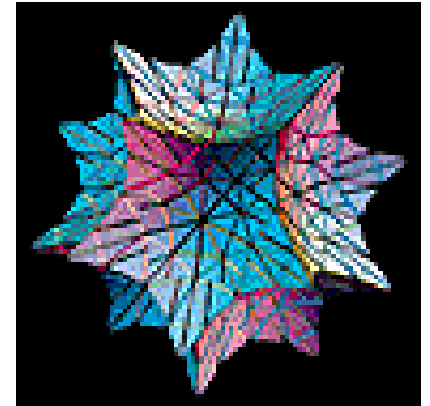


Introduction to Scientific Computing

October 20, 2004



Presented by the

ITC Research Computing Support Group

Kathy Gerber, Ed Hall, Katherine Holcomb, Tim F. Jost Tolson

- Introduction to Parallel Computing— Wednesday, October 27 at 3:30 PM
- Mathematics on the Desktop— Wednesday, November 3 at 3:30 PM
- Optimization— Wednesday, November 10 at 3:30 PM

Introduction to Scientific Computing

Katherine Holcomb

ITC Research Computing Support Group

res-consult@virginia.edu

Topics

- Computer Architectures
- Computer Representation of Numbers
- Interpreted Languages
- Compiled Languages
- Program Development
- Using the PBS Queuing System

Computer Architecture in a (Very Small) Nutshell

- The central processing unit (cpu) executes instructions sequentially
- Memory stores the program and its data
- Some kind of I/O subsystem enables programs to go in and results to come out
- Communications channels allow different parts to exchange data
- Functions are synchronized by a clock (usually some kind of solid-state oscillator)

CPU

- The CPU contains some number of registers that store data while it is in use
- Most CPUs execute one instruction per clock cycle (some pipelining cpus can execute multiple instructions per cycle)
- A single program operation generally requires many (tens or hundreds) of clock cycles

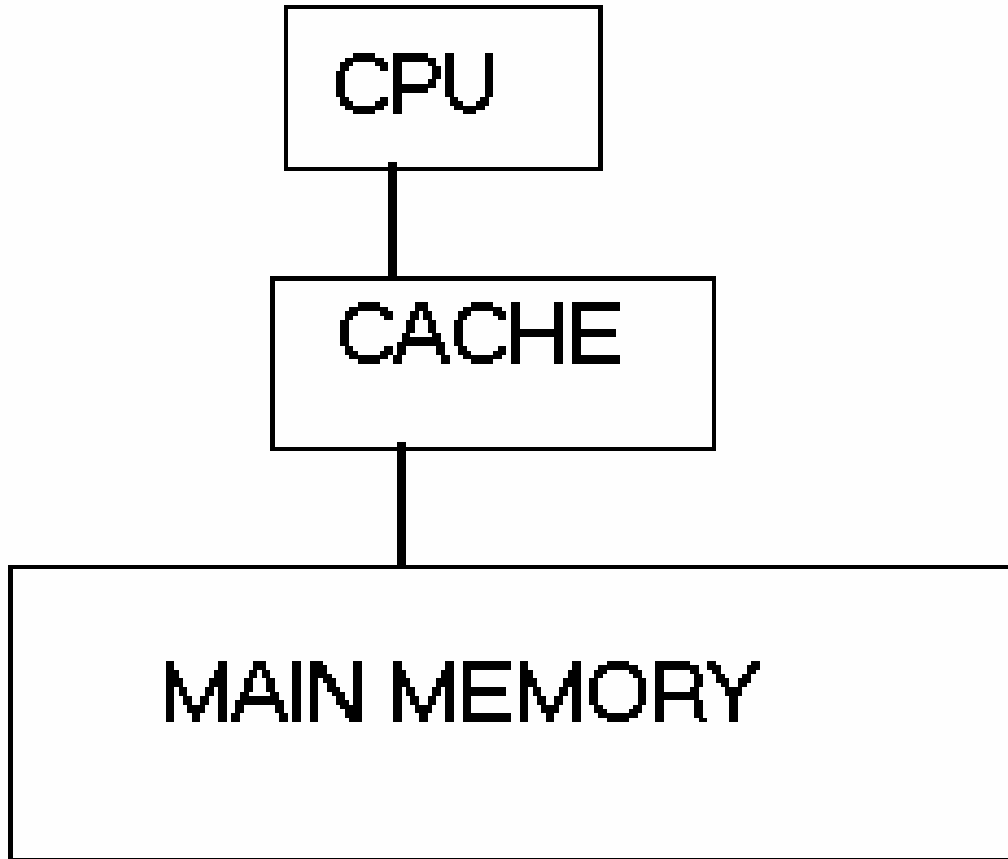
Memory

- Memory stores data in *words* that in nearly all modern architectures consist of either 32 bits (4 bytes) or 64 bits (8 bytes)
- Most is random-access memory (RAM) – meaning that elements need not be accessed in order
- Access time for most modern memory is 10 to 80 ns

Memory Speed

- Memory *latency* is the time required to fetch the first item of a group requested by the cpu
- Memory *bandwidth* is the quantity of data transferred per second
- Most memory is very slow (both in latency and in bandwidth). CPU manufacturers accommodate this by placing a small amount of very (expensive) high-speed memory into a *cache* connected directly to the cpu.

Single Processor System



Cache Optimization

- When the cpu requests data, a chunk is transferred into cache. This is called a cache line
- If data are requested that are not currently in cache, the entire cache line is overwritten. If consecutive program operations result in a new cache line being loaded, this is called a cache miss.

Avoiding Some Cache Misses

- Fortran

- Good

```
do j=1,10
  do i=1,10
    a(i,j) = b(i,j) + c(i,j)
  enddo
enddo
```

Best

```
A = B+C
```

Bad

```
do i=1,10
  do j=1,10
    a(i,j) = b(i,j)
              + c(i,j)
  enddo
enddo
```

Fortran is column-major ordered—this is the natural ordering for functions $f(x,y)$

Avoiding Some Cache Misses (cont.)

- C, C++ etc.

- Good

```
– for i=1;i=10;++i; {  
    for j=1;j=10;++j; {  
        a[i][j]=b[i][j]+c[i][j]  
    }  
}
```

- Bad

```
for j=1;j=10;++j {  
    for i=1;i=10;++i {  
        a[i][j] = b[i][j]+c[i][j]  
    }  
}
```

C and its relatives are row-major ordered—this is the natural ordering for matrices

Computer Representation of Numbers

- Computers represent every number as a finite string of binary digits (0,1). (Humans generally write these numbers in base 16, i.e. hexadecimal format)
- Because of the finite size of each representation, floating-point numbers are *not* the real numbers. Many real numbers map to the *same* floating-point number.

Properties of Reals/Floats

- Real numbers are *dense*: between any two there is another
- Real numbers are *complete*: an arithmetic operation on any two reals always results in another real
- Real numbers are *associative*
- Floats have none of these properties

Single and Double Precision

- In nearly all modern architectures, single-precision floating-point numbers occupy 4 bytes (32 bits)
- Double precision occupies 8 bytes (64 bits)
- With the exception of intermediate registers in some architectures (e.g. 80 bits for Intel), any higher precision, if available, must be done in software and is extremely slow

IEEE Floating-Point Formats

- Single precision: 32 bits consisting of 1 sign bit, 8 exponent bits, 23 bits for the mantissa
 - Expressed in decimals, the range is from about 10^{-38} to 10^{38} with approximately 6-7 significant digits
- Double precision: 64 bits consisting of 1 sign bit, 11 exponent bits, 52 mantissa bits
 - Decimal range is from about 10^{-308} to 10^{308} with approximately 15 significant digits

Roundoff Error

- The finiteness of the set of floats introduces an inevitable **roundoff error** to any numerical calculation
- If the roundoff error remains bounded over the course of the calculation, the algorithm is **stable**. If it grows in an unbounded manner, the algorithm is **unstable**.

Numerical Analysis in Two Slides

- Numerical analysis is the mathematics of solving equations by a finite sequence of arithmetic operations
- Many subfields including linear equations, optimization, and ordinary and partial differential equations

Numerical Analysis (cont.)

- Continuous functions must be replaced by an approximation that can be readily computed, e.g. polynomials, sinusoidals, piecewise polynomials (e.g. splines), etc. This is a *discretization* process.
- Example: Taylor series
 - $f(x+h) = f(x) + f'(x)h + (1/2)f''(x)h^2 + \dots$

Truncation Error

- When a function is discretized, an error called **truncation error** is introduced

- Example: truncate Taylor series after second term:

- $f(x+h) = f(x) + f'(x)h + \dots$

- This would lead to an $O(h^2)$ method

The sum of the dropped terms is the difference between the actual value and the approximate value: this is the truncation error

Truncation errors can mimic physical effects, e.g. viscosity in fluid-dynamics calculations—occasionally this is desirable, usually it is not

Not every method leads to truncation error, but roundoff error is *always* present

HPC Resources from ITC

- Two Linux clusters (Aspen and Birch)
- Other Unix systems: sun-1.unixlab.virginia.edu through sun-8.unixlab.virginia.edu
- Orange (Sun) and Teal (SGI) – to be improved soon
- Licensed software (Matlab, Mathematica, etc.)
- See <http://www.itc.virginia.edu/research/talks/home.html> for further information about HPC resources

Interpreted Programming Languages

- Converted line by line either to machine language or to an intermediate called bytecode by an *interpreter*
- Usually easy to use, good for prototyping, but typically slow to execute
- Cross-platform as long as the interpreter is present
- Examples: Perl, Python, Java (with JVM), many commercial products such as Matlab

Compiled Programming Languages

- Convert source code into machine language (binary) via a program called a compiler
- Often somewhat demanding to learn and use
- Produce highly efficient executables
- Can link external libraries into the executable
- The executable works *only* on the platform on which it was compiled!
(Platform=architecture+operating system)

Program Development

- Compilation
- Makefiles
- Debugging
- Checkpointing

Available Compilers

- ITC Supported Unix Compilers

www.itc.virginia.edu/research/compilers.html

- ITC Supported Linux Compilers

www.itc.virginia.edu/research/pgi/

www.itc.virginia.edu/research/intel/

Compiling Your Program

- Example using PGI Fortran

```
pgf90 -O mycode.f90
```

Produces an executable named a.out

```
pgf90 -g mycode.f90
```

Adds symbols for debugger

```
pgf90 -O -o myexec mycode.f90
```

Executable is named myexec

Compiling with Separate Linking

- `pgf90 -c -O mycode.f90`
- `pgf90 -c -O mysub.f90`
- To link:
`pgf90 -o myexec mycode.o mysub.o`

Using Libraries

- `pgf90 -O -I/usr/local/somecode/inc -c mycode.f90`
- `pgf90 -O -c mysub.f90`

- `pgf90 -L/usr/local/somecode/lib -o myexec
mycode.o mysub.o -lsomelib`
- In this example, the library linked must be of the form `/usr/local/somecode/lib/libsomelib.a`

Makefiles

- Automates compilation of programs
- Shortens compilation by keeping track of what has been changed/needs recompiling
- Simplifies inclusion of multiple compiler flags, e.g. for debugging and optimization

www.itc.virginia.edu/research/make.html

Debugging Software

- Must compile with debug flag (usually `-g`):
disables optimization
- `dbx` or similar is found on most Unix systems
www.itc.virginia.edu/research/debug.html
- `gdb` is the GNU debugger for `gcc`, `g++` and `g77`
www.gnu.org/software/gdb/gdb.html
- TotalView for serial and parallel programs
www.itc.virginia.edu/research/totalview/

Performance

- Once your code is developed, you should optimize it
 - Compiler optimization: use `-O` flag (most compilers). Do not use both `-g` and `-O`
 - Hand optimization: first *profile* the code. Options to use for profiling vary by compiler—check the man page for your compiler
 - Example for Intel compiler: use `-pg` flag for *both* compilation and linking steps
- When run, a profiling log is created which can be examined with programs such as `prof` or `gprof`

Checkpointing

- Prevent losing computation results due to premature program termination resulting from machine crash or cpu time limits.
- Periodically save program state (variables) to a file which could later be read into the program so that computation can proceed from that state.
- Allows monitoring progress of running program
- Especially useful for parallel programs

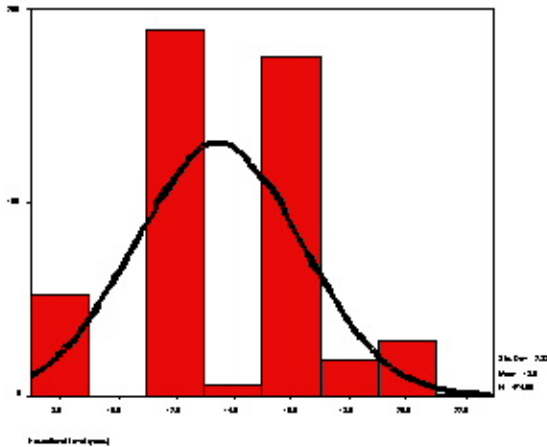
Portable Batch System

- Queueing system – resource manager
- Job scripts are prepared on the frontend and submitted to the queue manager with the command `qsub`
- Refer to Aspen or Birch tutorial for information about using PBS, with many example scripts
 - www.itc.virginia.edu/research/linux-cluster/aspen/tutor.html
 - www.itc.virginia.edu/research/linux-cluster/birch/tutor.html

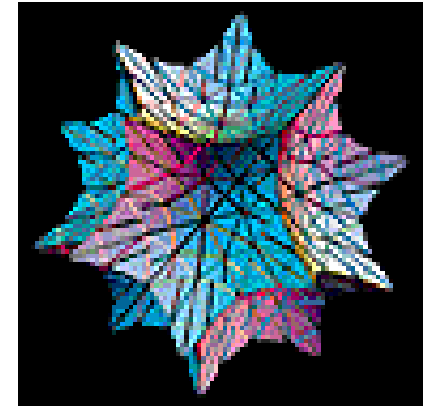
Sample PBS Script

```
#!/bin/sh
#PBS -l nodes=1:ppn=1
#PBS -l walltime=12:00:00
#PBS -o output_filename
#PBS -j oe
#PBS -m bea
#PBS -M userid@virginia.edu

cd $PBS_O_WORKDIR
./myexec args
```



Upcoming Talk



Introduction to Parallel Computing—
Wednesday, October 27 at 3:30 PM

Talks are online at

www.itc.virginia.edu/research/talks