Using Compiler Techniques to Improve Automatic Performance Modeling

ARNAMOY BHATTACHARYYA, GRZEGORZ KWASNIEWSKI, TORSTEN HOEFLER

AUTOMATING PERFORMANCE MODEL GENERATION

- Representing performance as function of program inputs

\[ T(N) = t N^3 \]

- \( t = 2.2 \text{ns}^{-1} \)

Why?

1. Scalability
2. Insight into requirements

We want to generate models **ON THE FLY**

**AUTOMATING PERFORMANCE MODEL GENERATION**

- Existing techniques:
  - Static: Counts Loop iterations
  - Dynamic: Use ML on profiled data

**Problems:**

- Static Analysis [SPAA ‘14]: *Imprecise* sometimes
- Dynamic Analysis [PACT ‘14]: *Overhead restrictions*

When **ON THE FLY**, overhead should be negligible
**AUTOMATING PERFORMANCE MODEL GENERATION**

**Best of both worlds**

- **Static** → **Hybrid**

  - Precision – Upto 10% improvement (on avg. 4.5%)
  - Overhead – Upto 65% reduction (on average 25%)

---

**Our technique**

- **Precision**
- **Overhead**
Dynamic Analysis

- Main Idea:
  - Generating a model by “selecting” the best features from a bag of “candidate features”

  \[ p = \{ t_i^k \log^l t_i^k, k, l \in \mathbb{R}, t_i \in I \} \]

- Example:
  - Program input: \( n \)
  - We select \( k=\{1,2\} \) and \( l = \{0,1\} \)
  - Bag of features = \( n, n^2, n \log n, n^2 \log n \)
  - ON THE FLY feature selection

Dynamic Analysis

- Main Idea:
  - Generating a model by “selecting” the best features from a bag of “candidate features”

  \[ p = \{ t_i^k \log^l t_i^k, k, l \in \mathbb{R}, t_i \in I \} \]

- Example with two inputs:
  - Program input: \( n, m \)
  - We select \( k=\{1,2\} \) and \( l = \{0,1\} \)
  - Bag of features = \( n, n^2, n \log n, n^2 \log n, n^2 \log^2 n, m, m^2, m \log m, m^2 \log m \)
  - BUT… What about terms like \( n^m \) or \( n^{3 \log m} \)??
Static Analysis

Count # of iterations as a function of program inputs

- Existing methods –
  - i) Polyhedral Model
  - ii) Hoefler – Kwasnewski method [SPAA ‘14]

“Better” than polyhedral!

1) Over approximation (e.g. iter_variable = iter_variable * 2 )
2) No support for non-constant updates

```c
j=1;
k=5;
while (j>0){
    j=j+k;
k--;
}
```
Static Analysis

Problem:

- Still can't handle some specific loops (e.g. indirection in loop condition)

```plaintext
do j=lastrow-firstrow+1 sum = 0.d0
   do k=rowstr(j),rowstr(j+1)-1
      sum = sum + a(k)*p(COLIDX(k))
   enddo
   w(j) = sum
endo
```

- Give `undef` terms in the model
  
  # iterations = (lastrow – firstrow) * `undef`

Our Contribution: Combining two

- Extract predictors (even interaction terms 😊) from the static model

- If `undef`, use profiling and ML dynamically.

- Include interaction terms in bag of features.
Our technique

Precision

Overhead

### Overhead reduction:

1. Batched model update
   - **ON THE FLY** modeling
   - Function call overhead

```c
if (cell_coord[i, c] > 0, ncells) then
    do k = 0, cell_size[2], c-1
        do j = 0, cell_size[2], c-1
            do i = cell_size[2], c-2, cell_size[2], c-1
                do m = 1, 5
                    out_buffer(as(0)+p0) = u(w[i,j,k,c])
                    p0 = p0 + 1
                end do
            end do
        end do
    end do
end do
```

6 million iterations, for 16 processes
1.6% overhead just from this loop
23% in total
Overhead reduction:
1. Batched model update
   - ON THE FLY modeling
   - Function call overhead

   ```
   if (cell_coord(i,0) .ne. small) then
      do j = 0, cell_size(3,0)-1
         do i = 1, cell_size(1,0)-1
            do m = 1, 5
               out_buffer(as(0),ip0) = u(w,i,j,k,c)
               ip0 = ip0 + 1
            end do
         end do
      end do
   end do
   ```

   6 million iterations, for 16 processes
   1.6% overhead just from this loop
   23% in total

   - Solution:
   - Function call **once per batch**
   - Batch size optimization

---

Overhead reduction:
2. Performance Model Similarities

```plaintext
do i = mm,m0,-1
   z( jg(1,i,0), jg(2,i,0), jg(3,i,0) ) = -1.0D0
enddo
```

```plaintext
do i = mm,m1,-1
   z( jg(1,i,1), jg(2,i,1), jg(3,i,1) ) = +1.0D0
enddo
```
Overhead reduction:
2. Performance Model Similarities

do i = mm, m0, -1
   z( jg(1,i,0), jg(2,i,0), jg(3,i,0) ) = -1.0D0
enddo

do i = mm, m1, -1
   z( jg(1,i,1), jg(2,i,1), jg(3,i,1) ) = +1.0D0
enddo

Loop 1: \( c_1 \) * (\( mm - m0 \))  
Loop 2: \( c_2 \) * (\( mm - m1 \))

Solution:

*Program Dependence Graph* based *similarity detection*

Model one loop *per* similarity group
Overhead reduction:
3. Regulate Frequency of model update

Models get more precise over time ON THE FLY

Solution:
Prediction hit counter
Delay the next update using exponential backoff

\[ q = b \cdot \text{rand}(0, 2^{h_c}) \]

Experimental Evaluation

- Measurement of Precision:
  - PARS (Predicted adjusted R-square) (values between 0 - 1)
  - Lack of fit (LOF) (p-value < 0.05 tells better models are possible)

- Measurement of Overhead Reduction:
  - Software details:
    - LLVM 3.3
    - NAS and MILC benchmarks
    - Intel core-i7 3.4 GHz quad-core machine where each core is 2-way multi-threaded
Accuracy improvement

On Average 4.5%

Overhead reduction:

2. Performance Model Similarities

What percentage of loops can we cluster?

Best 50% (BT), Worst 2% (kid_su3_rmd), Average 26%

How precise is clustering?

Best 100% (FT), Worst 90% (kid_su3_rmd), Average 93%
Overhead reduction:
3. Regulate Frequency of model update

What percentage loops have changing behaviour?

Best gp_quark_prop – 8%
Best MG – 0%
Average – 1.5%

On Average 25% reduction
Conclusion

- Combine existing static and dynamic approaches
- Improve precision upto 10%
- Reduce overhead upto 65%
- Hope that low-overhead automatic model generation techniques will become popular to system engineers.

Questions?
Sample models

```c
sum=0.0;
FORALLSITES(i,s){
  for(dir=XUP;dir<=TUP;dir++)
    sum+= (double)ahmat_mag_sq(s->mom[dir])
    -4.0;
}
```

\[ f(P) = nx \cdot ny \cdot nz \cdot nt \cdot (1.56 \cdot TUP \\
- 0.49 \cdot XUP + 0.45) + 0.001 \]

Previous approach:
- 45 terms!!
- PARS of 0.75
- p-value 0.01

New approach:
- PARS of 0.88
- p-value 0.40