UCLouvain
Institut de recherche en mathématique et physique
Centre de Cosmologie, Physique des Particules et Phénoménologie

MCnet

MG5amC: customise output

Olivier Mattelaer

Creation of matrix-element in …

- Two types of work to be done
  - ➡ Creates the “basic” functionS that are the “real” kernel of the computation
    - ixxxx
    - ffv2_3, ….
  - ➡ Creates the files that calls such function one by one
    - Includes also initialization/rambo/…

CUDA output

Generate p p > t t~
output standalone_gpu

```
[PROC_SA_GPU_sm_1]$ ls SubProcesses/
Makefile              P1_Sigma_sm_gg_ttx     cmeck.cc              perf.py               runTest.cc            timermap.h
Memory.h              P1_Sigma_sm_uux_ttx    nvtx.h                profile.sh            timer.h
[PROC_SA_GPU_sm_1]$ ls SubProcesses/P1_Sigma_sm_gg_ttx/
CPPProcess.cc  Makefile      Memory.h       cmeck_sa.cc    gCPPProcess.cu  gCPPProcess.h  gcheck_sa.cu  nvtx.h        perf.py       runTest.cc    timer.h       timermap.h
[PROC_SA_GPU_sm_1]$ ls src
HelAmps_sm.cu         Makefile              Parameters_sm.h       mgOnGpuConfig.h       rambo.cc              read_slha.cc
HelAmps_sm.h          Parameters_sm.cc      grambo.cu             mgOnGpuTypes.h        rambo.h               read_slha.h
[PROC_SA_GPU_sm_1]$
```

A lot of file here are processes independent

- perf.py
- grambo.cu
- read_slha.cc

Typically template file copy/paste
To output

Some file depends on the model and/or process

- Parameters_sm.X (madgraph)
- Helamps_sm.X (aloha)
- (g)CPPProcess.X (madgraph)

Heavy logic going on to determine those functions.

Will not cover “Parameters_sm.X” here (do not think that we need to change that one)

Environment setup

- Madgraph does not use git (yet)
- So please install bzr (bazaar)
- bzr branch lp:~maddevelopers/mg5amcnlo/2.7.0_gpu

- actually not 100% needed since we will create a PLUGIN for simplicity here but we need that specific branch for the rest of the workshop.

Step 1:

- First modification to the code would be to add support for a new “output standalone_cuda” command within MG5aMC
  - ➡ First step make that command identical to the current “standalone_gpu” code
- Instruction for that:
  - ➡ cd PLUGIN
  - ➡ untar directory from indico

```
[TUTO_SA_OUTPUT]$ ls
__init__.py     output.py
```

Edit metadata __init__.py

```
File Edit Options Buffers Tools Python Help
## import the required files
# example: import maddm_interface as maddm_interface # local file
#          import madgraph.various.cluster as cluster #MG5 distribution file
# Three types of functionality are allowed in a plugin
#   1. new output mode
#   2. new cluster support
#   3. new interface

# 1. Define new output mode
#    example: new_output = {'myformat': MYCLASS}
#    madgraph will then allow the command "output myformat PATH"
#    MYCLASS should inherated of the class madgraph.iolibs.export_v4.VirtualExporter
import output
new_output = {'standalone_cuda':output.CUDAExporter}

# 2. Define new way to handle the cluster.
#    example new_cluster = {'mycluster': MYCLUSTERCLASS}
#    allow "set cluster_type mycluster" in madgraph
#    MYCLUSTERCLASS should inherated from madgraph.various.cluster.Cluster
new_cluster = {}


# 3. Define a new interface (allow to add/modify MG5 command)
#    This can be activated via ./bin/mg5_aMC --mode=PLUGINNAME
## Put None if no dedicated command are required
new_interface = None


########################## CONTROL VARIABLE ####################################
__author__ = ''
__email__ = ''
__version__ = (1,0,0)
minimal_mg5amcnlo_version = (2,3,4)
maximal_mg5amcnlo_version = (1000,1000,1000)
latest_validated_version = (2,4,0)
```

Key: name of output

value: class to use

Check content of output.py

```
import madgraph.iolibs.export_cpp as export_cpp
import madgraph.various.misc as misc
from madgraph import MG5DIR

import os
pjoin = os.path.join

class MY_CPP_Standalone(export_cpp.ProcessExporterGPU):
    # class structure information
    # object
    #  - VirtualExporter(object) [in madgraph/iolibs/export_v4.py]
    #  - ProcessExporterCPP(VirtualExporter) [in madgraph/iolibs/export_cpp.py]
    #  - ProcessExporterGPU(ProcessExporterCPP) [in madgraph/iolibs/export_cpp.py]
    #      Note: only change class attribute
    #  - MY_CPP_Standalone(ProcessExporterGPU)
    #      This class
```

```
def __init__(self, *args, **opts):
    misc.sprint("Initialise the exporter")
    return super(MY_CPP_Standalone, self).__init__(*args, **opts)

def copy_template(self, model):

    misc.sprint("initialise the directory")
    return super(MY_CPP_Standalone, self).copy_template(model)


def generate_subprocess_directory(self, subproc_group,
                                  fortran_model, me=None):

    misc.sprint('create the directory')
    return super(MY_CPP_Standalone, self).generate_subprocess_directory(subproc_group, fortran_model, me)
```

Step 1: create an output

This PLUGIN does not do anything new compare to the current CUDA class (i.e. epoch 2). So the output will be identical.

Run madgraph command:

```
generate p p > t t~
output standalone_gpu TEST_SA_GPU
generate p p > t t~
output standalone_cuda TEST_SA_CUDA
```

Check that created directory are identical

```
[2.7.0_gpu]$ diff -r TEST_SA_GPU/ TEST_SA_CUDA/
[2.7.0_gpu]$
```

Step 2: modify ALOHA

In the output file, uncomment the line for specifying a new create_model_class

```
#For model/aloha exporter (typically not used)
create_model_class =  export_cpp.UFOModelConverterGPU
import PLUGIN.TUTO_SA_OUTPUT.model_hangling as model_handling
create_model_class = model_handling.UFOModelConverterGPU
```

- This still does not change anything (still dummy subclass)
  - ➡ Freedom to edit any code

model_handling.py

```
class ALOHAWriterForGPU(aloha_writers.ALOHAWriterForGPU):

    extension = '.cu'
    prefix ='__device__'
    realoperator = '.real()'
    imagoperator = '.imag()'
    ci_definition = 'cxtype cI = cxtype(0., 1.);\n'

    type2def = {}
    type2def['int'] = 'int '
    type2def['double'] = 'fptype '
    type2def['complex'] = 'cxtype '
    type2def['pointer_vertex'] = '*' # using complex<double> * vertex)
    type2def['pointer_coup'] = ''


class  UFOModelConverterGPU(export_cpp.UFOModelConverterGPU):

    #aloha_writer = 'cudac' #this was the default mode assigned to GPU
    aloha_writer = ALOHAWriterForGPU # this is equivalent to the above line but allow to edit it obviously.
```

PROC_SA_GPU_sm_0 > src > HelAmps_sm.cu

```cpp
//==========================================================================
// This file has been automatically generated for C++ Standalone by
// MadGraph5_aMC@NLO v. 2.9.5, 2021-08-22
// By the MadGraph5_aMC@NLO Development Team
// Visit launchpad.net/madgraph5 and amcatnlo.web.cern.ch
//==========================================================================

#include <cmath>
#include <cstring>
#include <cstdlib>
#include <iomanip>
#include <iostream>

#include "mgOnGpuConfig.h"
#include "mgOnGpuTypes.h"

mgDebugDeclare();

namespace MG5_sm
{

using mgOnGpu::nw6;

//--------------------------------------------------------------------------

__device__
inline const fptype& pIparIp4Ievt(const fptype * momenta1d,  // input: mo
const int ipar,
const int ip4,
const int ievt)
{
  // mapping for the various scheme AOS, OSA, ...

  using mgOnGpu::np4;
  using mgOnGpu::npar;
  const int neppM = mgOnGpu::neppM;  // ASA layout: constant at compile-t
  fptype (*momenta)[npar][np4][neppM] = (fptype (*)[npar][np4][neppM])
      momenta1d; // cast to multiD array pointer (AOSOA)
  const int ipagM = ievt/neppM;  // #eventpage in this iteration
  const int ieppM = ievt%neppM;  // #event in the current eventpage in th
  // return allmomenta[ipagM*npar*np4*neppM + ipar*neppM*np4 + ip4*neppM
  // ieppM]; // AOSOA[ipagM][ipar][ip4][ieppM]
  return momenta[ipagM][ipar][ip4][ieppM];
}

//--------------------------------------------------------------------------

__device__ void ixxxxx(const fptype * allmomenta, const fptype& fmass, co
```

Template input from
madgraph/iolibs/template_files/gpu/
cpp_hel_amps_cc.inc
Note this is a python template

Can be changed in
model_handling.py
In UFOModelConverterGPU class

Template input from
aloha/template_files/gpu/helas.cu
Quite long file with all the function
finishing with “xxxx”
(initial/final state function))
Pure copy/paste
Template path can be changed from
model_handling.py

```
621     fo[4] = chi1;
622     fo[5] = chi0;
623   }
624   return;
625 }
626 __device__ void FFV2_0(const cxtype F1[], const cxtype F2[], const cxtype V3[],
627     const cxtype COUP, cxtype * vertex)
628 {
629   cxtype cI = cxtype(0., 1.);
630   cxtype TMP0;
631   TMP0 = (F1[2] * (F2[4] * (V3[2] + V3[5]) + F2[5] * (V3[3] + cI * (V3[4]))) +
632       F1[3] * (F2[4] * (V3[3] - cI * (V3[4])) + F2[5] * (V3[2] - V3[5])));
633   (*vertex) = COUP * - cI * TMP0;
634 }
635
636
637 __device__ void FFV2_3(const cxtype F1[], const cxtype F2[], const cxtype COUP,
638     const fptype M3, const fptype W3, cxtype V3[])
639 {
640   cxtype cI = cxtype(0., 1.);
641   fptype OM3;
642   fptype P3[4];
643   cxtype TMP1;
644   cxtype denom;
645   OM3 = 0.;
646   if (M3 != 0.)
647     OM3 = 1./(M3 * M3);
648   V3[0] = +F1[0] + F2[0];
649   V3[1] = +F1[1] + F2[1];
650   P3[0] = -V3[0].real();
651   P3[1] = -V3[1].real();
652   P3[2] = -V3[1].imag();
653   P3[3] = -V3[0].imag();
654   TMP1 = (F1[2] * (F2[4] * (P3[0] + P3[3]) + F2[5] * (P3[1] + cI * (P3[2]))) +
655       F1[3] * (F2[4] * (P3[1] - cI * (P3[2])) + F2[5] * (P3[0] - P3[3])));
656   denom = COUP/((P3[0] * P3[0]) - (P3[1] * P3[1]) - (P3[2] * P3[2]) - (P3[3] *
657       P3[3]) - M3 * (M3 - cI * W3));
658   V3[2] = denom * (-cI) * (F1[2] * F2[4] + F1[3] * F2[5] - P3[0] * OM3 * TMP1);
659   V3[3] = denom * (-cI) * (-F1[2] * F2[5] - F1[3] * F2[4] - P3[1] * OM3 *
660       TMP1);
661   V3[4] = denom * (-cI) * (-cI * (F1[2] * F2[5]) + cI * (F1[3] * F2[4]) - P3[2]
662       * OM3 * TMP1);
663   V3[5] = denom * (-cI) * (-F1[2] * F2[4] - P3[3] * OM3 * TMP1 + F1[3] *
664       F2[5]);
665 }
666
667
668 __device__ void FFV4_0(const cxtype F1[], const cxtype F2[], const cxtype V3[],
669     const cxtype COUP, cxtype * vertex)
```

End of previous template

Amplitude function
Generated line by line by python code
(see later)

Propagator function
Generated line by line by python code
(see later)
Same structure as above

```
__device__ void FFV2_0(const cxtype F1[], const cxtype F2[], const cxtype V3[],
    const cxtype COUP, cxtype * vertex)
{
  cxtype cI = cxtype(0., 1.);
  cxtype TMP0;
  TMP0 = (F1[2] * (F2[4] * (V3[2] + V3[5]) + F2[5] * (V3[3] + cI * (V3[4]))) +
      F1[3] * (F2[4] * (V3[3] - cI * (V3[4])) + F2[5] * (V3[2] - V3[5])));
  (*vertex) = COUP * - cI * TMP0;
}
```

This function is done via a series of sub-function that can be modified in model_handling.py

```
__device__ void FFV2_0(const cxtype F1[], const cxtype F2[], const cxtype V3[],
    const cxtype COUP, cxtype * vertex)
{
```

Output of function get_header_txt

From class attribute self.prefix

```
  cxtype cI = cxtype(0., 1.);
  cxtype TMP0;
```

From function get_declaration_txt

From class attribute self.type2def

From class attribute self.ci_definition

```
  TMP0 = (F1[2] * (F2[4] * (V3[2] + V3[5]) + F2[5] * (V3[3] + cI * (V3[4]))) +
      F1[3] * (F2[4] * (V3[3] - cI * (V3[4])) + F2[5] * (V3[2] - V3[5])));
  (*vertex) = COUP * - cI * TMP0;
```

From function define_expression

All the multiplication/addition are from “write_obj”
Using change_var_format
get_fct_format,
shit_indices

```
}
```

From function get_foot_txt

ALOHAWriterForGPU

Class ALOHAWriterForGPU

source code

```
   WriteALOHA --+
                |
ALOHAWriterForCPP ---
                    |
                ALOHAWriterForGPU
```

Nested Classes	[hide private]
Inherited from ALOHAWriterForCPP: Writer	

Instance Methods		[hide private]
	get_h_text(self, couplings=None) Return the full contents of the .h file	source code
	write_obj_Add_test(self, obj, prefactor=True) Turns addvariable into a string	source code
	write_MultVariable_test(self, obj, prefactor=True) Turn a multvariable into a string	source code
	get_header_txt(self, name=None, couplings=None, mode='') Define the Header of the fortran file.	source code
Inherited from ALOHAWriterForCPP: change_number_format, change_var_format, define_expression, define_symmetry, get_declaration_txt, get_fct_format, get_foot_txt, get_momenta_txt, get_one_momenta_def, shift_indices, write, write_combined, write_combined_cc **Inherited from** WriteALOHA: __init__, define_argument_list, define_content, get_P_sign, get_momentum_conservation_sign, make_call_list, make_declaration_list, pass_to_HELAS, write_MultContainer, write_MultVariable, write_indices_part, write_obj, write_obj_Add, write_variable, write_variable_id		

You can overclass all those functions within model_handling.py

In itself, gpu only overwrites get_header_txt
(and some class attributes related to formatting) -> see model_handling.py

You can also decide to modify directly one of the mother class and then use bazaar to push the change in that branch (or do a merge request) —if this make more sense—

Step 3: change the function call

The modified functions are called from CPPProcess.cc
Currently with “beautiful” ifdef;

```
#ifdef __CUDACC__
  vxxxxx(allmomenta, 0., cHel[ihel][0], -1, w[0], 0);
#else
  vxxxxx(allmomenta, 0., cHel[ihel][0], -1, w[0], ievt, 0);
#endif


#ifdef __CUDACC__
  vxxxxx(allmomenta, 0., cHel[ihel][1], -1, w[1], 1);
#else
  vxxxxx(allmomenta, 0., cHel[ihel][1], -1, w[1], ievt, 1);
#endif


#ifdef __CUDACC__
  oxxxxx(allmomenta, cIPD[0], cHel[ihel][2], +1, w[2], 2);
#else
  oxxxxx(allmomenta, cIPD[0], cHel[ihel][2], +1, w[2], ievt, 2);
#endif
```

So next step is to modify those type of lines
-> like remove the ifdef (challenge for Andrea)
-> pass to kernel call (challenge for Stefan)

HelasCallWriter

Those line are written by the “HelasCallWriter”. Let subclass this one as well.
By uncommenting the line defining aloha_exporter in output.py

```
# typically not defined but usufull for this tutorial the class for writing helas routine
#aloha_exporter = None
#aloha_exporter = model_handling.GPUFOHelasCallWriter
```

Relevant function:
- get_external -> defines the line(s) call for the external particle
- get_wavefunction_call -> defines the line(s) call for the propagator
- get_amplitude_call -> defines the line(s) call for the amplitude

A bunch of caching is done within the function:
- generate_helas_call (not nicely factorised: template for the helas call itself)

The “main” driver for the writing of the matrix-element is
- get_matrix_element_call (call all the above as needed and defined the color matrix)
  -> where we will need to add the multi-channel computation!

Line by line function

```
#ifdef __CUDACC__
ixxxxx(allmomenta, cIPD[0], cHel[ihel][3], -1, w[3], 3);
#else
ixxxxx(allmomenta, cIPD[0], cHel[ihel][3], -1, w[3], ievt, 3);
#endif

VVV1P0_1(w[0], w[1], cxtype(cIPC[0], cIPC[1]), 0., 0., w[4]);
// Amplitude(s) for diagram number 1
FFV1_0(w[3], w[2], w[4], cxtype(cIPC[2], cIPC[3]), &amp;amp[0]);
jamp[0] += +cxtype(0, 1) * amp[0];
jamp[1] += -cxtype(0, 1) * amp[0];
FFV1_1(w[2], w[0], cxtype(cIPC[2], cIPC[3]), cIPD[0], cIPD[1], w[4]);
// Amplitude(s) for diagram number 2
FFV1_0(w[3], w[4], w[1], cxtype(cIPC[2], cIPC[3]), &amp;amp[0]);
jamp[0] += -amp[0];
FFV1_2(w[3], w[0], cxtype(cIPC[2], cIPC[3]), cIPD[0], cIPD[1], w[4]);
// Amplitude(s) for diagram number 3
FFV1_0(w[4], w[2], w[1], cxtype(cIPC[2], cIPC[3]), &amp;amp[0]);
jamp[1] += -amp[0];
// double CPPProcess::matrix_1_gg_ttx() {
```

get_external

get_external_line

get_wavefunction_call

get_amplitude_call

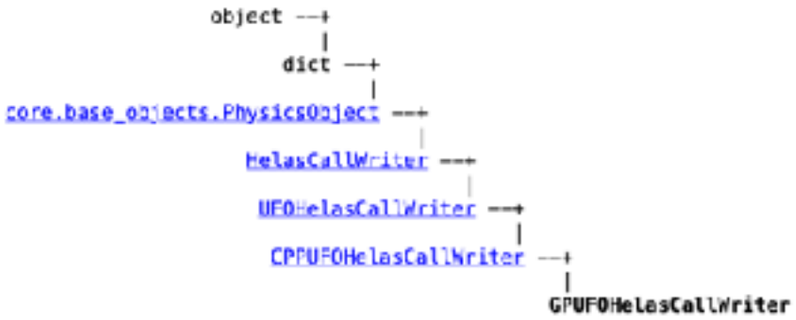
get_matrix_element_call

format_coupling

```
export_cpp.OneProcessExporterGPU.coeff(*coeff)
```

Class GPUFOHelasCallWriter

source code



Nested Classes

[hide private]

Inherited from core.base_objects.PhysicsObject: PhysicsObjectError

Instance Methods

[hide private]

	format_coupling(self, call) Format the coupling so any minus signs are put in front	source code
	get_external(self, wf, argument)	source code
	get_external_line(self, wf, argument)	source code
	generate_helas_call(self, argument) Routine for automatic generation of C++ Helas calls according to just the spin structure of the interaction.	source code
	get_matrix_element_calls(self, matrix_element, color_amplitudes) Return a list of strings, corresponding to the Helas calls for the matrix element	source code

Inherited from UFOHelasCallWriter: get_amplitude_call, get_wavefunction_call, write_factor

Inherited from HelasCallWriter: __init__, add_amplitude, add_wavefunction, default_setup, filter, get_amplitude_calls, get_born_ct_helas_calls, get_loop_amp_helas_calls, get_loop_matrix_element_calls, get_model_name, get_sorted_keys, get_sqso_target_skip_code, get_wavefunction_calls

Inherited from core.base_objects.PhysicsObject: __getitem__, __repr__, __str__, get, is_valid_prop, set

Inherited from dict: __cmp__, __contains__, __delitem__, __eq__, __ge__, __getattribute__, __gt__, __iter__, __le__, __len__, __lt__, __ne__, __new__, __setitem__, __sizeof__, clear, copy, fromkeys, has_key, items, iteritems, iterkeys, itervalues, keys, pop, popitem, setdefault, update, values, viewitems, viewkeys, viewvalues

Inherited from object: __delattr__, __format__, __reduce__, __reduce_ex__, __setattr__, __subclasshook__

Static Methods

[hide private]

Inherited from HelasCallWriter: customize_argument_for_all_other_helas_object, default_customize_argument_for_all_other_helas_object

Class Variables

[hide private]

	findcoupling = re.compile(r'pars->(-*[\d\w_]+)\s*,')

Inherited from HelasCallWriter: mother_dict

Inherited from dict: __hash__

Step 4: change the rest of that file

```
oneprocessclass = export_cpp.OneProcessExporterGPU # responsible for P directory
```

```
FFV1_0(w[4], w[2], w[1], cxtype(cIPC[2], cIPC[3]), &amp;amp[0]);
jamp[1] += -amp[0];
// double CPPProcess::matrix_1_gg_ttx() {



// Local variables

// The color matrix;
static const double denom[ncolor] = {3, 3};
static const double cf[ncolor][ncolor] = {{16, -2}, {-2, 16}};
```

```
// Sum and square the color flows to get the matrix element
for(int icol = 0; icol < ncolor; icol++ )
{
  cxtype ztemp = cxmake(0, 0);
  for(int jcol = 0; jcol < ncolor; jcol++ )
    ztemp = ztemp + cf[icol][jcol] * jamp[jcol];
  meHelSum = meHelSum + cxreal(ztemp * conj(jamp[icol]))/denom[icol];
}

// Store the leading color flows for choice of color
// for(i=0;i < ncolor; i++)
// jamp2[0][i] += real(jamp[i]*conj(jamp[i]));

mgDebug(1, __FUNCTION__);
return;
```

get_color_matrix_line

From template
template_files/gpu/
process_matrix.inc
Template picked via class variable:

single_process_template
(model_handling)

Basic subclassing example

As before this is keeping everything identical to mother class

```
class OneProcessExporterGPU(export_cpp.OneProcessExporterGPU):

        # Static variables (for inheritance)
    process_dir = '.'
    include_dir = '.'
    template_path = os.path.join(_file_path, 'iolibs', 'template_files')
    __template_path = os.path.join(_file_path, 'iolibs', 'template_files')
    process_template_h = 'gpu/process_h.inc'
    process_template_cc = 'gpu/process_cc.inc'
    process_class_template = 'gpu/process_class.inc'
    process_definition_template = 'gpu/process_function_definitions.inc'
    process_wavefunction_template = 'cpp_process_wavefunctions.inc'
    process_sigmaKin_function_template = 'gpu/process_sigmaKin_function.inc'
    single_process_template = 'gpu/process_matrix.inc'
    cc_ext = 'cu'
```

Typical method of work:

1. Find which template you need to edit
2. Copy the template in your PLUGIN directory
3. Edit it and modify the associated class attribute
4. No Template -> need to check with function does the work then

Diagram definition

Class OneProcessExporterGPU

source code

Class to take care of exporting a set of matrix elements to C++ format.

Nested Classes [hide private]

Inherited from OneProcessExporterCPP: ProcessExporterCPPError

Instance Methods [hide private]

	`__init__(self, *args, **opts)` Initiate with matrix elements, helas call writer, process string, path.	source code
	`generate_process_files(self)` Generate the .h and .cc files needed for C++, for the processes described by multi_matrix_element	source code
	`edit_check_sa(self)`	source code
	`edit_mgonGPU(self)`	source code
	`get_initProc_lines(self, matrix_element, color_amplitudes)` Get initProc_lines for function definition for Pythia 8 .cc file	source code
	`get_reset_jamp_lines(self, color_amplitudes)` Get lines to reset jamps	source code
	`get_process_function_definitions(self, write=True)` The complete Pythia 8 class definition for the process	source code
	`get_process_class_definitions(self, write=True)` The complete class definition for the process	source code
	`get_all_sigmaKin_lines(self, color_amplitudes, class_name)` Get sigmaKin_process for all subprocesses for Pythia 8 .cc file	source code
	`write_process_h_file(self, writer)` Write the class definition (.h) file for the process	source code
	`write_process_cc_file(self, writer)` Write the class member definition (.cc) file for the process described by matrix_element	source code

Inherited from OneProcessExporterCPP: get_calculate_wavefunctions, get_class_specific_definition_matrix, get_color_matrix_lines, get_default_converter, get_den_factor_line, get_helicity_matrix, get_jamp_lines, get_matrix_single_process, get_process_info_lines, get_process_name, get_sigmaHat_lines, get_sigmaKin_lines, get_sigmaKin_single_process

Inherited from object: __delattr__, __format__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __sizeof__, __str__, __subclasshook__

Class Methods [hide private]

Inherited from OneProcessExporterCPP: read_template_file

Break down of the file

```
//--------------------------------------------------------------------------
// This file has been automatically generated for C++ Standalone by
// MadGraph5_aMC@NLO v. 2.9.5, 2021-08-22
// By the MadGraph5_aMC@NLO Development Team
// Visit launchpad.net/madgraph5 and amcatnlo.web.cern.ch
//--------------------------------------------------------------------------


#include "../../src/HelAmps_sm.cu"


#include <algorithm>
#include <iostream>
#include "mgOnGpuTypes.h"
#include "mgOnGpuConfig.h"

#include "gCPPProcess.h"

//--------------------------------------------------------------------------
// Class member functions for calculating the matrix elements for
// Process: g g > t t~ WEIGHTED<=2 @1

#ifdef __CUDACC__
namespace gProc
#else
namespace Proc
#endif
{

using mgOnGpu::np4;  // 4: the dimension of 4-momenta (E,px,py,pz)
using mgOnGpu::npar;  // number of particles in total (initial + final)
using mgOnGpu::ncomb;  // number of helicity combinations

#ifdef __CUDACC__
__device__ __constant__ int cHel[ncomb][npar];
__device__ __constant__ fptype cIPC[4];
__device__ __constant__ fptype cIPD[2];
__device__ __constant__ int cNGoodHel[1];
__device__ __constant__ int cGoodHel[ncomb];
#else
static int cHel[ncomb][npar];
static fptype cIPC[4];
static fptype cIPD[2];
#endif

//--------------------------------------------------------------------------
using mgOnGpu::nwf;
using mgOnGpu::nw6;
//--------------------------------------------------------------------------
```

Template:
process_cc.inc

```
//--------------------------------------------------------------------------
// This file has been automatically generated for C++ Standalo
%(info_lines)s
//--------------------------------------------------------------------------

%(hel_amps_def)s


#include <algorithm>
#include <iostream>
#include "mgOnGpuTypes.h"
#include "mgOnGpuConfig.h"

#include "gCPPProcess.h"

%(process_function_definitions)s
```

```
self.get_process_function_definitions
```

Template:
process_function_definition.i

Value feed via function o
The generated process

```
// of |M|^2 over helicities for the given event

__device__ void calculate_wavefunctions(int ihel, const fptype
    fptype &meHelSum
#ifndef __CUDACC__
, const int ievt
#endif
)
{
  using namespace MG5_sm;
  mgDebug(0, __FUNCTION__);
  cxtype amp[1];  // was 3
  const int ncolor = 2;
  cxtype jamp[ncolor];
  // Calculate wavefunctions for all processes
  using namespace MG5_sm;
  cxtype w[nwf][nw6];
  for(int i = 0; i < 2; i++ )
  {
    jamp[i] = cxtype(0., 0.);
  }

#ifdef __CUDACC__
  vxxxxx(allmomenta, 0., cHel[ihel][0], -1, w[0], 0);
#else
  vxxxxx(allmomenta, 0., cHel[ihel][0], -1, w[0], ievt, 0);
#endif
```

Not from template but from:

`get_process_class_definitions`

Template:
process_function_definition.inc

```
CPPProcess::CPPProcess(int numiterations, int gpublocks, int gputhreads,
bool verbose, bool debug)
: m_numiterations(numiterations), gpu_nblocks(gpublocks),
gpu_nthreads(gputhreads), m_verbose(verbose),
dim(gpu_nblocks * gpu_nthreads)
{

  // Helicities for the process - nodim
  static const int tHel[ncomb][nexternal] = {{-1, -1, -1, -1}, {-1, -1, -1, 1},
      {-1, -1, 1, -1}, {-1, -1, 1, 1}, {-1, 1, -1, -1}, {-1, 1, -1, 1}, {-1, 1,
      1, -1}, {-1, 1, 1, 1}, {1, -1, -1, -1}, {1, -1, -1, 1}, {1, -1, 1, -1},
      {1, -1, 1, 1}, {1, 1, -1, -1}, {1, 1, -1, 1}, {1, 1, 1, -1}, {1, 1, 1,
      1}};
#ifdef __CUDACC__
  checkCuda(cudaMemcpyToSymbol(cHel, tHel, ncomb * nexternal * sizeof(int)));
#else
  memcpy(cHel, tHel, ncomb * nexternal * sizeof(int));
#endif

  // SANITY CHECK: GPU memory usage may be based on casts of fptype[2] to cxtype
  assert(sizeof(cxtype) == 2 * sizeof(fptype));
}

CPPProcess::~CPPProcess() {}

const std::vector<fptype> &CPPProcess::getMasses() const {return mME;}

//--------------------------------------------------------------------------
// Initialize process.
```

Template:
process_function_definition.inc

Value feed via function of
The generated process

```
//--------------------------------------------------------------------------
// Initialize process.

void CPPProcess::initProc(string param_card_name)
{
  // Instantiate the model class and set parameters that stay fixed
  pars = Parameters_sm::getInstance();
  SLHAReader slha(param_card_name, m_verbose);
  pars->setIndependentParameters(slha);
  pars->setIndependentCouplings();
  if (m_verbose)
  {
    pars->printIndependentParameters();
    pars->printIndependentCouplings();
  }
  pars->setDependentParameters();
  pars->setDependentCouplings();
  // Set external particle masses for this matrix element
  mME.push_back(pars->ZERO);
  mME.push_back(pars->ZERO);
  mME.push_back(pars->mdl_MT);
  mME.push_back(pars->mdl_MT);

  static cxtype tIPC[2] = {pars->GC_10, pars->GC_11};
  static double tIPD[2] = {pars->mdl_MT, pars->mdl_WT};

#ifdef __CUDACC__
  checkCuda(cudaMemcpyToSymbol(cIPC, tIPC, 2 * sizeof(cxtype)));
  checkCuda(cudaMemcpyToSymbol(cIPD, tIPD, 2 * sizeof(fptype)));
#else
  memcpy(cIPC, tIPC, 2 * sizeof(cxtype));
  memcpy(cIPD, tIPD, 2 * sizeof(fptype));
#endif
```

Template:
process_function_definition.inc

Value feed via function of
The generated process

```cpp
//--------------------------------------------------------------------------

#ifdef __CUDACC__
__global__
void sigmaKin_getGoodHel(const fptype * allmomenta,  // input: momenta as AOSOA[npagM][npar][4][neppM] with nevt=npagM*neppM
bool * isGoodHel)  // output: isGoodHel[ncomb] - device array
{
  const int nprocesses = 1;  // FIXME: assume process.nprocesses == 1
  fptype meHelSum[nprocesses] = {0};  // all zeros
  fptype meHelSumLast = 0;
  for (int ihel = 0; ihel < ncomb; ihel++ )
  {
    // NB: calculate_wavefunctions ADDS |M|^2 for a given ihel to the running
    // sum of |M|^2 over helicities for the given event
    calculate_wavefunctions(ihel, allmomenta, meHelSum[0]);
    if (meHelSum[0] != meHelSumLast)
    {
      isGoodHel[ihel] = true;
      meHelSumLast = meHelSum[0];
    }
  }
}
#endif

//--------------------------------------------------------------------------

#ifdef __CUDACC__
void sigmaKin_setGoodHel(const bool * isGoodHel)  // input: isGoodHel[ncomb] - host array
{
  int nGoodHel[1] = {0};
  int goodHel[ncomb] = {0};
  for (int ihel = 0; ihel < ncomb; ihel++ )
  {
    // std::cout << "sigmaKin_setGoodHel ihel=" << ihel << ( isGoodHel[ihel] ?
    // " true" : " false" ) << std::endl;
    if (isGoodHel[ihel])
    {
      goodHel[nGoodHel[0]] = ihel;
      nGoodHel[0]++;
    }
  }
  checkCuda(cudaMemcpyToSymbol(cNGoodHel, nGoodHel, sizeof(int)));
  checkCuda(cudaMemcpyToSymbol(cGoodHel, goodHel, ncomb * sizeof(int)));
}
#endif
```

Template:
process_function_definition.inc

```
__global__ void sigmaKin(const fptype * allmomenta, fptype * allMEs
#ifndef __CUDACC__
, const int nevt  // input: #events (for cuda: nevt == ndim == gpublo
#endif
)
{
  // Set the parameters which change event by event
  // Need to discuss this with Stefan
  // pars->setDependentParameters();
  // pars->setDependentCouplings();

#ifndef __CUDACC__
  const int maxtry = 10;
  static unsigned long long sigmakin_itry = 0;  // first iteration ove
  static bool sigmakin_goodhel[ncomb] = {false};
#endif

  // Reset color flows


  // start sigmakin lines
```

Template:
process_function_definition.inc

```
  mgDebugInitialise();
  // Set the parameters which change event by event
  // Need to discuss this with Stefan
  // pars->setDependentParameters();
  // pars->setDependentCouplings();
  // Reset color flows

#ifndef __CUDACC__
  //** START LOOP ON IEVT **
  for (int ievt = 0; ievt < nevt; ++ievt)
#endif
  {
#ifdef __CUDACC__
    const int idim = blockDim.x * blockIdx.x + threadIdx.x;  // eventA
    const int ievt = idim;
    // printf( "sigmakin: ievt %d\n", ievt );
#endif

    // Denominators: spins, colors and identical particles
    const int nprocesses = 1;  // FIXME: assume process.nprocesses ==
    const int denominators[1] = {256};

    // Reset the "matrix elements" - running sums of |M|^2 over helici
    // the given event
```

Template:
process_sigmakin_function.inc