EIGENVALUE COMPUTATIONS ON GRAPHICS PROCESSING UNITS

Andreas Selinger, Denis Ojdanić, Karl Rupp, and Erasmus Langer
Institute for Microelectronics

Many algorithms have the potential to dramatically increase their performance by using the processing power of Graphics Processing Units (GPUs). The bisection algorithm for tridiagonal symmetric matrices is one example. We adapted and corrected an implementation by NVIDIA and showed its performance in comparison to other implementations and algorithms.

THE EIGENVALUE PROBLEM

Finding the eigenvalues of a linear system is a very common problem in physics and engineering. Important examples are the solution of the Schrödinger equation or the simulation of mechanical oscillations. The problem can be written as

$$\mathbf{Av} = \lambda\mathbf{v},$$

where $\mathbf{A}$ is a regular $n \times n$ matrix, $\mathbf{v} \neq \mathbf{0}$ is an eigenvector of $\mathbf{A}$ and λ is an eigenvalue of $\mathbf{A}$.
We consider a symmetric real matrix $\mathbf{A}$, for which all eigenvalues are real. In general, it is very expensive to compute the eigenvalues, so they are only computed numerically in practice. Therefore it is important to choose a fast algorithm, especially when computing the eigenvalues of large systems.

BISECTION ALGORITHM

The heart of the bisection algorithm is a function we call EigenvalueCount $C(x)$, which computes the number of eigenvalues smaller than a scalar x. This yields that in an interval $[x_1, x_2[$, there are $C(x_2) - C(x_1)$ eigenvalues. Now the key to find all eigenvalues is to divide an interval, which is guaranteed to contain all eigenvalues (*Gerschgorin interval*), into two intervals. All resulting intervals with eigenvalues are further divided. This procedure is repeated until the bounds of each interval containing eigenvalues are so tight, that we can tell the eigenvalues as accurately as desired [1].

PARALLELIZATION OF THE ALGORITHM

With the high number of processors residing on a GPU, it can run general-purpose computations simultaneously at a very high performance. Since the computations of all eigenvalues on a level of the interval tree do not depend on each other, they can be processed very efficiently and in parallel on the GPU, instead of computing one eigenvalue after another on the CPU.

Unfortunately, the NVIDIA implementation failed when applied to matrices with multiple eigenvalues. We fixed this error, which was caused by a false termination condition and by a race condition. Furthermore, we ported the CUDA version of the implementation to OpenCL to make it not only suitable for NVIDIA-GPUs, but also for AMD-GPUs.

Performance Analysis

Figure 1 shows the performance of the parallel bisection algorithm in comparison to serial algorithms. By using the GPU, a performance gain on the order of one magnitude over the reference TQL1 implementation was obtained.

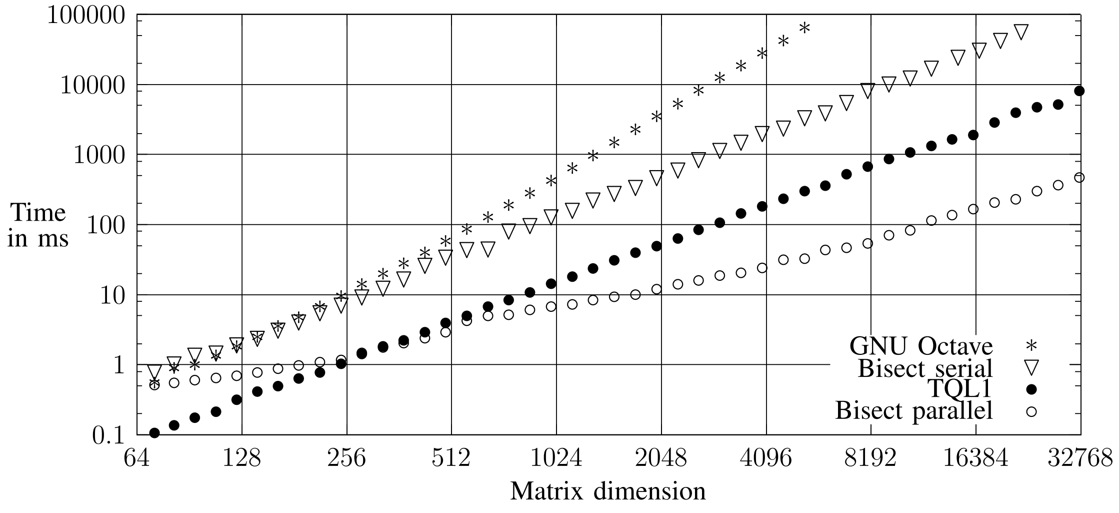

Fig. 1. Performance of different algorithms: In our tests, the serial algorithms ran on an Intel Core i7 960 CPU and the parallel bisection algorithm ran on an NVIDIA GeForce GTX 580 GPU.

Conclusion

Compared to CPU-based implementations for tridiagonal symmetric matrices, a magnificent speedup was achieved with the parallel algorithm. Our implementation is available in the free open source software library ViennaCL [2].

References

[1] C. Lessig, "Eigenvalue Computation with CUDA," *NVIDIA CUDA Toolkit*, 2007. [Online]. Available: http://docs.nvidia.com/cuda/cuda-samples/index.html#eigenvalues

[2] ViennaCL. . [Online]. Available: http://viennacl.sourceforge.net