Improved Event Generation at NLO and NNLO

or

Extending MCFM to include NNLO processes

W. Giele, RadCor 2015

NNLO in MCFM:

- Jettiness approach:
  Using already well tested NLO MCFM as the double real and virtual-real generator of NNLO
  W+jet: Boughezal, Focke,Liu,Petriello
  H+jet: Boughezal, Focke, Giele, Liu, Petriello.

- Better phase space:
  Generating double real and virtual-real events from fixed born phase space.
  Extending the Matrix Element Method to Next-to-Leading Order (Campbell, Giele, Williams)

- Fast performance on modern processors:
  Using multi-threading/multi-core (this talk)
  A Multi-Threaded Version of MCFM (Campbell, Ellis, Giele)

Single core vs Multiple cores

- The power used by a processor is given by $P=C\ V^2\ F$
  - C is the capacitance being switched/cycle
  - V is the voltage (determines rate at which capacitance charges/discharges)
  - F is clock frequency of processor
- By putting in 2 cores and halving the frequency we keep the same performance
  - $C \rightarrow 2C$, $F \rightarrow F/2$, $V \rightarrow V/2$
- So that $P \rightarrow P/4$. So by reducing the frequency one can increase the performance of the chip by putting in many cores

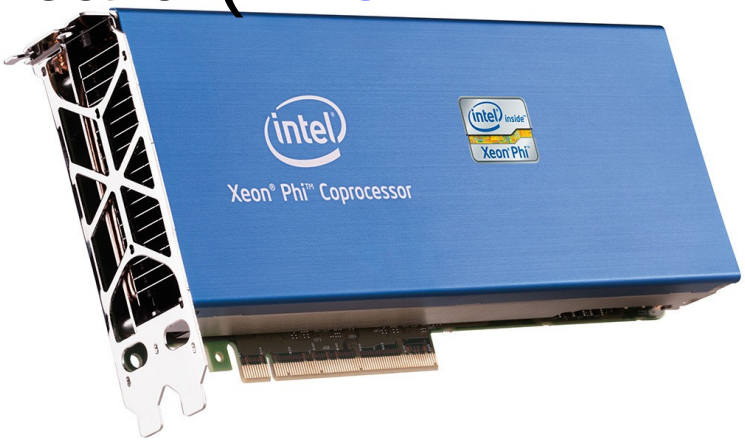
GPU's vs Multi-Core CPU's

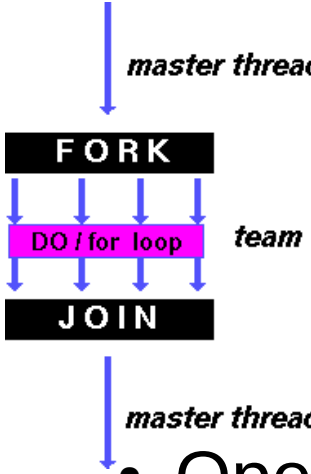
- GPU's have many cores (up to 2880) giving a potential huge acceleration.
- For example at LO a speed-up of ~500 can be obtained on a desktop

  Thread-Scalable Evaluation of Multi-Jet Observables; (Giele, Stavenga, Winter)
- Need to carefully reprogram existing programs using CUDA and careful memory layout
- Special hardware required
- Not really usable for MCFM

GPU's vs Multi-Core CPU's

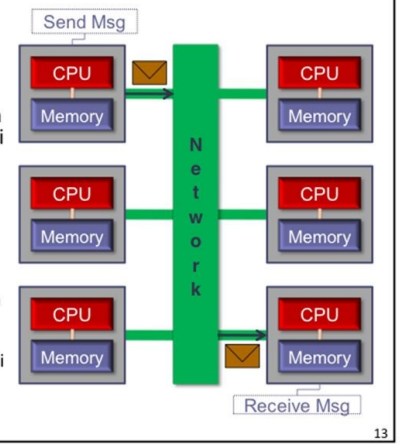
- Programs can easily be made to use multiple cores in parallel using openMP and MPI with minimal changes to the code
- Program will accelerate on laptops, desktops, clusters depending on number of cores available
- Processors are developing quickly, optimized to run openMP and other parallel methods
- E.g. The Xeon Phi co-processor card (240 threads, 1Ghz, 28MB cache)
- Xeon Phi E5 2699 processor (18 cores, 2.3-3.6Ghz, 45Mb cache)





openMP vs MPI

- OpenMP has a strict standard and is included in the gnu and intel compilers (just compile with the openmp flag)
- A program is can be made to run in parallel by adding simple pragma's, but can be challanging to debug
- OpenMP requires a unified memory address space for all threads (i.e. all threads access the same memory)
  - Advantage: cache and memory is shared between all threads
  - Disadvantage: Does not run on clusters
- Perfect for running on single multi-core processor (laptop, desktop, single node on cluster). It is “hidden” from the user as it automatically uses all available threads when program compiled with openMP flag.

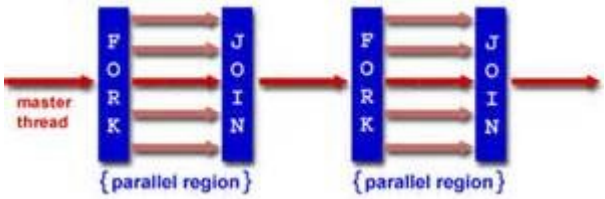


openMP vs MPI

- To extend the parallelism to multiple nodes one can use MPI
- It runs independent programs each with its own memory space
- Data can be transferred between the MPI processes by passing “messages”
- MPI has several flavors and is less standardized. It requires to be installed and libraries and compiler extensions need to be made.
- We developed a hybrid MCFM/MPI MCFM version. So on each processor we can use its threads using openMP, while we use MPI to link the different openMP processes

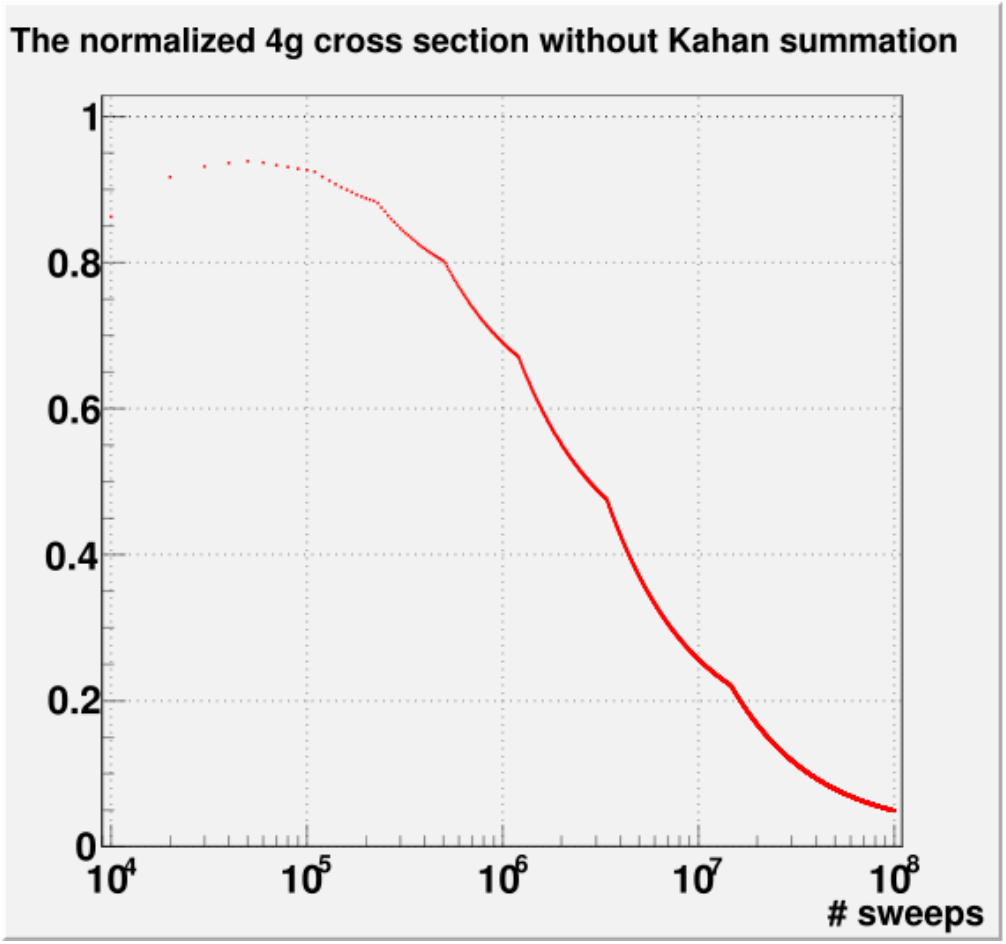
Parallelism in event MC's

- The easiest way to use multiple threads is making Vegas parallel.
- The events to be evaluated are spread over all threads, while there is only a single Vegas grid
  - e.g. using 100 processors linked with MPI, each using openMP with 5 threads gives us 500 threads. If Vegas uses 10x1,000,000 calls,each thread will evaluate 10x2,000 events. Between the 10 sweeps the Vegas grid is updated using all 500x2,000 events
- We ensure we get the same result using one thread or 500 threads. (This is indispensable for validation and debugging)



Parallelism in event MC's

- To get the same (and correct) result, independent of over how many threads the run is using one has to use Kahan summation to ensure no rounding errors occurs.
- Rounding errors in summation are an issue when using large number of events



Parallelism in event MC's

- Vegas code:

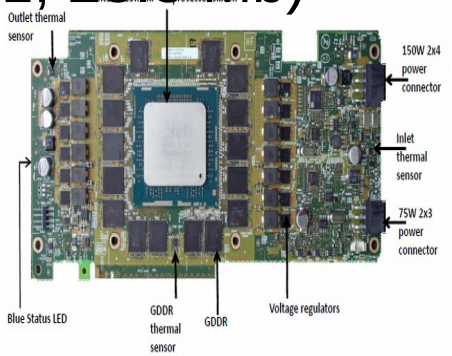
```
          call mpi_bcast(xi,ngrid*MXDIM,mpi_double_precision,
     .                   0,mpi_comm_world,ierr)
!$omp  parallel do
!$omp& schedule(dynamic)
!$omp& default(private)
!$omp& shared(incall,xi,ncall,ndim,sfun,sfun2,d,cfun,cfun2,cd)
!$omp& shared(rank,size)
          do calls = 1, ncall/size
```

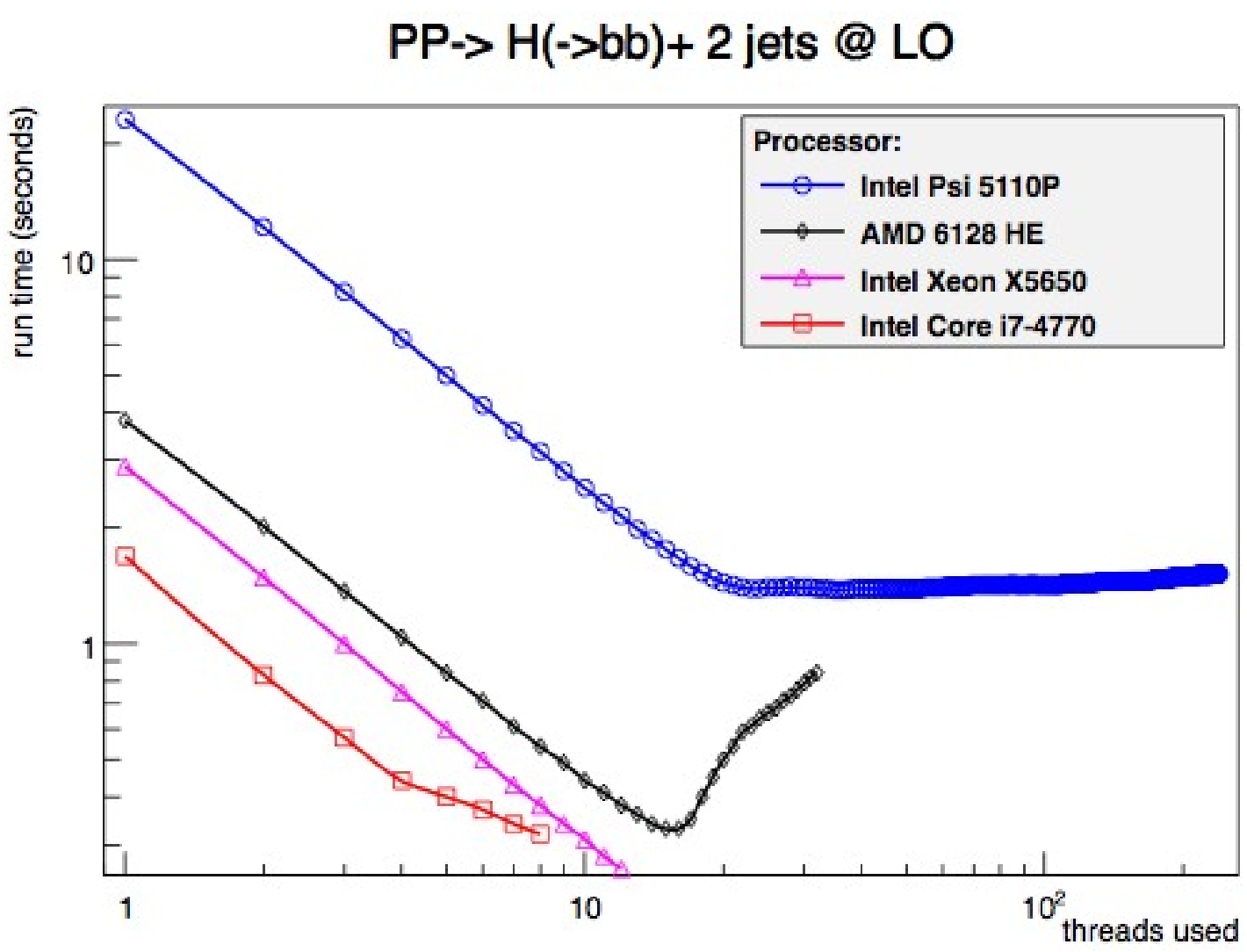
- This is all as far as MPI is concerned (except of combining histograms when run is terminating)
- For openMP one has to specify whether variables are local to the thread of global
- Note: MPI has seperate vegas file/histograms per task,while openMP has a single veas file and histograms.

Performance of openMP MCFM

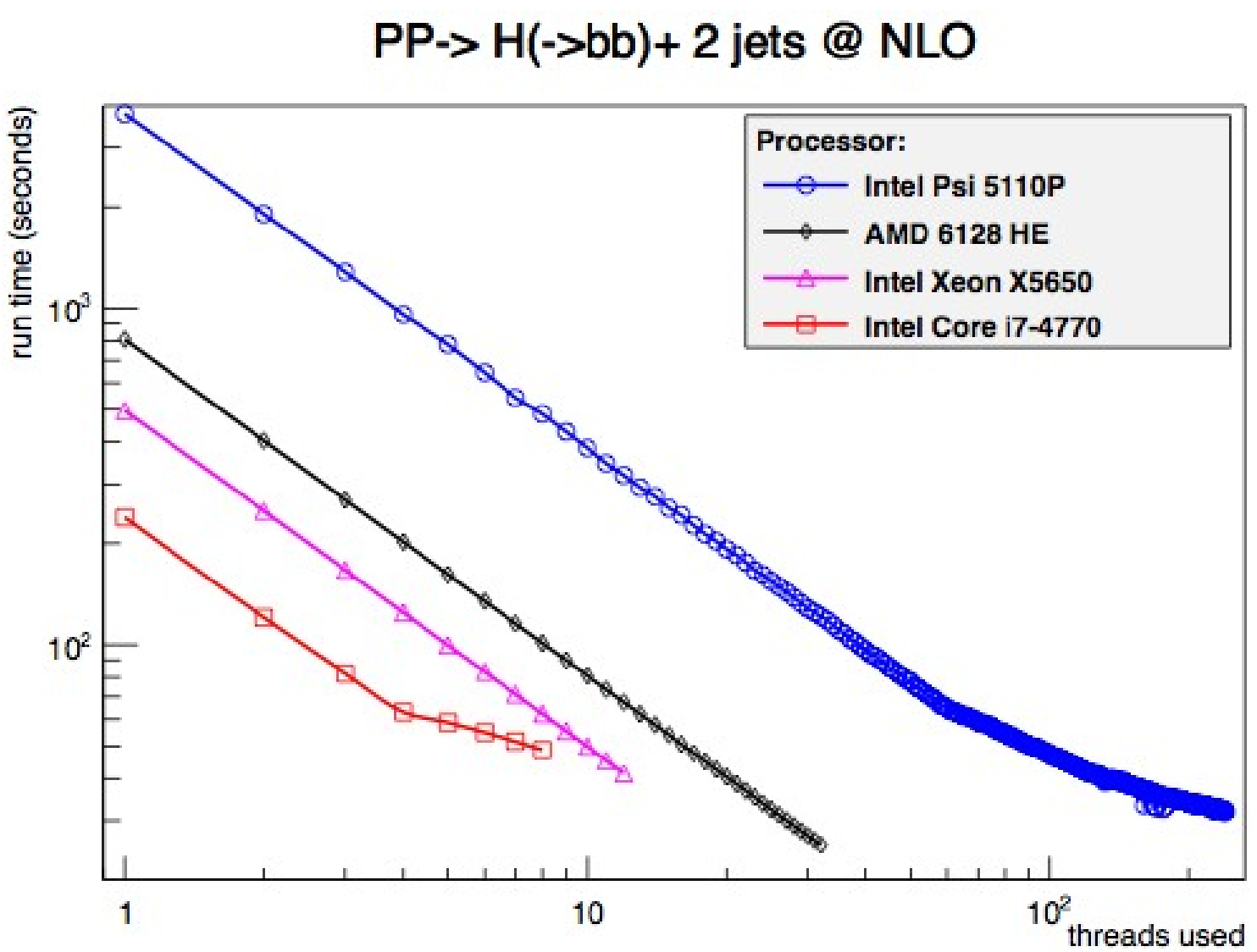
(Campbell, Ellis, Giele)

- We use H+2 jets at NLO as an example
- This process is 'used' in the H+1 jet at NNLO in the jettiness approach so speedy evaluation is crucial
- We use 4 different configurations to test openMP MCFM:
  - Standard desktop using an Intel core I7-4770 (4 cores, 3.4Ghz, 8MB cache)
  - Double Intel x5650 processor (2x(6 cores, 2.66Ghz,12Mb cache))
  - Quadruple AMD 6128 HE opteron (4x(8 cores, 2Ghz, 12Mb cache)
  - Xeon Phi co-processor (60 cores/240 threads, 1.1Ghz, 28.5 Mb)

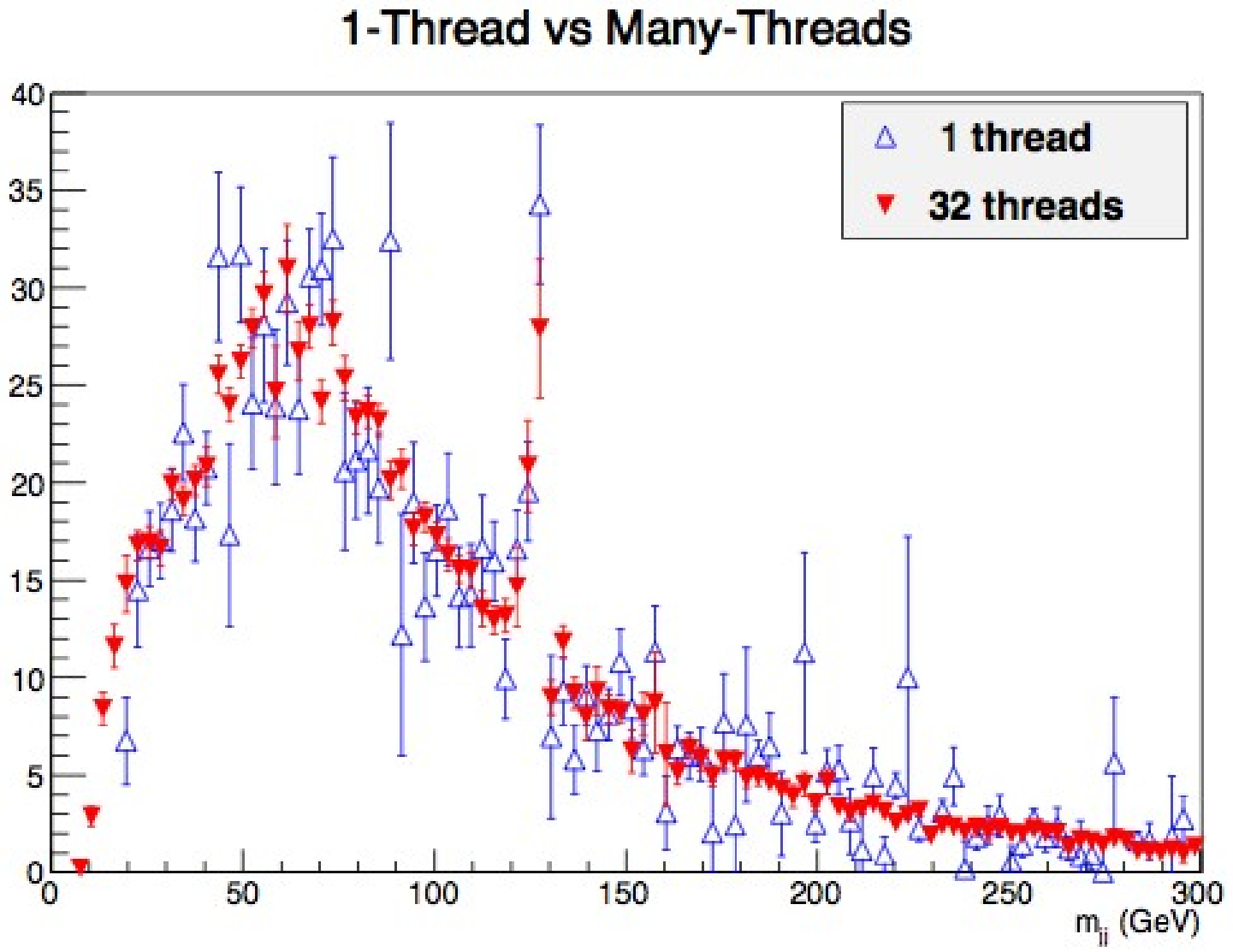




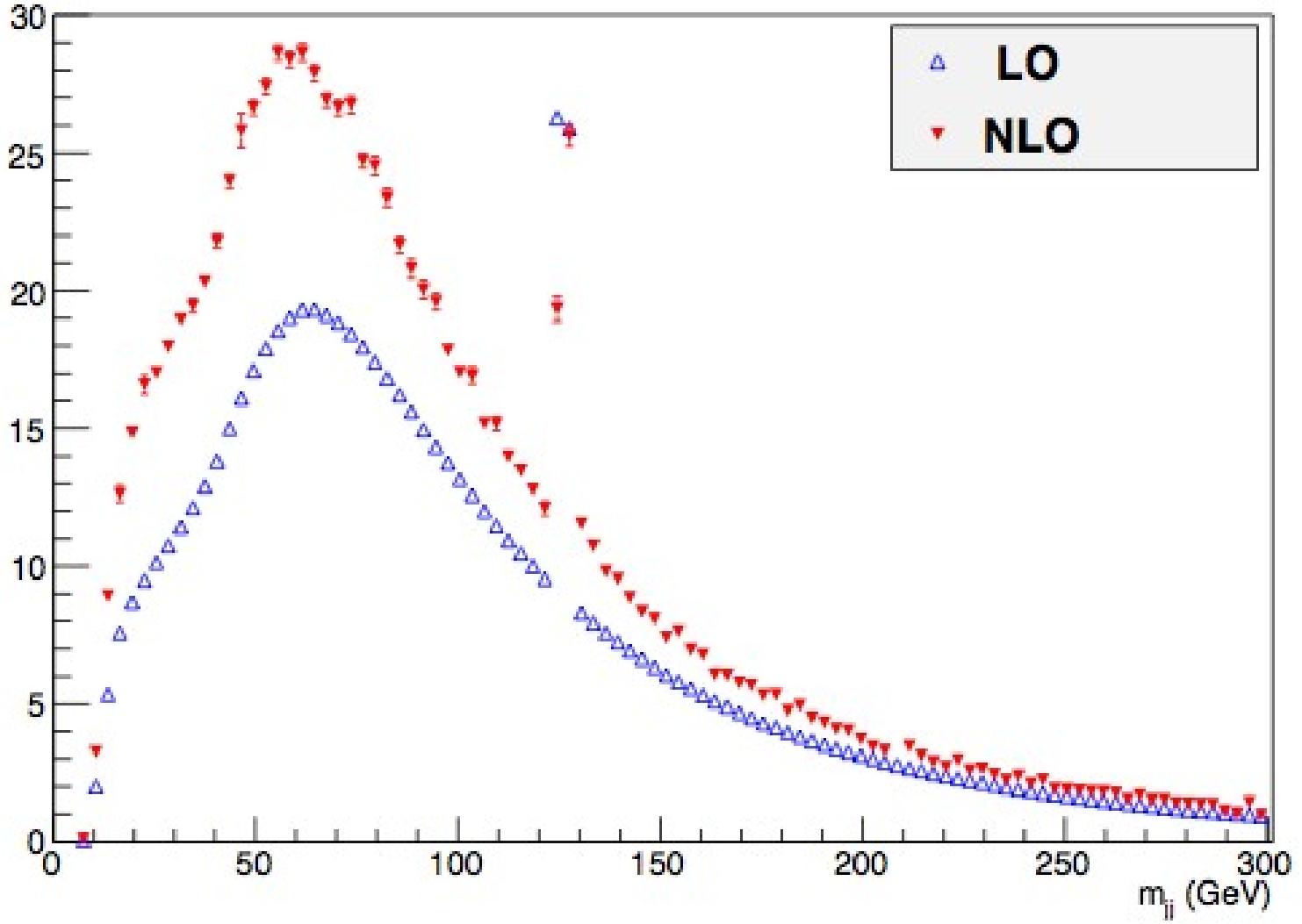
- 4x1,000+10x10,000 events, 10 dimensional integration.
- I7: The processor has 4 cores each with 2 hyper threads
- The AMD and Xeon Psi become bandwidth bound (run time becomes dominated by memory fetch time)



- 4x1,000+10x10,000 events, 13 dimensional integration.
- NLO on the other hand is computational bound and scales as one would expect
- The xeon-phi exhibits a bit of slowdown above 60 (has 60 cores)



- The dijet invariant mass distribution (5 GeV bins) at NLO
- 1 hour of runtime using 1 thread on the Intel I7 vs 32 threads on the quad AMD Opteron

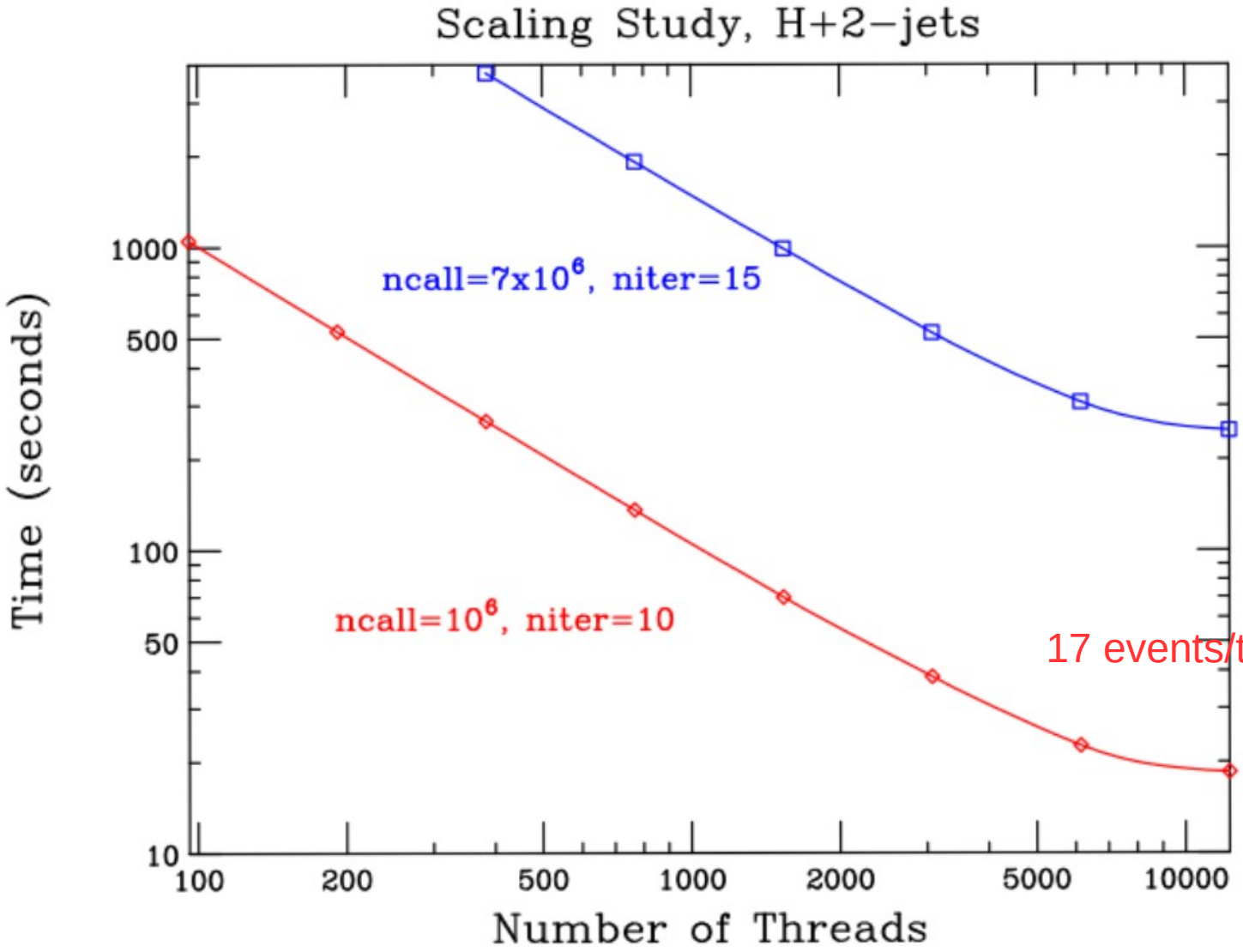


- The dijet invariant mass distribution (5 GeV bins) at NLO
- 4x1,500,000+10x15,000,000 events
- LO: 12 min on 12 threaded dual Intel Xeon X5650
- NLO: 22 hours on the 32 threaded quad AMD Opteron

Performance of hybrid MCFM

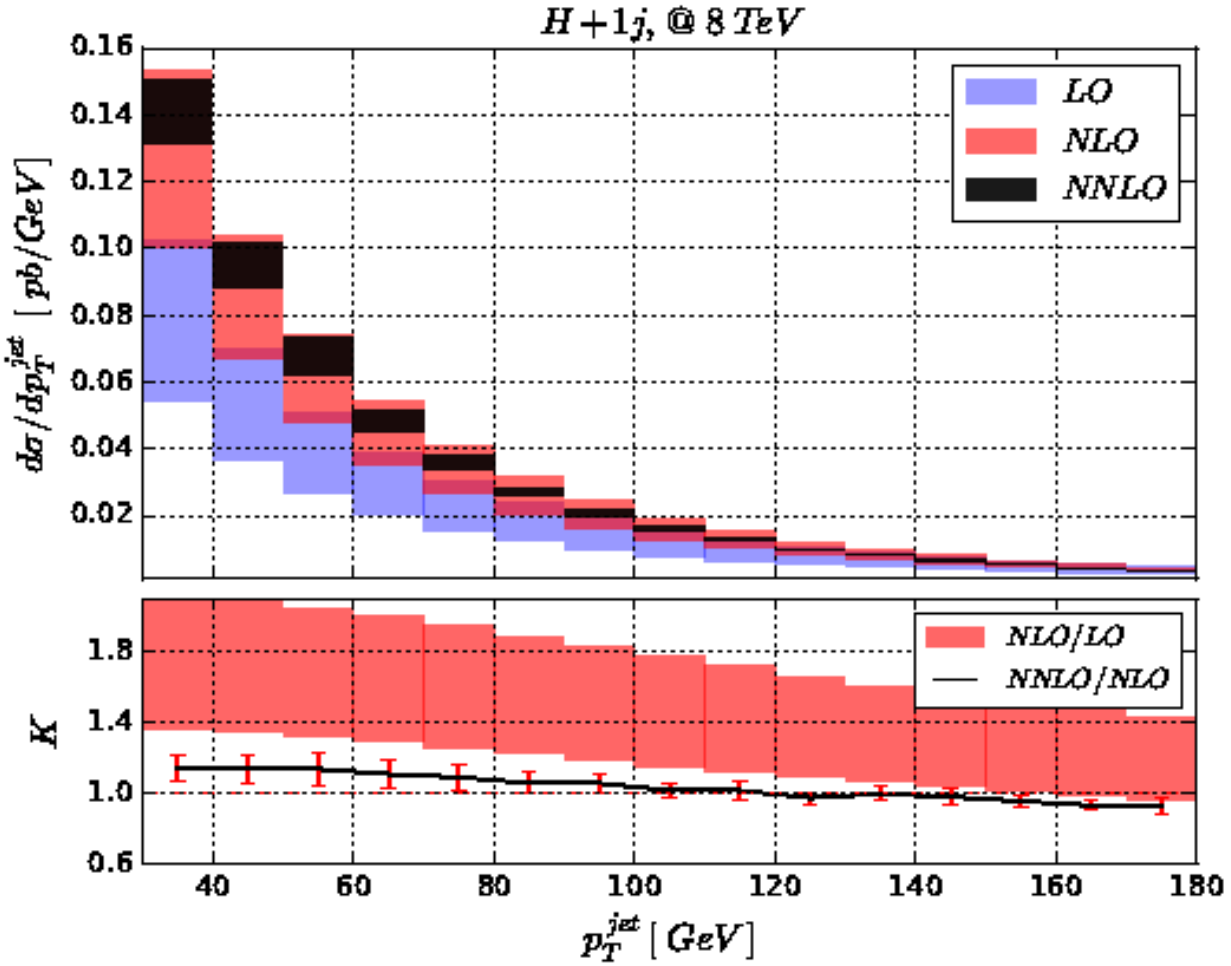
Boughezal, Focke, Giele, Liu, Petriello

- The hybrid mode is for running on clusters
- While NLO can easily be run on desktops, for NNLO clusters are still needed for reasonable run times
- Care needs to be taken with integer counters
- How to best use Vegas still under study (one does not want to freeze up the adaptive grid)



- quad 6-core intel chips per node on NERSC
  → 6 openMP threads/MPI task
- Scales as expected up to ~5,000 threads

H+jet: Boughezal, Focke, Giele, Liu, Petriello.



- We used 15x10^7 events on 1,000 thread
- This results in ~30 min of runtime for the results in the published paper.
- With better jettiness phase space generators this time will improve further.

Conclusions

- We have successfully made a hybrid openMP/MPI version of MCFM
- It will give accelerations from laptops to large clusters using a single Vegas grid
- OpenMP allows one to keep cache unified on the chip.
- MPI links up all the processors, still using a single vegas grid
- MCFM can fully take advantage of the new chips with more and more cores and the clusters based on these chips (multi-petaflop clusters).