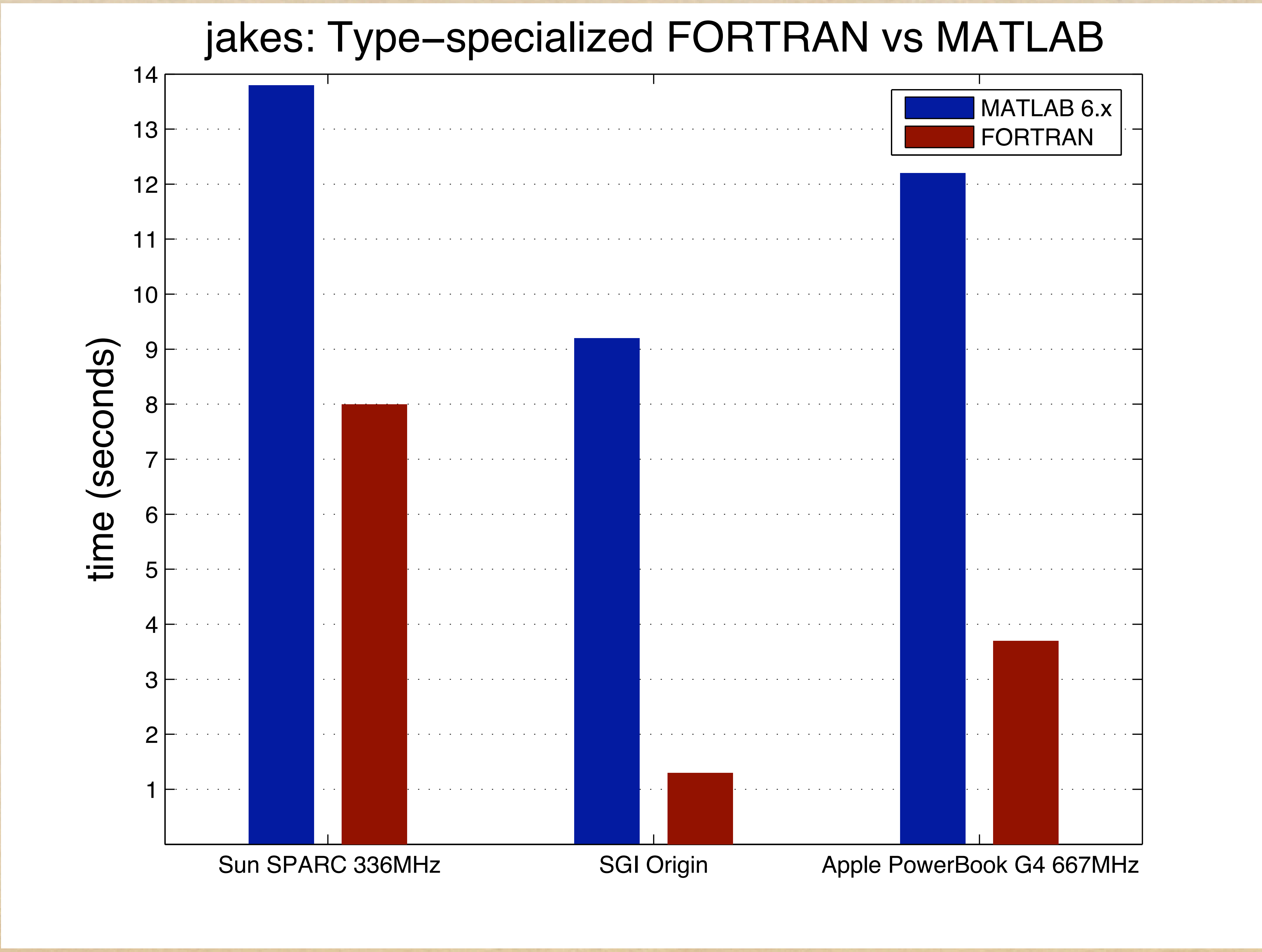
Lowering MATLAB by Rewriting:

Arun Chauhan
Indiana University

MATLAB Language

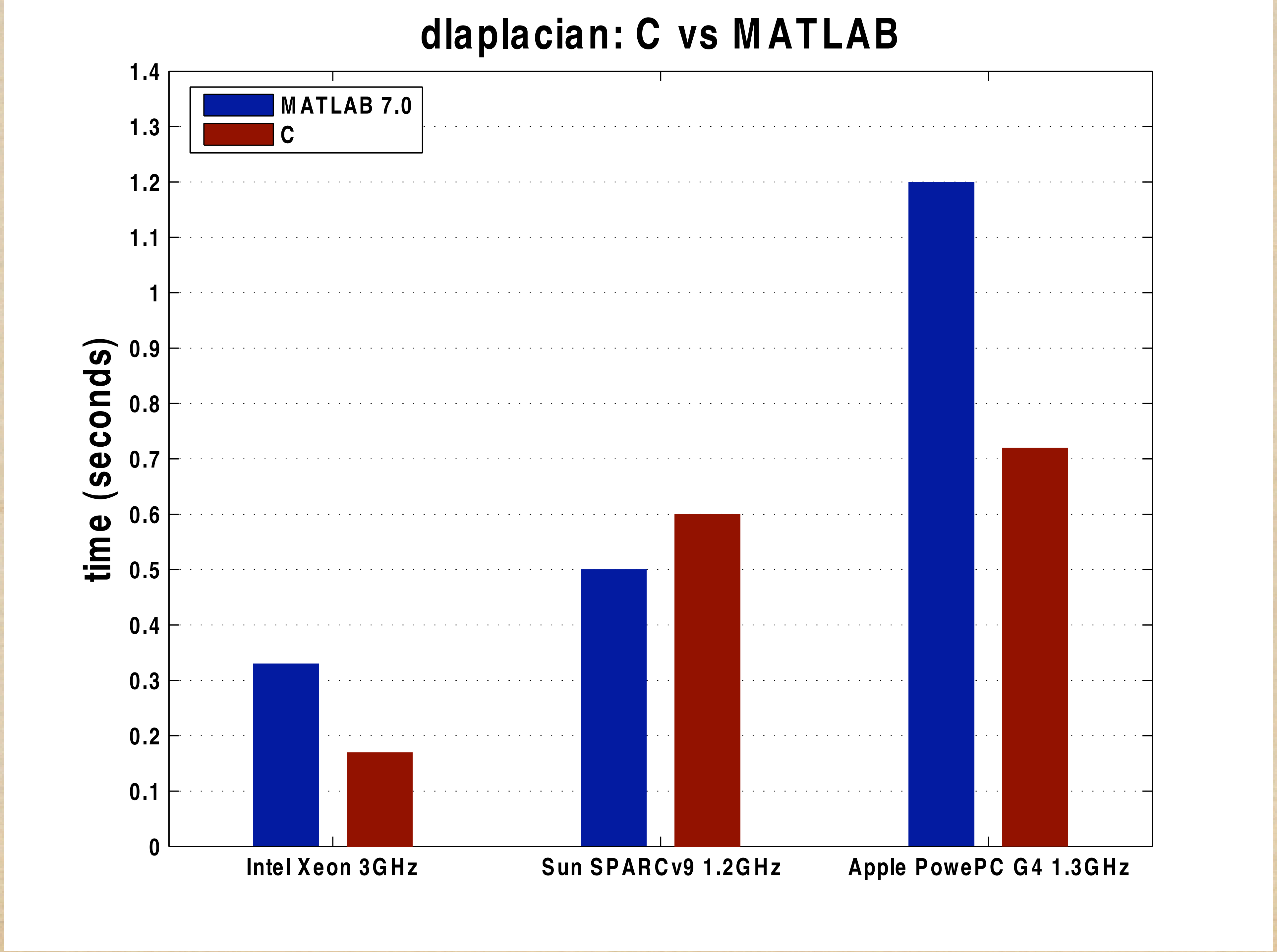
- System vs Language
- "Core MATLAB" vs "Full MATLAB"
- Lowering the core MATLAB language
- Computational core in applications
  - 80-20 rule

An Old Study ...



jakes: Type−specialized FORTRAN vs MATLAB
time (seconds)
14
13
12
11
10
9
8
7
6
5
4
3
2
1
MATLAB 6.x
FORTRAN
Sun SPARC 336MHz
SGI Origin
Apple PowerBook G4 667MHz

… And a New One


dlaplacian: C vs MATLAB
MATLAB 7.0
C
time (seconds)
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Intel Xeon 3GHz
Sun SPARCv9 1.2GHz
Apple PowePC G4 1.3GHz

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

Context-Dependency

- The **end** keyword in subscripts
  - A(end, end-1, end+2) = ...
  - ... = A(end, end-1, end+2)

Context-Dependency

- Context-dependent return values
  - function [x, y] = foo () ...
  - x + foo()
  - [x, y] + foo()
  - [x, y, z] + foo()

Array Subscripting

- $A(x) = Y(p, q) * Z$
  - 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 > Type Inference

Recognizing Combinations

```
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
```

Optimizing Patterns

```
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

```
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!