Adaptive Instruction Cache Optimizations

By Maxime Chevalier-Boisvert

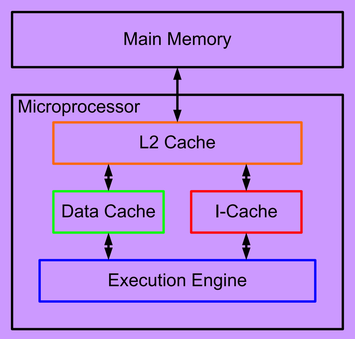
Supervised by Prof. Clark Verbrugge

McGill
COMPUTER SCIENCE

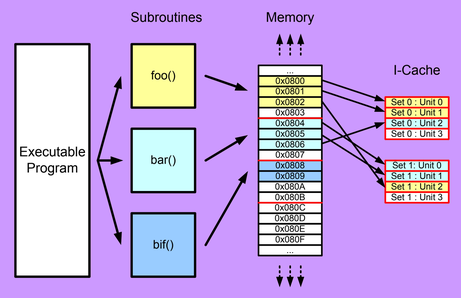
A Strange Fact...

- Minor, normally irrelevant changes to a program's source code can greatly affect its performance…
- Adding useless code can actually make a program up to 15% faster!
- This strange fact is due to the way the instruction cache of modern microprocessors works

What's an Instruction Cache (i-cache)?

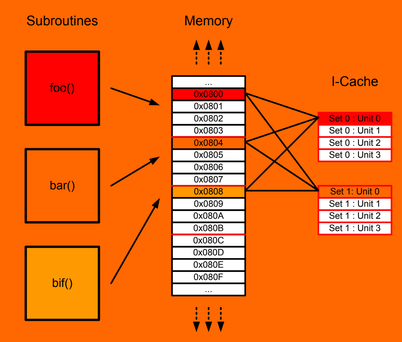


It's a part of every microprocessor



Computer programs (executable instructions) are cached into it

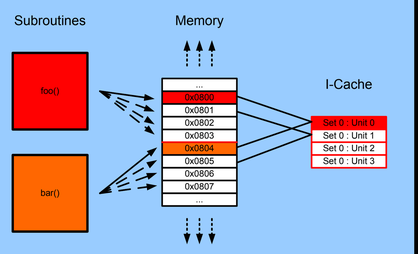
The Problem: I-Cache Conflicts



- Several «hot » subroutines in memory
- Must be stored in cache to be executed
- May technically all fit in cache space
- May not be possible to store them all in cache at once!
- Conflicts cause poorer performance (bad!)

Solving Conflicts

- Code can be moved in memory
- This can solve conflicts
- Can also create new conflicts
- Changing code alignment can affect performance
- Change can be dramatic (+/- 15% possible)



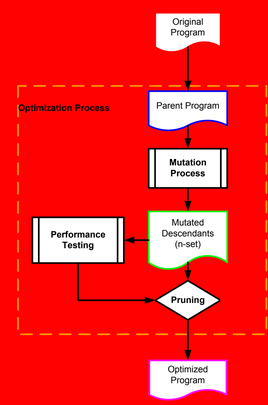
Random Mutations

VS

- In nature, many mutations do not lead to improvements, but some do!
- What if we randomly mutated programs?
- Could generate programs that perform better or worse...

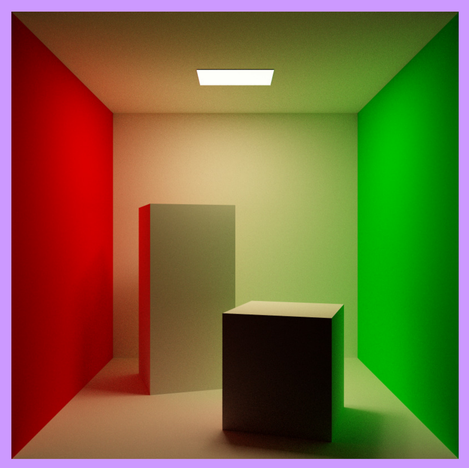
Our Method

1. Generate a large set of mutant versions of the original program
2. Test the performance of each version
3. Select the best performing program
4. Done! We have an optimized program



Test Setting

- Tested our technique on the Radiant 3D Rendering program (made for COMP400 project)
- This program has a complex inner logic and is computationally intensive

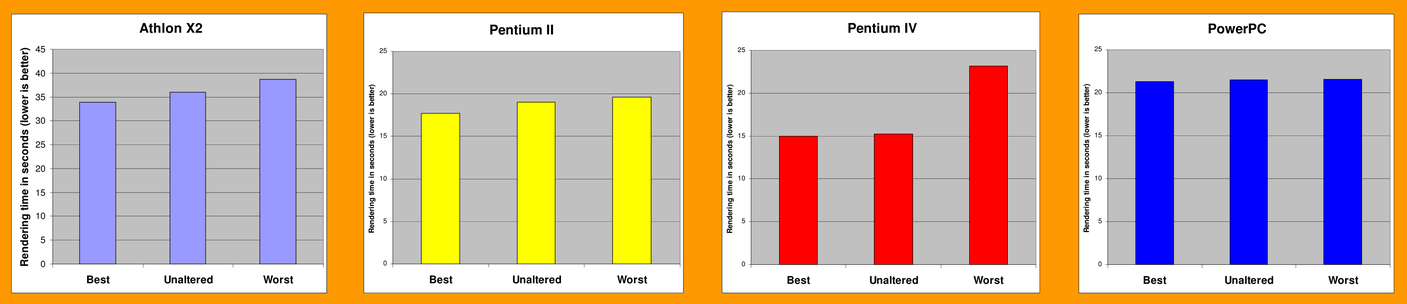


- We tested our algorithm with 50 mutations of the Radiant program
- Benchmarked the final results using SABS (tool from previous year NSERC project)
- Tested best, worst and unaltered variations
- Performed tests on multiple platforms (Debian Linux, Mandrake Linux, Windows 2000)
- Tested multiple microprocessors (Athlon XP, Athlon X2, Pentium II, Pentium IV, PowerPC)
- Tested Multiple compilers (GCC, MSVC)

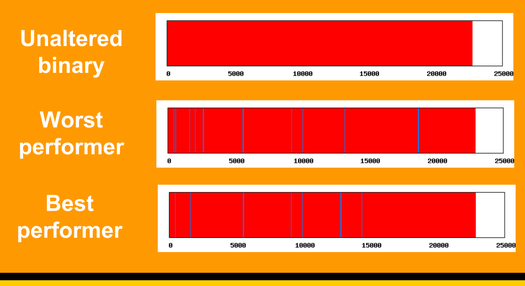
Special Thanks to:
- Prof. Clark Verbrugge
- Prof. Laurie Hendren
- Gregory B. Prokopsky

Sable Research Group
www.sable.mcgill.ca

Results



Blue bars below show inserted code in sample program



- Performance improvements were obtained on all configurations tested
- Most notable improvement was 6.8% on Pentium II processor, with 13% variation between best and worst
- Most important variation on Pentium IV (54%)
- Very small amount of code introduced, but very significant changes
- Measurement error relatively insignificant (< 0.1%)

Future Work

- Algorithm could be improved for faster convergence (potential for machine learning, genetic algorithms)
- Our technique is already applicable as-is
- Can be used for software releases
- Could be used in embedded systems
- Very JIT (Just-In-Time Compiler) friendly

- Could be particularly useful for server software, or specialized, computationally intensive, scientific applications