MG4GPU STATUS

FOR CMS-MG JOINT MEETING

24.09.10

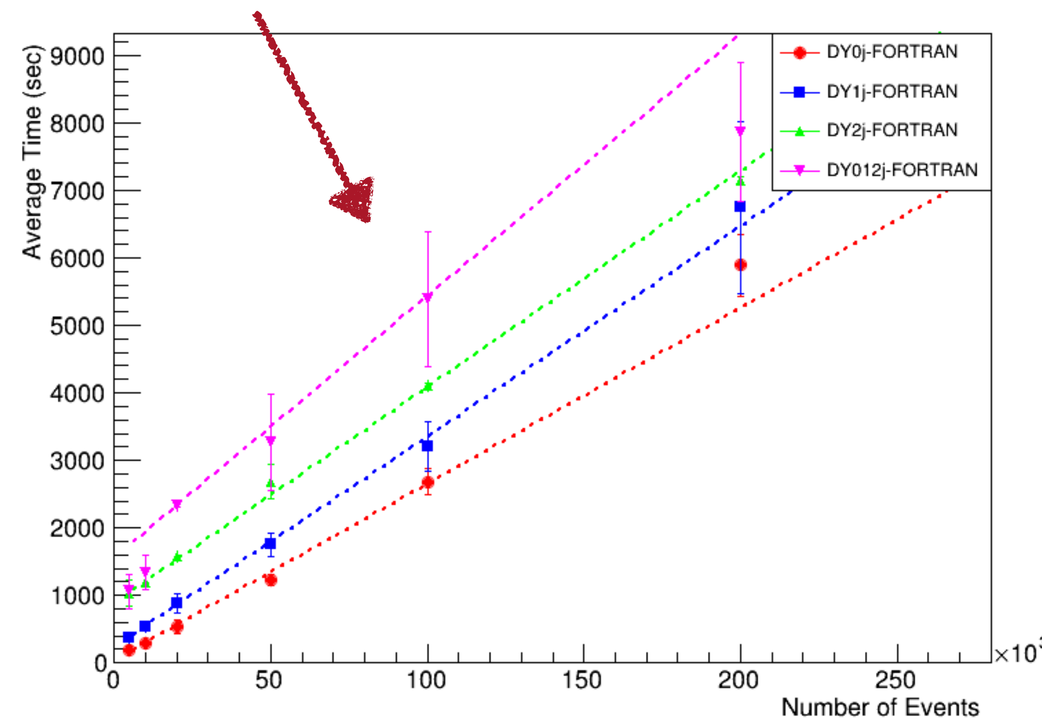
RECAP

Gridpack Production

- Baseline MG Repo - Synced on 240819
- Most slots in LXPLUS are in use😢 - moved to the server in SNU for FORTRAN/CPP test
  - ⇨ Gain has been calculated based on nb_core (how to make statement okay?)

Event generation

- Tested in LXPLUS - requesting single core exclusively
- Tested with no. of evts - 5K / 10K / 20K / 50K / 100K / 200K
- Partial results have been shown - Why DY+012j takes longer time than DY+0/1/2j?
  - ⇨ Tested w/ other backends / TT+0123j



GRIDPACK PRODUCTION

Drell-yan (high multiplicity)

	FORTRAN (nb_core = 32)	CPP (nb_core = 32)	CUDA (nb_core = 16)
DY+3j	22h 39m	9h 4m	4h 18m
DY+4j	-	-	3d 22h
DY+01234j	-	-	2d 10h (H100) nb_core = 28

- Clearly(!) see the improvements in DY+3j, x2.5 for CPP and x11 for CUDA
- Regarding DY+4j/01234j - Needs to process O(10k) grids…
- For CPP gridpacks, generating in SNU server with 80 cores, need additional 2 weeks
- For FORTRAN, tested in several servers…
  - SNU (80 cores): ~ 3 months
  - NERSC-CPU (256 cores): It is fast, but restricted by time limit (24 / 48 hours, based on QOS)
  - cms-connect (256 cores): Hardly matchable (1~2 days), easily disconnected

GRIDPACK PRODUCTION

TTbar - finalized!

	nb_core = 16 FORTRAN	nb_core = 16 CPP	nb_core = 16 CUDA	nb_core = 12 CUDA - H100
TT+0j	5m 47s	7m 15s	4m 41s	-
TT+1j	11m 8s	10m 43s	7m 7s	-
TT+2j	74m 52s	38m 25s	21m 47s	-
TT+3j	nb_core = 80 2d 4h	nb_core = 80 15h 51m	nb_core = 6 8h 11m	4h 53m
TT+0123j	nb_core = 80 2d 3h	nb_core = 80 1d 7h	8h 24m	4h 52s

- Improvements observed throughout the whole processes - x2 for CPP / x3.5 for CUDA for TT+2j
- Hugh improvement for TT+3j / 0123j! ~ x2 for CPP / x39 for CUDA (for Inclusive, **based on nb_core**)
- Only 6 madevents possible to be submitted for TT+3j/0123j - gg →ttxggg takes ~ 6GB GPU memory
- Additional test with 12 madevents using H100 (~ 96 GB)

EVENT GENERATION

Test Environment

- Using single core(requesting 1 CPUs) in lxplus condor
- Test timing for 5k, 10k, 20k, 50k, 100k, 200k event generation
- Each process x nevt configuration tested 8 times - take avg & stddev

TT2j - 5000

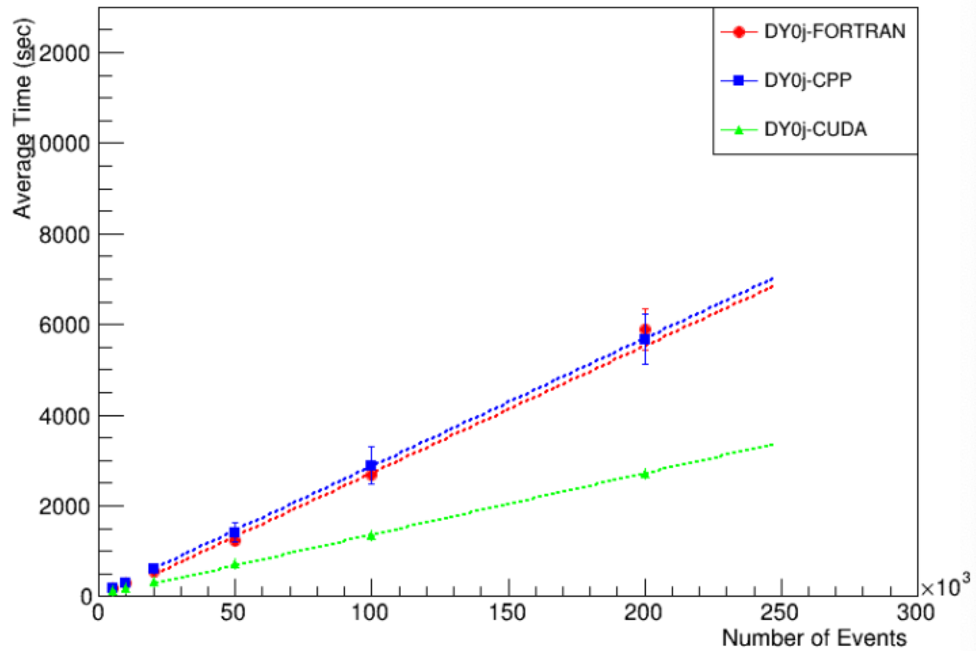
	FORTRAN		CPP		CUDA	
0	28m9.804s	1689.804	20m38.510s	1238.51	6m47.389s	407.389
1	29m8.745s	1748.745	20m2.217s	1202.217	7m45.196s	465.196
2	28m45.357s	1725.357	19m54.762s	1194.762	6m53.662s	413.662
3	27m32.752s	1652.752	19m52.637s	1192.637	7m50.752s	470.752
4	28m22.442s	1702.442	21m0.456s	1260.456	6m18.169s	378.169
5	27m29.009s	1649.009	19m47.381s	1187.381	6m43.447s	403.447
6	27m21.541s	1641.541	20m11.514s	1211.514	6m0.925s	360.925
7	28m47.916s	1727.916	20m31.408s	1231.408	6m19.430s	379.43
avg		1692.196		1214.861		409.871
err		40.832		26.011		19.770

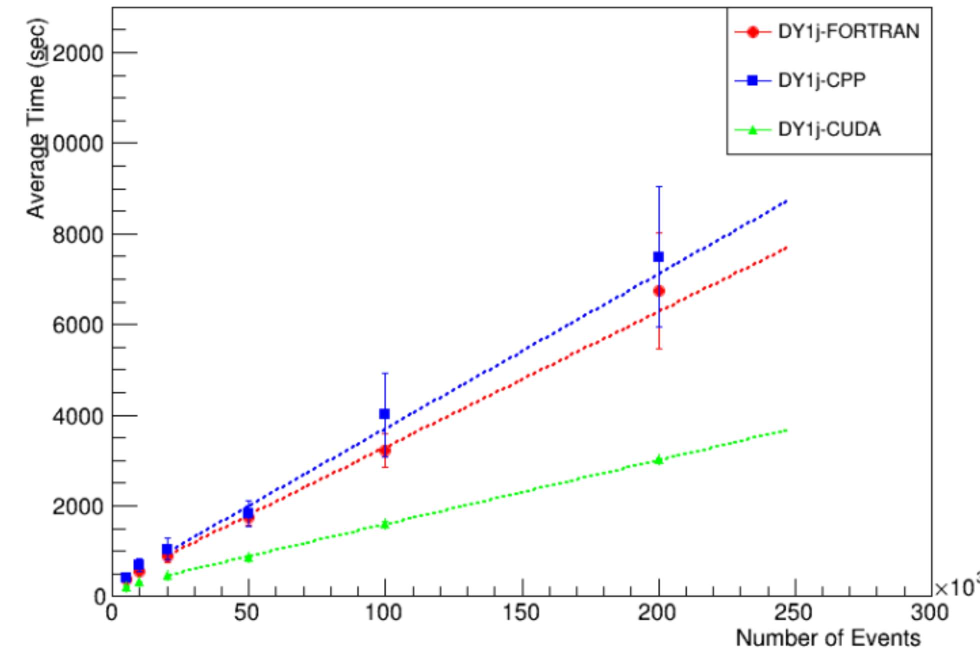
These numbers used for results

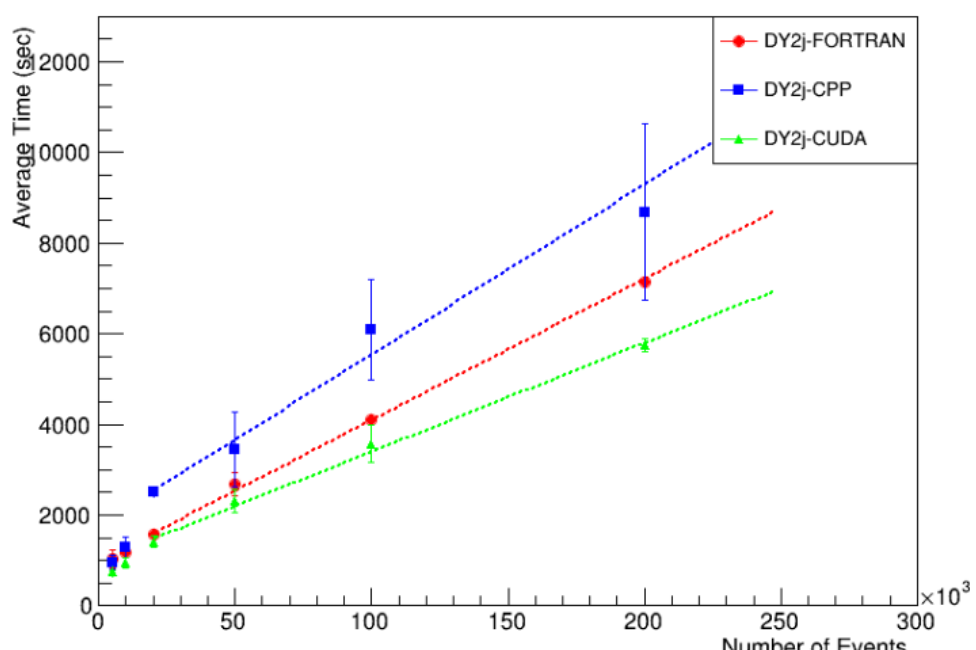
- Done for TT, Done for low multiplicity DYs, DY+3j

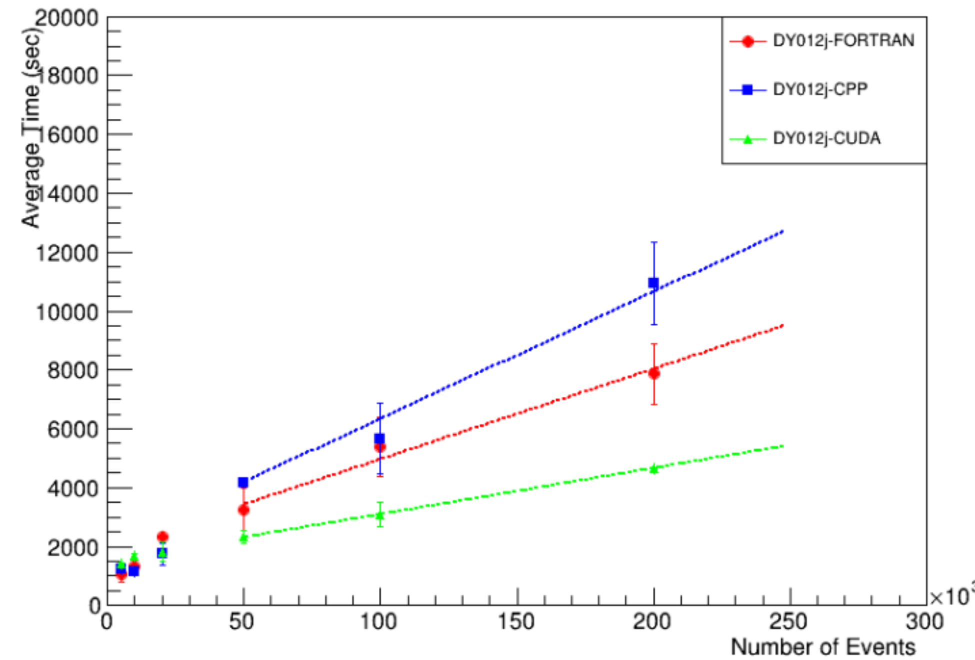
EVENT GENERATION

Drell-Yan (Backend Comparison)





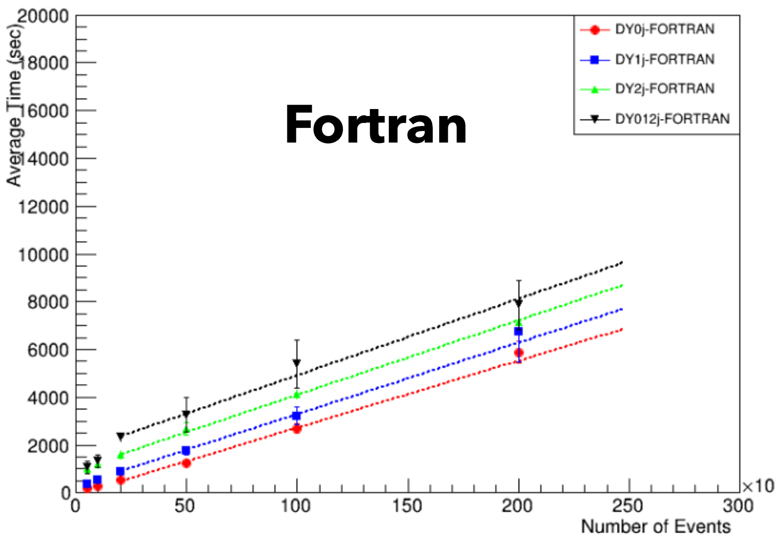


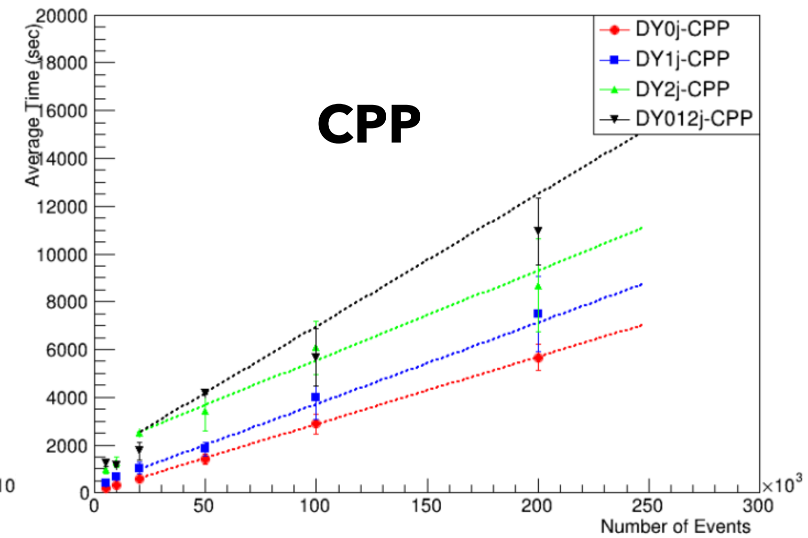


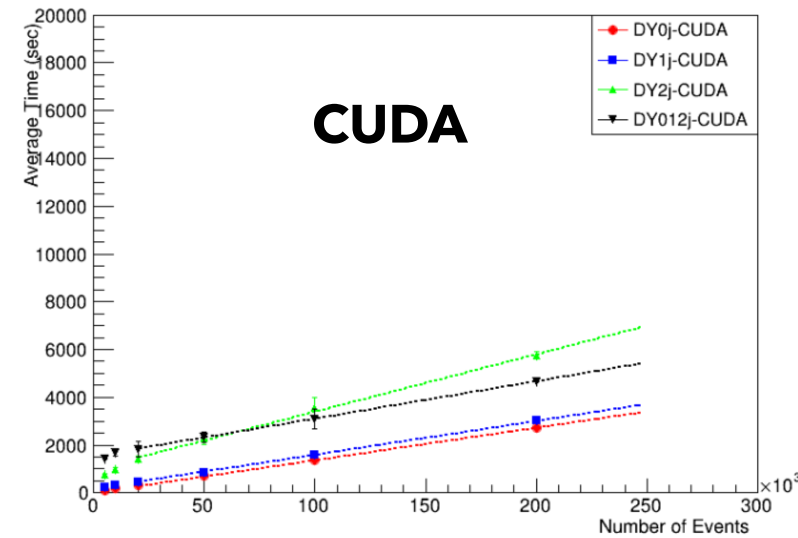
✓ Clearly see x2-3 improvement in CUDA!
No improvement viable for AVX2 Vectorization - even worse?

EVENT GENERATION

Drell-Yan (Inclusive vs. Exclusive)

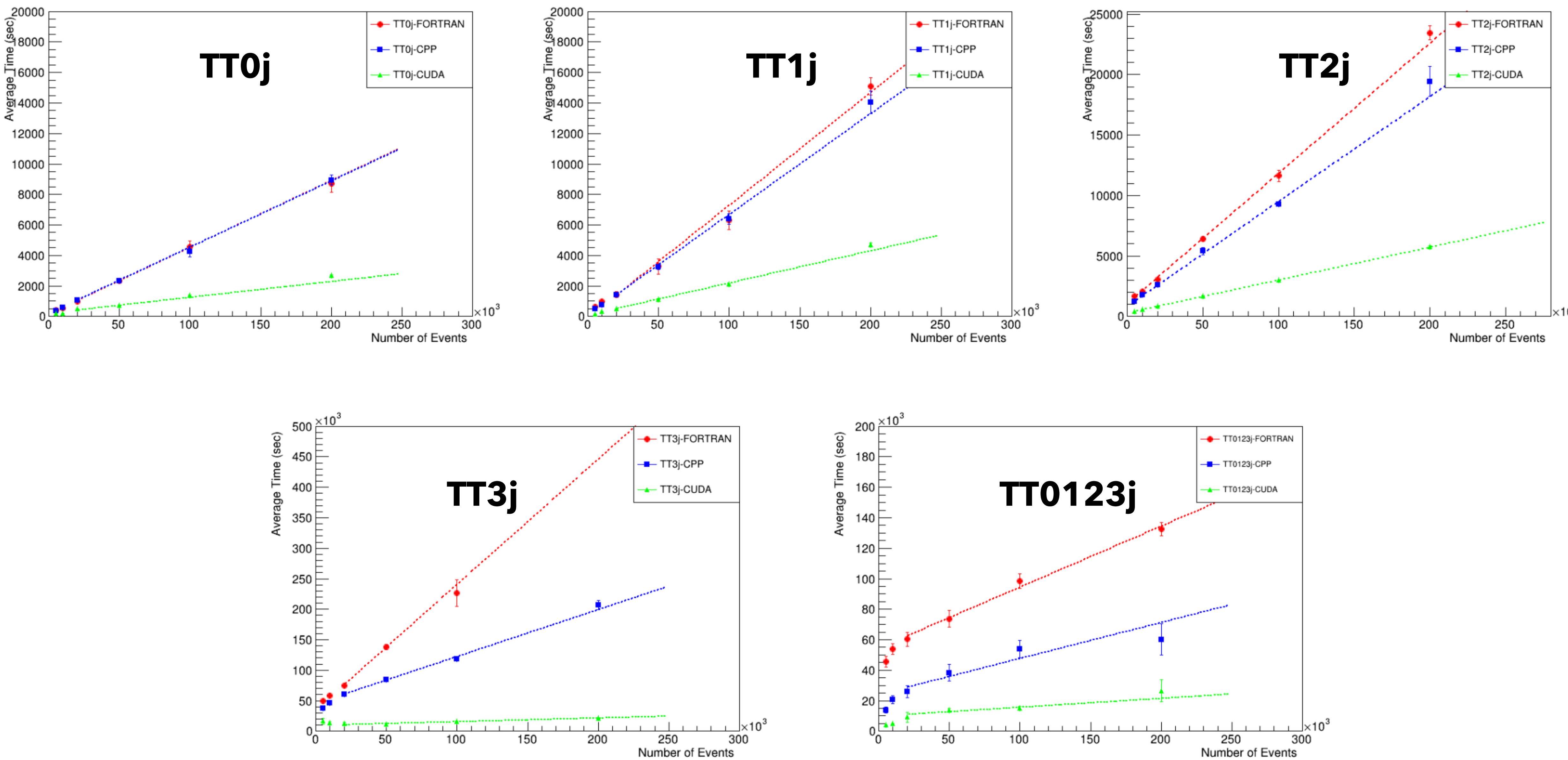






- Inclusive sample generation takes more time… or at least comparable with maximum jets

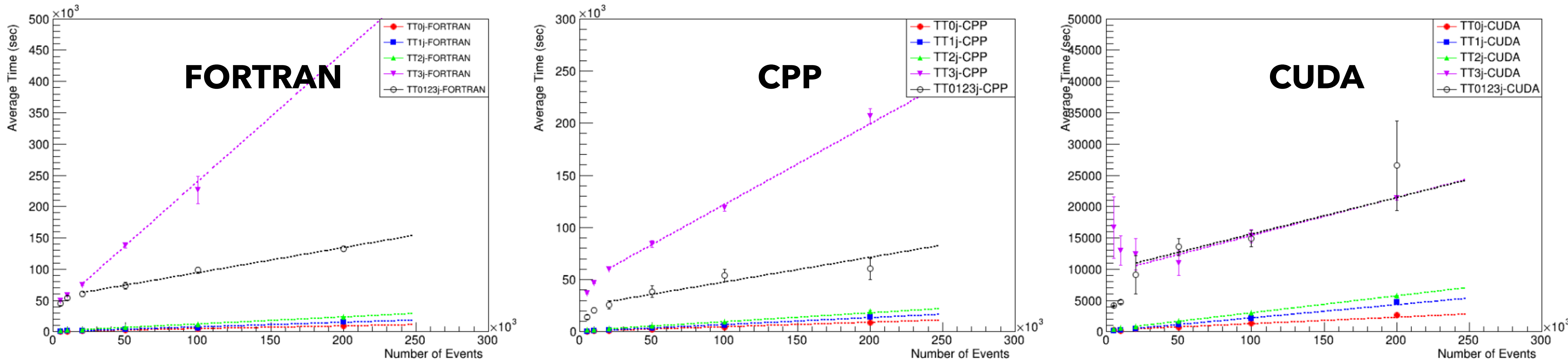
TTbar (Backend Comparison)



✓ ~x2.5(5) improvements from CPP(CUDA) event generation

EVENT GENERATION

TTbar (Inclusive vs. Exclusive)



- For FORTRAN/CPP, inclusive resides b/w 012j & 3j - as expected
- For CUDA, inclusive comparable with 3j
  - Naive guess: Does no. of processes (or grids?) matter?
    More time to spend roaming around subdirectories than Vagas optimization?
- Hard to check what's going on behind the scenes - no printing messages

ROADMAP

- **For CMS upgrade week (Sep. 17)**
- For the status talk, I want to make numbers explainable
  - For Gridpack Generation, looks okay
  - For Event Generation, 1. Why T(inc.) > T(exc.)? 2. Why CPP takes more than FORTRAN for DYs?
- **For Pre-approval / CHEP (Early Oct. ~)**
- We have additional 3 week before pre-approval
- How we will treat DY+4j/01234j?
  - For CPP, might possible to produce gridpacks, but very tight time limits
  - For FORTRAN, even 256 parallelization needs about a month…
- Several options checked:
  - In NERSC Slurm: C/R option with podman-hpc… need super privileged
  - Splitting CODGEN and INTEGRATE steps and split the batch jobs: no more support or at least need for development
- Most of the progress made from ttbar - Is it feasible to drop DY+4j? (and make inclusive study up to 3 jets)?
- For the difference speed for process definition (e.g. uux_epemgg v.s. pp_epemjj), how it should be treated?
- Multi-backend options? (If we have enough time?)

BACK UP

STATUS - GRIDPACK PRODUCTION

Drell-yan (low multiplicity)

	FORTRAN (nb_core = 16)	CPP (nb_core = 16)	CUDA (nb_core = 16)
DY+0j	7m 15s	6m 29s	5m 21s
DY+1j	10m 13s	9m 59s	11m 39s
DY+2j	72m 10s	69m 49s	51m 14s
DY+012j	75m 48s	84m 42s	59m 36s

- First inclusive results - DY+2j dominates in gridpack production
- Compatible timing for FORTRAN ~ CPP (AVX2) - might expect more improvements in AVX512?

GRIDPACK PRODUCTION

Summary

- Improvements from the latest numbers - now also viable in CPP
- Some processes (e.g. gg_ttxggg) takes huge amount of GPU memory
  - ⇨ Parallelization level (nb_core) resticted by (LargestMadeventMemory / TotalMemory)
  - ⇨ Room for improvement in inclusive processes - low multiplicity processes consume low memory, even though high multiplicity processes are time - consuming
  - ⇨ We don't have SUPER MEMORY SINGLE GPU possible to submit O(100) jobs...
    Viable for Super fast Gridpack Production if Multi-GPU setup supported!
- Experience with single H100 - x2 memory, ~x2 madevent jobs, ~/2 timing
- DY4j / TT3j too slow in FORTRAN/CPP set-up, many HPCs are in use:< - hard to produce numbers

EVENT GENERATION

Summary

- Clear improvement (x2-4, depending on the process) shown in CUDA!
  Not large (or no) improvement for AVX2 supports....
- AVX512 Supports? Again, lack of machines.
- DY+012j inclusive sample takes more time than DY+0j/1j/2j
  - Comparison flamegraphs for DY+2j / DY+012j would help
- Further test with higher multiplicities are on-going