Loki: A Polymorphic Array of Simple Processors
Daniel Bates, Alex Bradbury, and Robert Mullins

Overview

Loki is an all-purpose, robust, homogeneous computing fabric. It consists of hundreds or thousands of individual cores and memories interconnected by a chip-wide network. It is targeted at embedded computing applications.

Characteristics:
Loki aims to combine the flexibility of an FPGA approach with the high performance and low power of an ASIC, while remaining a good target for high-level languages. The network-centric design blurs the boundaries between cores.

Specialisation-execution:
- Communication-exposed architecture allows fine-grained control
- Group cores together at run-time to form more powerful processors or accelerators

Architecture

Execution Patterns

Motivation

Challenges:
- Power limits require a smaller percentage of transistors to be active
- Need to accept device failures
- Poor interconnect scaling
- Amdahl’s law
- Design and verification cost
- Complexity
- Greater levels of control flow in new embedded applications

Aims:
- Explore the design space between FPGAs and many-core processors
- Our goal is a simple, robust, any-purpose fabric
- Power efficiency is achieved by specialising:
  - execution datapaths
  - communication
  - memory subsystem
- Purely software-programmable

Design Space Exploration

Determine the best possible design for any combination of constraints:
- Parametrise the design as much as possible
  - Eventually, explore using architectural transformations
  - Perform a wide sweep over the design space
  - Using the results of exploration, determine the best targets for optimisation and/or innovation

The design space is too large to explore thoroughly:
- It must be carefully constrained, sampled and pruned
- Interesting approaches to exploration include machine learning, genetic algorithms, tabu search, and simulated annealing

As in logic synthesis, the compilation process may be influenced by user-specified constraints (e.g. performance, area, power).

Harnessing run-time data:
- Using the results of run-time execution allows the compiler to make better decisions
- Cores may be used to collect statistics to dynamically improve overlay performance and adapt to changes in input data

Compilation

Language: Loki-C (inspired by ARM’s SoC-C). Supports streaming level parallelism through additions such as code and data placement annotations. Further extensions will add data-level parallelism and other features.

Implementation: LLVM compiler infrastructure with a clang-based frontend.

In addition to the usual phases of compilation, the compiler will:
- Discover parallelism in all its forms
- Exploit static analyses and programmer annotations
- Assign tasks to cores, possibly acting as virtual VLIW, SIMD or data-driven execution engines
- Choose when to dedicate cores as helper engines such as DMA, schedulers, smart cache controllers, and predictors
- Generate memories appropriate for the application, e.g. scratchpads or caches
- Minimise the distance between frequently communicating code
- Construct communication infrastructure, e.g. software routers or FIFOs
- Generate and optimise pipeline control code

Communication scheduling
- Generate code to hide communication latency wherever possible
- Perform final optimisations. These may involve localised layout changes

Overlay generation and load-balancing

Layout ("place and route")