Genetic improvement of GPU code

Jhe-Yu (Jerry) Liou, Stephanie Forrest, Carole-Jean Wu

Computer Science and Engineering

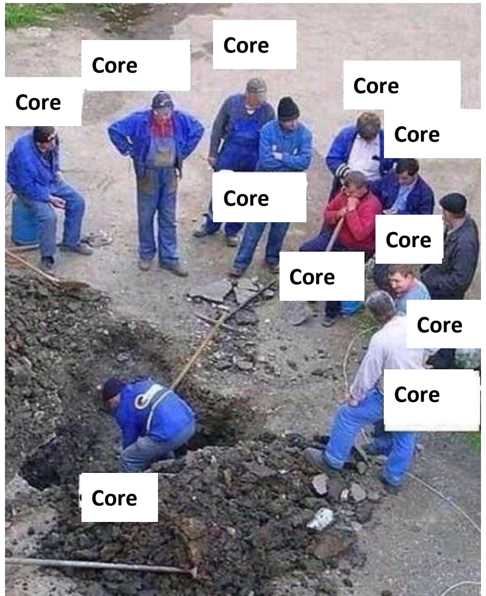
Biodesign institute

Arizona State University, Tempe, AZ

Motivation

- GPU is the de-facto co-processor for computation-intensive applications
  - Deep learning
  - Image processing
  - Protein folding…

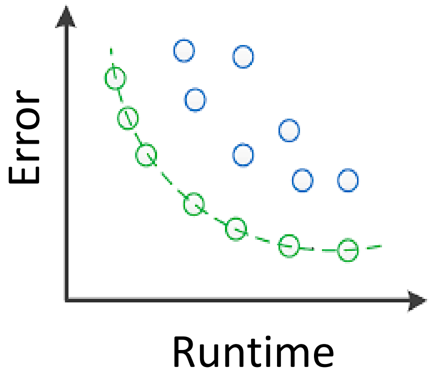
- GPU programs are often poorly optimized
  - Optimization requires both architecture/domain expertise
  - *C++*-like programming interface encourages novice programmers



Approach:
Use Genetic Programming to find optimizations

- GPU programs are usually small, but critical to performance
  - Search space is smaller
  - Any improvement can be significant

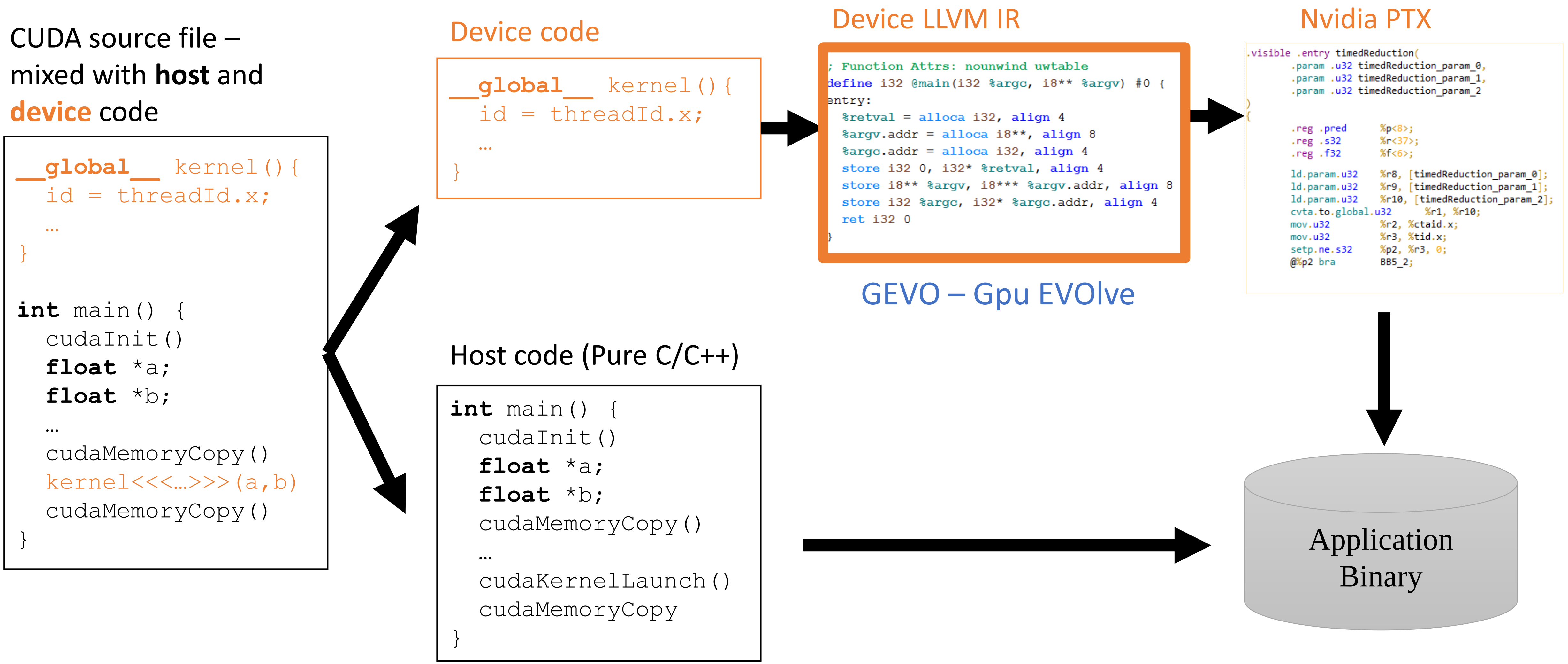
- Many GPU applications are error-tolerant
  - More resilient to the program transformation from GP
  - Error can be co-optimized along with performance (multi-objective)



Outline

- Motivation
- Proposed Design – GEVO
- Experimental Setup
- Result and Analysis
- Conclusion

Compilation flow of GPU programs



CUDA source file – mixed with **host** and **device** code

```
__global__ kernel(){
  id = threadId.x;
  …
}

int main() {
  cudaInit()
  float *a;
  float *b;
  …
  cudaMemoryCopy()
  kernel<<<…>>>(a,b)
  cudaMemoryCopy()
}
```

Device code

```
__global__ kernel(){
  id = threadId.x;
  …
}
```

Host code (Pure C/C++)

```
int main() {
  cudaInit()
  float *a;
  float *b;
  cudaMemoryCopy()
  …
  cudaKernelLaunch()
  cudaMemoryCopy
}
```

Device LLVM IR

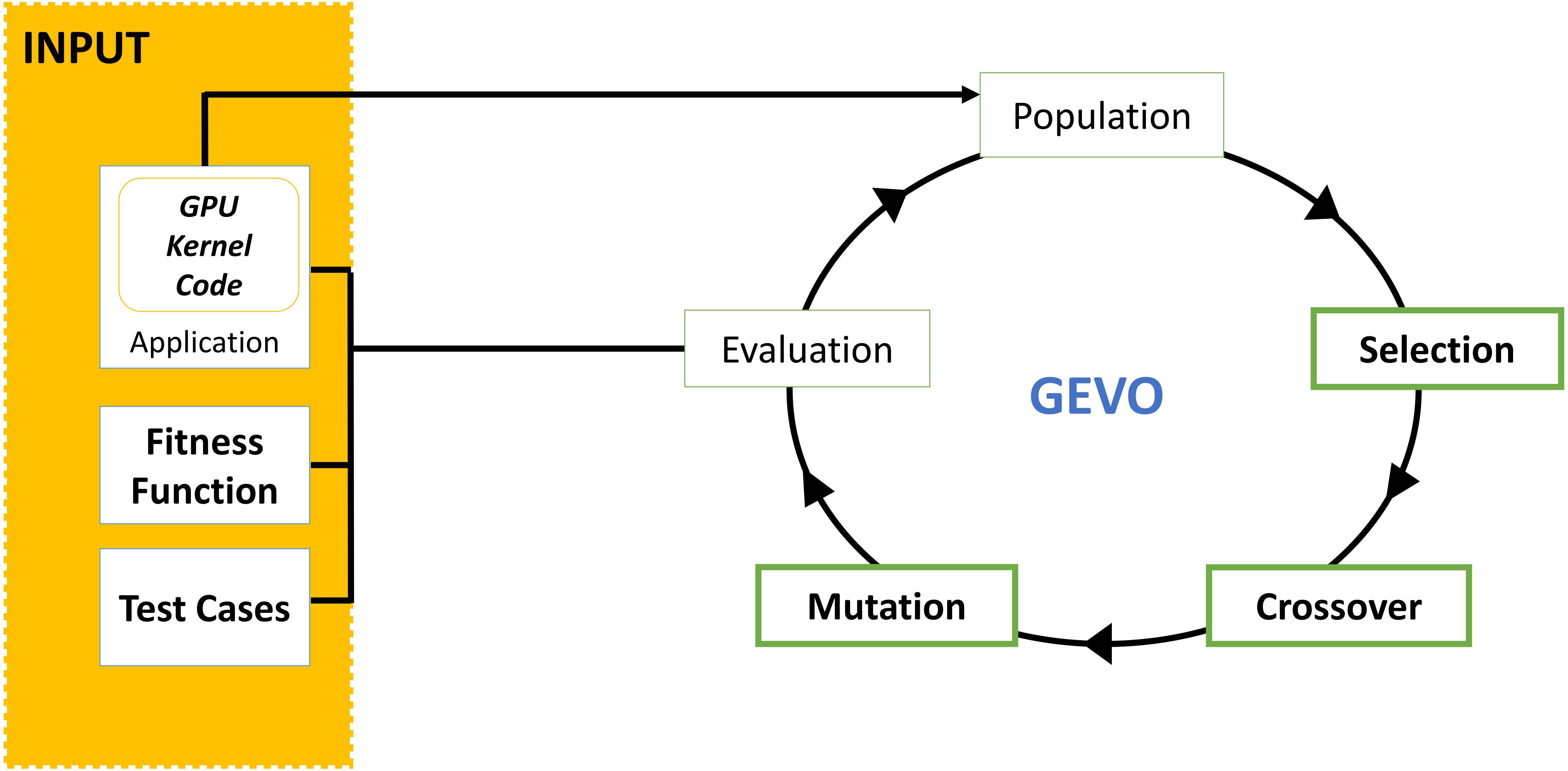
```
; Function Attrs: nounwind uwtable
define i32 @main(i32 %argc, i8** %argv) #0 {
entry:
  %retval = alloca i32, align 4
  %argv.addr = alloca i8**, align 8
  %argc.addr = alloca i32, align 4
  store i32 0, i32* %retval, align 4
  store i8** %argv, i8*** %argv.addr, align 8
  store i32 %argc, i32* %argc.addr, align 4
  ret i32 0
}
```

GEVO – Gpu EVOlve

Nvidia PTX

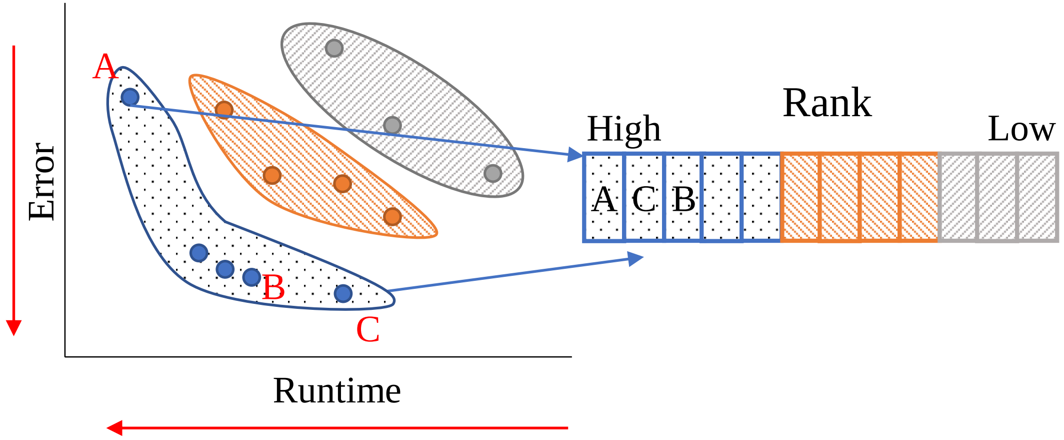
```
.visible .entry timedReduction(
        .param .u32 timedReduction_param_0,
        .param .u32 timedReduction_param_1,
        .param .u32 timedReduction_param_2
)
{

        .reg .pred      %p<8>;
        .reg .s32       %r<37>;
        .reg .f32       %f<6>;

        ld.param.u32    %r8, [timedReduction_param_0];
        ld.param.u32    %r9, [timedReduction_param_1];
        ld.param.u32    %r10, [timedReduction_param_2];
        cvta.to.global.u32      %r1, %r10;
        mov.u32         %r2, %ctaid.x;
        mov.u32         %r3, %tid.x;
        setp.ne.s32     %p2, %r3, 0;
        @%p2 bra        BB5_2;
```

Overview of Gpu EVOlution (GEVO)



Selection

- Multi-objective selection: (runtime, error)
- NSGA-II : Non-dominated Sorting Genetic Algorithm [1]

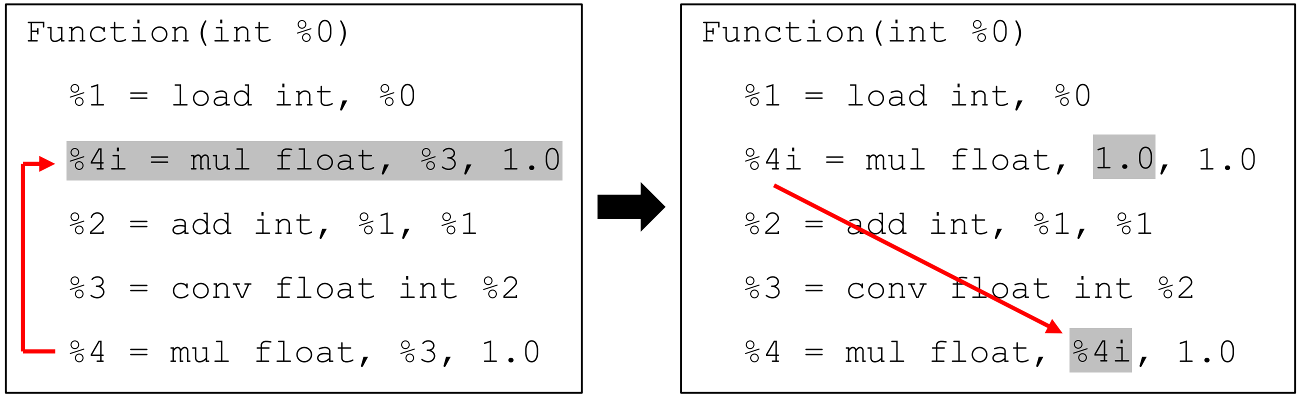


- Combine dominance and crowding distance for ranking

[1] Deb et al., "A fast and elitist multiobjective genetic algorithm: NSGA-II,“IEEE Transactions on Evolutionary Computation, 2002

Mutation

- Copy, delete, move, replace, swap instructions/operands
- Often breaks LLVM syntax: requires repairs



Copy an instruction

```
Function(int %0)
  %1 = load int, %0
  %4i = mul float, %3, 1.0
  %2 = add int, %1, %1
  %3 = conv float int %2
  %4 = mul float, %3, 1.0
```

```
Function(int %0)
  %1 = load int, %0
  %4i = mul float, 1.0, 1.0
  %2 = add int, %1, %1
  %3 = conv float int %2
  %4 = mul float, %4i, 1.0
```

Connect the input

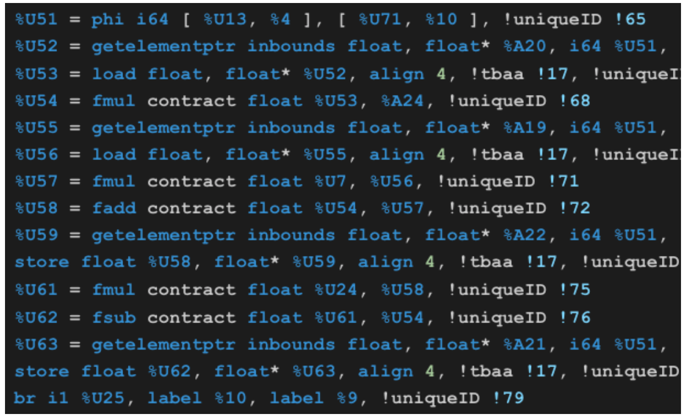
Apply the output

Individual representation

LLVM-IR + Patch(mutation)

Individual

LLVM-IR



```
%U51 = phi i64 [ %U13, %4 ], [ %U71, %10 ], !uniqueID !65
%U52 = getelementptr inbounds float, float* %A20, i64 %U51,
%U53 = load float, float* %U52, align 4, !tbaa !17, !uniqueI
%U54 = fmul contract float %U53, %A24, !uniqueID !68
%U55 = getelementptr inbounds float, float* %A19, i64 %U51,
%U56 = load float, float* %U55, align 4, !tbaa !17, !uniqueI
%U57 = fmul contract float %U7, %U56, !uniqueID !71
%U58 = fadd contract float %U54, %U57, !uniqueID !72
%U59 = getelementptr inbounds float, float* %A22, i64 %U51,
store float %U58, float* %U59, align 4, !tbaa !17, !uniqueID
%U61 = fmul contract float %U24, %U58, !uniqueID !75
%U62 = fsub contract float %U61, %U54, !uniqueID !76
%U63 = getelementptr inbounds float, float* %A21, i64 %U51,
store float %U62, float* %U63, align 4, !tbaa !17, !uniqueID
br i1 %U25, label %10, label %9, !uniqueID !79
```

Patch

Copy 3, 4

Move 9, 3

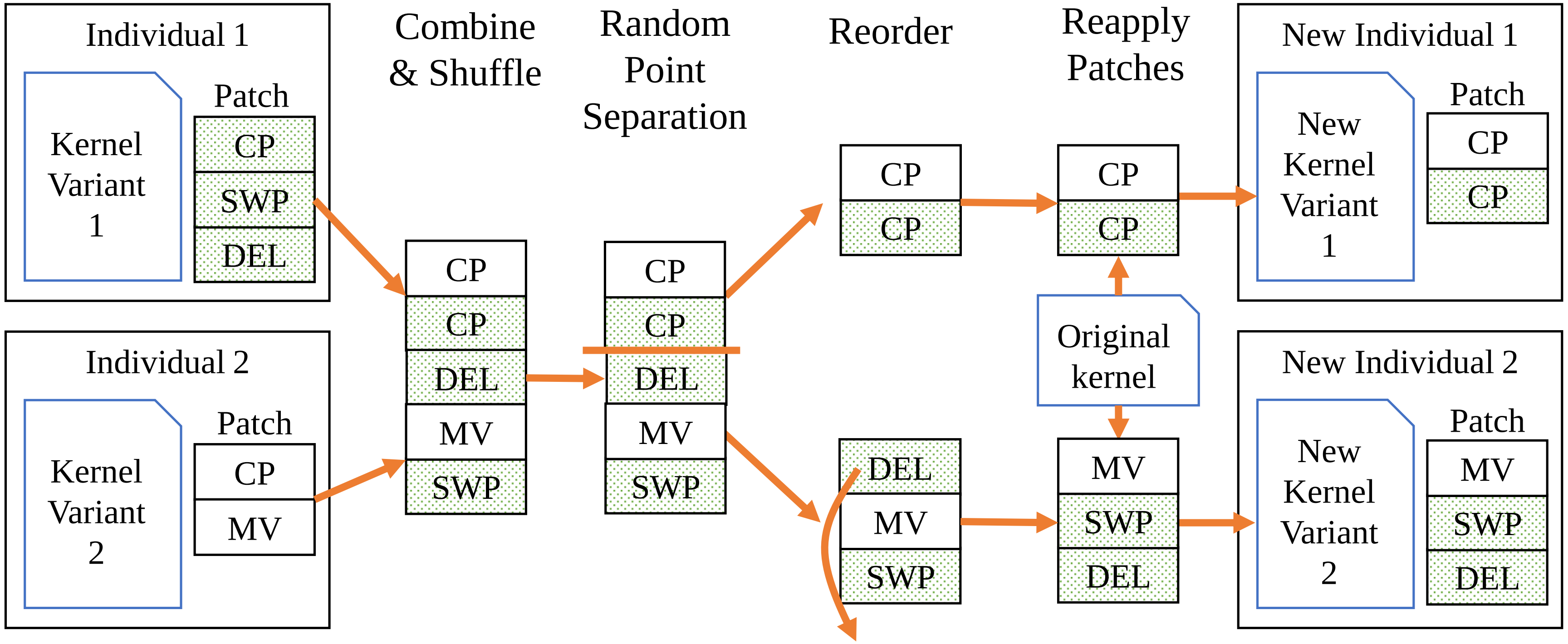
Del 4

Mutation

Crossover

Crossover

- Uses patch-based representation



Outline

- Motivation
- Proposed Design – GEVO
- Experimental Setup
- Results and Analysis
- Conclusion

Experimental Setup

- Platform
  - GPU: Nvidia P100
  - Driver: CUDA 9.2 with Nvidia driver 410
  - CUDA kernel Compiler: Clang/LLVM-8.0
- GEVO Parameters
  - Population size: 256
  - Cross rate: 80%
  - Mutation rate: 30%
  - Search time: 48 hours (20 – 100 generations)

Benchmarks

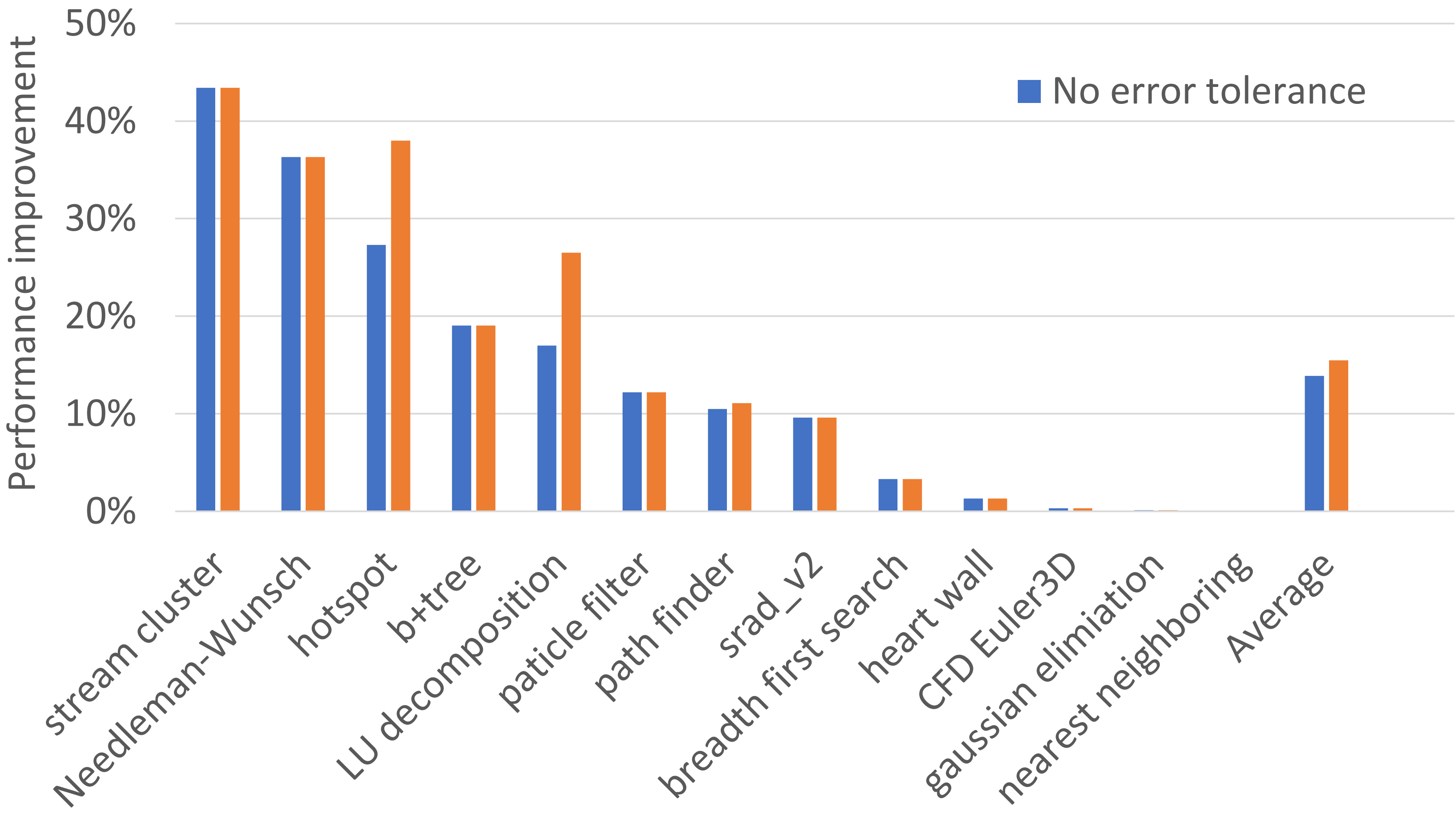
	Applications	Error metric	Test suites	Post-optimization validation
Rodinia benchmark suites [2] **(GPGPU)**	• Bfs • B+tree • … • Particle filter • Stream cluster (13 applications)	Max raw output difference	Built-in data generator	Held-out tests
ML workloads trained using ThunderSVM [3]	• MNIST • a9a	Model training error	Training datasets	• Testing datasets • MNIST large dataset

[2] S. Che et al., "Rodinia: A benchmark suite for heterogeneous computing,“ IISWC 2009
[3] W. Zei et al., “ThunderSVM: A Fast SVM Library on GPUs and CPUs”, JMLS 2018

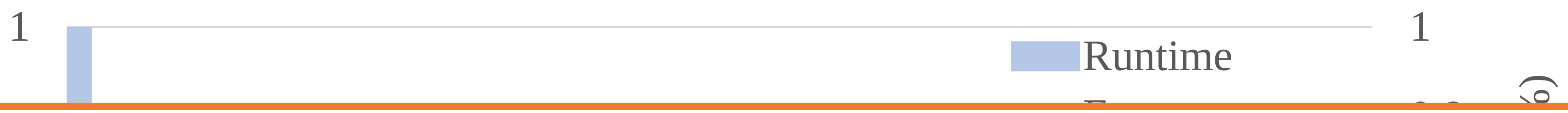
Outline

- Motivation
- Proposed Design – GEVO
- Experimental Setup
- Results and Analysis
  - Rodinia benchmark suite
  - ML workloads trained under ThunderSVM
- Conclusion

GEVO results – Rodinia



Temporal analysis – hotspot (epistasis)



1. Sub-optimal individual can be served as the stepping stone for better optimization combination

2. This implies error tolerance can be used for circumventing and reaching other program spaces.

- Observed 3 key mutations, introducing 0.3 error rate individually, but only incurring 0.1 error rate when combined.

Optimization analysis – remove redundant store (LU decomposition)

(a) Unmodified

```
__shared__ s[BLOCK];
int c = CONST;
int tid = ThreadId.x;
for(i=0; i < 16; i++)
  s[tid] = init(tid);
__syncthread();


for(i=0; i < 16; i++)
  s[tid] = s[tid] - s[i]*s[i];

s[tid] = s[tid] / c;
__syncthread();
```

(b) Post-Compilation

```
__shared__ s[BLOCK];
int c = CONST;
int tid = ThreadId.x;
for(i=0; i < 16; i++)
  s[tid] = init(tid);
__syncthread();

float temp = s[tid];
for(i=0; i < 16; i++) {
  temp = temp - s[i]*s[i];
  s[tid] = temp; }
s[tid] = temp / c;
__syncthread();
```

(c) GEVO Optimized

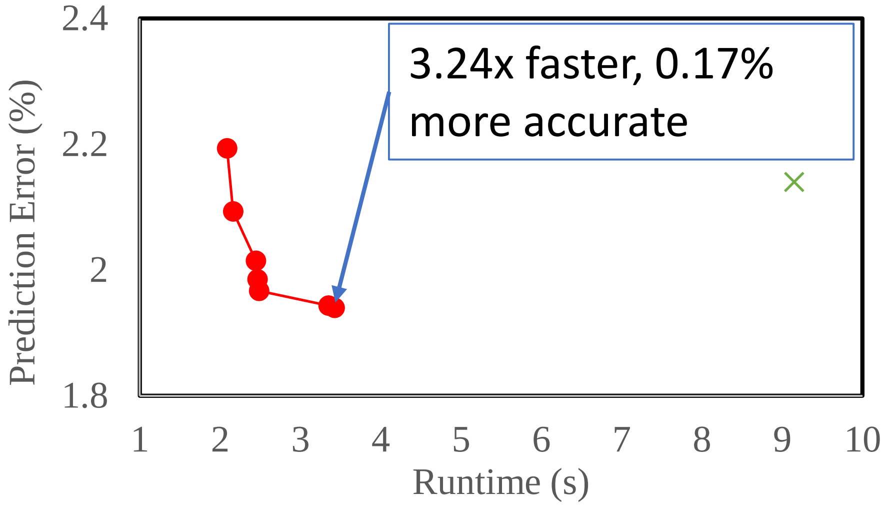
```
__shared__ s[BLOCK];
int c = CONST;
int tid = ThreadId.x;
for(i=0; i < 16; i++)
  s[tid] = init(tid);
__syncthread();

float temp = s[tid];
for(i=0; i < 16; i++)
  temp = temp - s[i]*s[i];
  ~~s[tid] = temp;~~
s[tid] = temp / c;
__syncthread();
```

- Interpretation: The GPU executes the load instruction without waiting for the outstanding store instruction to be finished, and renders the store instruction redundant.

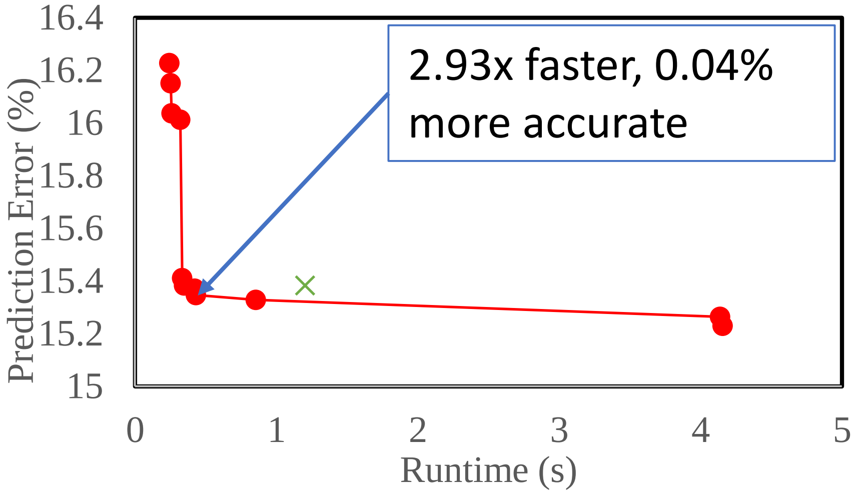
Representative Rodinia optimizations

Architecture-specific	Application-specific
Removing redundant synchronization primitives • Hotspot • LU decomposition • Needleman-Wunch	Removing conditional execution • Hotspot • LU decomposition • Particle filter
Removing redundant stores • LU decomposition	Loop perforation • Stream cluster • LU decomposition • Hotspot
	Memoization • Hotspot

GEVO results – ML workloads in ThunderSVM



MNIST



a9a

- Supersede the baseline in both objectives!
- Same prediction error trend on testing dataset
- 10x training time reduction on the MNIST large dataset (1182 mins to 121 mins)
  - with nearly the same training accuracy (100% to 99.997%)

Optimization analysis – Terminate the loop earlier (MNIST)

```
...
While (1)
  // select f Up
  if (is_I_up(…))
    f_val_reduce[tid] = f;
  up_val = f_val_reduce[…];

  // select f Low
  if (is_I_low(…))
    // f_val_reduce[tid] = -f;
    f_val_reduce[tid] = 1 – f;
  down_val = f_val_reduce[…];

  if (up_val – down_val < epsilon)
    break;
```

- Sequential minimal optimization
  - Iteratively optimizes solution until the progress being slow down.
- GEVO changes the terminal condition, to exit the loop earlier
  - The accuracy isn't affected by this change.
- This might only be applicable for particular type of dataset

Conclusion

- GEVO finds 3 classes of optimization:
  - Architecture-specific
  - Application-specific
  - Dataset-specific
- Machine learning is a promising GEVO target
  - Error tolerant
  - Expensive training times
  - Currently experimenting with deep learning frameworks
- Multi-objective search allows GEVO to find stepping stones to explore larger program space.

Thanks for Yours Attention!

Genetic improvement of GPU code

Jhe-Yu (Jerry) Liou, Stephanie Forrest, Carole-Jean Wu

Computer Science and Engineering

Biodesign institute

Arizona State University, Tempe, AZ

ASU Ira A. Fulton Schools of Engineering

ARIZONA STATE UNIVERSITY

Main loop of GEVO

Initialization

Selection

Crossover &
Mutation

Elitism

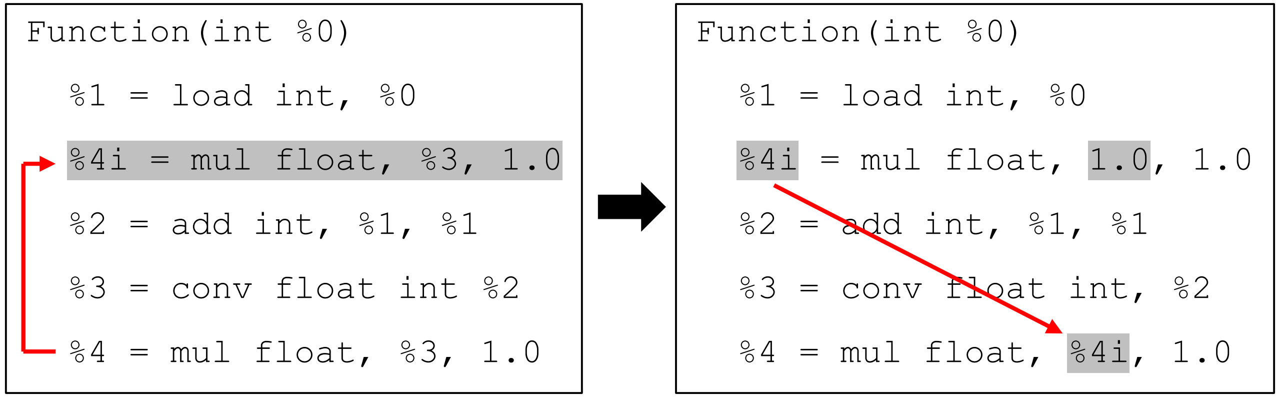
```
pop = Initialization(POP_SIZE, PROGRAM)
for all individual from pop
  Mutate(individual) * 3
rank = NonDominateSorting(pop)

while
  offspring = SelTournment(pop, rank, POP_SIZE)
  elites    = SelBest(pop, rank, POP_SIZE/4)

  for every 2 individuals (ind1,ind2) from Offspring
    if random(0,1) < CROSS_RATE
      Crossover(ind1, ind2)
  for every ind from Offspring
    if random(0,1) < MUTATE_RATE
      Mutate(ind)

  rank = NonDominateSorting(elites + offspring)
  pop = SelBest(elites + offspring, rank, POP_SIZE)
```

POP_SIZE = 256
CROSS_RATE = 0.8
MUTATE_RATE = 0.3

Mutation

- Copy, delete, move, replace, swap instructions/operands
- Often breaks syntax: requires repairs

Copy an instruction



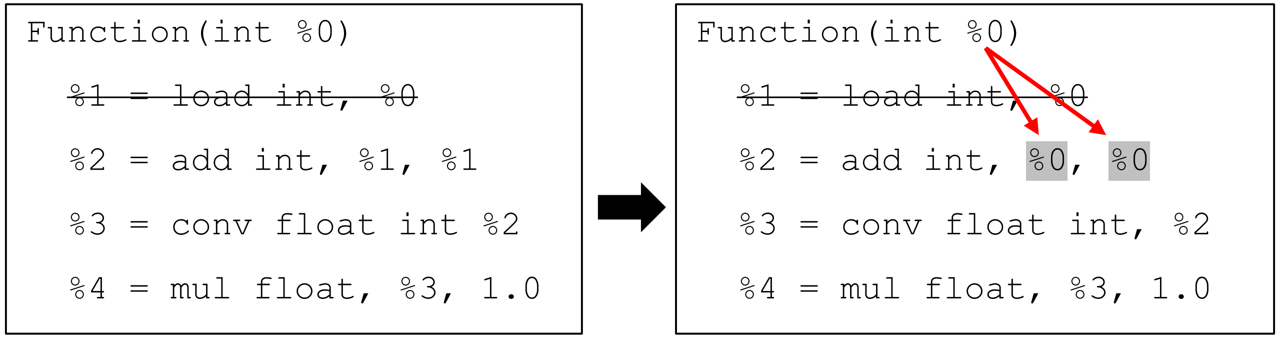
```
Function(int %0)
  %1 = load int, %0
  %4i = mul float, %3, 1.0
  %2 = add int, %1, %1
  %3 = conv float int %2
  %4 = mul float, %3, 1.0
```

```
Function(int %0)
  %1 = load int, %0
  %4i = mul float, 1.0, 1.0
  %2 = add int, %1, %1
  %3 = conv float int, %2
  %4 = mul float, %4i, 1.0
```

Connect the input

Apply the output

delete an instruction



```
Function(int %0)
  ~~%1 = load int, %0~~
  %2 = add int, %1, %1
  %3 = conv float int %2
  %4 = mul float, %3, 1.0
```

```
Function(int %0)
  ~~%1 = load int, %0~~
  %2 = add int, %0, %0
  %3 = conv float int, %2
  %4 = mul float, %3, 1.0
```

Connect the broken dependence chain

Optimization analysis – Removing conditional branch (Particle filter)

- Use inner if statement to exit loop
  - It is guaranteed by the application algorithm

- This single mutation results in 6% speedup over the baseline

```
// CDF and u are both global
// memory with size of N
int tid = ThreadId.x …;

for (x=0; ~~x<N~~; x++) {
  if (CDF[x] >= u[tid]) {
      index = x;
      break;
  }
}
```

Optimization analysis – Removing redundant barrier (Needleman-Wunch)

```
__shared__ int temp[...][...];
__shared__ int ref[...];
int tid = threadId.x;

ref[tid] = referrence[...];
__syncthreads();
temp[tid +1][0] = matrix_cuda[...];
__syncthreads();
temp[0][tid+1] = matrix_cuda[...];
__syncthreads();

for (int i=0; i<BLOCK_SIZE; i++)
  temp[tid][tid] =
    temp[i][0] + temp[0][i] + ref[i];
```

- The 1st and 2nd syncthreads() are not needed