

Tree amplitudes on modern GPUs

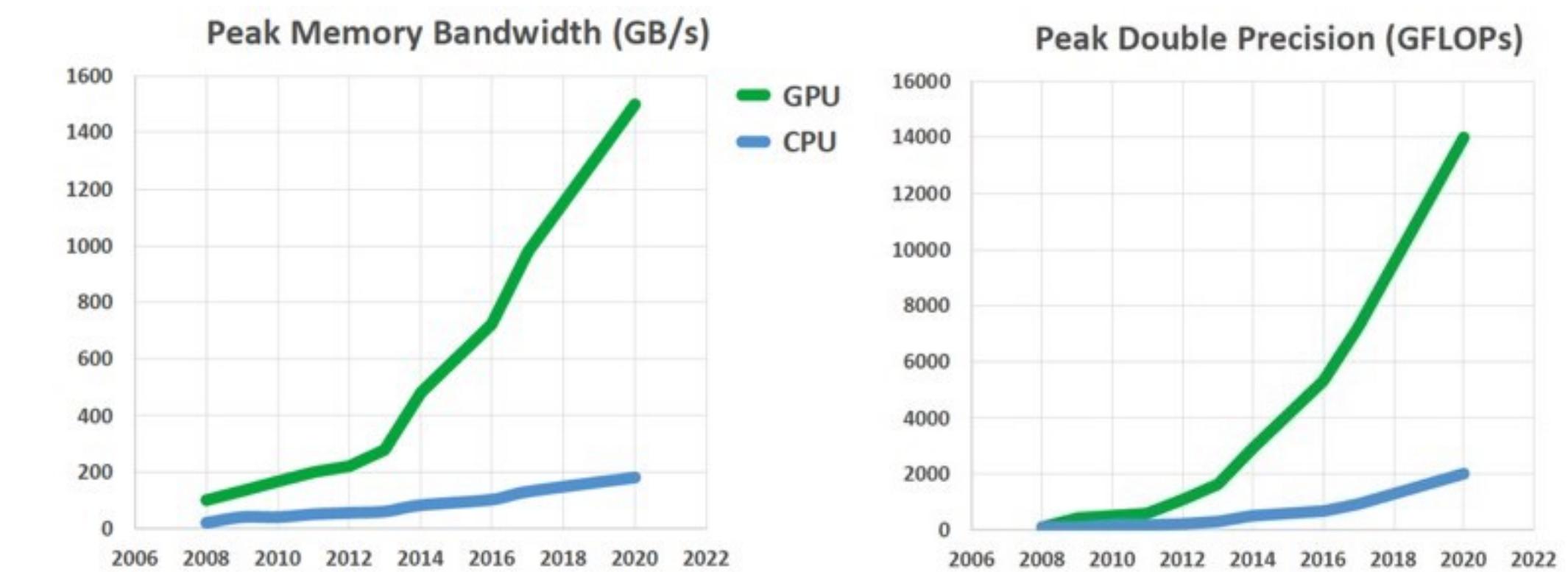
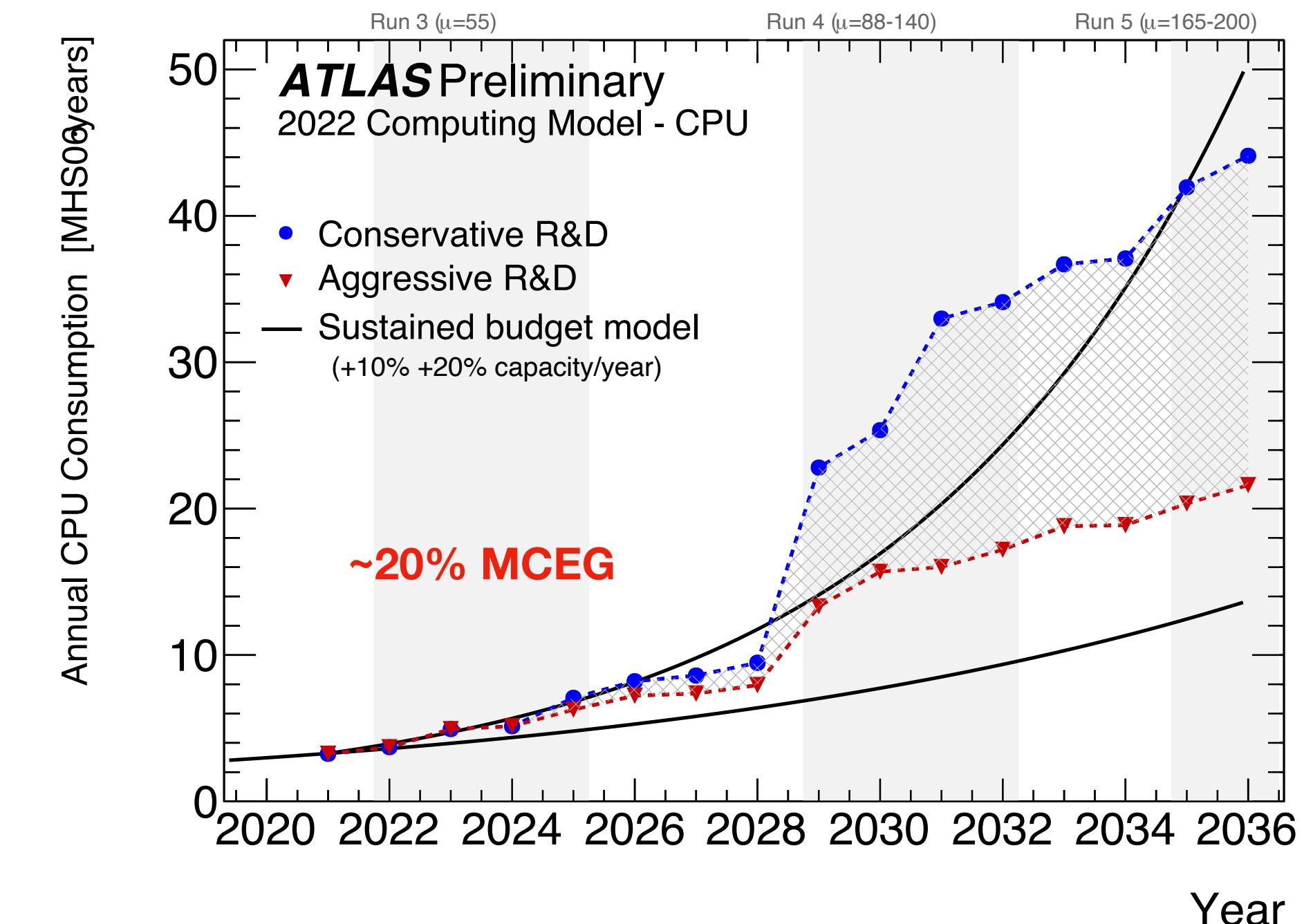
A case study for novel event generators

[2106.06507]

Enrico Bothmann*, W. Giele, S. Höche, J. Isaacson, M. Knobbe

Introduction/Motivation

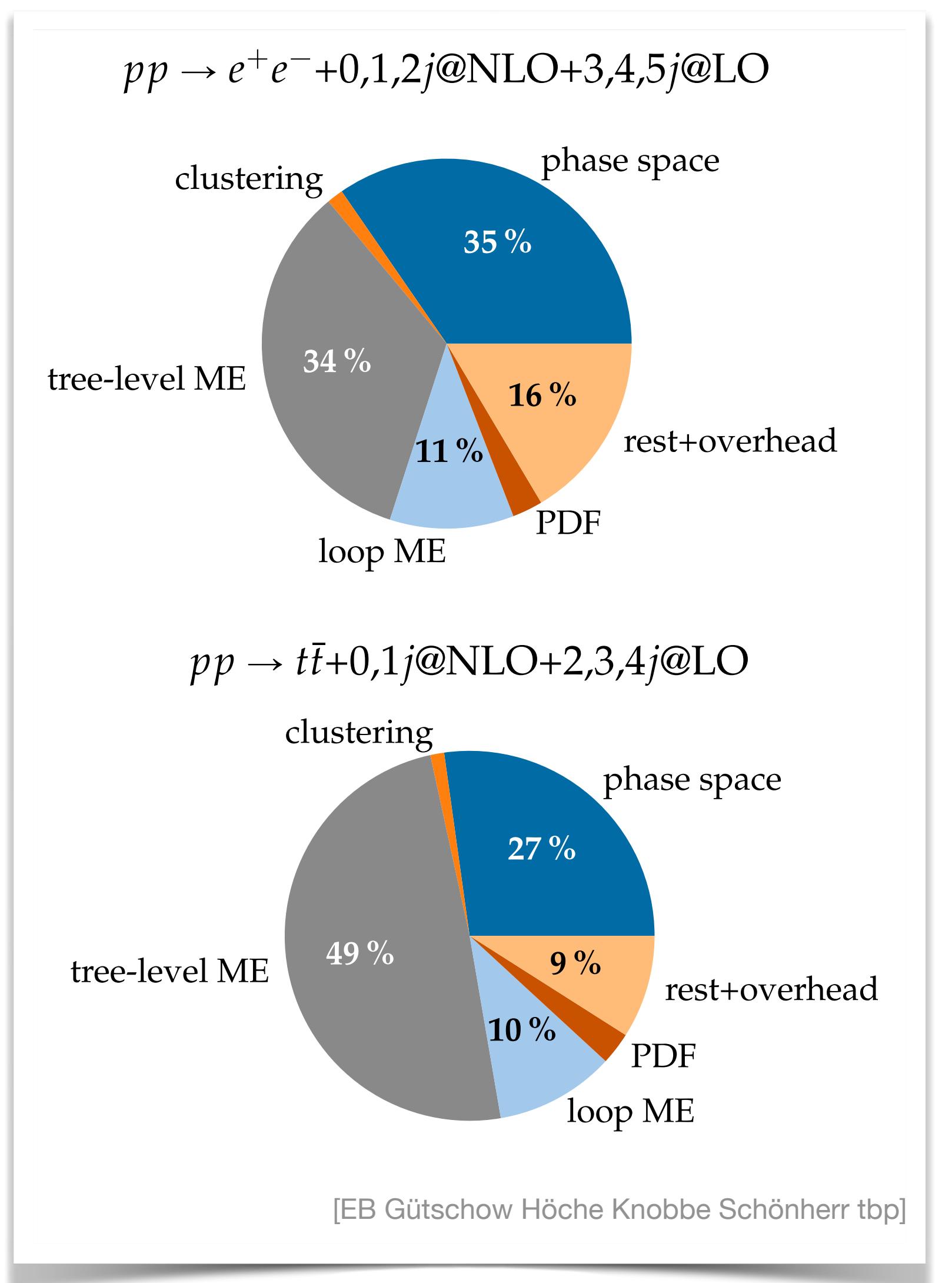
- MC embarrassingly parallel → perfect for GPU (SIMT)
- Can build on early 2010s CUDA studies
[Hagiwara, Kanzaki et al. 0908.4403, 0909.5257, 1010.2107, 1305.0708], [Giele Stavenga Winter 1002.3446]
- 10x bandwidth, 30x DP GFLOPS within 10 years
- profit from HPC trends (FOMO)
GPU-accelerated clusters, new parallelism abstractions:
KOKKOS etc.
- new urgency: HL-LHC era requirements
[ATLAS HL-LHC Computing CDR, Valassi et al Challenges in MCEG software for HL-LHC arXiv:2004.13687]
- elusive signals: push towards complex and/or high multiplicity procs



Introduction/Motivation

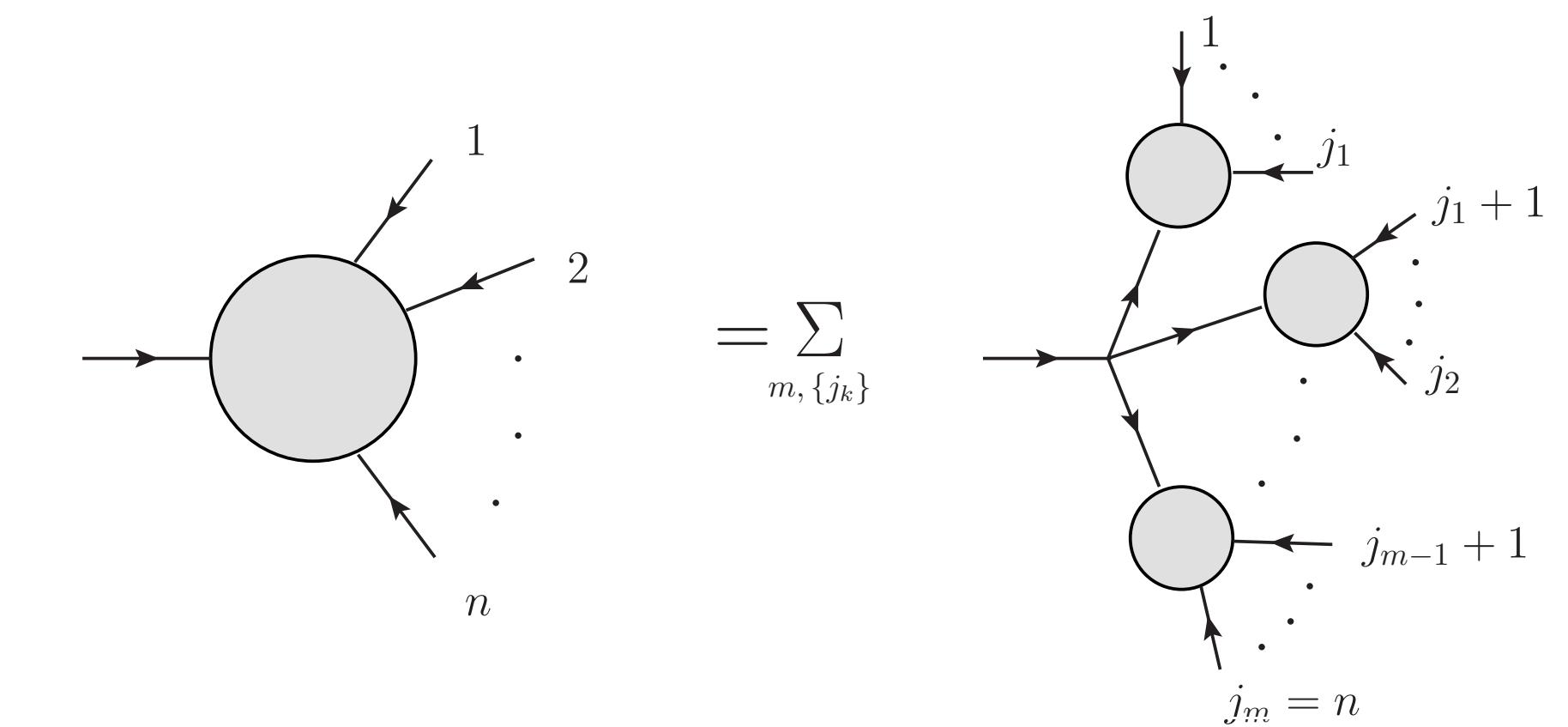
Attack bottleneck of MC event generation

- Focus on standard candle processes ($V+jets$, $t\bar{t}+jets$) which need tons of statistics
 - finding: limited by tree-level ME and PS sampling
[production setups after optimisation \[EB Gütschos Höche Knobbe Schönherr tbp\]](#)
- Pathfinder study: new CUDA-accelerated BlockGen code [\[2106.06507\]](#)
 - Gluon-only amplitudes as a testbed
following up on W. Giele's 2010 proof-of-principle implementation Tess [\[Giele Stavenga Winter 1002.3446\]](#), but no approximations or gluon-specific symmetries
 - Investigate different recursive strategies on CPU and GPU
[trying new combinations of colour and helicity treatments](#)
 - Compare with each other and to existing MCEG codes
- Related efforts by other groups
 - MadGraph renewed effort using KOKKOS et al [\[Valassi et al 2106.12631\]](#) → Talk by Andrea Valassi (yesterday)
 - MadFlow using TensorFlow [\[Carrazza et al 2106.102796\]](#) → Talk by Juan Martinez (current session)

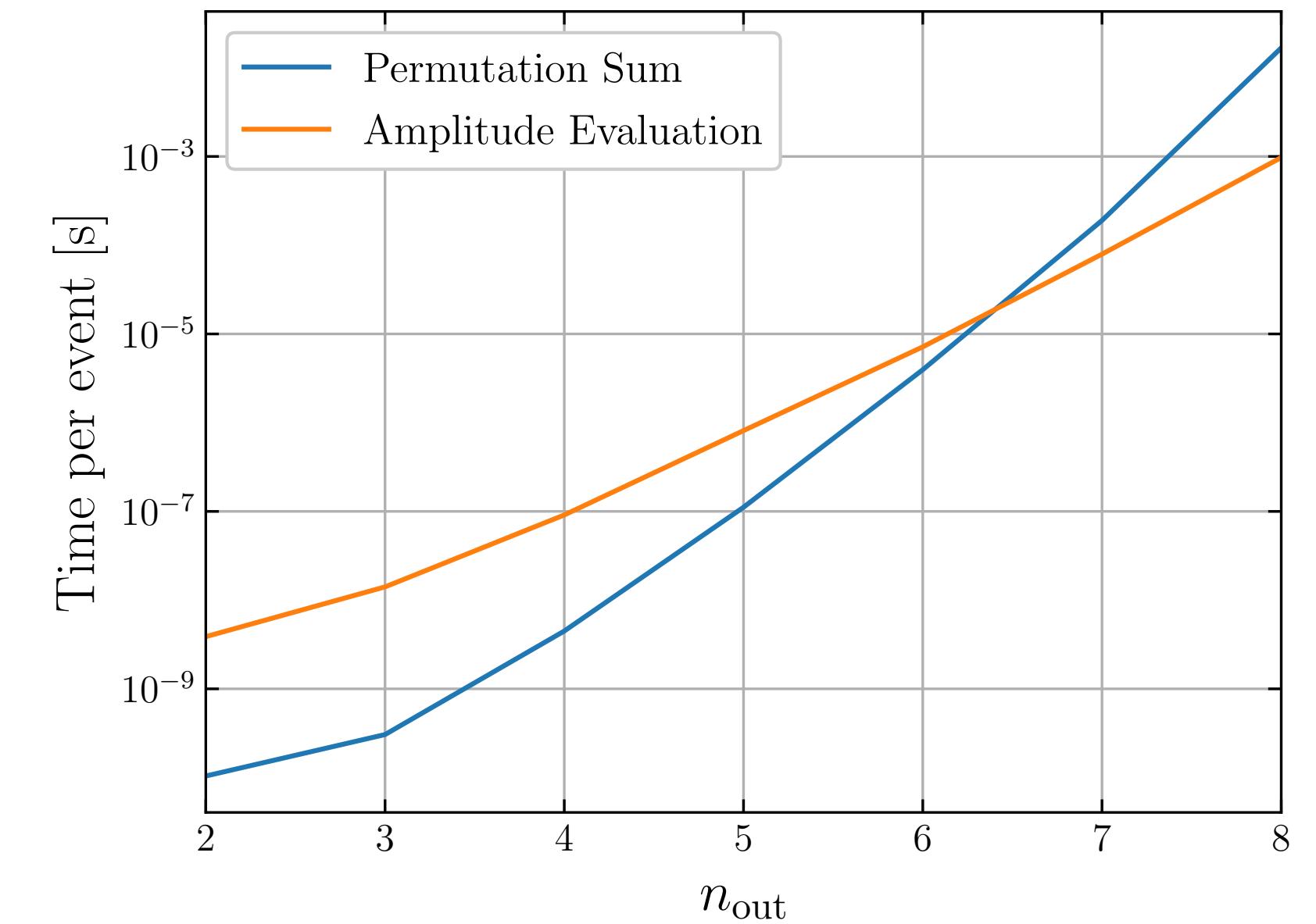


BlockGen algorithms

- All use Berends-Giele recursion for optimal scaling $\mathcal{O}(n^3 \dots n^4)$ for amplitude evaluation with number of particles n , instead of approx. $\mathcal{O}((2n)!)$ using Feynman diagrams
- Colour treatment varies:
 - Leading-Colour variant **BlockGen-LC** for direct comparison with 2010 study [Giele Stavenga Winter 1002.3446]
 - Colour-ordered and -summed variant **BlockGen-CO $_{\Sigma}$**
 - minimal memory for amplitude evaluation (because colour d.o.f. stripped)
 - but $\mathcal{O}(n!^2)$ growth of colour matrix (storage and colour sum evaluation)
 - Colour-dressed and (continuously) -sampled variant **BlockGen-CD $_{MC}$**



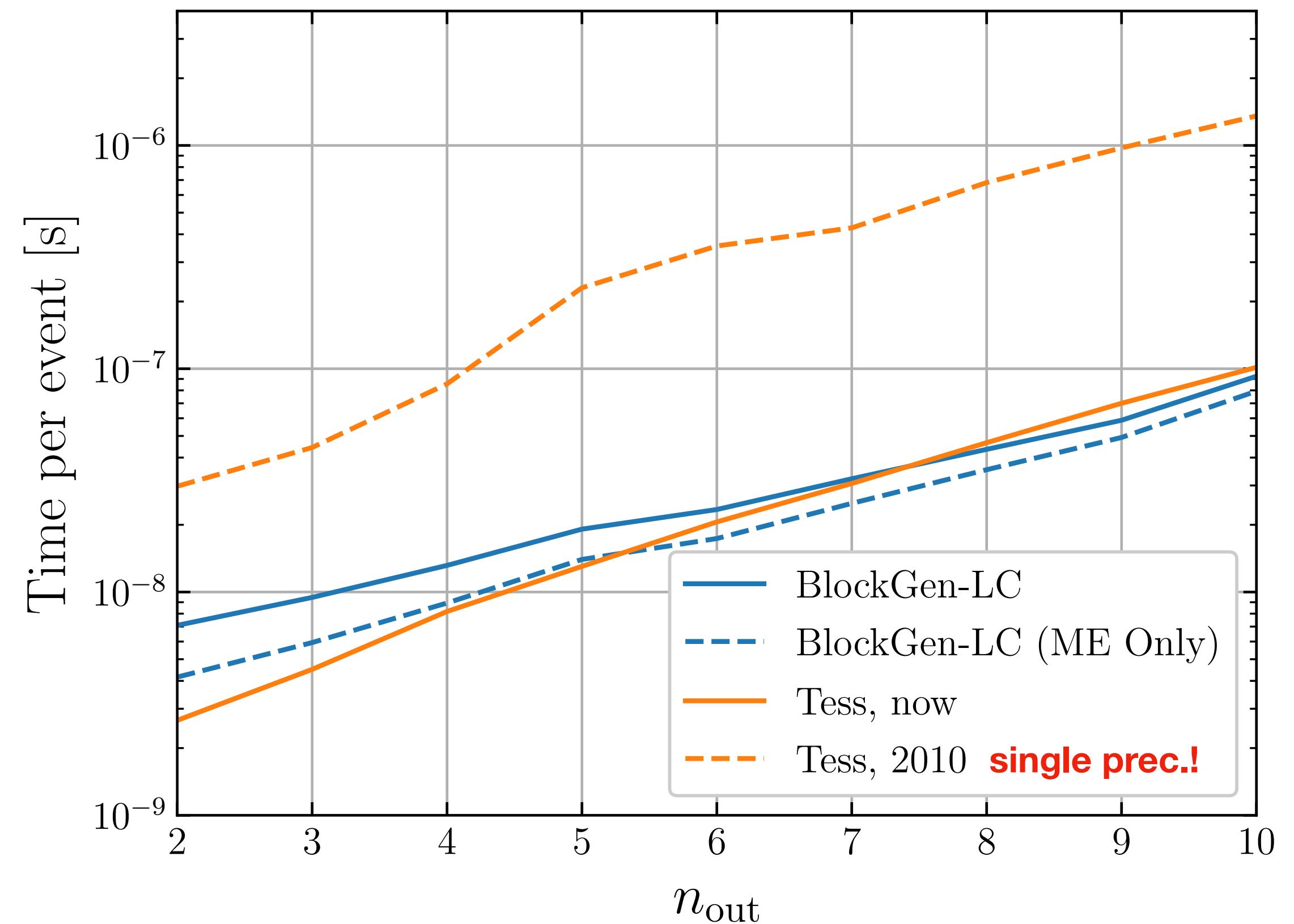
$$\mathcal{A}_{a_1 \dots a_n}^{\lambda_1 \dots \lambda_n}(p_1, \dots, p_n) = \sum_{\vec{\sigma} \in S_{n-2}} (F^{a_{\sigma_2}} \dots F^{a_{\sigma_{n-1}}})_{a_1 a_n} A^{\lambda_1 \dots \lambda_n}(p_1, p_{\sigma_2}, \dots, p_{\sigma_{n-1}}, p_n)$$



“Historical” comparison

2010's Tess vs. BlockGen-LC

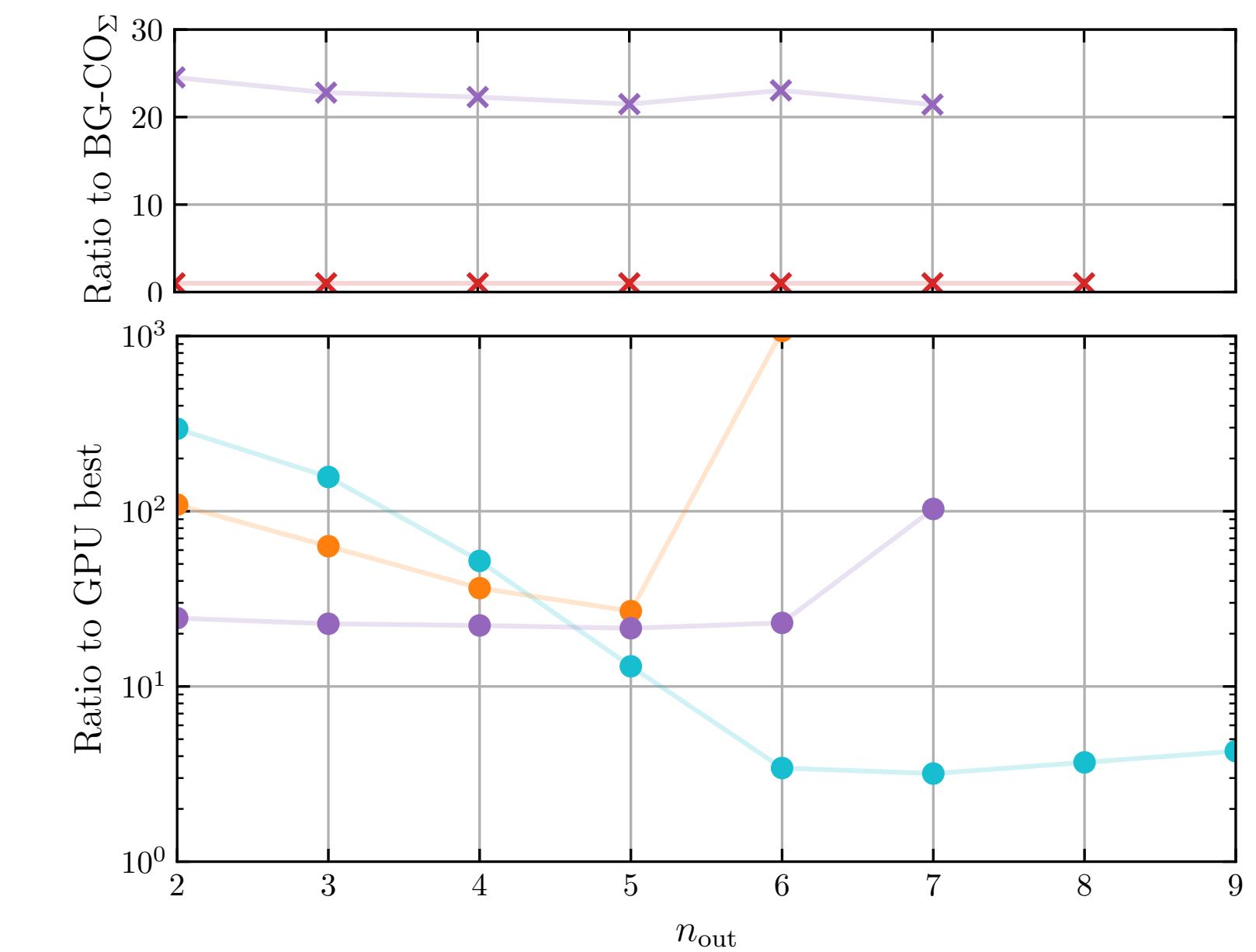
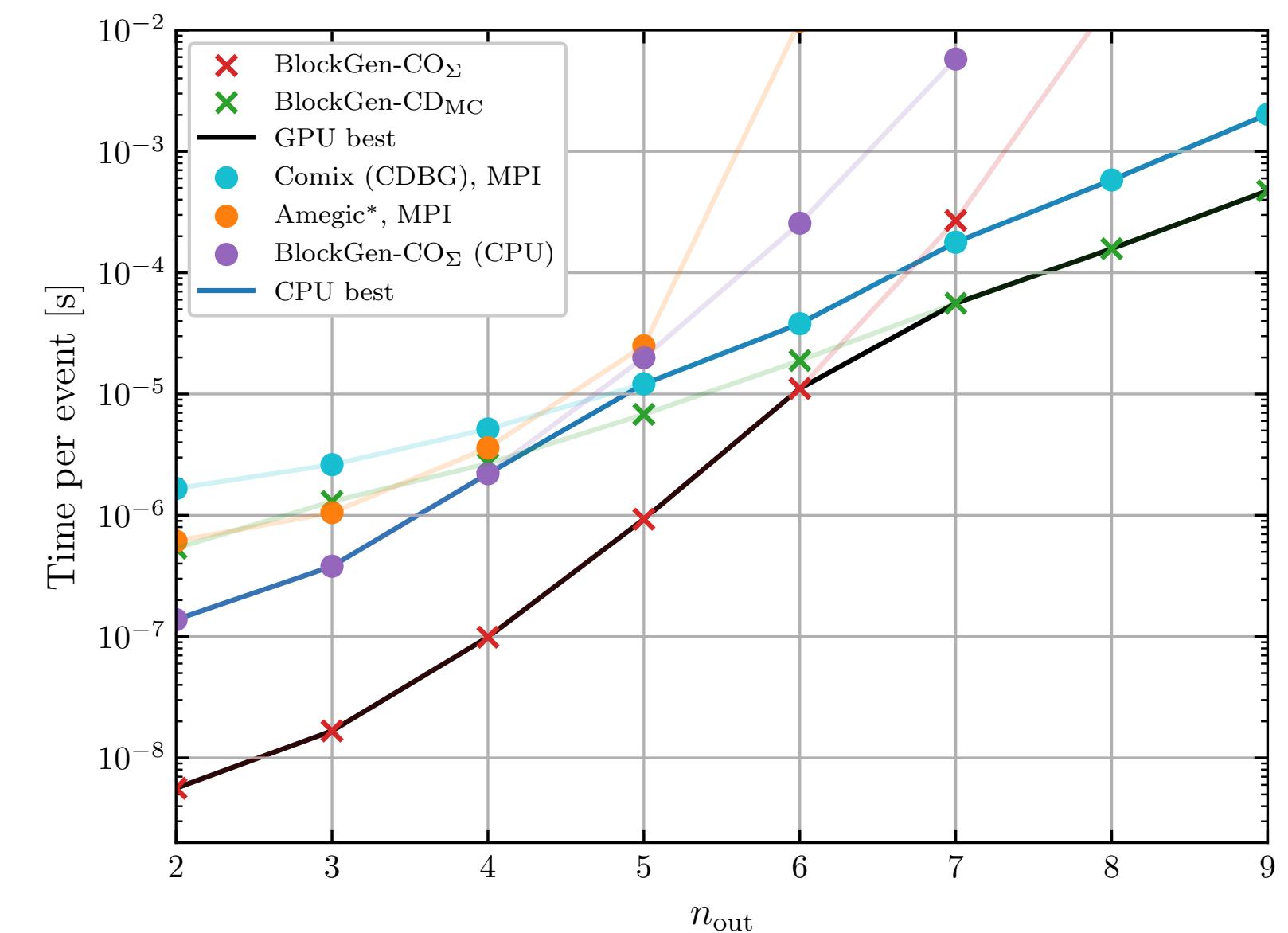
- Hardware x20 compared to 2010
NVIDIA V100 (16GB global memory, 5,120 CUDA cores, 6144KB L2 cache)
- much simpler BlockGen-LC code performs similarly
 - without managing cache manually (“shared memory”) or using intra-event parallelisation
 - possible reason: more+unified chip-controlled L2 cache, and larger throughput
- new Hardware faster & needs less optimisation effort



CPU vs. GPU

a chip-to-chip comparison

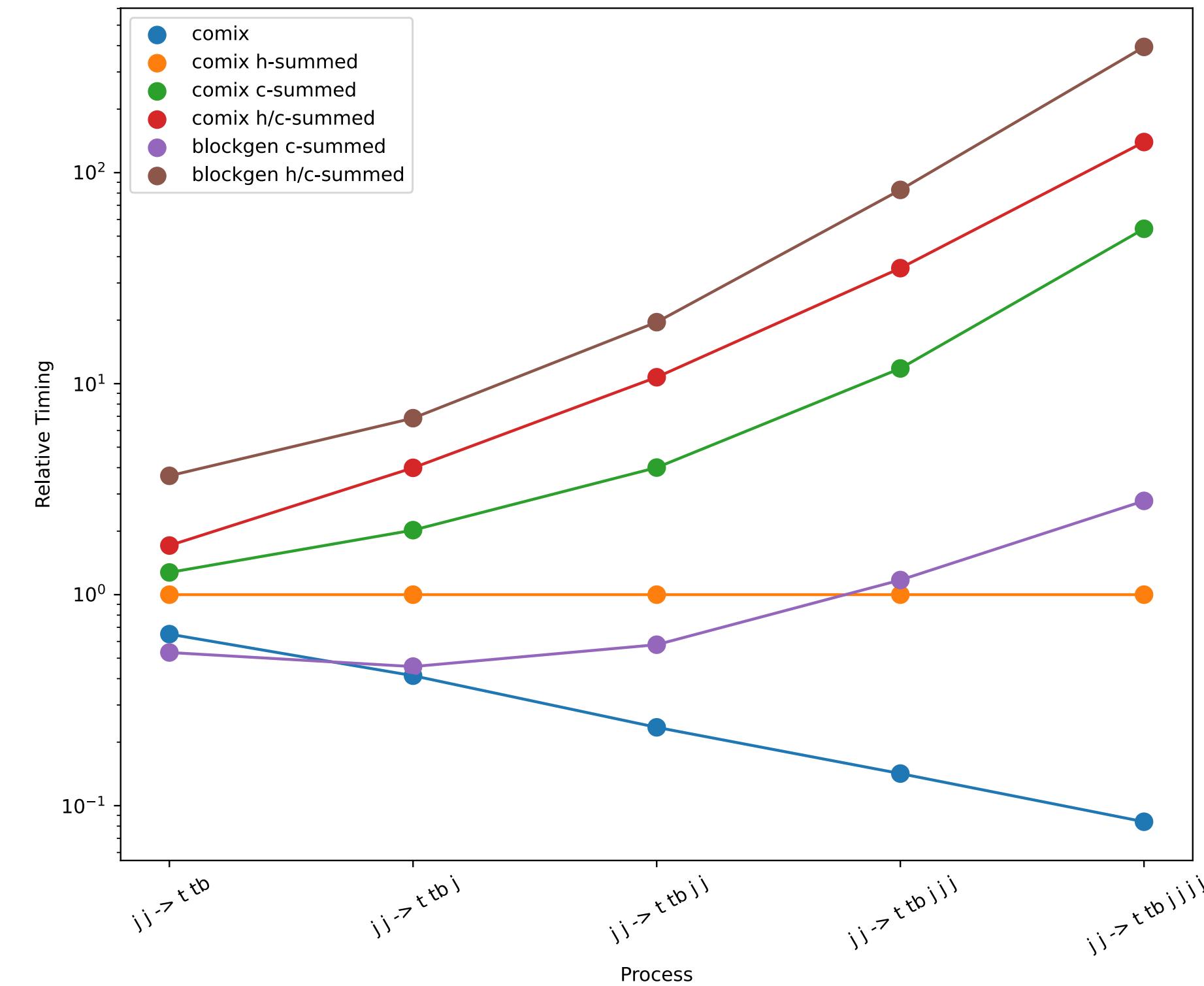
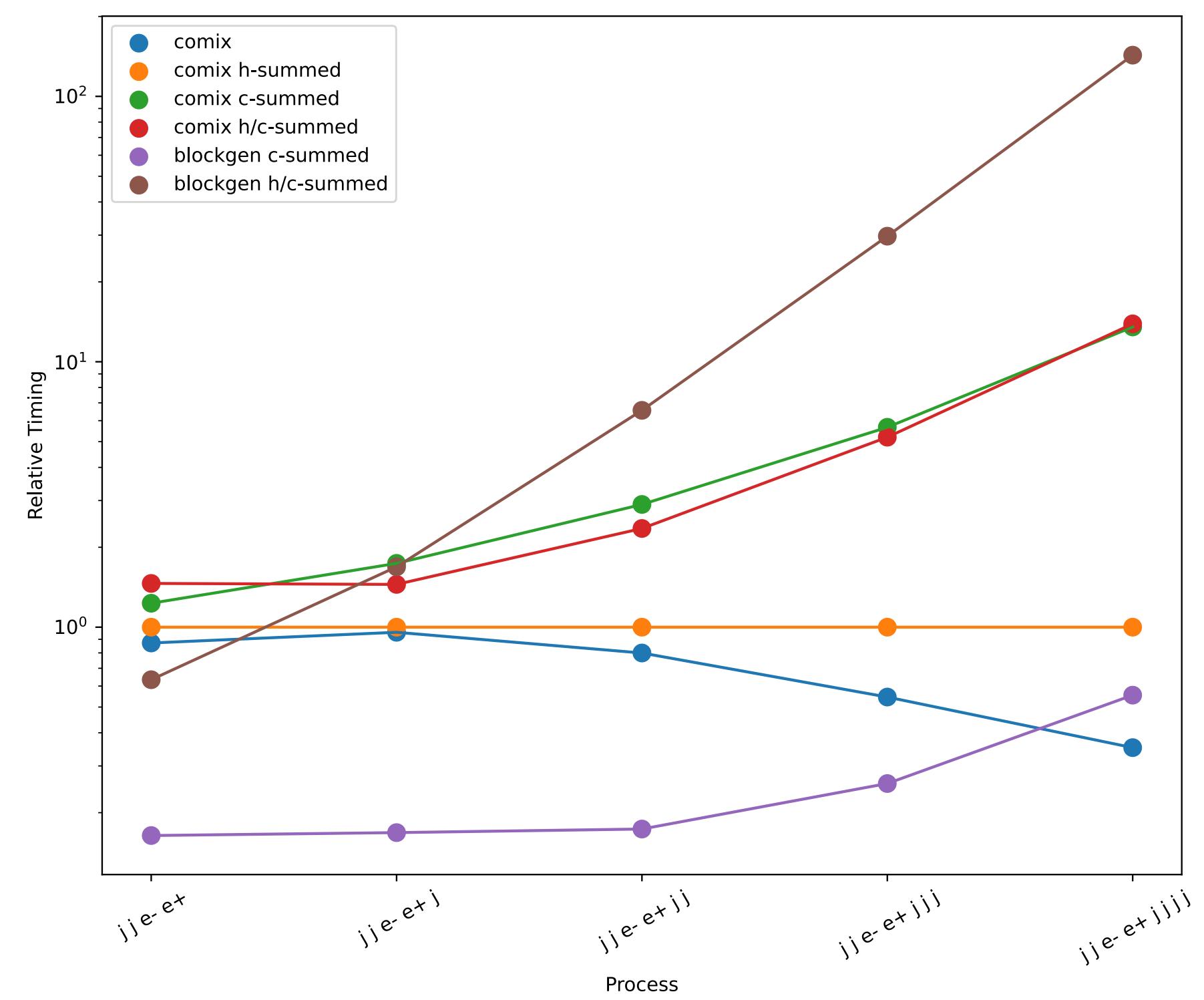
- CPU: Intel Xeon E5-2650 v2
2.6 GHz, 20 MB cache
 - use all 16 threads on the CPU (via MPI) → use “whole” chip
- GPU: NVIDIA V100
16 GB mem, 5120 cores, 6144KB L2
- BlockGen-CO $_{\Sigma}$ runs 20–25x times faster on GPU compared to CPU
- for $n_{\text{out}} < 6$, BlockGen-CO $_{\Sigma}$ wins, 10–25x faster than best CPU code
- for $n_{\text{out}} > 6$, BlockGen-CD $_{\text{MC}}$ best, but COMIX comes close
- Caveats
 - newer Intel Xeon Platinum might be more fair, expect factor of 2, mainly from doubled cache size
 - BG codes use real numbers only, complex numbers would slow them down by factor of 2, unavoidable with fermionic currents
 - on the other hand: colour sampling codes would be penalised by factors of about 2



WIP: Building a standard candle parton-level event generator

- generalise BlockGen to full QCD, $V+jets$ & $t\bar{t}+jets$
- single-threaded ME calculator part is DONE:
 - Colour-ordered Berends–Giele recursion
 - First time in a code aimed for production: colour d.o.f.'s use minimal basis of $(n - 2)!/k!$ amplitudes
[T. Melia 1304.7809 & 1312.0599 & 1509.03297; H. Johansson, A. Ochirov, 1507.00332]
 - Compare to more familiar gluon-only case: "naive" fundamental-representation decomposition gives $(n - 1)!$ amplitudes, but the minimal adjoint-representation decomposition only gives $(n - 2)!$ amplitudes
 - Similar multi-gluon performance as COMIX, but faster for many quarks ($40\times$ for $d\bar{d} \rightarrow u\bar{u}c\bar{c}\bar{s}\bar{s}$)
 - real-world performance will be a mix of channels → next slide

WIP: Building a standard candle parton-level event generator



REMAINING STEPS

- GPU port of matrix element calculation [Taylor Childers, Rui Wang](#)
→ again hope for factor of 10 or more in chip-to-chip comparison
- addition of recursive phase-space generator to have fully accelerated tree-level generator
- ... then eventually plug it into Sherpa as the seed event provider

Conclusions

- explored algorithm space based on BG recursion for CPU vs. GPU hardware
 - no process-specific or overly complicated optimisations
- algorithm that performs best in chip-to-chip comparison ...
 - BlockGen-CO_Σ below $n_{\text{out}} = 7$ (speedup 10-25x)
 - BlockGen-CD_{MC} $n_{\text{out}} \geq 7$ (speedup ~4x)
- for practical purposes, BlockGen-CO_Σ good starting point for currently ongoing extensions
 - eventual goal: parton-level generator that delivers seed events to SHERPA for further processing

Backup

Introduction

Past and present GPU-accelerated event generation efforts

Related toolchain (PDF, PS ...)

- Stratified phase-space sampling (ZMCintegral) using Tensorflow/Numba
[Wu Zhang Pang Wang arXiv:1902.07916]
- VEGAS using OpenCL and MPI
[Grasseau:2015vfa]
- VegasFlow, PDFFlow using TensorFlow
[Carrazza et al 2010.09341 2105.10529]
- LHAPDF using KOKKOS
[yet unreleased]
- ME method using HELAS/MadGraph and CUDA/OpenCL [Schouten et al. 1407.7595, Grasseau:2015qhg, Grasseau:2019sih]

Loops

- Feynman loop integrals [Yuasa:2013qza]
- pySecDec multi-loop integrals/amplitudes using CUDA [G. Heinrich, S. Jones et al 1811.11720]

(Tree-level) ME generators

Early conceptual studies

- HELAS/HEGET/MadGraph+VEGAS/BASES using CUDA (2010–2012)
[Hagiwara, Kanzaki et al. 0908.4403, 0909.5257, 1010.2107, 1305.0708]
- Berends–Giele gluon amplitudes using CUDA
[Giele Stavenga Winter 1002.3446]

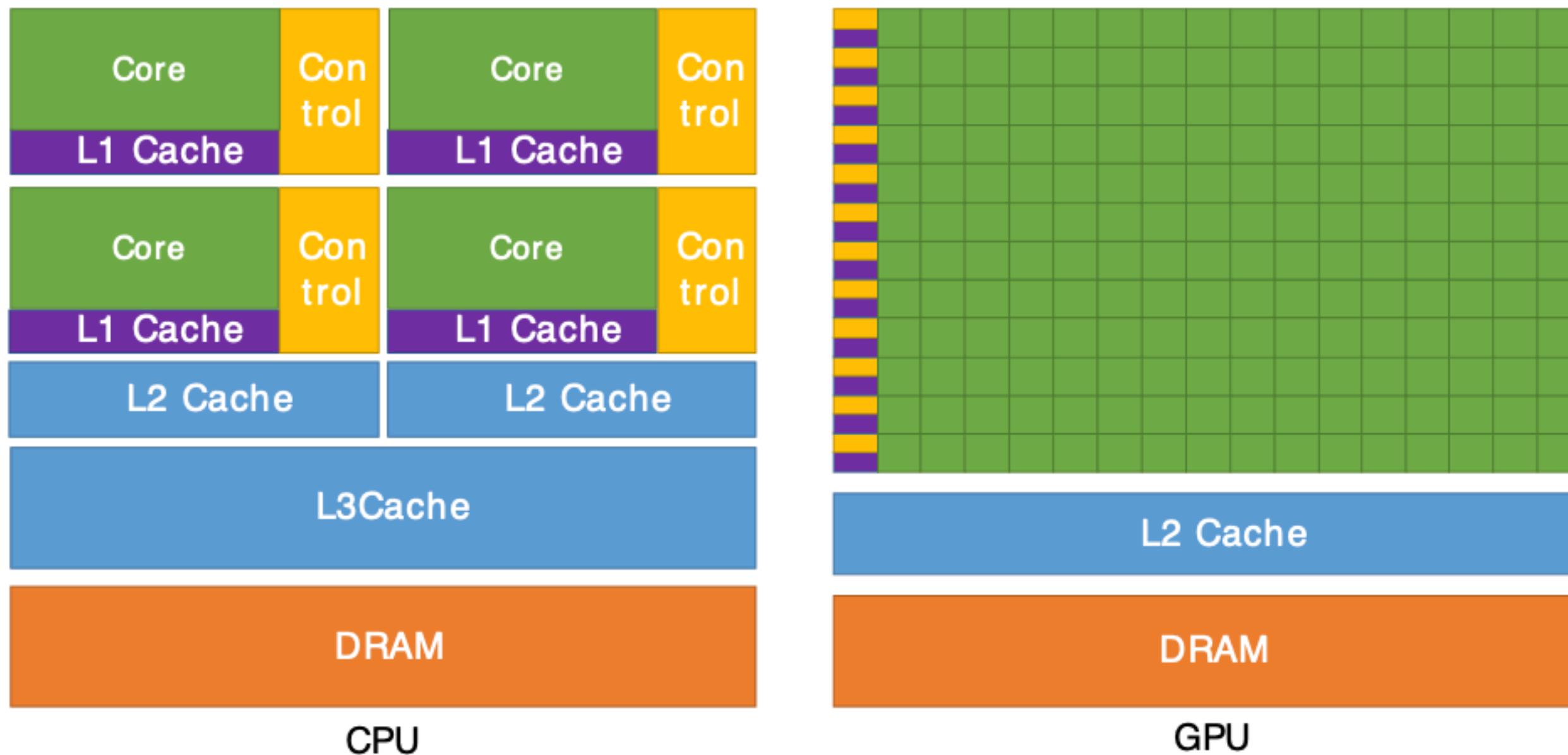
Recent efforts

- MadFlow using TensorFlow [Carrazza et al 2106.102796]
- WIP MadGraph renewed effort using KOKKOS et al [Valassi Roiser Mattelaer Hagebock 2106.12631]
- Berends–Giele gluon amplitudes using CUDA
[arXiv:2106.06507 EB Giele Höche Isaacson Knobbe]

GPU programming

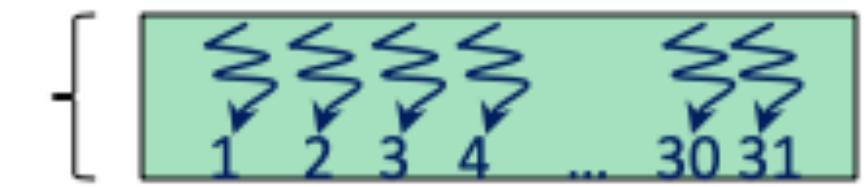
- Lock-step (SIMT) → avoid thread divergence
- Coalescent memory access
- (Block-)shared memory latency is roughly 100x lower than uncached global memory latency

Figure 1. The GPU Devotes More Transistors to Data Processing



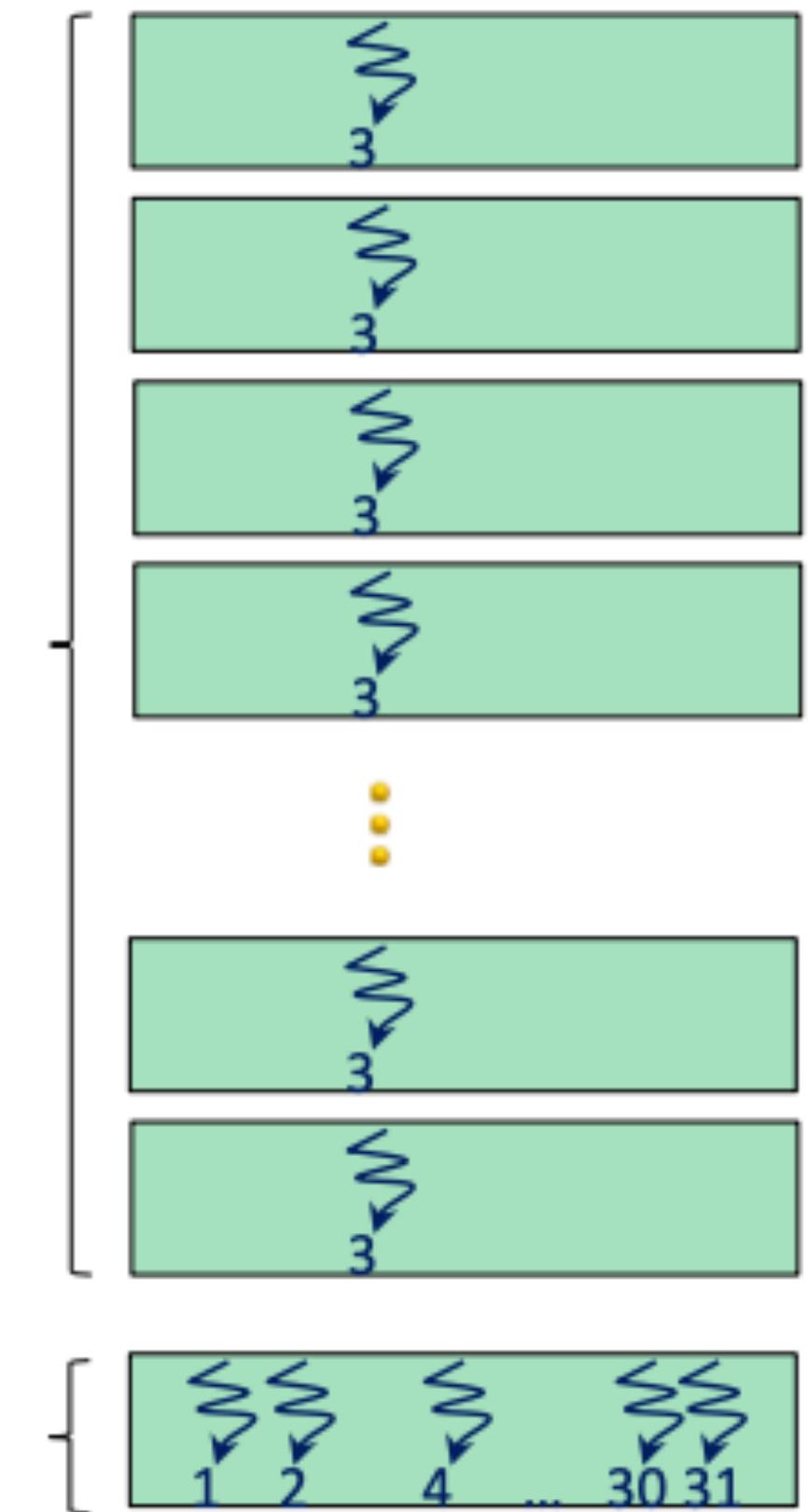
Coalescent memory load/write :

```
array[threadIdx.x] = 0.0;
```



Thread divergence, no SIMT :

```
if(threadIdx.x == 3) {  
    for(i=0; i < 1000; i++) {  
        array[threadIdx.x]++;  
    }  
} else {  
    array[threadIdx.x] = 3.14;  
}  
  
if(threadIdx.x == 3) {  
    for(i=0; i < 1000; i++) {  
        array[threadIdx.x]++;  
    }  
} else {  
    array[threadIdx.x] = 3.14;  
}
```



Amplitudes

- Colour and spin summed squared n-gluon tree-level amplitude \mathcal{A} :

$$|\mathcal{A}(1, \dots, n)|^2 = \sum_{a_1 \dots a_n} \mathcal{A}_{a_1 \dots a_n}^{\mu_1 \dots \mu_n}(p_1, \dots, p_n) \left(\mathcal{A}_{a_1 \dots a_n}^{\nu_1 \dots \nu_n}(p_1, \dots, p_n) \right)^{\dagger} \prod_{i=1}^n P_{\mu_i \nu_i}(p_i)$$

Colour sum treatment of external polarisations,
usually written: $P_{\mu\nu}(p) = \sum_{\lambda} \epsilon_{\mu}^{\lambda}(p) \epsilon_{\nu}^{\lambda\dagger}(p)$

- two considerations:
 - efficiently calculate individual amplitudes $\mathcal{A}_{a_1 \dots a_n}^{\mu_1 \dots \mu_n}(p_1, \dots, p_n)$
 - efficiently sum over colour and helicity (perhaps sample in a MC fashion)

Helicity/colour summing/sampling

- Instead of summing, use discrete Monte-Carlo sum over various λ
- Alternative: use continuous sampling of polarisations via
[Draggiotis Kleiss Papadopoulos hep-ph/9807207, hep-ph/0202201]

$$e_\mu(p, \phi) = e^{i\phi} \epsilon_\mu^+(p) + e^{-i\phi} \epsilon_\mu^-(p), \quad P_{\mu\nu}(p) = \frac{1}{2\pi} \int_0^{2\pi} d\phi e_\mu(p, \phi) e_\nu^*(p, \phi)$$

- (for gluons can choose real-valued polarisations \rightarrow real-valued amplitudes)
- Less obvious: can also make colour a continuous variable

- Gluon colour polarisation vector $\eta^a(z, z') = \eta_i(z) T_{ij}^a \eta_j(z')$, represented by two real 3D spherical unit vectors
- similar to construction in [Draggiotis Kleiss Papadopoulos hep-ph/9807207, hep-ph/0202201], but 4 instead of 5 integration variables
- summed squared amplitude using continuous polarisations given by

$$|\mathcal{A}(1, \dots, n)|^2 = \prod_i \int \frac{d\phi_i}{2\pi} dz_i dz'_i \left| \mathcal{A}_{z_1 z'_1 \dots z_n z'_n}^{\phi_1 \dots \phi_n}(p_1, \dots, p_n) \right|^2$$

$$\eta_i(z) \rightarrow \eta_i(\theta, \phi) = \begin{pmatrix} \cos \theta \\ \sin \theta \cos \phi \\ \sin \theta \sin \phi \end{pmatrix}$$

$$\int dz \rightarrow \frac{N_c}{4\pi} \int_{-1}^1 d\cos \theta \int_0^{2\pi} d\phi$$

Colour bases and decompositions

- evaluate colour structure as part of the amplitude OR factorise amplitude first into sum of colour factors and corresponding kinematic parts=colour-ordered amplitudes (colour decomposition)
 - which is best depends on size, bandwidth and access pattern of fast memory of the hardware!
- colour decomposition for gluon-only, using adjoint basis (\rightarrow minimum number of $(n - 2)!$ colour-ordered amplitudes A):
[\[Berends Giele Nucl. Phys. B294, 700 \(1987\), Del Duca Dixon Maltoni hep-ph/9910563, Del Duca Frizzo Maltoni hep-ph/9909464\]](#)

$$\mathcal{A}_{a_1 \dots a_n}^{\lambda_1 \dots \lambda_n}(p_1, \dots, p_n) = \sum_{\vec{\sigma} \in S_{n-2}} (F^{a_{\sigma_2}} \dots F^{a_{\sigma_{n-1}}})_{a_1 a_n} A^{\lambda_1 \dots \lambda_n}(p_1, p_{\sigma_2}, \dots, p_{\sigma_{n-1}}, p_n)$$

$$|\mathcal{A}(1, \dots, n)|^2 = \sum_{\vec{\sigma}, \vec{\sigma}' \in S_{n-2}} \mathcal{C}_{\vec{\sigma} \vec{\sigma}'} \sum_{\lambda_1 \dots \lambda_n} A^{\lambda_1 \lambda_{\sigma_2} \dots \lambda_{\sigma_{n-1}} \lambda_n}(p_1, p_{\sigma_2}, \dots, p_{\sigma_{n-1}}, p_n) \times A^{\lambda_1 \lambda'_{\sigma'_2} \dots \lambda'_{\sigma'_{n-1}} \lambda_n}(p_1, p_{\sigma'_2}, \dots, p_{\sigma'_{n-1}}, p_n)^\dagger$$

- get $(n - 2)!((n - 2)! - 1)/2 = \mathcal{O}((n - 2)!^2)$ colour coefficients $\mathcal{C}_{\vec{\sigma} \vec{\sigma}'} = \sum_{a_1 \dots a_n} (F^{a_{\sigma_2}} \dots F^{a_{\sigma_{n-1}}})_{a_1 a_n} (F^{a'_{\sigma'_2}} \dots F^{a'_{\sigma'_{n-1}}})_{a_1 a_n}^*$
- colour-flow decomposition, treating gluon as $N_c \times N_c$ field:
[\[Maltoni, Paul, Stelzer, Willenbrock hep-ph/0209271\]](#)

$$\mathcal{A}_{i_1 j_1 \dots i_n j_n}^{\lambda_1 \dots \lambda_n}(p_1, \dots, p_n) = \sum_{\vec{\sigma} \in S_{n-1}} \delta^{i_1 \bar{j}_{\sigma_2}} \delta^{i_{\sigma_2} \bar{j}_{\sigma_3}} \dots \delta^{i_{\sigma_n} \bar{j}_1} A^{\lambda_1 \dots \lambda_n}(p_1, p_{\sigma_2}, \dots, p_{\sigma_n})$$

- more amplitudes than in adjoint basis, namely $(n - 1)!$, but straightforward to find primitive amplitudes for a given colour point (number varies!), trivial colour factor, nice properties for sampling colours instead of summing; great!(?)

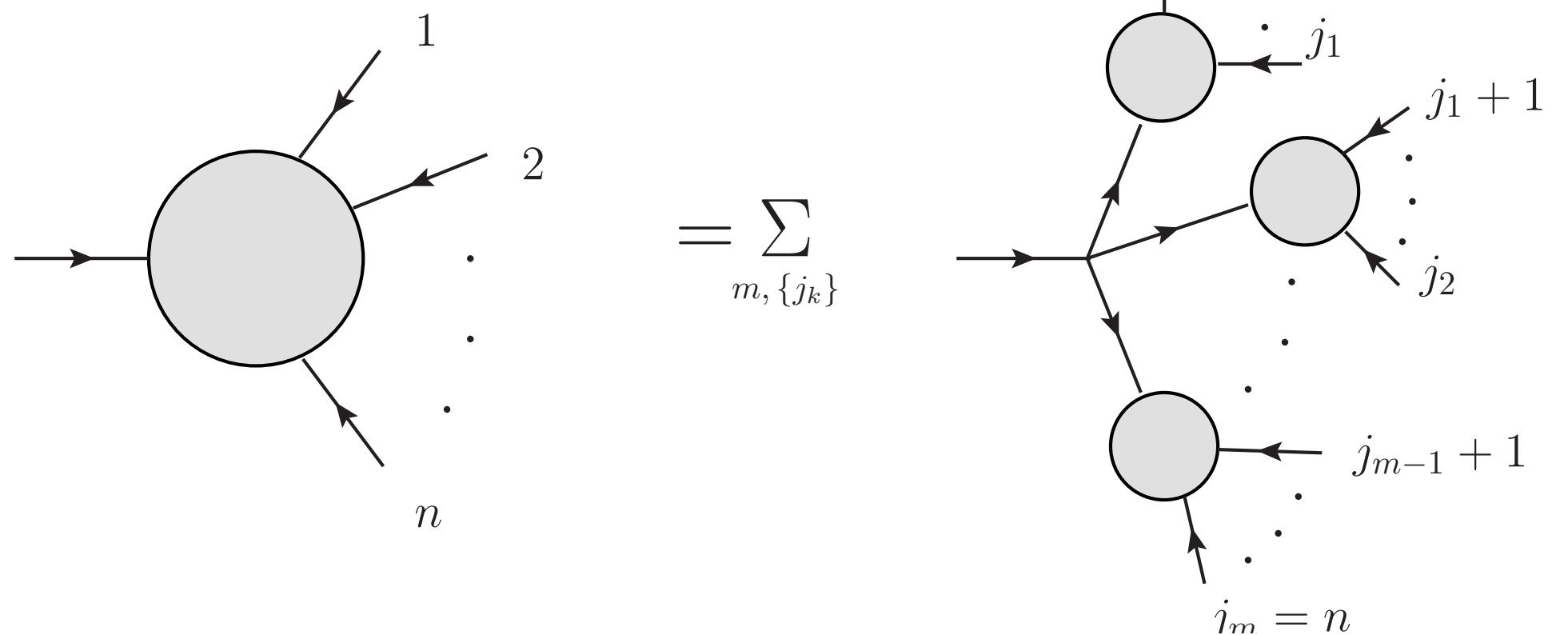
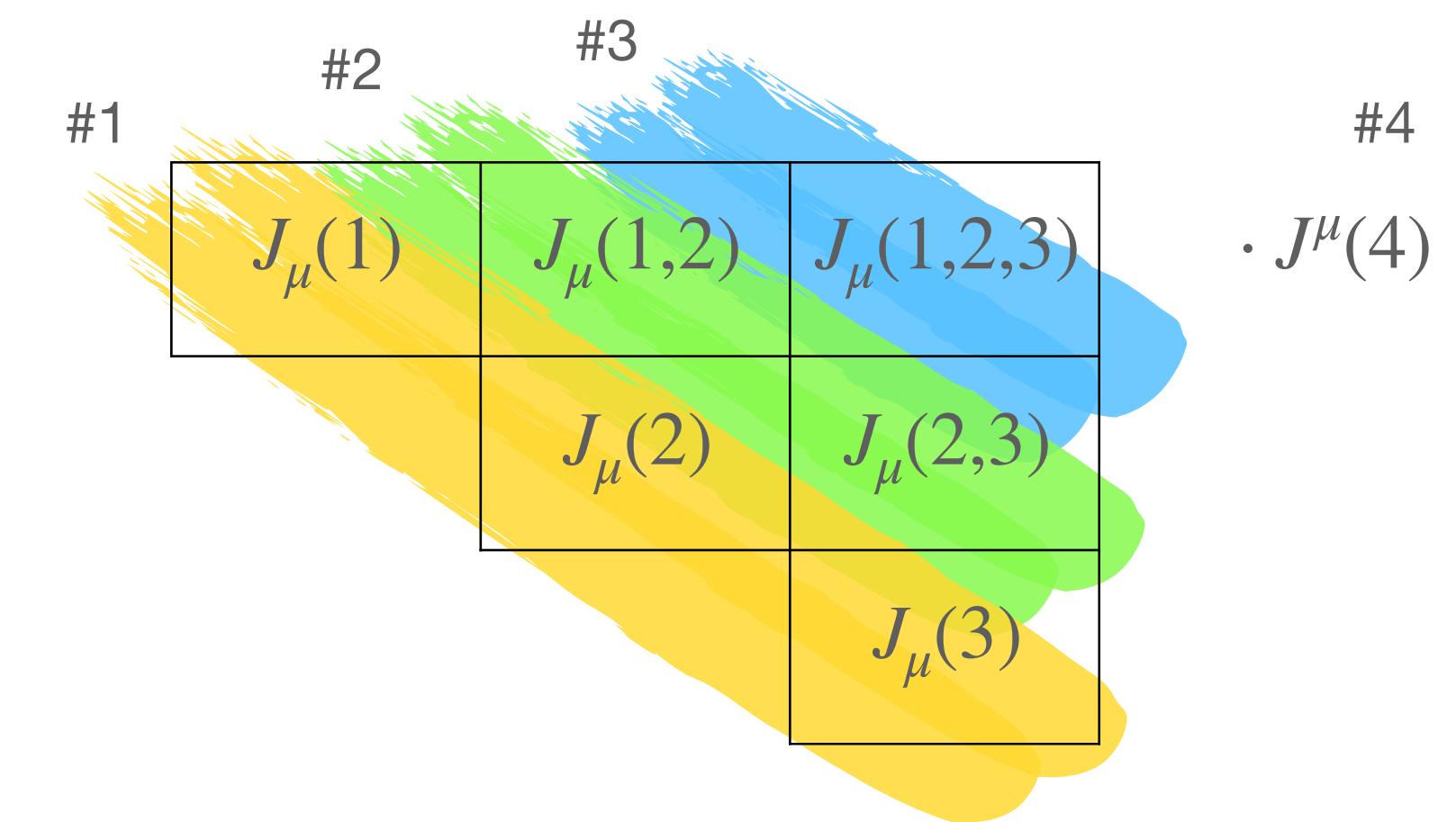
Berends–Giele recursion

[Berends Giele Nucl. Phys. B306, 759 (1988)]

- calculate colour-ordered (or „dressed“, i.e. with embedded colour) amplitudes
- starting point: external particle currents $J_\mu(i)$ = external helicity states (#1)
- uses a recursive computation of off-shell currents (#2, #3)

$$J_\mu(1,2,\dots,n) = \frac{-ig_{\mu\nu}}{p_{1,n}^2} \left\{ \sum_{k=1}^{n-1} V_3^{\nu\kappa\lambda}(p_{1,k}, p_{k+1,n}) J_\kappa(1,\dots,k) J_\lambda(k+1,\dots,n) + \sum_{j=1}^{n-2} \sum_{k=j+1}^{n-1} V_4^{\nu\rho\kappa\lambda} J_\rho(1,\dots,j) J_\kappa(j+1,\dots,k) J_\lambda(k+1,\dots,n) \right\}$$

- in the end, construct with remaining external particle current,
 $A(1,\dots,n) = J_\mu(n)p_{1,n}^2 J^\mu(1,\dots,n-1)$ (#4)
- $\mathcal{O}(n^4)$ from iteration over four-gluon vertices
 (instead of approx. $\mathcal{O}((2n)!)$ using diagrams)
 can be reduced to $\mathcal{O}(n^3)$ by using an auxiliary tensor field, but needs more memory
- colour-dressed version looks structurally the same,
 most convenient for colour flow, either discrete or continuous
 important difference: indices not fully ordered → easier to store subcurrents for reuse
 [Duhr Höche Maltoni hep-ph/0607057]



Tess and BlockGen-LC

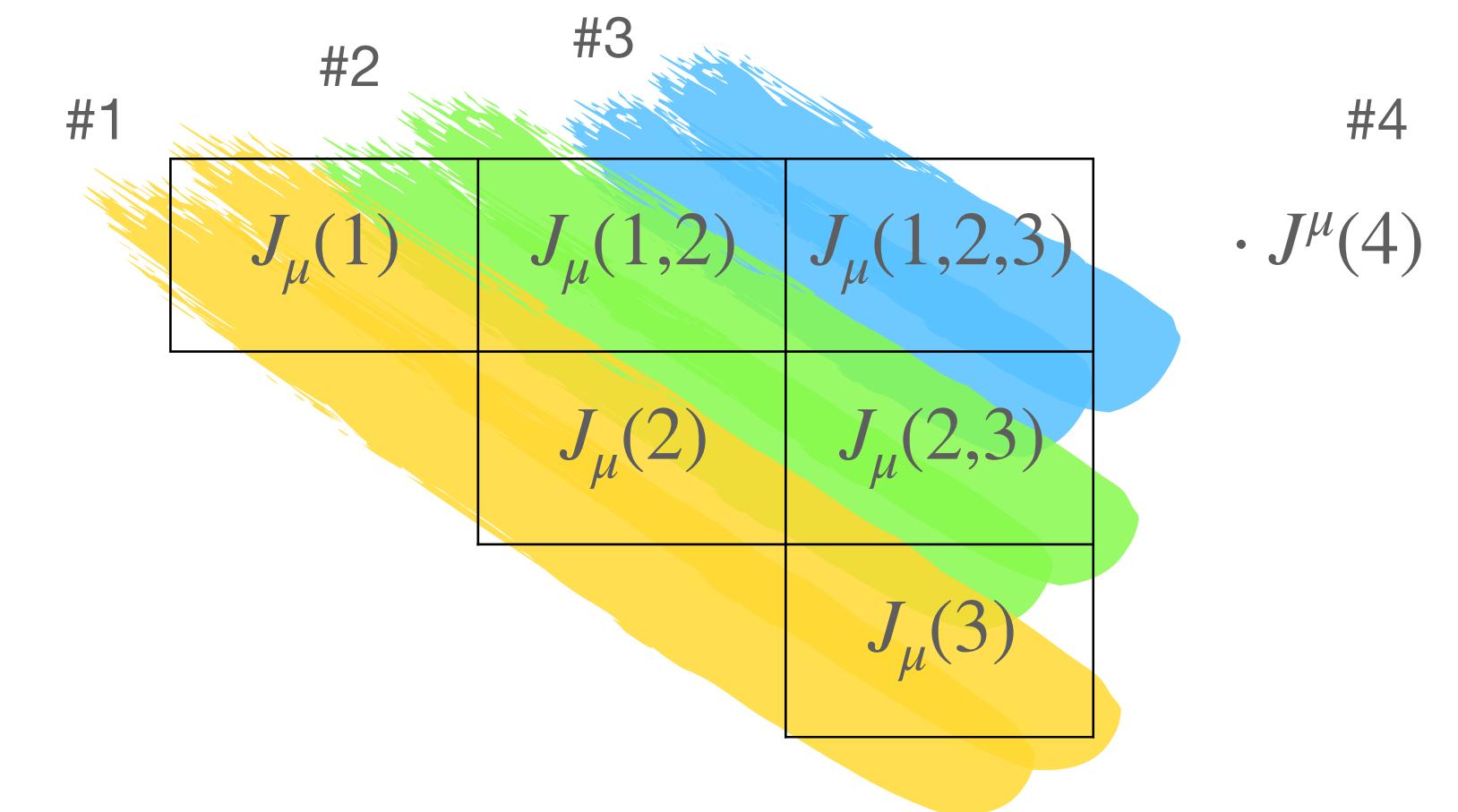
Starting point and comparison benchmark

[Giele Stavenga Winter 1002.3446]

- Adjoint colour-ordered BG recursion, sampling of real polarisation vectors (real-valued algorithm)
- Parallelisation over events, *and* over currents of the recursion
(*up to* $(n - 1)$ threads cooperate on a single event)
- currents and momentum sums on self-managed cache (shared memory) $\sim \mathcal{O}(n^2)$
- Leading-colour, thus avoiding $(n - 2)!^2$ scaling of colour sum

$$\left| \mathcal{A}_n(1, \dots, n) \right|^2 = N_c^{n-2} (N_c^2 - 1) \left(\sum_{\sigma \in S_{n-1}} \left| A^{\lambda_1 \dots \lambda_n} (p_1, p_{\sigma_2}, \dots, p_{\sigma_n}) \right|^2 + \mathcal{O}\left(\frac{1}{N_c^2}\right) \right)$$

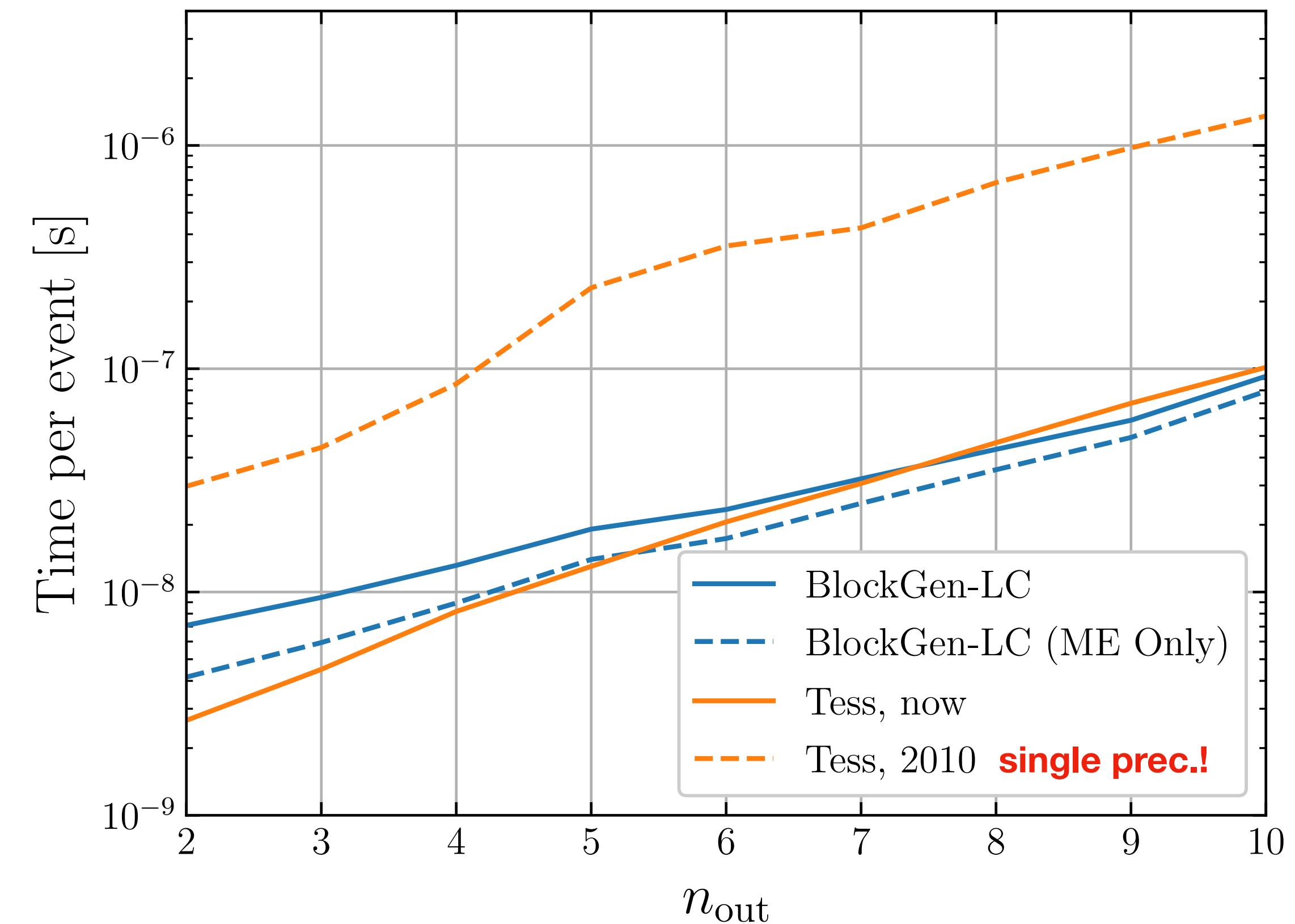
- Re-implemented without shared memory or intra-event parallelisation → BlockGen-LC



Leading colour

Tess vs. BlockGen-LC

- Hardware x20 compared to 2010
NVIDIA V100 (16GB global memory, 5,120 CUDA cores, 6144KB L2 cache)
- simpler BG-LC code performs similarly
 - not using shared memory or intra-event parallelisation not an issue
 - possible reason: more+unified chip-controlled L2 cache, and larger throughput



Full Colour Summed BlockGen-CO $_{\Sigma}$

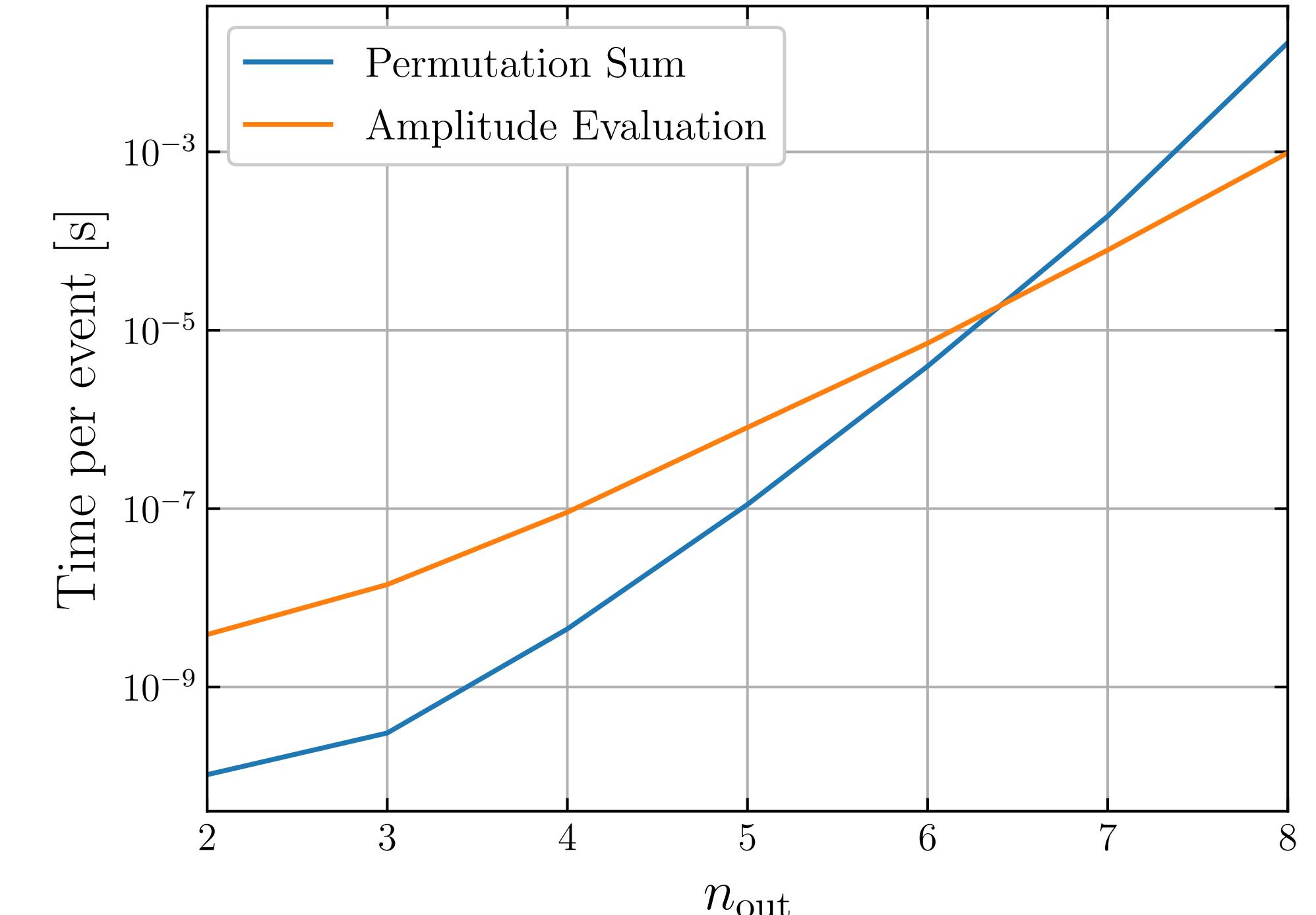
- colour-ordered & -summed BlockGen-CO $_{\Sigma}$
- minimal adjoint colour decomposition:
 $(n - 2)!$ primitive amplitudes
- using FORM-precalculated (stored in global memory)

$$\mathcal{C}_{\vec{\sigma}\vec{\sigma}'} = \sum_{a_1 \dots a_n} (F^{a_{\sigma_2}} \dots F^{a_{\sigma_{n-1}}})_{a_1 a_n} (F^{a'_{\sigma'_2}} \dots F^{a'_{\sigma'_{n-1}}})_{a_1 a_n}^*$$

- summing $(n - 2)!((n - 2)! + 1)/2$ amplitudes eventually overtakes amplitude evaluation

$$|\mathcal{A}(1, \dots, n)|^2 = \sum_{\vec{\sigma}, \vec{\sigma}' \in S_{n-2}} \mathcal{C}_{\vec{\sigma}\vec{\sigma}'} \sum_{\lambda_1 \dots \lambda_n} A^{\lambda_1 \lambda_{\sigma_2} \dots \lambda_{\sigma_{n-1}} \lambda_n} (p_1, p_{\sigma_2}, \dots, p_{\sigma_{n-1}}, p_n) \times A^{\lambda_1 \lambda'_{\sigma'_2} \dots \lambda'_{\sigma'_{n-1}} \lambda_n} (p_1, p_{\sigma'_2}, \dots, p_{\sigma'_{n-1}}, p_n)^{\dagger}$$

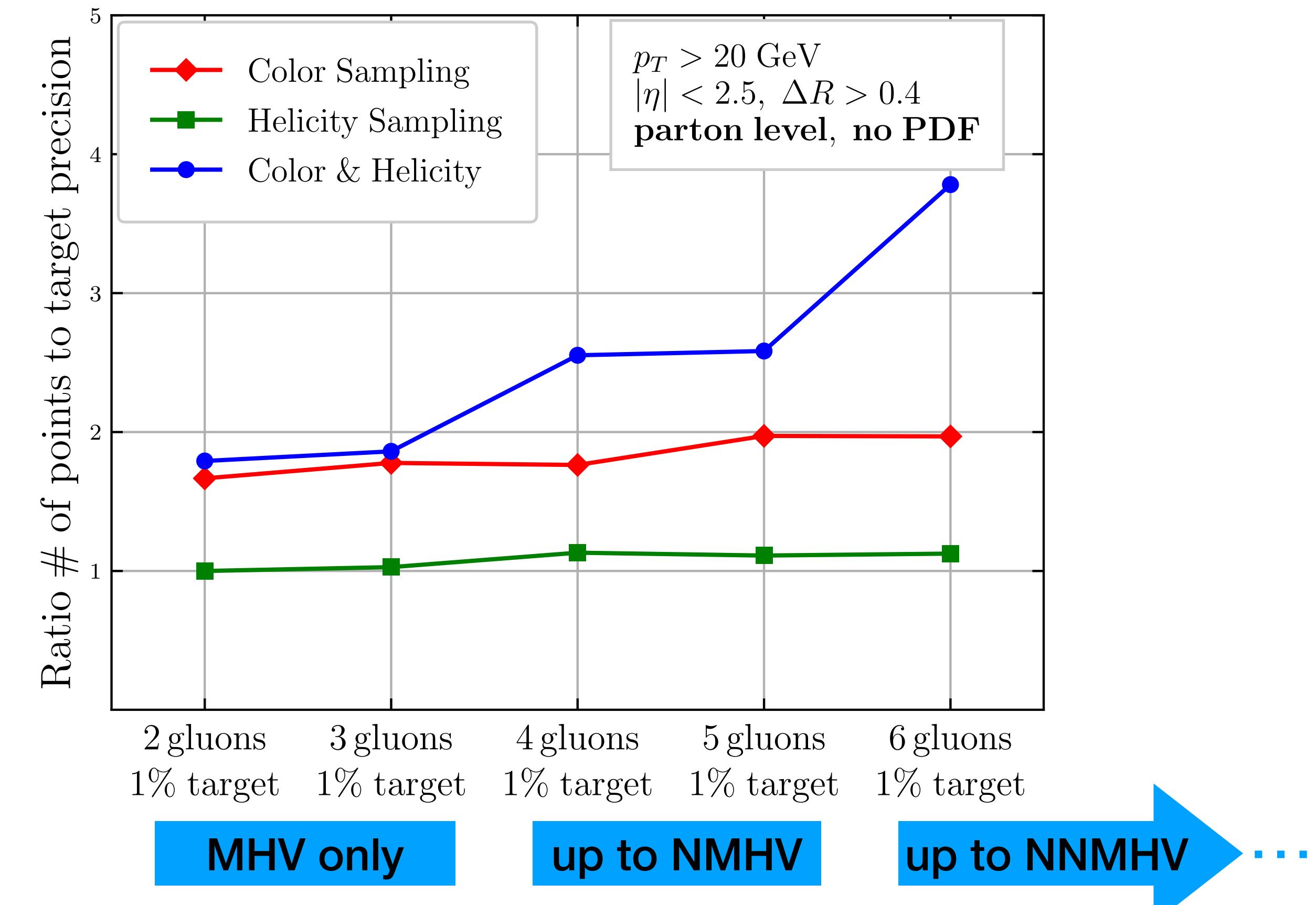
- relevance for practical applications?



Summing vs. sampling

Can we afford just to sample?

- study realistic parton-level calculation using COMIX
- how many more points do we need?
- colour OR helicity \rightarrow ~constant overhead
- combined sampling overhead increases with n in $N^n\text{MHV}$
- likely sampling always better than summing \rightarrow define sampled algos



Full Colour Sampled BlockGen

- colour-ordered & -sampled BlockGen- CO_{MC}

- sample colours using color-flow decomposition

$$\mathcal{A}_{i_1 j_1 \dots i_n j_n}^{\lambda_1 \dots \lambda_n}(p_1, \dots, p_n) = \sum_{\vec{\sigma} \in S_{n-1}} \delta^{i_1 \bar{J}_{\sigma_2}} \delta^{i_{\sigma_2} \bar{J}_{\sigma_3}} \dots \delta^{i_{\sigma_n} \bar{J}_1} A^{\lambda_1 \dots \lambda_n} (p_1, p_{\sigma_2}, \dots, p_{\sigma_n})$$

- number of contributing amplitudes depends on colour configuration

- colour-dressed & -sampled BlockGen- CD_{MC}

- Continuos colour „polarisations“ → less integer arithmetics, more floating-point arithmetics
 - needs significantly more memory for recursion, storing all possible colour flows for each current:

$$\mathcal{J}_{\mu ij}(1, \dots, n) = \sum_{\vec{\sigma} \in S_n} \delta_{ij_{\sigma_1}} \delta_{i_{\sigma_1} j_{\sigma_2}} \dots \delta_{i_{\sigma_n} j} J_{\mu} (\sigma_1, \dots, \sigma_n)$$

CPU single-threaded

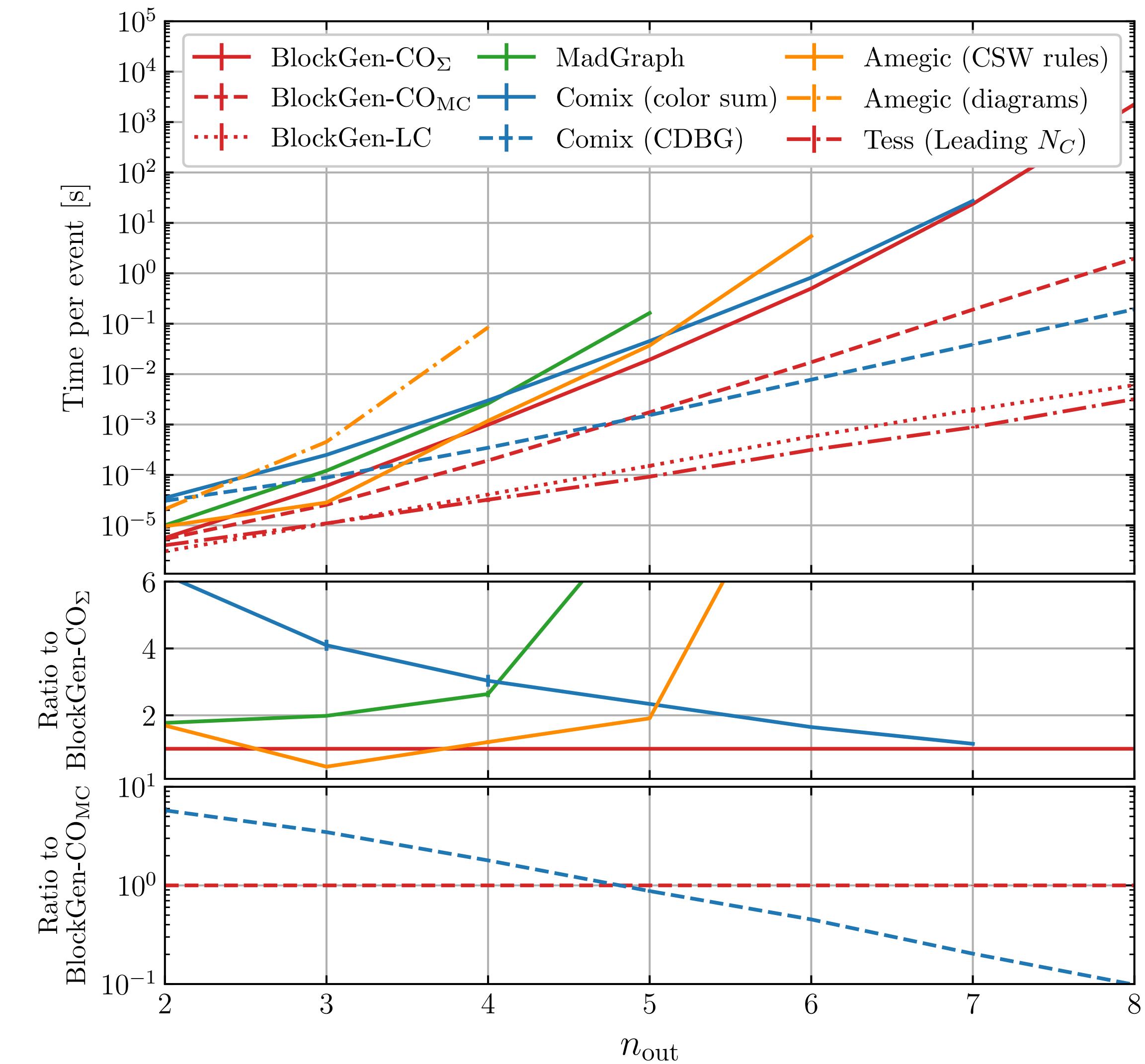
- for all results:

$\sqrt{s} = 14 \text{ TeV}$, RAMBO

$p_T > 20 \text{ GeV}$, $|\eta| < 2.5$, $\Delta R > 0.4$

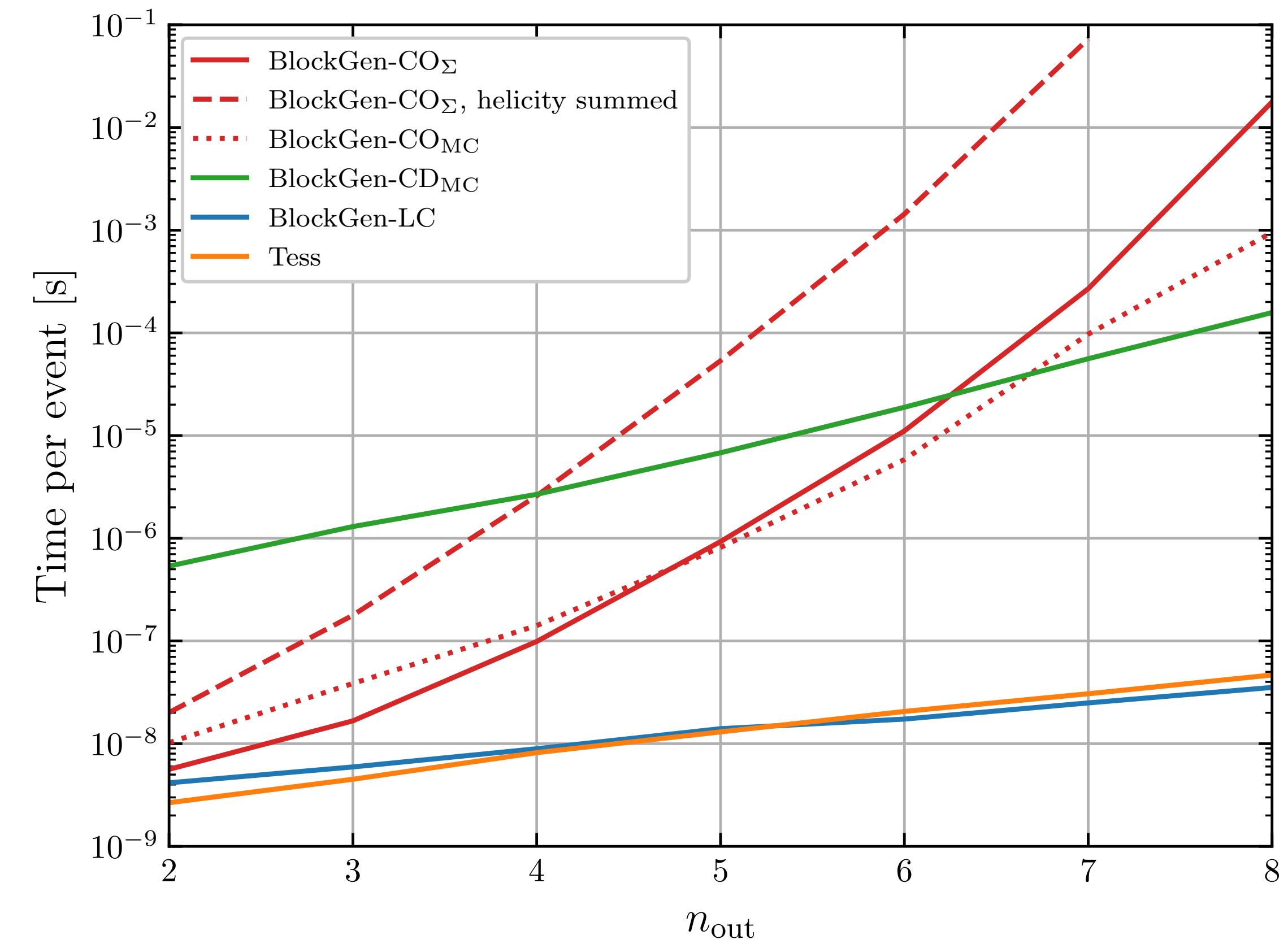
fixed $\mu_R = m_Z$ and $\alpha_s(m_Z) = 0.118$

- right: single-threaded performance on CPU
- helicity summed (except Tess); COMIX profits from reuse of precomputed sub-currents across helicity configs
- even taking into account penalty of sampling, it is significantly faster compared to summing



GPU multi-threaded

- helicity-sampled
- now $\text{CO}_\Sigma \sim \text{CO}_{\text{MC}}$
 - highlights a problem of the latter for SIMT: varying number of valid colour, configs \rightarrow threads not in lock-step
 - CO_{MC} scaling still better due to absence of $(n - 2)!^2$ colour sum scaling
- CD_{MC} has excellent scaling, but offset by $\sim 10^2$ due to increased memory use of recursion



Johannsen–Ochirov–Melia decomposition

Minimal colour basis for arbitrary QCD amplitudes

[T. Melia 1304.7809 & 1312.0599 & 1509.03297; H. Johannsson, A. Ochirov, 1507.00332]

- gluon-only starting from fundamental representation, gives $(n - 1)!$ gluon CO amplitudes:

$$\mathcal{A}_{n,0}^{\text{tree}} = \sum_{\sigma \in S_{n-1}(\{2, \dots, n\})} \text{Tr} (T^{a_1} T^{a_{\sigma(2)}} \dots T^{a_{\sigma(n)}}) A(1, \sigma(2), \dots, \sigma(n))$$

- not minimal, use Kleiss-Kuijf relations $A(1, \beta, 2, \alpha) = (-1)^{|\beta|} \sum_{\sigma \in \alpha \sqcup \beta^T} A(1, 2, \sigma)$ [Kleiss-Kuijf Nucl.Phys. B312 (1989) 616]
- allows to fix second index in gluon primitives, gives minimal Del-Duca–Dixon–Maltoni (DDM) decomposition:

$$\mathcal{A}_{n,0}^{\text{tree}} = \sum_{\sigma \in S_{n-2}(\{3, \dots, n\})} \tilde{f}^{a_2 a_{\sigma(3)} b_1} \tilde{f}^{b_1 a_{\sigma(4)} b_2} \dots \tilde{f}^{b_{n-3} a_{\sigma(n)} a_1} A(1, 2, \sigma(3), \dots, \sigma(n))$$

- Only $(n - 2)!$ truly independent CO amplitudes remain. Also works for single quark line.
[Kosower Lee Nair Phys.Lett. B201 (1988) 85, Mangano Nucl.Phys. B309 (1988) 461]

Johannsen–Ochirov–Melia decomposition

Minimal colour basis for arbitrary QCD amplitudes

[T. Melia 1304.7809 & 1312.0599 & 1509.03297; H. Johannsson, A. Ochirov, 1507.00332]

- Generalisation to $k > 1$ quark pairs found by Melia (primitive basis) and Johannsson/Ochirov (corresponding colour factors, later proven by Melia, ~2015)
- minimal basis of $(n - 2)!/k!$ CO amplitudes given by
 $\{A(\underline{1}, \bar{2}, \sigma) \mid \sigma \in \text{Dyck}_{k-1} \times \{\text{gluon insertions}\}_{n-2k}\}$

cubic graphs

$k \setminus n$	3	4	5	6	7	8
0	1	3	15	105	945	10395
1	1	3	15	105	945	10395
2	-	1	5	35	315	3465
3	-	-	-	7	63	693
4	-	-	-	-	-	99

$$\nu(n, k) = \frac{(2n-5)!!}{(2k-1)!!} \text{ for } 2k \leq n$$

- colour factors $C(\underline{1}, \bar{2}, \sigma) = (-1)^{k-1} \{2 | \sigma | 1\}$
- | | | |
|-----------------|---------------|------------------------------------|
| \underline{q} | \rightarrow | $\{q \mid T^b \otimes \Xi_{l-1}^b$ |
| \bar{q} | \rightarrow | $ q\}$ |
| g | \rightarrow | $\Xi_l^{a_g}$ |

- (BCJ relations (potentially) offer further reduction to $(n - 3)!(2k - 2)/k!$ amplitudes)

[H. Johannsson, A. Ochirov, 1507.00332]

- NOTE: colour-ordered is always $(n - 1)!$, so $(n - 2)!/k!$ is a massive improvement, in particular when several fermion pairs k are present

Melia primitive amplitudes

$k \setminus n$	3	4	5	6	7	8
0	1	2	6	24	120	720
1	1	2	6	24	120	720
2	-	1	3	12	60	360
3	-	-	-	4	20	120
4	-	-	-	-	-	30

$$\varkappa(n, k) = (n - 2)!/k!.$$

Continuous colour

- Do colour ordering (next slide)
- Do discrete summation over (adjoint) color indices, or use continuous „color polarisation vector“, start with dyadic decomposition:

[Draggiotis Kleiss Papadopoulos hep-ph/9807207, hep-ph/0202201]

- Quarks: Find colour vectors η with $\delta_{ij} = \int dz \eta_i(z) \eta_j(z)$, parametrisation: $\eta_i(z) \rightarrow \eta_i(\theta, \phi) = \begin{pmatrix} \cos \theta \\ \sin \theta \cos \phi \\ \sin \theta \sin \phi \end{pmatrix}$,
- $$\int dz \rightarrow \frac{N_c}{4\pi} \int_{-1}^1 d\cos \theta \int_0^{2\pi} d\phi$$

- For gluons, we reduce to quark case first:

$$\delta^{ab} = \text{Tr}(T^a T^b) = T_{ij}^a T_{kl}^b \delta_{jk} \delta_{il} = \int dz dz' \eta_i(z) T_{ij}^a \eta_j(z') \eta_k(z') T_{kl}^b \eta_l(z) = \int dz dz' \eta^a(z, z') \eta^{b\dagger}(z, z')$$

- Here, defined gluon colour polarisation vector $\eta^a(z, z') = \eta_i(z) T_{ij}^a \eta_j(z')$, represented by two real 3D spherical unit vectors
- similar to construction in [Draggiotis Kleiss Papadopoulos hep-ph/9807207, hep-ph/0202201], but four instead of five integration variables

- summed squared amplitude then given by $|\mathcal{A}(1, \dots, n)|^2 = \prod_i \int \frac{d\phi_i}{2\pi} dz_i dz'_i \left| \mathcal{A}_{z_1 z'_1 \dots z_n z'_n}^{\phi_1 \dots \phi_n}(p_1, \dots, p_n) \right|^2$

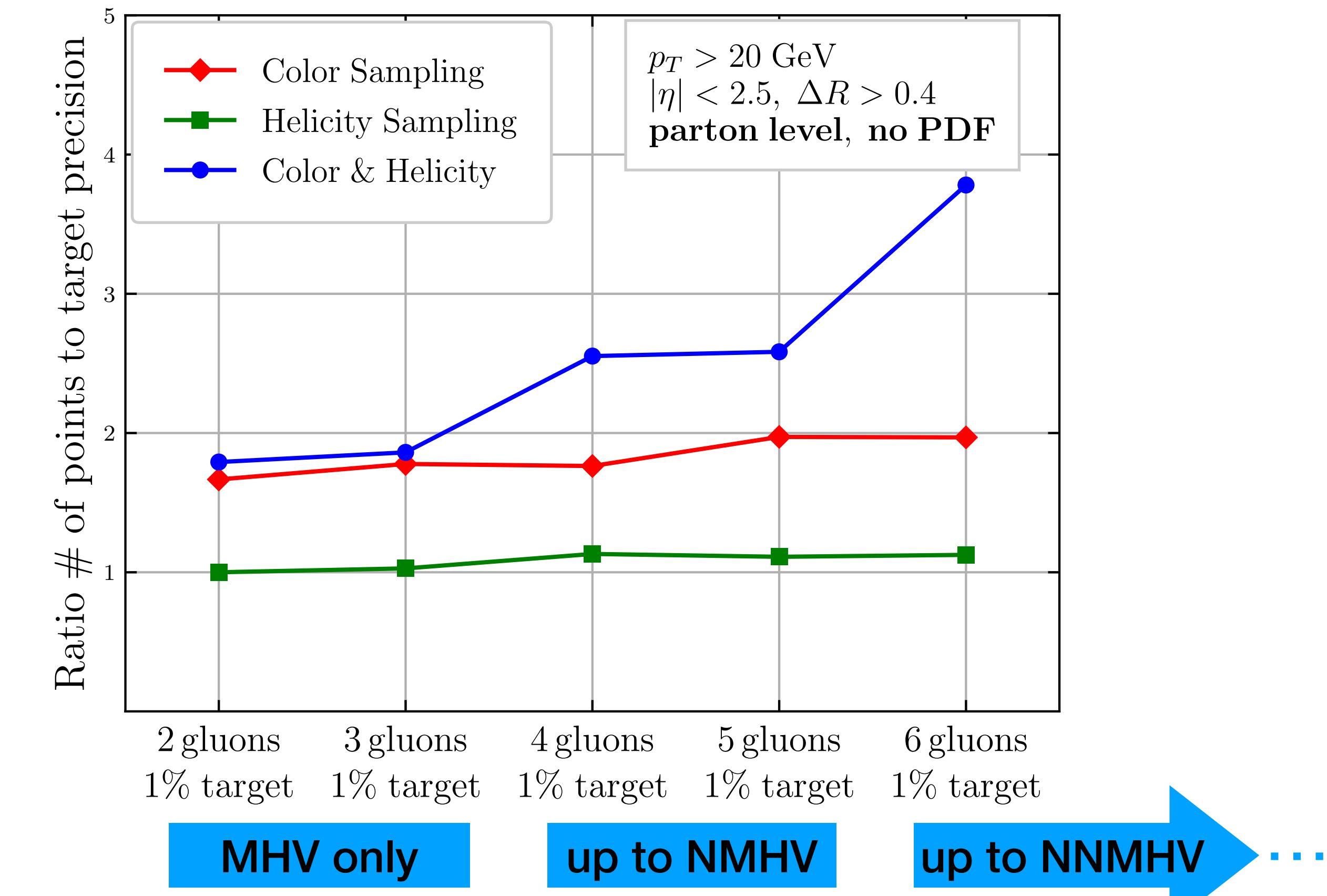
Summing vs. sampling

Can we afford just to sample?

- near-perfect for helicity due to selection of first two negative helicities according to invariants $P \sim \langle i|j\rangle^2$, compare MHV amplitudes

$$A_{g^n}^{\text{tree}}(\dots, i^-, \dots, j^-, \dots) = \frac{\langle ij\rangle^4}{\langle 12\rangle\langle 23\rangle\dots\langle(n-1)n\rangle\langle n1\rangle}$$

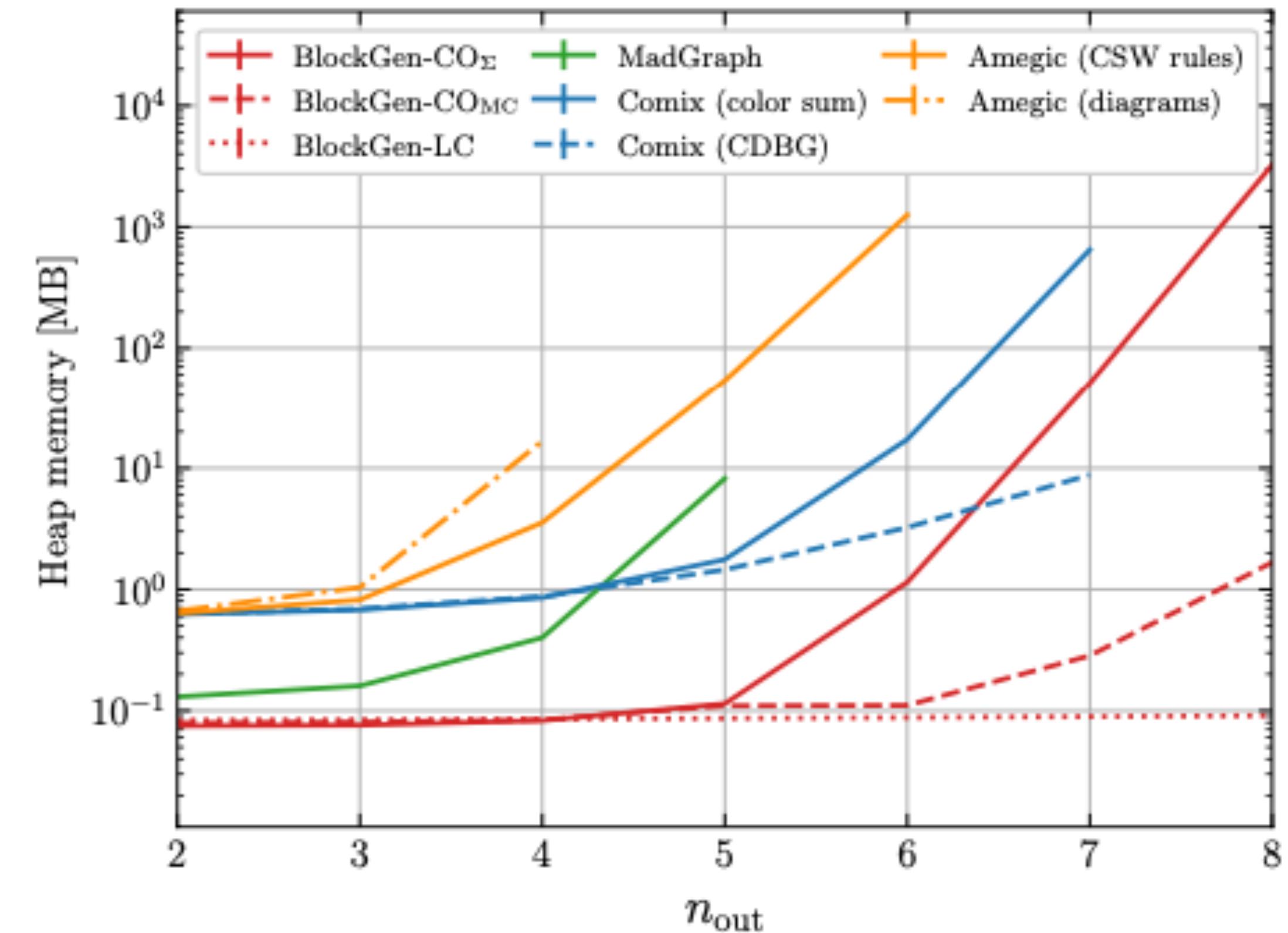
- combined sampling overhead increases with n in $N^n\text{MHV}$, likely due to ignoring correlation between colour and helicity configuration



CPU memory usage

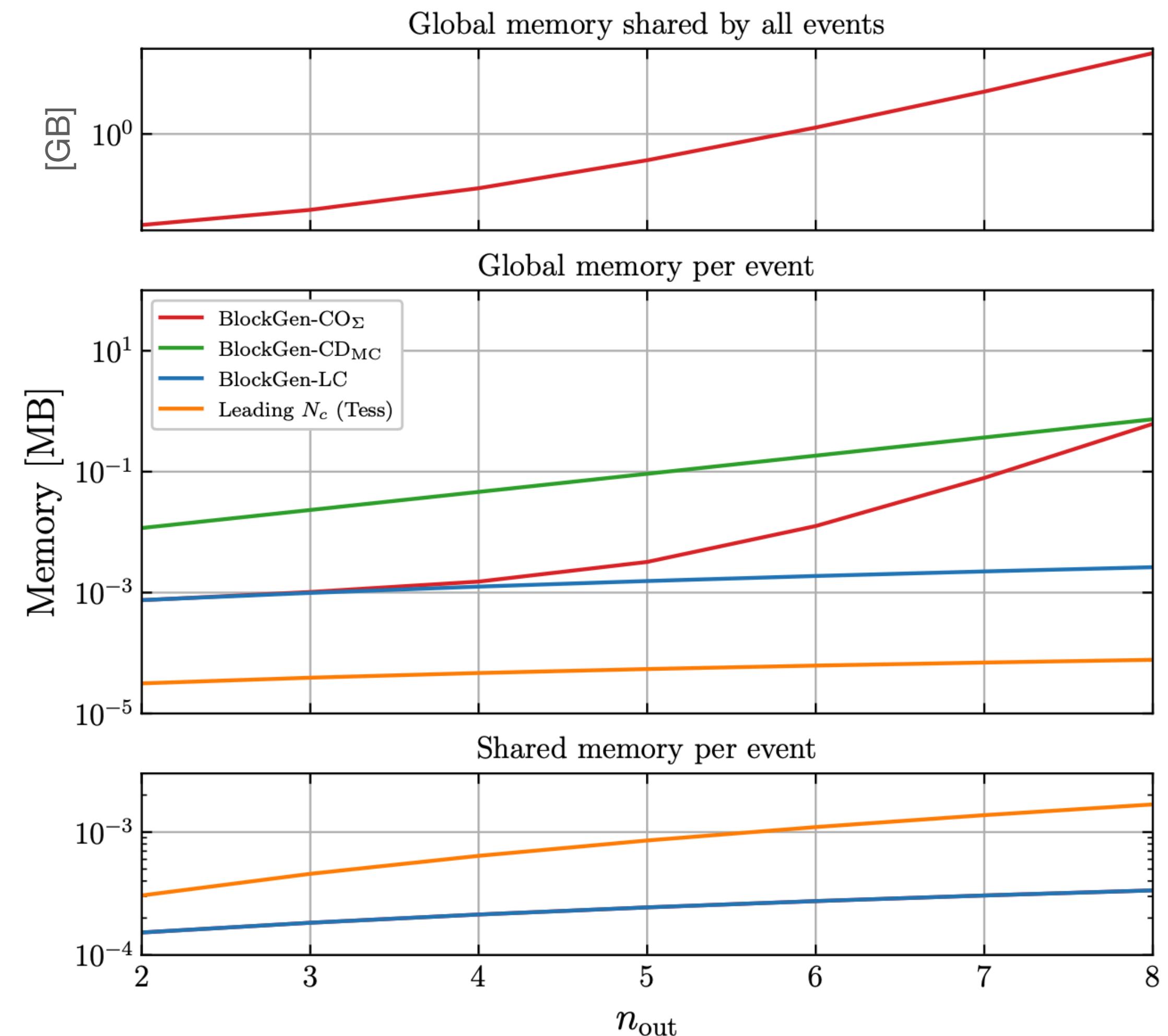
Single-threaded

-
- all helicity summed
 - MG 2.9.2 standalone-cpp with custom MC loop, Comix/Amegic as part of Sherpa 2.x
 - RAM, except MG: exe size
 - COMIX and BG smallest footprint
 - COMIX = 10-25 x BG, because it stores all contributing sub-currents for re-use when helicity sub-set is same



GPU Memory usage per thread & shared by threads

- BG-CO $_{\Sigma}$ stores its colour matrix in event-independent global mem $\sim \mathcal{O}((n - 2)!)^2$ → limits to $n_{\text{out}} \leq 8$, otherwise beyond 10 GB
- BG-CO $_{\Sigma}$ per-event mem: $\mathcal{O}((n - 2)!)$ color-ordered amplitudes
- BG-CD $_{\text{MC}}$ starts higher, storage governed by complex valued currents and tensors $\mathcal{O}(2^n)$
 - → BG-CO $_{\Sigma}$ more memory-lean for $n_{\text{out}} \lesssim 8$ and hence due to mem-boundness expected to perform better
- BG uses shared mem to store momenta sums used in recursion



QCD processes

- Colour summing branches off,
 - BUT BG colour-summed „stays in the game“
 - BG does really well for quark-only processes
 - ... but worse for gluonic processes

