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.

### Programming Heterogenous Systems

For the sake of readability and clarity, the textual content is presented here:

**Example:** $av + bv \Rightarrow (a+b)v$

```c
#pragma def is_sc_field(F, [a, b])
#pragma def sc_field(F, float, +, *, /)

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

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

**Legend:**
- Code before
- Code after

### 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.

### Transformation Rules and Rule Language

```c
for (lmod(i); rel1; rel2; mod(l)) {s1} for (lmod(i); rel1; rel2; mod(l)) {s2} \rightarrow for (lmod(i); rel1; rel2; mod(l)) {s1; s2}
```

```c
when \( s1 = \{ s2; rel1; mod2; rel3; mod4; \} \), pure \( t1; t2; \) \( \Rightarrow \) \( lmod(l); \) writes(\( mod(l) \)) = \( l \)
```

```c
metrics conc, task
```

```c
join assignments { cstmts(body0.1); cexpr(val1.vl); cstmts(body0.2); cexpr(val2.vl); cexpr(val1.vl) = subs(cexpr(val2.vl), cexpr(val1.vl)); cstmts(body0.3); }
```

```c
condition { no_mod(vl, cstmts(body0.2)); no_mod(body0.2, cexpr(val1.vl)); pure(cexpr(val1.vl)); }
```

```c
... generate: { cstmts(body0.1); cstmts(body0.2); cexpr(val1.vl) = subs(cexpr(val2.vl), cexpr(val1.vl)); cstmts(body0.3); }
```

```c
metrics: { metric1: delta.metric1; metric2: delta.metric2; }
```

```c
... 
```

### 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).

---

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

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

**Example:** $av + bv \Rightarrow (a+b)v$

![Diagram showing code transformations](https://www.polca-project.eu)