Towards a Rule-Based Approach to Generate High-Performance Scientific Code

Guillermo Vigueras, Salvador Tamarit, Manuel Carro, and Julio Mariño

Programming Heterogenous Systems

Strong trend towards the integration of various types of computing elements (FPGA, GPU, DSP, ...).

- **Benefits**: Cost-effective alternative to more traditional architectures.
- **Drawbacks**:
  - *Additional complexity in hardware and software:* developers must take care of different architecture-independent features.
  - *Programming these systems is restricted to a few experts:* hinders widespread adoption and increases the likelihood of bugs.
  - *Portability is greatly limited:* adapting existing scientific algorithms is laborious.

Approach

Framework for sound, mechanical transformation of programs written in an architecture-agnostic way to versions suitable for heterogeneous platforms.

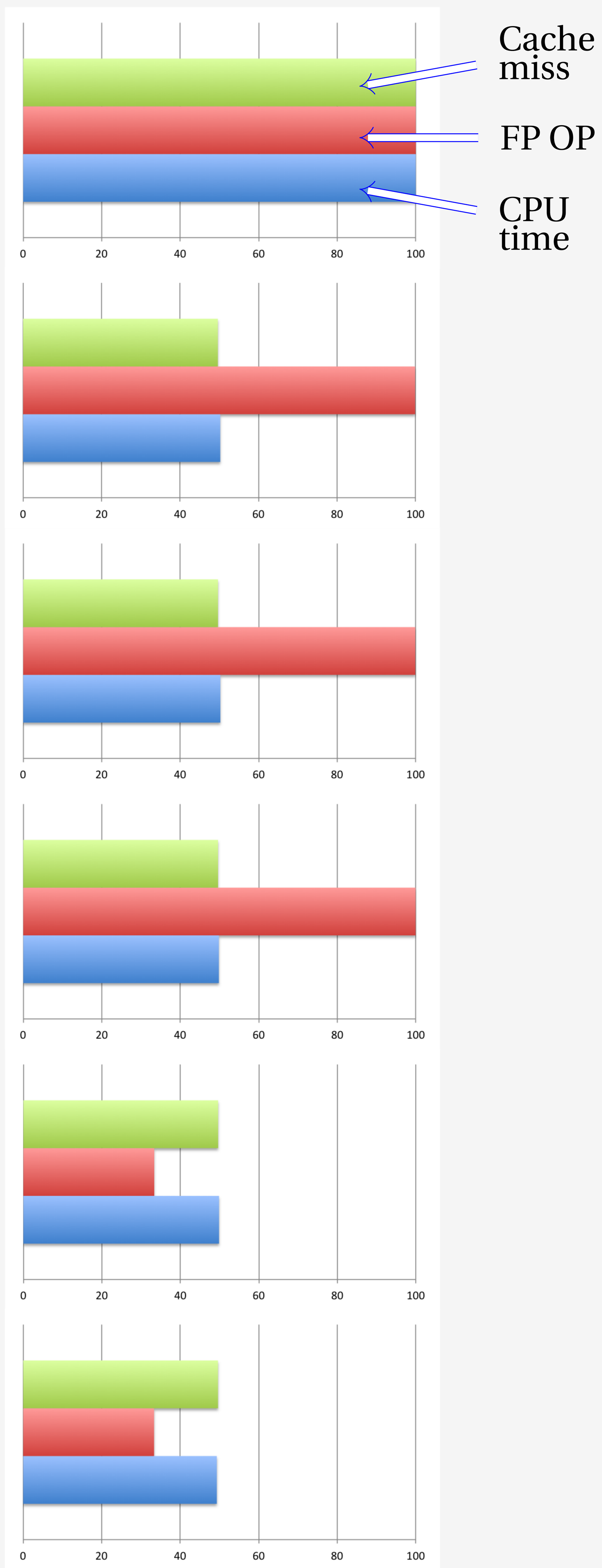
- **Transformation steps**: modeled with rules.
  - *Extensible:* written in a C-like language (inspired by CTT and CML).
  - *Metrics:* measurement of impact on *adequacy properties*.
- **Rule firing**:
  - Syntactic and semantic conditions should be fulfilled.
  - Metrics approximate the impact of applying a given transformation on run-time non-functional properties.
  - Used to decide candidate rules to be applied by guiding a heuristic search.
- **Program properties**:
  - Stated by hand with pragmas, or
  - Automatically inferred.

Example: av+bv ⇒ (a+b)v

```
#pragma def is_vc_space(V,[c,v])
#pragma def vc_space(V,F,+,*)
#pragma def is_sc_field(F,[a,b])
#pragma def sc_field(F,float,+,-,*,/)
```

Legend:
Code before
Code after

Compilation with -O3




```
float c[N], v[N], a, b;

for(int i=0;i<N;i++)
  c[i] = a*v[i];

for(int i=0;i<N;i++)
  c[i] += b*v[i];
```

(For-LoopFusion)

```
for(int i=0;i<N;i++) {
  c[i] = a*v[i];
  c[i] += b*v[i];
}
```

(SplitAdditionAssign)

```
for(int i=0;i<N;i++) {
  c[i] = a*v[i];
  c[i] = c[i] + b*v[i];
}
```

(JoinAssignments)

```
for(int i=0;i<N;i++){
  c[i] = a*v[i] + b*v[i];
}
```

(UndoDistribute)

```
for(int i=0;i<N;i++){
  c[i] = (a + b) * v[i];
}
```

(LoopInvCodeMotion)

```
float k = a + b;
for(int i=0;i<N;i++){
  c[i] = k * v[i];
}
```

Transformation Rules and Rule Language

$$\texttt{for}\ (l{=}e_{ini}\,;rel(l,e_{end})\,;mod(l))\,\{s_1\}$$
$$\texttt{for}\ (l{=}e_{ini}\,;rel(l,e_{end})\,;mod(l))\,\{s_2\} \Rightarrow \texttt{for}\ (l{=}e_{ini}\,;rel(l,e_{end})\,;mod(l))\,\{s_1\,;s_2\}$$

when $s_1 \not\mapsto \{s_2, e_{ini}, e_{end}\}$, rel pure, $(s_1;s_2) \not\mapsto \{l, e_{ini}, e_{end}\}$, writes$(mod(l)) = \{l\}$
metrics conc(), **task**()

(For-LoopFusion)

$$s_1;\ l = e_1;\ s_2;\ l = e_2;\ s_3; \Rightarrow s_1;\ s_2;\ l = e_2[e_1/l];\ s_3;$$

when $s_2 \not\mapsto\ l, s_2 \not\leftarrow\ l, s_2 \not\mapsto\ e_1, e_1$ pure

(JoinAssignments)

$$f(g(e_1,e_3), g(e_2,e_3)) \Rightarrow g(f(e_1,e_2),e_3)$$

when e_1, e_2, e_3 pure, g distributes_over f

(UndoDistribute)

```
join_assignments {
 pattern: {
  cstmts(body0_1);
  cexpr(v1) = cexpr(val_v1);
  cstmts(body0_2);
  cexpr(v1) = cexpr(val_v2);
  cstmts(body0_3);
 }
 condition: {
  no_mod_use(cexpr(v1),cstmts(body0_2));
  no_mod(cstmts(body0_2),cexpr(val_v1));
  pure(cexpr(val_v1));
 }
 ...
```

```
 ...
 generate: {
  cstmts(body0_1);
  cstmts(body0_2);
  cexpr(v1) = subs(cexpr(val_v2),
    cexpr(v1), cexpr(val_v1));
  cstmts(body0_3);
 }
 metrics: {
  metric_1: delta_metric_1;
  metric_2: delta_metric_2;
  ...
}
```

Summary and Future Work

- **Distinguishing features**:
  - Focus on scientific code.
  - Aiming at heterogeneous platforms.
  - Use of mathematical properties of the code.
  - Quantitative measures of non-functional, run-time properties guide which transformations apply.
- **Tool under implementation** (see QR code).

polca-project-contact@software.imdea.org

FP7-ICT-2013.3.4
Ref: 610686
www.polca-project.eu

imdea software institute

POLITÉCNICA