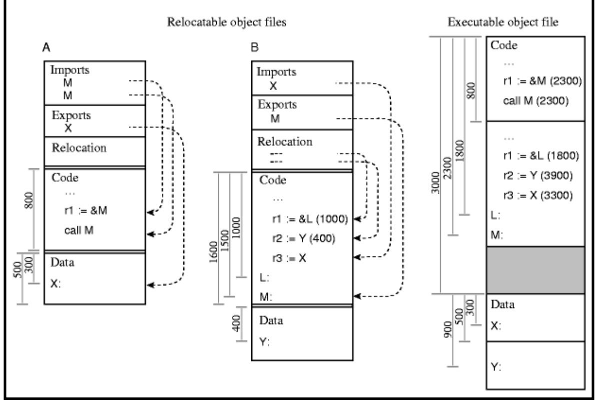
Linking, Parallelism, Wrap-Up

CSE401
Winter 2008

Agenda

- Linking
- Parallelism
- What we did in this course
- Final Exam
- Evals

Static Linking Example



Libraries

- contain lots of code, you don't need all of it
- linkers search the library and only pull in the code that you need.
- libraries are often stored in a special format to make this easier.

Dynamic Linking

Observations:

- Several instances of a program are often live at the same time.
- Programs share code (graphics routines)
- Libraries often improve over time

Dynamic Linking (cont.)

- OS sets up a mapping so that all instances of the same program share the same read-only copy of the code.

Parallel Programming

Why Parallel Programming?

- Predict Weather
- Predict Spread of SARS
- Predict path of hurricanes
- Predict oil slick propagation
- Model growth of bio-plankton/fisheries
- Structural Simulations
- Predict path of forest fires
- Model formation of galaxies


```
do i= 1 to max,
    a[i] = b[i] + c[i] * d[i]
end do
```


Approaches

- Parallel Algorithms
- Parallel Language
- Message passing (low-level)
- Parallelizing compilers

Fortran for parallelism

- **Fortran 90** - Array language. Triplet notation for array sections. Operations and intrinsic functions possible on array sections.
- **High Performance Fortran (HPF)** - Similar to Fortran 90, but includes data layout specifications to help the compiler generate efficient code.

- ZPL - array-based language at UW. Compiles into C code (highly portable).
- C* - C extended for parallelism

Parallelizing Compilers

Automatically transform a sequential program into a parallel program.

1. Identify loops whose iterations can be executed in parallel.
2. Often done in stages.

Q: Which loops can be run in parallel?

Q: How should we distribute the work/data?

Data Dependences

Flow dependence - RAW. Read-After-Write. A "true" dependence. Read a value after it has been written into a variable.

Anti-dependence - WAR. Write-After-Read. Write a new value into a variable after the old value has been read.

Output dependence - WAW. Write-After-Write. Write a new value into a variable and then later on write another value into the same variable.

14

Example

1: A = 90;

2: B = A;

3: C = A + D

4: A = 5;

15

A parallelizing compiler must identify loops that do not have dependences BETWEEN ITERATIONS of the loop.

Example:

```
do I = 1, 1000
   A(I) = B(I) + C(I)
   D(I) = A(I)
end do
```

16

```
Fork one thread for each processor
Each thread executes the loop:
  do I = my_lo, my_hi
         A(I) = B(I) + C(I)
         D(I) = A(I)
   end do
Wait for all threads to finish
  before proceeding.
```

17

Another Example

```
do I = 1, 1000
   A(I) = B(I) + C(I)
   D(I) = A(I+1)
end do
```

18

Yet Another Example

```
do I = 1, 1000
   A( X(I) ) = B(I) + C(I)
   D(I) = A( X(I) )
end do
```

19

Can we improve this?

```
for (i=0; i < 1000, i++){
   for (j=0; j < 1000, j++){
     A[j][i] = B[j][i] + C[j];
  }
}
```

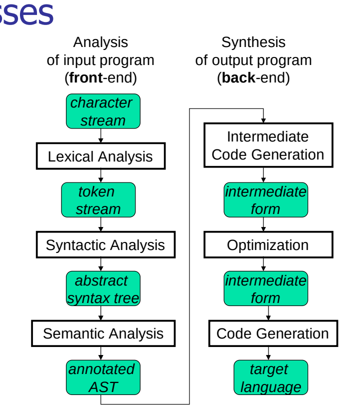
20

Course Project

- Start with a MiniJava complier in Java … improve it
  - Add:
    - Comments
    - Floating-point values
    - Arrays
    - Static (class) variables
    - For loops
    - Break Statements
    - … And more
  - Completed in stages over the term
  - Strongly encouraged: Work in teams, but only if joint work, not divided work

Grading Basis
•Correctness
•Clarity of design/impl
•Quality of test cases

Compiler Passes



First Step: Lexical Analysis

"Scanning", "tokenizing"

Read in characters, clump into tokens

- strip out whitespace & comments in the process

23

Specifying tokens: Regular Expressions

Example:

Ident ::= Letter AlphaNum*
Integer ::= Digit+
AlphaNum ::= Letter | Digit
Letter ::= 'a' | ... | 'z' | 'A' | ... | 'Z'
Digit ::= '0' | ... | '9'

24

Second Step: Syntactic Analysis

"Parsing" -- Read in tokens, turn into a tree based on syntactic structure

- report any errors in syntax

25

Specifying Syntax: Context-free Grammars

EBNF is a popular notation for CFG's

Example:

```
Stmt ::= if (Expr ) Stmt [else Stmt]
    | while (Expr ) Stmt
    | ID = Expr;
    | ...
Expr ::= Expr + Expr | Expr < Expr | ...
    | ! Expr
    | Expr . ID ( [Expr {, Expr}] )
    | ID
    | Integer
    | (Expr)
    | ...
```

EBNF specifies *concrete syntax* of language; parser constructs tree of the *abstract syntax* of the language

26

Third Step: Semantic Analysis

"Name resolution and type checking"

- Given AST:
  - figure out what declaration each name refers to
  - perform type checking and other static consistency checks
- Key data structure: symbol table
  - maps names to info about name derived from declaration
  - tree of symbol tables corresponding to nesting of scopes
- Semantic analysis steps:
  1. Process each scope, top down
  2. Process declarations in each scope into symbol table for scope
  3. Process body of each scope in context of symbol table

27

Fourth Step: Intermediate Code Gen

- Given annotated AST & symbol tables, translate into lower-level intermediate code
- Intermediate code is a separate language
  - Source-language independent
  - Target-machine independent
- Intermediate code is simple and regular
  - Good representation for doing optimizations

Might be a reasonable target language itself, e.g. Java bytecode

28

Fifth Step: Optimization

Identify inefficiencies in intermediate or target code

Replace with equivalent but better sequences

- equivalent => "has the same externally visible behavior"

Target-independent optimizations best done on IL code

Target-dependent optimizations best done on target code

"Optimize" overly optimistic

- Optimize => "usually improve"

Scope of study for optimizations:

- Peephole, local, global (intraprocedural) and interprocedural
- Larger scope => better optimization but more cost and complexity

29

Sixth Step: Target Machine Code Gen

Translate intermediate code into target code

- Need to do:
  - Instruction selection: choose target instructions for (subsequences) of IR instructions
  - Register allocation: allocate IR code variables to registers, spilling to memory when necessary
  - Compute layout of each procedures stack frames and other runtime data structures
  - Emit target code

30

Why Study Compilers?

- Better Understanding Of Implementation Issues in Programming Languages:
  - How Is "This" Implemented?
  - Why Does "This" Run So Slowly?
- Translation appears several places:
  - Processing command line parameters
  - Converting files/programs from one language/format to another

CSE 401: Intro to Compiler Construction

Goals

- Learn principles and practice of language translation
  - Bring together theory and pragmatics of previous classes
  - Understand compile-time vs run-time processing
- Study interactions among
  - Language features
  - Implementation efficiency
  - Compiler complexity
  - Architectural features
- Gain more experience with OO design
- Gain more experience with working in a team
- Gain experience working with SW someone else wrote

Final Exam

- Our final exam will be held **2:30-4:20 p.m. Wednesday, Mar. 19, 2008** in our regular classroom.
- The exam will be comprehensive, but will have a focus on material covered since the midterm.
- EC questions on material from Monday and today**.**
- Ruth will hold office hours on Mon March 17 and Tues 18th, Wed 19th times TBA.
- I will post exam materials on our course web page.