UCB CS294-88
Declarative Design:
High-Level Descriptions and
Automated Design-Space Exploration
Introduction

Jonathan Bachrach and Krste Asanović

EECS UC Berkeley

January 22, 2013

First Questions

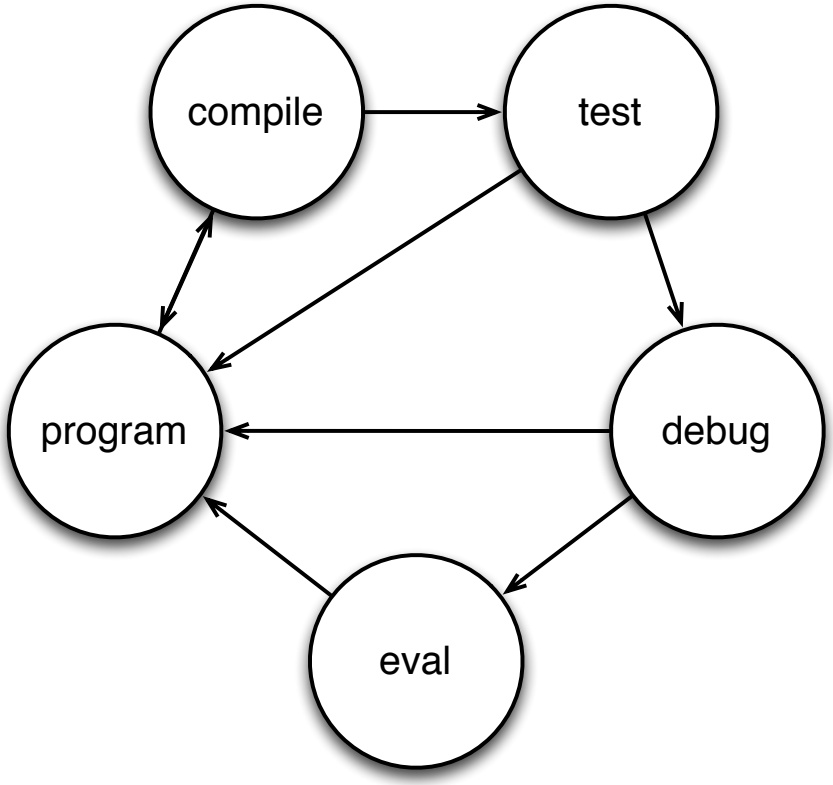
How many people have **not**

- taken cs250?
- used Chisel?
- used an HDL?
- used an FPGA?

- goal: Make energy efficient hardware design dramatically easier
- idea 1: Raise level of abstraction further
- idea 2: Increase design space exploration
- class: Seminar lectures, readings, projects
- desire: Jump start research in Aspire

Design Iteration

- programming
- compiling
- testing
- debugging
- evaluating



“Iron Law of Design Performance”

TD = TP + TC + TT + TD + TE

- up until now easy wins from Moore's law
  - speed, area, and power improved exponentially
- now power constrained
  - first parallelism came to rescue
  - next SOCs are leading the charge but now
  - need specialization to get even more energy efficiency
  - more ops/sec requires less energy/op
- billions of transistors
  - complexity growing
  - huge cost to build chips
  - now can't afford to switch all transistors and number is decreasing exponentially

- Specialization
- High Level Design
- Fast Simulation
- Massive Design Space Exploration

Specialization Status Quo

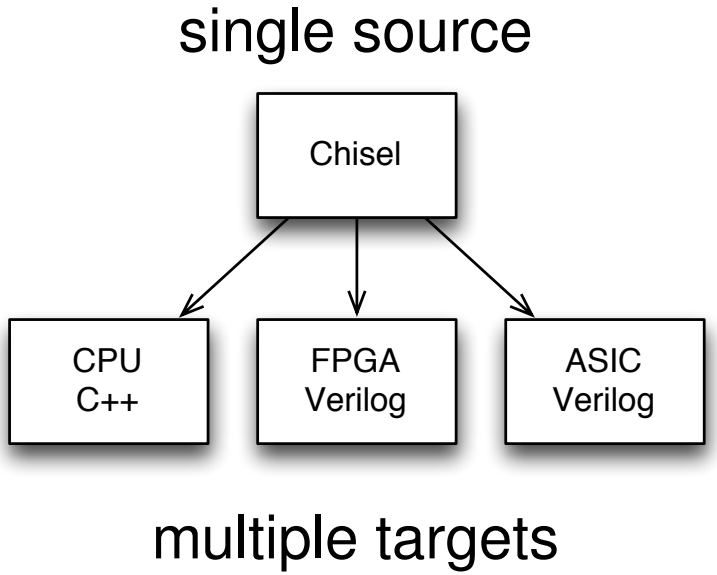
- Detailed microarchitecture with major investment
- Limited parameterization and implementation options
- Painful and slow simulation
- Limited design space exploration
- Think recent cs250 projects

High Level Design Ideas

- programming language concepts – chisel
- parameterized architectural templates – generators
- algorithm separated from implementation (factored)
- software / hardware codesign

Chisel is ...

- Best of hardware and software design ideas
- Embedded within Scala language to leverage mindshare and language design
- Algebraic construction and wiring
- Hierarchical, object oriented, and functional construction
- Abstract data types and interfaces
- Bulk connections
- Multiple targets
  - Simulation and synthesis
  - Memory IP is target-specific

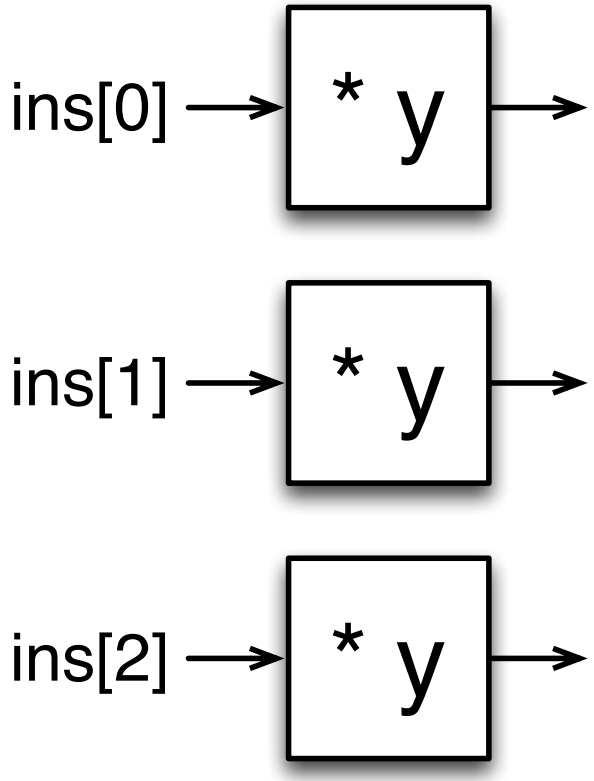


- Chisel is just a set of class definitions in Scala and when you write a Chisel program you are actually writing a Scala program,
- Chisel programs produce and manipulate a data structure in Scala using a convenient textural language layered on top of Scala,
- Chisel makes it possible to create powerful and reusable hardware components using modern programming language concepts, and
- the same Chisel description can generate different types of output

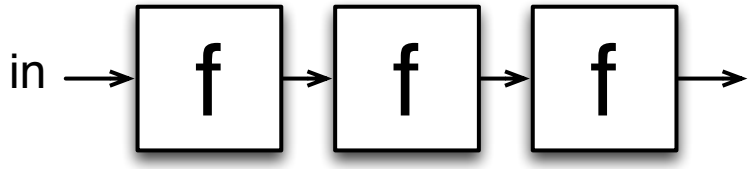
- abstract data types and type inference
- object orientation
- functional programming
- domain specific languages
- design patterns

Example: Functional Composition

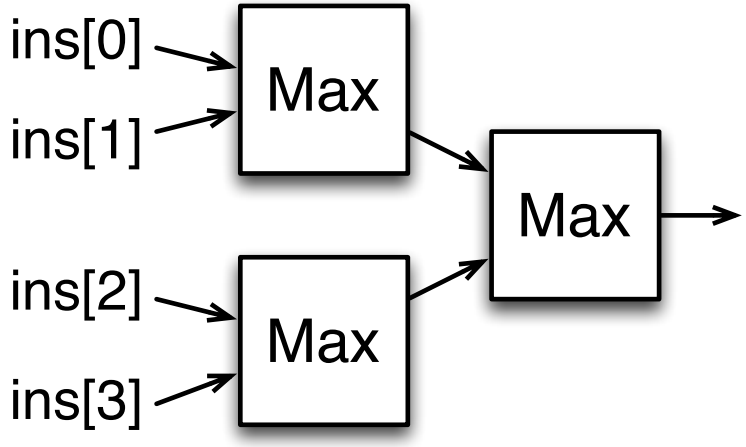

```
Map(ins, x => x * y)
```




```
Chain(n, in, x => f(x))
```




```
Reduce(ins, Max)
```



- procedural construction of RTL
- parameters drive construction
  - mechanism to thread parameters through hierarchical construction
  - application interface for end clients – XML or JSON
  - different kinds of parameters – set, constrained, and free
  - encode backend dependencies
- hardware / software codesign
  - split into hardware and software
  - support software like compilers, assemblers, loaders etc

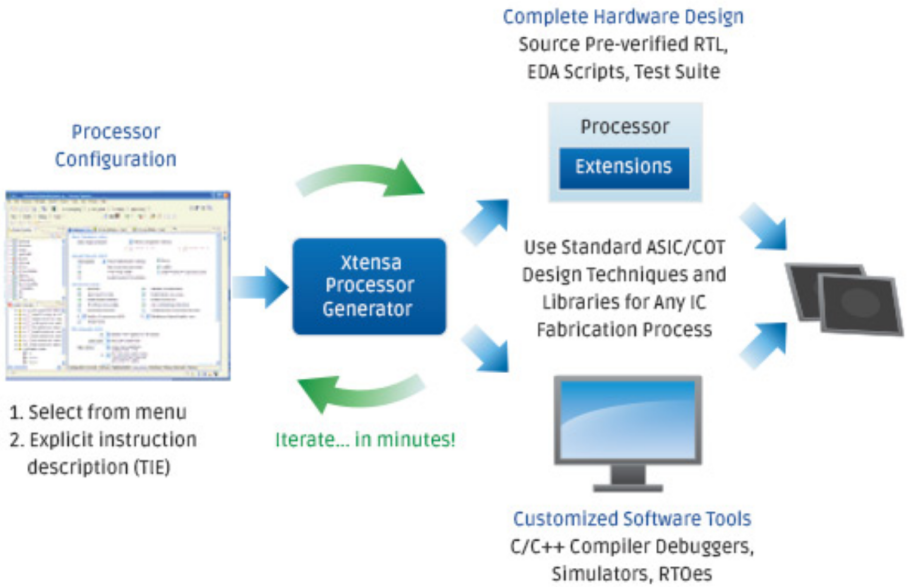
Example: Generator


```
class Cache(cache_type:    Int = DIR_MAPPED,
            associativity: Int = 1,
            line_size:     Int = 128,
            cache_depth:   Int = 16,
            write_policy:  Int = WRITE_THRU
           ) extends Component {
  val io = new Bundle() {
    val cpu = new IoCacheToCPU()
    val mem = new IoCacheToMem().flip()
  }
  val addr_idx_width = log2(cache_depth).toInt
  val addr_off_width = log2(line_size/32).toInt
  val addr_tag_width = 32 - addr_idx_width - addr_off_width - 2
  val log2_assoc     = log2(associativity).toInt
  ...
  if (cache_type == DIR_MAPPED)
    ...
```

- need auxiliary software to be generated (e.g., compiler)
- not easy to identify where hardware / software boundary is
- want to leave decision to profiling
- more productive to write in one language
- could imagine creative hardware and JITs

Example: Codesign – Tensilica

- design application specific instruction processor (ASIP)
- designed in architectural language
- add application specific instructions to base processor
- automatically generates compiler, assembler, core, etc
- $150K per chip

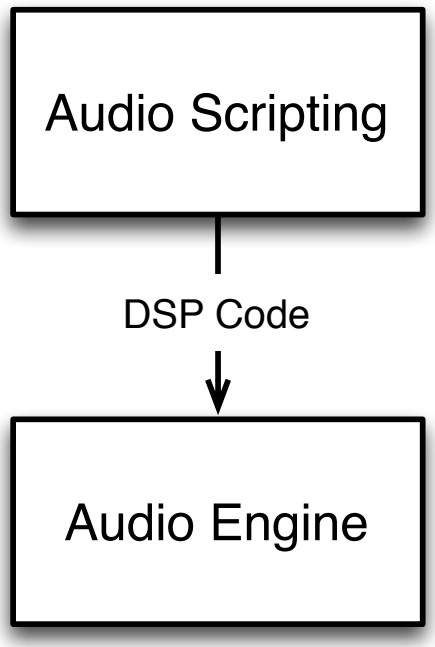


Tensilica has automated the process of creating customized dataplane processors.

from `http://www.tensilica.com/learning-center/dataplane-design`

Example: Wiring All The Way Down

- Can write both audio scripts and engines in Chisel
- Can choose which part is baked into hardware
- For example, can map entire DSP to FPGA or ASIC



- design programmed at high level
- use common patterns for transforming high level design into implementation
- can actually implement these patterns and their transformations directly in Chisel
- in 294-88 we will focus on reifying patterns into chisel

Example: Multiported Memory

- true multiported
- banked multiported
- stream-buffered multiported
- cached multiported
- replicated state multiported

Example: Rate Balancing

- to make faster use parallelism – expand space / shrink time
  - unrolling (for processing units)
  - banking (for memories)
  - multiporting (for memories)
  - widen links (for networks)
- to make slower use time multiplexing – shrink space / expand time
  - share links with bus
  - share memories with common memory
  - use multiple cycles on single port
  - multithread computations onto a common pipeline
  - schedule a dataflow graph onto a single ALU

Example: Time Multiplexing in Chisel

- time multiplex many components using fewer
- vec of virtual components
- stateful wires

```
val cores = TVec(n){ (new Core()).io }
... wire cores together ...
```

- layer language on top of Chisel (in Scala)
- new datatypes
- operator overloading
- macros

Example: EDSL – Process Language

- natural way to write controllers
- functional combinators
- state update orthogonal
- can implement in a variety of fashions

- DO...
- EXEC(c) a / EXEC a
- STOP
- SKIP / WAIT(n)
- SEQ(a, ...)
- PAR(a, ...)
- ALT(c, a1, a2)
- WHILE(c) a / LOOP a

Example: Chisel Audio Support

- Flo and Dbl data types and ops
- Add FP support in C++ backend
- Audio harness with mics, speakers, and controls

Example: Factored Design – Halide

- image processing DSL
- factoring into dataflow and scheduling

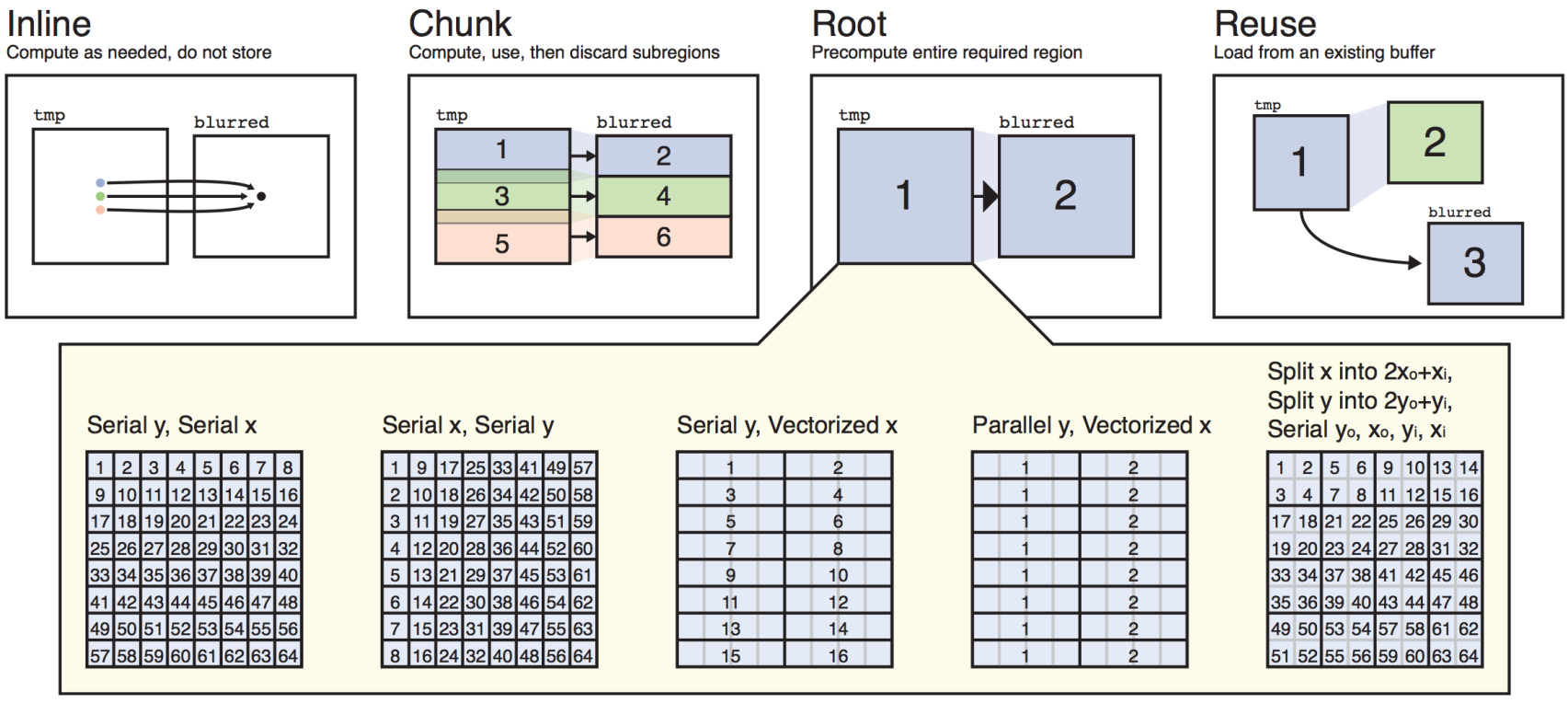
```
Func halide_blur (Func in) {
  Func tmp, blurred;
  Var x, y, xi, yi;

  // The algorithm
  tmp(x,  y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3
  blurred(x, y) = (tmp(x, y-1) + tmp(x, y) + tmp(x, y+1))/3;

  // The schedule
  blurred.tile(x, y, xi, yi, 256, 32)
         .vectorize(xi, 8).parallel(y);
  tmp.chunk(x).vectorize(x, 8);

  return blurred;
}
```

Halide Scheduling



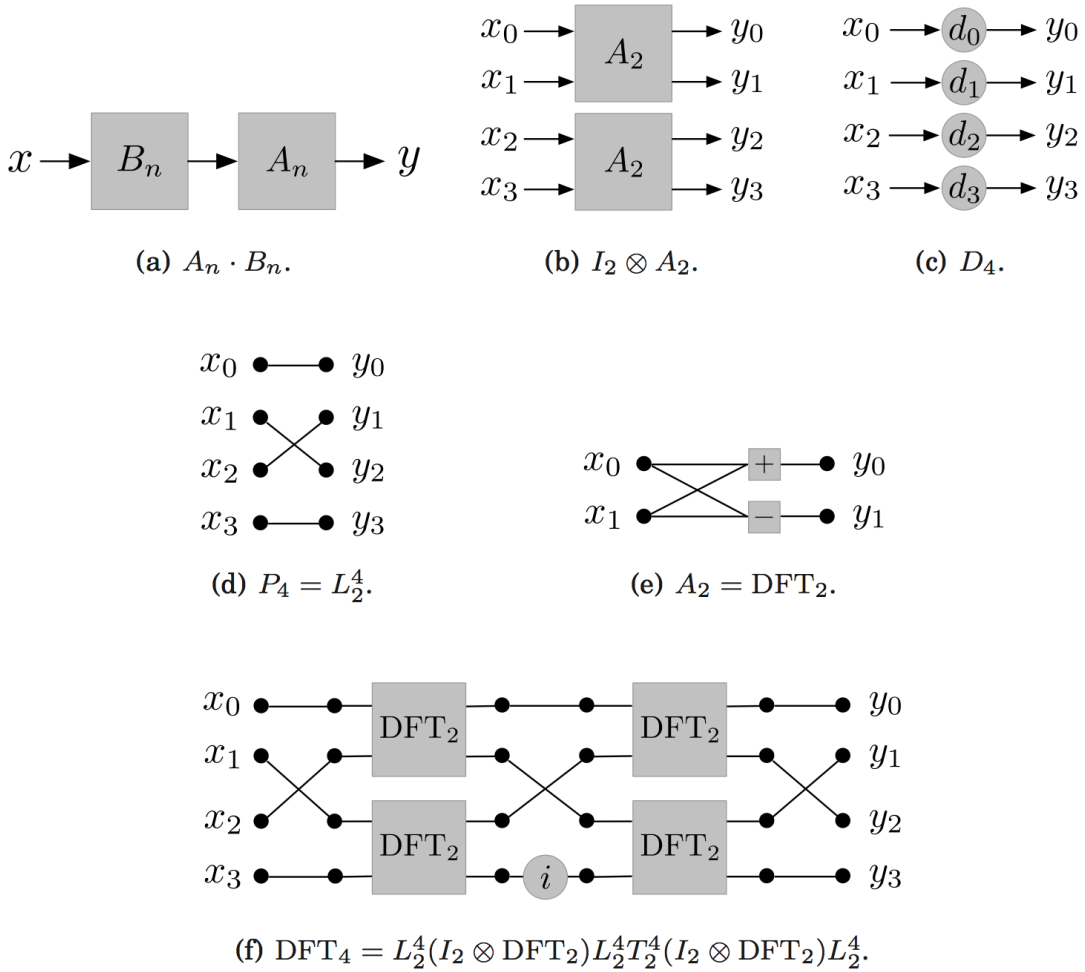
from Decoupling Algorithms from Schedules for Easy Optimization of Image Processing Pipelines by Ragan-Kelley, Adams, Paris, Levoy Amarasinghe, Durand in siggraph 2012

- hardware / software co-optimization framework for DSP application
- specification of FFT using series of matrix operations
- select streaming or iterative reuse of operations
- easy exploration of design space for given algorithm

$$DFT_4 = L^4_2(I_2 \otimes DFT_2)L^4_2 T^4_2(I_2 \otimes DFT_2)L^4_2 \quad (1)$$

$$\prod_{I_o=0}^{k/d-1} \left(\prod_{I1=0}^{d-1} (I_{nm/w} \otimes^{sr} (I_{w/n} \otimes A_n)) \right) \quad (2)$$

Spiral Formula Elements



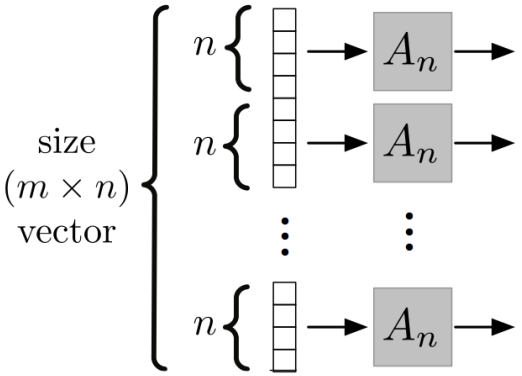
(a) $A_n \cdot B_n$. (b) $I_2 \otimes A_2$. (c) D_4.

(d) $P_4 = L^4_2$. (e) $A_2 = \mathrm{DFT}_2$.

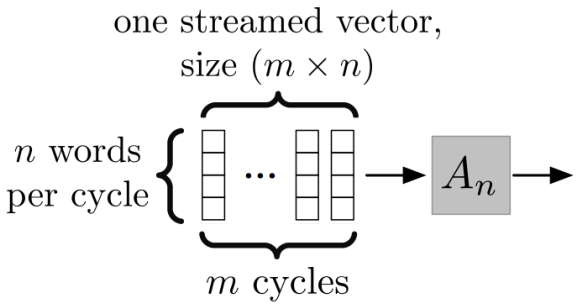
(f) $\mathrm{DFT}_4 = L^4_2(I_2 \otimes \mathrm{DFT}_2)L^4_2T^4_2(I_2 \otimes \mathrm{DFT}_2)L^4_2$.

Fig. 1. Examples of formula elements and their corresponding combinational datapaths.

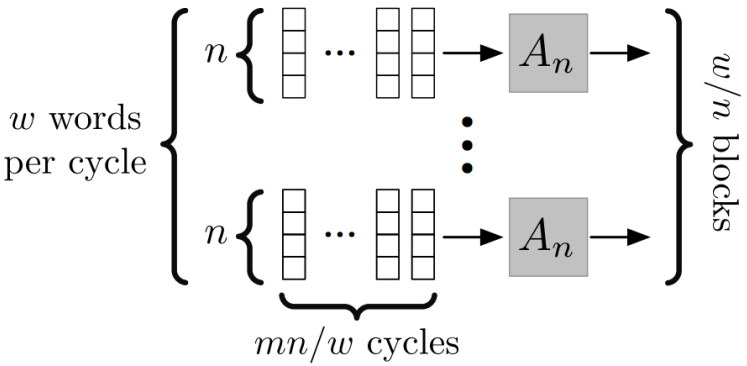
from Computer Generation of Hardware for Linear Digital Signal Processing Transforms by MILDER, FRANCHETTI, and HOE



(a) No streaming reuse (width = mn): $I_m \otimes A_n$.



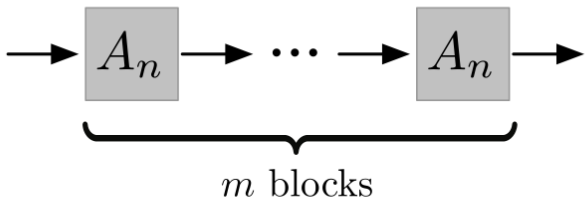
(b) Full streaming reuse (width = n): $I_m \otimes^{\mathrm{sr}} A_n$.



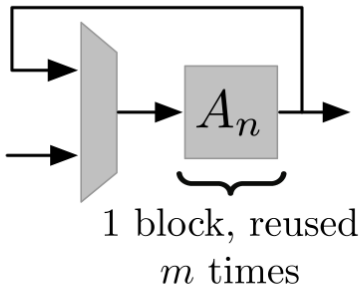
(c) Partial streaming reuse (width = w): $I_{mn/w} \otimes^{\mathrm{sr}} \left(I_{w/n} \otimes A_n\right)$.

Fig. 3. Examples of streaming reuse.

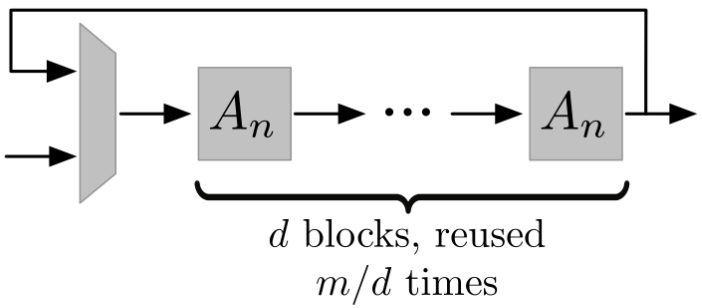
from Computer Generation of Hardware for Linear Digital Signal Processing Transforms by MILDER, FRANCHETTI, and HOE



(a) No iterative reuse (depth = m): $\prod_m A_n$.



(b) Full iterative reuse (depth = 1): $\prod_m^{\text{ir}} A_n$.



(c) Partial iterative reuse (depth = d): $\prod_{m/d}^{\text{ir}} \left(\prod_d A_n\right)$.

Fig. 5. Examples of iterative reuse.

from Computer Generation of Hardware for Linear Digital Signal Processing Transforms by MILDER, FRANCHETTI, and HOE

Design Space Exploration

- complexity
  - solution space – (e.g., joules / op, area, ...)
  - problem space – (e.g., cache size, number cores, ...)
- optimization strategies
  - pareto optimality – dominates
  - blended solution – cost function with constraints
- evaluation
  - simulation
  - analytical
- exploration
  - basic
  - pruning

Pareto Dominance and Optimality

- assume n objectives
- pareto dominant solution – is at least better in one objective while being at least the same in the others
- pareto optimal solution – if no other solution dominates it

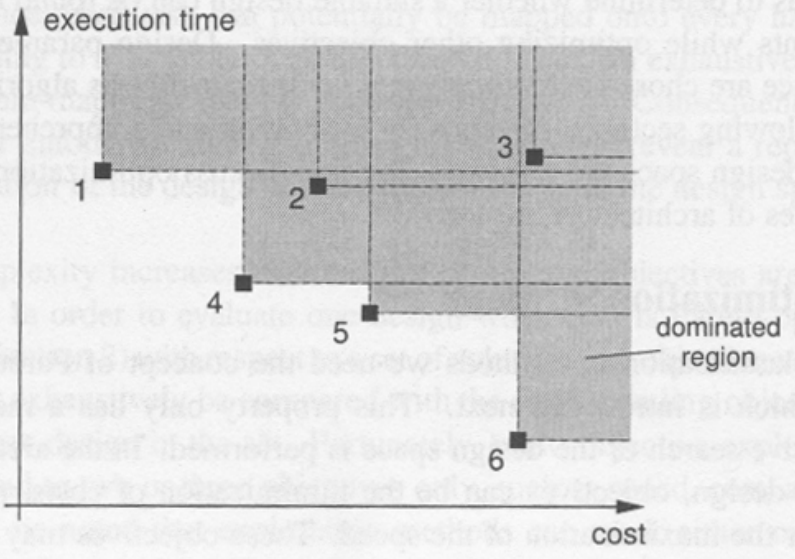


Figure 5.2. Two-dimensional design space with Pareto-optimal designs 1, 4, 5, and 6.

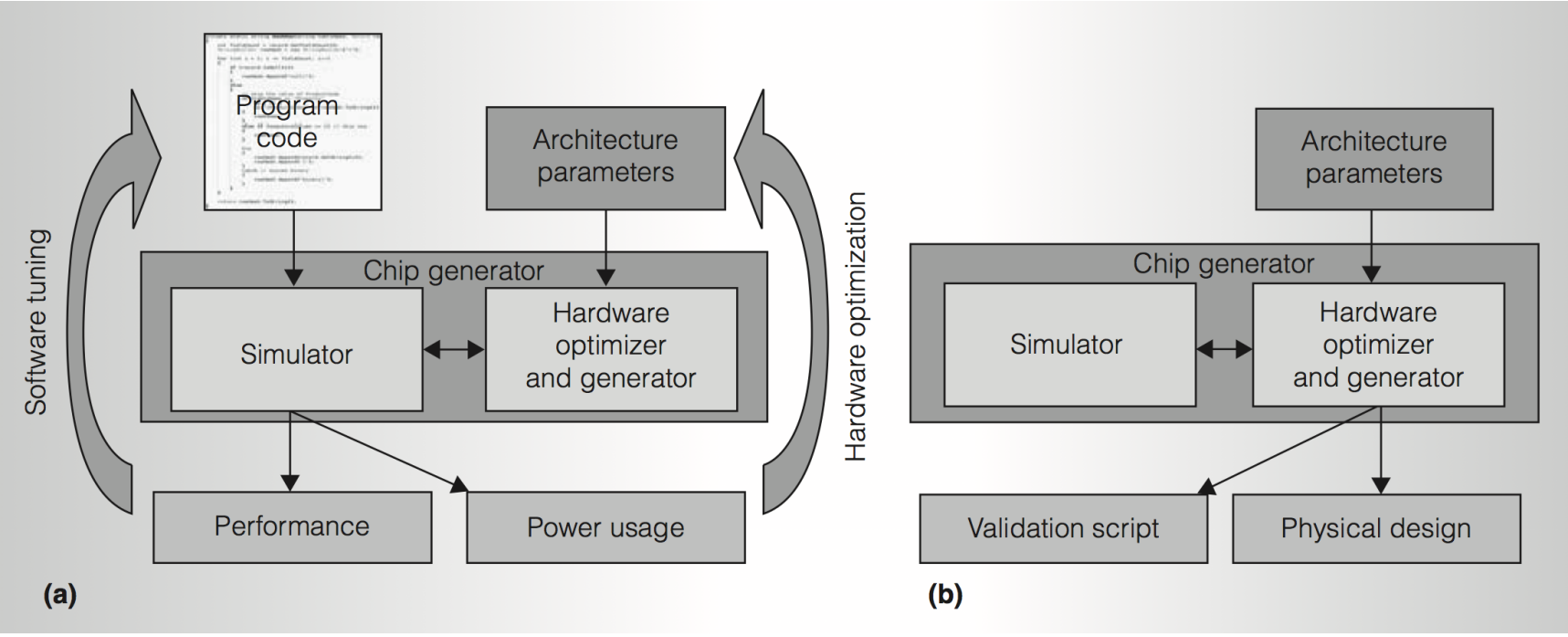
from Building ASIPS: The MESCAL Methodology by Gries, Keutzer, Meyr, Martin Springer Book

Exploration Techniques

- basic
  - exhaustive
  - randomly sampling
  - guided search
  - ad hoc techniques
- pruning
  - hierarchical exploration
  - Subsampling of the design space
  - subdividing the design space into independent parts
  - sensitivity analysis of design parameters

Exploration Workflow

- input energy budget
- simulation for performance
- ASIC workflow for energy usage
- fed back for refinement



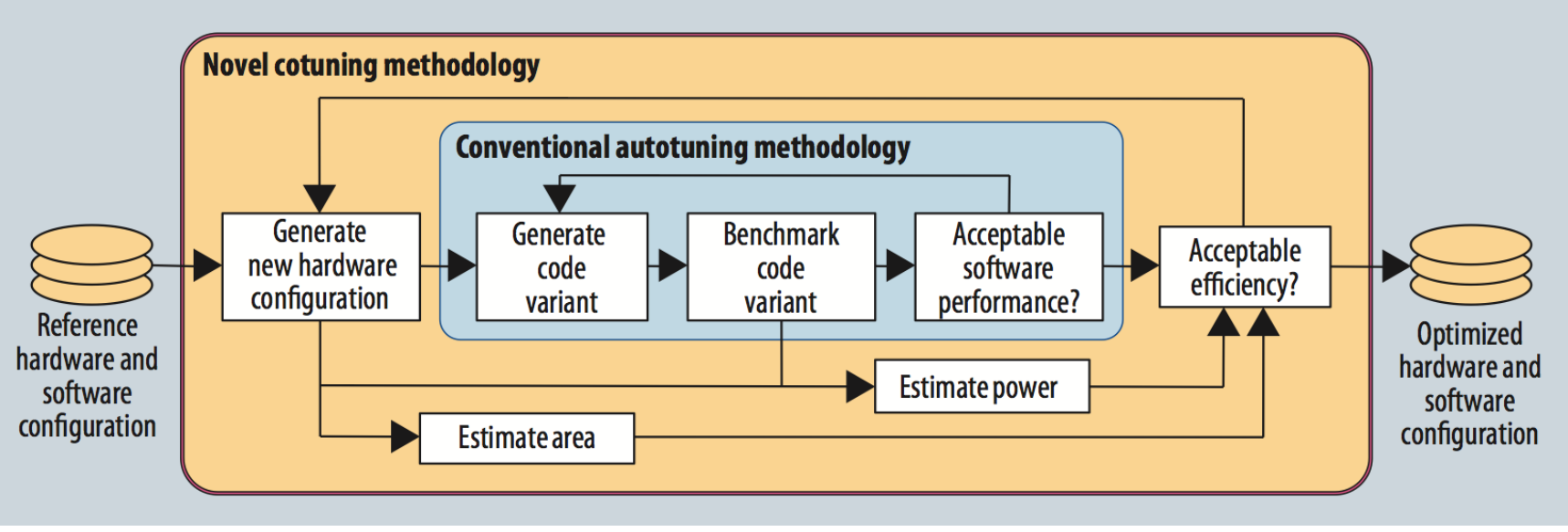
from Rethinking Digital Design: Why Design Must Change by Shacham et al in IEEE Micro magazine

Difficulty of Automatic Optimization

- no gradients and expensive to evaluate design points
- can't afford to explore all combinations
  - potentially explore variance independently
- some optimization approaches only work with restricted formulation

Example: Green Flash

- hardware / software codesign
- grid of experiments

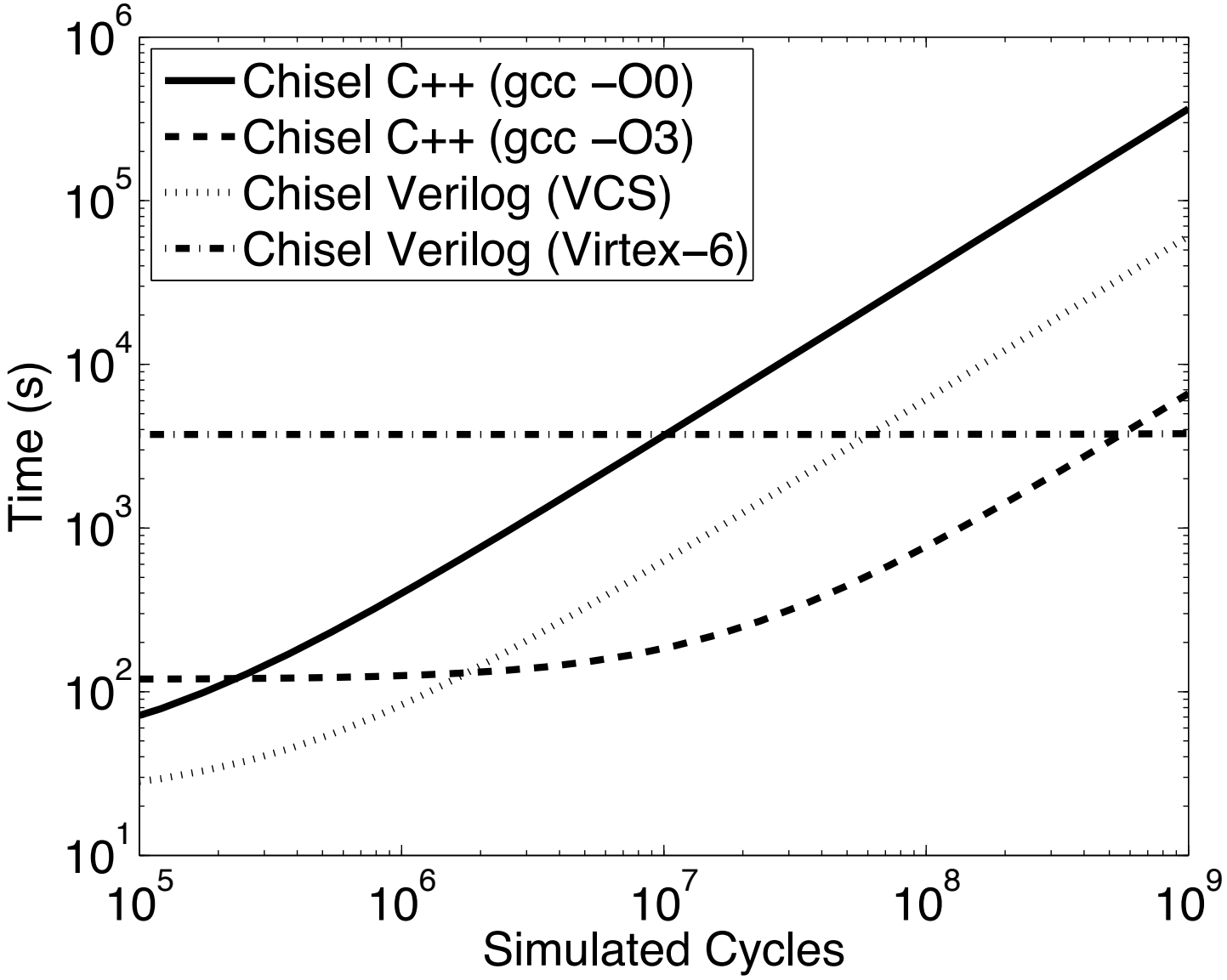


from Energy-Efficient Computing for Extreme-Scale Science by Donofrio et al in IEEE Computer magazine

Fast Evaluation

- setup
  - bottleneck in design loop
  - performance in terms of CPI for instance
  - delay and power estimation through ASIC workflow
- challenges
  - FPGAs slow to synthesize for
  - ASIC workflow slow
  - designs bigger than FPGAs

Overheads for Various Simulation Techniques

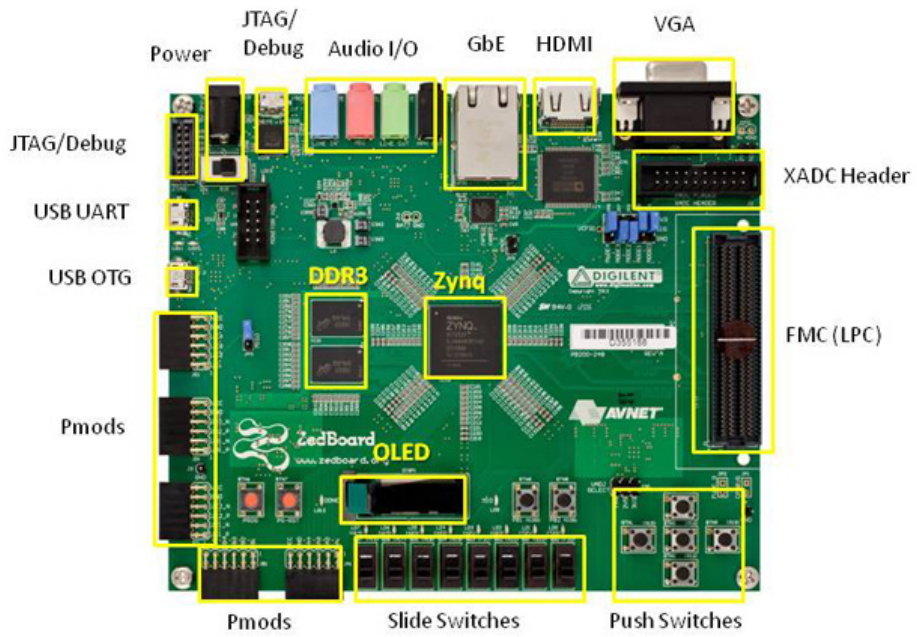


Class Specifics

- seminar with lots of participation
- kickoff of Aspire
- grading 25% based on participation and 75% on projects
- lecture / reading / response
- reading response = good + bad + three questions
- projects (using zedboard)
- start thinking about projects early

Zedboard

- 2 core ARM with FPU running linux
- Xilinx fabric programmed using Xilinx Vivado workflow
- ARM talks to fabric using AXI lite



* SD card cage and QSPI Flash reside on backside of board

- Generators
- Design Patterns
- Embedded Domain Specific Language
- Factored Design
- Hardware / Software Codesign
- Fast Emulation
- Design Space Exploration

Generators

- parameterize
- multiple implementations
- create interface to user or optimizer
- examples
  - processors
  - NOC

- implement a design pattern or family of patterns in Chisel
- example ideas
  - multiported memory
  - rate balancing

- invent a new higher level description of hardware
- example ideas
  - functional programming
  - combinators
  - actors

- split into pure algorithm and resource allocation and scheduling
- example domains
  - video
  - audio
  - DSP
  - xactors
  - processors

- hardware and software designed together
- example ideas
  - tensilica style processor generator
  - scripting all the way down
  - hardware and algorithm evolved simulataneously

Fast Emulation

- performance estimation
- examples
  - Power aware simulation
  - Multiprocessor emulator
  - GPU backend for chisel

Design Space Exploration

- guided exploration of design space
- examples
  - criteria
  - user interface
  - pruning techniques
  - optimization techniques

Assignment for Next Class

- read **Rethinking Digital Design: Why Design Must Change?**
- come prepared with
  - the good
  - the bad
  - three questions
- website will be up by the end of the day.