Programming Languages and Compilers

Jeff Nucciarone
AERSP 597B
Sept. 20, 2004
Programming Languages

- Fortran
- C
- C++
- Java
- many others
Why use Standard Programming Languages?

- Programming tedious
  - requiring detailed knowledge of
    - instructions,
    - registers,
    - CPU layout,
    - memory
- Source code was numerical notation
  - octal code
- machine or assembly code
Why use Standard Programming Languages?

- method highly inconvenient
  - non-standard
  - non-portable
  - time consuming
  - difficult to debug
- led to IBM's development of FORTRAN
  - John Backus
Revolution of FORTRAN

- Goals
  - simple to understand
  - easy to learn
- Programmers relieved of burden of using assembly language
- Allowed any scientist or engineer to concentrate on problem at hand
  - no longer needed to be a system expert
Evolution of FORTRAN

- Rapid adoption
- Dialect appeared
  - led to problems in portability
  - required knowledge of extentions
- Gave rise to standards bodies
- 1966 First 'standard' FORTRAN
  - far from perfect, not everyone interpreted or adopted rules the same
  - F77 / F90 / F95/ F2000
Standards Bodies

- **X3J3 - US Fortran Standards Committee**
  - performs technical work of producing the standard (both as a US national and an international standard)

- **ISO/IEC JTC1/SC22/WG5 ("WG5" for short)**
  - coordinates international comment X3J3 work
  - gives general advice on the direction in which the development of the standard should be heading
Standards Questions

- Should language be innovative?
- Small and simple, or big and powerful?
- Will the standard be easy or difficult to implement?
- Should older features be dropped from standard or must all codes work forever?
- Usefulness of language subsets
- Is it what the community needs / wants?
- What does the future hold?
Ad-Hoc and Vendor Standards

- Java
  - Sun Microsystems
- C#
  - Microsoft
- High Performance Fortran
  - Informal standards body
Role of the Compiler

- Early programming
  - tedious
  - highly efficient
- Early FORTAN
  - tradeoff between easy to learn / use and efficiency
- Early compiler
  - great deal of attention
  - small loss in efficiency
Role of the Compiler

- **Translate source code written in a high level language to object code or machine language**
  - write source code in a straightforward manner
  - express intentions clearly
  - allow the compiler to make choices about implementation details that lead to efficient execution
- **Rarely results in executables that are optimal**
Compiler Optimization

- Takes an intermediate representation of source code and replaces it with a better version
- high-level redundancy in the source program (such as an inefficient algorithm) remains unchanged
- Most compilers “jack of all trades”
- typically only deals with a small part of a entire program at a time
  - at most a module at a time
  - usually only a procedure at a time
- the result is compiler is unable to consider at least some important contextual information
Compiler Optimization

- "Post pass" optimizers
  - work at assembly code level

- Hand optimization
  - requires knowledge of processor / memory layout

- Early compilers not as good at optimizing, requiring a lot of hand tuning

- hand tuned code not as portable
  - what runs great here may run poorly elsewhere
Optimization Types

- **peephole**
  - few instructions at a time
  - late in optimization process
- **local**
  - basic block
- **loop optimization**
  - set of basic blocks making up a loop
  - hoist out loop invariant code
Optimization Types

- Intraprocedural
  - acts on control flow graph
    - abstract representation of procedure or program
    - maintained internally by compiler
- Interprocedural
  - optimize interactions between procedures
  - most powerful of all
Optimization Factors

- Number of CPUs
- Memory subsystem layout
  - e.g. number, type and size of caches
- Type and number of control units
  - e.g. FPU, IPU
- Number of processor registers
Optimization Factors

- **Avoid redundancy**
  - If something has already been computed, it's generally better to store it and reuse it later, instead of recomputing it.

- **Less code**
  - There is less work for the CPU, cache, and memory.

- **Straight line code, fewer jumps**
  - Less complicated code. Jumps interfere with the prefetching of instructions, thus slowing down code.
Optimization Factors

- **Code locality**
  - Pieces of code executed close together in time should be placed close together in memory, which increases spatial locality

- **Extract more information from code**
  - The more information the compiler has, the better it can optimize.

- **Use optimal routines where available**
  - More on this later.....

- **Modern compilers have become very good at optimization, little need to hand tune**
Numerical Libraries

- Developing reliable and accurate code takes effort
- Standardized numeric libraries present cost effective way of solving problems
- No need to reinvent wheel
- Reduced burden on programmer to optimize
Benefit of Using Numerical Libraries

- Reduce development time
  - esp. debugging time, *or at least that is the plan*
  - reduction of porting costs
- performance benefits
  - out of the box
- good investment
  - time savings
  - optimization
Supported Languages

- Fortran
  - most all dialects
  - maturity of language and compilers negate need for some older routines
- C
- C++
- Java
- others
BLAS

- Level 1
  - vector-vector operations
- Level 2
  - matrix-vector operations
- Level 3
  - matrix-matrix operations
Who is out there?

- IMSL
- NAG
- Intel MKL
- AMD ACML
- netlib.org
- ATLAS
- GOTO
- many many many more
Specialty libraries

- IBM MASS
  - for AIX systems
  - trade precision for speed
- IBM ESSL
  - for AIX systems
- Sparse Matrix
- FFT (FFTW)
- many many many many more
How to choose

- Features that you need
- Standard and portable?
- How well does it work?
  - does it work at all
  - how optimal
- Cost
  - cost / benefit
How to use

- Check documentation!!!!!!
  - review sample codes
- System / Installation dependent
  - consult with system support staff and / or user guides for specific instructions
- Link time option
  - no need to recompile main code to switch between libraries
Some examples

- Live demo
  - matrix inversion
Beyond Numerics...

- Packages to parallelize applications
- Packages to aid parallelization
- Application packages
  - G03, CPMD, VASP
  - Matlab, Mathematica
Thank You...
More Information

- e-mail
  - beatnic@aset.psu.edu
- Please visit the following web pages
  - http://www.kcl.ac.uk/kis/support/cit/staff/brian/forsaga.html