Kernel Tuning Toolkit

1.0

Generated by Doxygen 1.8.13

Contents

Chapter 1

KTT - Kernel Tuning Toolkit

KTT is a C++ tuning library for OpenCL and CUDA kernels. Project is currently in late beta stage with all of the baseline functionality available.

Main features

- Ability to define kernel tuning parameters like thread count, vector data types and loop unroll factors in order to optimize computation for particular device
- Support for iterative kernel launches and composite kernels
- Ability to automatically ensure correctness of tuned computation with reference kernel or C++ function
- Ability to run kernels with low overhead after finding optimal configuration during kernel tuning
- Support for multiple compute APIs, switching between CUDA and OpenCL requires only minor changes in C++ code (eg. changing the kernel source file), no library recompilation is needed
- Large number of customization options, including ability to specify custom tolerance threshold for floating-point argument validation, ability to change kernel compiler flags and more

Getting started

- Documentation for KTT API can be found `here`.
- Newest version of KTT library can be found `here`.
- Prebuilt binaries are currently available only for some platforms. Other platforms require manual build.
- Prebuilt binaries for Nvidia include both CUDA and OpenCL support, binaries for AMD and Intel include only OpenCL support.

Examples

Examples showcasing KTT functionality are located inside examples folder. List of currently available examples:

- `compute_api_info (OpenCL / CUDA)`: basic example showing how to retrieve detailed information about compute API platforms and devices through KTT API
- `simple (OpenCL / CUDA)`: basic example showing how to run simple kernel with KTT framework, utilizes reference class, no actual autotuning is done
- `coulomb_sum_2d (OpenCL)`: advanced example which utilizes large number of tuning parameters, thread modifiers and constraints
- `coulomb_sum_3d_iterative (OpenCL)`: 3D version of previous example, utilizes tuning manipulator to iteratively launch 2D kernel
- `coulomb_sum_3d (OpenCL)`: alternative to iterative version, utilizes several tuning parameters and reference kernel
- `nbody (OpenCL)`: advanced example which utilizes tuning parameters, multiple constraints and validation of multiple arguments with reference kernel
- `reduction (OpenCL)`: advanced example which utilizes reference class, tuning manipulator and several tuning parameters

Building KTT

- KTT can be built as a dynamic (shared) library using command line build tool Premake. Currently supported operating systems are Linux and Windows.
- The prerequisites to build KTT are:
  - C++14 compiler, for example Clang 3.5, GCC 5.0, MSVC 19.0 (Visual Studio 2015) or newer
  - OpenCL or CUDA library, supported SDKs are AMD APP SDK 3.0, Intel SDK for OpenCL and NVIDIA CUDA Toolkit 7.5 or newer
  - `Premake 5` (alpha 12 or newer)
- Build under Linux (inside KTT root folder):
  - ensure that path to vendor SDK is correctly set in the environment variables
  - run `./premake5 gmake` to generate makefile
  - run `cd build` to get inside build directory
  - afterwards run `make config={configuration}_{architecture}` to build the project (eg. `make config=release_x86_64`)
- Build under Windows (inside KTT root folder):
  - ensure that path to vendor SDK is correctly set in the environment variables, this should be done automatically during SDK installation
  - run `premake5.exe vs2015` (or `premake5.exe vs2017`) to generate Visual Studio project files
  - open generated solution file and build the project inside Visual Studio
- Following build options are available:
  - `--outdir=path` specifies custom build directory, default build directory is `build`
  - `--platform=vendor` specifies SDK used for building KTT, useful when multiple SDKs are installed
  - `--no-examples` disables compilation of examples
  - `--tests` enables compilation of unit tests
  - `--no-cuda` disables inclusion of CUDA API during compilation, only affects Nvidia platform

Original project

KTT is based on `CLTune project`. Some parts of KTT API are similar to CLTune API, however internal structure was almost completely rewritten from scratch. Portions of code for following features were ported from CLTune:

- PSO and annealing searcher
- Generation of kernel configurations
- Tuning parameter constraints

Chapter 2

Namespace Documentation

2.1 ktt Namespace Reference

All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Classes

- class ArgumentOutputDescriptor

  Class which can be used to retrieve kernel argument data when calling certain KTT API methods.
- class DeviceInfo

  Class which holds information about a compute API device.
- class DimensionVector

  Class which holds information about either global or local thread size of a single kernel.
- class ParameterPair

  Class which holds single value for one kernel parameter.
- class PlatformInfo

  Class which holds information about a compute API platform.
- class ReferenceClass

  Class which can be used to compute reference output for selected kernel arguments inside regular C++ method. In order to use this functionality, new class which publicly inherits from reference class has to be defined.
- class Tuner

  Class which serves as the main part of public API for KTT library.
- class TuningManipulator

  Class which can be used to customize kernel launch in order to run some part of computation on CPU, utilize iterative kernel launches, kernel compositions and more. In order to use this functionality, new class which publicly inherits from tuning manipulator class has to be defined.

Typedefs

- using ArgumentId = size_t

  Data type for referencing kernel arguments in KTT.
- using KernelId = size_t

  Data type for referencing kernels in KTT.

Enumerations

- enum ArgumentAccessType { ArgumentAccessType::ReadOnly, ArgumentAccessType::WriteOnly, ArgumentAccessType::ReadWrite }

  Enum for access type of kernel arguments. Specifies whether kernel argument is used for input or output by compute API kernel function.
- enum ArgumentDataType {
  ArgumentDataType::Char, ArgumentDataType::UnsignedChar, ArgumentDataType::Short, ArgumentData↩Type::UnsignedShort,
  ArgumentDataType::Int, ArgumentDataType::UnsignedInt, ArgumentDataType::Long, ArgumentDataType::↩UnsignedLong,
  ArgumentDataType::Half, ArgumentDataType::Float, ArgumentDataType::Double, ArgumentDataType::↩Custom }

  Enum for data type of kernel arguments. Specifies the data type of elements inside single kernel argument.
- enum ArgumentMemoryLocation { ArgumentMemoryLocation::Device, ArgumentMemoryLocation::Host, ArgumentMemoryLocation::HostZeroCopy }

  Enum for memory location of kernel arguments. Specifies the memory from which the argument data will be accessed by compute API functions and kernels.
- enum ArgumentUploadType { ArgumentUploadType::Scalar, ArgumentUploadType::Vector, Argument↩UploadType::Local }

  Enum for upload type of kernel arguments. Specifies which compute API function should be used internally by KTT library to make the argument accessible to kernel functions.
- enum ComputeApi { ComputeApi::Opencl, ComputeApi::Cuda, ComputeApi::Vulkan }

  Enum for compute API used by KTT library. It is utilized during tuner creation.
- enum DeviceType {
  DeviceType::CPU, DeviceType::GPU, DeviceType::Accelerator, DeviceType::Default,
  DeviceType::Custom }

  Enum for type of a device. It is based on device types supported by OpenCL API.
- enum Dimension { Dimension::X, Dimension::Y, Dimension::Z }

  Enum for dimensions. Dimensions are utilized during specification of parameters which modify kernel thread sizes.
- enum DimensionVectorType { DimensionVectorType::Global, DimensionVectorType::Local }

  Enum for dimension vector type. Specifies whether a single dimension vector holds global or local kernel thread dimensions.
- enum GlobalSizeType { GlobalSizeType::Opencl, GlobalSizeType::Cuda, GlobalSizeType::Vulkan }

  Enum for format of global thread size. Specifies the format of global thread size specified by user during kernel addition.
- enum PrintFormat { PrintFormat::Verbose, PrintFormat::CSV }

  Enum for format of printed results. Specifies the format used during printing of tuning results.
- enum SearchMethod { SearchMethod::FullSearch, SearchMethod::RandomSearch, SearchMethod::PSO, SearchMethod::Annealing }

  Enum for search method used to explore configuration space during kernel tuning.
- enum ThreadModifierAction { ThreadModifierAction::Add, ThreadModifierAction::Subtract, ThreadModifier↩Action::Multiply, ThreadModifierAction::Divide }

  Enum for modifier action for kernel parameters which modify thread size.
- enum ThreadModifierType { ThreadModifierType::None, ThreadModifierType::Global, ThreadModifierType↩::Local }

  Enum for modifier type for kernel parameters. Specifies whether kernel parameter value affects corresponding kernel thread size.
- enum TimeUnit { TimeUnit::Nanoseconds, TimeUnit::Microseconds, TimeUnit::Milliseconds, TimeUnit::↩Seconds }

  Enum for time unit used during printing of kernel results.
- enum ValidationMethod { ValidationMethod::AbsoluteDifference, ValidationMethod::SideBySideComparison, ValidationMethod::SideBySideRelativeComparison }

  Enum for validation method used during validation of floating-point output arguments.

Functions

- KTT_API std::ostream & operator<< (std::ostream &outputTarget, const DeviceInfo &deviceInfo)

  Output operator for device info class.
- KTT_API std::ostream & operator<< (std::ostream &outputTarget, const DimensionVector &dimension↩ Vector)

  Output operator for dimension vector class.
- KTT_API std::ostream & operator<< (std::ostream &outputTarget, const ParameterPair &parameterPair)

  Output operator for parameter pair class.
- KTT_API std::ostream & operator<< (std::ostream &outputTarget, const PlatformInfo &platformInfo)

  Output operator for platform info class.

2.1.1 Detailed Description

All classes, methods and type aliases related to KTT library are located inside ktt namespace.

2.1.2 Enumeration Type Documentation

2.1.2.1 ArgumentAccessType

`enum ktt::ArgumentAccessType [strong]`

Enum for access type of kernel arguments. Specifies whether kernel argument is used for input or output by compute API kernel function.

Enumerator

ReadOnly	Specifies that kernel argument is read-only. Attempting to modify the argument may result in error.
WriteOnly	Specifies that kernel argument is write-only. Attempting to read the argument may result in error.
ReadWrite	Specifies that kernel argument can be both read and modified.

2.1.2.2 ArgumentDataType

`enum ktt::ArgumentDataType [strong]`

Enum for data type of kernel arguments. Specifies the data type of elements inside single kernel argument.

Enumerator

Char	8-bit signed integer type.
UnsignedChar	8-bit unsigned integer type.
Short	16-bit signed integer type.
UnsignedShort	16-bit unsigned integer type.

Enumerator

Int	32-bit signed integer type.
UnsignedInt	32-bit unsigned integer type.
Long	64-bit signed integer type.
UnsignedLong	64-bit unsigned integer type.
Half	16-bit floating-point type.
Float	32-bit floating-point type.
Double	64-bit floating-point type.
Custom	Custom data type, usually defined by user. Custom data type has to be trivially copyable. It can be for example struct or class.

2.1.2.3 ArgumentMemoryLocation

```
enum ktt::ArgumentMemoryLocation  [strong]
```

Enum for memory location of kernel arguments. Specifies the memory from which the argument data will be accessed by compute API functions and kernels.

Enumerator

Device	Argument data will be accessed from device memory. This is recommended setting for devices with dedicated memory, eg. discrete GPUs.
Host	Argument data will be accessed from host memory. This is recommended setting for CPUs and devices without dedicated memory, eg. integrated GPUs.
HostZeroCopy	Argument data will be accessed from host memory without explicitly creating additional compute API buffer. This flag cannot be used for writable arguments during regular kernel tuning. It can be used for any arguments during kernel tuning by step and kernel running. Note that even when this flag is used, extra buffer copy is still sometimes created internally by compute API. This bevaiour depends on particular API and device.

2.1.2.4 ArgumentUploadType

```
enum ktt::ArgumentUploadType  [strong]
```

Enum for upload type of kernel arguments. Specifies which compute API function should be used internally by KTT library to make the argument accessible to kernel functions.

Enumerator

Scalar	Argument will be uploaded as a scalar. Scalar arguments are uploaded into kernel as a local copy.
Vector	Argument will be uploaded as a vector.
Local	Argument will be located in local memory. Kernel arguments cannot be directly transferred into local memory from host memory. Assigning local memory argument to kernel from KTT API simply means that the compute API will allocate enough local memory to hold number of elements specified for the argument.

2.1.2.5 ComputeApi

```
enum ktt::ComputeApi  [strong]
```

Enum for compute API used by KTT library. It is utilized during tuner creation.

Enumerator

Opencl	Tuner will use OpenCL as compute API.
Cuda	Tuner will use CUDA as compute API.
Vulkan	Tuner will use Vulkan as compute API. Vulkan API is not supported by KTT library yet.

2.1.2.6 DeviceType

```
enum ktt::DeviceType  [strong]
```

Enum for type of a device. It is based on device types supported by OpenCL API.

Enumerator

CPU	Device type corresponding to CPU device type in OpenCL.
GPU	Device type corresponding to GPU device type in OpenCL. All available devices in CUDA API will also have this device type.
Accelerator	Device type corresponding to accelerator device type in OpenCL.
Default	Device type corresponding to default device type in OpenCL.
Custom	Device type corresponding to custom device type in OpenCL.

2.1.2.7 Dimension

```
enum ktt::Dimension  [strong]
```

Enum for dimensions. Dimensions are utilized during specification of parameters which modify kernel thread sizes.

Enumerator

X	Kernel parameter will modify thread size in dimension X.
Y	Kernel parameter will modify thread size in dimension Y.
Z	Kernel parameter will modify thread size in dimension Z.

2.1.2.8 DimensionVectorType

```
enum ktt::DimensionVectorType  [strong]
```

Enum for dimension vector type. Specifies whether a single dimension vector holds global or local kernel thread dimensions.

Enumerator

Global	Dimension vector holds global kernel thread dimensions.
Local	Dimension vector holds local kernel thread dimensions.

2.1.2.9 GlobalSizeType

```
enum ktt::GlobalSizeType  [strong]
```

Enum for format of global thread size. Specifies the format of global thread size specified by user during kernel addition.

Enumerator

Opencl	Global thread size uses OpenCL format for NDRange dimensions specification.
Cuda	Global thread size uses CUDA format for grid dimensions specification.
Vulkan	Currently unspecified format.

2.1.2.10 PrintFormat

```
enum ktt::PrintFormat  [strong]
```

Enum for format of printed results. Specifies the format used during printing of tuning results.

Enumerator

Verbose	Format suitable for printing to console or log file.
CSV	Format suitable for printing into CSV file, allows easier data analysis and vizualization.

2.1.2.11 SearchMethod

```
enum ktt::SearchMethod  [strong]
```

Enum for search method used to explore configuration space during kernel tuning.

Enumerator

FullSearch	All kernel configurations will be explored. No additional search parameters are needed.
RandomSearch	Explores random fraction of kernel configurations. The fraction size is controlled with parameter.
PSO	Explores fraction of kernel configurations using particle swarm optimization method. The fraction size is controlled with parameter. Additional parameters specify swarm size and swarm influences.
Annealing	Explores fraction of kernel configurations using simulated annealing method. The fraction size is controlled with parameter. Additional parameter specifies maximum temperature.

2.1.2.12 ThreadModifierAction

```
enum ktt::ThreadModifierAction  [strong]
```

Enum for modifier action for kernel parameters which modify thread size.

Enumerator

Add	Kernel parameter will add its value to corresponding kernel thread size.
Subtract	Kernel parameter will subtract its value from corresponding kernel thread size.
Multiply	Kernel parameter will multiply corresponding kernel thread size by its value.
Divide	Kernel parameter will divide corresponding kernel thread size by its value.

2.1.2.13 ThreadModifierType

```
enum ktt::ThreadModifierType  [strong]
```

Enum for modifier type for kernel parameters. Specifies whether kernel parameter value affects corresponding kernel thread size.

Enumerator

None	Parameter value does not affect any thread sizes of corresponding kernel.
Global	Parameter value affects global thread size of corresponding kernel.
Local	Parameter value affects local thread size of corresponding kernel.

2.1.2.14 TimeUnit

```
enum ktt::TimeUnit  [strong]
```

Enum for time unit used during printing of kernel results.

Enumerator

Nanoseconds	Timings inside kernel results will be printed in nanoseconds.
Microseconds	Timings inside kernel results will be printed in microseconds.
Milliseconds	Timings inside kernel results will be printed in milliseconds.
Seconds	Timings inside kernel results will be printed in seconds.

2.1.2.15 ValidationMethod

```
enum ktt::ValidationMethod  [strong]
```

Enum for validation method used during validation of floating-point output arguments.

Enumerator

AbsoluteDifference	Calculates sum of differences between each pair of elements, then compares the sum to specified threshold.
SideBySideComparison	Calculates difference for each pair of elements, then compares the difference to specified threshold.
SideBySideRelativeComparison	Calculates difference for each pair of elements, then compares the difference divided by reference value to specified threshold.

2.1.3 Function Documentation

2.1.3.1 operator<<() [1/4]

```
std::ostream & ktt::operator<< (
            std::ostream & outputTarget,
            const ParameterPair & parameterPair )
```

Output operator for parameter pair class.

Parameters

outputTarget	Location where information about parameter pair is printed.
parameterPair	Parameter pair object that is printed.

Returns

Output target to support chain of output operations.

2.1.3.2 operator<<() [2/4]

```
std::ostream & ktt::operator<< (
             std::ostream & outputTarget,
             const PlatformInfo & platformInfo )
```

Output operator for platform info class.

Parameters

outputTarget	Location where information about platform is printed.
platformInfo	Platform info object that is printed.

Returns

Output target to support chain of output operations.

2.1.3.3 operator<<() [3/4]

```
std::ostream & ktt::operator<< (
             std::ostream & outputTarget,
             const DimensionVector & dimensionVector )
```

Output operator for dimension vector class.

Parameters

outputTarget	Location where information about dimension vector is printed.
dimensionVector	Dimension vector object that is printed.

Returns

Output target to support chain of output operations.

2.1.3.4 operator<<() [4/4]

```
std::ostream & ktt::operator<< (
             std::ostream & outputTarget,
             const DeviceInfo & deviceInfo )
```

Output operator for device info class.

Parameters

outputTarget	Location where information about device is printed.
deviceInfo	Device info object that is printed.

Returns

Output target to support chain of output operations.

Chapter 3

Class Documentation

3.1 ktt::ArgumentOutputDescriptor Class Reference

Class which can be used to retrieve kernel argument data when calling certain KTT API methods.

`#include <argument_output_descriptor.h>`

Public Member Functions

- ArgumentOutputDescriptor (const ArgumentId id, void ∗outputDestination)

  Constructor, which creates new output descriptor object for specified kernel argument.
- ArgumentOutputDescriptor (const ArgumentId id, void ∗outputDestination, const size_t outputSizeInBytes)

  Constructor, which creates new output descriptor object for specified kernel argument.
- ArgumentId getArgumentId () const

  Getter for id of argument tied to output descriptor.
- void ∗ getOutputDestination () const

  Getter for pointer to destination buffer tied to output descriptor.
- size_t getOutputSizeInBytes () const

  Getter for data size retrieved with output descriptor.

3.1.1 Detailed Description

Class which can be used to retrieve kernel argument data when calling certain KTT API methods.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 ArgumentOutputDescriptor() [1/2]

```
ktt::ArgumentOutputDescriptor::ArgumentOutputDescriptor (
            const ArgumentId id,
            void * outputDestination )  [explicit]
```

Constructor, which creates new output descriptor object for specified kernel argument.

Parameters

id	Id of vector argument which is retrieved.
outputDestination	Pointer to destination where vector argument data will be copied. Destination buffer size needs to be equal or greater than argument size.

3.1.2.2 ArgumentOutputDescriptor() [2/2]

```
ktt::ArgumentOutputDescriptor::ArgumentOutputDescriptor (
            const ArgumentId id,
            void * outputDestination,
            const size_t outputSizeInBytes )  [explicit]
```

Constructor, which creates new output descriptor object for specified kernel argument.

Parameters

id	Id of vector argument which is retrieved.
outputDestination	Pointer to destination where vector argument data will be copied. Destination buffer size needs to be equal or greater than specified output size.
outputSizeInBytes	Size of output in bytes which will be copied to specified destination, starting with first byte in argument.

3.1.3 Member Function Documentation

3.1.3.1 getArgumentId()

```
ArgumentId ktt::ArgumentOutputDescriptor::getArgumentId ( ) const
```

Getter for id of argument tied to output descriptor.

Returns

Id of argument tied to output descriptor.

3.1.3.2 getOutputDestination()

```
void * ktt::ArgumentOutputDescriptor::getOutputDestination ( ) const
```

Getter for pointer to destination buffer tied to output descriptor.

Returns

Pointer to destination buffer tied to output descriptor.

3.1.3.3 getOutputSizeInBytes()

```
size_t ktt::ArgumentOutputDescriptor::getOutputSizeInBytes ( ) const
```

Getter for data size retrieved with output descriptor.

Returns

Data size retrieved with output descriptor. Returns 0 if entire argument is retrieved.

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

- source/api/argument_output_descriptor.h

3.2 ktt::DeviceInfo Class Reference

Class which holds information about a compute API device.

```
#include <device_info.h>
```

Public Member Functions

- DeviceInfo (const size_t id, const std::string &name)

    Constructor, which creates new device info object.
- size_t getId () const

    Getter for id of device assigned by KTT library.
- std::string getName () const

    Getter for name of device retrieved from compute API.
- std::string getVendor () const

    Getter for name of device vendor retrieved from compute API.
- std::string getExtensions () const

    Getter for list of supported device extensions retrieved from compute API.
- DeviceType getDeviceType () const

    Getter for type of device. See DeviceType for more information.
- std::string getDeviceTypeAsString () const

    Getter for type of device converted to string. See DeviceType for more information.
- uint64_t getGlobalMemorySize () const

    Getter for global memory size of device retrieved from compute API.
- uint64_t getLocalMemorySize () const

    Getter for local memory (shared memory in CUDA) size of device retrieved from compute API.
- uint64_t getMaxConstantBufferSize () const

    Getter for constant memory size of device retrieved from compute API.
- uint32_t getMaxComputeUnits () const

    Getter for maximum parallel compute units (multiprocessors in CUDA) count of device retrieved from compute API.
- size_t getMaxWorkGroupSize () const

    Getter for maximum work-group (thread block in CUDA) size of device retrieved from compute API.
- void setVendor (const std::string &vendor)

    Setter for name of device vendor.
- void setExtensions (const std::string &extensions)

Setter for list of supported device extensions.
- void setDeviceType (const DeviceType &deviceType)

  Setter for type of device.
- void setGlobalMemorySize (const uint64_t globalMemorySize)

  Setter for global memory size of device.
- void setLocalMemorySize (const uint64_t localMemorySize)

  Setter for local memory size of device.
- void setMaxConstantBufferSize (const uint64_t maxConstantBufferSize)

  Setter for constant memory size of device.
- void setMaxComputeUnits (const uint32_t maxComputeUnits)

  Setter for maximum compute units count of device.
- void setMaxWorkGroupSize (const size_t maxWorkGroupSize)

  Setter for maximum work-group size of device.

3.2.1 Detailed Description

Class which holds information about a compute API device.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 DeviceInfo()

```
ktt::DeviceInfo::DeviceInfo (
            const size_t id,
            const std::string & name )  [explicit]
```

Constructor, which creates new device info object.

Parameters

id	Id of device assigned by KTT library.
name	Name of device retrieved from compute API.

3.2.3 Member Function Documentation

3.2.3.1 getDeviceType()

```
DeviceType ktt::DeviceInfo::getDeviceType ( ) const
```

Getter for type of device. See DeviceType for more information.

Returns

Type of device.

3.2.3.2 getDeviceTypeAsString()

```
std::string ktt::DeviceInfo::getDeviceTypeAsString ( ) const
```

Getter for type of device converted to string. See DeviceType for more information.

Returns

Type of device converted to string.

3.2.3.3 getExtensions()

```
std::string ktt::DeviceInfo::getExtensions ( ) const
```

Getter for list of supported device extensions retrieved from compute API.

Returns

List of supported device extensions retrieved from compute API.

3.2.3.4 getGlobalMemorySize()

```
uint64_t ktt::DeviceInfo::getGlobalMemorySize ( ) const
```

Getter for global memory size of device retrieved from compute API.

Returns

Global memory size of device retrieved from compute API.

3.2.3.5 getId()

```
size_t ktt::DeviceInfo::getId ( ) const
```

Getter for id of device assigned by KTT library.

Returns

Id of device assigned by KTT library.

3.2.3.6 getLocalMemorySize()

```
uint64_t ktt::DeviceInfo::getLocalMemorySize ( ) const
```

Getter for local memory (shared memory in CUDA) size of device retrieved from compute API.

Returns

Local memory size of device retrieved from compute API.

3.2.3.7 getMaxComputeUnits()

```
uint32_t ktt::DeviceInfo::getMaxComputeUnits ( ) const
```

Getter for maximum parallel compute units (multiprocessors in CUDA) count of device retrieved from compute API.

Returns

Maximum parallel compute units count of device retrieved from compute API.

3.2.3.8 getMaxConstantBufferSize()

```
uint64_t ktt::DeviceInfo::getMaxConstantBufferSize ( ) const
```

Getter for constant memory size of device retrieved from compute API.

Returns

Constant memory size of device retrieved from compute API.

3.2.3.9 getMaxWorkGroupSize()

```
size_t ktt::DeviceInfo::getMaxWorkGroupSize ( ) const
```

Getter for maximum work-group (thread block in CUDA) size of device retrieved from compute API.

Returns

Maximum work-group size of device retrieved from compute API.

3.2.3.10 getName()

```
std::string ktt::DeviceInfo::getName ( ) const
```

Getter for name of device retrieved from compute API.

Returns

Name of device retrieved from compute API.

3.2.3.11 getVendor()

```
std::string ktt::DeviceInfo::getVendor ( ) const
```

Getter for name of device vendor retrieved from compute API.

Returns

Name of device vendor retrieved from compute API.

3.2.3.12 setDeviceType()

```
void ktt::DeviceInfo::setDeviceType (
             const DeviceType & deviceType )
```

Setter for type of device.

Parameters

deviceType	Type of device.

3.2.3.13 setExtensions()

```
void ktt::DeviceInfo::setExtensions (
             const std::string & extensions )
```

Setter for list of supported device extensions.

Parameters

extensions	List of supported device extensions.

3.2.3.14 setGlobalMemorySize()

```
void ktt::DeviceInfo::setGlobalMemorySize (
             const uint64_t globalMemorySize )
```

Setter for global memory size of device.

Parameters

globalMemorySize	Global memory size of device.

3.2.3.15 setLocalMemorySize()

```
void ktt::DeviceInfo::setLocalMemorySize (
             const uint64_t localMemorySize )
```

Setter for local memory size of device.

Parameters

localMemorySize	Local memory size of device.

3.2.3.16 setMaxComputeUnits()

```
void ktt::DeviceInfo::setMaxComputeUnits (
             const uint32_t maxComputeUnits )
```

Setter for maximum compute units count of device.

Parameters

maxComputeUnits	Maximum compute units count of device.

3.2.3.17 setMaxConstantBufferSize()

```
void ktt::DeviceInfo::setMaxConstantBufferSize (
             const uint64_t maxConstantBufferSize )
```

Setter for constant memory size of device.

Parameters

maxConstantBufferSize	Constant memory size of device.

3.2.3.18 setMaxWorkGroupSize()

```
void ktt::DeviceInfo::setMaxWorkGroupSize (
             const size_t maxWorkGroupSize )
```

Setter for maximum work-group size of device.

Parameters

maxWorkGroupSize	Maximum work-group size of device.

3.2.3.19 setVendor()

```
void ktt::DeviceInfo::setVendor (
             const std::string & vendor )
```

Setter for name of device vendor.

Parameters

vendor	Name of device vendor.

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

- source/api/device_info.h

3.3 ktt::DimensionVector Class Reference

Class which holds information about either global or local thread size of a single kernel.

```
#include <dimension_vector.h>
```

Public Member Functions

- DimensionVector ()

  Default constructor, creates dimension vector with thread sizes in all dimensions set to 1.
- DimensionVector (const size_t sizeX)

Constructor which creates dimension vector with specified thread size in dimension x and thread sizes in other dimensions set to 1.

- DimensionVector (const size_t sizeX, const size_t sizeY)

  Constructor which creates dimension vector with specified thread sizes in dimensions x and y and thread size in dimension z set to 1.
- DimensionVector (const size_t sizeX, const size_t sizeY, const size_t sizeZ)

  Constructor which creates dimension vector with specified thread sizes in all dimensions.
- DimensionVector (const std::vector< size_t > &vector)

  Constructor which creates dimension vector with thread sizes based on up to first three elements of provided vector. If size of vector is less than 3, remaining thread sizes are set to 1.
- void setSizeX (const size_t sizeX)

  Setter for thread size in dimension x.
- void setSizeY (const size_t sizeY)

  Setter for thread size in dimension y.
- void setSizeZ (const size_t sizeZ)

  Setter for thread size in dimension z.
- void multiply (const DimensionVector &factor)

  Multiplies thread sizes by values provided by specified dimension vector.
- void divide (const DimensionVector &divisor)

  Divides thread sizes by values provided by specified dimension vector.
- void modifyByValue (const size_t value, const ThreadModifierAction &modifierAction, const Dimension modifierDimension)

  Modifies thread size in single dimension based on provided value and action.
- size_t getSizeX () const

  Getter for thread size in dimension x.
- size_t getSizeY () const

  Getter for thread size in dimension y.
- size_t getSizeZ () const

  Getter for thread size in dimension z.
- size_t getTotalSize () const

  Getter for total thread size. Total thread size is calculated by multiplying thread sizes in each dimension.
- std::vector< size_t > getVector () const

  Converts dimension vector to STL vector. Resulting vector will always contain 3 elements.
- bool operator== (const DimensionVector &other) const

  Comparison operator for dimension vector. Compares thread sizes in all 3 dimensions.
- bool operator!= (const DimensionVector &other) const

  Comparison operator for dimension vector. Compares thread sizes in all 3 dimensions.

3.3.1 Detailed Description

Class which holds information about either global or local thread size of a single kernel.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 DimensionVector() [1/4]

```
ktt::DimensionVector::DimensionVector (
            const size_t sizeX )  [explicit]
```

Constructor which creates dimension vector with specified thread size in dimension x and thread sizes in other dimensions set to 1.

Parameters

sizeX	Thread size in dimension x.

3.3.2.2 DimensionVector() [2/4]

```
ktt::DimensionVector::DimensionVector (
            const size_t sizeX,
            const size_t sizeY )  [explicit]
```

Constructor which creates dimension vector with specified thread sizes in dimensions x and y and thread size in dimension z set to 1.

Parameters

sizeX	Thread size in dimension x.
sizeY	Thread size in dimension y.

3.3.2.3 DimensionVector() [3/4]

```
ktt::DimensionVector::DimensionVector (
            const size_t sizeX,
            const size_t sizeY,
            const size_t sizeZ )  [explicit]
```

Constructor which creates dimension vector with specified thread sizes in all dimensions.

Parameters

sizeX	Thread size in dimension x.
sizeY	Thread size in dimension y.
sizeZ	Thread size in dimension z.

3.3.2.4 DimensionVector() [4/4]

```
ktt::DimensionVector::DimensionVector (
            const std::vector< size_t > & vector )  [explicit]
```

Constructor which creates dimension vector with thread sizes based on up to first three elements of provided vector. If size of vector is less than 3, remaining thread sizes are set to 1.

Parameters

vector	Source vector for dimension vector thread sizes.

3.3.3 Member Function Documentation

3.3.3.1 divide()

```
void ktt::DimensionVector::divide (
            const DimensionVector & divisor )
```

Divides thread sizes by values provided by specified dimension vector.

Parameters

divisor	Source of values for thread size division.

3.3.3.2 getSizeX()

```
size_t ktt::DimensionVector::getSizeX ( ) const
```

Getter for thread size in dimension x.

Returns

Thread size in dimension x.

3.3.3.3 getSizeY()

```
size_t ktt::DimensionVector::getSizeY ( ) const
```

Getter for thread size in dimension y.

Returns

Thread size in dimension y.

3.3.3.4 getSizeZ()

```
size_t ktt::DimensionVector::getSizeZ ( ) const
```

Getter for thread size in dimension z.

Returns

Thread size in dimension z.

3.3.3.5 getTotalSize()

```
size_t ktt::DimensionVector::getTotalSize ( ) const
```

Getter for total thread size. Total thread size is calculated by multiplying thread sizes in each dimension.

Returns

Total thread size.

3.3.3.6 getVector()

```
std::vector< size_t > ktt::DimensionVector::getVector ( ) const
```

Converts dimension vector to STL vector. Resulting vector will always contain 3 elements.

Returns

Converted STL vector.

3.3.3.7 modifyByValue()

```
void ktt::DimensionVector::modifyByValue (
             const size_t value,
             const ThreadModifierAction & modifierAction,
             const Dimension modifierDimension )
```

Modifies thread size in single dimension based on provided value and action.

Parameters

value	Value which will modifies thread size in single dimension based on specified action.
modifierAction	Specifies which operation should be performed with thread size and specified value.
modifierDimension	Specifies which dimension will be affected by the action.

3.3.3.8 multiply()

```
void ktt::DimensionVector::multiply (
             const DimensionVector & factor )
```

Multiplies thread sizes by values provided by specified dimension vector.

Parameters

factor	Source of values for thread size multiplication.

3.3.3.9 operator"!=()

```
bool ktt::DimensionVector::operator!= (
             const DimensionVector & other ) const
```

Comparison operator for dimension vector. Compares thread sizes in all 3 dimensions.

Returns

True if dimension vectors are not equal. False otherwise.

3.3.3.10 operator==()

```
bool ktt::DimensionVector::operator== (
             const DimensionVector & other ) const
```

Comparison operator for dimension vector. Compares thread sizes in all 3 dimensions.

Returns

True if dimension vectors are equal. False otherwise.

3.3.3.11 setSizeX()

```
void ktt::DimensionVector::setSizeX (
             const size_t sizeX )
```

Setter for thread size in dimension x.

Parameters

sizeX	Thread size in dimension x.

3.3.3.12 setSizeY()

```
void ktt::DimensionVector::setSizeY (
             const size_t sizeY )
```

Setter for thread size in dimension y.

Parameters

sizeY	Thread size in dimension y.

3.3.3.13 setSizeZ()

```
void ktt::DimensionVector::setSizeZ (
             const size_t sizeZ )
```

Setter for thread size in dimension z.

Parameters

sizeZ	Thread size in dimension z.

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

- source/api/dimension_vector.h

3.4 ktt::ParameterPair Class Reference

Class which holds single value for one kernel parameter.

```
#include <parameter_pair.h>
```

Public Member Functions

- ParameterPair ()

  Default constructor, creates parameter pair with empty name and value set to zero.
- ParameterPair (const std::string &name, const size_t value)

Constructor which creates parameter pair for integer parameter.
- ParameterPair (const std::string &name, const double value)

  Constructor which creates parameter pair for floating-point parameter.
- void setValue (const size_t value)

  Setter for value of an integer parameter.
- std::string getName () const

  Returns name of a parameter.
- size_t getValue () const

  Returns integer representation of parameter value.
- double getValueDouble () const

  Returns floating-point representation of parameter value.
- bool hasValueDouble () const

  Checks if parameter value was specified as floating-point.

3.4.1 Detailed Description

Class which holds single value for one kernel parameter.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 ParameterPair() [1/2]

```
ktt::ParameterPair::ParameterPair (
            const std::string & name,
            const size_t value )  [explicit]
```

Constructor which creates parameter pair for integer parameter.

Parameters

name	Name of a parameter.
value	Value of a parameter.

3.4.2.2 ParameterPair() [2/2]

```
ktt::ParameterPair::ParameterPair (
            const std::string & name,
            const double value )  [explicit]
```

Constructor which creates parameter pair for floating-point parameter.

Parameters

name	Name of a parameter.
value	Value of a parameter.

3.4.3 Member Function Documentation

3.4.3.1 getName()

```
std::string ktt::ParameterPair::getName ( ) const
```

Returns name of a parameter.

Returns

Name of a parameter.

3.4.3.2 getValue()

```
size_t ktt::ParameterPair::getValue ( ) const
```

Returns integer representation of parameter value.

Returns

Integer representation of parameter value.

3.4.3.3 getValueDouble()

```
double ktt::ParameterPair::getValueDouble ( ) const
```

Returns floating-point representation of parameter value.

Returns

Floating-point representation of parameter value.

3.4.3.4 hasValueDouble()

```
bool ktt::ParameterPair::hasValueDouble ( ) const
```

Checks if parameter value was specified as floating-point.

Returns

True if parameter value was specified as floating-point, false otherwise.

3.4.3.5 setValue()

```
void ktt::ParameterPair::setValue (
             const size_t value )
```

Setter for value of an integer parameter.

Parameters

value	New value of an integer parameter.

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

- source/api/parameter_pair.h

3.5 ktt::PlatformInfo Class Reference

Class which holds information about a compute API platform.

```
#include <platform_info.h>
```

Public Member Functions

- PlatformInfo (const size_t id, const std::string &name)

  Constructor, which creates new platform info object.
- size_t getId () const

  Getter for id of platform assigned by KTT library.
- std::string getName () const

  Getter for name of platform retrieved from compute API.
- std::string getVendor () const

  Getter for name of platform vendor retrieved from compute API.
- std::string getVersion () const

  Getter for platform version retrieved from compute API.
- std::string getExtensions () const

  Getter for list of supported platform extensions retrieved from compute API.

- void setVendor (const std::string &vendor)

  Setter for name of platform vendor.
- void setVersion (const std::string &version)

  Setter for platform version.
- void setExtensions (const std::string &extensions)

  Setter for list of supported platform extensions.

3.5.1 Detailed Description

Class which holds information about a compute API platform.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 PlatformInfo()

```
ktt::PlatformInfo::PlatformInfo (
            const size_t id,
            const std::string & name )  [explicit]
```

Constructor, which creates new platform info object.

Parameters

id	Id of platform assigned by KTT library.
name	Name of platform retrieved from compute API.

3.5.3 Member Function Documentation

3.5.3.1 getExtensions()

```
std::string ktt::PlatformInfo::getExtensions ( ) const
```

Getter for list of supported platform extensions retrieved from compute API.

Returns

List of supported platform extensions retrieved from compute API.

3.5.3.2 getId()

```
size_t ktt::PlatformInfo::getId ( ) const
```

Getter for id of platform assigned by KTT library.

Returns

Id of platform assigned by KTT library.

3.5.3.3 getName()

```
std::string ktt::PlatformInfo::getName ( ) const
```

Getter for name of platform retrieved from compute API.

Returns

Name of platform retrieved from compute API.

3.5.3.4 getVendor()

```
std::string ktt::PlatformInfo::getVendor ( ) const
```

Getter for name of platform vendor retrieved from compute API.

Returns

Name of platform vendor retrieved from compute API.

3.5.3.5 getVersion()

```
std::string ktt::PlatformInfo::getVersion ( ) const
```

Getter for platform version retrieved from compute API.

Returns

Platform version retrieved from compute API.

3.5.3.6 setExtensions()

```
void ktt::PlatformInfo::setExtensions (
            const std::string & extensions )
```

Setter for list of supported platform extensions.

Parameters

extensions	List of supported platform extensions.

3.5.3.7 setVendor()

```
void ktt::PlatformInfo::setVendor (
             const std::string & vendor )
```

Setter for name of platform vendor.

Parameters

vendor	Name of platform vendor.

3.5.3.8 setVersion()

```
void ktt::PlatformInfo::setVersion (
             const std::string & version )
```

Setter for platform version.

Parameters

version	Platform version.

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

- source/api/platform_info.h

3.6 ktt::ReferenceClass Class Reference

Class which can be used to compute reference output for selected kernel arguments inside regular C++ method. In order to use this functionality, new class which publicly inherits from reference class has to be defined.

```
#include <reference_class.h>
```

Public Member Functions

- virtual ∼ReferenceClass ()=default

Reference class destructor. Inheriting class can override destructor with custom implementation. Default implementation is provided by KTT library.

- virtual void computeResult ()=0

  Computes reference output for all kernel arguments validated by the class and stores it for later retrieval by tuner. Inheriting class must provide implementation for this method.
- virtual void ∗ getData (const ArgumentId id)=0

  Returns pointer to buffer containing reference output for specified kernel argument. This method will be called only after running computeResult() method. It can be called multiple times for same kernel argument. Inheriting class must provide implementation for this method.
- virtual size_t getNumberOfElements (const ArgumentId id) const

  Returns number of validated elements returned by getData() method for specified kernel argument. This method will be called only after running computeResult() method. It can be called multiple times for same kernel argument. Inheriting class can override this method, which is useful in conjuction with Tuner::setValidationRange() method. If number of validated elements equals zero, all elements in corresponding kernel argument will be validated.

3.6.1 Detailed Description

Class which can be used to compute reference output for selected kernel arguments inside regular C++ method. In order to use this functionality, new class which publicly inherits from reference class has to be defined.

3.6.2 Member Function Documentation

3.6.2.1 getData()

```
void * ktt::ReferenceClass::getData (
            const ArgumentId id )  [pure virtual]
```

Returns pointer to buffer containing reference output for specified kernel argument. This method will be called only after running computeResult() method. It can be called multiple times for same kernel argument. Inheriting class must provide implementation for this method.

Parameters

id	Id of kernel argument for which reference output is retrieved. This can be used by inheriting class to support validation of multiple kernel arguments.

Returns

Pointer to buffer containing reference output for specified kernel argument.

3.6.2.2 getNumberOfElements()

```
size_t ktt::ReferenceClass::getNumberOfElements (
            const ArgumentId id ) const  [inline], [virtual]
```

Returns number of validated elements returned by getData() method for specified kernel argument. This method will be called only after running computeResult() method. It can be called multiple times for same kernel argument. Inheriting class can override this method, which is useful in conjuction with Tuner::setValidationRange() method. If number of validated elements equals zero, all elements in corresponding kernel argument will be validated.

Parameters

id	Id of kernel argument for which number of validated elements is retrieved. This can be used by inheriting class to support validation of multiple kernel arguments.

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

- source/api/reference_class.h

3.7 ktt::Tuner Class Reference

Class which serves as the main part of public API for KTT library.

```
#include <tuner_api.h>
```

Public Member Functions

- Tuner (const size_t platformIndex, const size_t deviceIndex)

  Constructor, which creates new tuner object for specified platform and device. Tuner uses OpenCL as compute API. Indices for available platforms and devices can be retrieved by calling printComputeApiInfo() method.
- Tuner (const size_t platformIndex, const size_t deviceIndex, const ComputeApi &computeApi)

  Constructor, which creates new tuner object for specified platform, device and compute API. Indices for available platforms and devices can be retrieved by calling printComputeApiInfo() method. If specified compute API is CUDA, platform index is ignored.
- ∼Tuner ()

  Tuner destructor.
- KernelId addKernel (const std::string &source, const std::string &kernelName, const DimensionVector &globalSize, const DimensionVector &localSize)

  Adds new kernel to tuner from source inside string. Requires specification of kernel name and default global and local thread sizes.
- KernelId addKernelFromFile (const std::string &filePath, const std::string &kernelName, const Dimension↩ Vector &globalSize, const DimensionVector &localSize)

  Adds new kernel to tuner from file. Requires specification of kernel name and default global and local thread sizes.
- void setKernelArguments (const KernelId id, const std::vector< ArgumentId > &argumentIds)

  Sets kernel arguments for specified kernel by providing corresponding argument ids.
- void addParameter (const KernelId id, const std::string &parameterName, const std::vector< size_t > &parameterValues)

  Adds new integer parameter for specified kernel, providing parameter name and list of allowed values. When the corresponding kernel is launched, parameters will be added to kernel source code as preprocessor definitions. During the tuning process, tuner will generate configurations for combinations of kernel parameters and their values.
- void addParameterDouble (const KernelId id, const std::string &parameterName, const std::vector< double > &parameterValues)

  Adds new floating-point parameter for specified kernel, providing parameter name and list of allowed values. When the corresponding kernel is launched, parameters will be added to kernel source code as preprocessor definitions. During the tuning process, tuner will generate configurations for combinations of kernel parameters and their values.

- void addParameter (const KernelId id, const std::string &parameterName, const std::vector< size_t > &parameterValues, const ThreadModifierType &modifierType, const ThreadModifierAction &modifierAction, const Dimension &modifierDimension)

  Adds new integer parameter for specified kernel, providing parameter name and list of allowed values. When the corresponding kernel is launched, parameters will be added to kernel source code as preprocessor definitions. During the tuning process, tuner will generate configurations for combinations of kernel parameters and their values.
- void addConstraint (const KernelId id, const std::function< bool(std::vector< size_t >)> &constraintFunction, const std::vector< std::string > &parameterNames)

  Adds new constraint for specified kernel. Constraints are used to prevent generating of invalid configurations (eg. conflicting parameter values).
- void setTuningManipulator (const KernelId id, std::unique_ptr< TuningManipulator > manipulator)

  Sets tuning manipulator for specified kernel. Tuning manipulator enables customization of kernel execution. This is useful in several cases, eg. running part of the computation in C++ code, utilizing iterative kernel launches or composite kernels. See TuningManipulator for more information.
- KernelId addComposition (const std::string &compositionName, const std::vector< KernelId > &kernelIds, std::unique_ptr< TuningManipulator > manipulator)

  Creates a kernel composition using specified kernels. Following methods can be used with kernel compositions and will call the corresponding method for all kernels inside the composition: setKernelArguments(), addParameter() (both versions), addConstraint().
- void addCompositionKernelParameter (const KernelId compositionId, const KernelId kernelId, const std↩ ::string &parameterName, const std::vector< size_t > &parameterValues, const ThreadModifierType &modifierType, const ThreadModifierAction &modifierAction, const Dimension &modifierDimension)

  Calls addParameter() method (version with thread modifier) for a single kernel inside specified kernel composition. Does not affect standalone kernels or other compositions.
- void setCompositionKernelArguments (const KernelId compositionId, const KernelId kernelId, const std↩ ::vector< ArgumentId > &argumentIds)

  Calls setKernelArguments() method for a single kernel inside specified kernel composition. Does not affect standalone kernels or other compositions.
- template<typename T >
  ArgumentId addArgumentVector (const std::vector< T > &data, const ArgumentAccessType &accessType)

  Adds new vector argument to tuner. Makes copy of argument data, so the source data vector remains unaffected by tuner operations. Argument data will be accessed from device memory during its usage by compute API.
- template<typename T >
  ArgumentId addArgumentVector (std::vector< T > &data, const ArgumentAccessType &accessType, const ArgumentMemoryLocation &memoryLocation, const bool copyData)

  Adds new vector argument to tuner. Allows choice for argument memory location and whether argument data is copied to tuner.
- template<typename T >
  ArgumentId addArgumentScalar (const T &data)

  Adds new scalar argument to tuner. All scalar arguments are read-only.
- template<typename T >
  ArgumentId addArgumentLocal (const size_t localMemoryElementsCount)

  Adds new local memory (shared memory in CUDA) argument to tuner. All local memory arguments are read-only and cannot be initialized from host memory. In case of CUDA API usage, local memory arguments cannot be directly set as kernel function arguments. Setting a local memory argument to kernel in CUDA means that corresponding amount of memory will be allocated for kernel to use. In that case, all local memory argument ids should be specified at the end of the vector when calling setKernelArguments() method.
- void tuneKernel (const KernelId id)

  Starts the tuning process for specified kernel. Creates configuration space based on combinations of provided kernel parameters and constraints. The configurations will be launched in order that depends on specified SearchMethod.
- void tuneKernelByStep (const KernelId id, const std::vector< ArgumentOutputDescriptor > &output)

  Performs one step of the tuning process for specified kernel. When this method is called inside tuner for the first time, creates configuration space based on combinations of provided kernel parameters and constraints. Each time this method is called, launches single kernel configuration. If all configurations were already tested, runs kernel using the best configuration. Output data can be retrieved by providing output descriptors.

- void runKernel (const KernelId id, const std::vector< ParameterPair > &configuration, const std::vector< ArgumentOutputDescriptor > &output)

  Runs specified kernel using provided configuration. Does not perform result validation.
- void setSearchMethod (const SearchMethod &method, const std::vector< double > &arguments)

  Specifies search method which will be used during kernel tuning. Number of required search arguments depends on the search method. Default search method is full search, which requires no search arguments.
- void setPrintingTimeUnit (const TimeUnit &unit)

  Sets time unit used during printing of results inside printResult() methods. Default time unit is microseconds.
- void setInvalidResultPrinting (const bool flag)

  Toggles printing of results from failed kernel runs. Invalid results will be separated from valid results during printing. Printing of invalid results is disabled by default.
- void printResult (const KernelId id, std::ostream &outputTarget, const PrintFormat &format) const

  Prints tuning results for specified kernel to specified output stream. Valid results will be printed only if methods tuneKernel() or tuneKernelByStep() were already called for corresponding kernel.
- void printResult (const KernelId id, const std::string &filePath, const PrintFormat &format) const

  Prints tuning results for specified kernel to specified file. Valid results will be printed only if methods tuneKernel() or tuneKernelByStep() were already called for corresponding kernel.
- std::vector< ParameterPair > getBestConfiguration (const KernelId id) const

  Returns the best configuration found for specified kernel. Valid configuration will be returned only if methods tune↩ Kernel() or tuneKernelByStep() were already called for corresponding kernel.
- void setReferenceKernel (const KernelId id, const KernelId referenceId, const std::vector< ParameterPair > &referenceConfiguration, const std::vector< ArgumentId > &validatedArgumentIds)

  Sets reference kernel for specified kernel. Reference kernel output will be compared to tuned kernel output in order to ensure correctness of computation. Reference kernel uses only single configuration and cannot be composite.
- void setReferenceClass (const KernelId id, std::unique_ptr< ReferenceClass > referenceClass, const std↩ ::vector< ArgumentId > &validatedArgumentIds)

  Sets reference class for specified kernel. Reference class output will be compared to tuned kernel output in order to ensure correctness of computation.
- void setValidationMethod (const ValidationMethod &method, const double toleranceThreshold)

  Sets validation method and tolerance threshold for floating-point argument validation. Default validation method is side by side comparison. Default tolerance threshold is 1e-4.
- void setValidationRange (const ArgumentId id, const size_t range)

  Sets validation range for specified argument to specified validation range. Only elements within validation range, starting with the first element, will be validated. All elements are validated by default.
- void setArgumentComparator (const ArgumentId id, const std::function< bool(const void ∗, const void ∗)> &comparator)

  Sets argument comparator for specified kernel argument. Arguments with custom data type cannot be compared using built-in comparison operators and require user to provide a comparator. Comparator can also be optionally added for arguments with built-in data types.
- void setCompilerOptions (const std::string &options)

  Sets compute API compiler options to specified options. There are no default options for OpenCL back-end. Default option for CUDA back-end is "--gpu-architecture=compute_30".
- void printComputeApiInfo (std::ostream &outputTarget) const

  Prints basic information about available platforms and devices to specified output stream. Also prints indices assigned to them by KTT library.
- std::vector< PlatformInfo > getPlatformInfo () const

  Retrieves detailed information about all available platforms (eg. platform name, vendor). See PlatformInfo for more information.
- std::vector< DeviceInfo > getDeviceInfo (const size_t platformIndex) const

  Retrieves detailed information about all available devices (eg. device name, memory capacity) on specified platform. See DeviceInfo for more information.
- DeviceInfo getCurrentDeviceInfo () const

  Retrieves detailed information about device (eg. device name, memory capacity) used by the tuner. See DeviceInfo for more information.

- void setAutomaticGlobalSizeCorrection (const bool flag)

    Toggles automatic correction for global size, which ensures that global size in each dimension is always a multiple of local size in corresponding dimension. Performs a roundup to the nearest higher multiple. Automatic global size correction is disabled by default.
- void setGlobalSizeType (const GlobalSizeType &type)

    Sets global size specification type to specified compute API style. In OpenCL, NDrange size is specified as number of work-items in a work-group multiplied by number of work-groups. In CUDA, grid size is specified as number of threads in a block divided by number of blocks. This method makes it possible to use OpenCL style in CUDA and vice versa. Default global size type is the one corresponding to compute API of the tuner.
- void setLoggingTarget (std::ostream &outputTarget)

    Sets the target for info messages logging to specified output stream. Default logging target is `std::clog`.
- void setLoggingTarget (const std::string &filePath)

    Sets the target for info messages logging to specified file. Default logging target is `std::clog`.

3.7.1 Detailed Description

Class which serves as the main part of public API for KTT library.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 Tuner() [1/2]

```
ktt::Tuner::Tuner (
            const size_t platformIndex,
            const size_t deviceIndex )  [explicit]
```

Constructor, which creates new tuner object for specified platform and device. Tuner uses OpenCL as compute API. Indices for available platforms and devices can be retrieved by calling printComputeApiInfo() method.

Parameters

platformIndex	Index for platform used by created tuner.
deviceIndex	Index for device used by created tuner.

3.7.2.2 Tuner() [2/2]

```
ktt::Tuner::Tuner (
            const size_t platformIndex,
            const size_t deviceIndex,
            const ComputeApi & computeApi )  [explicit]
```

Constructor, which creates new tuner object for specified platform, device and compute API. Indices for available platforms and devices can be retrieved by calling printComputeApiInfo() method. If specified compute API is CUDA, platform index is ignored.

Parameters

platformIndex	Index for platform used by created tuner.
deviceIndex	Index for device used by created tuner.
computeApi	Compute API used by created tuner.

3.7.3 Member Function Documentation

3.7.3.1 addArgumentLocal()

```
template<typename T >
template< typename T > ArgumentId ktt::Tuner::addArgumentLocal (
             const size_t localMemoryElementsCount )  [inline]
```

Adds new local memory (shared memory in CUDA) argument to tuner. All local memory arguments are read-only and cannot be initialized from host memory. In case of CUDA API usage, local memory arguments cannot be directly set as kernel function arguments. Setting a local memory argument to kernel in CUDA means that corresponding amount of memory will be allocated for kernel to use. In that case, all local memory argument ids should be specified at the end of the vector when calling setKernelArguments() method.

Parameters

localMemoryElementsCount	Specifies how many elements of provided data type the argument contains.

Returns

Id assigned to kernel argument by tuner. The id can be used in other API methods.

3.7.3.2 addArgumentScalar()

```
template<typename T >
template< typename T > ArgumentId ktt::Tuner::addArgumentScalar (
             const T & data )  [inline]
```

Adds new scalar argument to tuner. All scalar arguments are read-only.

Parameters

data	Argument data provided as single scalar value. The data type must be trivially copyable. Bool data type is currently not supported.

Returns

Id assigned to kernel argument by tuner. The id can be used in other API methods.

3.7.3.3 addArgumentVector() [1/2]

```
template<typename T >
template< typename T > ArgumentId ktt::Tuner::addArgumentVector (
             const std::vector< T > & data,
             const ArgumentAccessType & accessType )  [inline]
```

Adds new vector argument to tuner. Makes copy of argument data, so the source data vector remains unaffected by tuner operations. Argument data will be accessed from device memory during its usage by compute API.

Parameters

data	Argument data provided in std::vector. Provided data type must be trivially copyable. Bool data type is currently not supported.
accessType	Access type of argument specifies whether argument is used for input or output. See ArgumentAccessType for more information.

Returns

Id assigned to kernel argument by tuner. The id can be used in other API methods.

3.7.3.4 addArgumentVector() [2/2]

```
template<typename T >
template< typename T > ArgumentId ktt::Tuner::addArgumentVector (
             std::vector< T > & data,
             const ArgumentAccessType & accessType,
             const ArgumentMemoryLocation & memoryLocation,
             const bool copyData )  [inline]
```

Adds new vector argument to tuner. Allows choice for argument memory location and whether argument data is copied to tuner.

Parameters

data	Argument data provided in std::vector. Provided data type must be trivially copyable. Bool data type is currently not supported.
accessType	Access type of argument specifies whether argument is used for input or output. See ArgumentAccessType for more information.
memoryLocation	Memory location of argument specifies whether argument will be accessed from device or host memory during its usage by compute API. See ArgumentMemoryLocation for more information.
copyData	Flag which specifies whether the argument is copied inside tuner. If set to false, tuner will store reference of source data vector and will access it directly during kernel launch operations. This results in lower memory overhead, but relies on a user to keep data in source vector valid.

Returns

Id assigned to kernel argument by tuner. The id can be used in other API methods.

3.7.3.5 addComposition()

```
KernelId ktt::Tuner::addComposition (
            const std::string & compositionName,
            const std::vector< KernelId > & kernelIds,
            std::unique_ptr< TuningManipulator > manipulator )
```

Creates a kernel composition using specified kernels. Following methods can be used with kernel compositions and will call the corresponding method for all kernels inside the composition: setKernelArguments(), addParameter() (both versions), addConstraint().

Kernel compositions do not inherit any parameters or constraints from the original kernels. Setting kernel arguments and adding parameters or constraints to kernels inside given composition will not affect the original kernels or other compositions. Tuning manipulator is required in order to launch kernel composition with tuner. See Tuning↩ Manipulator for more information.

Parameters

compositionName	Name of kernel composition. The name is used during output printing.
kernelIds	Ids of kernels which will be included in the composition.
manipulator	Tuning manipulator for the composition.

Returns

Id assigned to kernel composition by tuner. The id can be used in other API methods.

3.7.3.6 addCompositionKernelParameter()

```
void ktt::Tuner::addCompositionKernelParameter (
            const KernelId compositionId,
            const KernelId kernelId,
            const std::string & parameterName,
            const std::vector< size_t > & parameterValues,
            const ThreadModifierType & modifierType,
            const ThreadModifierAction & modifierAction,
            const Dimension & modifierDimension )
```

Calls addParameter() method (version with thread modifier) for a single kernel inside specified kernel composition. Does not affect standalone kernels or other compositions.

Parameters

compositionId	Id of composition which includes the specified kernel.
kernelId	Id of kernel inside the composition for which the parameter is added.

Parameters

parameterName	Name of a parameter. Parameter names for single kernel must be unique.
parameterValues	Vector of allowed values for the parameter.
modifierType	Type of thread modifier. See ThreadModifierType for more information.
modifierAction	Action of thread modifier. See ThreadModifierAction for more information.
modifierDimension	Dimension which will be affected by thread modifier. See Dimension for more information.

3.7.3.7 addConstraint()

```
void ktt::Tuner::addConstraint (
            const KernelId id,
            const std::function< bool(std::vector< size_t >)> & constraintFunction,
            const std::vector< std::string > & parameterNames )
```

Adds new constraint for specified kernel. Constraints are used to prevent generating of invalid configurations (eg. conflicting parameter values).

Parameters

id	Id of kernel for which the constraint is added.
constraintFunction	Function which returns true if provided combination of parameter values is valid. Returns false otherwise.
parameterNames	Names of kernel parameters which will be affected by the constraint function. The order of parameter names will correspond to the order of parameter values inside constraint function vector argument.

3.7.3.8 addKernel()

```
KernelId ktt::Tuner::addKernel (
            const std::string & source,
            const std::string & kernelName,
            const DimensionVector & globalSize,
            const DimensionVector & localSize )
```

Adds new kernel to tuner from source inside string. Requires specification of kernel name and default global and local thread sizes.

Parameters

source	Kernel source code written in corresponding compute API language.
kernelName	Name of kernel function inside kernel source code.
globalSize	Dimensions for base kernel global size (eg. grid size in CUDA).
localSize	Dimensions for base kernel local size (eg. block size in CUDA).

Returns

Id assigned to kernel by tuner. The id can be used in other API methods.

3.7.3.9 addKernelFromFile()

```
KernelId ktt::Tuner::addKernelFromFile (
             const std::string & filePath,
             const std::string & kernelName,
             const DimensionVector & globalSize,
             const DimensionVector & localSize )
```

Adds new kernel to tuner from file. Requires specification of kernel name and default global and local thread sizes.

Parameters

filePath	Path to file with kernel source code written in corresponding compute API language.
kernelName	Name of kernel function inside kernel source code.
globalSize	Dimensions for base kernel global size (eg. grid size in CUDA).
localSize	Dimensions for base kernel local size (eg. block size in CUDA).

Returns

Id assigned to kernel by tuner. The id can be used in other API methods.

3.7.3.10 addParameter() [1/2]

```
void ktt::Tuner::addParameter (
             const KernelId id,
             const std::string & parameterName,
             const std::vector< size_t > & parameterValues )
```

Adds new integer parameter for specified kernel, providing parameter name and list of allowed values. When the corresponding kernel is launched, parameters will be added to kernel source code as preprocessor definitions. During the tuning process, tuner will generate configurations for combinations of kernel parameters and their values.

Parameters

id	Id of kernel for which the parameter is added.
parameterName	Name of a parameter. Parameter names for single kernel must be unique.
parameterValues	Vector of allowed values for the parameter.

3.7.3.11 addParameter() [2/2]

```
void ktt::Tuner::addParameter (
            const KernelId id,
            const std::string & parameterName,
            const std::vector< size_t > & parameterValues,
            const ThreadModifierType & modifierType,
            const ThreadModifierAction & modifierAction,
            const Dimension & modifierDimension )
```

Adds new integer parameter for specified kernel, providing parameter name and list of allowed values. When the corresponding kernel is launched, parameters will be added to kernel source code as preprocessor definitions. During the tuning process, tuner will generate configurations for combinations of kernel parameters and their values.

This version of method allows the parameter to act as thread size modifier. Parameter value modifies number of threads in either global or local space in specified dimension. Form of modification depends on thread modifier action argument. If there are multiple thread modifiers present for same space and dimension, actions are applied in the order of parameters' addition.

Parameters

id	Id of kernel for which the parameter is added.
parameterName	Name of a parameter. Parameter names for single kernel must be unique.
parameterValues	Vector of allowed values for the parameter.
modifierType	Type of thread modifier. See ThreadModifierType for more information.
modifierAction	Action of thread modifier. See ThreadModifierAction for more information.
modifierDimension	Dimension which will be affected by thread modifier. See Dimension for more information.

3.7.3.12 addParameterDouble()

```
void ktt::Tuner::addParameterDouble (
            const KernelId id,
            const std::string & parameterName,
            const std::vector< double > & parameterValues )
```

Adds new floating-point parameter for specified kernel, providing parameter name and list of allowed values. When the corresponding kernel is launched, parameters will be added to kernel source code as preprocessor definitions. During the tuning process, tuner will generate configurations for combinations of kernel parameters and their values.

Parameters

id	Id of kernel for which the parameter is added.
parameterName	Name of a parameter. Parameter names for single kernel must be unique.
parameterValues	Vector of allowed values for the parameter.

3.7.3.13 getBestConfiguration()

```
std::vector< ParameterPair > ktt::Tuner::getBestConfiguration (
            const KernelId id ) const
```

Returns the best configuration found for specified kernel. Valid configuration will be returned only if methods tune↩ Kernel() or tuneKernelByStep() were already called for corresponding kernel.

Parameters

id	Id of kernel for which the best configuration is returned.

Returns

Best configuration found for specified kernel. See ParameterPair for more information.

3.7.3.14 getCurrentDeviceInfo()

```
DeviceInfo ktt::Tuner::getCurrentDeviceInfo ( ) const
```

Retrieves detailed information about device (eg. device name, memory capacity) used by the tuner. See DeviceInfo for more information.

Returns

Information about device used by the tuner.

3.7.3.15 getDeviceInfo()

```
std::vector< DeviceInfo > ktt::Tuner::getDeviceInfo (
            const size_t platformIndex ) const
```

Retrieves detailed information about all available devices (eg. device name, memory capacity) on specified platform. See DeviceInfo for more information.

Parameters

platformIndex	Index of platform for which the device information is retrieved.

Returns

Information about all available devices on specified platform.

3.7.3.16 getPlatformInfo()

```
std::vector< PlatformInfo > ktt::Tuner::getPlatformInfo ( ) const
```

Retrieves detailed information about all available platforms (eg. platform name, vendor). See PlatformInfo for more information.

Returns

Information about all available platforms.

3.7.3.17 printComputeApiInfo()

```
void ktt::Tuner::printComputeApiInfo (
            std::ostream & outputTarget ) const
```

Prints basic information about available platforms and devices to specified output stream. Also prints indices assigned to them by KTT library.

Parameters

outputTarget	Location where the information is printed.

3.7.3.18 printResult() [1/2]

```
void ktt::Tuner::printResult (
            const KernelId id,
            std::ostream & outputTarget,
            const PrintFormat & format ) const
```

Prints tuning results for specified kernel to specified output stream. Valid results will be printed only if methods tuneKernel() or tuneKernelByStep() were already called for corresponding kernel.

Parameters

id	Id of kernel for which the results are printed.
outputTarget	Location where the results are printed.
format	Format in which the results are printed. See PrintFormat for more information.

3.7.3.19 printResult() [2/2]

```
void ktt::Tuner::printResult (
            const KernelId id,
            const std::string & filePath,
            const PrintFormat & format ) const
```

Prints tuning results for specified kernel to specified file. Valid results will be printed only if methods tuneKernel() or tuneKernelByStep() were already called for corresponding kernel.

Parameters

id	Id of kernel for which the results are printed.
filePath	Path to file where the results are printed.
format	Format in which the results are printed. See PrintFormat for more information.

3.7.3.20 runKernel()

```
void ktt::Tuner::runKernel (
            const KernelId id,
            const std::vector< ParameterPair > & configuration,
            const std::vector< ArgumentOutputDescriptor > & output )
```

Runs specified kernel using provided configuration. Does not perform result validation.

Parameters

id	Id of kernel which is run.
configuration	Configuration under which the kernel will be launched. See ParameterPair for more information.
output	User-provided memory locations for kernel arguments which should be retrieved. See ArgumentOutputDescriptor for more information.

3.7.3.21 setArgumentComparator()

```
void ktt::Tuner::setArgumentComparator (
            const ArgumentId id,
            const std::function< bool(const void *, const void *)> & comparator )
```

Sets argument comparator for specified kernel argument. Arguments with custom data type cannot be compared using built-in comparison operators and require user to provide a comparator. Comparator can also be optionally added for arguments with built-in data types.

Parameters

id	Id of argument for which the comparator is set.
comparator	Function which receives two elements with data type matching the data type of specified kernel argument and returns true if the elements are equal. Returns false otherwise.

3.7.3.22 setAutomaticGlobalSizeCorrection()

```
void ktt::Tuner::setAutomaticGlobalSizeCorrection (
            const bool flag )
```

Toggles automatic correction for global size, which ensures that global size in each dimension is always a multiple of local size in corresponding dimension. Performs a roundup to the nearest higher multiple. Automatic global size correction is disabled by default.

Parameters

flag	If true, automatic global size correction is enabled. It is disabled otherwise.

3.7.3.23 setCompilerOptions()

```
void ktt::Tuner::setCompilerOptions (
            const std::string & options )
```

Sets compute API compiler options to specified options. There are no default options for OpenCL back-end. Default option for CUDA back-end is "--gpu-architecture=compute_30".

Parameters

options	Compute API compiler options. If multiple options are used, they need to be separated by a single space character.

3.7.3.24 setCompositionKernelArguments()

```
void ktt::Tuner::setCompositionKernelArguments (
            const KernelId compositionId,
            const KernelId kernelId,
            const std::vector< ArgumentId > & argumentIds )
```

Calls setKernelArguments() method for a single kernel inside specified kernel composition. Does not affect standalone kernels or other compositions.

Parameters

composition↩ Id	Id of composition which includes the specified kernel.
kernelId	Id of kernel inside the composition for which the arguments are set.
argumentIds	Ids of arguments to be used by specified kernel inside the composition. Order of ids must match the order of kernel arguments specified in kernel function. Argument ids for single kernel must be unique.

3.7.3.25 setGlobalSizeType()

```
void ktt::Tuner::setGlobalSizeType (
             const GlobalSizeType & type )
```

Sets global size specification type to specified compute API style. In OpenCL, NDrange size is specified as number of work-items in a work-group multiplied by number of work-groups. In CUDA, grid size is specified as number of threads in a block divided by number of blocks. This method makes it possible to use OpenCL style in CUDA and vice versa. Default global size type is the one corresponding to compute API of the tuner.

Parameters

type	Global size type which is set for tuner. See GlobalSizeType for more information.

3.7.3.26 setInvalidResultPrinting()

```
void ktt::Tuner::setInvalidResultPrinting (
             const bool flag )
```

Toggles printing of results from failed kernel runs. Invalid results will be separated from valid results during printing. Printing of invalid results is disabled by default.

Parameters

flag	If true, printing of invalid results is enabled. It is disabled otherwise.

3.7.3.27 setKernelArguments()

```
void ktt::Tuner::setKernelArguments (
             const KernelId id,
             const std::vector< ArgumentId > & argumentIds )
```

Sets kernel arguments for specified kernel by providing corresponding argument ids.

Parameters

id	Id of kernel for which the arguments are set.
argumentIds	Ids of arguments to be used by specified kernel. Order of ids must match the order of kernel arguments specified in kernel function. Argument ids for single kernel must be unique.

3.7.3.28 setLoggingTarget() [1/2]

```
void ktt::Tuner::setLoggingTarget (
            std::ostream & outputTarget )
```

Sets the target for info messages logging to specified output stream. Default logging target is `std::clog`.

Parameters

outputTarget	Location where tuner info messages are printed.

3.7.3.29 setLoggingTarget() [2/2]

```
void ktt::Tuner::setLoggingTarget (
            const std::string & filePath )
```

Sets the target for info messages logging to specified file. Default logging target is `std::clog`.

Parameters

filePath	Path to file where tuner info messages are printed.

3.7.3.30 setPrintingTimeUnit()

```
void ktt::Tuner::setPrintingTimeUnit (
            const TimeUnit & unit )
```

Sets time unit used during printing of results inside printResult() methods. Default time unit is microseconds.

Parameters

unit	Time unit which will be used inside printResult() methods. See TimeUnit for more information.

3.7.3.31 setReferenceClass()

```
void ktt::Tuner::setReferenceClass (
            const KernelId id,
            std::unique_ptr< ReferenceClass > referenceClass,
            const std::vector< ArgumentId > & validatedArgumentIds )
```

Sets reference class for specified kernel. Reference class output will be compared to tuned kernel output in order to ensure correctness of computation.

Parameters

id	Id of kernel for which reference class is set.

Parameters

referenceClass	Reference class which produces reference output for specified kernel. See ReferenceClass for more information.
validatedArgumentIds	Ids of kernel arguments which will be validated. The validated arguments must be vector arguments and cannot be read-only.

3.7.3.32 setReferenceKernel()

```
void ktt::Tuner::setReferenceKernel (
            const KernelId id,
            const KernelId referenceId,
            const std::vector< ParameterPair > & referenceConfiguration,
            const std::vector< ArgumentId > & validatedArgumentIds )
```

Sets reference kernel for specified kernel. Reference kernel output will be compared to tuned kernel output in order to ensure correctness of computation. Reference kernel uses only single configuration and cannot be composite.

Parameters

id	Id of kernel for which reference kernel is set.
referenceId	Id of reference kernel. This can be the same as validated kernel. This can be useful in cases where kernel has a configuration which is known to produce correct results.
referenceConfiguration	Configuration under which the reference kernel will be launched to produce reference output.
validatedArgumentIds	Ids of kernel arguments which will be validated. The validated arguments must be vector arguments and cannot be read-only.

3.7.3.33 setSearchMethod()

```
void ktt::Tuner::setSearchMethod (
            const SearchMethod & method,
            const std::vector< double > & arguments )
```

Specifies search method which will be used during kernel tuning. Number of required search arguments depends on the search method. Default search method is full search, which requires no search arguments.

Parameters

method	Search method which will be used during kernel tuning. See SearchMethod for more information.
arguments	Arguments necessary for specified search method to work. Following arguments are required for corresponding search method, the order of arguments is important: • RandomSearch - fraction • PSO - fraction, swarm size, global influence, local influence, random influence • Annealing - fraction, maximum temperature

Fraction argument specifies the number of configurations which will be explored, eg. when fraction is set to 0.5, 50% of all configurations will be explored.

3.7.3.34 setTuningManipulator()

```
void ktt::Tuner::setTuningManipulator (
            const KernelId id,
            std::unique_ptr< TuningManipulator > manipulator )
```

Sets tuning manipulator for specified kernel. Tuning manipulator enables customization of kernel execution. This is useful in several cases, eg. running part of the computation in C++ code, utilizing iterative kernel launches or composite kernels. See TuningManipulator for more information.

Parameters

id	Id of kernel for which the tuning manipulator is set.
manipulator	Tuning manipulator for specified kernel.

3.7.3.35 setValidationMethod()

```
void ktt::Tuner::setValidationMethod (
            const ValidationMethod & method,
            const double toleranceThreshold )
```

Sets validation method and tolerance threshold for floating-point argument validation. Default validation method is side by side comparison. Default tolerance threshold is 1e-4.

Parameters

method	Validation method which will be used for floating-point argument validation. See ValidationMethod for more information.
toleranceThreshold	Output validation threshold. If difference between tuned kernel output and reference output is within tolerance threshold, the tuned kernel output will be considered correct.

3.7.3.36 setValidationRange()

```
void ktt::Tuner::setValidationRange (
            const ArgumentId id,
            const size_t range )
```

Sets validation range for specified argument to specified validation range. Only elements within validation range, starting with the first element, will be validated. All elements are validated by default.

Parameters

id	Id of argument for which the validation range is set.
range	Range inside which the argument elements will be validated, starting from the first element.

3.7.3.37 tuneKernel()

```
void ktt::Tuner::tuneKernel (
            const KernelId id )
```

Starts the tuning process for specified kernel. Creates configuration space based on combinations of provided kernel parameters and constraints. The configurations will be launched in order that depends on specified Search↩ Method.

Parameters

id	Id of kernel for which the tuning begins.

3.7.3.38 tuneKernelByStep()

```
void ktt::Tuner::tuneKernelByStep (
            const KernelId id,
            const std::vector< ArgumentOutputDescriptor > & output )
```

Performs one step of the tuning process for specified kernel. When this method is called inside tuner for the first time, creates configuration space based on combinations of provided kernel parameters and constraints. Each time this method is called, launches single kernel configuration. If all configurations were already tested, runs kernel using the best configuration. Output data can be retrieved by providing output descriptors.

Parameters

id	Id of kernel for which the tuning by step begins.
output	User-provided memory locations for kernel arguments which should be retrieved. See ArgumentOutputDescriptor for more information.

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

- source/tuner_api.h

3.8 ktt::TuningManipulator Class Reference

Class which can be used to customize kernel launch in order to run some part of computation on CPU, utilize iterative kernel launches, kernel compositions and more. In order to use this functionality, new class which publicly inherits from tuning manipulator class has to be defined.

```
#include <tuning_manipulator.h>
```

Public Member Functions

- virtual ∼TuningManipulator ()

  Tuning manipulator destructor. Inheriting class can override destructor with custom implementation. Default implementation is provided by KTT library.
- virtual void launchComputation (const KernelId id)=0

  This method is responsible for directly running the computation and ensuring that correct results are computed. It may utilize any other method inside the tuning manipulator as well as any user-defined methods. Any other tuning manipulator methods run from this method only affect current invocation of launchComputation() method. Inheriting class must provide implementation for this method.
- virtual bool enableArgumentPreload () const

  Controls whether arguments for all kernels that are part of manipulator will be automatically uploaded to corresponding compute API buffers before any kernel is run in the current invocation of launchComputation() method. Argument preload is turned on by default.
- void runKernel (const KernelId id)

  Runs kernel with specified id using thread sizes based on the current configuration.
- void runKernel (const KernelId id, const DimensionVector &globalSize, const DimensionVector &localSize)

  Runs kernel with specified id using specified thread sizes.
- DimensionVector getCurrentGlobalSize (const KernelId id) const

  Returns global thread size of specified kernel based on the current configuration.
- DimensionVector getCurrentLocalSize (const KernelId id) const

  Returns local thread size of specified kernel based on the current configuration.
- std::vector< ParameterPair > getCurrentConfiguration () const

  Returns configuration used inside current invocation of launchComputation() method.
- void updateArgumentScalar (const ArgumentId id, const void ∗argumentData)

  Updates specified scalar argument.
- void updateArgumentLocal (const ArgumentId id, const size_t numberOfElements)

  Updates specified local memory argument.
- void updateArgumentVector (const ArgumentId id, const void ∗argumentData)

  Updates specified vector argument. Does not modify argument size.
- void updateArgumentVector (const ArgumentId id, const void ∗argumentData, const size_t numberOf←Elements)

  Updates specified vector argument. Possibly also modifies argument size.
- void getArgumentVector (const ArgumentId id, void ∗destination) const

  Retrieves specified vector argument.
- void getArgumentVector (const ArgumentId id, void ∗destination, const size_t numberOfElements) const

  Retrieves part of specified vector argument.
- void changeKernelArguments (const KernelId id, const std::vector< ArgumentId > &argumentIds)

  Changes kernel arguments for specified kernel by providing corresponding argument ids.
- void swapKernelArguments (const KernelId id, const ArgumentId argumentIdFirst, const ArgumentId argumentIdSecond)

  Swaps positions of specified kernel arguments for specified kernel.
- void createArgumentBuffer (const ArgumentId id)

  Transfers specified kernel argument to a buffer from which it can be accessed by compute API. This method should be utilized only if argument preload is disabled. See enableArgumentPreload() for more information.
- void destroyArgumentBuffer (const ArgumentId id)

  Destroys compute API buffer for specified kernel argument. This method should be utilized only if argument preload is disabled. See enableArgumentPreload() for more information.

Static Public Member Functions

- static size_t getParameterValue (const std::string &parameterName, const std::vector< ParameterPair > &parameterPairs)

    Returns integer value of specified parameter from provided vector of parameters.
- static double getParameterValueDouble (const std::string &parameterName, const std::vector< Parameter↩ Pair > &parameterPairs)

    Returns floating-point value of specified parameter from provided vector of parameters.

Friends

- class **KernelRunner**

3.8.1 Detailed Description

Class which can be used to customize kernel launch in order to run some part of computation on CPU, utilize iterative kernel launches, kernel compositions and more. In order to use this functionality, new class which publicly inherits from tuning manipulator class has to be defined.

3.8.2 Member Function Documentation

3.8.2.1 changeKernelArguments()

```
void ktt::TuningManipulator::changeKernelArguments (
            const KernelId id,
            const std::vector< ArgumentId > & argumentIds )
```

Changes kernel arguments for specified kernel by providing corresponding argument ids.

Parameters

id	Id of kernel for which the arguments are changed.
argumentIds	Ids of arguments to be used by specified kernel. Order of ids must match the order of kernel arguments specified in kernel function. Argument ids for single kernel must be unique.

3.8.2.2 createArgumentBuffer()

```
void ktt::TuningManipulator::createArgumentBuffer (
            const ArgumentId id )
```

Transfers specified kernel argument to a buffer from which it can be accessed by compute API. This method should be utilized only if argument preload is disabled. See enableArgumentPreload() for more information.

Parameters

id	Id of argument for which the buffer is created.

3.8.2.3 destroyArgumentBuffer()

```
void ktt::TuningManipulator::destroyArgumentBuffer (
            const ArgumentId id )
```

Destroys compute API buffer for specified kernel argument. This method should be utilized only if argument preload is disabled. See enableArgumentPreload() for more information.

Parameters

id	Id of argument for which the buffer is destroyed.

3.8.2.4 enableArgumentPreload()

```
bool ktt::TuningManipulator::enableArgumentPreload ( ) const  [virtual]
```

Controls whether arguments for all kernels that are part of manipulator will be automatically uploaded to corresponding compute API buffers before any kernel is run in the current invocation of launchComputation() method. Argument preload is turned on by default.

Turning this behavior off is useful when utilizing kernel compositions where different kernels use different arguments which would not all fit into available memory. Buffer creation and deletion can be then controlled by using create↵ ArgumentBuffer() and destroyArgumentBuffer() methods for corresponding arguments. Any leftover arguments after launchComputation() method finishes will still be automatically cleaned up. Inheriting class can override this method.

Returns

Flag which controls whether the argument preload is enabled or not.

3.8.2.5 getArgumentVector() [1/2]

```
void ktt::TuningManipulator::getArgumentVector (
            const ArgumentId id,
            void * destination ) const
```

Retrieves specified vector argument.

Parameters

id	Id of vector argument which is retrieved.
destination	Pointer to destination where vector argument data will be copied. Destination buffer size needs to be equal or greater than argument size.

3.8.2.6 getArgumentVector() [2/2]

```
void ktt::TuningManipulator::getArgumentVector (
             const ArgumentId id,
             void * destination,
             const size_t numberOfElements ) const
```

Retrieves part of specified vector argument.

Parameters

id	Id of vector argument which is retrieved.
destination	Pointer to destination where vector argument data will be copied. Destination buffer size needs to be equal or greater than size of specified number of elements.
numberOfElements	Number of elements which will be copied to specified destination, starting with first element.

3.8.2.7 getCurrentConfiguration()

```
std::vector< ParameterPair > ktt::TuningManipulator::getCurrentConfiguration ( ) const
```

Returns configuration used inside current invocation of launchComputation() method.

Returns

Current configuration. See ParameterPair for more information.

3.8.2.8 getCurrentGlobalSize()

```
DimensionVector ktt::TuningManipulator::getCurrentGlobalSize (
             const KernelId id ) const
```

Returns global thread size of specified kernel based on the current configuration.

Parameters

id	Id of kernel for which the global size is retrieved. It must either match the id used to launch kernel from tuner API or be included inside composition which was launched from tuner API.

Returns

Global thread size of specified kernel.

3.8.2.9 getCurrentLocalSize()

```
DimensionVector ktt::TuningManipulator::getCurrentLocalSize (
             const KernelId id ) const
```

Returns local thread size of specified kernel based on the current configuration.

Parameters

id	Id of kernel for which the local size is retrieved. It must either match the id used to launch kernel from tuner API or be included inside composition which was launched from tuner API.

Returns

Local thread size of specified kernel.

3.8.2.10 getParameterValue()

```
static size_t ktt::TuningManipulator::getParameterValue (
             const std::string & parameterName,
             const std::vector< ParameterPair > & parameterPairs )  [static]
```

Returns integer value of specified parameter from provided vector of parameters.

Returns

Integer value of specified parameter.

3.8.2.11 getParameterValueDouble()

```
static double ktt::TuningManipulator::getParameterValueDouble (
             const std::string & parameterName,
             const std::vector< ParameterPair > & parameterPairs )  [static]
```

Returns floating-point value of specified parameter from provided vector of parameters.

Returns

Floating-point value of specified parameter.

3.8.2.12 launchComputation()

```
void ktt::TuningManipulator::launchComputation (
             const KernelId id )  [pure virtual]
```

This method is responsible for directly running the computation and ensuring that correct results are computed. It may utilize any other method inside the tuning manipulator as well as any user-defined methods. Any other tuning manipulator methods run from this method only affect current invocation of launchComputation() method. Inheriting class must provide implementation for this method.

When tuning manipulator is used, total execution duration is calculated from two components. First component is the sum of execution times of all kernel launches inside this method. Second component is the execution time of the method itself, minus the execution times of kernel launches. Initial buffer transfer times are not included in the total duration, same as in the case of kernel tuning without manipulator. Other buffer update and retrieval times are included in the second component.

Parameters

id	Id of a kernel or kernel composition which was used to launch kernel from tuner API.

3.8.2.13 runKernel() [1/2]

```
void ktt::TuningManipulator::runKernel (
             const KernelId id )
```

Runs kernel with specified id using thread sizes based on the current configuration.

Parameters

id	Id of kernel which is run. It must either match the id used to launch kernel from tuner API or be included inside composition which was launched from tuner API.

3.8.2.14 runKernel() [2/2]

```
void ktt::TuningManipulator::runKernel (
             const KernelId id,
             const DimensionVector & globalSize,
             const DimensionVector & localSize )
```

Runs kernel with specified id using specified thread sizes.

Parameters

id	Id of kernel which is run. It must either match the id used to launch kernel from tuner API or be included inside composition which was launched from tuner API.
globalSize	Dimensions for global size with which the kernel is run.
localSize	Dimensions for local size with which the kernel is run.

3.8.2.15 swapKernelArguments()

```
void ktt::TuningManipulator::swapKernelArguments (
            const KernelId id,
            const ArgumentId argumentIdFirst,
            const ArgumentId argumentIdSecond )
```

Swaps positions of specified kernel arguments for specified kernel.

Parameters

id	Id of kernel for which the arguments are swapped.
argumentIdFirst	Id of the first argument which is swapped.
argumentIdSecond	Id of the second argument which is swapped.

3.8.2.16 updateArgumentLocal()

```
void ktt::TuningManipulator::updateArgumentLocal (
            const ArgumentId id,
            const size_t numberOfElements )
```

Updates specified local memory argument.

Parameters

id	Id of local memory argument which is updated.
numberOfElements	Number of local memory elements inside updated argument. Data types for old and new data match.

3.8.2.17 updateArgumentScalar()

```
void ktt::TuningManipulator::updateArgumentScalar (
            const ArgumentId id,
            const void * argumentData )
```

Updates specified scalar argument.

Parameters

id	Id of scalar argument which is updated.
argumentData	Pointer to new data for scalar argument. Data types for old and new data have to match.

3.8.2.18 updateArgumentVector() [1/2]

```
void ktt::TuningManipulator::updateArgumentVector (
             const ArgumentId id,
             const void * argumentData )
```

Updates specified vector argument. Does not modify argument size.

Parameters

id	Id of vector argument which is updated.
argumentData	Pointer to new data for vector argument. Number of elements and data types for old and new data have to match.

3.8.2.19 updateArgumentVector() [2/2]

```
void ktt::TuningManipulator::updateArgumentVector (
             const ArgumentId id,
             const void * argumentData,
             const size_t numberOfElements )
```

Updates specified vector argument. Possibly also modifies argument size.

Parameters

id	Id of vector argument which is updated.
argumentData	Pointer to new data for vector argument. Data types for old and new data have to match.
numberOfElements	Number of elements inside updated vector argument.

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

- source/api/tuning_manipulator.h

Chapter 4

File Documentation

4.1 source/api/argument_output_descriptor.h File Reference

Functionality related to retrieving kernel output with KTT API.

```
#include "ktt_platform.h"
#include "ktt_types.h"
```

Classes

- class ktt::ArgumentOutputDescriptor

  Class which can be used to retrieve kernel argument data when calling certain KTT API methods.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

4.1.1 Detailed Description

Functionality related to retrieving kernel output with KTT API.

4.2 source/api/device_info.h File Reference

Functionality related to retrieving information about compute API devices.

```
#include <cstdint>
#include <iostream>
#include <string>
#include "ktt_platform.h"
#include "enum/device_type.h"
```

Classes

- class ktt::DeviceInfo

  Class which holds information about a compute API device.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Functions

- KTT_API std::ostream & ktt::operator<< (std::ostream &outputTarget, const DeviceInfo &deviceInfo)

  Output operator for device info class.

4.2.1 Detailed Description

Functionality related to retrieving information about compute API devices.

4.3 source/api/dimension_vector.h File Reference

Functionality related to specifying thread sizes of a kernel.

```
#include <iostream>
#include <vector>
#include "ktt_platform.h"
#include "enum/dimension.h"
#include "enum/thread_modifier_action.h"
```

Classes

- class ktt::DimensionVector

  Class which holds information about either global or local thread size of a single kernel.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Functions

- KTT_API std::ostream & ktt::operator<< (std::ostream &outputTarget, const DimensionVector &dimension↩ Vector)

  Output operator for dimension vector class.

4.3.1 Detailed Description

Functionality related to specifying thread sizes of a kernel.

4.4 source/api/parameter_pair.h File Reference

Functionality related to holding a value for one kernel parameter.

```
#include <cstddef>
#include <iostream>
#include <string>
#include "ktt_platform.h"
```

Classes

- class ktt::ParameterPair

  Class which holds single value for one kernel parameter.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Functions

- KTT_API std::ostream & ktt::operator<< (std::ostream &outputTarget, const ParameterPair &parameterPair)

  Output operator for parameter pair class.

4.4.1 Detailed Description

Functionality related to holding a value for one kernel parameter.

4.5 source/api/platform_info.h File Reference

Functionality related to retrieving information about compute API platforms.

```
#include <iostream>
#include <string>
#include "ktt_platform.h"
```

Classes

- class ktt::PlatformInfo

  Class which holds information about a compute API platform.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Functions

- KTT_API std::ostream & ktt::operator<< (std::ostream &outputTarget, const PlatformInfo &platformInfo)

  Output operator for platform info class.

4.5.1 Detailed Description

Functionality related to retrieving information about compute API platforms.

4.6 source/api/reference_class.h File Reference

Functionality related to validating kernel output with reference class.

```
#include "ktt_types.h"
```

Classes

- class ktt::ReferenceClass

  Class which can be used to compute reference output for selected kernel arguments inside regular C++ method. In order to use this functionality, new class which publicly inherits from reference class has to be defined.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

4.6.1 Detailed Description

Functionality related to validating kernel output with reference class.

4.7 source/api/tuning_manipulator.h File Reference

Functionality related to customizing kernel runs with tuning manipulator.

```
#include <cstddef>
#include <utility>
#include <vector>
#include "ktt_platform.h"
#include "ktt_types.h"
#include "api/dimension_vector.h"
#include "api/parameter_pair.h"
```

Classes

- class ktt::TuningManipulator

  Class which can be used to customize kernel launch in order to run some part of computation on CPU, utilize iterative kernel launches, kernel compositions and more. In order to use this functionality, new class which publicly inherits from tuning manipulator class has to be defined.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

4.7.1 Detailed Description

Functionality related to customizing kernel runs with tuning manipulator.

4.8 source/enum/argument_access_type.h File Reference

Definition of enum for access type of kernel arguments.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Enumerations

- enum ktt::ArgumentAccessType { ktt::ArgumentAccessType::ReadOnly, ktt::ArgumentAccessType::Write↩Only, ktt::ArgumentAccessType::ReadWrite }

  Enum for access type of kernel arguments. Specifies whether kernel argument is used for input or output by compute API kernel function.

4.8.1 Detailed Description

Definition of enum for access type of kernel arguments.

4.9 source/enum/argument_data_type.h File Reference

Definition of enum for data type of kernel arguments.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Enumerations

- enum ktt::ArgumentDataType {
  ktt::ArgumentDataType::Char, ktt::ArgumentDataType::UnsignedChar, ktt::ArgumentDataType::Short, ktt::↩
  ArgumentDataType::UnsignedShort,
  ktt::ArgumentDataType::Int, ktt::ArgumentDataType::UnsignedInt, ktt::ArgumentDataType::Long, ktt::↩
  ArgumentDataType::UnsignedLong,
  ktt::ArgumentDataType::Half, ktt::ArgumentDataType::Float, ktt::ArgumentDataType::Double, ktt::Argument↩
  DataType::Custom }

  Enum for data type of kernel arguments. Specifies the data type of elements inside single kernel argument.

4.9.1 Detailed Description

Definition of enum for data type of kernel arguments.

4.10 source/enum/argument_memory_location.h File Reference

Definition of enum for memory location of kernel arguments.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Enumerations

- enum ktt::ArgumentMemoryLocation { ktt::ArgumentMemoryLocation::Device, ktt::ArgumentMemory↩
  Location::Host, ktt::ArgumentMemoryLocation::HostZeroCopy }

  Enum for memory location of kernel arguments. Specifies the memory from which the argument data will be accessed by compute API functions and kernels.

4.10.1 Detailed Description

Definition of enum for memory location of kernel arguments.

4.11 source/enum/argument_upload_type.h File Reference

Definition of enum for upload type of kernel arguments.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Enumerations

- enum ktt::ArgumentUploadType { ktt::ArgumentUploadType::Scalar, ktt::ArgumentUploadType::Vector, ktt::↵ ArgumentUploadType::Local }

  Enum for upload type of kernel arguments. Specifies which compute API function should be used internally by KTT library to make the argument accessible to kernel functions.

4.11.1 Detailed Description

Definition of enum for upload type of kernel arguments.

4.12 source/enum/compute_api.h File Reference

Definition of enum for compute APIs supported by KTT library.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Enumerations

- enum ktt::ComputeApi { ktt::ComputeApi::Opencl, ktt::ComputeApi::Cuda, ktt::ComputeApi::Vulkan }

  Enum for compute API used by KTT library. It is utilized during tuner creation.

4.12.1 Detailed Description

Definition of enum for compute APIs supported by KTT library.

4.13 source/enum/device_type.h File Reference

Definition of enum for type of a device.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Enumerations

- enum ktt::DeviceType {
  ktt::DeviceType::CPU, ktt::DeviceType::GPU, ktt::DeviceType::Accelerator, ktt::DeviceType::Default,
  ktt::DeviceType::Custom }

  Enum for type of a device. It is based on device types supported by OpenCL API.

4.13.1 Detailed Description

Definition of enum for type of a device.

4.14 source/enum/dimension.h File Reference

Definition of enum for dimension.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Enumerations

- enum ktt::Dimension { ktt::Dimension::X, ktt::Dimension::Y, ktt::Dimension::Z }

  Enum for dimensions. Dimensions are utilized during specification of parameters which modify kernel thread sizes.

4.14.1 Detailed Description

Definition of enum for dimension.

4.15 source/enum/dimension_vector_type.h File Reference

Definition of enum for dimension vector type.

Namespaces

- ktt

    All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Enumerations

- enum ktt::DimensionVectorType { ktt::DimensionVectorType::Global, ktt::DimensionVectorType::Local }

    Enum for dimension vector type. Specifies whether a single dimension vector holds global or local kernel thread dimensions.

4.15.1 Detailed Description

Definition of enum for dimension vector type.

4.16 source/enum/global_size_type.h File Reference

Definition of enum for format of global thread size.

Namespaces

- ktt

    All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Enumerations

- enum ktt::GlobalSizeType { ktt::GlobalSizeType::Opencl, ktt::GlobalSizeType::Cuda, ktt::GlobalSizeType::↩ Vulkan }

    Enum for format of global thread size. Specifies the format of global thread size specified by user during kernel addition.

4.16.1 Detailed Description

Definition of enum for format of global thread size.

4.17 source/enum/print_format.h File Reference

Definition of enum for format of printed results.

Namespaces

- ktt

    All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Enumerations

- enum ktt::PrintFormat { ktt::PrintFormat::Verbose, ktt::PrintFormat::CSV }

  Enum for format of printed results. Specifies the format used during printing of tuning results.

4.17.1 Detailed Description

Definition of enum for format of printed results.

4.18 source/enum/search_method.h File Reference

Definition of enum for search method used to explore configuration space during kernel tuning.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Enumerations

- enum ktt::SearchMethod { ktt::SearchMethod::FullSearch, ktt::SearchMethod::RandomSearch, ktt::Search↩ Method::PSO, ktt::SearchMethod::Annealing }

  Enum for search method used to explore configuration space during kernel tuning.

4.18.1 Detailed Description

Definition of enum for search method used to explore configuration space during kernel tuning.

4.19 source/enum/thread_modifier_action.h File Reference

Definition of enum for modifier action for kernel parameters.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Enumerations

- enum ktt::ThreadModifierAction { ktt::ThreadModifierAction::Add, ktt::ThreadModifierAction::Subtract, ktt::↩ ThreadModifierAction::Multiply, ktt::ThreadModifierAction::Divide }

  Enum for modifier action for kernel parameters which modify thread size.

4.19.1 Detailed Description

Definition of enum for modifier action for kernel parameters.

4.20 source/enum/thread_modifier_type.h File Reference

Definition of enum for modifier type for kernel parameters.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Enumerations

- enum ktt::ThreadModifierType { ktt::ThreadModifierType::None, ktt::ThreadModifierType::Global, ktt::↩ ThreadModifierType::Local }

  Enum for modifier type for kernel parameters. Specifies whether kernel parameter value affects corresponding kernel thread size.

4.20.1 Detailed Description

Definition of enum for modifier type for kernel parameters.

4.21 source/enum/time_unit.h File Reference

Definition of enum for time unit used during printing of kernel results.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Enumerations

- enum ktt::TimeUnit { ktt::TimeUnit::Nanoseconds, ktt::TimeUnit::Microseconds, ktt::TimeUnit::Milliseconds, ktt::TimeUnit::Seconds }

  Enum for time unit used during printing of kernel results.

4.21.1 Detailed Description

Definition of enum for time unit used during printing of kernel results.

4.22 source/enum/validation_method.h File Reference

Definition of enum for validation method used during validation of floating-point output arguments.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Enumerations

- enum ktt::ValidationMethod { ktt::ValidationMethod::AbsoluteDifference, ktt::ValidationMethod::SideBySide↩ Comparison, ktt::ValidationMethod::SideBySideRelativeComparison }

  Enum for validation method used during validation of floating-point output arguments.

4.22.1 Detailed Description

Definition of enum for validation method used during validation of floating-point output arguments.

4.23 source/ktt_platform.h File Reference

Preprocessor definitions which ensure compatibility for multiple compilers.

4.23.1 Detailed Description

Preprocessor definitions which ensure compatibility for multiple compilers.

4.24 source/ktt_types.h File Reference

Definitions for KTT type aliases.

```
#include <cstddef>
```

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

Typedefs

- using ktt::ArgumentId = size_t

  Data type for referencing kernel arguments in KTT.
- using ktt::KernelId = size_t

  Data type for referencing kernels in KTT.

4.24.1 Detailed Description

Definitions for KTT type aliases.

4.25 source/tuner_api.h File Reference

Public API for KTT library.

```
#include <functional>
#include <iostream>
#include <memory>
#include <ostream>
#include <string>
#include <typeinfo>
#include <type_traits>
#include <vector>
#include "ktt_platform.h"
#include "ktt_types.h"
#include "enum/argument_access_type.h"
#include "enum/argument_data_type.h"
#include "enum/argument_memory_location.h"
#include "enum/argument_upload_type.h"
#include "enum/compute_api.h"
#include "enum/dimension.h"
#include "enum/global_size_type.h"
#include "enum/print_format.h"
#include "enum/time_unit.h"
#include "enum/search_method.h"
#include "enum/thread_modifier_action.h"
#include "enum/thread_modifier_type.h"
#include "enum/validation_method.h"
#include "api/argument_output_descriptor.h"
#include "api/device_info.h"
#include "api/dimension_vector.h"
#include "api/platform_info.h"
#include "api/reference_class.h"
#include "api/tuning_manipulator.h"
#include "half.hpp"
```

Classes

- class ktt::Tuner

  Class which serves as the main part of public API for KTT library.

Namespaces

- ktt

  All classes, methods and type aliases related to KTT library are located inside ktt namespace.

4.25.1 Detailed Description

Public API for KTT library.

Index