University of Pittsburgh

HPCSoC Modeling and Simulation Implications

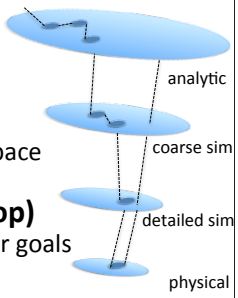
(Sharing three "concerns" from an academic research user perspective using free, open tools. Solutions left to the reader. ☺)

Bruce Childers
childers@cs.pitt.edu
http://www.cs.pitt.edu/~childers
Dietrich School of Arts and Sciences
Department of Computer Science

Concern 1: High-level Design Exploration

Degrees of freedom increase

- Custom compute: processor, memory, IO/storage, network
- Software and hardware both can be customized
- Modeling at scale, breadth of choice at scale

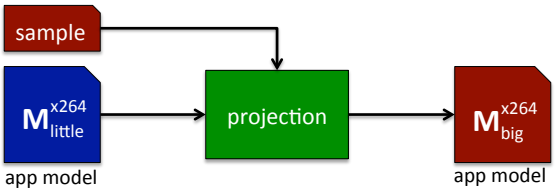


1. **Increasing the level of abstraction**
   - Explore more choices in finite time
   - More layers & higher abstraction to quickly traverse space
   - Multi-layered, multi-fidelity sim & modeling
2. **Agent-based optimization (with engineer-in-loop)**
   - Agents find design neighborhoods directed by engineer goals
   - Move up/down between layers – zoom in/out
   - Confidence aware
3. **Agent "learn" (i.e., models refined)**
   - Refine higher-level models with lower-level discoveries
   - E.g., regression & machine learning models
   - Interacts with the sim & models – need interfaces

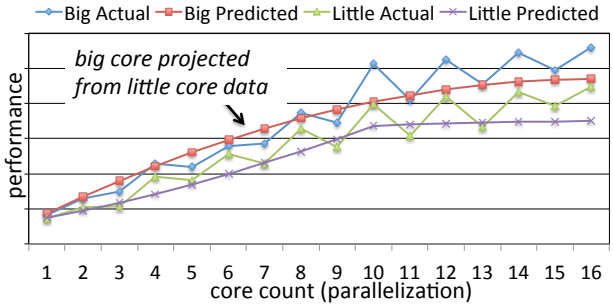
Example: Refine Model to Predict Another Core Type

Example: Little to Big Core

1. Existing little model
2. Sample actual big
3. Transform (project) existing model to a new mode



Multivariate piece-wise linear regression with projection



Concern 2: Modeling and Simulation of Sys. Software

- Not just the hardware but also the software
  - Of course, application models are useful
- **At some point in design flow, the models are lowered enough that we should consider system software as well**
  - Programming model, Development environment (debugging, testing), Compiler, OS/runtime
- **Supporting role**: What is need and cost/disruption to SW layer for custom hardware? How well does it work?
- **First-class citizen**: What customizations can be made to the software layer and how well do they work?

E.g., Compiler

Evaluate an accelerator design possibility

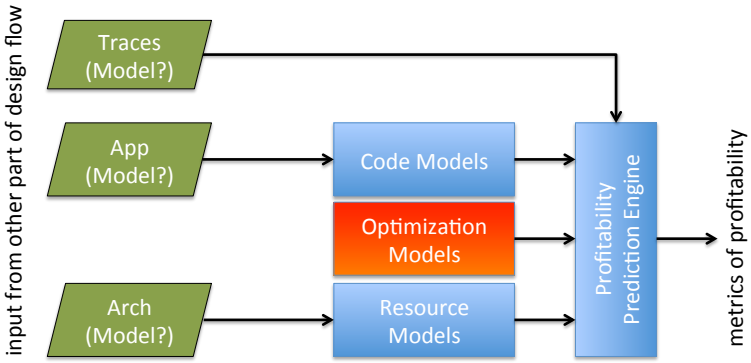
1. Partitioning: divide application between resource types
2. Mapping: assign to specific resources of each type
3. Optimization: transform to best use the type

How "good" is the compiler?

How does the compiler affect the end result (PPR)?

We want to know how well the compiler will do its job without actually doing it (or implementing), especially in early design!

Modeling the Compiler



- Code, Resource models automatically inferred
- Optimization models: Require effort.
- **Model-driven Code Optimization (mostly scalar)**
  - Modeled for profitability for scalar, some locality optimizations
  - Used FPO+models to find good opt. sequences (phase orderings)

E.g., OS (Persistent memory, NVM)

- Persistent memory
  - Flash, PCM, STT-RAM, etc.
  - Different operation, structure, integration
- What about SW implications?
  - Block device through file system
  - Block device with lightweight layer
  - Directly through read/write instructions

Need models to evaluate: choice, system design (disruption), PPR (metrics)

Samsung F2FS
Optimized for Flash storage
Sequential writes desired
Log-structured file system
Optimization interaction w/FTL

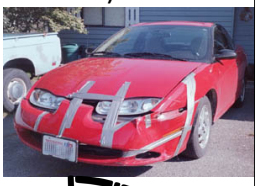
< Aged >

Items		Ext4	F2FS	Improv.
Contact sync time (seconds)		437	375	17%
App install time (seconds)		362	370	-2%
RLBench (seconds)		99.4	85.1	17%
IOZone With AppInstall (MB/s)	Write	7.3	7.8	7%
	Read	16.2	18.1	12%

Joo-Young Hwang, F2FS: A New File System Designed for Flash Storage in Mobile, Embedded Linux Conference, Europe, 2012

Concern 3: Too much Duct Tape

- Complex systems, abstraction, multi-fidelity, multi-level, composition, big collection of tools & models....
- Awkwardly integrated tools (Python? Perl? Bash? Awk?)
- Do you ***trust*** the result??? Really?
- Properties of trust [static & dynamic]
  - Model assumptions (correct, what is/isn't assumed)
  - Bug-free (yea, yea) model implementation
  - Integration & composition
  - Meaningful composition
- Methods, measures of trustworthiness



E.g., ModSim 2014 - Gem5: Performance bug in core pipeline discovered when integrated with power and thermal modeling

Trust

- (My comments reflect an "open" SoC economy.)
- Often "feels" overly ad hoc (ok, random!)
  - Favorite frameworks, languages, libraries, tools, etc.
  - Side effect of "more important work" (publish or perish!)
- Need more principled, formal & accountable
  - Good ***software engineering*** (a **real** problem for "free" artifacts)
  - Documented***,*** reasonably robust implementation
  - What is and is ***not*** modeled, degree of testing
  - ***Interfaces***, at least some definition, preferred formalism for reasoning
    - E.g., Coq was used to build a formally verified compiler! Experts, though...
  - Computable ***confidence metric***(s) (uncertainty quantification)

The End