Multi-Objective Intrinsic Evolution of Embedded Systems (MOVES)

Paul Kaufmann, Marco Platzner

Computer Engineering Group

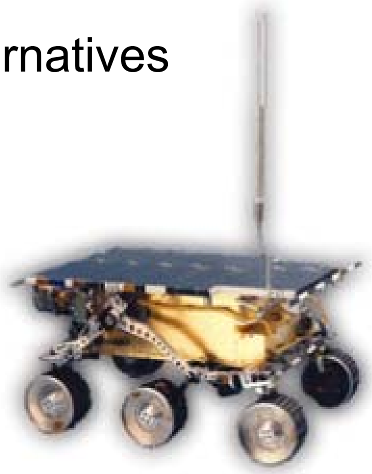
University of Paderborn

{paulk, platzner}@uni-paderborn.de

Motivation / Vision

- investigate intrinsic evolution as a mechanism to achieve self-adaptation and –optimization for autonomous embedded systems

- an embedded system ...
  - adapts to slow changes by simulated evolution
    - typically, change of environment
  - adapts to radical changes by switching to pre-evolved alternatives
    - typically, change in computational resources
  - requires intrinsic evolution for autonomous operation



Outline

- ✓ motivation/vision
- Δ to last status meeting
- coarse-grained CGP model
- EvoCaches: adaptation of cache mappings

Δ to Last Status Meeting (1)

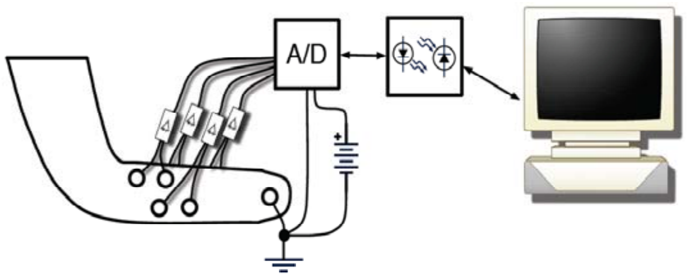
- working areas
  1. models and algorithms
  2. system architectures
  3. case studies, evaluation

- last status meeting
  - evolutionary algorithms: tackling scalability, comparing GA with MOEAs
  - reconfigurable SoC: hardware accelerator for k-NN thinning
  - application examples: prosthetic hand controllers

- new work done
  - comparing GA with MOEAs for hardware evolution
  - <u>coarse-granular cartesian genetic programming (CGP) model</u>
  - experiments with the functional unit row architecture for classification tasks (cooperation with University of Oslo)

Δ to Last Status Meeting (2)

- new work done (cont'd)
  - reconfigurable SoC
    - hardware accelerator for k-NN thinning [Schumacher et al., FCCM '09] [Schumacher et al., FPL '09]
  - prosthetic hand controllers [Boschmann et al., TAR '09] [Boschmann et al., TIPS '09]



→ spin-off project "Adaptive Prothetik" funded by BMWi
(University of Paderborn, Orthopädietechnik Winkler, iXtronics, DLR)

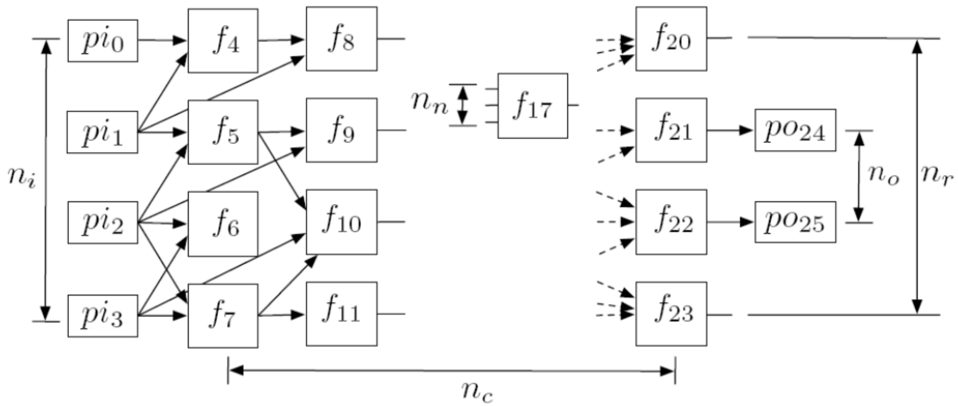
  - EvoCaches: application-specific cache mappings [Kaufmann et al., AHS '09]

Outline

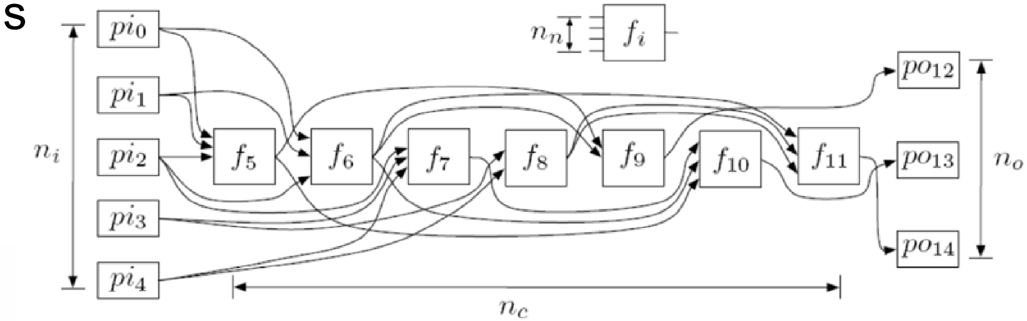
- ✓ motivation/vision
- ✓ Δ to last status meeting
- coarse-grained CGP model
- EvoCaches: adaptation of cache mappings

CGP Models

- cartesian genetic programming (CGP) [Miller & Thomson, EuroGP '00]
  - array of combinational blocks connected by feed-forward wires
  - chromosome defines configuration of the array

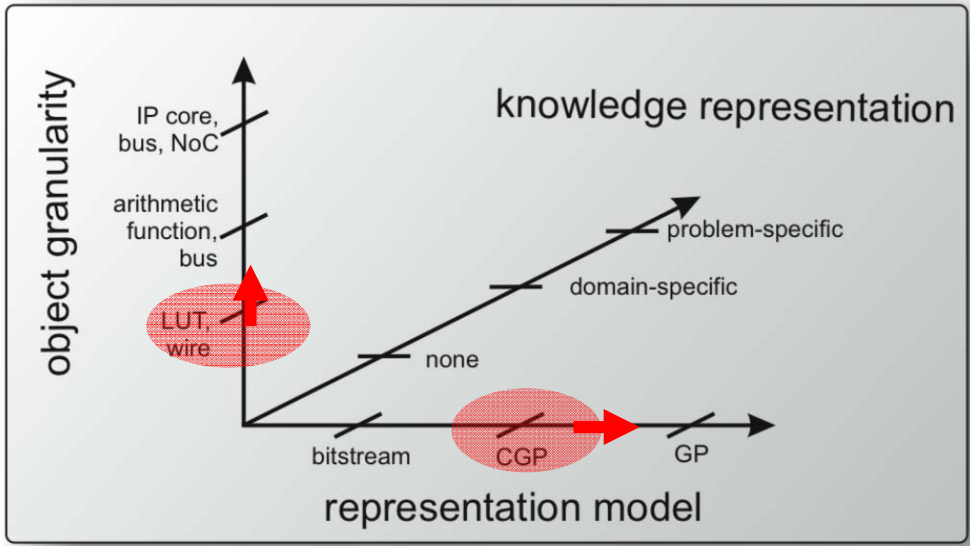


- ECGP model [Walker & Miller, EuroGP '04]
  - single row of functional blocks
  - no restriction on wire lengths
  - 1+4 evolutionary strategy



Scalability and CGP

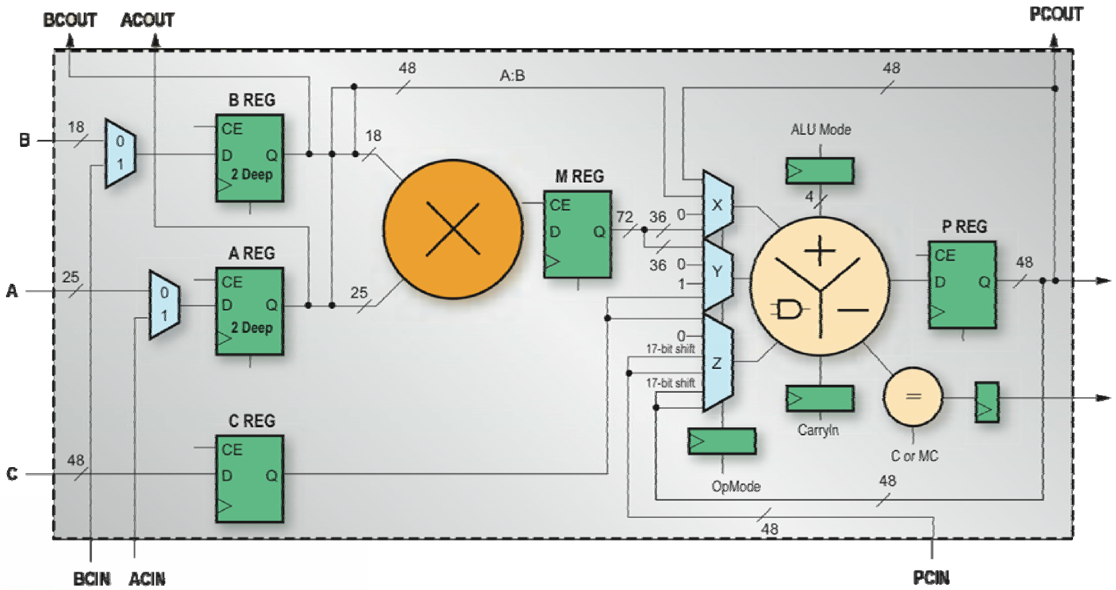
- scalability can be tackled along three dimensions
  - object granularity
  - knowledge representation
  - representation model



- coarse-grained CGP model
  - moving from LUTs and wires to functional units and buses
    → supported by modern FPGA technologies
  - moving from structural to behavioral models
    → challenge is to maintain an efficient mapping to real hardware

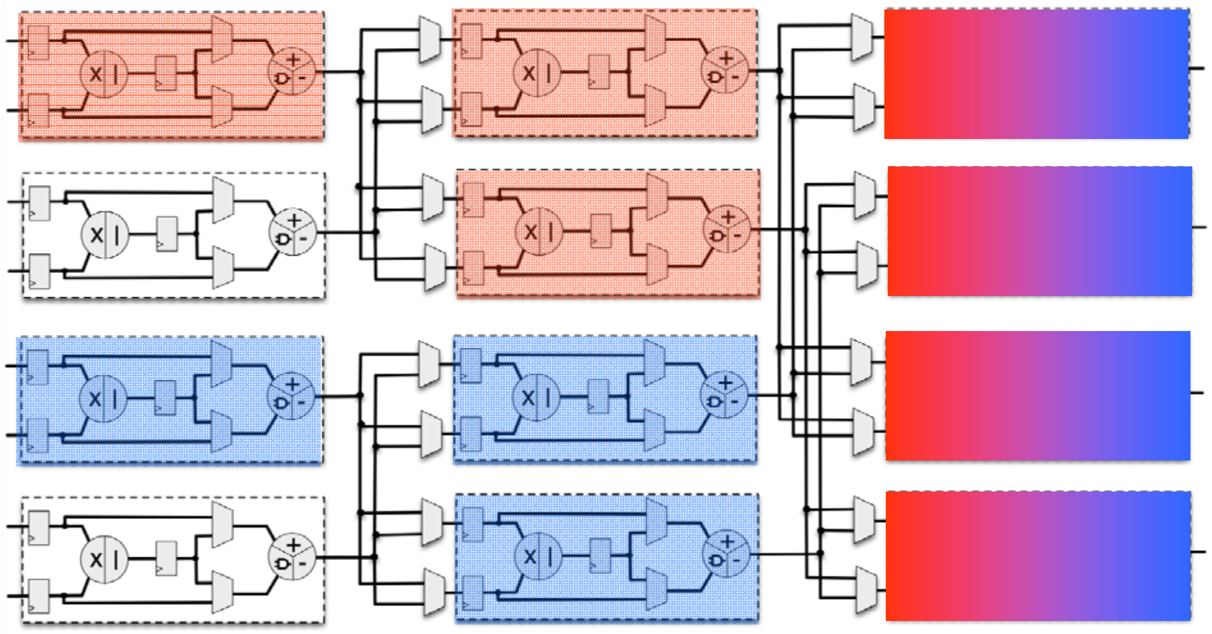
Coarse-grained CGP: Block

- implementation on Xilinx FPGA technology using DSP48 blocks
  - multiply (18x25), add, sub, compare (48x48)
  - bit-wise OR, AND, NOT, NOR, NAND, XOR, and XNOR
  - wider data types by cascading multiple DSP48E blocks
- configuration through user logic registers
  - also denoted as “Virtual Reconfiguration“ (VRC)
  - fast reconfiguration (one clock cycle)



Coarse-grained CGP: Interconnect

- hyper cube inspired interconnect
  - general feed-forward interconnect too expensive in terms of hardware

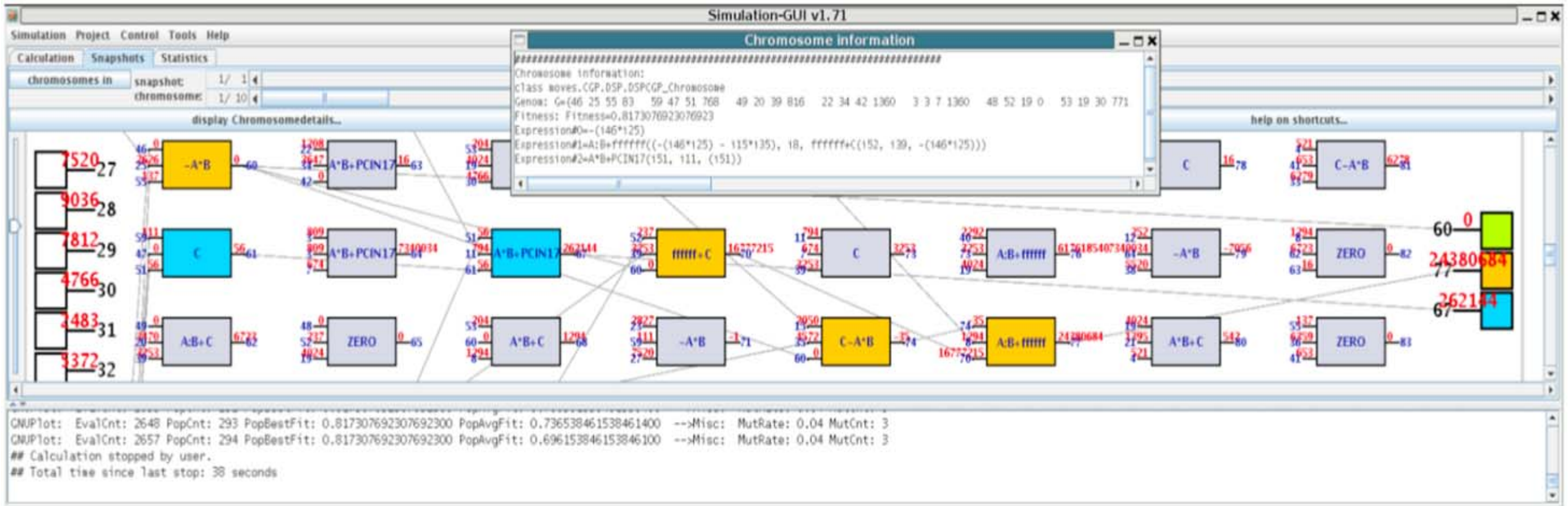


- trade-off
  - include placement and routing in the CGP chromosome, or
  - map an evolved (restricted) DAG to hardware

Coarse-grained CGP: Status

- proof-of-concept implementation on Xilinx Virtex 5
  - Linux@PowerPC and CGP core
    - access DSP48E via CPU bus
    - integration into Linux device driver hierarchy

- integration of the coarse-grained CGP model into the MOVES toolbox
  - DSP48E functional model

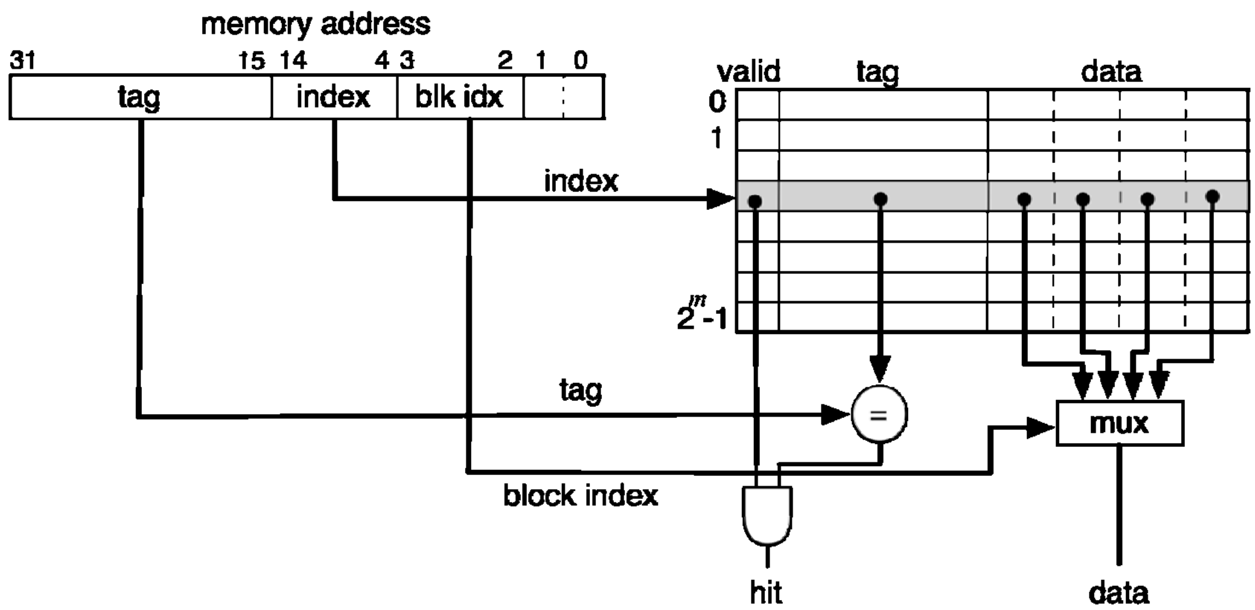


Outline

- ✓ motivation/vision
- ✓ Δ to last status meeting
- ✓ coarse-grained CGP model
- EvoCaches: adaptation of cache mappings

EvoCache (1)

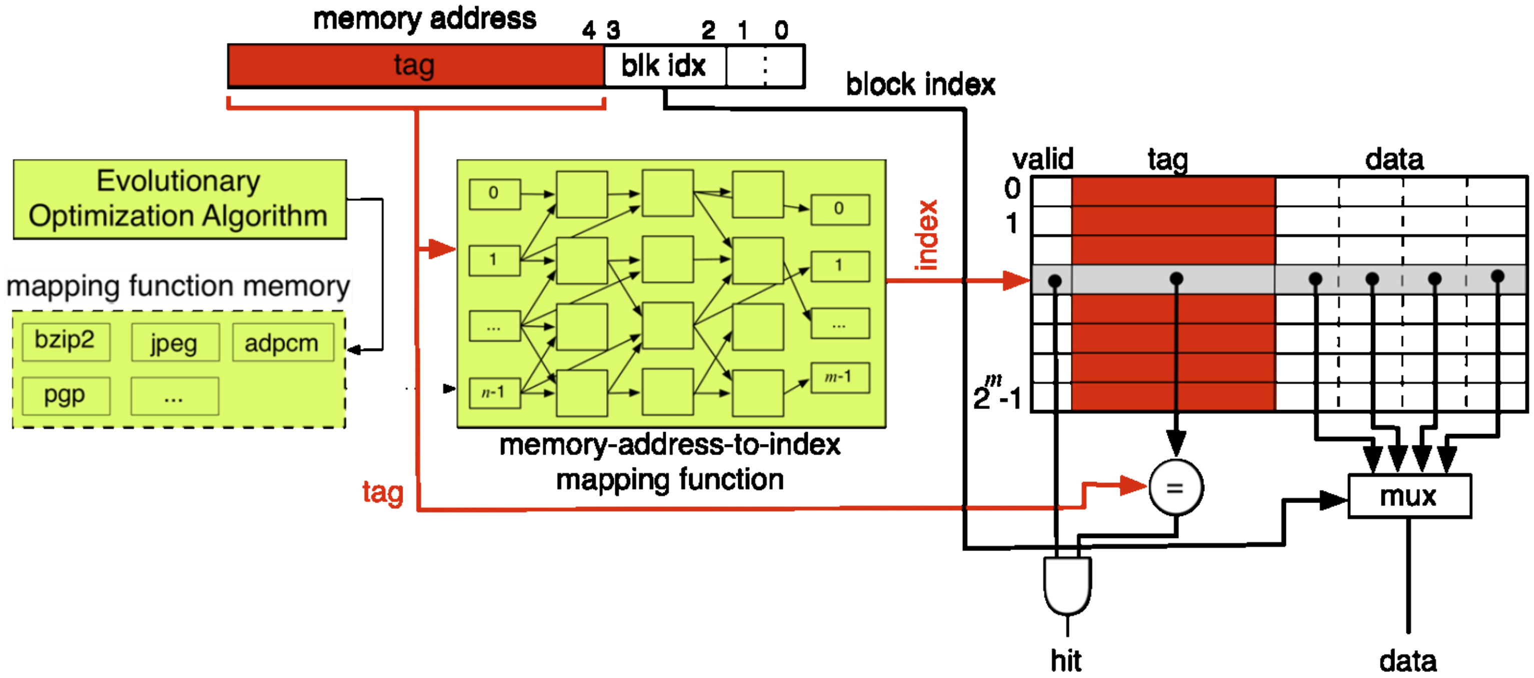
- classic direct-mapped cache fetches data entry by
  - addressing the cache line (index)
  - detecting collisions (tag)
  - addressing inside the cache line (block index)



- set-associative caches reduce collisions

EvoCache (2)

- idea: application-specific memory-address → index mapping function



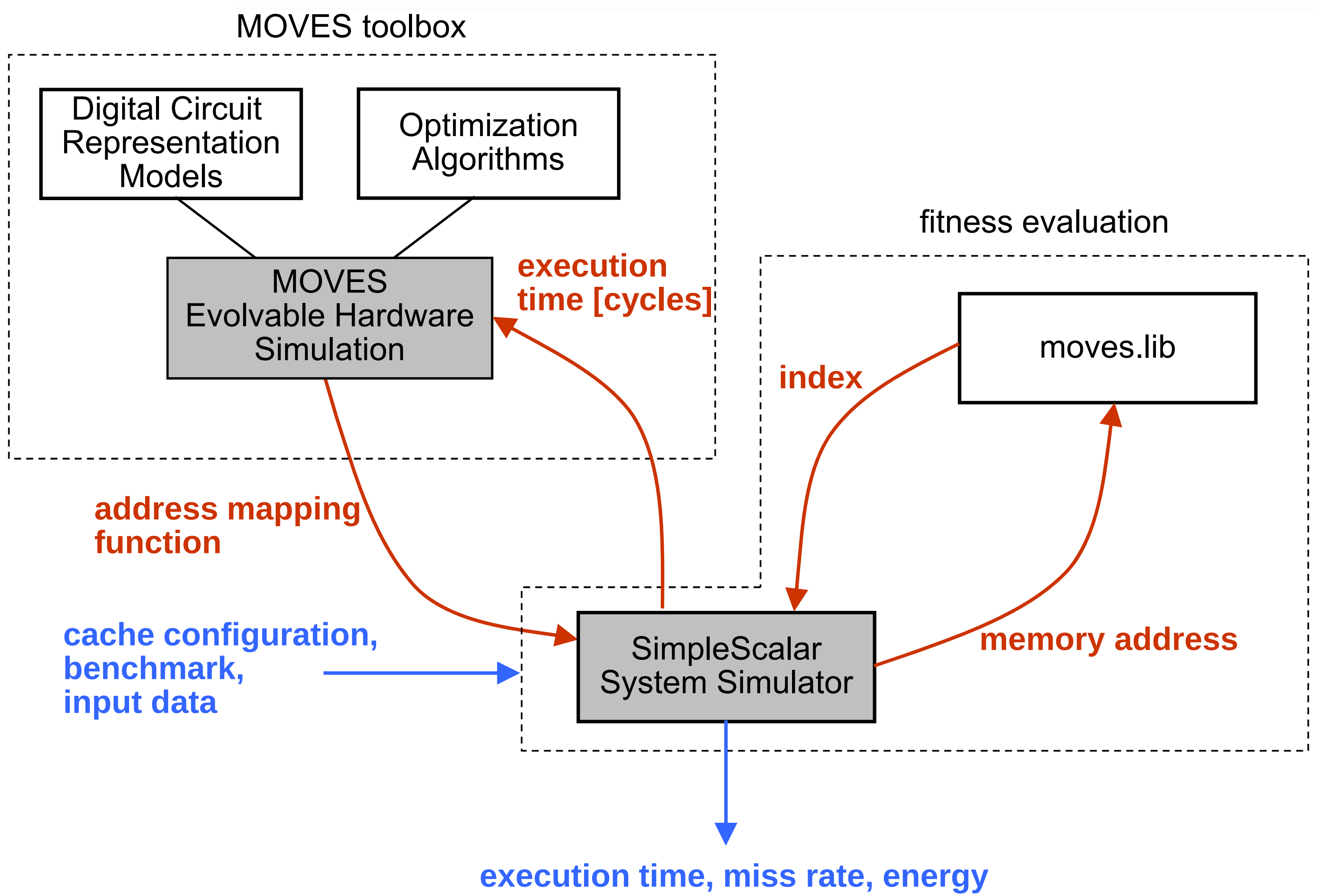
- reconfigurable circuit maps tag to cache line
- optimization algorithm determines the mapping

Related Work on Adaptive Caches

- optimize cache configuration: size, associativity, replacement strategy
  - 2-3 static configurations to improve IPC [Ranganathan et al., ISCA '00]
  - selective cache ways to reduce power consumption [Albonesi, MICRO '99]
  - self-tuning by simple search heuristic to reduce energy for accessing memory [Zhang et al., ACM TECS '04]

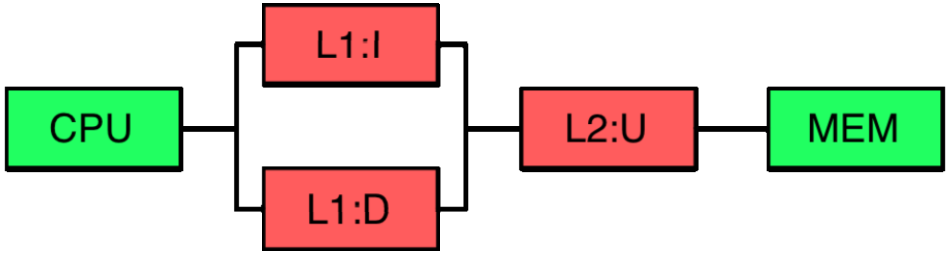
- optimize cache address to index mapping
  - bit-juggling in index and block index fields to reduce miss-rate [Stanca et al., EuroPar '00]
  - XOR-ing two memory address bits to reduce miss-rate [Vandierendonck et al., DATE '06]
  - EvoCache approach
    - more complex circuit model
    - optimization by evolutionary techniques
    - optimize for execution time, determine miss-rate and energy

EvoCache Tool Setup



Experiment Configuration

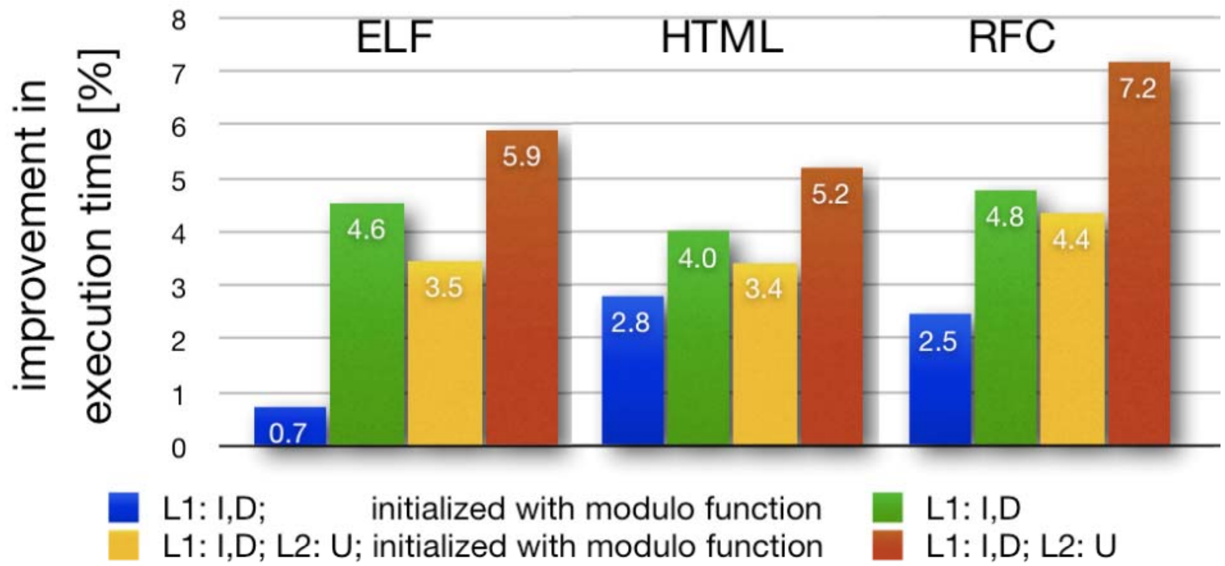
- hardware representation model
  - CGP: 4-LUTs, one row, 32 columns, unconstrained wire length
  - 1+4 evolutionary strategy
- cache configuration
  - L1:I, L1:D: 2-way, 1 cycle hit time, 64 lines, block size 16 bytes, LRU
  - L2:U: 4-way, 6 cycles hit time, 128 lines, block size 32 bytes, LRU
  - we optimize either {L1:I, L1:D} or {L1:I, L1:D, L2:U}



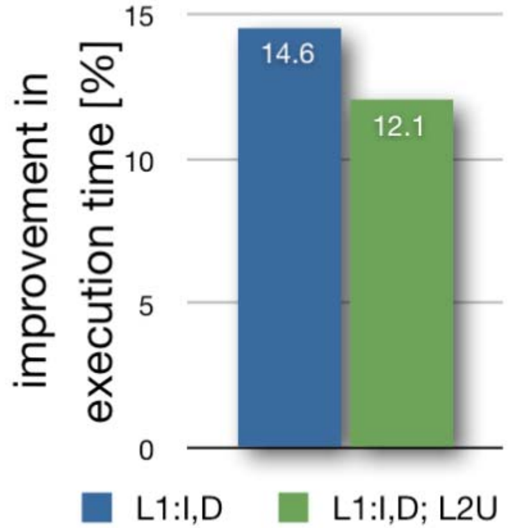
- benchmarks
  - bzip2 – text file compression
  - JPEG – image compression

Generalization Performance

bzip2



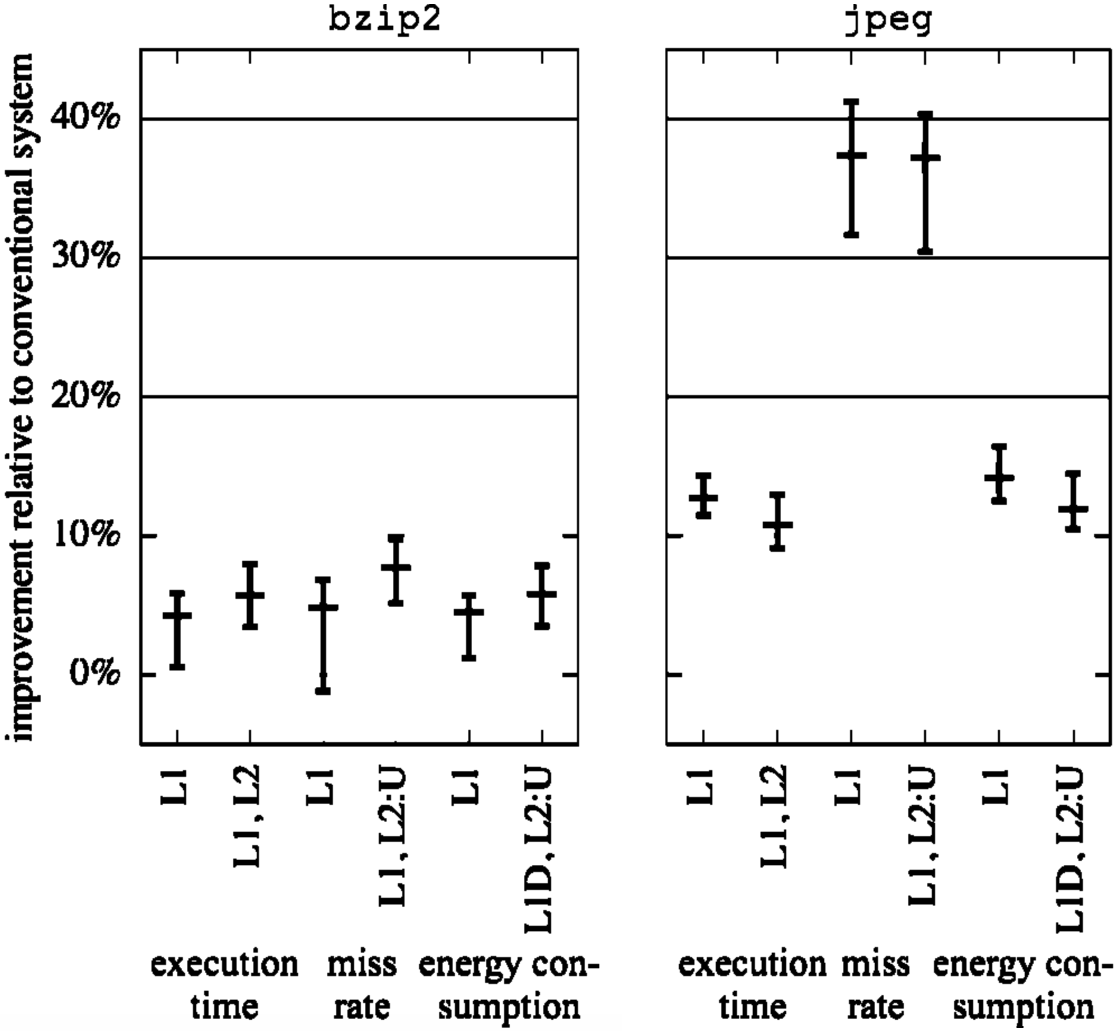
JPEG



– best bzip2 training circuit verified on three data sets
  - 10 Linux executable files (ELF)
  - 10 HTML web sites (HTML)
  - 10 RFC plain text files (RFC)

– best JPEG training circuit verified on data set of ten pictures

Miss Rate and Energy Consumption



EvoCache Issues, Further Work

- cache implementation
  - hit time increase: best mapping functions had depth of 3-6 LUTs
  - area increase: best mapping functions had 14-19 LUTs, larger tag
- system integration
  - application binaries carry EvoCache configuration (backward compatible)
  - what to do at context switches?
- experiments with high-performance CPUs and workloads
  - simulation times are excessive for caches of modern high-performance CPUs and realistically large workloads
  - need to use execution traces, statistical sampling techniques
- online optimization
  - nodes gather traces and generate mapping functions using spare cycles
  - evolutionary optimization in a distributed manner (island-GA)

Outline

- ✓ motivation/vision
- ✓ Δ to last status meeting
- ✓ coarse-grained CGP model
- ✓ EvoCaches: adaptation of cache mappings

Thank you for your attention!