Documentation for repository

EMAM2CPP

Created by Sascha Schneiders

2018 - May - 22th

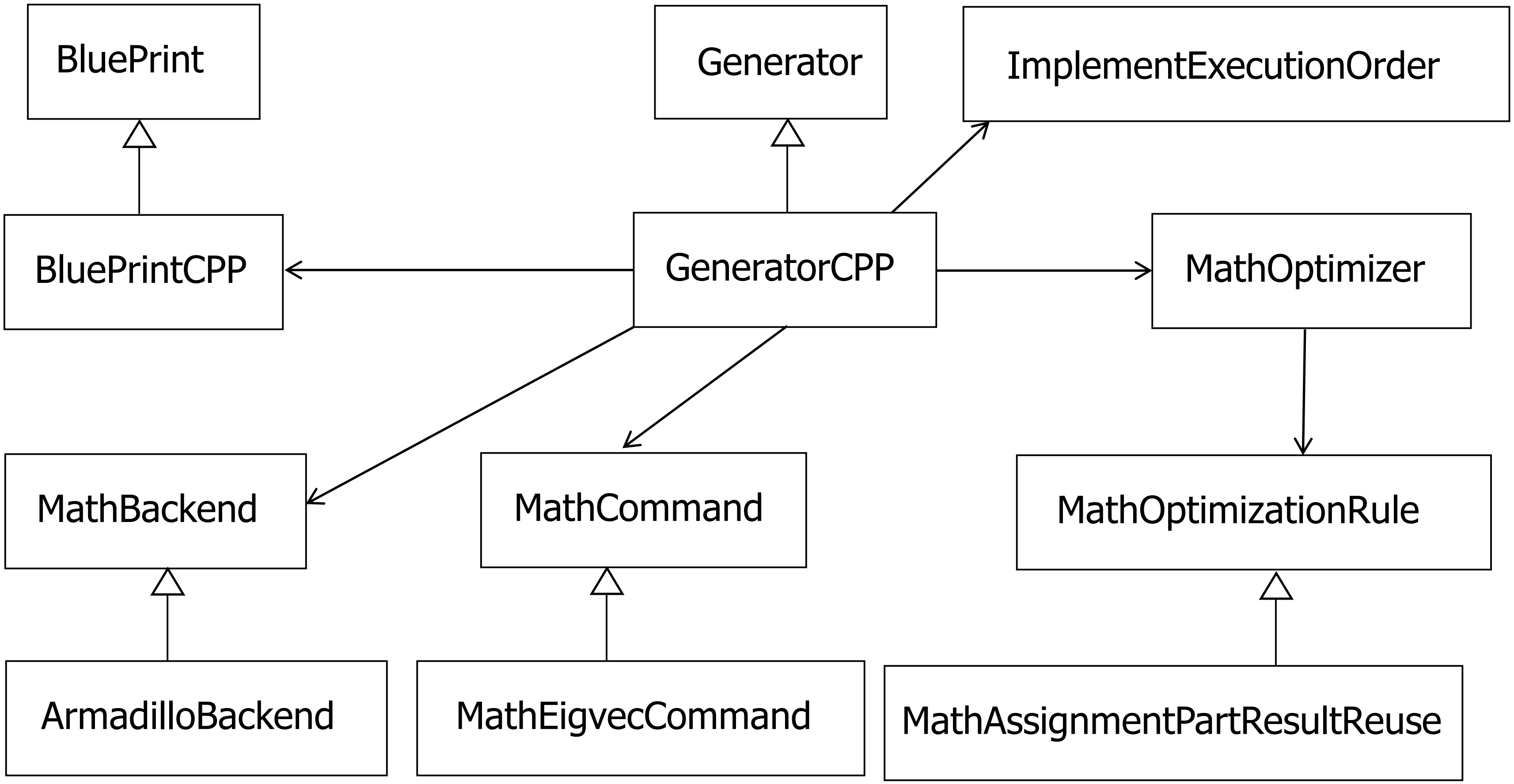
Purpose

- Automatically generate executable C++ code out of EmbeddedMontiArcMath models
- Automatically generate integration code for the MontiSim simulator
- Optimize generated code by utilizing algebraic and threading optimizations

Most Important Classes

- Namely:
  - Generator(GeneratorCPP)
  - BluePrint(BluePrintCPP)
  - MathCommand(MathAbsCommand,MathEigvalCommand)
  - ImplementExecutionOrder
  - MathBackend(ArmadilloBackend)
  - MathOptimizier
  - MathOptimizationRule(MathAssignmentPartResultReuse)

Class Relations



GeneratorCPP

- Represents an instance of the generator
- Can be used to generate the code of a single ExpandedComponentInstanceSymbol
- Can be used to generate all files needed for an ExpandedComponentInstanceSymbol
- Test Coverage: 100%

BlueprintCPP

- Stores all information for one transformed component
- This includes:
  - Variables
  - Methods
  - Includes
- Test Coverage: 74%

MathCommand

- Base class for all math commands
- A math command is a built in function of the generator/EMAM language that is automatically transformed
- This includes functions like ones, atan, eigvec
- Test Coverage: 85%

ImplementExecutionOrder

- Computes the components execution based on the Simulink execution order specification:
  - If a block drives the direct-feedthrough port of another block, the block must appear in the sorted order ahead of the block that it drives.
  - Blocks that do not have direct-feedthrough inputs can appear anywhere in the sorted order as long as they precede any direct-feedthrough blocks that they drive.
- Test Coverage: 93%

MathBackend

- Math library/framework that is used as the backend for matrix and other complex mathematical operations
- Backend classes: ArmadilloBackend, OctaveBackend
- Has methods for retrieving the name of common functions/types in the target language(e.g. matrix type)
- Test Coverage: 100%

MathOptimizer & MathOptimizationRule

- Stores all registered MathOptimizationRules that are applied when the generator uses algebraic optimizations
- Some rules are:
  - MathAssignmentPartResultReuse(a*b+a*b -> c+c with c=a*b)
  - MathDiagonalMatrixOptimizations(matsqrt(D) -> matdiagsqrt(D))
  - MathMatrixMultiplicationOrder(a*b*c -> a*(b*c) if faster)
- Test Coverage:
  - MathOptimizer: 75%
  - MathOptimizationRule: 100%

Software Quality

- Code Quality(according to codeclimate.com): C
- Test Coverage:
  - Whole: 79%
  - Handwritten code: 79%
  - Generated code: (There is no code generated into target folder)

Utilization(Excerpt)

- Used in the following projects:
  - SpectralCluster
  - PacMan
  - MontiSim
  - EMAM2WASM

Threading Optimization

- ImplementExecutionOrder calculates independent subcomponents
- Independent subcomponents can then be executed in different threads
- Can be enabled and disabled depending on input
- Drawback: Too many threads do not provide that much benefit, if not enough cores are available on the target processor due to thread creation overhead
- Future Work: Devise threading algorithm that restructures program execution to only use x threads with (roughly) the same workload

Algebraic Optimizations

- Basic mathematical instruction reordering if amount of operations can be reduced by using the commutative law
- This includes reordering:
  - Multiplications(Matrix and Scalar)
  - Removing unused variable assignments
- Associative law is used to change instructions like AB+AC to A(B+C) if this reduces the amount of operations
- All equations from a component are taken into consideration
- Does also use matrix properties like diagonal to automatically compute fast matrix inverse and fast matrix squareroot

Prof. Dr. B. Rumpe
Lehrstuhl für
Software Engineering
RWTH Aachen

Algebraic Optimization Example

Original:

EMA

```
component MatrixModifier {
 ports in Q(-oo:oo)^{1000,2} mat1,
       in Q(-oo:oo)^{2,1000} mat2,
       in Q(-oo:oo)^{1000,2} mat3,
       in Q(-oo:oo)^{2,10000} mat4,
       in Q(-oo:oo)^{2,10000} mat5,
       out Q(-oo:oo)^{1000,10000} matOut;

 implementation Math{
    Q^{1000,1000} h1 = mat1 * mat2;
    Q^{1000,1000} h2 = mat3 * mat4;
    Q^{1000,1000} h3 = h1 * h2;
    matOut = h3 * mat5;
 }
}
```

Algebraic Optimization Example

Transformed:

EMA

```
component MatrixModifier {
 ports in Q(-oo:oo)^{1000,2} mat1,
       in Q(-oo:oo)^{2,1000} mat2,
       in Q(-oo:oo)^{1000,2} mat3,
       in Q(-oo:oo)^{2,10000} mat4,
       in Q(-oo:oo)^{2,10000} mat5,
       out Q(-oo:oo)^{1000,10000} matOut;

 implementation Math{
    matOut = (mat1*(mat2*mat3))*(mat4*mat5);
 }
}
```

Future Work

- Refactoring and cleanup(Remove deprecated methods)
- Integrate utilizing projects as tests to improve test coverage and robustness
- Implement additional optimizations