View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Springer - Publisher Connector

Eur. Phys. J. C (2011) 71: 1559
DOI 10.1140/epjc/s10052-011-1559-8

THE EUROPEAN PHYSICAL JOURNAL C

Special Article - Tools for Experiment and Theory

Monte Carlo integration on GPU

J. Kanzaki[a]

KEK, Tsukuba 305-0801, Japan

Received: 11 October 2010 / Revised: 19 January 2011 / Published online: 8 February 2011
© The Author(s) 2011. This article is published with open access at Springerlink.com

Abstract We use a graphics processing unit (GPU) for fast computations of Monte Carlo integrations. Two widely used Monte Carlo integration programs, VEGAS and BASES, are parallelized for running on a GPU. By using W^+ plus multi-gluon production processes at LHC, we test the integrated cross sections and execution time for programs written in FORTRAN and running in the CPU and those running on a GPU. The integrated results agree with each other within statistical errors. The programs run about 50 times faster on the GPU than on the CPU.

1 Introduction

The GPU (Graphics Processing Unit) was originally developed to quickly perform the multitude of calculations necessary to render complex moving images on computer displays. It achieves this speed by using many multi-processors, which means it could be used as a powerful parallel processor not only for graphics applications but also for general purpose computations. GPUs have already been used in scientific applications which require a large number of calculations to process data in the fields of astrophysics and fluid dynamics. Recently, GPUs have also been used in elementary particle physics, to calculate cross sections [1–3]. In these studies programs run on the GPU were shown to be about 100 times faster than those run on the CPU.

Decreasing the computation time for simulating events by two orders of magnitude could dramatically improve the efficiency of analysis in the field of elementary particle physics. In this paper, we show that general purpose Monte Carlo integration programs can be adopted to run on the GPU, opening the door to fast and economical computations in all areas of research that makes use of Monte Carlo method.

[a] e-mail: junichi.kanzaki@kek.jp

2 Monte Carlo integration programs

Scattering amplitudes of physics processes at the LHC energies are expressed as complex functions of momenta and helicities of external particles. The kinematical distributions of the produced particles are obtained by integrating the squared amplitudes over the phase space of the external particles. The large number of dimensions associated with this phase space, as well as the need for differential distributions and experimental cuts, make these problems particularly suited to Monte Carlo integration techniques.

As the number of final state particles increases, the computation time necessary to obtain accurate results grows quickly. There are two contributing factors. First, the number of dimensions that must be integrated is proportional to the number of final state particles. So the number of points at which the function should be evaluated increases with the number of final state particles. Second, the complexity of the scattering amplitude also increases with the number of external particles. Therefore, integration of the differential cross section becomes a very time consuming task for multi-particle production processes, and often limits which processes can be accurately simulated. Using the GPU may significantly reduce this computation time and will contribute to the improvement of the efficiency of physics analysis at LHC and elsewhere.

The program VEGAS [4] and its variants are widely used for Monte Carlo integration. They are based on an iterative and adaptive Monte Carlo scheme. In these programs each axis of variable is divided into grids, thus the integrand volume is divided into hyper cubes. Monte Carlo integration is performed in each hypercube and variances from the hypercubes are used to define new grid spacings which are used in the next iteration step. The variance of total integral is reduced iteration by iteration. BASES [5, 6] is one variant of VEGAS that was developed at KEK, and has been widely used in particle physics calculations for colliders.

In this paper we study the parallelization of VEGAS and BASES, for running on a GPU.

3 Parallelization of Monte Carlo integration program

3.1 Program structure

Multi-dimensional integration programs, VEGAS and BASES, have the following common structure:

1. initialize parameters,
2. generate N points consisting of a set of k random numbers within a k-dimensional hyperspace,
3. evaluate an integrand function at the generated space point,
4. sum up values of the integrand function and their squares for all phase space points and also within each hyper cube, and compute their averages and variances,
5. optimize grid spacing after accumulating N function values,
6. repeat steps 2–5 up to M iterations or until the desired accuracy is reached.

In BASES, after M iterations (grid optimization phase) are done, further iteration steps are executed in order to improve the accuracy of the integration (integration phase). The results of this integration phase are used for event generation by the program SPRING [5, 6].

3.2 Parallelization for the GPU

Before developing GPU versions of the programs, we measured the fraction of CPU time spent on each step in the original version of FORTRAN programs. We found that almost all (98–99%) of CPU time was used in step 3, calculating the integrand function. This fraction grows as the number of sampling points grows and the complexity of the integrand function grows. Therefore a significant reduction of the total CPU time is expected by parallelizing the function calls at all sampling points with GPU.

In order to transfer the whole system of the Monte Carlo integration, all programs should be written in CUDA [7], C/C++ system platform developed for general purpose computing on a GPU. Because both VEGAS and BASES are originally written in FORTRAN, we first converted them into C code and then transformed them further into CUDA codes.

Especially, VEGAS is written in the old style of FORTRAN with many "GOTO" statements. It had to be rewritten in a modern style of FORTRAN control structures with a reduced number of "GOTO"'s. Then, FORTRAN source codes should be carefully converted into C codes with a special care for the difference of array indexing between the two programming languages.

For the development of the GPU programs based on the converted C programs, the structure of the program should be carefully considered to achieve better performance by parallelization. Because the generation of space points (step 2) and the evaluation of an integrand function (step 3) can be executed independently for each point in phase space, these steps can be parallelized on GPU. In the original CPU programs, the accumulations and summations of the integrand function values (step 4) are included in a loop over phase space points for the function evaluation (step 3). Because the accumulations and summations are not independent between points in phase space, they have to be separated from the function evaluation step (step 3) on the GPU. Hence, the generation of phase space points (step 2) and the evaluation of the integrand function at all phase space points (step 3) are computed in the GPU in parallel, and their values are transferred to CPU memory. The computed function values are accumulated on the CPU and the grid parameters are optimized based on the accumulated information (steps 4–5). These steps are iterated and the variance of the integral is reduced.

Due to the limited support for double precision calculations on the GPU that we used for this study [1, 2], floating point computations on the GPU were done in single precision.[1] We compared the results and performances for the programs in FORTRAN and C on the CPU, and on the GPU.

4 Computing environments

4.1 GPU and its host PC

We used a GeForce GTX285 by NVIDIA [9] for the computation of cross sections of physics processes using Monte Carlo integration. The GeForce GTX285 is connected with PCI Express2$\times$16 bus has 30 streaming multi-processors (SM). Since each SM has 8 streaming processors (SP), the GTX285 GPU card has 240 SP in total. Other parameters of the GTX285 are summarized in Table 1.

The GTX285 is controlled by a Linux PC running the Fedora10 (64 bit) operating system. The parameters of host computer are summarized in Table 2.

In order to compile programs for the GPU, we used the CUDA version 2.3 toolkit which is obtained from the

Table 1 Parameters of GTX285

Number of multiprocessor	30
Number of core	240
Total amount of global memory	2 GB
Total amount of constant memory	64 kB
Total amount of shared memory per thread block	16 kB
Total number of registers available per thread block	16 k
Clock rate	1.48 GHz

[1] This limitation is relaxed for NVIDIA's GPUs with newer architecture [8].

Table 2 Host PC environment

CPU	Core i7 2.67 GHz
L2 Cache	8 MB
Memory	6 GB
Bus Speed	1.333 GHz
OS	Fedora 10 (64 bit)

Table 3 Development environment

nvcc	Rel. 2.3 (V0.2.1221)
CUDA Driver	Ver. 2.30
CUDA Runtime	Ver. 2.30
gcc	4.3.2 (Red Hat 4.3.2-7)
gfortran	4.3.2 (Red Hat 4.3.2-7)

NVIDIA site [9]. The programs in FORTRAN and C were compiled using gfortran and gcc respectively, which are automatically installed with Fedora 10. The versions of the compilers are summarized in Table 3.

4.2 Process time measurement

For comparisons of execution time, we measured the time between the start and end of VEGAS/BASES programs, i.e. between step 1 and the completion of step 6, including steps 4 and 5 that were processed on the CPU. For FORTRAN programs, an intrinsic procedure of gfortran, "`cpu_time`", is used for the measurement of the elapsed CPU time. For the C and GPU programs, a system call, "`getrusage`", is used for the time measurements.

5 Physics process

In order to test the GPU version of VEGAS and BASES, called gVEGAS and gBASES[2] respectively, we compare the total cross sections for multi-particle production processes at the LHC. In particular, we report results on the following processes

$$u\overline{d} \to W^+(\to \mu^+\nu_\mu) + n \text{ gluons} \quad (n = 0 \sim 4) \tag{1}$$

with semi-realistic final state cuts at LHC. The dimension of the integral is $3(n+2)-4$ from phase space, 2 from the parton distributions (PDF), and 1 for the helicity summation, and hence $3n+5$; hence phase space varies from a 5-dimensional integral for no gluon ($n=0$) to a 17-dimensional integral for 4 gluons ($n=4$).

[2]Sample source codes of gVEGAS are available on the web page: http://madgraph.kek.jp/KEK/GPU/gVEGAS/example/. The source codes of gBASES will also become available soon together with the event generation package, SPRING.

Table 4 $u\overline{d} \to W^+(\to \mu^+\nu_\mu) + \text{gluons}$

Number of gluons	Number of diagrams	Number of color bases
0	1	1
1	2	1
2	8	2
3	54	6
4	516	24

The degree of complexity (length) of the integral function can be estimated from the number of contributing Feynman diagrams and the number of independent color-basis vectors. They are listed in Table 4. Previous studies [1, 2] have shown that the performance of GPU computations is limited by the product of these two numbers, and the processes in (1) cover program size of four orders of magnitude difference.

In order to simulate realistic LHC experiments, we introduce the following final state cuts. For gluons,

$$|\eta_i| < 5, \tag{2a}$$

$$p_{\mathrm{T}i} > 20 \text{ GeV}, \tag{2b}$$

$$p_{\mathrm{T}ij} > 20 \text{ GeV}, \tag{2c}$$

where η_i and $p_{\mathrm{T}i}$ are the rapidity and the transverse momentum of the ith jet, respectively, in the pp collisions rest frame along the right-moving ($p_z = |p|$) proton momentum direction, and $p_{\mathrm{T}ij}$ is the relative transverse momentum [10] between the jets i and j defined by

$$p_{\mathrm{T}ij} \equiv \min(p_{\mathrm{T}i}, p_{\mathrm{T}j})\,\Delta R_{ij}, \tag{3a}$$

$$\Delta R_{ij} = \sqrt{\Delta\eta_{ij}^2 + \Delta\phi_{ij}^2}. \tag{3b}$$

Here ΔR_{ij} measures the boost-invariant angular separation between jets. For μ^+ from W^+ decay, we require

$$|\eta_l| < 2.5, \tag{4a}$$

$$p_{\mathrm{T}l} > 20 \text{ GeV} \tag{4b}$$

As for the parton distribution functions (PDF), we use the set CTEQ6L1 [11] and the factorization scale is chosen to be the Z boson mass. The QCD coupling constant is also fixed as $\alpha_s(m_Z)_{\overline{\mathrm{MS}}} = 0.118$ [12].

For the computation of helicity amplitudes of these processes, HELAS [13, 14] for FORTRAN programs and its C/GPU version, HEGET [1, 2] are used.

6 Results

6.1 Parameters of the integration programs

In order to make a fair comparison of the computational performance of the various codes, it is important that they make

the same number of calculations. The behavior of the Monte Carlo integration by VEGAS and BASES, can be controlled by the following parameters:

- number of total function calls in one iteration step (`NCALL`),
- number of maximum iteration steps (`ITMX`), and
- desired accuracy of the integration (`ACC`).

`NCALL` is the number N in step 5 and `ITMX` is the number M in step 6 in Sect. 3.1. Iteration steps of BASES are separated into two phases: the grid optimization step and the integration step. Accordingly, `ITMX` and `ACC` are also separated as:

- number of maximum iteration steps (`ITMX1`), and
- desired accuracy of integration (`ACC1`)

for the grid optimization phase, and

- number of maximum iteration steps (`ITMX2`), and
- desired accuracy of integration (`ACC2`)

for the integration phase.

Parameter values used in this study are summarized in Table 5. In order to keep the total number of calculations the same for all of the programs, all desired accuracies, `ACC` for VEGAS and `ACC1` and `ACC2`, are set to an extremely small value (0.001%) which cannot be reached by MC sampling of `NCALL` × `ITMX` points used in this study: see Table 5. For BASES, the number of iteration steps for the grid optimization and integration phases are set to be equal (`ITMX1` = `ITMX2`), and their sum is set the same as `ITMX` of VEGAS programs (`ITMX1` + `ITMX2` = `ITMX`). In summary, we accumulate 10^7 sample points for processes up to two gluons ($n = 0, 1, 2$) and 10^8 points for those with more gluons ($n = 3$ and 4).

Table 5 Parameters for integrations

Number of gluons	NCALL	ITMX	ITMX1	ITMX2
0	10^6	10	5	5
1	10^6	10	5	5
2	10^6	10	5	5
3	10^7	10	5	5
4	10^7	10	5	5

6.2 Total cross section computation

Total cross sections for the processes in (1) with experimental cuts (2–4) are listed in Table 6. They are computed with programs in FORTRAN, C and CUDA (GPU). Cross sections from the different programs agree with each other within their statistical errors. In addition, they agree with the results from the event generator MadGraph/MadEvent [15–17].

6.3 Parameters of the kernel program

A function program, which is called from a CPU program and executed on a GPU, is called a *kernel*. The kernel program which is executed on each streaming processor is called a *thread*. In the CUDA programming model a set of threads forms a *thread block*. Threads in a thread block can share data through shared memory of the GPU. The maximum size of a thread block for the GTX285 is 512. The size of a thread block can be changed within this limit when the kernel program is executed on a GPU. With a single call of a kernel function from a CPU program, multiple thread blocks are executed in parallel on the GPU. A set of thread blocks executed with a single kernel call is called a *grid* of thread blocks. The total number of threads executed in parallel with a single kernel call becomes "the number of thread blocks in a grid" × "the size of a thread block".

The performance of GPU programs largely depends on parameters of kernel programs executed on GPU. Most significant parameters which affect the process time of programs are:

- number of registers allocated to a thread, and
- number of threads in a thread block.

If a thread is allocated more registers, the performance of the GPU programs can improve. But, because the maximum

Table 6 Total cross sections of $u\bar{d} \to W^+(\to \mu^+\nu_\mu) + n$-gluons computed by programs in FORTRAN, C, CUDA (GPU) and MadGraph/MadEvent

No. of	VEGAS			BASES			MG/ME	[fb]
gluons	FORTRAN	C	GPU	FORTRAN	C	GPU		
0	2.137 ± 0.001	2.138 ± 0.001	2.137 ± 0.001	2.137 ± 0.001	2.137 ± 0.001	2.137 ± 0.001	2.138 ± 0.002	$\times 10^6$
1	1.783 ± 0.001	1.783 ± 0.001	1.780 ± 0.001	1.785 ± 0.001	1.784 ± 0.001	1.782 ± 0.001	1.773 ± 0.003	$\times 10^5$
2	1.873 ± 0.007	1.853 ± 0.006	1.843 ± 0.006	1.876 ± 0.007	1.883 ± 0.010	1.870 ± 0.007	1.874 ± 0.002	$\times 10^4$
3	2.868 ± 0.008	2.881 ± 0.009	2.832 ± 0.010	2.860 ± 0.010	2.855 ± 0.014	2.907 ± 0.012	2.845 ± 0.005	$\times 10^3$
4	6.186 ± 0.041	6.054 ± 0.081	6.157 ± 0.073	6.078 ± 0.134	6.191 ± 0.068	6.385 ± 0.235	6.070 ± 0.010	$\times 10^2$

Table 7 Process time for a single function call in VEGAS and BASES on CPU with FORTRAN or C, and on GPU with CUDA. Numbers in the parentheses of the FORTRAN and C columns are the ratio of process time relative to that of GPU

No. of gluons	VEGAS [μsec]			BASES [μsec]		
	FORTRAN	C	GPU	FORTRAN	C	GPU
0	1.32 (63.8)	1.06 (51.2)	0.0207	1.78 (68.7)	1.39 (53.5)	0.0260
1	2.19 (68.8)	1.73 (54.6)	0.0318	2.97 (75.0)	2.24 (56.6)	0.0396
2	4.19 (84.2)	2.96 (59.5)	0.0497	4.97 (88.3)	3.35 (59.6)	0.0563
3	11.1 (101)	7.00 (63.6)	0.110	11.7 (103)	7.02 (62.2)	0.113
4	72.1 (77.8)	37.4 (40.4)	0.927	61.6 (66.2)	31.8 (34.2)	0.931

number of registers available per thread block is limited to 16 k (Table 1), the size of thread blocks becomes smaller and the level of parallelism becomes lower. In this study we use 64 registers allocated to a thread and 256 threads in a block. From the detailed study of dependence of performance on these parameters we find that they give almost the best performance for all processes in this study.

The number of thread blocks in a grid (= a set of thread blocks), which is executed with a single kernel call, is set to be equal to `NCALL`, so that one iteration of Monte Carlo integration steps is executed by a single kernel call.

6.4 Process time comparisons

In Table 7 the measured process time for a single function call is listed for each program. As explained above, the process time per single function call is obtained by dividing the total computation time by 10^7 for processes with up to two gluons ($n = 0, 1, 2$) and by 10^8 for those with more gluons ($n = 3$ and 4).

Numbers in parentheses in the FORTRAN and C columns in Table 7 are the ratio of the process time as compared to that of the GPU. About a factor of 50 times more sampling is possible with the GPU as compared to the C programs on CPU. During the comparison of process time, we find that the original FORTRAN codes run slower than the C-versions. Because the total process time of the CPU programs is dominated by the function (amplitude) computation, this FORTRAN-to-C ratio originates from the difference in process time of the amplitude computation. We find that arithmetic operations of complex numbers are faster in the C programs than in the FORTRAN programs. In particular, the addition of complex numbers which appears frequently in the amplitude computation is processed about 60% faster in the C programs. We use in-line functions for the computations of complex numbers in C, which have better efficiency compared with built-in complex functions in FORTRAN.

In Fig. 1 the process time for a single function call is plotted versus the number of gluons in the final state. In Fig. 2 ratios of the process time between programs on the CPU (FORTRAN/C) and those on the GPU are plotted. The differences between process time for VEGAS and BASES are small.

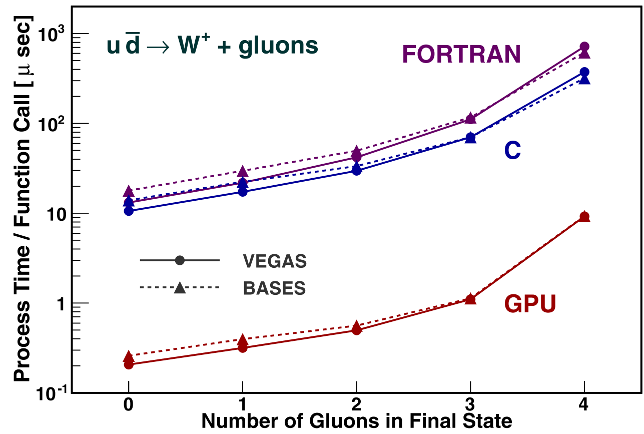

Fig. 1 Process time of a single function call for $u\bar{d} \to W^+(\to \mu^+\nu_\mu) + n$-gluons

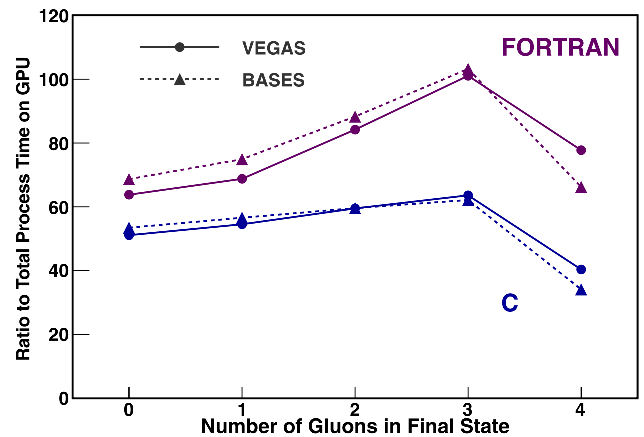

Fig. 2 Process time ratios of FORTRAN and C programs to the corresponding GPU program

Programs which are executed on the GPU can run about 50 times faster than those in C on the CPU, and even greater compared to the original FORTRAN code running on the CPU.

When the final state has 4 gluons, the size of the GPU program becomes large and requires more access to local memory. Previous studies indicate that large GPU programs [1, 2] show worse performance. Still the VEGAS (BASES) program for the 4 gluon production process runs

40 (34) times faster on GPU than the C-program runs on CPU.

7 Summary

Based on the programs VEGAS and BASES written in FORTRAN, we have developed the Monte Carlo integration programs, gVEGAS and gBASES, which can be executed on NVIDIA's GPU using the CUDA development kit. We have tested their performance with the computation of total cross sections for processes, $u\overline{d} \to W^+(\to \mu^+\nu_\mu) + n$-gluons ($n = 0 \sim 4$), in pp collisions at $\sqrt{s} = 14$ TeV. The total cross sections agree with each other within statistical errors for all programs. Both the VEGAS and BASES programs run about 50 times faster on the GPU than the same programs written in C running on the CPU. Compared with FORTRAN programs the GPU version programs show more than 60 times better performance in execution time. For the process with 4 gluons, the size of GPU programs becomes large and their relative performance is somewhat reduced.

Acknowledgements The author wishes to thank Kaoru Hagiwara for his encouragement and advice throughout this work. He also thanks Naotoshi Okamura in valuable discussions. This work is supported in part by the Grant-in-Aid for Scientific Research from the Japan Society for the Promotion of Science (No. 20340064).

Open Access This article is distributed under the terms of the Creative Commons Attribution Noncommercial License which permits any noncommercial use, distribution, and reproduction in any medium, provided the original author(s) and source are credited.

Appendix: Sample codes for gVEGAS

Sample source codes of the gVEGAS program are available from the web page: http://madgraph.kek.jp/KEK/GPU/gVEGAS/example/.

They include a minimum set of source files which are necessary to do Monte Carlo integration with the VEGAS algorithm on GPU, but do not include `Makefile` which largely depends on user's environment of development.

User programs

Sample codes include two user programs: `gVegasMain.cu` and `gVegasFunc.cu`. They should be customized by user to the task one intends to perform.

gVegasMain.cu

`gVegasMain.cu` includes a sample main program for Monte Carlo integration where users can set parameters for the integration. Typical parameters are:

`ncall` number of sample points per iteration
`itmx` maximum number of iterations
`acc` required accuracy during iterations
`nBlockSize` size of a thread block of a kernel program on GPU.

For the benefit of developers of GPU programs, this sample main program has a simple interface for the input of these parameters. With the use of CUDA utility functions, they can be given to a compiled program through a list of arguments as:

"program" `-n=`*ncall0* `-i=`*itmx* `-a=`*acc0*
`-b=`*nBlockSize*

For `itmx` and `nBlockSize` given values are used directly. On the other hand, actual values of `ncall` and `acc` are calculated as:

$$\texttt{ncall} = 1024 \times \texttt{ncall0}$$

$$\texttt{acc} = 0.00001 \times \texttt{acc0}$$

from given values.

gVegasFunc.cu

User function program integrated in the program is described in `gVegasFunc.cu`. The calling sequence of user functions is

```
float func(float* rx, float wgt)
```

where `rx` includes a set of variables and `wgt` is a function weight.

Internal programs

The gVEGAS consists of the following programs which are included in the sample codes:

`gVegas.cu` main program of `gVEGAS` system
`gVegasCallFunction.cu` kernel program which runs on GPU called from `gVegas.cu`.
`xorshift.cu` random number generator on GPU.

Header files

The following header files which are necessary for the gVEGAS system are also included in the sample codes:

`gvegas.h` includes `nBlockSize` which user can set in `gVegasMain.cu`
`vegasconst.h` includes internal constants which are located at constant memory of GPU
`vegas.h` includes internal `gVEGAS` parameters
`kernels.h` a list of kernel programs which are included at CUDA compilation.

References

1. K. Hagiwara, J. Kanzaki, N. Okamura, D. Rainwater, T. Stelzer, Eur. Phys. J. C **66**, 477 (2010). arXiv:0908.4403
2. K. Hagiwara, J. Kanzaki, N. Okamura, D. Rainwater, T. Stelzer, Eur. Phys. J. C **70**, 513 (2010). arXiv:0909.5257
3. W. Giele, G. Stavenga, J. Winter, FERMILAB-PUB-10-025-T, Feb 2010. arXiv:1002.3446
4. G.P. Lepage, J. Comput. Phys. **27**, 192 (1978)
5. S. Kawabata, Comput. Phys. Commun. **41**, 127 (1986)
6. S. Kawabata, Comput. Phys. Commun. **88**, 309 (1995)
7. http://www.nvidia.com/object/cuda_home_new.html
8. http://www.nvidia.com/object/fermi_architecture.html
9. http://www.nvidia.com/page/home.html
10. S. Catani, Y.L. Dokshitzer, M.H. Seymour, B.R. Webber, Nucl. Phys. B **406**, 187 (1993)
11. CTEQ Collaboration, H.L. Lai et al., Eur. Phys. J. C **12**, 375 (2000)
12. K. Nakamura et al. (Particle Data Group), J. Phys. G **37**, 075021 (2010)
13. K. Hagiwara, H. Murayama, I. Watanabe, Nucl. Phys. B **367**, 257 (1991)
14. H. Murayama, I. Watanabe, K. Hagiwara, KEK-Report 91-11 (1992)
15. T. Stelzer, W.F. Long, Comput. Phys. Commun. **81**, 357 (1994)
16. F. Maltoni, T. Stelzer, J. High Energy Phys. **0302**, 027 (2003)
17. J. Alwall et al., J. High Energy Phys. **0709**, 028 (2007)