Parallel programming with SkIml

Quentin Carbonneaux François Clément Pierre Weis

INRIA

April 19th, 2013

Skeleton programming

Traditional approaches to parallelism (MPI, OpenMP)

- intrusive: mix sequential instructions with parallel primitives;
- low level notations and concepts;
- fine tune of parallelism; very efficient parallel programs;
- error prone: very demanding in programming/debugging effort.

SkIml approach

- non intrusive: parallel code is apart from sequential code;
- skeleton combinators: high level parallel programming schemes;
- skeleton algebra: compositional description of parallelism;
- reliable: deterministic parallel execution;
- Domain Specific Language embeded in OCaml.

Skeleton algebra

- skeletons are functions over data streams;
- coarse grain parallelism;
- task parallel combinators: `pipe`, `farm`;
- data parallel combinators: `prod`, `sum`, `farm_vector`, `rails`;
- control combinator: `loop`.

Safety

- well defined semantics: given by the sequential interpretation;
- proof feasibility: proofs for all basic combinators imply proofs for all programs;
- weak adequacy theorem: sequential and parallel versions are compiled from the same source code;
- strong adequacy theorem: sequential and parallel versions always give the same results.

Skeletons in practice

Development methodology

- develop and debug using the sequential semantics;
- run heavy computations in parallel after a simple recompilation.

Example

Deploy `nw` independent workers computing `f`, then compose `g`:

```
farm (skl () -> f, nw) ||| skl () -> g;;
```

Abstraction over combinators

`make_domain`: specialized combinator for domain decomposition.

Foreign languages (C, C++, Fortran)

External communication layer: Pio (polyglot I/O library).

SkIml is free software available at `http://sklml.inria.fr/`.