MarGotAspect

An AspectC++ generator for the auto-tuning framework Margot

Arcari Leonardo
Galimberti Andrea
@ Politecnico di Milano

Project context

Road to new computing architectures

Processors technology has encountered a **clear threshold** in the impossibility of getting more performance by means of just increasing the frequency or the complexity of a processor, especially because of **power limitations**.

In recent years, performance improvements have instead been obtained by:

- **multiprocessor** architectures, **clustering** many processors whose coordination and usage is controlled by a **single operating system** and that usually share memory through a **shared address space**;
- **heterogeneous** architectures, which exploit **coprocessors** in order to handle particular tasks more efficiently than general-purpose processors.

HPC and Embedded Systems

In the **High Performance Computing** scenario, whose goal is to reach the **Exascale level** in the next few years, it is fundamental for the design of computing systems to not just increase the computational performance, but even more importantly to **increase** the **energy efficiency** (FLOPS/Watt).

Performance-to-power ratio is also a key factor in the **Embedded Systems** field, historically characterized by **low-power constraints**, but in which multiprocessor solutions have more recently started to be widely used as **performance requirements** have been steadily increasing.

Solution: application adaptivity and autotuning

One solution to the HPC performance/power efficiency problem is:

- at **compile-time**, to express strategies with regards to **adaptivity**, **energy** and **performance** of the system;
- at **run-time**, to enforce application **auto-tuning** and both **resource** and **power management**.

One example of a framework providing these functionalities is the mARGOt framework.

mARGOt framework

Provides **dynamic adaptivity** to an application, in order to face changes in its execution environment or in its requirements.

Exploits the information gathered during a DSE in order to guide the selection of the most suitable configuration of the application **software knobs**.

Defines a set of **constraints** on the application performance and a **rank function** that expresses how good is an **operating point** for the application.

Senses the execution environments at run-time, by means of run-time **monitors**, and automatically selects the best **configuration** that fits the observed situation.

mARGOt framework

But what if we want to “**manually”** configure the adaptivity and auto-tuning of a code block while the application is running?

- through a message protocol, or
- through the update of some variable inside the application, which hasn’t been configured as a mARGOt software knob

The goal of our project is to achieve this functionality, in the context of C/C++ applications, by exploiting the **code-insertion** capability given by Aspect-C++.

AspectC++

Aspect C++ is a language following the **aspect-oriented programming paradigm**, and allows to insert code blocks by means of expression matching.

A block of code, called **advice**, is associated to specific **join points** (e.g., the assignment of a variable, a function call, the construction of an object) through matching expression called **pointcuts**.

mARGOt - AspectC++ integration

We are writing advice code containing mARGOt framework function calls and which can be associated to **different use cases**.

This **autotuning code** would otherwise have to be **manually inserted** into an application, instead we are leaving the application **code completely unmodified**.

A **separate aspect file** is going to contain code to:

- start and stop performance metrics monitors;
- change the goals or the state of the framework, based on some condition changing at execution-time.

Project description

Use case (1)

Alright, my functional requirements are fine. But I have no clue about the running time of that *expensive* function. Let's **monitor** it!

Use case (1)

We just need to pick up a **margot::time_monitor** to measure the execution time of function *do_work()*

Steps required so far

```
int main() {
  margot::init();
  for (int i = 0; i<repetitions; ++i) {
    margot::foo::start_monitor();
    do_work(trials);
    margot::foo::stop_monitor();
    margot::foo::log();
  }
}
```

A little bit entangled right? What if we have many functions to monitor invoked many times in our codebase?

What we'd rather see

```
int main() {
  for (int i = 0; i<repetitions; ++i) {
    do_work(trials);
  }
} // Functional code
```

```
// description file
monitor a call to do_work with
argument trials. You know how
to do it!
```

Use case (2)

Oh, but I don’t have all the work I wanna monitor inside a function… I just need to **monitor** this *region of interest*.

Use case (2)

Ok, ok... so it's still the same right? I just need to run some start/stop methods from mARGOt. Oh wait. What if my software knobs should be tuned due to some environmental changes?

Steps required so far

```
int main() {
  margot::init();
  for (int i = 0; i<repetitions; ++i) {
    if (margot::foo::update(trials))
      margot::foo::manager.configuration_applied()
;
    margot::foo::start_monitor();
    // whatever code
    // spanning some lines
    margot::foo::stop_monitor();
    margot::foo::log();
  }
}
```

What we'd rather see

```
int main() {
  int sw_knob = 10;
  margot_bar_start_roi(sw_knob);
  // whatever code
  // spanning some lines
  margot_bar_end_roi();
} // Functional code
```

```
// description file
monitor region of interest from here
to there. These are the software
knobs we use.
```

Use case (3)

Great! My function now returns within *600 ms*. Too bad that power consumption is high. I need to lower the performance **when on battery charge**...

Use case (3)

That's easy. A simple call to **goal::execution_time_goal.set()** under some condition does the trick. Right?

Steps required so far

```
bool battery_energy = false;
int main() {
  margot::init();
  for (int i = 0; i<repetitions; ++i) {
    // check if the configuration is different wrt the
    // previous one
    if (margot::foo::update(trials)) {
      margot::foo::manager.configuration_applied();
    }
    if (!battery_energy) goal::execution_time_goal.set(600);
    else goal::execution_time_goal.set(1000);
    /* monitor do_work */
  }
}

/* changes to battery_energy status somewhere */
```

What we'd rather see

```
int main() {
  for (int i = 0; i<repetitions; ++i) {
    do_work(trials);
  }
} // Functional code
```

```
// description file
set goal to 600 when
battery_energy changes to
false. When it changes to true
it's time to slow down. Set
goal to 1000
```

Use case (4)

Alright, after some domain analysis I identified 3 possible **environment states** that require different sets of constraints. I just need to **add some conditions**...

Use case (4)

Cool, mARGOt helps me. I *just* describe the set of constraints and I can change the state according to current system status...

Steps required so far

```
int status = 0;
int main() {
  margot::init();
  for (int i = 0; i<repetitions; ++i) {
    if (margot::foo::update(trials))
      margot::foo::manager.configuration_applied();
    if (status == 0)
      manager.change_active_state("first_state");
    else if (0 < status && status < 3)
      manager.change_active_state("second_state");
    else if (status >= 3)
      manager.change_active_state("third_state");

    /* monitor do_work */
  }
}
/* changes to status somewhere */
```

What we'd rather see

```
int main() {
  for (int i = 0; i<repetitions; ++i) {
    do_work(trials);
  }
} // Functional code
```

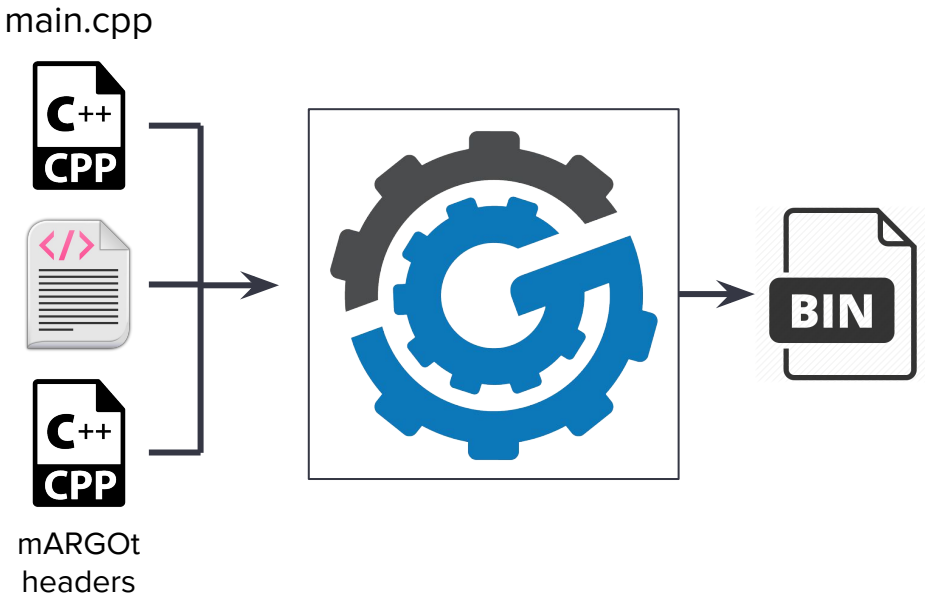
```
// description file
Watch changes to status. In case of
change apply following rules:

status = 0 → state: first_state
0 < status < 3 → state: second_state
status >= 3 → state: third_state
```

MarGotAspect (1)

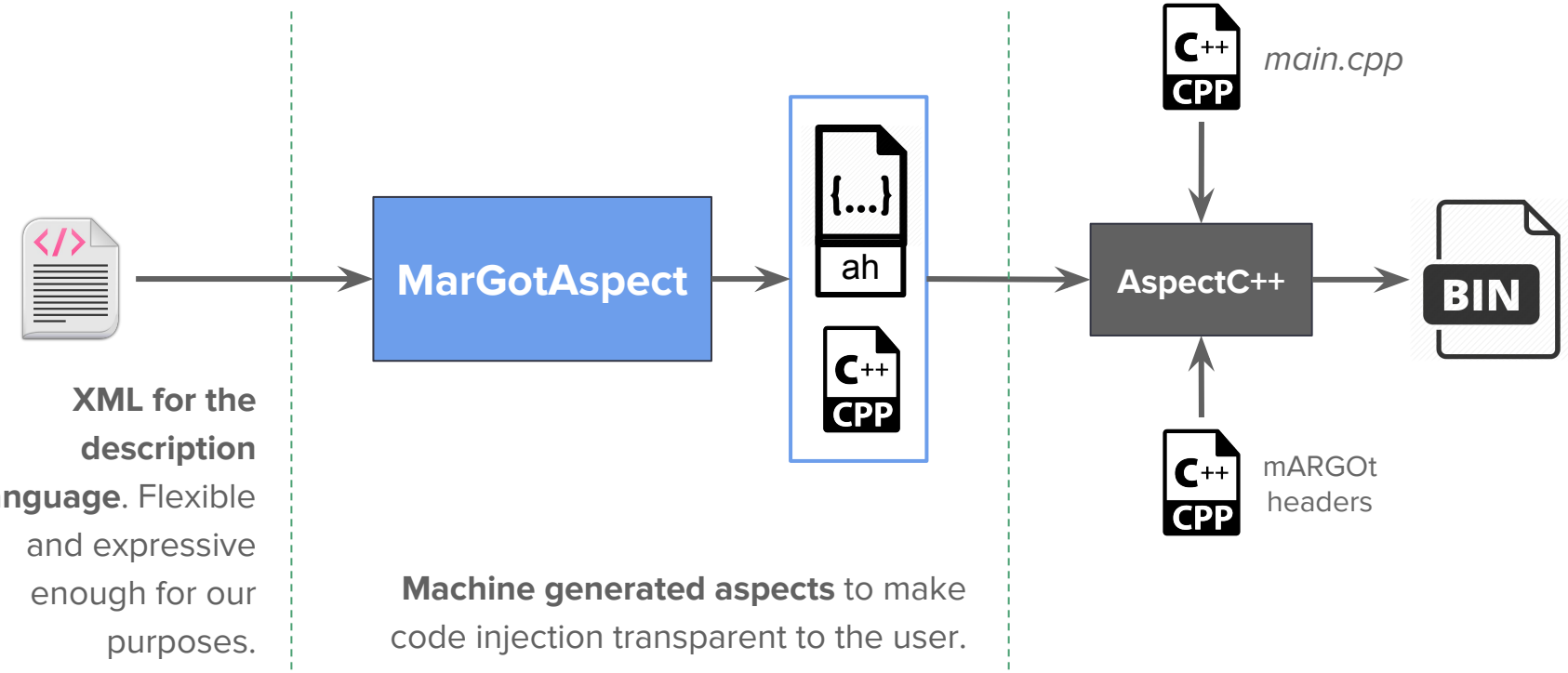
is an attempt to exploit Aspect Oriented Programming within the mARGOt framework in a transparent way to the application developer.

Ideal features:

- Write your functional code.
- Describe non-functional requirements *somewhere else!*
- MarGotAspect generates the AspectC++ code for you.
- Weave them together.



MarGotAspect (2)



XML for the description language. Flexible and expressive enough for our purposes.

Machine generated aspects to make code injection transparent to the user.

Demo