Optimizing Expression Selection for Lookup Table Program Transformation

Chris Wilcox, Michelle Mills Strout, James M. Bieman

Presented by:
Chris Hu, John Penington, Steven Schmatz, Zephaniah Hill

Agenda

- **Introduction & Background - Zephaniah Hill**
- Lookup Table Optimization - John Penington
- Implementation - Chris Hu
- Results & conclusions - Steven Schmatz

What is LUT Optimization?

- Preload a table with input output pairs
- Run the table instead of the function when it’s called
- Interpolate or use nearest value for values with no table entries
    - This can become very difficult for non-numerical functions

LUT Simple Example:

```
x = 0

For x < 1000

        exp(x%4)

        x++
```

n	exp(n)
0	1
1	2.718
2	7.389
3	20.08

LUT Applications:

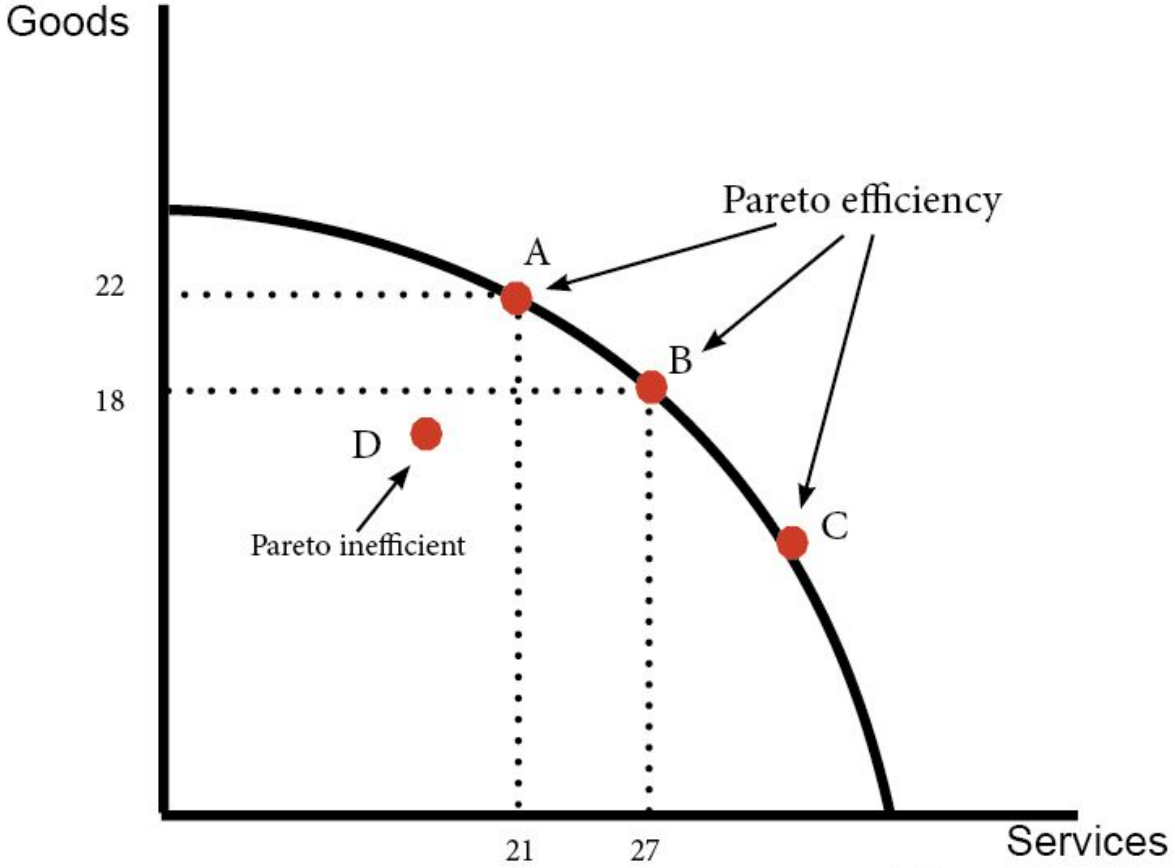
- Fastest Fourier Transform in the West (FFTW)
- Rapid Radiative Transfer Model (RRTM) - A climate change model
- Flight Simulators

How Mesa Addresses Current LUT Issues

- Manual insertion of LUTs in source code:
  - Slow, error prone
  - Makes code hard to read
  - Makes accuracy and error calculations difficult
  - Writing performance code by hand can be difficult

- Using the newest Mesa version:
  - Automatically locates optimization candidates
  - Provides error estimation and a performance model
  - Finds a set of pareto optimal LUT transforms

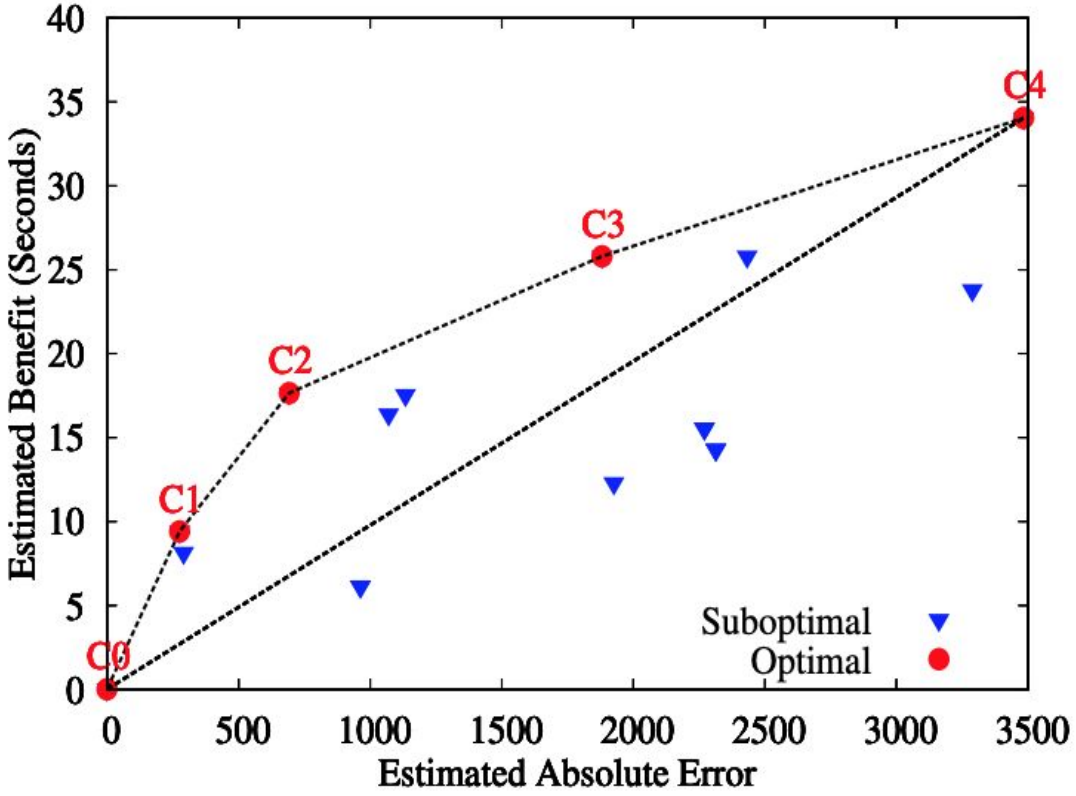
Pareto Optimal as a Concept



Pareto Optimal
in LUT Optimization

Goods = Error

Services =
Computation Time



Agenda

- Introduction & Background - Zephaniah Hill
- **Lookup Table Optimization - John Penington**
- Implementation - Chris Hu
- Results & conclusions - Steven Schmatz

Defining LUT Optimization Problem

INPUT VALUES
D_i: real - input domain for expression
M_i: real - maximum slope for expression
B_i: real - potential benefit for expression
CS: integer - cache size

VARIABLE DEFINITIONS
X_i: boolean - expression selector
S_i: integer - table size $(0 < S_i < CS)$
E_i: real - computed as $(D_i/S_i) * (M_i/2.0)$

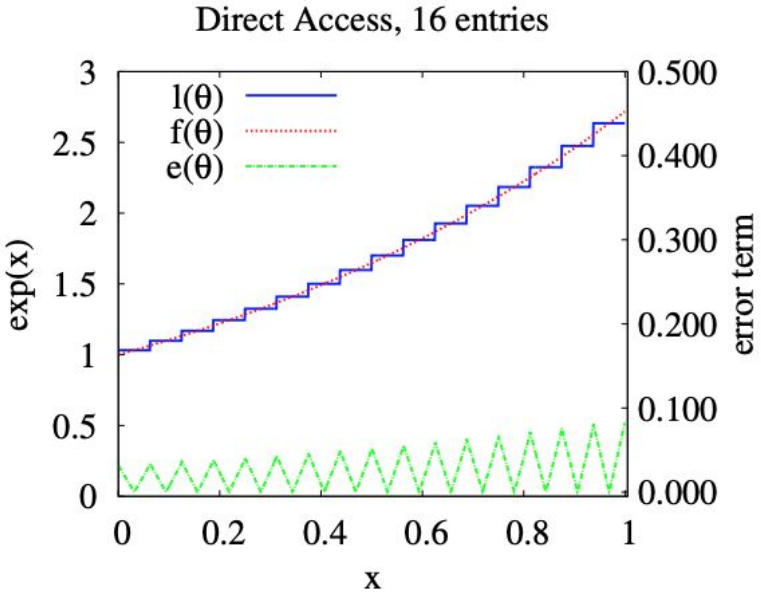
PROBLEM OBJECTIVES
maximize TotalBenefit = $\sum((X_i?1:0)*B_i)$ - maximize benefit
minimize TotalError = $\sum((X_i?1:0)*E_i)$ - minimize error

CACHE CONSTRAINT
$\sum S_i = CS$ - use entire cache

INTERSECTION CONSTRAINTS
$X_i + X_j \leq 1$, for intersecting X_i and X_j

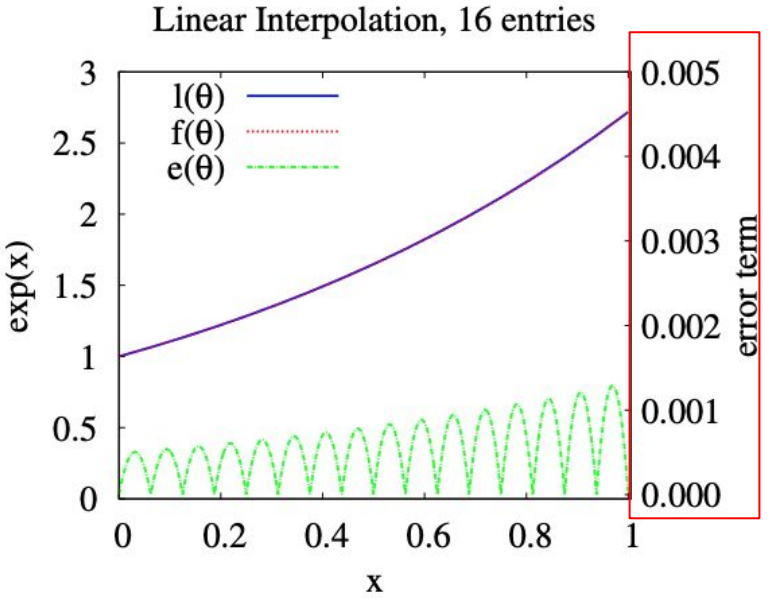
Error (Direct Access)

$$\text{MaxError} = \left(\frac{\text{Domain}}{\text{Size}}\right) * \left(\frac{\text{MaxSlope}}{2}\right)$$



Error (Linear Interpolation)

$$\text{MaxError} = \left(\frac{\text{Domain}}{\text{Size}}\right)^2 * \left(\frac{\text{MaxDelta}}{8}\right)$$



Performance Modeling

$$\mathrm{B} = (\mathrm{Cost}(\mathrm{Op}) * \mathrm{Count}(\mathrm{Op}) - \mathrm{Cost}(\mathrm{Access})) * \mathrm{Freq}$$

Linear Interpolation: divide benefit by cost of interpolating

Example

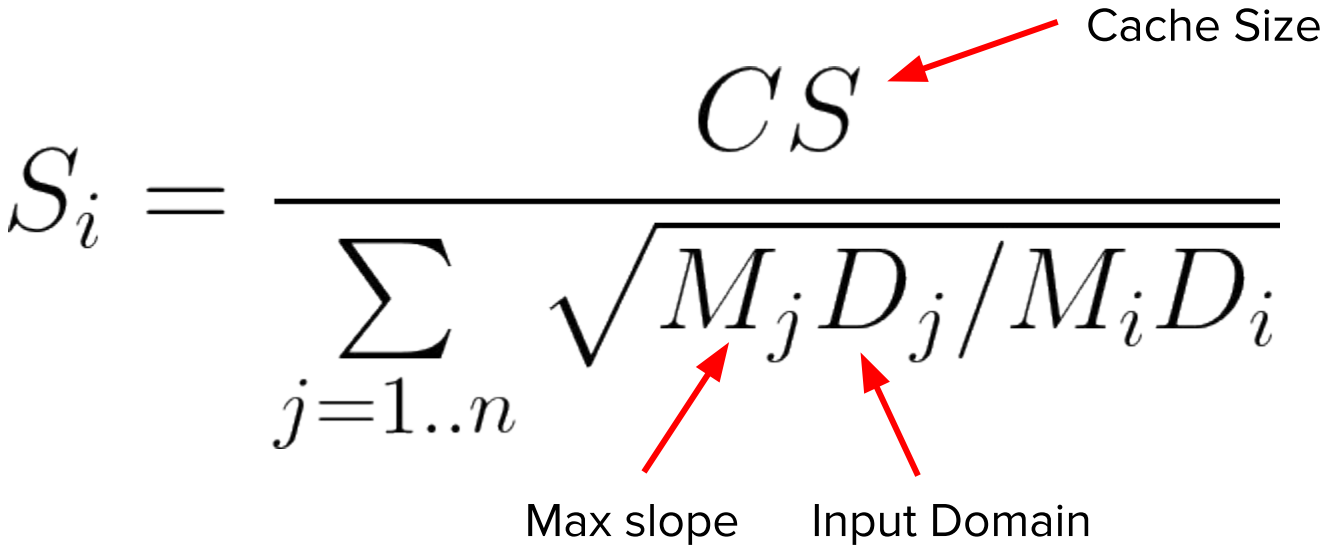
- Sine call takes 45ns
- LUT access takes 7.4ns
- Frequency is 10^8

DA = (45ns*1 - 7.4ns) * 10^8 = 3.76s

LI = 3.76s / 1.8 = 2.089s

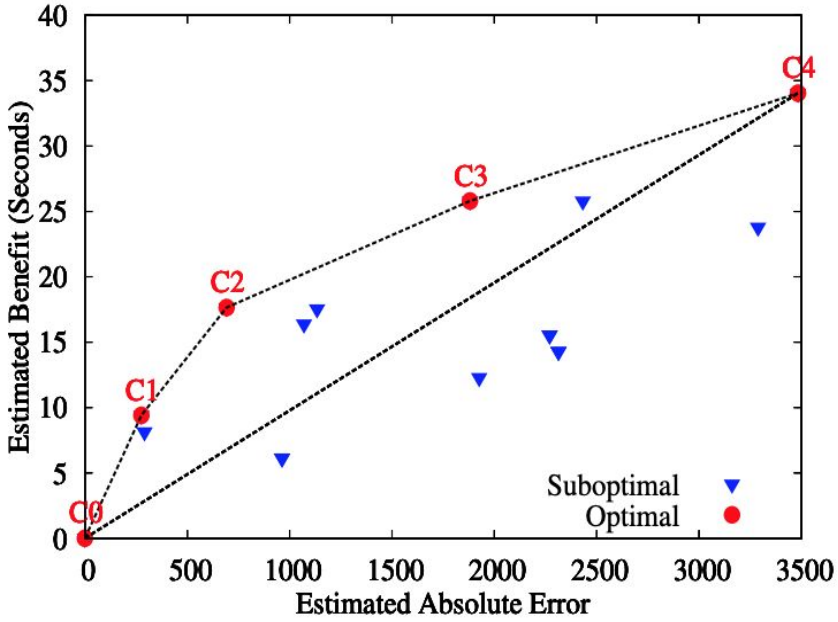
Local Optimization

- Find optimal allocation of cache resources for the set of expressions in each solution



Global Optimization

- Sort solutions by estimated error and performance to find the Pareto optimal solutions
- Select the solutions that lie on the convex hull (calculated with Graham Scan algorithm)

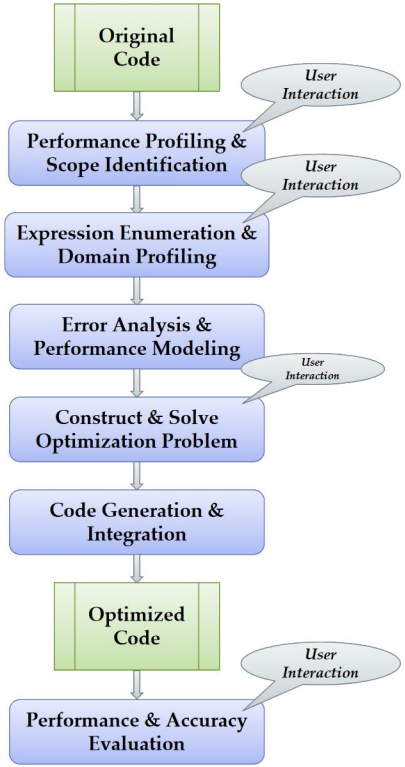


Agenda

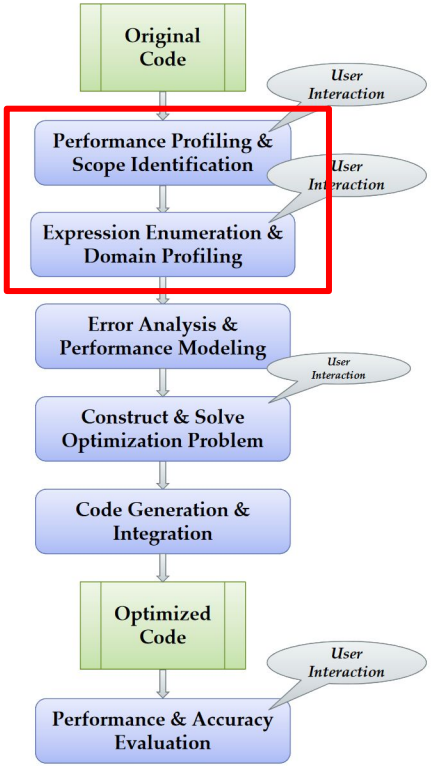
- Introduction & Background - Zephaniah Hill
- Lookup Table Optimization - John Penington
- **Implementation - Chris Hu**
- Results & conclusions - Steven Schmatz

Implementation (Mesa Tool)

- Program that automatically generates lookup table optimizations

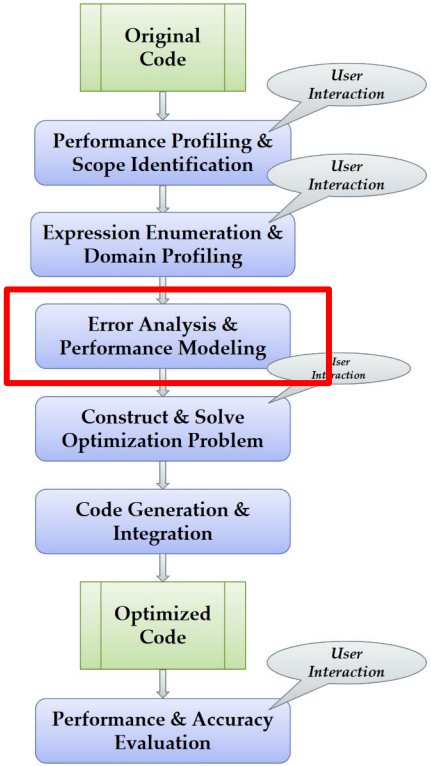


Implementation (Mesa Tool)



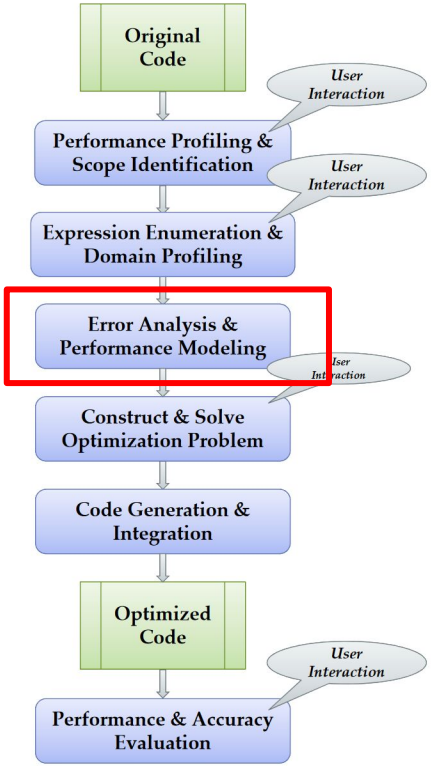
- First profile run:
  - User profiles original code to identify most costly subroutines
- Second profile run:
  - Determines expressions that can be optimized
  - Determines the domain of each expression
  - Paper only considers certain math library calls (sin, asin, sinh, cos, acos, cosh, exp, log, ...)
  - Some expressions are redundant and/or be combined

Implementation (Mesa Tool)



- Calculate for each expression:
  - Maximum slope
  - Error
  - Performance benefit

Implementation (Mesa Tool)

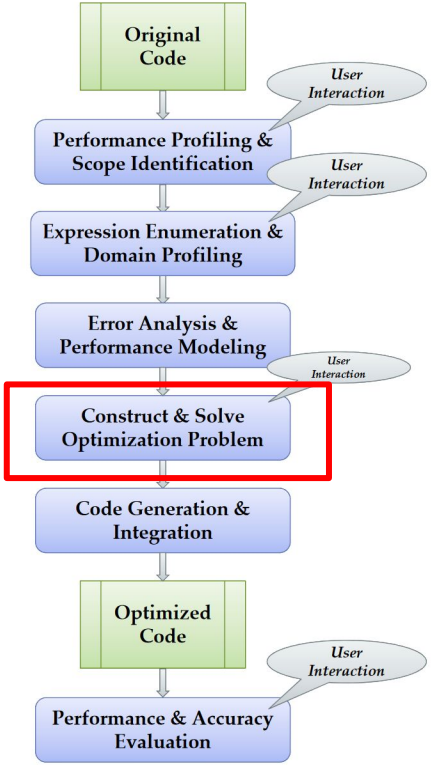



```
#pragma LUTOPTIMIZE
double ScatterSample(Sample sample, ...
{
S35     double dProduct;
S36     double dSum0 = 1.0;
S37     double dSum1 = 1.0;
S38
S39     // Iterate geometry
S40     for (int j = 0; j < vGeometry.size(); j++)
S41     {
S42         dProduct = (sample.x * vGeometry[j].x) ...
S43         dSum0 += exp(dProduct) + sin(dProduct);
S44         dSum1 += exp(dProduct) + cos(dProduct);
S45     }
S46
S47     // Return answer
S48     return dSum0 * dSum0 + dSum1 * dSum1;
}
```

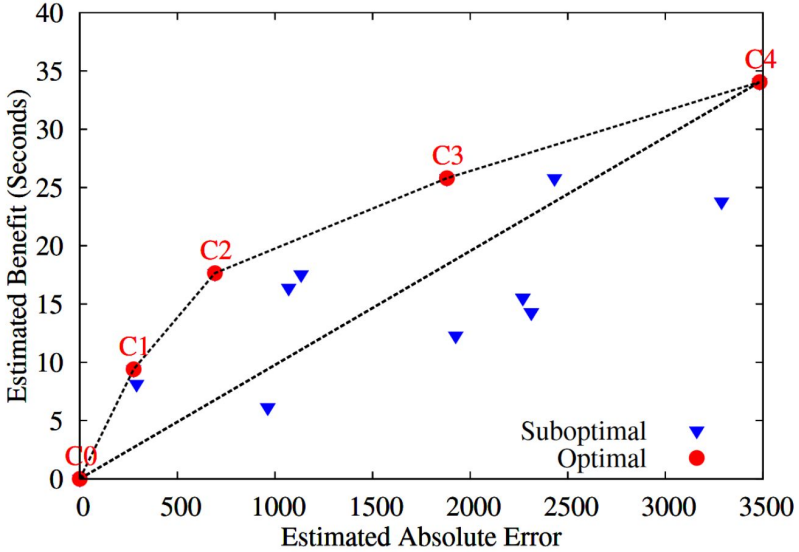
Table III
INPUT DATA FOR OPTIMIZATION PROBLEM.

Expression Identifier	Expression Description	Statement Identifiers	D_i	M_i	B_i
X0	exp(dProduct)	S43	2.44	3.31	6.15s
X1	sin(dProduct)	S43	2.44	1.00	8.15s
X2	exp(dProduct) + sin(dProduct)	S43	2.44	3.67	16.40s
X3	exp(dProduct)	S44	2.44	3.31	6.15s
X4	cos(dProduct)	S44	2.44	0.95	9.40s
X5	exp(dProduct) + cos(dProduct)	S44	2.44	2.38	17.65s
X6	exp(dProduct)	S43, S44	2.44	3.31	12.30s

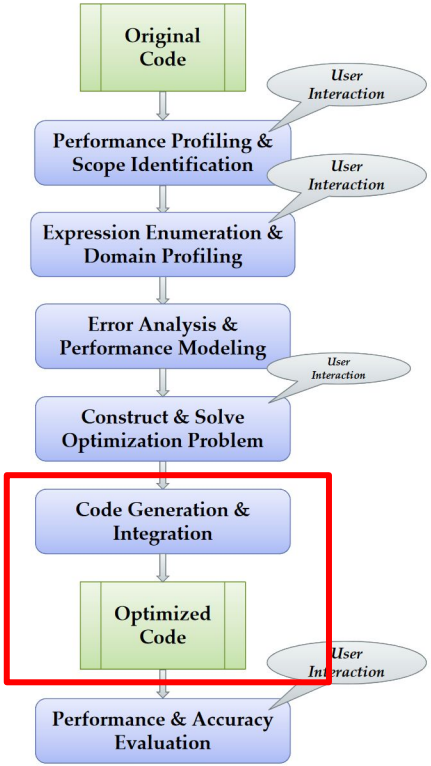
Implementation (Mesa Tool)



- Generate possible solutions and presents optimal solutions to user



Implementation (Mesa Tool)



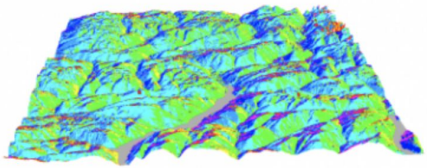

```
// Start of  code generated by Mesa, version 2.0
// LUT constants
const double X5_lower = -1.2461340000e+00;
const double X5_upper =  1.1962890000e+00;
const double X5_gran  =  5.2249019600e-06;
class CLut {
public:
  // LUT Constructor
  CLut() {
    double dIn, dOut;
    for (dIn=X5_lower; dIn<=X5_upper; dIn+=X5_gran) {
      dOut = X5_orig(dInput+(X5_gran/2.0));
      X5_data.push_back(dOut);
    }
  }
  // LUT Destructor
  ~CLut() {
    X5_data.clear();
  }
  // LUT function for expression
  float X5_lut(float X5_param) {
    X5_param -= X5_lower;
    int uIndex = (int) (X5_param*(1.0/X5_gran));
    return(X5_data[uIndex]);
  }
  // Original expression
  double X5_orig(double dProduct) {
    return (exp(dProduct)+cos(dProduct));
  }
private:
  // LUT data structures
  std::vector<float> X2_data;
};
// Object instantiation
CLut clut;
// End of code generated by Mesa, version 2.0

// Expressions replaced by Mesa
S43 dSum0 += clut.X2_lut(dProduct);
S44 dSum1 += clut.X5_lut(dProduct);
```

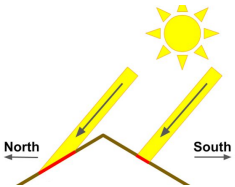
Agenda

- Introduction & Background - Zephaniah Hill
- Lookup Table Optimization - John Penington
- Implementation - Chris Hu
- **Results & conclusions - Steven Schmatz**

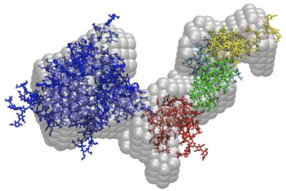
Computation Benchmarks

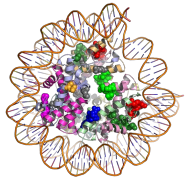
USGS Precipitation Runoff
Slope aspect + solar radiation



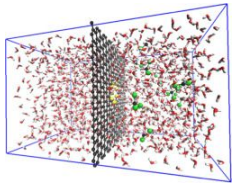


Molecule density analysis
Small angle x-ray scattering

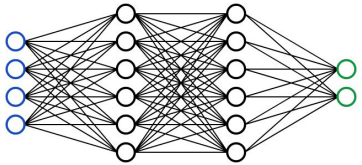




Cornell University
Molecular dynamics program



Neural net library (in-house)



Processing cost

Expression coalescing:
Z = exp(a) + exp(b)

(Intel Core 2 Duo, E8300, family 6, model 23, 2.83GHz, single core)

Application Name	Lines of Code	Number of Expressions	Possible Solutions	Actual Solutions	Pareto Solutions	Processing Time
PRMS Slope Aspect	35	9	512	384	**9**	**13.7s**
PRMS Slope Aspect	35	11	2048	425	**9**	**15.5s**
PRMS Solar Radiation	7	6	64	64	**8**	**14.1s**
SAXS Discrete	60	3	8	4	**3**	**11.2s**
SAXS Discrete	60	3	8	4	**3**	**16.5s**
SAXS Continuous	30	5	32	20	**4**	**10.8s**
Stillinger-Weber	44	6	64	36	**3**	**9.3s**
Neural Network (logistics)	5	2	4	3	**2**	**4.9s**
Neural Network (hypertan)	5	1	2	2	**2**	**2.8s**

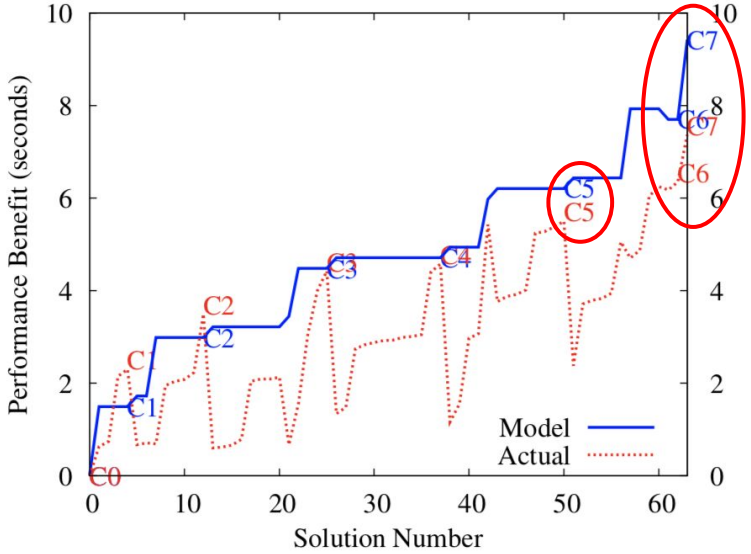
Performance boost

(Intel Core 2 Duo, E8300, family 6, model 23, 2.83GHz, single core)

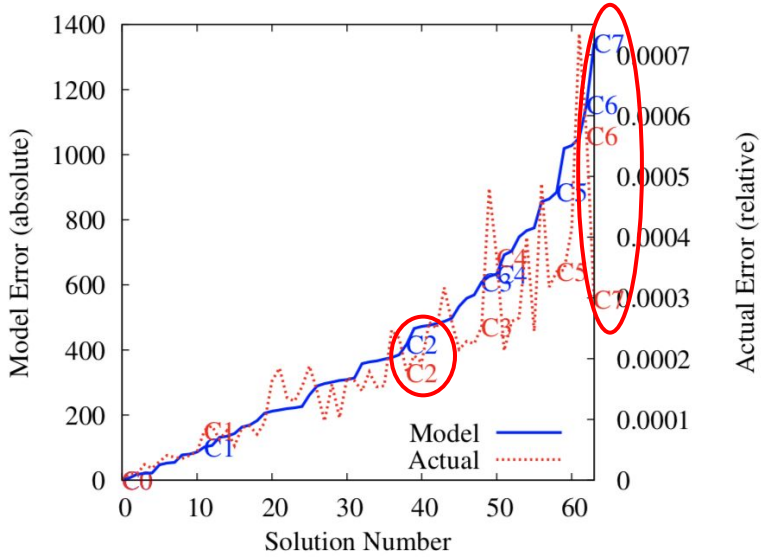
Application Name	Original Time	Optimized Time	Performance Speedup	Maximum Error	Memory Usage
Saxs Scattering (discrete)	196.2s	29.0s	6.8X	$4.06 \times 10^{-3}\%$	4MB
Saxs Scattering (continuous)	10.1s	2.5s	4.0X	$1.48 \times 10^{-4}\%$	4MB
Stillinger-Weber (simulation)	14.6s	10.4s	1.4X	$2.91 \times 10^{-2}\%$	1MB
Neural Network (logistics)	8.0s	3.6s	2.2X	$8.70 \times 10^{-2}\%$	4MB
Neural Network (hypertan)	10.9s	3.9s	2.8x	$6.30 \times 10^{-1}\%$	4MB
PRMS (slope aspect)	242ns	56ns	4.3X	$8.21 \times 10^{-6}\%$	4MB
PRMS (solar radiation)	13.7s	6.1s	2.2X	$2.97 \times 10^{-4}\%$	4MB

Model evaluation (solar radiation example)

Why an error here?



And here?



Clear benefits

Large performance increase (up to 6.8x demonstrated)
Minimal memory requirements (4MB) *What happens when we can use more?*
Less than 1% error threshold
Computes optimal LUT cache allocations
Models allow for error + benefit prediction
Saves programmer time from weeks of LUT tuning to less than a day of work

Threats to validity

Just six scientific applications - not many benchmarks, all in one domain *It should generalize to other elementary function call-based procedures*
LUT-based architectures depend on relative performance of memory access vs. function evaluation *Can change as architectures evolve*
Does not consider propagated error
Does not account for compiler optimizations - can overestimate benefit

Questions?