System-level ISA (SISA)

Wenhao Jia, Margaret Martonosi, Princeton University; Kelly Shaw, U. Richmond.

Design Technologies for Concurrent Systems
Task 1.1.2.5

GSRC Annual Symposium

Sep 3-4, 2009

Motivation

- Future CMPs:
  - Increasing core counts
  - Increasing heterogeneity (e.g. special functional units)
  - Runtime events (core failure, per-core V/F scaling, …)
- Current parallelization methods are limited in many aspects:
  - Schedulers designed for relatively few number of cores
  - Poor portability across different hardware
  - Insufficiently responsive to runtime events

SISA Approach

- Coarse-grained, System-level ISA
  - Instructions are aggregated into chunks that expose maximal parallelism
  - Communication and dependences are explicitly marked so that they can be managed and optimized
- Code optimization throughout the lifetime of a program
  - Dynamically adapt to increasing core counts in future generations
- Efficient and flexible runtime task scheduler/manager

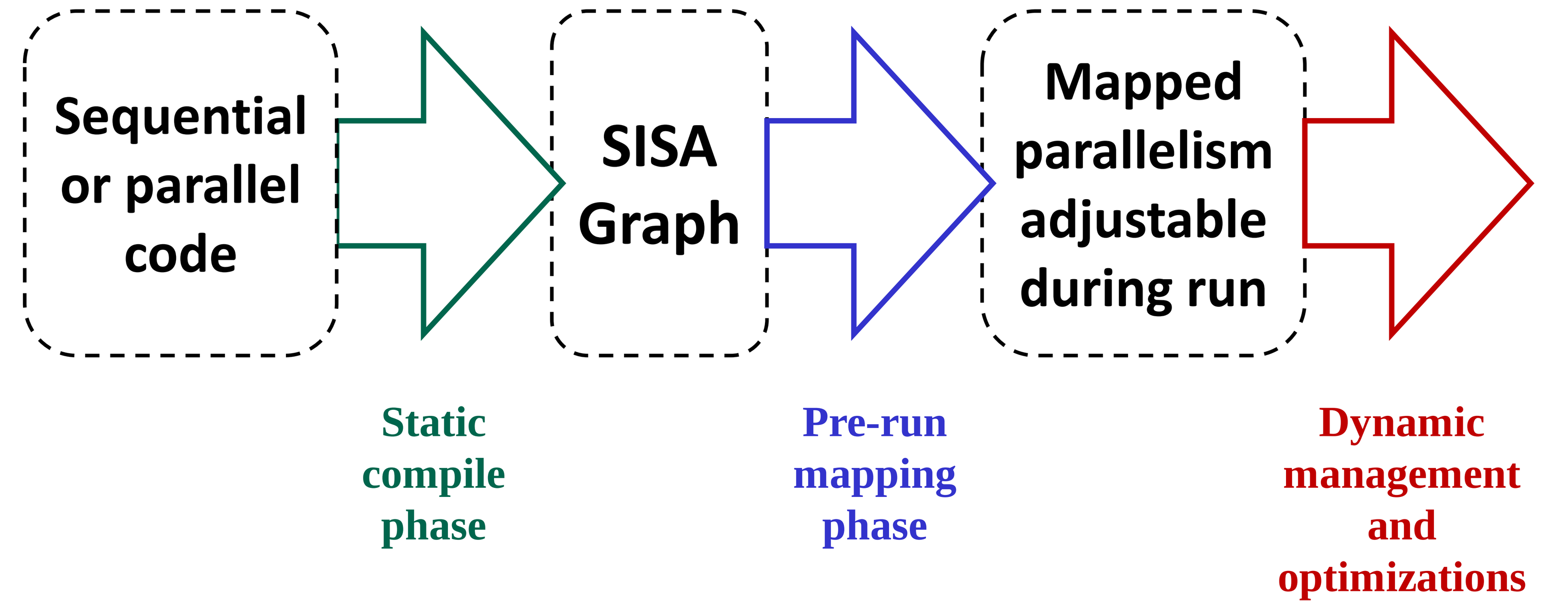
Goals

- Performance portability across varying hardware implementations
- Memory and communication optimizations
- Support for heterogeneity and specialized functional units
- Task migration and restart for reliability events

Acknowledgements

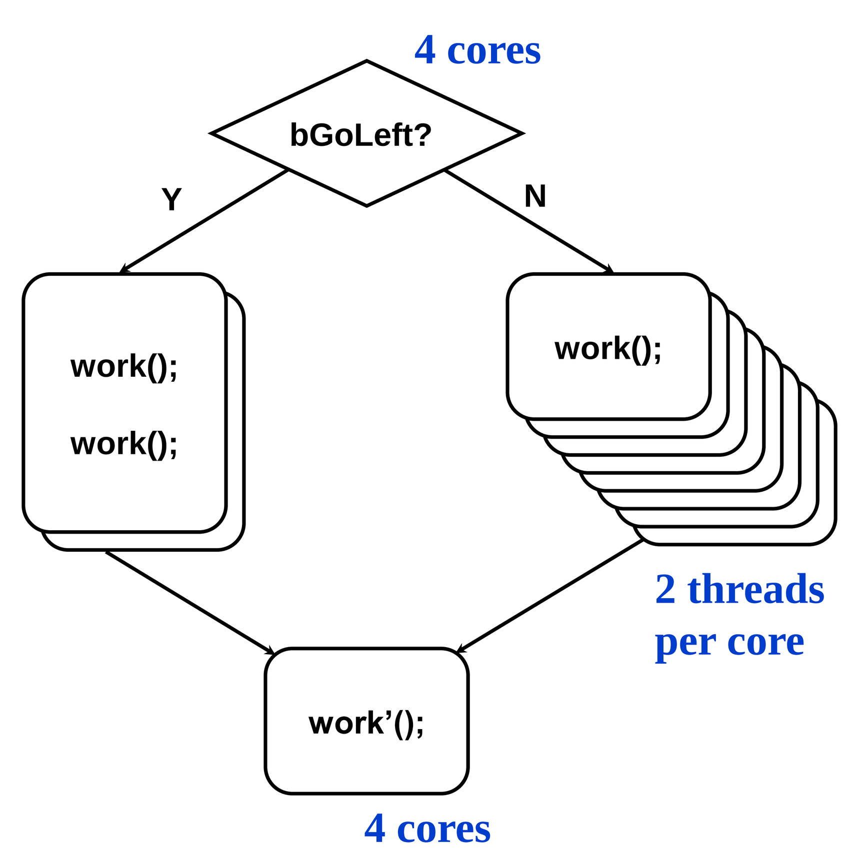
- The authors acknowledge the support of the Gigascale Systems Research Center, one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program.
- In addition, this work was supported in part by National Science Foundation grant CCF-0916971

Framework

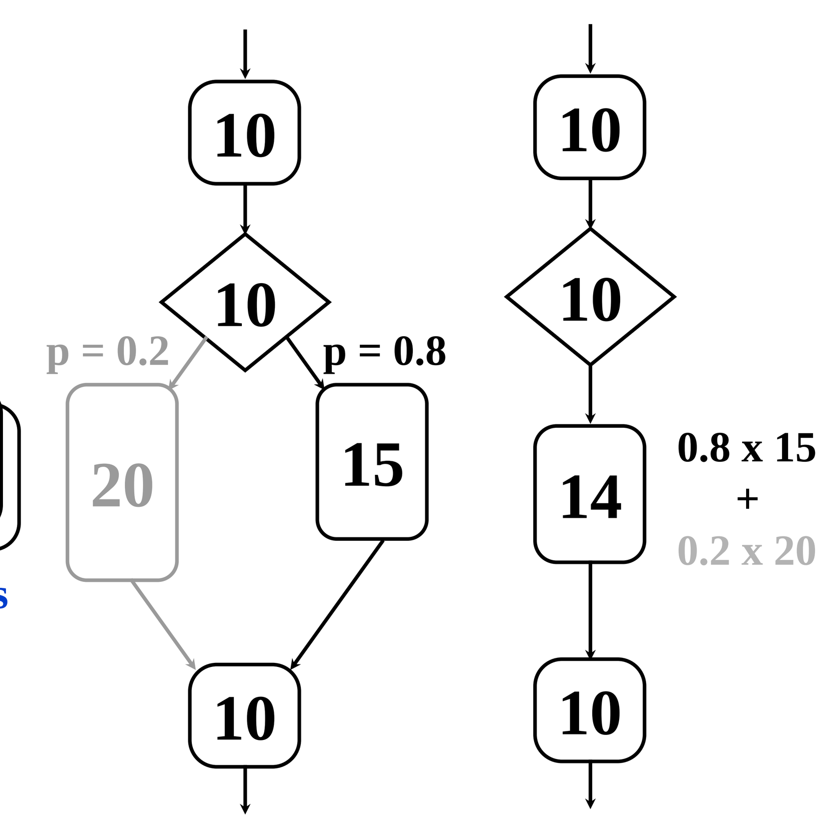


SISA Graph

- Key characteristics
  - Instructions organized into computational chunks
  - Communication flows and data dependences explicitly marked
  - Chunks contain no global side-effects – can be aborted or migrated freely



Core assignment

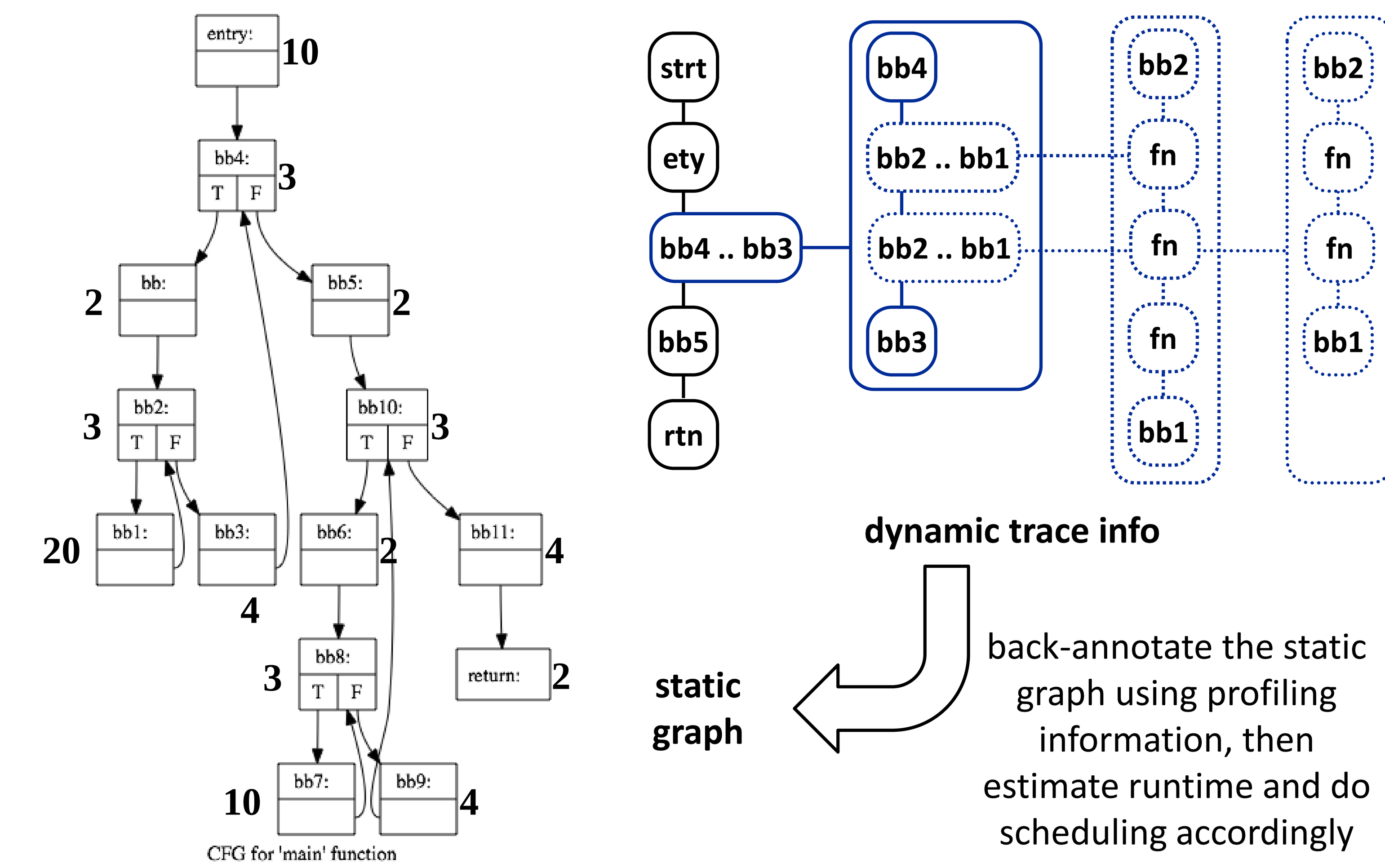


If removal

Dynamic Manager

- SISA's high-level information and abstraction enable more efficient and effective dynamic management
  - SISA-guided Adaptive Parallelism
  - Memory mapping and communication optimization
  - Chunk criticality prediction

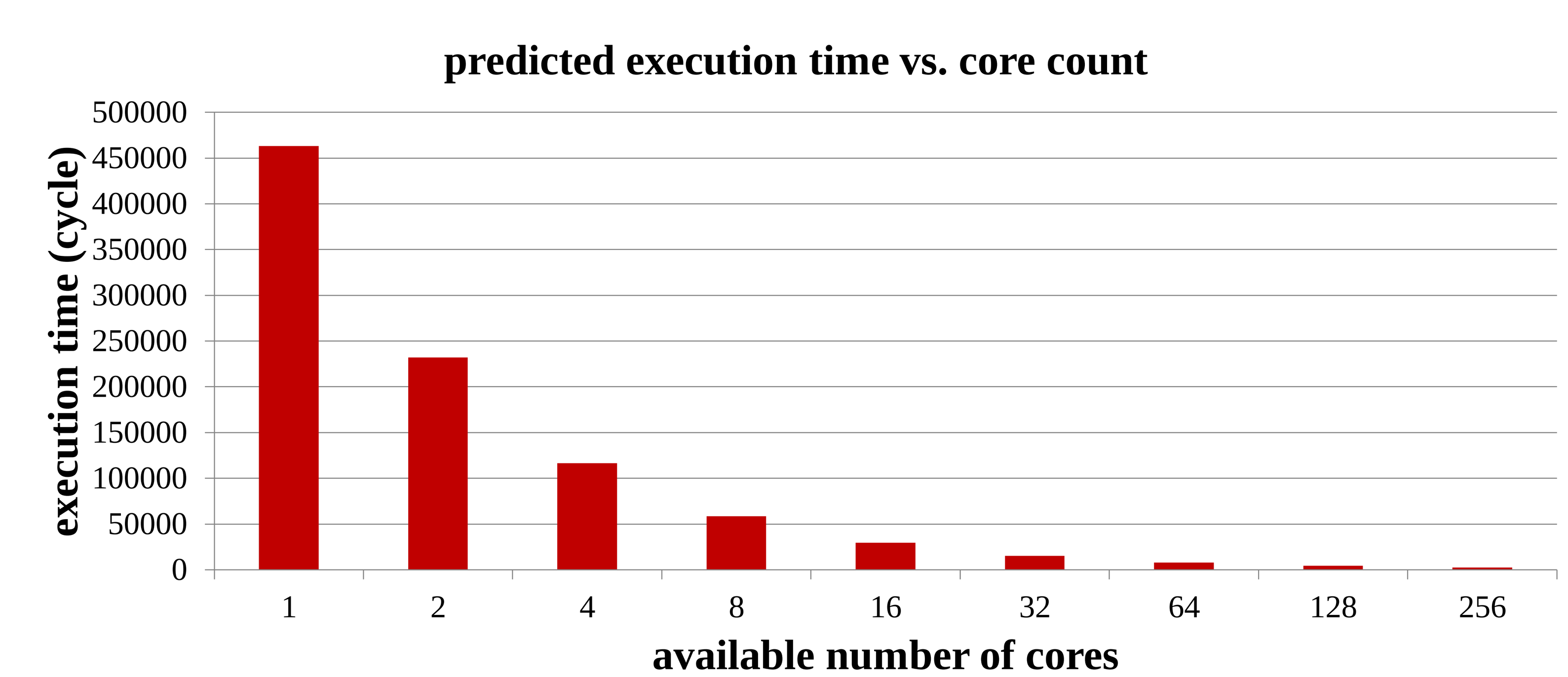
Implementation



- Make use of the Low-Level Virtual Machine (LLVM) project
- SISA libraries implemented as LLVM passes
  - Take advantage of LLVM's wide range of front-end compilers and back-end code emitters
  - Existing platforms for rich features including alias analysis, dependence analysis, profiling, etc.

Runtime Predictor

- To be able to predict a thread's runtime is an important task of the runtime manager



- PARSEC 2.0 benchmark blackscholes-simsmall input set
- Assume all the loops are DOALL loops
- Plotting runtime cycle count vs. available cores