Performance Comparison of a CUDA interval arithmetic library with a standard interval arithmetic library

D. Jailsingh Nayak and P. S. V. Nataraj
GPU Centre of Excellence, Systems and Control Engineering Group ,IIT Bombay, India

Abstract

We develop a CUDA based interval arithmetic library for GPU users. This library is based on the ideas in C-XSC interval library [1]. We compare the performance of the developed CUDA interval library with that of the C-XSC interval library for different interval arithmetic operations. The CUDA interval library is found to be much faster than the standard C-XSC library. This CUDA interval arithmetic library allows advanced interval techniques, such as interval global optimization, to be performed in comparatively very little time on GPUs .

Motivation

Interval arithmetic [2] is the arithmetic of quantities that lie within specified ranges (i.e., intervals) Interval arithmetic is especially useful when working with data that is subject to measurement errors or uncertainties. Interval arithmetic, however, is generally slow, This motivates us to develop an interval arithmetic library based on the NVIDIA GPU for speeding up interval arithmetic operations. The library will be useful in advanced interval methods, for instance, in performing interval global optimization on GPUs.

Resources used for the performance tests :
CPU : Intel(R) Xeon(R) @1.80GHz
GPU : NVIDIA Tesla C2070
MS Visual Studio 2012, CUDA 7
Work done: Compared results & performance of developed CUDA interval library versus those of C-XSC interval toolbox (serial code) on data set of 10 million random intervals

Method for tests: We adopted the following method for conducting the performance tests
Step (1): Initialize the input intervals and number of thread's resource allocation

Step (2): Generate a large interval data sets, in this case, set of 10 million random intervals for inputs

Step (3): Execute interval arithmetic using CUDA rounding operations on GPU: upward and downward rounding with respect to unit least precession (ULP) [3]. The ULP provides round toward 0, round toward $+\infty$ (or) $-\infty$.

Step (4): Transfer the memory copy from GPU to CPU

Step (5): Vary the number of threads and go to Step 3.

Interval Operations	Timing for C-XSC Code (serial) on CPU (milli-secs)	No of Threads	Timing for developed CUDA Code (parallel) On GPU (micro-secs)	Speedup factor using parallel CUDA code over serial C-XSC code
Basic Interval Arithmetic (+, -, *, /)	2.05	256	1.6	1281
		128	2.1	976
		64	3.2	640
Trigonometric functions (sin, cos, tan, arcsin, etc)	17.56	256	43.45	404
		128	66.92	262
		64	71.76	245
Hyperbolic Trigonometry functions (sinh, cosh, tanh, arcsinh ,etc)	13.56	256	52.17	260
		128	79.67	170
		64	94.9	143
Elementary functions (power , exponential , log, square root, etc)	3.75	256	20.45	183
		128	33.2	113
		64	48.9	77

Analysis and Summary

The Table alongside compares the timings obtained with C-XSC code (serial code on CPUs) with those of the developed CUDA code from our interval arithmetic library on GPUs. The timings are given for each arithmetic operation/function, averaged over 10 million random intervals. The table shows that significant speedups - of a couple of orders of magnitude - are obtained using the developed CUDA interval library, over the existing serial C-XSC library for interval arithmetic...

References

[1] R. Klatte, U. Kulisch, A. Wiethoff, C. Lawo and M. Rauch, C-XSC: A C++ Class Library for Scientific Computing, Springer-Verlag, 1993.

[2] R.E.Moore, R.B. Kearfott and M.J. Cloud, Introduction to Interval Analysis, SIAM, 2009.

[3] D. Goldberg, What every computer scientist should know about floating-point arithmetic, *ACM Computing Surveys*, 23(1):5–48, 1991.