Lowering MATLAB by Rewriting:

Arun Chauhan
Indiana University
MATLAB Language

- System vs Language
- “Core MATLAB” vs “Full MATLAB”
- Lowering the core MATLAB language
- Computational core in applications
  - 80-20 rule
An Old Study ...

The diagram compares the time (in seconds) it takes to execute specialized FORTRAN vs MATLAB code on different platforms:

- **Sun SPARC 336MHz**
- **SGI Origin**
- **Apple PowerBook G4 667MHz**

The graph shows that MATLAB 6.x generally takes less time than FORTRAN across all platforms, with the exception of the Apple PowerBook G4, where FORTRAN is slightly faster. The specific times are not clearly visible in the image for each platform, but the trend indicates that MATLAB is faster for the Sun SPARC and SGI Origin, while FORTRAN is faster for the Apple PowerBook G4.
... And a New One

![Graph showing comparison between C and MATLAB for dlaplacian function](image-url)

- **dlaplacian: C vs MATLAB**
- **x-axis**: time (seconds)
- **y-axis**: time (seconds)
- **Comparisons**:
  - Intel Xeon 3GHz
  - Sun SPARCv9 1.2GHz
  - Apple PowerPC G4 1.3GHz

- **Legend**:
  - MATLAB 7.0
  - C

---

Arun Chauhan, Indiana University
The Language

- High-level operations
- Heavily overloaded operators
  - "generic" operations
- Rich array semantics
- Extensive support libraries
Imperative Constructs

- Simple statements
- FOR loop
- WHILE loop
- SWITCH statement
- IF ... ELSEIF ... ELSE
- Function calls (arguments by value)
Semantics

- "Usual" C-like semantics for language constructs
  - true FOR-loop (unlike C)
- Rich semantics for array manipulation
- 2-dimensional array the basic unit of computation
The *end* keyword in subscripts

- $A(\text{end}, \text{end}-1, \text{end}+2) = \ldots$
- $\ldots = A(\text{end}, \text{end}-1, \text{end}+2)$
Context-Dependency

- Context-dependent return values
  - function \([x, y] = \text{foo}() \)
  - \(x + \text{foo}()\)
  - \([x, y] + \text{foo}()\)
  - \([x, y, z] + \text{foo}()\)
Array Subscripting

- $A(x) = Y(p, q) \times Z$
  - $x$: scalar, array
  - $p, q$: scalar, array
  - $Z$: scalar, array
Implications for Type Inf.

- Limited use of backward flow
  - exceptions: max, min, point-wise arithmetic
- Assumption of correctness
- Complicated case analysis
  - easier expressed procedurally
Inferring Types

- Data-flow analysis
- Forward flow of information
- Dynamic inference
  - continuous spectrum of choices between specialization and dynamic analysis
Compiling Type Inference
Compiling $\geq$ Type Inference
Recognizing Combinations

```matlab
function [s, r, j_hist] = min_sr1 (xt, h, m, alpha)
    ...
    while ~ok
        ...
        invsr = change_form_inv (sr0, h, m, low_rp);
        big_f = change_form (xt-invsr, h, m);
        ...
        while iter_s < 3*m
            ...
            invdr0 = change_form_inv (sr0, h, m, low_rp);
            sssdr = change_form (invdr0, h, m);
            ...
            end
        ...
        invsr = change_form_inv (sr0, h, m, low_rp);
        big_f = change_form (xt-invsr, h, m);
        ...
        while iter_r < n1*n2
            ...
            invdr0 = change_form_inv (sr0, h, m, low_rp);
            sssdr = change_form (invdr0, h, m);
            ...
            end
        ...
    end
end
```
for ii = 1:200
    ...
    chan = jakes_mp1(16500, 160, ii, num_paths);
    ...
    for snr = 2:2:20
       ...
       [s,x,ci,h,L,a,y,n0] = newcodesig (NO, l, num_paths, M, snr, chan, sig_pow_paths);
       ...
       [o1,d1,d2,d3,mf,m] = codesdhd (y, a, h, NO, Tm, Bd, M, B, n0);
       ...
    end
    ...
end
Optimizing Patterns

```matlab
init_vals = jakes_mp1_init (16500, 160, num_paths);

for ii = 1:200
    ...
    chan = jakes_mp1_delta (init_vals, ii);
    ...
    for snr = 2:2:20
        ...
        [s,x,ci,h,L,a,y,n0] = newcodesig (NO, l, num_paths, M, snr, chan, sig_pow_paths);
        ...
        [o1,d1,d2,d3,mf,m] = codesdhd (y, a, h, NO, Tm, Bd, M, B, n0);
        ...
    end
...
end
```
Rewriting
Rewriting

- Source-level Transformations
- Stratego rewriting system
Rewriting

- Source-level Transformations
  - Stratego rewriting system
- Capturing Domain Knowledge
  - definition of “knowledge”
  - capturing and utilizing the “knowledge”
Rewriting

- Source-level Transformations
  - Stratego rewriting system
- Capturing Domain Knowledge
  - definition of “knowledge”
  - capturing and utilizing the “knowledge”
- Lowering
Lessons

- Hybrid model of compilation
  - focus on the computational core
  - overlaps with the JIT approach
- Critical importance of libraries
- Careful implementation of array semantics
Challenges

- Identify and develop libraries
- Build domain-specific transformation model(s)
  - annotation language
  - cost model
- Implement!