Accuracy-Aware Program Transformations

Saša Misailović

MIT CSAIL

Joint work with
Zeyuan Allen Zhu, Jonathan Kelner, Martin Rinard
Trend #1: Computations operate on enormous data sets

Trend #2: Big computations never execute perfectly

Trend #3: Huge space of alternative computations

Current paradigm of perfect software does not account for these trends

Process not all but just enough data

Enable programs to adapt and keep executing

Automate the process

Calls for a new paradigm!
Approximate Computations
Trend #1: Computations operate on enormous data sets. Process not all but just enough data.

Trend #2: Big computations never execute perfectly. Enable programs to adapt and keep executing.

Trend #3: Huge space of alternative computations. Automate the process.

Current paradigm of perfect software does not account for these trends. Calls for a new paradigm!
Tradeoff Space

- Time
- Error

Diagram showing a scatter plot with points scattered across the Time (vertical axis) and Error (horizontal axis) dimensions.
Optimization Framework

- Original program
- Input specification
- Transformation specification

Transformed Program
Tradeoffs and Configurations
Optimization Framework

Original program

Tree of Map-fold Tasks

Transformed Program

Tradeoffs and Configurations
Original program

Tree of Map-fold Tasks

Optimization Framework

Transformed Program

Can execute at various points in the tradeoff space
Optimization Framework

Tradeoffs and Configurations

Tradeoff Curve: (1+\(\varepsilon\))-optimal

[POPL ’12]
Search for Optimized Programs

Property: With high probability the result of the optimized program is within the specified error bound.

Query: Generate the program that executes in minimal time.
Example: Blackscholes

```plaintext
function Blackscholes (option)
    T = option.Time
    S = option.Strike
    V = option.Volatility
    t1 = ContinuousF1 (T, S, V)
    t2 = ContinuousF2 (T, S, V)
    return ContinuousF3 ( t1, t2, option)
```

```plaintext
function main (options)
    return sum ( map Blackscholes options )
```
Example: Blackscholes

```plaintext
function Blackscholes (option)
    T = option.Time
    S = option.Strike
    V = option.Volatility
    t1 = ContinuousF1 (T, S, V)
    t2 = ContinuousF2 (T, S, V)
    return ContinuousF3 ( t1, t2, option)

function main (options)
    return sum ( map Blackscholes options )
```

**Input specification:**
- option.Strike $\in [s_0, s_1]$  
- option.Time $\in [t_0, t_1]$  
- option.Volatility $\in [v_0, v_1]$  
- Size of options $\in [n_0, n_1]$  

Example: Blackscholes

```plaintext
function Blackscholes (option)
    T = option.Time
    S = option.Strike
    V = option.Volatility
    t1 = ContinuousF1 (T, S, V)
    t2 = ContinuousF2 (T, S, V)
    return ContinuousF3 (t1, t2, option)

function main (options)
    return sum ( map Blackscholes options )
```

Transformations:
- **Functions**: specialize implementations for expected input ranges
- Polynomial interpolation
- Transformed Computation:
  ```plaintext
  function ContinuousF1 (·, ·, ·) if inputs in expected ranges
  execute randomly one of the alternative versions
  else
  execute original version
  ```
Example: Blackscholes

```plaintext
function Blackscholes (option)
    T = option.Time
    S = option.Strike
    V = option.Volatility
    t1 = ContinuousF1 (T, S, V)
    t2 = ContinuousF2 (T, S, V)
    return ContinuousF3 (t1, t2, option)
```

```plaintext
function main (options)
    return sum (map Blackscholes options)
```

Transformations:

- **Reductions**: Randomly skip some of the inputs
- Extrapolate the result
- Value of each inputs computed only when required
Example: Blackscholes

```python
function Blackscholes (option)
    T = option.Time
    S = option.Strike
    V = option.Volatility
    t1 = ContinuousF1 (T, S, V)
    t2 = ContinuousF2 (T, S, V)
    return ContinuousF3 ( t1, t2, option)

function main (options)
    return sum ( map Blackscholes options )
```
Example: Blackscholes

```plaintext
function Blackscholes (option)
    T = option.Time
    S = option.Strike
    V = option.Volatility
    t1 = ContinuousF1 (T, S, V)
    t2 = ContinuousF2 (T, S, V)
    return ContinuousF3 ( t1, t2, option)

function main (options)
    return sum ( map Blackscholes options )
```

Result:
- Tradeoff curve of the whole program
- Randomized program that delivers tradeoffs
Blackscholes: Tradeoff Curve

![Graph showing the tradeoff curve between time (ms) and 95% error ($). The x-axis represents 95% error ($), ranging from 0 to 2, and the y-axis represents time (ms), ranging from 1.3 to 2.3. The curve shows a steep initial decrease followed by a nearly flat line at around 1.5 ms.]

- Time (ms)
  - 0
  - 0.5
  - 1
  - 1.5
  - 2

- 95% Error ($)
  - 1.3
  - 1.5
  - 1.7
  - 1.9
  - 2.1
  - 2.3

The graph illustrates the tradeoff between time and error, with a sharp decrease at the beginning and a plateau thereafter.
**Blackscholes: Tradeoff Curve**

**Property:** With probability at least 0.95, a portfolio value that the transformed program computes differs by at most $1 from the original program.

**Query:** Execute the program configuration that runs in a minimum amount of time.
Challenge: Reasoning about Uncertainty

• Expressions for error emergence and propagation
  • Probabilistic analysis
• Influences form of computations we can analyze, tightness of bounds, optimality of the search for transformed programs
• Core algorithm: our POPL ‘12 paper

Also Check: Michael Carbin’s Reasoning about Relaxed Programs Talk (Friday session)
Trend #1: Computations operate on enormous data sets. Process not all but just enough data.

Trend #2: Big computations never execute perfectly. Enable programs to adapt and keep executing.

Trend #3: Huge space of alternative computations. Automate the process.

Current paradigm of perfect software does not account for these trends. Calls for a new paradigm!