Runtime Support for Algol-Like Languages —

Comp 412

Copyright 2017, Keith D. Cooper & Linda Torczon, all rights reserved.
Students enrolled in Comp 412 at Rice University have explicit permission to make copies of these materials for their personal use.
Faculty from other educational institutions may use these materials for nonprofit educational purposes, provided this copyright notice is preserved.

Chapter 6 in EaC2e
Support for Name Spaces

In an ALL, the compiler needs
• Compile-time mechanism for name resolution
• Runtime mechanism to compute an address from a name

Compiler must emit code that builds & maintains the runtime structures for addressability

In an OOL, the compiler needs
• Compile-time mechanism for name resolution
• Runtime mechanism to compute an address from a name

Compiler must emit code that builds & maintains the runtime structures for addressability

This lecture focuses on runtime support for ALLs.
How Do We Address Variables?

Local variable of current procedure
• Convert to static coordinate
• Add \textit{offset} to $r_{\text{ARP}}$ and load

Local variable in surrounding lexical scope
• Convert to static coordinate
• Convert \textit{level} to ARP for procedure at that level
• Add \textit{offset} to that ARP and load

Local variables of surrounding scopes, static variables at various scopes, and global variables are often called “free variables” in the PL literature
Finding the Correct AR

Each AR has a field for addressability

One common scheme uses access links

• Each AR has a link to the AR of its lexical ancestor (*surrounding scope*)
• Lexical ancestor need not be the procedure that called it
• Compiler can use static coordinate to generate code that chases the chain of access links to find the correct AR
  ✦ Reference to \(<p, 16>\) runs up chain to find level \(p\)'s AR, then adds 16
  ✦ Number of dereferences is \((c − p)\), where \(c\) is the current lexical level
• Cost of access is proportional to lexical distance between \(c\) and \(p\)

\(r_{ARP}\) is a physical register, dedicated to holding the ARP

COMP 412, Fall 2017
Using Access Links

Finding the Correct AR

<table>
<thead>
<tr>
<th>Static Coord</th>
<th>Generated Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;3, 8&gt;</td>
<td>loadAI $r_{ARP}, 8 \Rightarrow r_{10}$</td>
</tr>
<tr>
<td>&lt;2,12&gt;</td>
<td>loadAI $r_{ARP}, 4 \Rightarrow r_{10}$</td>
</tr>
<tr>
<td></td>
<td>loadAI $r_{ARP}, 12 \Rightarrow r_{10}$</td>
</tr>
<tr>
<td>&lt;1,16&gt;</td>
<td>loadAI $r_{ARP}, 4 \Rightarrow r_{10}$</td>
</tr>
<tr>
<td></td>
<td>loadAI $r_{ARP}, 4 \Rightarrow r_{10}$</td>
</tr>
<tr>
<td></td>
<td>loadAI $r_{ARP}, 12 \Rightarrow r_{10}$</td>
</tr>
</tbody>
</table>

procedure fee {
    procedure fie {
        procedure foe {
            call foe()
        }
        call fie()
    }
}
Finding the Correct AR

Maintaining Access Links

At each call:

- In pre-call sequence, caller must find the appropriate ARP

- Assume caller is at level $k$
  - Call at level $k+1$
    - Use current ARP as link
  - Call at level $j$, $0 < j \leq k$
    - Walk up access link chain to level $j-1$
    - Use that ARP as link
  - Call at level $j$, $0 < j \leq k$
    - Use NULL as link
    - Caller & callee share no lexical context

- As long as the code is correct
  - Caller’s chain will be long enough
  - Callee cannot walk off end of chain
Finding the Correct AR

Each AR has a field for addressability

The Linkage Can Also Use A Display

- Global vector of pointers to the ARs of surrounding procedures
- Any nameable AR can be reached in constant time
  - Current AR is reached through ARP
  - Surrounding scopes are reached by treating display as a vector
- Reference to \(<p, 16>\), for \(p < k\), looks up the level \(p \text{ ARP}\) in the display and adds 16 to it
- Cost of access is constant
Finding the Correct AR

procedure fee {
    procedure fie {
        procedure foe {
        }
        call foe()
    }
    call fie()
}
Finding the Correct AR

Maintaining A Display

- Display maintenance is simple, as long as the compiler writer is willing to “waste” one display slot
- In prolog of each level $j$ procedure
  - Save the level $j$ display entry into the procedure’s own AR (addressability slot)
  - Store current ARP into level $j$ display slot
- In epilog of each level $j$ procedure
  - Restore old level $j$ display entry from the current AR’s addressability slot

Simple, clean, and constant time
## Access Links versus Display

### Access Links
- Maintenance adds overhead to each call
  - Overhead depends on nesting
- Adds overhead on each reference to a local in surrounding scope
  - Overhead depends on nesting
- If ARs outlive procedure, access links still work
  - Access links between ARs ensures that they are live (& uncollected)

### Display
- Maintenance adds overhead to each call & return
  - Overhead is constant
- Adds overhead on each reference to a local in surrounding scope
  - Overhead is constant
- If ARs outlive procedure, display links are not preserved
  - Need to preserve them in a way that ensures ARs are live (collector issue)

### Your mileage will vary
- Depends on ratio of non-local accesses to calls
- Depends on demand for registers

*(available register to hold \_display\_ ?)*
Access Links and Displays

The Meta Issue

• Decision to use one or the other is made when the linkage is designed
  – May be beyond the control of the compiler writer

• Lexical levels for procedures and variables are determined in parsing
  – Requires the use of a lexically scoped symbol table

• Code to maintain access link or display is emitted at compile time
  – When compiler generates a procedure or a call, it emits the code
  – **Access links:** Lexical levels are known at compile time, so compiler can generate the code to find a level $j$ display (*knows how far to run up the chain*)
  – **Display:** Lexical levels are known at compile time, so compiler can generate the save (in prolog) and restore (in epilog)

• Either an access link or a display is a runtime object
  – Exists at runtime,
  – Points to ARs, which are runtime objects
Other Runtime Services

We have talked about storage for ARs, for static & global data, for heap allocated data. What other runtime services are needed?

• Input & output subsystems, including files systems, sockets, streams, pipes, **DMA**, network connections, and the like
• Process management, including concurrency
• Interfaces to accelerators, such as **GPUs**, **FPGAs**, other processors
• Heap management, including allocation, deallocation, and collection