Facile: Fast, Accurate, and Interpretable Basic-Block Throughput Prediction

Andreas Abel, Saarland University*
Shrey Sharma, Saarland University
Jan Reineke, Saarland University

* now at Google

This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 101020415)
Performance Prediction/Analysis

Hardware

Software
• Full applications
• Functions
• Loops
• Basic blocks

Tool
• Dynamic
  • Profiling/sampling
  • Tracing
• Static
  • Analytical
  • Simulation-based
  • ML-based
• Hybrid

Output
• Execution time
• Throughput
• Resource Usage
• Bottlenecks
• Hotspots
Performance Prediction/Analysis

Hardware  (Intel) x86 systems

Software
- Full applications
- Functions
- Loops
- Basic blocks

Tool
- Dynamic
  - Profiling/sampling
  - Tracing
- Static
  - Analytical
  - Simulation-based
  - ML-based
- Hybrid

Output
- Execution time
- Throughput
- Resource Usage
- Bottlenecks
- Hotspots
x86 Basic-Block Throughput Prediction

Intel® Architecture Code Analyzer (IACA)

uiCA - The uops.info Code Analyzer

Instruction THroughput Estimator using MAchine Learning (ITHEMAL)
Use cases for basic block predictors

Manual performance analysis/optimization

As a cost model for compilers/superoptimizers
State of the art

Skylake

Average Error

Time (ms)

IACA 2.3
OSACA
LLVM-MCA-15
CQA
Ithemal
uiCA

y
Average Error
Time (ms)
x
Background: Pipeline of Intel Core CPUs

Diagram showing the pipeline stages:
- Front End:
  - Predecoeder
  - Instruction Queue (IQ)
  - DSB
  - Instruction Decode Queue (IDQ)
  - Renamer / Allocator
  - Reorder Buffer
  - Scheduler
  - Back End:
    -alu, add
    -alu, mult
    -load, scf
    -load, sfo
    -store, data
    -alu, jmp
Analytical throughput predictor

\[ TP = \max\{\text{FrontEnd, Issue, Ports, Precedence}\} \]
Analytical throughput predictor: Issue

\[ TP = \max\{\text{FrontEnd, Issue, Ports, Precedence}\} \]

\[ \text{Issue} = \frac{n}{i} \]

- in cycles per iteration
- number of uops of the benchmark
- issue width
Analytical throughput predictor: Front end

\[ TP = \max\{\text{FrontEnd, Issue, Ports, Precedence}\} \]

\[
\text{FrontEnd} = \begin{cases} 
  \max\{\text{Predec}, \text{Dec}\} & \text{if benchmark is affected by the JCC erratum} \\
  \text{LSD} & \text{else if LSD is enabled and } \#\text{\muops} \leq \text{IDQWidth} \\
  \text{DSB} & \text{else}
\end{cases}
\]

\[
\text{LSD} = \left[ \frac{n \cdot u}{i} \right] \\
\]

in cycles per iteration

number of uops

unrolling factor

issue width
Analytical throughput predictor: Front end

\[ TP = \max\{\text{FrontEnd}, \text{Issue}, \text{Ports}, \text{Precedence}\} \]

FrontEnd = \begin{cases} 
\max\{\text{Predec}, \text{Dec}\} & \text{if benchmark is affected by the JCC erratum} \\
\text{LSD} & \text{else if LSD is enabled and} \\
\text{DSB} & \text{#uops} \leq \text{IDQ Width} \\
\text{DSB} & \text{else}
\end{cases}

DSB = \begin{cases} 
\left\lfloor \frac{n}{w} \right\rfloor & l < 32, \\
\frac{n}{w} & l \geq 32.
\end{cases}

in cycles per iteration

number of uops

length (in bytes)

DSB width
Analytical throughput predictor: Front end

\[ TP = \max\{\text{FrontEnd}, \text{Issue}, \text{Ports}, \text{Precedence}\} \]

FrontEnd = \begin{cases} 
\max\{\text{Predec}, \text{Dec}\} & \text{if benchmark is affected by the JCC erratum} \\
\text{LSD} & \text{else if LSD is enabled and } \#\muops \leq \text{IDQWidth} \\
\text{DSB} & \text{else} 
\end{cases}
Analytical throughput predictor: Ports

\[ TP = \max\{\text{FrontEnd}, \text{Issue}, \text{Ports}, \text{Precedence}\} \]

Examples:

- \(0|1|5\) ➔ 1/3
- \(0|1\) and \(1|5\) ➔ 2/3

• In general: can be solved using a linear program
• We developed a simpler, more efficient heuristic approach
Analytical throughput predictor: Precedence

\[ TP = \max\{\text{FrontEnd, Issue, Ports, Precedence}\} \]

inp cycles per iteration

\[ \text{loop:} \]

\[ \text{dec rax} \]

\[ \text{dec rax} \]

\[ \text{jnz loop} \]
Implementation
Automatic generation of microbenchmarks to measure the latency, throughput and execution port usage of individual instructions.
Evaluation

Skylake

IACA 2.3 | OSACA | LLVM-MCA-15 | CQA | Ithemal | uiCA | FACILE

Average Error | Time (ms)
Accuracy
Accuracy
Execution time
Summary & Future Work

- Accurate and efficient open-source analytical basic-block TP predictor
- Provides insights into what the bottlenecks are

- Integrate into compilers/superoptimizers
- Combine with branch prediction or memory hierarchy models
- Combine static and dynamic analyses

github.com/andreas-abel/uiCA/blob/master/facile.py