Towards Optimal Custom Instruction Processors

Wayne Luk
Kubilay Atasu, Rob Dimond and Oskar Mencer

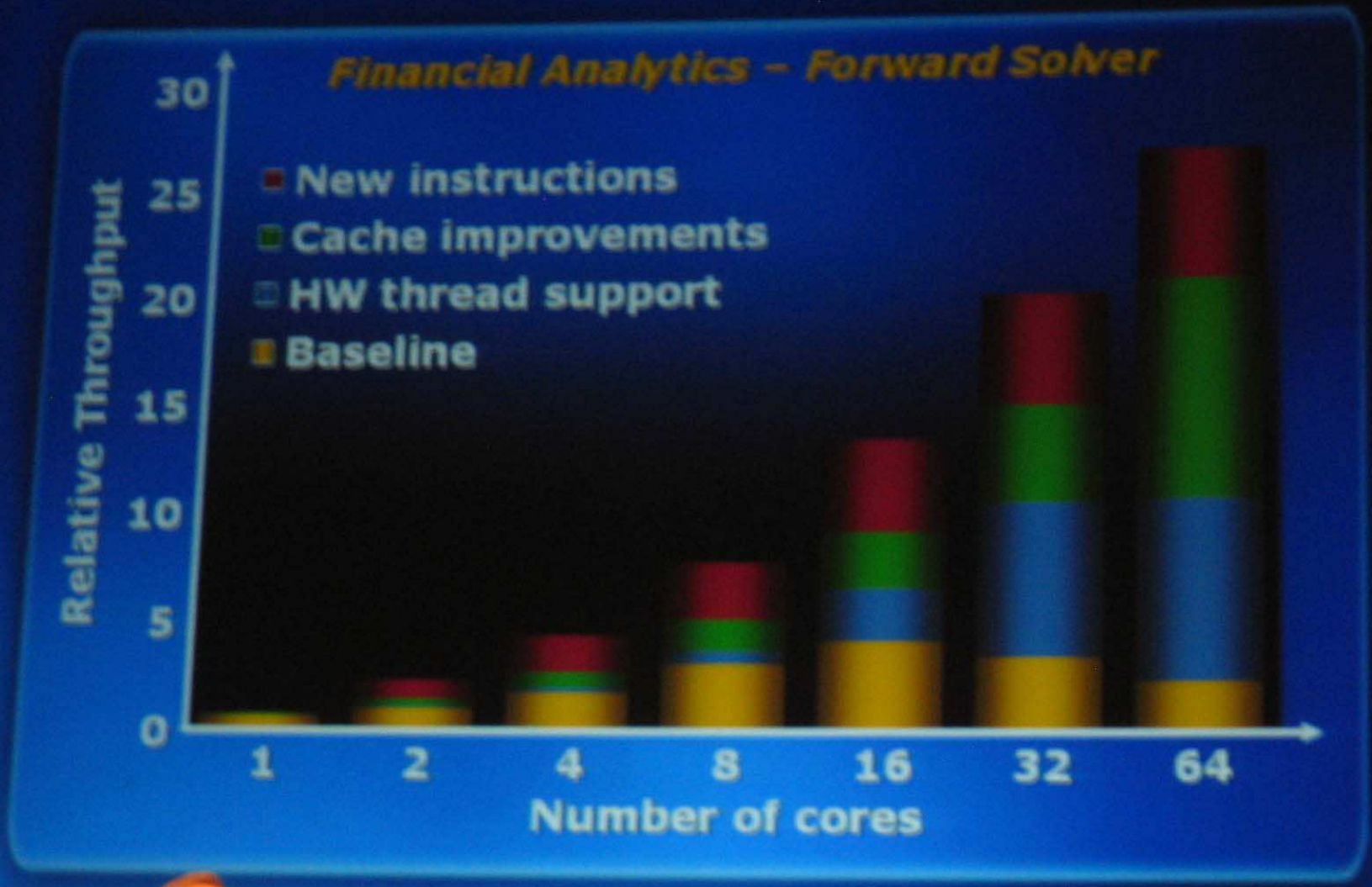
Department of Computing
Imperial College London

HOT CHIPS 18

Overview

1. background: extensible processors
2. design flow: C to custom processor silicon
3. instruction selection: bandwidth/area constraints
4. application-specific processor synthesis
5. results: 3x area delay product reduction
6. current and future work + summary

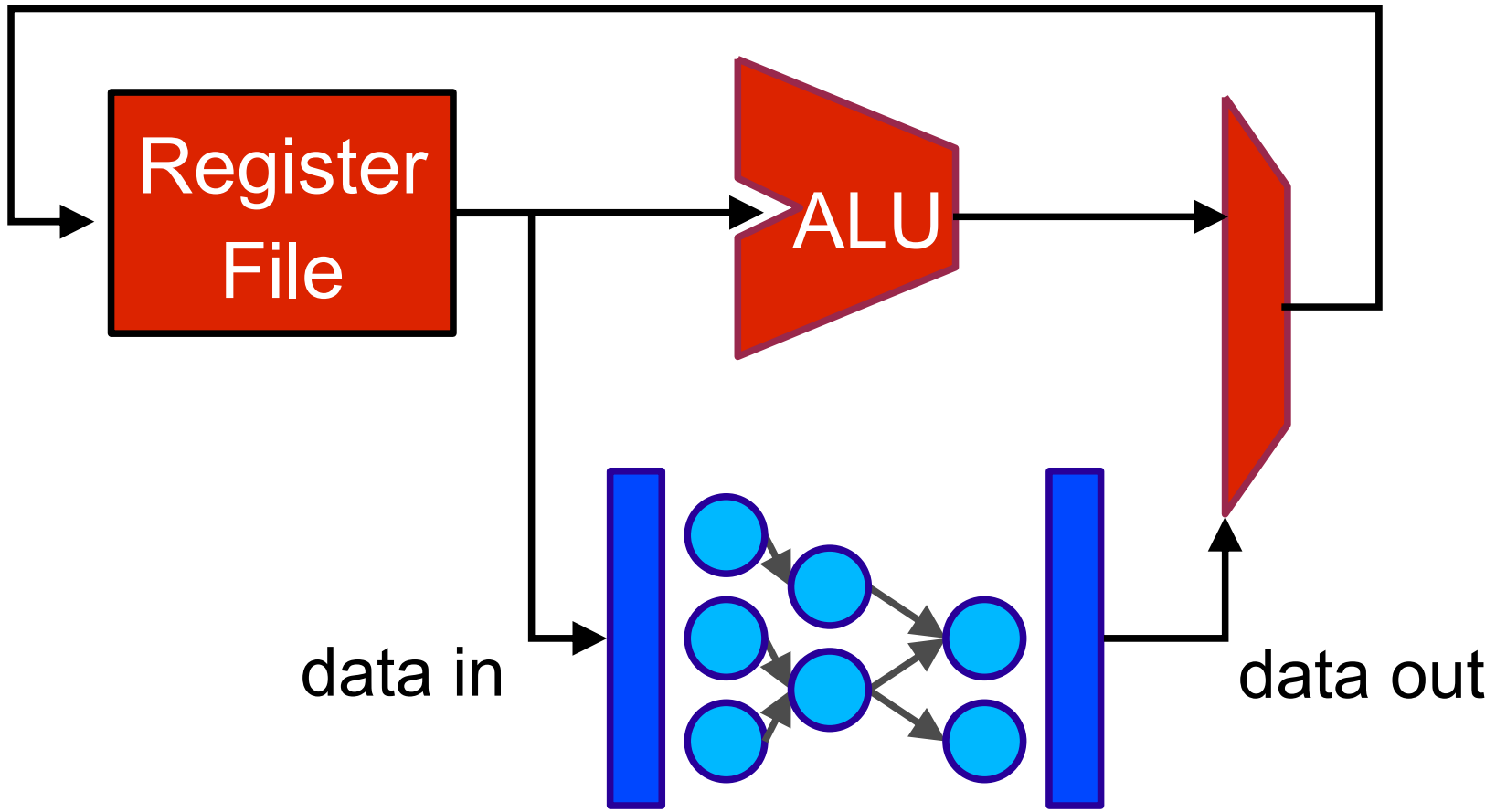
Architecture-Algorithm Co-Design



intel

1. Instruction-set extensible processors

- base processor + custom logic
  - partition data-flow graphs into custom instructions



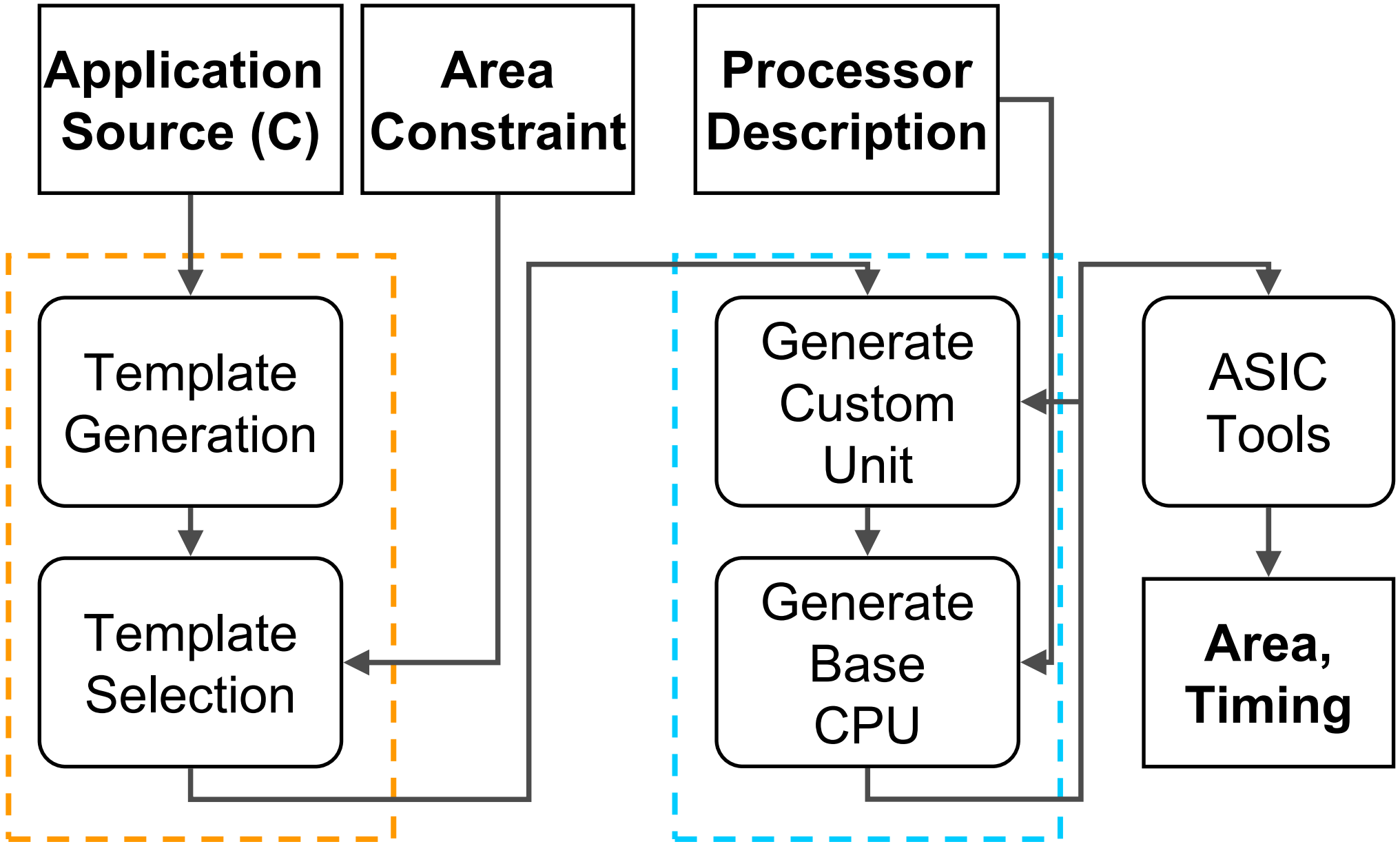
Previous work

- many techniques, e.g.
  - Atasu et al. (DAC 03, CODES 05)
  - Biswas et al. (DATE 05), Cong et al. (FPGA 04)
  - Clark et al. (MICRO 03, HOT CHIPS 04)
  - Goodwin and Petkov (CASES 03)
  - Sun et al. (ICCAD 03), Yu et al. (CASES 04)
- current challenges
  - optimality and robustness of heuristics
  - complete tool chain: application to silicon
  - research infrastructure for custom processor design

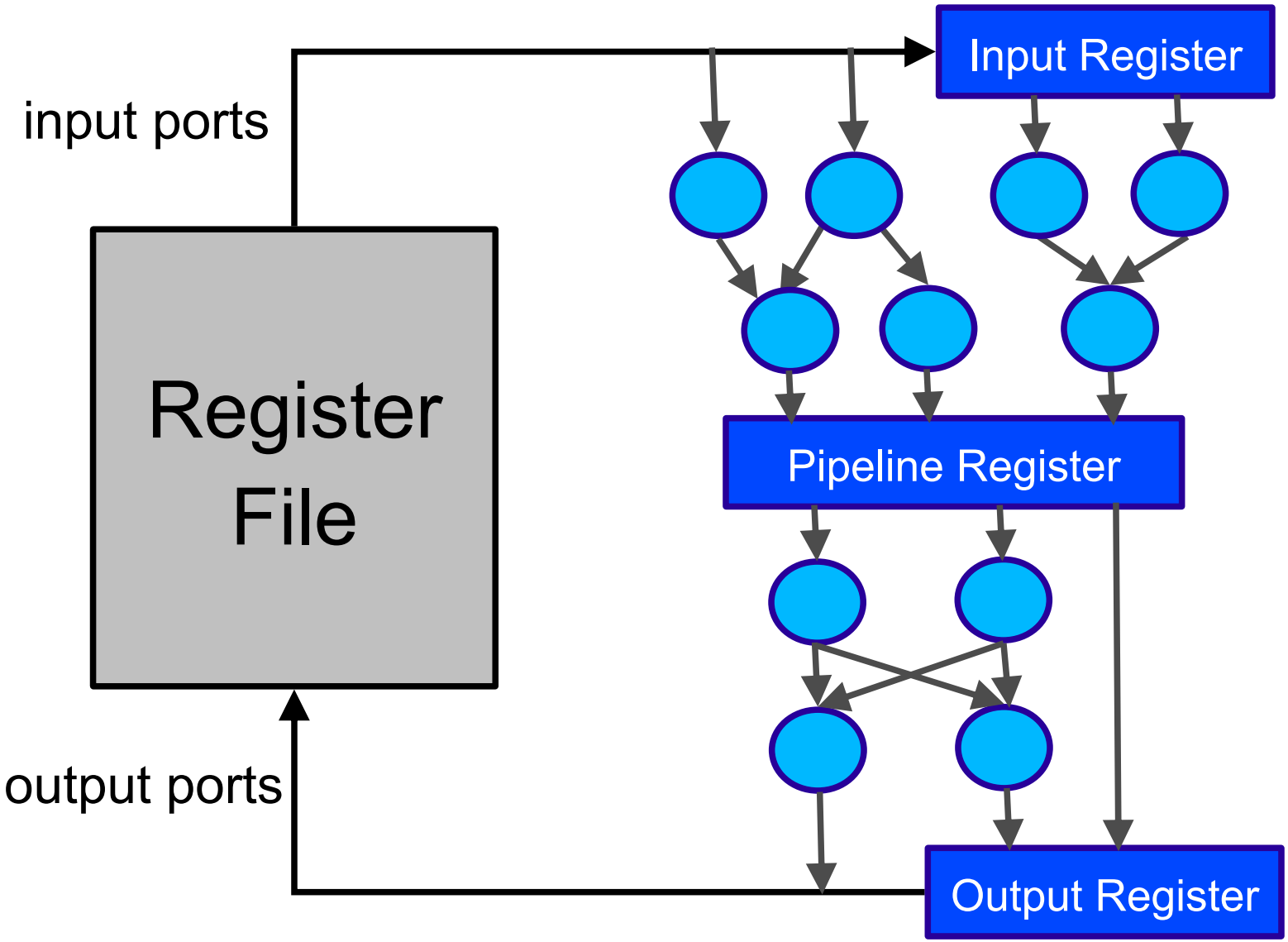
2. Custom processor research at Imperial

- focus on effective optimization techniques
  - e.g. Integer Linear Programming (ILP)
- complete tool-chain
  - high-level descriptions to custom processor silicon
- open infrastructure for research in
  - custom processor synthesis
  - automatic customization techniques
- current tools
  - optimizing compiler (Trimaran) for custom CPUs
  - custom processor synthesis tool

Application to custom processor flow



Custom instruction model



3. Optimal instruction identification

- minimize schedule length of program data flow graphs (DFGs)
- subject to constraints
  - convexity: ensure feasible schedules
  - fixed processor critical path: pipeline for multi-cycle instructions
  - fixed data bandwidth: limited by register file ports
- steps: based on Integer Linear Pogramming (ILP)

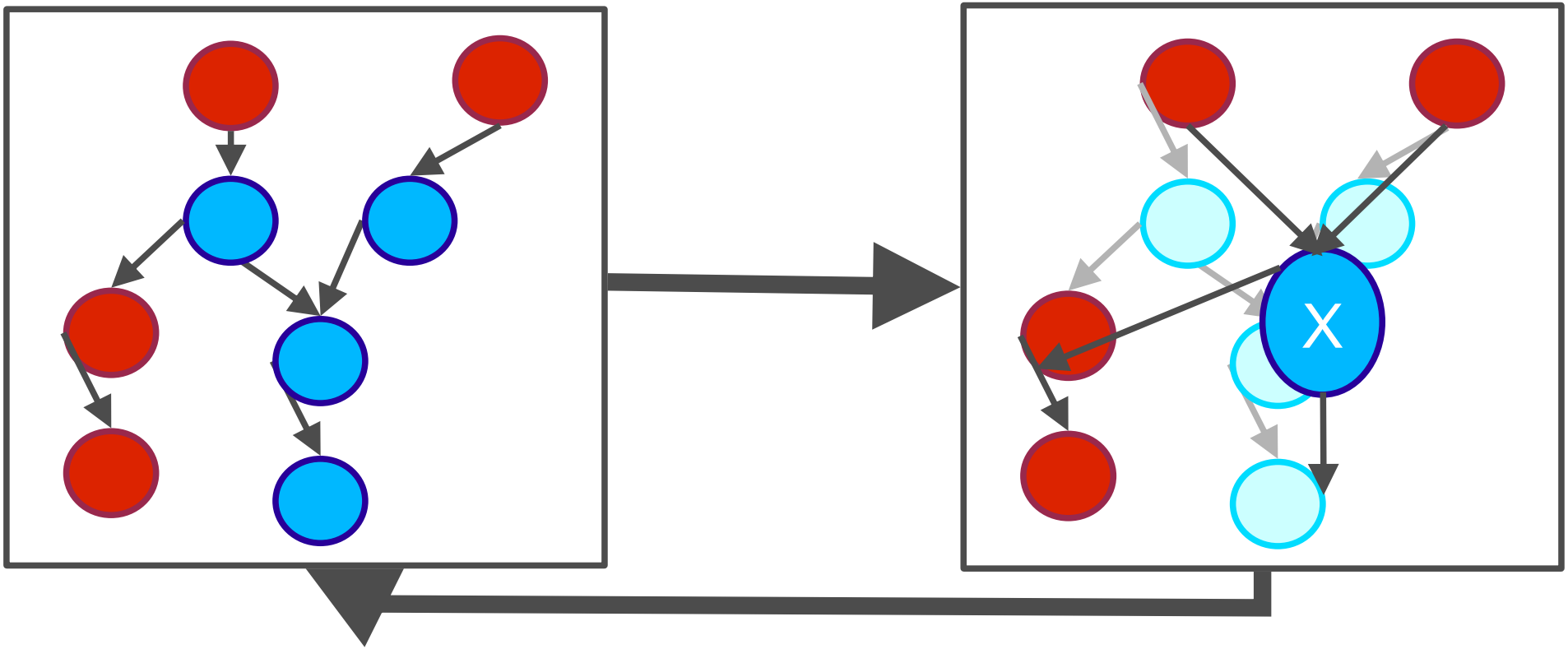
  a. template generation

  b. template selection

a. Template generation

1. Solve ILP for DFG to generate a template

2. Collapse template to a single DFG node

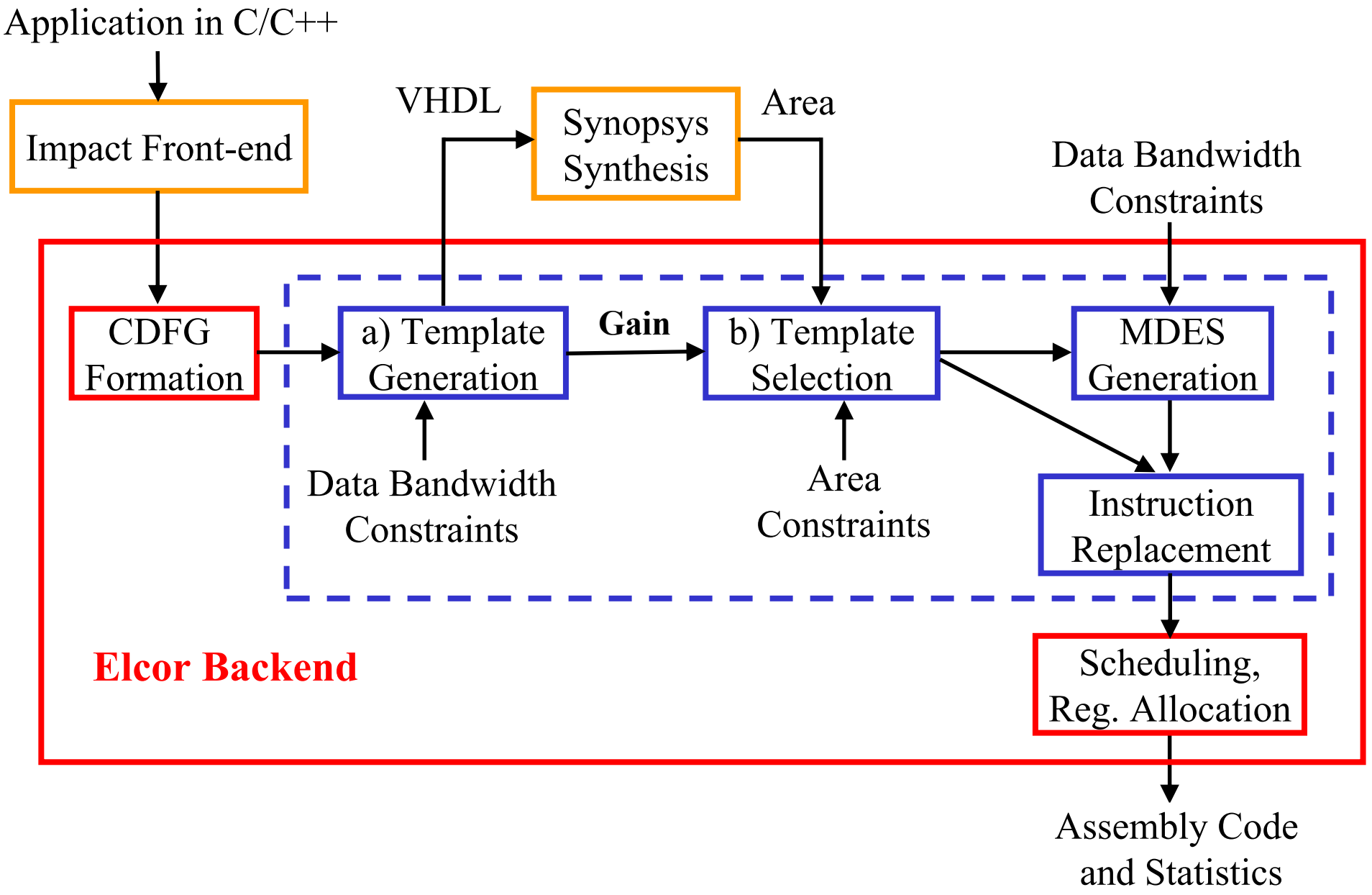


3. Repeat while (objective > 0)

b. Template selection

- determine isomorphism classes
  - find templates that can be implemented using the same instruction
  - calculate speed-up potential of each class
- solve Knapsack problem using ILP
  - maximize speedup within area constraint

Optimizing compilation flow



4. Application-specific processor synthesis

- design space exploration framework
  - Processor Component Library
  - specialized structural description
- prototype: MIPS integer instruction set
  - custom instructions
  - flexible micro-architecture
- evaluate using actual implementation
  - timing and area

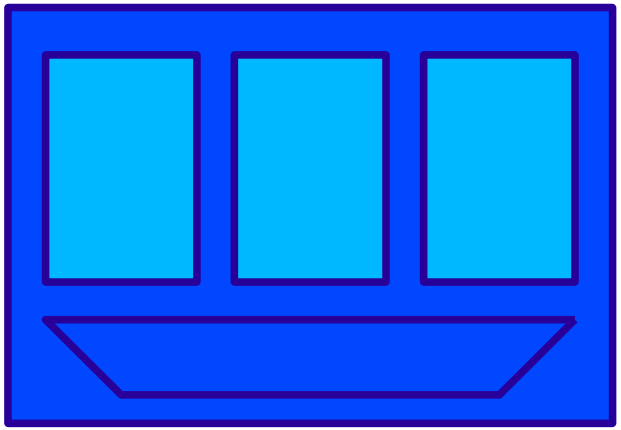
Processor synthesis flow

from compiler

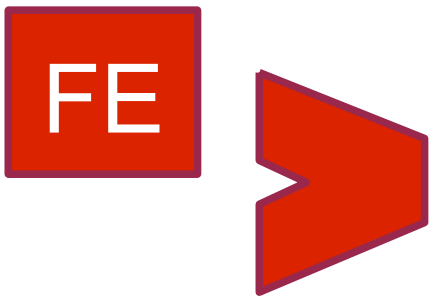
Custom
Data paths

- merging
- add state registers
- processor interface

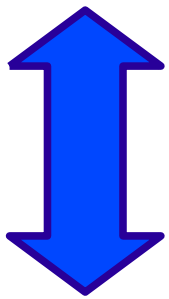
Custom
Processor



Processor
Component Library



- pipeline description
- parameters



interface

- data in/out
- stall control

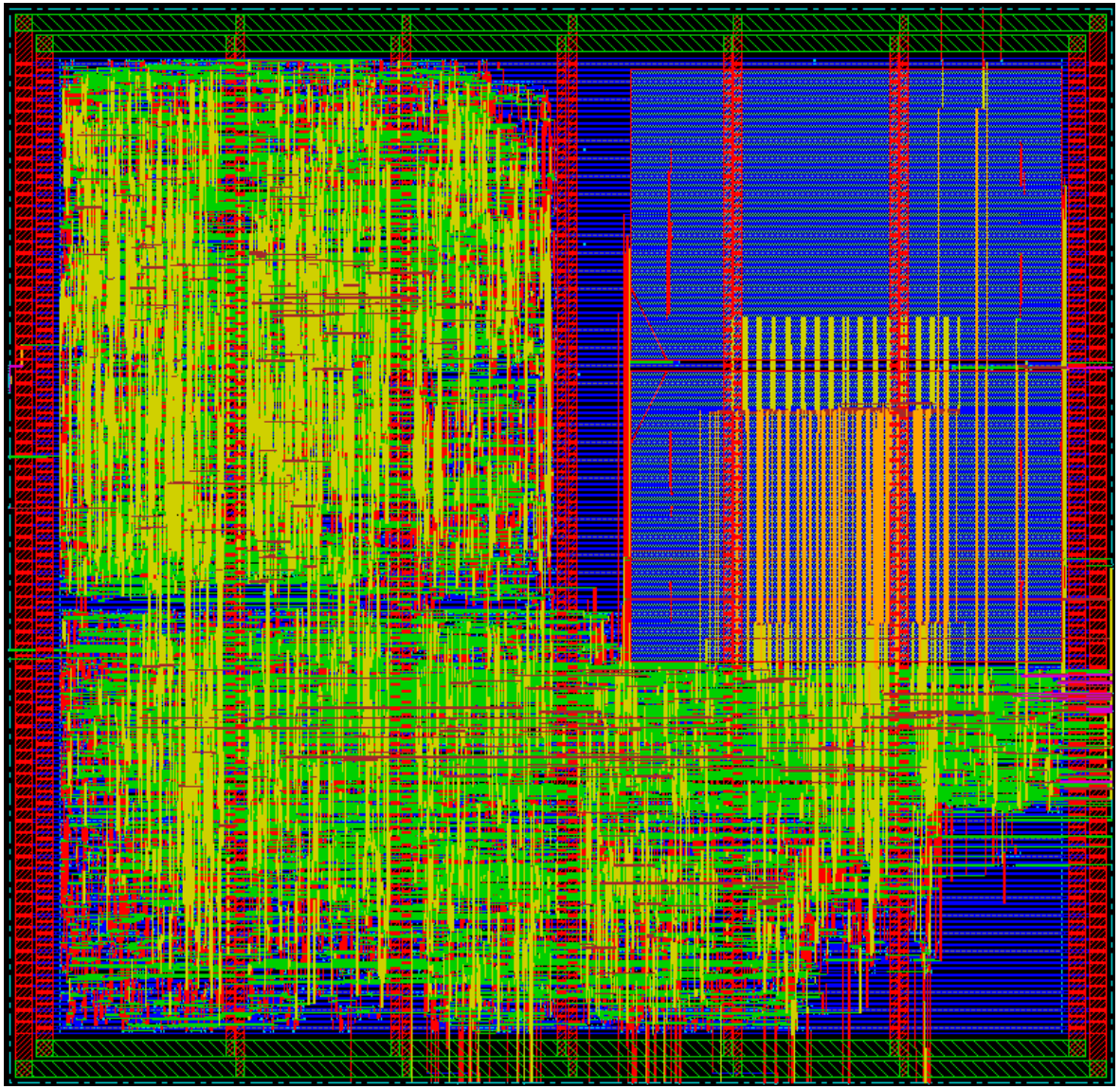
Implementation

- based on Python scripts
  - structural meta-language for processors
  - combine RTL (Verilog/VHDL) IP blocks
  - module generators for custom units
- generate 100s of designs automatically
  - ASIC processor cores
  - complete system on FPGA: CPU + memory + I/O

5. Results

- cryptography benchmarks: C source
  - AES decrypt, AES encrypt, DES, MD5, SHA
- 4/5 stage pipelined MIPS base processor
  - 0.225mm^2 area, 200 MHz clock speed
  - single issue processor
  - register file with 2 input ports, 1 output port
- processors synthesized to 130nm library
  - Synopsys DC and Cadence SoC Encounter
  - also synthesize to Xilinx FPGA for testing

AES Decryption Processor

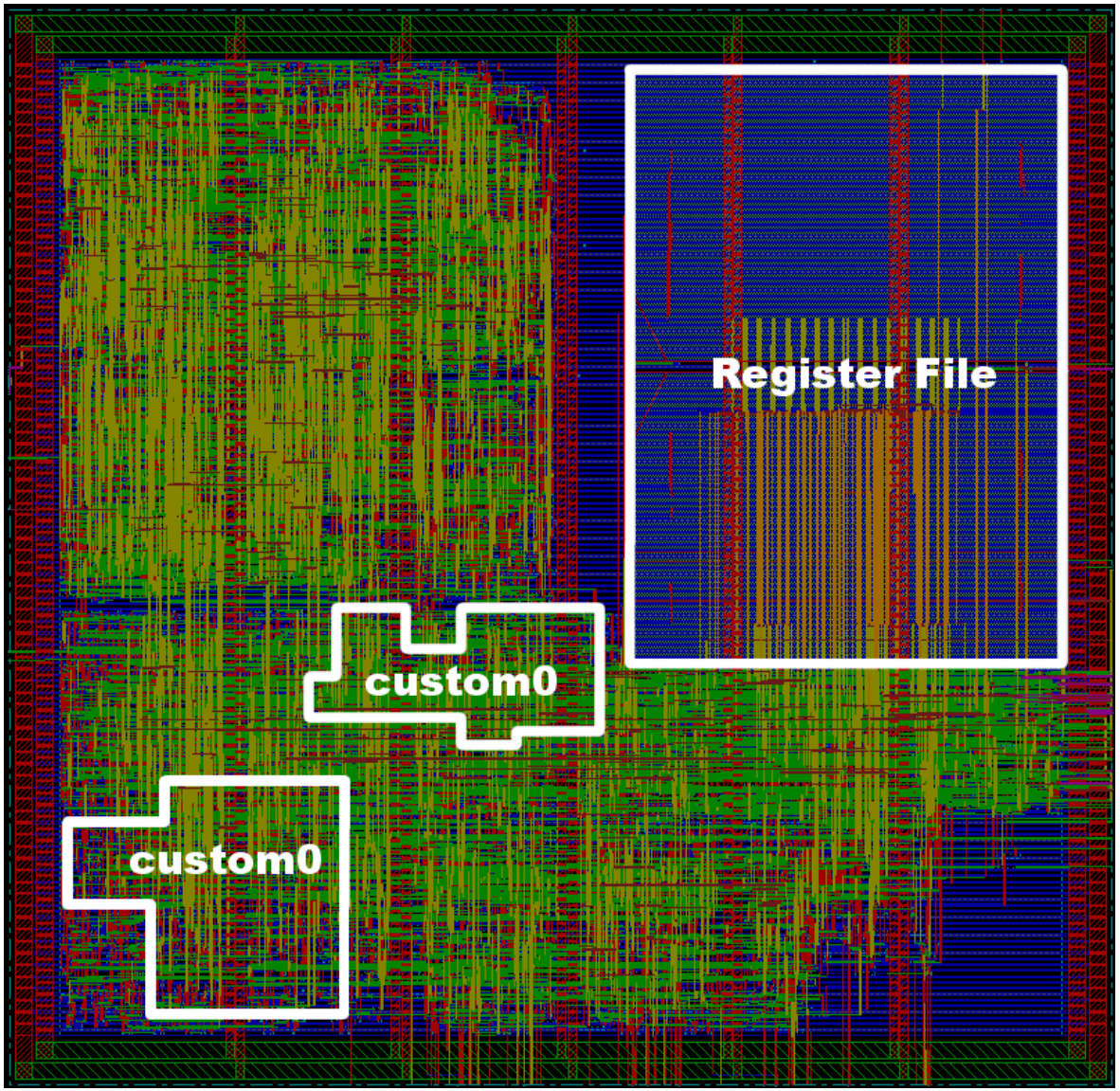


130nm CMOS
200MHz
0.307mm^2

35% area cost (mostly one instruction)

76% cycle reduction

AES Decryption Processor

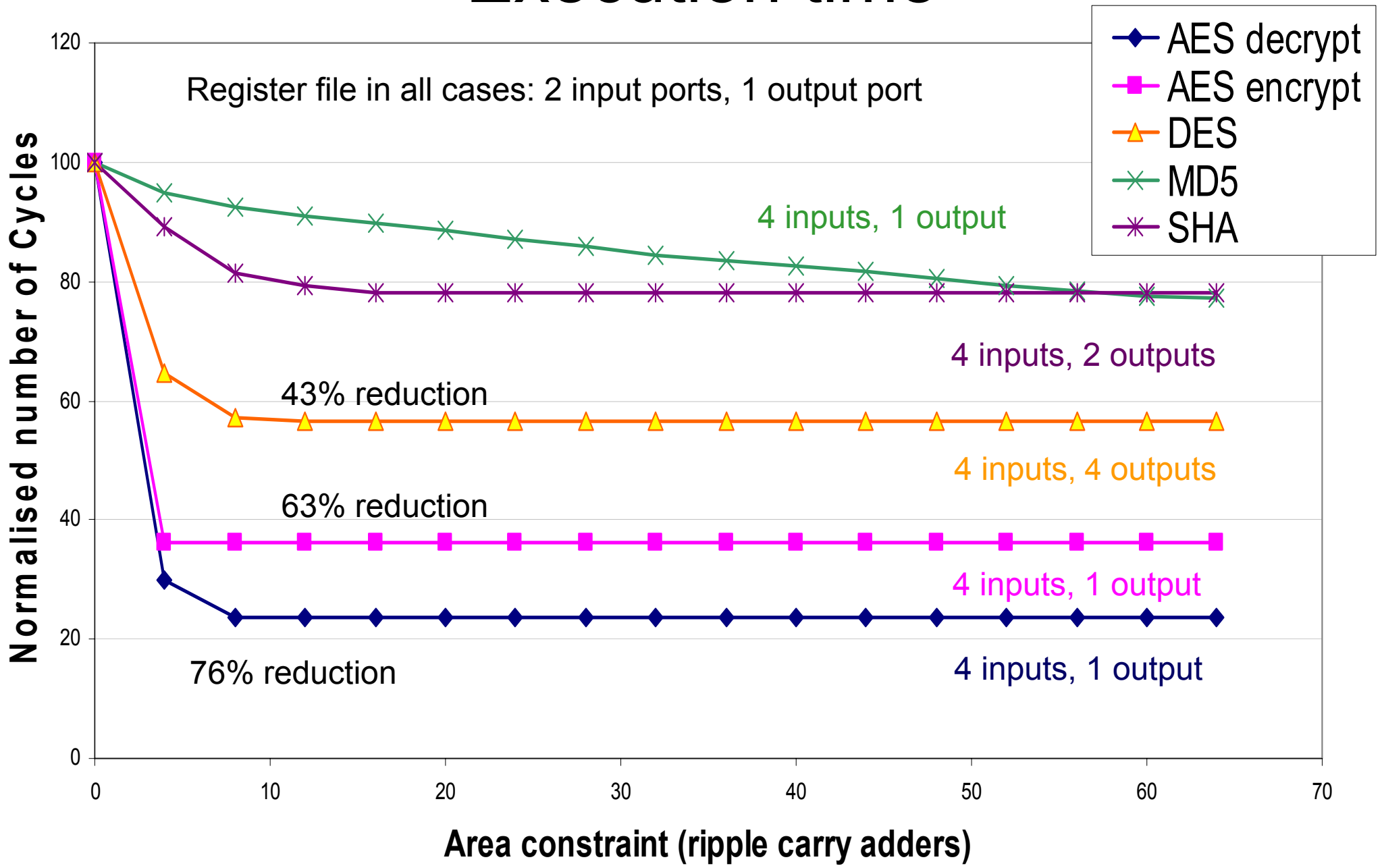


130nm CMOS
200MHz
0.307mm^2

35% area cost
(mostly one
instruction)

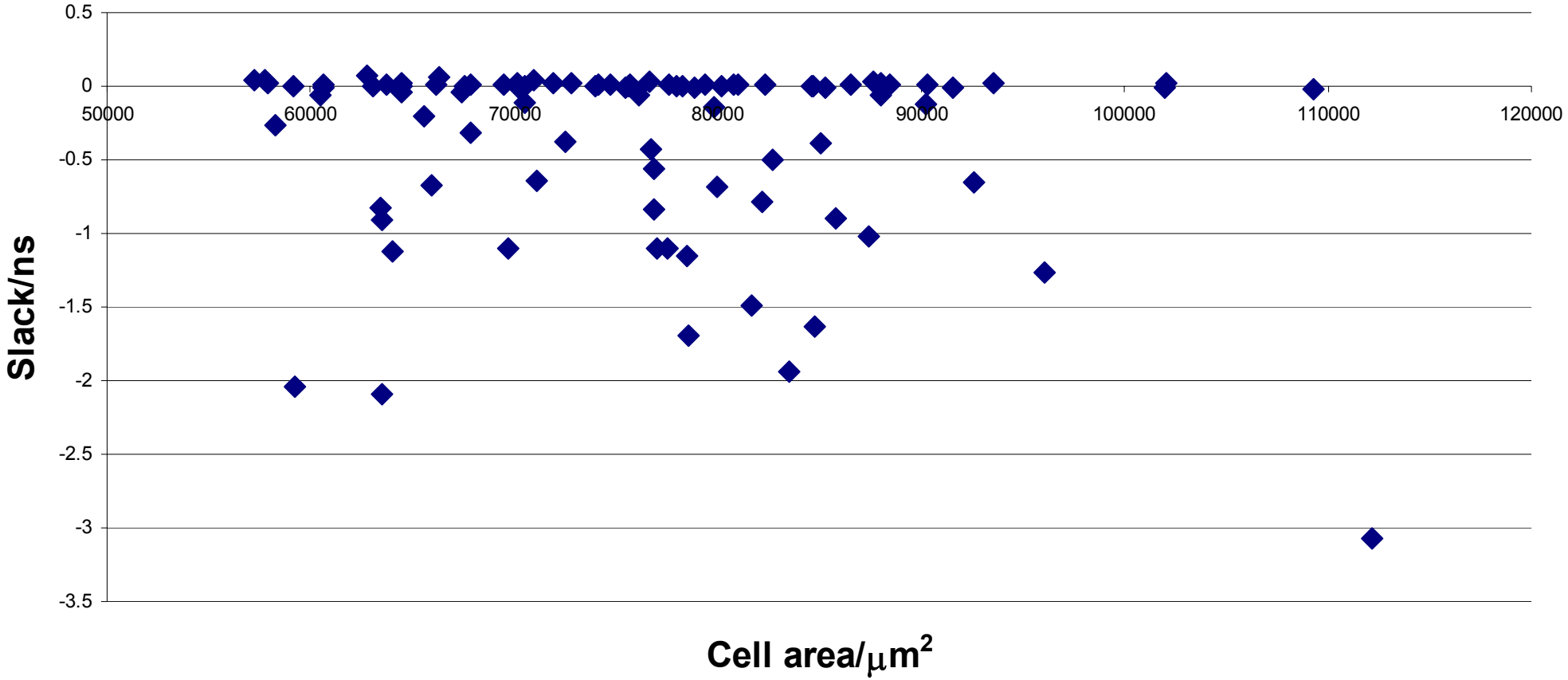
76% cycle
reduction

Execution time

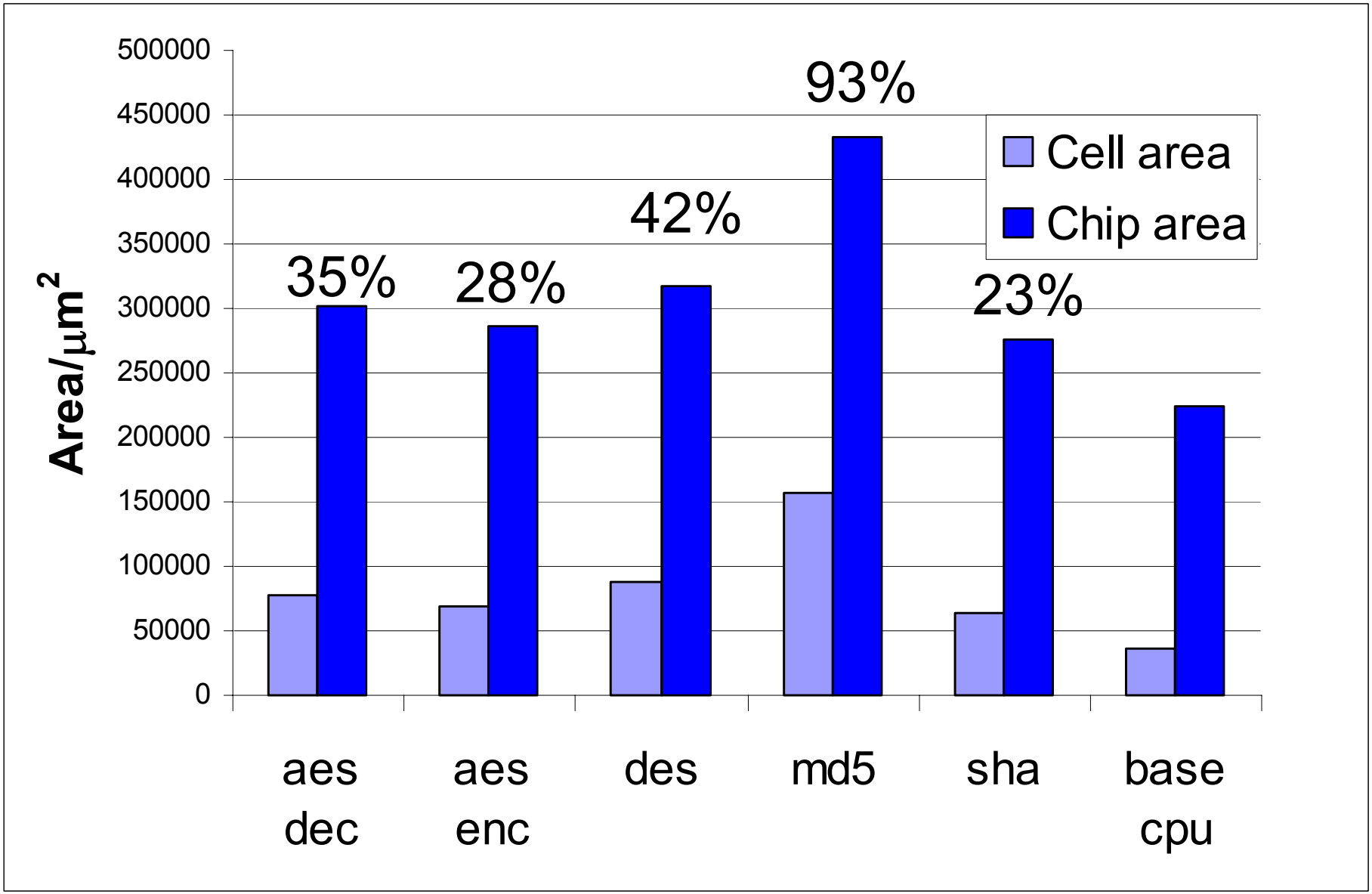


Timing

- 48% of designs meet timing at 200MHz without manual optimization



Area (for maximum speedup)



6. Current and future work

- support memory access in custom instructions
  - automate data partitioning for memory access
  - automate SIMD load/store instructions for state registers
- use architectural techniques e.g. shadow registers
  - improve bandwidth without additional register file ports
- study trade-offs for VLIW style
  - multiple issue vs custom instructions
- extend compiler: e.g. ILP model for cyclic graphs
  - adapt software pipelining for hardware
- more applications: multimedia, multi-core, reconfig.
  - include power consumption etc in optimization

Summary

- complete flow from C to custom processor
- automatic instruction set extension
  - based on integer linear programming
  - optimize schedule length under constraints
- application-specific processor synthesis
  - complete flow: permits real hardware evaluation
- up to 76% reduction in execution cycles
  - 3x area delay product reduction
- max speedup: 23% to 93% area overhead