Professorship for Computer Science
Operating Systems and System-Level Programming

Ideas from Synthesis Kernel for Modern Systems

Optimization techniques and SuperOptimizers

December 23, 2022

Madhu Mohan Nelemane

Content

Introduction

- The Synthesis Kernel [1] introduces runtime code generation as one of the key ideas for an efficient operating system.
- Runtime Code generation enables optimization
- [1] proposes common optimization techniques like - constant folding, constant propagation and procedure inlining
- The Kernel and the subsequent paper on Superoptimizer by Henry Massalin tend to achieve the goal of aiding the expert to optimize better.

Synthesis OS Kernel

- Synthesis optimized code for frequently used Kernel routines - queues, buffers, switchers, interrupt handlers, system call dispatchers
- Fine-grained scheduling - Deduce CPU time for each task through measurement and data accumulation
- Tackling the goal of high throughput with low latency
- Optimization through design changes in various functions of the kernel - Diffuse Kernel, I/O, process management, services and interfaces

Optimizations from Synthesis OS

- Constant Folding, Constant Propagation and Procedure Inlining are used
- Data-aware optimizations using information available at compile-time
- TradeOff: Cost saving must exceed code generation cost
- Methods used by Synthesis for code generation:
  - Facoring invariants - partial evaluation for constraints
  - Collapsing Layers - similar to inlining reaching multiple layers
  - Executable Data structures - self-traversing - Jobqueue with Startjob and Stopjob sequences
- Challenges: Size, Protection and Cache Coherence

Traditional Optimization Tricks

Earlier processors had constraints of memory and CPU cycles and every cycle and pipeline stalls mattered.
Imagine running a video decoding application on a 16 KB RAM and 128 Mhz CPU.

1. Replace decision constructs with logical and arithmetic operations
2. Utilize processor's parallel processing capabilities avoiding pipeline stalls
   - Unroll loops to avoid pipeline dependencies
   - Algorithmic changes to parallelize Load/Store with MAC instructions
3. Precompute constants and constant expressions
4. Inline smaller frequently called functions/procedures

The last two ideas were also mentioned in [1]

Some Examples of Code-level Optimizations

```c
int maximum(int a, int b) {
    if (a > b)
        return a;
    else
        return b;
}
```

Can be replaced by:

```c
int maximum(int a, int b) {
    return -(((b - a) >> 31)*a + ((a - b) >> 31)*b);
}
```

Pipeline Stages and Loop Unrolling

This is a sample set of processor pipeline stages of a RISC processor and can be different for other CPUs. Assumption: Each instruction takes 1 cycle unless otherwise mentioned.

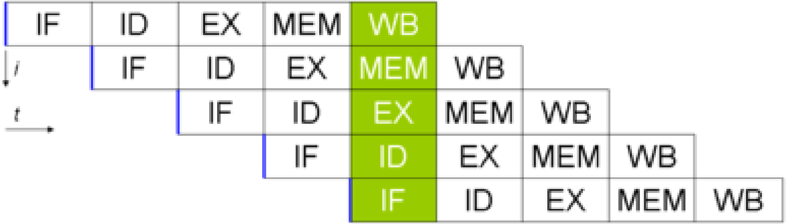


Figure 1: Pipeline Stages RISC from Wikipedia on RISC Pipeline [2]

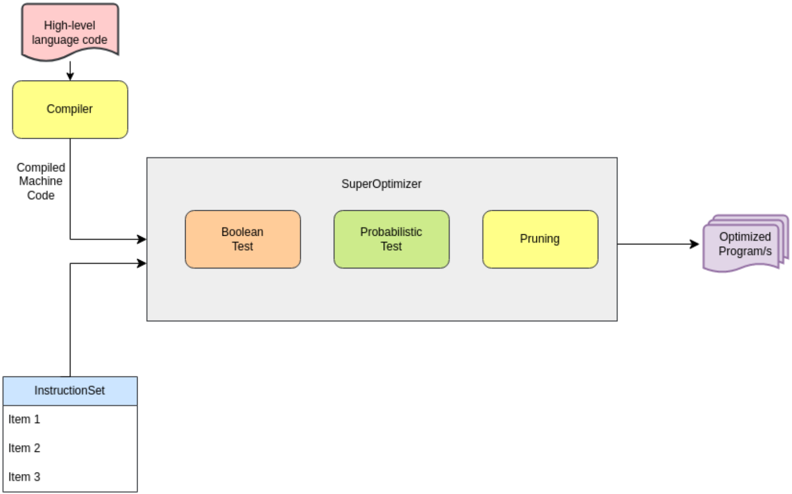
Loop Unrolling Example

```
Loop:
  load R1, (R6)
  load R2, (R6+1)
  mac R1, R2, R3   ..2ticks
  store R3, (R9)
  add R6, 2
  inc R7
EndLoop
```

```
load R1, (R6)
load R2, (R6+1)
add R6, 2
Loop:
  Load R4, (R6)
  mac R1, R2, R3  ..2ticks
  Load R5, (R6+1)

  add R6, 2
  store R3, (R9)

  Load R1,(R6)
  mac R4, R5, R7
  Load R2, (R6+1)

  add R6, 2
  store R7, (R9)
  inc R9
Endloop
```

Superoptimizer Methods

Goal: Given an instruction set, search for the smallest program for a task. Introduced first by Henrz Massalin [3].



Limitations and Applications

- Exponential growth of search time with instruction set (several hours)
- Modeling a pointer is difficult (take whole memory into account)
- Machine-independence
- Effective in register-register operations
- Design of RISC architectures
- For program snippets overlooked by compiler (eg> multiplication by constants)
- Best used to aid assembly language programmers

Automatic Generation of Peephole Superoptimizers

- Peephole optimizers replace sequence of instructions with faster sequence
- Utilizes pattern-matching to decide rules - Automation to use superoptimization
- implemented as a network-based search engine on a database of learned optimizations

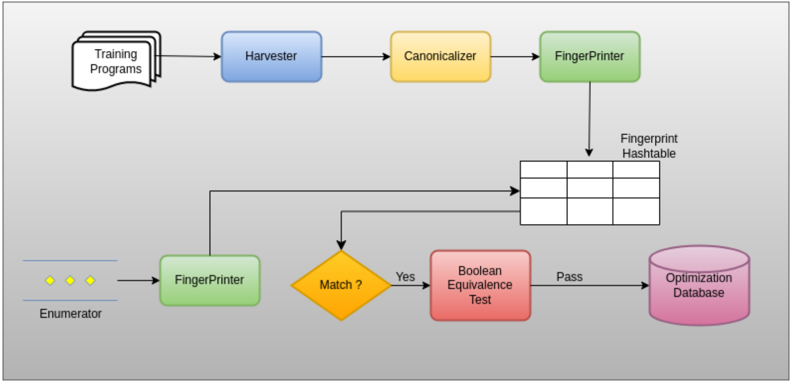


Figure 3: Architecture of automated SuperOptimizer [4]

GNU Superoptimizer

- Gives the shortest instruction sequence
- Available as an installable package
- Supports multiple CPU Architectures
- SPARC, MC68000, MC68020, M88000, POWER, POWERPC, AM29K, I386, I960 (for i960 1.0), I960B (for I960B 1.1), PYR, ALPHA, HPPA, SH

To compile and run the superopt on a GNU Machine

```
> make CPU=-D<cpu_name_from_list> superopt
>
> superopt -f<goal-function> | -all  [-assembly]
[-max-cost n] [-shifts] [-extracts] [-no-carry-insns]
[-extra-cost n]
```

Conclusions

- Optimized programs are always beneficial particularly so in low-level systems
- Ideas in Synthesis are relevant and have been used and improved since
- Superoptimization can provide great performance benefits
- Limitations on size, effort, and time can be countered through automation
- Modern compilers try to incorporate some of the techniques

Future Work:

- Improvements have been made on the superoptimization originally proposed in [3]
- Goal-oriented, automated and efficient superoptimization techniques have evolved over the years
- Standard compilers also have tools to aid superoptimization
- Further research is required into reducing the search space, optimal equivalence test and possible application of machine learning techniques to train superoptimizers can be thought through.

References I

[1] H. Massalin, "Synthesis: An efficient implementation of fundamental operating system services," *Phd. Thesis, Columbia University*, 1992.

[2] Wikipedia, the free encyclopedia, "Atomic force microscopy," 2022, [Online; accessed Dec 22, 2022]. [Online]. Available: https://en.wikipedia.org/wiki/File:Fivestagespipeline.png

[3] H. Massalin, "Superoptimizer - a look at the smallest program," *ACM ASPLOS II*, 1987.

[4] S. Bansal and A. Aiken, "Automatic generation of peephole superoptimizers," *ACM ASPLOS II*, 2006.

[5] It's time for a modern synthesis kernel. [Online]. Available: https://blog.regehr.org/archives/1676

[6] Gnu superoptimizer. [Online]. Available: https://www.gnu.org/software/superopt/

Questions ?

Madhu Mohan Nelemane

sridhara-madhu-mohan.nelemane@stud.uni-bamberg.de