Introduction to Scientific Computing, Part I

C. David Sherrill

School of Chemistry and Biochemistry

Georgia Institute of Technology

Outline

- Requirements of scientific computing
- Some definitions
- Computer architectures
- Benchmarking
- Benchmarks for Quantum Chemistry

21st Century Computing:

- **Very** complex programs (100's or 1000's of developers)
- Graphical — ease of use critical
- Not much math

Examples: Graphical operating systems (Windows); word processors; spreadsheets; large databases; graphics programs (Photoshop); Web browsers; games

Scientific Computing:

- Complex programs (10^6 lines, perhaps 1-30 developers)
- No time to develop graphical interface
- Much math — floating point very important! "Computationally expensive."

Needs of scientific computing can be vastly different than a user-friendly graphical program. Java makes great applets but is horribly slow for computations.

Some definitions

Bandwidth: The rate at which data can flow.

Bus: A physical data pathway connecting, e.g., the CPU to a graphics card or other device.

Cache: Small storage area for frequently accessed data which provides faster data access.

CPU: Central Processing Unit, or "processor," this is the brain of the computer and does most computational work.

Disk: A magnetic storage device, typically a hard disk drive, used to store data which won't fit in memory. Much slower access than memory.

Floating point: Data representing a real number, or operations (such as multiplication) on such data. Longer/costlier than integers.

Instruction: An elementary, low-level command that the CPU understands. Each CPU has an "instruction set" that it can interpret.

Integer: Data representing an integer, or operations (such as multiplication) on such data.

Kernel: The core of the operating system. Linux is actually an OS kernel; the support software comes from the GNU project (MIT).

Memory: Typically refers to random access memory (RAM) (or "physical

memory"). "Virtual memory" simulates additional RAM by using disk space.

OS: Operating system. The kernel plus essential support software (e.g., file utility programs, system management tools, graphical user interface).

Register: One of the handful of memory locations on the CPU itself.

Swap space: (Also *paging space*) Disk space used to store data which won't fit in physical memory, i.e., virtual memory.

Virtual memory: Maps logical memory addresses to physical memory addresses. Program memory is divided into *pages*, some of which may actually reside on disk.

Word: A fixed number of bytes appropriate for a given data type. Often 4 bytes (32 bits) for an integer and 8 bytes (64 bits) for a floating point number.

RISC vs CISC

CISC (Complex Instruction Set Computers): Older machines, many current and most older PC's. Advantages: program requires fewer instructions (fits in less memory, requires fewer memory fetches). Disadvantages: compilers can't figure out how to take advantage of complex instructions (including Pentium MMX and SSE instructions!).

RISC (Reduced Instruction Set Computers): Modern workstations (e.g., IBM, Compaq) beginning in mid 1980's. PC's moving this way since 1990's (IBM/Motorola/Apple PowerPC). More memory now, faster access due to caches and pipelines — advantages of CISC not as great. Easier to pipeline due to smaller set of instructions.

Pipelines

Instructions typically take more than one clock cycle to execute. In pipelining, after launching one instruction, you immediately launch another on the next clock tick, without waiting for the first one to complete. Possible for CISC, easier for RISC (uniform instruction length, simple addressing modes).

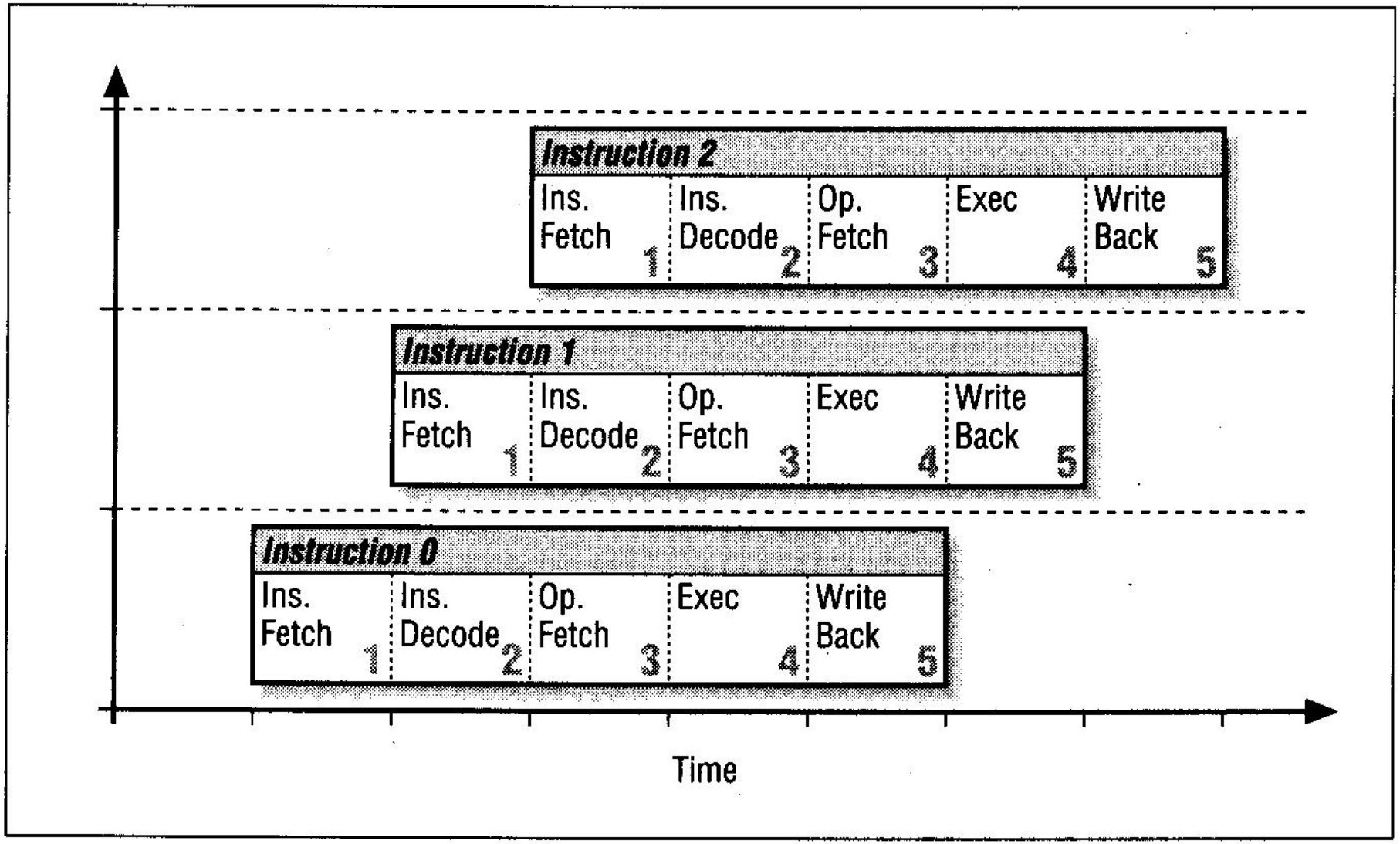

Figure 2-3: Three instructions in flight through one pipeline

Parallel RISC

Superscalar: Multiple pipelines (e.g., floating point and integer pipelines). Depend on compiler to give good mix of instructions and branch processor to oversee their scheduling. Examples: IBM RS/6000, DEC Alpha, even Intel Pentium.

Superpipeline: Break up stages of instruction into smaller, simpler, faster stages, making pipeline go faster.

Long Instruction Word: Like superscalar but depend entirely on compiler to make the instruction stream parallel; puts RISC floating point and integer operations together into big instruction.

Memory and Caches

Advances in memory technology have not kept up with advances in processor speed. Hence, accessing memory substantially delays computations. Caches store frequently accessed data in a small, expensive, high-speed memory area. When we fetch a new memory element, that element *and the memory around it* are transferred to a *cache line* of the cache.

Direct Mapped Cache: If the computer has a 32K cache, then in the direct mapped scheme, memory location 0 is mapped to cache location 0, as are memory locations 64K, 96K, 128K, etc. Problem: if we need data elements more than 32K apart, we'll never find the next item in the cache! (*Cache miss; cache thrashing*).

Fully Associative Cache: Each cache line can map to any memory location. Performs well, very expensive.

Set Associative Cache: Two or four (or more) direct mapped caches side by side; less likely to miss if we hop back and forth between two areas in memory.

```
   REAL*4 A(1024), B(1024)
   DO 10 I=1,1024
     A(I) = A(I) * B(I)
10 CONTINUE
   END
```

Memory Pages

In the *virtual memory* scheme, memory is divided into *pages*. Each program addresses its memory as a block from 0 to N, even though this may be distributed nonsequentially in physical memory. A *page table* translates virtual memory locations to physical memory locations. Simple for the program, bad for performance.

A *translation lookaside buffer* (TLB) is a special cache for page tables, speeding up virtual to physical translation.

A *page fault* results when a requested memory location is not in cache (*cache miss*) or in the TLB (*TLB miss*) or in the list of valid pages (page invalid or on disk). TLB is refreshed and new page is created or loaded from disk (*swapped*).

Benchmarking

User time: The time spent by the CPU on the user's computation.

System time: The time spent by the system in tasks required by the computation; typically I/O time.

Wall time: The actual time elapsed by a "clock on the wall." This is the most relevant time and the most useful benchmark assuming the machine is not busy with other things.

Standard Benchmarks: e.g., SPEC benchmarks

User Benchmarks: Most relevant

Suggested Reading

"High Performance Computing," Kevin Dowd (O'Reilly, Sebastopol, CA, 1993).

"C++ and C efficiency," David Spuler (Prentice Hall, New York, 1992).