Cog Compute Graph

External Reference Specification

Version 1

Greg Snider

The Compute Graph is Cog’s model of a massively-parallel computation. This document describes how it’s produced, who uses it, its internal structure, and how it is exchanged between different program components.

# 1. Overview

A Compute Graph models a Cog computation. It is produced by the Cog compiler and written in serial form to a binary file. This binary file is used by several other components:

1. The Native Runtime system to build and execute the computation specified by the Compute Graph.
2. The Cog Debugger to display and control the execution of the Compute Graph.
3. A Cog application so that it may write inputs (sensors) and write outputs (actuators) of the Compute Graph

Cog Compiler

Cog Runtime

Cog Application

Cog Debugger

Compute Graph (binary file)

*uses*

*uses*

*uses*

*produces*

A Compute Graph is composed of tensor fields (corresponding to edges in the graph) and operators (corresponding to nodes in the graph). All information flows through tensor fields and all computation is done by operators.

tensor field

operator

tensor field

tensor field

(feedback operator)

<== operator

<==

Operators have one or more ordered inputs (tensor fields) and one or more ordered outputs (tensor fields). User-defined operators and most user-defined operators use a dataflow abstraction, where inputs flow through, and are transformed by, an operator to produce its outputs. For the most part, a Compute Graph is a directed, acyclic graph.

There are two special operators, though, that allow feedback within a Compute Graph. The <== operator copies its single input field to its single output with a delay of one cycle; this is sometimes called the “mutation” operator. The <~~ operator copies its input to its output, but with an unknown delay of a few cycles; this is sometimes called the “lazy mutation” operator. The lazy feedback operator allows for greater optimization and faster execution of Compute Graphs, and is useful where feedback is used to create tensor fields that “learn” or “adapt” at a slow rate.

# 2. Compute Graph Structure

Since Compute Graphs must be understood by Scala and C / C++ code, there are parallel data structures to represent them:

*C version:*

struct CogComputeGraph {

int majorReleaseNumber; // Major release number of Cog compiler.

int minorReleaseNumber; // Minor release number of Cog compiler.

char\* name; // User-defined name of Compute Graph.

int fieldCount; // Number of fields in the Compute Graph

CogTensorField fields[]; // Descriptor of each field.

float\*\* fieldsInitialData[]; // Initial data for each field (optional).

int operatorCount; // Number of operators.

CogOperator operators[]; // Descriptor of each operator.

int operatorCodeCount/ // Number of operator code segments.

CogOperatorCode operatorCode[]; // Operator code segments.

}

struct CogOperator {

CogOperatorCode\* operatorCode; // Implementation of the operator.

int inputCount; // Number of inputs to the operator.

CogTensorField\* inputs[]; // Pointers to the input fields.

int outputCount; // Number of outputs from the operator.

CogTensorField\* outputs[]; // Pointers to the output fields.

}

struct CogOperatorCode {

char\* name; // Name of the operator.

enum CogCodeType codeType; // Type of code implementing the operator.

char\* code; // Code implementing the operator.

}

enum CogCodeType {

COG\_OPENCL, // OpenCL code.

COG\_CUDA // CUDA code (not yet supported).

}

*Scala version:*

class CogComputeGraph {

val majorReleaseNumber: Int

val minorReleaseNumber: Int

val name: String

val fields: Array[CogTensorField]

val fieldsInitialData: Array[Option[Array[Float]]]

val operators: Array[CogOperator]

val operatorCode: Array[CogOperatorCode]

}

class CogOperator {

val operatorCode: CogOperatorCode

val inputs: Array[CogTensorField]

val outputs: Array[CogTensorField]

}

class CogOperatorCode {

val name: String

val codeType: CogCodeType

val code: String)

}

sealed abstract class CogCodeType

case object CogOpenCL extends CogCodeType

case object CogCUDA extends CogCodeType

The CogTensorField class/struct is defined in the Cog Tensor Field External Reference Specification.

# 3. Read and Writing the Compute Graph File

The Compute Graph file is written only by the Cog Compiler, so the Scala version of the CogComputeGraph class contains a method for doing that:

class CogComputeGraph {

// data fields defined previously are here.

def write(filename: String): Void

}

Since Scala and C both need to read the Compute Graph file, a method (Scala) and function (C) are provided:

object CogComputeGraph {

def readFromFile(filename: String): CogComputeGraph

}

CogComputeGraph\* readCogComputeGraph(char \*filename)