GPUMixer: Performance-Driven Floating-Point Tuning for GPU Scientific Applications

Ignacio Laguna, Paul C. Wood, Ranvijay Singh, Saurabh Bagchi

ISC High Performance 2019

June 17th, 2019, Frankfurt, Germany

Lawrence Livermore National Laboratory

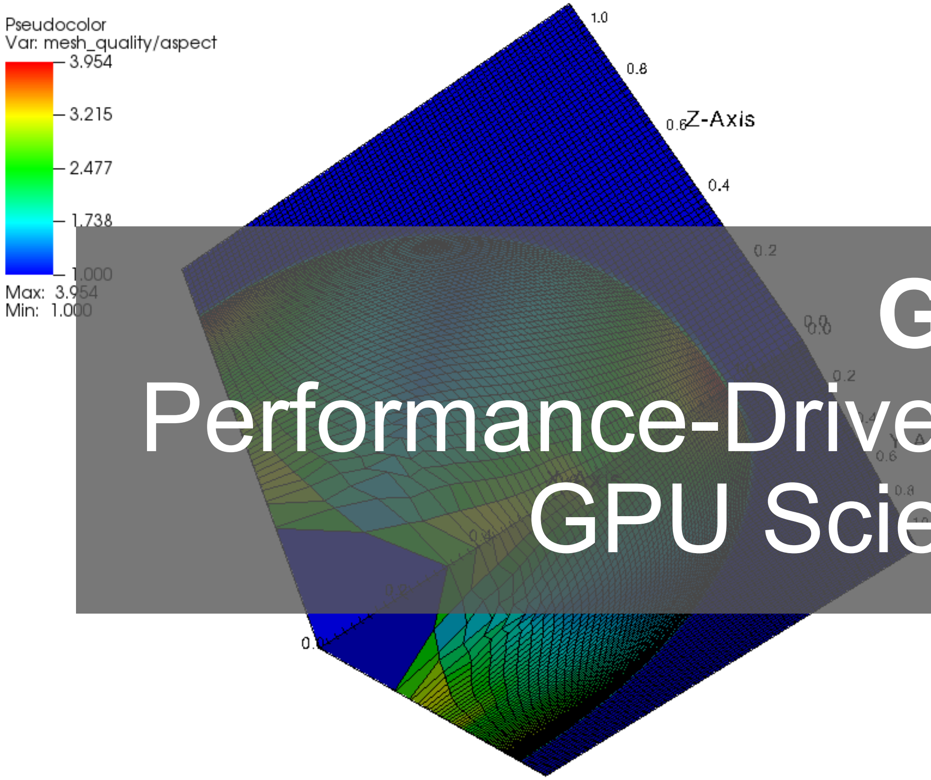
CASC Center for Applied Scientific Computing

Purdue University

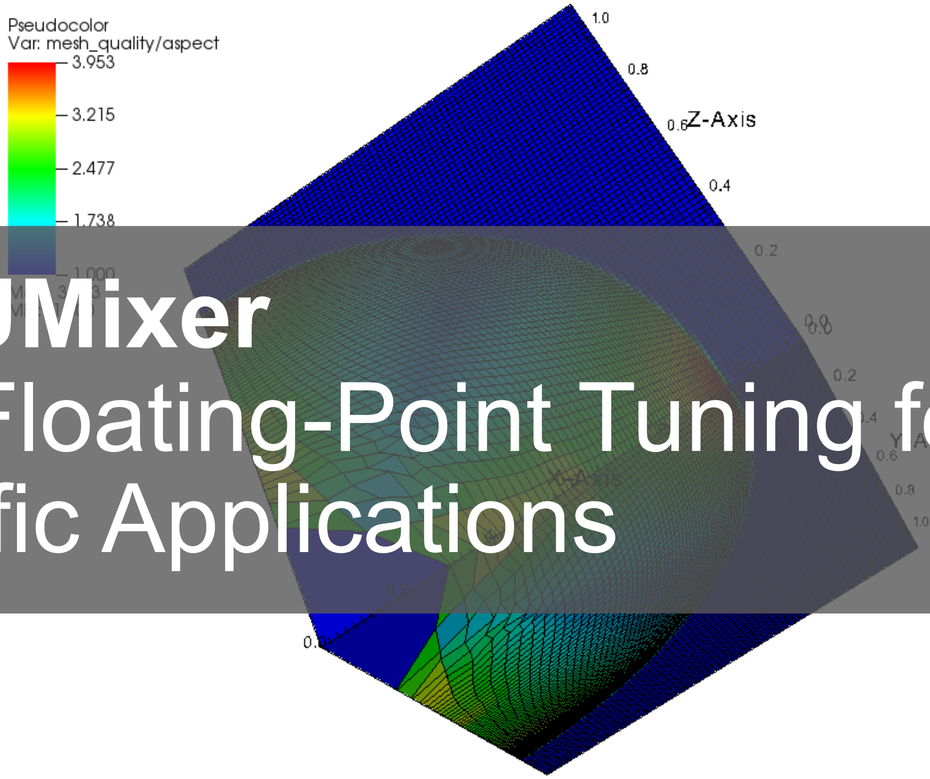
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-PRES-777665)

LULESH

Run 1



Run 2



GPUMixer

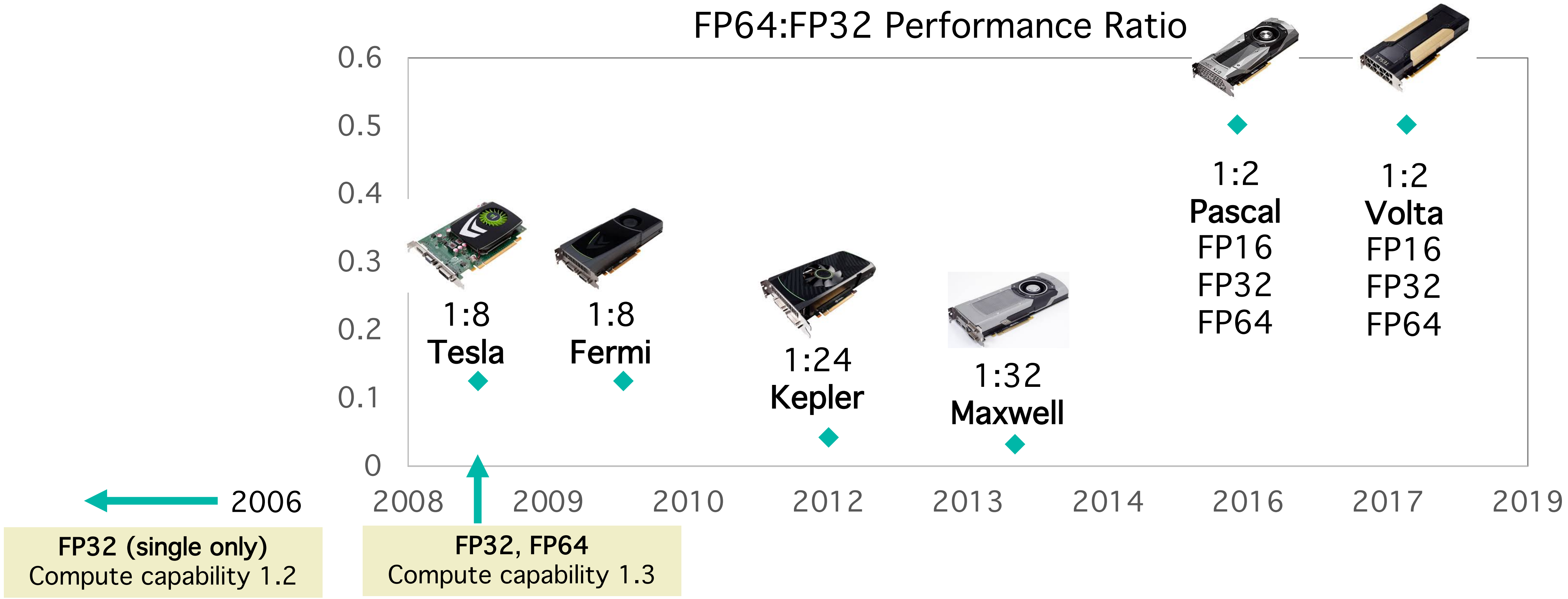
Performance-Driven Floating-Point Tuning for GPU Scientific Applications

FP64 (double precision)

Mixed-Precision (FP64 & FP32)

6 digits of accuracy, **10%** speedup
3 digits of accuracy, **46%** speedup

Floating-Point Precision Levels in NVIDIA GPUs Have Increased

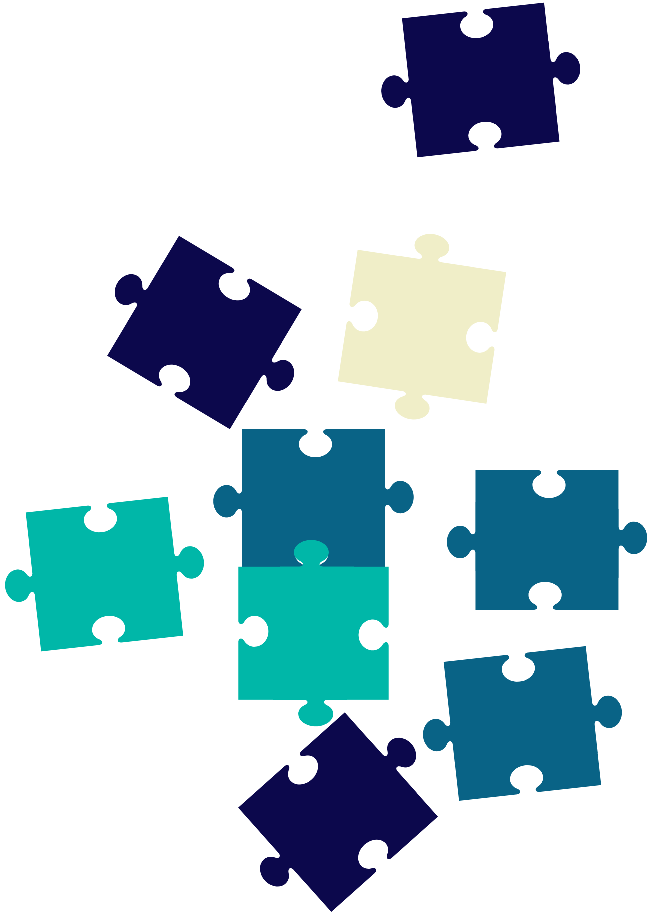


Mixed-Precision Programing is Challenging

- Scientific programs have many variables
- {FP32, FP64} precision: 2^N combinations
- {FP16, FP32, FP64} precision: 3^N combinations

```
double compute(...)
{
  ...
  return r;
}

int main()
{
  double x;
  x = compute() + ...
}
```



Example of Mixed-Precision Tuning

Force computation kernel in **n-body simulation** (CUDA)

```
__global__ void bodyForce(double *x, double *y,
 double *z, double *vx, double *vy, double *vz,
 double dt, int n)
{
 int i = blockDim.x * blockIdx.x + threadIdx.x;
 if (i < n) {
  double Fx=0.0; double Fy=0.0; double Fz=0.0;
  for (int j = 0; j < n; j++) {
   double dx = x[j] - x[i];
   double dy = y[j] - y[i];
   double dz = z[j] - z[i];
   double distSqr = dx*dx + dy*dy + dz*dz + 1e-9;
   double invDist = rsqrt(distSqr);
   double invDist3 = invDist * invDist * invDist;
   Fx += dx*invDist3; Fy += dy*invDist3; Fz += dz*invDist3;
  }
  vx[i] += dt*Fx; vy[i] += dt*Fy; vz[i] += dt*Fz;
 }
}
```

double -> float

Error of particle position (x,y,z)

$$\left|\frac{x-x_0}{x}\right| + \left|\frac{y-y_0}{y}\right| + \left|\frac{z-z_0}{z}\right|$$

(x,y,z): baseline position

(x_0,y_0,z_0): new configuration

Example of Mixed-Precision Tuning (2)

Force computation kernel in **n-body simulation** (CUDA)

```
__global__ void bodyForce(double *x, double *y,
 double *z, double *vx, double *vy, double *vz,
 double dt, int n)
{
 int i = blockDim.x * blockIdx.x + threadIdx.x;
 if (i < n) {
  double Fx=0.0; double Fy=0.0; double Fz=0.0;
  for (int j = 0; j < n; j++) {
   double dx = x[j] - x[i];
   double dy = y[j] - y[i];
   double dz = z[j] - z[i];
   double distSqr = dx*dx + dy*dy + dz*dz + 1e-9;
   double invDist = rsqrt(distSqr);
   double invDist3 = invDist * invDist * invDist;
   Fx += dx*invDist3; Fy += dy*invDist3; Fz += dz*invDist3;
  }
  vx[i] += dt*Fx; vy[i] += dt*Fy; vz[i] += dt*Fz;
 }
}
```

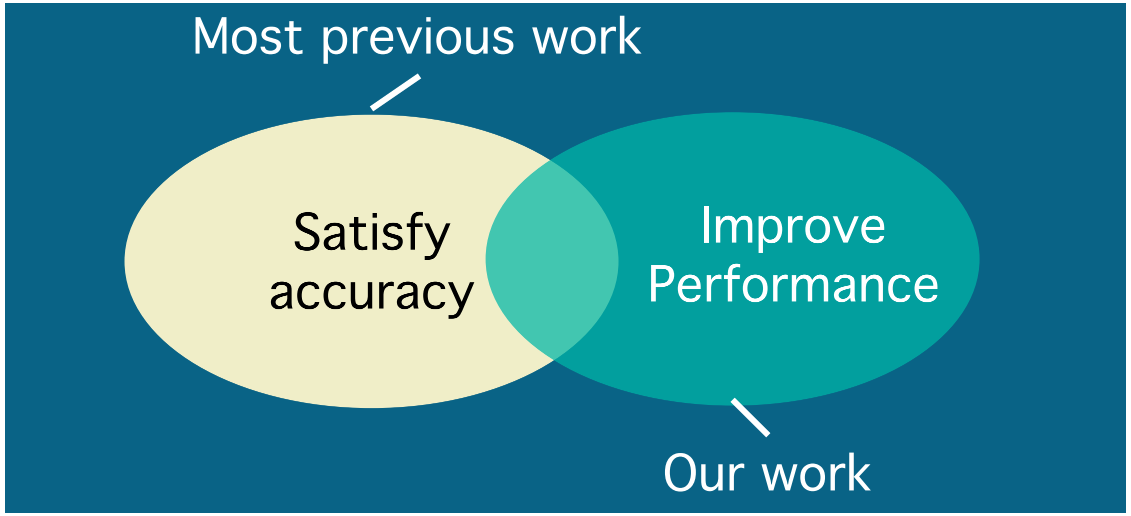
No.	Variables in FP32	Error	Speedup(%)
1	All	15.19	53.70
2	invDist3	4.08	5.78
3	distSqr	1.93	-43.35
4	invDist3, invDist, distSqr	1.80	11.69

Floating-Point Mixed-Precision Configurations

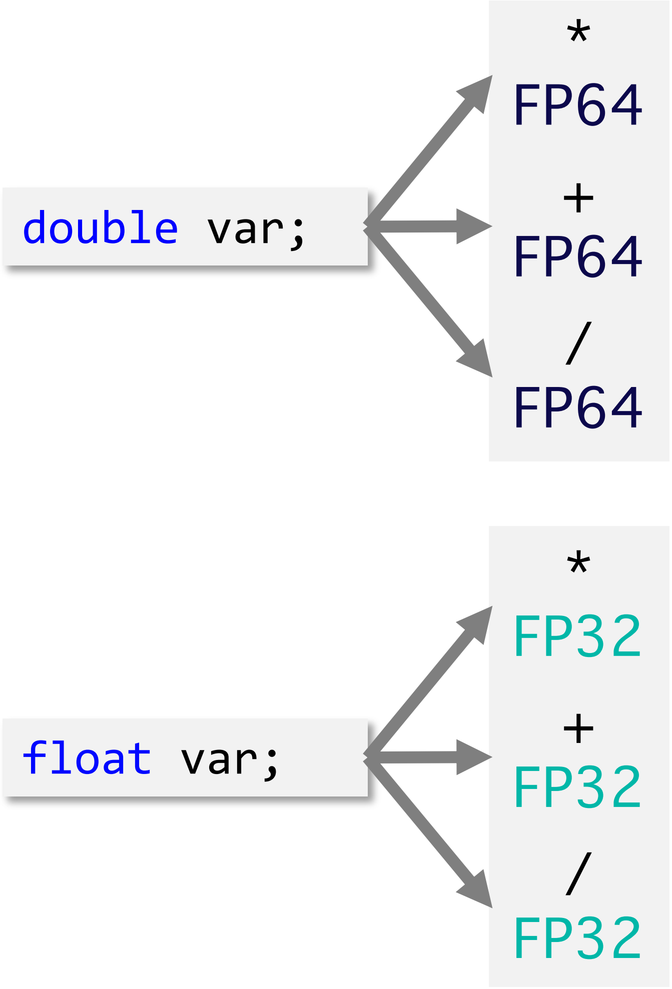
Configuration: set of operations $N = n_1 + n_2$

- N: total program operations
- n_1: precision level 1
- n_2: precision level 2

Config 1	Config 2	Config 3
+, FP64	+, FP32	+, FP32
−, FP64	−, FP64	−, FP64
/, FP64	/, FP64	/, FP32



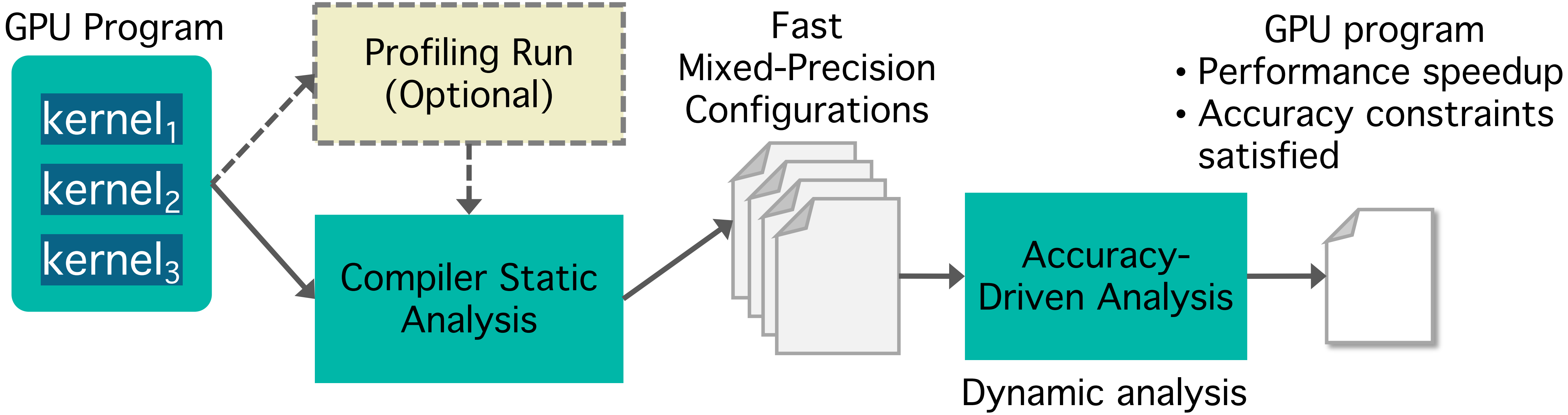
Program Variables are not the Right Level to Define Configurations



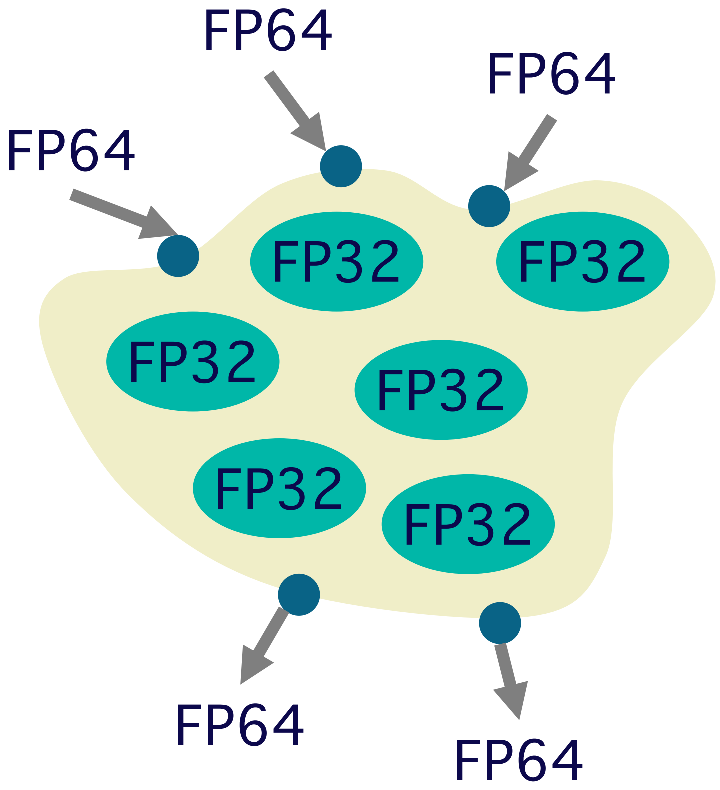
What if the optimal configuration is?

* FP64 + FP32 / FP64	or	* FP32 + FP32 / FP64

GPUMixer Overview



Fast Imprecise Sets (FISets) for Mixed-Precision



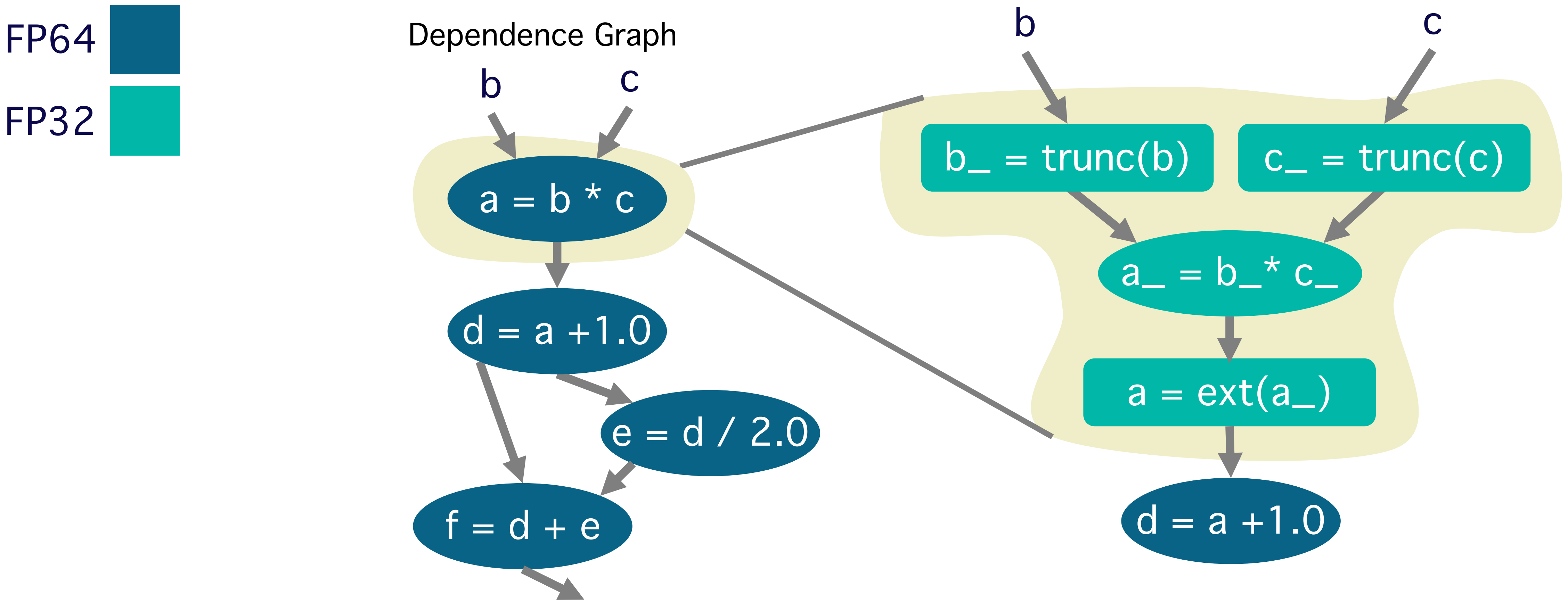
- Type cast operations are costly
- Performance model
  - Arithmetic-to-cast ratio

$$r_{ac} = {}^{O}/_{C}$$

O = arithmetic operations
C = type casting operations

$$r_{ac} \gg 1.0$$

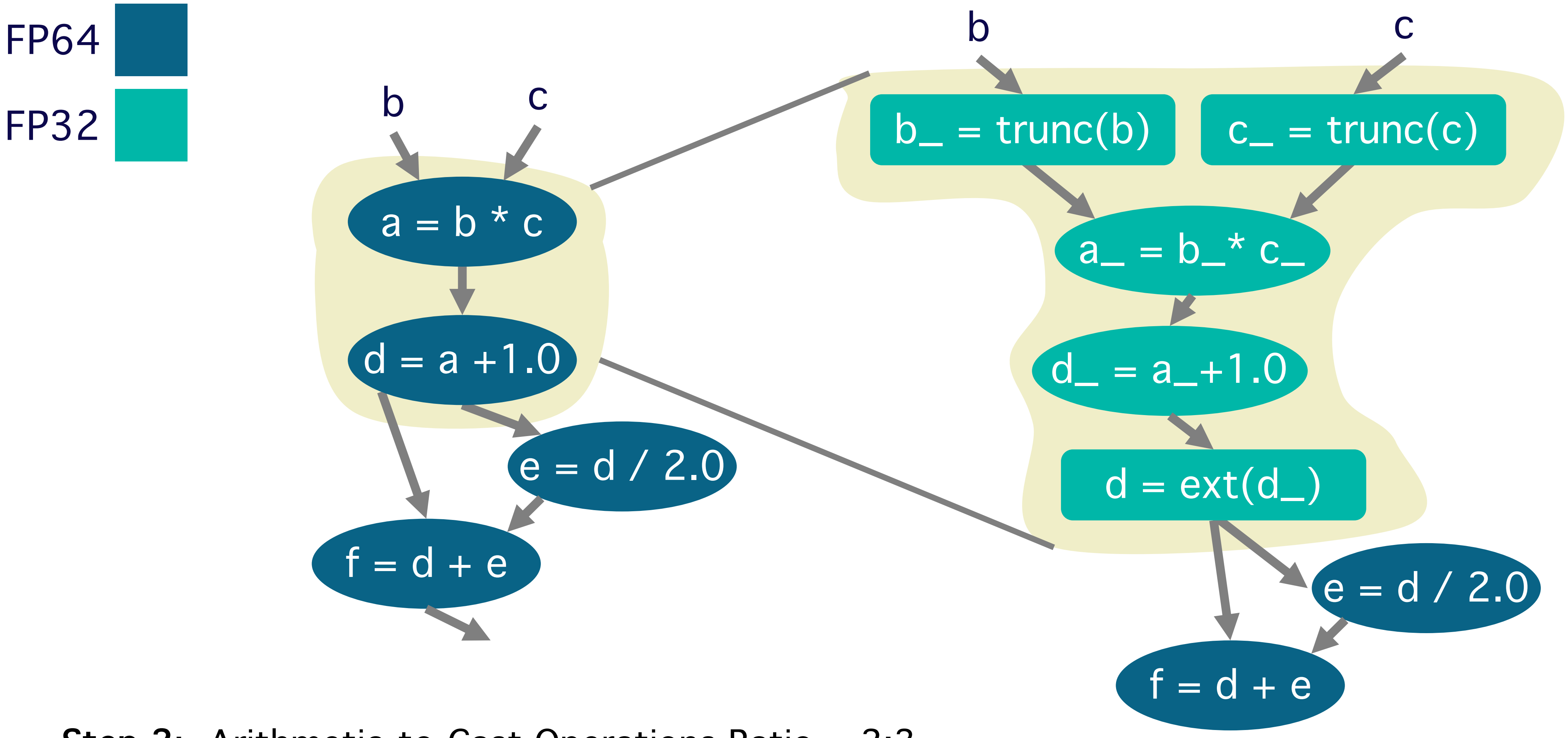
Algorithm for FISet Identification



Step 1: Arithmetic-to-Cast Operations Ratio = 1:3

$$r_{ac} = {}^{O}/_{C} = {}^{1}/_{3}$$ ✗

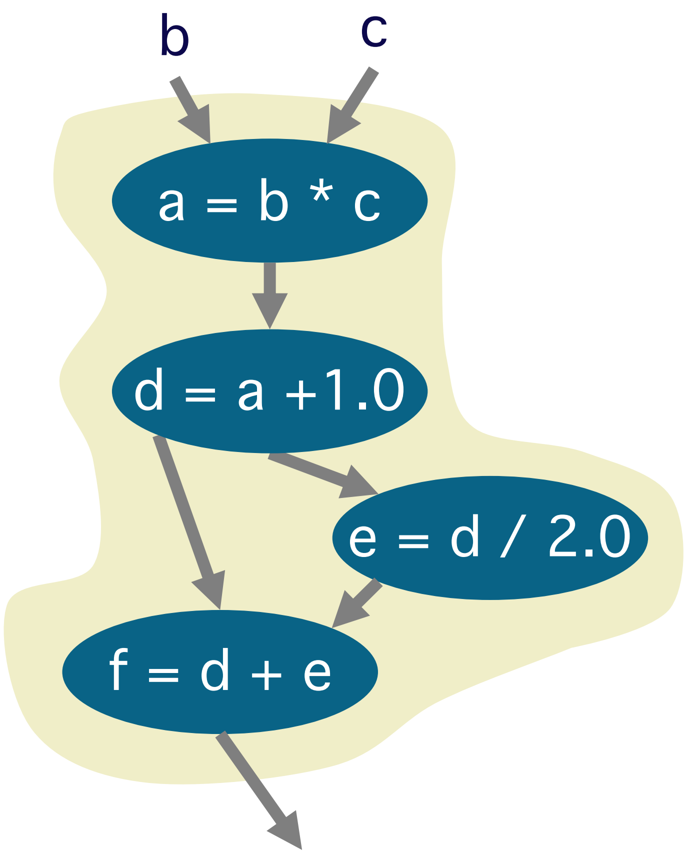
Algorithm for FISet Identification (2)



Step 2: Arithmetic-to-Cast Operations Ratio = 2:3

$$r_{ac} = {}^{O}/_{C} = {}^{2}/_{3}$$

Algorithm for FISet Identification (After N Steps)



- 3 type cast operations
- 4 arithmetic operations

Step N: Arithmetic-to-Cast Operations Ratio = 4:3

$$r_{ac} = {}^{O}/_{C} = {}^{4}/_{3} > 1$$

More details of the algorithm in the paper

Calculating FISets in Loops

- Model: $L_0 > L_1 > L_2 > \ldots$
  - L_0 encloses L_1, L_1 encloses L_2, …
- **Case 1**: all nodes of the FISet are in the same L_x
  - No special treatment

Case 2
$L_x > L_y$
L_x : Arithmetic operations
L_y: Casting operations

```
Loop Lx(...)
{
 a*b
  Loop Ly(...)
  {
    Cast
  }
}
```

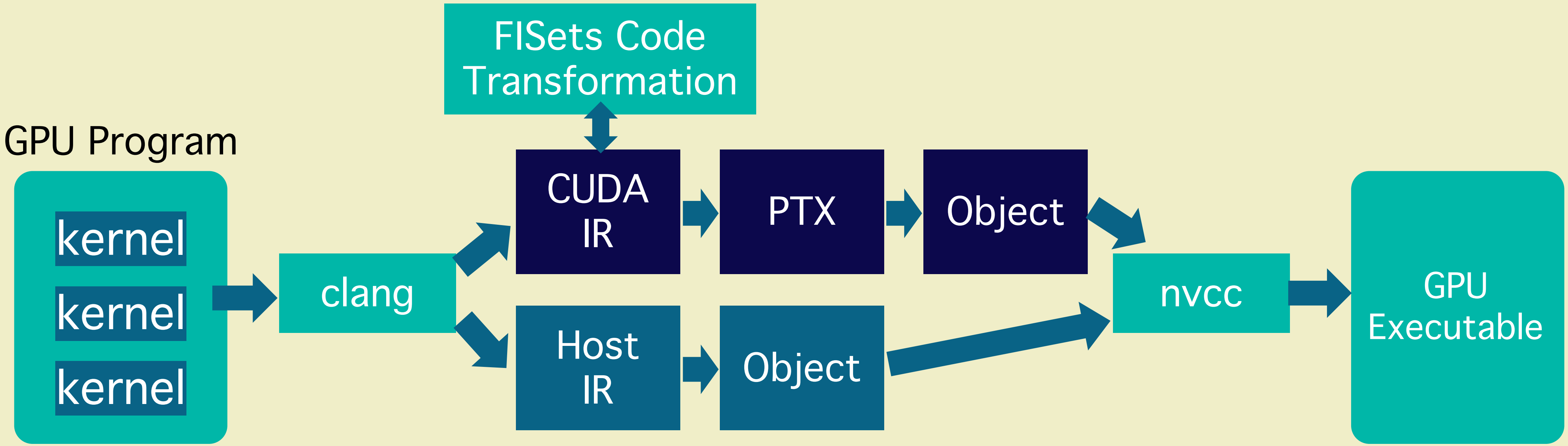
Arithmetic operations will executed equal or more times than casting

Case 3
$L_x > L_y$
L_x : Casting operations
L_y: Arithmetic operations

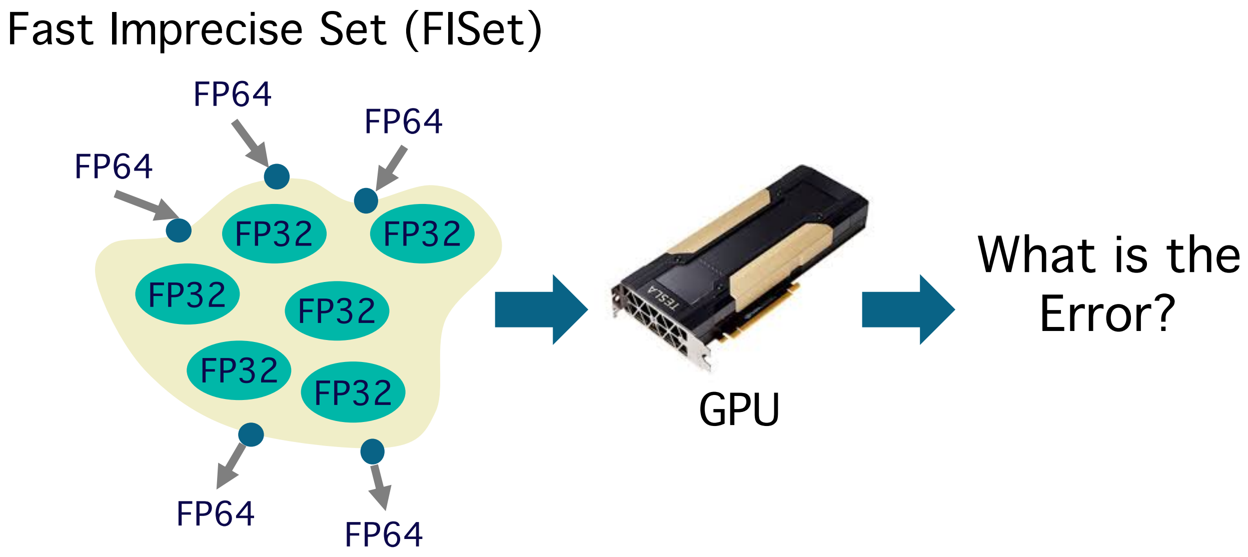
```
Loop Lx(...)
{
  Cast
  Loop Ly(...)
  {
    a*b
  }
}
```

Casting operations may be executed more times than arithmetic operations

Compilation Process is Based on Clang/LLVM



Shadow Computations are Used to Calculate the Error Introduced in a FISet



- FISets introduce error
- Techniques exist on serial code
- Could not use any on CUDA
- Our method:
  - **Shadow computations**
  - Used before in serial code

Overview of Shadow Computations

Program Execution

CUDA Kernel	Shadow Operations	Relative Error
FISet 1		
(FP64) a = x + y	a_ = x_ + y_ (FP32)	$error_a$
(FP64) b = a * z	b_ = a_ * z_ (FP32)	$error_b$
…	…	…

Total Error
(for FISet 1)

$$error = \left|\frac{v_{FP64} - v_{FP32}}{v_{FP64}}\right|$$

v_{FP64} : result of FP64 operation
v_{FP32} : result of FP32 operation

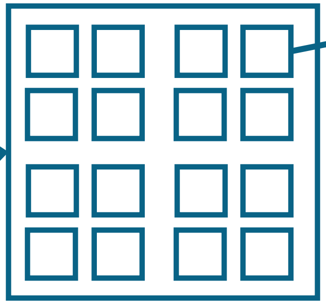
Our CUDA Runtime System Keeps Track of the Per-Thread Error

```
main() {

  kernel1<<<N,M>>>();
  kernel2<<<N,M>>>();
  kernel3<<<N,M>>>();

}
```

GPU Kernel



Runtime System

- Keep global-memory data structure
  `total_error [INST][THREADS]`
  `INST``: number of static instructions`
  `THREADS``: kernel threads`
- No synchronization needed
- Single location may be executed several times
  - Every error is aggregated
- **Total error** aggregated at the end of kernel

Trial Runs are Sorted by FISet Total Error (Default Search)

Program Configurations (1 for each FISet)	Total Error
FISet a	0.00012
FISet b	0.00448
FISet c	0.00619
…	…

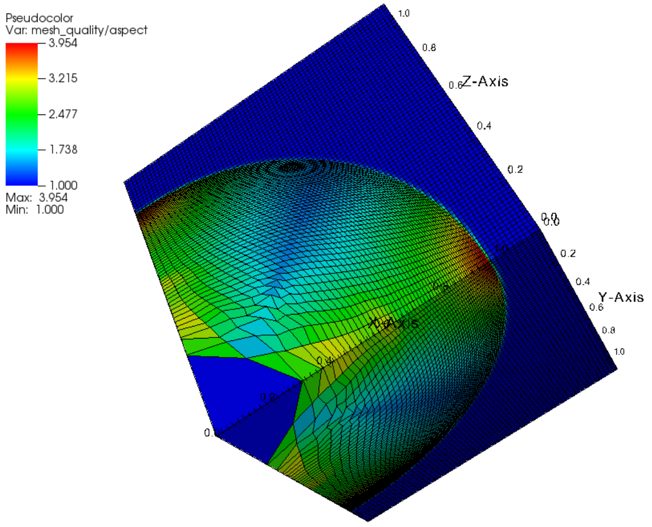
Run Program

Check if output satisfies accuracy constraints

Evaluation

- Comparison approach: **Precimonious** [Rubio-Gonzalez et al. SC'13]
  - Uses generic search algorithm: *delta-debugging*
  - Original version doesn't consider parallel code
  - Implemented our own version for CUDA (called Precimonious-GPU)

- Three CUDA programs: LULESH, CoMD, CFD (Rodinia)



- LLNL system
  - NVIDIA Tesla P100 GPUs
  - Clang 4.0
  - CUDA 8.0

Three Modes of Operation

Example of 3 digits of accuracy
FP64: **3.14**15
Mixed-Precision: **3.14**79

1

- User specifies **accuracy threshold**
- Search based in FISet total error

2

- User specifies **accuracy threshold** and **performance speedup**
- Accuracy has priority
- Search like in mode 1
- Ends when both constraints are satisfied

3

- User specifies **accuracy threshold** and **performance speedup**
- Performance has priority
- Search based on the ratio r_{ac} (start with the largest ratio)
- Ends when both constraints are satisfied

Overhead of Shadow Computations and Threshold Settings

- Overhead of shadow computation analysis: **24× average**
  - LULESH: 61×
  - CoMD: 1.5×
  - CFD: 11 ×
  - It is run only once for a given input
- Accuracy levels: 3, 6, 9 digits
- Performance speedups levels: 5%, 10%, 15%, 20%

GPUMixer Results – Performance Speedup

No. of trial runs

	Error Thold. (digits)	Mode 1	Mode 2 Performance Threshold 5%	10%	15%	20%	Mode 3 Performance Threshold 5%	10%	15%	20%
LULESH	3	9.8% (1)	9.8% (1)	30.4% (2)	30.4% (2)	$\boxed{30.4\%\ (2)}$	46.4% (1)	46.4% (1)	46.4% (1)	$\boxed{46.4\%\ (1)}$
	6	0.3% (12)	8.4% (79)	–	–	–	–	–	–	–
	9	0.3% (12)	–	–	–	–	–	–	–	–
CoMD	3	24.2% (1)	24.2% (1)	24.2% (1)	24.2% (1)	24.2% (1)	10.9% (1)	10.9% (1)	37.5% (7)	37.5% (7)
	6	24.2% (1)	24.2% (1)	24.2% (1)	24.2% (1)	24.2% (1)	10.9% (1)	10.9% (1)	37.5% (7)	$\boxed{37.5\%\ (7)}$
	9	2.3% (3)	19.7% (62)	19.7% (62)	19.7% (62)	–	19.3% (8)	19.3% (8)	19.3% (8)	–
CFD	3	8.3% (1)	8.3% (1)	13.3% (3)	$\boxed{15.3\%\ (35)}$	–	5.1% (9)	12.6% (15)	15.1% (39)	–
	6	8.34% (1)	8.3% (1)	13.3% (3)	15.3% (35)	–	5.1% (9)	12.6% (15)	$\boxed{15.1\%\ (39)}$	–
	9	–	–	–	–	–	–	–	–	–

- We can find good configurations only in a few runs (1-3 runs)
- Mode 1 takes only a few runs (but can't find very good cases)

- Mode 2 finds configurations with better performance than mode 1

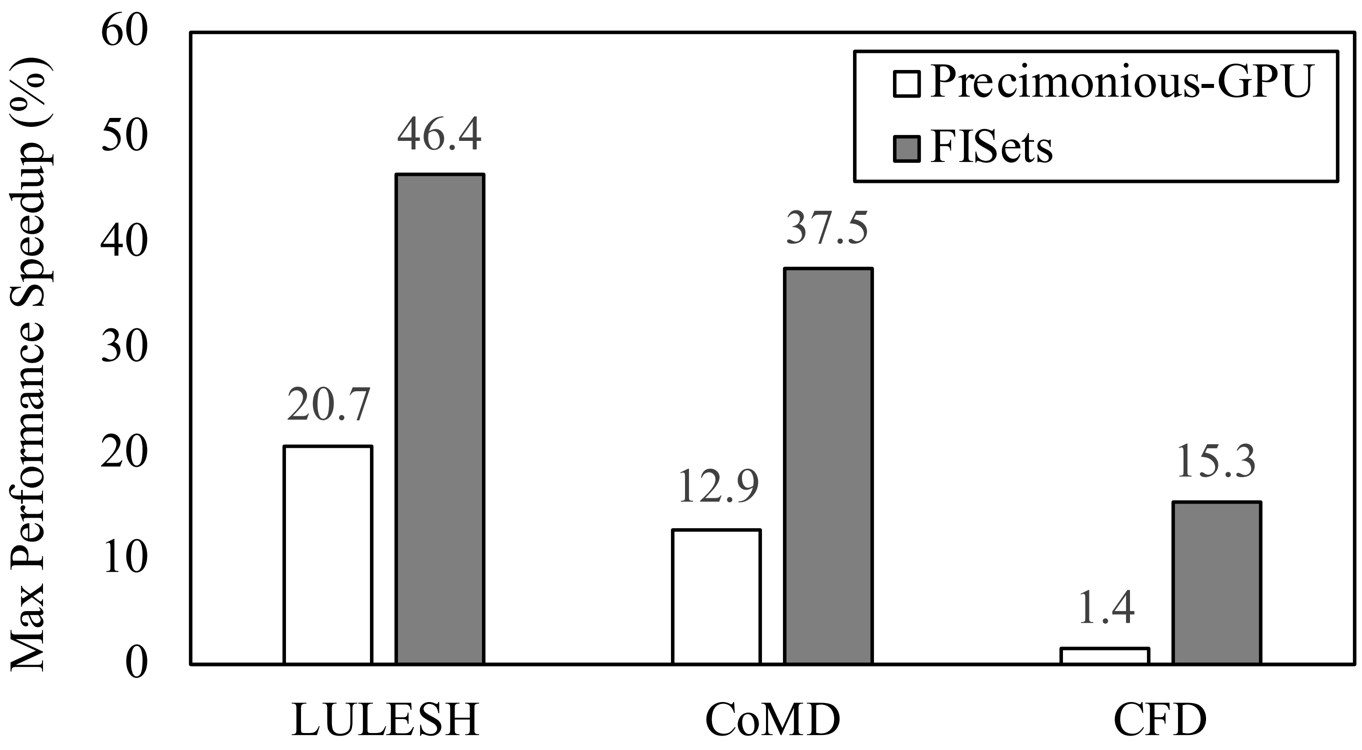
- Highest performance improvements are found with mode 3

Precimonious-GPU Results

	Error Thold. (digits)	Performance Threshold			
		5%	10%	15%	20%
LULESH	3	11.6% (11)	11.4% (11)	17.4% (32)	20.7% (34)
	6	11.5% (11)	11.4 (11)	–	–
	9	–	–	–	–
CoMD	3	12.6% (2)	12.9% (2)	–	–
	6	13.6% (2)	12.7% (2)	–	–
	9	5.4% (24)	–	–	–
CFD	3	–	–	–	–
	6	–	–	–	–
	9	–	–	–	–

- Useful in finding configurations quickly
- It didn't find configurations with higher speedup than GPUMixer

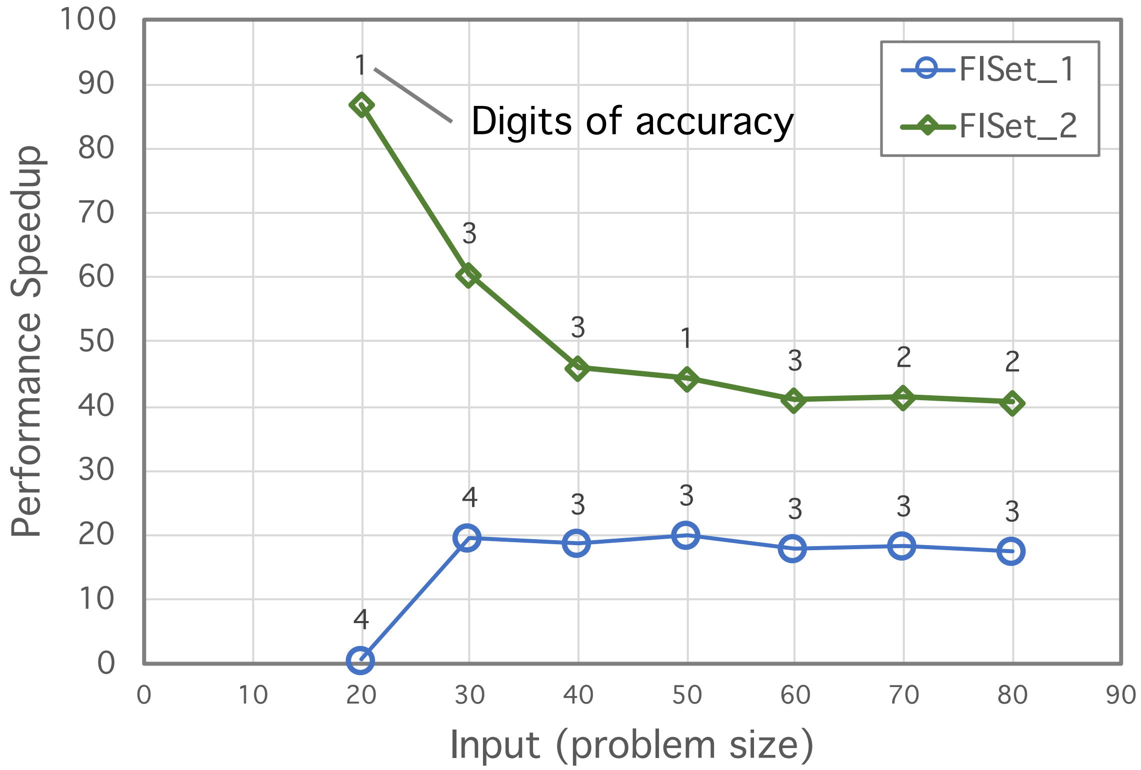
Comparison of Maximum Performance Speedup Between GPUMixer and Precimonious-GPU



- GPUMixer can find configurations with up to 46% of ideal speedup
  - Versus 20.7% for Precimonious-GPU

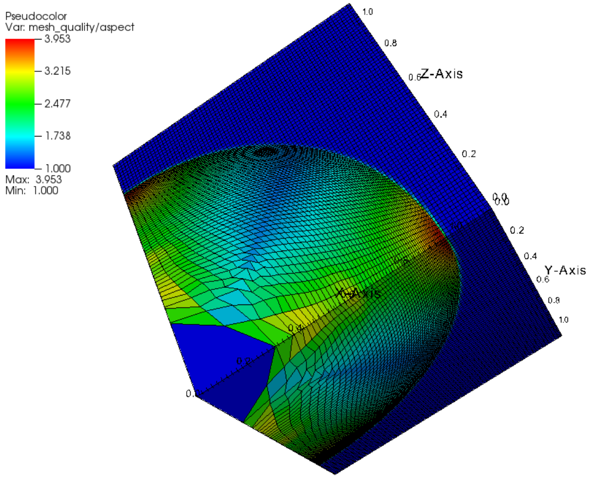
Input Sensitivity Analysis in LULESH

- Two FISet configurations
  - FISet 1, low $r_{ac} = 2.08$
  - FISet 2, high $r_{ac} = 6.90$
- Speedups stabilize for large inputs
- FISet 1 has higher digits of accuracy
  - FISet 1 has fewer FP32 operations



In Summary

Mixed-Precision
10%-46% speedup



1 Automatic mixed-precision tuning can improve performance in GPU applications

2 We present the first framework for automatic FP tuning in GPUs; **we focus on performance improvements**

4 FISets can be found via static analysis; can be implemented in a compiler

3 GPUMixer gets performance improvements of up to 46% of ideal speedup (20% only in state-of-the art)

Thank you for the nomination!

Lawrence Livermore National Laboratory