UNIVERSITÄT SIEGEN

TP1

Adaptive Multi-Channel Integration with MPI

wihtin WHIZARD – arXiv:1811.09711

Simon Braß

November 27, 2018

Universität Siegen – Theoretische Physik I
http://www.tp.nt.uni-siegen.de/+brass/

Adaptive Multi-Channel Monte Carlo

Typical Monte Carlo Event Generator

- Computation of total cross-section
  - → *Monte Carlo* integration
- Generation of physical events
  - → *Monte Carlo* unweighting algorithm
- Acceptance-rejection method
  - Accept event with weight w on $\frac{w}{w_{\text{max}}} \leq r, r \in (0,1)$
  - unweighted, physical event.
- Unweighting efficieny $\epsilon = \frac{\langle w \rangle_N}{w_{\text{max}}}$.

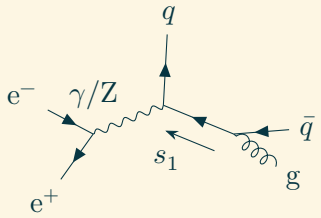
Computation time

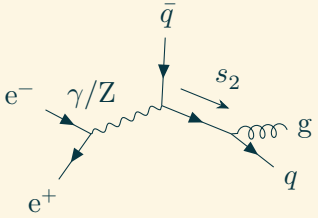
- Integration *converges* with $\propto \frac{1}{\sqrt{N}}$ and *efficieny* $\epsilon < 1 \times 10^{-2}\,\%$
  - → increase sampling number N increasing computation time
  - → adaptive method, e.g. multi-channel, stratified or importance sampling
- Parallelization → need to *completely* understand the algorithms
  - → Reduce computation time
  - → Utilize computing hardware more efficiently

Multi-channel integration

- Matrix elements have different analytical structures and interference terms
  → resonances, (logarithmic-enhanced) final and/or initial state radiation, ...
- Phasespace parametrization of physical manifold Ω on the unit hypercube U_i
  → channel $\phi_i : U_i \mapsto \Omega$
- *Partition of unity* $f(x) = \sum_i \alpha_i f(x)$ and $\sum_i \alpha_i = 1$ [Kleiss and Pittau 1994]

$$I = \int_\Omega f(p)\,\mathrm{d}\mu(p) = \sum_i \alpha_i \int_0^1 f\left(\phi_i(x)\right) \phi_i'(x)\,\mathrm{d}x$$





Process $\mathrm{e}^+\mathrm{e}^- \to q\bar{q}\mathrm{g}$ with Z-resonance and soft/collinear singularities in $s_{1,2}$.

Monte Carlo integration

- Expectation value for $f(x)$ over a *uniform probability density*

$$E(f) = \int f(x) \mathbb{1}_x \, \mathrm{d}\mu(x) \quad \text{and} \quad V(f) \propto 1/N$$

- Importance sampling with $x = G^{-1}(r)$ and $g(x) \propto f(x)$
  - large $f(x)$, large $g(x)$
- Stratified sampling adapts to variance $V(f)$
  - large and rapidly changing $f(x)$, more dense sampling
- Stratified sampling converges *faster* than importance sampling
  - Stratified sampling is unpractical for higher dimensions

VEGAS

- Adaptive algorithm for *factorizable* probability density $g(x)$ [Lepage 1978]

$$g(x) = \prod_i \frac{1}{N\Delta x_i},\ \Delta x_i = x_{i+1} - x_i \quad \text{and} \quad 0 \leq x_0 \leq \cdots \leq x_N \leq 1$$

- Different implementations [Lepage 1980], GSL, VAMP, ...
- Importance and Stratified Sampling
  - Automatically switch between both sample modes
- Integration grid with superficial stratification grid
  → Match under stratified sampling
- Stratified sampling → phasespace division

$$I = \int_U f(x)\,\mathrm{d}x = \sum_{i\in I} \int_{U_i} f(x)\,\mathrm{d}x,\ \text{with } U = \cup U_i,$$

  - Sample each division U_i with $N \geq 2$.

VAMP

- Overall probability density

$$g(p) = \sum_i \alpha_i g_i(\phi_i^{-1}(p))\phi_i^{-1\prime}(p)$$

- Sample each *channel* with probability density $\frac{\mathrm{d}G_i}{\mathrm{d}x} = g_i(\phi_i(x))$
- Compute overall integral

$$I = \sum_i \alpha_i \int_0^1 \frac{f(\phi_i(x))}{g(\phi_i(x))}\phi_i'(x)\,\mathrm{d}G_i(x).$$

- Adapt $g_i(x)$ with VEGAS → Vegas AMPlified [Ohl 1999]
- Channel weights α_i are adapted by channel variances

$$\alpha_i \to \alpha_i' = \frac{\alpha_i V_i(f,\vec{\alpha})^\beta}{\sum \alpha_i V_i(f,\vec{\alpha})^\beta}, \quad \beta > 0$$

Comparison: Multi-channel and flat parametrization

process	$\sigma_{\text{multi-channel}}$/fb	Eff. %	σ_{RAMBO}/fb	Eff. %
$e^+e^- \to \mu^+\mu^-$	$1.063\,864(22) \times 10^2$	41.22(0)	$1.063\,864\,5(4) \times 10^2$	42.69(0)
$e^+e^- \to \mu^+\mu^-\gamma$	$2.8788(10) \times 10^1$	7.95(25)	$2.891(14) \times 10^1$	0.112(19)
$e^+e^- \to \mu^+\mu^-\gamma\gamma$	$9.049(6) \times 10^{-1}$	3.95(11)	$9.07(5) \times 10^{-1}$	0.048(12)
$e^+e^- \to \mu^+\mu^-\gamma\gamma\gamma$	$1.0873(15) \times 10^{-2}$	1.93(6)	$1.082(7) \times 10^{-2}$	0.024(14)
$jj \to W^-(\to e^-\bar{\nu}_e)j$	$9.368(6) \times 10^5$	2.9(4)	$9.39(4) \times 10^5$	0.038(27)
$jj \to W^-(\to e^-\bar{\nu}_e)jj$	$2.8813(27) \times 10^5$	2.40(17)	$2.91(4) \times 10^5$	0.008(4)
$jj \to W^-(\to e^-\bar{\nu}_e)jjj$	$7.967(13) \times 10^4$	1.03(5)	$7.62(16) \times 10^4$	0.0(0)
$gg \to W^-(\to e^-\bar{\nu}_e)qq$	$2.4106(24) \times 10^4$	1.27(16)	$2.411(20) \times 10^4$	0.020(9)
$gg \to W^-(\to e^-\bar{\nu}_e)qqg$	$9.150(15) \times 10^3$	0.81(7)	$8.86(18) \times 10^3$	0.004(5)
$gg \to W^-(\to e^-\bar{\nu}_e)qqgg$	$2.366(15) \times 10^3$	0.152(18)	$1.54(9) \times 10^3$	0.0(0)

Table 1: Comparison of adaptive Monte Carlo integration in **WHIZARD** [Kilian, Ohl, and Reuter 2011]. Using multi-channel parametrization using *wood* or flat paramtrization using **RAMBO** [Kleiss, Stirling, and Ellis 1986], [Plätzer 2013].

Parallelization of Multi-Channel Monte Carlo

Overview

Shared Memory

- Memory accessible by multiple programs or threads of a process
- Multi-threading
  - `POSIX` system library
  - `OpenMP`

Distributed Memory

- Multiple processor systems with distinctive and private memory
- Message-passing
  - Tasks only have access to local data
  - Remote data requires communication

Standards

OpenMP

- Open Multi-Processing (shared-memory)
- Compiler directives, library routines and environment variables
  - → Provides efficient resource management
- Supports `Fortran`, `C` and `C++`
  - → Fortran 2008 *support* in `OpenMP` 5.0 (Nov 2018)
- Implemented by all major compilers: `GCC`, `Intel`, `LLVM`, ...

MPI

- Message-Passing Interface (distributed-memory)
  - → *Current version*: 3.1 (2015)
  - → `Fortran 2008` bindings and *non-blocking* collective communication
  - → *De-facto* standard
- Abstract concepts of communication between independent processes
- Implementation by an external library: `OpenMPI`, `MPICH`, `Intel MPI`

Comparison of paradigms

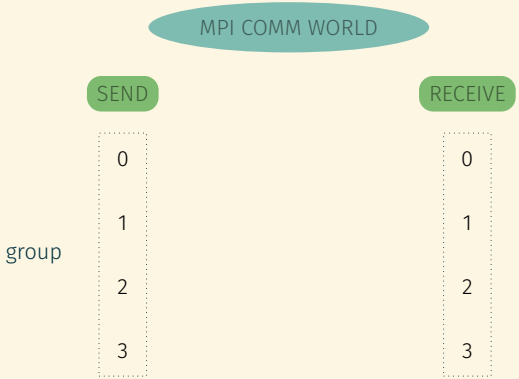
OpenMP

- Shared-memory systems restricted to single machines or highly specialized hardware
  - → Number of cores $N_{\text{Proc}} \leq \mathcal{O}(10)$
  - → Defining *shared variables* and *program sections* to be run in parallel
  - → Introduces race conditions
- Used to parallelize simpler structures
  - → Search for channel probabilities
  - → Helicity-based computation of matrix elements

MPI

- Distributed-memory systems have better scalability
  - → Limited by physical resources only
  - → `MPI` implemented *directly* into the source code
- Used for our parallelization!

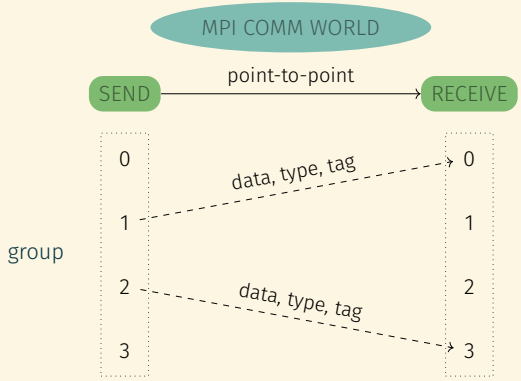
Communication Basics



Communicators

- Ordered set of process identifiers
- Tagged with an integer rank
- Communicator combines group and contex
- Predefined: `MPI_COMM_WORLD`

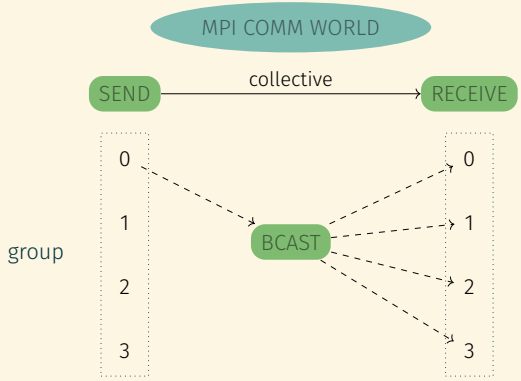
Communication Basics



Point-to-Point Communication

- Sender and receiver
- Data + data type + tag = message
- `MPI_SEND`, `MPI_RECV`, ...

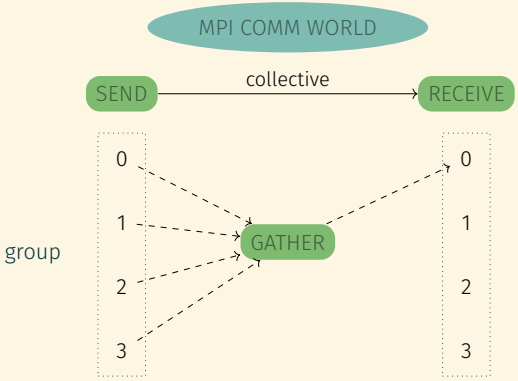
Communication Basics



Collective Communication

- One-to-all, all-to-one, all-to-all
- Data + data type
- `MPI_BCAST`, `MPI_GATHER`, `MPI_REDUCE`

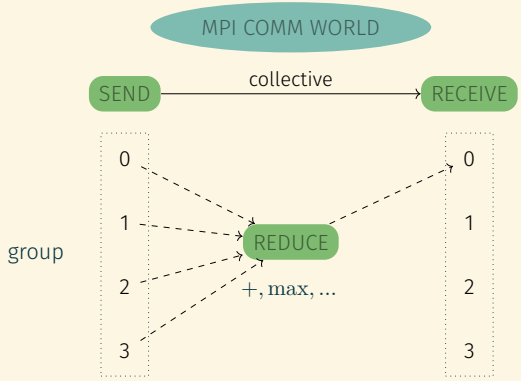
Communication Basics



Collective Communication

- One-to-all, all-to-one, all-to-all
- Data + data type
- `MPI_BCAST`, `MPI_GATHER`, `MPI_REDUCE`

Communication Basics



Collective Communication

- One-to-all, all-to-one, all-to-all
- Data + data type
- `MPI_BCAST`, `MPI_GATHER`, `MPI_REDUCE`

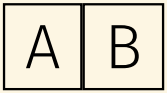
Blocking vs. Non-blocking Communication

Blocking program flow halts until communication completes

Non-blockiung directly return to calling function, communication in background

Increased parallel portion

by usage of non-blocking communication



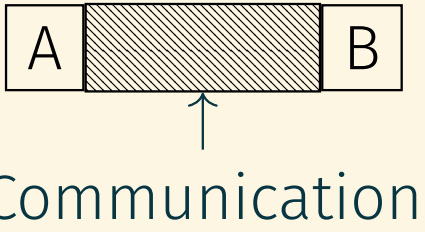
Blocking vs. Non-blocking Communication

Blocking program flow halts until communication completes

Non-blockiung directly return to calling function, communication in background

Increased parallel portion

by usage of non-blocking communication



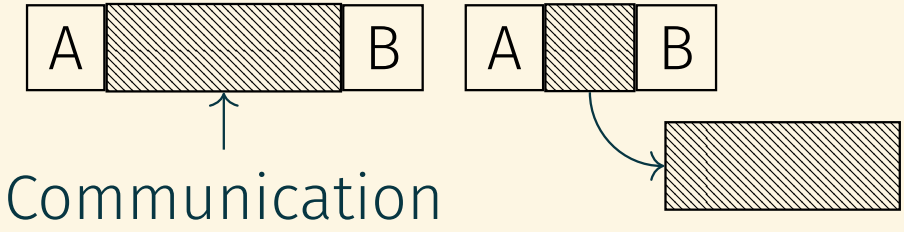
Blocking vs. Non-blocking Communication

Blocking program flow halts until communication completes

Non-blockiung directly return to calling function, communication in background

Increased parallel portion

by usage of non-blocking communication



Parallelization of Integration

VEGAS

- Parallelization of $\sum_{i \in I} I_i$ with phasespace division $U = \cup U_i$
- Number of stratas *depends* on number of calls → no division possible
- Workflow
  1. Broadcast grid to all workers
  2. Compute assigned divisions
  3. Master collects results and adapts grid → conserves numerical properties

VAMP

- Parallelization over $\sum_i \alpha_i \int \frac{f(\phi_i(x)))}{g((\phi_i(x)))} \phi_i'(x) \, \mathrm{d}G(x)$
- Number of channels process-dependent
- Workflow
  1. Broadcast all grids to all workers
  2. Compute and adapt assigned channels
  3. Master collects all grids and results → conserves numerical properties

Details of Implementation

- Implementation in WHIZARD event generator [Kilian, Ohl, and Reuter 2011]
- Reimplementation of VAMP in Fortran 2008 → VAMP2
  - Fully-functional VAMP replacement, supports channel equivalences
- Master/Slave ansatz
  - Only master has I/O
  - Preserves numerical properties
- Prefer VEGAS over *multi-channel* parallelization
  - Number of *stratas* $N_{\text{boxes}} \geq 2^d \approx \mathcal{O}(100)$ to $\mathcal{O}(1000)$ or higher for stratified sampling
  - Number of *channels* $N_{\text{c}} \approx \mathcal{O}(10)$ to $\mathcal{O}(100)$ depending on process
- Automatically switch between parallelization modes
- Fix *numerical properties* of serial and parallel runs with RNGstream [L'ecuyer et al. 2002]
  - Independent random number stream for each channel/strata
- VEGAS parallelization over $d_{\parallel} = \lfloor d/2 \rfloor$ with subspace $d_{\perp} = \lceil d/2 \rceil$ [Kreckel 1997]
- *Collective* and non-blocking communication: MPI_BCAST, MPI_REDUCE

Ahmdal's Law

- Parallel portion p of a programm
- Speedup for N workers

$$S(p;N) = \frac{T(p;N)}{T(0;1)} = \frac{1}{(1-p) + p/N}$$

- Parallelization efficiency depends heavily on parallel portion

$$\lim_{N\to\infty} S(p;N) = \frac{1}{1-p}$$

- Benchmarks with full-partonic and $\mathrm{e}^+\mathrm{e}^-$ processes with `WHIZARD`

Results

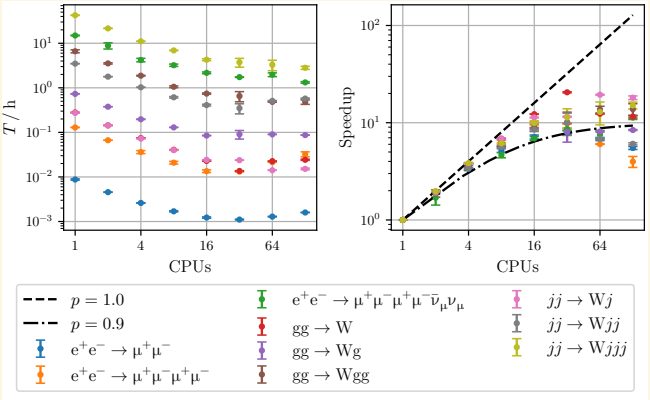


Figure 1: Benchmark for different complex processes. For reference Ahmdal's law is shown.

Discussion

- Speedup of $\mathcal{O}(10)$
  - not optimized for cluster topology
- Saturation after $N = 32$
- Parallel portion $9.0 \times 10^1\,\%$

Flavour Benchmark

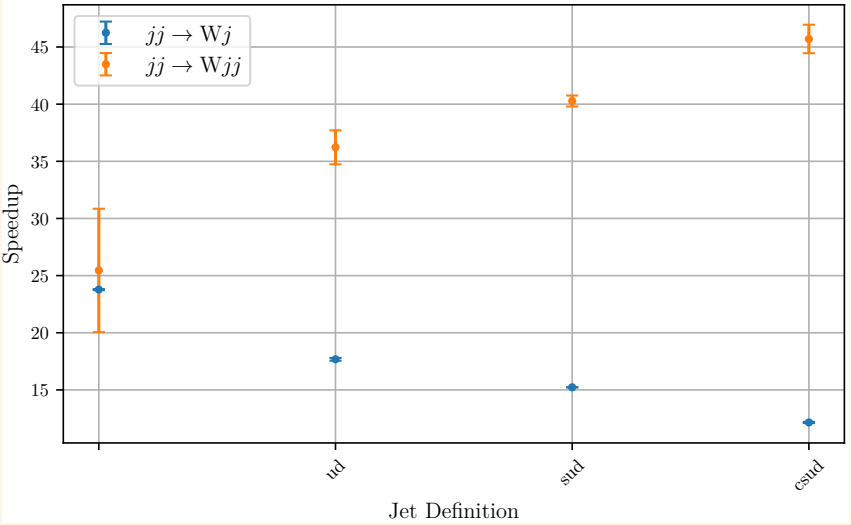


Figure 2: Benchmark for increasing jet flavours → increasing ME evaluation time. **Optimized** on cluster topology with 60 workers → $45\times$ **faster**.

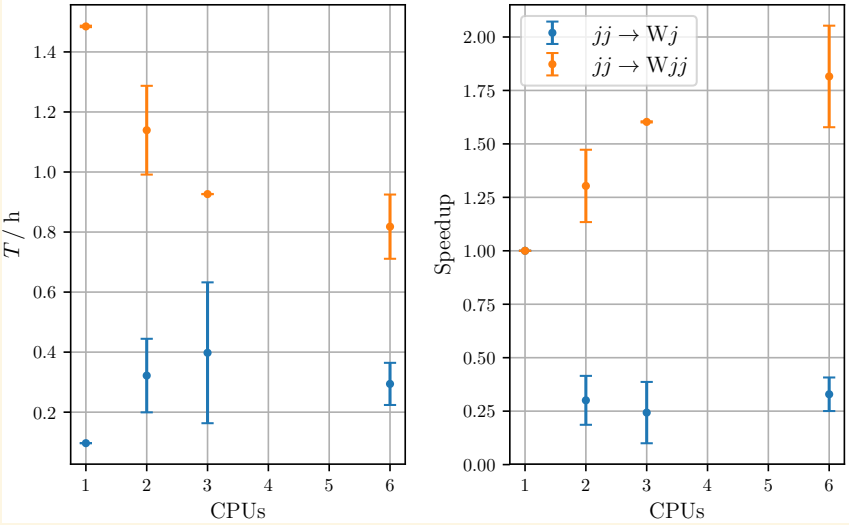


Figure 3: 20 workers with increasing number of threads → doubled speedup, but only *favorable* for **more complicated** processes.

Conclusion i

MPI-only Parallelization

- Parallelization of the adaptive Multi-Channel integration and VEGAS
  - → Automatically switch between VEGAS and multi-channel parallelization
- Speedups of $\mathcal{O}(10) - \mathcal{O}(100)$
- Complicated processes → limitation by available memory per computing node

Hybrid-Parallelization

- Hybrid ansatz with OpenMP and MPI
- Memory-intense application
  - → OpenMP for on-node parallelization, MPI for node communication

Conclusion ii

Outlook

- Profiling to increase parallel portion
- Implementation of server client model
- Packing of VAMP2 as independent package
  - → *Currently* only as part of WHIZARD

Thank you for your attention!

Thank you for your attention!

Backup

Using WHIZARD with MPI i

- Prerequisite:
  - MPI library: *OpenMPI* (≥ `2.0.0`), MPICH
    - → ask your administrator
    - → `./configure FC=mpifort --enable-fc-mpi`
  - Prepare a hostfile
    - → not required on a cluster

```
mpirun --hostfile myhosts -np 20 --output-filename mpi.log whizard
 ↪  process.sin
```

Using WHIZARD with MPI ii

Hostfile

```
$ cat myhosts
caesium slots=12
xenon   slots=12
neon    slots=5
```

Sindarin

```
$integration_method = "vamp2"
$rng_method = "rng_stream"
```

Improving runtime i

Phasespace Parametrization

- Default phasespace configuration with `cascades` *very* slow
- *DAG* representation of Feynman diagrams from `O'Mega` → cascades2 [Utsch 2018]
  - Respects restrictions on matrix elements
  - Includes all available models and not limited to `SM`
  - Improved runtime over `cascades` up to 215x faster

Sindarin

```
?omega_write_phs_output = true
$phs_method = "fast_wood"
```

Improving runtime ii

Thread-based Parallelization with OpenMP

- Loop over helicity amplitudes in O'Mega [Nejad, Ohl, and Reuter 2015]
- Phasespace configuration in *cascades2* and generation [Utsch 2018]
- Computation of *multi-channel* probability density
  - → Linear or binary search over all grids

Improving runtime iii

Configure

- `OpenMP` commonly *supported* by (most) compilers
- Enabled by configuration flag

```
./configure --enable-fc-openmp
```

Improving runtime iv

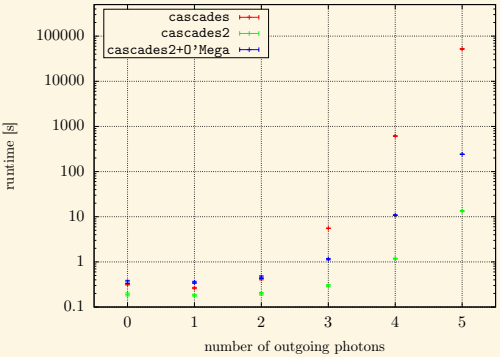


Figure 4: Runtimes of `cascades` and `cascades2`, e.g. $gg \to W^+ t \bar{b} + n\gamma$ [Utsch 2018].

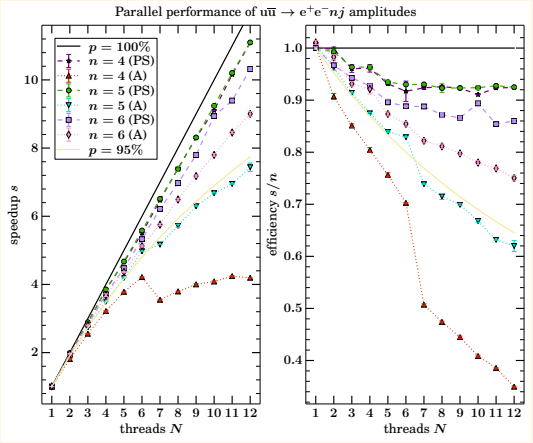


Figure 5: Speedup for *thread-based* parallelization with **OpenMP**, e.g. Drell-Yan [Nejad, Ohl, and Reuter 2015].

Message-Passing Interface (History)

- MPI is a message-passing library interface specification
  - MPI is not a programming language
  - MPI is not an implementation
  - MPI defines only the interface for the communication procedures
  - MPI usage is more abstract and more complicate than OpenMP
- MPI-1 in 1994, MPI-2 in 1997, MPI-3 in 2012 and MPI-3.1 in 2015
- MPI-3.1 supports Fortran 2008 language bindings and *non-blocking* collective communications

Message-Passing Interface

- Abstracts concepts of communication between independent processes
- Introduces concepts of point-to-point and collective communication
  - → Point-to-point communication
  - → Collective communication
  - → Definition of a message
  - → Blocking and non-blocking communication → Ahmdal's Law
  - → Communicator, topology and process management
  - → Unified API for C and Fortran
- Master slave model
- Server client model
  - → Main executable, simplified integration-only client

Adaptive Integration

- Adpative and iterative VEGAS algorithm based on Importance and Stratified sampling
- Normalized step-function $p(x)$ as probability density function $g(x)$

$$g(x) = \frac{1}{N\Delta x_i}, \quad x_i - \Delta x_i \leq x_i < x_{i+1}, \quad i \in \{1, ..., N\}$$

- Steps x_i define bin length $\Delta x_i = x_i - x_{i-1}$ with $0 = x_0 < ... < x_N = 1$
- Integrate and collect (squared) weights
- Adapt bin width with weights

Implementation

Stratification Grid

- Additioanl stratifiaction grid
- Divide unit hypercube in *equal-sized* subcubes
- Access stratification grid with (box-)coordinates $\vec{c} = \{c_1, c_2, ..., c_N\}, \quad c_i \in \mathbb{N}$
- Switch between importance and (genuine) stratified sampling

Adaptive Algorithm

- Access bins with (bin-)coordinates $\vec{b} = \{b_1, b_2, ..., b_M\}, \quad b_i \in \mathbb{N}$
- Calculate x_i from bin and box coordinates
- Transform box coordinates $\vec{c}$ and a random number $\vec{r} \in U^{\otimes d}$ to actual position on the adaptive grid $\vec{x}$

$$\vec{x} = \phi\left(\vec{c} + \vec{r}\right)$$

Example

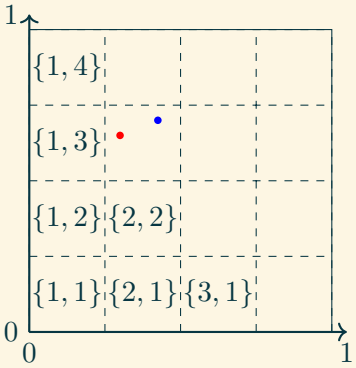


Figure 6: Stratified grid with two sample points $\{2, 3\}$.

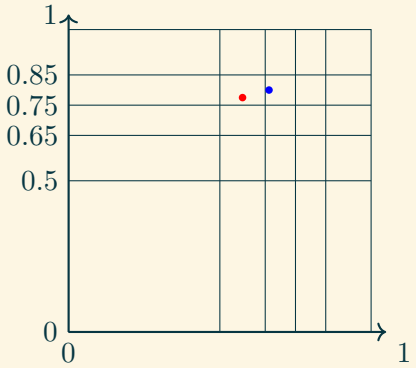


Figure 7: Adpative grid with five bins and transformed points from the stratified grid.

Parallelization over the Stratified Grid

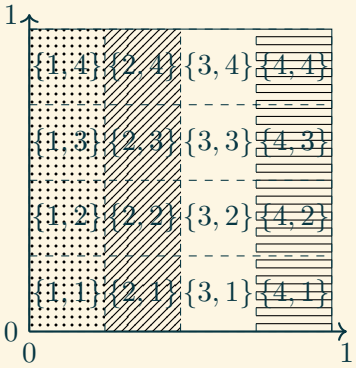


Figure 8: Stratified grid is divided for parallelization.

- Broadcast adaptive Grid non-blocking to all worker
- Each worker gets a division of the strtatifed grid assigned $\{c_1, ..., c_{D_\parallel}\}$ and $D_\parallel = \lfloor D/2 \rfloor$
- Each worker computes its division with its own (independent) stream of random numbers
- Master reduces non-blocking the results from all workers

MPI example

Point-to-Point Communication

```
if (rank == sender) then
   call MPI_send (data(:n), n, MPI_INTEGER, receiver, tag, comm, ierr)
else if (rank == receiver) then
   call MPI_recv (data(:n), n, MPI_INTEGER, sender, tag, comm, ierr)
end if
```

References

- Kilian, Wolfgang, Thorsten Ohl, and Jürgen Reuter (2011). ?WHIZARD: Simulating Multi-Particle Processes at LHC and ILC? In: *Eur. Phys. J.* C71, p. 1742. DOI: 10.1140/epjc/s10052-011-1742-y. arXiv: 0708.4233 [hep-ph].
- Kleiss, R. and R. Pittau (1994). ?Weight optimization in multichannel Monte Carlo? In: *Comput. Phys. Commun.* 83, pp. 141–146. DOI: 10.1016/0010-4655(94)90043-4. arXiv: hep-ph/9405257 [hep-ph].
- Kleiss, R., W. James Stirling, and S. D. Ellis (1986). ?A New Monte Carlo Treatment of Multiparticle Phase Space at High-energies? In: *Comput. Phys. Commun.* 40, p. 359. DOI: 10.1016/0010-4655(86)90119-0.

References ii

- Kreckel, Richard (Nov. 1997). ?Parallelization of adaptive MC integrators? In: *Computer Physics Communications* 106.3, pp. 258–266. DOI: 10.1016/s0010-4655(97)00099-4. URL: https://doi.org/10.1016%2Fs0010-4655%2897%2900099-4.
- L'ecuyer, Pierre et al. (2002). ?An object-oriented random-number package with many long streams and substreams? In: *Operations research* 50.6, pp. 1073–1075.
- Lepage, G. Peter (1978). ?A New Algorithm for Adaptive Multidimensional Integration? In: *J. Comput. Phys.* 27, p. 192. DOI: 10.1016/0021-9991(78)90004-9.
- – (1980). ?VEGAS: AN ADAPTIVE MULTIDIMENSIONAL INTEGRATION PROGRAM? In:

References iii

- Nejad, Bijan Chokoufe, Thorsten Ohl, and Jürgen Reuter (Nov. 2015). ?Simple, parallel virtual machines for extreme computations? In: *Computer Physics Communications* 196, pp. 58–69. DOI: 10.1016/j.cpc.2015.05.015. URL: https://doi.org/10.1016%2Fj.cpc.2015.05.015.
- Ohl, Thorsten (1999). ?Vegas revisited: Adaptive Monte Carlo integration beyond factorization? In: *Comput. Phys. Commun.* 120, pp. 13–19. DOI: 10.1016/S0010-4655(99)00209-X. arXiv: hep-ph/9806432 [hep-ph].
- Plätzer, Simon (2013). ?RAMBO on diet? In: arXiv: 1308.2922 [hep-ph].
- Utsch, Manuel (Apr. 2018). ?Parametrisierung des Phasenraums im Monte-Carlo-Eventgenerator WHIZARD? MA thesis. Unversität Siegen.