Optimizing Compilers for Modern Architectures

Chen Ding

CS255/455 Advanced Programming Systems
Spring 2014

Chapter 1, Optimizing Compilers for Modern Architectures, Allen and Kennedy

Early History of Computer Architecture

- 1963 IBM 7094, 1 MIPS
- 1964 CDC 6600, 9 Mflops
- 1968
  - CDC 7600, 40 Mflops
  - Intel founded
  - Rand proposed ARPA net
- 1971 IBM 360/195, cache memory
- 1975 Cray I, 133 Mflops
- 1992 DEC Alpha
- Techniques used
  - pipelining, lookahead, SIMD
  - multi-bank memory, cache

1K x 1K Matrix Multiply

- Machine and compiler (tested around 2003)
  - SGI, MIPS R12K 250MHz, MIPSpro compiler
  - Intel, Intel Pentium4 2GHz, GCC compiler
  - IBM, Power4 1GHz, Xlc compiler
  - Sun, Ultra5 360MHz, Sun compiler

	Intel 2GHz	IBM 1GHz	Sun 360MHz	SGI 250MHz
no opt				
scalar opt				
loop opt				

Case study

- Matrix multiply
  - C(J,I)=C(J,I)+A(J,K)*B(K,I)
- For pieplines
  - unroll J loop by 4
- For a vector machine
  - unroll J loop by 64
- On Sun Starfire
  - parallel do for I loop
- On Intel Paragon
  - parallelize J loop
- On Cray T90, 32 vector processors
  - vectorize J and parallelize I

Lessons learned

- Explicit representation of parallelism in a source language is not sufficient. Whenever the program is tailored to a specific architecture, it loses efficiency when ported from one machine to another.
- All parallel forms can be derived from initial sequential source by relatively simple transformations
- Increasing lifetime of software and decreasing lifetime of hardware, program optimization should be best left to the compiler.

Summary

- Uniprocessor performance
  - key challenge
    - sustaining the instruction and data supply
  - hardware solution
    - pipelining, lookahead, vector, VLIW, GPU
    - memory hierarchy
  - software challenge
    - automatic optimization
- Synchronous parallelism
  - e.g. A[1:n] = A[1:n] + B[1:n]
    - read all values before computing on them
- Automatic vectorization next week (the Allen-Kennedy algorithm)