DESY, Zeuthen, Germany

25 April 2019

Developments in Numerical Multi-loop Computations: Sector Decomposition & Quasi-Monte Carlo Integration

Stephen Jones

Borowka, Greiner, Heinrich, Jahn, Kerner, Luisoni,
Schlenk, Schubert, Scyboz, Vryonidou, Zirke

CPC (2019) (Accepted) [1811.11720]
CPC 222 (2018) 313 [1703.09692]
CPC 196 (2015) 470 [1502.06595]

Higher Order Computations

QCD Factorisation Formula

$$\mathrm{d}\sigma = \int \mathrm{d}x_a \mathrm{d}x_b f(x_a) f(x_b) \mathrm{d}\hat{\sigma}_{ab}(x_a, x_b) F_J + \mathcal{O}\left((\Lambda/Q)^m\right)$$

PDFs/ Input parameters

Hard Scattering Matrix Element

Non-perturbative effects ~ few %

With $\alpha_s \sim 0.1$

Typically: NLO ~ 10% correction, NNLO ~ 1% correction

However, there are important exceptions:

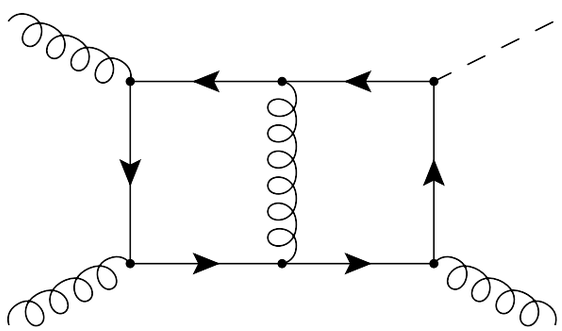
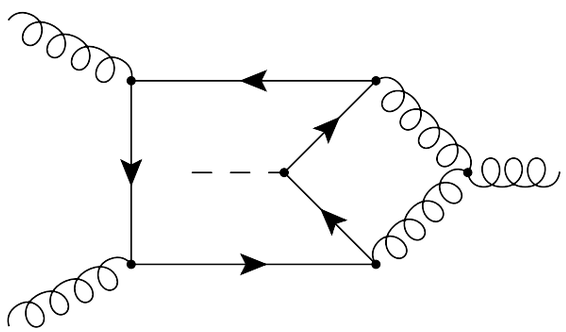
- Higgs Boson production (NLO ~100%, NNLO ~10%, N3LO ~ 2%)
- New partonic channels can open (e.g. di-boson production)

and

- Distributions can be modified substantially (even if σ_{tot} is stable)

Multi-loop Amplitudes

Consider a (multi-)loop amplitude:

$\mathcal{M}^{\mu\nu\rho} \sim$  $+$  $+\ldots$

Feynman diagrams → Projectors ($F = P_{\mu\nu\rho}\mathcal{M}^{\mu\nu\rho}$) → Integral Reduction (or other methods e.g. Numerical unitarity)

$$F = \sum_i c_i I_i$$

Large (#terms/degree) coefficient
Rational function of kinematics
Handled with specialist symbolic manipulation programs

Feynman integrals
Sometimes can be computed analytically ($\Longrightarrow$ fast)
We will discuss recent success in computing them numerically

Outline

Feynman Integrals & Sector Decomposition

- Feynman Parametrisation
- Sector Decomposition

Numerical Integration

- Quasi-Monte Carlo (QMC) Integration
- QMC & Variance Reduction
- Integration Package: qmc
- All-in-one Program: pySecDec

Physics Applications

- Higgs Boson Pair Production & Higgs Boson + Jet Production

Feynman Integrals

After applying Feynman rules we obtain integrals in momentum space:

$$I = \int \prod_{l=1}^{L} \left[\mathrm{d}^D k_l\right] \frac{1}{\prod_{j=1}^{N} P_j^{\nu_j}}$$

$L-$ loops

$N-$ propagators

$D-$ spacetime dimension

1) Feynman Parametrise and compute momentum integrals

$$I = (-1)^{N_\nu} \frac{\Gamma(N_\nu - LD/2)}{\prod_{j=1}^{N} \Gamma(\nu_j)} \int_0^\infty \prod_{j=1}^{N} \mathrm{d}x_j \; x_j^{\nu_j - 1} \delta(1 - \sum_{i=1}^{N} x_i) \frac{\mathcal{U}^{N_\nu - (L+1)D/2}(\vec{x})}{\mathcal{F}^{N_\nu - LD/2}(\vec{x}, s_{ij})}$$

Here $\mathcal{U}, \mathcal{F}$ are the 1st and 2nd Symanzik polynomials

Have exchanged L D-dim. momentum integrals for N param. integrals

Case:

A) Integral I is finite as $D \to 4$, may compute numerically

B) I is ill-defined as $D \to 4$, regulate $D = 4 - 2\epsilon$, disentangle singular behaviour and expand integral about $\epsilon \to 0$

Divergences

From the master formula, 3 possibilities for poles in ϵ to arise:

1. Overall $\Gamma(N_\nu - LD/2)$ diverges (single UV pole)
2. $\mathcal{U}(\vec{x})$ vanishes for some $x = 0$ and is in denominator (UV sub-divergences)
3. $\mathcal{F}(\vec{x}, s_{ij})$ vanishes on the boundary and is in denominator (IR divergences)

Outside the Euclidean region ($\forall s_{ij} < 0$) there is a further possibility:

4. $\mathcal{F}(\vec{x}, s_{ij})$ vanishes inside the integration region (May give: Landau singularity)

Can be handled by via contour deformation

Soper 99; Binoth, Guillet, Heinrich, Pilon, Schubert 05; Nagy, Soper 06; Anastasiou, Beerli, Daleo 07; Beerli 08; Borowka, Carter, Heinrich 12; Borowka 14;

If only condition 1 leads to a divergence the integral is **Quasi-finite**

Sector Decomposition

2) After integrating out δ we are faced with integrals of the form:

$$I_i = \int_0^1 \left(\prod_{j=1}^{N-1} \mathrm{d}x_j x_j^{\nu_j - 1} \right) \frac{\mathcal{U}_i(\vec{x})^{\mathrm{expo}\mathcal{U}(\epsilon)}}{\mathcal{F}_i(\vec{x}, s_{ij})^{\mathrm{expo}\mathcal{F}(\epsilon)}}$$

Powers depending on ϵ

Polynomials in F.P

Which may contain overlapping singularities which appear when several $x_j \to 0$ simultaneously (corresponding to UV/IR singularities)

Sector decomposition maps each integral to integrals of the form:

$$I_{ik} = \int_0^1 \left(\prod_{j=1}^{N-1} \mathrm{d}x_j x_j^{a_j - b_j \epsilon} \right) \frac{\mathcal{U}_{ik}(\vec{x})^{\mathrm{expo}\mathcal{U}(\epsilon)}}{\mathcal{F}_{ik}(\vec{x}, s_{ij})^{\mathrm{expo}\mathcal{F}(\epsilon)}}$$

Singularity structure can be read off

$\mathcal{U}_{ik}(\vec{x}) = 1 + u(\vec{x})$

$\mathcal{F}_{ik}(\vec{x}) = -s_0 + f(\vec{x})$

$u(\vec{x}), f(\vec{x})$ **have no constant term**

Hepp 66; Denner, Roth 96; Binoth, Heinrich 00

Sector Decomposition

One technique **Iterated Sector Decomposition** repeat:

Binoth, Heinrich 00

$$\int_0^1 \mathrm{d}x_1 \int_0^1 \mathrm{d}x_2 \frac{1}{(x_1+x_2)^{2+\epsilon}} \quad \leftarrow \textbf{Overlapping singularity for} \quad x_1, x_2 \to 0$$

$$= \int_0^1 \mathrm{d}x_1 \int_0^1 \mathrm{d}x_2 \frac{1}{(x_1+x_2)^{2+\epsilon}} (\theta(x_1 - x_2) + \theta(x_2 - x_1))$$

$$= \int_0^1 \mathrm{d}x_1 \int_0^{x_1} \mathrm{d}x_2 \frac{1}{(x_1+x_2)^{2+\epsilon}} + \int_0^1 \mathrm{d}x_2 \int_0^{x_2} \mathrm{d}x_1 \frac{1}{(x_1+x_2)^{2+\epsilon}}$$

$$= \int_0^1 \mathrm{d}x_1 \int_0^1 \mathrm{d}t_2 \frac{x_1}{(x_1+x_1 t_2)^{2+\epsilon}} + \int_0^1 \mathrm{d}x_2 \int_0^1 \mathrm{d}t_1 \frac{x_2}{(x_2 t_1+x_2)^{2+\epsilon}}$$

$$= \int_0^1 \mathrm{d}x_1 \int_0^1 \mathrm{d}t_2 \frac{x_1^{-1-\epsilon}}{(1+t_2)^{2+\epsilon}} + \int_0^1 \mathrm{d}x_2 \int_0^1 \mathrm{d}t_1 \frac{x_2^{-1-\epsilon}}{(t_1+1)^{2+\epsilon}} \quad \leftarrow \textbf{Singularities factorised}$$

If this procedure terminates depends on order of decomposition steps

An alternative strategy **Geometric Sector Decomposition** always terminates; both strategies are implemented in pySecDec.

Kaneko, Ueda 10; See also: Bogner, Weinzierl 08; Smirnov, Tentyukov 09

Sector Decomposition (II)

3) Expand in ϵ (simple case $a = -1$ "Logarithmic Divergence"):

$$\int_0^1 \mathrm{d}x x^{-1-b\epsilon} g(x) = \frac{g(0)}{-b\epsilon} + \int_0^1 \mathrm{d}x x^{-b\epsilon} \left[\frac{g(x) - g(0)}{x}\right]$$

Poles ← $\frac{g(0)}{-b\epsilon}$

Finite ← $\left[\frac{g(x) - g(0)}{x}\right]$

`subtraction' of $g(0)$

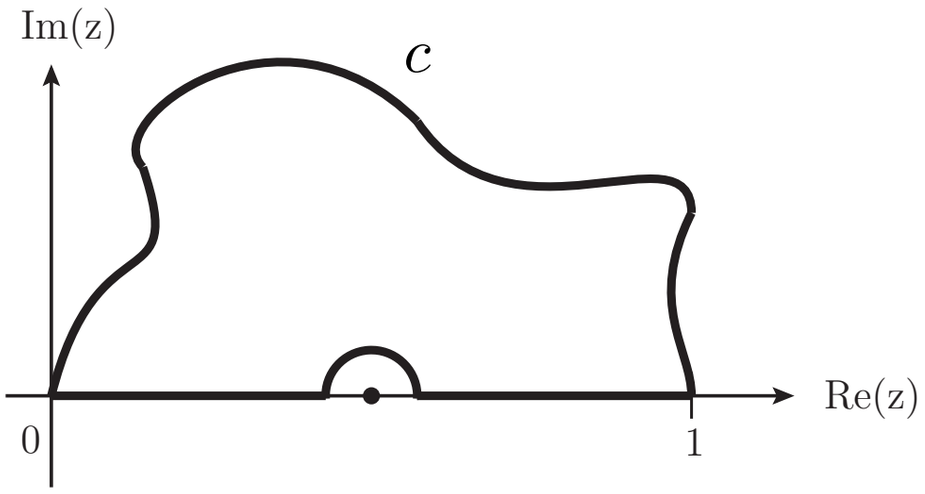
By Definition: $g(0) \neq 0, g(0)$ finite

4) Numerically integrate

Key Point: Sector Decomposed integrals can be expanded in ϵ and numerically integrated

Contour Deformation

(Physical region) $\mathcal{F}$ may vanish on a hypersurface within the integration domain, use Cauchy's theorem to avoid unphysical poles on the real axis



$$\oint_c \mathrm{d}z I(z) = \int_0^1 \mathrm{d}x I(x) + \int_1^0 \mathrm{d}z I(z) = 0$$

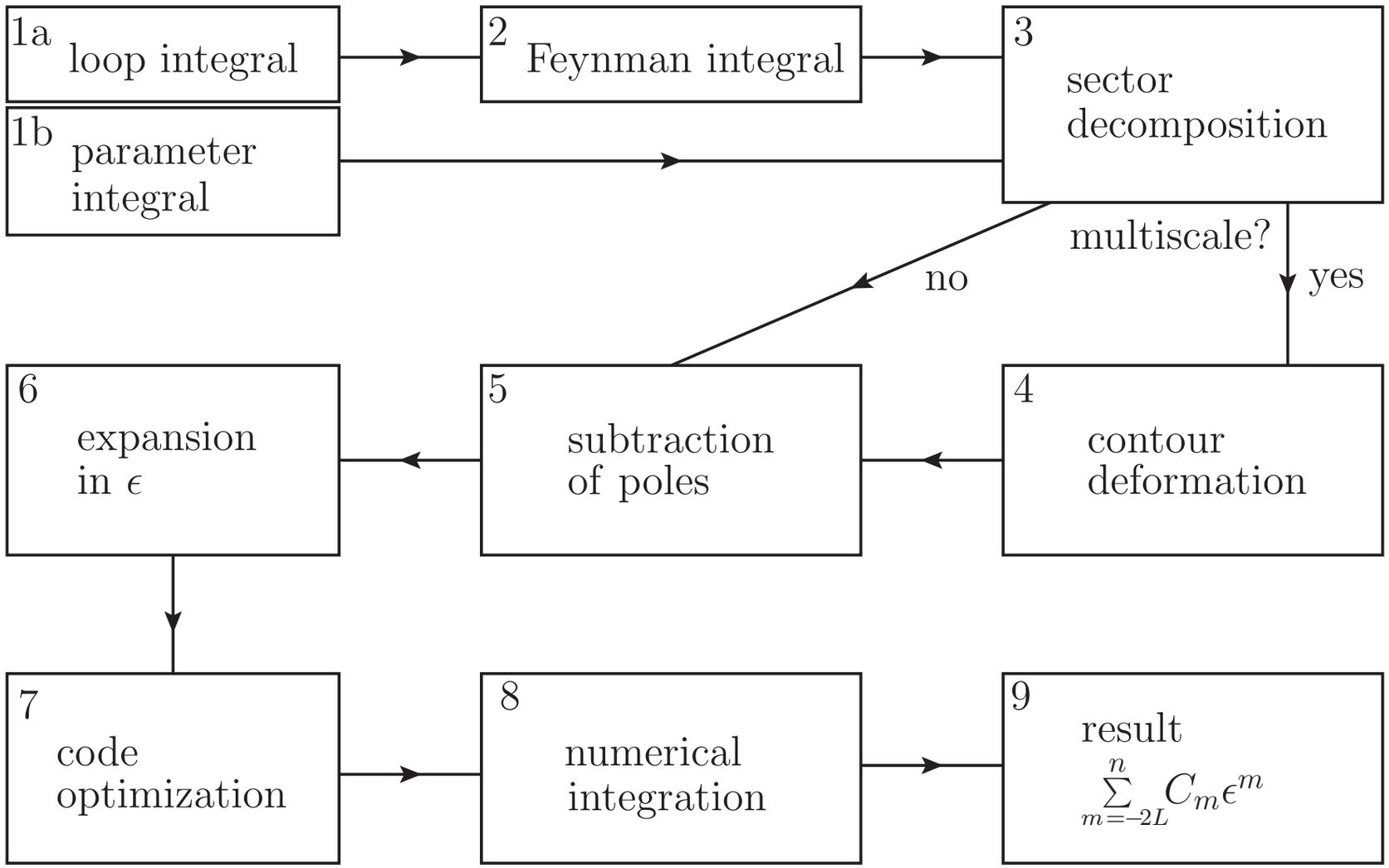
Must obey Feynman (causal) prescription: $\mathcal{F}(\vec{x}) \to \mathcal{F}(\vec{x}) - i\delta$

Deformation: $z_k(\vec{x}) = x_k - i\tau_k(\mathbf{x}), \quad \tau_k(\vec{x}) = \lambda_k x_k(1-x_k)\dfrac{\partial \mathcal{F}(\vec{x})}{\partial x_k}$

Expanding $\mathcal{F}(\vec{z})$ about $\vec{z} = \vec{x}$ (i.e. sufficiently small λ_k):

$$\begin{aligned}\mathcal{F}(\vec{z}) =& \mathcal{F}(\vec{x}) - i\sum_k \lambda_k x_k(1-x_k)\left(\frac{\partial \mathcal{F}(\vec{x})}{\partial x_k}\right)^2 - \frac{1}{2}\sum_j\sum_k \tau_j\tau_k\left(\frac{\partial^2 \mathcal{F}(\vec{x})}{\partial x_j x_k}\right) \\ &+ \frac{i}{6}\sum_j\sum_k\sum_l \tau_j\tau_k\tau_l\left(\frac{\partial^3 \mathcal{F}(\vec{x})}{\partial x_j x_k x_l}\right) + \ldots\end{aligned}$$

pySecDec: Schematic Workflow



Numerical Integration

Numerical Integration

$$I[f] \equiv \int_{[0,1]^d} \mathrm{d}\mathbf{x}\ f(\mathbf{x}) \quad \approx \quad Q[f] = \frac{1}{N}\sum_{i=1}^{N} w_i\ f(\mathbf{x}_i)$$

Goal: select points to minimise integration error

$$\varepsilon \equiv |I[f] - Q[f]|$$

Monte Carlo:

Randomly select sampling points

$$\varepsilon \approx \mathrm{Var}[f]/\sqrt{N}, \quad \varepsilon \sim \mathcal{O}(N^{-1/2})$$

Improves slowly with N

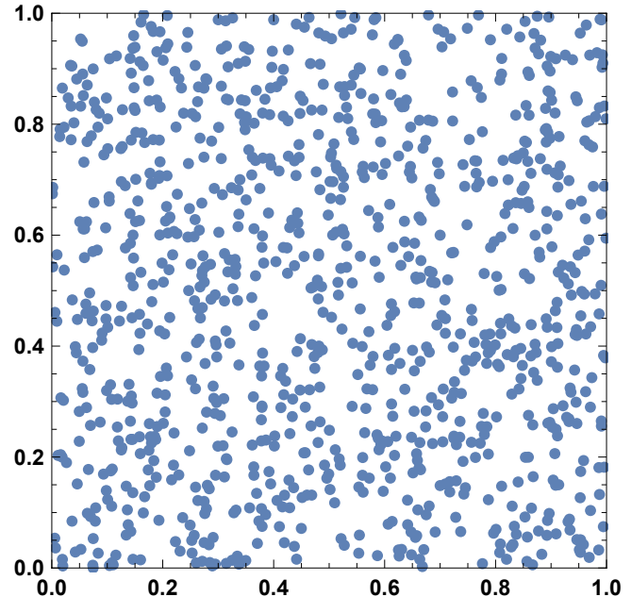
Quasi-Monte Carlo

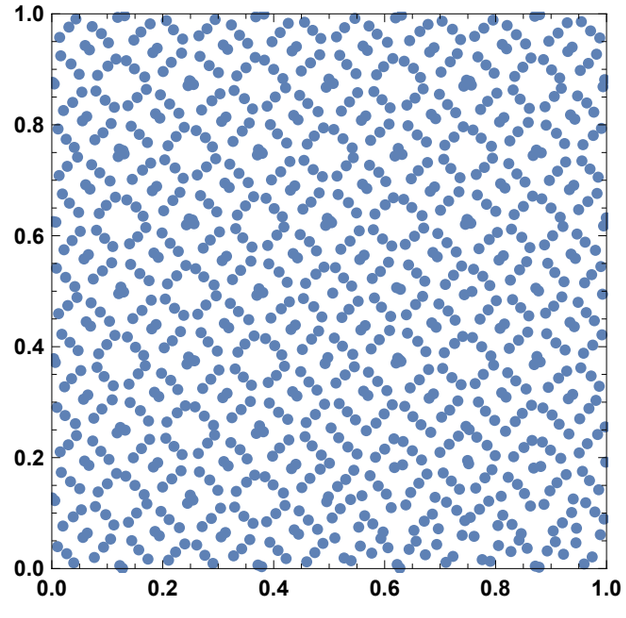
Select points with low discrepancy D_N

$$\varepsilon \leq D_N \cdot \mathrm{Var}[f], \quad \varepsilon \sim \mathcal{O}(\log^d(N)/N)$$

Poor performance for large d

Both methods implemented in Cuba Hahn 04; Hahn14





Quasi-Monte Carlo (Rank 1 Lattices)

Quasi-Monte Carlo (QMC) in a Weighted Function Space

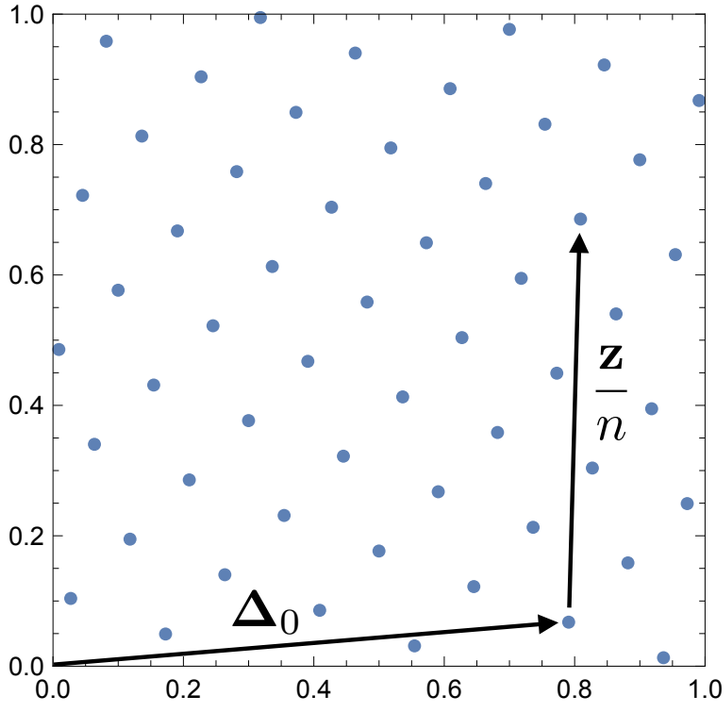
First applications to loop integrals, see:

Li, Wang, Yan, Zhao 15; de Doncker, Almulihi, Yuasa 17, 18; de Doncker, Almulihi 17; Kato, de Doncker, Ishikawa, Yuasa 18

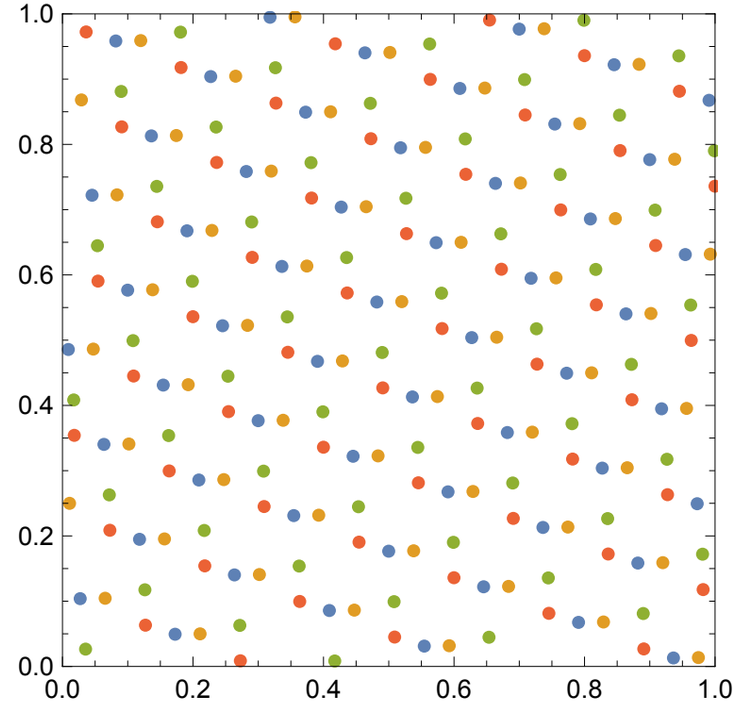
$\varepsilon \leq e_{\boldsymbol{\gamma}} \cdot ||f||_{\boldsymbol{\gamma}}, \quad \varepsilon \sim \mathcal{O}(N^{-1})$ or better

$$I[f] \approx \bar{Q}_{n,m}[f] \equiv \frac{1}{m} \sum_{k=0}^{m-1} Q_n^{(k)}[f], \quad Q_n^{(k)}[f] \equiv \frac{1}{n} \sum_{i=0}^{n-1} f\left(\left\{ \frac{i\mathbf{z}}{n} + \boldsymbol{\Delta}_k \right\} \right)$$

- $\mathbf{z}$ - Generating vec.
- $\boldsymbol{\Delta}_k$ - Random shift vec.
- $\{\}$ - Fractional part
- n - # Lattice points
- m - # Random shifts



1 Shift



4 Shifts

Unbiased error estimate computed using (10-50) random shifts

Weighted Function Spaces

Assign weights $\gamma_{\mathfrak{u}}$ to each subset of dimensions $\mathfrak{u} \subseteq \{1, \ldots, d\}$ Review: Dick, Kuo, Sloan 13

Sobolev Space

Functions with square integrable first derivatives

Norm $$||f||_{\boldsymbol{\gamma}}^2 = \sum_{\mathfrak{u} \subseteq \{1,\ldots,d\}} \frac{1}{\gamma_{\mathfrak{u}}} \int_{[0,1]^{|\mathfrak{u}|}} \left(\int_{[0,1]^{d-|\mathfrak{u}|}} \frac{\partial^{|\mathfrak{u}|} f(\mathbf{x})}{\partial \mathbf{x}_{\mathfrak{u}}} \mathrm{d}\mathbf{x}_{-\mathfrak{u}} \right)^2 \mathrm{d}\mathbf{x}_{\mathfrak{u}}$$

Worst-case error $$e_{\boldsymbol{\gamma}}^2 \leq \left(\frac{1}{\psi(n)} \sum_{\emptyset \neq \mathfrak{u} \subseteq \{1,\ldots,d\}} \gamma_{\mathfrak{u}}^{\lambda} \left(\frac{2\zeta(2\lambda)}{(2\pi^2)^{\lambda}} \right)^{|\mathfrak{u}|} \right)^{\frac{1}{\lambda}}$$

$\forall \lambda \in (1/2, 1]$

$\varepsilon \sim \mathcal{O}(n^{-1})$

Korobov Space

Periodic functions which are α times differentiable in each variable

$$||f||_{\boldsymbol{\gamma}}^2 = \sum_{\mathbf{h} \in \mathbb{Z}^d} \frac{\prod_{j \in \mathfrak{u}(\mathbf{h})} |h_j|^{2\alpha}}{\gamma_{\mathfrak{u}(\mathbf{h})}} |\hat{f}(\mathbf{h})|^2$$

↑ Fourier Coefficient

$$e_{\boldsymbol{\gamma}}^2 \leq \left(\frac{1}{\psi(n)} \sum_{\emptyset \neq \mathfrak{u} \subseteq \{1,\ldots,d\}} \gamma_{\mathfrak{u}}^{\lambda} \left(2\zeta(2\alpha\lambda)\right)^{|\mathfrak{u}|} \right)^{\frac{1}{\lambda}}$$

$\forall \lambda \in (1/(2\alpha), 1]$, smoothness α

$\varepsilon \sim \mathcal{O}(n^{-\alpha})$

Generating vector $\mathbf{z}$ precomputed for a **fixed** number of lattice points, chosen to minimise the worst-case error, we use component-by-component (CBC) construction Nuyens 07

In our public code, we distribute lattice rules generated using product weights: $\gamma_{\mathfrak{u}} = \prod_{i \in \mathfrak{u}} \gamma_i, \ \gamma_i = 1/d$ produced for a Korobov space with $\alpha = 2$

Periodizing Transforms

Lattice rules work especially well for continuous, smooth and periodic functions
Functions can be periodized by a suitable change of variables: $\mathbf{x} = \phi(\mathbf{u})$

$$I[f] \equiv \int_{[0,1]^d} \mathrm{d}\mathbf{x}\ f(\mathbf{x}) = \int_{[0,1]^d} \mathrm{d}\mathbf{u}\ \omega_d(\mathbf{u}) f(\phi(\mathbf{u}))$$

$$\phi(\mathbf{u}) = (\phi(u_1), \ldots, \phi(u_d)), \qquad \omega_d(\mathbf{u}) = \prod_{j=1}^{d} \omega(u_j) \qquad \text{and} \qquad \omega(u) = \phi'(u)$$

Korobov transform: $\omega(u) = 6u(1-u), \quad \phi(u) = 3u^2 - 2u^3$

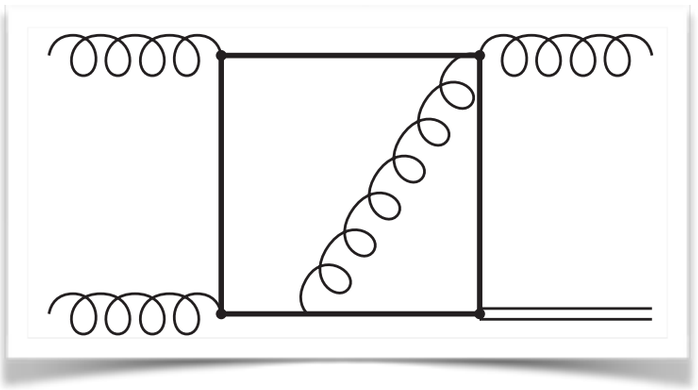
Sidi transform: $\omega(u) = \pi/2 \sin(\pi u), \quad \phi(u) = 1/2(1 - \cos \pi t)$

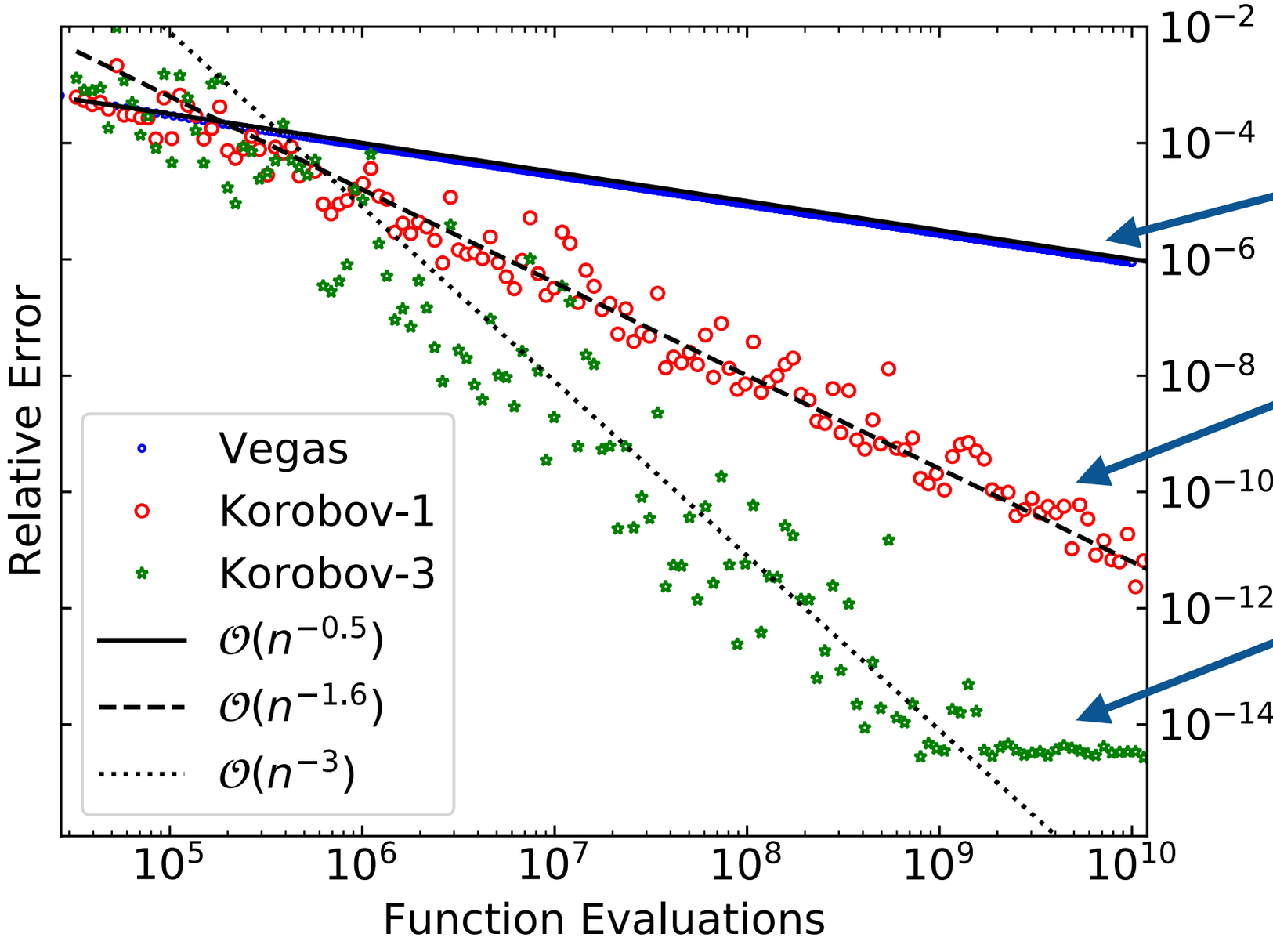
Baker transform: $\phi(u) = 1 - |2u - 1|$

Scaling

Example:

Sector Decomposed HJ Integral





Monte Carlo scaling

Better than ``guaranteed" n^{-1} scaling

Limited by machine precision (double)

Variance Reduction

Can improve performance of numerical integrator by flattening integrand via a variable transformation $y = p(x)$

$$I = \int_0^1 \mathrm{d}y \; f(y) = \int_0^1 \mathrm{d}x \; p'(x) f(p(x)) \qquad \text{s.t.} \qquad p'(x) \propto |f(p(x))|^{-1}$$

VEGAS:

Assume integrand separable $f(\mathbf{x}) = g(x_1)g(x_2)\cdots g(x_d)$

Iteratively approximate $p(x)$ with piecewise linear function

But: Spoils smoothness of integrand (bad for QMC)

Alternative:

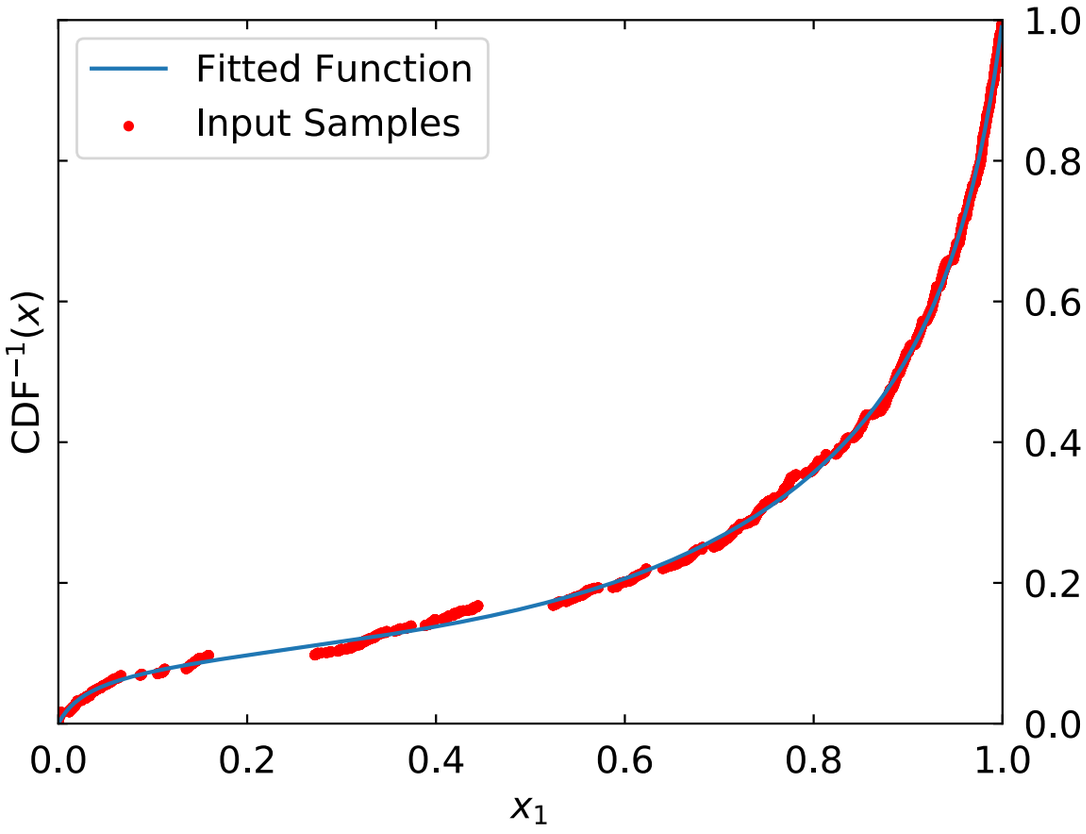
Choose a smooth function for $p(x)$, for example:

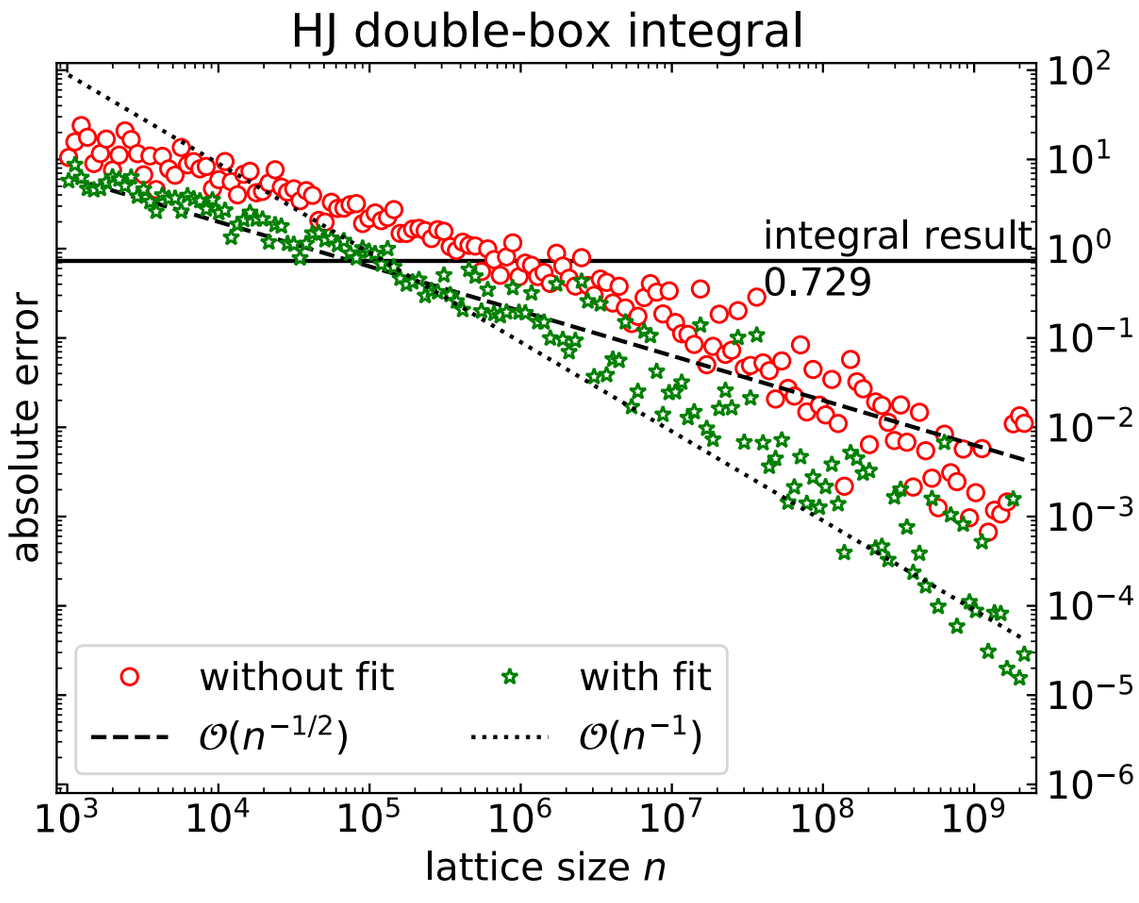
$$p(x) = a_2 \cdot x \, \frac{a_0 - 1}{a_0 - x} + a_3 \cdot x \, \frac{a_1 - 1}{a_1 - x} + a_4 \cdot x + a_5 \cdot x^2 + \left(1 - \sum_{i=2}^{5} a_i\right) \cdot x^3$$

Fit parameters a_i to inverse cumulative distribution function (CDF)

Variance Reduction (II)

1) Pre-sample integrand
2) In each dimension: fit inverse CDF
3) Use fit as variable transformation, obtain flatter integrand without spoiling smoothness





Example: HJ Integral

Small n: ~ 3x improvement
Large n: ~ 10x improvement

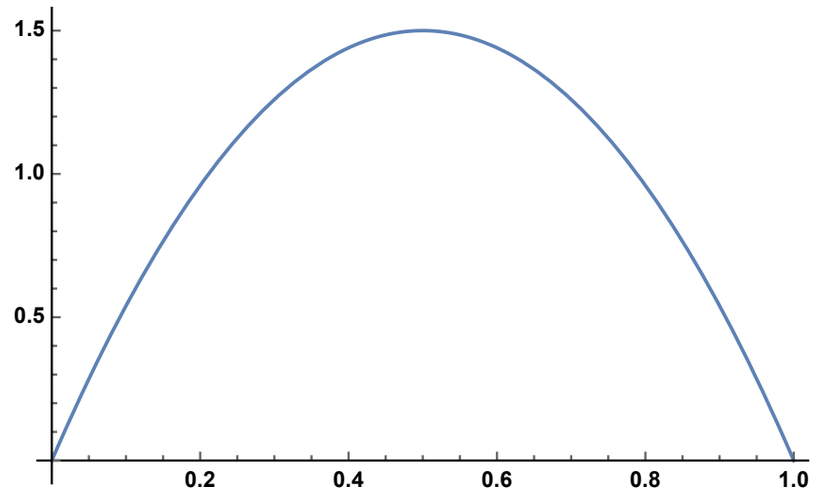
QMC Scaling preserved

Curse of Dimensionality

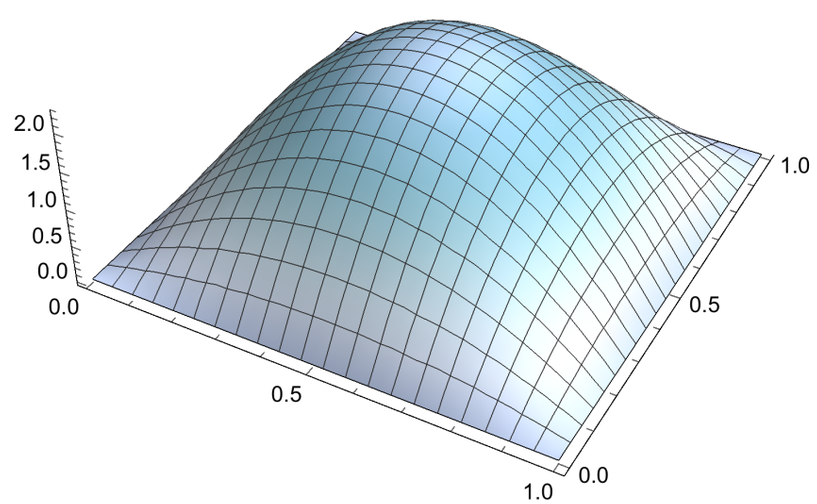
Even if the worst-case error of our lattice rule is independent of d, can still have problems integrating in a higher number of dimensions ($d \gtrsim 8$) due to the periodizing transform

Example: $f(\mathbf{x}) = 1$

$d = 1$
$F(u_1) = \omega(u_1)$



$d = 2$
$F(u_1, u_2) = \omega(u_1)\omega(u_2)$



Korobov & Sidi transforms can increase the norm of the function, the increase can be exponentially large for large d

Kuo, Sloan, Woźniakowski 07

In this case, using the Baker transform can be a good choice

qmc: Design

QMC integration is embarrassingly parallel and simple to implement

We aimed to write a public package that is:

1) Fast
2) Easy to use
3) Supports multiple CPUs/GPUs (on a single machine)
4) Flexible and reasonably easy to develop and extend

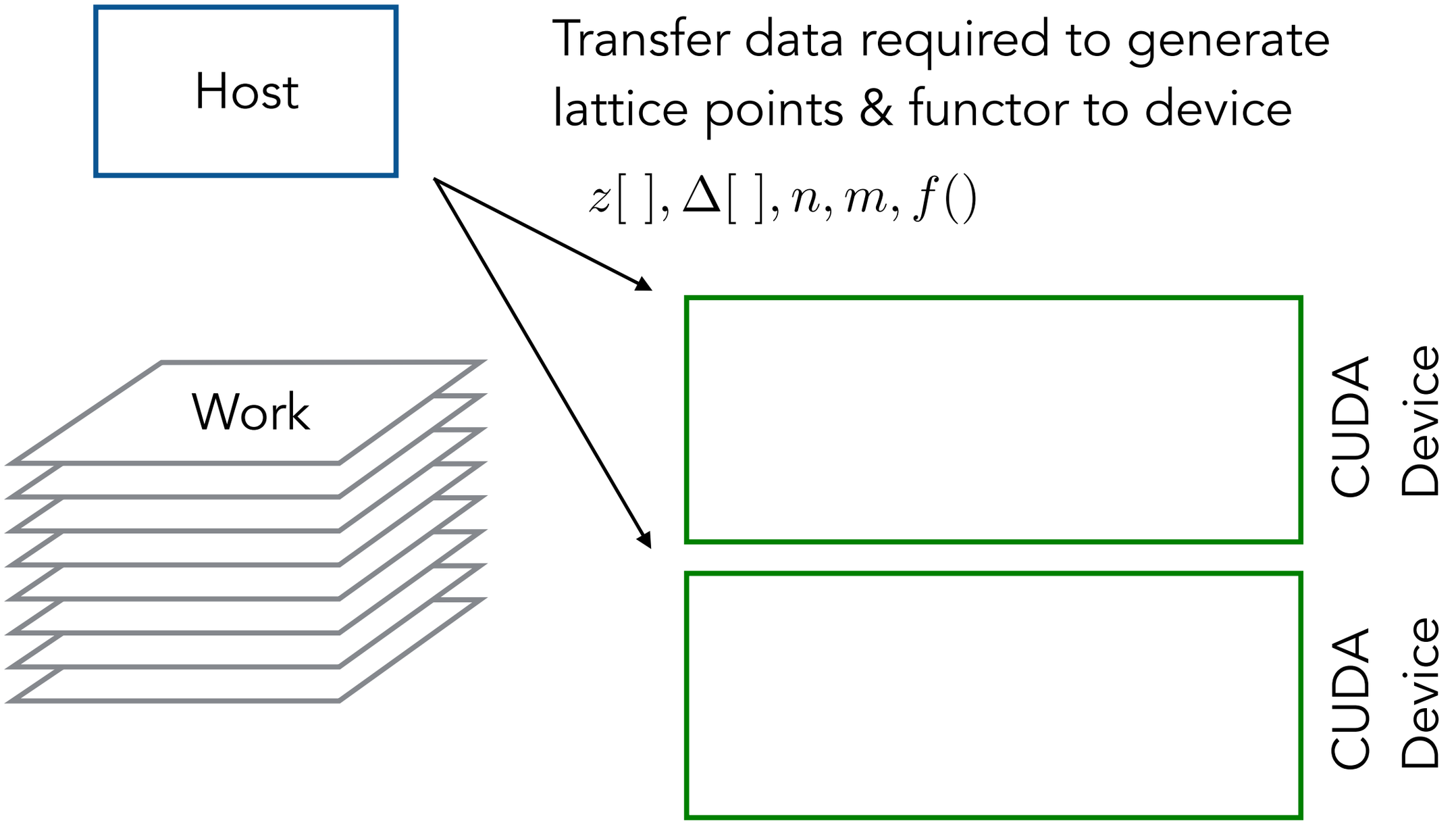
Language Choices:

CUDA (fairly developed language, decent compiler)

c++11 (object orientated, std::thread, templates, CUDA support)

qmc: Implementation

1) Devices are initialised

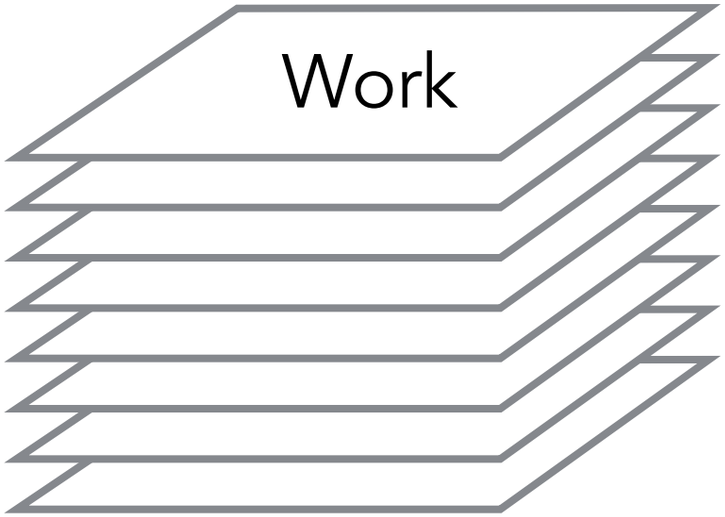


qmc: Implementation

2) (Loop) Work is requested from central work queue by each device

Host

Only index of point(s) to calculate needs to be transferred
Partial results are accumulated by each thread on each device



i

i'

$f\left(\left\{\frac{i\mathbf{z}}{n}+\mathbf{\Delta}_0\right\}\right)$	$f\left(\left\{\frac{i\mathbf{z}}{n}+\mathbf{\Delta}_1\right\}\right)$	…
$f\left(\left\{\frac{(i+1)\mathbf{z}}{n}+\mathbf{\Delta}_0\right\}\right)$	$f\left(\left\{\frac{(i+1)\mathbf{z}}{n}+\mathbf{\Delta}_1\right\}\right)$	…
…	…	

Thread 0
Thread 1
…

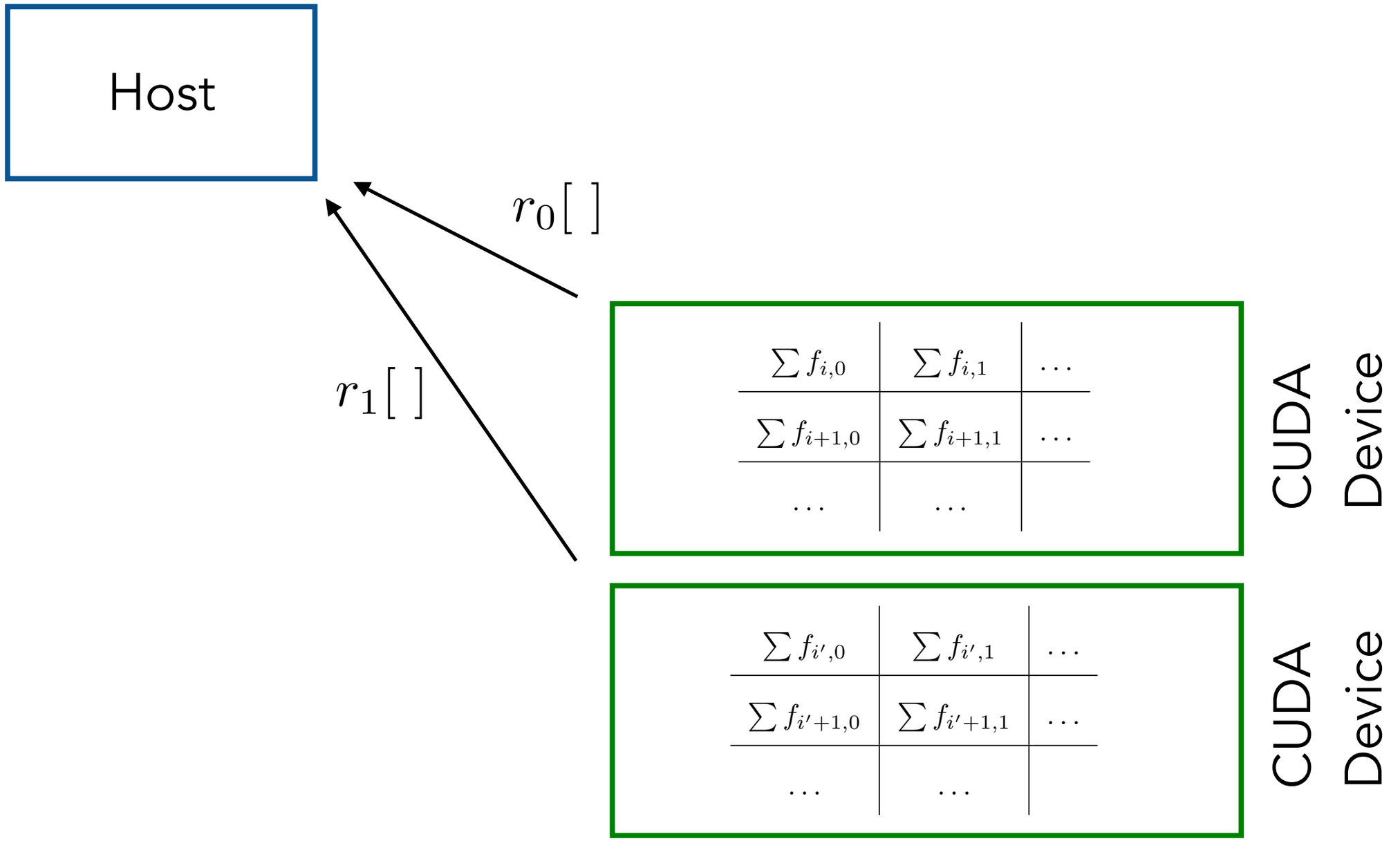
$f\left(\left\{\frac{i'\mathbf{z}}{n}+\mathbf{\Delta}_0\right\}\right)$	$f\left(\left\{\frac{i'\mathbf{z}}{n}+\mathbf{\Delta}_1\right\}\right)$	…
$f\left(\left\{\frac{(i'+1)\mathbf{z}}{n}+\mathbf{\Delta}_0\right\}\right)$	$f\left(\left\{\frac{(i'+1)\mathbf{z}}{n}+\mathbf{\Delta}_1\right\}\right)$	…
…	…	

Thread 0
Thread 1
…

Work is actually just an atomically locked counter

qmc: Implementation

3) Once all work is completed, partial results are transferred back to host for final reduction



GPU Usage for Calculating Feynman Integrals

FIESTA4 (Cuba with GPUs + Sector Decomposition) Smirnov 16

- Widely used in community, e.g. 4-loop $\overline{\text{MS}}$-on-shell quark mass relation in QCD Marquard, Smirnov, Smirnov, Steinhauser, Wellmann 16

Lattice Rules + Sector Decomposition

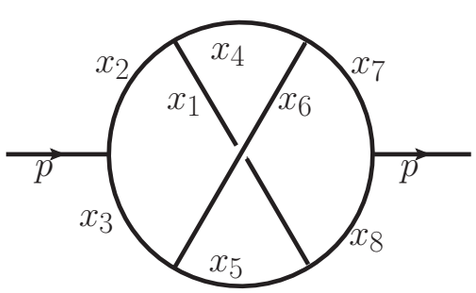
- 2-loop QCD Higgs Boson pair production
- 2-loop QCD Higgs Boson + Jet production

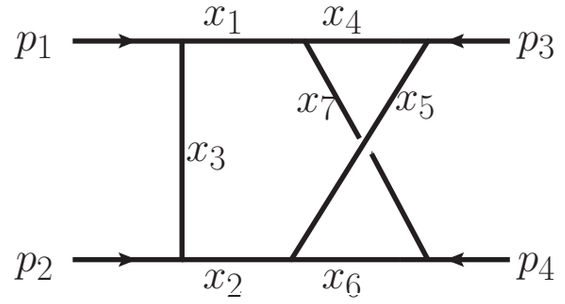
Li, Wang, Yan, Zhao 15;
Borowka, Greiner, Heinrich, SPJ, Kerner, Schlenk, Schubert, Zirke 16;
SPJ, Kerner, Luisoni 18

Lattice Rules + $\epsilon \to 0$ Extrapolation

- 2-3 loop box/self-energy diagrams

de Doncker, Almulihi, Yuasa 17, 18





Monte Carlo Integration of Hepp Sectors Volkov 18

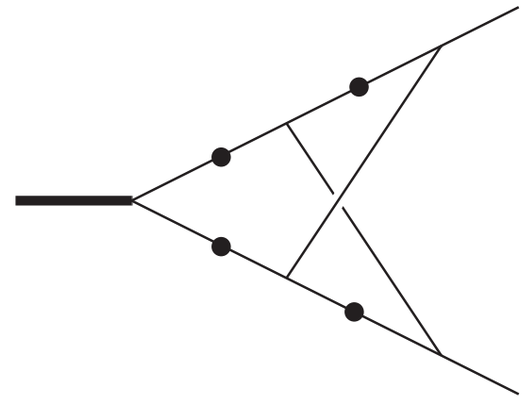
- 2-4 loop QED contributions to the electron magnetic moment
- 5-6 loop "ladder" form factor integrals

qmc: Usage

Let's compute a finite Feynman integral (à la de Doncker, Almulihi, Yuasa 17)

```cpp
#include <iostream>
#include "qmc.hpp"
struct formfactor2L_t {
    const unsigned long long int number_of_integration_variables = 5;
#ifdef __CUDACC__
    __host__ __device__
#endif
    double operator()(const double arg[]) const
    {
        // Simplex to cube transformation
        double x0  = arg[0];
        double x1  = (1.-x0)*arg[1];
        double x2  = (1.-x0-x1)*arg[2];
        double x3  = (1.-x0-x1-x2)*arg[3];
        double x4  = (1.-x0-x1-x2-x3)*arg[4];
        double x5  = (1.-x0-x1-x2-x3-x4);
        double wgt =
        (1.-x0)*
        (1.-x0-x1)*
        (1.-x0-x1-x2)*
        (1.-x0-x1-x2-x3);
        if(wgt <= 0) return 0;
        // Integrand
        double u=x2*(x3+x4)+x1*(x2+x3+x4)+(x2+x3+x4)*x5+x0*(x1+x3+x4+x5);
        double f=x1*x2*x4+x0*x2*(x1+x3+x4)+x0*(x2+x3)*x5;
        double n=x0*x1*x2*x3;
        double d = f*f*u*u;
        return wgt*n/d;
    }
} formfactor2L;
```

$(8-2\epsilon)$



Note: can compile this code with or without CUDA

```cpp
int main() {
    integrators::Qmc<double,double,5,integrators::transforms::Korobov<3>::type> integrator;
    integrator.minn = 100000000; // (optional) lattice size
    integrators::result<double> result = integrator.integrate(formfactor2L);
    std::cout << "integral = " << result.integral << std::endl;
    std::cout << "error    = " << result.error    << std::endl;
    return 0;
}
```

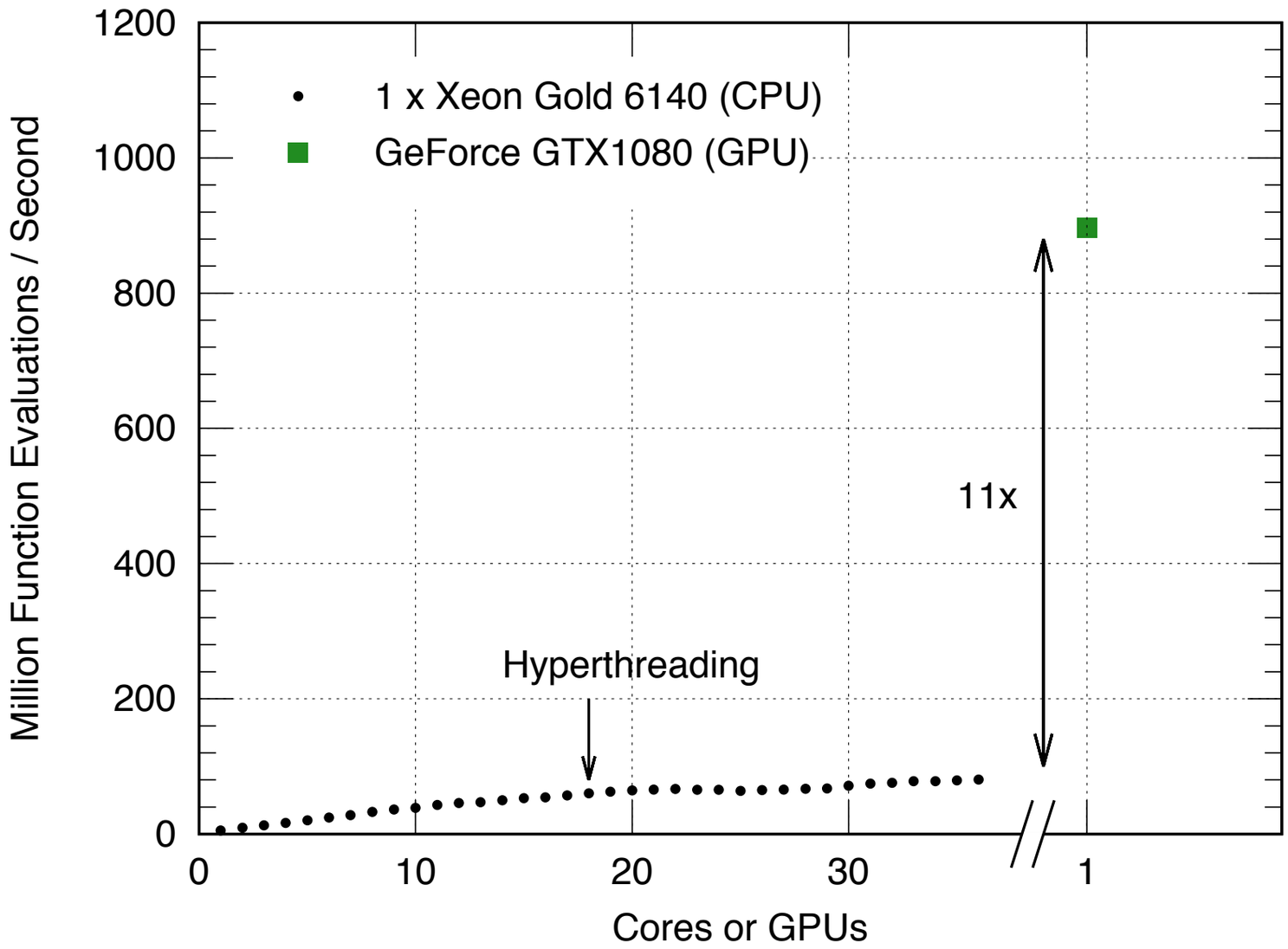
```
$ nvcc -O3 -arch=sm_70 -std=c++11 -x cu -I../src 102_ff2_demo.cpp -o 102_ff2_demo.out -lgsl -lgslcblas && ./102_ff2_demo.out
integral = 0.27621
error    = 4.49751e-07
```

(Agrees with analytic result)

qmc: Performance

Accuracy limited by number of function evaluations

Can accelerate this using Graphics Processing Units (GPUs)



Device	M Func. Evals/s	Speedup
Xeon 6140	80.6	-
GTX1080	897	11x

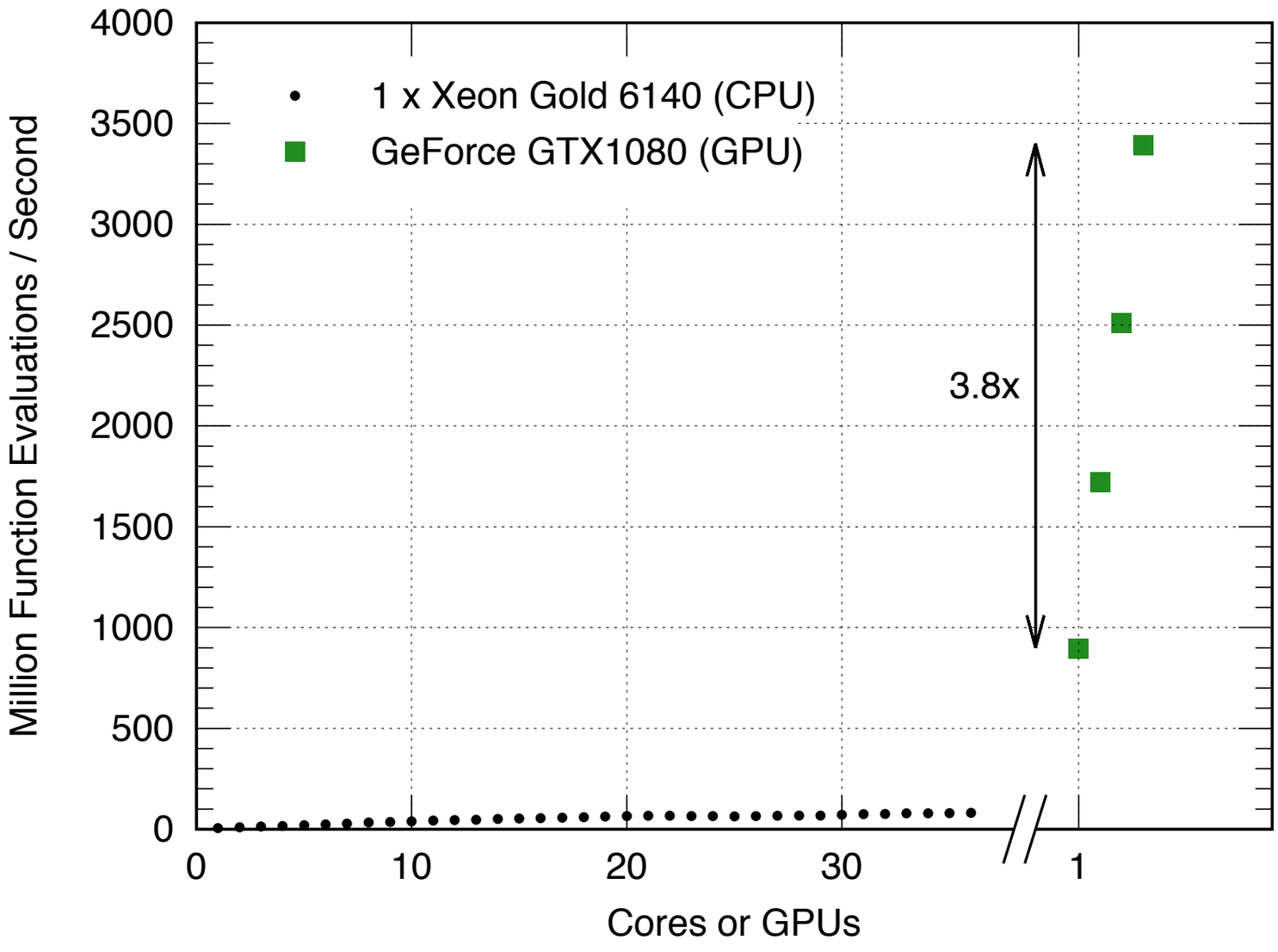
Note: Performance gain highly dependent on integrand & hardware!

Still room for further optimisations (both for CPU and GPU)

qmc: Performance

Accuracy limited by number of function evaluations

Can accelerate this using Graphics Processing Units (GPUs)



Device	M Func. Evals/s	Speedup
Xeon 6140	80.6	-
GTX1080	897	11x

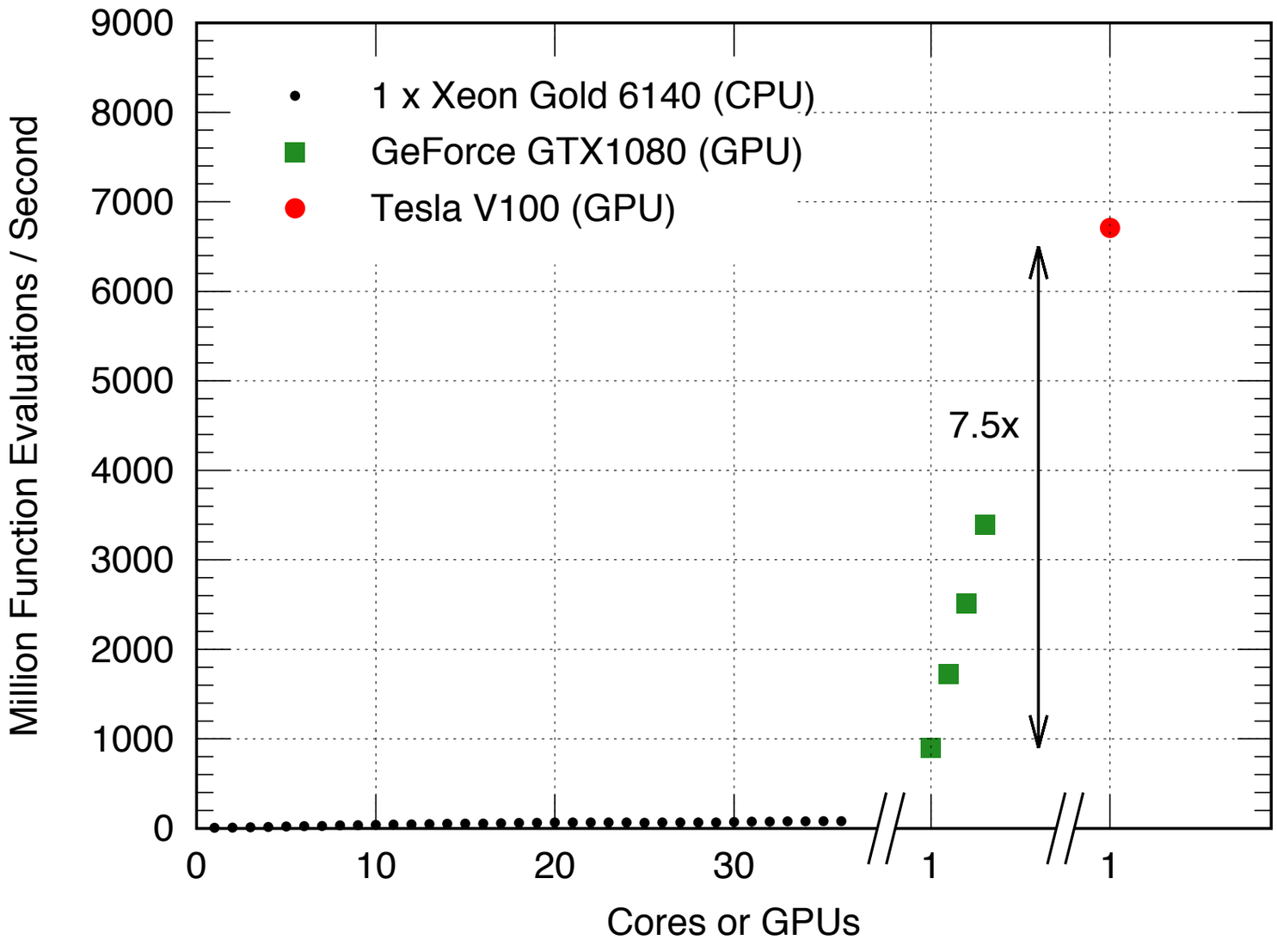
Note: Performance gain highly dependent on integrand & hardware!

Still room for further optimisations (both for CPU and GPU)

qmc: Performance

Accuracy limited by number of function evaluations

Can accelerate this using Graphics Processing Units (GPUs)



Device	M Func. Evals/s	Speedup
Xeon 6140	80.6	-
GTX1080	897	11x
Tesla V100	6710	83x

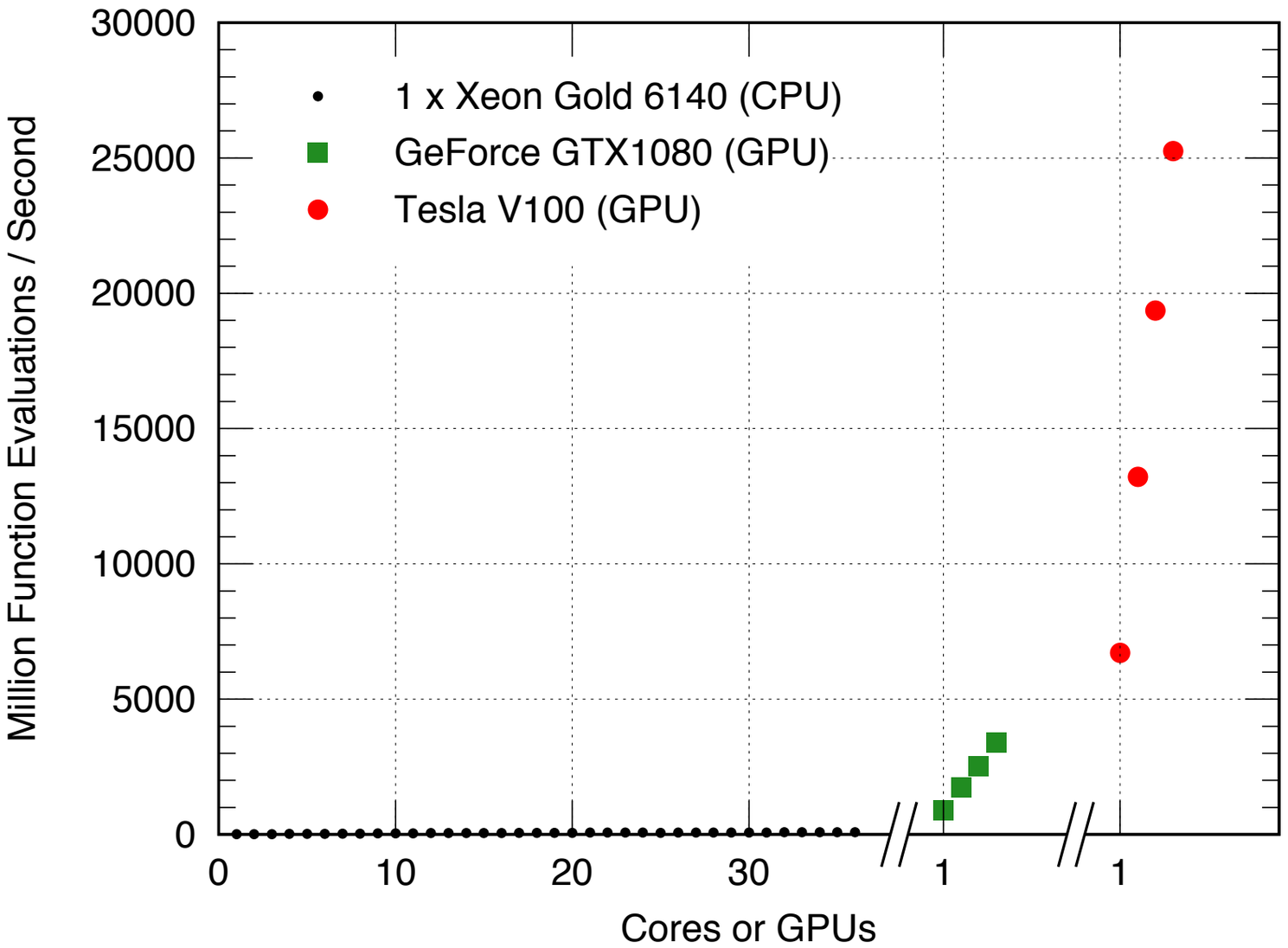
Note: Performance gain highly dependent on integrand & hardware!

Still room for further optimisations (both for CPU and GPU)

qmc: Performance

Accuracy limited by number of function evaluations

Can accelerate this using Graphics Processing Units (GPUs)



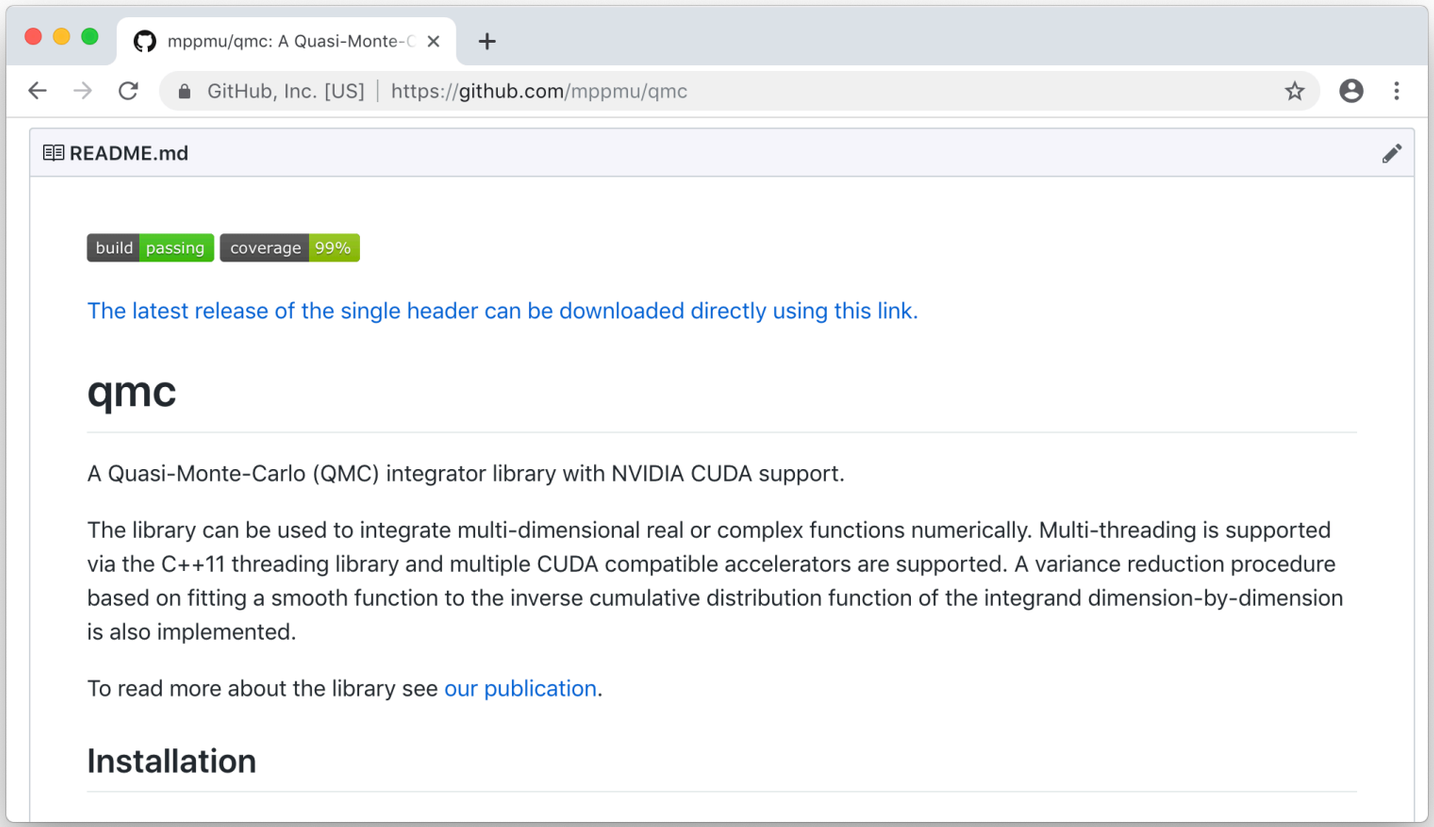
Device	M Func. Evals/s	Speedup
Xeon 6140	80.6	-
GTX1080	897	11x
Tesla V100	6710	83x

Note: Performance gain highly dependent on integrand & hardware!

Still room for further optimisations (both for CPU and GPU)

qmc: Installation

qmc: distributed as a standalone single header c++11 library
quite flexible: define your own lattices, integral transforms, floating-point types (e.g… Boost multi-precision)



Publicly available (Github)

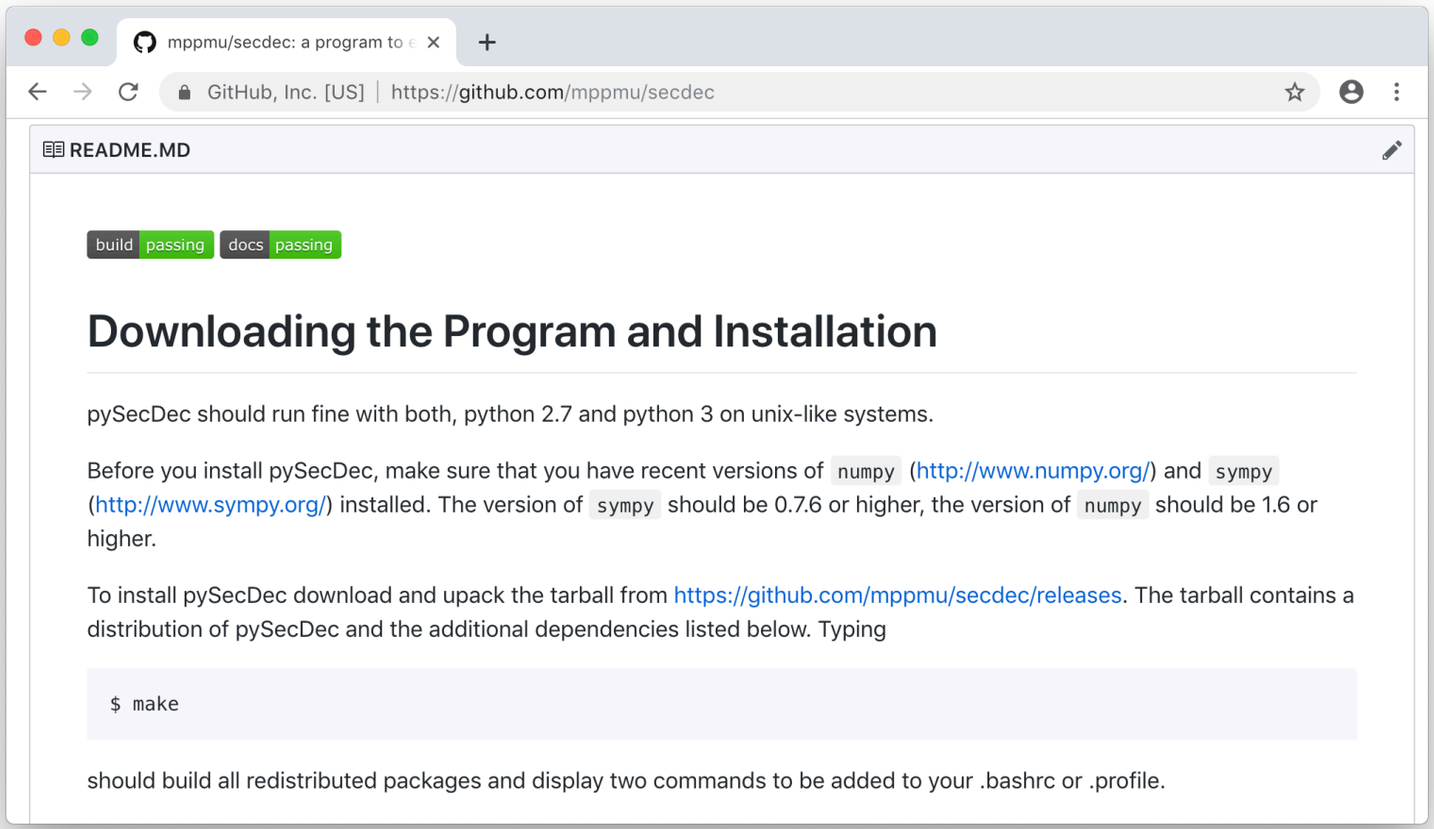
Extensive tests (CI) and coverage

Borowka, Heinrich, Jahn, SPJ, Kerner, Schlenk

pySecDec

pySecDec: a program to numerically evaluate dimensionally regulated parameter integrals (written in python, FORM & c++)

Vermaseren 00; Kuipers, Ueda, Vermaseren 13; Ruijl, Ueda, Vermaseren 17



Publicly available (Github)

Extensive tests (CI) and documentation

Borowka, Heinrich, Jahn, SPJ, Kerner, Schlenk, Zirke

New: Quasi-Monte Carlo integration & CUDA GPU Support

Borowka, Heinrich, Jahn, SPJ, Kerner, Schlenk 18

Other Sector Decomposition Tools

Several other codes implement sector decomposition

Public:

- sector_decomposition + CSectors (uses GiNaC)
  http://wwwthep.physik.uni-mainz.de/~stefanw/sector_decomposition
  Bogner, Weinzierl 07; Gluza, Kajda, Riemann, Yundin 10

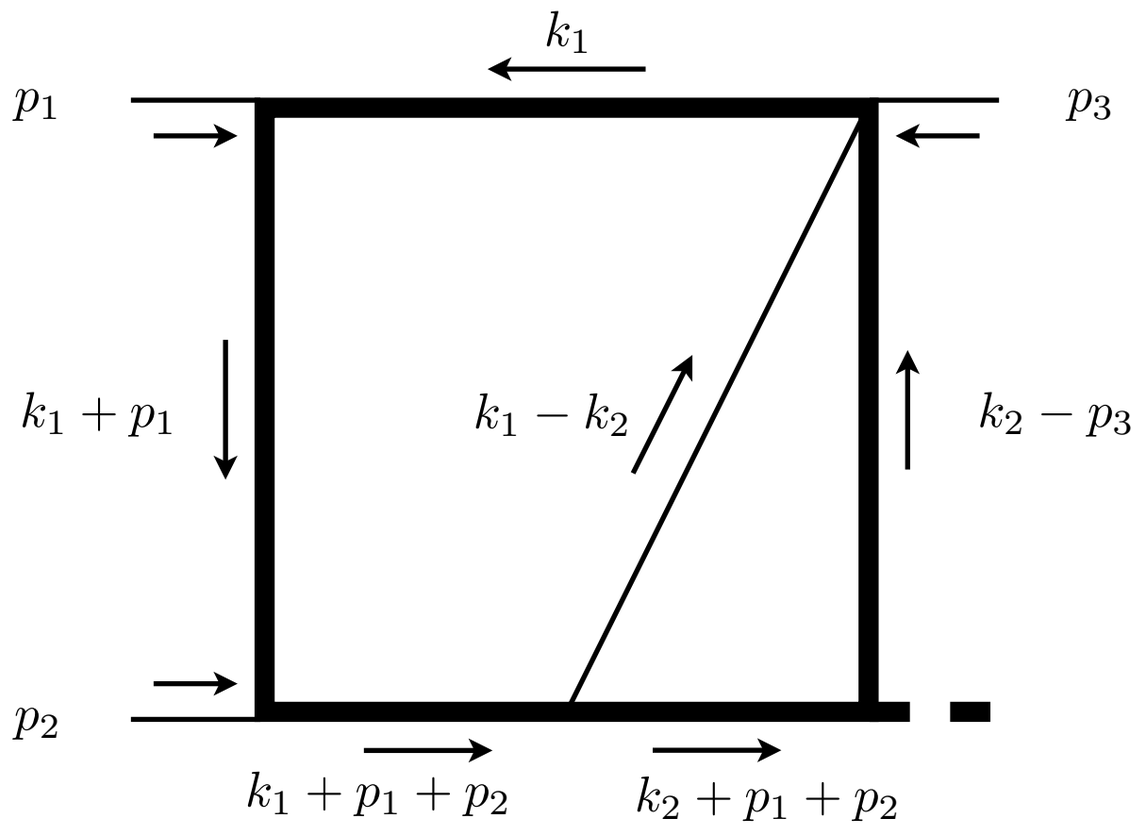
- FIESTA (uses Mathematica, C)
  http://science.sander.su/FIESTA.htm
  Supports integration on GPUs
  A. Smirnov, V. Smirnov, Tentyukov 08, 09, 13, 15; Smirnov 16

(Currently) Private:

- FORM & Python Implementations
  Fujimoto, Kaneko and Ueda 08, 10
- NIFT
  Zhao (in preparation)

Example: elliptic2L_euclidean

Massive 2-loop Box



Propagators:

$$(k_1)^2 - m^2$$
$$(k_1 + p_1 + p_2)^2 - m^2$$
$$(k_2 + p_1 + p_2)^2 - m^2$$
$$(k_1 + p_1)^2 - m^2$$
$$(k_1 - k_2)^2$$
$$(k_2 - p_3)^2 - m^2$$

Bonciani, Del Duca, Frellesvig, Henn, Moriello, Smirnov 16

Scalar Products:

$$p_1 \cdot p_1 = 0 \qquad p_2 \cdot p_2 = 0 \qquad p_3 \cdot p_3 = 0$$
$$p_1 \cdot p_2 = s/2 \qquad p_1 \cdot p_3 = t/2 \qquad p_2 \cdot p_3 = (m_H^2 - s - t)/2$$

Example: elliptic2L_euclidean (II)

Step 1: Write input files

generate_elliptic2L_euclidean.py

```
from pySecDec.loop_integral import loop_package
import pySecDec as psd

li = psd.loop_integral.LoopIntegralFromPropagators(
propagators = ['k1**2-msq', \
               '(k1+p1+p2)**2-msq', \
               '(k2+p1+p2)**2-msq', \
               '(k1+p1)**2-msq', \
               '(k1-k2)**2', \
               '(k2-p3)**2-msq'],
loop_momenta = ['k1','k2'],
replacement_rules = [
                        ('p1*p1',0),
                        ('p2*p2',0),
                        ('p3*p3',0),
                        ('p1*p2','s/2'),
                        ('p2*p3','pp4/2-s/2-t/2'),
                        ('p1*p3','t/2')
                    ]
)

Mandelstam_symbols = ['s','t','pp4']
mass_symbols = ['msq']

loop_package(
name = 'elliptic2L_euclidean',
loop_integral = li,
real_parameters = Mandelstam_symbols + mass_symbols,
additional_prefactor = '(-s/msq)**(3/2)',
requested_order = 0,
contour_deformation = False
)
```

Or: LoopIntegralFromGraph (Adjacency list)

Propagators

Scalar products

Result multiplied by this

Euclidean region

Example: elliptic2L_euclidean (III)

Step 1: Write input files

integrate_elliptic2L_euclidean.py

```
from pySecDec.integral_interface import IntegralLibrary
elliptic2L_euclidean = IntegralLibrary('elliptic2L_euclidean/elliptic2L_euclidean_pylink.so')
elliptic2L_euclidean.use_Qmc(minn=10000000,transform='korobov5')
s, t, pp4, msq = [-4./3.,-16./5.,-100./39.,1.]
_, _, result = elliptic2L_euclidean([s, t, pp4, msq])
print(result)
```

Select qmc integrator

Step 2: Generate & build c++ library, run integrator

```
$ python generate_elliptic2L_euclidean.py
$ CXX=nvcc SECDEC_WITH_CUDA=sm_70 make -C elliptic2L_euclidean
$ time python integrate_elliptic2L_euclidean.py
 + (2.4707419914073156e-01 +/- 4.3607559980517135e-16) + O(eps)
real        0m2.576s
```

Compile with nvcc

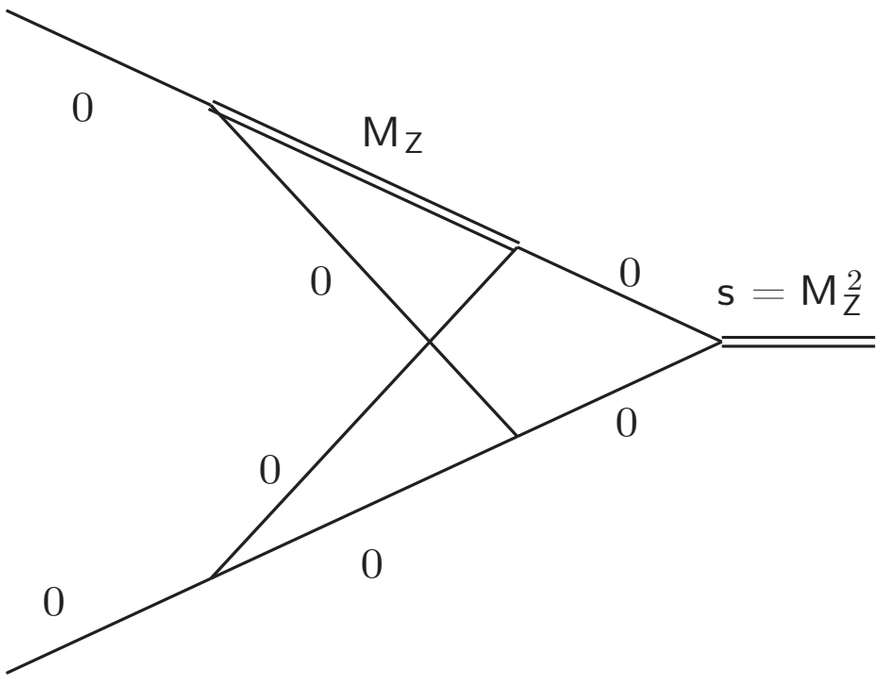
2 x Xeon Gold 6140 + 4 x Tesla V100

Integration performed on all CPU cores & GPUs on the system

Agrees with analytic result: Bonciani, Del Duca, Frellesvig, Henn, Moriello, Smirnov 16

Sector Decomposition Issues (I)

Issue: $\mathcal{F}$ polynomial does not have definite sign (no Euclidean region)



$$\mathcal{F}/m_Z^2 = x_3^2x_5 + x_3^2x_4 + x_2x_3x_5 + x_2x_3x_4$$
$$+ x_1x_3x_5 + x_1x_3x_4 + x_1x_3^2 + x_1x_2x_3$$
$$+ x_0x_3x_4 + x_0x_3^2 + x_0x_2x_3 - x_1x_2x_4$$
$$- x_0x_1x_5 - x_0x_1x_4 - x_0x_1x_2 - x_0x_1x_3$$

Here caused by internal & external m_Z

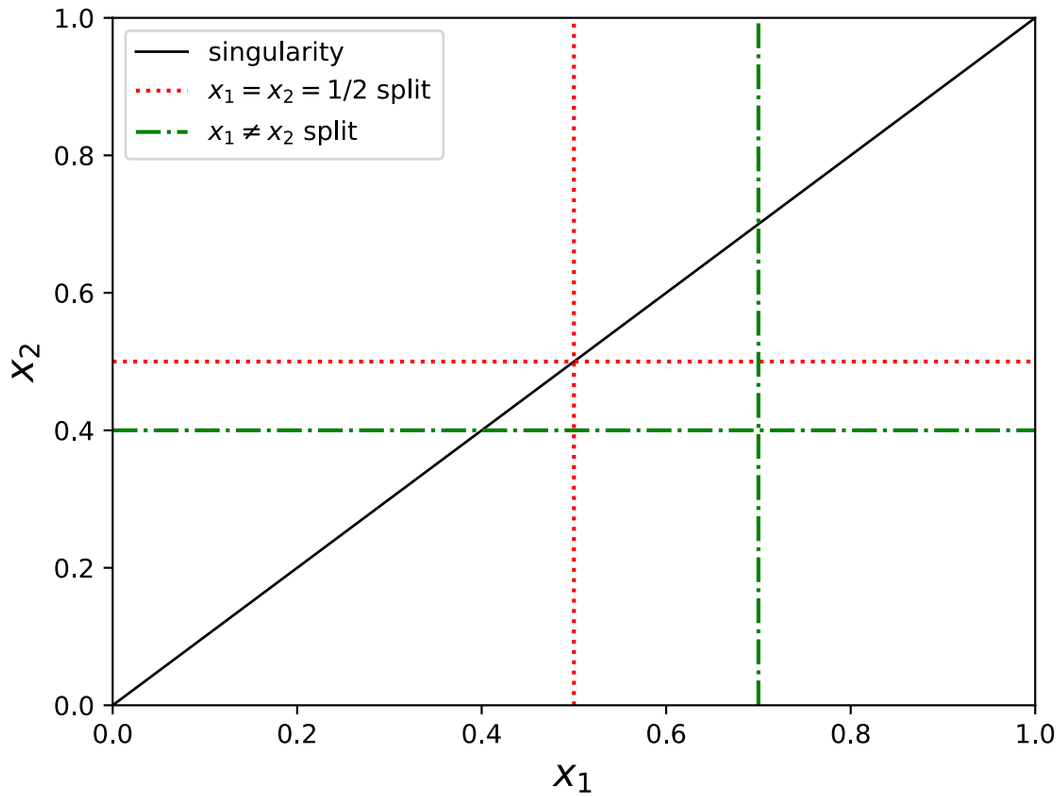
Sector decomposition resolves singularities as $\{x_i, \ldots, x_j\} \to 0$ may still have singularities as $\{x_i, \ldots, x_j\} \to 1$ which can not be avoided by a contour deformation

Can try mapping $x_j = 1 - x_j$ but this will move singularities at $x_j = 0$

Sector Decomposition Issues (II)

Solution: Split integrand then remap singularities at 1 to 0 (Split=True)

Example: $\int_0^1 \mathrm{d}x_1 \int_0^1 \mathrm{d}x_2 \, (x_1 - x_2 - i\delta)^{-2+\epsilon}$



Jahn 18

$$\int_0^1 \mathrm{d}x_j = \int_0^{r_j} \mathrm{d}x_j + \int_{r_j}^1 \mathrm{d}x_j$$

Remap to the unit hypercube:

$$x_j = r_j z_j \quad \text{and} \quad x_j = (r_j - 1)z + 1$$

Must be careful to ensure that split integrand does not again have singularities at $x_j = 1$ (pySecDec splits each variable at a random point)

Sector Decomposition Issues (III)

Issue: Sector decomposition generates integrals with worse than ``logarithmic poles''

$$\int \mathrm{d}x \; x^{a-b\epsilon} I(x) \qquad \begin{array}{ll} a = -1 & \text{Logarithmic} \\ a = -2 & \text{Linear} \\ a \leq -2 & \text{Higher} \end{array}$$

Can identify this with pySecDec by examining src/pole_structures.cpp

```
pole_structures = {{{0,0,0,0,1,-2},{0,0,-2,0,1,0},{-2,0,0,0,1,0}, ...
```

Linear poles

Solution:

1) pySecDec can handle this case directly with subtraction
2) Repeatedly apply integration by parts to soften singularity

Numerical Integration: A wider point of view

Integration by parts (IBP) identities:

$$\int \mathrm{d}^d p_i \frac{\partial}{\partial p_i^\mu} \left[q^\mu \, \mathbf{I}'(p_1, \ldots, p_l; k_1, \ldots, k_m) \right] = 0$$

Tkachov 81; Chetyrkin 81

Produce linear relations between integrals

Can perform e.g. Gaussian elimination on system of equations

Relate integrals to a smaller set (a basis) of **Master integrals**

IBPs have proven very useful for avoiding problematic integrals in the context of both analytic and numerical computations

IBPs are of course useful if we are interested in some physical result (e.g. an amplitude) but can also be used to express problematic integrals in terms of better behaved integrals

Numerical Integration: A wider point of view (II)

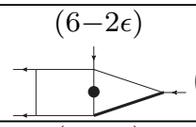
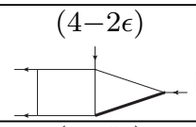
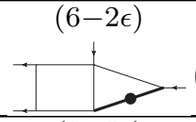
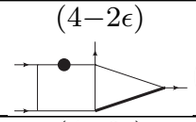
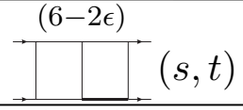
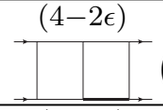
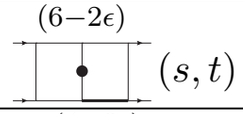
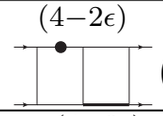
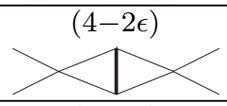
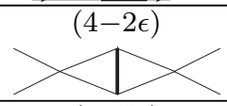
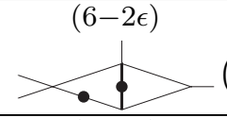
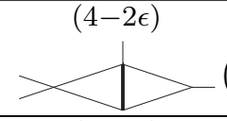
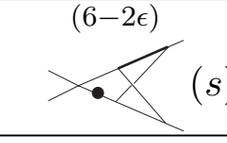
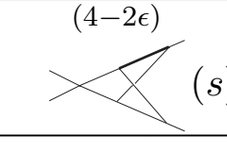
Always possible to pick **finite basis** of integrals, rewrite integrals using:

- Dimension Shifts Tarasov 96; Lee 10
- Dots (propagator taken to higher power)

Panzer 14; von Manteuffel, Panzer, Schabinger 15

Two-loop EW-QCD Drell-Yan

von Manteuffel, Schabinger 17

Finite Basis...			Conventional...		
	201 s	2.34×10^{-4}		384 s	8.12×10^{-4}
	150 s	4.83×10^{-4}		56538 s	1.67×10^{-2}
	280 s	1.00×10^{-3}		214135 s	8.29×10^{-3}
	294 s	1.21×10^{-3}		3484378 s	30.9
	91 s	3.76×10^{-4}		87 s	3.76×10^{-4}
	17 s	5.15×10^{-4}		20 s	1.95×10^{-4}
	119 s	2.32×10^{-3}		118 s	2.12×10^{-3}
Total/Max:	3995 s	5.84×10^{-3}	Total/Max:	5136862 s	30.9

← **Rel. Err.**

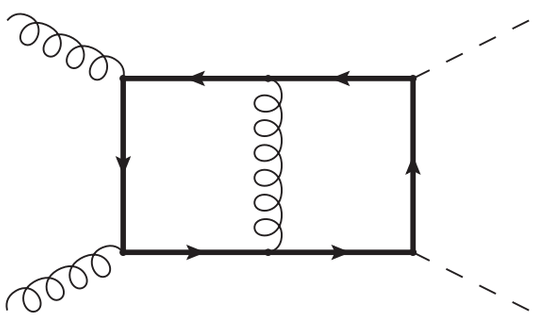
Huge decrease in time to numerically integrate and relative error

Physics Applications

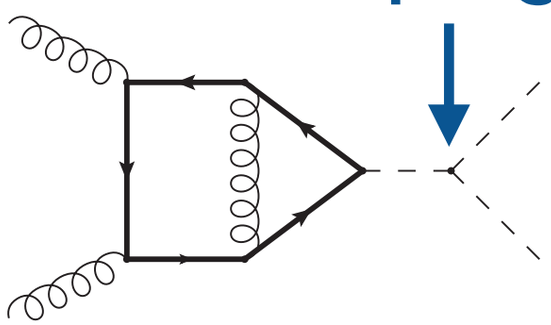
HH and HJ Production @ NLO

Di-Higgs Boson and Higgs Boson+Jet production are loop induced

HH Production



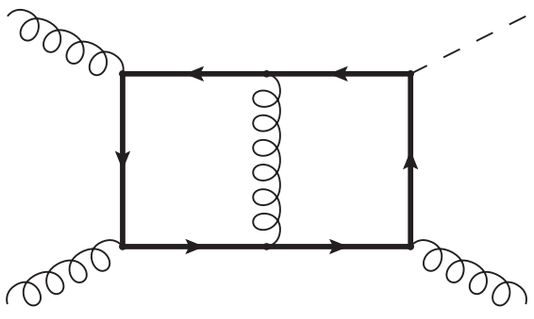
HHH coupling



Sensitive to Higgs self coupling, probe of EW symmetry breaking

$$\frac{m_H^2}{2}H^2 + \boxed{\frac{m_H^2}{2v}H^3} + \frac{m_H^2}{8v^2}H^4$$

HJ Production



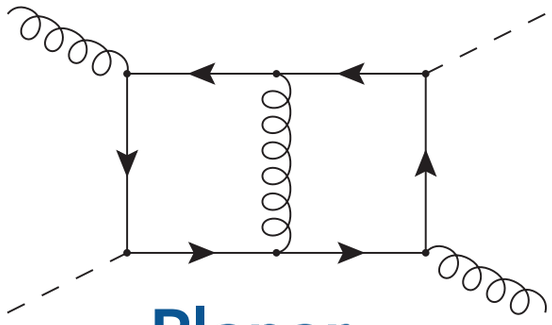
Contributes to Higgs p_T distribution
At large p_T can "probe inside the loop", distributions can be modified by heavy BSM particles

NLO corrections (2-loop) are currently known only numerically

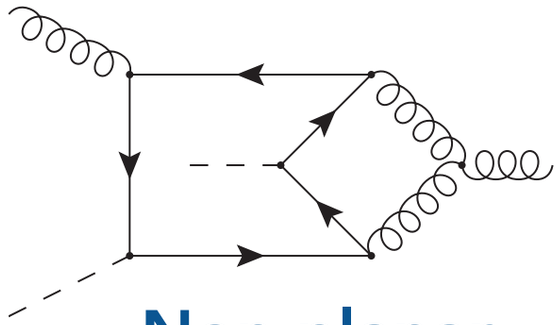
Higgs Boson Production: NLO QCD Results

NLO QCD corrections to HJ/HH production with full top quark mass:
≤7 propagator, 4-point, 2-loop diagrams, 4 mass scales (s, t, m_T, m_H)

HH:



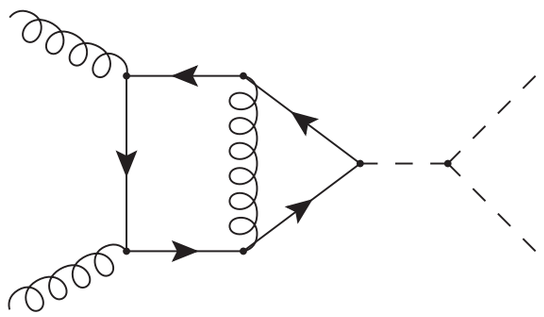
Planar



Non-planar

Reducible

Degrassi, Giardino, Gröber 16

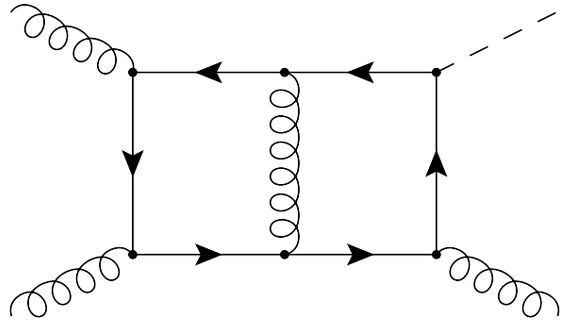


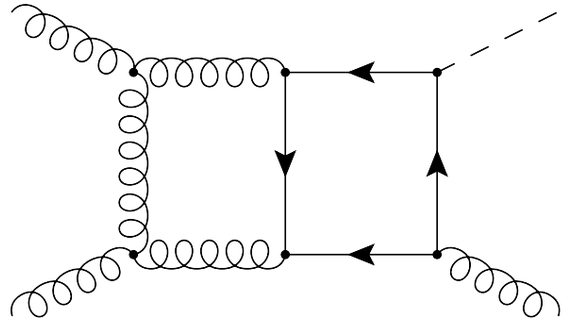
$gg \to H$

Spira, Djouadi et al. 93, 95; Bonciani, P. Mastrolia 03,04; Anastasiou, Beerli et al. 06;

$H \to Z\gamma$ Gehrmann, Guns, Kara 15;

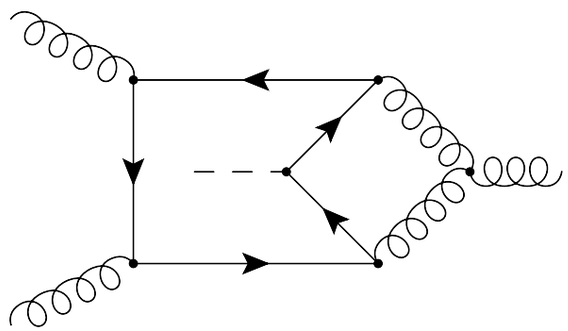
HJ:

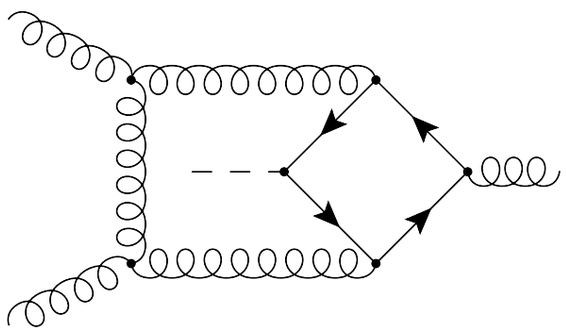




Planar

Bonciani, Del Duca, Frellesvig, Henn, Moriello, Smirnov 16; (See also Primo, Tancredi 16)





Non-planar

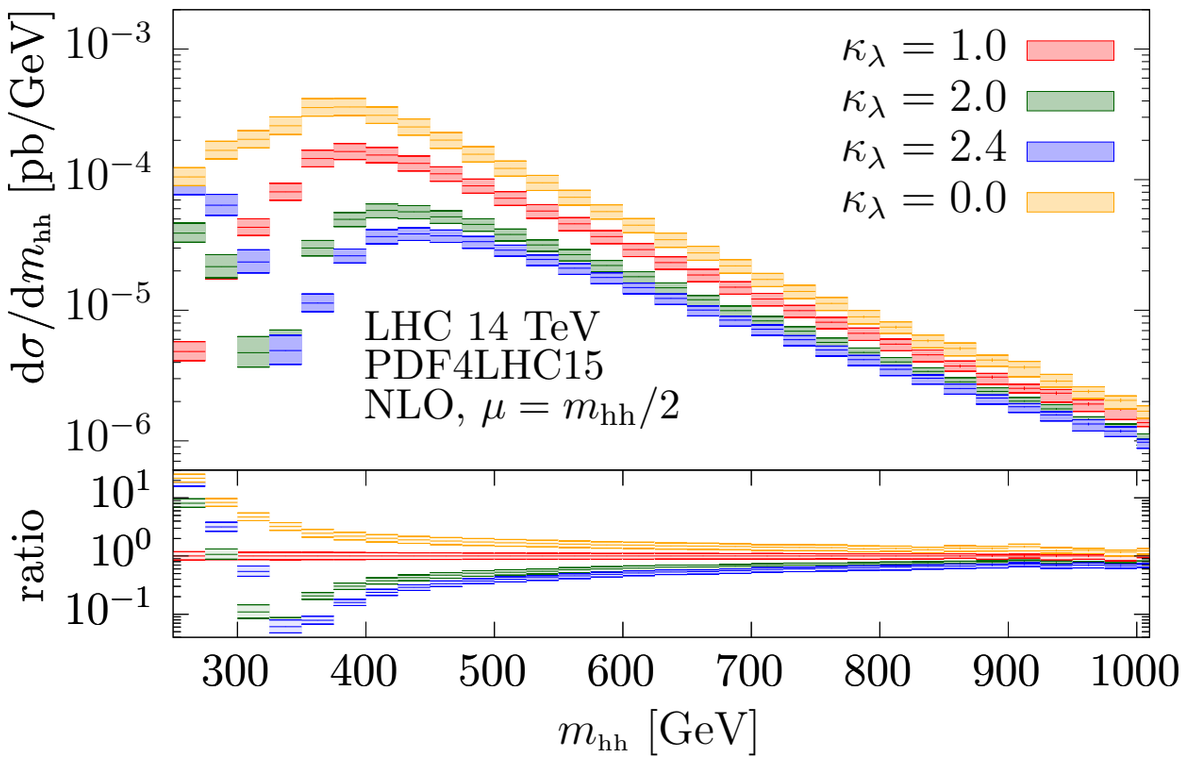
Comparison of HJ and HH

	HJ production	**HH production**
#Form factors	4+2	2
Full reduction	✓	only planar
(quasi-) finite basis	✓	only planar
#Master integrals including crossings	458	327*
#Master integrals neglecting crossings	120	215*
#Integrals after sector decomposition and expansion in ϵ	22675	11244
Code size coefficients	~340 MB	~80 MB
Code size integrals	~330 MB	~580 MB
Compile time coefficients	~2 weeks	few days
Compile time integrals	~4 hours	~1-2 days
Time for linking the program	~3-4 days	few hours

Slide: Matthias Kerner, Radcor 2017

* HH non-planar not fully reduced

Higgs Boson Pair Results



	$\sigma_{\rm LO}$ (fb)	$\sigma_{\rm NLO}$ (fb)
B.I. HEFT	$19.85^{+27.6\%}_{-20.5\%}$	$38.32^{+18.1\%}_{-14.9\%}$
FTapprox	$19.85^{+27.6\%}_{-20.5\%}$	$34.26^{+14.7\%}_{-13.2\%}$
Full Theory	$19.85^{+27.6\%}_{-20.5\%}$	$\mathbf{32.91^{+13.6\%}_{-12.6\%}}$

Borowka, Greiner, Heinrich, SPJ, Kerner, Schlenk, Schubert, Zirke 16;

Interfaced to public Monte-Carlo+Parton Shower codes via grid of precomputed phase-space points

Heinrich, SPJ, Kerner, Luisoni, Vryonidou 17

Dependence on Higgs Boson self-coupling now also available within Monte-Carlo framework

Heinrich, SPJ, Kerner, Luisoni, Scyboz 19

Several expanded results also known analytically

Gröber, Maier, Rauh 17; Davies, Mishima, Steinhauser, Wellmann 18, 18; Xu, Yang 18; Bonciani, Degrassi, Giardino, Gröber 18

HH recently recomputed by another group (also using numerical methods)

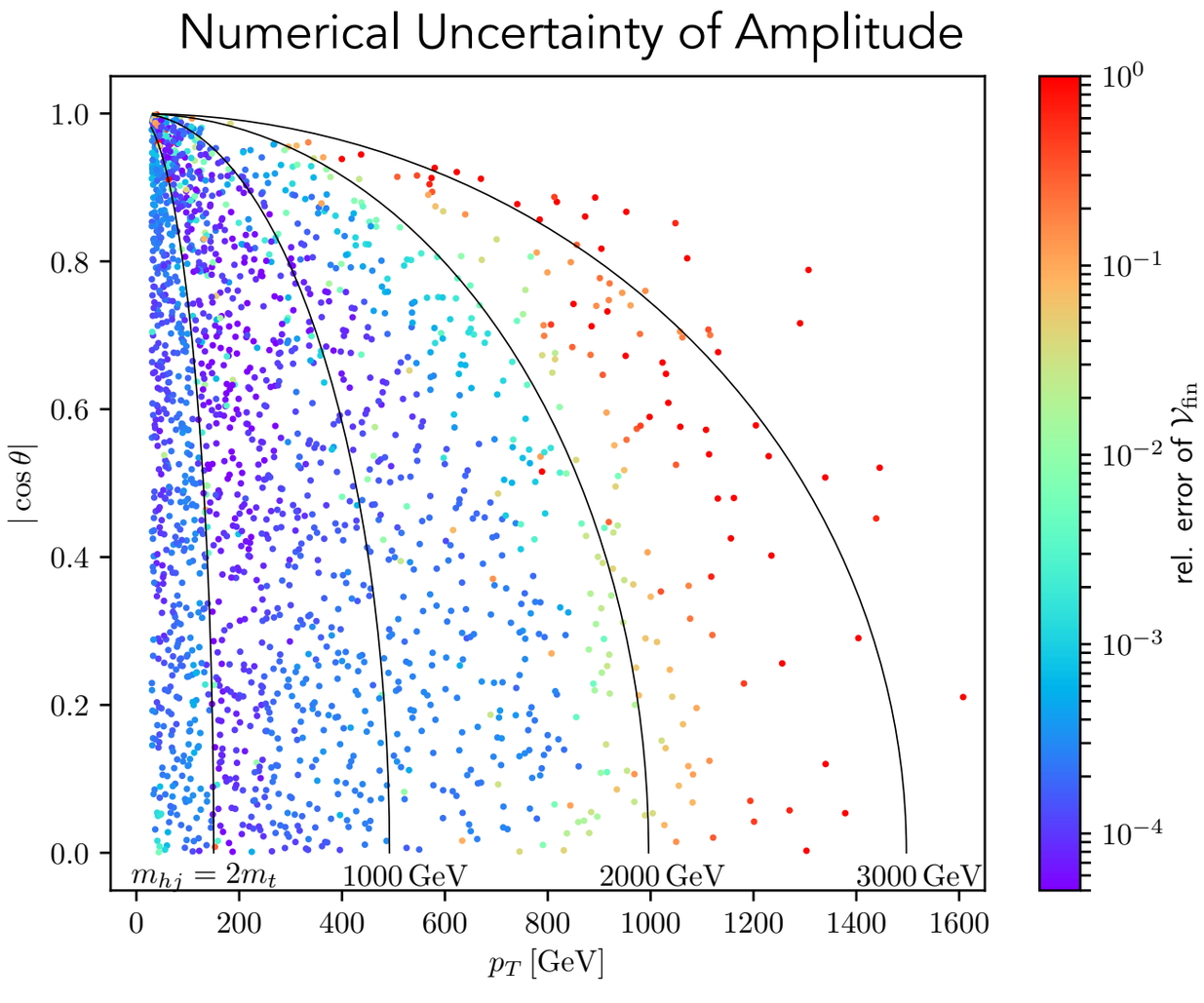
Baglio, Campanario, Glaus, Mühlleitner, Spira, Streicher 18

HJ Numerical Stability

Numerical evaluation of virtual amplitude:

- accuracy goal: 0.5% for each form factor
- wall-clock limit: 2d GPU-time (Tesla K20X GPU)

Thanks to MPCDF for compute resources



Accuracy reached for $|\mathcal{M}|^2$

- Better than per-mill for most points below $m_{hj} = 1.5$ TeV
- Region $m_{hj} \geq 2$ TeV numerically challenging
- Forward region challenging

Run time per point:

- Minimum: 1.3h
- Median: 15h

HJ Numerical Stability (II)

MI Basis Change:

- Consider quasi-finite integrals (prefer finite integrals)
- Brute force combinations of masters, factor denominators and check that dimension factorises (achieved)
- Prefer $\mathcal{F}$ to have only exponent -1 (if in denominator)
- Prefer simple denominator factors
- Prefer computing fewer orders in epsilon for each master
- Prefer simpler numerators (check number of terms/file size)

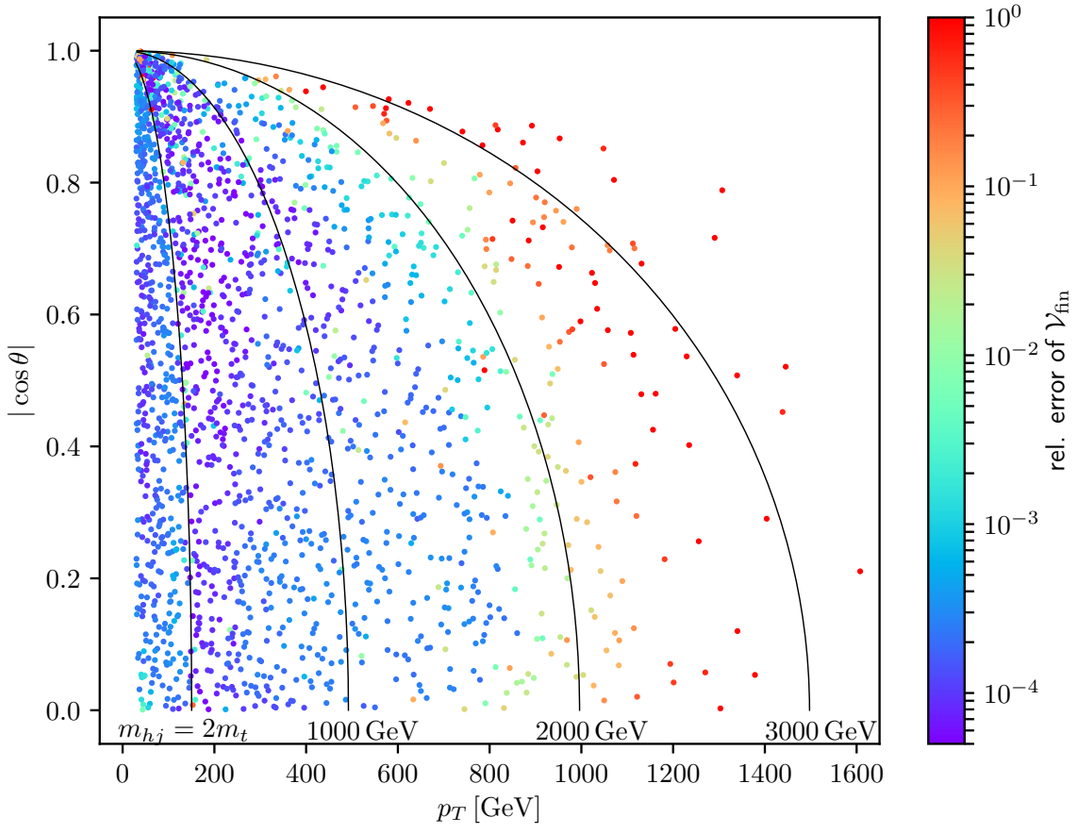
See: Matthias Kerner, Loops and Legs Proceedings 2018

Numerical Improvements:

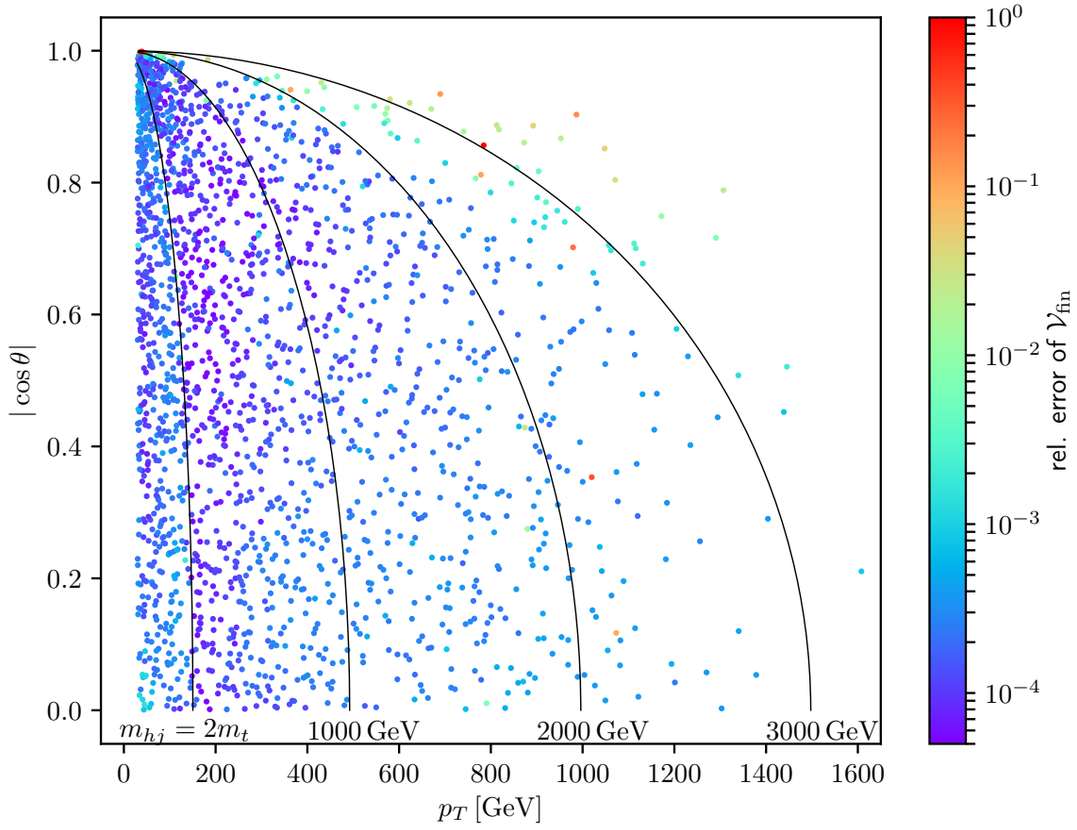
- Improve modular arithmetic implementation (needed for larger lattices)
- Do not try to further evaluate integrals if $\mathrm{rel.\ err.} < 10^{-14}$

HJ Numerical Stability (III)

Before basis change:



After basis change:



Phase-space points significantly more stable:

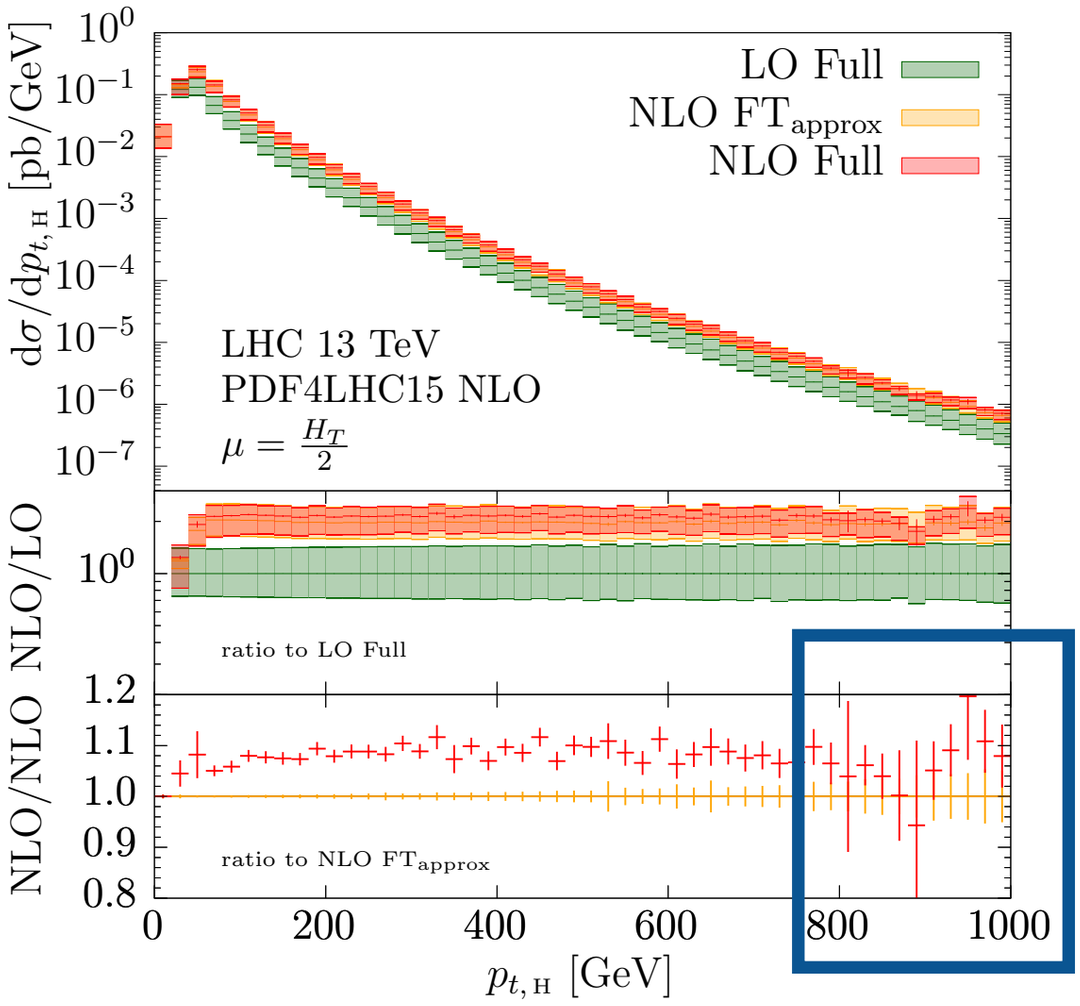
- Good accuracy around top quark threshold
- Huge improvement in accuracy at larger invariant mass (2-3 TeV)
- Improvement in the forward region

Coefficient code size: 340 MB → 100 MB

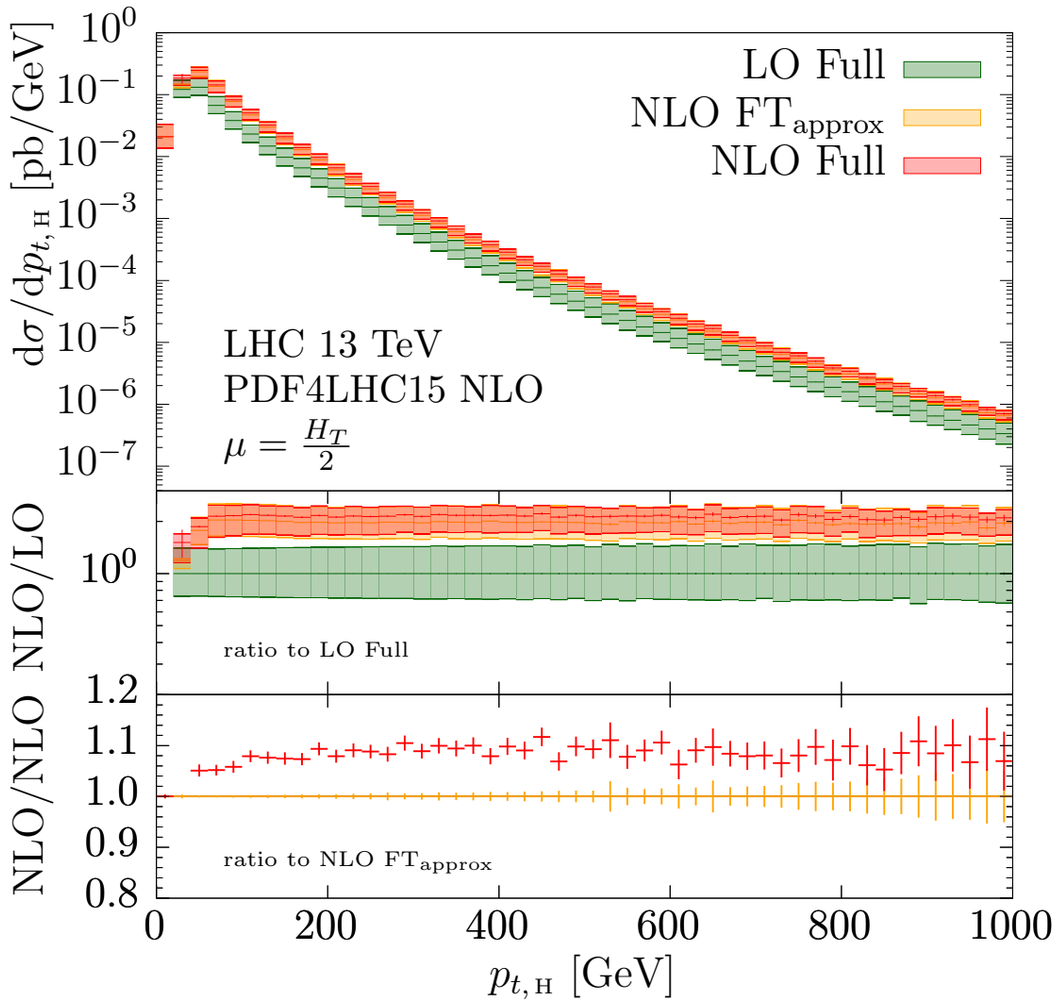
Median runtime 15h → <2h

HJ Results (IV): Higgs Boson pT

Before basis change:



After basis change:



SPJ, Kerner, Luisoni 18

Recomputing unstable points improves fluctuations in tail

Low fraction of points excluded 3/2004

Conclusion

Numerical Multi-loop Calculations

- Described sector decomposition procedure and pySecDec
- Public release of QMC integration code with CUDA GPU support
- Discussed basis choice which improves numerical performance

Physics Applications

- Presented HH & HJ at NLO with full top quark mass dependence
- 2-loop virtual amplitude calculated numerically on GPUs

Future

- Complete study of HJ
- Several multi-scale 2→2 processes left to attack
- Likely significant room for improving the numerical integration: writing better optimised integrand functions, improved variance reduction techniques (NN?), digital nets (?), …

Thank you for listening!

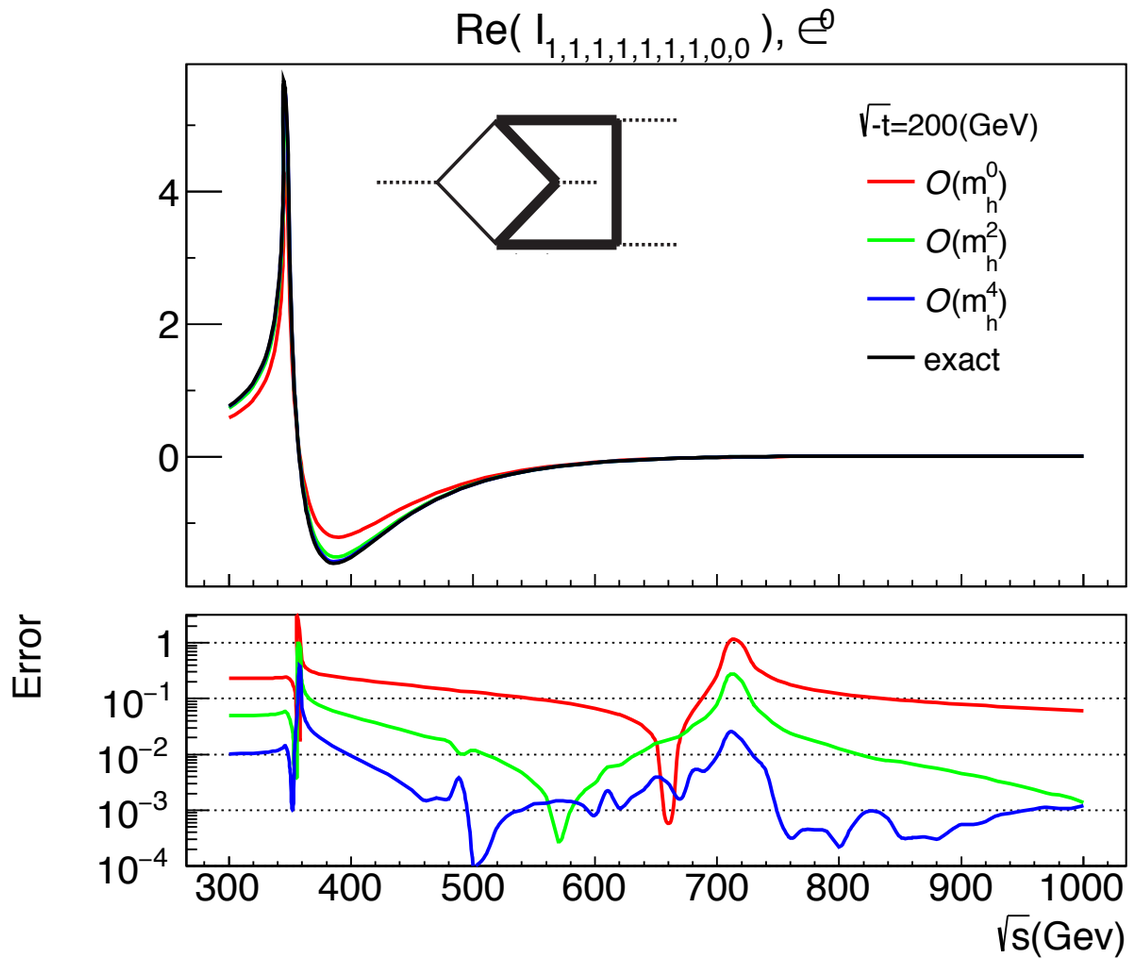
Backup

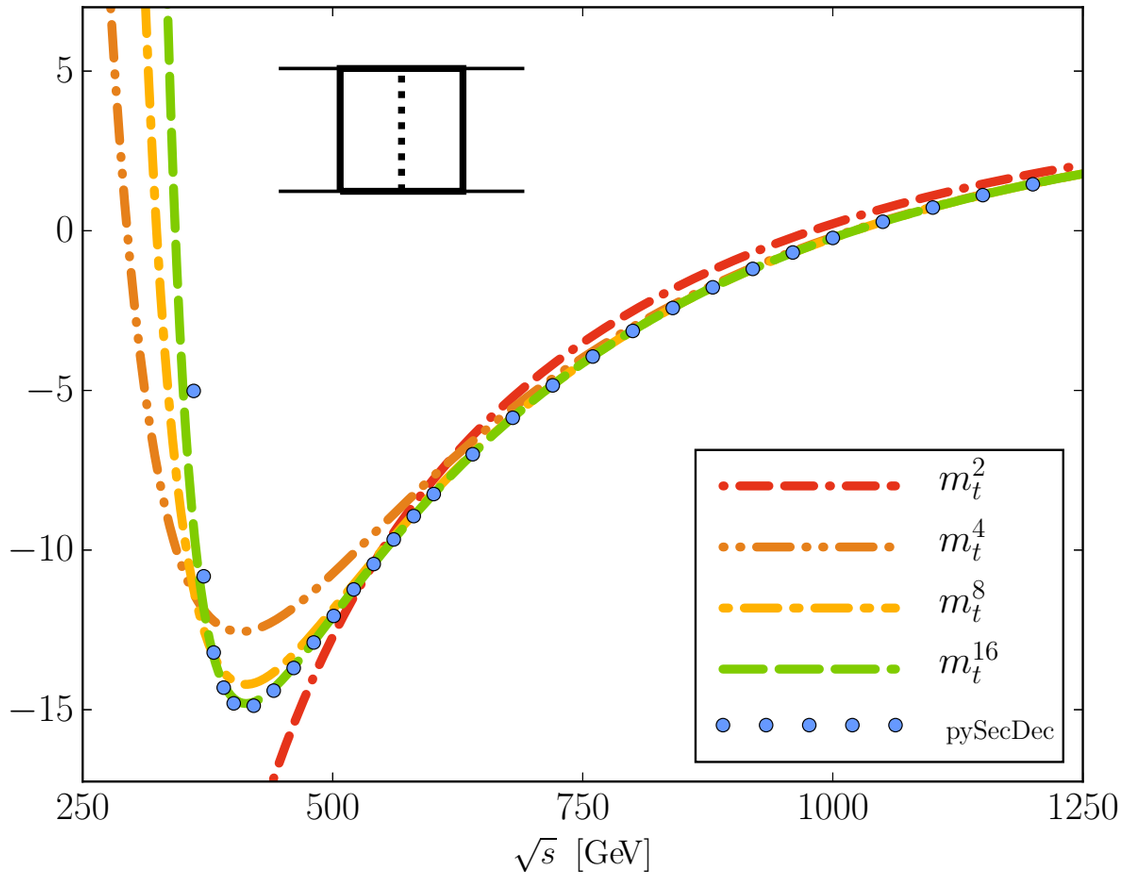
HH Production via Expansions

Expansion about small $m_H^2, m_T^2 \ll |s|, |t|$ can be applied to HH

Davies, Mishima, Steinhauser, Wellmann 18, 18;

Agreement between expanded integrals and numerical results (where expansion is valid)





Alternatively:

Expand only about small m_H^2

Larger range of validity, some integrals significantly more involved (Elliptic) Xu, Yang 18

HH Production via Expansions (II)

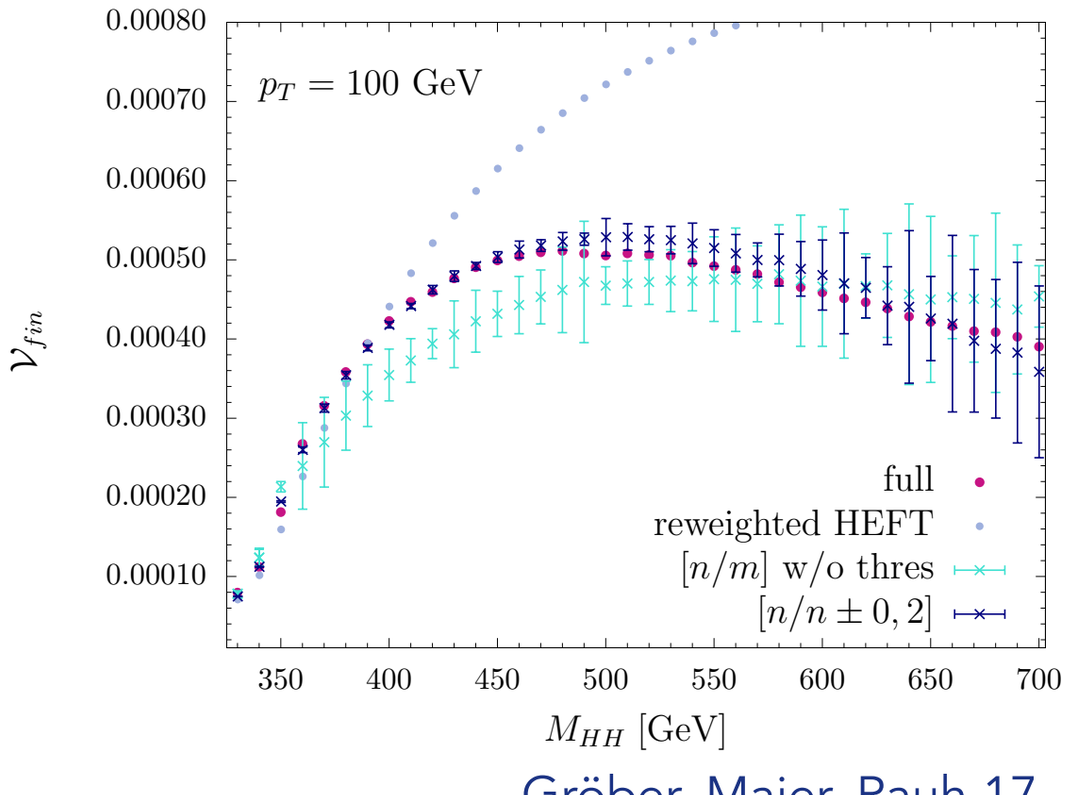
But, there is more than one way to skin a cat…

Produce a Padé approximation using:
Large- m_T expansion + threshold expansion

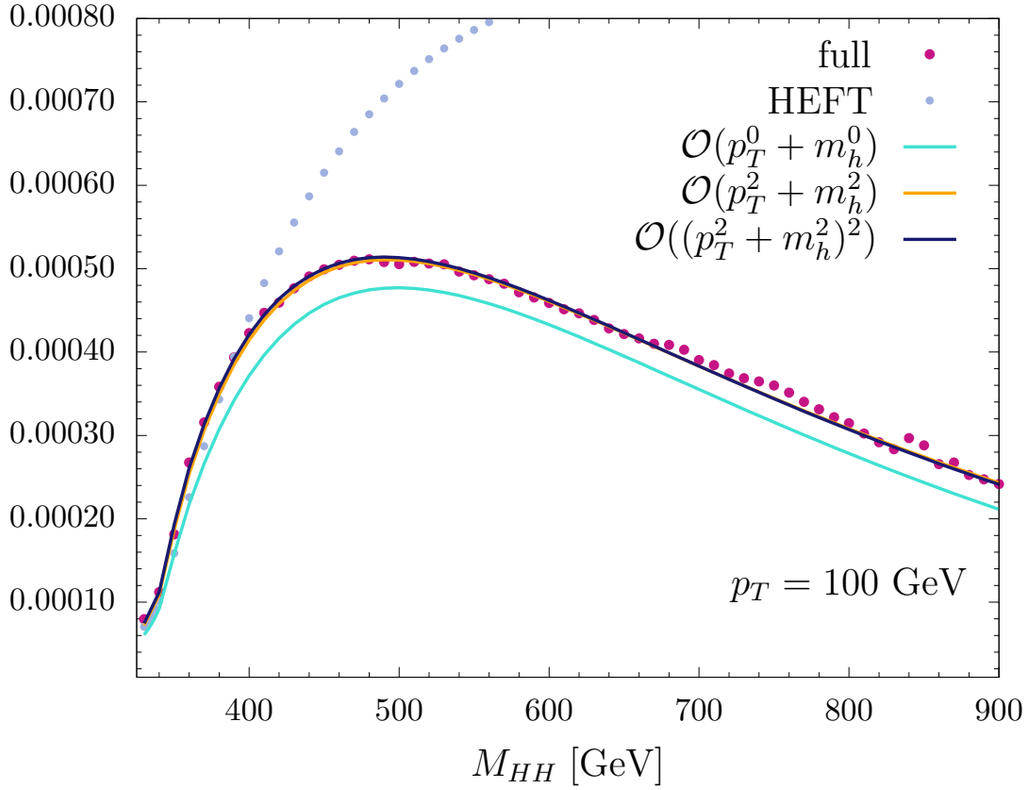
Incorporate non-analytic threshold corrections into approximation

Method applicable to more processes:

$$gg \to H^{(*)}, HZ, ZZ$$



Gröber, Maier, Rauh 17



Have: $p_T^2 + m_H^2 \leq \hat{s}/4$

Expand in: $p_T^2 + m_H^2$

Solve remaining dependence on $\hat{s}, m_T$

Invariant mass distribution agrees well with full (numerical) result up to ~900 GeV

Bonciani, Degrassi, Giardino, Gröber 18

HJ Expansion

Alternatively can consider Higgs boson & top quark masses as small

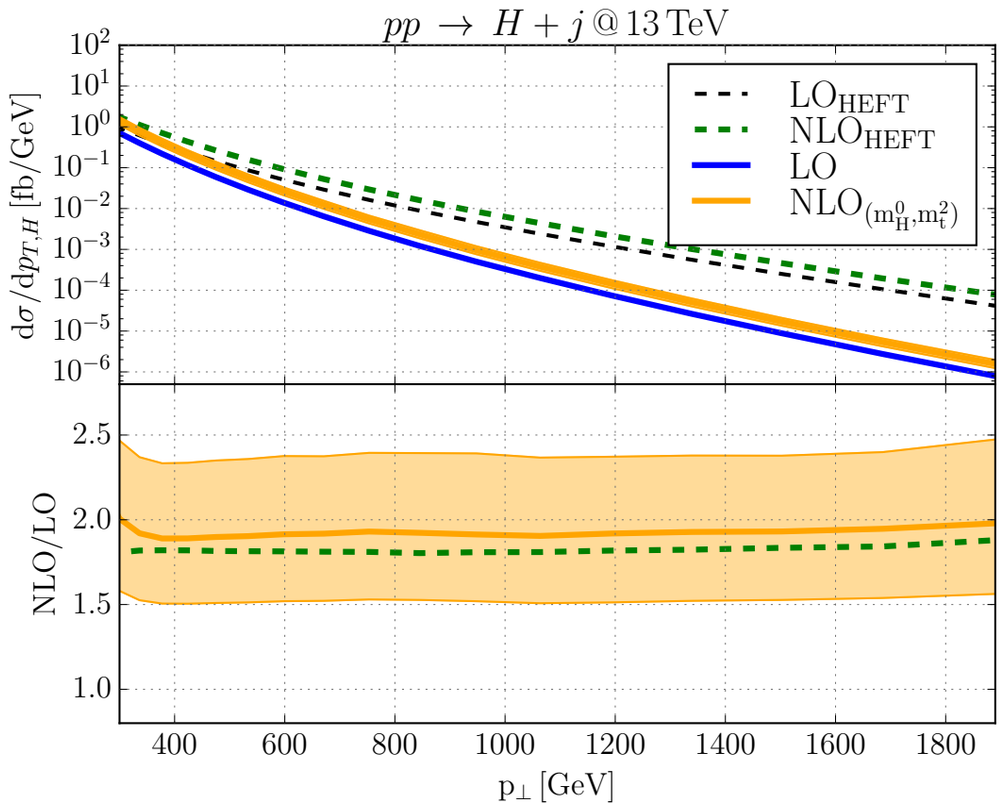
Introduce variables:

$$\eta = -\frac{m_H^2}{4m_T^2}, \quad \kappa = -\frac{m_T^2}{s}, \quad z = \frac{u}{s}$$

Expand integrals to $\mathcal{O}(\eta^0\kappa^1)$ justified for $m_H^2, m_T^2 \ll |s| \sim |t| \sim |u|$,

For example at large $p_T^2 = ut/s$

Kudashkin, Melnikov, Wever 17



Expanded 2-loop virtuals can be combined with full reals to predict Higgs boson p_T distribution above top threshold Lindert, Kudashkin, Melnikov, Wever 18; Neumann 18

$$\frac{K^{\mathrm{SM}}}{K^{\mathrm{HTL}}} = 1.04 \ldots 1.06$$

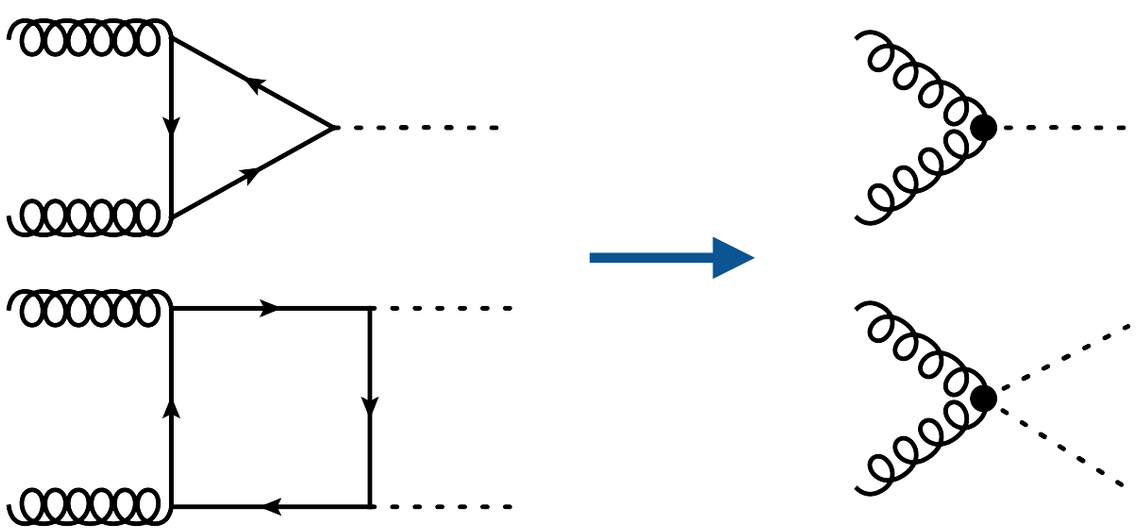
Minor difference to full result, due to missing $\mathcal{O}(\eta^1)$ terms?

Heavy Top Limit

Heavy Top Limit (HTL): $m_T \to \infty$

Effective tree-level couplings between gluons and Higgs

Lowers number of loops by 1



HTL valid for

$$\sqrt{\hat{s}} \ll 2m_T$$

HH production for

$$2m_H < \sqrt{\hat{s}}$$

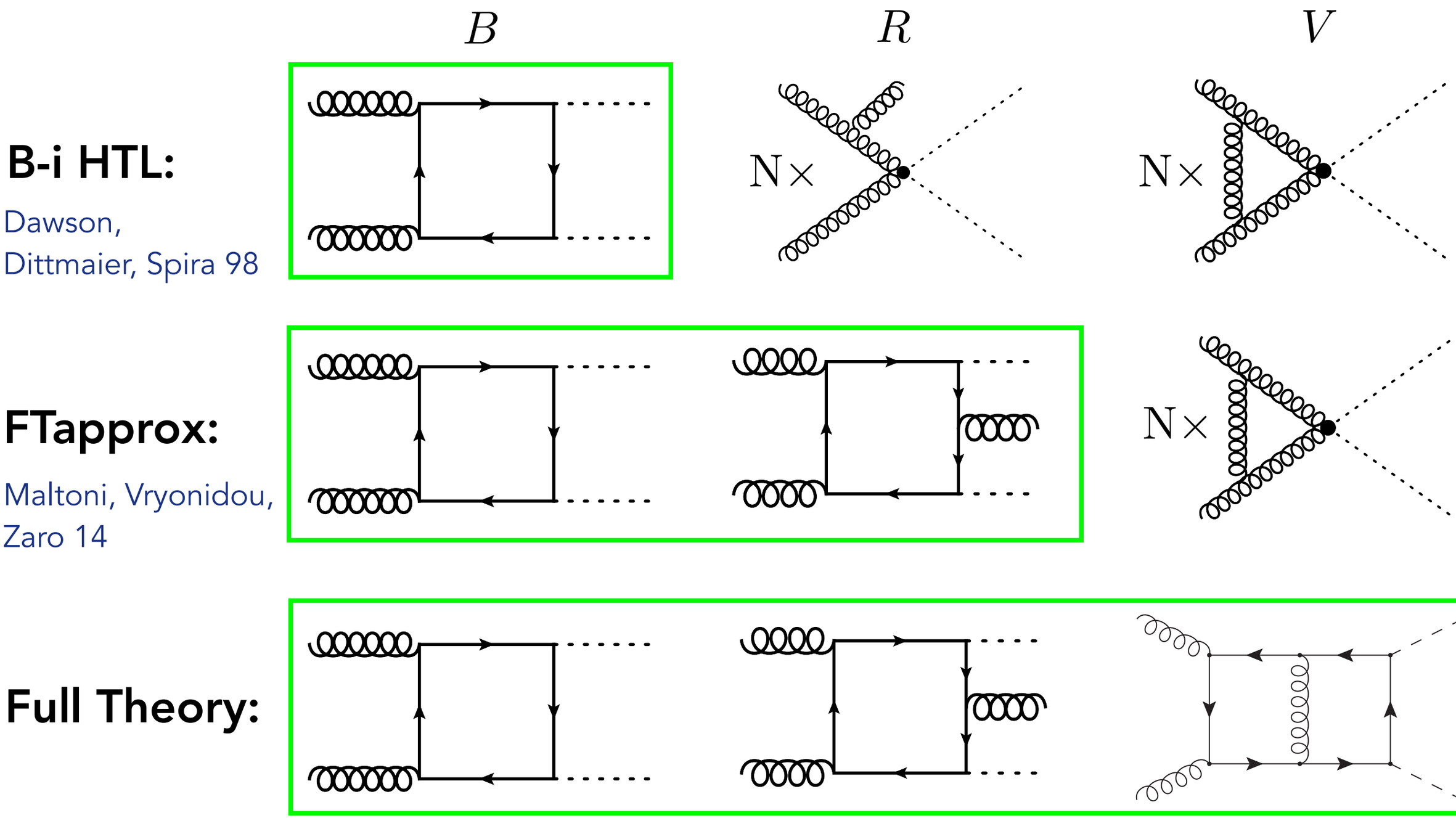
Small energy range in which HTL is technically justified

Born improved NLO HTL:

$$\mathrm{d}\sigma_{\mathrm{NLO}}(m_T) \approx \mathrm{d}\bar{\sigma}_{\mathrm{NLO}}(m_T) \equiv \underbrace{\frac{\mathrm{d}\sigma_{\mathrm{LO}}(m_T)}{\mathrm{d}\sigma_{\mathrm{LO}}(m_T \to \infty)}}_{\mathrm{N}} \mathrm{d}\sigma_{\mathrm{NLO}}(m_T \to \infty)$$

Spira et al. (HPAIR)

HH Approximations @ NLO (Schematically)



Borowka, Greiner, Heinrich, SPJ, Kerner, Schlenk, Schubert, Zirke 16;
Borowka, Greiner, Heinrich, SPJ, Kerner, Schlenk, Zirke 16;
Baglio, Campanario, Glaus, Mühlleitner, Spira, Streicher 18;

Amplitude Evaluation

Use Quasi-Monte Carlo (QMC) integration $\mathcal{O}(n^{-1})$ error scaling

Li, Wang, Yan, Zhao 15; Review: Dick, Kuo, Sloan 13;

Implemented in OpenCL, evaluated on GPUs

Entire 2-loop amplitude evaluated with a single code

$$F = \sum_i \left(\sum_j C_{i,j} \epsilon^j \right) \left(\sum_k I_{i,k} \epsilon^k \right) = \epsilon^{-2} \left[C^{(L)}_{1,-2} I^{(L)}_{1,0} + \ldots \right] + \epsilon^{-1} \left[C^{(L)}_{1,-1} I^{(L)}_{1,0} + \ldots \right] + \ldots$$

coeff. integral

compute once

Dynamically set target precision for each sector, minimising time:

$$T = \sum_i t_i + \bar{\lambda} \left(\sigma^2 - \sum_i \sigma_i^2 \right), \quad \sigma_i \sim t_i^{-e}$$

$\bar{\lambda}$ – Lagrange multiplier

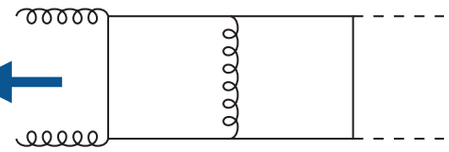
σ – precision goal

σ_i – error estimate

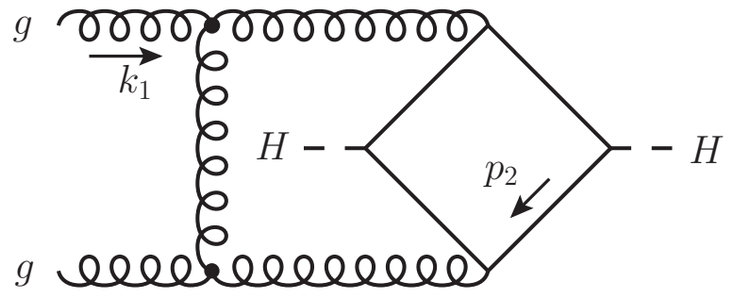
Amplitude Evaluation

Contributing integrals: $\sqrt{\hat{s}} = 327.25\,\text{GeV}, \sqrt{-\hat{t}} = 170.05\,\text{GeV}, M^2 = \hat{s}/4$

integral	value	error	time [s]
...			
F1_011111110_ord0	(0.484, 4.96e-05)	(4.40e-05, 4.23e-05)	11.8459
...			
N3_111111100_k1p2k2p2_ord0	(0.0929, -0.224)	(6.32e-05, 5.93e-05)	235.412
N3_111111100_1_ord0	(-0.0282, 0.179)	(8.01e-05, 9.18e-05)	265.896
N3_111111100_k1p2k1p2_ord0	(0.0245, 0.0888)	(5.06e-05, 5.31e-05)	282.794
N3_111111100_k1p2_ord0	(-0.00692, -0.108)	(3.05e-05, 3.05e-05)	433.342



$$I(s,t,m_t^2,m_h^2) = -\left(\frac{\mu^2}{M^2}\right)^{2\varepsilon} \Gamma(3+2\epsilon) M^{-4} \left(\frac{A_{-2}}{\epsilon^2} + \frac{A_{-1}}{\epsilon^1} + A_0 + \mathcal{O}(\epsilon)\right)$$



Sector Decomposition

sector	integral value	error	time [s]	#points
5	(-1.34e-03, 2.00e-07)	(2.38e-07, 2.69e-07)	0.255	1310420
6	(-1.58e-03, -9.23e-05)	(7.44e-07, 5.34e-07)	0.266	1310420
...				
41	(0.179, -0.856)	(1.10e-05, 1.22e-05)	29.484	79952820
42	(0.359, -1.308)	(1.40e-06, 1.58e-06)	80.24	211436900
44	(0.0752, -1.185)	(5.44e-07, 6.76e-07)	99.301	282904860

Slide: Matthias Kerner

hhgrid

Amplitude is slow to evaluate:

Accuracy goal: 3% for F_1, 5-20% for F_2 (depending on F_2/F_1)

GPU Time/PS point: 80 min - 2 days (median 2 hours)

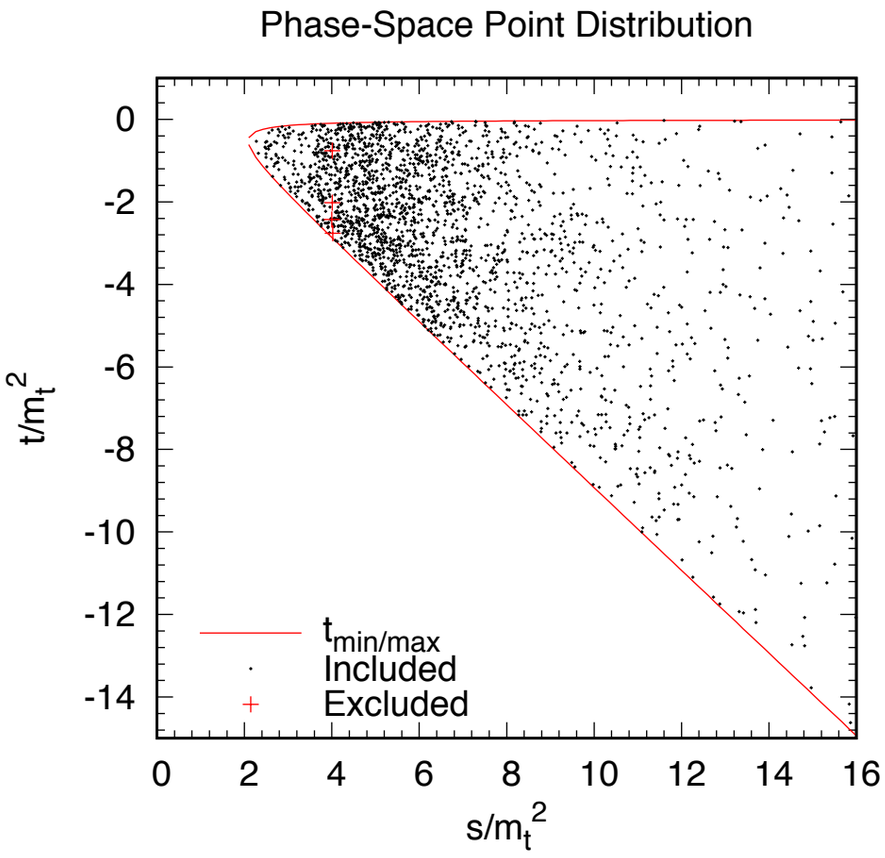
If a theoretical calculation is done, but it can not be used by any experimentalists, does it make a sound? - J. Huston

Constructed a grid in Mandelstam s, t

Based on 3746 phase-space points

Interfaced to POWHEG, MG5_amc@NLO

Sherpa



Heinrich, SPJ, Kerner, Luisoni, Vryonidou 17; SPJ, Kuttimalai 18

https://github.com/mppmu/hhgrid

Grid Details

Matrix element takes hours per phase-space point

Can not put directly into a Monte Carlo

But: Virtual matrix element depends only on $\hat{s}, \hat{t}$ (fixed m_T, m_H)

Can build 2D grid of our phase-space points and interpolate

Parametrisation

$$x = f(\beta(s)), \quad c_\theta = |\cos\theta| = \left|\frac{s + 2t - 2m_H^2}{s\beta(s)}\right|, \quad \beta = \left(1 - \frac{4m_H^2}{s}\right)^{\frac{1}{2}}$$

Choose $f(\beta)$ according to cumulative distribution function of phase space points used in the original calculation

Obtain nearly uniform distribution in (x, c_θ) unit square

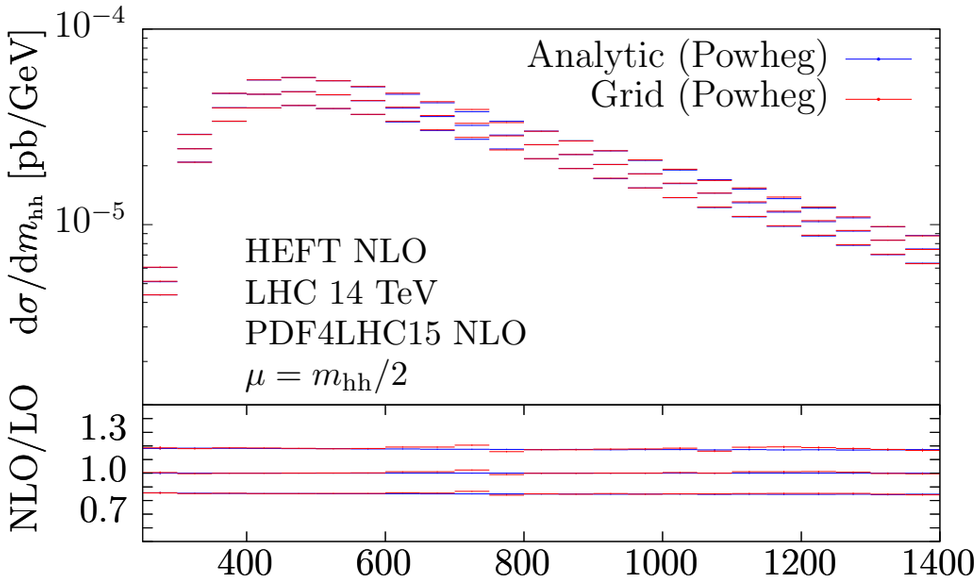
Interpolate with Clough-Tocher using the SciPy package

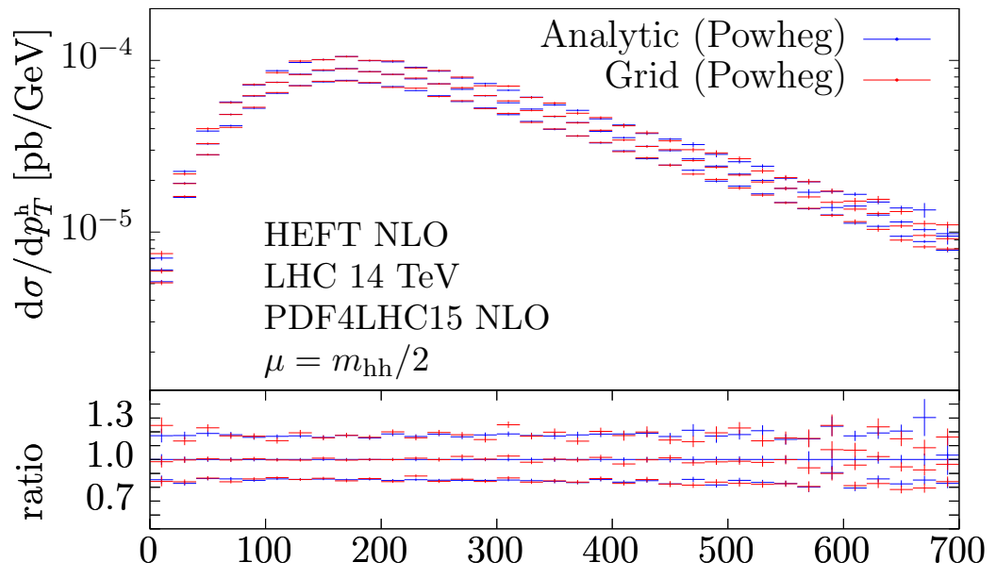
Clough, Tocher 65

2402+1344 events used for POWHEG/MG5_amc@NLO

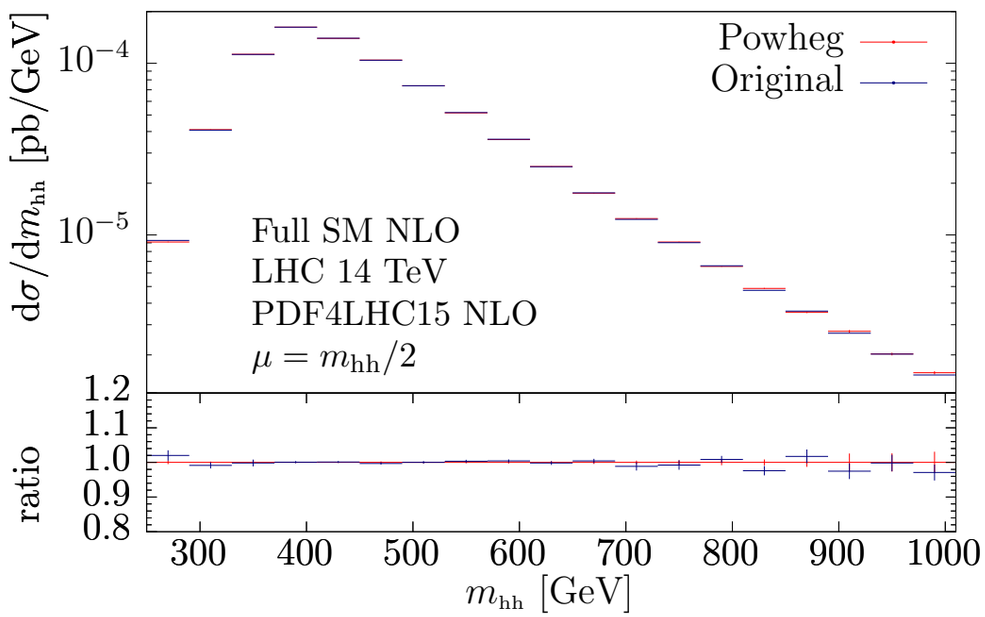
Grid Validation

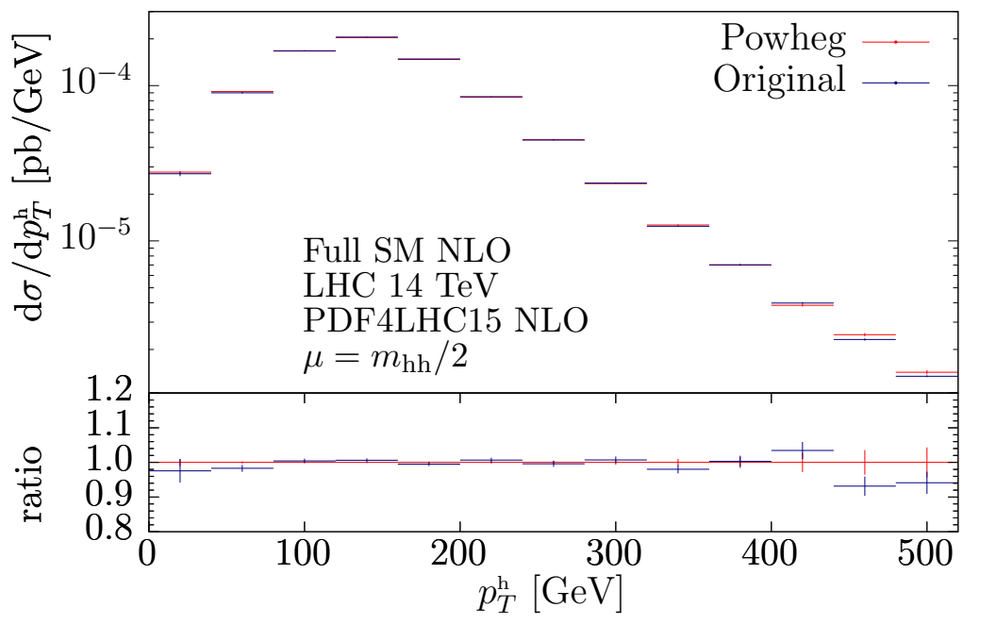
Use HEFT to study validity of grid





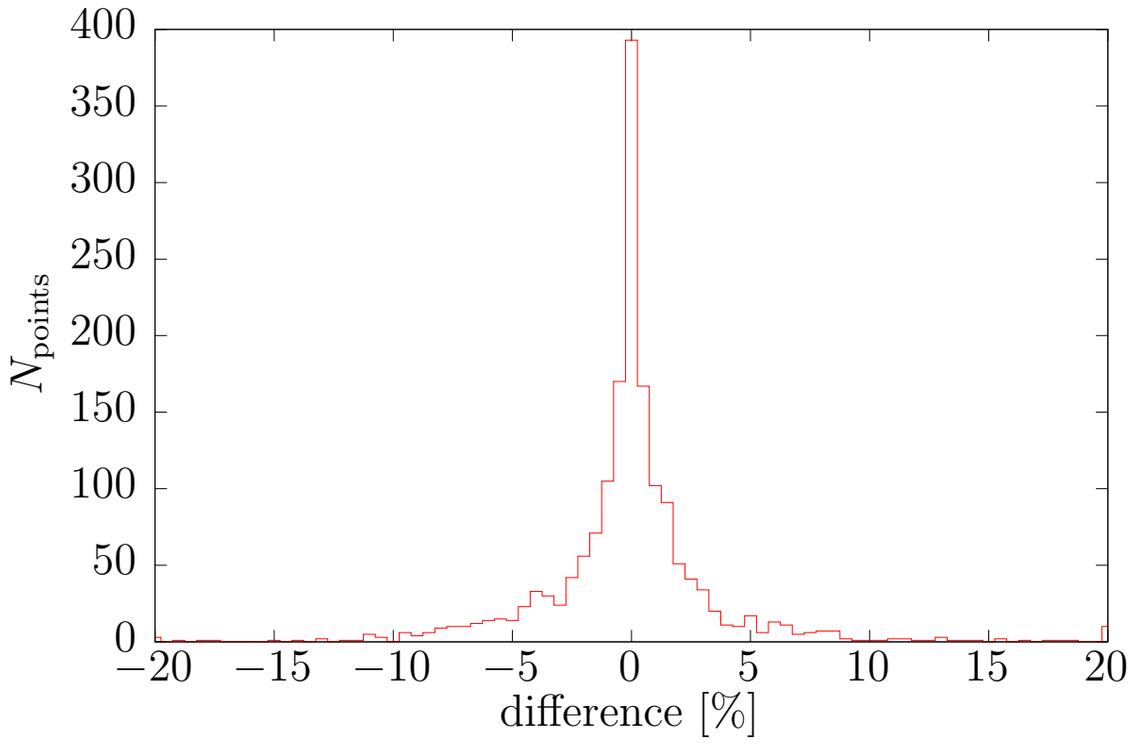
Full SM compare POWHEG (grid) with our original results



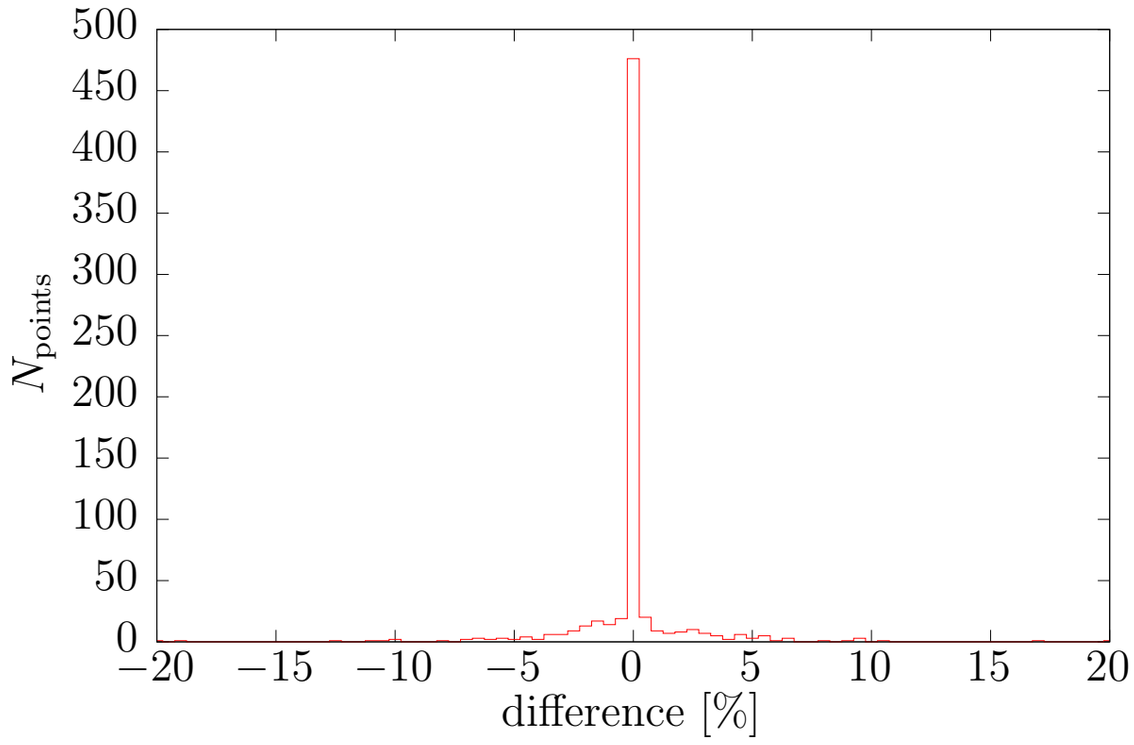


Heinrich, SPJ, Kerner, Luisoni, Vryonidou 17

Grid Stability



Grid vs grid with 50% of points



Grid vs grid with 80% of points

HH Checks

Real Emission / Subtraction Terms

- Independence of dipole-cut α_{cut} parameter Nagy 03
- Agreement with literature Maltoni, Vryonidou, Zaro 14
- Agreement with FKS (`POWHEG/MG5_amc@NLO`)

  Frixione, Kunszt, Signer 96; Nason 04; Frixione et al 07; Alioli et al. 10; J. Alwall et al. 14

Virtual Corrections

- Two calculations of amplitude up to reduction
- Amplitude result invariant under $t \leftrightarrow u$
- Pole cancellation
- Mass renormalization using two methods:
  counter-term insertion vs. calculating $\mathrm{d}\mathcal{M}^{\mathrm{LO}}/\mathrm{d}m_T^2$ numerically
- Agreement of contributions $gg \to H \to HH$ with `SusHi` Harlander, Liebler, Mantler 13,16
- Convergence of $1/m_T$ expansion to full result
  where agreement is expected