CUDAnuSQuIDS

Generated by Doxygen 1.8.11

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

Class Documentation

2.1 cudanusquids::CudaNusquids< NFLVS_, body_t, Op_t > Class Template Reference

This is the main class of of the library.

Public Member Functions

- template<class Nus_t >
  CudaNusquids (Nus_t &other)

  Constructor to change body type.
- CudaNusquids (std::shared_ptr< ParameterObject > &params, int batchsizeLimit)

  Construct CudaNusquids from ParameterObject.
- CudaNusquids (CudaNusquids &&other)

  Move constructor.
- CudaNusquids & operator= (CudaNusquids &&other)

  Move assignment operator.
- int getCosineDeviceIndex (int index_path) const

  Return device Id of GPU which processes the index_path-th path.
- double EvalMassAtNode (int flavor, int index_path, int index_rho, int index_energy)

  Get expectation value in mass Basis.
- double EvalFlavorAtNode (int flavor, int index_path, int index_rho, int index_energy)

  Get expectation value in flavor Basis.
- void evolve ()

  Evolve neutrinos along the specified paths from path begin to path end.
- void mixingParametersChanged ()

  Notify a CudaNusquids instance that mixing parameters in parameter object changed.
- void simulationFlagsChanged ()

  Notify a CudaNusquids instance that simulation parameters which enable / disable physics in parameter object changed.
- void initialFluxChanged ()

  Notify a CudaNusquids instance that initial flux in parameter object changed.
- void additionalDataChanged ()

  Notify a CudaNusquids instance that additional data in parameter object changed.
- void setBody (const Body &body_, int deviceId)

  Set body for GPU with device id deviceId.
- void setTracks (const std::vector< Track > &tracks_)

  Set neutrino tracks. Must be one track per path.
- ode::RKstats getRKstats (int index_path) const

  Get Runge-Kutta stats after evolution.

2.1.1 Detailed Description

template<int NFLVS_, class body_t, class Op_t = PhysicsOps>
class cudanusquids::CudaNusquids< NFLVS_, body_t, Op_t >

This is the main class of of the library.

Parameters

*NFLVS_*	Number of neutrino flavors
body_t	The body type which provides density lookup
Op_t	The operators for physics simulation. (optional, default behaviour provided by struct PhysicsOps)

2.1.2 Constructor & Destructor Documentation

2.1.2.1 template<int NFLVS_, class body_t , class Op_t = PhysicsOps> template<class Nus_t > cudanusquids::CudaNusquids< NFLVS_, body_t, Op_t >::CudaNusquids (Nus_t & *other*) [inline]

Constructor to change body type.

other will be moved to ∗this. other must not be accessed afterwards.

2.1.2.2 template<int NFLVS_, class body_t , class Op_t = PhysicsOps> cudanusquids::CudaNusquids< NFLVS_, body_t, Op_t >::CudaNusquids (std::shared_ptr< ParameterObject > & *params*, int *batchsizeLimit*) [inline]

Construct CudaNusquids from ParameterObject.

batchsizeLimit determines the maximum number of neutrino paths, which are processed simultaneously per GPU. If the number of paths is greater than batchsizeLimit, paths are processed in chunks of size batchsizeLimit This effectively controls the GPU memory usage, since there only has to be enough memory to process batchsizeLimit paths

2.1.3 Member Function Documentation

2.1.3.1 template<int NFLVS_, class body_t , class Op_t = PhysicsOps> void cudanusquids::CudaNusquids< NFLVS_, body_t, Op_t >::additionalDataChanged ( ) [inline]

Notify a CudaNusquids instance that additional data in parameter object changed.

Needs to be called before CudaNusquids::evolve() after calls to ParameterObject::registerAdditionalData ParameterObject::clearAdditionalData

2.1.3.2 template<int NFLVS_, class body_t , class Op_t = PhysicsOps> ode::RKstats cudanusquids::CudaNusquids< NFLVS_, body_t, Op_t >::getRKstats (int *index_path*) const [inline]

Get Runge-Kutta stats after evolution.

members of RKstats: unsigned int steps; // number of required steps unsigned int repeats; // number of repeated steps Status status; // success or failure

2.1.3.3 template<int NFLVS_, class body_t , class Op_t = PhysicsOps> void cudanusquids::CudaNusquids< NFLVS_, body_t, Op_t >::initialFluxChanged ( ) [inline]

Notify a CudaNusquids instance that initial flux in parameter object changed.

Needs to be called before CudaNusquids::evolve() after calls to ParameterObject::setInitialFlux

2.1.3.4 template<int NFLVS_, class body_t , class Op_t = PhysicsOps> void cudanusquids::CudaNusquids< NFLVS_, body_t, Op_t >::mixingParametersChanged ( ) [inline]

Notify a CudaNusquids instance that mixing parameters in parameter object changed.

Needs to be called before CudaNusquids::evolve() after calls to ParameterObject::Set_MixingAngle, ParameterObject::Set_SquareMassDifference, ParameterObject::Set_CPPhase

2.1.3.5 template<int NFLVS_, class body_t , class Op_t = PhysicsOps> void cudanusquids::CudaNusquids< NFLVS_, body_t, Op_t >::setBody (const Body & *body_*, int *deviceId*) [inline]

Set body for GPU with device id deviceId.

The deviceId must be identical to the one for which the body was created

2.1.3.6 template<int NFLVS_, class body_t , class Op_t = PhysicsOps> void cudanusquids::CudaNusquids< NFLVS_, body_t, Op_t >::simulationFlagsChanged ( ) [inline]

Notify a CudaNusquids instance that simulation parameters which enable / disable physics in parameter object changed.

Needs to be called before CudaNusquids::evolve() after calls to ParameterObject::Set_IncludeOscillations, ParameterObject::Set_NonCoherentRhoTerms, ParameterObject::Set_InteractionsRhoTerms ParameterObject::Set_NCInteractions ParameterObject::Set_TauRegeneration ParameterObject::Set_GlashowResonance

The documentation for this class was generated from the following file:

- include/cudanuSQuIDS/cudanusquids.cuh

2.2 cudanusquids::ParameterObject Class Reference

Manages simulation parameters.

`#include <parameterobject.hpp>`

Public Member Functions

- ParameterObject (int numPaths, nusquids::marray< double, 1 > energylist, int n_flvs, nusquids::Neutrino↩ Type neutype, bool interactions, std::shared_ptr< nusquids::NeutrinoCrossSections > ncs=nullptr)

  Constructor.
- void Set_Basis (nusquids::Basis basis)

  Set simulation basis. mass or interaction picture.
- nusquids::Basis getBasis () const

  Get simulation basis.
- nusquids::NeutrinoType getNeutrinoType () const

  Get type of simulated neutrinos.
- void Set_MixingAngle (unsigned int i, unsigned int j, double angle)

  Set theta_(i+1)_(j+1)
- void Set_SquareMassDifference (unsigned int i, double diff)

  Set m_i - m_1.
- void Set_CPPhase (unsigned int i, unsigned int j, double angle)

  Set delta_cp_(i+1)_(j+1)
- void Set_h_min (double opt)

  Set minimum integration step size.
- void Set_h_max (double opt)

  Set maximum integration step size.
- void Set_h (double opt)

  Set begin integration step size for adaptive stepsize integration.
- void Set_rel_error (double opt)

  Set maximum relative integration error.
- void Set_abs_error (double opt)

  Set maximum absolute integration error.
- void Set_NumSteps (unsigned int opt)

  Set number of steps for fixed stepsize integration.
- double Get_h_min () const

  Get minimum integration step size.
- double Get_h_max () const

  Get maximum integration step size.
- double Get_h () const

  begin integration step size for adaptive stepsize integration
- double Get_rel_error () const

  Set maximum relative integration error.
- double Get_abs_error () const

  Set maximum absolute integration error.
- unsigned int Get_NumSteps () const

  Set number of steps for fixed stepsize integration.
- nusquids::marray< double, 1 > GetERange () const

  Get list of neutrino energies.
- const squids::SU_vector & GetFlavorProj (unsigned int flv, unsigned int rho=0) const

  Get flavor projector. If rho == 1, get projector for anti-neutrino.
- const squids::SU_vector & GetMassProj (unsigned int flv, unsigned int rho=0) const

  Get mass projector. If rho == 1, get projector for anti-neutrino.
- const squids::Const & GetParams () const

  Get object with squids constants.
- int GetNumE () const

  Get number of energy bins.

- int GetNumNeu () const

  Get number of flavors.
- int GetNumRho () const

  Get number of neutrino types.
- int getNumPaths () const

  Get number of paths.
- bool Get_CanUseInteractions () const

  Check if interactions can be used.
- void Set_IncludeOscillations (bool opt)

  Enable / Disable neutrino oscillation.
- bool Get_IncludeOscillations () const

  Check if neutrino oscillation is enabled.
- void Set_NonCoherentRhoTerms (bool opt)

  Enable / Disable non-coherent terms.
- bool Get_NonCoherentRhoTerms () const

  Check if non-coherent terms are enabled.
- void Set_InteractionsRhoTerms (bool opt)

  Enable / Disable interaction terms.
- bool Get_InteractionsRhoTerms () const

  Check if interaction terms are enabled.
- void Set_NCInteractions (bool opt)

  Enable / Disable neutral current interactions. Will only be calculated if Get_InteractionsRhoTerms() == true.
- void Set_TauRegeneration (bool opt)

  Enable / Disable Tau regeneration. Will only be calculated if Get_InteractionsRhoTerms() == true.
- void Set_GlashowResonance (bool opt)

  Enable / Disable Glashow resonance. Will only be calculated if Get_InteractionsRhoTerms() == true.
- void Set_ProgressBar (bool opt)

  Enable / Disable progress bar. A Progressbar should only be used with a single GPU.
- bool Get_ProgressBar () const

  Check if progress bar is enabled.
- void Set_DeviceIds (const std::vector< int > &ids)

  Set device ids for GPUs that should be used for simulation.
- const std::vector< int > & Get_DeviceIds () const

  Get device Ids.
- void Set_SolverType (cudanusquids::SolverType solverType_)

  Set solver type. Either Version1 or Version2.
- cudanusquids::SolverType Get_SolverType () const

  Get solver type.
- void Set_StepperType (cudanusquids::ode::StepperType stepperType_)

  Set integration method. RK4 (4th order Runge-Kutta)
- cudanusquids::ode::StepperType Get_StepperType () const

  Get integration method.
- template<class Func >
  void setInitialFlux (Func fluxGenerator, nusquids::Basis fluxbasis_)

  Set initial neutrino flux from generator, specified in basis fluxbasis, either mass or flavor.
- void setInitialFlux (const std::vector< double > &initialFlux, nusquids::Basis fluxbasis)

  Set initial neutrino flux from list, specified in basis fluxbasis, either mass or flavor.
- const std::vector< double > & Get_InitialFlux () const

  Get initial neutrino flux list as it was set by setInitialFlux.
- nusquids::Basis Get_FluxBasis () const

  Get flux basis as it was set by setInitialFlux.

- void registerAdditionalData (size_t size)

  Create a GPU array of size bytes which can be accessed by custom physics operations. The array contents are left uninitialized.
- void registerAdditionalData (size_t size, const char *data)

  Create a GPU array of size bytes which can be accessed by custom physics operations size bytes are copied from data to the GPU array.
- void clearAdditionalData ()

  Delete every GPU array created by registerAdditionalData.

2.2.1 Detailed Description

Manages simulation parameters.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 cudanusquids::ParameterObject::ParameterObject (int *numPaths*, nusquids::marray< double, 1 > *energylist*, int *n_flvs*, nusquids::NeutrinoType *neutype*, bool *interactions*, std::shared_ptr< nusquids::NeutrinoCrossSections > *ncs* = `nullptr`)

Constructor.

Parameters

numPaths	Number of neutrino trajectories
energylist	List of neutrino energies [eV].
n_flvs	Number of neutrino flavors
neutype	neutrino,antineutrino, or both (simultaneous solution).
interactions	If interactions can be used.
ncs	Cross section object. (optional)

2.2.3 Member Function Documentation

2.2.3.1 void cudanusquids::ParameterObject::registerAdditionalData (size_t *size*)

Create a GPU array of size bytes which can be accessed by custom physics operations. The array contents are left uninitialized.

The additional GPU arrays are made available in member void** additionalData; of struct Physics additionalData[0] holds a pointer to the first registered GPU array, additionalData[1] holds a pointer to the second registered GPU array, and so on. The arrays are not exclusive to a specific path, but can be accessed by all neutrino paths. Thus, the arrays must be created large enough to fit the data for the maximum number of parallel simulated paths

2.2.3.2 void cudanusquids::ParameterObject::registerAdditionalData (size_t *size*, const char * *data*)

Create a GPU array of size bytes which can be accessed by custom physics operations size bytes are copied from data to the GPU array.

The additional GPU arrays are made available in member void** additionalData; of struct Physics additionalData[0] holds a pointer to the first registered GPU array, additionalData[1] holds a pointer to the second registered GPU array, and so on. The arrays are not exclusive to a specific path, but can be accessed by all neutrino paths. Thus, the arrays must be created large enough to fit the data for the maximum number of parallel simulated paths

2.2.3.3 void cudanusquids::ParameterObject::setInitialFlux (const std::vector< double > & *initialFlux*, nusquids::Basis *fluxbasis*)

Set initial neutrino flux from list, specified in basis fluxbasis, either mass or flavor.

initialFlux is flat array of dimensions [number of paths][number of neutrino types][number of energies][number of flavors]

The documentation for this class was generated from the following file:

- include/cudanuSQuIDS/parameterobject.hpp

2.3 cudanusquids::Physics< NFLV_, body_t, Op_t > Class Template Reference

Handles the derivation of all bins of a specific neutrino path at a specific time.

Public Attributes

- const double * energyList
  List of energy bins. length n_energies.
- const double * cstates
  pointer to current state. during derivation, cstates == y, else cstates == states
- const double * b0proj
  mass projectors.
- const double * b1proj
  flavor projectors.
- const double * dm2
  Matrix of mass differences.
- const double * H0_array
  Time-independent hamiltonion for each energy bin.
- const double * delE
  Energy difference between the energy bins.
- double * states
  Neutrino states.
- double * evolB1proj
  Flavor projectors in the interaction picture.
- double * fluxes
  The current neutrino fluxes.
- void ** additionalData
  user-defined arrays specified via ParameterObject::registerAdditionalData
- double density
  The current density.
- double electronFraction
  The current electron fraction.
- double t
  The current time.
- int max_n_cosines
  maximum number of simultaneous paths
- int n_rhos

Number of neutrino types. n_rhos = 2 if neutrinoType == both, else n_rhos = 1.

- int n_energies

  Number of energy bins.
- int indexInBatch

  The cosine bin local to the current batch. indexInBatch < max_n_cosines.
- int globalPathId

  The cosine bin.
- BodyType body

  The body.
- BodyType::Track track

  The track.
- InteractionStructureGpu intstruct

  Cross-section lookup table.
- InteractionStateGpu intstate

  Inverse interaction lengths lookup table.
- nusquids::Basis basis

  Basis of states. (mass or interaction)
- nusquids::NeutrinoType neutrinoType

  NeutrinoType.

2.3.1 Detailed Description

template<int NFLV_, class body_t, class Op_t>
class cudanusquids::Physics< NFLV_, body_t, Op_t >

Handles the derivation of all bins of a specific neutrino path at a specific time.

2.3.2 Member Data Documentation

2.3.2.1 template<int NFLV_, class body_t , class Op_t > const double* cudanusquids::Physics< NFLV_, body_t, Op_t >::b0proj

mass projectors.

density matrix, NFLV x NFLV. The i-th entry for mass basis f is b0proj[f + b0offset * i];

2.3.2.2 template<int NFLV_, class body_t , class Op_t > const double* cudanusquids::Physics< NFLV_, body_t, Op_t >::b1proj

flavor projectors.

density matrix, NFLV x NFLV. The i-th entry for flavor basis f with index_rho is b1proj[index_rho * NFLV + flv + i * b1offset]

2.3.2.3 template<int NFLV_, class body_t , class Op_t > const double* cudanusquids::Physics< NFLV_, body_t, Op_t >::delE

Energy difference between the energy bins.

Length n_energies. delE[i+1] = energyList[i+1] - energyList[i], delE[0] = 0.0

2.3.2.4 template<int NFLV_, class body_t , class Op_t > const double* cudanusquids::Physics< NFLV_, body_t, Op_t >::dm2

Matrix of mass differences.

density matrix, NFLV x NFLV

2.3.2.5 template<int NFLV_, class body_t , class Op_t > double* cudanusquids::Physics< NFLV_, body_t, Op_t >::evolB1proj

Flavor projectors in the interaction picture.

density matrix, NFLV x NFLV. Get pointer to first element of specific density matrix via double* evolb1data = getPitchedElement(evolB1proj, index_rho * NFLV * NFLV * NFLV + index_flv * NFLV * NFLV, index_energy, evolB1pitch); then get i-th matrix element with evolb1data[i * evoloffset];

2.3.2.6 template<int NFLV_, class body_t , class Op_t > double* cudanusquids::Physics< NFLV_, body_t, Op_t >::fluxes

The current neutrino fluxes.

Stores the current neutrino fluxes during a derivation step. Get pointer via double* fluxptr = getPitchedElement(fluxes,index_rho * NFLV, flv * fluxOffset + index_energy, fluxPitch) Then get flux with double flux = *fluxptr;

2.3.2.7 template<int NFLV_, class body_t , class Op_t > const double* cudanusquids::Physics< NFLV_, body_t, Op_t >::H0_array

Time-independent hamiltonion for each energy bin.

density matrix, NFLV x NFLV. Get pointer to first element of energy bin index_energy via double* h0data = getPitchedElement(H0_array, 0, index_energy, h0pitch); then get i-th matrix element with h0data[i * h0offset];

2.3.2.8 template<int NFLV_, class body_t , class Op_t > double* cudanusquids::Physics< NFLV_, body_t, Op_t >::states

Neutrino states.

density matrix, NFLV x NFLV. Get pointer to first element of density matrix of energy bin index_energy and index_rho via double* statedata = getPitchedElement(cstates, index_rho * NFLV * NFLV, index_energy, statesPitch); then get i-th matrix element with statedata[i * statesOffset];

The documentation for this class was generated from the following file:

- include/cudanuSQuIDS/physics.cuh

2.4 cudanusquids::PhysicsOps Class Reference

Defines the physical operators used for simulation.

Public Member Functions

- template<class Physics >
  DEVICEQUALIFIER void addToPrederive (Physics &base, double time) const

  Perform custom updates before a derivation step.
- template<class Physics >
  DEVICEQUALIFIER void H0 (const Physics &base, double out[ ], int index_rho, int index_energy) const

  Calculate time-independent part of the Hamiltonian.
- template<class Physics >
  DEVICEQUALIFIER void HI (const Physics &base, double out[ ], int index_rho, int index_energy) const

  Calculate time-dependent part of the Hamiltonian.
- template<class Physics >
  DEVICEQUALIFIER void GammaRho (const Physics &base, double out[ ], int index_rho, int index_energy) const

  Calculate absorbtion and attenuation.
- template<class Physics >
  DEVICEQUALIFIER void InteractionsRho (const Physics &base, double out[ ], int index_rho, int index_↩ energy) const

  Calculate neutrino interactions.

2.4.1 Detailed Description

Defines the physical operators used for simulation.

2.4.2 Member Function Documentation

2.4.2.1 template<class Physics > DEVICEQUALIFIER void cudanusquids::PhysicsOps::GammaRho (const Physics & *base,* double *out[ ],* int *index_rho,* int *index_energy*) const [inline]

Calculate absorbtion and attenuation.

Parameters

base	The physics object to which this function is applied
out	Output array with length Physics::NFLV ∗ Physics::NFLV
index_rho	0, if neutrinotype != Both. Else, 0 = neutrino, 1 = antineutrino
index_energy	Index of energy bin.

2.4.2.2 template<class Physics > DEVICEQUALIFIER void cudanusquids::PhysicsOps::H0 (const Physics & *base,* double *out[ ],* int *index_rho,* int *index_energy*) const [inline]

Calculate time-independent part of the Hamiltonian.

Parameters

base	The physics object to which this function is applied
out	Output array with length Physics::NFLV ∗ Physics::NFLV
index_rho	0, if neutrinotype != Both. Else, 0 = neutrino, 1 = antineutrino
index_energy	Index of energy bin.

2.4.2.3 template<class Physics > DEVICEQUALIFIER void cudanusquids::PhysicsOps::HI (const Physics & *base,* double *out[ ],* int *index_rho,* int *index_energy*) const [inline]

Calculate time-dependent part of the Hamiltonian.

Parameters

base	The physics object to which this function is applied
out	Output array with length Physics::NFLV ∗ Physics::NFLV
index_rho	0, if neutrinotype != Both. Else, 0 = neutrino, 1 = antineutrino
index_energy	Index of energy bin.

2.4.2.4 template<class Physics > DEVICEQUALIFIER void cudanusquids::PhysicsOps::InteractionsRho (const Physics & *base,* double *out[ ],* int *index_rho,* int *index_energy*) const [inline]

Calculate neutrino interactions.

Parameters

base	The physics object to which this function is applied
out	Output array with length Physics::NFLV ∗ Physics::NFLV
index_rho	0, if neutrinotype != Both. Else, 0 = neutrino, 1 = antineutrino
index_energy	Index of energy bin.

The documentation for this class was generated from the following file:

- include/cudanuSQuIDS/physics.cuh

Index