Some Improvements on KaniCUDA – a program synthesizer for CUDA

Chengkai Yang Advisor: 増原英彦

Background: KaniCUDA

- Replacement of global memory with shared memory improves runtime performance of CUDA programs.
- KaniCUDA[1] is a program synthesizer that replaces global memory access with shared memory access for the user.
- The user should declare shared memory, candidate variables, and global memory access.
- KaniCUDA emulates the program, profile the values of shared memory index and candidate variables, and calculates the shared memory access expression.
- KaniCUDA returns the optimized program back to the user.

[1] 蟹暁, 共有メモリ最適化のためのGPGPU プログラム合成器, 東京工業大学'18

Background: An Example

Original Program

```
__global__ void diffusion_kernel(float* in,
                                 float* out,
                                 int nx, int ny, int nz,
                                 float ce, float cw, float cn, float cs,
                                 float ct, float cb, float cc) {
  profile("threadIdx.x threadIdx.y blockDim.x blockDim.y csb c i j");
  int i = blockDim.x * blockIdx.x + threadIdx.x;
  int j = blockDim.y * blockIdx.y + threadIdx.y;
  int c = i + j * nx;
  int xy = nx * ny;
  __shared__ float sb[BLOCK_X * BLOCK_Y];
  int csb = threadIdx.x + threadIdx.y * blockDim.x;
  for (int k = 0; k < nz; ++k) {
    sb[csb] = in[c];
    int w = (i == 0)        ? c : c - 1;
    int e = (i == nx-1)     ? c : c + 1;
    int n = (j == 0)        ? c : c - nx;
    int s = (j == ny-1)     ? c : c + nx;
    int b = (k == 0)        ? c : c - xy;
    int t = (k == nz-1)     ? c : c + xy;
    out[c] =
        cc * in[c]
      + cw * __opt__.in[w]
      + ce * __opt__.in[e]
      + cs * __opt__.in[s]
      + cn * __opt__.in[n]
      + cb * in[b]
      + ct * in[t];
    c += xy;
  }
}
```

Candidate variables

Initialize shared memory

Global memory access

Emulate the program and profile the variables

Profiles

```
tid bid id smid threadIdx.x threadIdx.y blockDim.x blockDim.y csb c i j
0 0 1 1 0 0 3 4 0 0 0 0
1 0 2 2 1 0 3 4 1 1 1 0
2 0 3 N 2 0 3 4 2 2 2 0
3 0 10 4 0 1 3 4 3 9 0 1
4 0 11 5 1 1 3 4 4 10 1 1
5 0 12 N 2 1 3 4 5 11 2 1
6 0 19 7 0 2 3 4 6 18 0 2
7 0 20 8 1 2 3 4 7 19 1 2
8 0 21 N 2 2 3 4 8 20 2 2
9 0 28 10 0 3 3 4 9 27 0 3
10 0 29 11 1 3 3 4 10 28 1 3
```

Goal: smid = an expression using candidate variables

Generate and test arithmetic and logic expressions
(I rewrote this part)

Optimized Program

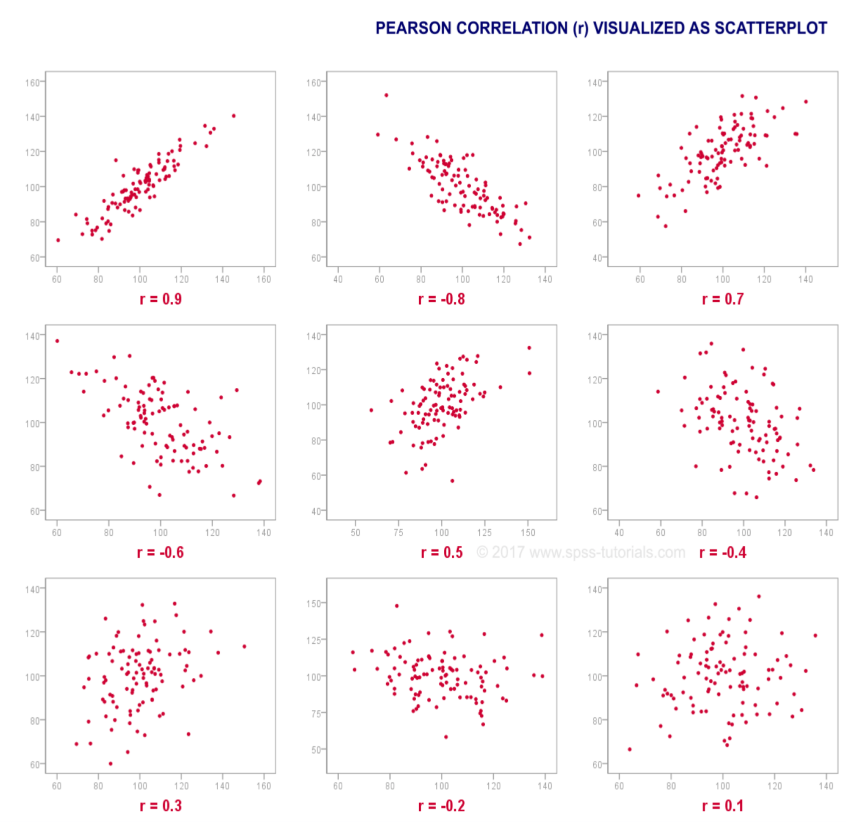
```
for (int k = 0; k < nz; ++k) {
  sb[csb] = in[c];
  int w = (i == 0)        ? c : c - 1;
  int e = (i == nx-1)     ? c : c + 1;
  int n = (j == 0)        ? c : c - nx;
  int s = (j == ny-1)     ? c : c + nx;
  int b = (k == 0)        ? c : c - xy;
  int t = (k == nz-1)     ? c : c + xy;
  out[c] =
      cc * in[c]
    + cw * ((((i - 1)!=(blockDim.y + 1))&&(i!=blockDim.x)) ? sb[(threadIdx.x==0) ? csb + 1 - 2 : csb ] : in
    [w])
    + ce * ((((blockDim.y + 1)!=i)&&((blockDim.x - 1)!=i)) ? sb[((threadIdx.x + 1)==blockDim.x) ? csb + 1
    : csb ] : in[e])
    + cs * ((((csb - 1)<j)||(blockDim.x!=threadIdx.y)) ? sb[(blockDim.x==threadIdx.y) ? csb + blockDim.x :
    csb ] : in[s])
    + cn * (((j==0)||(threadIdx.y!=0)) ? sb[(threadIdx.y==0) ? csb + 1 - blockDim.y : threadIdx.x ] : in[
    n])
    + cb * in[b]
    + ct * in[t];
  c += xy;
```

Optimization: Motivation

- Runtime performance of KaniCUDA is limited by its naively implemented expression generation and verification.
- KaniCUDA generates expressions for all candidate variables, but very few of them are used in the result. The data could be pre-processed to eliminate irrelevant variables.
- KaniCUDA represents expression AST with java objects, which are slower than arrays.
- KaniCUDA generates and stores sufficiently many expressions and then verify each expression. It takes up more time and space. An expression should be tested immediately after it is generated.
- KaniCUDA does not avoid equivalent expressions, such as x + y and y + x.

Optimization: Profile Analysis

- Examine the correlation between smid and candidate variables.
- Pearson correlation coefficient, denoted by r, is a measure of the strength of a linear association between two variables.
- Search on related variables first. If no expression is correct, continue searching on other variables.

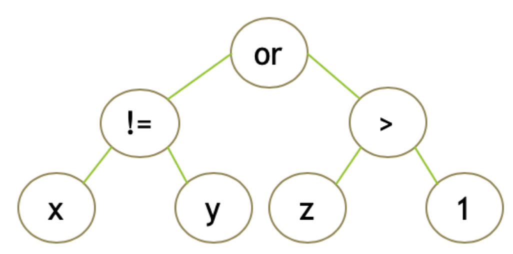


Optimization: Arithmetic Expression

- Arithmetic expressions are written in a linear style.
- Three basic components of arithmetic expression
  - Unit : constant or variable — ex. 2, x, y
  - Term : unit, or unit * unit — ex. y, y*x, y*x*2
  - Expression: term +- term +- term… — ex. y, y + x, y*x + y + 2
- Identifiers 1 : constant 2 : variable 3 : + 4 : - 5 : *
  - Ex. constant list = [1, 2] — variable list = [x, y, z]
  - [1, 0] = 1 — [2, 1] = y
  - [2, 0, 4, 1, 1] = x – 2 — [2, 0, 5, 1, 1, 3, 2, 2] = x*2 + z
- Avoid equivalent expressions due to symmetry of + and *.
  - Give each term a unique value.
    - Value of unit = index (start from 1) of the constant/variable
    - Value of Term = (string of values of each unit)$_{number\ of\ units}$
    - Ex. Units: [1, 2], [x, y, z]
    - v(2) = 2, v(y) = 4, v(y * y * 2) = 442_5= 4 * 25 + 4 * 5 + 2 * 1 = 122
  - Sort the terms with their values.
    - Value of current unit should not exceed value of previous unit
    - Ex. y * x is okay, x * y is not
    - Value of current term should not exceed value of previous term
    - Ex. x * 2 + y is okay, y + x * 2 is not, y - x - y * 2 is not
    - Note: this rule applies to +/-, but not to transition from + to -
    - Ex. x * 2 - y is okay, y - x * 2 is also okay
- Continuation-passing style
  - Functions take an argument: an explicit "continuation", i.e. a callback function.
  - It allows different expressions to be generated based on one expression.
  - It allows an expression to be tested immediately after it is generated.

Optimization: Logic Expression

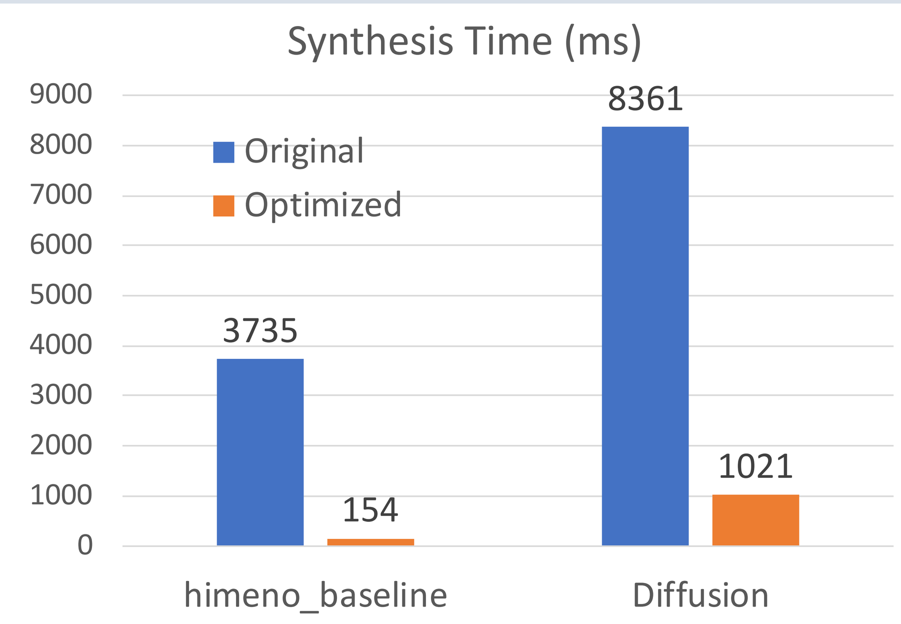
- Logic expression is represented in a binary tree style (left & right)
- Ex. (x != y) || (z > 1)
- Identifiers -1 : && -2 : || -4 : != -5 : < -6 : > 1 : constant 2 : variable 3 : + 4 : -



Ex.
constant list = [1, 2] — variable list = [x, y, z]
[1, 1] = 2, — [2, 0] = x
[-2, -4, 2, 0, 1, 1, -5, 2, 2, 3, 2, 1, 1, 0] = (x != 2) || (z < y + 1)

Conclusion

- The runtime performance has been significantly improved.
- The expression generator could be useful in other projects.
- Future work
  - More flexible expression generator.
  - User determined different parameters sets for profiles



Environment:

- OS: macOS Mojave version 10.14.5
- CPU: 2.2 GHz Intel Core i7
- Memory: 16 GB 1600 MHz DDR3