Automatic Mixed Precision Floating Point Tuning for GPU Applications

Pradeep Venkata Kotipalli[1], Ranvijay Singh[1], Paul Wood[2], Ignacio Laguna[3], **Saurabh Bagchi**[1]

1: Purdue University; 2: Johns Hopkins University; 3: Lawrence Livermore National Laboratory

US Department of Energy contract DEAC52-07NA27344

Motivation

- Mixed precision computing is an effective solution to increase performance and reduce computation energy of applications with intense floating point operations
- Throughput ratio of FP64: FP32: FP16 in modern GPUs is 1:2:4
- Scientific and Machine Learning applications rely heavily on GPUs for heavy floating point computations
- Balancing the trade off between mixed precision application level error and performance is challenging

Problem Statement

- Identify the *optimal* precision vector (PV), the precision levels of all the floating point variables in the program
- Tradeoff between error and runtime
  - All variables tuned to the lowest precision level leads to the greatest improvement in performance but results in a high error
  - All variables tuned to the highest precision level results in zero error but is performance inefficient
- Given an error tolerance margin, the task is to find the PV that gives the fastest runtime while staying below the error tolerance

Beware of Casting Cost

```
float64 x;
float64 y;

…
z = x+y;
…
```

(A)

```
float32 x;
float32 y;

…
z = x+y;
…
```

(B)

```
float32 x;
float64 y;

…
z = x+y;
…
```

(C)

Say Error(B) > Tolerable threshold

- Runtime: C > A > B
- C is the most expensive due to automatic casting up of *x*
- In our GPU architecture:
  - FP64 add = 2 (time) units, FP32 add = 1 unit, Casting = 4 units

Our Contributions

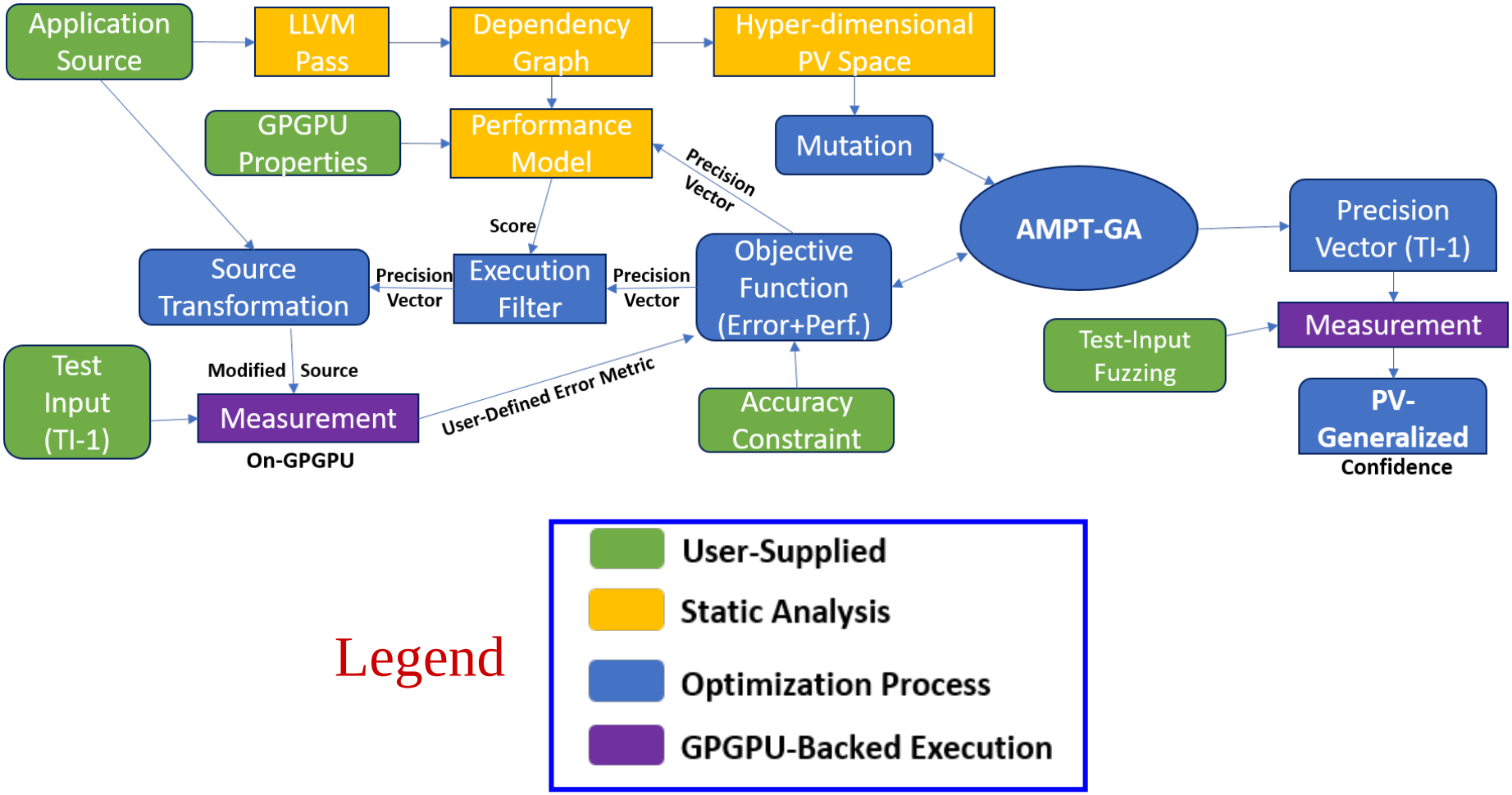
1. Developed a *casting-aware* PV search technique
2. Developed a static performance model which estimates the performance of a PV relative to other PVs *without* actually executing the program
3. Deployed a Genetic Algorithm (GA) for optimal PV search in discontinuous space and showed that it can tolerate input data dependency
4. Showed that our technique benefits a complex DOE application, LULESH

Shortcomings of Prior Work

- They do not support parallel GPU codes as they rely on serial instrumentation or profiling support that does not cover the GPU programming and execution model
- Accuracy-based approaches do not account for the casting effects on performance and tend to increase the runtime
- The previous state-of-the-art **Precimonious**[1] uses delta debugging technique leading to local minima
- They do not use static analysis to aid runtime search
- Some use hard-to-scale local accuracy models[2]

[1] Cindy Rubio-González, C. Nguyen, H. Nguyen, J. Demmel, W. Kahan, K. Sen, D.H. Bailey, C. Iancu, D. Hough. "Precimonious: Tuning assistant for floating-point precision." In *Supercomputing* 2013.
[2] Wei-Fan Chiang, Mark Baranowski, Ian Briggs, Alexey Solovyev, Ganesh Gopalakrishnan, and Zvonimir Rakamarić, "Rigorous Floating-point Mixedprecision Tuning," POPL 2017.

Our Solution Approach: AMPT-GA



Objective Function Formulation

- The objective function used in GA:

$$\theta = perf(PV) + (error(PV) - T) \cdot K$$

where T is the error threshold and constant K >> max perf(PV) if error > T, else K = 1

- Choose PV such that θ is minimized
- We depend on actual executions to evaluate the error of each PV
- However, we reduce the number of executions through the execution filter backed up by the performance model

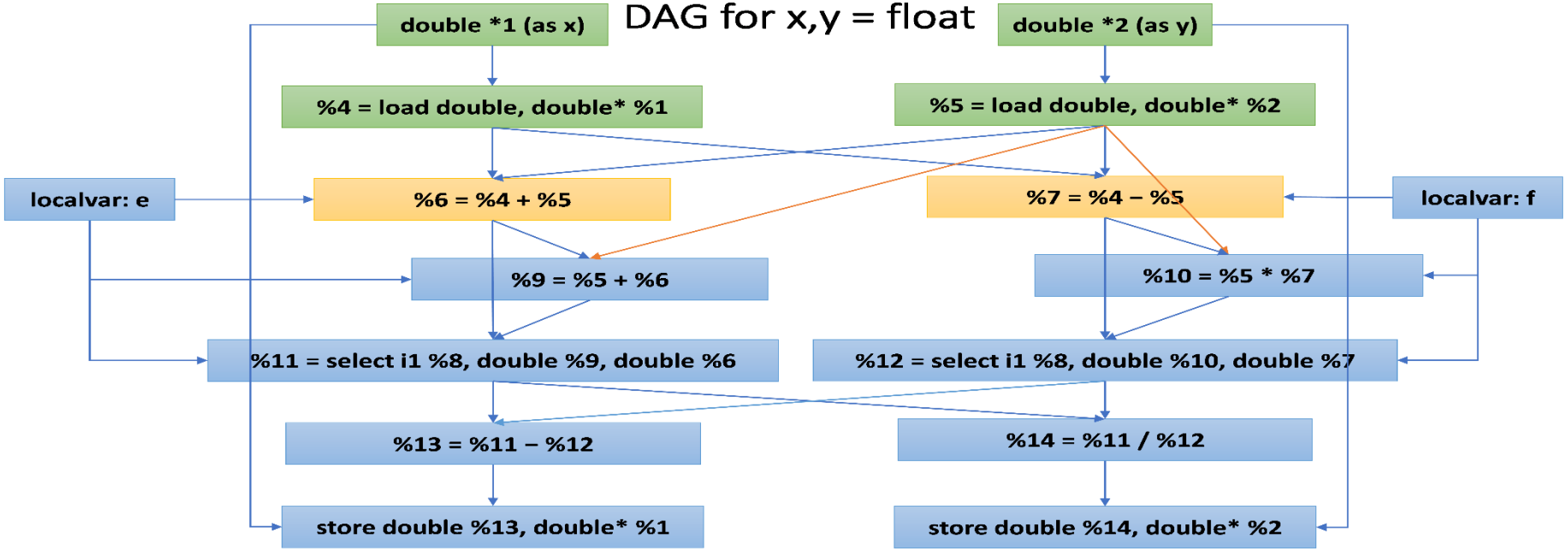
- Casting-aware GA search: Mutation function is modified to change some sets of variables together to avoid casting penalties

Static Analysis for Performance Model

- Static performance model and IR dependency graph estimate performance gain from a to-be-searched PV, ***without program execution***
- Takes a PV and kernel source code and estimates #floating point and cast operations

```
void func4(int n, double &x, double &y)
{
  double e = x + y;
  double f = x - y;
  if (n < 42)
  {
    e += y;
    f *= y;
  }
  x = e - f;
  y = e / f;
}
```

Legend
- float value
- double value
- cast value
- operation result is same type
- operation result is cast



Evaluation Plan

- Evaluation is meant to answer questions:

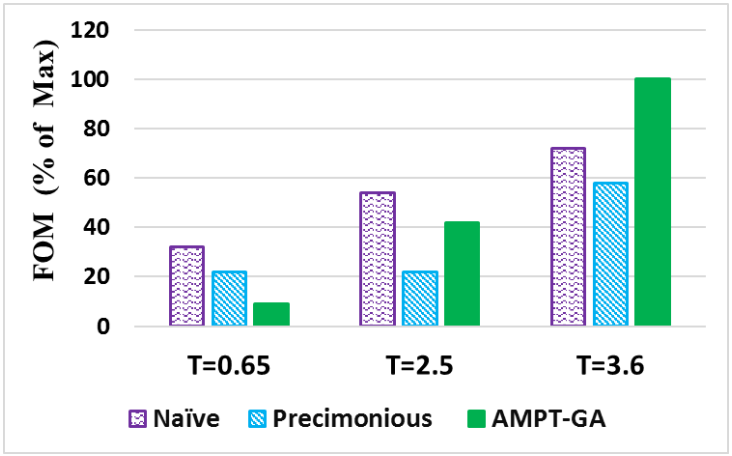
1. Does the selected Precision Vector give the lowest runtime?
2. How many program executions are needed to decide optimal PV?
3. What is the efficiency of the approach? (Combines #1 and #2)

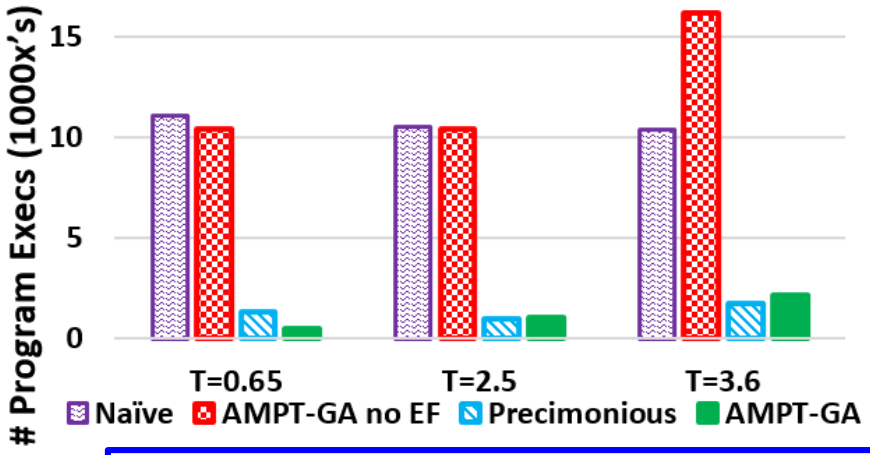
- Target applications

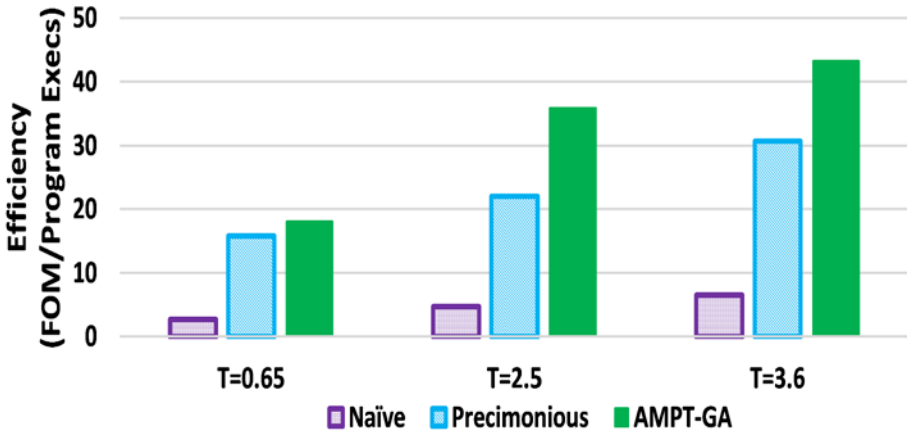
1. LULESH: A proxy app by the US DOE for benchmarking large-scale clusters because it is representative of many large codes. It models hydrodynamics equations, which describe motion of materials when subject to forces
2. Three programs from the Rodinia GPU benchmark suite: LavaMD, Backprop and CFD.

- Evaluation done on NVIDIA Tesla P100 GPU

Evaluation on LULESH



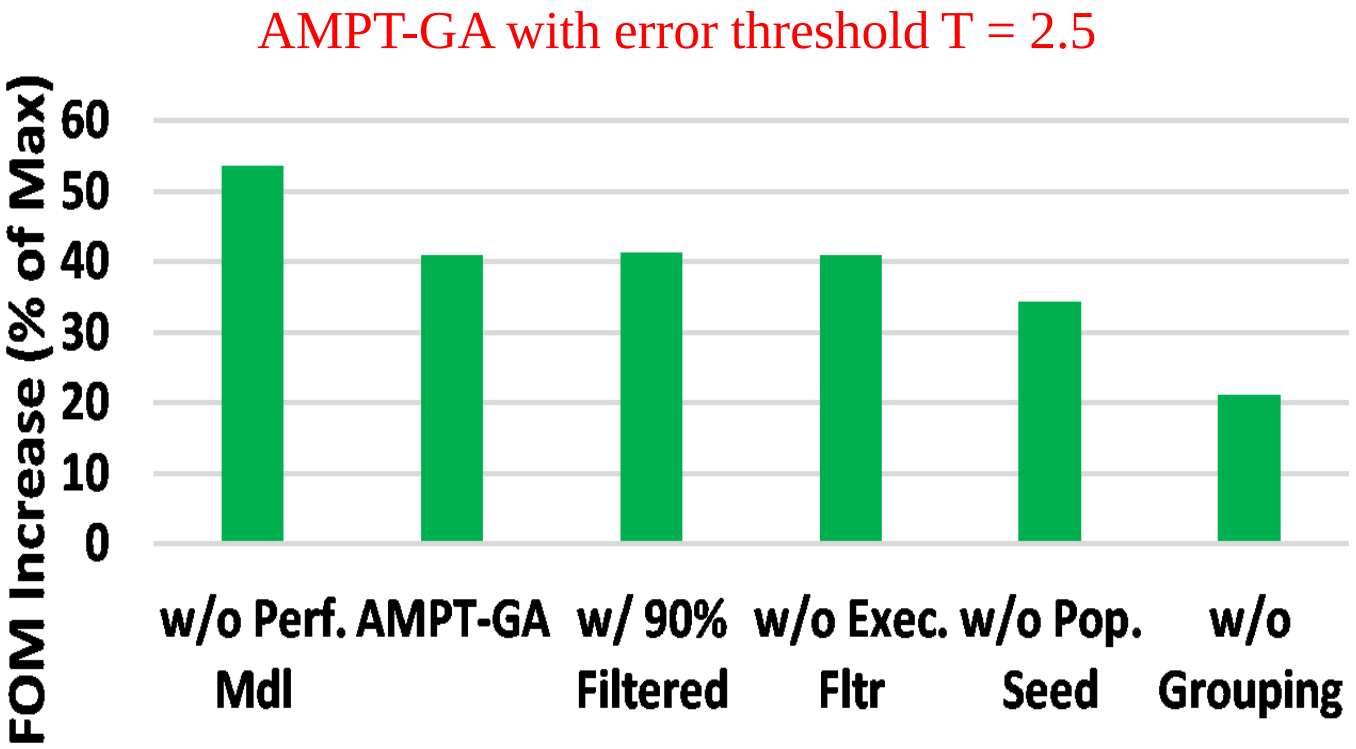




1. AMPT-GA is the most efficient technique considering the improvement in FOM (performance) and the executions incurred
2. Naïve achieves highest FOM but is hugely inefficient
3. AMPT-GA execution filter is useful
4. Precimonious suffers due to frequent casting penalties

PURDUE UNIVERSITY

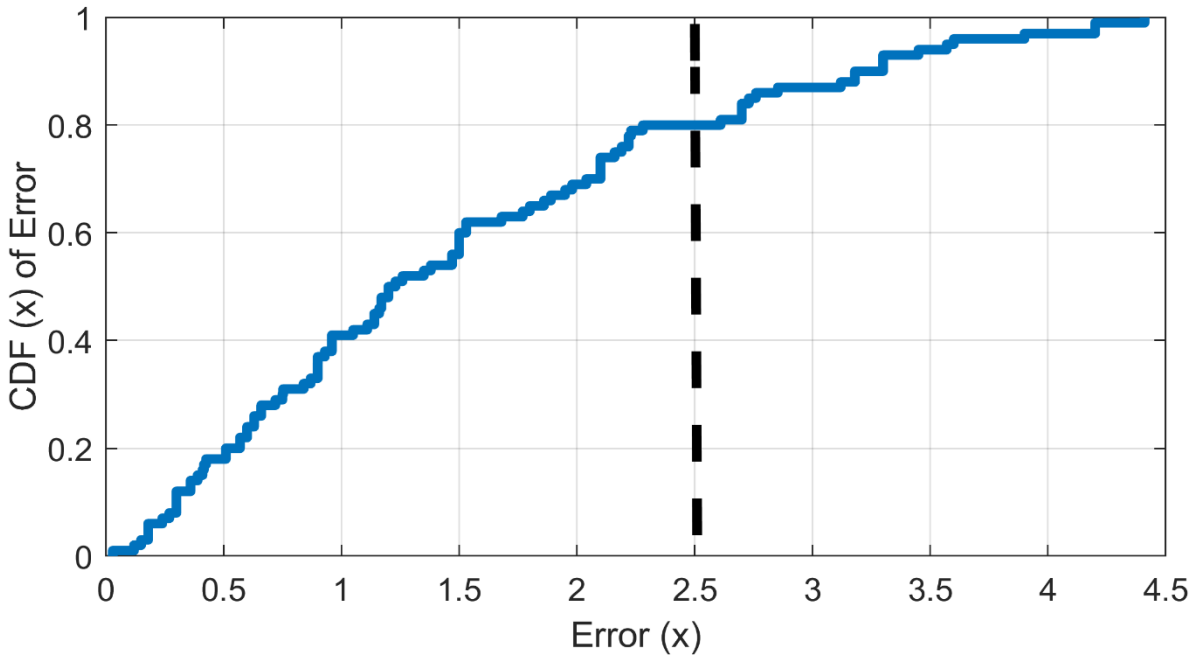
Component-wise Evaluation on LULESH



- Hyper-dimensional mutation (grouping) has highest impact
- Static performance model and execution filtering has no impact on performance, but it greatly improves efficiency

Input Data Dependence Evaluation on LULESH

AMPT-GA with error threshold T = 2.5



PV obtained by AMPT-GA satisfies the error threshold for 80% of the randomly chosen input sets

Take Aways

1. AMPT-GA introduces an efficient way to search the best performing floating point precision given the tolerable error threshold
2. AMPT-GA outperforms Precimonious in efficiency by 14-63%
3. AMPT-GA is capable of handling applications with large number of variables without exponentially blowing up the number of actual executions through performance model and execution filter
4. AMPT-GA is the first mixed floating precision technique for GPU kernels

- Open problem: Efficiency can be improved if we could design an error model that can be queried without program execution