Ithemal: Accurate, Portable and Fast Basic Block Throughput Estimation using Deep Neural Networks

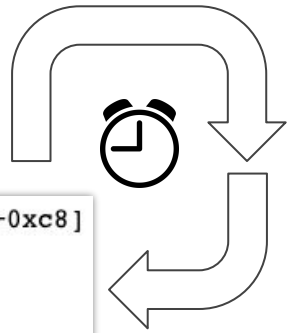
Paper Authors: Charith Mendis, Alex Renda, Saman Amarasinghe, Michael Carbin

Presenters: Yongyu Deng, Ziqing Xu, Daniel Geng, Max Hamilton

Throughput of a Basic Block

- **Throughput** - number of cycles needed to execute a block in *steady state*

- **Uses**
  - Register Allocation
  - Instruction Scheduling



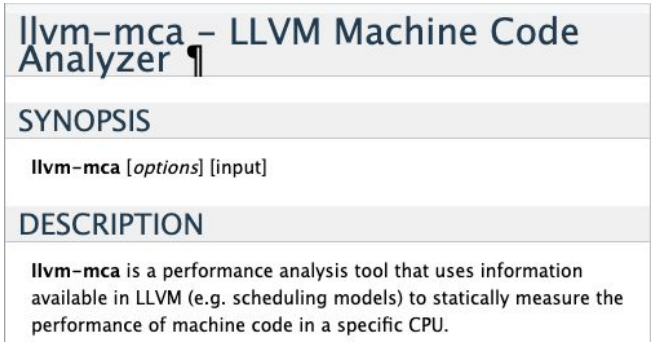
```
mov     rdx,QWORD PTR [rsp+0xc8]

mov     rdi,r13
mov     rsi,rax
```

Basic block from Clang

Calculating Throughput

- Brute Force (Dynamic Analysis)
  - Just run the block in a loop until steady state
- Static Code Analyzers
  - **LLVM-MCA**
    - LLVM Machine Code Analyzer
    - Pushed in 2018 by Andrew Di Biagio (Sony)
  - **IACA** (End of Life)
    - Intel Architecture Code Analyzer
    - Uses closed source info about Intel microprocessors
  - Both use analytical models to calculate throughput
  - (And to be fair, these tools do quite a bit more than just throughput analysis)



llvm-mca – LLVM Machine Code Analyzer ¶

SYNOPSIS

llvm-mca [*options*] [input]

DESCRIPTION

llvm-mca is a performance analysis tool that uses information available in LLVM (e.g. scheduling models) to statically measure the performance of machine code in a specific CPU.

https://llvm.org/docs/CommandGuide/llvm-mca.html

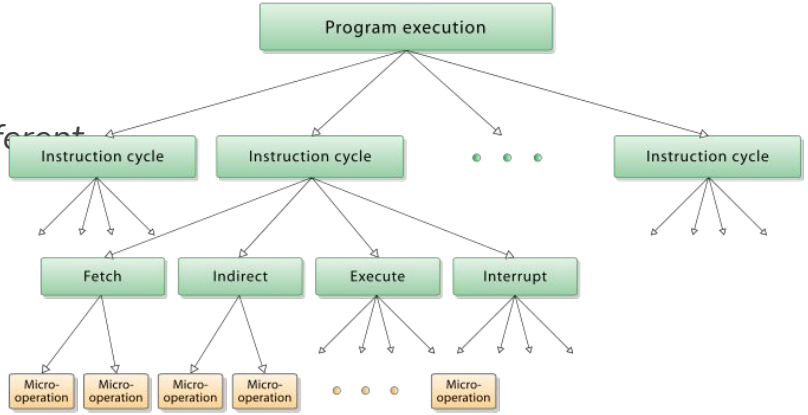
Issues with Current Methods

- Dynamic Analysis
  - Slow and expensive (needs to run until steady state)
  - Requires sandboxing, which adds overhead

- Static Analysis
  - Relies heavily on the model
  - Writing a model takes **time**, can be **error-prone**, and requires **knowledge of the processor**
  - Tradeoffs between **accuracy** and **portability/speed**

Difficulties in Model Building

- Microarchitectures
  - ISAs (such as x86-64) are implemented with *different **microarchitectures***
  - “Macro”-instructions are translated to “micro”-instructions
  - Micro-ops can then be optimized through:
    - Micro-op **fusion**
    - **Out-of-order** execution of micro-ops
    - **Register renaming**
  - This makes writing the model **very complicated**

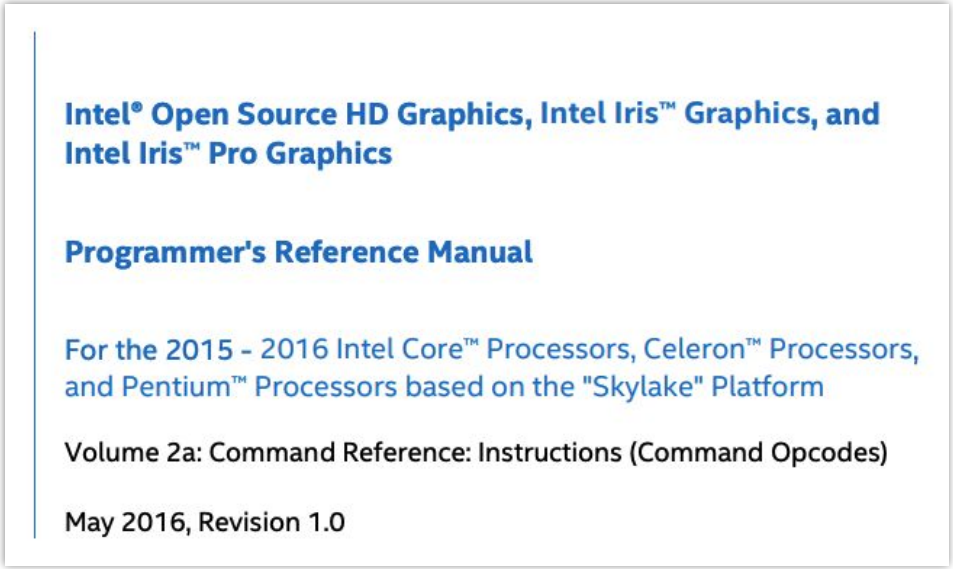


High-level diagram

Difficulties in Model Building

- Portability
  - ISAs (x86-64) are relatively stable, but new microarchitectures are introduced frequently
    - 2012 - Ivy Bridge
    - 2013 - Haswell
    - 2015 - Skylake
  - Microarchitectures are not open-sourced, requiring guesswork
  - **Incomplete** and **incorrect** documentation
    - Often produced through reverse engineering

Intel Skylake chips



This Skylake manual is 1292 pages long, and is only volume 2a out of 21!

Ithemal: A Data Driven Approach

A Data Driven Approach

- Why model microprocessors by hand, when we can just learn it...
- High level idea: generate data, feed to deep learning model
- Only requires description of the Instruction Set Architecture (ISA)

Motivating Examples

	`vxorps xmm0, xmm0,xmm0`	`mov [rbp+0x70], rax` `mov rax, 0x01`	`shl rbx, 0x02` `mov rdi, rbx`
	(a)	(b)	(c)
Actual	32	103	83
llvm-mca	100	100	50
IACA	24	84	96
Ithemal	35	102	83

Table of x86-64 assembly code, with the actual measured throughput (number of cycles to execute a basic block in steady-state), and estimate throughput by llvm-mca, IACA, and Ithemal

Motivating Examples

x86-64 assembly

	vxorps xmm0, xmm0,xmm0	mov [rbp+0x70], rax mov rax, 0x01	shl rbx, 0x02 mov rdi, rbx
	(a)	(b)	(c)
Actual	32	103	83
llvm-mca	100	100	50
IACA	24	84	96
Ithemal	35	102	83

Table of x86-64 assembly code, with the actual measured throughput (number of cycles to execute a basic block in steady-state), and estimate throughput by llvm-mca, IACA, and Ithemal

Motivating Examples

x86-64 assembly

	vxorps xmm0, xmm0,xmm0	mov [rbp+0x70], rax mov rax, 0x01	shl rbx, 0x02 mov rdi, rbx
	(a)	(b)	(c)
Actual	32	103	83
llvm-mca	100	100	50
IACA	24	84	96
Ithemal	35	102	83

Throughputs

Table of x86-64 assembly code, with the actual measured throughput (number of cycles to execute a basic block in steady-state), and estimate throughput by llvm-mca, IACA, and Ithemal

Motivating Examples

	`vxorps xmm0, xmm0,xmm0`	`mov [rbp+0x70], rax` `mov rax, 0x01`	`shl rbx, 0x02` `mov rdi, rbx`
	(a)	(b)	(c)
Actual	32	103	83
llvm-mca	100	100	50
IACA	24	84	96
Ithemal	35	102	83

- **Implementation Errors**
  - (a) zeros out `xmm0` by `xor`-ing
  - Zeroing is very very common, and is implemented with a faster, optimized data path
  - IACA is accurate, while llvm-mca is not

Motivating Examples

	`vxorps xmm0, xmm0,xmm0`	`mov [rbp+0x70], rax` `mov rax, 0x01`	`shl rbx, 0x02` `mov rdi, rbx`
	(a)	(b)	(c)
Actual	32	103	83
llvm-mca	100	100	50
IACA	24	84	96
Ithemal	35	102	83

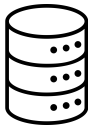
- **Implementation Errors**
  - (b) implements a pair of `mov` instructions
  - IACA identifies a micro-op fusion opportunity, and predicts a lower cycle count
  - This fusion opportunity is not actually used

Motivating Examples

	`vxorps xmm0, xmm0,xmm0`	`mov [rbp+0x70], rax` `mov rax, 0x01`	`shl rbx, 0x02` `mov rdi, rbx`
	(a)	(b)	(c)
Actual	32	103	83
llvm-mca	100	100	50
IACA	24	84	96
Ithemal	35	102	83

- **Documentation Errors**
  - (c) left shifts `rbx` and then moves it to `rdi`, a data dependency
  - llvm-mca uses the documentation
  - But the documentation assumes no dependency

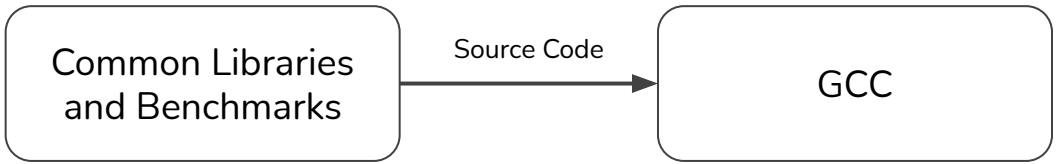
High Level Approach

1. Dataset creation: (x86-64 instructions -> clock cycles) 
2. Tokenize 
3. Train a hierarchical RNN 

Dataset Generation

Common Libraries and Benchmarks

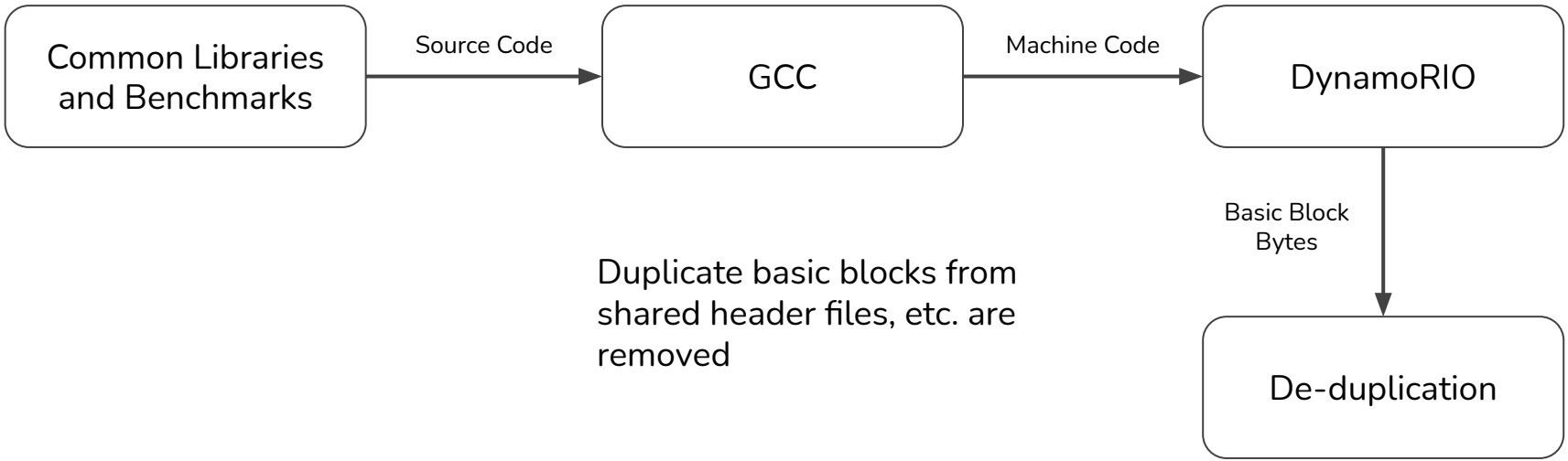
Dataset Generation



Dataset Generation

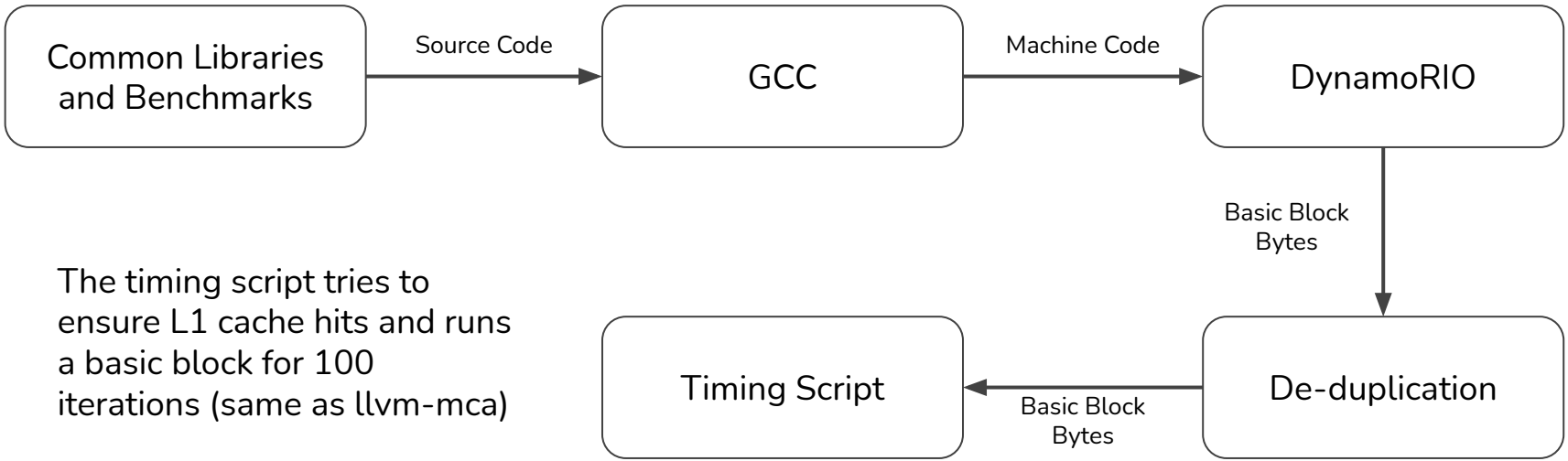
DynamoRIO extracts the byte representation of basic blocks

Dataset Generation



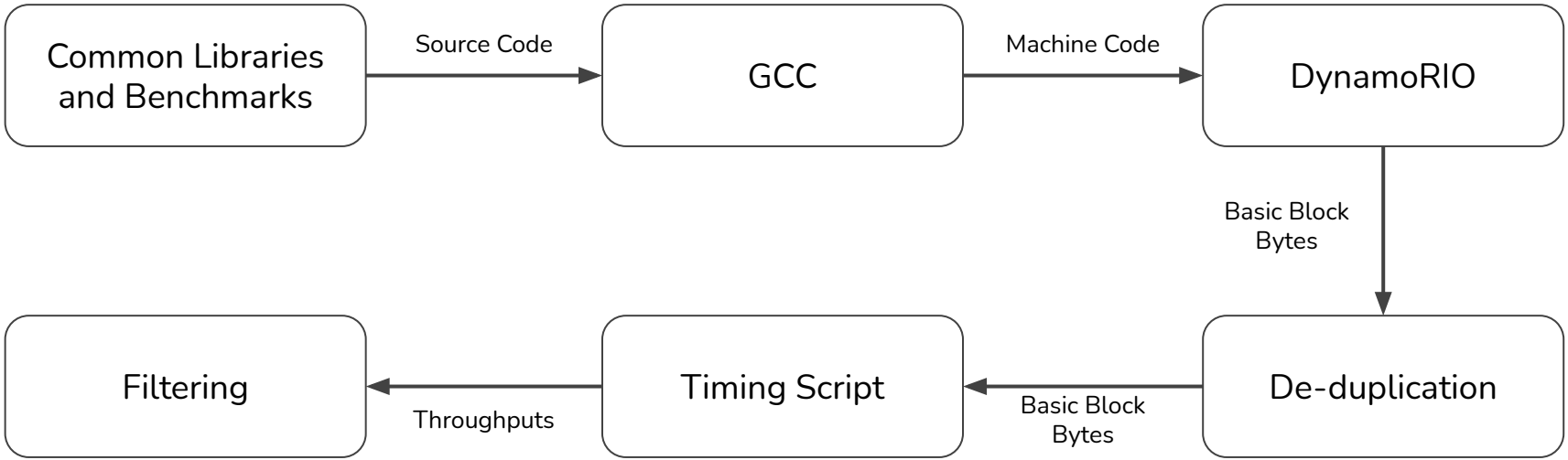
Duplicate basic blocks from shared header files, etc. are removed

Dataset Generation



The timing script tries to ensure L1 cache hits and runs a basic block for 100 iterations (same as llvm-mca)

Dataset Generation



Results that had too many L1 cache misses or were preempted are discarded

Dataset Generation

Benchmark suite	Description	#Total Blocks	#Unique Blocks
Linux Shared Libraries	linux loaders, standard library and other utilities	313846	103977
SPEC2006 (SPEC, 2006)	benchmark suite with compilers, chess engines, video compression and various simulation applications. Commonly used for benchmarking compilers	247047	141051
SPEC2017 (SPEC, 2017)	similar to SPEC2006, but with a larger variety	616899	234588
NAS (NASA, 1991–2014)	benchmarks with stencil computations (dense loops)	3935	1813
polybench-3.1 (Pouchet, 2012)	polyhedral compilation test suite (dense loops)	1900	859
TSVC (Maleki et al., 2011)	suite for testing compiler auto-vectorization	5129	2350
cortexsuite (Venkata et al., 2009)	computer vision workloads including neural networks	6582	3968
simd (Ihar et al., 2018)	heavily hand vectorized image processing library (exposes lot of SSE2, AVX, AVX2 variants)	212544	25462
compilers/interpreters	clang (Lattner & Adve, 2004) and different versions of python (2.7,3.5)	2746275	924663
end user applications	gimp filters, firefox, open-office, rhythmbox, etc.	83555	35513
Full Dataset		4237712	1416473

Tokenization

- We have (x86-64 instructions, throughput) pairs
- We need to transform the instructions into a form usable by a deep learning model
- Common format in NLP are **tokens**
  - Have a token for each register and instruction
  - Additional “semantic” tokens

Tokenization

`mul ecx` → (`mul`, <S>, `eax`, `ecx`, <D>, `edx`, `eax`, <E>)

Insert tokens for

- Sources - <S>
- Destinations - <D>
- End - <E>

Tokenization

`mul ecx` → (`mul`, <S>, `eax`, `ecx`, <D>, `edx`, `eax`, <E>)

`add ecx, 0xc7` → (`add`, <S>, `CONST`, <D>, `ecx`, <E>)

All constant immediates get mapped to the `CONST` token

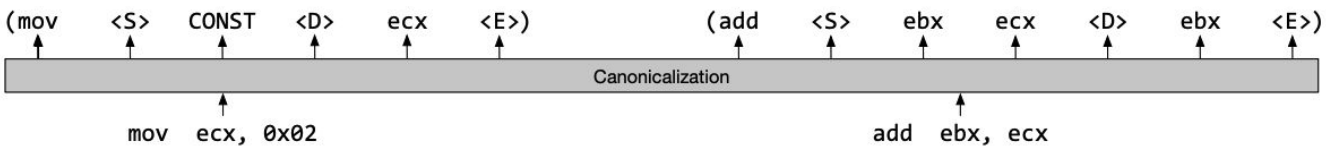
Tokenization

`mul ecx`	→	(`mul`, <S>, `eax`, `ecx`, <D>, `edx`, `eax`, <E>)
`add ecx, 0xc7`	→	(`add`, <S>, `CONST`, <D>, `ecx`, <E>)
`mov [rbp+0x70], rax`	→	(`mov`, <S>, `rax`, <D>, <M>, `rbp`, </M>, <E>)

Address offsets are tokenized by wrapping in “<M> … </M>”

Hierarchical LSTM Model

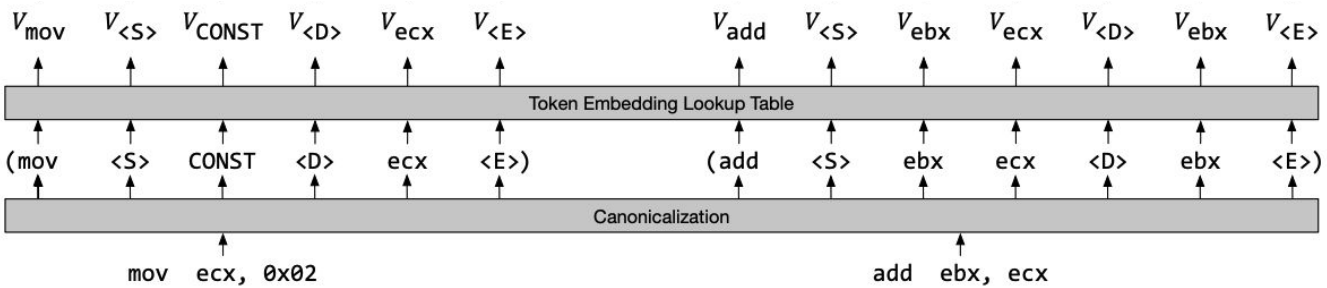
First **canonicalize** (tokenize)



Hierarchical LSTM Model

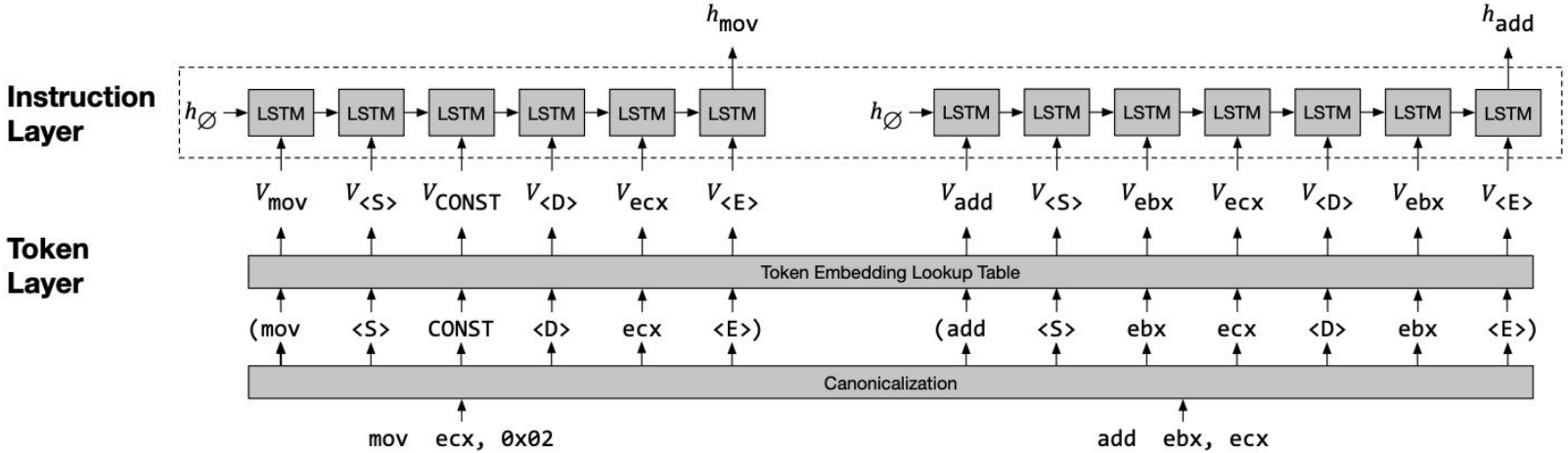
Map tokens to **token embeddings** (vectors)

Token Layer



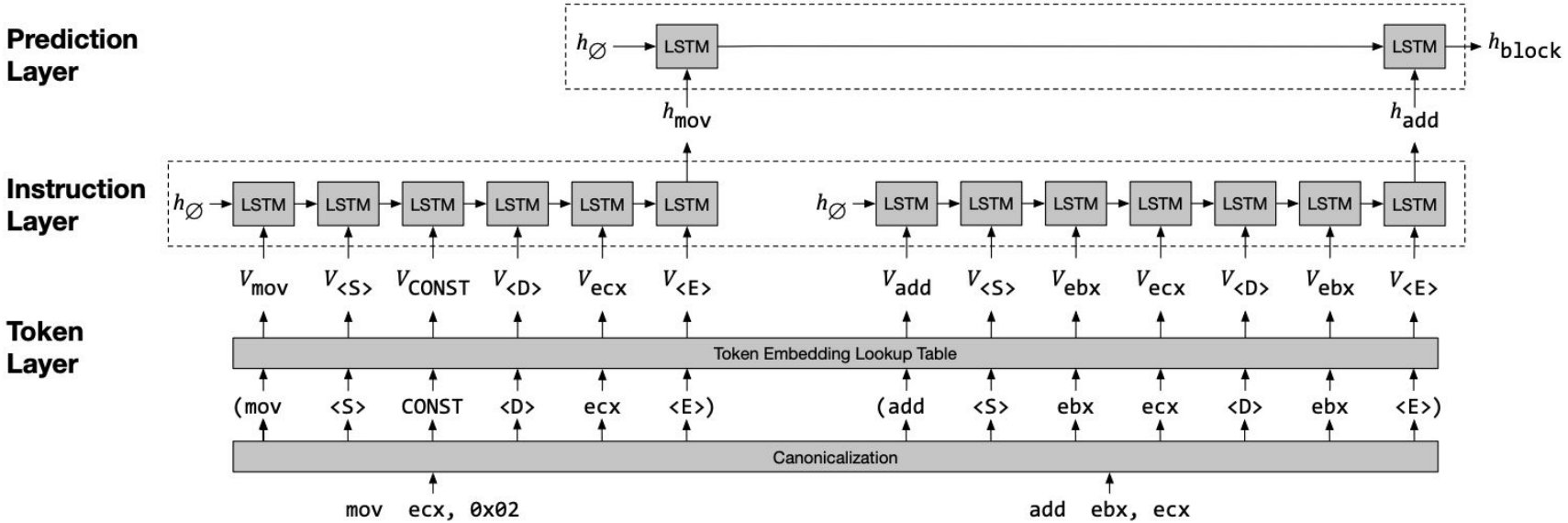
Hierarchical LSTM Model

Process each instruction into an **instruction embedding**



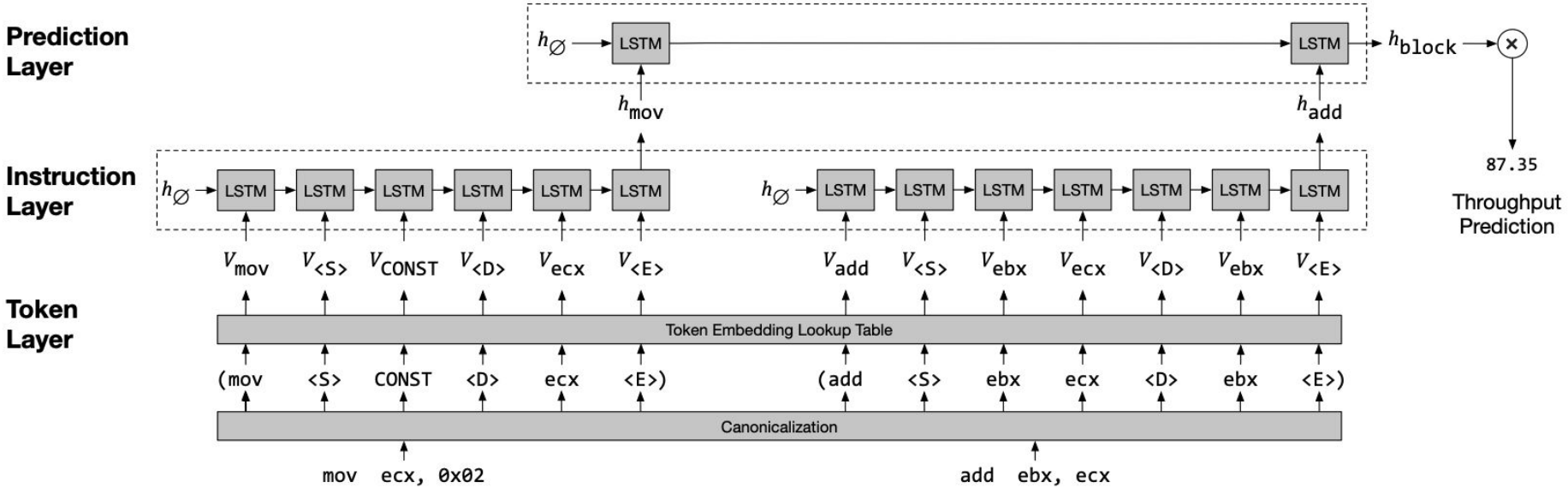
Hierarchical LSTM Model

Process **instruction embeddings** into a single **block embedding**



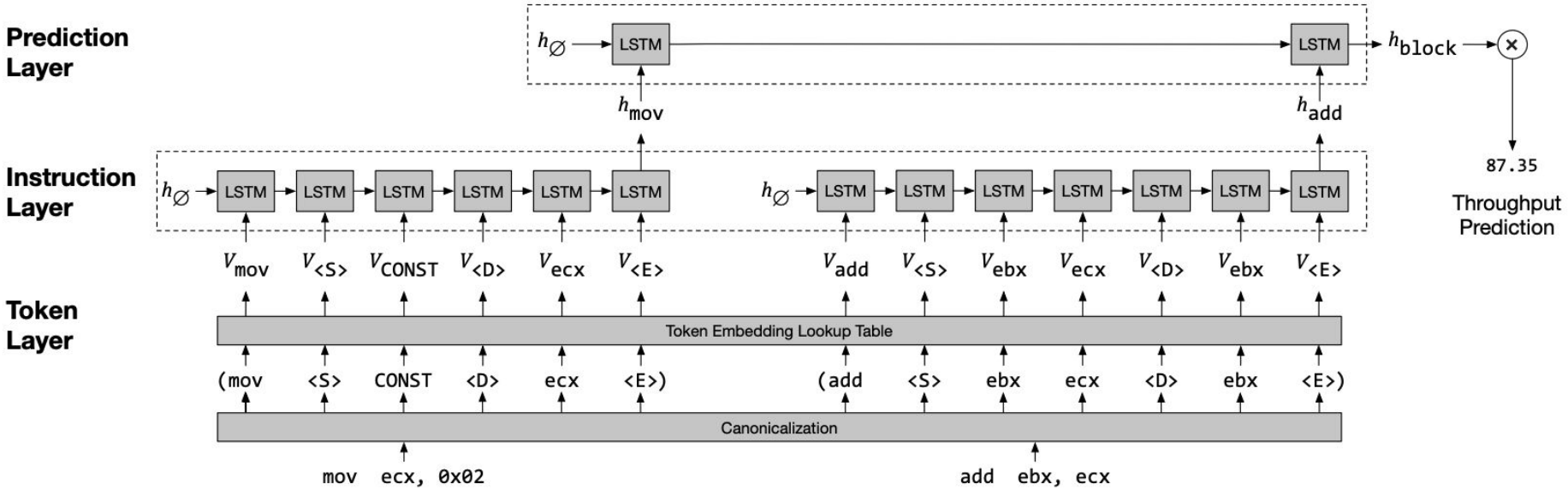
Hierarchical LSTM Model

Apply a linear layer to the **block embedding** to predict the throughput



Hierarchical LSTM Model

Apply a linear layer to the **block embedding** to predict the throughput



Training is standard, with the loss: $\mathcal{L}(\texttt{pred}, \texttt{actual}) = \frac{|\texttt{pred} - \texttt{actual}|}{\texttt{actual}}$

Results

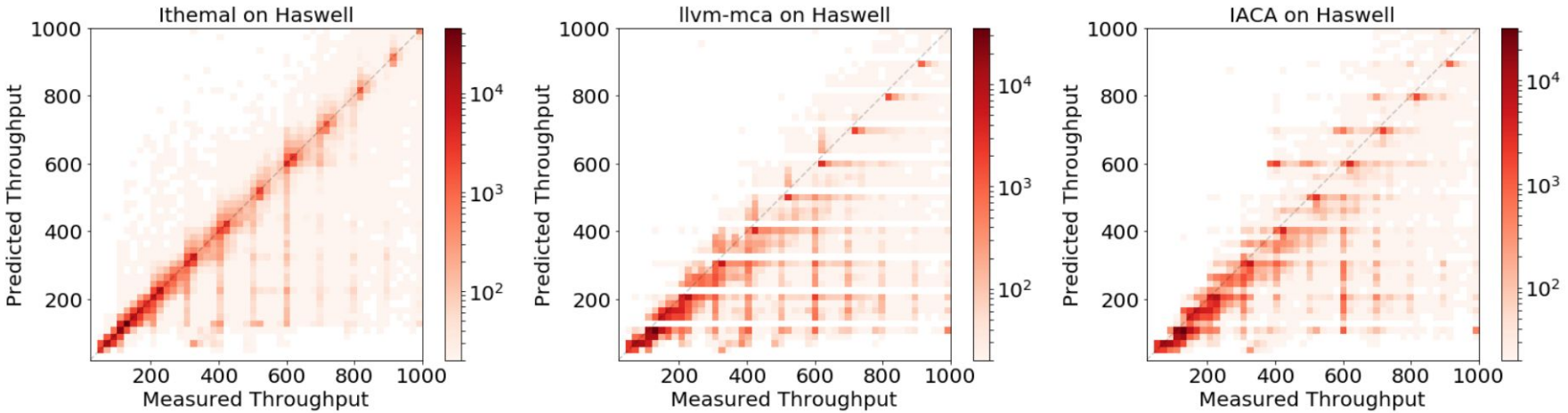


Figure 2. Heatmaps for measured and predicted throughput values under different models for basic blocks with measured throughput values less than 1000 cycles (Haswell)

Results

Micro-architecture	Method	Error	Spearman Correlation	Pearson Corr.
Ivy Bridge	llvm-mca	0.181	0.902	0.777
	Ithemal	**0.089**	**0.955**	**0.913**
Haswell	llvm-mca	0.200	0.890	0.790
	IACA	0.209	0.917	0.833
	Ithemal	**0.089**	**0.960**	**0.918**
Skylake	llvm-mca	0.239	0.852	0.729
	IACA	0.167	0.926	0.835
	Ithemal	**0.079**	**0.960**	**0.895**

Table 3. Average error for different models and microarchitectures

Method	Throughput (Instructions / second)
llvm-mca	492
IACA	541
Ithemal	560
Empirical execution	13

Table 4. Estimation throughputs for different estimators measured in instructions per second

Strengths

- Ithemal provides state-of-the-art prediction performance
- Its results beat the baselines across the board
- Able to make prediction without knowing the underlying microarchitecture
  - Process is **automated**
  - Reduces **time** and **manpower**
  - Reduces **errors**
  - Can be applied to **new microarchitectures**

Limitations

- Ithemal does not currently handle UNK tokens (i.e. jump instructions at the end of each basic block)
- Assumptions
  - All memory accesses are assumed to be L1 hits
  - Assumes no preemption
- Generalization
  - It’s unclear if the method simply memorizes throughputs
- Can only predict throughput for a single basic block
- Immediates and pointer offsets are mapped to a single token

Thank You!