Telescoping Languages

or High Performance Computing for Dummies

Arun Chauhan

Rice University
Two True Stories
Two True Stories

- the world of Digital Signal Processing
Two True Stories

• the world of Digital Signal Processing
  - almost everyone uses MATLAB
  - a large number uses MATLAB exclusively
  - almost everyone hates writing C code
  - prefer coding for an hour and letting it run for 7 days, than the other way round
  - often forced to rewrite programs in C
Two True Stories

- the world of Digital Signal Processing
  - almost everyone uses MATLAB
  - a large number uses MATLAB exclusively
  - almost everyone hates writing C code
  - prefer coding for an hour and letting it run for 7 days, than the other way round
  - often forced to rewrite programs in C

- linear algebra through MATLAB
Two True Stories

- the world of Digital Signal Processing
  - almost everyone uses MATLAB
  - a large number uses MATLAB exclusively
  - almost everyone hates writing C code
  - prefer coding for an hour and letting it run for 7 days, than the other way round
  - often forced to rewrite programs in C

- linear algebra through MATLAB
  - ARPACK—a linear algebra package to solve eigenvalue problems
  - prototyped in MATLAB
  - painfully hand translated to FORTRAN
Lessons

programming is an unavoidable fact of life to conduct research in science and engineering.

Users do not like programming in traditional languages. Users love domain-specific high-level scripting languages—MATLAB has over 500,000 worldwide licenses, Python, Perl, R, Mathematica.

Performance problems limit their use in the productivity connection.

Rice University: Telescoping Languages

April 7, 2003
Lessons

- programming is an unavoidable fact of life to conduct research in science and engineering
Lessons

- programming is an unavoidable fact of life to conduct research in science and engineering
- users do not like programming in traditional languages
Lessons

• programming is an unavoidable fact of life to conduct research in science and engineering

• users do not like programming in traditional languages

• users love domain-specific high-level scripting languages
  - MATLAB has over 500,000 worldwide licenses
  - Python, Perl, R, Mathematica
Lessons

• programming is an unavoidable fact of life to conduct research in science and engineering

• users do not like programming in traditional languages

• users love domain-specific high-level scripting languages
  - MATLAB has over 500,000 worldwide licenses
  - Python, Perl, R, Mathematica

• performance problems limit their use
Lessons

• programming is an unavoidable fact of life to conduct research in science and engineering

• users do not like programming in traditional languages

• users love domain-specific high-level scripting languages
  - MATLAB has over 500,000 worldwide licenses
  - Python, Perl, R, Mathematica

• performance problems limit their use

• the productivity connection
“It was our belief that if FORTRAN, during its first months, were to translate any reasonable ‘scientific’ source program into an object program only half as fast as its hand-coded counterpart, then acceptance of our system would be in serious danger... I believe that had we failed to produce efficient programs, the widespread use of languages like FORTRAN would have been seriously delayed.” —John Backus
Pushing the Level Again
Pushing the Level Again

effective compilation
Pushing the Level Again

effective compilation

efficient compilation
Fundamental Observation

- libraries are the key in optimizing high-level scripting languages

\[ a = x * y \quad \Rightarrow \quad a = \text{MATMULT}(x, y) \]
Fundamental Observation

- libraries are the key in optimizing high-level scripting languages

\[
a = x \times y \quad \Rightarrow \quad a = \text{MATMULT}(x, y)
\]

- libraries are high-level languages!
  - a large effort in HPC is towards writing libraries
  - domain-specific libraries make scripting languages useful and popular
  - high-level operations are largely “syntactic sugar”
Libraries as Black Boxes
Libraries as Black Boxes

Diagram:
- User program
- Compiler
- Library binaries
- Object code

Rice University: Telescoping Languages
April 7, 2003
Telescoping Languages Approach
Telescoping Languages Approach

- pre-compile libraries to minimize end-user compilation time
Telescoping Languages Approach

- pre-compile libraries to minimize end-user compilation time
- annotate libraries to capture specialized knowledge of library writers
Telescoping Languages Approach

- pre-compile libraries to minimize end-user compilation time
- annotate libraries to capture specialized knowledge of library writers
- generate specialized variants based on interesting contexts
Telescoping Languages Approach

- pre-compile libraries to minimize end-user compilation time
- annotate libraries to capture specialized knowledge of library writers
- generate specialized variants based on interesting contexts
- link appropriate versions into the user script
Telescopming Languages Approach

- pre-compile libraries to minimize end-user compilation time
- annotate libraries to capture specialized knowledge of library writers
- generate specialized variants based on interesting contexts
- link appropriate versions into the user script

analogous to offline indexing by search engines
Telescoping Languages: Entities

- Library writer
- Library compiler
- End user
- Script compiler
Telescoping Languages: Entities

library writer → write libraries → subroutine VMP (C, Z, ..., s) → "expect s to be mostly 1"

library compiler

end user

script compiler
Telescoping Languages: Entities

library writer

write libraries

write annotations

subroutine VMP (C, Z, ..., s)

“expect s to be mostly 1”

library compiler

specialize code

VMP_step1 (C, Z, ...)
Telescoping Languages: Entities

library writer

write libraries

subroutine VMP (C, Z, ... , s)

write annotations

“expect s to be mostly 1”

library compiler

specialize code

VMP_step1 (C, Z, ... )

end user

write script

call VMP (C, Z, ... , 1)

script compiler
Telescoping Languages: Entities

library writer
- write libraries
  - subroutine VMP (C, Z, ..., s)
- write annotations
  - “expect s to be mostly 1”

library compiler
- specialize code
  - VMP_step1 (C, Z, ...)

end user
- write script
  - call VMP (C, Z, ..., 1)

script compiler
- choose optimized variant
  - call VMP_step1 (C, Z, ...)
Telescoping Languages Approach

Domain library

Language building compiler

Script translator

Enhanced language compiler

Optimized object program
Challenges
Challenges

- identifying specialization opportunities
  - which kinds of specializations
  - how many
Challenges

• identifying specialization opportunities
  - which kinds of specializations
  - how many

• identifying high pay-off optimizations
  - must be applicable in telescoping languages context
  - should focus on these first
Challenges

- identifying specialization opportunities
  - which kinds of specializations
  - how many

- identifying high pay-off optimizations
  - must be applicable in telescoping languages context
  - should focus on these first

- enabling the library writer to express these transformations
  - guide the specialization
  - describe equivalences (identities)
Example from ARPACK

```matlab
function [V,H,f] = ArnoldiC (A,k,v);
    n = length(v);
    H = zeros(k);
    V = zeros(n,k);
    v = v/norm(v);
    w = A*v;
    alpha = v'*w;
    ...  
    for j = 2:k,
        beta = norm(f);
        v = f/beta;
        ...  
    end
```
Example from ARPACK

function \[ V, H, f \] = ArnoldiC \( (A, k, v) \);

\[
\begin{align*}
n &= \text{length}(v); \\
H &= \text{zeros}(k); \\
V &= \text{zeros}(n, k); \\
v &= v / \text{norm}(v); \\
w &= A * v; \\
alpha &= v^* w; \\
\ldots \\
\text{for } j = 2: k, & \\
\quad \beta &= \text{norm}(f); \\
\quad v &= f / \beta; \\
\ldots \\
\text{end}
\end{align*}
\]
Inferring Types
Inferring Types

- \textbf{type} \equiv \langle \tau, \delta, \sigma, \psi \rangle
  
  - \tau = \text{intrinsic type}, \text{e.g., int, real, complex, etc.}
  
  - \delta = \text{array dimensionality, 0 for scalars}
  
  - \sigma = \delta\text{-tuple of positive integers}
  
  - \psi = \text{“shape” of an array}
Inferring Types

• type \( \equiv <\tau, \delta, \sigma, \psi> \)
  - \( \tau = \) intrinsic type, e.g., int, real, complex, etc.
  - \( \delta = \) array dimensionality, 0 for scalars
  - \( \sigma = \delta\)-tuple of positive integers
  - \( \psi = \) “shape” of an array

• type inference in general
  - variable type = the “largest” set of values that preserves meaning
Inferring Types

- type ≡ <τ, δ, σ, ψ>
  - τ = intrinsic type, e.g., int, real, complex, etc.
  - δ = array dimensionality, 0 for scalars
  - σ = δ-tuple of positive integers
  - ψ = “shape” of an array

- type inference in general
  - variable type = the “largest” set of values that preserves meaning

- type inference for type-based specialization
  - all valid configurations of types are needed
Formulating the Problem  
(joint work with Cheryl McCosh)

\[
v = v/\text{norm}(v);
\]
\[
w = A^{}v;
\]

- each operation or function call imposes certain “constraints” on the types of its arguments and return values
- the type of a variable is the “smallest” one that meets all the constraints
- incomparable types may give rise to multiple valid configurations of variable types
Solving the Problem

- the problem is hard to solve in general
- efficient solution is possible under certain conditions
- the idea is to reduce it to the clique problem
  - a constraint defines a level
  - clauses in a constraint are nodes at that level
  - an edge whenever two clauses are “compatible”
  - a clique defines a valid type configuration
- some type information must still be inferred dynamically
  - novel technique called slice hoisting
Results: ARPACK

ARPACK: Type-specialized FORTRAN vs MATLAB

Rice University: Telescoping Languages

April 7, 2003
Experimental Evaluation

jakes: Type-specialized FORTRAN vs MATLAB

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

Rice University: Telescoping Languages  
April 7, 2003
Moving Beyond

- study of DSP applications
- library identities play a key role
- identified high-payoff well-known optimization techniques
  - vectorization caused 33 times speedup in one case!
  - others: common subexpression elimination, constant propagation, beating and dragging along
- discovered two novel optimizations
  - procedure strength reduction
  - procedure vectorization
Procedure Strength Reduction

\[ f(c_1, c_2, i, c_3); \]

\[ \text{for } i = 1: N \]

\[ f\text{ iter}(i); \]

\[ \text{end} \]
Procedure Strength Reduction

\[
\text{for } i = 1:N \\
\quad \ldots \\
\quad f (c_1, c_2, i, c_3); \\
\quad \ldots \\
\text{end}
\]
Procedure Strength Reduction

for i = 1:N
    ...
    f (c₁, c₂, i, c₃);
    ...
end

f_init (c₁, c₂, c₃);
for i = 1:N
    ...
    f_iter (i);
    ...
end
Experimental Evaluation

The figure on the left shows the speedups for top-level procedures in Ciss relative to unoptimized original versions. The speedups range from 0 to 3.5, with optimizations for procedures jakes_mp1, newcodesig, codesdhd, and the whole program. The right figure displays the distribution of total execution time among top-level procedures in Ciss, comparing optimized and original versions. The execution time is measured in thousands of seconds, with a range from 0 to 25.
Contributions

- demonstrating the viability of scripting languages for library generation through the telescoping languages approach

- specific technical contributions
  - identification of high-payoff optimizations
  - discovery of two new optimizations
  - development of a novel type-inference algorithm

- telescoping infrastructure
  - MATLAB compiler with C / FORTRAN library generator
Future Directions

• annotation language to describe transformations

• techniques to speculatively optimize code
  - database techniques

• time and space trade-offs in library generation
  - AI techniques

• applying the ideas to automatic parallelization

• dynamically evolving systems like the computation grid

• other domains
  - VLSI design
Past Work

- runtime execution model for irregular parallel applications
- parallelization techniques for high performance multimedia applications
- algorithmic techniques for parallel Cholesky factorization on SMP
- parallelization of weather forecasting application for an SMP
Conclusion

- need to make a move towards high-level languages for high-performance computing
- telescoping languages provide the compiler technology to enable this move
- type-based speculative specialization a primary step
- a novel type-inference algorithm enables this step
- identified high-payoff optimizations
- discovered two new optimizations

http://www.cs.rice.edu/~achauhan/
Bonus Material
Procedure Vectorization
for $i = 1:N$
    $f(c_1, c_2, i, A[i]);$
end

...
for \( i = 1: N \)
\[
\begin{align*}
  f \ (c_1, c_2, i, A[i]);
\end{align*}
\]
end

\[\ldots\]

function \( f \) (a_1, a_2, a_3, a_4)
\[
\text{<body of } f \rangle
\]

function \( f_{\text{vect}} \) (c_1, c_2, [1:N], A)
\[
\text{<body of } f_{\text{vect}} \rangle
\]
for \( i = 1: N \)
end