Approximate Trigonometric Functions in GPGPU

sam@cs.brown.edu

Dec 15, 2015

Advanced Computer Architecture

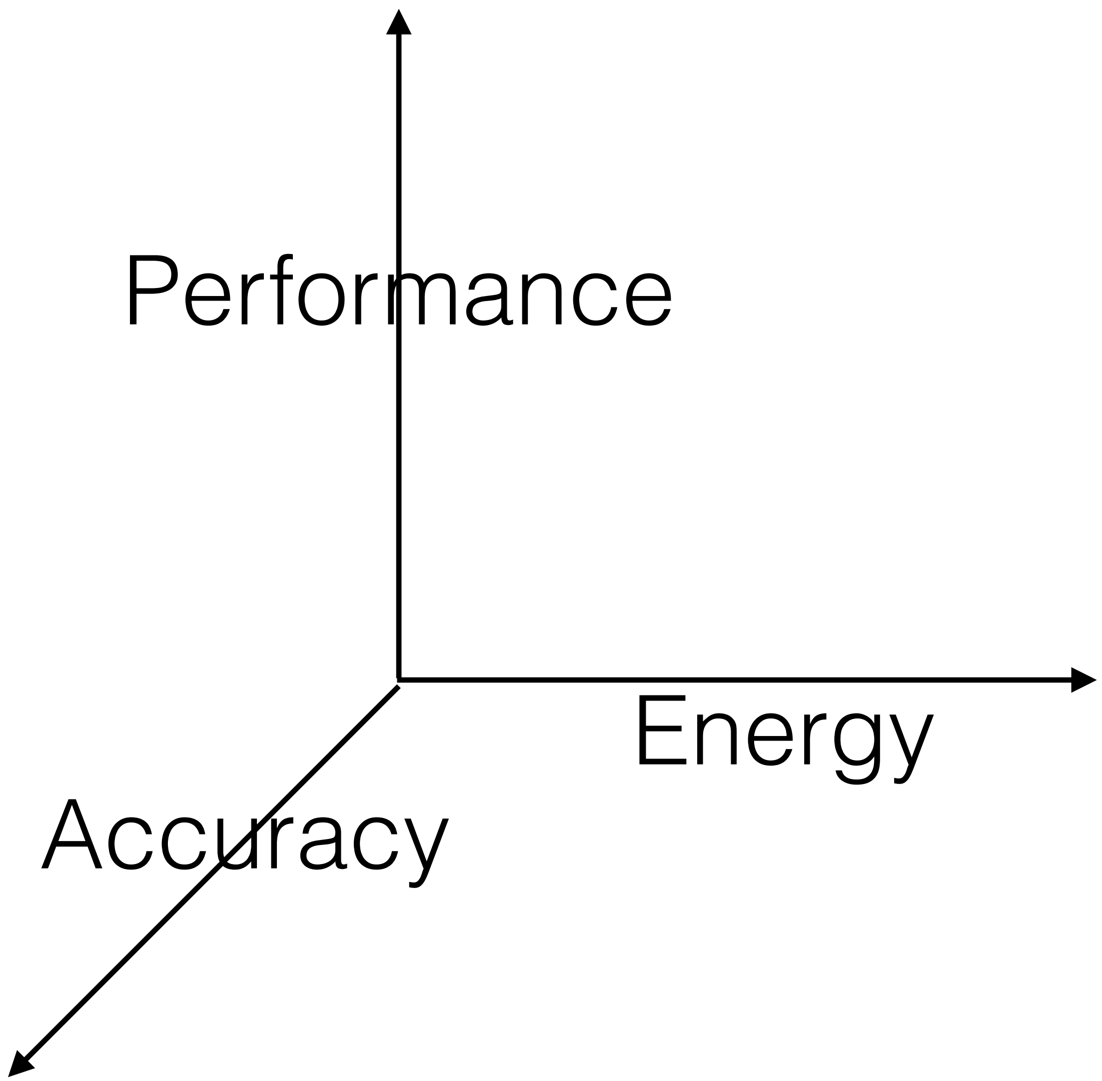
Outline

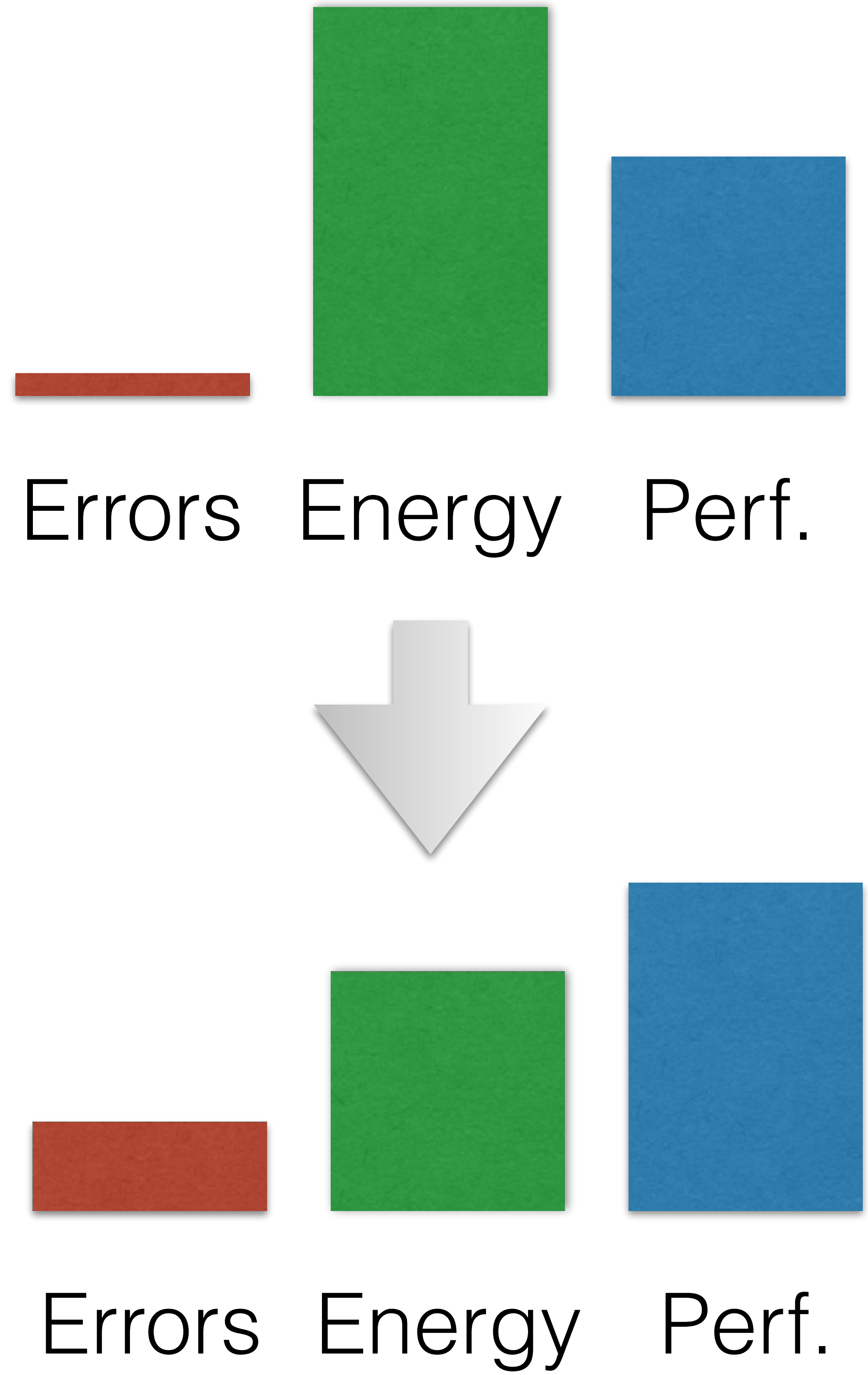
- Background
- Precise approximate trigonometric functions in GPGPU
- Our approximation approaches
- Experiments

Background

- Many emerging applications do not require perfect executions [1].
  - Input data is inexact
    - sensor data
  - Multiple acceptable outputs
    - machine learning algorithms
  - Imprecise output
    - Image rendering, sound and video processing

- Goal: trade off accuracy for additional energy savings and performance gain.





Approximate Trigonometric Functions

- Computer arithmetics are *finite* sequence of algebraic operations (add, multiply, root)

- Transcendental functions like *sin, cos* and *exp* *cannot* be expressed in finite sequence of algebraic operations.

- Traditionally *high precision approximation* routines in either software or recently in hardware.

CUDA Maths Library: software-level high precision approximation

```
__global__ void sin_array(float *a, int N)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx<N) a[idx] = sinf(a[idx]);
}
```

Linker

```
static __forceinline__ float __internal_accurate_sinf(float
{
  float z;
  int i;

  if (__isinff(a)) {
    a = __fmul_rn (a, CUDART_ZERO_F);
  }
  a = __internal_trig_reduction_kernel(a, &i);
  z = __internal_sin_cos_kernel(a, i);
#if __CUDA_ARCH__ < 200
  if (a == CUDART_ZERO_F) {
    z = __fmul_rn (a, CUDART_ZERO_F);
  }
#endif /* __CUDA_ARCH__ < 200 */
  return z;
}
```

```
static __forceinline__ float __internal_sin_cos_kernel(float x, int i)
{
#if __CUDA_ARCH__ >= 200
  float x2, z;
  x2 = __fmul_rn (x, x);
  if (i & 1) {
    z =                              2.44331571e-5f;
    z = __internal_fmad (z, x2, -1.38873163e-3f);
  } else {
    z =                             -1.95152959e-4f;
    z = __internal_fmad (z, x2,  8.33216087e-3f);
  }
  if (i & 1) {
    z = __internal_fmad (z, x2,  4.16666457e-2f);
    z = __internal_fmad (z, x2, -5.00000000e-1f);
  } else {
    z = __internal_fmad (z, x2, -1.66666546e-1f);
    z = __internal_fmad (z, x2,  0.0f);
  }
  x = __internal_fmad (z, x, x);
  if (i & 1) x = __internal_fmad (z, x2, 1.0f);
  if (i & 2) x = __internal_fmad (x, -1.0f, CUDART_ZERO_F);
#else /* __CUDA_ARCH__ >= 200 */
  if (i & 1) {
    x = __internal_cos_kernel(x);
  } else {
    x = __internal_sin_kernel(x);
  }
  if (i & 2) {
    x = __internal_fmad (x, -1.0f, CUDART_ZERO_F);
  }
#endif /* __CUDA_ARCH__ >= 200 */
  return x;
}
```

GPU Hardware Routines: Intrinsic functions

```
__global__ void sin_array(float *a, int N)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx<N) a[idx] = __sinf(a[idx]);
}
```

GCC for PTX target

```
$LDWbegin__Z9sin_arrayPfi:
    mov.u16      %rh1, %ctaid.x;
    mov.u16      %rh2, %ntid.x;
    mul.wide.u16     %r1, %rh1, %rh2;
    cvt.u32.u16      %r2, %tid.x;
    add.u32      %r3, %r2, %r1;
    ld.param.s32     %r4, [__cudaparm__Z9sin_arrayPfi_N];
    setp.le.s32      %p1, %r4, %r3;
    @%p1 bra     $Lt_1_1026;
    .loc    16   19   0
    ld.param.u64     %rd1, [__cudaparm__Z9sin_arrayPfi_a];
    cvt.s64.s32      %rd2, %r3;
    mul.wide.s32     %rd3, %r3, 4;
    add.u64      %rd4, %rd1, %rd3;
    ld.global.f32    %f1, [%rd4+0];
    sin.approx.f32   %f2, %f1;
    st.global.f32    [%rd4+0], %f2;
```

GPU

PTX Instructions

GTX480 (Fermi)

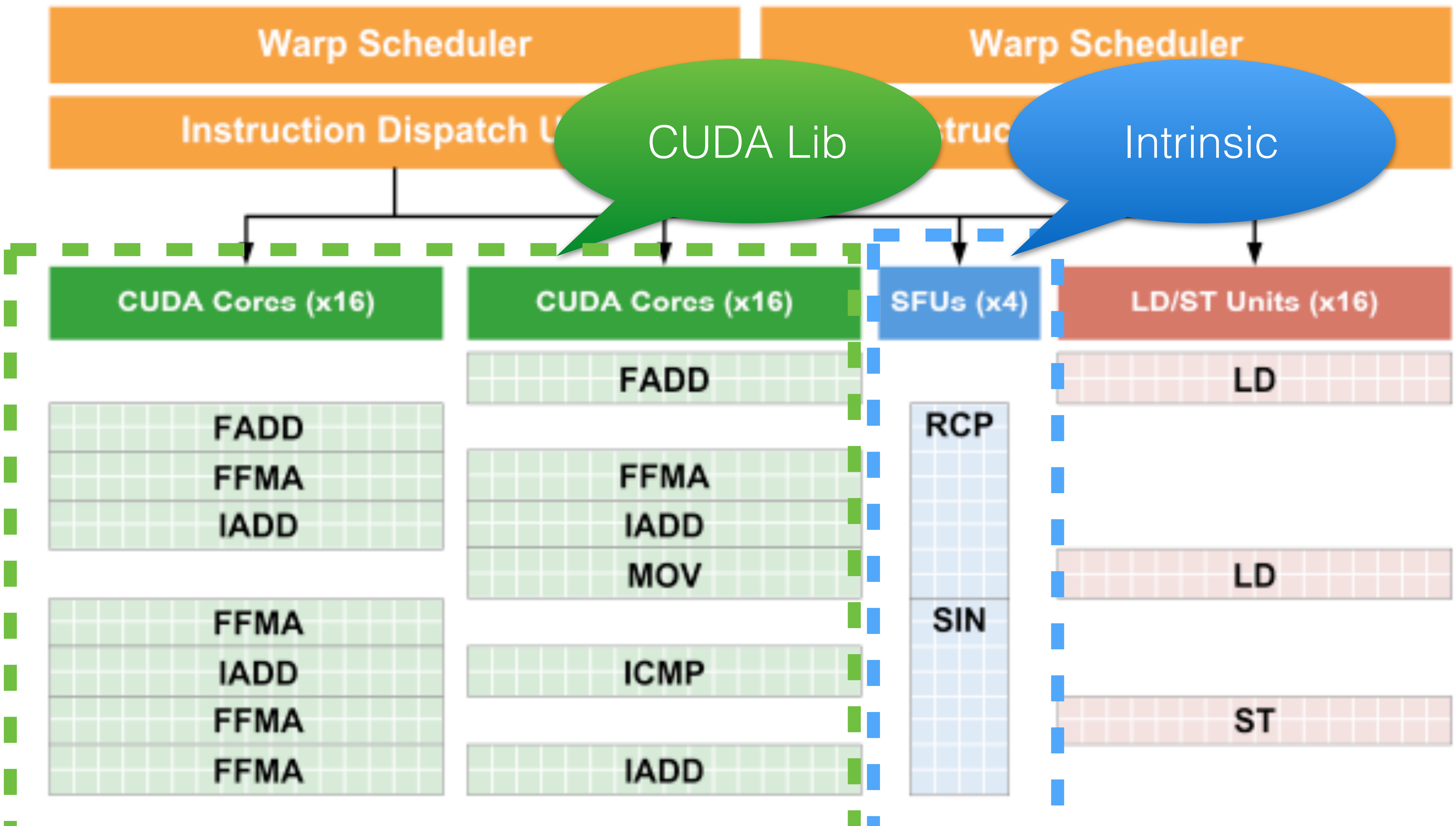


Figure 7. A total of 32 instructions from one or two warps can be dispatched in each cycle to any two of the four execution blocks within a Fermi SM: two blocks of 16 cores each, one block of four Special Function Units, and one block of 16 load/store units. This figure shows how instructions are issued to the execution blocks. (Source: NVIDIA)

Our Approach

- What can we improve upon?
  - Current software or hardware approaches focus on *precise approximation*.
  - Can we trade off precision for energy savings and/or performance gain?
    - Simplification of computation
      - Approximate ALU (SFU)
      - Approximate CUDA Maths Lib

- Chebychev Approximation [4]
    - find polynomial of degree <= n to minimize maximum error.

$$\max_{a \le x \le b} |f(x) - p(x)|.$$

- Chebychev Polynomials [4]
    - good set of nodes for polynomial interpolation

$$x_k = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{2k-1}{2n}\pi\right) \ , k = 1, \ldots, n.$$

    - interpolation error bound

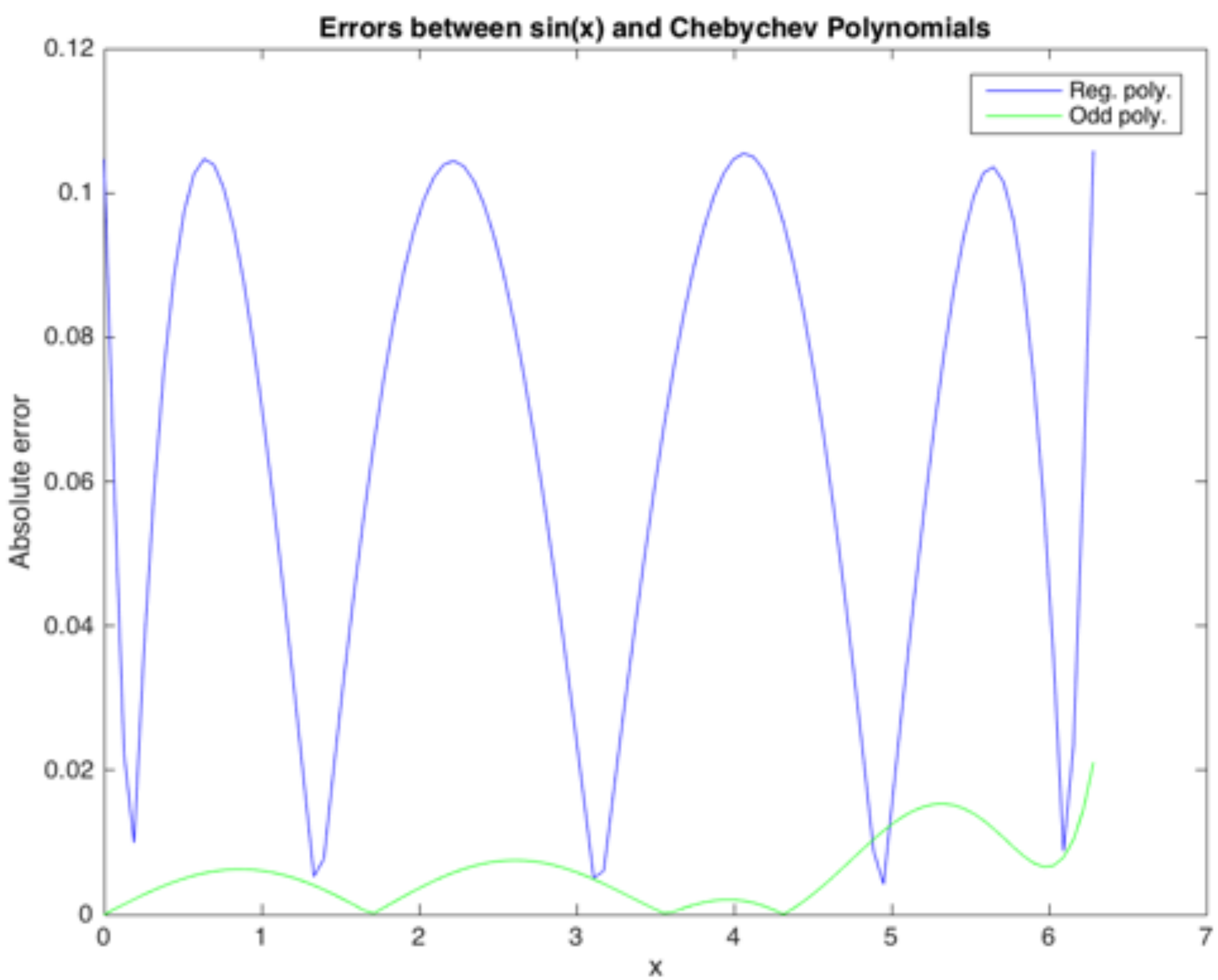
$$|f(x) - P_{n-1}(x)| \le \frac{1}{2^{n-1} n!} \left(\frac{b-a}{2}\right)^n \max_{\xi \in [a,b]} \left| f^{(n)}(\xi) \right|.$$

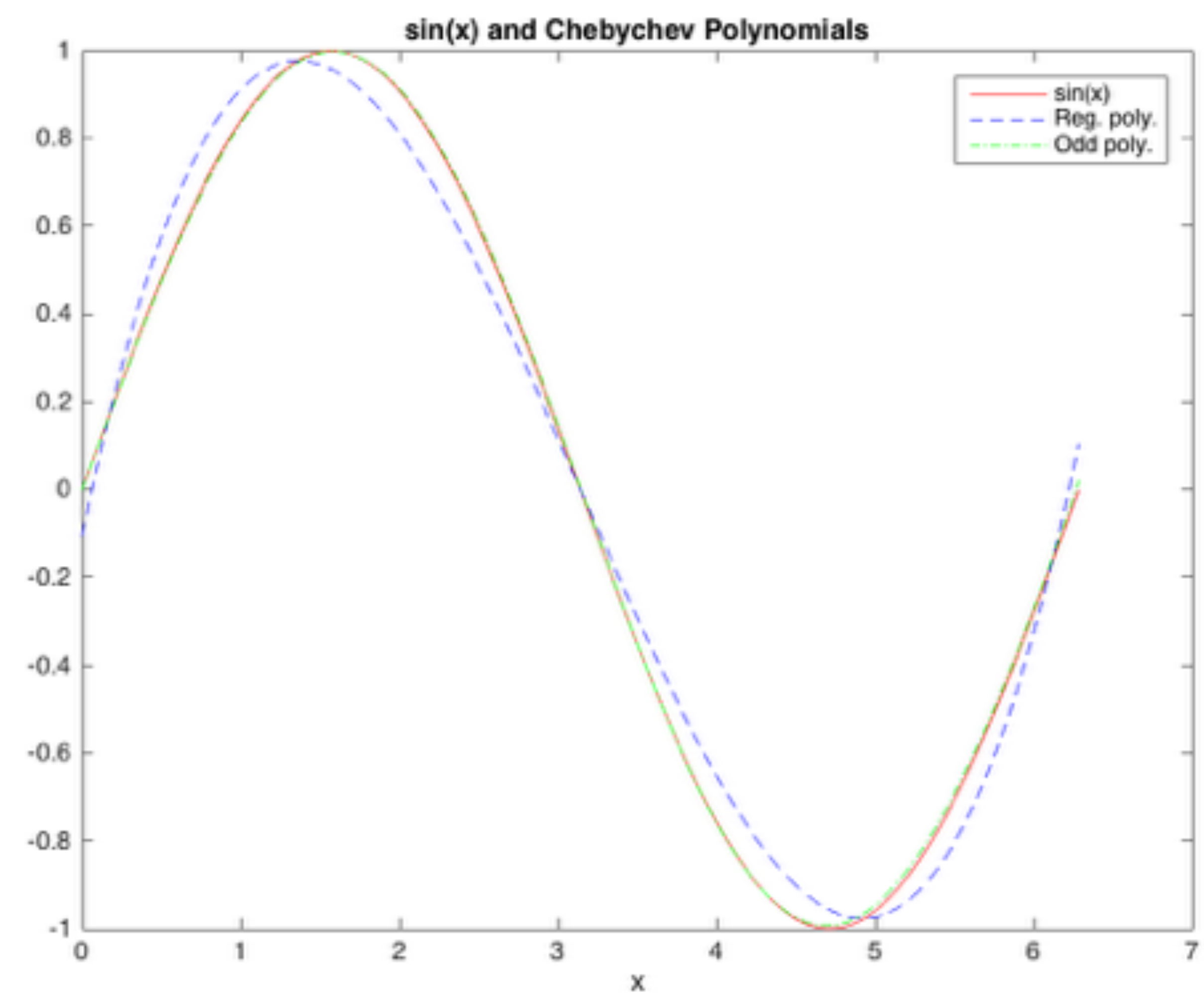
- Use Remez Algorithm [5] to calculate offline (e.g. by the compiler).
    - Solve linear system of equations.

$$b_0 + b_1 x_i + ... + b_n x_i^n + (-1)^i E = f(x_i)$$

- Our specific optimization:
  - Sine is odd function, can use odd polynomials to achieve better approximation with the similar computation.
  - Can trade off error bound with faster Remez runtime.

```
y1 = sin(x);
y2 = -1.047e-1 + x * 1.749 + x.^2 * -8.191e-1 + x.^3 * 8.691e-2 + x.^4 * -3.390e-156;
y3 = x * 9.886e-1 + x.^3 * -1.605e-1 + x.^5 * 7.410e-3 + x.^7 * -1.396e-4 + x.^9 * 9.846e-7;
```

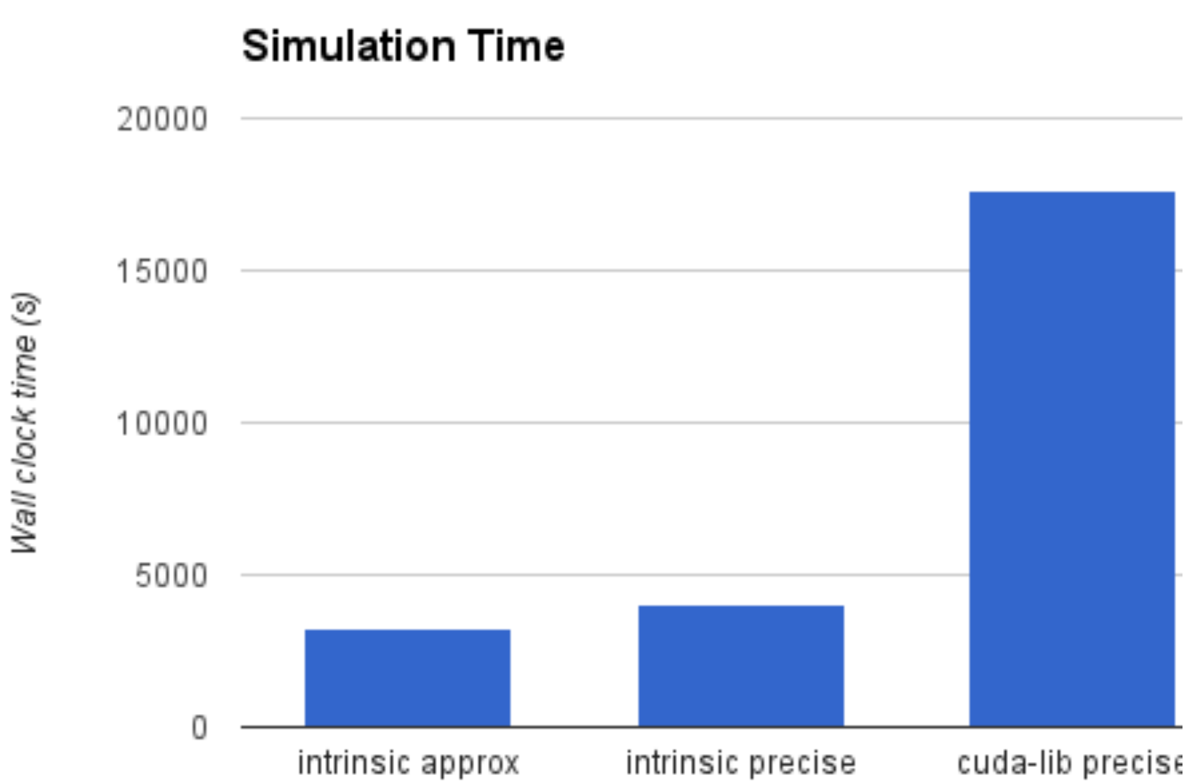


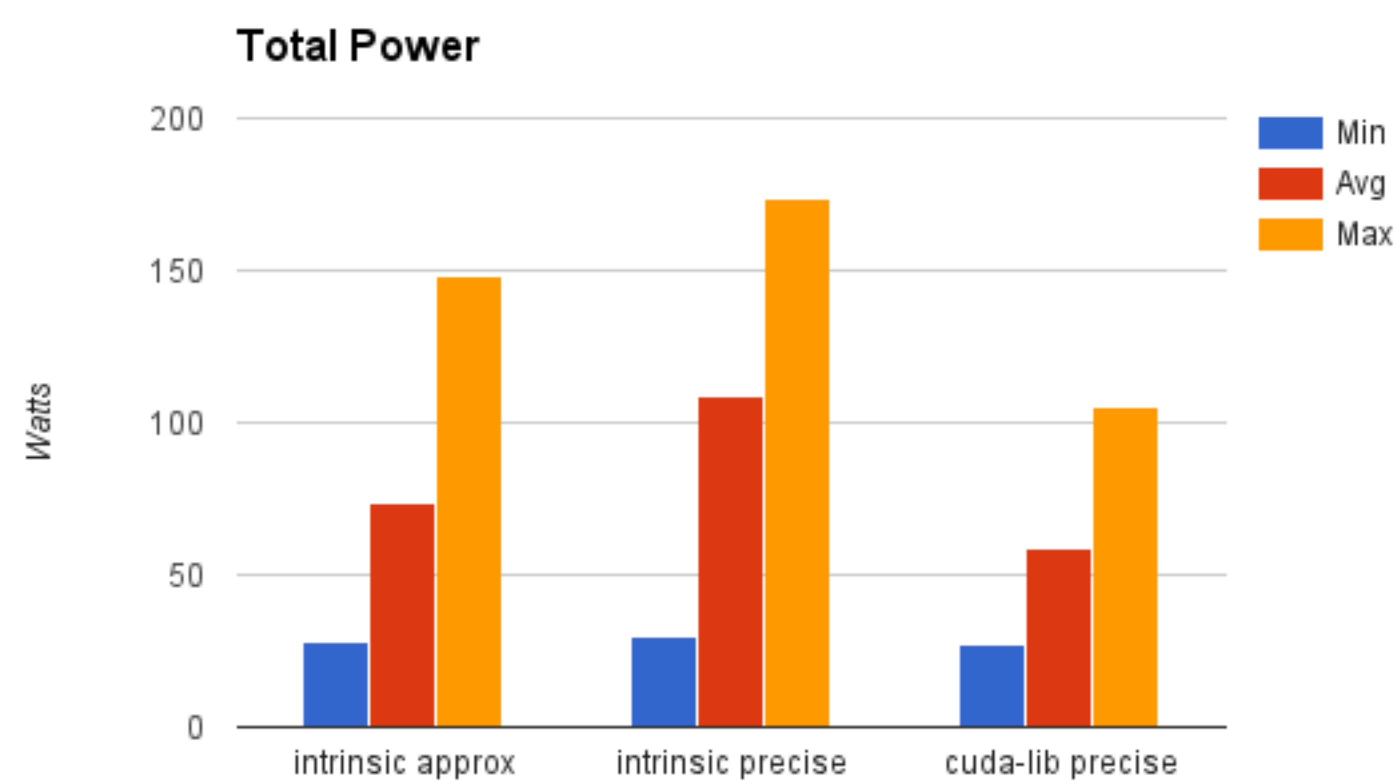


Experiments with GPGPU-Sim and GPUWattch

- Comparisons
  - HW
    - Intrinsic Approx
    - Intrinsic Precise
  - SW
    - Cuda-lib Precise
    - Cuda-lib Approx (missing)

- Benchmark: Parboil, MRI-Q [6].
- Result of Intrinsic Approx
  - **5.5x** speedup over Cuda-lib Precise
  - **1.2x** speedup over Intrinsic Precise.
  - Power consumption from **185%** in Intrinsic Precise to **125%** compared to Cuda-lib Precise.





Reference

1. https://homes.cs.washington.edu/~luisceze/ceze-approx-overview.pdf

2. Ali Bakhoda, George Yuan, Wilson W. L. Fung, Henry Wong, Tor M. Aamodt, Analyzing CUDA Workloads Using a Detailed GPU Simulator, in IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Boston, MA,April 19-21, 2009.

3. http://www.nvidia.com/content/pdf/fermi_white_papers/p.glaskowsky_nvidia's_fermi-the_first_complete_gpu_architecture.pdf

4. https://en.wikipedia.org/wiki/Chebyshev_nodes

5. https://en.wikipedia.org/wiki/Remez_algorithm

6. John A. Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid,vLi-Wen Chang, Nasser Anssari, Geng Daniel Liu, Wen-mei W. Hwu. IMPACT Technical Report, IMPACT-12-01, University of Illinois, at Urbana-Champaign, March 2012