DSL and Source to Source Compilation

Hands-on-approach
with
Clava+LARA

Pedro Pinto, João Cardoso

2018-11-03 – PACT 2018

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

SPeCS
Special-Purpose Computing
Systems, languages and tools

Outline

- Overview
  - Clava
  - LARA
- Hands-on approach
  - First LARA Strategy (Call Graph)
  - Code Instrumentation (Logging)
  - Clava APIs (Time and Energy Measurement)
  - CMake Plugin (Auto-Parallelization)
  - Third-party Libraries (DSE)
  - ANTAREX component Integration (mARGOt)

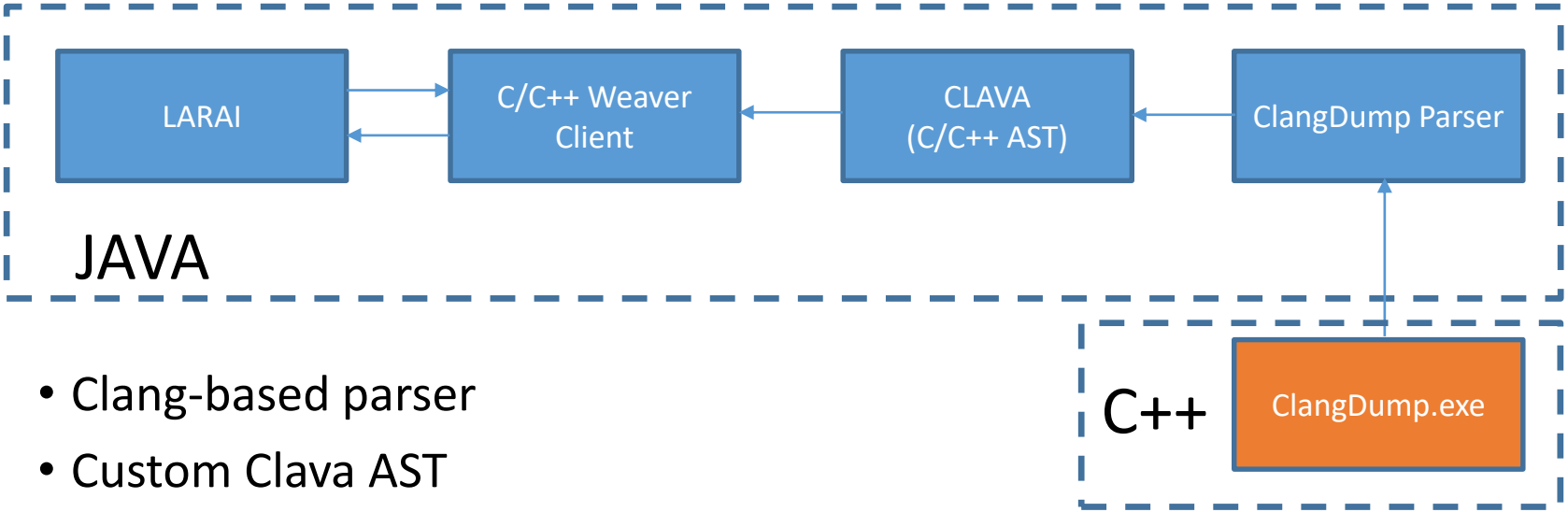
Clava Installation

- All platforms: download JAR
  - http://specs.fe.up.pt/tools/clava.jar
- Linux: script clava-update
  - http://specs.fe.up.pt/tools/clava/clava-update
- CMake Plugin:
  - https://github.com/specs-feup/clava/tree/master/CMake

Clava

- Source-to-source C/C++ compiler (weaver)
- User-defined strategies written in LARA
- Several kinds of strategies possible
  - Analysis, Generation, Insertion, Modification
- Open-source
  - github.com/specs-feup/clava

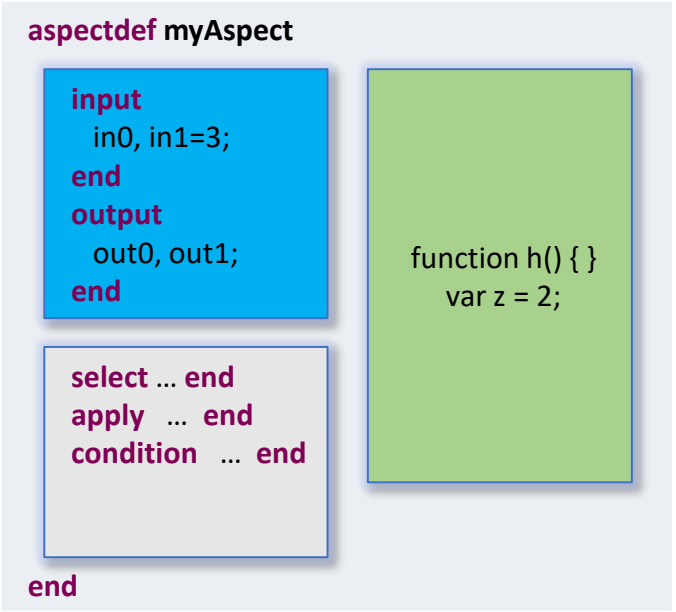
Clava - Toolflow



- Clang-based parser
- Custom Clava AST
  - AST-based transformations
- LARA framework

The LARA Language

- JavaScript-based language
- Strategies written separately from application logic code
- Not tied to a specific target language
  - *Weavers* binds LARA code to a target language
  - Current languages: Java, **C**, **C++** and MATLAB



J. M.P. Cardoso, T. Carvalho, J. G. de F. Coutinho, W. Luk, R. Nobre, P. C. Diniz, Z. Petrov, "**LARA: An Aspect-Oriented Programming Language for Embedded Systems,**" in *Int'l Conf. on Aspect-Oriented Software Development (AOSD'12)*, Potsdam, Germany, March 25-30, 2012.

Main LARA Features

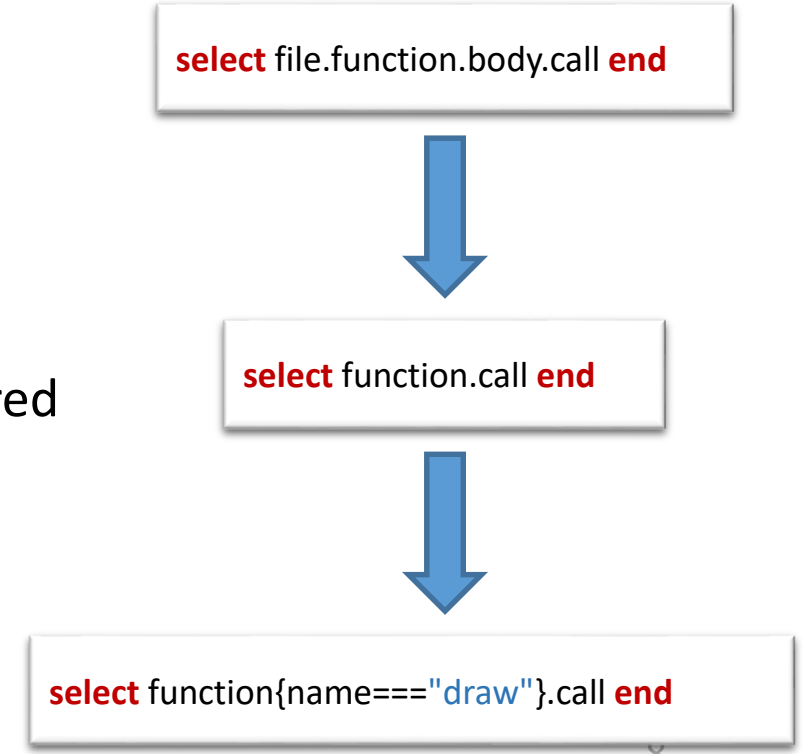
- **Declarative select-apply** clauses
  - **Select** points of interest in the code
  - **Apply** code transformations over them
- Modularity and reuse based on **calling aspects** and **using parameters**
- **Composition of strategies** based on other strategies

```
select method end
apply
    …
end
```

```
apply
  call LoopTiling(64);
  call Timer("ns");
end
```

LARA Select

- Access points on the source code
- Uses an hierarchical point chain
  - Defined in the language specification
- Points not present in the chain are inferred
- Filtering based on attributes



LARA Apply

- Iterates over the selected points (prefixed with $)
- Any point in the select statement can be accessed
- Can access point attributes
- Can change the application

insert before|after|replace

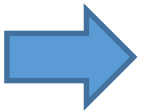
- For injecting code in input application source code

exec

- For executing a compiler action

def

- For defining the value of an attribute

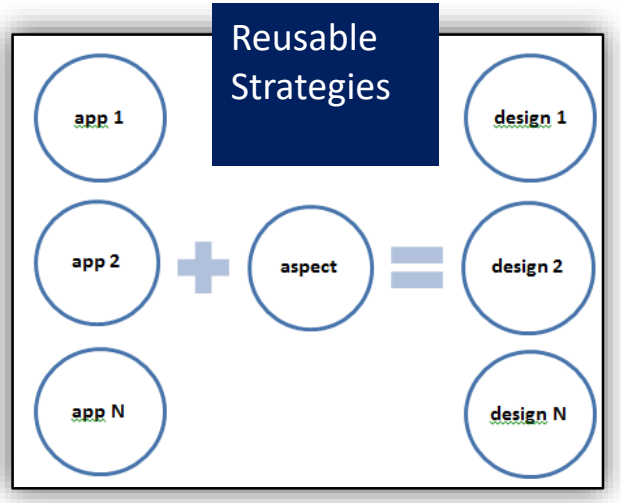


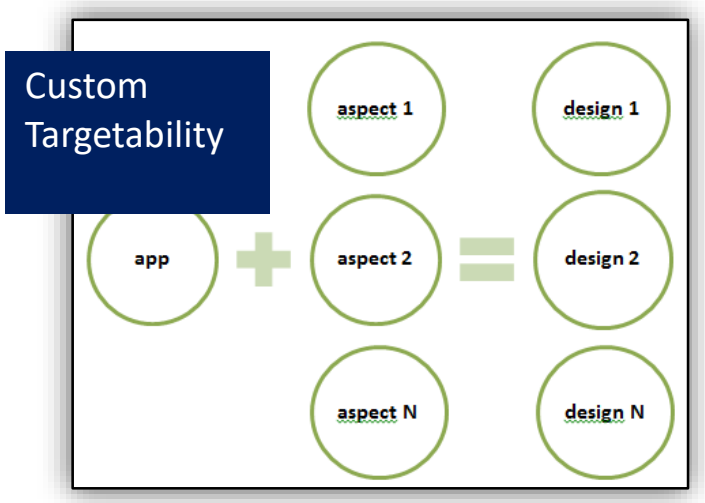
```
select function{name=="draw"}.call end
apply
  $call.insert before 'code to inject';
  insert before 'more code';
end
```

```
$loop.exec interchange($innerLoop);
```

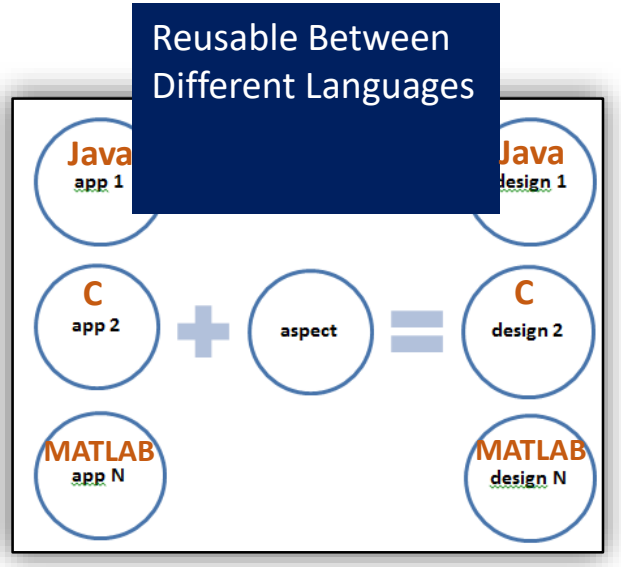
```
$var.def  type='float';
```

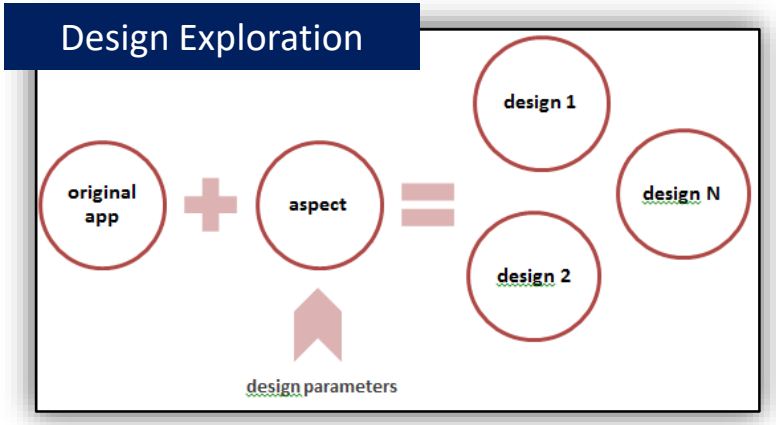
LARA Reusability and Targetability





LARA Reusability and DSE





Hands-on – Getting Started

- Linux
  - Get script: specs.fe.up.pt/tools/clava/clava-update
  - Put in path and run (may require sudo)
- Windows
  - Get jar: specs.fe.up.pt/tools/clava.jar
  - Get Cmake files: https://bit.ly/2EVVnF7
- Tutorial files
  - specs.fe.up.pt/tutorials/PACT2018.zip

Hands-on

- 1. CallGraph

- Objectives
  - Read some LARA code
  - Load and run a LARA strategy

Hands-on

- 2. Logging

- Objectives
  - Write some LARA code (exercise)
  - Introduction to LARA APIs

Hands-on

- 3. Measurements

- Objectives
  - Use LARA APIs (exercise)
  - Introduction to Clava documentation

Hands-on

- 4. AutoPar

- Objectives
  - Auto-parallelize code
  - Introduction to CMake plugin

Hands-on

- 5. Exploration

- Objectives
  - Use the CMake plugin
  - Perform Design-Space Exploration

Hands-on

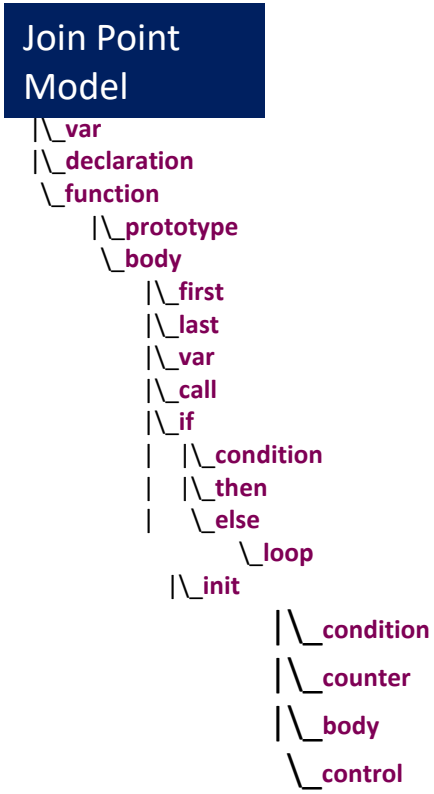
- 6. mARGOt Integration

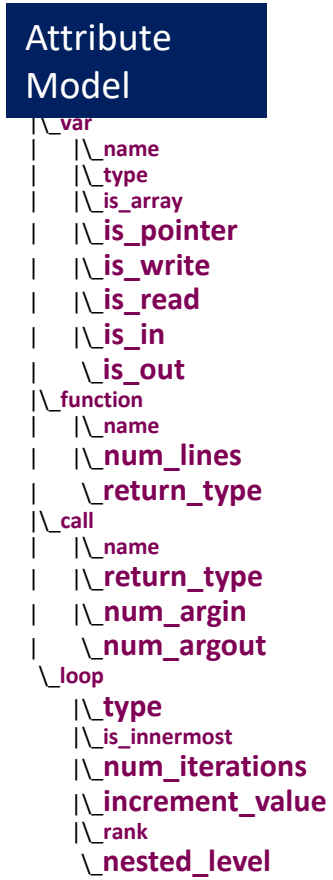
- Objectives
  - Use mARGOt Clava API to:
    - Generate configuration
    - Perform exploration
    - Instrument the code
  - Run the enhanced application

Backup Slides

The LARA Language

- Join Point Model
  - Allows the front-end to adapt to other target programming languages
- Attribute Model
  - Allows LARA to access join point values and to associate values to join points
- Action Model
  - Allows LARA to express actions





AOP Approach

- Several AOP languages
- No reusability between AOP languages
- Flexibility on the join point capture
- Include the support of code transformations

Concerns related to code transformations and compiler optimizations:

- Performance, Power, Energy
- Parallelism, Concurrency
- Monitoring, Test, Debug
- Safety, Security
- Targeting hardware accelerators, multicore and manycore architectures
- Different tool flows
- Fully explore compiler optimizations

Existence of many crosscutting concerns!

LARA Compilation Flow