TECHNISCHE
UNIVERSITÄT
DRESDEN

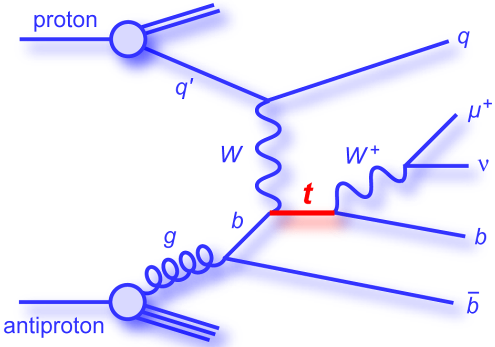
Anton Reinhard
Technische Universität Dresden

Optimizations on DAG-Representations of Domain-Specific Computations for Heterogeneous Systems and Application to Quantum Electrodynamics

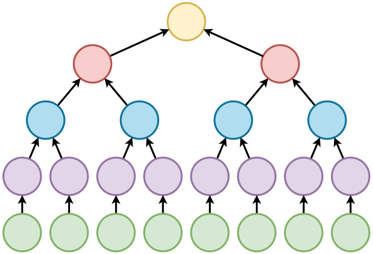
JuliaHEP, 23.05.2024

Structure

1. Introduction
2. The Pipeline
   2.1 Generating Diagrams
   2.2 Translation to DAGs
   2.3 Optimization
   2.4 Code Generation
   2.5 Execution
3. Summary & Future Work



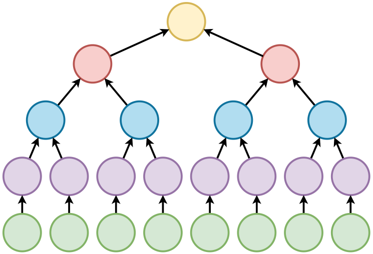
Introduction - Motivation



Goals:

- Use graph representation for high-level optimizations

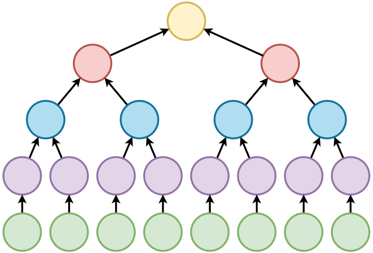
Introduction - Motivation



Goals:

- Use graph representation for high-level optimizations
- Scale the code with the process

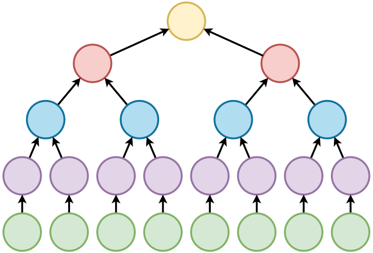
Introduction - Motivation



Goals:

- Use graph representation for high-level optimizations
- Scale the code with the process
- Support multiple platforms (CPU, GPU) with generic code

Introduction - Motivation



Goals:

- Use graph representation for high-level optimizations
- Scale the code with the process
- Support multiple platforms (CPU, GPU) with generic code
- Benefit from all available hardware

Introduction - Language

- Implementation done in Julia

Introduction - Language

- Implementation done in Julia
- **Why Julia?**

Introduction - Language

- Implementation done in Julia
- **Why Julia?**
    1. Multiple dispatch is helpful for elegantly implementing particle interactions

Introduction - Language

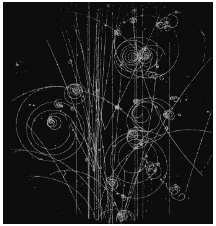
- Implementation done in Julia
- **Why Julia?**
  1. Multiple dispatch is helpful for elegantly implementing particle interactions
  2. DAG analysis, optimization, and code generation easily in the same language and same session

Introduction - Language

- Implementation done in Julia
- **Why Julia?**
  1. Multiple dispatch is helpful for elegantly implementing particle interactions
  2. DAG analysis, optimization, and code generation easily in the same language and same session
  3. Interfacing with existing code of the QED.jl project

Introduction - Application: Quantum Field Theory

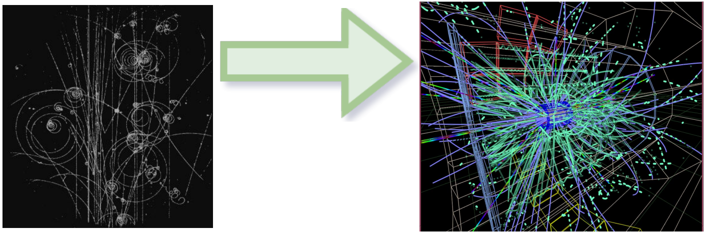
– Experimentation and observation



Experimentation

Introduction - Application: Quantum Field Theory

- Experimentation and observation needs computation and simulation

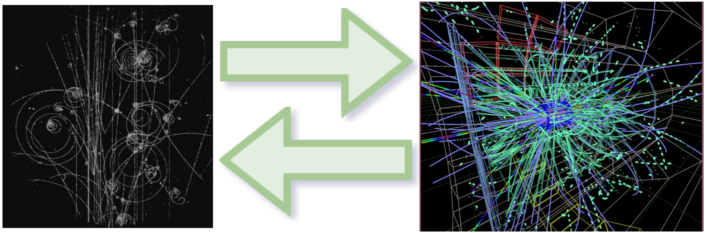


Experimentation

Simulation

Introduction - Application: Quantum Field Theory

- Experimentation and observation needs computation and simulation

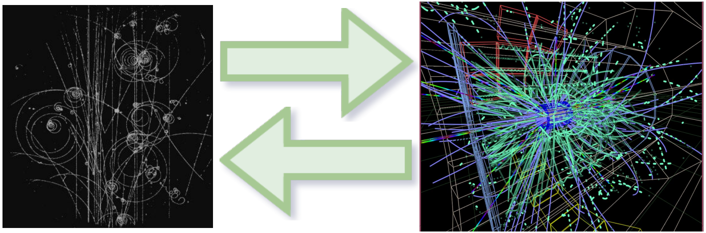


Experimentation

Simulation

Introduction - Application: Quantum Field Theory

- Experimentation and observation needs computation and simulation
- Currently very difficult to simulate processes involving even just ten particles in the final state



Experimentation **Simulation**

Introduction - Quantum Electrodynamics (QED)

N-photon Compton scattering processes: $\mathrm{e}^- + \gamma \rightarrow \mathrm{e}^- + \mathrm{n}\gamma$

Introduction - Quantum Electrodynamics (QED)

N-photon Compton scattering processes: $\mathrm{e}^- + \gamma \rightarrow \mathrm{e}^- + \mathrm{n}\gamma$

Why?

Introduction - Quantum Electrodynamics (QED)

N-photon Compton scattering processes: $\mathrm{e}^- + \gamma \rightarrow \mathrm{e}^- + \mathrm{n}\gamma$

Why?

- Very simple local structure

Introduction - Quantum Electrodynamics (QED)

N-photon Compton scattering processes: $\mathrm{e}^- + \gamma \rightarrow \mathrm{e}^- + \mathrm{n}\gamma$

Why?

- Very simple local structure
- Easy to generate

Introduction - Quantum Electrodynamics (QED)

N-photon Compton scattering processes: $\mathrm{e}^- + \gamma \rightarrow \mathrm{e}^- + \mathrm{n}\gamma$

Why?

- Very simple local structure
- Easy to generate
- Grows very quickly: $\mathcal{O}(\mathrm{n}!)$ **diagrams with** $\mathcal{O}(\mathrm{n})$ **vertices each**

Introduction - Quantum Electrodynamics (QED)

N-photon Compton scattering processes: $\mathrm{e}^- + \gamma \rightarrow \mathrm{e}^- + \mathrm{n}\gamma$

Why?

- Very simple local structure
- Easy to generate
- Grows very quickly: $\mathcal{O}(\mathrm{n}!)$ **diagrams with** $\mathcal{O}(\mathrm{n})$ **vertices each**
- Properties of the resulting processes and DAGs can be verified

Introduction - Quantum Electrodynamics (QED)

N-photon Compton scattering processes: $\mathrm{e}^- + \gamma \rightarrow \mathrm{e}^- + \mathrm{n}\gamma$

Why?

- Very simple local structure
- Easy to generate
- Grows very quickly: $\mathcal{O}(\mathrm{n}!)$ **diagrams with** $\mathcal{O}(\mathrm{n})$ **vertices each**
- Properties of the resulting processes and DAGs can be verified
- Important process for QED

Introduction - Quantum Electrodynamics (QED)

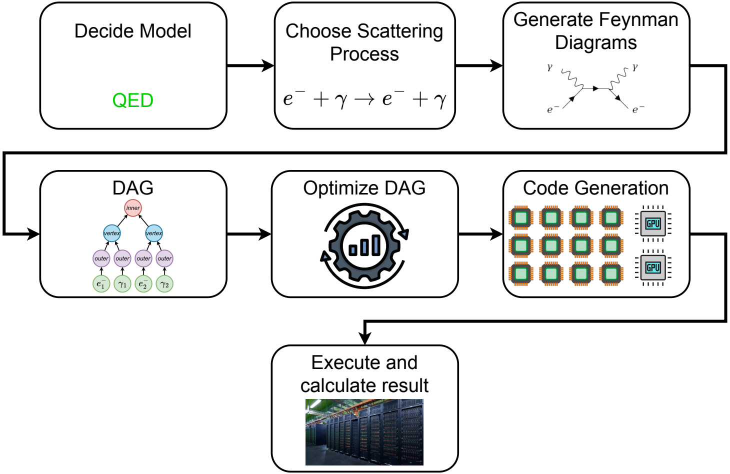
N-photon Compton scattering processes: $\mathrm{e}^- + \gamma \rightarrow \mathrm{e}^- + \mathrm{n}\gamma$

Why?

- Very simple local structure
- Easy to generate
- Grows very quickly: $\mathcal{O}(\mathrm{n}!)$ **diagrams with** $\mathcal{O}(\mathrm{n})$ **vertices each**
- Properties of the resulting processes and DAGs can be verified
- Important process for QED

$\Longrightarrow$ Use $\mathrm{e}^- + \gamma \rightarrow \mathrm{e}^- + \gamma$ as simplest example case.

The Pipeline



Decide Model

QED

Choose Scattering Process

$e^- + \gamma \to e^- + \gamma$

Generate Feynman Diagrams

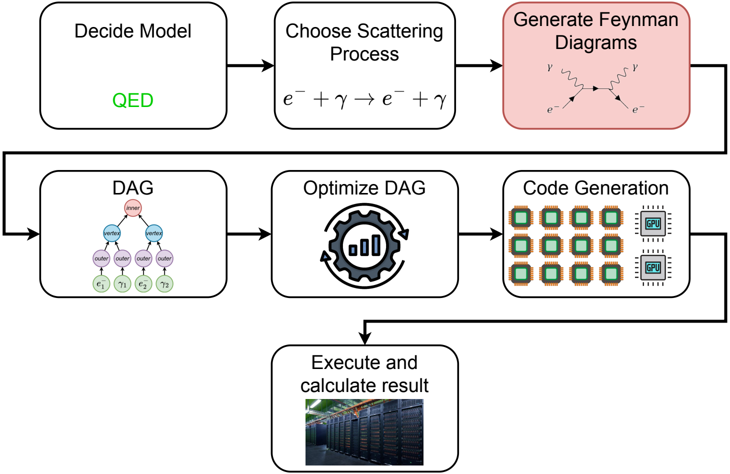
DAG

Optimize DAG

Code Generation

Execute and calculate result

The Pipeline



Decide Model

QED

Choose Scattering Process

$e^- + \gamma \rightarrow e^- + \gamma$

Generate Feynman Diagrams

DAG

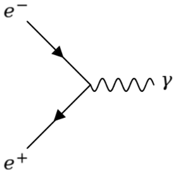
Optimize DAG

Code Generation

Execute and calculate result

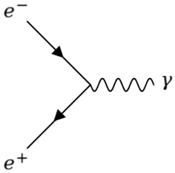
The Pipeline - Finding Feynman Diagrams in QED

One vertex:



The Pipeline - Finding Feynman Diagrams in QED

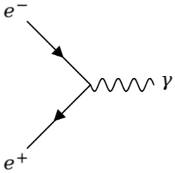
One vertex:



Connect all incoming particles with all outgoing particles in all possible unique ways, using only this vertex!

The Pipeline - Finding Feynman Diagrams in QED

One vertex:

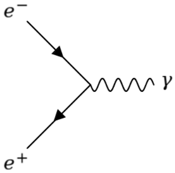


Connect all incoming particles with all outgoing particles in all possible unique ways, using only this vertex!

Example: $\mathrm{e}^- + \gamma \rightarrow \mathrm{e}^- + \gamma$

The Pipeline - Finding Feynman Diagrams in QED

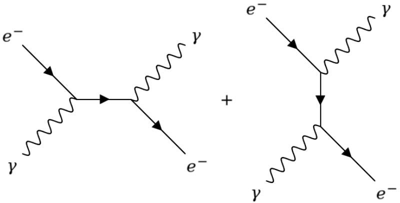
One vertex:



Connect all incoming particles with all outgoing particles in all possible unique ways, using only this vertex!

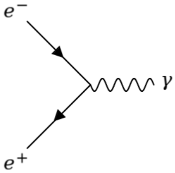
Example: $\mathrm{e}^- + \gamma \rightarrow \mathrm{e}^- + \gamma$

Two Feynman diagrams for this process!



The Pipeline - Finding Feynman Diagrams in QED

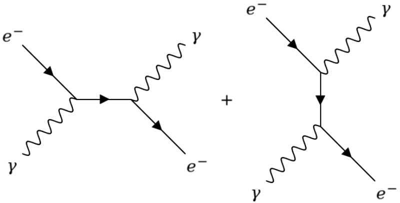
One vertex:



Connect all incoming particles with all outgoing particles in all possible unique ways, using only this vertex!

Example: $\mathrm{e}^- + \gamma \rightarrow \mathrm{e}^- + \gamma$

Two Feynman diagrams for this process!



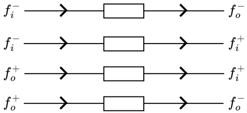
Generally, for scattering processes $\mathrm{e}^- + \gamma \rightarrow \mathrm{e}^- + \mathrm{n}\gamma$, there are $(\mathrm{n} + 1)!$ Feynman diagrams!

The Pipeline - Number of Feynman Diagrams

The Pipeline - Number of Feynman Diagrams

$\mathrm{e}\ldots$ electron-positron pairs, $\mathrm{u}\ldots$ muon-antimuon pairs, $\mathrm{t}\ldots$ tauon-antitauon pairs, $\mathrm{m}\ldots$ photons

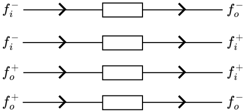
$$\mathrm{N}_{\mathrm{diags}}(\mathrm{e}, \mathrm{u}, \mathrm{t}, \mathrm{m}) =$$



The Pipeline - Number of Feynman Diagrams

$\mathrm{e}\ldots$electron-positron pairs, $\mathrm{u}\ldots$muon-antimuon pairs,
$\mathrm{t}\ldots$tauon-antitauon pairs, $\mathrm{m}\ldots$photons

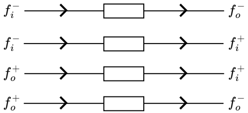
$$\mathrm{N_{diags}}(\mathrm{e}, \mathrm{u}, \mathrm{t}, \mathrm{m}) =$$



The Pipeline - Number of Feynman Diagrams

$\mathrm{e}\ldots$ electron-positron pairs, $\mathrm{u}\ldots$ muon-antimuon pairs, $\mathrm{t}\ldots$ tauon-antitauon pairs, $\mathrm{m}\ldots$ photons

$$\mathrm{N_{diags}}(\mathrm{e},\mathrm{u},\mathrm{t},\mathrm{m}) = \mathrm{e}!\cdot\mathrm{u}!\cdot\mathrm{t}!$$

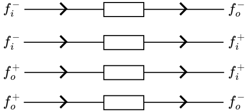


The Pipeline - Number of Feynman Diagrams

$\mathrm{e}\ldots$ electron-positron pairs, $\mathrm{u}\ldots$ muon-antimuon pairs, $\mathrm{t}\ldots$ tauon-antitauon pairs, $\mathrm{m}\ldots$ photons

$$\mathrm{N}_{\mathrm{diags}}(\mathrm{e},\mathrm{u},\mathrm{t},\mathrm{m}) = \mathrm{e}!\cdot\mathrm{u}!\cdot\mathrm{t}!\cdot\frac{(3\mathrm{n}-3)!}{(2\mathrm{n}-1)!}$$

where $\mathrm{n} := \mathrm{e}+\mathrm{u}+\mathrm{t}$

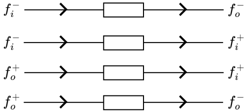


The Pipeline - Number of Feynman Diagrams

$\mathrm{e}\ldots$electron-positron pairs, $\mathrm{u}\ldots$muon-antimuon pairs,
$\mathrm{t}\ldots$tauon-antitauon pairs, $\mathrm{m}\ldots$photons

$$N_{\mathrm{diags}}(\mathrm{e},\mathrm{u},\mathrm{t},\mathrm{m}) = \mathrm{e}! \cdot \mathrm{u}! \cdot \mathrm{t}! \cdot \frac{(3\mathrm{n}-3)!}{(2\mathrm{n}-1)!} \cdot \binom{\mathrm{m}+3\mathrm{n}-3}{3\mathrm{n}-3}$$

where $\mathrm{n} := \mathrm{e}+\mathrm{u}+\mathrm{t}$

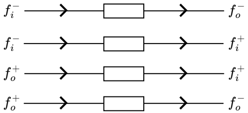


The Pipeline - Number of Feynman Diagrams

$\mathrm{e}\ldots$ electron-positron pairs, $\mathrm{u}\ldots$ muon-antimuon pairs,
$\mathrm{t}\ldots$ tauon-antitauon pairs, $\mathrm{m}\ldots$ photons

$$\mathrm{N}_{\mathrm{diags}}(\mathrm{e},\mathrm{u},\mathrm{t},\mathrm{m}) = \mathrm{e}! \cdot \mathrm{u}! \cdot \mathrm{t}! \cdot \frac{(3\mathrm{n}-3)!}{(2\mathrm{n}-1)!} \cdot \binom{\mathrm{m}+3\mathrm{n}-3}{3\mathrm{n}-3} \cdot \mathrm{m}!$$

where $\mathrm{n} := \mathrm{e}+\mathrm{u}+\mathrm{t}$

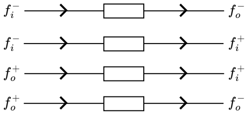


The Pipeline - Number of Feynman Diagrams

$\mathrm{e}\dots$electron-positron pairs, $\mathrm{u}\dots$muon-antimuon pairs,
$\mathrm{t}\dots$tauon-antitauon pairs, $\mathrm{m}\dots$photons

$$
\begin{aligned}
\mathrm{N_{diags}}(\mathrm{e},\mathrm{u},\mathrm{t},\mathrm{m}) &= \mathrm{e}!\cdot\mathrm{u}!\cdot\mathrm{t}!\cdot\frac{(3\mathrm{n}-3)!}{(2\mathrm{n}-1)!}\cdot\binom{\mathrm{m}+3\mathrm{n}-3}{3\mathrm{n}-3}\cdot\mathrm{m}! \\
&= \frac{(\mathrm{m}+3\mathrm{n}-3)!}{(2\mathrm{n}-1)!}\cdot\mathrm{e}!\cdot\mathrm{u}!\cdot\mathrm{t}!
\end{aligned}
$$

where $\mathrm{n} := \mathrm{e}+\mathrm{u}+\mathrm{t}$



The Pipeline - Number of Feynman Diagrams

$$N_{\text{diags}}(e, u, t, m) = \frac{(m + 3n - 3)!}{(2n - 1)!} \cdot e! \cdot u! \cdot t!$$

n	e	u	t	0	1	2	3	4	5	6
1	1	0	0	1	1	2	6	24	120	720
2	1	1	0	1	4	20	120	840	6 720	60 480
2	2	0	0	2	8	40	240	1 680	13 440	120 960
3	1	1	1	6	42	336	3 024	30 240	332 640	3 991 680
3	2	1	0	12	84	672	6 048	60 480	665 280	7 983 360
3	3	0	0	36	252	2 016	18 144	181 440	1 995 840	23 950 080
4	2	1	1	144	1 440	15 840	190 080	2 471 040	34 594 560	518 918 400
4	2	2	0	288	2 880	31 680	380 160	4 942 080	69 189 120	1 037 836 800
4	3	1	0	432	4 320	47 520	570 240	7 413 120	103 783 680	1 556 755 200

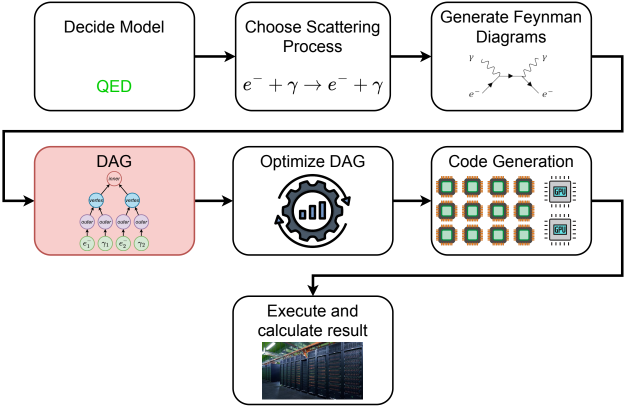
The Pipeline - Number of Feynman Diagrams

$$N_{\text{diags}}(e, u, t, m) = \frac{(m + 3n - 3)!}{(2n - 1)!} \cdot e! \cdot u! \cdot t!$$

n	e	u	t	0	1	2	3	4	5	6
1	1	0	0	1	1	2	6	24	120	720
2	1	1	0	1	4	20	120	840	6 720	60 480
2	2	0	0	2	8	40	240	1 680	13 440	120 960
3	1	1	1	6	42	336	3 024	30 240	332 640	3 991 680
3	2	1	0	12	84	672	6 048	60 480	665 280	7 983 360
3	3	0	0	36	252	2 016	18 144	181 440	1 995 840	23 950 080
4	2	1	1	144	1 440	15 840	190 080	2 471 040	34 594 560	518 918 400
4	2	2	0	288	2 880	31 680	380 160	4 942 080	69 189 120	1 037 836 800
4	3	1	0	432	4 320	47 520	570 240	7 413 120	103 783 680	1 556 755 200

The Pipeline - Feynman Diagram Translation

How do we get from the Feynman diagrams to a DAG?



Decide Model

QED

Choose Scattering Process

$e^- + \gamma \to e^- + \gamma$

Generate Feynman Diagrams

DAG

Optimize DAG

Code Generation

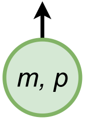
Execute and calculate result

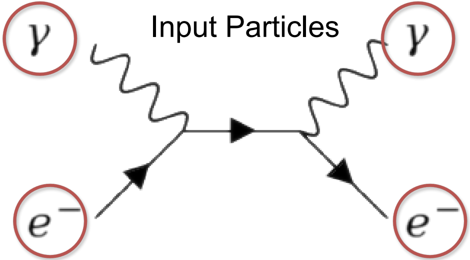
The Pipeline - Feynman Diagram Translation

How do we get from the Feynman diagrams to a DAG?

Input particles:

- Four-momentum $p = (p_0, p_1, p_2, p_3)$
- Particle mass m



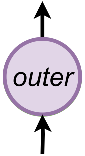


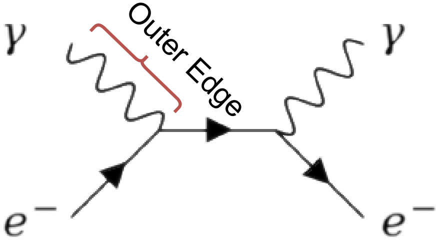
The Pipeline - Feynman Diagram Translation

How do we get from the Feynman diagrams to a DAG?

Outer edges (particle state):

- $\mathrm{u}(\mathrm{p})$, $\overline{\mathrm{u}}(\mathrm{p})$, $\mathrm{v}(\mathrm{p})$, $\overline{\mathrm{v}}(\mathrm{p})$, ε_μ, $\varepsilon_\mu^\star$
- Carry particle state along



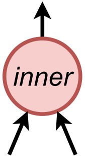


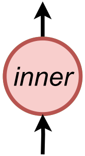
The Pipeline - Feynman Diagram Translation

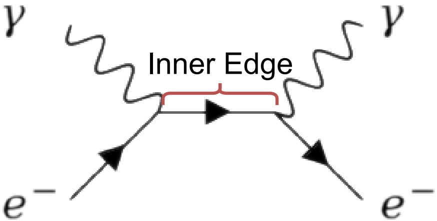
How do we get from the Feynman diagrams to a DAG?

Inner edges (particle propagator):

- $\frac{\mathrm{im}}{\mathrm{p}^2 - \mathrm{m}^2 + \mathrm{i}\varepsilon}$, $\frac{\mathrm{i}\mathrm{g}_{\mu\nu}}{\mathrm{q}^2 + \mathrm{i}\varepsilon}$
- Carry particle state along





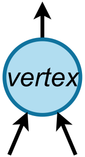


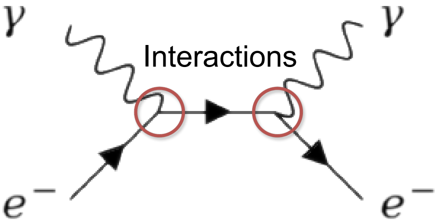
The Pipeline - Feynman Diagram Translation

How do we get from the Feynman diagrams to a DAG?

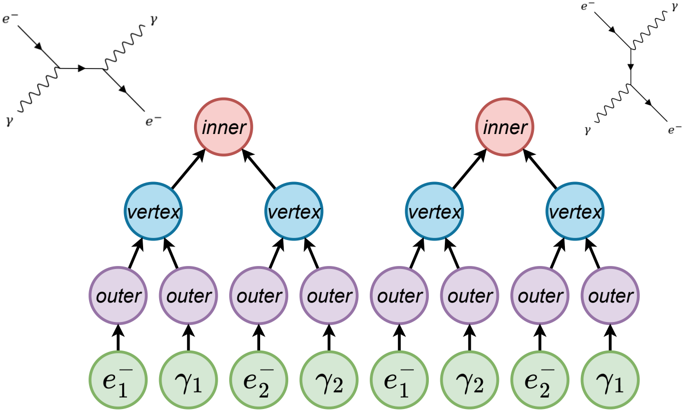
Vertices:

- $-\mathrm{ie}\gamma^{\mu}$
- Use conservation of momentum to get new particle state

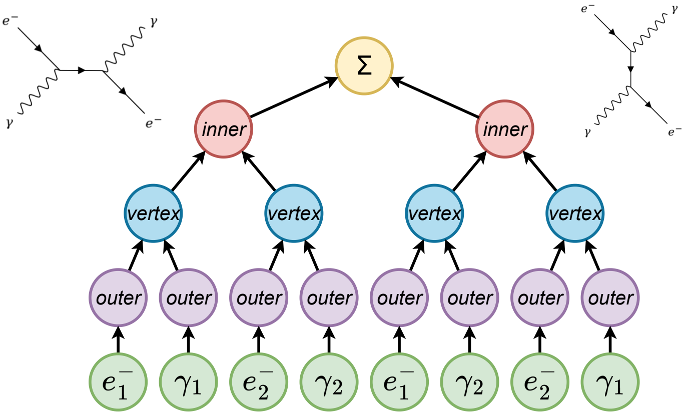




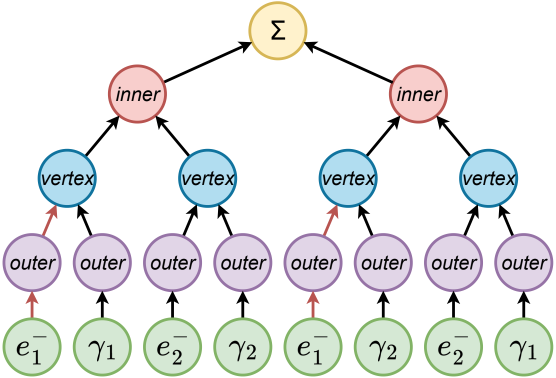
The Pipeline - The (Naive) Directed Acyclic Graph (**DAG**)



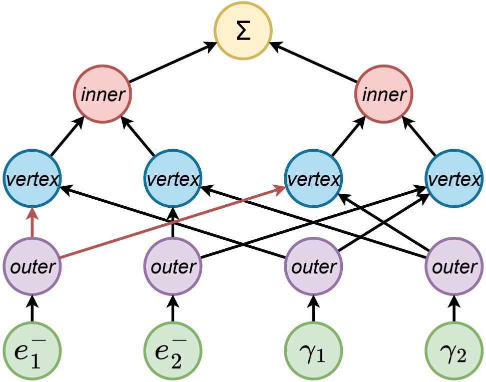
The Pipeline - The (Naive) Directed Acyclic Graph (**DAG**)



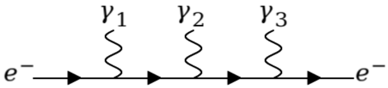
The Pipeline - The (Naive) DAG, Reduced



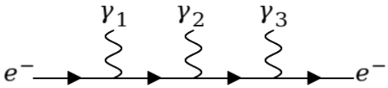
The Pipeline - The (Naive) DAG, Reduced



Intermission - Complexity of N-Photon Compton Processes

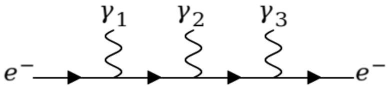


Intermission - Complexity of N-Photon Compton Processes

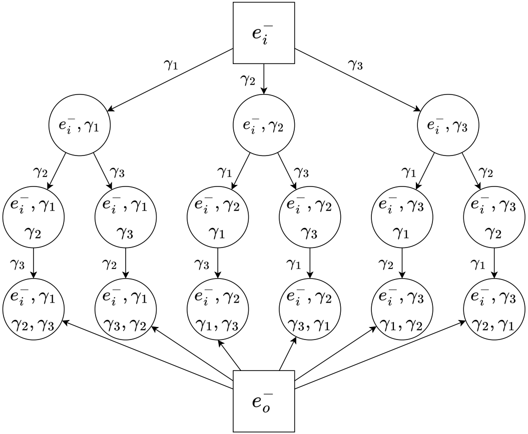


6 possible diagrams, each with 10 parts
→ 60 compute nodes?

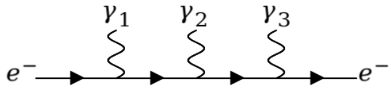
Intermission - Complexity of N-Photon Compton Processes



6 possible diagrams, each with 10 parts
→ 60 compute nodes?

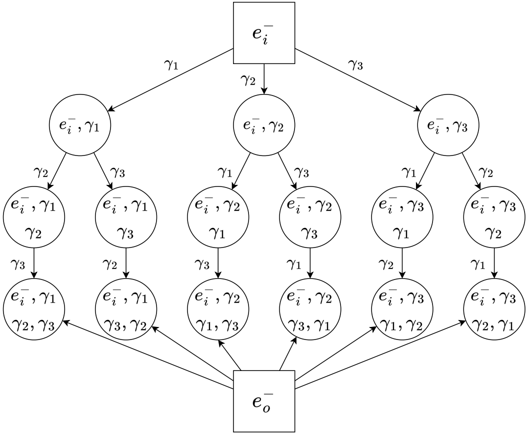

e_i^-
γ_1
γ_2
γ_3
e_i^-, γ_1
e_i^-, γ_2
e_i^-, γ_3
e_i^-, γ_1 γ_2
e_i^-, γ_1 γ_3
e_i^-, γ_2 γ_1
e_i^-, γ_2 γ_3
e_i^-, γ_3 γ_1
e_i^-, γ_3 γ_2
e_i^-, γ_1 γ_2, γ_3
e_i^-, γ_1 γ_3, γ_2
e_i^-, γ_2 γ_1, γ_3
e_i^-, γ_2 γ_3, γ_1
e_i^-, γ_3 γ_1, γ_2
e_i^-, γ_3 γ_2, γ_1
e_o^-

Intermission - Complexity of N-Photon Compton Processes

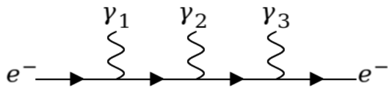


6 possible diagrams, each with 10 parts
→ 60 compute nodes?

Reusing diagram parts reduces the complexity!

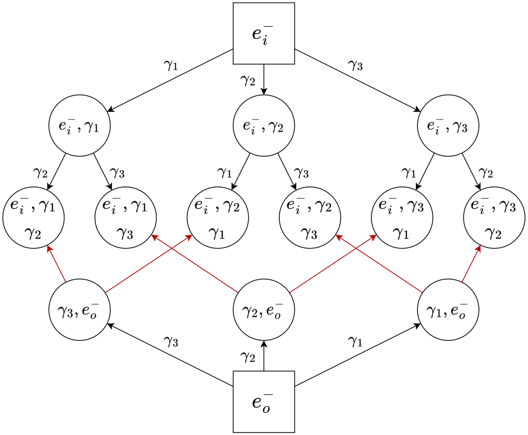

e_i^-
γ_1
γ_2
γ_3
e_i^-, γ_1
e_i^-, γ_2
e_i^-, γ_3
γ_2
γ_3
γ_1
γ_3
γ_1
γ_2
e_i^-, γ_1 γ_2
e_i^-, γ_1 γ_3
e_i^-, γ_2 γ_1
e_i^-, γ_2 γ_3
e_i^-, γ_3 γ_1
e_i^-, γ_3 γ_2
γ_3
γ_2
γ_3
γ_1
γ_2
γ_1
e_i^-, γ_1 γ_2, γ_3
e_i^-, γ_1 γ_3, γ_2
e_i^-, γ_2 γ_1, γ_3
e_i^-, γ_2 γ_3, γ_1
e_i^-, γ_3 γ_1, γ_2
e_i^-, γ_3 γ_2, γ_1
e_o^-

Intermission - Complexity of N-Photon Compton Processes

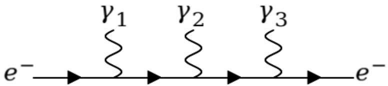


6 possible diagrams, each with 10 parts
→ 60 compute nodes?

Reusing diagram parts reduces the complexity!

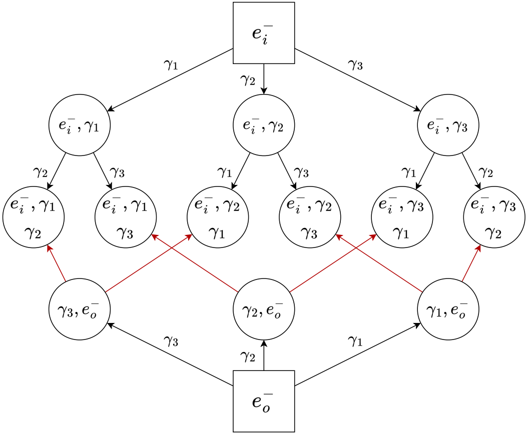


Intermission - Complexity of N-Photon Compton Processes

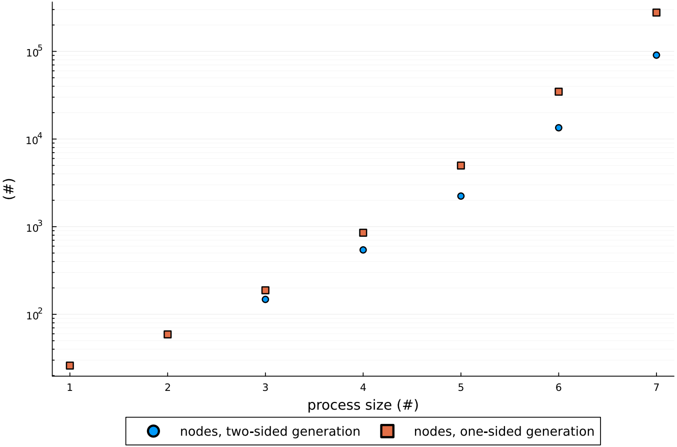


6 possible diagrams, each with 10 parts
→ 60 compute nodes?

Reusing diagram parts reduces the complexity!
Reusing parts from both sides is even better!

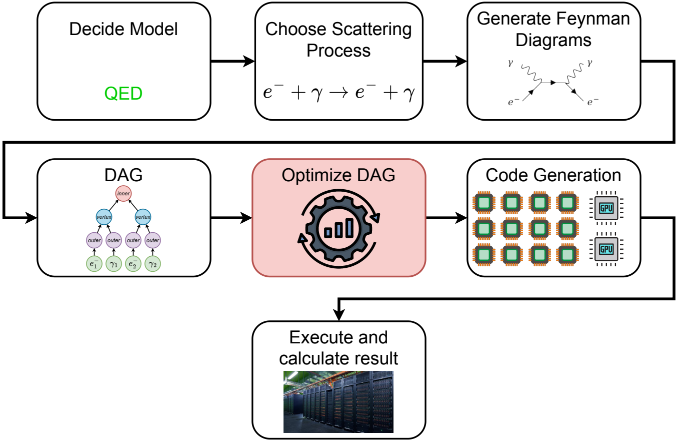


Intermission - Complexity of N-Photon Compton Processes



- $\mathcal{O}(\frac{n!}{\frac{n}{2}!})$ versus $\mathcal{O}(n!)$

The Pipeline - DAG Optimization Operations



Decide Model

QED

Choose Scattering Process

$e^- + \gamma \rightarrow e^- + \gamma$

Generate Feynman Diagrams

DAG

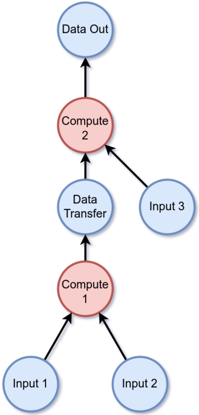
Optimize DAG

Code Generation

Execute and calculate result

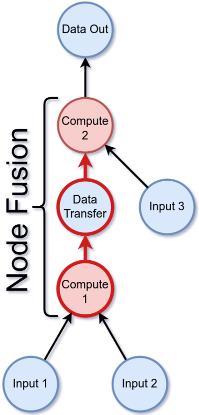
The Pipeline - DAG Optimization Operations

Node Fusion



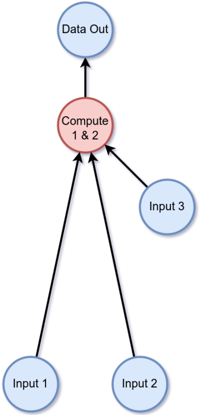
The Pipeline - DAG Optimization Operations

Node Fusion



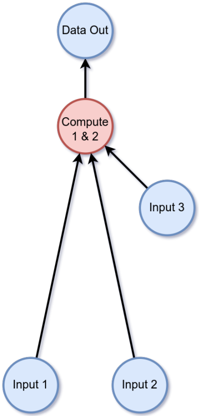
The Pipeline - DAG Optimization Operations

Node Fusion

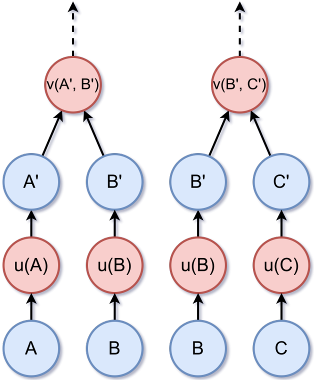


The Pipeline - DAG Optimization Operations

Node Fusion

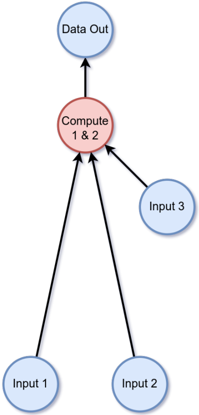


Node Reduction

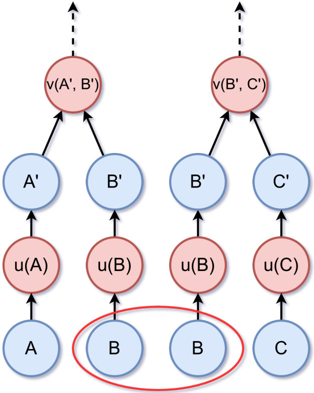


The Pipeline - DAG Optimization Operations

Node Fusion



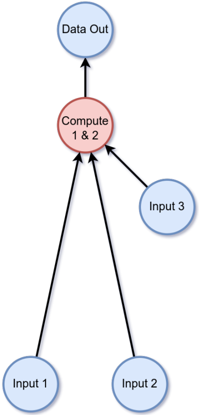
Node Reduction



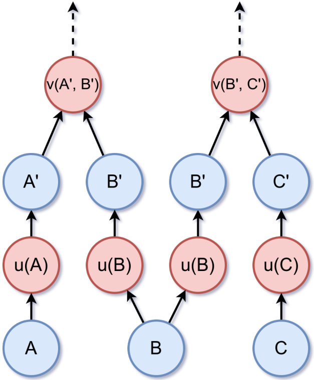
Node Reduction

The Pipeline - DAG Optimization Operations

Node Fusion

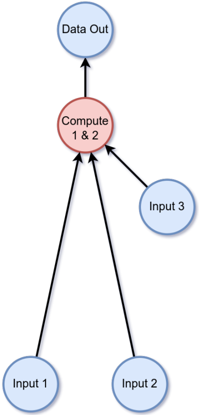


Node Reduction

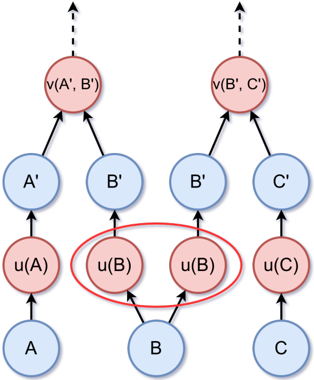


The Pipeline - DAG Optimization Operations

Node Fusion



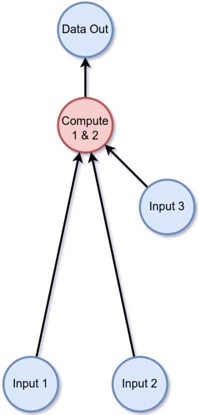
Node Reduction



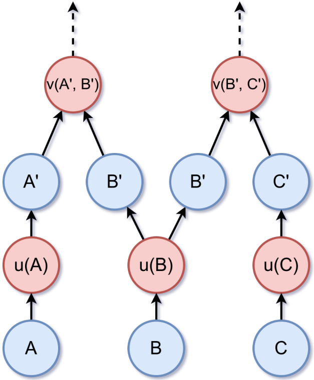
Node Reduction

The Pipeline - DAG Optimization Operations

Node Fusion

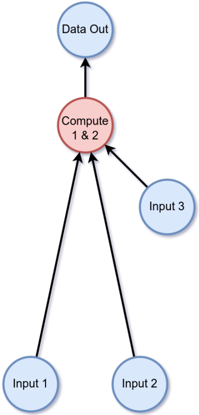


Node Reduction

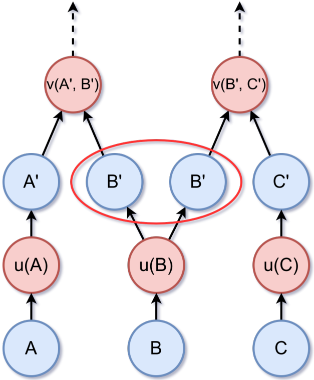


The Pipeline - DAG Optimization Operations

Node Fusion



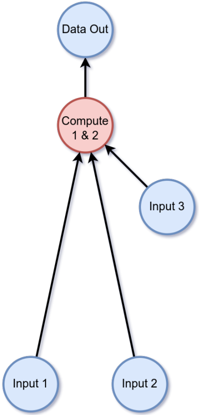
Node Reduction



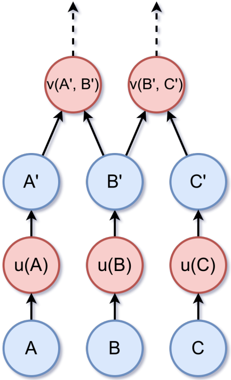
Node Reduction

The Pipeline - DAG Optimization Operations

Node Fusion

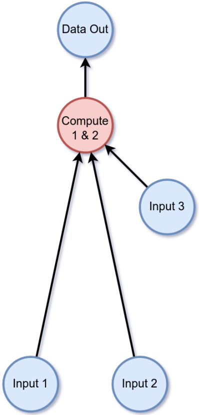


Node Reduction

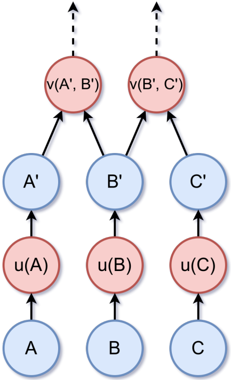


The Pipeline - DAG Optimization Operations

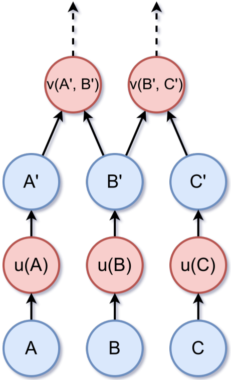
Node Fusion



Node Reduction

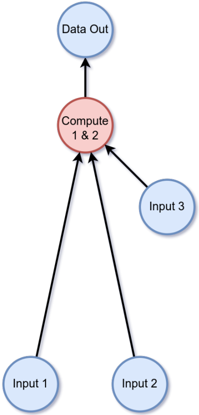


Node Split

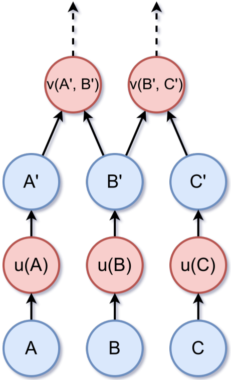


The Pipeline - DAG Optimization Operations

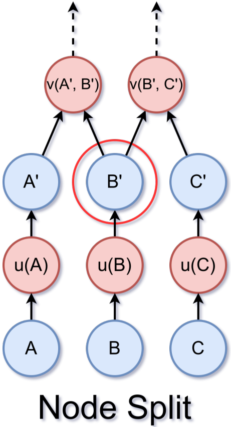
Node Fusion



Node Reduction

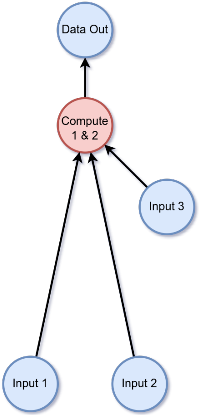


Node Split

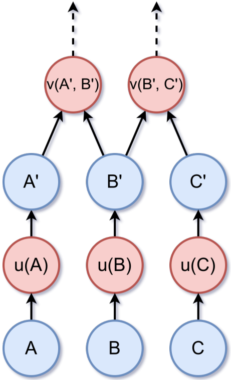


The Pipeline - DAG Optimization Operations

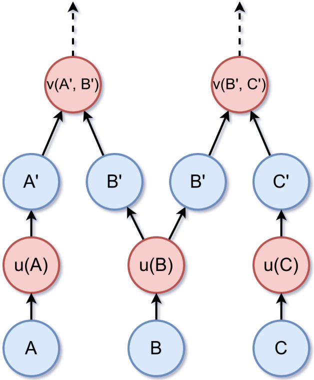
Node Fusion



Node Reduction

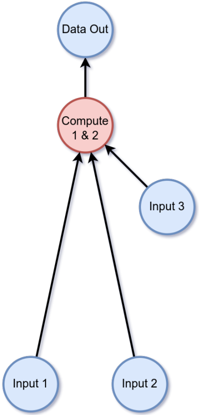


Node Split

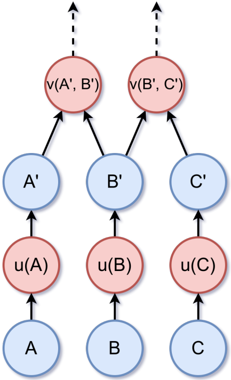


The Pipeline - DAG Optimization Operations

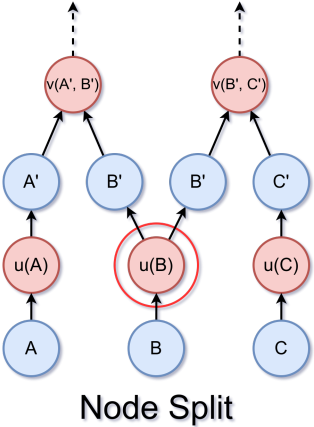
Node Fusion



Node Reduction

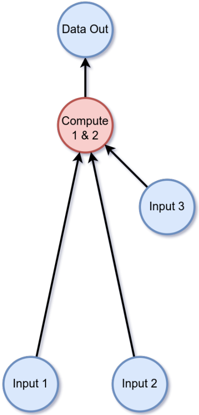


Node Split

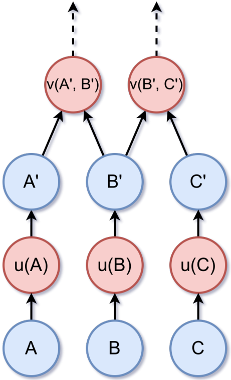


The Pipeline - DAG Optimization Operations

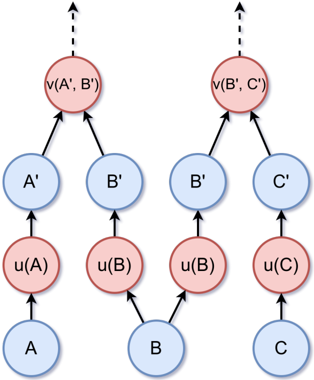
Node Fusion



Node Reduction

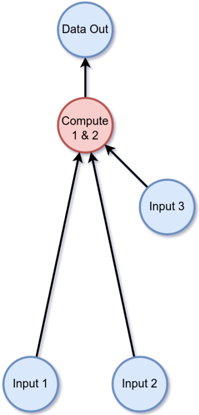


Node Split

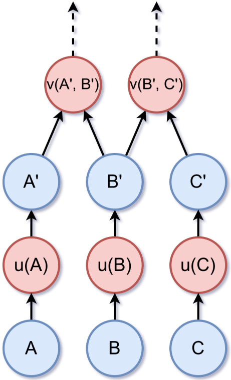


The Pipeline - DAG Optimization Operations

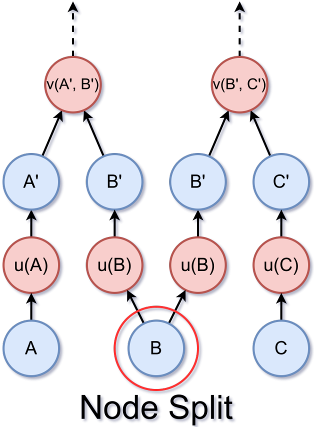
Node Fusion



Node Reduction

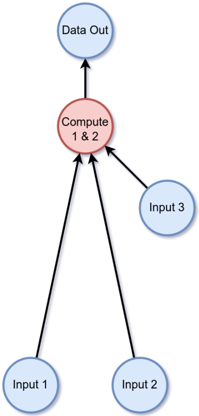


Node Split

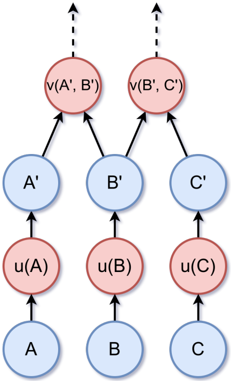


The Pipeline - DAG Optimization Operations

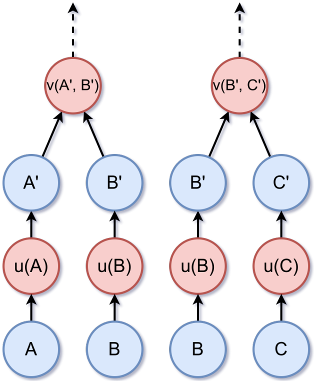
Node Fusion



Node Reduction

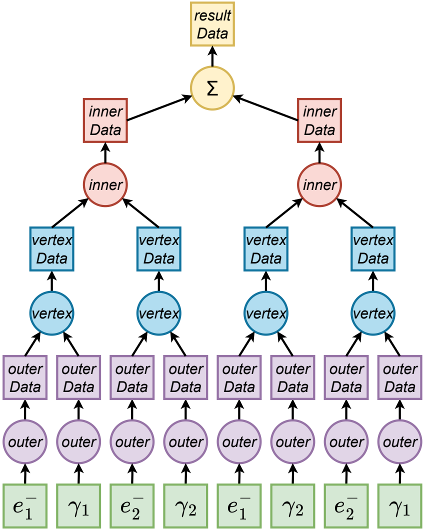


Node Split



The Pipeline - DAG Optimization

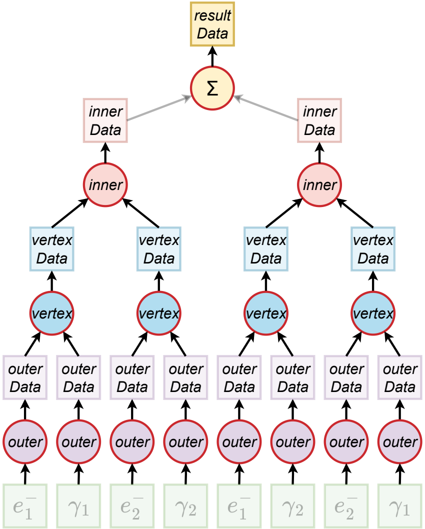
What are we optimizing?



The Pipeline - DAG Optimization

What are we optimizing?

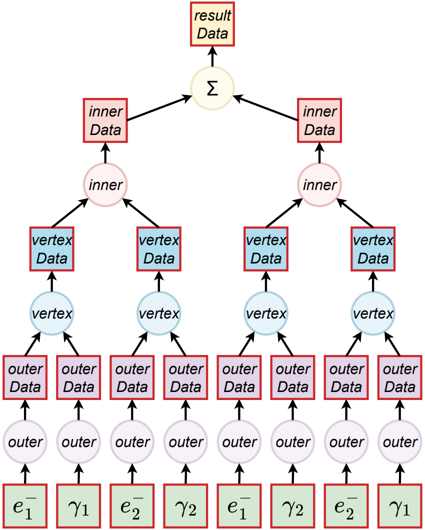
- Compute Effort



The Pipeline - DAG Optimization

What are we optimizing?

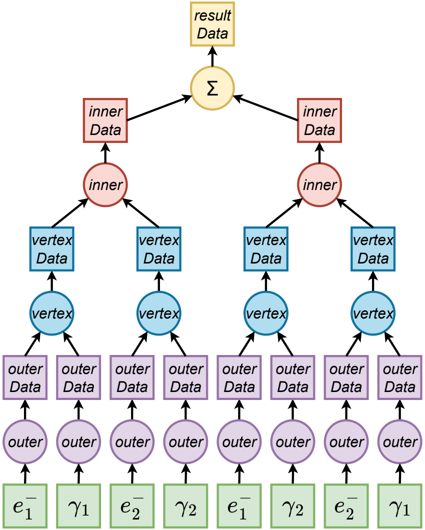
- Compute Effort
- Data Transfer



The Pipeline - DAG Optimization

What are we optimizing?

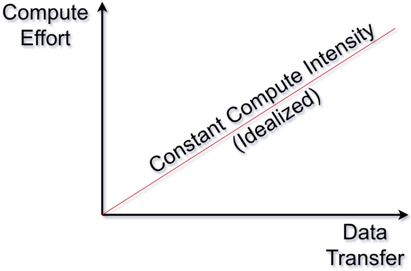
- Compute Effort
- Data Transfer
- Compute Intensity $= \frac{\text{Compute Effort}}{\text{Data Transfer}}$

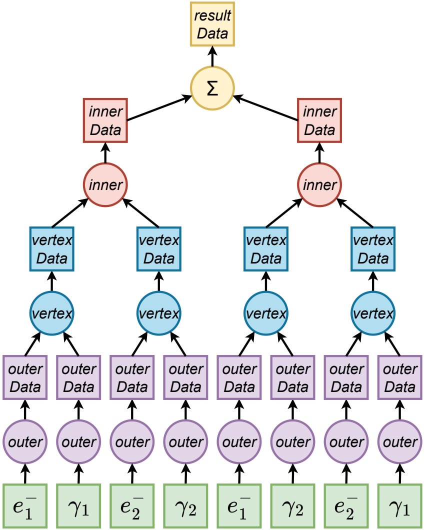


The Pipeline - DAG Optimization

What are we optimizing?

- Compute Effort
- Data Transfer
- Compute Intensity $= \frac{\text{Compute Effort}}{\text{Data Transfer}}$

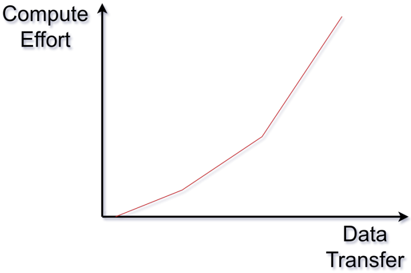


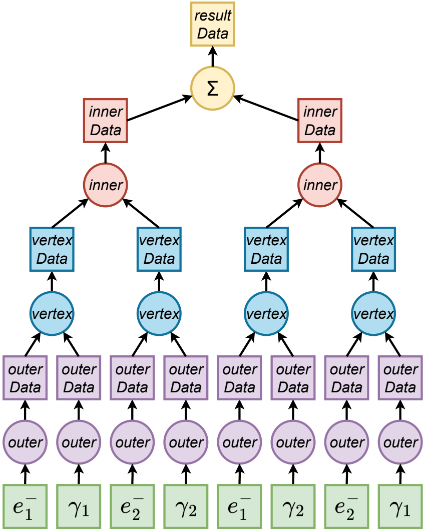


The Pipeline - DAG Optimization

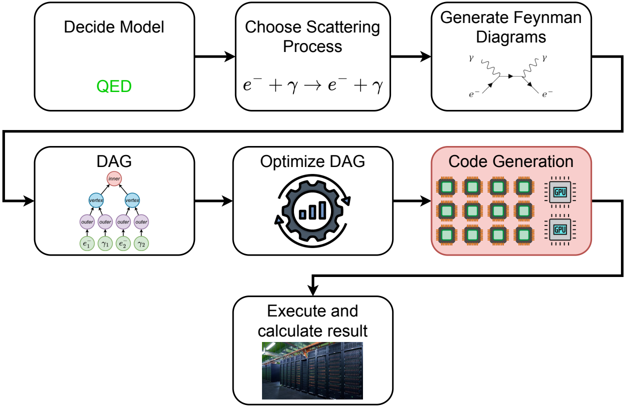
What are we optimizing?

- Compute Effort
- Data Transfer
- Compute Intensity $= \frac{\text{Compute Effort}}{\text{Data Transfer}}$





The Pipeline - Code Generation



Decide Model

QED

Choose Scattering Process

$e^- + \gamma \to e^- + \gamma$

Generate Feynman Diagrams

DAG

Optimize DAG

Code Generation

Execute and calculate result

The Pipeline - Code Generation

DAG

- Get graph, a scheduler, and machine information

The Pipeline - Code Generation

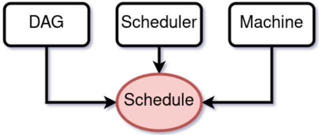
- Get graph, a scheduler, and machine information

The Pipeline - Code Generation

DAG Scheduler Machine

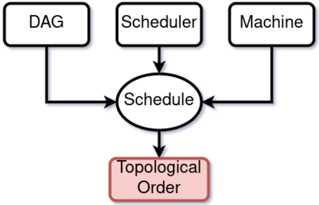
- Get graph, a scheduler, and machine information

The Pipeline - Code Generation



- Get graph, a scheduler, and machine information
- Use scheduler interface

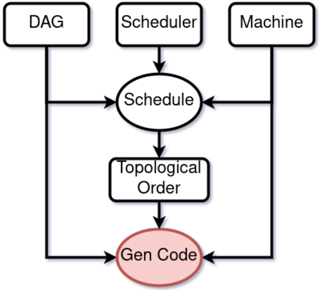
The Pipeline - Code Generation



- Get graph, a scheduler, and machine information
- Use scheduler interface to create a topological ordering of tasks for each device

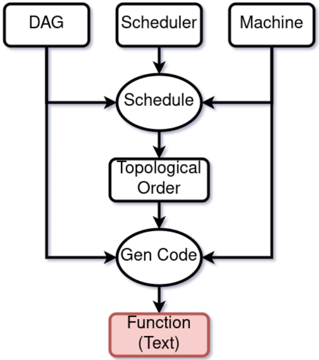
The Pipeline - Code Generation

- Get graph, a scheduler, and machine information
- Use scheduler interface to create a topological ordering of tasks for each device
- For each task in the ordering, generate code using the scheduled device



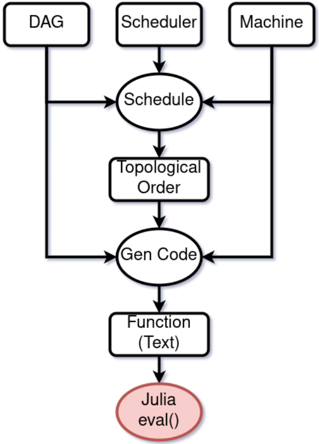
The Pipeline - Code Generation

- Get graph, a scheduler, and machine information
- Use scheduler interface to create a topological ordering of tasks for each device
- For each task in the ordering, generate code using the scheduled device
- Evaluate the function code



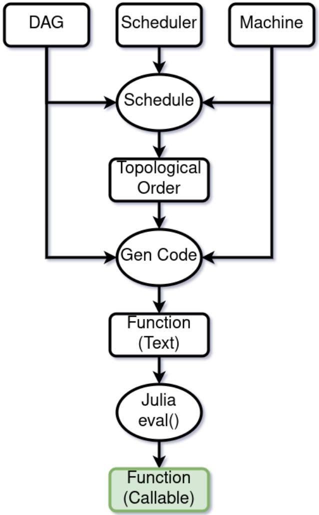
The Pipeline - Code Generation

- Get graph, a scheduler, and machine information
- Use scheduler interface to create a topological ordering of tasks for each device
- For each task in the ordering, generate code using the scheduled device
- Evaluate the function code into a function

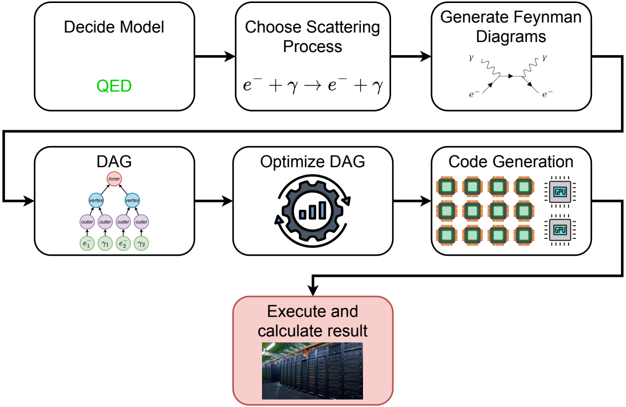


The Pipeline - Code Generation

- Get graph, a scheduler, and machine information
- Use scheduler interface to create a topological ordering of tasks for each device
- For each task in the ordering, generate code using the scheduled device
- Evaluate the function code into a function



The Pipeline - Execute



Decide Model

QED

Choose Scattering Process

$e^- + \gamma \rightarrow e^- + \gamma$

Generate Feynman Diagrams

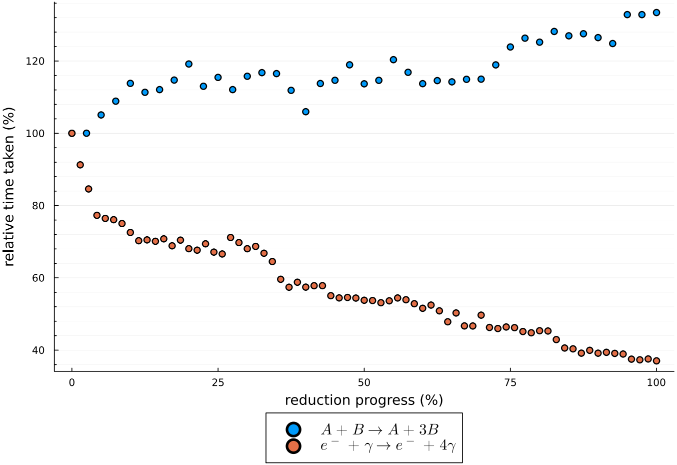
DAG

Optimize DAG

Code Generation

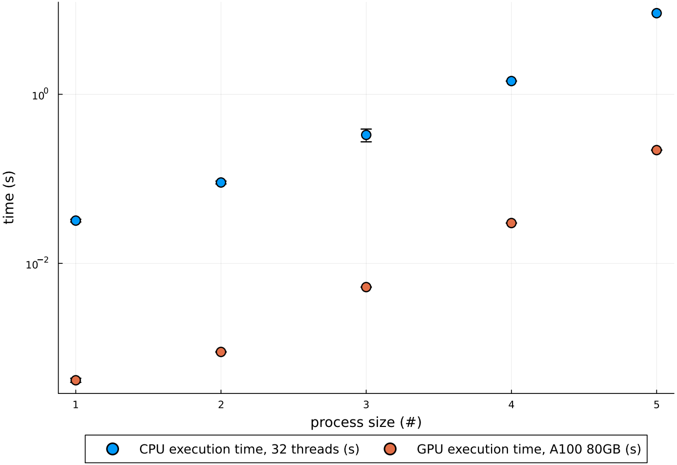
Execute and calculate result

The Pipeline - Reduction Effects on ABC vs QED



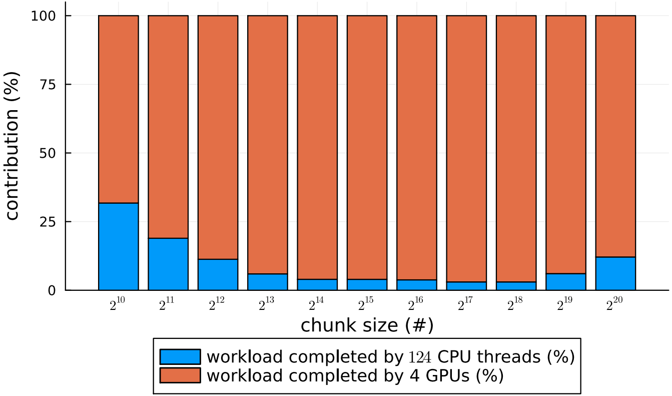
- ABC-Model is structurally like QED but with smaller tasks
- Execution on CPU
- Showing relative time taken compared to unreduced graph (lower is better)

The Pipeline - QED Performance CPU vs GPU



- Time taken for execution of $2^{20} \approx 1$ million samples
- CPU: 32 cores of AMD EPYC™ 7763
- GPU: 1 Nvidia Tesla A100 SXM4

The Pipeline - QED Performance Heterogeneous Execution



- Execution of $2^{30} \approx 1$ billion samples for 5-photon Compton
- CPU: 124 cores of AMD EPYC™ 7763
- GPU: 4 Nvidia Tesla A100 SXM4
- Sample distribution onto available hardware chunks of various sizes

Summary

We can

- represent the necessary computation to evaluate Feynman diagrams as DAGs.
- provide search space for optimizers through node operations.
- generate efficient code and dynamically compile and run it on multiple target devices.

Summary

We can

- represent the necessary computation to evaluate Feynman diagrams as DAGs.
- provide search space for optimizers through node operations.
- generate efficient code and dynamically compile and run it on multiple target devices.

Findings:

- The complexity of the calculations for QED processes depends on the diagram generation method.
- Optimizers can help the compiler, but building block size matters.
- Little unexpected overhead is introduced by Julia's GPU libraries.

Future Work

- Include GPUs in the scheduling of DAGs
- Compare different optimization algorithms and cost functions
- Determine a machine's scaling functions and working point graph using microbenchmarks
- More types of node operations: node vectorization and term rewriting
- Extend theory improvements and diagram counting to other Quantum Field Theories
- Apply to other promising fields outside of particle physics

Acknowledgements

Supervisor: Dr. Uwe Hernandez Acosta[3,4]
Supervising Professor: Prof. Dr.-Ing. Jerónimo Castrillón[1]
Supervising Professor: Prof. Dr. Thomas D. Kühne[2,3]
Thanks: Simeon Ehrig[3,4] **& René Widera**[4]

[1] Chair for Compiler Construction, TU Dresden
[2] Professorship for Computational Systems Science, TU Dresden
[3] Center for Advanced Systems Understanding (CASUS)
[4] Helmholtz-Zentrum Dresden-Rossendorf (HZDR)

References

[CH05] John Clark and Derek Allan Holton. *A first look at graph theory*. Reprint. Hackensack [u.a.]: World Scientific, 2005. ISBN: 9789810204891. URL: `http://slubdd.de/katalog?TN_libero_mab2`.

[HW79] John A Hartigan and Manchek A Wong. „Algorithm AS 136: A k-means clustering algorithm". In: *Journal of the royal statistical society. series c (applied statistics)* 28.1 (1979), pp. 100–108.

[BFD18] Tim Besard, Christophe Foket, and Bjorn De Sutter. „Effective extensible programming: unleashing Julia on GPUs". In: *IEEE Transactions on Parallel and Distributed Systems* 30.4 (2018), pp. 827–841.

[Kar+12] Stefan Karpinski et al. *Why we created Julia*. Feb. 2012. URL: `https://julialang.org/blog/2012/02/why-we-created-julia/`.

[Luo+21] Jinhong Luo et al. „Learning to optimize DAG scheduling in heterogeneous environment". In: *arXiv:2103.06980* (2021).

[Val+21] Andrea Valassi et al. „Design and engineering of a simplified workflow execution for the MG5aMC event generator on GPUs and vector CPUs". In: *EPJ Web of Conferences*. Vol. 251. EDP Sciences. 2021, p. 03045.

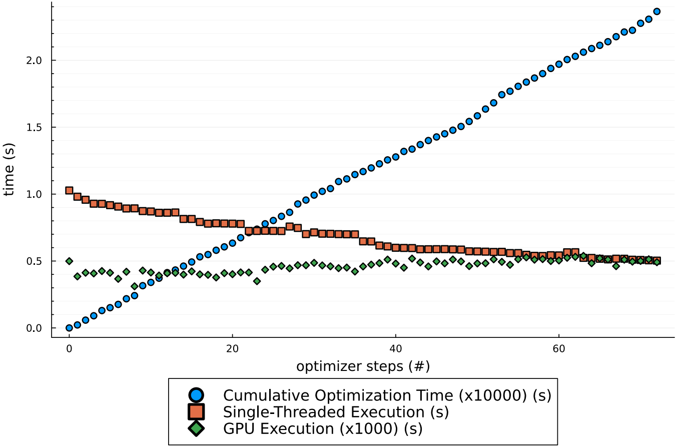
[Chu+22] Valentin Churavy et al. „Bridging HPC Communities through the Julia Programming Language". In: *arXiv:2211.02740* (2022).

[Gri08] David Griffiths. *Introduction to elementary particles*. 2., revised edition. Wiley-VCH Verlag GmbH & Co. KGaA, 2008.

[Her+24] Uwe Hernandez Acosta et al. *QED project: QEDbase.jl (v0.1.6) and QEDprocesses.jl (v0.1.0)*. Feb. 2024. URL: `https://github.com/QEDjl-project`.

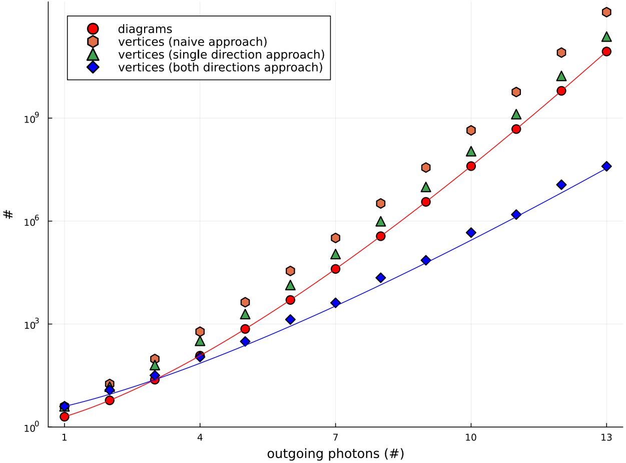
[KN28] Oskar Klein and Yoshio Nishina. „The scattering of light by free electrons according to Dirac's new relativistic dynamics". In: *Nature* 122.3072 (1928), pp. 398–399.

Backup - Analysis vs Execution Speed



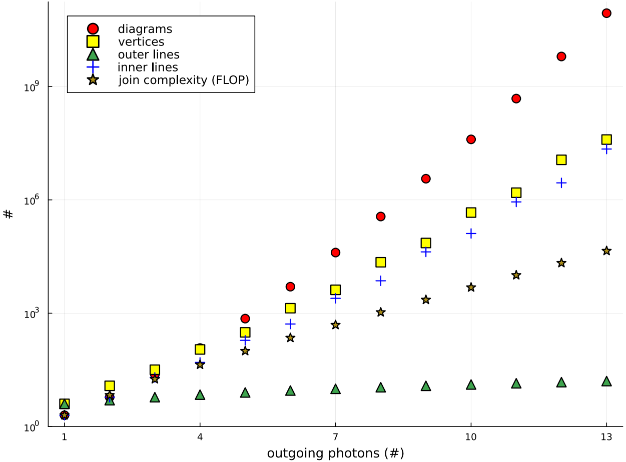
- Cumulative time taken to optimize (reduction) versus execution time at state
- Note the factors

Backup - Vertex Amounts



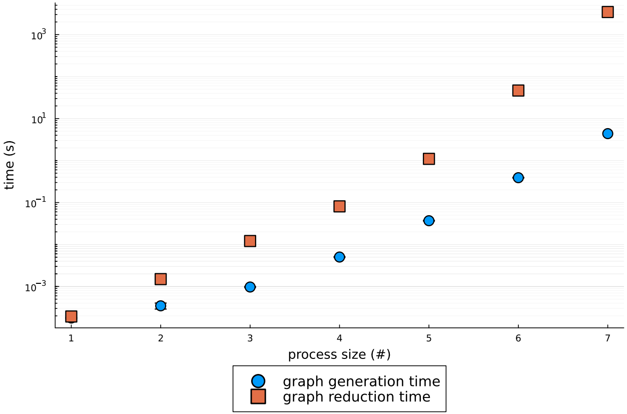
- Numbers of diagrams and vertices in generated DAGs for given n-photon Compton processes
- $(n+1)!$ is shown in red
- $2\frac{(n+1)!}{\frac{n+1}{2}!}$ is shown in blue

Backup - Optimal Complexity with Binomial Join Nodes



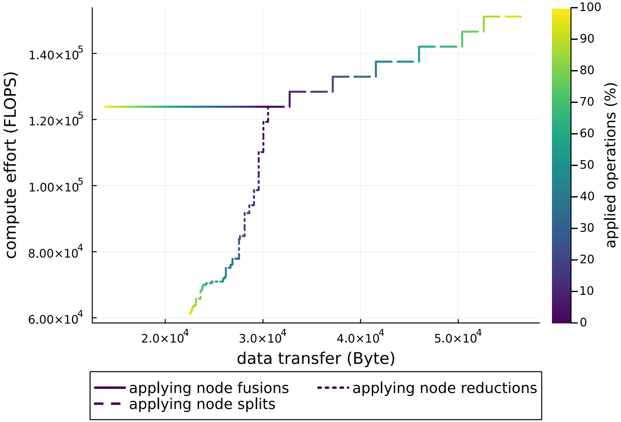
- The number of diagrams eventually scales faster than the number of nodes

Backup - DAG Generation Times



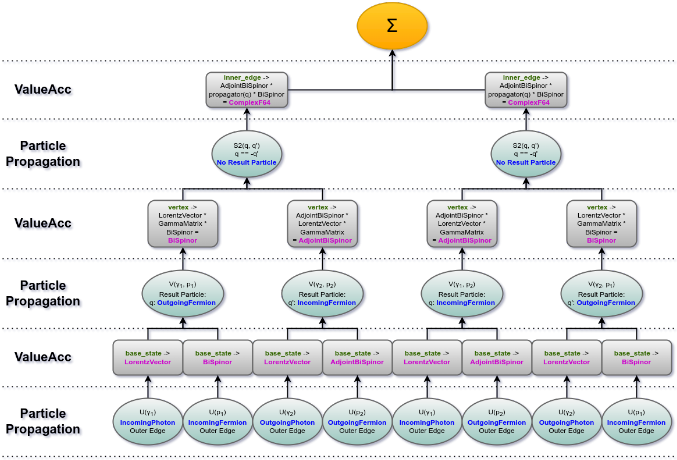
- Time to generate the DAG for given n-photon Compton processes

Backup - Optimizer Effects on Compute Intensity



- Compute Effort and Data Transfer for a 3-photon Compton process
- As operations are applied, the compute intensity changes

Backup - Data Types in the DAG



Σ

ValueAcc

inner_edge -> AdjointBiSpinor * propagator(q) * BiSpinor = ComplexF64

inner_edge -> AdjointBiSpinor * propagator(q) * BiSpinor = ComplexF64

Particle Propagation

S2(q, q') q == -q' No Result Particle

S2(q, q') q == -q' No Result Particle

ValueAcc

vertex -> LorentzVector * GammaMatrix * BiSpinor = BiSpinor

vertex -> AdjointBiSpinor * LorentzVector * GammaMatrix = AdjointBiSpinor

vertex -> AdjointBiSpinor * LorentzVector * GammaMatrix = AdjointBiSpinor

vertex -> LorentzVector * GammaMatrix * BiSpinor = BiSpinor

Particle Propagation

$V(\gamma_1, p_1)$ Result Particle: q: OutgoingFermion

$V(\gamma_2, p_2)$ Result Particle: q': IncomingFermion

$V(\gamma_1, p_2)$ Result Particle: q: IncomingFermion

$V(\gamma_2, p_1)$ Result Particle: q': OutgoingFermion

ValueAcc

base_state -> LorentzVector

base_state -> BiSpinor

base_state -> LorentzVector

base_state -> AdjointBiSpinor

base_state -> LorentzVector

base_state -> AdjointBiSpinor

base_state -> LorentzVector

base_state -> BiSpinor

Particle Propagation

$U(\gamma_1)$ IncomingPhoton Outer Edge

$U(p_1)$ IncomingFermion Outer Edge

$U(\gamma_2)$ OutgoingPhoton Outer Edge

$U(p_2)$ OutgoingFermion Outer Edge

$U(\gamma_1)$ IncomingPhoton Outer Edge

$U(p_2)$ OutgoingFermion Outer Edge

$U(\gamma_2)$ OutgoingPhoton Outer Edge

$U(p_1)$ IncomingFermion Outer Edge

- Data types change throughout the graph
- The result is a complex number