DKS

Generated by Doxygen 1.8.6

Mon Aug 21 2017 10:42:51

Contents

Chapter 1

Main Page

The aim of DKS is to allow the creation of fast fine tuned kernels using device specific frameworks such as CUDA, OpenCL, OpenACC and OpenMP and accelerator libraries such as Thrust, Nvidia CUDA libraries, Intel MKL or others. On top of that, DKS allows the easy use of these kernels in host applications without providing any device or framework specific details. This approach facilitates the integration of different types of devices in the existing applications with minimal code changes and makes the device and the host code a lot more manageable.

The main parts of DKS are:

- DKSBase - provides the basic communication functions between host application and hardware accelerators including memory manegement, data transfer and synchronization.
- DKSOPAL - provides functions for Object Oriented Particle Accelerator library to offload FFTPoisson calculations and particle matter interaction using Monte Carlo simulations to GPU and Intel MIC
- DKSBaseMuSR - provides functions to perform parameter fitting for musrfit on the GPU
- DKSImageRecon - provides functions to perform PET image reconstruction on the GPU
- DKSFFT - provides functions to perform FFT on the GPU and Intel MIC

Developed by Uldis Locans

For further information contact: locans.uldis@psi.ch - Uldis Locans

DKS on gitlab

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

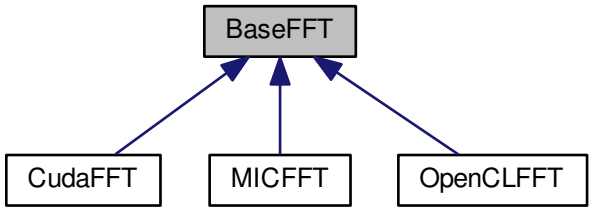
Class Documentation

4.1 BaseFFT Class Reference

Abstract class defining methods for DKS FFT class.

`#include <FFT.h>`

Inheritance diagram for BaseFFT:



Public Member Functions

- virtual int setupFFT (int ndim, int N[3])=0

  Setup FFT - init FFT library used by chosen device.
- virtual int setupFFTRC (int ndim, int N[3], double scale=1.0)=0

  Setup real to complex FFT - init FFT library used by chosen device.
- virtual int setupFFTCR (int ndim, int N[3], double scale=1.0)=0

  Setup real to complex complex to real FFT - init FFT library used by chosen device.
- virtual int destroyFFT ()=0

  Clean up.
- virtual int executeFFT (void ∗mem_ptr, int ndim, int N[3], int streamId=-1, bool forward=true)=0

  Exectute C2C FFT.
- virtual int executeIFFT (void ∗mem_ptr, int ndim, int N[3], int streamId=-1)=0

  Exectute inverse C2C FFT.
- virtual int normalizeFFT (void ∗mem_ptr, int ndim, int N[3], int streamId=-1)=0

  Normalize the FFT or IFFT.

- virtual int executeRCFFT (void *real_ptr, void *comp_ptr, int ndim, int N[3], int streamId=-1)=0

  Exectute R2C FFT.
- virtual int executeCRFFT (void *real_ptr, void *comp_ptr, int ndim, int N[3], int streamId=-1)=0

  Exectute C2R FFT.
- virtual int normalizeCRFFT (void *real_ptr, int ndim, int N[3], int streamId=-1)=0

  Normalize CR FFT.

Protected Member Functions

- bool useDefaultPlan (int ndim, int N[3])

  Check if FFT plan is created for the needed dimension and FFT size.

Protected Attributes

- int **defaultN** [3]
- int **defaultNdim**

4.1.1 Detailed Description

Abstract class defining methods for DKS FFT class.

Used by CudaFFT, OpenCLFFT and MICFFT to create device specific FFT classes.

4.1.2 Member Function Documentation

4.1.2.1 virtual int BaseFFT::destroyFFT ( ) [pure virtual]

Clean up.

Implemented in OpenCLFFT, MICFFT, and CudaFFT.

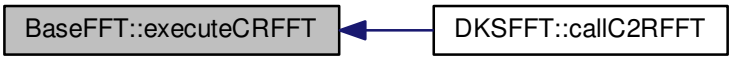
4.1.2.2 virtual int BaseFFT::executeCRFFT (void * *real_ptr*, void * *comp_ptr*, int *ndim*, int *N[3]*, int *streamId* = -1) [pure virtual]

Exectute C2R FFT.

real_ptr - real output data from the C2R FFT, comp_ptr - complex input data for the FFT.

Implemented in OpenCLFFT, MICFFT, and CudaFFT.

Here is the caller graph for this function:



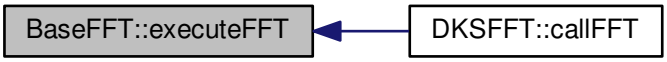
4.1.2.3 virtual int BaseFFT::executeFFT (void * *mem_ptr*, int *ndim*, int *N[3]*, int *streamId* = -1, bool *forward* = true) [pure virtual]

Exectute C2C FFT.

mem_ptr - memory ptr on the device for complex data. Performs in place FFT.

Implemented in OpenCLFFT, MICFFT, and CudaFFT.

Here is the caller graph for this function:



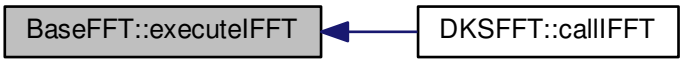
4.1.2.4 virtual int BaseFFT::executeIFFT (void * *mem_ptr*, int *ndim*, int *N[3]*, int *streamId* = -1) [pure virtual]

Exectute inverse C2C FFT.

mem_ptr - memory ptr on the device for complex data. Performs in place FFT.

Implemented in OpenCLFFT, MICFFT, and CudaFFT.

Here is the caller graph for this function:



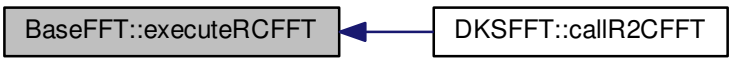
4.1.2.5 virtual int BaseFFT::executeRCFFT (void * *real_ptr*, void * *comp_ptr*, int *ndim*, int *N[3]*, int *streamId* = -1) [pure virtual]

Exectute R2C FFT.

real_ptr - real input data for FFT, comp_ptr - memory on the device where results for the FFT are stored as complex numbers.

Implemented in OpenCLFFT, MICFFT, and CudaFFT.

Here is the caller graph for this function:



4.1.2.6 virtual int BaseFFT::normalizeFFT (void * *mem_ptr*, int *ndim*, int *N[3]*, int *streamId* = -1) [pure virtual]

Normalize the FFT or IFFT.

mem_ptr - memory to complex data.

Implemented in OpenCLFFT, MICFFT, and CudaFFT.

Here is the caller graph for this function:

4.1.2.7 virtual int BaseFFT::setupFFT (int *ndim*, int *N[3]*) [pure virtual]

Setup FFT - init FFT library used by chosen device.

Implemented in OpenCLFFT, MICFFT, and CudaFFT.

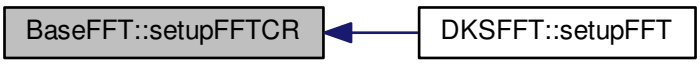
Here is the caller graph for this function:

4.1.2.8 virtual int BaseFFT::setupFFTCR (int *ndim*, int *N[3]*, double *scale* = 1.0) [pure virtual]

Setup real to complex complex to real FFT - init FFT library used by chosen device.

Implemented in OpenCLFFT, MICFFT, and CudaFFT.

Here is the caller graph for this function:

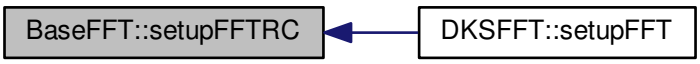


4.1.2.9 virtual int BaseFFT::setupFFTRC (int *ndim*, int *N[3]*, double *scale* = 1.0) [pure virtual]

Setup real to complex FFT - init FFT library used by chosen device.

Implemented in OpenCLFFT, MICFFT, and CudaFFT.

Here is the caller graph for this function:

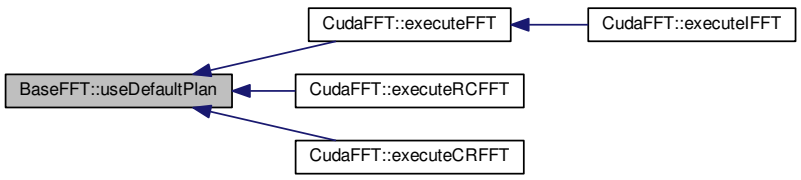


4.1.2.10 bool BaseFFT::useDefaultPlan (int *ndim*, int *N[3]*) [inline], [protected]

Check if FFT plan is created for the needed dimension and FFT size.

Returns true if the plan has been created and false if no plan for specified dimension and size exists.

Here is the caller graph for this function:



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

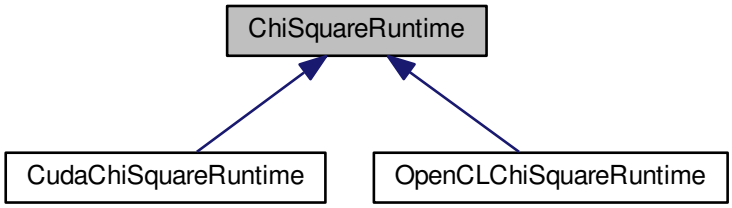
- src/Algorithms/FFT.h

4.2 ChiSquareRuntime Class Reference

Interface to implement ChiSquareRuntime class for musrfit.

`#include <ChiSquareRuntime.h>`

Inheritance diagram for ChiSquareRuntime:



Public Member Functions

- virtual ~ChiSquareRuntime ()

  Default constructor.
- virtual int compileProgram (std::string function, bool mlh=false)=0

  Compile GPU programm generated at runtime.
- virtual int launchChiSquare (int fitType, void *mem_data, void *mem_err, int length, int numpar, int numfunc, int nummap, double timeStart, double timeStep, double &result)=0

  Launche the compiled chiSquare kernel.
- virtual int writeParams (const double *params, int numparams)=0

  Write the parameter values to the GPU.
- virtual int writeFunc (const double *func, int numfunc)=0

  Write the function values to the GPU.
- virtual int writeMap (const int *map, int nummap)=0

  Write map values to the GPU.
- virtual int initChiSquare (int size_data, int size_param, int size_func, int size_map)=0

  Allocate temporary memory needed for the chi square calucaltios on the device.
- virtual int freeChiSquare ()=0

  Free device memory allocated for chi square calculations.
- virtual int checkChiSquareKernels (int fitType, int &threadsPerBlock)=0

  Check if available device can run the chi square GPU code.
- int setConsts (double N0, double tau, double bkg)

  Set N0, tau and bgk values to use for the kernel.
- int setConsts (double alpha, double beta)

  Set alpha and beta values to use for the kernel.
- int setKernelParams (int numBlocks, int blockSize)

  Set number of blocks and threads.
- int getOperations (int &oper)

  Get the number of operations in compiled kernel.

Protected Member Functions

- void **setN0** (double value)
- void **setTau** (double value)
- void **setBKG** (double value)
- void **setAlpha** (double value)
- void **setBeta** (double value)

Protected Attributes

- double **N0_m**
- double **tau_m**
- double **bkg_m**
- double **alpha_m**
- double **beta_m**
- bool **initDone_m**
- void * **mem_chisq_m**
- void * **mem_param_m**
- void * **mem_func_m**
- void * **mem_map_m**
- int **numBlocks_m**
- int **blockSize_m**
- char * **ptx_m**

Friends

- class **DKSBaseMuSR**

4.2.1 Detailed Description

Interface to implement ChiSquareRuntime class for musrfit.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 virtual ChiSquareRuntime::~ChiSquareRuntime ( ) [inline], [virtual]

Default constructor.

Default destructor.

4.2.3 Member Function Documentation

4.2.3.1 int ChiSquareRuntime::getOperations (int & *oper*) [inline]

Get the number of operations in compiled kernel.

Count the number of operation in the ptx file for the compiled program.

Here is the caller graph for this function:

4.2.3.2 int ChiSquareRuntime::setConsts (double *N0,* double *tau,* double *bkg*) [inline]

Set N0, tau and bgk values to use for the kernel.

If values changes between data sets this needs to be called before every kernel call. Returns DKS_SUCCESS.

Here is the caller graph for this function:

4.2.3.3 int ChiSquareRuntime::setConsts (double *alpha,* double *beta*) [inline]

Set alpha and beta values to use for the kernel.

If values changes between data sets this needs to be called before every kernel call. Returns DKS_SUCCESS.

4.2.3.4 int ChiSquareRuntime::setKernelParams (int *numBlocks,* int *blockSize*) [inline]

Set number of blocks and threads.

Used to set parameters obtained from auto-tuning

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

- src/Algorithms/ChiSquareRuntime.h

4.3 compare_particle_small Struct Reference

Operator used in thrust sort to compare particles by label.

Public Member Functions

- void **set_threshold** (int t)
- __host__ __device__ bool **operator()** (CUDA_PART_SMALL p1, CUDA_PART_SMALL p2)
- __host__ __device__ bool **operator()** (CUDA_PART_SMALL p1)

Public Attributes

- int **threshold**

4.3.1 Detailed Description

Operator used in thrust sort to compare particles by label.

Used to move dead particles to the end of array, since they have label -1 or -2.

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

- src/CUDA/CudaCollimatorPhysics.cuh

4.4 CUDA_PART2 Struct Reference

Structure for storing particle on GPU as SoA.

Public Attributes

- int * **label**
- unsigned * **localID**
- double3 * **Rincol**
- double3 * **Pincol**
- long * **lDincol**
- int * **Bininco**l
- double * **DTincol**
- double * **Qincol**
- long * **LastSecincol**
- double3 * **Bfincol**
- double3 * **Efincol**

4.4.1 Detailed Description

Structure for storing particle on GPU as SoA.

Structure for OPAL particle, can be used to store particles on the GPU in structure of arrays.

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

- src/CUDA/CudaCollimatorPhysics.cuh

4.5 CUDA_PART2_SMALL Struct Reference

Structure for storing particle on GPU Structure for OPAL particle, can be used to store particles on the GPU in structure of arrays, contains only data that are used by the GPU kernels, the rest of the particle data must be kept on the host side.

Public Attributes

- int * **label**
- unsigned * **localID**
- double3 * **Rincol**
- double3 * **Pincol**

4.5.1 Detailed Description

Structure for storing particle on GPU Structure for OPAL particle, can be used to store particles on the GPU in structure of arrays, contains only data that are used by the GPU kernels, the rest of the particle data must be kept on the host side.

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

- src/CUDA/CudaCollimatorPhysics.cuh

4.6 CUDA_PART_SMALL Struct Reference

Structure for storing particle on GPU as AoS Structure for OPAL particle, can be used to store particles on the GPU in array of structures, contains only data that are used by the GPU kernels, the rest of the particle data must be kept on the host side.

Public Attributes

- int **label**
- unsigned **localID**
- double3 **Rincol**
- double3 **Pincol**

4.6.1 Detailed Description

Structure for storing particle on GPU as AoS Structure for OPAL particle, can be used to store particles on the GPU in array of structures, contains only data that are used by the GPU kernels, the rest of the particle data must be kept on the host side.

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

- src/CUDA/CudaCollimatorPhysics.cuh

4.7 CudaBase Class Reference

CUDA base class handles device setup and basic communication with the device.

Public Member Functions

- int getDefaultThreads ()

  Get default threads per block.
- int cuda_createCurandStates (int size, int seed=-1)

  Init cuda random number (cuRand) states.
- int cuda_deleteCurandStates ()

  Delete curandState.
- int cuda_createRandomNumbers (void *mem_ptr, int size)

  Create 'size' random numbers on the device and save in mem_ptr array.
- curandState * cuda_getCurandStates ()

  Get a pointer to curand states.
- int cuda_createStream (int &streamId)

  Create a cuda stream and set streamId to index refering to this stream.

- int cuda_addStream (cudaStream_t tmpStream, int &streamId)
  add existing cuda stream to the list.
- int cuda_deleteStream (int id)
  delete cuda stream.
- int cuda_deleteStreams ()
  delete all streams.
- int cuda_setStream (int id)
  set stream to use.
- int cuda_getStreamId ()
  get stream that is used.
- int cuda_defaultStream ()
  reset to default stream.
- int cuda_numberOfStreams ()
  get number of streams.
- cudaStream_t cuda_getStream (int id)
  get stream.
- cublasHandle_t cuda_getCublas ()
  Get default cublass handle.
- int cuda_getDevices ()
  get information on cuda devices.
- int cuda_getDeviceCount (int &ndev)
  Get CUDA device count.
- int cuda_getDeviceName (std::string &device_name)
  Get the name of the device.
- int cuda_setDevice (int device)
  Set CUDA device to use.
- int cuda_getUniqueDevices (std::vector< int > &devices)
  Get unique devices.
- int cuda_setUp ()
  Initialize connection to the device.
- void * cuda_allocateMemory (size_t size, int &ierr)
  Allocate memory on cuda device.
- template<typename T >
  int cuda_allocateHostMemory (T *&ptr, size_t size)
  Allocate host memory in pinned memory Return: success or error code.
- template<typename T >
  int cuda_zeroMemory (T *mem_ptr, size_t size, int offset=0)
  Zero CUDA memory.
- template<typename T >
  int cuda_zeroMemoryAsync (T *mem_ptr, size_t size, int offset=0, int streamId=-1)
  Zero CUDA memory async.
- template<typename T >
  int cuda_writeData (T *mem_ptr, const void *in_data, size_t size, int offset=0)
  Write data to memory Retrun: success or error code.
- template<typename T >
  int cuda_writeDataAsync (T *mem_ptr, const void *in_data, size_t size, int streamId=-1, int offset=0)
  Write data assynchonuously Return: success or error code.
- template<typename T >
  int cuda_readData (const T *mem_ptr, void *out_data, size_t size, int offset=0)
  Read data from memory Return: success or error code.
- template<typename T >
  int cuda_readDataAsync (const T *mem_ptr, void *out_data, size_t size, int streamId=-1, int offset=0)

Read data async from device memory Return: success or error code.

- int cuda_freeMemory (void *mem_ptr)

  Free memory on device Return: success or error code.

- int cuda_freeHostMemory (void *mem_ptr)

  Free page locked memory on host Return: success or erro code.

- template<typename T >
  void * cuda_pushData (const void *in_data, size_t size, int &ierr)

  Allcate memory and write data (push) Return: pointer to memory object.

- template<typename T >
  int cuda_pullData (T *mem_ptr, void *out_data, size_t size, int &ierr)

  Read data and free memory (pull) Return: success or error code.

- int cuda_syncDevice ()

  Sync cuda device.

- template<typename T >
  int cuda_hostRegister (T *ptr, int size)

  Page-lock host memory.

- template<typename T >
  int cuda_hostUnregister (T *ptr)

  Release page locked memory.

- int cuda_memInfo ()

  Print device memory info (total, used, avail) Return: success or error code.

Protected Attributes

- cublasHandle_t **defaultCublas**
- curandState * **defaultRndState**
- int **defaultRndSet**

4.7.1 Detailed Description

CUDA base class handles device setup and basic communication with the device.

Handles devicew setup, memory manegement, data transfers and stream setup for asynchronous data transfers and kernel executions.

4.7.2 Member Function Documentation

4.7.2.1 int CudaBase::cuda_addStream (cudaStream_t *tmpStream*, int & *streamId*)

add existing cuda stream to the list.

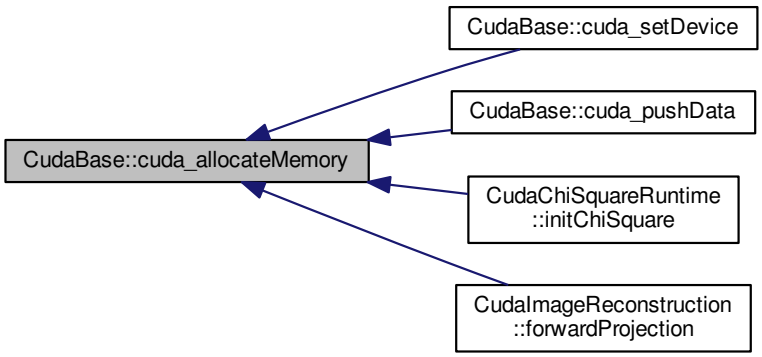
Return: success or error code.

4.7.2.2 void* CudaBase::cuda_allocateMemory (size_t *size*, int & *ierr*)

Allocate memory on cuda device.

Return: pointer to memory object

Here is the caller graph for this function:

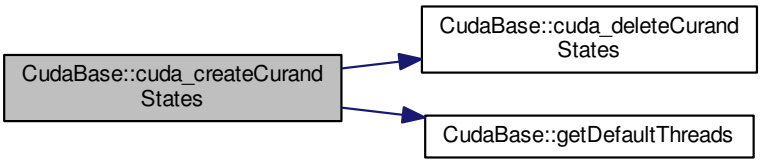


4.7.2.3 int CudaBase::cuda_createCurandStates (int *size*, int *seed* = -1)

Init cuda random number (cuRand) states.

Create an array of type curandState with "size" elements on the GPU and create a curandState with different seed for each array entry. If no seed is given create a seed based on current time. Return success or error code

Here is the call graph for this function:



4.7.2.4 int CudaBase::cuda_createStream (int & *streamId*)

Create a cuda stream and set streamId to index refering to this stream.

Return success or error code

4.7.2.5 int CudaBase::cuda_defaultStream ( )

reset to default stream.

Return: success or error code

4.7.2.6 int CudaBase::cuda_deleteCurandStates ( )

Delete curandState.

Delete curandState array on the GPU and free memory. Return success or error code

Here is the caller graph for this function:

4.7.2.7 int CudaBase::cuda_deleteStream (int *id*)

delete cuda stream.

success or error code

4.7.2.8 int CudaBase::cuda_deleteStreams ( )

delete all streams.

success or error code

4.7.2.9 int CudaBase::cuda_getDeviceCount (int & *ndev*)

Get CUDA device count.

Sets the number of devices on the platform that can use CUDA.

4.7.2.10 int CudaBase::cuda_getDeviceName (std::string & *device_name*)

Get the name of the device.

QUery the device properties of the used device and set the string device_name

4.7.2.11 int CudaBase::cuda_getDevices ( )

get information on cuda devices.

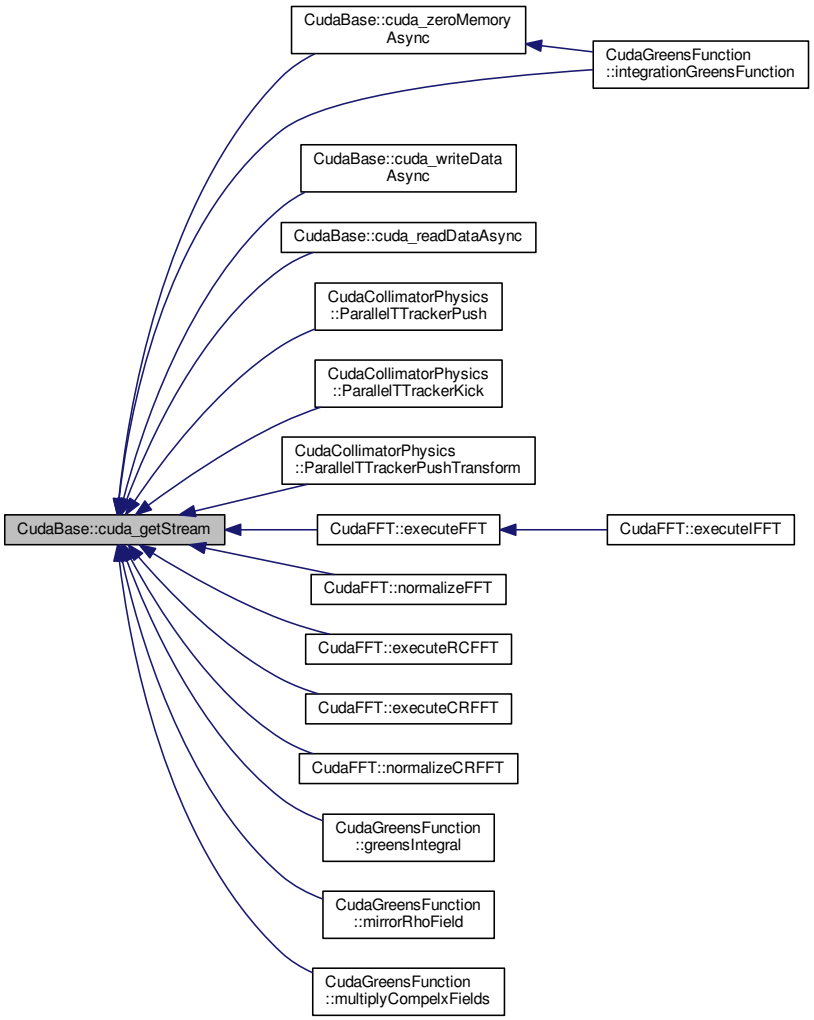
Return: success or error code

4.7.2.12 cudaStream_t CudaBase::cuda_getStream (int *id*)

get stream.

Return: stream

Here is the caller graph for this function:



4.7.2.13 int CudaBase::cuda_getStreamId ( )

get stream that is used.

Return: return id of curretn stream

4.7.2.14 int CudaBase::cuda_getUniqueDevices (std::vector< int > & *devices*)

Get unique devices.

Get array of indeces with the unique CUDA devices available on the paltform

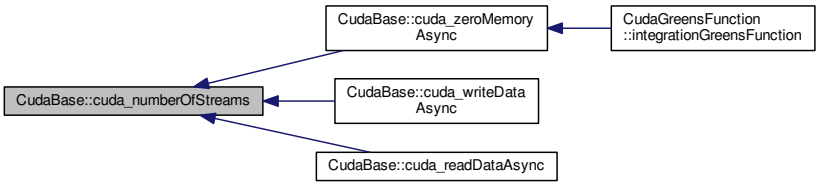
Here is the caller graph for this function:

4.7.2.15 int CudaBase::cuda_numberOfStreams ( )

get number of streams.

Return: success or error code

Here is the caller graph for this function:

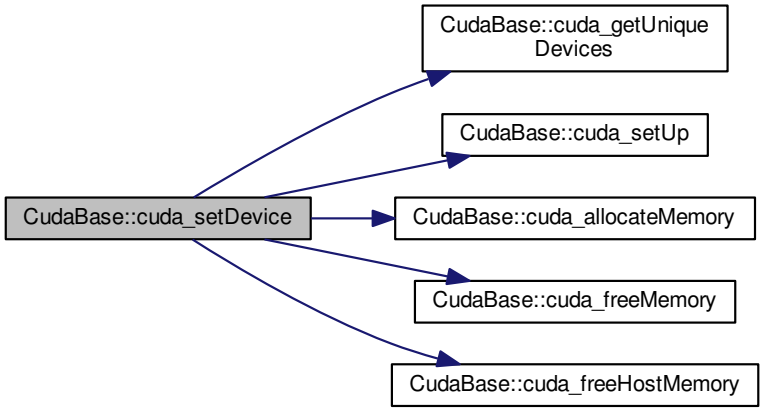


4.7.2.16 int CudaBase::cuda_setDevice (int *device*)

Set CUDA device to use.

If device passed in is larger than the number of devices use the default:0 and return DKS_ERROR

Here is the call graph for this function:



4.7.2.17 int CudaBase::cuda_setStream (int *id*)

set stream to use.

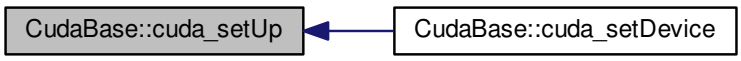
success or error code

4.7.2.18 int CudaBase::cuda_setUp ( )

Initialize connection to the device.

Only needed when runtime compilation is used. Return: success or error code

Here is the caller graph for this function:



4.7.2.19 int CudaBase::cuda_syncDevice ( ) [inline]

Sync cuda device.

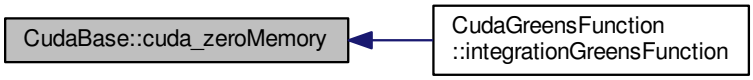
Waits till all the tasks on the GPU are finished. Return: success or error code

4.7.2.20 template<typename T > int CudaBase::cuda_zeroMemory (T * *mem_ptr*, size_t *size*, int *offset* = 0) [inline]

Zero CUDA memory.

Set all the elements of the array on the device to zero.

Here is the caller graph for this function:

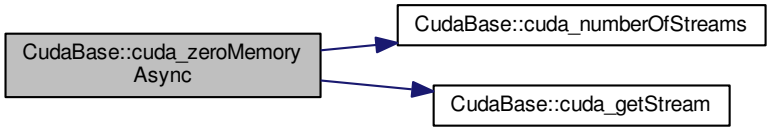


4.7.2.21 template<typename T > int CudaBase::cuda_zeroMemoryAsync (T * *mem_ptr*, size_t *size*, int *offset* = 0, int *streamId* = -1) [inline]

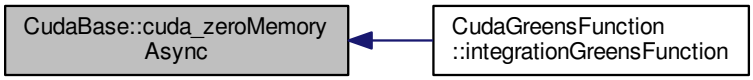
Zero CUDA memory async.

Set all the elements of the array on the device to zero.

Here is the call graph for this function:



Here is the caller graph for this function:



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

- src/CUDA/CudaBase.cuh
- src/CUDA/CudaBase.cu

4.8 CudaChiSquare Class Reference

Deprecated, CUDA simpleFit implementation of ChiSquare.

Public Member Functions

- CudaChiSquare (CudaBase ∗base)

  Constructor which gets CudaBase as argument.
- int **cuda_PHistoTFFcn** (void ∗mem_data, void ∗mem_par, void ∗mem_chisq, double fTimeResolution, double fRebin, int sensors, int length, int numpar, double &result)
- int **cuda_singleGaussTF** (void ∗mem_data, void ∗mem_t0, void ∗mem_par, void ∗mem_result, double fTimeResolution, double fRebin, double fGoodBinOffset, int sensors, int length, int numpar, double &result)
- int **cuda_doubleLorentzTF** (void ∗mem_data, void ∗mem_t0, void ∗mem_par, void ∗mem_result, double fTimeResolution, double fRebin, double fGoodBinOffset, int sensors, int length, int numpar, double &result)

4.8.1 Detailed Description

Deprecated, CUDA simpleFit implementation of ChiSquare.

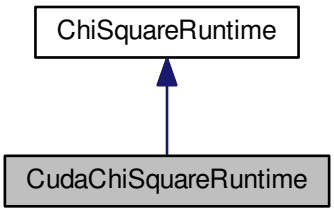
The documentation for this class was generated from the following files:

- src/CUDA/CudaChiSquare.cuh
- src/CUDA/CudaChiSquare.cu

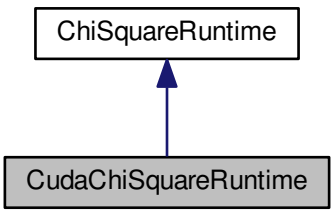
4.9 CudaChiSquareRuntime Class Reference

CUDA implementation of ChiSquareRuntime class.

Inheritance diagram for CudaChiSquareRuntime:



Collaboration diagram for CudaChiSquareRuntime:



Public Member Functions

- CudaChiSquareRuntime (CudaBase *base)

  Constructor with CudaBase argument.
- CudaChiSquareRuntime ()

  Default constructor init cuda device.
- ~CudaChiSquareRuntime ()

  Default destructor.
- int compileProgram (std::string function, bool mlh=false)

  Compile program and save ptx.
- int launchChiSquare (int fitType, void *mem_data, void *mem_err, int length, int numpar, int numfunc, int nummap, double timeStart, double timeStep, double &result)

  Launch selected kernel.
- int writeParams (const double *params, int numparams)

  Write params to device.
- int writeFunc (const double *func, int numfunc)

  Write functions to device.
- int writeMap (const int *map, int nummap)

  Write maps to device.
- int initChiSquare (int size_data, int size_param, int size_func, int size_map)

  Allocate temporary memory needed for chi square.
- int freeChiSquare ()

  Free temporary memory allocated for chi square.
- int checkChiSquareKernels (int fitType, int &threadsPerBlock)

  Check if CUDA device is able to run the chi square kernel.

Additional Inherited Members

4.9.1 Detailed Description

CUDA implementation of ChiSquareRuntime class.

Implements ChiSquareRuntime interface to allow musrfit to use CUDA to target Nvidia GPU.

4.9.2 Member Function Documentation

4.9.2.1 int CudaChiSquareRuntime::checkChiSquareKernels (int *fitType,* int & *threadsPerBlock*) [inline], [virtual]

Check if CUDA device is able to run the chi square kernel.

Redundant - all new CUDA devices that support RT compilation will also support double precision, there are no other requirements to run chi square on GPU

Implements ChiSquareRuntime.

4.9.2.2 int CudaChiSquareRuntime::compileProgram (std::string *function,* bool *mlh* = false) [virtual]

Compile program and save ptx.

Add function string to the calcFunction kernel and compile the program Function must be valid C math expression. Parameters can be addressed in a form par[map[idx]]

Implements ChiSquareRuntime.

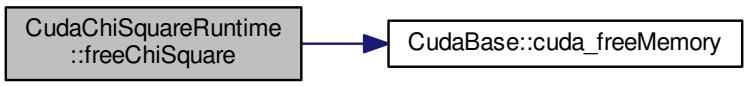
4.9.2.3 int CudaChiSquareRuntime::freeChiSquare ( ) [virtual]

Free temporary memory allocated for chi square.

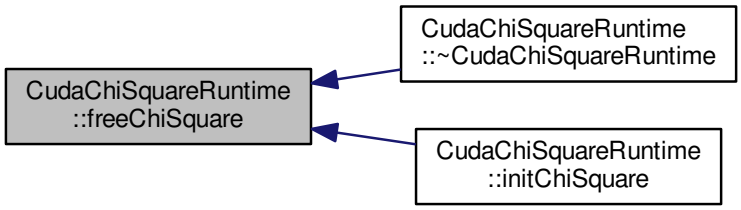
Frees the chisq temporary memory and memory for params, functions and maps

Implements ChiSquareRuntime.

Here is the call graph for this function:



Here is the caller graph for this function:



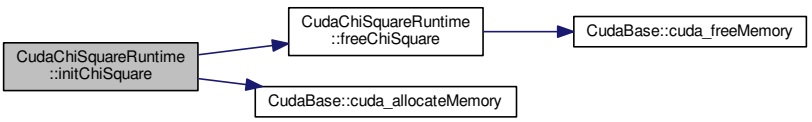
4.9.2.4 int CudaChiSquareRuntime::initChiSquare (int *size_data,* int *size_param,* int *size_func,* int *size_map*) [virtual]

Allocate temporary memory needed for chi square.

Initializes the necessary temporary memory for the chi square calculations. Size_data needs to the maximum number of elements in any datasets that will be used for calculations. Size_param, size_func and size_map are the maximum number of parameters, functions and maps used in calculations.

Implements ChiSquareRuntime.

Here is the call graph for this function:



4.9.2.5 int CudaChiSquareRuntime::launchChiSquare (int *fitType*, void * *mem_data*, void * *mem_err*, int *length*, int *numpar*, int *numfunc*, int *nummap*, double *timeStart*, double *timeStep*, double & *result*) [virtual]

Launch selected kernel.

Launched the selected kernel from the compiled code. Result is put in &result variable.

Implements ChiSquareRuntime.

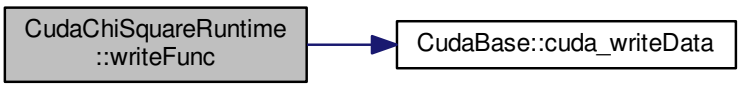
4.9.2.6 int CudaChiSquareRuntime::writeFunc (const double * *func*, int *numfunc*) [virtual]

Write functions to device.

Write function values from double array to mem_func_m memory on the device.

Implements ChiSquareRuntime.

Here is the call graph for this function:



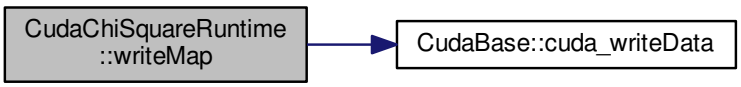
4.9.2.7 int CudaChiSquareRuntime::writeMap (const int * *map*, int *nummap*) [virtual]

Write maps to device.

Write map values from int array to mem_map_m memory on the device.

Implements ChiSquareRuntime.

Here is the call graph for this function:



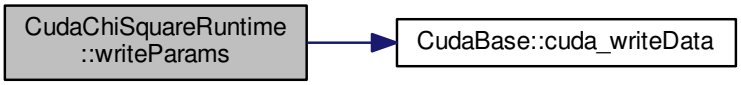
4.9.2.8 int CudaChiSquareRuntime::writeParams (const double * *params*, int *numparams*) [virtual]

Write params to device.

Write params from double array to mem_param_m memory on the device.

Implements ChiSquareRuntime.

Here is the call graph for this function:



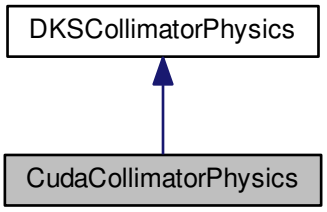
The documentation for this class was generated from the following files:

- src/CUDA/CudaChiSquareRuntime.cuh
- src/CUDA/CudaChiSquareRuntime.cu

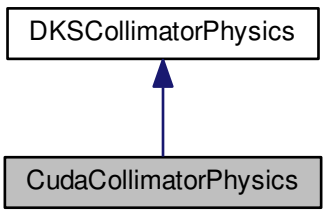
4.10 CudaCollimatorPhysics Class Reference

CudaCollimatorPhysics class based on DKSCollimatorPhysics interface.

Inheritance diagram for CudaCollimatorPhysics:



Collaboration diagram for CudaCollimatorPhysics:



Public Member Functions

- CudaCollimatorPhysics (CudaBase *base)

  Constructor with CudaBase as argument.
- CudaCollimatorPhysics ()

  Empty constructor.
- ~CudaCollimatorPhysics ()

  Destructor.
- int CollimatorPhysics (void *mem_ptr, void *par_ptr, int numpartices, bool enableRutherforScattering=true)

  Execute collimator physics kernel.
- int CollimatorPhysicsSoA (void *label_ptr, void *localID_ptr, void *rx_ptr, void *ry_ptr, void *rz_ptr, void *px_ptr, void *py_ptr, void *pz_ptr, void *par_ptr, int numparticles)

  Special calse CollimatorPhysics kernel that uses SoA instead of AoS.
- int CollimatorPhysicsSort (void *mem_ptr, int numparticles, int &numaddback)

  Sort particle array on GPU.
- int CollimatorPhysicsSortSoA (void *label_ptr, void *localID_ptr, void *rx_ptr, void *ry_ptr, void *rz_ptr, void *px_ptr, void *py_ptr, void *pz_ptr, void *par_ptr, int numparticles, int &numaddback)

  Special calse CollimatorPhysicsSort kernel that uses SoA instead of AoS.
- int ParallelTTrackerPush (void *r_ptr, void *p_ptr, int npart, void *dt_ptr, double dt, double c, bool usedt=false, int streamId=-1)

  BorisPusher push function for integration from OPAL.
- int ParallelTTrackerKick (void *r_ptr, void *p_ptr, void *ef_ptr, void *bf_ptr, void *dt_ptr, double charge, double mass, int npart, double c, int streamId=-1)

  BorisPusher kick function for integration from OPAL.
- int ParallelTTrackerPushTransform (void *x_ptr, void *p_ptr, void *lastSec_ptr, void *orient_ptr, int npart, int nsec, void *dt_ptr, double dt, double c, bool usedt=false, int streamId=-1)

  BorisPusher push function with transformto function form OPAL.

Additional Inherited Members

4.10.1 Detailed Description

CudaCollimatorPhysics class based on DKSCollimatorPhysics interface.

Contains kerenls that execute CollimatorPhysics functions form OPAL. For detailed documentation on CollimatorPhysics functions see OPAL documentation.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 CudaCollimatorPhysics::CudaCollimatorPhysics (CudaBase * *base*) [inline]

Constructor with CudaBase as argument.

Create a new instace of the CudaCollimatorPhysics using existing CudaBase object.

4.10.2.2 CudaCollimatorPhysics::CudaCollimatorPhysics ( ) [inline]

Empty constructor.

Create a new instance of CudaCollimatorPhysics with its own CudaBase.

4.10.2.3 CudaCollimatorPhysics::~CudaCollimatorPhysics ( ) [inline]

Destructor.

Destroy CudaBase object if it was created by CudaCollimatorPhysics constructor.

4.10.3 Member Function Documentation

4.10.3.1 int CudaCollimatorPhysics::CollimatorPhysicsSoA (void * *label_ptr,* void * *localID_ptr,* void * *rx_ptr,* void * *ry_ptr,* void * *rz_ptr,* void * *px_ptr,* void * *py_ptr,* void * *pz_ptr,* void * *par_ptr,* int *numparticles*) [inline], [virtual]

Special calse CollimatorPhysics kernel that uses SoA instead of AoS.

Used only on the MIC side, was not implemented on the GPU.

Implements DKSCollimatorPhysics.

4.10.3.2 int CudaCollimatorPhysics::CollimatorPhysicsSort (void * *mem_ptr,* int *numparticles,* int & *numaddback*) [virtual]

Sort particle array on GPU.

Count particles that are dead (label -1) or leaving material (label -2) and sort particle array so these particles are at the end of array

Implements DKSCollimatorPhysics.

4.10.3.3 int CudaCollimatorPhysics::CollimatorPhysicsSortSoA (void * *label_ptr,* void * *localID_ptr,* void * *rx_ptr,* void * *ry_ptr,* void * *rz_ptr,* void * *px_ptr,* void * *py_ptr,* void * *pz_ptr,* void * *par_ptr,* int *numparticles,* int & *numaddback*) [inline], [virtual]

Special calse CollimatorPhysicsSort kernel that uses SoA instead of AoS.

Used only on the MIC side, was not implemented on the GPU.

Implements DKSCollimatorPhysics.

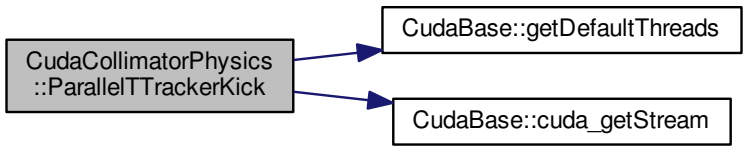
4.10.3.4 int CudaCollimatorPhysics::ParallelTTrackerKick (void * *r_ptr,* void * *p_ptr,* void * *ef_ptr,* void * *bf_ptr,* void * *dt_ptr,* double *charge,* double *mass,* int *npart,* double *c,* int *streamId* = −1) [virtual]

BorisPusher kick function for integration from OPAL.

ParallelTTracker integration from OPAL implemented in cuda. For more details see ParallelTTracler docomentation in opal

Implements DKSCollimatorPhysics.

Here is the call graph for this function:



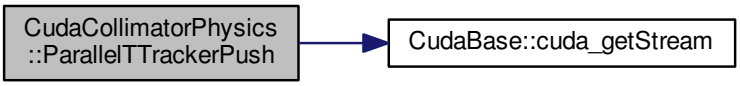
4.10.3.5 int CudaCollimatorPhysics::ParallelTTrackerPush (void * *r_ptr*, void * *p_ptr*, int *npart*, void * *dt_ptr*, double *dt*, double *c*, bool *usedt* = `false`, int *streamId* = `-1`) [virtual]

BorisPusher push function for integration from OPAL.

ParallelTTracker integration from OPAL implemented in cuda. For more details see ParallelTTracler docomentation in opal

Implements DKSCollimatorPhysics.

Here is the call graph for this function:



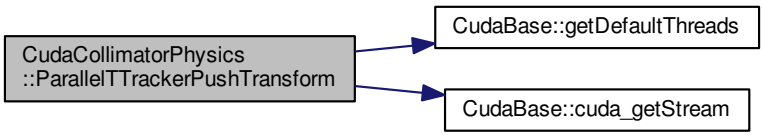
4.10.3.6 int CudaCollimatorPhysics::ParallelTTrackerPushTransform (void * *x_ptr*, void * *p_ptr*, void * *lastSec_ptr*, void * *orient_ptr*, int *npart*, int *nsec*, void * *dt_ptr*, double *dt*, double *c*, bool *usedt* = `false`, int *streamId* = `-1`) [virtual]

BorisPusher push function with transformto function form OPAL.

ParallelTTracker integration from OPAL implemented in cuda. For more details see ParallelTTracler docomentation in opal

Implements DKSCollimatorPhysics.

Here is the call graph for this function:



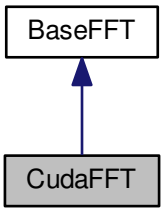
The documentation for this class was generated from the following files:

- src/CUDA/CudaCollimatorPhysics.cuh
- src/CUDA/CudaCollimatorPhysics.cu

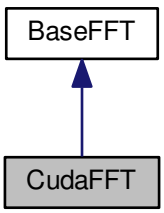
4.11 CudaFFT Class Reference

Cuda FFT class based on BaseFFT interface.

Inheritance diagram for CudaFFT:



Collaboration diagram for CudaFFT:



Public Member Functions

- CudaFFT (CudaBase *base)

  Constructor with CudaBase as argument.
- CudaFFT ()

  constructor
- ~CudaFFT ()

  destructor
- int setupFFT (int ndim, int N[3])

  Init cufftPlans witch can be reused for all FFTs of the same size and type Return: success or error code.
- int setupFFTRC (int ndim, int N[3], double scale=1.0)

  Setup real to complex FFT - init FFT library used by chosen device.
- int setupFFTCR (int ndim, int N[3], double scale=1.0)

  Setup real to complex complex to real FFT - init FFT library used by chosen device.
- int destroyFFT ()

  Destroy default FFT plans Return: success or error code.
- int executeFFT (void *mem_ptr, int ndim, int N[3], int streamId=-1, bool forward=true)

  Exectute C2C FFT.
- int executeIFFT (void *mem_ptr, int ndim, int N[3], int streamId=-1)

  Exectute inverse C2C FFT.
- int normalizeFFT (void *mem_ptr, int ndim, int N[3], int streamId=-1)

  Normalize the FFT or IFFT.
- int executeRCFFT (void *real_ptr, void *comp_ptr, int ndim, int N[3], int streamId=-1)

  Exectute R2C FFT.
- int executeCRFFT (void *real_ptr, void *comp_ptr, int ndim, int N[3], int streamId=-1)

  Exectute C2R FFT.
- int normalizeCRFFT (void *real_ptr, int ndim, int N[3], int streamId=-1)

  Normalize CR FFT.

Additional Inherited Members

4.11.1 Detailed Description

Cuda FFT class based on BaseFFT interface.

Uses cuFFT library to perform FFTs on nvidias GPUs.

4.11.2 Member Function Documentation

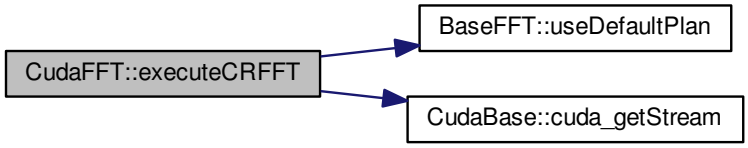
4.11.2.1 int CudaFFT::executeCRFFT (void * *real_ptr*, void * *comp_ptr*, int *ndim*, int *N[3]*, int *streamId* = `-1`) `[virtual]`

Exectute C2R FFT.

real_ptr - real output data from the C2R FFT, comp_ptr - complex input data for the FFT.

Implements BaseFFT.

Here is the call graph for this function:



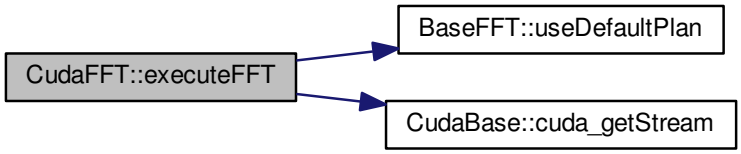
4.11.2.2 int CudaFFT::executeFFT (void * *mem_ptr*, int *ndim*, int *N[3]*, int *streamId* = -1, bool *forward* = true) [virtual]

Exectute C2C FFT.

mem_ptr - memory ptr on the device for complex data. Performs in place FFT.

Implements BaseFFT.

Here is the call graph for this function:



Here is the caller graph for this function:

4.11.2.3 int CudaFFT::executeIFFT (void * *mem_ptr*, int *ndim*, int *N[3]*, int *streamId* = -1) [virtual]

Exectute inverse C2C FFT.

mem_ptr - memory ptr on the device for complex data. Performs in place FFT.

Implements BaseFFT.

Here is the call graph for this function:

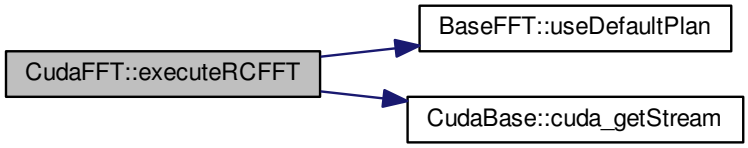
4.11.2.4 int CudaFFT::executeRCFFT (void * *real_ptr*, void * *comp_ptr*, int *ndim*, int *N[3]*, int *streamId* = -1) [virtual]

Exectute R2C FFT.

real_ptr - real input data for FFT, comp_ptr - memory on the device where results for the FFT are stored as complex numbers.

Implements BaseFFT.

Here is the call graph for this function:



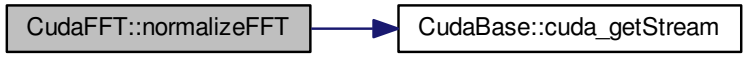
4.11.2.5 int CudaFFT::normalizeFFT (void * *mem_ptr*, int *ndim*, int *N[3]*, int *streamId* = -1) [virtual]

Normalize the FFT or IFFT.

mem_ptr - memory to complex data.

Implements BaseFFT.

Here is the call graph for this function:



4.11.2.6 int CudaFFT::setupFFTCR (int *ndim*, int *N[3]*, double *scale* = 1.0) [inline], [virtual]

Setup real to complex complex to real FFT - init FFT library used by chosen device.

Implements BaseFFT.

4.11.2.7 int CudaFFT::setupFFTRC (int *ndim*, int *N[3]*, double *scale* = 1.0) [inline], [virtual]

Setup real to complex FFT - init FFT library used by chosen device.

Implements BaseFFT.

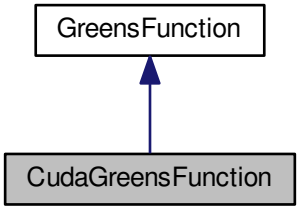
The documentation for this class was generated from the following files:

- src/CUDA/CudaFFT.cuh
- src/CUDA/CudaFFT.cu

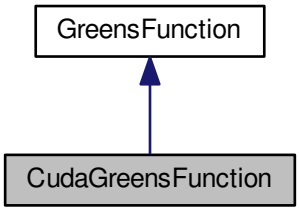
4.12 CudaGreensFunction Class Reference

CUDA implementation of GreensFunction calculation for OPALs Poisson Solver.

Inheritance diagram for CudaGreensFunction:



Collaboration diagram for CudaGreensFunction:



Public Member Functions

- CudaGreensFunction (CudaBase ∗base)

  Constructor with CudaBase argument.

- int greensIntegral (void ∗tmpgreen, int I, int J, int K, int NI, int NJ, double hr_m0, double hr_m1, double hr_m2, int streamId=-1)

  Info: calc itegral on device memory (taken from OPAL src code) Return: success or error code.

- int integrationGreensFunction (void ∗rho2_m, void ∗tmpgreen, int I, int J, int K, int streamId=-1)

  Info: integration of rho2_m field (taken from OPAL src code) Return: success or error code.

- int mirrorRhoField (void ∗rho2_m, int I, int J, int K, int streamId=-1)

  Info: mirror rho field (taken from OPAL src code) Return: succes or error code.

- int multiplyCompelxFields (void ∗ptr1, void ∗ptr2, int size, int streamId=-1)

  Info: multiply complex fields already on the GPU memory, result will be put in ptr1 Return: success or error code.

4.12.1 Detailed Description

CUDA implementation of GreensFunction calculation for OPALs Poisson Solver.

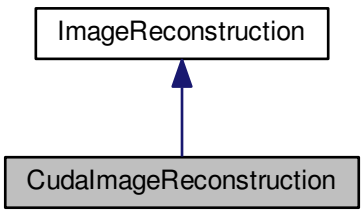
The documentation for this class was generated from the following files:

- src/CUDA/CudaGreensFunction.cuh
- src/CUDA/CudaGreensFunction.cu

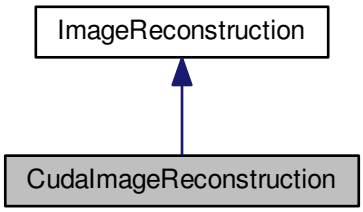
4.13 CudaImageReconstruction Class Reference

CUDA implementation of ImageReconstruction interface.

Inheritance diagram for CudaImageReconstruction:



Collaboration diagram for CudaImageReconstruction:



Public Member Functions

- CudaImageReconstruction ()
  Constructor.
- CudaImageReconstruction (CudaBase *base)
  Constructor with base.
- ~CudaImageReconstruction ()
  Destructor.
- int calculateSource (void *image_space, void *image_position, void *source_position, void *avg, void *std, float diameter, int total_voxels, int total_sources, int start=0)
  CUDA implementation of caluclate source.
- int calculateBackground (void *image_space, void *image_position, void *source_position, void *avg, void *std, float diameter, int total_voxels, int total_sources, int start=0)
  Cuda implementation of calculate background.
- int calculateSources (void *image_space, void *image_position, void *source_position, void *avg, void *std, void *diameter, int total_voxels, int total_sources, int start=0)
  Caluclate source for differente sources.
- int calculateBackgrounds (void *image_space, void *image_position, void *source_position, void *avg, void *std, void *diameter, int total_voxels, int total_sources, int start=0)
  Calculate background for differente sources.
- int generateNormalization (void *recon, void *image_position, void *det_position, int total_det)
  Generate normalization.
- int forwardProjection (void *correction, void *recon, void *list_data, void *det_position, void *image_position, int num_events)
  Calculate forward projection.
- int backwardProjection (void *correction, void *recon_corrector, void *list_data, void *det_position, void *image_position, int num_events, int num_voxels)
  Calculate backward projection.
- int setDimensions (int voxel_x, int voxel_y, int voxel_z, float voxel_size)
  Set the voxel dimensins on device.
- int setEdge (float x_edge, float y_edge, float z_edge)
  Set the image edge.
- int setEdge1 (float x_edge1, float y_edge1, float z_edge1, float z_edge2)
  Set the image edge1.
- int setMinCrystalInRing (float min_CrystalDist_InOneRing, float min_CrystalDist_InOneRing1)
  Set the minimum crystan in one ring values.
- int setParams (float matrix_distance_factor, float phantom_diameter, float atten_per_mm, float ring_diameter)
  Set all other required parameters for reconstruction.

Additional Inherited Members

4.13.1 Detailed Description

CUDA implementation of ImageReconstruction interface.

4.13.2 Member Function Documentation

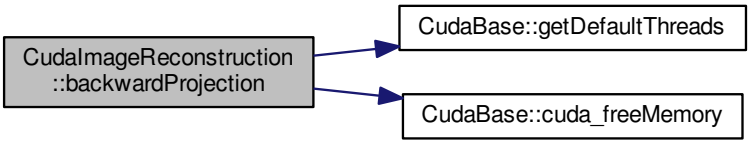
4.13.2.1 int CudaImageReconstruction::backwardProjection (void * *correction*, void * *recon_corrector*, void * *list_data*, void * *det_position*, void * *image_position*, int *num_events*, int *num_voxels*) [virtual]

Calculate backward projection.

For image reconstruction calculates backward projections. see recon.cpp for details

Implements ImageReconstruction.

Here is the call graph for this function:



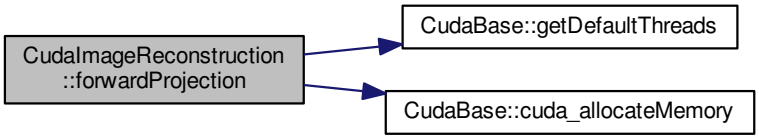
4.13.2.2 int CudaImageReconstruction::forwardProjection (void * *correction*, void * *recon*, void * *list_data*, void * *det_position*, void * *image_position*, int *num_events*) [virtual]

Calculate forward projection.

For image reconstruction calculates forward projections. see recon.cpp for details

Implements ImageReconstruction.

Here is the call graph for this function:



4.13.2.3 int CudaImageReconstruction::generateNormalization (void * *recon,* void * *image_position,* void * *det_position,* int *total_det*) [virtual]

Generate normalization.

Goes trough detectors pairs and if detector pair crosses image launches seperate kernel that updates voxel values in the image on the slope between these two detectors.

Implements ImageReconstruction.

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

- src/CUDA/CudaImageReconstruction.cuh
- src/CUDA/CudaImageReconstruction.cu

4.14 Device Class Reference

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

- src/DKSDevice.h

4.15 DKSAutoTuning Class Reference

DKS autotuning class, allows to auto-tune the defince function.

`#include <DKSAutoTuning.h>`

Public Member Functions

- DKSAutoTuning (DKSBase *base, std::string api, std::string device, int loops=100)

  Constructor.
- ~DKSAutoTuning ()

  Destructor.
- void setFunction (std::function< int()> f, std::string name, bool evaluate_time=true)

  Set function to auto tune.
- void setFunction (std::function< double()> f, std::string name, bool evaluate_time=false)

  Set function to auto tune.
- template<typename T1 >
  void addParameter (T1 *value, T1 min, T1 max, T1 step, std::string name)

  Set parameter for auto tuning.
- void clearParameters ()

  Delete all added parameters.
- void exaustiveSearch ()

  Perform exaustive search evaluating all the parameter configurations.
- void lineSearch ()

  Perform line-search auto-tuning by variying parameters one at a time.
- void hillClimbing (int restart_loops=1)

  Perform hill climbing.
- void simulatedAnnealing (double Tstart, double Tstep)

  Perfor simulated annealing to find the parameters.

4.15.1 Detailed Description

DKS autotuning class, allows to auto-tune the defince function.

Executes the defined function for auto-tuning and searches for optimal parameters to improve the function execution time. The function that is auto-tuned, parameters and the ranges need to be set. Includes multiple search methods, that searches the parameter space to finde the optimal solution. 1) exaustive search 2) line search 3) hill climbimg 4) simulated annealing

4.15.2 Constructor & Destructor Documentation

4.15.2.1 DKSAutoTuning::∼DKSAutoTuning ( )

Destructor.

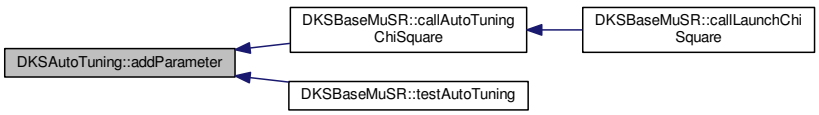
4.15.3 Member Function Documentation

4.15.3.1 template<typename T1 > void DKSAutoTuning::addParameter (T1 * *value*, T1 *min*, T1 *max*, T1 *step*, std::string *name*) [inline]

Set parameter for auto tuning.

Provide a pointer to a parameter that will be changed during auto-tuning and a min-max value for this element

Here is the caller graph for this function:

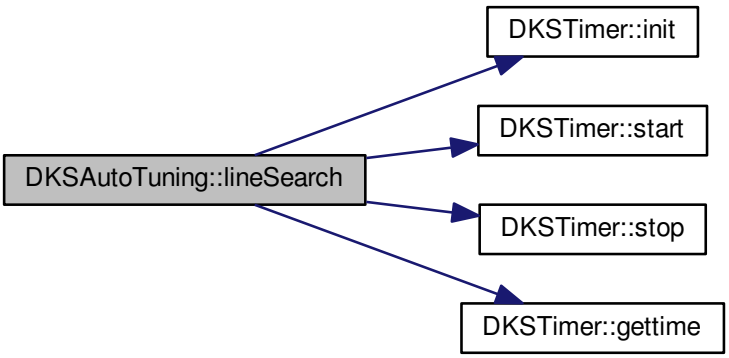


4.15.3.2 void DKSAutoTuning::lineSearch ( )

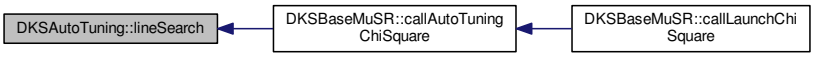
Perform line-search auto-tuning by variying parameters one at a time.

After one parameter is auto-tuned the next on is varied

Here is the call graph for this function:



Here is the caller graph for this function:

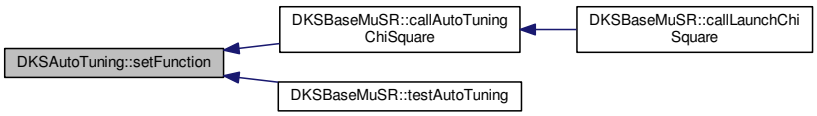


4.15.3.3 void DKSAutoTuning::setFunction (std::function< int()> *f*, std::string *name*, bool *evaluate_time* = `true`) `[inline]`

Set function to auto tune.

Caller of setFunction is responsible to bind the correct parameters to the function with std::bind.

Here is the caller graph for this function:



4.15.3.4 void DKSAutoTuning::setFunction (std::function< double()> *f*, std::string *name*, bool *evaluate_time* = `false`) `[inline]`

Set function to auto tune.

Caller of setFunction is responsible to bind the correct parameters to the function with std::bind.

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

- src/AutoTuning/DKSAutoTuning.h
- src/AutoTuning/DKSAutoTuning.cpp

4.16 DKSAutoTuningTester Class Reference

Tester class for auto-tuning search algorithms.

`#include <DKSAutoTuningTester.h>`

Public Member Functions

- double **peaksZ** ()

Friends

- class **DKSBaseMuSR**

4.16.1 Detailed Description

Tester class for auto-tuning search algorithms.

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

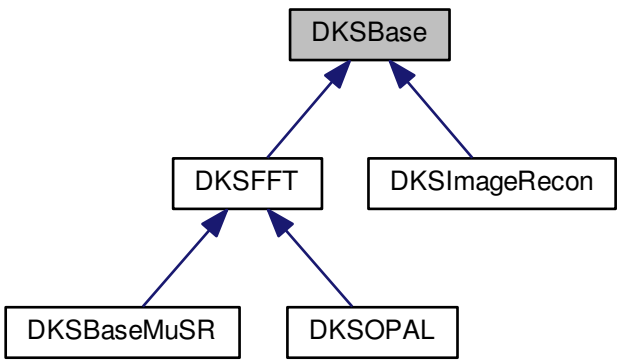
- src/AutoTuning/DKSAutoTuningTester.h

4.17 DKSBase Class Reference

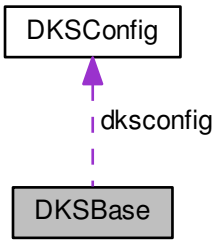
API for handling communication function calls to DKS library.

`#include <DKSBase.h>`

Inheritance diagram for DKSBase:



Collaboration diagram for DKSBase:



Public Member Functions

- DKSBase ()
  Default constructor.
- DKSBase (const char *api_name, const char *device_name)
  Constructor that sets api and devcie to use with DKS.
- ~DKSBase ()
  Destructor.
- int setupDevice ()
  Function to initialize objects based on the device used.
- void setAutoTuningOn ()
  Turn on auto tuning.
- void setAutoTuningOff ()
  Turn of auto tuning.
- bool isAutoTuningOn ()
  Get status of auto tuning.
- void setUseConfigOn ()
  Turn on use of config file.
- void setUseConfigOff ()
  Turn off use of config file.
- bool isUseConfigOn ()
  Check if using config file.
- int setDevice (const char *device_name, int length=-1)
  Set device to use with DKS.
- int setAPI (const char *api_name, int length=-1)
  Set framework to use with DKS.
- int getDevices ()
  Prints information about all available devices.
- int getDeviceCount (int &ndev)
  Returns device count.
- int getDeviceName (std::string &device_name)
  Get the name of the device in use.
- int setDefaultDevice (int device)
  Set the device to use.

- int getDeviceList (std::vector< int > &devices)

  Get unique devices.
- int initDevice ()

  Inititialize DKS.
- int createStream (int &streamId)

  Create stream for async execution.
- int closeHandle (void *mem_ptr)

  Send pointer to device memory from one MPI process to another.
- int syncDevice ()

  Wait till all tasks running on device are completed.
- template<typename T >
  void * pushData (const void *data_in, int elements, int &ierr)

  Allocate memory and transfer data to device.
- template<typename T >
  int pullData (void *mem_ptr, void *data_out, int elements)

  Read data from device and free device memory.
- template<typename T >
  void * allocateMemory (int elements, int &ierr)

  Allocate memory on device and return pointer to device memory.
- template<typename T >
  int allocateHostMemory (T *&ptr, int size)

  Allocates host memory as page-locked.
- template<typename T >
  int freeHostMemory (T *&ptr, int size)

  Free host page-locked memory.
- template<typename T >
  int registerHostMemory (T *ptr, int size)

  Page lock allocated host memory.
- template<typename T >
  int unregisterHostMemory (T *ptr)

  Unregister page locked memory.
- template<typename T >
  int writeData (void *mem_ptr, const void *data, int elements, int offset=0)

  Write data from host to device.
- template<typename T >
  int writeDataAsync (void *mem_ptr, const void *data, int elements, int streamId=-1, int offset=0)

  Write data to device using async write.
- template<typename T >
  int readData (const void *mem_ptr, void *out_data, int elements, int offset=0)

  Gather 3D data from multiple mpi processes to one memory region.
- template<typename T >
  int readDataAsync (const void *mem_ptr, void *out_data, int elements, int streamId=-1, int offset=0)

  Performs an async data read from device.
- template<typename T >
  int freeMemory (void *mem_ptr, int elements)

  Free memory allocated on device.
- int callCreateRandomNumbers (void *mem_ptr, int size)

  Create random numbers on the device and fille mem_data array.
- int callInitRandoms (int size, int seed=-1)

  Init random number states and save for reuse on device.
- int callMemInfo ()

  Print memory information on device (total, used, available) TODO: opencl and mic imlementation.

- void oclEventInfo ()

  Test function to profile opencl kernel calls.
- void oclClearEvents ()

  Test function to profile opencl kernel calls.

Protected Member Functions

- bool apiOpenCL ()

  Check if current API is set to OpenCL.
- bool apiCuda ()

  Check if current API is set to CUDA.
- bool apiOpenMP ()

  Check if current API is set to OpenMP.
- bool deviceGPU ()

  Check if device is GPU.
- bool deviceCPU ()

  Check if device is CPU.
- bool deviceMIC ()

  Check if device is MIC.
- int loadOpenCLKernel (const char *kernel_name)

  Get cbase pointer.
- std::string **getAPI** ()
- std::string **getDevice** ()

Protected Attributes

- DKSConfig **dksconfig**

4.17.1 Detailed Description

API for handling communication function calls to DKS library.

DKSBase class uses CudaBase, OpenCLBase and MICBase to handle setup of device, memory manegement, data transfer and other basic communication functions between the host and device.

4.17.2 Constructor & Destructor Documentation

4.17.2.1 DKSBase::~DKSBase ( )

Destructor.

Free DKS resources.

4.17.3 Member Function Documentation

4.17.3.1 template<typename T > int DKSBase::allocateHostMemory (T *& *ptr*, int *size*) [inline]

Allocates host memory as page-locked.

Used for memroy allocation on the host side for pointer ptr for size elements. Page locked memory improves data transfer rates between host and device and allows async data transfer and kernel execution. Reurns succes or error code. TODO: opencl and mic implementations needed.

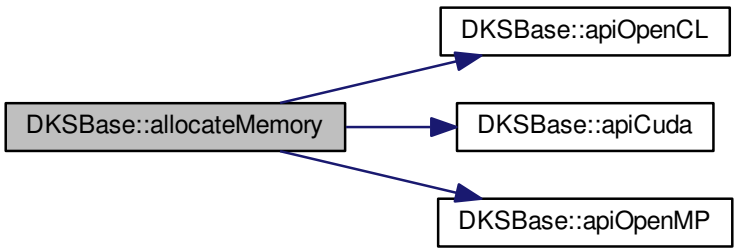
Here is the call graph for this function:

4.17.3.2 template<typename T > void* DKSBase::allocateMemory (int *elements*, int & *ierr*) [inline]

Allocate memory on device and return pointer to device memory.

Allocates memory of type T, elements specifies the number of elements for which memory should be allocated. If memory allocation fails ierr is set to error code. Returns void pointer to device memory.

Here is the call graph for this function:



4.17.3.3 bool DKSBase::apiCuda ( ) [protected]

Check if current API is set to CUDA.

Return true/false wether curretn api is cuda

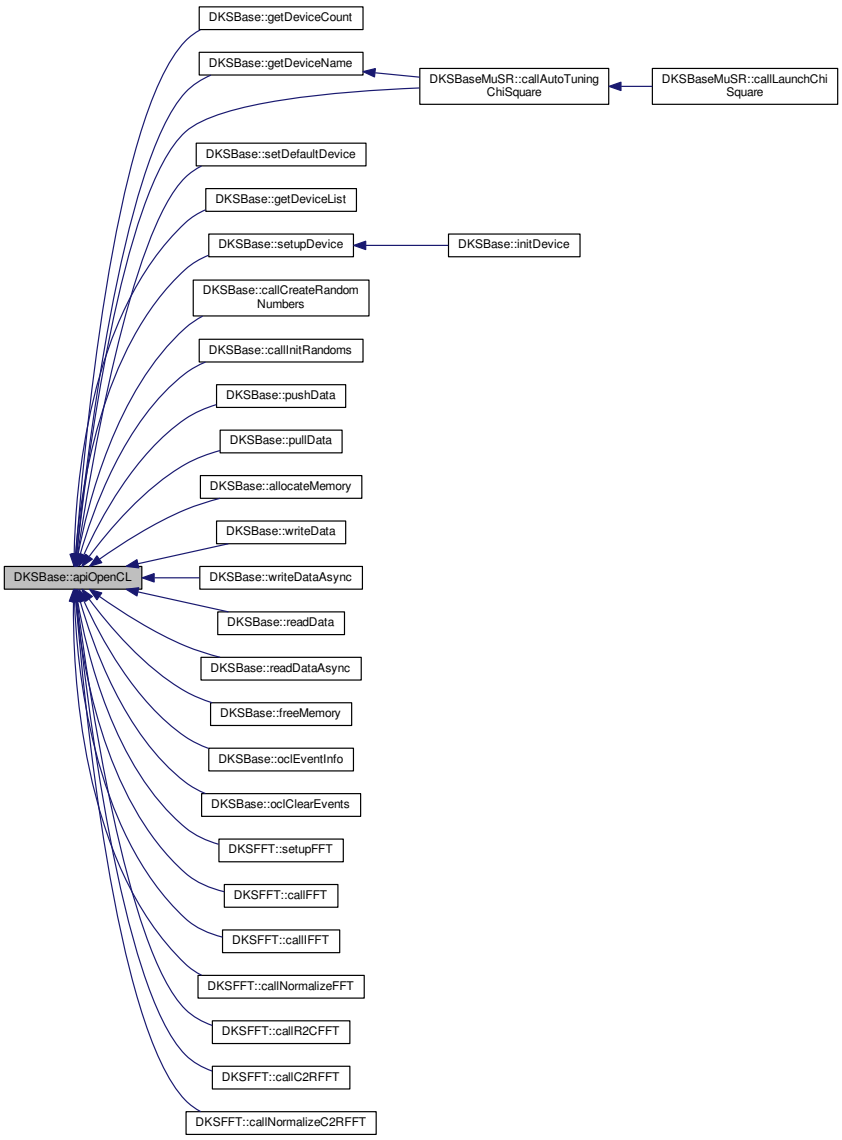
Here is the caller graph for this function:

4.17.3.4 bool DKSBase::apiOpenCL ( ) [protected]

Check if current API is set to OpenCL.

Return true/false wether current api is opencl

Here is the caller graph for this function:

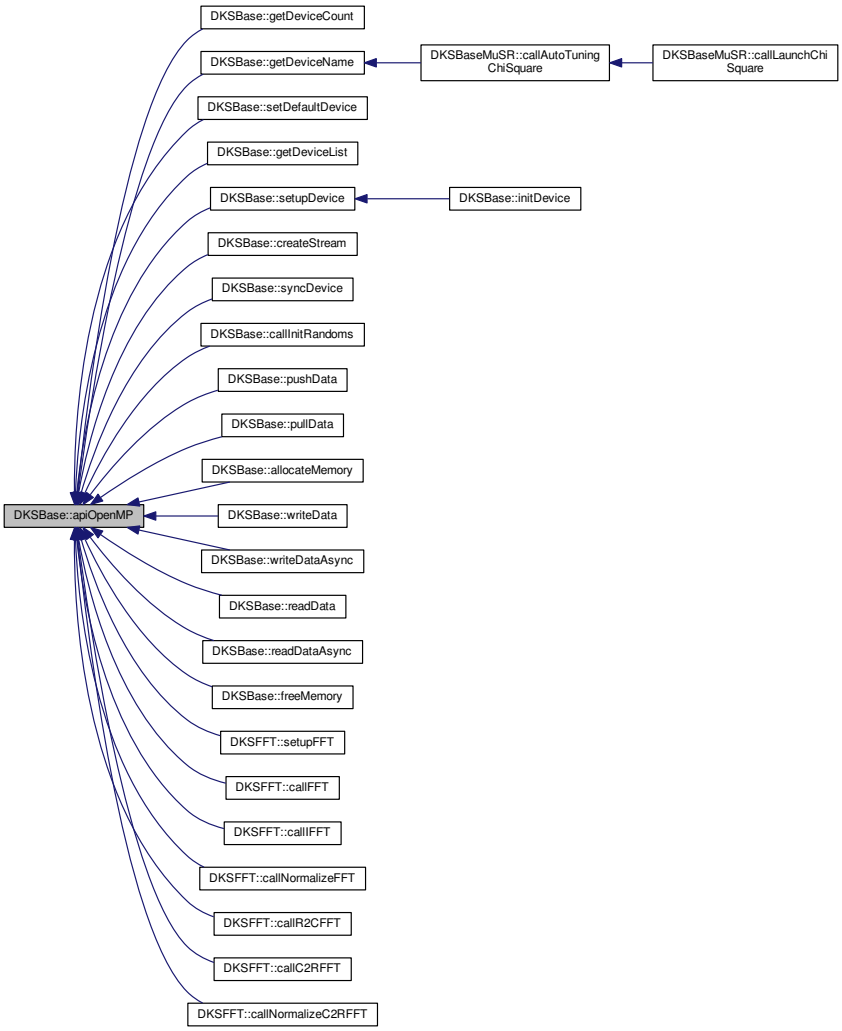


4.17.3.5 bool DKSBase::apiOpenMP ( ) [protected]

Check if current API is set to OpenMP.

Return true/false whether current api is OpenMP

Here is the caller graph for this function:

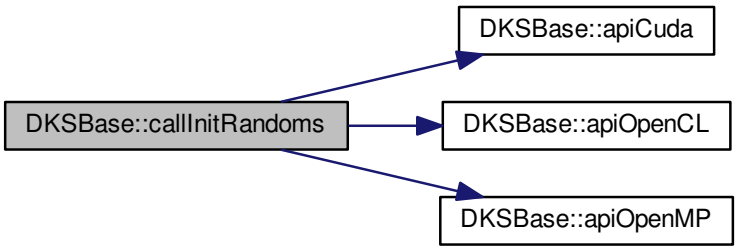


4.17.3.6 int DKSBase::callInitRandoms (int *size*, int *seed* = -1)

Init random number states and save for reuse on device.

If seed is -1, a random seed based on current time is taken. TODO: opencl and mic implementations.

Here is the call graph for this function:



4.17.3.7 int DKSBase::closeHandle (void * *mem_ptr*)

Send pointer to device memory from one MPI process to another.

Implemented only if mpi compiler is used to build DKS. Implemented only for cuda. Uses cuda icp. Gets icp handle of memory allocated on device pointed by mem_ptr does MPI_Send to dest process where matching receivePointer should be called. Returns success or error code. TODO: opencl and mic cases still need implementations Receive pointer to device memory from another MPI process. Implemented only if mpi compiler is used to build DKS. Implemented only for cuda. Uses cuda icp. Uses MPI_Recv to get icp handle from another MPI process and opens a reference to this memory. Togeter with sendPointer function allows multiple MPI processes to share one memory region of the device. Returns success or error code. TODO: opencl and mic cases still need implementations Close handle to device memory. If receivePointer is used to open memory handle allocated by another MPI process closeHandle should be called to free resources instead of freeMemory. Returns success or error code. TODO: opencl and mic cases still need implementations.

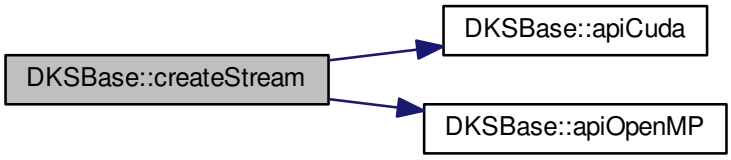
Here is the call graph for this function:

4.17.3.8 int DKSBase::createStream (int & *streamId*)

Create stream for async execution.

Function to create different streams with device to allow assync kernel execution and data transfer. Currently implemented for CUDA with cuda streams. streamId will be can be used later use the created stream. Returns success or error code. TODO: for opencl use different contexts similar as cuda streams to achieve async execution. TODO: for intel mic look at library (libxstream) from Hans Pabst.

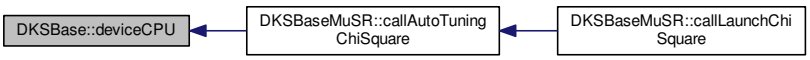
Here is the call graph for this function:



4.17.3.9 bool DKSBase::deviceCPU ( ) [protected]

Check if device is CPU.

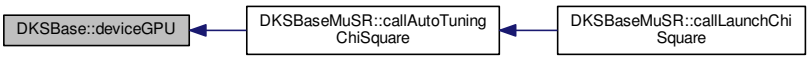
Here is the caller graph for this function:



4.17.3.10 bool DKSBase::deviceGPU ( ) [protected]

Check if device is GPU.

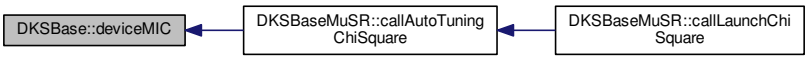
Here is the caller graph for this function:



4.17.3.11 bool DKSBase::deviceMIC ( ) [protected]

Check if device is MIC.

Here is the caller graph for this function:

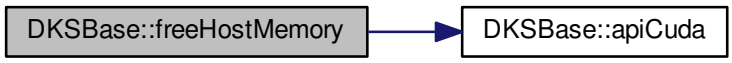


4.17.3.12 template<typename T > int DKSBase::freeHostMemory (T *& *ptr,* int *size*) [inline]

Free host page-locked memory.

Used to free page-locked memory on the host that was allocated using allocateHostMemory. ptr is the host pointer where page-locked memory was allocated, size - number of elements held by the memroy.

Here is the call graph for this function:

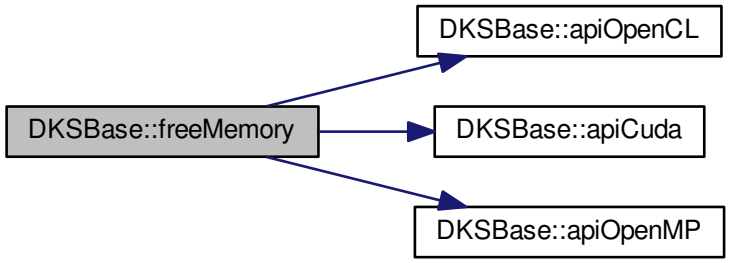


4.17.3.13 template<typename T > int DKSBase::freeMemory (void * *mem_ptr,* int *elements*) [inline]

Free memory allocated on device.

Free memory referenced by mem_ptr, elements - number of elements in memory, T - data type.

Here is the call graph for this function:

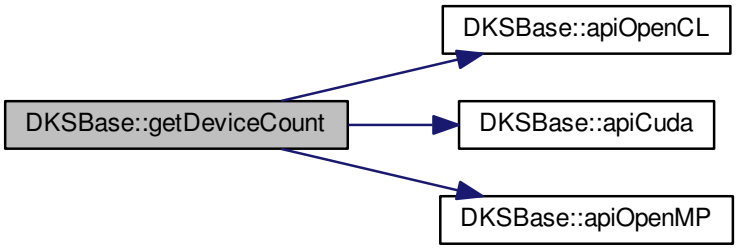


4.17.3.14 int DKSBase::getDeviceCount (int & *ndev*)

Returns device count.

Saves the number of the devices available on the platform to ndev.

Here is the call graph for this function:

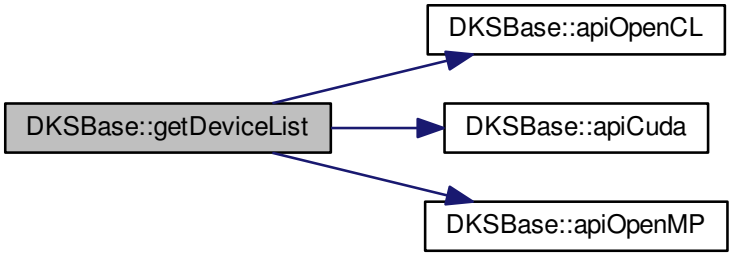


4.17.3.15 int DKSBase::getDeviceList (std::vector< int > & *devices*)

Get unique devices.

Get a list of all the unique devices available on the platform. When API and device type for DKS is set, getDeviceList can get all the unique devices available for this API and device type. Used for autotuning if multiple different GPUs are installed on the system.

Here is the call graph for this function:

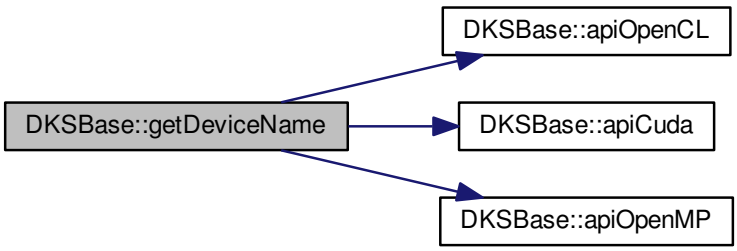


4.17.3.16 int DKSBase::getDeviceName (std::string & *device_name*)

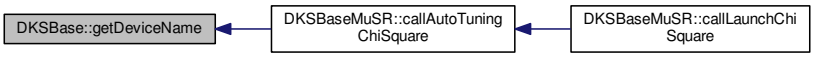
Get the name of the device in use.

Query the device that is used and get the naem of the device. The name is saved in the device_name string. Returns DKS_SUCCESS

Here is the call graph for this function:



Here is the caller graph for this function:



4.17.3.17 int DKSBase::getDevices ( )

Prints information about all available devices.

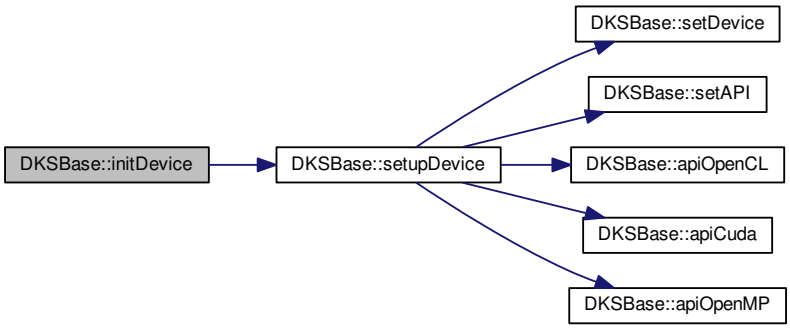
Calls CUDA, OpenCL and MIC functions to query for available devices for each framework and pirnts information about each device. Length specifies the number of characters in api_name array Returns success or error code

4.17.3.18 int DKSBase::initDevice ( )

Inititialize DKS.

Set framework and device to use. If OpenCL is used create context with device. Return success or error code.

Here is the call graph for this function:



4.17.3.19 int DKSBase::loadOpenCLKernel (const char * *kernel_name*) [protected]

Get cbase pointer.

Call OpenCL base to load specified kenrel file.

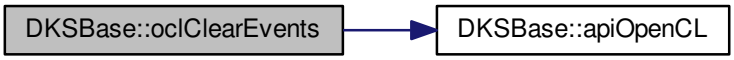
Here is the caller graph for this function:

4.17.3.20 void DKSBase::oclClearEvents ( ) [inline]

Test function to profile opencl kernel calls.

Used for debuging and timing purposes only.

Here is the call graph for this function:



4.17.3.21 void DKSBase::oclEventInfo ( ) [inline]

Test function to profile opencl kernel calls.

Used for debuging and timing purposes only.

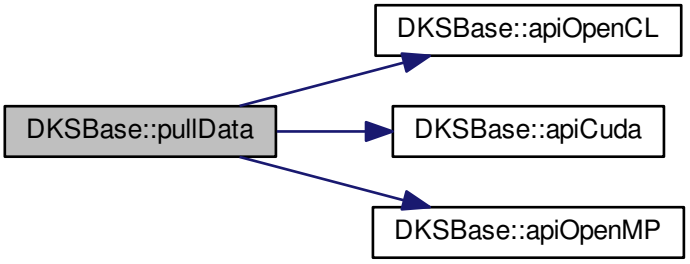
Here is the call graph for this function:

4.17.3.22 template<typename T > int DKSBase::pullData (void * *mem_ptr,* void * *data_out,* int *elements*) [inline]

Read data from device and free device memory.

Reads data from device pointed by mem_ptr into data_out pointer. Elements specifies the number of data elements to read, T specifies the datatype of elements to copy. Returns error code if read data or free memory fails.

Here is the call graph for this function:

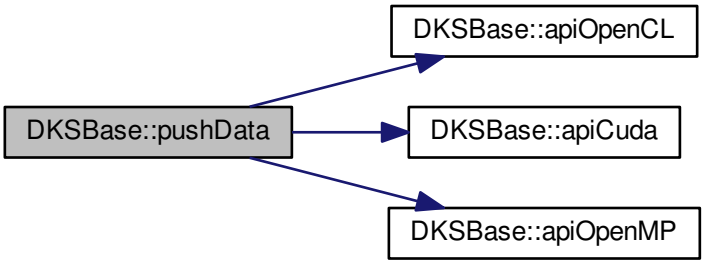


4.17.3.23 template<typename T > void* DKSBase::pushData (const void * *data_in,* int *elements,* int & *ierr*) [inline]

Allocate memory and transfer data to device.

Returns a void pointer which can be used in later kernels to reference allocated device memory. data_in pointer to data to be transfered to device, elements is the number of data elements to transfer, T - type of data to transfer. If memory allocation or data transfer fails ierr will be set to error code.

Here is the call graph for this function:



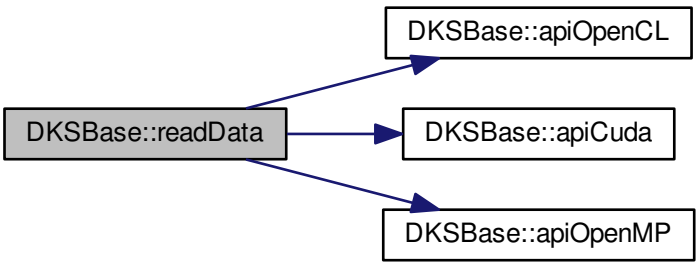
4.17.3.24 template<typename T > int DKSBase::readData (const void * *mem_ptr,* void * *out_data,* int *elements,* int *offset* = 0) [inline]

Gather 3D data from multiple mpi processes to one memory region.

When multiple processes share the same device memory using sendPointer and receivePointer gather3DDataAsync allows each process to write data to its memory region. Uses async writes. mem_ptr - device pointer, data - host pointer, Ng - global dimensions of data, Nl - local data dimensions, id - starting indexes in global domain for each process streamId - stream to use for data transfers. Returns success or error code. Scatter 3D data to multiple

MPI processes from one device memory region. When multiple processes share the same device memory using sendPointer and receivePointer scatter3DDataAsync allows each process to read data from its memory region. Uses async reads. mem_ptr - device pointer, data - host pointer, Ng - global dimensions of data, Nl - local data dimensions, id - starting indexes in global domain for each process streamId - stream to use for data transfers. Returns success or error code. Create MPI subarray for 3D data gather and scatter using cuda aware MPI. If multiple MPI processes share device and cuda aware MPI is used for data transfer creates a MPI subarray so each MPI process can write and read to its own memory region. N_global - global domain dimensions, N_local - local domain dimensions, datatype - MPI datatype Gather 3D data from multiple MPI processes to device using cuda aware MPI. Using cuda aware mpi allows to gather data to one device memory region allocated by one of the mpi processes. mem_ptr - device pointer, data - host memory pointer, size - number of elements to transfer, stype - data type of elements, N_global - global dimensions of the domain, N_local - local domain dimensions, idx,idy,idz - starting indexes in global domain for each process, numNodes - number of processes, myNode - current node, rootNode - node that allocated device memory, comm - MPI communicator TODO: opencl and mic implementations (solution other than cuda aware mpi needed). Gather 3D data from multiple MPI processes to device using cuda aware MPI and non blocking gather. For detailed parameter description see gather3DData docs. TODO: opencl and mic implementations (solution other than cuda aware mpi needed). Scatter 3D data from device to multiple MPI processes using cuda aware MPI. If multiple MPI prcesses share one device allows to scatter 3D data regions from device memory allocated by one of the processes to all other MPI processes. For detailed parameter description see gather3DData docs. TODO: opencl and mic implementations (solution other than cuda aware mpi needed). Read data from device memory. Read data referenced by mem_ptr int out_data. Elements indicates the number of data elements to read and offset is the offset on the device from start of the memroy. Data type to read is specified by T. Performs a blocking read.

Here is the call graph for this function:

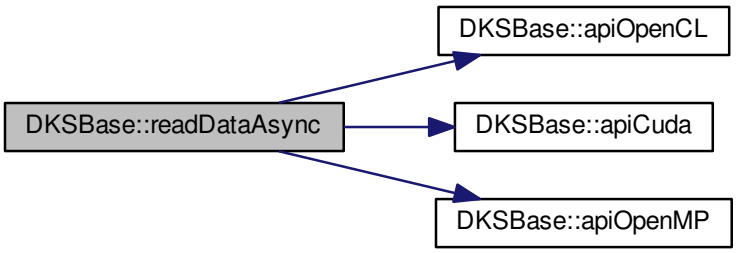


4.17.3.25 template<typename T > int DKSBase::readDataAsync (const void * *mem_ptr*, void * *out_data*, int *elements*, int *streamId* = -1, int *offset* = 0) [inline]

Performs an async data read from device.

Queues data read from device and returns control to host. stream id specifies stream to use for the read. Device async read can be performed if host memroy is page-locked and strema other than default -1 is used. For other parameter detailed description see readData function. TODO: opencl and mic implementations (currently reverts to blocking reads).

Here is the call graph for this function:



4.17.3.26 template<typename T > int DKSBase::registerHostMemory (T * *ptr,* int *size*) [inline]

Page lock allocated host memory.

Page locked memory improves data transfer between host and device (true for cuda and opencl, maybe also mic). ptr - pointer to memory that needs to be page locked, size - number of elements in array. TODO: mic and opencl implementations needed

Here is the call graph for this function:

4.17.3.27 int DKSBase::setAPI (const char * *api_name,* int *length* = -1)

Set framework to use with DKS.

Sets framework and API that DKS uses to execute code on device. Supported API's are OpenCL, CUDA and OpenMP. Returns success or error code. Length specifies the number of characters in api_name array (length - deprecated).

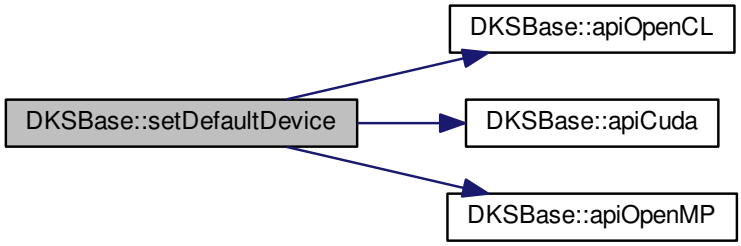
Here is the caller graph for this function:

4.17.3.28 int DKSBase::setDefaultDevice (int *device*)

Set the device to use.

Pass the index of the device to use by dks.

Here is the call graph for this function:

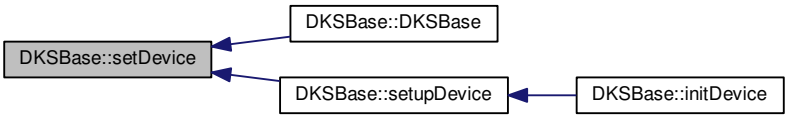


4.17.3.29 int DKSBase::setDevice (const char * *device_name,* int *length* = -1)

Set device to use with DKS.

Sets specific device to use with DKS. Supported devices are -gpu and -mic. Length specifies the number of characters in device_name array (length - deprecated). Return success or error code.

Here is the caller graph for this function:

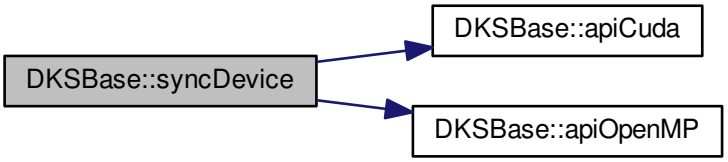


4.17.3.30 int DKSBase::syncDevice ( )

Wait till all tasks running on device are completed.

Forces a device synchronization - waits till all tasks on the device are complete. Implemented for cuda. Forces sync only in context in witch it is called - only waits for tasks launched by process calling syncDevice. If multiple processes launch different tasks each process is responsible for its own synchronization. Returns success or error code. TODO: opencl and mic implementations still necessary

Here is the call graph for this function:



4.17.3.31 template<typename T > int DKSBase::unregisterHostMemory (T * *ptr*) [inline]

Unregister page locked memory.

TODO: opencl and mic implementations needed·

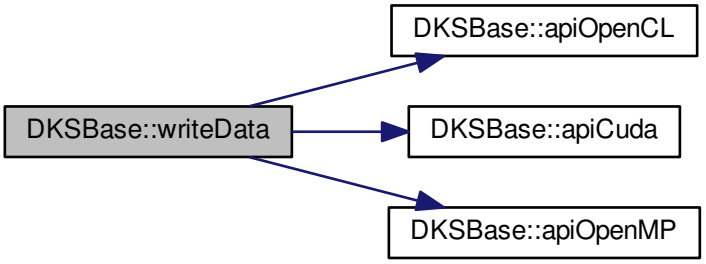
Here is the call graph for this function:

4.17.3.32 template<typename T > int DKSBase::writeData (void * *mem_ptr,* const void * *data,* int *elements,* int *offset* = 0) [inline]

Write data from host to device.

Write data from data to device memory referenced by mem_ptr. Elements spicify the number of elements to write, offset specifies the offset from the first element. Returns success or error code. Performs a blocking write - control to the host is returned only when data transfer is complete.

Here is the call graph for this function:

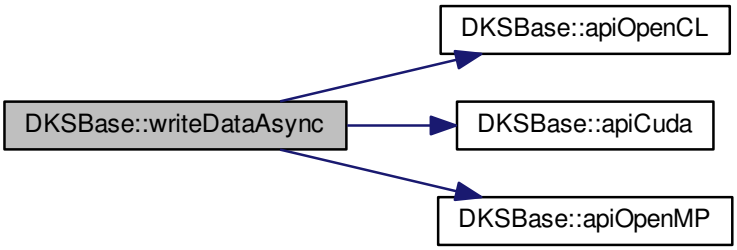


4.17.3.33 template<typename T > int DKSBase::writeDataAsync (void * *mem_ptr,* const void * *data,* int *elements,* int *streamId* = -1, int *offset* = 0) [inline]

Write data to device using async write.

Queue a async data write and return control to host imediately. mem_ptr - device memory pointer, data - host memory pointer, elements - number of data elements to write streamId - stream id to use, offset - offset on device from first element For trully async execution on cuda stream other than default needs to be created and device memory must be page-locked. Otherwise functions just asynchronosly with respect to host. TODO: mic and opencl implementations needed (goes to blocking writes)

Here is the call graph for this function:



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

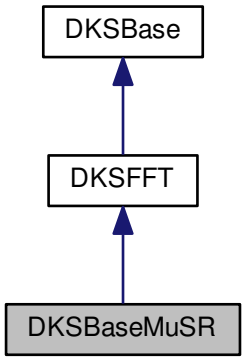
- src/DKSBase.h
- src/DKSBase.cpp

4.18 DKSBaseMuSR Class Reference

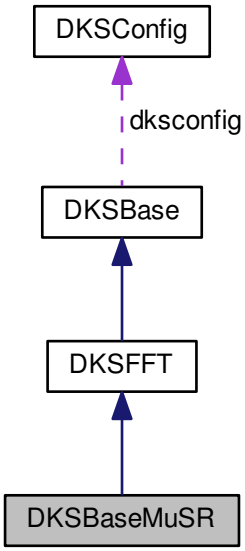
API to handle musrfit calls to DKS library.

`#include <DKSBaseMuSR.h>`

Inheritance diagram for DKSBaseMuSR:



Collaboration diagram for DKSBaseMuSR:



Public Member Functions

- int callCompileProgram (std::string function, bool mlh=false)

  Compile the program with kernels to be run.
- int callLaunchChiSquare (int fitType, void ∗mem_data, void ∗mem_err, int length, int numpar, int numfunc, int nummap, double timeStart, double timeStep, double &result)

Launch chi square calculation on data set writen in mem_data memory on device.

- int callAutoTuningChiSquare (int fitType, void *mem_data, void *mem_err, int length, int numpar, int numfunc, int nummap, double timeStart, double timeStep, double &result, std::vector< int > &config)

  Launch auto-tuning of chisquare function for the selected device.

- int callSetConsts (double N0, double tau, double bkg)

  Set N0, tau and BKG values for the run.

- int callSetConsts (double alpha, double beta)

  Set alpha and beta values for the run.

- int initChiSquare (int size_data, int size_param, int size_func, int size_map)

  Init chisquare calculations.

- int freeChiSquare ()

  Free temporary device storage allocated for chi^2 kernel.

- int writeParams (const double *params, int numparams)

  Write params to device.

- int writeFunctions (const double *func, int numfunc)

  Write function values to device.

- int writeMaps (const int *map, int numfunc)

  Write map indexes to device.

- int checkMuSRKernels (int fitType)

  Check if device can run necessary kernels.

- int checkMuSRKernels (int fitType, int &threadsPerBlock)

  Perform the same check as checkMuSRKernels(int fitType) and return max threads per block.

- int testAutoTuning ()

  Debug function to test auto-tuning search functions.

- int getOperations (int &oper)

  Get the number of operations in compiled kernel.

Additional Inherited Members

4.18.1 Detailed Description

API to handle musrfit calls to DKS library.

Using ChiSquareRuntime interface allows to call chi square functions on the GPU or CPU using CUDA or OpenCL.

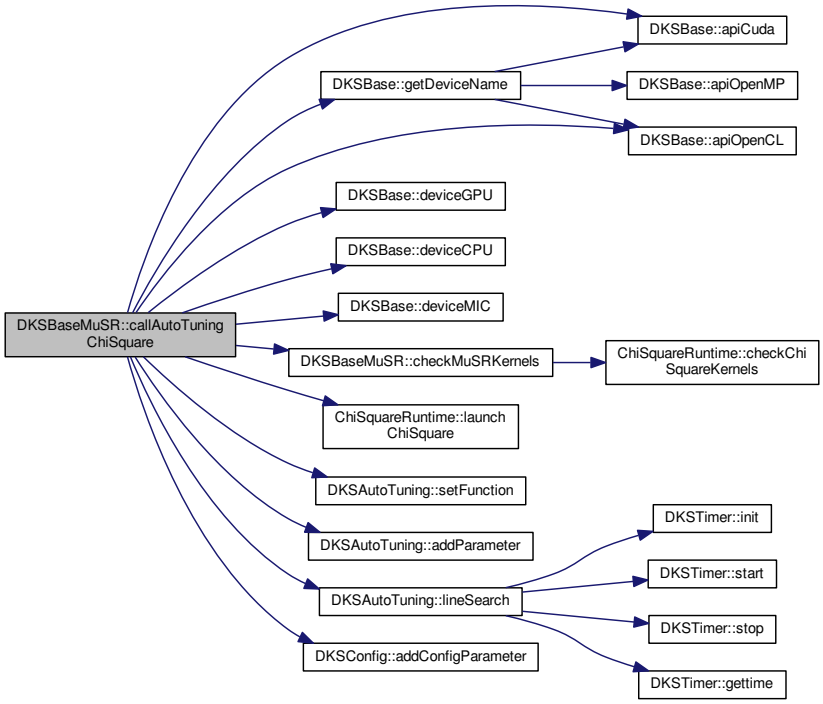
4.18.2 Member Function Documentation

4.18.2.1 int DKSBaseMuSR::callAutoTuningChiSquare (int *fitType*, void * *mem_data*, void * *mem_err*, int *length*, int *numpar*, int *numfunc*, int *nummap*, double *timeStart*, double *timeStep*, double & *result*, std::vector< int > & *config*)

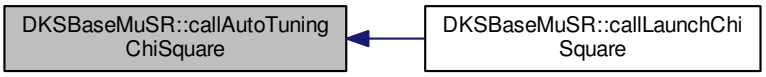
Launch auto-tuning of chisquare function for the selected device.

Creates a function pointer to callLaunchChiSquare with necessary arguments bind to function call. CUDA and OpenCL version - gives AutoTuning class access to numThreads parameter which is varied to find the optimal value by AutoTuning class. Uses brute force method to test all the values.

Here is the call graph for this function:



Here is the caller graph for this function:

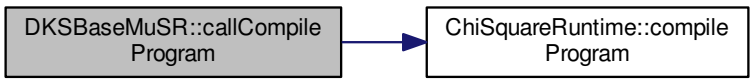


4.18.2.2 int DKSBaseMuSR::callCompileProgram (std::string *function*, bool *mlh* = `false`)

Compile the program with kernels to be run.

String function contains the string that will be added to the code to compile in the function: **device** double f-Theory(double t, double ∗p, double ∗f, int ∗m); Function string must be a valid C math expression. It can contain operators, math functions and predefined functions listed in: http://lmu.web.psi.ch/musrfit/user/-MUSR/MusrFit.html#A_4.3_The_THEORY_Block Predifined functions can be accessed by the abbreviation given in the table Parameters can be accesed in form p[idx] or p[m[idx]] - where p represents parameter array m represents map array and idx is the index to use from the maps. Precalculated function values can be accessed the same way - f[idx] or f[m[idx]]. Returns DKS_SUCCESS if everythin runs successfully, otherwise returns DKS_ERROR. If DKS is compiled with debug flag enabled prints DKS error message in case something fails

Here is the call graph for this function:

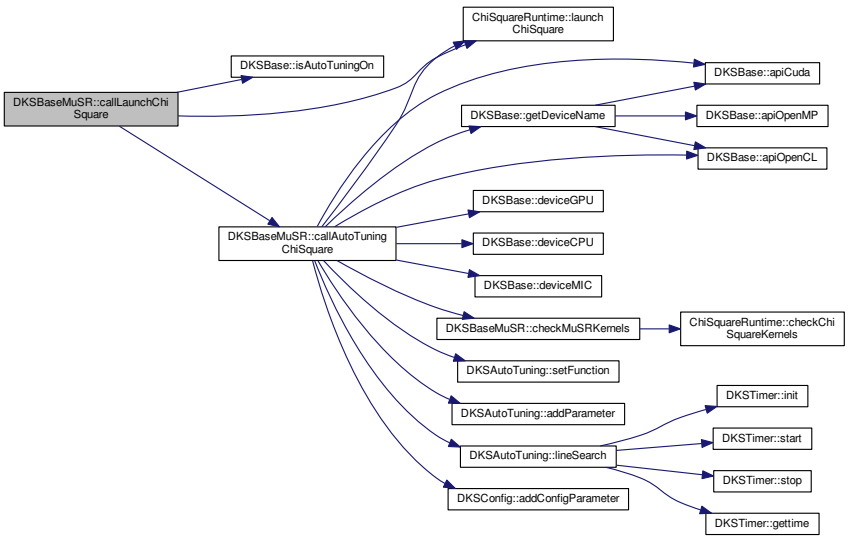


4.18.2.3 int DKSBaseMuSR::callLaunchChiSquare (int *fitType,* void * *mem_data,* void * *mem_err,* int *length,* int *numpar,* int *numfunc,* int *nummap,* double *timeStart,* double *timeStep,* double & *result*)

Launch chi square calculation on data set writen in mem_data memory on device.

mem_par, mem_map and mem_func hold pointers to parameter, function and map values for this data set (parameter array is one for all the data sets, maps and functions change between data sets). Resulting chi square value for this dataset will be put in result variable. Returns DKS_SUCCESS if everythin runs successfully, otherwise returns DKS_ERROR. If DKS is compiled with debug flag enabled prints DKS error message in case something fails

Here is the call graph for this function:



4.18.2.4 int DKSBaseMuSR::callSetConsts (double *N0,* double *tau,* double *bkg*)

Set N0, tau and BKG values for the run.

Needs to be called before kernel launch if these values are changing

Here is the call graph for this function:

4.18.2.5 int DKSBaseMuSR::callSetConsts (double *alpha,* double *beta*)

Set alpha and beta values for the run.

Needs to be called before kernel launch if these values are changing

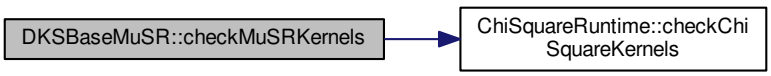
Here is the call graph for this function:

4.18.2.6 int DKSBaseMuSR::checkMuSRKernels (int *fitType*)

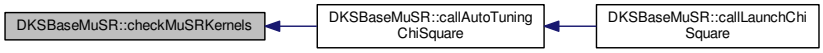
Check if device can run necessary kernels.

Check selected device properties to see if device suports double precision and if device can run the necessary number of work_items / work_groups to successfully execute CUDA/OpenCL kernels.

Here is the call graph for this function:



Here is the caller graph for this function:

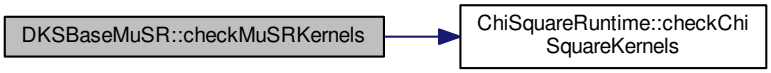


4.18.2.7 int DKSBaseMuSR::checkMuSRKernels (int *fitType,* int & *threadsPerBlock*)

Perform the same check as checkMuSRKernels(int fitType) and return max threads per block.

Used for autotuning to check what is the device limit for threads per block to correctly set the upper bound when searching the parameter space.

Here is the call graph for this function:

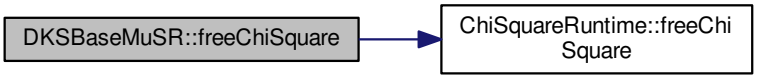


4.18.2.8 int DKSBaseMuSR::freeChiSquare ( )

Free temporary device storage allocated for chi^2 kernel.

Return error code if freeing the device fails.

Here is the call graph for this function:

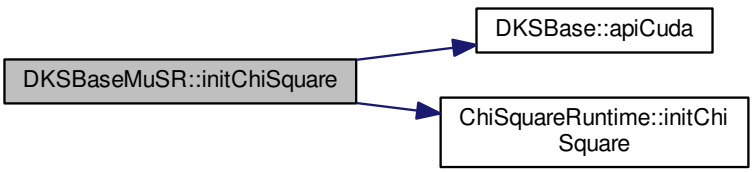


4.18.2.9 int DKSBaseMuSR::initChiSquare (int *size_data,* int *size_param,* int *size_func,* int *size_map*)

Init chisquare calculations.

Size is the maximum number of elements in any of the data sets used.

Here is the call graph for this function:



4.18.2.10 int DKSBaseMuSR::writeFunctions (const double * *func,* int *numfunc*)

Write function values to device.

Write precalculated function values to device, memory for functions on device is handled by DKS.

Here is the call graph for this function:

4.18.2.11 int DKSBaseMuSR::writeMaps (const int * *map,* int *numfunc*)

Write map indexes to device.

Write map indexes to use in defined theory function to devive. Memory for map indexes is handeld by DKS.

Here is the call graph for this function:

4.18.2.12 int DKSBaseMuSR::writeParams (const double * *params,* int *numparams*)

Write params to device.

Write pramas from double array to device, params device memory is managed by DKS.

Here is the call graph for this function:

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

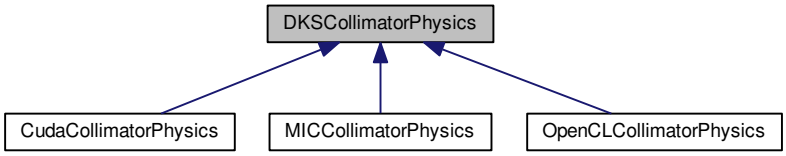
- src/DKSBaseMuSR.h
- src/DKSBaseMuSR.cpp

4.19 DKSCollimatorPhysics Class Reference

Interface to impelment particle matter interaction for OPAL.

`#include <CollimatorPhysics.h>`

Inheritance diagram for DKSCollimatorPhysics:



Public Member Functions

- virtual int CollimatorPhysics (void *mem_ptr, void *par_ptr, int numpartices, bool enableRutherfor-Scattering=true)=0

  Execute collimator physics kernel.
- virtual int CollimatorPhysicsSoA (void *label_ptr, void *localID_ptr, void *rx_ptr, void *ry_ptr, void *rz_ptr, void *px_ptr, void *py_ptr, void *pz_ptr, void *par_ptr, int numparticles)=0

  Special calse CollimatorPhysics kernel that uses SoA instead of AoS.
- virtual int CollimatorPhysicsSort (void *mem_ptr, int numparticles, int &numaddback)=0

  Sort particle array on GPU.
- virtual int CollimatorPhysicsSortSoA (void *label_ptr, void *localID_ptr, void *rx_ptr, void *ry_ptr, void *rz_ptr, void *px_ptr, void *py_ptr, void *pz_ptr, void *par_ptr, int numparticles, int &numaddback)=0

  Special calse CollimatorPhysicsSort kernel that uses SoA instead of AoS.
- virtual int ParallelTTrackerPush (void *r_ptr, void *p_ptr, int npart, void *dt_ptr, double dt, double c, bool usedt=false, int streamId=-1)=0

  BorisPusher push function for integration from OPAL.
- virtual int ParallelTTrackerKick (void *r_ptr, void *p_ptr, void *ef_ptr, void *bf_ptr, void *dt_ptr, double charge, double mass, int npart, double c, int streamId=-1)=0

  BorisPusher kick function for integration from OPAL.
- virtual int ParallelTTrackerPushTransform (void *x_ptr, void *p_ptr, void *lastSec_ptr, void *orient_ptr, int npart, int nsec, void *dt_ptr, double dt, double c, bool usedt=false, int streamId=-1)=0

  BorisPusher push function with transformto function form OPAL.

Protected Attributes

- int **numBlocks_m**
- int **blockSize_m**

4.19.1 Detailed Description

Interface to impelment particle matter interaction for OPAL.

4.19.2 Member Function Documentation

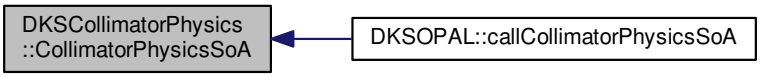
4.19.2.1 virtual int DKSCollimatorPhysics::CollimatorPhysicsSoA (void * *label_ptr,* void * *localID_ptr,* void * *rx_ptr,* void * *ry_ptr,* void * *rz_ptr,* void * *px_ptr,* void * *py_ptr,* void * *pz_ptr,* void * *par_ptr,* int *numparticles*) [pure virtual]

Special calse CollimatorPhysics kernel that uses SoA instead of AoS.

Used only on the MIC side, was not implemented on the GPU.

Implemented in CudaCollimatorPhysics, OpenCLCollimatorPhysics, and MICCollimatorPhysics.

Here is the caller graph for this function:



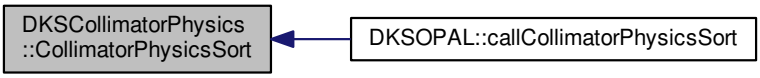
4.19.2.2 virtual int DKSCollimatorPhysics::CollimatorPhysicsSort (void * *mem_ptr,* int *numparticles,* int & *numaddback*) [pure virtual]

Sort particle array on GPU.

Count particles that are dead (label -1) or leaving material (label -2) and sort particle array so these particles are at the end of array

Implemented in CudaCollimatorPhysics, OpenCLCollimatorPhysics, and MICCollimatorPhysics.

Here is the caller graph for this function:



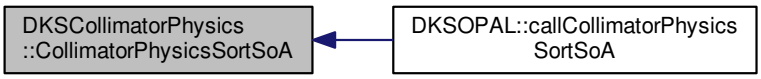
4.19.2.3 virtual int DKSCollimatorPhysics::CollimatorPhysicsSortSoA (void * *label_ptr,* void * *localID_ptr,* void * *rx_ptr,* void * *ry_ptr,* void * *rz_ptr,* void * *px_ptr,* void * *py_ptr,* void * *pz_ptr,* void * *par_ptr,* int *numparticles,* int & *numaddback*) [pure virtual]

Special calse CollimatorPhysicsSort kernel that uses SoA instead of AoS.

Used only on the MIC side, was not implemented on the GPU.

Implemented in CudaCollimatorPhysics, OpenCLCollimatorPhysics, and MICCollimatorPhysics.

Here is the caller graph for this function:



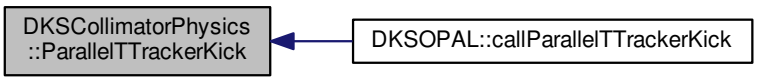
4.19.2.4 virtual int DKSCollimatorPhysics::ParallelTTrackerKick (void * *r_ptr,* void * *p_ptr,* void * *ef_ptr,* void * *bf_ptr,* void * *dt_ptr,* double *charge,* double *mass,* int *npart,* double *c,* int *streamId* = -1) [pure virtual]

BorisPusher kick function for integration from OPAL.

ParallelTTracker integration from OPAL implemented in cuda. For more details see ParallelTTracler docomentation in opal

Implemented in CudaCollimatorPhysics.

Here is the caller graph for this function:



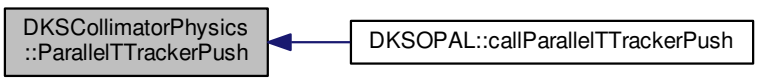
4.19.2.5 virtual int DKSCollimatorPhysics::ParallelTTrackerPush (void * *r_ptr,* void * *p_ptr,* int *npart,* void * *dt_ptr,* double *dt,* double *c,* bool *usedt* = false, int *streamId* = -1) [pure virtual]

BorisPusher push function for integration from OPAL.

ParallelTTracker integration from OPAL implemented in cuda. For more details see ParallelTTracler docomentation in opal

Implemented in CudaCollimatorPhysics, OpenCLCollimatorPhysics, and MICCollimatorPhysics.

Here is the caller graph for this function:



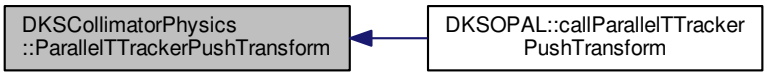
4.19.2.6 virtual int DKSCollimatorPhysics::ParallelTTrackerPushTransform (void * *x_ptr*, void * *p_ptr*, void * *lastSec_ptr*, void * *orient_ptr*, int *npart*, int *nsec*, void * *dt_ptr*, double *dt*, double *c*, bool *usedt* = `false`, int *streamId* = `-1`) `[pure virtual]`

BorisPusher push function with transformto function form OPAL.

ParallelTTracker integration from OPAL implemented in cuda. For more details see ParallelTTracler docomentation in opal

Implemented in CudaCollimatorPhysics, OpenCLCollimatorPhysics, and MICCollimatorPhysics.

Here is the caller graph for this function:



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

- src/Algorithms/CollimatorPhysics.h

4.20 DKSConfig Class Reference

Class to save and load DKS autotunning configs.

`#include <DKSConfig.h>`

Public Member Functions

- DKSConfig ()

  Constructor, set home variable.
- int loadConfigFile ()

  Load autotuinig.xml into tree variable if this file exists.
- int saveConfigFile ()

  Save autotuning.xml file.
- int addConfigParameter (const std::string api, const std::string device, const std::string name, const std::string func, int size, std::string param, int value)

  Add config parameter to tree.
- int getConfigParameter (const std::string api, const std::string device, const std::string name, const std::string func, int size, std::string param, int &value)

  Get config parameter from the tree.

4.20.1 Detailed Description

Class to save and load DKS autotunning configs.

Autotuning settings are saved and loaded from $HOME/.config/DKS/autotuning.xml. Uses boost xml_parser to read and write the xml file and boost property tree to store the xml content. TODO: need an update boost::filesystem is disabled at the moment, no configuration file is saved so the auto-tuning has no effect.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 DKSConfig::DKSConfig ( )

Constructor, set home variable.

If home directory is not set config file can not be read or saved

Here is the call graph for this function:

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

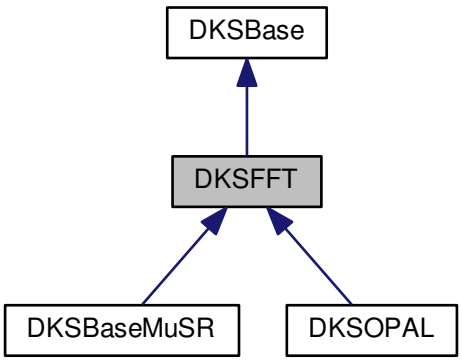
- src/AutoTuning/DKSConfig.h
- src/AutoTuning/DKSConfig.cpp

4.21 DKSFFT Class Reference

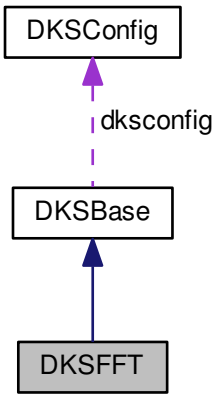
API to handel calls to DKSFFT.

`#include <DKSFFT.h>`

Inheritance diagram for DKSFFT:



Collaboration diagram for DKSFFT:



Public Member Functions

- int setupFFT (int ndim, int N[3])

  Setup FFT function.
- int **setupFFTRC** (int ndim, int N[3], double scale=1.0)
- int **setupFFTCR** (int ndim, int N[3], double scale=1.0)
- int callFFT (void *data_ptr, int ndim, int dimsize[3], int streamId=-1)

  Call complex-to-complex fft.
- int callIFFT (void *data_ptr, int ndim, int dimsize[3], int streamId=-1)

  Call complex-to-complex ifft.
- int callNormalizeFFT (void *data_ptr, int ndim, int dimsize[3], int streamId=-1)

  Normalize complex to complex ifft.
- int callR2CFFT (void *real_ptr, void *comp_ptr, int ndim, int dimsize[3], int streamId=-1)

  Call real to complex FFT.
- int callC2RFFT (void *real_ptr, void *comp_ptr, int ndim, int dimsize[3], int streamId=-1)

  Call complex to real iFFT.
- int callNormalizeC2RFFT (void *real_ptr, int ndim, int dimsize[3], int streamId=-1)

  Normalize compelx to real ifft.

Additional Inherited Members

4.21.1 Detailed Description

API to handel calls to DKSFFT.

Using DKSFFT interface executes FFT on GPUs, CPUs and MICs using cuFFT, clFFT or MKL libraries.

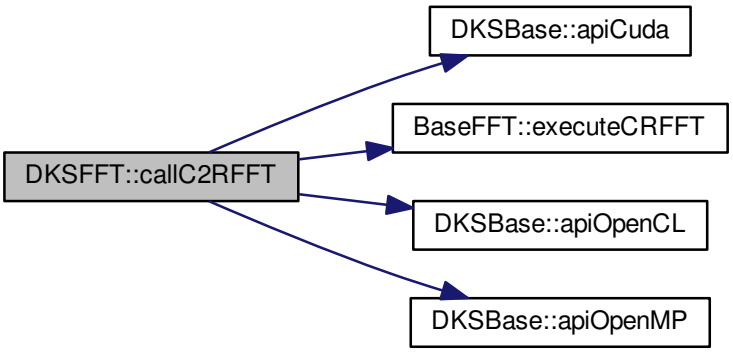
4.21.2 Member Function Documentation

4.21.2.1 int DKSFFT::callC2RFFT (void * *real_ptr,* void * *comp_ptr,* int *ndim,* int *dimsize[3],* int *streamId* = -1)

Call complex to real iFFT.

Executes out of place complex to real ifft, real_ptr points to real data, comp_pt - points to complex data, ndim - dimension of data, dimsize size of each dimension. real_ptr size should be dimsize[0]∗dimsize[1]∗disize[2], comp-_ptr size should be atleast (dimsize[0]/2+1)∗dimsize[1]∗dimsize[2] TODO: opencl and mic implementations.

Here is the call graph for this function:

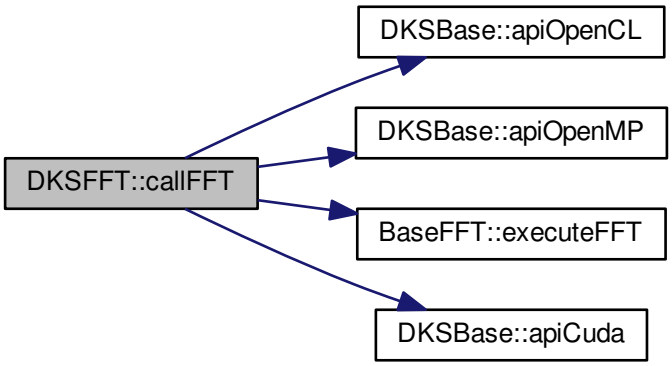


4.21.2.2 int DKSFFT::callFFT (void * *data_ptr,* int *ndim,* int *dimsize[3],* int *streamId* = -1)

Call complex-to-complex fft.

Executes in place complex to compelx fft on the device on data pointed by data_ptr. stream id can be specified to use other streams than default. TODO: mic implementation

Here is the call graph for this function:

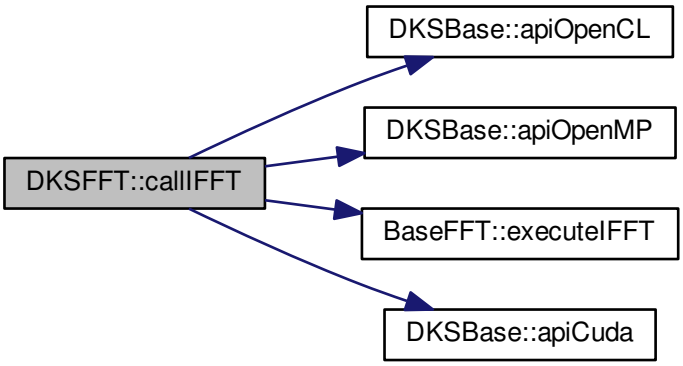


4.21.2.3 int DKSFFT::callIFFT (void * *data_ptr,* int *ndim,* int *dimsize[3],* int *streamId* = -1)

Call complex-to-complex ifft.

Executes in place complex to compelx ifft on the device on data pointed by data_ptr. stream id can be specified to use other streams than default. TODO: mic implementation.

Here is the call graph for this function:

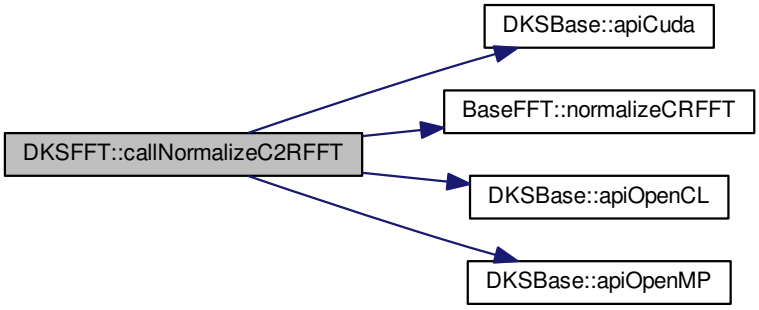


4.21.2.4 int DKSFFT::callNormalizeC2RFFT (void * *real_ptr,* int *ndim,* int *dimsize[3],* int *streamId* = -1)

Normalize compelx to real ifft.

Cuda, mic and OpenCL implementations return ifft unscaled, this function divides each element by fft size. TODO: opencl and mic implementations.

Here is the call graph for this function:

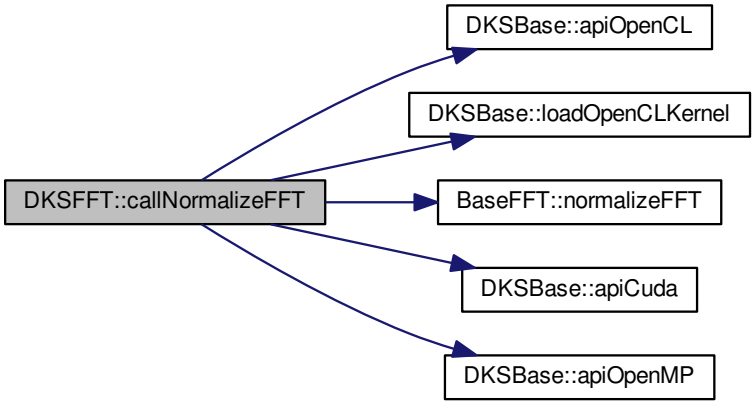


4.21.2.5 int DKSFFT::callNormalizeFFT (void * *data_ptr,* int *ndim,* int *dimsize[3],* int *streamId* = -1)

Normalize complex to complex ifft.

Cuda, mic and OpenCL implementations return ifft unscaled, this function divides each element by fft size TODO: mic implementation.

Here is the call graph for this function:

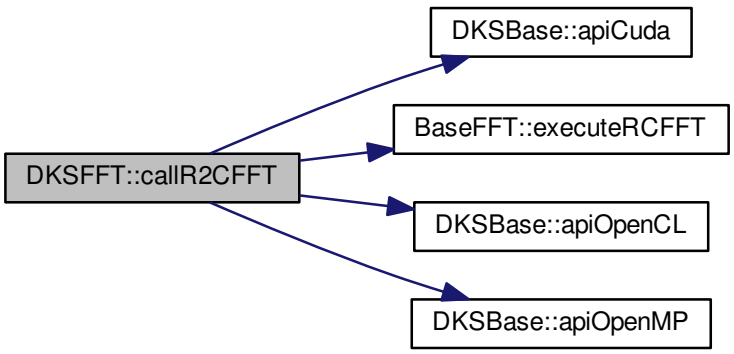


4.21.2.6 int DKSFFT::callR2CFFT (void * *real_ptr,* void * *comp_ptr,* int *ndim,* int *dimsize[3],* int *streamId* = -1)

Call real to complex FFT.

Executes out of place real to complex fft, real_ptr points to real data, comp_pt - points to complex data, ndim - dimension of data, dimsize size of each dimension. real_ptr size should be dimsize[0]∗dimsize[1]∗disize[2], comp-_ptr size should be atleast (dimsize[0]/2+1)∗dimsize[1]∗dimsize[2] TODO: opencl and mic implementations

Here is the call graph for this function:

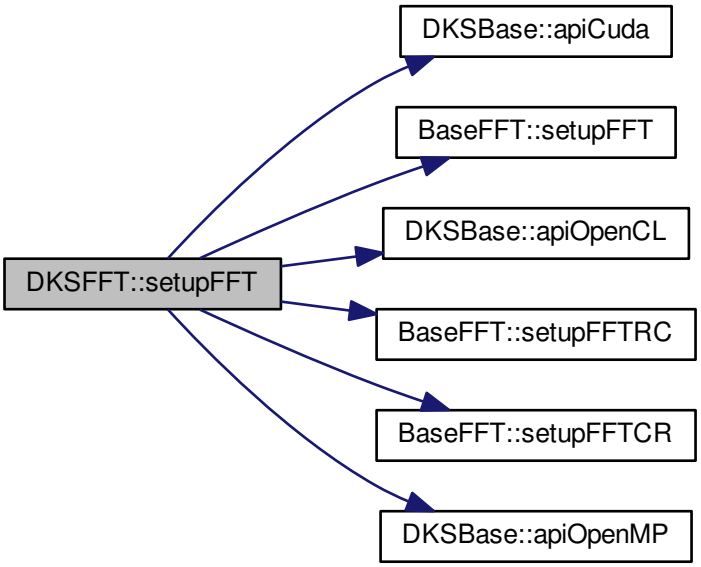


4.21.2.7 int DKSFFT::setupFFT (int *ndim*, int *N[3]*)

Setup FFT function.

Initializes parameters for fft executuin. If ndim $>$ 0 initializes handles for fft calls. If ffts of various sizes are needed setupFFT should be called with ndim 0, in this case each fft will do its own setup according to fft size and dimensions. TODO: opencl and mic implementations

Here is the call graph for this function:



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

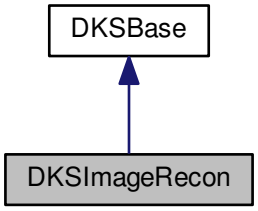
- src/DKSFFT.h
- src/DKSFFT.cpp

4.22 DKSImageRecon Class Reference

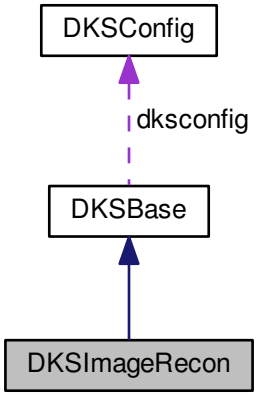
API to handle PET image reconstruction calls.

`#include <DKSImageReconstruction.h>`

Inheritance diagram for DKSImageRecon:



Collaboration diagram for DKSImageRecon:



Public Member Functions

- int callCalculateSource (void *image_space, void *image_position, void *source_position, void *avg, void *std, float diameter, int total_voxels, int total_sources, int start=0)

  Image reconstruction analaysis calculate source.
- int callCalculateBackground (void *image_space, void *image_position, void *source_position, void *avg, void *std, float diameter, int total_voxels, int total_sources, int start=0)

  Image reconstruction analaysis calculate source.
- int callCalculateSources (void *image_space, void *image_position, void *source_position, void *avg, void *std, void *diameter, int total_voxels, int total_sources, int start=0)

  Image reconstruction analaysis calculate source.
- int callCalculateBackgrounds (void *image_space, void *image_position, void *source_position, void *avg, void *std, void *diameter, int total_voxels, int total_sources, int start=0)

  Image reconstruction analaysis calculate source.
- int callGenerateNormalization (void *recon, void *image_position, void *det_position, int total_det)

  Image reconstruction - generate normalization.

- int callForwardProjection (void *correction, void *recon, void *list_data, void *det_position, void *image_-position, int num_events)

  Image reconstruction - forward correction.
- int callBackwardProjection (void *correction, void *recon_corrector, void *list_data, void *det_position, void *image_position, int num_events, int num_voxels)

  Image reconstruction - backward projection.
- int setDimensions (int voxel_x, int voxel_y, int voxel_z, float voxel_size)

  Set the voxel dimensins on device.
- int setEdge (float x_edge, float y_edge, float z_edge)

  Set the image edge.
- int setEdge1 (float x_edge1, float y_edge1, float z_edge1, float z_edge2)

  Set the image edge1.
- int setMinCrystalInRing (float min_CrystalDist_InOneRing, float min_CrystalDist_InOneRing1)

  Set the minimum crystal in one ring values.
- int setParams (float matrix_distance_factor, float phantom_diameter, float atten_per_mm, float ring_diameter)

  Set all other required parameters for reconstruction.

Additional Inherited Members

4.22.1 Detailed Description

API to handle PET image reconstruction calls.

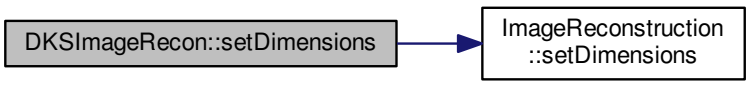
4.22.2 Member Function Documentation

4.22.2.1 int DKSImageRecon::setDimensions (int *voxel_x,* int *voxel_y,* int *voxel_z,* float *voxel_size*)

Set the voxel dimensins on device.

Values are stored in GPU memory and used in forward and backward projection calculations. Call set function once to transfer the values from host side to GPU. If value changes on the host side set functions needs to be called again to update GPU values.

Here is the call graph for this function:

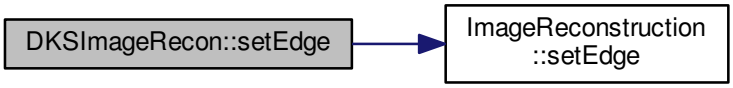


4.22.2.2 int DKSImageRecon::setEdge (float *x_edge,* float *y_edge,* float *z_edge*)

Set the image edge.

Values are stored in GPU memory and used in forward and backward projection calculations. Call set function once to transfer the values from host side to GPU. If value changes on the host side set functions needs to be called again to update GPU values.

Here is the call graph for this function:

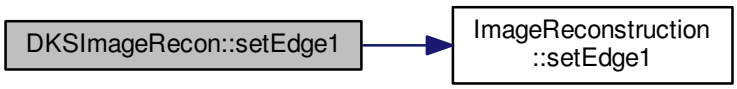


4.22.2.3 int DKSImageRecon::setEdge1 (float *x_edge1,* float *y_edge1,* float *z_edge1,* float *z_edge2*)

Set the image edge1.

Values are stored in GPU memory and used in forward and backward projection calculations. Call set function once to transfer the values from host side to GPU. If value changes on the host side set functions needs to be called again to update GPU values.

Here is the call graph for this function:

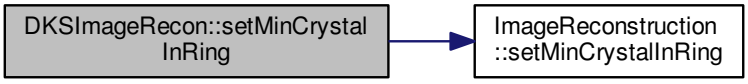


4.22.2.4 int DKSImageRecon::setMinCrystalInRing (float *min_CrystalDist_InOneRing,* float *min_CrystalDist_InOneRing1*)

Set the minimum crystal in one ring values.

Values are stored in GPU memory and used in forward and backward projection calculations. Call set function once to transfer the values from host side to GPU. If value changes on the host side set functions needs to be called again to update GPU values.

Here is the call graph for this function:

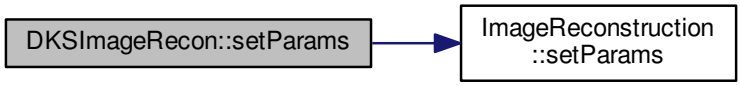


4.22.2.5 int DKSImageRecon::setParams (float *matrix_distance_factor,* float *phantom_diameter,* float *atten_per_mm,* float *ring_diameter*)

Set all other required parameters for reconstruction.

Values are stored in GPU memory and used in forward and backward projection calculations. Call set function once to transfer the values from host side to GPU. If value changes on the host side set functions needs to be called again to update GPU values.

Here is the call graph for this function:



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

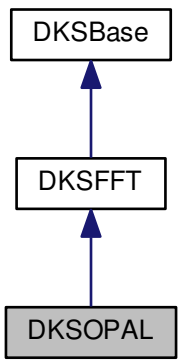
- src/DKSImageReconstruction.h
- src/DKSImageReconstruction.cpp

4.23 DKSOPAL Class Reference

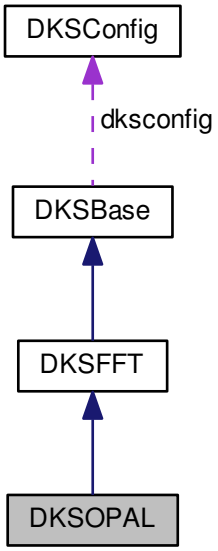
API to handle OPAL calls to DKS library.

`#include <DKSOPAL.h>`

Inheritance diagram for DKSOPAL:



Collaboration diagram for DKSOPAL:



Public Member Functions

- **DKSOPAL** (const char *api_name, const char *device_name)
- int **initDevice** ()
- int callGreensIntegral (void *tmp_ptr, int I, int J, int K, int NI, int NJ, double hz_m0, double hz_m1, double hz_m2, int streamId=-1)

  Integrated greens function from OPAL FFTPoissonsolver.cpp put on device.
- int callGreensIntegration (void *mem_ptr, void *tmp_ptr, int I, int J, int K, int streamId=-1)

  Integrated greens function from OPAL FFTPoissonsolver.cpp put on device.
- int callMirrorRhoField (void *mem_ptr, int I, int J, int K, int streamId=-1)

  Integrated greens function from OPAL FFTPoissonsolver.cpp put on device.
- int callMultiplyComplexFields (void *mem_ptr1, void *mem_ptr2, int size, int streamId=-1)

  Element by element multiplication.
- int callCollimatorPhysics (void *mem_ptr, void *par_ptr, int numparticles, int numparams, int &numaddback, int &numdead, bool enableRutherfordScattering=true)

  Monte carlo code for the degrader from OPAL classic/5.0/src/Solvers/CollimatorPhysics.cpp on device.
- int callCollimatorPhysics2 (void *mem_ptr, void *par_ptr, int numparticles, bool enableRutherfordScattering=true)

  Monte carlo code for the degrader from OPAL classic/5.0/src/Solvers/CollimatorPhysics.cpp on device.
- int callCollimatorPhysicsSoA (void *label_ptr, void *localID_ptr, void *rx_ptr, void *ry_ptr, void *rz_ptr, void *px_ptr, void *py_ptr, void *pz_ptr, void *par_ptr, int numparticles)

  Monte carlo code for the degrader from OPAL classic/5.0/src/Solvers/CollimatorPhysics.cpp on device.
- int callCollimatorPhysicsSort (void *mem_ptr, int numparticles, int &numaddback)

  Monte carlo code for the degrader from OPAL classic/5.0/src/Solvers/CollimatorPhysics.cpp on device.
- int callCollimatorPhysicsSortSoA (void *label_ptr, void *localID_ptr, void *rx_ptr, void *ry_ptr, void *rz_ptr, void *px_ptr, void *py_ptr, void *pz_ptr, void *par_ptr, int numparticles, int &numaddback)

  Monte carlo code for the degrader from OPAL classic/5.0/src/Solvers/CollimatorPhysics.cpp on device.

- int callParallelTTrackerPush (void ∗r_ptr, void ∗p_ptr, int npart, void ∗dt_ptr, double dt, double c, bool usedt=false, int streamId=-1)

  Integration code from ParallelTTracker from OPAL.

- int callParallelTTrackerPushTransform (void ∗x_ptr, void ∗p_ptr, void ∗lastSec_ptr, void ∗orient_ptr, int npart, int nsec, void ∗dt_ptr, double dt, double c, bool usedt=false, int streamId=-1)

  Integration code from ParallelTTracker from OPAL.

- int callParallelTTrackerPush (void ∗r_ptr, void ∗p_ptr, void ∗dt_ptr, int npart, double c, int streamId=-1)

  Integration code from ParallelTTracker from OPAL.

- int callParallelTTrackerKick (void ∗r_ptr, void ∗p_ptr, void ∗ef_ptr, void ∗bf_ptr, void ∗dt_ptr, double charge, double mass, int npart, double c, int streamId=-1)

  Integration code from ParallelTTracker from OPAL.

Additional Inherited Members

4.23.1 Detailed Description

API to handle OPAL calls to DKS library.

Gives access to DKSCollimatorPhysics, GreensFunction and DKSFFT, as well as all the DKSBase functions.

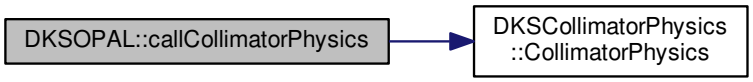
4.23.2 Member Function Documentation

4.23.2.1 int DKSOPAL::callCollimatorPhysics (void ∗ *mem_ptr,* void ∗ *par_ptr,* int *numparticles,* int *numparams,* int & *numaddback,* int & *numdead,* bool *enableRutherfordScattering* = `true`)

Monte carlo code for the degrader from OPAL classic/5.0/src/Solvers/CollimatorPhysics.cpp on device.

For specifics check OPAL docs and CudaCollimatorPhysics class documentation. TODO: opencl and mic implementations.

Here is the call graph for this function:



4.23.2.2 int DKSOPAL::callCollimatorPhysics2 (void ∗ *mem_ptr,* void ∗ *par_ptr,* int *numparticles,* bool *enableRutherfordScattering* = `true`)

Monte carlo code for the degrader from OPAL classic/5.0/src/Solvers/CollimatorPhysics.cpp on device.

For specifics check OPAL docs and CudaCollimatorPhysics class documentation. TODO: opencl and mic implementations.

Here is the call graph for this function:

4.23.2.3 int DKSOPAL::callCollimatorPhysicsSoA (void * *label_ptr*, void * *localID_ptr*, void * *rx_ptr*, void * *ry_ptr*, void * *rz_ptr*, void * *px_ptr*, void * *py_ptr*, void * *pz_ptr*, void * *par_ptr*, int *numparticles*)

Monte carlo code for the degrader from OPAL classic/5.0/src/Solvers/CollimatorPhysics.cpp on device.

For specifics check OPAL docs and CudaCollimatorPhysics class documentation. Test function for the MIC to test SoA layout vs AoS layout used in previous versions

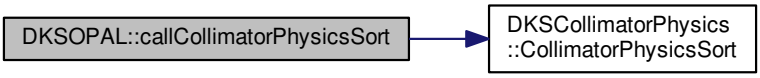
Here is the call graph for this function:

4.23.2.4 int DKSOPAL::callCollimatorPhysicsSort (void * *mem_ptr*, int *numparticles*, int & *numaddback*)

Monte carlo code for the degrader from OPAL classic/5.0/src/Solvers/CollimatorPhysics.cpp on device.

For specifics check OPAL docs and CudaCollimatorPhysics class documentation. TODO: opencl and mic implementations.

Here is the call graph for this function:

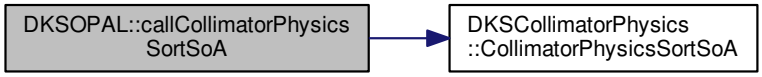


4.23.2.5 int DKSOPAL::callCollimatorPhysicsSortSoA (void * *label_ptr*, void * *localID_ptr*, void * *rx_ptr*, void * *ry_ptr*, void * *rz_ptr*, void * *px_ptr*, void * *py_ptr*, void * *pz_ptr*, void * *par_ptr*, int *numparticles*, int & *numaddback*)

Monte carlo code for the degrader from OPAL classic/5.0/src/Solvers/CollimatorPhysics.cpp on device.

For specifics check OPAL docs and CudaCollimatorPhysics class documentation. TODO: opencl and mic implementations.

Here is the call graph for this function:



4.23.2.6 int DKSOPAL::callGreensIntegral (void * *tmp_ptr*, int *I*, int *J*, int *K*, int *NI*, int *NJ*, double *hz_m0*, double *hz_m1*, double *hz_m2*, int *streamId* = -1)

Integrated greens function from OPAL FFTPoissonsolver.cpp put on device.

For specifics check OPAL docs. TODO: opencl and mic implementations.

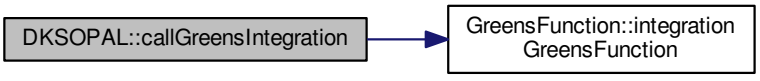
Here is the call graph for this function:

4.23.2.7 int DKSOPAL::callGreensIntegration (void * *mem_ptr*, void * *tmp_ptr*, int *I*, int *J*, int *K*, int *streamId* = -1)

Integrated greens function from OPAL FFTPoissonsolver.cpp put on device.

For specifics check OPAL docs. TODO: opencl and mic implementations.

Here is the call graph for this function:

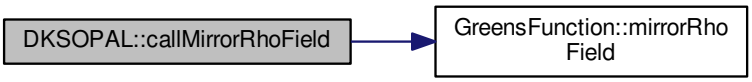


4.23.2.8 int DKSOPAL::callMirrorRhoField (void * *mem_ptr*, int *I*, int *J*, int *K*, int *streamId* = -1)

Integrated greens function from OPAL FFTPoissonsolver.cpp put on device.

For specifics check OPAL docs. TODO: opencl and mic implementations.

Here is the call graph for this function:

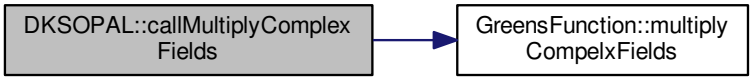


4.23.2.9 int DKSOPAL::callMultiplyComplexFields (void * *mem_ptr1,* void * *mem_ptr2,* int *size,* int *streamId* = -1)

Element by element multiplication.

Multiplies each element of mem_ptr1 with corresponding element of mem_ptr2, size specifies the number of elements in mem_ptr1 and mem_ptr2 to use. Results are put in mem_ptr1. TODO: opencl and mic implementations.

Here is the call graph for this function:

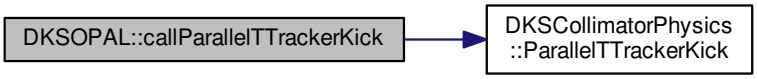


4.23.2.10 int DKSOPAL::callParallelTTrackerKick (void * *r_ptr,* void * *p_ptr,* void * *ef_ptr,* void * *bf_ptr,* void * *dt_ptr,* double *charge,* double *mass,* int *npart,* double *c,* int *streamId* = -1)

Integration code from ParallelTTracker from OPAL.

For specifics check OPAL docs and CudaCollimatorPhysics class docs

Here is the call graph for this function:

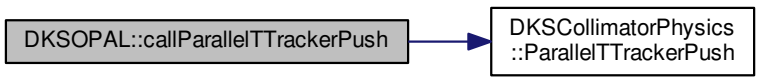


4.23.2.11 int DKSOPAL::callParallelTTrackerPush (void * *r_ptr,* void * *p_ptr,* int *npart,* void * *dt_ptr,* double *dt,* double *c,* bool *usedt* = false, int *streamId* = -1)

Integration code from ParallelTTracker from OPAL.

For specifics check OPAL docs and CudaCollimatorPhysics class docs

Here is the call graph for this function:

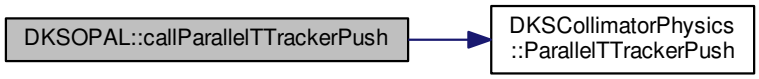


4.23.2.12 int DKSOPAL::callParallelTTrackerPush (void * *r_ptr*, void * *p_ptr*, void * *dt_ptr*, int *npart*, double *c*, int *streamId* = -1)

Integration code from ParallelTTracker from OPAL.

For specifics check OPAL docs and CudaCollimatorPhysics class docs

Here is the call graph for this function:

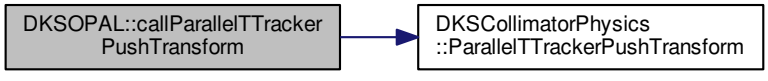


4.23.2.13 int DKSOPAL::callParallelTTrackerPushTransform (void * *x_ptr*, void * *p_ptr*, void * *lastSec_ptr*, void * *orient_ptr*, int *npart*, int *nsec*, void * *dt_ptr*, double *dt*, double *c*, bool *usedt* = false, int *streamId* = -1)

Integration code from ParallelTTracker from OPAL.

For specifics check OPAL docs and CudaCollimatorPhysics class docs

Here is the call graph for this function:



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

- src/DKSOPAL.h
- src/DKSOPAL.cpp

4.24 DKSSearchStates Class Reference

Used by auto-tuning search algorithms to move between parameter configurations.

`#include <DKSSearchStates.h>`

Public Member Functions

- DKSSearchStates (Parameters params)

  Constructor alwats takes params array as variable.
- void setCurrentState (Parameters current_state)

  Set current state using parameter vector.
- void setCurrentState (States current_state)

  set current state using the state vector
- void initCurrentState ()

  init random current state
- States getCurrentState ()

  get current state
- States getNextNeighbour ()

  get next neighbour state.
- States getRandomNeighbour ()

  get random neighbour state
- void getNeighbours (int dist=1)

  calculate all the neighbour states
- bool nextNeighbourExists ()

  Chech if there are more neighbours to evaluate Return true if more neighbors exist, false if we are at the last neighbour.
- void moveToNeighbour ()

  move to next neighbour.
- void saveCurrentState (double current_time)

  Save the current state and the evaluation time of the current state.
- void printCurrentState (double time=0.0)

  Print current state.
- void printNeighbour (double time=0.0)

  Print current neighbour info.
- void printInfo ()

  Print info.
- void printBest ()

  Print the best saved state.

4.24.1 Detailed Description

Used by auto-tuning search algorithms to move between parameter configurations.

Allows to move from one parameter stat to another, get neighboring states, move to neighboring states and save state information. Print functions are available for debugging purposes, to follow how algorithm muves between sates.

4.24.2 Constructor & Destructor Documentation

4.24.2.1 DKSSearchStates::DKSSearchStates (Parameters *params*)

Constructor alwats takes params array as variable.

set the current state so that number of parameters and parameter bounds are known

Params array is needed to know how many params will be searched and what are thou bounds of each parameter.

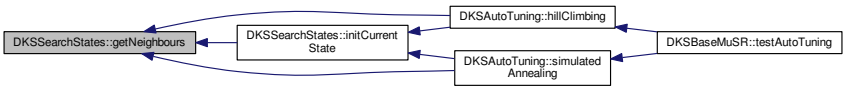
4.24.3 Member Function Documentation

4.24.3.1 void DKSSearchStates::getNeighbours (int *dist* = `1`)

calculate all the neighbour states

Get all the possible neighbours of the current state.

Here is the caller graph for this function:



4.24.3.2 States DKSSearchStates::getNextNeighbour ( )

get next neighbour state.

if there are no next neighbore stay at the curretn neighbour

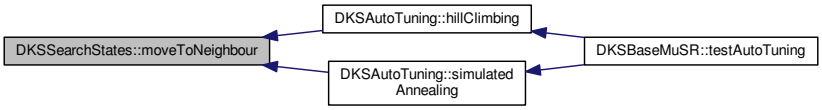
Here is the caller graph for this function:

4.24.3.3 void DKSSearchStates::moveToNeighbour ( )

move to next neighbour.

set the current state as the next neighbour, calculate the neighbours of the new current state.

Here is the caller graph for this function:

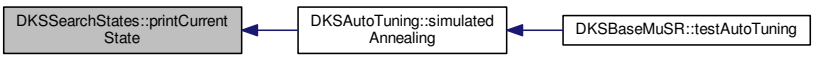


4.24.3.4 void DKSSearchStates::printCurrentState (double *time* = 0.0)

Print current state.

cout the current state. Mostly used for debuging purposes

Here is the caller graph for this function:



4.24.3.5 void DKSSearchStates::printInfo ( )

Print info.

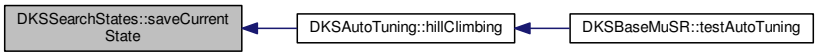
Print the whole info about the search: current state, current neighbour, total neighbors

4.24.3.6 void DKSSearchStates::saveCurrentState (double *current_time*)

Save the current state and the evaluation time of the current state.

If evaluation time of the current state is better than the evaluation time of the best state, save the current state as best.

Here is the caller graph for this function:



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

- src/AutoTuning/DKSSearchStates.h
- src/AutoTuning/DKSSearchStates.cpp

4.25 DKSStream Class Reference

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

- src/DKSStream.h

4.26 DKSTimer Class Reference

Custom timer class.

`#include <DKSTimer.h>`

Public Member Functions

- DKSTimer ()

  Init DKSTimer by seting timer to zero.
- void init (std::string n)

  Init the timer.
- void start ()

  Start the timer.
- void stop ()

  Stop the timer.
- void reset ()

  Reset timervalue to zero.
- double gettime ()

  Return elapsed time in seconds.
- void print ()

  Print timer.

4.26.1 Detailed Description

Custom timer class.

Allows to insert timers in the code to get function exectution times.

4.26.2 Constructor & Destructor Documentation

4.26.2.1 DKSTimer::DKSTimer ( )

Init DKSTimer by seting timer to zero.

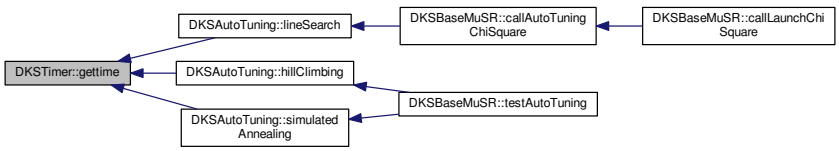
4.26.3 Member Function Documentation

4.26.3.1 double DKSTimer::gettime ( )

Return elapsed time in seconds.

Return the value of timervalue

Here is the caller graph for this function:

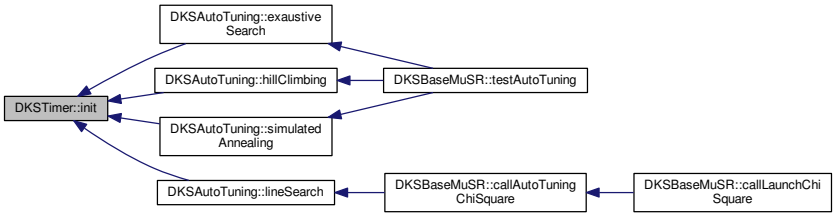


4.26.3.2 void DKSTimer::init (std::string *n*)

Init the timer.

Set the name for timer and clear all values

Here is the caller graph for this function:



4.26.3.3 void DKSTimer::print ( )

Print timer.

Print the elapsed time of the timer

4.26.3.4 void DKSTimer::reset ( )

Reset timervalue to zero.

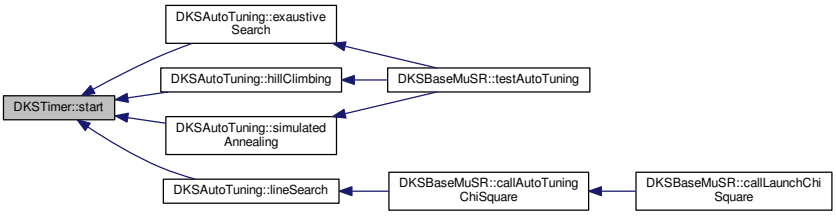
Set timervalue, timeStart and timeEnd to zero

4.26.3.5 void DKSTimer::start ( )

Start the timer.

Get the curret time with gettimeofday and save in timeStart

Here is the caller graph for this function:

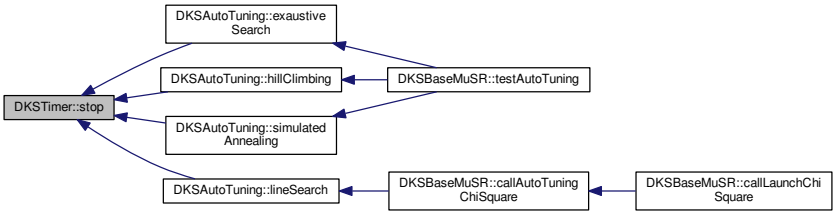


4.26.3.6 void DKSTimer::stop ( )

Stop the timer.

Get the curretn time with gettimeofday and save in timeEnd Calculate elapsed time by timeEnd - timeStart and add to timervalue

Here is the caller graph for this function:



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

- src/Utility/DKSTimer.h
- src/Utility/DKSTimer.cpp

4.27 Double3 Struct Reference

Double3 structure for use in OpenCL code.

`#include <OpenCLCollimatorPhysics.h>`

Public Attributes

- double **x**
- double **y**
- double **z**

4.27.1 Detailed Description

Double3 structure for use in OpenCL code.

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

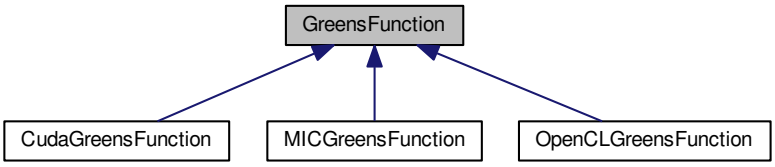
- src/OpenCL/OpenCLCollimatorPhysics.h

4.28 GreensFunction Class Reference

Interface to implement Greens function calculations for OPAL.

`#include <GreensFunction.h>`

Inheritance diagram for GreensFunction:



Public Member Functions

- virtual int greensIntegral (void *tmpgreen, int I, int J, int K, int NI, int NJ, double hr_m0, double hr_m1, double hr_m2, int streamId=-1)=0

  calc greens integral, as defined in OPAL.
- virtual int integrationGreensFunction (void *rho2_m, void *tmpgreen, int I, int J, int K, int streamId=-1)=0

  integration if rho2_m, see OPAL for more details.
- virtual int mirrorRhoField (void *rho2_m, int I, int J, int K, int streamId=-1)=0

  mirror rho2_m field.
- virtual int multiplyCompelxFields (void *ptr1, void *ptr2, int size, int streamId=-1)=0

  multiply two complex fields from device memory.

4.28.1 Detailed Description

Interface to implement Greens function calculations for OPAL.

4.28.2 Member Function Documentation

4.28.2.1 virtual int GreensFunction::greensIntegral (void * *tmpgreen*, int *I*, int *J*, int *K*, int *NI*, int *NJ*, double *hr_m0*, double *hr_m1*, double *hr_m2*, int *streamId* = -1) [pure virtual]

calc greens integral, as defined in OPAL.

Implemented in CudaGreensFunction, OpenCLGreensFunction, and MICGreensFunction.

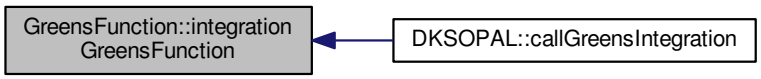
Here is the caller graph for this function:

4.28.2.2 virtual int GreensFunction::integrationGreensFunction (void * *rho2_m*, void * *tmpgreen*, int *I*, int *J*, int *K*, int *streamId* = -1) [pure virtual]

integration if rho2_m, see OPAL for more details.

Implemented in CudaGreensFunction, OpenCLGreensFunction, and MICGreensFunction.

Here is the caller graph for this function:

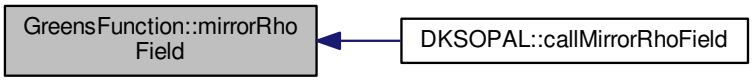


4.28.2.3 virtual int GreensFunction::mirrorRhoField (void * *rho2_m*, int *l*, int *J*, int *K*, int *streamId* = -1) [pure virtual]

mirror rho2_m field.

Implemented in CudaGreensFunction, OpenCLGreensFunction, and MICGreensFunction.

Here is the caller graph for this function:

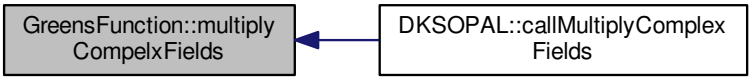


4.28.2.4 virtual int GreensFunction::multiplyCompelxFields (void * *ptr1*, void * *ptr2*, int *size*, int *streamId* = -1) [pure virtual]

multiply two complex fields from device memory.

Implemented in CudaGreensFunction, OpenCLGreensFunction, and MICGreensFunction.

Here is the caller graph for this function:



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

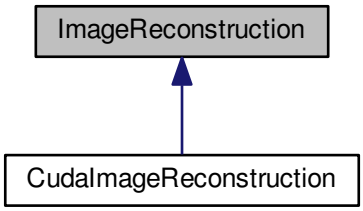
- src/Algorithms/GreensFunction.h

4.29 ImageReconstruction Class Reference

Interface to implement PET image reconstruction.

`#include <ImageReconstruction.h>`

Inheritance diagram for ImageReconstruction:



Public Member Functions

- virtual int calculateSource (void ∗image_space, void ∗image_position, void ∗source_position, void ∗avg, void ∗std, float diameter, int total_voxels, int total_sources, int start=0)=0

  Caluclate source.
- virtual int calculateBackground (void ∗image_space, void ∗image_position, void ∗source_position, void ∗avg, void ∗std, float diameter, int total_voxels, int total_sources, int start=0)=0

  Calculate background.
- virtual int calculateSources (void ∗image_space, void ∗image_position, void ∗source_position, void ∗avg, void ∗std, void ∗diameter, int total_voxels, int total_sources, int start=0)=0

  Caluclate source using differente sources.
- virtual int calculateBackgrounds (void ∗image_space, void ∗image_position, void ∗source_position, void ∗avg, void ∗std, void ∗diameter, int total_voxels, int total_sources, int start=0)=0

  Places two sphere at each voxel position, calculates the avg value and std value of pixels.
- virtual int generateNormalization (void ∗recon, void ∗image_position, void ∗det_position, int total_det)=0

  Generate normalization.
- virtual int forwardProjection (void ∗correction, void ∗recon, void ∗list_data, void ∗det_position, void ∗image_-position, int num_events)=0

  Calculate forward projection.
- virtual int backwardProjection (void ∗correction, void ∗recon_corrector, void ∗list_data, void ∗det_position, void ∗image_position, int num_events, int num_voxels)=0

  Calculate backward projection.
- virtual int setDimensions (int voxel_x, int voxel_y, int voxel_z, float voxel_size)=0

  Set the voxel dimensins on device.
- virtual int setEdge (float x_edge, float y_edge, float z_edge)=0

  Set the image edge variables on the device.
- virtual int setEdge1 (float x_edge1, float y_edge1, float z_edge1, float z_edge2)=0

  Set the image edge1 on the device.
- virtual int setMinCrystalInRing (float min_CrystalDist_InOneRing, float min_CrystalDist_InOneRing1)=0

  Set the minimum crystan in one ring values on the device.
- virtual int setParams (float matrix_distance_factor, float phantom_diameter, float atten_per_mm, float ring_-diameter)=0

  Set all other required parameters for reconstruction.

Protected Attributes

- void * **m_event_branch**

4.29.1 Detailed Description

Interface to implement PET image reconstruction.

4.29.2 Member Function Documentation

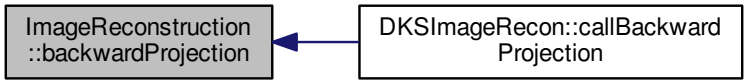
4.29.2.1 virtual int ImageReconstruction::backwardProjection (void * *correction*, void * *recon_corrector*, void * *list_data*, void * *det_position*, void * *image_position*, int *num_events*, int *num_voxels*) [pure virtual]

Calculate backward projection.

For image reconstruction calculates backward projections. see recon.cpp for details

Implemented in CudaImageReconstruction.

Here is the caller graph for this function:



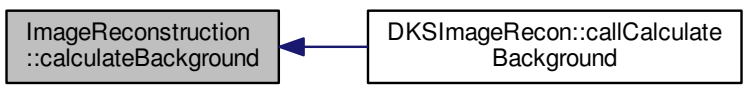
4.29.2.2 virtual int ImageReconstruction::calculateBackground (void * *image_space*, void * *image_position*, void * *source_position*, void * *avg*, void * *std*, float *diameter*, int *total_voxels*, int *total_sources*, int *start* = 0) [pure virtual]

Calculate background.

Places two sphere at each voxel position, calculates the avg value and std value of pixels that are inside the larger sphere, but are outside of the smaller sphere. The diameter of the smaller speher is given by parameter diameter, diameter of the larger sphere is 2*diameter.

Implemented in CudaImageReconstruction.

Here is the caller graph for this function:



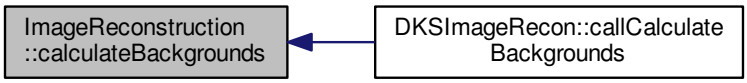
4.29.2.3 virtual int ImageReconstruction::calculateBackgrounds (void * *image_space*, void * *image_position*, void * *source_position*, void * *avg*, void * *std*, void * *diameter*, int *total_voxels*, int *total_sources*, int *start* = 0) [pure virtual]

Places two sphere at each voxel position, calculates the avg value and std value of pixels.

that are inside the larger sphere, but are outside of the smaller sphere. The diameter of the smaller sphere is given by *diameter array, diameter of the larger sphere is 2*diameter of the smaller sphere.

Implemented in CudaImageReconstruction.

Here is the caller graph for this function:



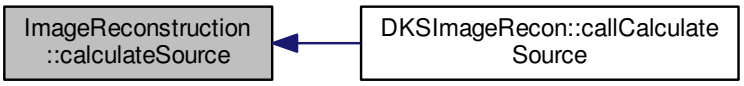
4.29.2.4 virtual int ImageReconstruction::calculateSource (void * *image_space*, void * *image_position*, void * *source_position*, void * *avg*, void * *std*, float *diameter*, int *total_voxels*, int *total_sources*, int *start* = 0) [pure virtual]

Caluclate source.

Places a sphere at each voxel position and calculate the avg value and std value of pixels that are inside this sphere. All the sphere used have the same diameter.

Implemented in CudaImageReconstruction.

Here is the caller graph for this function:



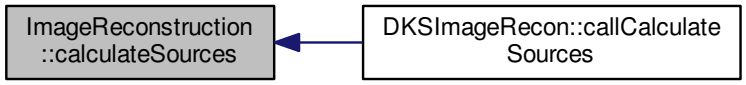
4.29.2.5 virtual int ImageReconstruction::calculateSources (void * *image_space*, void * *image_position*, void * *source_position*, void * *avg*, void * *std*, void * *diameter*, int *total_voxels*, int *total_sources*, int *start* = 0) [pure virtual]

Caluclate source using differente sources.

Places two sphere at each voxel position, calculates the avg value and std value of pixels that are inside the larger sphere, but are outside of the smaller sphere. The diameter of the each sphere is given by *diameter array.

Implemented in CudaImageReconstruction.

Here is the caller graph for this function:



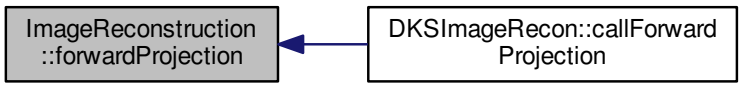
4.29.2.6 virtual int ImageReconstruction::forwardProjection (void * *correction*, void * *recon*, void * *list_data*, void * *det_position*, void * *image_position*, int *num_events*) [pure virtual]

Calculate forward projection.

For image reconstruction calculates forward projections. see recon.cpp for details

Implemented in CudaImageReconstruction.

Here is the caller graph for this function:



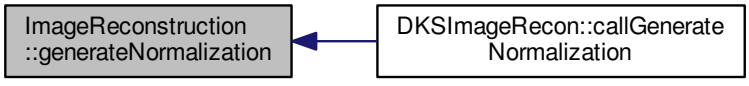
4.29.2.7 virtual int ImageReconstruction::generateNormalization (void * *recon*, void * *image_position*, void * *det_position*, int *total_det*) [pure virtual]

Generate normalization.

Goes trough detectors pairs and if detector pair crosses image launches seperate kernel that updates voxel values in the image on the slope between these two detectors.

Implemented in CudaImageReconstruction.

Here is the caller graph for this function:



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

- src/Algorithms/ImageReconstruction.h

4.30 ListEvent Struct Reference

Struct that holds pair of detectors that registered an envent.

`#include <ImageReconstruction.h>`

Public Attributes

- unsigned **detA**: 16
- unsigned **detB**: 16

4.30.1 Detailed Description

Struct that holds pair of detectors that registered an envent.

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

- src/Algorithms/ImageReconstruction.h

4.31 MICBase Class Reference

MIC Base class handles device setup and basic communication with the device.

`#include <MICBase.h>`

Public Member Functions

- MICBase ()
  constructor
- ~MICBase ()
  destructor
- int mic_createRandStreams (int size)
  Create MKL rand streams for each thread Return: success or error code.
- int mic_deleteRandStreams ()
  Delete MKL rand streams Return: succes or error code.
- int mic_createStream (int &streamId)
  Create a new signal for the mic.
- int & mic_getStream (int id)
  Info: get the signal from the vector.
- int mic_deleteStreams ()
  Info: delete streams.
- int mic_setDeviceId (int id)
  Info: set device id.
- int mic_getDevices ()
  Info: get mic devices.
- template<typename T >
  void * mic_allocateMemory (int size)
  Allocate memory on MIC device.

- template<typename T >
  int mic_writeData (void *data_ptr, const void *data, int size, int offset=0)

  Transfer data to device.
- template<typename T >
  int mic_writeDataAsync (void *data_ptr, const void *data, int size, int streamId=-1, int offset=0)

  Write data to device, non-blocking.
- template<typename T >
  int mic_readData (const void *data_ptr, void *result, int size, int offset=0)

  Read data from device Return: success or error code.
- template<typename T >
  int mic_readDataAsync (const void *data_ptr, void *result, int size, int streamId=-1, int offset=0)

  Read data from device waiting for signal Return: success or error code.
- int mic_syncDevice ()

  Wait till all the signals are complete Return siccess or error code.
- template<typename T >
  int mic_freeMemory (void *data_ptr, int size)

  Free memory on device Return: success or error code.
- template<typename T >
  void * mic_pushData (const void *data, int size)

  Allocate memory and write data to device Return: success or error code.
- template<typename T >
  int mic_pullData (void *data_ptr, void *result, int size)

  Read data and free memory on device Return: success or erro code.

Public Attributes

- void * **defaultRndStream**
- void * **testPtr**
- int **m_device_id**

Protected Attributes

- int **defaultRndSet**

4.31.1 Detailed Description

MIC Base class handles device setup and basic communication with the device.

Handles devicew setup, memory manegement and data transfers.

4.31.2 Member Function Documentation

4.31.2.1 template<typename T > void* MICBase::mic_allocateMemory (int *size*) [inline]

Allocate memory on MIC device.

Return: success or error code

4.31.2.2 int MICBase::mic_createStream (int & *streamId*)

Create a new signal for the mic.

Signals can be used for assynchronous data transfers. Return: success or error code

4.31.2.3 int MICBase::mic_deleteStreams ( )

Info: delete streams.

Return: success or error code

4.31.2.4 int MICBase::mic_getDevices ( )

Info: get mic devices.

Prints information about mic devices. Return: success or error code

4.31.2.5 int & MICBase::mic_getStream (int *id*)

Info: get the signal from the vector.

Return: mic signal

4.31.2.6 int MICBase::mic_setDeviceId (int *id*)

Info: set device id.

Return: success or error code

4.31.2.7 template<typename T > int MICBase::mic_writeData (void * *data_ptr,* const void * *data,* int *size,* int *offset* = 0) [inline]

Transfer data to device.

Return: success or error code

4.31.2.8 template<typename T > int MICBase::mic_writeDataAsync (void * *data_ptr,* const void * *data,* int *size,* int *streamId* = -1, int *offset* = 0) [inline]

Write data to device, non-blocking.

Return: success or error code

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

- src/MIC/MICBase.h
- src/MIC/MICBase.cpp

4.32 MICChiSquare Class Reference

Deprecated, OpenMP + offload to Xeon Phi implementation of ChiSquare for MIC devices.

`#include <MICChiSquare.h>`

Public Member Functions

- **MICChiSquare** (MICBase *base)
- int **mic_chi2** (double *O, double *E, double *result, int size)
- int **mic_Nt** (double *nt, double *p, int psize, int nsize, int jsize, double deltaT=1)
- int **mic_sum** (double *data, double *result, int size)

4.32.1 Detailed Description

Deprecated, OpenMP + offload to Xeon Phi implementation of ChiSquare for MIC devices.

Not complete and untested because of the poor performance of first MIC devices.

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

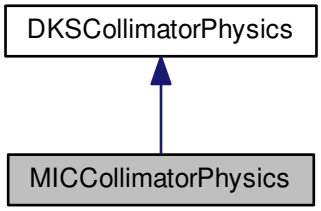
- src/MIC/MICChiSquare.h
- src/MIC/MICChiSquare.cpp

4.33 MICCollimatorPhysics Class Reference

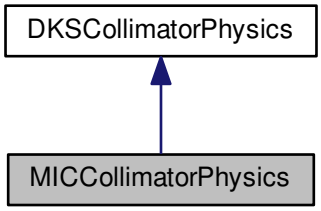
MICCollimatorPhysics class based on DKSCollimatorPhysics interface.

`#include <MICCollimatorPhysics.h>`

Inheritance diagram for MICCollimatorPhysics:



Collaboration diagram for MICCollimatorPhysics:



Public Member Functions

- **MICCollimatorPhysics** (MICBase *base)
- int CollimatorPhysics (void *mem_ptr, void *par_ptr, int numparticles, bool enableRutherforScattering=true)

  Execute collimator physics kernel.

- int CollimatorPhysicsSoA (void *label_ptr, void *localID_ptr, void *rx_ptr, void *ry_ptr, void *rz_ptr, void *px_ptr, void *py_ptr, void *pz_ptr, void *par_ptr, int numparticles)

  Special calse CollimatorPhysics kernel that uses SoA instead of AoS.
- int CollimatorPhysicsSort (void *mem_ptr, int numparticles, int &numaddback)

  Sort particle array on GPU.
- int CollimatorPhysicsSortSoA (void *label_ptr, void *localID_ptr, void *rx_ptr, void *ry_ptr, void *rz_ptr, void *px_ptr, void *py_ptr, void *pz_ptr, void *par_ptr, int numparticles, int &numaddback)

  Special calse CollimatorPhysicsSort kernel that uses SoA instead of AoS.
- int ParallelTTrackerPush (void *r_ptr, void *p_ptr, int npart, void *dt_ptr, double dt, double c, bool usedt=false, int streamId=-1)

  BorisPusher push function for integration from OPAL.
- int ParallelTTrackerPushTransform (void *x_ptr, void *p_ptr, void *lastSec_ptr, void *orient_ptr, int npart, int nsec, void *dt_ptr, double dt, double c, bool usedt=false, int streamId=-1)

  BorisPusher push function with transformto function form OPAL.

Additional Inherited Members

4.33.1 Detailed Description

MICCollimatorPhysics class based on DKSCollimatorPhysics interface.

Implementes OPALs collimator physics class for particle matter interactions using OpenMP and offload mode targetomg Intel Xeon Phi processors. For detailed documentation on CollimatorPhysics functions see OPAL documentation.

4.33.2 Member Function Documentation

4.33.2.1 int MICCollimatorPhysics::CollimatorPhysicsSoA (void * *label_ptr,* void * *localID_ptr,* void * *rx_ptr,* void * *ry_ptr,* void * *rz_ptr,* void * *px_ptr,* void * *py_ptr,* void * *pz_ptr,* void * *par_ptr,* int *numparticles*) [virtual]

Special calse CollimatorPhysics kernel that uses SoA instead of AoS.

Used only on the MIC side, was not implemented on the GPU.

Implements DKSCollimatorPhysics.

4.33.2.2 int MICCollimatorPhysics::CollimatorPhysicsSort (void * *mem_ptr,* int *numparticles,* int & *numaddback*) [virtual]

Sort particle array on GPU.

Count particles that are dead (label -1) or leaving material (label -2) and sort particle array so these particles are at the end of array

Implements DKSCollimatorPhysics.

4.33.2.3 int MICCollimatorPhysics::CollimatorPhysicsSortSoA (void * *label_ptr,* void * *localID_ptr,* void * *rx_ptr,* void * *ry_ptr,* void * *rz_ptr,* void * *px_ptr,* void * *py_ptr,* void * *pz_ptr,* void * *par_ptr,* int *numparticles,* int & *numaddback*) [virtual]

Special calse CollimatorPhysicsSort kernel that uses SoA instead of AoS.

Used only on the MIC side, was not implemented on the GPU.

Implements DKSCollimatorPhysics.

4.33.2.4 int MICCollimatorPhysics::ParallelTTrackerPush (void * *r_ptr*, void * *p_ptr*, int *npart*, void * *dt_ptr*, double *dt*, double *c*, bool *usedt* = false, int *streamId* = -1) [virtual]

BorisPusher push function for integration from OPAL.

ParallelTTracker integration from OPAL implemented in cuda. For more details see ParallelTTracler docomentation in opal

Implements DKSCollimatorPhysics.

4.33.2.5 int MICCollimatorPhysics::ParallelTTrackerPushTransform (void * *x_ptr*, void * *p_ptr*, void * *lastSec_ptr*, void * *orient_ptr*, int *npart*, int *nsec*, void * *dt_ptr*, double *dt*, double *c*, bool *usedt* = false, int *streamId* = -1) [virtual]

BorisPusher push function with transformto function form OPAL.

ParallelTTracker integration from OPAL implemented in cuda. For more details see ParallelTTracler docomentation in opal

Implements DKSCollimatorPhysics.

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

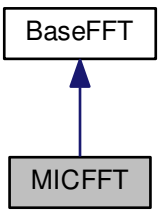
- src/MIC/MICCollimatorPhysics.h
- src/MIC/MICCollimatorPhysics.cpp

4.34 MICFFT Class Reference

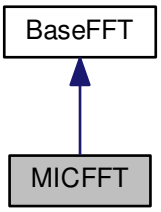
MIC FFT based on BaseFFT interface.

`#include <MICFFT.h>`

Inheritance diagram for MICFFT:



Collaboration diagram for MICFFT:



Public Member Functions

- **MICFFT** (MICBase *base)
- int setupFFT (int ndim, int N[3])

  Setup FFT - init FFT library used by chosen device.
- int setupFFTRC (int ndim, int N[3], double scale=1.0)

  Setup real to complex FFT - init FFT library used by chosen device.
- int setupFFTCR (int ndim, int N[3], double scale=1.0)

  Setup real to complex complex to real FFT - init FFT library used by chosen device.
- int executeFFT (void *mem_ptr, int ndim, int N[3], int streamId=-1, bool forward=true)

  Exectute C2C FFT.
- int executeIFFT (void *mem_ptr, int ndim, int N[3], int streamId=-1)

  Exectute inverse C2C FFT.
- int executeRCFFT (void *in_ptr, void *out_ptr, int ndim, int N[3], int streamId=-1)

  Exectute R2C FFT.
- int executeCRFFT (void *in_ptr, void *out_ptr, int ndim, int N[3], int streamId=-1)

  Exectute C2R FFT.
- int normalizeFFT (void *mem_ptr, int ndim, int N[3], int streamId=-1)

  Normalize the FFT or IFFT.
- int destroyFFT ()

  Info: destroy default FFT plans Return: success or error code.
- int normalizeCRFFT (void *real_ptr, int ndim, int N[3], int streamId=-1)

  Normalize CR FFT.

Additional Inherited Members

4.34.1 Detailed Description

MIC FFT based on BaseFFT interface.

uses MKL library to offload FFT on Intel Xeon Phi devices.

4.34.2 Member Function Documentation

4.34.2.1 int MICFFT::executeCRFFT (void * *real_ptr,* void * *comp_ptr,* int *ndim,* int *N[3],* int *streamId* = -1) [virtual]

Exectute C2R FFT.

real_ptr - real output data from the C2R FFT, comp_ptr - complex input data for the FFT.

Implements BaseFFT.

4.34.2.2 int MICFFT::executeFFT (void * *mem_ptr,* int *ndim,* int *N[3],* int *streamId* = -1, bool *forward* = true) [virtual]

Exectute C2C FFT.

mem_ptr - memory ptr on the device for complex data. Performs in place FFT.

Implements BaseFFT.

Here is the caller graph for this function:

4.34.2.3 int MICFFT::executeIFFT (void * *mem_ptr,* int *ndim,* int *N[3],* int *streamId* = -1) [virtual]

Exectute inverse C2C FFT.

mem_ptr - memory ptr on the device for complex data. Performs in place FFT.

Implements BaseFFT.

Here is the call graph for this function:

4.34.2.4 int MICFFT::executeRCFFT (void * *real_ptr,* void * *comp_ptr,* int *ndim,* int *N[3],* int *streamId* = -1) [virtual]

Exectute R2C FFT.

real_ptr - real input data for FFT, comp_ptr - memory on the device where results for the FFT are stored as complex numbers.

Implements BaseFFT.

4.34.2.5 int MICFFT::normalizeFFT (void * *mem_ptr*, int *ndim*, int *N[3]*, int *streamId* = -1) [virtual]

Normalize the FFT or IFFT.

mem_ptr - memory to complex data.

Implements BaseFFT.

4.34.2.6 int MICFFT::setupFFT (int *ndim*, int *N[3]*) [virtual]

Setup FFT - init FFT library used by chosen device.

Implements BaseFFT.

4.34.2.7 int MICFFT::setupFFTCR (int *ndim*, int *N[3]*, double *scale* = 1.0) [virtual]

Setup real to complex complex to real FFT - init FFT library used by chosen device.

Implements BaseFFT.

4.34.2.8 int MICFFT::setupFFTRC (int *ndim*, int *N[3]*, double *scale* = 1.0) [virtual]

Setup real to complex FFT - init FFT library used by chosen device.

Implements BaseFFT.

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

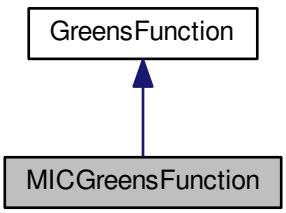
- src/MIC/MICFFT.h
- src/MIC/MICFFT.cpp

4.35 MICGreensFunction Class Reference

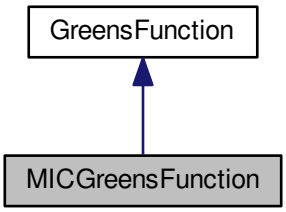
OpenMP offload implementation of GreensFunction calculation for OPALs Poisson Solver.

`#include <MICGreensFunction.hpp>`

Inheritance diagram for MICGreensFunction:



Collaboration diagram for MICGreensFunction:



Public Member Functions

- **MICGreensFunction** (MICBase ∗base)
- int greensIntegral (void ∗tmpgreen_, int I, int J, int K, int NI, int NJ, double hr_m0, double hr_m1, double hr_m2, int streamId=-1)

  calc greens integral, as defined in OPAL.
- int integrationGreensFunction (void ∗rho2_m, void ∗tmpgreen, int I, int J, int K, int stremaId=-1)

  integration if rho2_m, see OPAL for more details.
- int mirrorRhoField (void ∗rho2_m, int I, int J, int K, int streamId=-1)

  mirror rho2_m field.
- int multiplyCompelxFields (void ∗ptr1, void ∗ptr2, int size, int streamId=-1)

  multiply two complex fields from device memory.

4.35.1 Detailed Description

OpenMP offload implementation of GreensFunction calculation for OPALs Poisson Solver.

4.35.2 Member Function Documentation

4.35.2.1 int MICGreensFunction::greensIntegral (void ∗ *tmpgreen,* int *I,* int *J,* int *K,* int *NI,* int *NJ,* double *hr_m0,* double *hr_m1,* double *hr_m2,* int *streamId* = −1) [virtual]

calc greens integral, as defined in OPAL.

Implements GreensFunction.

4.35.2.2 int MICGreensFunction::integrationGreensFunction (void ∗ *rho2_m,* void ∗ *tmpgreen,* int *I,* int *J,* int *K,* int *streamId* = −1) [virtual]

integration if rho2_m, see OPAL for more details.

Implements GreensFunction.

4.35.2.3 int MICGreensFunction::mirrorRhoField (void ∗ *rho2_m,* int *I,* int *J,* int *K,* int *streamId* = −1) [virtual]

mirror rho2_m field.

Implements GreensFunction.

4.35.2.4 int MICGreensFunction::multiplyCompelxFields (void * *ptr1*, void * *ptr2*, int *size*, int *streamId* = -1) [virtual]

multiply two complex fields from device memory.

Implements GreensFunction.

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

- src/MIC/MICGreensFunction.hpp
- src/MIC/MICGreensFunction.cpp

4.36 OpenCLBase Class Reference

OpenCL base class to handle device setup and basic communication wiht the device.

`#include <OpenCLBase.h>`

Public Member Functions

- OpenCLBase ()
  constructor
- ~OpenCLBase ()
  destructor
- int ocl_createRndStates (int size)
  Allocate memory for size random number states and init the rnd states.
- int ocl_createRandomNumbers (void *mem_ptr, int size)
  Create an array of random numbers on the device.
- int ocl_deleteRndStates ()
  Destroy rnd states and free device memory.
- int ocl_getAllDevices ()
  Prints info about all the available platforms and devices.
- int ocl_getDeviceCount (int &ndev)
  Get the OpenCL device count for the set type of device.
- int ocl_getDeviceName (std::string &device_name)
  Get the name of the device currently us use.
- int ocl_setDevice (int device)
  Set the device to use for OpenCL kernels.
- int ocl_getUniqueDevices (std::vector< int > &devices)
  Get a list of all the unique devices of the same type that can run OpenCL kernels.
- int ocl_setUp (const char *device_name)
  Initialize OpenCL connection with a device of specified type.
- int ocl_loadKernel (const char *kernel_file)
  Given a OpenCL kernel file name loads the content and compile the OpenCL code.
- int ocl_loadKernelFromSource (const char *kernel_source, const char *opts=NULL)
  Build program from kernel source.
- cl_mem ocl_allocateMemory (size_t size, int &ierr)
  Allocate memory on the device.
- cl_mem ocl_allocateMemory (size_t size, int type, int &ierr)
  Allocate memory of specific type on device.
- template<typename T >
  int ocl_fillMemory (cl_mem mem_ptr, size_t size, T value, int offset=0)

Zero OpenCL memory buffer.

- int ocl_writeData (cl_mem mem_ptr, const void *in_data, size_t size, size_t offset=0, int blocking=CL_TRUE)

  Write data to device memory (needs ptr to mem object) Return: success or error code.
- int ocl_copyData (cl_mem src_ptr, cl_mem dst_ptr, size_t size)

  Copy data from one buffer on the device to another Return: success or error code.
- int ocl_createKernel (const char *kernel_name)

  Create kernel from compiled OpenCL program.
- int ocl_setKernelArg (int idx, size_t size, const void *arg_value)

  Set argiments for the kernel that will be launched.
- int ocl_executeKernel (cl_uint, const size_t *work_items, const size_t *work_grou_size=NULL)

  Execute selected kernel.
- int ocl_readData (cl_mem mem_ptr, void *out_data, size_t size, size_t offset=0, int blocking=CL_TRUE)

  Read data from device (needs pointer to mem object).
- int ocl_freeMemory (cl_mem mem_ptr)

  Free device memory (needs ptr to mem object).
- int ocl_cleanUp ()

  Free opencl resources.
- int ocl_deviceInfo (bool verbose=true)

  Print info of currently selected device.
- int **ocl_checkKernel** (const char *kernel_name, int work_group_size, bool double_precision, int &threadsPerBlock)
- void ocl_clearEvents ()

  Clear the event list.
- void ocl_eventInfo ()

  print information about kernel timings from event list.
- cl_command_queue ocl_getQueue ()

  Return current command queue.

Static Public Attributes

- static cl_context **m_context** = NULL
- static cl_command_queue **m_command_queue** = NULL

Protected Attributes

- int **defaultRndSet**
- cl_mem **defaultRndState**

4.36.1 Detailed Description

OpenCL base class to handle device setup and basic communication wiht the device.

Handles initialization of OpenCL device, memory manegement, data transfer and kernel launch. The OpenCL kernels are located in seperate files in OpenCLKernels folder, the OpenCLBase class contains methods to read the kernel files, compile the kernel codes and launch kernels from the compiled codes. Which kernel file needs to be loaded for the specif functin is handled by the base class that is launching the kernel.

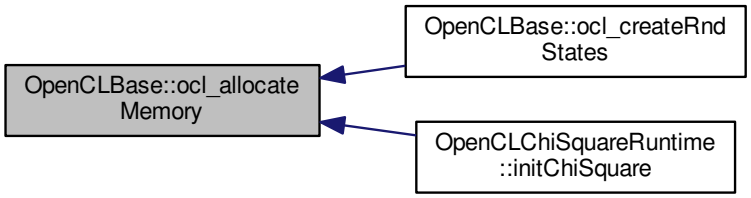
4.36.2 Member Function Documentation

4.36.2.1 cl_mem OpenCLBase::ocl_allocateMemory (size_t *size,* int & *ierr*)

Allocate memory on the device.

Return: return pointer to memory

Here is the caller graph for this function:



4.36.2.2 cl_mem OpenCLBase::ocl_allocateMemory (size_t *size,* int *type,* int & *ierr*)

Allocate memory of specific type on device.

The availabel types are cl_mem_flags type listed in OpenCL documentation: CL_MEM_READ_WRITE, CL_MEM_WRITE_ONLY, CL_MEM_USE_HOST_PTR, CL_MEM_ALLOC_HOST_PTR and CL_MEM_COPY_HOST_PTR. Return: return pointer to memory

4.36.2.3 int OpenCLBase::ocl_cleanUp ( )

Free opencl resources.

Deletes the kernel, compiled program, command queue and colese the connection to device by releasing the context. Return: success or error code

Here is the caller graph for this function:

4.36.2.4 void OpenCLBase::ocl_clearEvents ( )

Clear the event list.

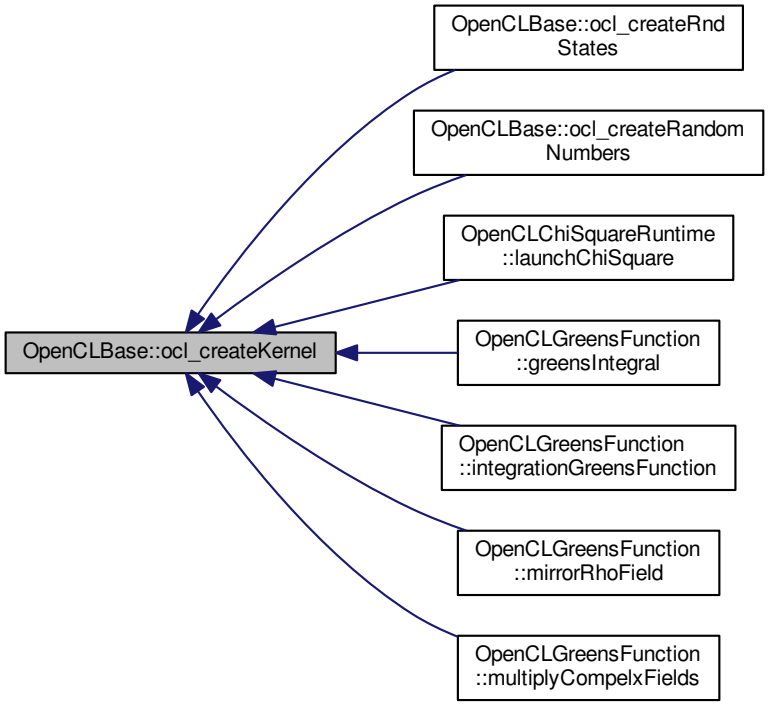
Events can be used for timing and synchronization purposes.

4.36.2.5 int OpenCLBase::ocl_createKernel (const char * *kernel_name*)

Create kernel from compiled OpenCL program.

Return: success or error code

Here is the caller graph for this function:

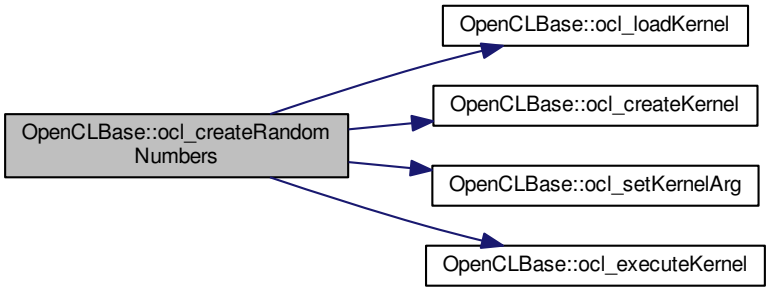


4.36.2.6 int OpenCLBase::ocl_createRandomNumbers (void * *mem_ptr,* int *size*)

Create an array of random numbers on the device.

Filles hte mem_ptr with random numbers.

Here is the call graph for this function:

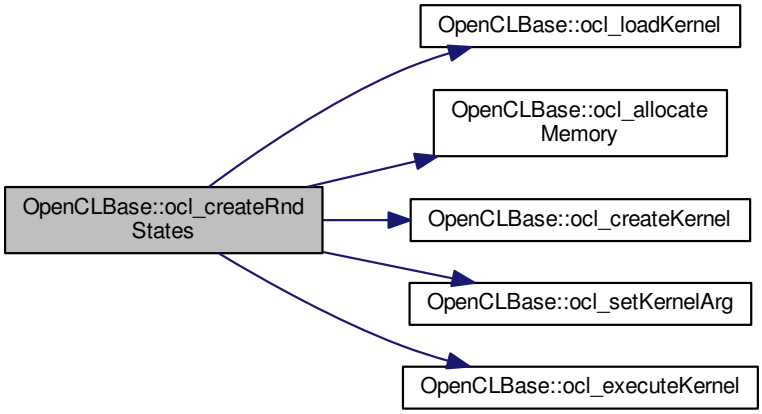


4.36.2.7 int OpenCLBase::ocl_createRndStates (int *size*)

Allocate memory for size random number states and init the rnd states.

Uses AMD clRng library for random numbers. This library is only compatible with AMD devices.

Here is the call graph for this function:

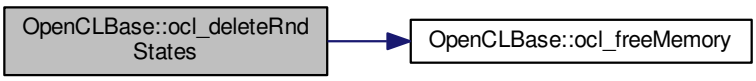


4.36.2.8 int OpenCLBase::ocl_deleteRndStates ( )

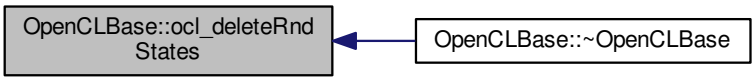
Destroy rnd states and free device memory.

Return: success or error code

Here is the call graph for this function:



Here is the caller graph for this function:

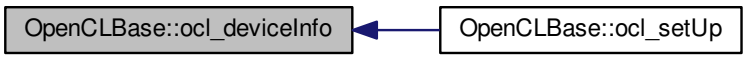


4.36.2.9 int OpenCLBase::ocl_deviceInfo (bool *verbose* = `true`)

Print info of currently selected device.

Mostly for debugging purposes, but in verbose mode can be used to see device properties. Return: success or error code

Here is the caller graph for this function:



4.36.2.10 void OpenCLBase::ocl_eventInfo ( )

print information about kernel timings from event list.

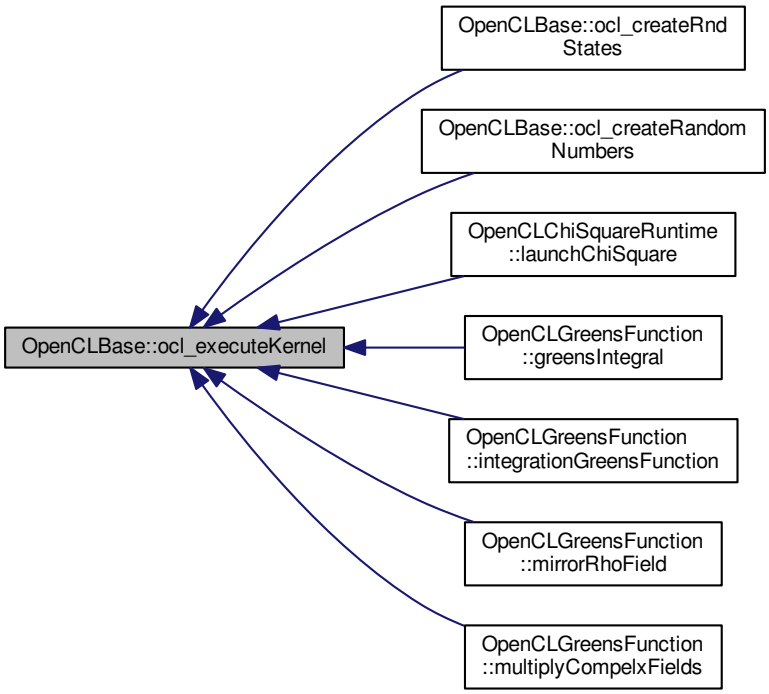
for debuging purposes

4.36.2.11 int OpenCLBase::ocl_executeKernel (cl_uint *ndim*, const size_t * *work_items*, const size_t * *work_grou_size* = `NULL`)

Execute selected kernel.

Before kenrel can be executed buildProgram must be executed, create kernel must be executed and kenre specifeid in execute kerenel must be in compiled source, and the necessary kernel arguments must be set. Return: success or error code

Here is the caller graph for this function:

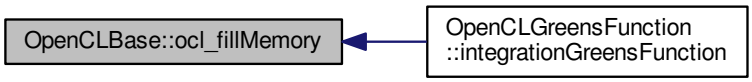


4.36.2.12 template<typename T > int OpenCLBase::ocl_fillMemory (cl_mem *mem_ptr*, size_t *size*, T *value*, int *offset* = 0) [inline]

Zero OpenCL memory buffer.

Set all the elemetns in the device array to zero.

Here is the caller graph for this function:

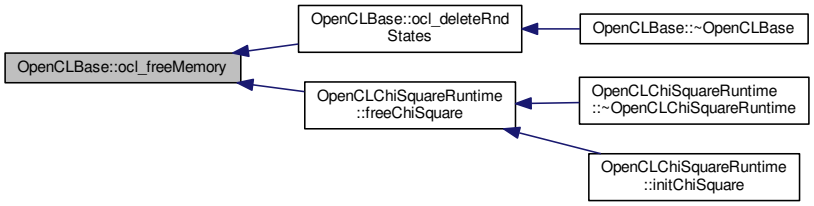


4.36.2.13 int OpenCLBase::ocl_freeMemory (cl_mem *mem_ptr*)

Free device memory (needs ptr to mem object).

Return: success or error code

Here is the caller graph for this function:



4.36.2.14 int OpenCLBase::ocl_getAllDevices ( )

Prints info about all the available platforms and devices.

Can be used for information purposes to see what devices are available on the system. ReturnL success or error code.

4.36.2.15 int OpenCLBase::ocl_getDeviceCount (int & *ndev*)

Get the OpenCL device count for the set type of device.

Device count is set in ndev parameter, returns success or error code.

4.36.2.16 int OpenCLBase::ocl_getUniqueDevices (std::vector< int > & *devices*)

Get a list of all the unique devices of the same type that can run OpenCL kernels.

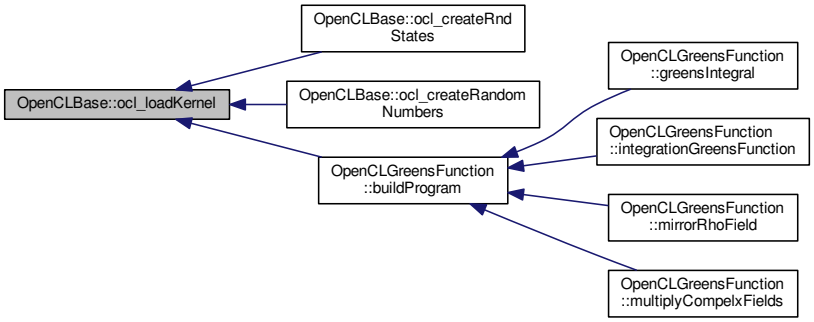
Used when GPUs of different types might be pressent on the system.

4.36.2.17 int OpenCLBase::ocl_loadKernel (const char * *kernel_file*)

Given a OpenCL kernel file name loads the content and compile the OpenCL code.

Load and compile opencl kernel file if it has changed. Return: success or error code

Here is the caller graph for this function:

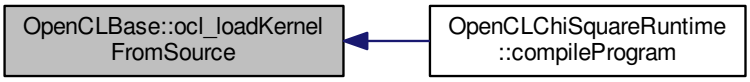


4.36.2.18 int OpenCLBase::ocl_loadKernelFromSource (const char * *kernel_source,* const char * *opts* = NULL)

Build program from kernel source.

Builds a program from source code provided in kernel_source. If compilation fails will return DKS_ERROR

Here is the caller graph for this function:



4.36.2.19 int OpenCLBase::ocl_readData (cl_mem *mem_ptr,* void * *out_data,* size_t *size,* size_t *offset* = 0, int *blocking* = CL_TRUE)

Read data from device (needs pointer to mem object).

Return: success or error code

4.36.2.20 int OpenCLBase::ocl_setDevice (int *device*)

Set the device to use for OpenCL kernels.

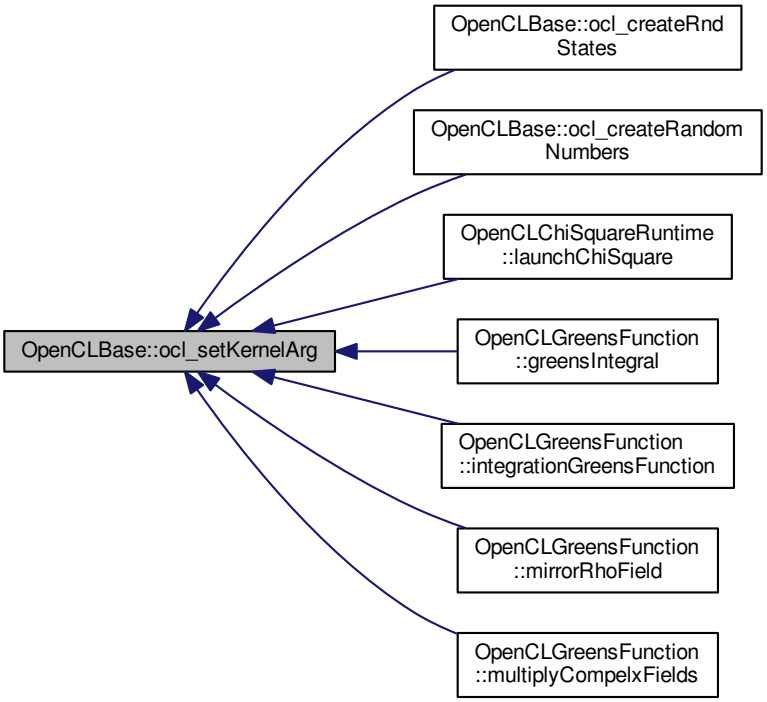
Device id to use is passed as integer.

4.36.2.21 int OpenCLBase::ocl_setKernelArg (int *idx,* size_t *size,* const void * *arg_value*)

Set argiments for the kernel that will be launched.

Return: success or error code

Here is the caller graph for this function:

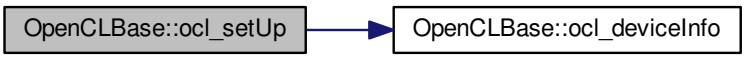


4.36.2.22 int OpenCLBase::ocl_setUp (const char * *device_name*)

Initialize OpenCL connection with a device of specified type.

Find if specified device is avaialble, creates a contex and command queue. Returns success or error code.

Here is the call graph for this function:



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

- src/OpenCL/OpenCLBase.h
- src/OpenCL/OpenCLBase.cpp

4.37 OpenCLChiSquare Class Reference

Deprecated, SimpleFit implementation of ChiSquare.

`#include <OpenCLChiSquare.h>`

Public Member Functions

- **OpenCLChiSquare** (OpenCLBase *base)
- int **ocl_PHistoTFFcn** (void *mem_data, void *mem_par, void *mem_result, double fTimeResolution, double fRebin, int sensors, int length, int numpar, double &result)
- int **ocl_singleGaussTF** (void *mem_data, void *mem_t0, void *mem_par, void *mem_result, double fTimeResolution, double fRebin, double fGoodBinOffset, int sensors, int length, int numpar, double &result)
- int **ocl_doubleLorentzTF** (void *mem_data, void *mem_t0, void *mem_par, void *mem_result, double fTimeResolution, double fRebin, double fGoodBinOffset, int sensors, int length, int numpar, double &result)

4.37.1 Detailed Description

Deprecated, SimpleFit implementation of ChiSquare.

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

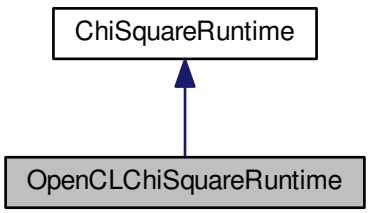
- src/OpenCL/OpenCLChiSquare.h
- src/OpenCL/OpenCLChiSquare.cpp

4.38 OpenCLChiSquareRuntime Class Reference

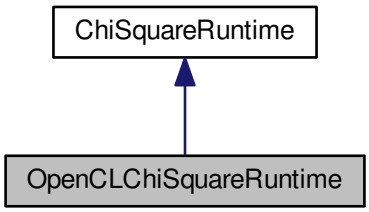
OpenCL implementation of ChiSquareRuntime class.

`#include <OpenCLChiSquareRuntime.h>`

Inheritance diagram for OpenCLChiSquareRuntime:



Collaboration diagram for OpenCLChiSquareRuntime:



Public Member Functions

- OpenCLChiSquareRuntime (OpenCLBase ∗base)

  Constructor wiht openclbase argument.
- OpenCLChiSquareRuntime ()

  Default constructor.
- ∼OpenCLChiSquareRuntime ()

  Default destructor.
- int compileProgram (std::string function, bool mlh=false)

  Compile program and save ptx.
- int launchChiSquare (int fitType, void ∗mem_data, void ∗mem_err, int length, int numpar, int numfunc, int nummap, double timeStart, double timeStep, double &result)

  Launch selected kernel.
- int writeParams (const double ∗params, int numparams)

  Write params to device.
- int writeFunc (const double ∗func, int numfunc)

  Write functions to device.
- int writeMap (const int ∗map, int nummap)

  Write maps to device.
- int initChiSquare (int size_data, int size_param, int size_func, int size_map)

  Allocate temporary memory needed for chi square.
- int freeChiSquare ()

  Free temporary memory allocated for chi square.
- int checkChiSquareKernels (int fitType, int &threadsPerBlock)

  Check MuSR kernels for necessary resources.

Additional Inherited Members

4.38.1 Detailed Description

OpenCL implementation of ChiSquareRuntime class.

Implements ChiSquareRuntime interface to allow musrfit to target devices that support OpenCL - Nvidia and AMD GPUs, Intel and AMD CPUs, Intel Xeon Phi.

4.38.2 Member Function Documentation

4.38.2.1 int OpenCLChiSquareRuntime::checkChiSquareKernels (int *fitType*, int & *threadsPerBlock*) [virtual]

Check MuSR kernels for necessary resources.

Query device properties to get if sufficient resources are available to run the kernels. Also checks if double precision is enabled on the device.

Implements ChiSquareRuntime.

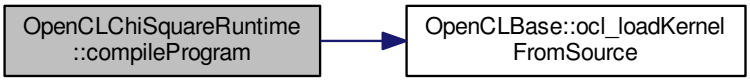
4.38.2.2 int OpenCLChiSquareRuntime::compileProgram (std::string *function*, bool *mlh* = false) [virtual]

Compile program and save ptx.

Add function string to the calcFunction kernel and compile the program Function must be valid C math expression. Parameters can be addressed in a form par[map[idx]]

Implements ChiSquareRuntime.

Here is the call graph for this function:



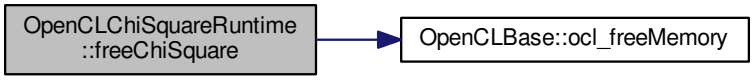
4.38.2.3 int OpenCLChiSquareRuntime::freeChiSquare ( ) [virtual]

Free temporary memory allocated for chi square.

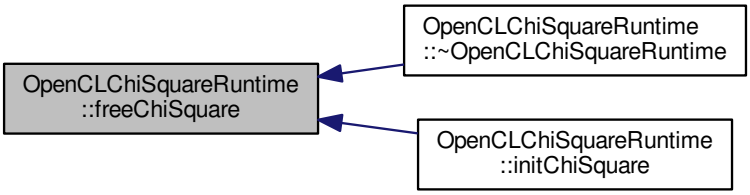
Frees the chisq temporary memory and memory for params, functions and maps

Implements ChiSquareRuntime.

Here is the call graph for this function:



Here is the caller graph for this function:



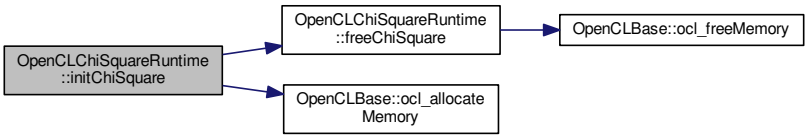
4.38.2.4 int OpenCLChiSquareRuntime::initChiSquare (int *size_data,* int *size_param,* int *size_func,* int *size_map*) [virtual]

Allocate temporary memory needed for chi square.

Initializes the necessary temporary memory for the chi square calculations. Size_data needs to the maximum number of elements in any datasets that will be used for calculations. Size_param, size_func and size_map are the maximum number of parameters, functions and maps used in calculations.

Implements ChiSquareRuntime.

Here is the call graph for this function:



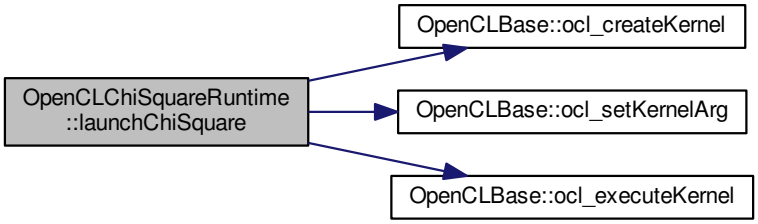
4.38.2.5 int OpenCLChiSquareRuntime::launchChiSquare (int *fitType,* void * *mem_data,* void * *mem_err,* int *length,* int *numpar,* int *numfunc,* int *nummap,* double *timeStart,* double *timeStep,* double & *result*) [virtual]

Launch selected kernel.

Launched the selected kernel from the compiled code. Result is put in &result variable

Implements ChiSquareRuntime.

Here is the call graph for this function:



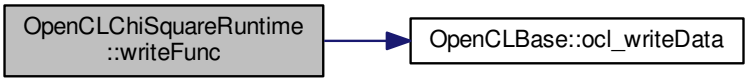
4.38.2.6 int OpenCLChiSquareRuntime::writeFunc (const double * *func*, int *numfunc*) [virtual]

Write functions to device.

Write function values from double array to mem_func_m memory on the device.

Implements ChiSquareRuntime.

Here is the call graph for this function:



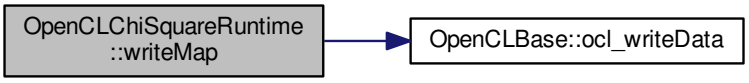
4.38.2.7 int OpenCLChiSquareRuntime::writeMap (const int * *map*, int *nummap*) [virtual]

Write maps to device.

Write map values from int array to mem_map_m memory on the device.

Implements ChiSquareRuntime.

Here is the call graph for this function:



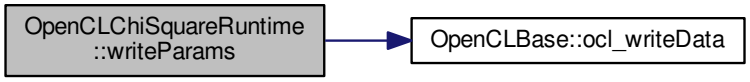
4.38.2.8 int OpenCLChiSquareRuntime::writeParams (const double * *params*, int *numparams*) [virtual]

Write params to device.

Write params from double array to mem_param_m memory on the device.

Implements ChiSquareRuntime.

Here is the call graph for this function:



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

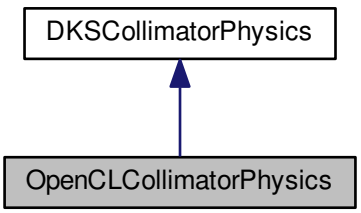
- src/OpenCL/OpenCLChiSquareRuntime.h
- src/OpenCL/OpenCLChiSquareRuntime.cpp

4.39 OpenCLCollimatorPhysics Class Reference

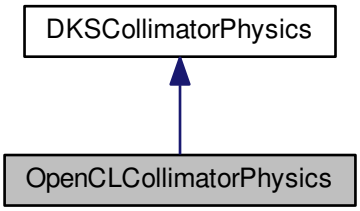
OpenCLCollimatorPhysics class based on DKSCollimatorPhysics interface.

`#include <OpenCLCollimatorPhysics.h>`

Inheritance diagram for OpenCLCollimatorPhysics:



Collaboration diagram for OpenCLCollimatorPhysics:



Public Member Functions

- OpenCLCollimatorPhysics (OpenCLBase ∗base)

  Constructor with OpenCLBase as argument.
- ∼OpenCLCollimatorPhysics ()

  Destructor.
- int CollimatorPhysics (void ∗mem_ptr, void ∗par_ptr, int numparticles, bool enableRutherforScattering=true)

  Execute collimator physics kernel.
- int CollimatorPhysicsSoA (void ∗label_ptr, void ∗localID_ptr, void ∗rx_ptr, void ∗ry_ptr, void ∗rz_ptr, void ∗px_ptr, void ∗py_ptr, void ∗pz_ptr, void ∗par_ptr, int numparticles)

  Special calse CollimatorPhysics kernel that uses SoA instead of AoS.
- int CollimatorPhysicsSort (void ∗mem_ptr, int numparticles, int &numaddback)

  Sort particle array on GPU.
- int CollimatorPhysicsSortSoA (void ∗label_ptr, void ∗localID_ptr, void ∗rx_ptr, void ∗ry_ptr, void ∗rz_ptr, void ∗px_ptr, void ∗py_ptr, void ∗pz_ptr, void ∗par_ptr, int numparticles, int &numaddback)

  Special calse CollimatorPhysicsSort kernel that uses SoA instead of AoS.
- int ParallelTTrackerPush (void ∗r_ptr, void ∗p_ptr, int npart, void ∗dt_ptr, double dt, double c, bool usedt=false, int streamId=-1)

  BorisPusher push function for integration from OPAL.
- int ParallelTTrackerPushTransform (void ∗x_ptr, void ∗p_ptr, void ∗lastSec_ptr, void ∗orient_ptr, int npart, int nsec, void ∗dt_ptr, double dt, double c, bool usedt=false, int streamId=-1)

  BorisPusher push function with transformto function form OPAL.

Additional Inherited Members

4.39.1 Detailed Description

OpenCLCollimatorPhysics class based on DKSCollimatorPhysics interface.

Implementes CollimatorPhysics for OPAL using OpenCL for execution on AMD GPUs.

4.39.2 Constructor & Destructor Documentation

4.39.2.1 OpenCLCollimatorPhysics::OpenCLCollimatorPhysics (OpenCLBase ∗ *base*) [inline]

Constructor with OpenCLBase as argument.

Create a new instace of the OpenCLCollimatorPhysics using existing OpenCLBase object.

4.39.3 Member Function Documentation

4.39.3.1 int OpenCLCollimatorPhysics::CollimatorPhysicsSoA (void * *label_ptr,* void * *localID_ptr,* void * *rx_ptr,* void * *ry_ptr,* void * *rz_ptr,* void * *px_ptr,* void * *py_ptr,* void * *pz_ptr,* void * *par_ptr,* int *numparticles*) [inline], [virtual]

Special calse CollimatorPhysics kernel that uses SoA instead of AoS.

Used only on the MIC side, was not implemented on the GPU.

Implements DKSCollimatorPhysics.

4.39.3.2 int OpenCLCollimatorPhysics::CollimatorPhysicsSort (void * *mem_ptr,* int *numparticles,* int & *numaddback*) [inline], [virtual]

Sort particle array on GPU.

Count particles that are dead (label -1) or leaving material (label -2) and sort particle array so these particles are at the end of array

Implements DKSCollimatorPhysics.

4.39.3.3 int OpenCLCollimatorPhysics::CollimatorPhysicsSortSoA (void * *label_ptr,* void * *localID_ptr,* void * *rx_ptr,* void * *ry_ptr,* void * *rz_ptr,* void * *px_ptr,* void * *py_ptr,* void * *pz_ptr,* void * *par_ptr,* int *numparticles,* int & *numaddback*) [inline], [virtual]

Special calse CollimatorPhysicsSort kernel that uses SoA instead of AoS.

Used only on the MIC side, was not implemented on the GPU.

Implements DKSCollimatorPhysics.

4.39.3.4 int OpenCLCollimatorPhysics::ParallelTTrackerPush (void * *r_ptr,* void * *p_ptr,* int *npart,* void * *dt_ptr,* double *dt,* double *c,* bool *usedt* = false, int *streamId* = -1) [inline], [virtual]

BorisPusher push function for integration from OPAL.

ParallelTTracker integration from OPAL implemented in cuda. For more details see ParallelTTracler docomentation in opal

Implements DKSCollimatorPhysics.

4.39.3.5 int OpenCLCollimatorPhysics::ParallelTTrackerPushTransform (void * *x_ptr,* void * *p_ptr,* void * *lastSec_ptr,* void * *orient_ptr,* int *npart,* int *nsec,* void * *dt_ptr,* double *dt,* double *c,* bool *usedt* = false, int *streamId* = -1) [inline], [virtual]

BorisPusher push function with transformto function form OPAL.

ParallelTTracker integration from OPAL implemented in cuda. For more details see ParallelTTracler docomentation in opal

Implements DKSCollimatorPhysics.

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

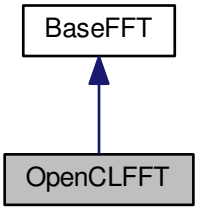
- src/OpenCL/OpenCLCollimatorPhysics.h
- src/OpenCL/OpenCLCollimatorPhysics.cpp

4.40 OpenCLFFT Class Reference

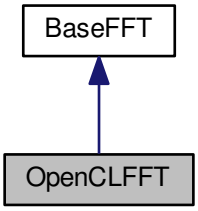
OpenCL FFT class based on BaseFFT interface.

`#include <OpenCLFFT.h>`

Inheritance diagram for OpenCLFFT:



Collaboration diagram for OpenCLFFT:



Public Member Functions

- **OpenCLFFT** (OpenCLBase *base)
- int executeFFT (void *data, int ndim, int N[3], int streamId=-1, bool forward=true)

  Exectute C2C FFT.
- int executeIFFT (void *data, int ndim, int N[3], int streamId=-1)

  Exectute inverse C2C FFT.
- int normalizeFFT (void *data, int ndim, int N[3], int streamId=-1)

  Normalize the FFT or IFFT.
- int setupFFT (int ndim, int N[3])

  Setup FFT - init FFT library used by chosen device.
- int setupFFTRC (int ndim, int N[3], double scale=1.0)

  Setup real to complex FFT - init FFT library used by chosen device.
- int setupFFTCR (int ndim, int N[3], double scale=1.0)

  Setup real to complex complex to real FFT - init FFT library used by chosen device.
- int destroyFFT ()

Clean up.

- int executeRCFFT (void *real_ptr, void *comp_ptr, int ndim, int N[3], int streamId=-1)

  Exectute R2C FFT.

- int executeCRFFT (void *real_ptr, void *comp_ptr, int ndim, int N[3], int streamId=-1)

  Exectute C2R FFT.

- int normalizeCRFFT (void *real_ptr, int ndim, int N[3], int streamId=-1)

  Normalize CR FFT.

Additional Inherited Members

4.40.1 Detailed Description

OpenCL FFT class based on BaseFFT interface.

Uses clFFT library to perform FFTs on AMD gpus. clFFT library works also on nvida GPUs and other devices that support OpenCL.

4.40.2 Member Function Documentation

4.40.2.1 int OpenCLFFT::destroyFFT ( ) [virtual]

Clean up.

Implements BaseFFT.

4.40.2.2 int OpenCLFFT::executeCRFFT (void * *real_ptr*, void * *comp_ptr*, int *ndim*, int *N[3]*, int *streamId* = `-1`) [virtual]

Exectute C2R FFT.

real_ptr - real output data from the C2R FFT, comp_ptr - complex input data for the FFT.

Implements BaseFFT.

4.40.2.3 int OpenCLFFT::executeFFT (void * *mem_ptr*, int *ndim*, int *N[3]*, int *streamId* = `-1`, bool *forward* = `true`) [virtual]

Exectute C2C FFT.

mem_ptr - memory ptr on the device for complex data. Performs in place FFT.

Implements BaseFFT.

Here is the caller graph for this function:

4.40.2.4 int OpenCLFFT::executeIFFT (void * *mem_ptr*, int *ndim*, int *N[3]*, int *streamId* = -1) [virtual]

Exectute inverse C2C FFT.

mem_ptr - memory ptr on the device for complex data. Performs in place FFT.

Implements BaseFFT.

Here is the call graph for this function:

4.40.2.5 int OpenCLFFT::executeRCFFT (void * *real_ptr*, void * *comp_ptr*, int *ndim*, int *N[3]*, int *streamId* = -1) [virtual]

Exectute R2C FFT.

real_ptr - real input data for FFT, comp_ptr - memory on the device where results for the FFT are stored as complex numbers.

Implements BaseFFT.

4.40.2.6 int OpenCLFFT::normalizeFFT (void * *mem_ptr*, int *ndim*, int *N[3]*, int *streamId* = -1) [virtual]

Normalize the FFT or IFFT.

mem_ptr - memory to complex data.

Implements BaseFFT.

4.40.2.7 int OpenCLFFT::setupFFT (int *ndim*, int *N[3]*) [virtual]

Setup FFT - init FFT library used by chosen device.

Implements BaseFFT.

4.40.2.8 int OpenCLFFT::setupFFTCR (int *ndim*, int *N[3]*, double *scale* = 1.0) [virtual]

Setup real to complex complex to real FFT - init FFT library used by chosen device.

Implements BaseFFT.

4.40.2.9 int OpenCLFFT::setupFFTRC (int *ndim*, int *N[3]*, double *scale* = 1.0) [virtual]

Setup real to complex FFT - init FFT library used by chosen device.

Implements BaseFFT.

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

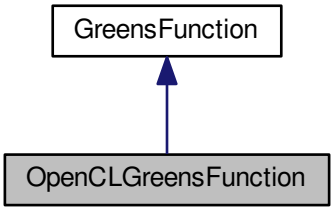
- src/OpenCL/OpenCLFFT.h
- src/OpenCL/OpenCLFFT.cpp

4.41 OpenCLGreensFunction Class Reference

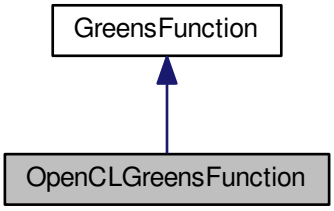
OpenCL implementation of GreensFunction calculation for OPALs Poisson Solver.

`#include <OpenCLGreensFunction.h>`

Inheritance diagram for OpenCLGreensFunction:



Collaboration diagram for OpenCLGreensFunction:



Public Member Functions

- OpenCLGreensFunction (OpenCLBase ∗base)

  Constructor with OpenCLBase argument.
- OpenCLGreensFunction ()

  Default constructor.
- ∼OpenCLGreensFunction ()

  Destructor.
- int buildProgram ()

  Load OpenCL kernel file containing greens function kernels.
- int greensIntegral (void ∗tmpgreen, int I, int J, int K, int NI, int NJ, double hr_m0, double hr_m1, double hr_m2, int streamId=-1)

  Info: calc itegral on device memory (taken from OPAL src code).
- int integrationGreensFunction (void ∗rho2_m, void ∗tmpgreen, int I, int J, int K, int streamId=-1)

  Info: integration of rho2_m field (taken from OPAL src code).
- int mirrorRhoField (void ∗rho2_m, int I, int J, int K, int streamId=-1)

Info: mirror rho field (taken from OPAL src code).

- int multiplyCompelxFields (void *ptr1, void *ptr2, int size, int streamId=-1)

  Info: multiply complex fields already on the GPU memory, result will be put in ptr1.

4.41.1 Detailed Description

OpenCL implementation of GreensFunction calculation for OPALs Poisson Solver.

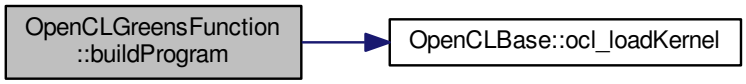
4.41.2 Member Function Documentation

4.41.2.1 int OpenCLGreensFunction::buildProgram ( )

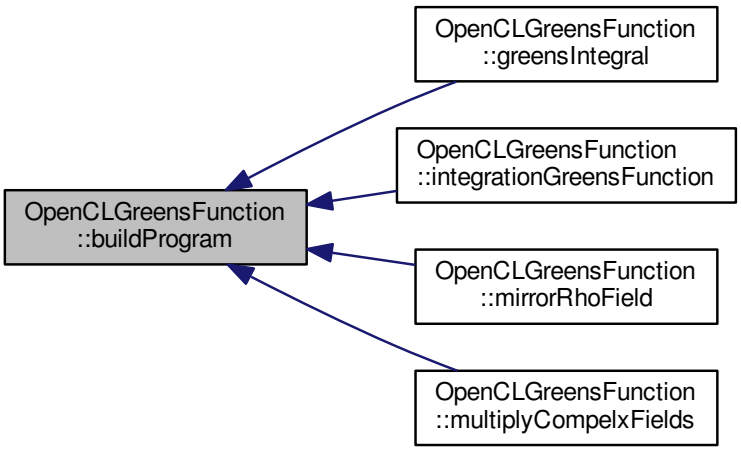
Load OpenCL kernel file containing greens function kernels.

m_base takes the kernel file and compiles the OpenCL programm.

Here is the call graph for this function:



Here is the caller graph for this function:



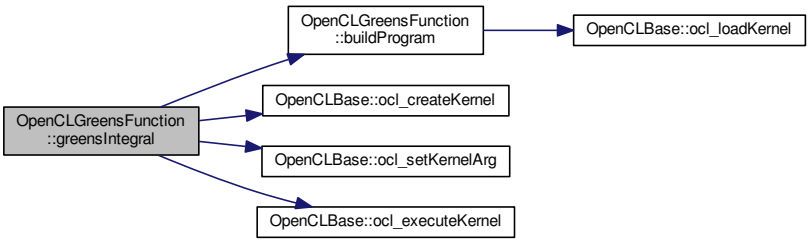
4.41.2.2 int OpenCLGreensFunction::greensIntegral (void * *tmpgreen*, int *I*, int *J*, int *K*, int *NI*, int *NJ*, double *hr_m0*, double *hr_m1*, double *hr_m2*, int *streamId* = -1) [virtual]

Info: calc itegral on device memory (taken from OPAL src code).

Return: success or error code

Implements GreensFunction.

Here is the call graph for this function:



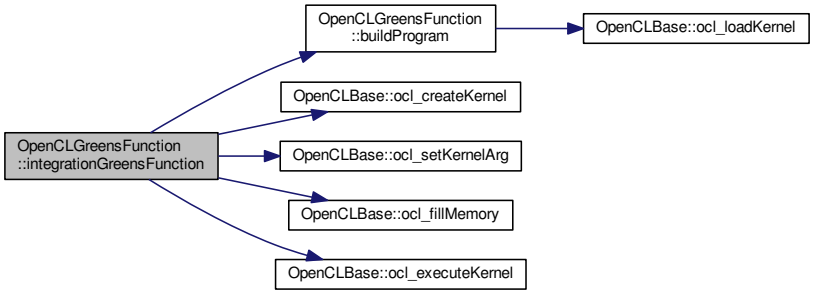
4.41.2.3 int OpenCLGreensFunction::integrationGreensFunction (void * *rho2_m*, void * *tmpgreen*, int *I*, int *J*, int *K*, int *streamId* = -1) [virtual]

Info: integration of rho2_m field (taken from OPAL src code).

Return: success or error code

Implements GreensFunction.

Here is the call graph for this function:



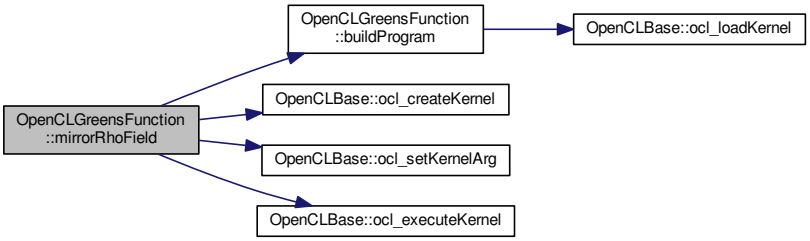
4.41.2.4 int OpenCLGreensFunction::mirrorRhoField (void * *rho2_m*, int *I*, int *J*, int *K*, int *streamId* = -1) [virtual]

Info: mirror rho field (taken from OPAL src code).

Return: succes or error code

Implements GreensFunction.

Here is the call graph for this function:



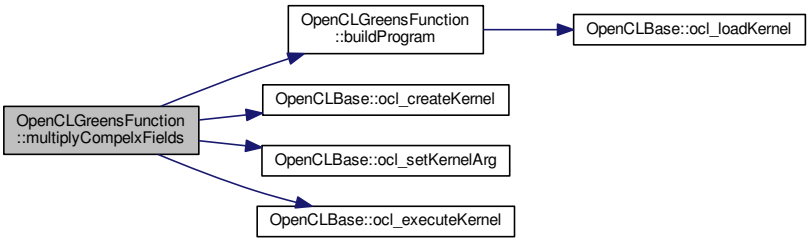
4.41.2.5 int OpenCLGreensFunction::multiplyCompelxFields (void * *ptr1*, void * *ptr2*, int *size*, int *streamId* = -1) [virtual]

Info: multiply complex fields already on the GPU memory, result will be put in ptr1.

Return: success or error code

Implements GreensFunction.

Here is the call graph for this function:



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

- src/OpenCL/OpenCLGreensFunction.h
- src/OpenCL/OpenCLGreensFunction.cpp

4.42 Parameter Class Reference

Parameter class allows to change the searchable parameters during the auto-tuning.

`#include <DKSSearchStates.h>`

Public Member Functions

- **Parameter** (int *_value, int _min, int _max, int _step, std::string _name)
- **Parameter** (double *_value, double _min, double _max, double _step, std::string _name)

- template<typename T >
  void **setValue** (T v)
- double **getValue** ()

Public Attributes

- double **min**
- double **max**
- double **step**
- std::string **name**

4.42.1 Detailed Description

Parameter class allows to change the searchable parameters during the auto-tuning.

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

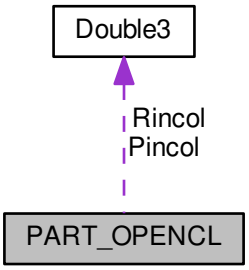
- src/AutoTuning/DKSSearchStates.h

4.43 PART_OPENCL Struct Reference

Structure for stroing particles in OpenCL code.

`#include <OpenCLCollimatorPhysics.h>`

Collaboration diagram for PART_OPENCL:



Public Attributes

- int **label**
- unsigned **localID**
- Double3 **Rincol**
- Double3 **Pincol**

4.43.1 Detailed Description

Structure for stroing particles in OpenCL code.

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

- src/OpenCL/OpenCLCollimatorPhysics.h

4.44 RNDState Struct Reference

struct for random number state.

`#include <OpenCLBase.h>`

Public Attributes

- double **s10**
- double **s11**
- double **s12**
- double **s20**
- double **s21**
- double **s22**
- double **z**
- bool **gen**

4.44.1 Detailed Description

struct for random number state.

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

- src/OpenCL/OpenCLBase.h

4.45 State Struct Reference

Struct to hold a auto-tuning state.

`#include <DKSSearchStates.h>`

Public Attributes

- double **value**
- double **min**
- double **max**
- double **step**

4.45.1 Detailed Description

Struct to hold a auto-tuning state.

Holds the current value, min, max and a step to witch a state can change.

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

- src/AutoTuning/DKSSearchStates.h

4.46 VoxelPosition Struct Reference

Struct to hold voxel position for PET image.

`#include <ImageReconstruction.h>`

Public Attributes

- float **x**
- float **y**
- float **z**

4.46.1 Detailed Description

Struct to hold voxel position for PET image.

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

- src/Algorithms/ImageReconstruction.h