Athena

0.1

Generated by Doxygen

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

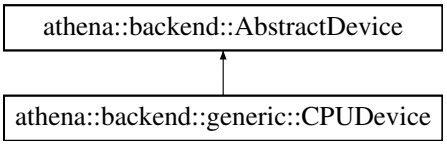
Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 athena::backend::AbstractDevice Class Reference

Inheritance diagram for athena::backend::AbstractDevice:



Public Member Functions

- unsigned long **getMaxThreadMemSize** ()
- void **setMaxThreadMemSize** (unsigned long size=0)
- virtual AbstractMemoryManager * **getMemoryManager** ()=0

Protected Attributes

- unsigned long **maxThreadMemorySize**
- unsigned long **maxThreads**
- unsigned long **memorySize**

4.1.1 Detailed Description

Definition at line 20 of file AbstractDevice.h.

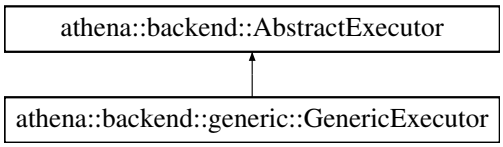
The documentation for this class was generated from the following files:

- backend/AbstractDevice.h
- backend/AbstractDevice.cpp

4.2 athena::backend::AbstractExecutor Class Reference

`#include <AbstractExecutor.h>`

Inheritance diagram for athena::backend::AbstractExecutor:



Public Member Functions

- virtual void execute ()=0
- virtual AbstractMemoryManager * getMemoryManager ()=0
- void setBytecode (std::vector< vm_word > &bytecode)

Protected Attributes

- std::vector< vm_word > **bytecode**

4.2.1 Detailed Description

An Executor is a Virtual Machine that runs Athena `bytecode`. AbstractExecutor is the base class for all executors.

Definition at line 27 of file AbstractExecutor.h.

4.2.2 Member Function Documentation

4.2.2.1 execute()

`virtual void athena::backend::AbstractExecutor::execute ( ) [pure virtual]`

Executes current bytecode. After execution threads state must be reset. However, memory state (Memory manager and its data) must persist.

Implemented in athena::backend::generic::GenericExecutor.

4.2.2.2 getMemoryManager()

```
virtual AbstractMemoryManager* athena::backend::AbstractExecutor::getMemoryManager ( )  [pure virtual]
```

Returns

Memory Manager for current device

Implemented in athena::backend::generic::GenericExecutor.

4.2.2.3 setBytecode()

```
void athena::backend::AbstractExecutor::setBytecode (
            std::vector< vm_word > & bytecode )
```

Sets new bytecode

Parameters

bytecode	Bytecode

Definition at line 16 of file AbstractExecutor.cpp.

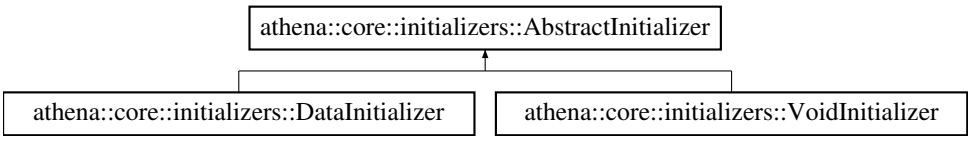
The documentation for this class was generated from the following files:

- backend/AbstractExecutor.h
- backend/AbstractExecutor.cpp

4.3 athena::core::initializers::AbstractInitializer Class Reference

```
#include <AbstractInitializer.h>
```

Inheritance diagram for athena::core::initializers::AbstractInitializer:



Public Member Functions

- virtual void initialize (athena::backend::AbstractMemoryManager *manager, Tensor *tensor)=0

4.3.1 Detailed Description

Initializers help developers load data into corresponding memory type. Every Tensor is assigned with an Initializer – a subclass of AbstractInitializer. Data will be loaded into memory according to initializer's parameters.

Definition at line 34 of file AbstractInitializer.h.

4.3.2 Member Function Documentation

4.3.2.1 initialize()

```
virtual void athena::core::initializers::AbstractInitializer::initialize (
            athena::backend::AbstractMemoryManager * manager,
            Tensor * tensor )  [pure virtual]
```

Loads data into memory. This method **must not** be called by developers. It is automatically called during initialization process.

Parameters

manager	Memory manager for the current device
tensor	A pointer to a Tensor object, that current initializer is assigned to

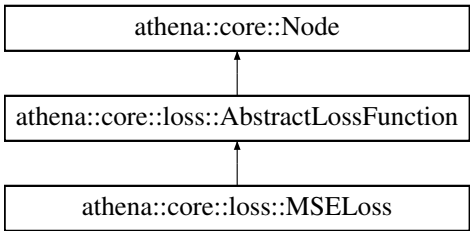
Implemented in athena::core::initializers::DataInitializer, and athena::core::initializers::VoidInitializer.

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

- core/initializers/AbstractInitializer.h

4.4 athena::core::loss::AbstractLossFunction Class Reference

Inheritance diagram for athena::core::loss::AbstractLossFunction:



Public Member Functions

- **AbstractLossFunction** (OpKernel *)

Additional Inherited Members

4.4.1 Detailed Description

Definition at line 21 of file AbstractLossFunction.h.

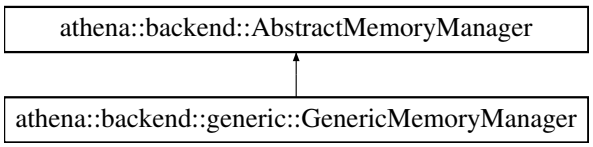
The documentation for this class was generated from the following files:

- core/loss/AbstractLossFunction.h
- core/loss/AbstractLossFunction.cpp

4.5 athena::backend::AbstractMemoryManager Class Reference

`#include <AbstractMemoryManager.h>`

Inheritance diagram for athena::backend::AbstractMemoryManager:



Public Member Functions

- virtual void **init** ()=0
- virtual void **deinit** ()=0
- void resetTable ()
- void addTensor (athena::core::Tensor *tensor)
- virtual void * getPhysicalAddress (vm_word virtualAddress)=0
- void loadAndLock (athena::core::Tensor *tensor)
- void loadAndLock (vm_word address)
- virtual void loadAndLock (vm_word address, unsigned long length)=0
- virtual void unlock (vm_word address)=0
- virtual void deleteFromMem (vm_word address)=0
- athena::core::Tensor * **getTensor** (vm_word address)
- void allocateAndLock (athena::core::Tensor *tensor)
- void allocateAndLock (vm_word address)
- virtual void allocateAndLock (vm_word address, unsigned long length)=0
- virtual void setData (vm_word tensorAddress, vm_word offset, vm_word length, void *data)=0
- virtual void **getData** (vm_word tensorAddress, vm_word offset, vm_word length, void *data)=0

Protected Attributes

- std::list< athena::core::Tensor *> **tensors**

4.5.1 Detailed Description

This class is an interface for physical memory managers. They provide conversion between virtual addresses and physical ones. A typical strategy for memory manager is to allocate as much memory as possible and then provide tensors with it. This class also encapsulates table of athena::core::Tensor objects. One can think of it as of variables table in a compiler.

Definition at line 30 of file AbstractMemoryManager.h.

4.5.2 Member Function Documentation

4.5.2.1 addTensor()

```
void athena::backend::AbstractMemoryManager::addTensor (
            athena::core::Tensor * tensor )
```

Adds Tensor to table

Parameters

tensor	Tensor, that will be added

Definition at line 20 of file AbstractMemoryManager.cpp.

4.5.2.2 allocateAndLock() [1/3]

```
void athena::backend::AbstractMemoryManager::allocateAndLock (
            athena::core::Tensor * tensor )
```

Allocate space in the fastest memory available without loading any data and lock it (prevent from being offloaded)

Parameters

tensor	Tensor memory is being allocated for

Definition at line 54 of file AbstractMemoryManager.cpp.

Referenced by athena::core::initializers::DataInitializer::initialize().

4.5.2.3 allocateAndLock() [2/3]

```
void athena::backend::AbstractMemoryManager::allocateAndLock (
            vm_word address )
```

Allocate space in the fastest memory available without loading any data and lock it (prevent from being offloaded)

Parameters

address	Virtual address of Tensor memory is being allocated for

Definition at line 60 of file AbstractMemoryManager.cpp.

4.5.2.4 allocateAndLock() [3/3]

```
virtual void athena::backend::AbstractMemoryManager::allocateAndLock (
            vm_word address,
            unsigned long length ) [pure virtual]
```

Allocate space in the fastest memory available without loading any data and lock it (prevent from being offloaded)

Parameters

address	Virtual address of Tensor memory is being allocated for
length	Length in bytes for the piece of memory that is being allocated

Implemented in athena::backend::generic::GenericMemoryManager.

4.5.2.5 deleteFromMem()

```
virtual void athena::backend::AbstractMemoryManager::deleteFromMem (
            vm_word address ) [pure virtual]
```

Mark corresponding memory chunk as free

Parameters

address	Virtual address

Implemented in athena::backend::generic::GenericMemoryManager.

4.5.2.6 getPhysicalAddress()

```
virtual void* athena::backend::AbstractMemoryManager::getPhysicalAddress (
            vm_word virtualAddress ) [pure virtual]
```

Convert virtual address to physical one

Parameters

virtualAddress	Virtual address, unsigned long from 0 to 2^64-1

Returns

Pointer to physical memory

Implemented in athena::backend::generic::GenericMemoryManager.

4.5.2.7 loadAndLock() [1/3]

```
void athena::backend::AbstractMemoryManager::loadAndLock (
            athena::core::Tensor * tensor )
```

Move data to the fastest memory type available (e.g. from hard drive to RAM) and lock it (prevent from being offloaded)

Parameters

tensor	Tensor containing data

Definition at line 25 of file AbstractMemoryManager.cpp.

4.5.2.8 loadAndLock() [2/3]

```
void athena::backend::AbstractMemoryManager::loadAndLock (
            vm_word address )
```

Move data to the fastest memory type available (e.g. from hard drive to RAM) and lock it (prevent from being offloaded)

Parameters

address	Virtual address of Tensor containing data

Definition at line 30 of file AbstractMemoryManager.cpp.

4.5.2.9 loadAndLock() [3/3]

```
virtual void athena::backend::AbstractMemoryManager::loadAndLock (
            vm_word address,
            unsigned long length ) [pure virtual]
```

Move data to the fastest memory type available (e.g. from hard drive to RAM) and lock it (prevent from being offloaded)

Parameters

address	Virtual address
length	Size of Tensor in bytes

Implemented in athena::backend::generic::GenericMemoryManager.

4.5.2.10 resetTable()

```
void athena::backend::AbstractMemoryManager::resetTable ( )
```

Clears table of Tensors

Definition at line 16 of file AbstractMemoryManager.cpp.

4.5.2.11 setData()

```
virtual void athena::backend::AbstractMemoryManager::setData (
            vm_word tensorAddress,
            vm_word offset,
            vm_word length,
            void * data )  [pure virtual]
```

Sets memory with the data

Parameters

tensorAddress	Virtual address of Tensor beginning (see Tensor::getStartAddress)
offset	Offset in bytes from the beginning
length	Length of piece of data in bytes
data	Pointer to data

Implemented in athena::backend::generic::GenericMemoryManager.

Referenced by athena::core::initializers::DataInitializer::initialize().

4.5.2.12 unlock()

```
virtual void athena::backend::AbstractMemoryManager::unlock (
            vm_word address )  [pure virtual]
```

Lets data be offloaded to a slower memory type (e.g. from RAM to HDD)

Parameters

address	Virtual address

Implemented in athena::backend::generic::GenericMemoryManager.

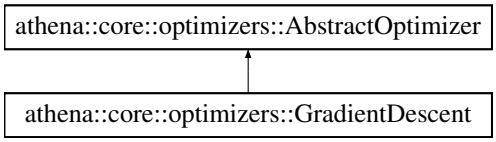
Referenced by athena::core::initializers::DataInitializer::initialize().

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

- backend/AbstractMemoryManager.h
- backend/AbstractMemoryManager.cpp

4.6 athena::core::optimizers::AbstractOptimizer Class Reference

Inheritance diagram for athena::core::optimizers::AbstractOptimizer:



Public Member Functions

- **AbstractOptimizer** (athena::core::loss::AbstractLossFunction *loss)
- void **init** (Session *session)
- virtual void **prepare** ()=0
- virtual void **minimize** ()=0

Protected Attributes

- std::vector< InputNode *> **headNodes**
- std::vector< vm_word > **bytecode**
- unsigned long **lastResultCell**
- Session * **session**
- athena::core::loss::AbstractLossFunction * **loss**

4.6.1 Detailed Description

Definition at line 24 of file AbstractOptimizer.h.

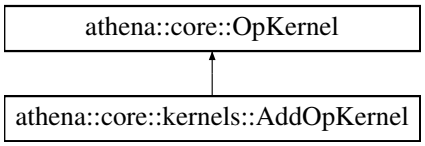
The documentation for this class was generated from the following files:

- core/optimizers/AbstractOptimizer.h
- core/optimizers/AbstractOptimizer.cpp

4.7 athena::core::kernels::AddOpKernel Class Reference

`#include <AddOpKernel.h>`

Inheritance diagram for athena::core::kernels::AddOpKernel:



Public Member Functions

- int getOperandsCount () override
- athena::core::TensorShape & getOutputShape (std::vector< athena::core::TensorShape > &shapes) override
- athena::core::TensorShape & getDerivativeShape (int d, std::vector< athena::core::TensorShape > &shapes) override
- std::vector< vm_word > **getOpBytecode** (std::vector< vm_word > args, vm_word resultCell) override
- std::vector< vm_word > getDerivativeBytecode (int d, std::vector< vm_word > args, vm_word resultCell) override

Additional Inherited Members

4.7.1 Detailed Description

Performs sum of 2 given Tensors

Definition at line 24 of file AddOpKernel.h.

4.7.2 Member Function Documentation

4.7.2.1 getDerivativeBytecode()

```
std::vector< vm_word > athena::core::kernels::AddOpKernel::getDerivativeBytecode (
            int d,
            std::vector< vm_word > args,
            vm_word resultCell )  [override], [virtual]
```

Generates bytecode to calculate partial derivative

Parameters

d	Number of variable with respect to which derivative is calculated
args	Function arguments
resultCell	Number of memory cell where results are saved

Returns

Implements athena::core::OpKernel.

Definition at line 39 of file AddOpKernel.cpp.

4.7.2.2 getDerivativeShape()

```
athena::core::TensorShape & athena::core::kernels::AddOpKernel::getDerivativeShape (
            int d,
            std::vector< athena::core::TensorShape > & shapes )  [override], [virtual]
```

It is important for some operations to have certain size of their operands

Parameters

shape	Original operand shape
dim	Dimensionality

Returns

New shape

Implements athena::core::OpKernel.

Definition at line 61 of file AddOpKernel.cpp.

4.7.2.3 getOperandsCount()

```
int athena::core::kernels::AddOpKernel::getOperandsCount ( )  [override], [virtual]
```

There can be unary, binary and other operations

Returns

Number of operands accepted

Implements athena::core::OpKernel.

Definition at line 18 of file AddOpKernel.cpp.

4.7.2.4 getOutputShape()

```
athena::core::TensorShape & athena::core::kernels::AddOpKernel::getOutputShape (
            std::vector< athena::core::TensorShape > & shapes )  [override], [virtual]
```

It is important for some operations to have certain size of their operands

Parameters

shape	Original operand shape
dim	Dimensionality

Returns

New shape

Implements athena::core::OpKernel.

Definition at line 56 of file AddOpKernel.cpp.

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

- core/kernels/AddOpKernel.h
- core/kernels/AddOpKernel.cpp

4.8 BasicOpCodeParams Struct Reference

4.8.1 Detailed Description

Definition at line 141 of file opcodes.h.

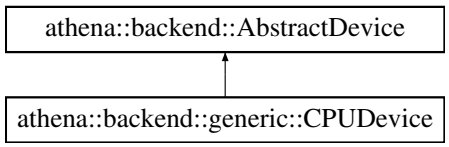
The documentation for this struct was generated from the following file:

- backend/opcodes.h

4.9 athena::backend::generic::CPUDevice Class Reference

`#include <CPUDevice.h>`

Inheritance diagram for athena::backend::generic::CPUDevice:



Public Member Functions

- AbstractMemoryManager * **getMemoryManager** () override

Additional Inherited Members

4.9.1 Detailed Description

This class represents a CPU It encapsulates Memory Manager

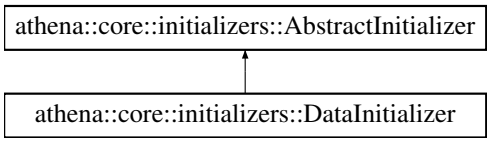
Definition at line 27 of file CPUDevice.h.

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

- backend/generic/CPUDevice.h
- backend/generic/CPUDevice.cpp

4.10 athena::core::initializers::DataInitializer Class Reference

Inheritance diagram for athena::core::initializers::DataInitializer:



Public Member Functions

- **DataInitializer** (const DataInitializer &src)
- DataInitializer & **operator=** (const DataInitializer &src)
- void **setData** (void *ptr, size_t length)
- void initialize (athena::backend::AbstractMemoryManager *manager, Tensor *tensor) override

4.10.1 Detailed Description

Definition at line 23 of file DataInitializer.h.

4.10.2 Member Function Documentation

4.10.2.1 initialize()

```
void athena::core::initializers::DataInitializer::initialize (
            athena::backend::AbstractMemoryManager * manager,
            Tensor * tensor )  [override], [virtual]
```

Loads data into memory. This method **must not** be called by developers. It is automatically called during initialization process.

Parameters

manager	Memory manager for the current device
tensor	A pointer to a Tensor object, that current initializer is assigned to

Implements athena::core::initializers::AbstractInitializer.

Definition at line 16 of file DataInitializer.cpp.

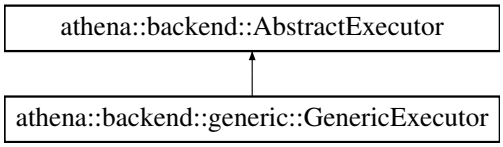
The documentation for this class was generated from the following files:

- core/initializers/DataInitializer.h
- core/initializers/DataInitializer.cpp

4.11 athena::backend::generic::GenericExecutor Class Reference

`#include <GenericExecutor.h>`

Inheritance diagram for athena::backend::generic::GenericExecutor:



Public Member Functions

- **GenericExecutor** (CPUDevice *cpuDevice)
- void execute () override
- AbstractMemoryManager * getMemoryManager () override

Additional Inherited Members

4.11.1 Detailed Description

GenericExecutor is the state of the art implementation of AbstractExecutor. While we try to make it work fast, the main goal of this implementation is to be mathematically correct and provide an example for more specific implementation.

GenericExecutor executes `bytecode` with standard CPU device. The actual implementations of bytecode commands use BLAS to speed up calculations. There are several accelerators available:

- `Apple Accelerate Framework`
- `OpenBLAS`
- `BLIS`

You can configure them during compile time. Other accelerators may be added later.

Definition at line 50 of file GenericExecutor.h.

4.11.2 Member Function Documentation

4.11.2.1 execute()

`void athena::backend::generic::GenericExecutor::execute ( ) [override], [virtual]`

Executes current bytecode. After execution threads state must be reset. However, memory state (Memory manager and its data) must persist.

Implements athena::backend::AbstractExecutor.

Definition at line 22 of file GenericExecutor.cpp.

4.11.2.2 getMemoryManager()

```
athena::backend::AbstractMemoryManager * athena::backend::generic::GenericExecutor::getMemoryManager
( )  [override], [virtual]
```

Returns

Memory Manager for current device

Implements athena::backend::AbstractExecutor.

Definition at line 31 of file GenericExecutor.cpp.

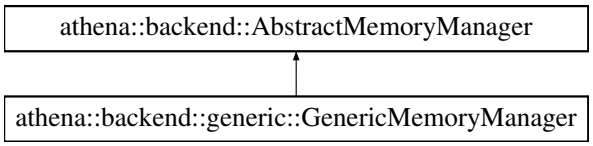
The documentation for this class was generated from the following files:

- backend/generic/GenericExecutor.h
- backend/generic/GenericExecutor.cpp

4.12 athena::backend::generic::GenericMemoryManager Class Reference

```
#include <GenericMemoryManager.h>
```

Inheritance diagram for athena::backend::generic::GenericMemoryManager:



Public Member Functions

- void init () override
- void deinit () override
- void * getPhysicalAddress (vm_word virtualAddress) override
- void loadAndLock (vm_word address, unsigned long length) override
- void allocateAndLock (vm_word address, unsigned long length) override
- void unlock (vm_word address) override
- void deleteFromMem (vm_word address) override
- void **setMemSize** (size_t memSize)
- void setData (vm_word tensorAddress, vm_word offset, vm_word length, void *data) override
- void **getData** (vm_word tensorAddress, vm_word offset, vm_word length, void *data) override
- void loadAndLock (athena::core::Tensor *tensor)
- void loadAndLock (vm_word address)
- virtual void loadAndLock (vm_word address, unsigned long length)=0
- void allocateAndLock (athena::core::Tensor *tensor)
- void allocateAndLock (vm_word address)
- virtual void allocateAndLock (vm_word address, unsigned long length)=0

Protected Member Functions

- void allocationThreadFunc (int laneId)
- void processQueueItem (QueueItem *item)

Protected Attributes

- std::list< SwapRecord *> **swapRecords**
- MemoryChunk * **memoryChunksHead**
- void * **memory**
- std::mutex **memoryChunksLock**
- std::vector< std::thread > **memLanes**
- size_t **allocatedMemory**
- std::queue< QueueItem *> **loadQueue**
- std::vector< bool > **laneFinished**

4.12.1 Detailed Description

This class implements AbstractMemoryManager interface for GenericExecutor. It pre-allocates RAM and uses persistent memory for swap. There are couple memory lanes - threads, that manage RAM. They monitor loadQueue for new queries and move data from hard drive to RAM if needed.

Definition at line 76 of file GenericMemoryManager.h.

4.12.2 Member Function Documentation

4.12.2.1 allocateAndLock() [1/4]

```
void athena::backend::generic::GenericMemoryManager::allocateAndLock (
            vm_word address,
            unsigned long length )  [override], [virtual]
```

Allocate space in the fastest memory available without loading any data and lock it (prevent from being offloaded)

Parameters

address	Virtual address of Tensor memory is being allocated for
length	Length in bytes for the piece of memory that is being allocated

Implements athena::backend::AbstractMemoryManager.

Definition at line 174 of file GenericMemoryManager.cpp.

4.12.2.2 allocateAndLock() [2/4]

```
void athena::backend::AbstractMemoryManager::allocateAndLock
```

Allocate space in the fastest memory available without loading any data and lock it (prevent from being offloaded)

Parameters

tensor	Tensor memory is being allocated for

Definition at line 54 of file AbstractMemoryManager.cpp.

4.12.2.3 allocateAndLock() [3/4]

```
void athena::backend::AbstractMemoryManager::allocateAndLock
```

Allocate space in the fastest memory available without loading any data and lock it (prevent from being offloaded)

Parameters

address	Virtual address of Tensor memory is being allocated for

Definition at line 60 of file AbstractMemoryManager.cpp.

4.12.2.4 allocateAndLock() [4/4]

```
virtual void athena::backend::AbstractMemoryManager::allocateAndLock
```

Allocate space in the fastest memory available without loading any data and lock it (prevent from being offloaded)

Parameters

address	Virtual address of Tensor memory is being allocated for
length	Length in bytes for the piece of memory that is being allocated

4.12.2.5 allocationThreadFunc()

```
void athena::backend::generic::GenericMemoryManager::allocationThreadFunc (
            int laneId ) [protected]
```

This is a thread function for memory lane-threads. It loads data to RAM and notifies corresponding threads

Parameters

laneId	

Definition at line 37 of file GenericMemoryManager.cpp.

Referenced by init().

4.12.2.6 deinit()

```
void athena::backend::generic::GenericMemoryManager::deinit ( )  [override], [virtual]
```

Free RAM and stop all threads-memory lanes

Implements athena::backend::AbstractMemoryManager.

Definition at line 84 of file GenericMemoryManager.cpp.

4.12.2.7 deleteFromMem()

```
void athena::backend::generic::GenericMemoryManager::deleteFromMem (
            vm_word address )  [override], [virtual]
```

Mark corresponding memory chunk as free

Parameters

address	Virtual address

Implements athena::backend::AbstractMemoryManager.

Definition at line 148 of file GenericMemoryManager.cpp.

4.12.2.8 getPhysicalAddress()

```
void * athena::backend::generic::GenericMemoryManager::getPhysicalAddress (
            vm_word virtualAddress )  [override], [virtual]
```

Convert virtual address to physical one

Parameters

virtualAddress	Virtual address, unsigned long from 0 to 2^64-1

Returns

Pointer to physical memory

Implements athena::backend::AbstractMemoryManager.

Definition at line 51 of file GenericMemoryManager.cpp.

4.12.2.9 init()

```
void athena::backend::generic::GenericMemoryManager::init ( )  [override], [virtual]
```

Initialize memory manager. That's where actual memory allocation happens. All configurations should be done before this method is called.

Implements athena::backend::AbstractMemoryManager.

Definition at line 18 of file GenericMemoryManager.cpp.

4.12.2.10 loadAndLock() [1/4]

```
void athena::backend::generic::GenericMemoryManager::loadAndLock (
            vm_word address,
            unsigned long length )  [override], [virtual]
```

Move data to the fastest memory type available (e.g. from hard drive to RAM) and lock it (prevent from being offloaded)

Parameters

address	Virtual address
length	Size of Tensor in bytes

Implements athena::backend::AbstractMemoryManager.

Definition at line 65 of file GenericMemoryManager.cpp.

4.12.2.11 loadAndLock() [2/4]

```
void athena::backend::AbstractMemoryManager::loadAndLock
```

Move data to the fastest memory type available (e.g. from hard drive to RAM) and lock it (prevent from being offloaded)

Parameters

address	Virtual address of Tensor containing data

Definition at line 30 of file AbstractMemoryManager.cpp.

4.12.2.12 loadAndLock() [3/4]

```
void athena::backend::AbstractMemoryManager::loadAndLock
```

Move data to the fastest memory type available (e.g. from hard drive to RAM) and lock it (prevent from being offloaded)

Parameters

tensor	Tensor containing data

Definition at line 25 of file AbstractMemoryManager.cpp.

4.12.2.13 loadAndLock() [4/4]

```
virtual void athena::backend::AbstractMemoryManager::loadAndLock
```

Move data to the fastest memory type available (e.g. from hard drive to RAM) and lock it (prevent from being offloaded)

Parameters

address	Virtual address
length	Size of Tensor in bytes

4.12.2.14 processQueueItem()

```
void athena::backend::generic::GenericMemoryManager::processQueueItem (
            QueueItem * item )  [protected]
```

This method does actual physical memory allocation

Parameters

item	

Definition at line 219 of file GenericMemoryManager.cpp.

4.12.2.15 setData()

```
void athena::backend::generic::GenericMemoryManager::setData (
            vm_word tensorAddress,
            vm_word offset,
            vm_word length,
            void * data )  [override], [virtual]
```

Sets memory with the data

Parameters

tensorAddress	Virtual address of Tensor beginning (see Tensor::getStartAddress)
offset	Offset in bytes from the beginning
length	Length of piece of data in bytes
data	Pointer to data

Implements athena::backend::AbstractMemoryManager.

Definition at line 195 of file GenericMemoryManager.cpp.

4.12.2.16 unlock()

```
void athena::backend::generic::GenericMemoryManager::unlock (
            vm_word address )  [override], [virtual]
```

Lets data be offloaded to a slower memory type (e.g. from RAM to HDD)

Parameters

address	Virtual address

Implements athena::backend::AbstractMemoryManager.

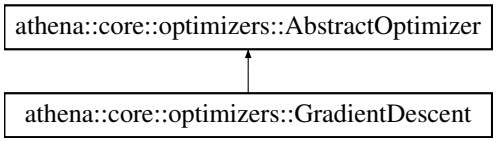
Definition at line 105 of file GenericMemoryManager.cpp.

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

- backend/generic/GenericMemoryManager.h
- backend/generic/GenericMemoryManager.cpp

4.13 athena::core::optimizers::GradientDescent Class Reference

Inheritance diagram for athena::core::optimizers::GradientDescent:



Public Member Functions

- **GradientDescent** (athena::core::loss::AbstractLossFunction *loss, float learningRate)
- void prepare () override
- void **minimize** () override

Protected Member Functions

- std::tuple< std::vector< vm_word >, vm_word > **getByteCode** (athena::core::loss::AbstractLossFunction *node)

Protected Attributes

- float **learningRate**

4.13.1 Detailed Description

Definition at line 21 of file GradientDescent.h.

4.13.2 Member Function Documentation

4.13.2.1 prepare()

```
void athena::core::optimizers::GradientDescent::prepare ( )  [override], [virtual]
```

Generate bytecode for backpropagation

The whole algorithm:

1. Calculate actual error E
2. Let's Q - queue of nodes, EQ - queue of errors
   (a) $node \to Q$
   (b) $E \to EQ$
3. For each node N in Q
   (a) $EQ \to E$
   (b) If this is variable node, adjust weights: $N = N - \alpha * E$
   (c) If this is regular node, for each incoming node I:
      i. If I is constant, skip
      ii. $E_i = D_i \odot E$, where E_i is the new error value, D_i - derivative of N with respect to I, $\odot$ - Hadamard (elementwise) product.
      iii. $E_i \to EQ$
      iv. $I \to Q$

Implements athena::core::optimizers::AbstractOptimizer.

Definition at line 24 of file GradientDescent.cpp.

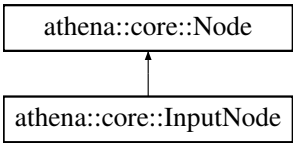
The documentation for this class was generated from the following files:

- core/optimizers/GradientDescent.h
- core/optimizers/GradientDescent.cpp

4.14 athena::core::InputNode Class Reference

`#include <InputNode.h>`

Inheritance diagram for athena::core::InputNode:



Public Member Functions

- **InputNode** (Tensor *input, bool isFrozen=true)
- bool isInputNode () override
- void setMappedMemCell (unsigned long cell)
- unsigned long getMappedMemCell ()
- void after (Node *) override
- Tensor * getData ()
- bool isFrozen ()
- void setFrozen (bool frozen)
- void **setInitializer** (athena::core::initializers::AbstractInitializer *initializer)
- athena::core::initializers::AbstractInitializer * **getInitializer** ()
- bool **isGarbage** () override

Additional Inherited Members

4.14.1 Detailed Description

Subclass of athena::core::Node Represents a node that has no predecessors

Definition at line 26 of file InputNode.h.

4.14.2 Member Function Documentation

4.14.2.1 after()

```
void athena::core::InputNode::after (
            Node *  )  [inline], [override], [virtual]
```

InputNodes can't be placed after other nodes in Athena's execution graph. This method does nothing

Reimplemented from athena::core::Node.

Definition at line 63 of file InputNode.h.

4.14.2.2 getData()

```
athena::core::Tensor * athena::core::InputNode::getData ( )
```

Get data associated with this InputNode

Returns

Pointer to Tensor

Definition at line 28 of file InputNode.cpp.

4.14.2.3 getMappedMemCell()

```
unsigned long athena::core::InputNode::getMappedMemCell ( )
```

Get the number of memory cell that is used to store tensor for this node

Returns

Memory cell number

Definition at line 24 of file InputNode.cpp.

4.14.2.4 isFrozen()

```
bool athena::core::InputNode::isFrozen ( )
```

InputNodes can be frozen. This means their tensors won't be changed during back propagation process (e.g. InputNode contains your input data). By default new InputNodes are frozen.

Returns

Current freeze state

Definition at line 32 of file InputNode.cpp.

4.14.2.5 isInputNode()

```
bool athena::core::InputNode::isInputNode ( )  [override], [virtual]
```

Check if it is an input node

Returns

true

Reimplemented from athena::core::Node.

Definition at line 16 of file InputNode.cpp.

4.14.2.6 setFrozen()

```
void athena::core::InputNode::setFrozen (
            bool frozen )
```

InputNodes can be frozen. This means their tensors won't be changed during back propagation process (e.g. InputNode contains your input data). By default new InputNodes are frozen.

Parameters

frozen	True - freeze node, False - unfreeze node (make it variable)

Definition at line 36 of file InputNode.cpp.

4.14.2.7 setMappedMemCell()

```
void athena::core::InputNode::setMappedMemCell (
            unsigned long cell )
```

Specify which memory cell will be used to store tensor for this node

Parameters

cell	Memory cell number

Definition at line 20 of file InputNode.cpp.

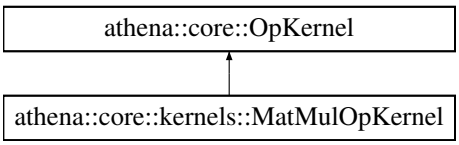
The documentation for this class was generated from the following files:

- core/InputNode.h
- core/InputNode.cpp

4.15 athena::core::kernels::MatMulOpKernel Class Reference

```
#include <MatMulOpKernel.h>
```

Inheritance diagram for athena::core::kernels::MatMulOpKernel:



Public Member Functions

- int getOperandsCount () override
- athena::core::TensorShape & getOutputShape (std::vector< athena::core::TensorShape > &shapes) override
- athena::core::TensorShape & getDerivativeShape (int d, std::vector< athena::core::TensorShape > &shapes) override
- std::vector< vm_word > **getOpBytecode** (std::vector< vm_word > args, vm_word resultCell) override
- std::vector< vm_word > getDerivativeBytecode (int d, std::vector< vm_word > args, vm_word resultCell) override

Additional Inherited Members

4.15.1 Detailed Description

Performs matrix multiplication of given Tensors. Matrix is a 2-D Tensor. The main restriction for this operation is that the number of columns for the first column must be equal to the number of rows for the second matrix. The reason to introduce this operation apart from Tensor product is that it is widely adopted by different acceleration mechanism (BLAS, cuBLAS, Accelerate Framework, etc)

Definition at line 29 of file MatMulOpKernel.h.

4.15.2 Member Function Documentation

4.15.2.1 getDerivativeBytecode()

```
std::vector< vm_word > athena::core::kernels::MatMulOpKernel::getDerivativeBytecode (
            int d,
            std::vector< vm_word > args,
            vm_word resultCell )  [override], [virtual]
```

Generates bytecode to calculate partial derivative

Parameters

d	Number of variable with respect to which derivative is calculated
args	Function arguments
resultCell	Number of memory cell where results are saved

Returns

Implements athena::core::OpKernel.

Definition at line 38 of file MatMulOpKernel.cpp.

4.15.2.2 getDerivativeShape()

```
athena::core::TensorShape & athena::core::kernels::MatMulOpKernel::getDerivativeShape (
            int d,
            std::vector< athena::core::TensorShape > & shapes )  [override], [virtual]
```

It is important for some operations to have certain size of their operands

Parameters

shape	Original operand shape
dim	Dimensionality

Returns

New shape

Implements athena::core::OpKernel.

Definition at line 63 of file MatMulOpKernel.cpp.

4.15.2.3 getOperandsCount()

```
int athena::core::kernels::MatMulOpKernel::getOperandsCount ( )  [override], [virtual]
```

There can be unary, binary and other operations

Returns

Number of operands accepted

Implements athena::core::OpKernel.

Definition at line 16 of file MatMulOpKernel.cpp.

4.15.2.4 getOutputShape()

```
athena::core::TensorShape & athena::core::kernels::MatMulOpKernel::getOutputShape (
            std::vector< athena::core::TensorShape > & shapes )  [override], [virtual]
```

It is important for some operations to have certain size of their operands

Parameters

shape	Original operand shape
dim	Dimensionality

Returns

New shape

Implements athena::core::OpKernel.

Definition at line 56 of file MatMulOpKernel.cpp.

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

- core/kernels/MatMulOpKernel.h
- core/kernels/MatMulOpKernel.cpp

4.16 athena::backend::generic::MemoryChunk Struct Reference

`#include <GenericMemoryManager.h>`

Public Attributes

- vm_word **virtualAddress**
- void * **begin**
- size_t **length**
- bool **isFree**
- bool **isLocked**
- MemoryChunk * **next**
- MemoryChunk * **prev**

4.16.1 Detailed Description

Describes single memory chunk that is allocated in RAM. Free status means there is no data in this chunk Locked status means this chunk is being used now and can't be unload to persistent memory.

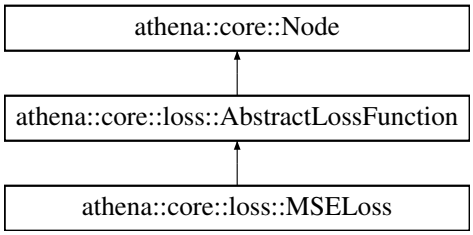
Definition at line 46 of file GenericMemoryManager.h.

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

- backend/generic/GenericMemoryManager.h

4.17 athena::core::loss::MSELoss Class Reference

Inheritance diagram for athena::core::loss::MSELoss:



Additional Inherited Members

4.17.1 Detailed Description

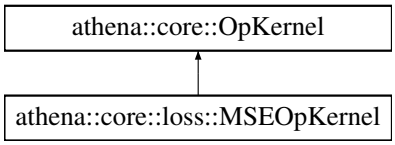
Definition at line 47 of file MSELoss.h.

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

- core/loss/MSELoss.h
- core/loss/MSELoss.cpp

4.18 athena::core::loss::MSEOpKernel Class Reference

Inheritance diagram for athena::core::loss::MSEOpKernel:



Public Member Functions

- int getOperandsCount () override
- athena::core::TensorShape & getOutputShape (std::vector< athena::core::TensorShape > &shapes) override
- athena::core::TensorShape & getDerivativeShape (int, std::vector< athena::core::TensorShape > &shapes) override
- std::vector< vm_word > **getOpBytecode** (std::vector< vm_word > args, vm_word resultCell) override
- std::vector< vm_word > getDerivativeBytecode (int d, std::vector< vm_word > args, vm_word resultCell) override

Additional Inherited Members

4.18.1 Detailed Description

Definition at line 22 of file MSELoss.h.

4.18.2 Member Function Documentation

4.18.2.1 getDerivativeBytecode()

```
std::vector< vm_word > athena::core::loss::MSEOpKernel::getDerivativeBytecode (
            int d,
            std::vector< vm_word > args,
            vm_word resultCell )  [override], [virtual]
```

Generates bytecode to calculate partial derivative

Parameters

d	Number of variable with respect to which derivative is calculated
args	Function arguments
resultCell	Number of memory cell where results are saved

Returns

Implements athena::core::OpKernel.

Definition at line 40 of file MSELoss.cpp.

4.18.2.2 getDerivativeShape()

```
athena::core::TensorShape & athena::core::loss::MSEOpKernel::getDerivativeShape (
            int d,
            std::vector< athena::core::TensorShape > & shapes )  [override], [virtual]
```

It is important for some operations to have certain size of their operands

Parameters

shape	Original operand shape
dim	Dimensionality

Returns

New shape

Implements athena::core::OpKernel.

Definition at line 59 of file MSELoss.cpp.

4.18.2.3 getOperandsCount()

```
int athena::core::loss::MSEOpKernel::getOperandsCount ( )  [override], [virtual]
```

There can be unary, binary and other operations

Returns

Number of operands accepted

Implements athena::core::OpKernel.

Definition at line 22 of file MSELoss.cpp.

4.18.2.4 getOutputShape()

```
athena::core::TensorShape & athena::core::loss::MSEOpKernel::getOutputShape (
            std::vector< athena::core::TensorShape > & shapes )  [override], [virtual]
```

It is important for some operations to have certain size of their operands

Parameters

shape	Original operand shape
dim	Dimensionality

Returns

New shape

Implements athena::core::OpKernel.

Definition at line 53 of file MSELoss.cpp.

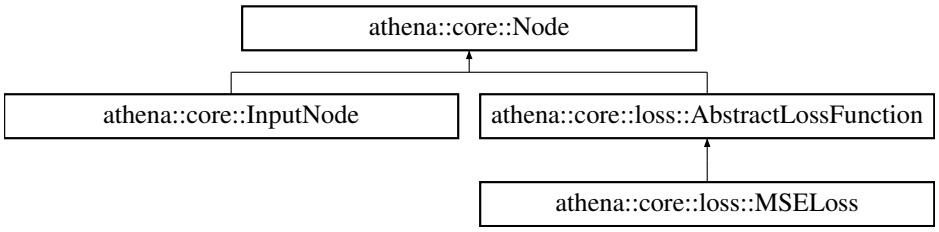
The documentation for this class was generated from the following files:

- core/loss/MSELoss.h
- core/loss/MSELoss.cpp

4.19 athena::core::Node Class Reference

`#include <Node.h>`

Inheritance diagram for athena::core::Node:



Public Member Functions

- **Node** (OpKernel *)
- virtual void after (Node *predecessor)
- virtual bool isInputNode ()
- OpKernel * **getOp** ()
- std::vector< Node *> & **getIncomingNodes** ()
- std::string **getName** ()
- void **addDerivative** (unsigned long d)
- vm_word **getDerivative** (unsigned long i)
- void **setCalculated** (unsigned long resCell)
- bool **isCalculated** ()
- vm_word **getResult** ()
- void **updateUsageCount** ()
- virtual bool **isGarbage** ()
- void **setPersistResult** ()

Protected Member Functions

- std::string **getRandomNodeName** ()

Protected Attributes

- std::vector< Node * > **incomingNodes**
- std::vector< Node * > **outcomingNodes**
- OpKernel * **operation**
- std::string **name**
- bool **calculated**
- std::vector< vm_word > **derivatives**
- unsigned long **resultCell**
- unsigned long **usageCount**
- bool **persistResult**

4.19.1 Detailed Description

A basic element of execution graph Each node has pointers to its predecessors and successors. It encapsulates operation and data.

Definition at line 29 of file Node.h.

4.19.2 Member Function Documentation

4.19.2.1 after()

```
void athena::core::Node::after (
            Node * predecessor )  [virtual]
```

Makes a new oriented edge in execution graph from predecessor to this node

Parameters

predecessor	A predecessor node

Reimplemented in athena::core::InputNode.

Definition at line 22 of file Node.cpp.

4.19.2.2 isInputNode()

```
bool athena::core::Node::isInputNode ( )  [virtual]
```

Check if it is an input node

Returns

false

Reimplemented in athena::core::InputNode.

Definition at line 33 of file Node.cpp.

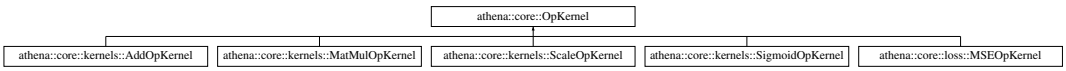
The documentation for this class was generated from the following files:

- core/Node.h
- core/Node.cpp

4.20 athena::core::OpKernel Class Reference

`#include <OpKernel.h>`

Inheritance diagram for athena::core::OpKernel:



Public Member Functions

- **OpKernel** (std::string name)
- virtual int getOperandsCount ()=0
- virtual athena::core::TensorShape & getOutputShape (std::vector< athena::core::TensorShape > &shapes)=0
- virtual athena::core::TensorShape & getDerivativeShape (int d, std::vector< athena::core::TensorShape > &shapes)=0
- virtual std::vector< vm_word > **getOpBytecode** (std::vector< vm_word > args, vm_word resultCell)=0
- virtual std::vector< vm_word > getDerivativeBytecode (int d, std::vector< vm_word > args, vm_word resultCell)=0

Protected Attributes

- std::string **name**

4.20.1 Detailed Description

Operation skeleton Each operation has OpCode

Definition at line 29 of file OpKernel.h.

4.20.2 Member Function Documentation

4.20.2.1 getDerivativeBytecode()

```
virtual std::vector< vm_word > athena::core::OpKernel::getDerivativeBytecode (
            int d,
            std::vector< vm_word > args,
            vm_word resultCell )  [pure virtual]
```

Generates bytecode to calculate partial derivative

Parameters

d	Number of variable with respect to which derivative is calculated
args	Function arguments
resultCell	Number of memory cell where results are saved

Returns

Implemented in athena::core::kernels::MatMulOpKernel, athena::core::kernels::SigmoidOpKernel, athena::core::loss::MSEOpKern
athena::core::kernels::AddOpKernel, and athena::core::kernels::ScaleOpKernel.

4.20.2.2 getDerivativeShape()

```
virtual athena::core::TensorShape& athena::core::OpKernel::getDerivativeShape (
            int d,
            std::vector< athena::core::TensorShape > & shapes )  [pure virtual]
```

It is important for some operations to have certain size of their operands

Parameters

shape	Original operand shape
dim	Dimensionality

Returns

New shape

Implemented in athena::core::kernels::MatMulOpKernel, athena::core::kernels::SigmoidOpKernel, athena::core::kernels::AddOpK
athena::core::kernels::ScaleOpKernel, and athena::core::loss::MSEOpKernel.

4.20.2.3 getOperandsCount()

```
virtual int athena::core::OpKernel::getOperandsCount ( )  [pure virtual]
```

There can be unary, binary and other operations

Returns

Number of operands accepted

Implemented in athena::core::kernels::MatMulOpKernel, athena::core::kernels::SigmoidOpKernel, athena::core::kernels::AddOpK
athena::core::kernels::ScaleOpKernel, and athena::core::loss::MSEOpKernel.

4.20.2.4 getOutputShape()

```
virtual athena::core::TensorShape& athena::core::OpKernel::getOutputShape (
            std::vector< athena::core::TensorShape > & shapes )  [pure virtual]
```

It is important for some operations to have certain size of their operands

Parameters

shape	Original operand shape
dim	Dimensionality

Returns

New shape

Implemented in athena::core::kernels::MatMulOpKernel, athena::core::kernels::SigmoidOpKernel, athena::core::kernels::AddOpK
athena::core::kernels::ScaleOpKernel, and athena::core::loss::MSEOpKernel.

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

- core/OpKernel.h

4.21 athena::backend::generic::QueueItem Struct Reference

`#include <GenericMemoryManager.h>`

Public Attributes

- vm_word **address**
- size_t **length**
- bool **alloc** = false
- std::condition_variable **loadHandle**
- std::mutex **m**
- bool **notified** = false

4.21.1 Detailed Description

Describes which Tensors should be loaded to RAM Alloc flag means we should not search for data in Swap

Definition at line 60 of file GenericMemoryManager.h.

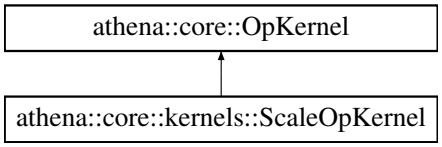
The documentation for this struct was generated from the following file:

- backend/generic/GenericMemoryManager.h

4.22 athena::core::kernels::ScaleOpKernel Class Reference

`#include <ScaleOpKernel.h>`

Inheritance diagram for athena::core::kernels::ScaleOpKernel:



Public Member Functions

- int getOperandsCount () override
- athena::core::TensorShape & getOutputShape (std::vector< athena::core::TensorShape > &shapes) override
- athena::core::TensorShape & getDerivativeShape (int d, std::vector< athena::core::TensorShape > &shapes) override
- std::vector< vm_word > **getOpBytecode** (std::vector< vm_word > args, vm_word resultCell) override
- std::vector< vm_word > getDerivativeBytecode (int d, std::vector< vm_word > args, vm_word resultCell) override

Additional Inherited Members

4.22.1 Detailed Description

Multiply Tensor by scalar

Definition at line 24 of file ScaleOpKernel.h.

4.22.2 Member Function Documentation

4.22.2.1 getDerivativeBytecode()

```
std::vector< vm_word > athena::core::kernels::ScaleOpKernel::getDerivativeBytecode (
            int d,
            std::vector< vm_word > args,
            vm_word resultCell )  [override], [virtual]
```

Generates bytecode to calculate partial derivative

Parameters

d	Number of variable with respect to which derivative is calculated
args	Function arguments
resultCell	Number of memory cell where results are saved

Returns

Implements athena::core::OpKernel.

Definition at line 35 of file ScaleOpKernel.cpp.

4.22.2.2 getDerivativeShape()

```
athena::core::TensorShape & athena::core::kernels::ScaleOpKernel::getDerivativeShape (
            int d,
            std::vector< athena::core::TensorShape > & shapes )  [override], [virtual]
```

It is important for some operations to have certain size of their operands

Parameters

shape	Original operand shape
dim	Dimensionality

Returns

New shape

Implements athena::core::OpKernel.

Definition at line 53 of file ScaleOpKernel.cpp.

4.22.2.3 getOperandsCount()

```
int athena::core::kernels::ScaleOpKernel::getOperandsCount ( )  [override], [virtual]
```

There can be unary, binary and other operations

Returns

Number of operands accepted

Implements athena::core::OpKernel.

Definition at line 16 of file ScaleOpKernel.cpp.

4.22.2.4 getOutputShape()

```
athena::core::TensorShape & athena::core::kernels::ScaleOpKernel::getOutputShape (
            std::vector< athena::core::TensorShape > & shapes )  [override], [virtual]
```

It is important for some operations to have certain size of their operands

Parameters

shape	Original operand shape
dim	Dimensionality

Returns

New shape

Implements athena::core::OpKernel.

Definition at line 47 of file ScaleOpKernel.cpp.

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

- core/kernels/ScaleOpKernel.h
- core/kernels/ScaleOpKernel.cpp

4.23 athena::core::Session Class Reference

```
#include <Session.h>
```

Public Member Functions

- void prepare (Node *logits)
- Tensor * run ()
- unsigned long **getResultCell** ()
- void **setExecutor** (athena::backend::AbstractExecutor *exec)
- athena::backend::AbstractExecutor * **getExecutor** ()
- athena::backend::VirtualMemory * **getMemory** ()

4.23.1 Detailed Description

The class encapsulates everything needed for a single training step

Definition at line 28 of file Session.h.

4.23.2 Member Function Documentation

4.23.2.1 prepare()

```
void athena::core::Session::prepare (
            Node * logits )
```

Generates bytecode for the whole graph

Parameters

logits	

Definition at line 18 of file Session.cpp.

4.23.2.2 run()

```
athena::core::Tensor * athena::core::Session::run ( )
```

does single training step

Returns

result tensor

Definition at line 126 of file Session.cpp.

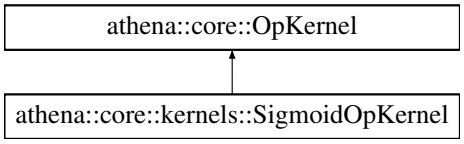
The documentation for this class was generated from the following files:

- core/Session.h
- core/Session.cpp

4.24 athena::core::kernels::SigmoidOpKernel Class Reference

```
#include <SigmoidOpKernel.h>
```

Inheritance diagram for athena::core::kernels::SigmoidOpKernel:



Public Member Functions

- int getOperandsCount () override
- athena::core::TensorShape & getOutputShape (std::vector< athena::core::TensorShape > &shapes) override
- athena::core::TensorShape & getDerivativeShape (int d, std::vector< athena::core::TensorShape > &shapes) override
- std::vector< vm_word > **getOpBytecode** (std::vector< vm_word > args, vm_word resultCell) override
- std::vector< vm_word > getDerivativeBytecode (int d, std::vector< vm_word > args, vm_word resultCell) override

Additional Inherited Members

4.24.1 Detailed Description

Apply sigmoid function to every element of Tensor. See https://en.wikipedia.org/wiki/Sigmoid_↩function for more info

Definition at line 26 of file SigmoidOpKernel.h.

4.24.2 Member Function Documentation

4.24.2.1 getDerivativeBytecode()

```
std::vector< vm_word > athena::core::kernels::SigmoidOpKernel::getDerivativeBytecode (
            int d,
            std::vector< vm_word > args,
            vm_word resultCell )  [override], [virtual]
```

Generates bytecode to calculate partial derivative

Parameters

d	Number of variable with respect to which derivative is calculated
args	Function arguments
resultCell	Number of memory cell where results are saved

Returns

Implements athena::core::OpKernel.

Definition at line 38 of file SigmoidOpKernel.cpp.

4.24.2.2 getDerivativeShape()

```
athena::core::TensorShape & athena::core::kernels::SigmoidOpKernel::getDerivativeShape (
            int d,
            std::vector< athena::core::TensorShape > & shapes )  [override], [virtual]
```

It is important for some operations to have certain size of their operands

Parameters

shape	Original operand shape
dim	Dimensionality

Returns

New shape

Implements athena::core::OpKernel.

Definition at line 53 of file SigmoidOpKernel.cpp.

4.24.2.3 getOperandsCount()

```
int athena::core::kernels::SigmoidOpKernel::getOperandsCount ( )  [override], [virtual]
```

There can be unary, binary and other operations

Returns

Number of operands accepted

Implements athena::core::OpKernel.

Definition at line 19 of file SigmoidOpKernel.cpp.

4.24.2.4 getOutputShape()

```
athena::core::TensorShape & athena::core::kernels::SigmoidOpKernel::getOutputShape (
            std::vector< athena::core::TensorShape > & shapes )  [override], [virtual]
```

It is important for some operations to have certain size of their operands

Parameters

shape	Original operand shape
dim	Dimensionality

Returns

New shape

Implements athena::core::OpKernel.

Definition at line 59 of file SigmoidOpKernel.cpp.

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

- core/kernels/SigmoidOpKernel.h
- core/kernels/SigmoidOpKernel.cpp

4.25 athena::backend::generic::SwapRecord Struct Reference

```
#include <GenericMemoryManager.h>
```

Public Attributes

- vm_word **address**
- size_t **length**
- std::string **filename**

4.25.1 Detailed Description

Describes single swap record - a file, that stores Tensor data

Definition at line 34 of file GenericMemoryManager.h.

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

- backend/generic/GenericMemoryManager.h

4.26 athena::core::Tensor Class Reference

`#include <Tensor.h>`

Public Member Functions

- **Tensor** (const TensorShape &shape, DataType dataType)
- const TensorShape & **getShape** () const
- DataType **getType** () const
- vm_word **getStartAddress** ()
- void **setStartAddress** (vm_word address)
- Tensor & **operator[]** (unsigned int idx)

4.26.1 Detailed Description

In mathematics **tensor** is an abstract object, expressing some definite type of multi-linear concept. See Wikipedia for more info.

In Athena Tensor is an abstraction to represent data inside computational graph. A 1-dimensional Tensor is either scalar or vector. A 2-dimensional Tensor is a matrix.

Definition at line 39 of file Tensor.h.

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

- core/Tensor.h
- core/Tensor.cpp

4.27 athena::core::TensorShape Class Reference

`#include <TensorShape.h>`

Public Member Functions

- **TensorShape** (std::vector< size_t > shape)
- **TensorShape** (unsigned long *shape, unsigned long length)
- **TensorShape** (const TensorShape &)
- TensorShape & **operator=** (const TensorShape &)
- unsigned long dimensions () const
- unsigned long dim (unsigned long n) const
- unsigned long totalSize () const
- const std::vector< unsigned long > & **getShape** () const
- bool operator== (const TensorShape &) const
- bool operator!= (const TensorShape &rhs) const

4.27.1 Detailed Description

Class represents size parameters for Tensor

Definition at line 25 of file TensorShape.h.

4.27.2 Member Function Documentation

4.27.2.1 dim()

```
unsigned long athena::core::TensorShape::dim (
            unsigned long n ) const
```

Gives size for certain dimension

Parameters

n	Dimension index (0 <= d < dimensions)

Returns

Size of dimension n

Definition at line 33 of file TensorShape.cpp.

Referenced by operator==().

4.27.2.2 dimensions()

```
unsigned long athena::core::TensorShape::dimensions ( ) const
```

Returns

Number of dimensions in Tensor

Definition at line 29 of file TensorShape.cpp.

Referenced by operator==().

4.27.2.3 operator"!=()

```
bool athena::core::TensorShape::operator!= (
            const TensorShape & rhs ) const
```

Parameters

rhs	TensorShape to be compared with

Returns

True if dimensions are different, else False

Definition at line 58 of file TensorShape.cpp.

4.27.2.4 operator==()

```
bool athena::core::TensorShape::operator== (
            const TensorShape & rhs ) const
```

Returns

True if dimensions are equal, else False

Definition at line 42 of file TensorShape.cpp.

4.27.2.5 totalSize()

```
unsigned long athena::core::TensorShape::totalSize ( ) const
```

Returns

Total number of elements in Tensor

Definition at line 17 of file TensorShape.cpp.

Referenced by athena::backend::VirtualMemory::allocate(), athena::backend::AbstractMemoryManager↩
::allocateAndLock(), and athena::backend::AbstractMemoryManager::loadAndLock().

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

- core/TensorShape.h
- core/TensorShape.cpp

4.28 athena::backend::VirtualMemory Class Reference

`#include <VirtualMemory.h>`

Public Member Functions

- vm_word allocate (athena::core::Tensor *tensor)
- void free (athena::core::Tensor *tensor)
- void free (vm_word virtualAddress)
- athena::core::Tensor * **getTensor** (vm_word address)

4.28.1 Detailed Description

Virtual memory is an abstraction of storage resources that are actually available on a given machine. Each thread has its own address space. In Athena's VM address space is linear. This means that valid addresses are 0 to 2^64 - 1. Address 0 is reserved for NULL value. When Tensor is initialized, it is given with a continuous block of virtual addresses. When one actually needs to access Tensor's data, Memory Manager allocates physical memory and converts virtual addresses to physical ones. This helps Athena to run in low-memory conditions. This class is heavily used in Session class to generate bytecode.
To discover more about Virtual Memory see article on Wikipedia

Definition at line 45 of file VirtualMemory.h.

4.28.2 Member Function Documentation

4.28.2.1 allocate()

```
vm_word athena::backend::VirtualMemory::allocate (
            athena::core::Tensor * tensor )
```

Allocates virtual memory for given Tensor

Parameters

tensor	Tensor object

Returns

Virtual Address of 0 element of Tensor

Definition at line 28 of file VirtualMemory.cpp.

4.28.2.2 free() [1/2]

```
void athena::backend::VirtualMemory::free (
            athena::core::Tensor * tensor )
```

Marks memory as free

Parameters

tensor	Corresponding tensor

Definition at line 137 of file VirtualMemory.cpp.

4.28.2.3 free() [2/2]

```
void athena::backend::VirtualMemory::free (
            vm_word virtualAddress )
```

Marks memory as free

Parameters

virtualAddress	

Definition at line 97 of file VirtualMemory.cpp.

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

- backend/VirtualMemory.h
- backend/VirtualMemory.cpp

4.29 athena::backend::VMemoryBlock Struct Reference

Public Attributes

- bool **isUsed**
- vm_word **startAddress**
- vm_word **endAddress**
- VMemoryBlock * **nextBlock**
- VMemoryBlock * **prevBlock**

4.29.1 Detailed Description

Definition at line 23 of file VirtualMemory.h.

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

- backend/VirtualMemory.h

4.30 athena::backend::VMState Struct Reference

`#include <VMState.h>`

Public Attributes

- unsigned long BC = 1
- unsigned long BP = 0
- unsigned long IP = 0

4.30.1 Detailed Description

VMState structure describes current state of executor thread

Definition at line 22 of file VMState.h.

4.30.2 Member Data Documentation

4.30.2.1 BC

`unsigned long athena::backend::VMState::BC = 1`

Batch Counter, number of batches

Definition at line 27 of file VMState.h.

4.30.2.2 BP

`unsigned long athena::backend::VMState::BP = 0`

Batch Pointer, number of current batch

Definition at line 32 of file VMState.h.

4.30.2.3 IP

`unsigned long athena::backend::VMState::IP = 0`

Instruction Pointer, points to instruction that is going to be executed

Definition at line 37 of file VMState.h.

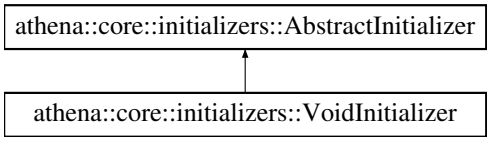
The documentation for this struct was generated from the following file:

- backend/VMState.h

4.31 athena::core::initializers::VoidInitializer Class Reference

`#include <VoidInitializer.h>`

Inheritance diagram for athena::core::initializers::VoidInitializer:



Public Member Functions

- void initialize (athena::backend::AbstractMemoryManager *, Tensor *) override

4.31.1 Detailed Description

VoidInitializer is the simplest initializer possible. It does not initialize Tensor in any way. It is the default initializer for newly created Tensor

Definition at line 25 of file VoidInitializer.h.

4.31.2 Member Function Documentation

4.31.2.1 initialize()

```
void athena::core::initializers::VoidInitializer::initialize (
            athena::backend::AbstractMemoryManager * manager,
            Tensor * tensor )  [override], [virtual]
```

Loads data into memory. This method **must not** be called by developers. It is automatically called during initialization process.

Parameters

manager	Memory manager for the current device
tensor	A pointer to a Tensor object, that current initializer is assigned to

Implements athena::core::initializers::AbstractInitializer.

Definition at line 16 of file VoidInitializer.cpp.

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

- core/initializers/VoidInitializer.h
- core/initializers/VoidInitializer.cpp

Chapter 5

File Documentation

5.1 backend/opcodes.h File Reference

Classes

- struct BasicOpCodeParams

Typedefs

- typedef unsigned long **vm_word**

Enumerations

- enum **OpCode** : vm_word {
  DEL = 0x0000000000,
  PUSH = 0x0100000000,
  POP = 0x0200000000,
  JMP = 0x0300000000,
  ADD = 0x0400000000,
  MATMUL = 0x0500000000,
  MKSCALAR = 0x0600000000,
  SCALE = 0x0700000000,
  SIGMOID = 0x0800000000,
  SIGMOID_DERIV = 0x0900000000,
  TRANSPOSE = 0x0A00000000,
  COPY = 0x0B00000000,
  MSE = 0x0C00000000,
  MSE_DERIV = 0x0D00000000,
  ALLOC = 0x0E00000000,
  MUL = 0x0F00000000,
  HADAMARD = 0x1000000000 }

5.1.1 Detailed Description

OpCode specifies digital codes for Athena VM's operations. Operation is encoded with operation code and its parameters. Bits 0..31 (assuming little-endian) are used by operation code. Bits 32..63 are used by operation parameters.

Description of Athena VM operations

Operation	Description	Parameters	Arguments
ADD	Produces sum of two Tensors		3 arguments: pointers to two summands and result Tensor
MATMUL	Produces product of two matrix Tensors	todo describe 2 parameters: transpose args	3 arguments: pointers to two matrices and result Tensor
MKSCALAR	Creates Tensor from scalar constant	todo describe 1 parameter: data type	2 arguments: scalar and result Tensor
SCALE	Multiplies every Tensor element by scalar	No parameters	3 arguments: pointers to scalar Tensor, source Tensor and result Tensor
SIGMOID	Applies sigmoid function to every element of Tensor	No parameters	2 arguments: source Tensor and result Tensor
SIGMOID_DERIV	Calculates derivative of sigmoid function	No parameters	2 arguments: source Tensor and result Tensor
TRANSPOSE	Transpose matrix Tensor	No parameters	2 arguments: source Tensor and result Tensor
TRANSPOSE	Transpose matrix Tensor	No parameters	2 arguments: source Tensor and result Tensor
COPY	Copy Tensor	No parameters	2 arguments: source Tensor and result Tensor
MSE	Calculate minimal squared error	No parameters	3 arguments: produced Tensor, label Tensor and result Tensor
MSE_DERIV	Calculate derivative of minimal squared error	No parameters	3 arguments: produced Tensor, label Tensor and result Tensor
ALLOC	Allocate memory for Tensor	todo describe one parameter: data type	First argument always specifies number of dimensions, last arg specifies virtual address of new Tensor.
MUL	Multiply two Tensors	No parameters	3 arguments: multiplier Tensors and result Tensor
HADAMARD	Do element-wise multiplication of two matrices	No parameters	3 arguments: multiplier Tensors and result Tensor

Index