# Quantitative High-Resolution TEM/STEM and Diffraction

## IM.1.P012
## GPU-accelerated Bloch wave calculation for quantitative structure factor determination

F. Wang[1], R. Pennington[1], C. Koch[1]

[1]Universität Ulm, Institut für Experimentelle Physik, Ulm, Germany

feng.wang@uni-ulm.de
Keywords: dynamical electron diffraction, Bloch waves, matrix exponential, GPU, LARBED, image simulation

Multiple scattering solves the crystallographic phase problem. This allows for structure-factor refinement by quantitative convergent-beam electron diffraction (QCBED) [1,2]. In QCBED, iterative refinement from dynamical electron diffraction data from multiple beam directions yields structure-factor amplitudes and phases, but QCBED only works for small unit cells and requires a good initial guess of the structure factors. In contrast, by using quantitative large-angle rocking-beam electron diffraction (QLARBED) [3] data from thin crystals (see, for example Fig 1b,c), even unknown crystal structures may be determined, even for large unit cells. This is possible because, for moderately thick crystals, the Bloch wave formalism for dynamical scattering can be expanded using a scattering path expansion [4] to find an initial guess for the structure factors [5]. This initial guess can then be refined using the classical QCBED approach. Fortunately, the initial guess for QLARBED need not be as precise as for QCBED, since, for thinner crystals, there are fewer local minima than for thick crystals. While QCBED must be applied only to crystals with small unit cells with only a few reflections, QLARBED is targeted at arbitrarily large structures and must therefore be able to handle many simultaneously excited beams. Therefore, QLARBED requires a very fast and scalable algorithm for computing dynamical diffraction intensities. In QCBED, this computation is mostly done with the Bloch wave method. For QLARBED, we also chose the Bloch wave method, because, compared to the multislice method, the Bloch wave method allows for beam tilt, Ewald-sphere curvature, and specimen surface orientation to be included in a very direct manner *i.e.* without having to deal with non-periodic boundary conditions or insufficiently thin slices [6].

In this work, we compare the numerical efficiency of multiple implementations of computing dynamical diffraction intensities. We discuss how using state-of-the-art graphics processing unit (GPU) hardware acceleration has helped to speed up the calculation by more than two orders of magnitude (see Fig. 2). For an optimal accuracy-time trade-off, a hybrid method using Taylor expansion and the scaling/squaring method has been employed to compute the matrix exponential [7, 8, 9].

1. C. Deininger, G. Necker and J. Mayer, Ultramicroscopy 54 (1994) pp. 15-30.
2. J.-M. Zuo, M. Kim, M. O'Keefe, and J.C.H. Spence, Nature 401 (1999) p. 49.
3. C.T. Koch, Ultramicroscopy 111 (2011) pp. 828-840.
4. C.T. Koch and J.C.H. Spence, J. Phys. A: Math. Gen. 36 (2003) pp. 803-816.
5. C.T. Koch, arXiv:0810.3811v1.
6. J.C.H. Spence and J.-M. Zuo, *Electron Microdiffraction*, (Plenum, New York) (1992).
7. C. B. Moler and C. F. Van Loan, SIAM Review 20 (1978) pp. 801-836.
8. C. B. Moler and C. F. Van Loan, SIAM Review 45 (2003) pp. 3-49.
9. M.L. Liou, Proc. Inst. Elect. Electron. Engrs. 54 (1966) pp. 20-23.
10. E. Jones, T. Oliphant, P. Peterson et al. SciPy: Open source scientific tools for Python. http://www.scipy.org.
11. J. R. Humphrey, D. K. Price, K. E. Spagnoli, A. L. Paolini, and E. J. Kelmelis, SPIE Defense and Security Symposium (DSS), 2010.
12. The authors acknowledge the Carl Zeiss Foundation as well as the German Research Foundation (DFG, Grant No. KO 2911/7-1).

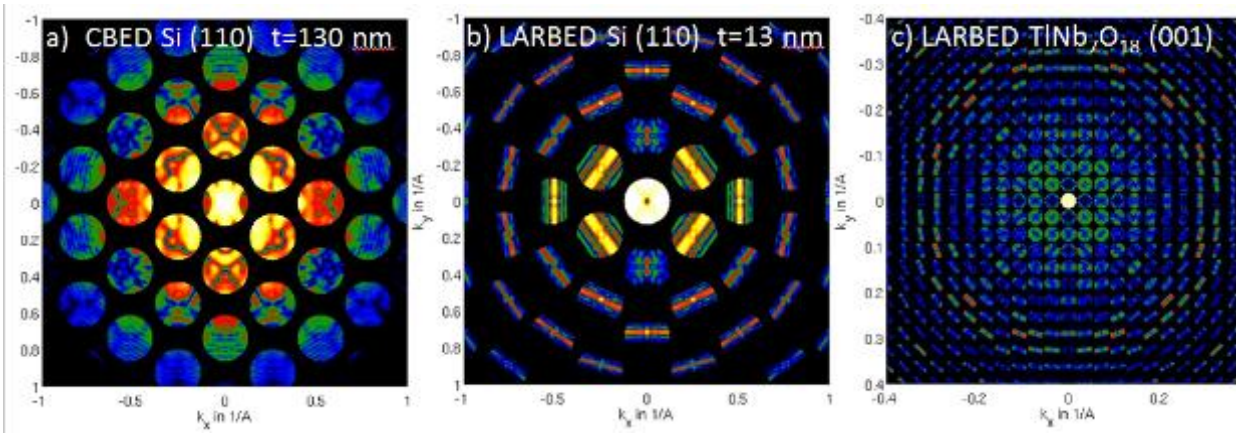


**Figure 1.** CBED and LARBED patterns computed using the GPU-based diffraction code. The patterns are displayed on a logarithmic scale. The simulation conditions were a) thickness = 130 nm, convergence-semi-angle $\alpha_{max}$ = 4.2 mrad, scan compensation = 0 (descan turned off); b) thickness = 13 nm, $\alpha_{max}$ = 70 mrad, scan compensation = 88%; c) thickness = 10 nm, $\alpha_{max}$ = 70 mrad, scan compensation = 99.2%. These patterns have been computed without using the Bethe approximation for any reflections.

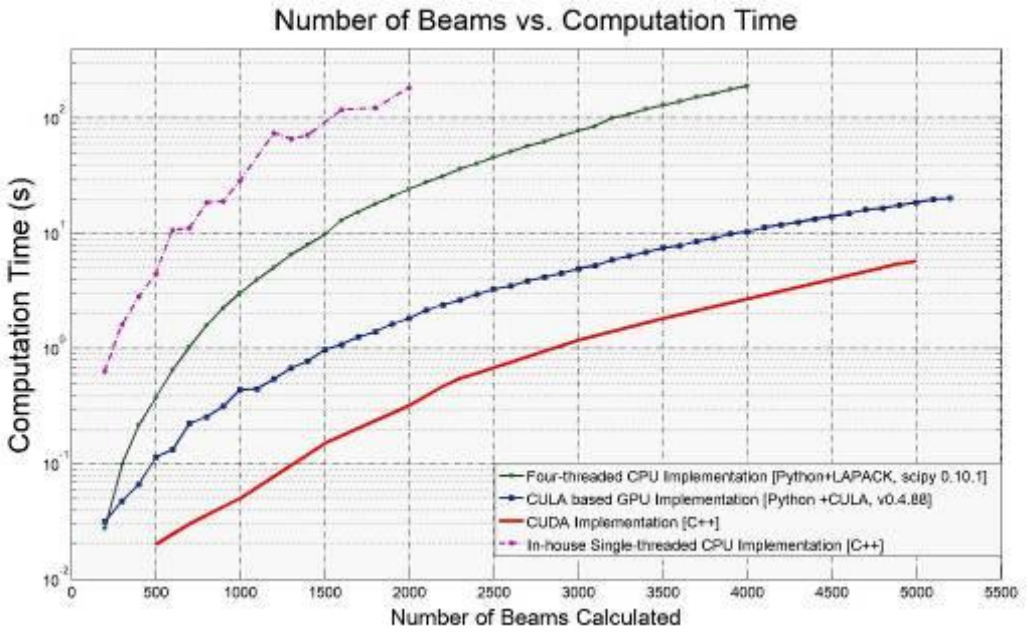


**Figure 2.** Matrix exponential implementation benchmark: The GPU-based CUDA code (running on an Nvidia Tesla K20) is 2 orders of magnitude faster than the parallelized CPU code (running on an Intel Xeon W3550). The purple and the green curves show the performance on the CPU: the purple one is a single-threaded native C++ implementation, while the green one is a LAPACK/BLAS-based implementation, using 4 CPU threads and the SciPy Python package [10]. The blue and the red curves show the performance on the GPU: the blue one is based on CULA [11] and written in Python, while the red one is a native CUDA C++ implementation. The algorithm corresponding to the purple, green and blue curves uses the Padé approximation (purple: order 14, green and blue: order 7). The algorithm corresponding to the red curve uses the scaling and squaring method with a 9th-order Taylor expansion.