Telescoping Languages

Domain Specific Languages for the Price of C
(or Fortran)

Arun Chauhan

Indiana University
Collaborators

- Ken Kennedy
- Bradley Broom
- Keith Cooper
- Rob Fowler
- John Garvin
- Chuck Koelbel
- Cheryl McCosh
- John Mellor-Crummey
- Linda Torczon
A True Story

- The world of Digital Signal Processing
  - almost everyone uses MATLAB
  - a large number uses MATLAB exclusively
  - almost everyone hates writing C code
  - readily tradeoff running time for development time
  - often forced to rewrite programs in C
A True Story

• The world of Digital Signal Processing
  - almost everyone uses MATLAB
  - a large number uses MATLAB exclusively
  - almost everyone hates writing C code
  - readily tradeoff running time for development time
  - often forced to rewrite programs in C

• Linear algebra through MATLAB
  - ARPACK prototyped in MATLAB
  - recoded in Fortran for performance
A True Story

- The world of Digital Signal Processing
  - almost everyone uses MATLAB
  - a large number uses MATLAB exclusively
  - almost everyone hates writing C code
  - readily tradeoff running time for development time
  - often forced to rewrite programs in C

- Linear algebra through MATLAB
  - ARPACK prototyped in MATLAB
  - recoded in Fortran for performance

The productivity connection
The Performance Gap

jakes: Type-specialized FORTRAN vs MATLAB

<table>
<thead>
<tr>
<th>Platform</th>
<th>FORTRAN</th>
<th>MATLAB 6.x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun SPARC 336MHz</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>SGI Origin</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Apple PowerBook G4 667MHz</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Telescoping Languages, Arun Chauhan, Indiana University

DSLOpt, 2004-08-19
It’s the Compilers

“We did not regard language design as a difficult problem, merely a simple prelude to the real problem: designing a compiler that could produce efficient programs.”

—John Backus, the “Father of Fortran”
It’s the Compilers

“We did not regard language design as a difficult problem, merely a simple prelude to the real problem: designing a compiler that could produce efficient programs.”

—John Backus, the “Father of Fortran”

Effective and Efficient compilation
The Big Picture

Human-Computer Interface

programming languages

- MATLAB, Python, R, functional
- Java
- C++, C, Fortran
- Machine / assembly

Telescoping Languages, Arun Chauhan, Indiana University
Fundamental Observation

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

\[ a = x \times y \Rightarrow a = \text{mclMtimes}(x, y) \]
Fundamental Observation

- Libraries are the key in optimizing high-level scripting languages
  \[
  a = x \times y \Rightarrow a = \text{mclMtimes}(x, y)
  \]

- Libraries practically **define** high-level scripting languages
  - high-level operations are often “syntactic sugar”
    * runtime libraries implement operations
  - a large effort in HPC is toward writing libraries
  - domain-specific libraries make scripting languages useful and popular
Hierarchy of Libraries

- C, Fortran
  - run-time library
- MATLAB
  - user-level libraries
- ...
- ...
- ...

Telescoping Languages, Arun Chauhan, Indiana University

DSLOpt, 2004-08-19
Libraries as Black Boxes

- Library
- Compiler
- Library Binaries
- User Program
- Compiler
- Object Code

Telescoping Languages, Arun Chauhan, Indiana University
Libraries as Black Boxes

- User program
- Compiler
- Library binaries
- Object code
Whole Program Compilation

- User program
- Library one
  - one
  - one.one
  - one.two
- Library two
- Compiler
- Object code

Telescoping Languages, Arun Chauhan, Indiana University  DSLOpt, 2004-08-19
Motivating Example

- **Specialization**
- **Speculate contexts**
  - utilize library writers’ specialized knowledge

```
procedure VMP (M, V, ..., s)

context

s = 1

s = ?

for i = 1, s, N
{
  ...
}
```

```
VMP_stridel (M, V, ... )

VMP (M, V, ... )
```
Telescoping Languages: Entities

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

library writer

write libraries

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

“expect s to often be 1”

library compiler

write annotations

document

end user

script compiler
Telescoping Languages: Entities

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

“expect s to often be 1”

library writer

write libraries

write annotations

library compiler

specialize code

end user

script compiler
Telescoping Languages: Entities

- **library writer**
  - write libraries
  - write annotations
  - procedure \( VMP \) \( (C, Z, \ldots, s) \)
    - “expect \( s \) to often be 1”

- **library compiler**
  - specialize code
  - \( VMP \) \( (C, Z, \ldots) \)

- **end user**

- **script compiler**
Telescoping Languages: Entities

library writer

write libraries

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

write annotations

"expect s to often be 1"

specialize code

library compiler

VMP_1 (C, Z, ...)

end user

script compiler
Telescoping Languages: Entities

- Library writer
  - Write libraries
- Library compiler
  - Write annotations
  - Specialize code
- End user
- Script compiler
  - Procedure VMP (C, Z, ..., s)
  - Expect s to often be 1
  - VMP (C, Z, ...)
  - VMP (C, Z, ...)
  - VMP (C, Z, ...)
  - VMP (C, Z, ...)

Telescoping Languages, Arun Chauhan, Indiana University
DSLOpt, 2004-08-19
Telescoping Languages: Entities

- **library writer**
  - write libraries
  - write annotations

- **library compiler**
  - specialize code

- **end user**
  - write script

- **script compiler**

**procedure VMP (C, Z, ... , s)**

- "expect s to often be 1"

- **VMP (C, Z, ...)**
  - **VMP.stridel (C, Z, ...)**

- **call VMP (C, Z, ... , 1)**
Telescoping Languages: Entities

**Library Writer**
- write libraries
- write annotations

**Library Compiler**
- specialize code

**End User**
- write script

**Script Compiler**
- choose optimized variant

```plaintext
procedure VMP (C, Z, ..., s)

"expect s to often be 1"

VMP (C, Z, ...)

VMP_stridel (C, Z, ...)

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

call VMP_stridel (C, Z, ...)
```
Overall Telescoping System

library generator

variant DB

KB

script compiler

C

vendor compiler
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
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

analogous to offline indexing by search engines
Library Compiler: Some Issues

- Dealing with high-level scripting languages
  - parsing and analyzing a library procedure written in a scripting language
  - translating into an intermediate language (C, Fortran)

- High-level transformations
  - identifying useful transformations

- Enabling the library writer to express library properties
  - stating facts about library procedures
  - describing specializations
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 = “structure” of an array

• Examples
  
  - x is scalar, integer
    ⇒ type of x = <int, 0, ⊥, ⊥>
  - y is 3-D 10 \times 5 \times 20 dense array of reals
    ⇒ type of y = <real, 3, <10,5,20>, dense>
“It is a capital mistake to theorize before one has data. In-sensibly one begins to twist facts to suit theories, instead of theories to suit facts.”

–Sir Arthur Conan Doyle in a *A Scandal in Bohemia*
Study of DSP Applications

- MATLAB applications from the ECE department
  - real applications being used in the DSP and image processing group
- Looked for high-level transformations
- Discovered
  - two novel procedure-level optimizations
  - relevance of several well known transformation techniques
Procedure Strength Reduction

for i = 1:N
    ...
    f (c_1, c_2, i, c_3);
    ...
end
Procedure Strength Reduction

```plaintext
for i = 1:N
    ...
    f(c_1, c_2, i, c_3);
    ...
end
```

```plaintext
f_init(c_1, c_2, c_3);
for i = 1:N
    ...
    f_iter(i);
    ...
end
```
Speedup by PSR

speedups for top-level procedures in ctss relative to unoptimized

- jakes_mp1: 3
- newcodesig: 1
- codesdhd: 1
- whole program: 1

distribution of the total execution time among top-level procedures in ctss

- Optimized:
  - jakes_mp1: 0.1
  - newcodesig: 0.1
  - codesdhd: 0.1
  - codeshd: 0.1

- Original:
  - jakes_mp1: 2.5
  - newcodesig: 2.5
  - codesdhd: 2.5
  - codeshd: 2.5

Telescoping Languages, Arun Chauhan, Indiana University
Procedure Vectorization

```
for i = 1:N
    α
    f (c₁, c₂, i, A[i]);
    β
end
...
function f (a₁, a₂, a₃, a₄)
    <body of f>
```
Procedure Vectorization

for i = 1:N
    α
    f(c₁, c₂, i, A[i]);
    β
end
...

function f(a₁, a₂, a₃, a₄)
    \textit{body of } f
end

for i = 1:N
    α
end
f vect(c₁, c₂, [1:N], A)
for i = 1:N
    β
end
...

function f vect(a₁, a₂, a₃, a₄)
    \textit{body of } f
end
Applying to jakes

Graph showing speedup (for 100 iterations) normalized for original and optimized versions.
High-payoff Optimizations

- Loop vectorization
- Library identities
- Common subexpression elimination
- Beating and dragging along
- Constant propagation
High-payoff Optimizations

- Loop vectorization
- Library identities
- Common subexpression elimination
- Beating and dragging along
- Constant propagation
High-payoff Optimizations

• Loop vectorization
• Library identities
• Common subexpression elimination
• Beating and dragging along
• Constant propagation
function z = jakes mpl (blength, speed, bnumber, N_Paths)
    ....
    for k = 1: N_Paths
        ....
        xc = sqrt(2) * cos(omega * t_step * j') ... 
            + 2 * sum(cos(pi * np / Num_osc) .* cos(omega * cos(2 * pi * np / N) * t_step .* jp));
        xs = 2 * sum(sin(pi * np / Num_osc) .* cos(omega * cos(2 * pi * np / N) * t_step .* jp));
        
        % for j = 1 : Num
        %     xc(j) = sqrt(2) * cos (omega * t_step * j);
        %     xs(j) = 0;
        % for n = 1 : Num_osc
        %     cosine = cos(omega * cos(2 * pi * n / N) * t_step * j);
        %     xc(j) = xc(j) + 2 * cos(pi * n / Num_osc) * cosine;
        %     xs(j) = xs(j) + 2 * sin(pi * n / Num_osc) * cosine;
        % end
        % end
        ....
    end
end
High-payoff Optimizations

- Loop vectorization
- Library identities
- Common subexpression elimination
- Beating and dragging along
- Constant propagation
function [s, r, j_hist] = min_srl (xt, h, m, alpha)
  ...
  while ~ok
    ...
    invsr = \textit{change\_form\_inv} (sr0, h, m, low\_rp);
    big.f = \textit{change\_form} (xt-invsr, h, m);
    ...
    while iter.s < 3*m
      ...
      invdr0 = \textit{change\_form\_inv} (sr0, h, m, low\_rp);
      sssdr = \textit{change\_form} (invdr0, h, m);
      ...
    end
    ...
    invsr = \textit{change\_form\_inv} (sr0, h, m, low\_rp);
    big.f = \textit{change\_form} (xt-invsr, h, m);
    ...
    while iter.r < n1*n2
      ...
      invdr0 = \textit{change\_form\_inv} (sr0, h, m, low\_rp);
      sssdr = \textit{change\_form} (invdr0, h, m);
      ...
    end
  end
end
XML-based Language

- Enables library writers to express transformations of interest
- Can specify type-based specializations
- Powerful enough to specify library identities
XML-based Language

- Enables library writers to express transformations of interest
- Can specify type-based specializations
- Powerful enough to specify library identities
- Serves as a driver for the source-level optimization phase
Example: Type-based Specialization

```xml
<specialization>
  <context>
    <type var="x" dims="0"/>
    <type var="y" dims="0"/>
  </context>
  <match>
    <simpleStmt>
      <function> generic_ADD </function>
      <input> <var>x</var> <var>y</var> </input>
      <output> <var>z</var> </output>
    </simpleStmt>
  </match>
  <substitute>
    <simpleStmt>
      <function> scalar_ADD </function>
      <input> <var>x</var> <var>y</var> </input>
      <output> <var>z</var> </output>
    </simpleStmt>
  </substitute>
</specialization>
```
Meanwhile, Elsewhere . . .

- **Compiling MATLAB**
  - FALCON, MaJIC (UIUC, Cornell)
  - MATCH (Northwestern)

- **Parallelizing MATLAB**
  - CONLAB (Sweden), Otter (Oregon State), MENHIR (Irisa), . . .
  - *P (MIT)

- **Annotations**
  - Broadway (UT Austin)

- **High-level programming systems**
  - POOMA, ROSE (LLNL)

- **Automatic library generation**
  - ATLAS (UTK), FFTW (MIT)
Concluding Remarks

• Need to raise the level of interface with computers
  - scripting languages raise the level of programming interface

• Scripting languages provide higher abstraction in programming languages but incur performance penalties

• Libraries need to be at the core of a compilation strategy for scripting languages
  - speculative specialization
  - incorporating expert knowledge of library writers

• Experience with MATLAB indicates that a library-centered approach pays off
Future Directions

• Parallel computation
  - speculation or specification of data distribution?
  - library identities

• Dynamically evolving systems (such as the computation grid)
  - speculatively specializing on possible scenarios
  - dynamically switching library versions
  - pre-building schedules
  - self-learning systems through feedback

• Library compilation ideas in other domains
  - VLSI design
  - component-based systems
Other Possible Directions

- Developing annotation language
- Refining techniques to speculatively optimize code
  - database techniques
- Time and space trade-offs in library generation
  - machine learning techniques
- Diversifying the source language systems
  - R, Python, Perl, etc.
- Self-learning systems
  - extracting general contexts from examples
  - incorporating feedback through maintenance-mode runs
http://www.cs.indiana.edu/~achauhan/