Procedure Strength Reduction: An Optimizing Strategy for Telescoping Languages

Arun Chauhan

and

Ken Kennedy
Motivation

• High Performance programming is hard
  – Increasingly a specialized activity
  – Shortage of programmers

• Enable end-users to program
  – Language should be high level
  – Should provide domain-specific features
  – Must have effective and efficient compilers
Current Scenario

• Object Oriented Languages
  – Targeted towards professionals
  – Still not sufficiently high-level for end-users

• Functional Programming Languages
  – Suffer from performance problems

• Scripting Languages (e.g., Matlab)
  – Preferred and used by end-users
  – Have domain specific libraries
  – But, no fast and effective compilers
Key Problems

• Libraries treated as black boxes
  – no library source code
Key Problems

- Libraries treated as black boxes
  - no library source code
- Translation to conventional languages
  - potentially very high compilation times
Key Problems

• Libraries treated as black boxes
  – no library source code

• Translation to conventional languages
  – potentially very high compilation times

• Expert knowledge on libraries discounted
  – potential optimization opportunities lost
Telescoping Languages Approach

Domain Library → Language-Building Compiler → Script Translator → Enhanced Language Compiler → Optimized Object Program
Compiling Telescopin Languages
Compiling Telescoping Languages

Compilation Framework

Library Compilation

Jump Functions

Identities

Specialization

Script Compilation
Compiling Telescoping Languages

Compilation Framework

Library Compilation
- Jump Functions
- Identities
- Specialization

Script Compilation
- Property Propagation
- Version Selection
Compiling Telescoping Languages

Compilation Framework

- Type Inferencing
- Library Compilation
  - Jump Functions
  - Identities
  - Specialization
- Script Compilation
  - Property Propagation
- Smart Run-Time System
  - Version Selection

Group Presentation, Feb 15th, 2001

Optimizations for Telescoping Languages
Example Codes

• Real DSP codes used by ECE wireless group
• Long Running codes, potential for optimization
• Written in Matlab (even though slow)
• Parts of the codes re–used extensively (candidates for domain–specific lib routines)
Useful Optimizations: Vectorization

```matlab
function z = jakes_mpl (blength, speed, bnumber, N_Paths)

....

for k = 1:N_Paths

....

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
```
Useful Optimizations: Vectorization

```matlab
function z = jakes_mp1 (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
    %   end
    %   xc(j) = s
    %   qrt(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
```

Group Presentation, Feb 15th, 2001
Useful Optimizations: CSE

```matlab
function z = jakes_mp1 (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));

....

end
```
Useful Optimizations: Preallocation

```matlab
function z = mdlOutputs (K, N, L, D, sprd_type, sprd_codes)
    ....

    for ii = 1:L
        ....

        U_ii(ii,:,:,:) = zeros(N, 2*(N+1)*K)
        for user_i = 1:K
            for chip_i = 1:N
                U_ii(ii,:,.....) = ....
            end
        end
    end
    ....

end
```
Useful Optimizations: Preallocation

```
function z = mdlOutputs (K, N, L, D, sprd_type, sprd_codes)

....

U_ii(:,:, :) = zeros(L, N, 2*(N+1)*K)
for ii = 1:L

....

% U_ii(ii,:,:,:) = zeros(N, 2*(N+1)*K)
for user_i = 1:K
    for chip_i = 1:N
        U_ii(ii,:,....) = ....
    end
end
end

....

end
```
Procedure Strength Reduction

- Procedure called inside loop
  - several arguments typically invariant
  - move invariant computations into init part
  - do incremental computations inside loop
Procedure Strength Reduction

- Procedure called inside loop
  - several arguments typically invariant
  - move invariant computations into init part
  - do incremental computations inside loop

```plaintext
for i = 1:N
    f (c1, c2, i, c3)
end
```
Procedure Strength Reduction

- Procedure called inside loop
  - several arguments typically invariant
  - move invariant computations into init part
  - do incremental computations inside loop

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

```
f_init (c1, c2, c3)
for i = 1:N
  f_iter (i)
end
```
Procedure Strength Reduction

....

for ii = 1:200
    chan = jakes_mpl (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
....
Procedure Strength Reduction

....

jakes_mp1_init (16500, 160, num_paths);
for ii = 1:200
    chan = jakes_mp1_iter (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
....
Procedure Vectorization

- Procedure called inside a loop
- Loop can be distributed around the call
  - interchange loop and call
  - vectorize the loop inside the procedure
Procedure Vectorization

for ii = 1:200
    chan = jakes mpl (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);
    [ol, d1, d2, d3, mf, m] = codesdhd (y, a, h, NO, Tm, Bd, M, B, n0);
end
end

....
Procedure Vectorization

....

for ii = 1:200
    chan = jakes_mpl (16500, 160, ii, num_paths);
end

for ii = 1:200
    ....

    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
....

Group Presentation, Feb 15th, 2001  Optimizations for Telescoping Languages
Procedure Vectorization

```matlab
chan = jakes_mp1_vectorized (16500, 160, [1:200], num_paths);

for ii = 1:200
    ....
    
    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
....
```
ctss: strength reduction

Optimized execution times for top-level procedures in ctss relative to unoptimized:

- jakes_mp1
- newcodesig
- codesdhd
- whole program

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

- Optimized
- Original
**jakes_mp1: vectorization**

![Bar chart showing comparison between original and vectorized execution times for 100 iterations.](chart.png)
chan_est: strength reduction
outage_lb_fad: strength reduction

![Bar Chart: Performance Improvement in outage_lb_fad](image)
Conclusion

• High pay-off optimizations
  – vectorization
  – common subexpression elimination
  – pre-allocation
  – beating and dragging along

• Two new optimizations
  – procedure strength reduction (10% – 50% gain)
  – procedure vectorization
Related Work

• Source level transformations
  – DeRose’s PhD (UIUC, 1995)
  – Menon & Pingali (Cornell, 1999)

• Currying in functional languages

• Automatic Differentiation
  – ADIFOR project

• APL
  – Abram’s PhD (Stanford, 1970)

• Translation to lower–level languages
  – MCC (Mathworks), MAJIC (UIUC), MATCH (NWU), Menhir (Irisa, France), CONLAB (Univ of Umea, Sweden), Otter (Oregon State Univ)
Current and Future Work

• Implementation
  - Matlab front-end ready
  - Need
    • jump fns, dependence, SSA, array section analysis
    • high-payoff optimizations
    • inter-procedural framework
    • variants database creation and lookup

• Theory
  - Type inferencing
  - Annotation language for library identities