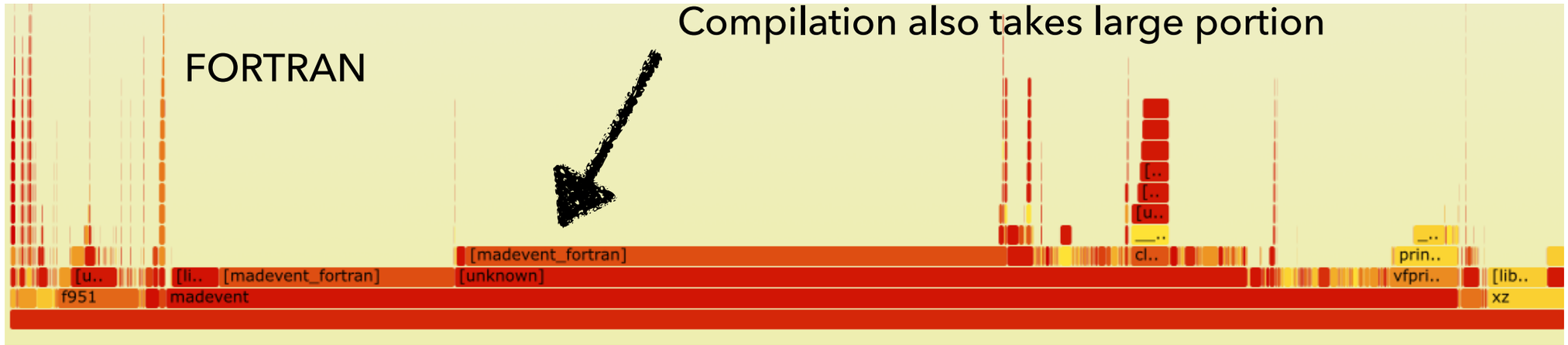
	72 Intel cores	72 Intel cores	batch job 16 AMD cores + 1 A100 GPU
	FORTRAN	CPP	CUDA
DY+0j	11m 31s	6m 32s	8m 1s
DY+1j	9m 28s	11m 7s	17m 20s
DY+2j	17m 15s	39m 33s	71m 25s
DY+3j	185m 35s	316m 58s	274m 44s
DY+4j	19362m 13s 13.5 days...	16242m 59s 11.3 days...	7682m 17s 5.3 days

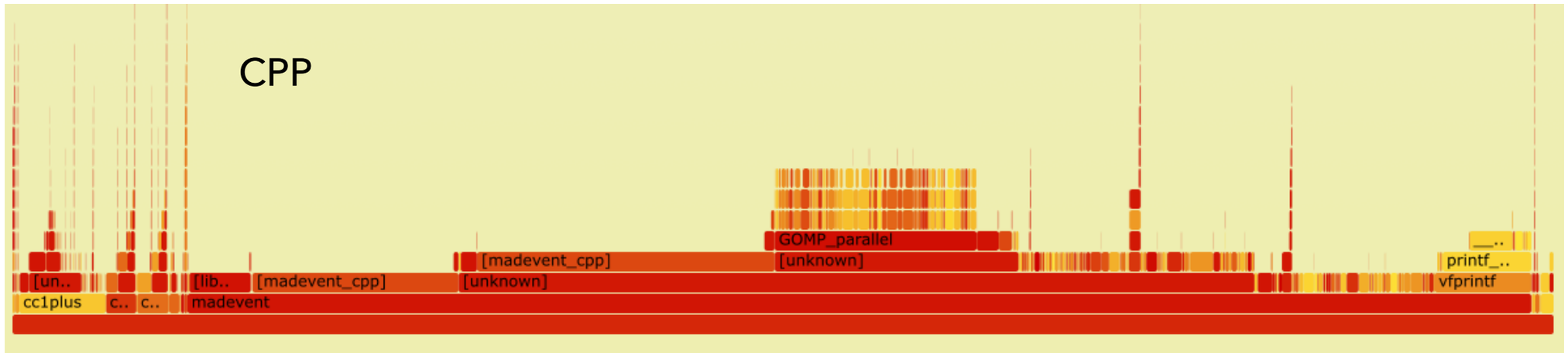
- Used time command to estimate full production time — The only improvement...why?
- Only CUDA environment is isolated - might exist some interruption by other jobs
- Improvement can only be seen in DY+4j...

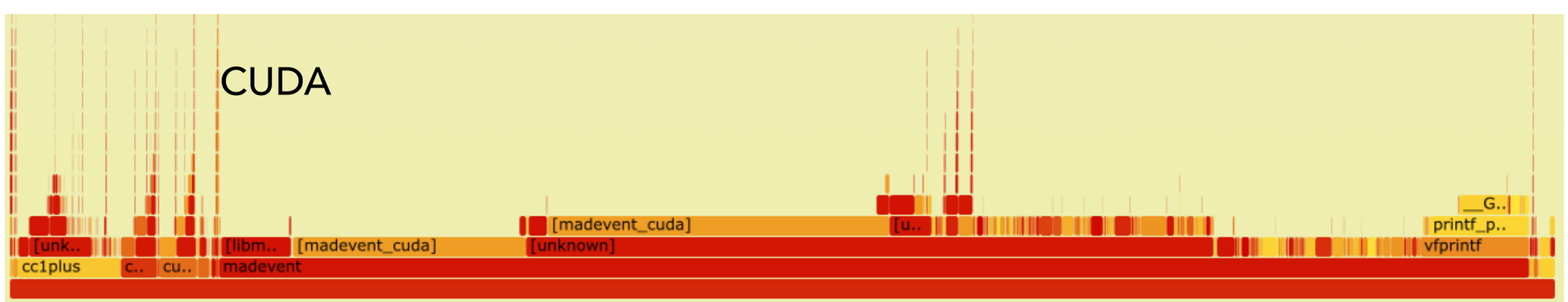
FLAMEGRAPHS

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...

ys?

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

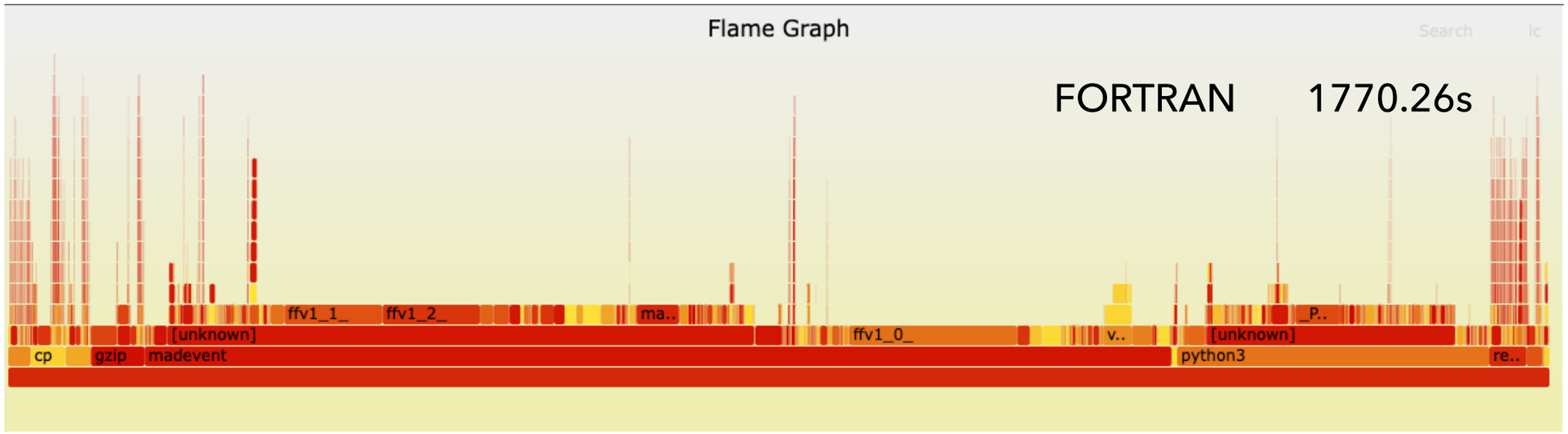
```
set sde_strategy 1
set vector_size 8192
set cudacpp_backend CUDA
```

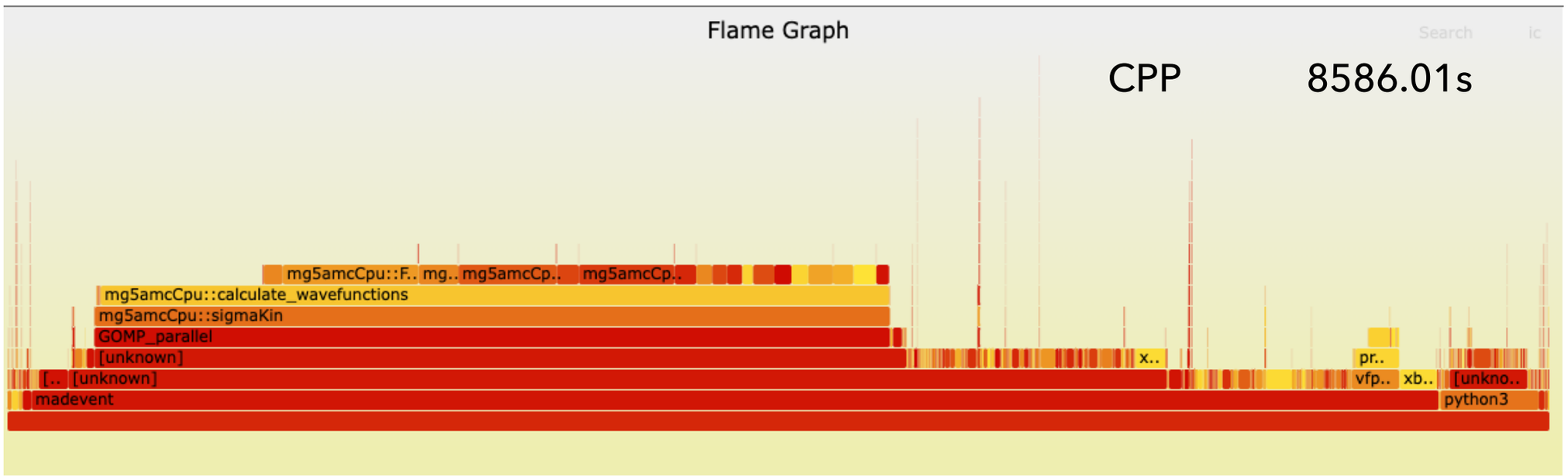
DY4j_LO_5f_CUDA_customizecards.dat

FLAMEGRAPHS

DY+3j (generating 20000 events)

svg files in [lxplus]



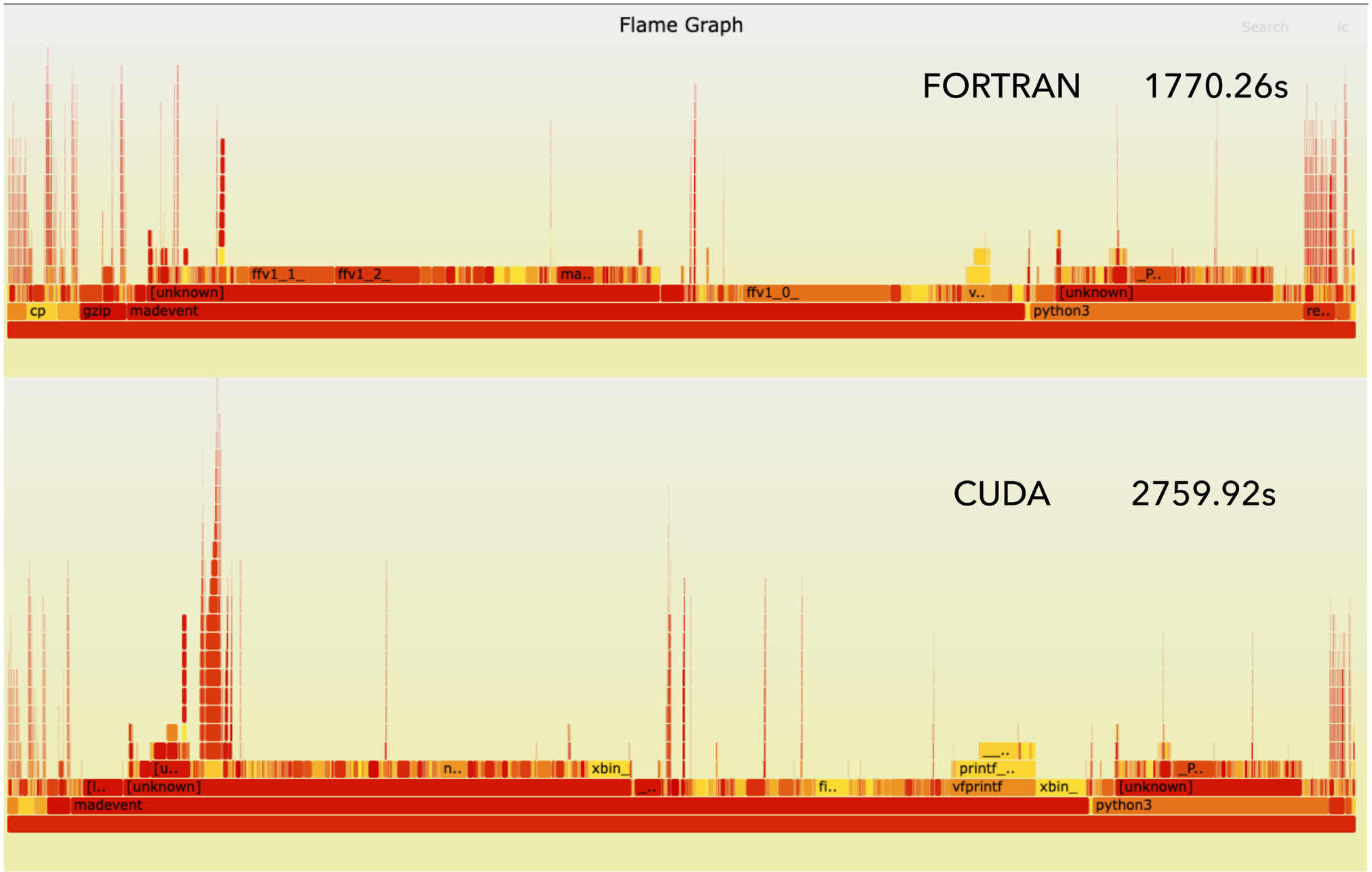


All jobs tested in lxplus8-gpu node

FLAMEGRAPHS

DY+3j (generating 20000 events)

svg files in [lxplus]



All jobs tested in lxplus8-gpu node