CS 370: Computer Organization

Computer Systems: Abstraction vs. Reality

Instructor:
Professor Stephen P. Carl

Quote of the Day

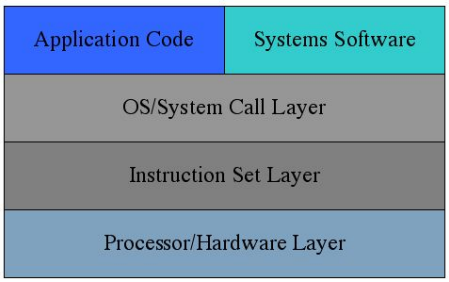
Do one thing at a time, and do it well. An interface should capture the minimum essentials of an abstraction. Don't generalize; generalizations are generally wrong.

– Butler Lampson
"Hints for Computer System Design"

Computing Systems as Abstractions

- **Most CS courses emphasize abstraction**
  - Abstract data types (CS 157/257)
  - Asymptotic analysis (CS 320)
- **Computer Systems are organized according to *layers of abstraction*.**
  - In general, abstraction helps engineers of all sorts to *manage complexity*
  - Abstraction helps insulate programmers from differences between various hardware platforms

Layers of Abstraction



CSci 370 Theme: Abstraction Is Important, *But Don't Forget Reality*

- **Abstractions have limits**
  - Especially in the presence of bugs
  - Understanding details of the underlying implementations: sometimes, you just need to
- **Useful outcomes**
  - Become a more effective programmer!
    - Find and eliminate bugs more efficiently
    - Tune systems for better performance
  - Preparation for later "systems" classes in CS
    - Operating Systems, Networking

Reality: There's more to system performance than asymptotic complexity

- **Constant factors matter too!**
- **Even an exact operation count does not predict performance**
  - Can easily see a 10:1 performance range depending on how code is written
  - Must optimize at multiple levels: algorithm, data representations, procedures, and loops
- **We must understand system to optimize performance**
  - How are programs compiled and executed?
  - How do we measure program performance and identify bottlenecks?
  - How do we improve performance without destroying code modularity and generality?

Memory System Performance Example

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  // access in row-major order
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  // access in column-major order
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

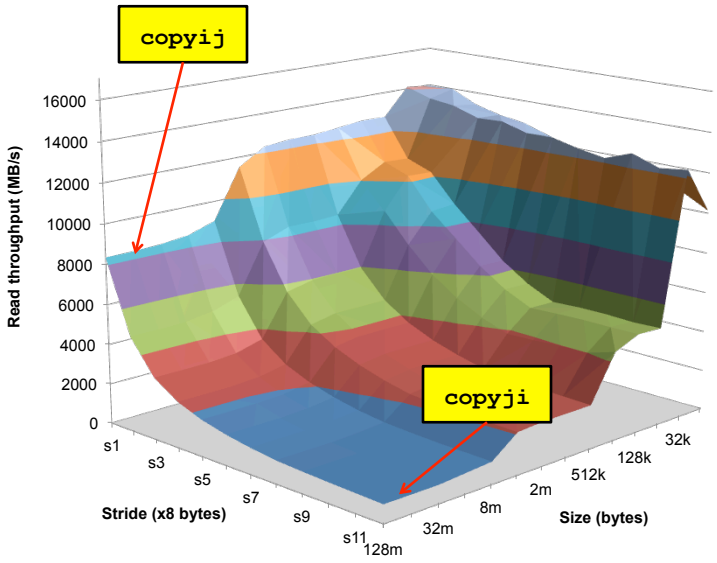
4.3ms **81.8ms**

(as measured on 2.0 GHz Intel Core i7)

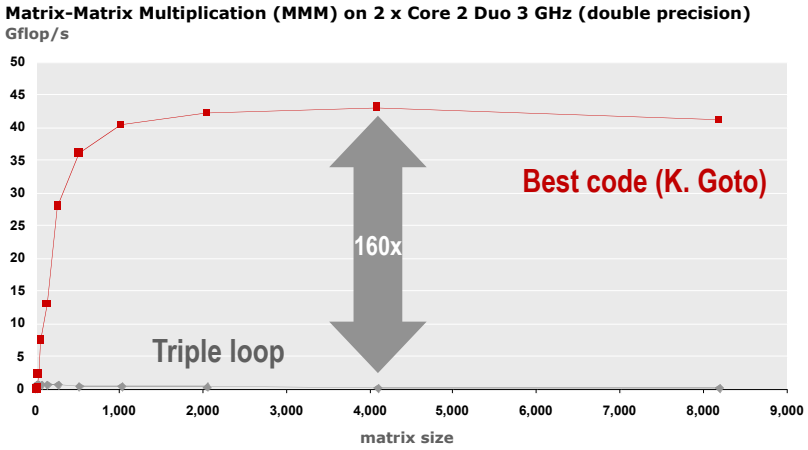
Performance depends on memory access patterns

- How code steps through multi-dimensional arrays

Why The Performance Differs

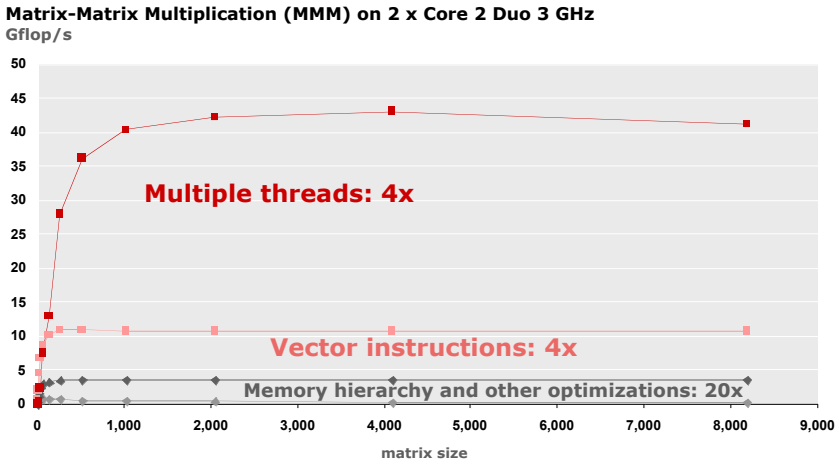


Another Example: Matrix Multiplication



- Standard desktop computer, typical compiler, using compiler optimizations
- Both implementations have exactly the same operations count $T_c = 2n^3$
- *What is going on?*

MMM Performance Analysis



- Each speedup is due to taking advantage of increasingly complex system resources
- *Effect: less register spills, less L1/L2 cache misses, less TLB misses*

Reality: Computers do more than just execute programs

- **They need to get data in and out**
  I/O system is critical to program reliability and performance
- **They communicate with each other over networks**
  Many system-level issues arise in presence of the network
  - Concurrent operations by autonomous processes
  - Coping with unreliable media
  - Cross platform compatibility
  - Complex performance issues

Acknowledgements

- **Slides accompanying textbook due to Bryant and O'Hallaron**
- **Technology Trends slides due to Dr. Andy Carle of UC/Berkeley**
- **Some figures from Hennessey and Patterson: Computer Organization and Design, 4th Ed.**