

Sketch of Research Plans Related to WG 2.11 – Program Generation

Jörg Striegnitz
Central Institute for Applied Mathematics and
John von Neumann Institute for Computing
Research Centre Jülich, Germany
J.Striegnitz@fz-juelich.de

1. Applied Program Transformation (medium-term project)

A huge class of modern Supercomputer architectures comes with a deeply nested cache hierarchy. Optimizing programs to utilize such cache architectures best is well known to be a hard problem. The situation is getting even worse with shared memory systems where parts of the cache hierarchy are shared among different CPUs (e.g. IBM's p690 series).

In the case of shared caches, if multiple programs run on a single machine, program optimization and data layout heavily depends on the number of processes to start, and how processes are mapped to CPUs (Caches) -- usually this mapping is performed by a special batch system and therefore changes from run to run.

We plan to develop a program transformation engine that optimizes programs right after the scheduler has assigned a set of CPUs to the associated job. The idea is to embed a DSL into an existing language to manipulate / generate code according to the available runtime resources. In effect, program execution will become a staged process as the transformation engine is also responsible for the compilation of the source.

The results of this project will also be applicable to Grid Computing, where the whole computing environment of an application may change from run to run. With respect to Grid Computing we also consider to check the applicability of our approach for heterogeneous environments, where different systems are involved at runtime.

2. Embedding DSLs into Existing Languages (long-term project)

Structural type analysis (aka intensional polymorphism) provides good means to embed DSLs into existing languages. The key idea of this approach is a dual representation of the DSL's abstract syntax tree: at the type-level it occurs as a nested type-constructor-term, whereas it is a value of exactly that type at the term level. Effectively, the type system acts as partial evaluator which translates the DSL into the host language.

Unfortunately, C++ is the only wide-spread language that allows for this approach and it may come as a surprise that it has not been designed with this application in mind.

Existing calculi to support structural type analysis have been designed carefully to offer decidable type systems. The type system itself is isomorphic to a simply typed lambda calculus without a fixed point combinator, but enriched with special constructs to define types and values by induction over the structure of monomorphic types. However, there are situations where the type-level language doesn't offer enough expressiveness to encode a translator for a DSL.

In this project we seek for answers to the following questions:

- How to effectively support structural type analysis for languages with separate compilation?
- What kind of DSL features demand the host system having an undecidable type system?
- Is it possible to enrich the type system to support such features while avoiding undecidability?
- Can we integrate structural type analysis into an existing language (like FORTRAN) such that we can embed DSLs, like we can do with C++? Will this new language be fully compatible with the FORTRAN standard?
- Is it possible to define a minimal language that can be extended by DSLs to become a superset of a more complex language (like FORTRAN and C)?