Telescoping Languages

or

High Performance Computing for Dummies – II

presentation by

Arun Chauhan

joint work with

Ken Kennedy
Other Collaborators

• Rice
  – John Mellor–Crummey
  – Rob Fowler
  – Bradley Broom
  – Keith Cooper
  – Linda Torczon

• Outside Rice
  – Jack Dongarra
  – Lennort Johnson
  – Dennis Gannon
Motivation

• Shortage of programmers
  – increasing application demands
  – rapidly changing architectures
  – need programmers for scientific applications too
Motivation

- Shortage of programmers
  - increasing application demands
  - rapidly changing architectures
  - need programmers for scientific applications too

- High Performance programming is hard
  - increasingly a specialized activity
  - more complex architectures
  - more high performance applications
One Solution

• Enable end–users to program
  – language should be high level
  – should provide domain–specific features
  – must have effective and efficient compilers
One Solution

• Enable end–users to program
  – language should be high level
  – should provide domain–specific features
  – must have effective and efficient compilers

• Scripting systems like MATLAB exist
  – very popular with end–users
  – lack effective and efficient compilers
Fundamental Observations

• Libraries extremely important
  – cannot treat libraries as black boxes
  – lib sources may not be available to end users

• Compiling user scripts must be fast
  – should follow principle of no surprise
Existing Approaches:
based on transforming to lower level languages
Existing Approaches:

based on transforming to lower level languages

- (potentially) high compilation time
- expert knowledge about libraries lost
Example 1

```matlab
function result = matrix_op (input_1, input_2, step)

i = 1
for j = 1:N
    result(i) = result(i) + input_1(j)*input_2(j)
i = i + step
end
```
Expert Knowledge

Example 1

```matlab
function result = matrix_op (input_1, input_2, step)

i = 1
for j = 1:N
    result(i) = result(i) + input_1(j)*input_2(j)
    i = i + step
end
```

Example 2

```matlab
....
x = sin (a);
y = cos (a);
....
```

```matlab
....
[x, y] = sincos (a);
....
```
Desiderata

- Utilize expert knowledge on libraries
- Fast compilation of user-scripts
- Still achieve high performance
Telescoping Languages Approach

- library writer
- library compiler
- end user
- script compiler
Telescoping Languages Approach

- write library procedures
- annotate library procedures

```
result = matrix_op (input_1, input_2, step)
```

"expect step to be non-zero most of the times"

- library writer
- library compiler
- end user
- script compiler
Telescoping Languages Approach

write library procedures

result = matrix_op (input_1, input_2, step)

"expect step to be non-zero most of the times"

annotate library procedures

extensively compile library routines

result = matrix_op_non_zero_step (...)

library writer

library compiler

end user

script compiler
Telescoping Languages Approach

- write library procedures
  - result = matrix_op (input_1, input_2, step)
  - "expect step to be non-zero most of the times"

- extensively compile library routines
  - result = matrix_op_non_zero_step (...)

- write scripts using libraries
  - r = matrix_op (A, B, 1)

- library writer
- library compiler
- end user
- script compiler
Telescoping Languages Approach

library writer

write library procedures

result = matrix_op (input_1, input_2, step)

annotate library procedures

"expect step to be non-zero most of the times"

library compiler

extensively compile library routines

result = matrix_op_non_zero_step (...)

done

end user

write scripts using libraries

r = matrix_op (A, B, 1)

script compiler

pick the most optimized version

r = matrix_op_non_zero_step (...)

Pizza Talk, Feb 26th, 2001
Telescoping Languages Approach

- User script
- Annotations
- Library
- Enhanced language compiler
- Language builder
- Optimized object code
Telescoping Languages Approach

Annotations → Language Builder

User Script → Enhanced Language Compiler → Optimized Object Code

Annotations → Language Builder

Library → Language Builder
Telescoping Performance

user script → enhanced language compiler → optimized object code

Telescoping Performance
Strength Reduction

```
for i = n_0:n_0+N
    ....
    x = i * c;
    y = g(x);
    ....
end
```
Strength Reduction

for $i = n_0:n_0+N$

.....

$x = i \times c$;

$y = g(x)$;

.....

end

$x = n_0 \times c$;

for $i = n_0:n_0+N$

.....

.....

$y = g(x)$;

$x = x + c$;

.....

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

```
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_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
....
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
....
....
ctss: strength reduction

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

- jakes_mp1: Optimized execution time
- newcodesig: Optimized execution time
- codesdhd: Optimized execution time
- whole program: Optimized execution time

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

- Optimized:
  - jakes_mp1
  - newcodesig
  - codesdhd

- Original:
  - jakes_mp1
  - newcodesig
  - codesdhd
chan_est: strength reduction

![Performance Improvement Graph]

- Original (per iteration)
- Init call
- Init (with preallocation)
- Iterative call

**Execution Time (seconds)**

**Performance Improvement in sML_chan_est**
outage_lb_fad: strength reduction

performance improvement in outage_lb_fad

execution time (thousands of seconds)
Conclusion

• Telescoping Languages approach
  – enable end-users write high perf. programs
  – libraries optimized as primitive operations
  – fast compilation of user scripts

• Procedure Strength Reduction
  – 10% – 50% gain