Gpufit: An open-source toolkit for GPU-accelerated curve fitting

Adrian Przybylski[1], Björn Thiel[1], Jan Keller-Findeisen[1], Bernd Stock[2], and Mark Bates[1,*]

[1]Department of NanoBiophotonics, Max Planck Institute for Biophysical Chemistry, Am Fassberg 11, Göttingen 37077, Germany

[2]Faculty of Natural Sciences and Technology, University of Applied Sciences and Arts, Von-Ossietzkystr. 99, Göttingen 37085, Germany

*Correspondence should be addressed to mark.bates@mpibpc.mpg.de

We present a general purpose, open-source software library for estimation of non-linear parameters by the Levenberg-Marquardt algorithm. The software, Gpufit, runs on a Graphics Processing Unit (GPU) and executes computations in parallel, resulting in a significant gain in performance. We measured a speed increase of up to 42 times when comparing Gpufit with an identical CPU-based algorithm, with no loss of precision or accuracy. Gpufit is designed such that it is easily incorporated into existing applications or adapted for new ones. Multiple software interfaces, including to C, Python, and Matlab, ensure that Gpufit is accessible from most programming environments. The full source code is published as an open source software repository, making its function transparent to the user and facilitating future improvements and extensions. As a demonstration, we used Gpufit to accelerate an existing scientific image analysis package, yielding significantly improved processing times for super-resolution fluorescence microscopy datasets.

Introduction

Optimization algorithms are widely used in science and engineering. In particular, when comparing data with a model function, numerical optimization methods may be applied to establish the suitability of the model and to determine the parameters which best describe the observations. One such method, which is generally applicable to models which may depend non-linearly on a set of parameters, is the Levenberg-Marquardt algorithm (LMA),[1] and this has become a standard approach for non-linear least squares curve fitting.[2,3]

Although the LMA is, in itself, an efficient algorithm, applications requiring many iterations of this procedure may encounter limitations due to the sheer number of calculations involved. The time required for the convergence of a fit, or a set of fits, can determine an application's feasibility, e.g. in the context of real-time data processing and feedback systems. Alternatively, in the case of very large datasets, the time required to solve a particular optimization problem may prove impractical.

In recent years, advanced graphics processing units (GPUs) and the development of general purpose GPU programming have enabled fast and parallelized computing by shifting calculations from the CPU to the GPU.[4] The large number of independent computing units available on a modern GPU enables the rapid execution of many instructions in parallel, with an overall computation power far exceeding that of a CPU. Languages such as CUDA C and OpenCL allow GPU-based programs to be developed in a manner similar to conventional software, but with an inherently parallelized structure. These developments have led to the creation of new GPU-accelerated tools, such as the MAGMA linear algebra library,[5,6] for example.

Here, we present Gpufit: a GPU-accelerated implementation of the Levenberg-Marquardt algorithm. Gpufit was developed to meet the need for a high performance, general-purpose nonlinear curve fitting library which is publicly available and open source. As expected, this software exhibits significantly faster execution than the equivalent CPU-based code, with no loss of precision or accuracy. In this report we discuss the design of Gpufit, characterize its performance in comparison to other CPU-based and GPU-based algorithms, and we demonstrate its use in a scientific data analysis application.

Results

Software design

The Gpufit library was designed to meet several criteria: *(i)* the software should make efficient use of the GPU resources in order to maximize execution speed, *(ii)* the interface should not require detailed knowledge of the GPU hardware, *(iii)* the source code should be modular and extendable, and *(iv)* the software should be accessible from multiple programming environments.

Gpufit implements the LMA entirely on the GPU, minimizing data transfers between CPU and GPU memory. Copying memory between the CPU and the GPU is a slow operation, and it is most efficient to handle large blocks of data at once. Therefore, Gpufit is designed to process multiple fits simultaneously, each with the same model function and data size, but allowing for unique starting parameters for each fit. At the start of a calculation, a chunk of input data and initial fit parameters are copied to the GPU, and upon completion the results are transferred back to CPU memory.

GPU architecture is based on a set of parallel multiprocessors, which divide computing tasks between blocks of processing threads, as illustrated in Supplementary Fig. S1. While determining how best to parallelize the LMA, we considered calculating one fit per thread block, and one data point per thread. However, we found that this approach yielded poor results due to the small number of simultaneously executed threads per multiprocessor, and therefore the design was modified to allow the calculation of multiple fits in each thread block. Also, a parallelized Gauss-Jordan algorithm was developed in order to efficiently solve for the correction ($\vec{\delta}_i$) to the fit parameters at each iteration (see equation (4) in Methods).

The number of computing blocks and threads may vary significantly depending on the GPU hardware, and our aim was to avoid the necessity for any hardware-specific configuration parameters in the Gpufit interface. Therefore, the software was designed to read the properties of the GPU at run-time, and automatically set parameters such as the number of blocks, threads, and the number of fits per thread block. This makes the use of a GPU transparent to the programmer, and allows the Gpufit interface to be no different from that for a conventional curve fitting function.

The Gpufit source code is modular, such that model functions and goodness-of-fit estimators are separate from the core sections of the code, and new functions or estimators may be added simply (see the Supplementary Information). In its initial release, Gpufit includes two different fit estimators: the standard weighted least-squares estimator (LSE), and a maximum likelihood estimator (MLE) which provides better fit results when the input data is characterized by Poisson statistics.[7] The modular concept of the Gpufit software is illustrated schematically in Supplementary Fig. S2. This modularity allows Gpufit to be quickly adapted to new applications, or modified to accommodate future developments.

Finally, Gpufit is written in C, CUDA C, and C++, and compiles to a Dynamic Link Library (DLL) providing a C interface, making it straightforward to call Gpufit from most programming environments. Furthermore, Gpufit bindings for Matlab and Python (e.g. the pyGpufit module) were implemented, forming part of the Gpufit distribution.

Performance characterization

We tested Gpufit by using the software to process randomized simulated datasets. The precision and accuracy of the fit results, as well as the number of fit iterations and the execution time, were measured. The input data consisted of 2D Gaussian functions defined by five parameters (see Supporting Information). Random noise (Gaussian or Poisson) was added to each data point. Test data was generated in Matlab and passed to the fit routines via their Matlab interfaces.

Algorithm validation

An initial step in characterizing Gpufit was to validate the function of the algorithm. For this purpose, we tested Gpufit against the well-established MINPACK library,[8] evaluating the precision and accuracy of the fit results, and the convergence of the fit. Gaussian noise was added to the input data such that a signal to noise ratio (SNR) could be defined. Upon testing, the two packages yielded almost identical fit precision over a wide range of SNR values, and converged in a similar number of steps (Supplementary Fig. S3). At very high SNR, differences appear due to the limited numerical precision of floating point operations in CUDA (single precision) vs. MINPACK (double precision). Fit accuracy was checked by comparing the distributions of the fit parameters (Supplementary Fig. S4), and no systematic deviation between the output of Gpufit and MINPACK was detected. Together, these measurements verify that Gpufit yields precise and accurate fit results, and converges similarly when compared to existing optimization software.

Execution speed and fit precision

The parallel architecture of the GPU enables significant speed improvements when a computation is amenable to being divided among multiple processors. To fairly evaluate the speed improvement obtained by shifting processing tasks from the CPU to the GPU, equivalent implementations of the LMA, running on both architectures, were required. We created a library called Cpufit to serve as the reference CPU-based fitting algorithm for testing purposes (see Methods). Cpufit implements precisely the same algorithm, section by section, as Gpufit. We verified that Cpufit and Gpufit returned numerically identical fit results given identical input data.

A comparison of execution speeds, plotted against the number of fits per function call (N), is shown in Fig. 1. As expected, the fitting speed on the CPU remains constant as N changes, due to the sequential execution of each fit calculation. On the other hand, for Gpufit the speed increases with increasing N. The more fits that are calculated in one execution, the greater the benefit of parallelization. The crossover point at which use of the GPU becomes advantageous is, in this case, approximately $N = 130$ fits. For large N the speed of GPU processing saturates, indicating that GPU resources are fully utilized. At $N = 10^8$ fits per function call, we measured a processing speed of more than 4.5 million fits per second, approximately 42 times faster than the same algorithm executed on the CPU.

Measurements of the execution time for each sub-section of the Cpufit and Gpufit code reveal the bottlenecks of the CPU-based algorithm, and how the computational workload is re-distributed on the GPU. For a set of 5×10^6 fits we measured the time duration of each step of the fitting process, and these results are shown in Fig. 2. For reference, program flowcharts for Cpufit and Gpufit, color coded according to processing time, are shown in Supplementary Fig. S5 and S6. In our tests, the most time-consuming task for Cpufit was the calculation of the model function, requiring more than one third of the total execution time. Gpufit completed the same calculation more than 100 times faster, due to the parallelization of the work. Similarly, all other steps in the fit process ran 10 – 100 times faster on the GPU. Gpufit includes additional operations, such as data transfers between CPU and GPU memory, however these did not impact performance when sufficient numbers of fits were processed.

Given the parallel computing capability of the GPU, it was not surprising that Gpufit outperformed an equivalent algorithm running on a CPU. In order to verify that our code is efficiently implemented, we therefore tested Gpufit against another GPU-based fitting library: GPU-LMFit.[9] These tests were limited to smaller datasets because GPU-LMFit is available only as a closed-source, 32-bit binary package, restricting the size of the memory it can address. Figure 3A shows the speed of the Gpufit and GPU-LMFit libraries measured as a function of the number of fits per function call (N), with the speed of the MINPACK library shown for reference. Both packages exhibited similar scaling in speed as the number of fits and the data size varied, however, Gpufit showed faster performance for all conditions tested. As the data size per fit was increased (Fig. 3B), the speeds became more comparable, indicating that Gpufit makes more efficient use of GPU resources for smaller fits. The increased speed comes with no loss of precision, as it was verified that the fit results returned by both Gpufit and GPU-LMFit have virtually identical numerical precision (see Supplementary Fig. S7).

When data is subject to counting statistics (i.e. when the noise has a Poisson distribution), curve fitting using an alternative goodness-of-fit measure based on maximum likelihood estimation (MLE) has been reported to yield more accurate fit results, when compared with least-squares fitting.[10,11] We tested the performance of this estimator using input data with simulated Poisson noise. The precision of the fit results, using either the unweighted LSE, weighted LSE, or MLE estimators, is shown in Supplementary Fig. S8. We found that the MLE estimator yielded better results than the LSE, particularly at small data values, although for larger values the two methods were approximately equivalent.

Application to super-resolution microscopy

Gpufit is well suited for rapid processing of large datasets, and its interface allows it to serve as a drop-in replacement for existing CPU-based fit routines. To demonstrate these capabilities in a real application, we integrated Gpufit into the image analysis pipeline of a super-resolution fluorescence microscopy experiment.

Stochastic optical reconstruction microscopy (STORM) is a method for fluorescence imaging of biological samples, which obtains an image resolution significantly higher than the classical “diffraction limit” of far-field optical microscopy.[12] This method relies heavily on image processing: the generation of a single STORM image requires thousands to millions of individual fitting operations, and this task may require minutes to hours of computation before the final image is obtained.

We integrated Gpufit into a recently published software package, Picasso,[13] which may be used to process raw STORM data into a super-resolved image of the sample. The Picasso software is written in Python, and is optimized to carry out the fit using a multi-threaded process running on all cores of the CPU. Moreover, Picasso uses "just in time" (JIT) compilation to optimize its execution speed. We modified this section of the Picasso source code, replacing the multi-core CPU-based fitting with a call to Gpufit, as illustrated schematically in Fig. 4A. Comparing the speed of Picasso before and after the modification demonstrates the benefit of Gpufit. When analyzing a raw STORM dataset (80000 images requiring 3.6 million fit operations), a fitting task which required 99.4s with standard Picasso was completed in only 2.2s after Gpufit was included, a 45-fold speed increase, with identical fit precision. The output STORM image is shown in Fig. 4B, and a comparison of the fitting time with and without Gpufit is shown in Fig. 4C.

Discussion

General purpose GPU computing is a relatively new technology, which is making an impact in many fields of science and engineering. The software introduced here, Gpufit, represents the first general purpose, open source implementation of a non-linear curve fitting algorithm for the GPU. It is intended as a high performance optimization tool, easily modified and adapted for new tasks, which can be rapidly implemented within existing data analysis applications.

In terms of performance, Gpufit exhibits similar precision and accuracy to other fitting libraries, but with significantly faster execution. In our measurements, curve fitting with Gpufit was approximately 42 times faster than the same algorithm run on the CPU. Gpufit also outperformed another GPU-based implementation of the LMA, GPU-LMFit, for all conditions tested. These values depend on the details of the fit and the computer hardware. Higher performance would be expected with a more powerful GPU (e.g. an Nvidia Tesla), or with multiple GPUs running in parallel. In addition to its speed, the principal advantages of Gpufit are its general purpose design, which may incorporate any model function or modified estimator, and the availability of the source code, which allows it to be compiled and run on multiple computing architectures.

Since Gpufit automatically distributes the curve fitting tasks over the blocks and threads of the GPU multiprocessors, the user is not required to know the details of the hardware, thereby allowing Gpufit to be used as a "drop-in" replacement for existing fit functions. To demonstrate this, we modified Picasso to make use of Gpufit rather than its own multi-core CPU fitting code. Despite the fact that curve fitting in Picasso was already parallelized (by virtue of its use of multi-threaded processing) we found that Gpufit outperformed the built-in Picasso curve fitting by a factor of 45 times (Fig. 4). The ease with which Gpufit was integrated into Picasso, and the gain in performance, show that our original design goals were met.

As of its initial release, the Gpufit package has several limitations. First, the fit model functions are built into the code at compilation time, and the addition or modification of a model function requires re-compilation of the source code. We also note that Gpufit requires the explicit calculation of the gradients and Hessian of the model, and expressions for these functions must be present in the code embodying the fit model function. As an open-source software project, however, we expect that Gpufit will continue to develop and improve, potentially removing these limitations. For example, runtime compilation of model functions written in CUDA would remove the requirement for re-building the source, and methods to approximate the gradient and Hessian numerically could be introduced. Finally, there is the potential for porting Gpufit to other general-purpose GPU computing languages, such as OpenCL, which would make the code functional on other GPU hardware platforms.

Using an inexpensive graphics card and a standard PC, we achieved speeds higher than 4.5 million fits per second for data that is typical of STORM experiments (Fig. 1). Considering recent developments in localization based super-resolution methods, in particular experiments in which >400 million individual fluorophore switching events were recorded,[14] data processing speed has become highly relevant. A tool such as Gpufit could make the difference between a researcher waiting minutes (with Gpufit) or hours (without) before the image can be examined. This rapid feedback has a greater importance than simply reducing waiting time – it enables the quick screening of samples and test conditions, ultimately leading to higher quality image data. Furthermore, we expect that Gpufit will facilitate the adoption of new, computationally demanding data analysis approaches, such as cubic spline fitting,[15] in order to further improve STORM image resolution.

Conclusions

We have developed a GPU-accelerated implementation of the Levenberg Marquardt algorithm, with significantly faster performance as compared with traditional CPU-based software. Gpufit is designed to be general purpose, and as such we expect it to be useful for diverse applications in science and engineering which may depend critically on rapid data processing, e.g. high-speed feedback systems and the recently described MINFLUX method for particle tracking and nanoscale imaging.[16] The Gpufit library is accessible from most programming environments, and the automatic configuration of GPU-specific parameters makes it simple to work with in practice. Finally, Gpufit makes efficient use of GPU computing resources to achieve high performance. Using Gpufit with our own hardware, we achieved a 42-fold improvement in execution time for batch-processing of curve fitting tasks, with no loss of precision or accuracy.

Gpufit is published as an open source software project, and is available via the Github software repository. Open source software development is advantageous from several standpoints: it enhances code integrity through review by users, and offers the possibility for users to fix bugs, add features, etc.[17] In addition, open source code makes the workings of an algorithm transparent to the user, which may be a crucial factor when a “black box” software tool is not sufficient, such as in scientific data analysis applications. Finally, we note that Marquardt’s original paper introducing his algorithm, published in 1963, concludes with the following sentence: “A FORTRAN program … embodying the algorithm described in this paper is available as IBM Share Program No. 1428”.[1] The “Share program” referred to by Marquardt represents one of the origins of the open software concept,[18] and we find it appropriate that Gpufit should be made available in a similar manner.

Methods

Levenberg-Marquardt algorithm

The LMA provides a general numerical procedure for fitting a non-linear model function to a set of data points. It may be considered as a combination of the method of steepest descent and Newton's method, having a high probability of convergence even when the initial parameter estimates are poor, and fast convergence near the minimum. The standard algorithm, as described by Marquardt,[1] minimizes iteratively the general least squares equation:

$$\chi^2(\vec{a}) \;=\; \sum_{n=0}^{N-1}\left(f_n(\vec{a})-z_n\right)^2 \tag{1}$$

where f_n are the model function values, $\vec{a}$ is the vector of model parameters, and z_n are the set of N data points. To find the minimum of $\chi^2(\vec{a})$ (chi-square), the algorithm performs an iterative search of the parameter space for a coordinate where the gradient of the function equals zero, i.e. $\nabla\chi^2(\vec{a})=0$. The gradient is approximated by a Taylor expansion:

$$0 \;=\; \nabla\chi^2\left(\vec{a}_i+\vec{\delta}_i\right) \;\approx\; \nabla\chi^2(\vec{a}_i)+H_{\chi^2}(\vec{a}_i)\,\vec{\delta}_i \tag{2}$$

where $H_{\chi^2}(\vec{a}_i)$ is the Hessian matrix of $\nabla\chi^2(\vec{a}_i)$, and $\vec{\delta}_i$ is a small correction to $\vec{a}_i$ (the index i corresponds to the iteration number). The expression for the Hessian includes the first and the second partial derivatives of $\chi^2(\vec{a})$, however terms containing the second derivatives are assumed negligible and ignored. Solving (2) for $\vec{\delta}_i$ yields the Newton step for the minimization of $\chi^2(\vec{a})$. Up to this point, the LMA is equivalent to Newton's Method.

A special characteristic of the LMA is the damping factor λ which controls the step size of each iteration by modifying the diagonal elements of the Hessian:

$$H'_{\chi^2,kl}(\vec{a}) \;=\; \begin{cases} H_{\chi^2,kl}(\vec{a}), & \text{for } k\neq l \\ H_{\chi^2,kl}(\vec{a})\cdot(1+\lambda), & \text{for } k=l \end{cases} \tag{3}$$

where k and l are the matrix indices. The positive factor λ is initialized with a small value, and thus initially the algorithm behaves like Newton's method. As the algorithm iterates, if the value of chi-square in the latest iteration is smaller than in the previous step, λ is decreased by a constant factor ν. Otherwise, λ is increased by the same factor. Increasing λ causes the LMA to tend towards the behavior of the method of steepest descent. In this manner, the LMA adjusts between the two methods, as the minimum is approached.

By transposing equation (2) and applying the damping factor (3), a system of linear equations is obtained which may be solved for $\vec{\delta}_i$, e.g. by the Gauss-Jordan method:[3]

$$\vec{\delta}_i \quad = \quad -H'^{-1}_{\chi^2}(\vec{a}_i)\cdot\nabla\chi^2(\vec{a}_i)\ . \tag{4}$$

If the iteration is successful (chi-square decreased), the difference $\vec{\delta}_i$ is added to the previous parameter values:

$$\vec{a}_{i+1} \quad = \quad \vec{a}_i \quad + \quad \vec{\delta}_i\ . \tag{5}$$

The damping factor λ is updated after each iteration, and convergence is tested. Any convenient convergence criterion may be used, but in general the overall convergence of the LMA depends on the relative size of the parameter adjustment in each iteration. As originally set out by Marquardt, with a reasonable choice of r and ε (e.g. $r = 10^{-3}$ and $\varepsilon = 10^{-5}$), the algorithm has converged when

$$\frac{\left|\delta_{i,j}\right|}{r+\left|a_{i,j}\right|} \quad < \quad \varepsilon \tag{6}$$

is satisfied for all parameters, where r is a small positive constant (to avoid division by zero), i is the iteration number, and j is the parameter index.

Estimators of best fit

Least squares estimation (LSE) is a common method for finding the parameters which yield the minimal deviation between observed data and a model function. The standard LMA minimizes the general LSE formula given by equation (1). However, it is also possible to include weighting factors in the calculation of chi-square, for example:

$$\chi^2_{\mathrm{LSE}}(\vec{a}) \quad = \quad \sum_{n=0}^{N-1}\left(\frac{f_n(\vec{a})-z_n}{\sigma_n}\right)^2 \tag{7}$$

where σ_n represents the uncertainty (standard deviation) of the data. This allows the precision of each data point to be taken into account.

In cases where the uncertainties of the data points are Poisson distributed, a maximum likelihood estimator (MLE) yields more precise parameter estimates than the LSE.[10,11] In this situation it is beneficial to use an alternative estimator with the LMA. A procedure has been described[7] in which the LSE formula (1) is replaced by the MLE equation for Poisson deviates, as follows:

$$\chi^2_{\mathrm{MLE}}(\vec{a}) \quad = \quad 2\left[\sum_{n=0}^{N-1}\left(f_n(\vec{a})-z_n\right) \ - \sum_{n=0,z_n\neq 0}^{N-1} z_n \ln\left(\frac{f_n(\vec{a})}{z_n}\right)\right]. \tag{8}$$

Using this estimator within the context of the LMA is relatively simple to implement, requiring only the calculation of the gradient and Hessian matrix of $\chi^2_{\mathrm{MLE}}(\vec{a})$, and the calculation of $\chi^2_{\mathrm{MLE}}(\vec{a})$ itself. As before, in the calculation of the Hessian matrix, terms containing second partial derivatives are ignored.

Gauss-Jordan elimination

The Gauss-Jordan method is a procedure for solving linear equation systems in matrix form, and we used this approach to find the solution of the set of equations (4) in the LMA. In addition, we parallelized the Gauss-Jordan algorithm for use with Gpufit. Our GPU implementation employs partial pivoting to ensure precise and numerically stable calculations.[3] The sorting step in the pivot operation was accomplished on the GPU by means of a parallel bitonic merge sort.[19]

Computer hardware

All tests were executed on a PC running the Windows 7 64-bit operating system and CUDA toolkit version 8.0. The PC hardware included an Intel Core i7 5820K CPU, running at 3.3 GHz, and 64 GB of RAM. The graphics card was an NVIDIA GeForce GTX 1080 GPU with 8 GB of GDDR5X memory. All source codes, including the Cpufit, Gpufit, and C/C++ Minpack libraries, were compiled using Microsoft Visual Studio 2013, with compiler optimizations enabled (release mode). Additional details of the test conditions and instructions for compiling the Cpufit and Gpufit software libraries are provided in the Supplementary Information and the Gpufit documentation.

Software for comparison tests

To evaluate its performance, Gpufit was compared against an equivalent CPU-based algorithm (called Cpufit) and two other curve fitting libraries: Minpack[8,20] and GPU-LMFit.[9] Cpufit is a standard implementation of the LMA based on published examples,[1,3,21] which we wrote in C++ for execution on the CPU. C++ Minpack is an open-source C/C++ implementation of MINPACK which runs on the CPU (we used the fitting function `lmder()` from this library).[8,20] GPU-LMFit is a closed-source implementation of the LMA (publicly available in 32-bit binary format) which runs on the GPU and provides the option of using LSE or MLE as the estimator.[9]

Code availability

The full source code for Gpufit, including external bindings, is available for download from a public software repository located at http://www.github.com/gpufit/Gpufit. The source code for the CPU-based reference algorithm, Cpufit, is included. The repository also contains a User's manual with instructions for building the source code, which may be read online at http://gpufit.readthedocs.io.

Data availability

The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

References

1 Marquardt, D. W. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *Journal of the Society for Industrial and Applied Mathematics* **11**, 431-441 (1963).

2 Moré, J. J. in *Numerical analysis* 105-116 (Springer, 1978).

3 Press, W. H., Teukolsky, S. A., Vetterling, W. T. & Flannery, B. P. *Numerical Recipes in FORTRAN; The Art of Scientific Computing*. (Cambridge University Press, 1993).

4 Du, P. *et al.* From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming. *Parallel Computing* **38**, 391-407 (2012).

5 Tomov, S., Nath, R., Du, P. & Dongarra, J. *MAGMA Users' Guide*, <http://icl.cs.utk.edu/magma/> (2009).

6 Dongarra, J. *et al.* in *Numerical Computations with GPUs* (ed Volodymyr Kindratenko) 1-26 (Springer, 2014).

7 Laurence, T. A. & Chromy, B. A. Efficient maximum likelihood estimator fitting of histograms. *Nature methods* **7**, 338-339 (2010).

8 Moré, J. J., Garbow, B. S. & Hillstrom, K. E. User guide for MINPACK-1. (1980).

9 Zhu, X. & Zhang, D. Efficient Parallel Levenberg-Marquardt Model Fitting towards Real-Time Automated Parametric Imaging Microscopy. *PLoS ONE* **8**, e76665 (2013).

10 Abraham, A. V., Ram, S., Chao, J., Ward, E. S. & Ober, R. J. Quantitative study of single molecule location estimation techniques. *Optics express* **17**, 23352-23373 (2009).

11 Smith, C. S., Joseph, N., Rieger, B. & Lidke, K. A. Fast, single-molecule localization that achieves theoretically minimum uncertainty. *Nature methods* **7**, 373-375 (2010).

12 Bates, M., Huang, B. & Zhuang, X. Super-resolution microscopy by nanoscale localization of photo-switchable fluorescent probes. *Curr. Opin. Chem. Biol.* **12**, 505-514 (2008).

13 Schnitzbauer, J., Strauss, M. T., Schlichthaerle, T., Schueder, F. & Jungmann, R. Super-resolution microscopy with DNA-PAINT. *Nat. Protocols* **12**, 1198-1228 (2017).

14 Legant, W. R. *et al.* High-density three-dimensional localization microscopy across large volumes. *Nat. Methods* **13**, 359-365 (2016).

15 Babcock, H. P., Moffitt, J. R., Cao, Y. & Zhuang, X. Fast compressed sensing analysis for super-resolution imaging using L1-homotopy. *Optics Express* **21**, 28583-28596 (2013).

16 Balzarotti, F. *et al.* Nanometer resolution imaging and tracking of fluorescent molecules with minimal photon fluxes. *Science* (2016).

17 Morin, A., Urban, J. & Sliz, P. A Quick Guide to Software Licensing for the Scientist-Programmer. *PLOS Computational Biology* **8**, e1002598 (2012).

18 Wikipedia. *SHARE (computing) --- Wikipedia The Free Encyclopedia*, <http://en.wikipedia.org/w/index.php?title=SHARE_(computing)> (2016).

19 Buck, I. & Purcell, T. in *GPU Gems* 621-636 (Addison Wesley, 2004).

20 Devernay, F. *C/C++ Minpack*, <http://devernay.free.fr/hacks/cminpack/> (2007).

21 Markwardt, C. B. in *Astronomical Data Analysis Software and Systems XVIII* Vol. 411 *ASP Conference Series* (eds D. Bohlender, P. Dowler, & D. Durand) 251-254 (Astronomical Society of the Pacific, Quebec, Canada, 2009).

Acknowledgements

We would like to thank Dr. C. Wurm and Dr. E. Rothermel for the preparation of the sample used for STORM imaging, and also Dr. V. Cordes for providing the primary antibody against GP210. We thank N. Warmbold for contributions to an early version of the Gauss-Jordan algorithm used in the Gpufit source code. M.B. gratefully acknowledges funding from the European Molecular Biology Organization (ALTF 800-2010) and the Max Planck Society.

Author Contributions

M.B., B.T., and B.S. conceived the project. A.P., B.T., J.K., and M.B. wrote the Cpufit and Gpufit source code. A.P. and M.B. carried out the quantitative evaluation of Gpufit. M.B. performed the STORM experiment. A.P. and J.K. wrote the external bindings for Gpufit. J.K. created the usage examples. J.K., A.P., M.B., and B.T. wrote the documentation. M.B., B.T., and B.S. supervised the research. M.B. wrote the manuscript.

Additional Information

Competing financial interests

The authors declare no competing financial interests.

Przybylski et al., Figure 1

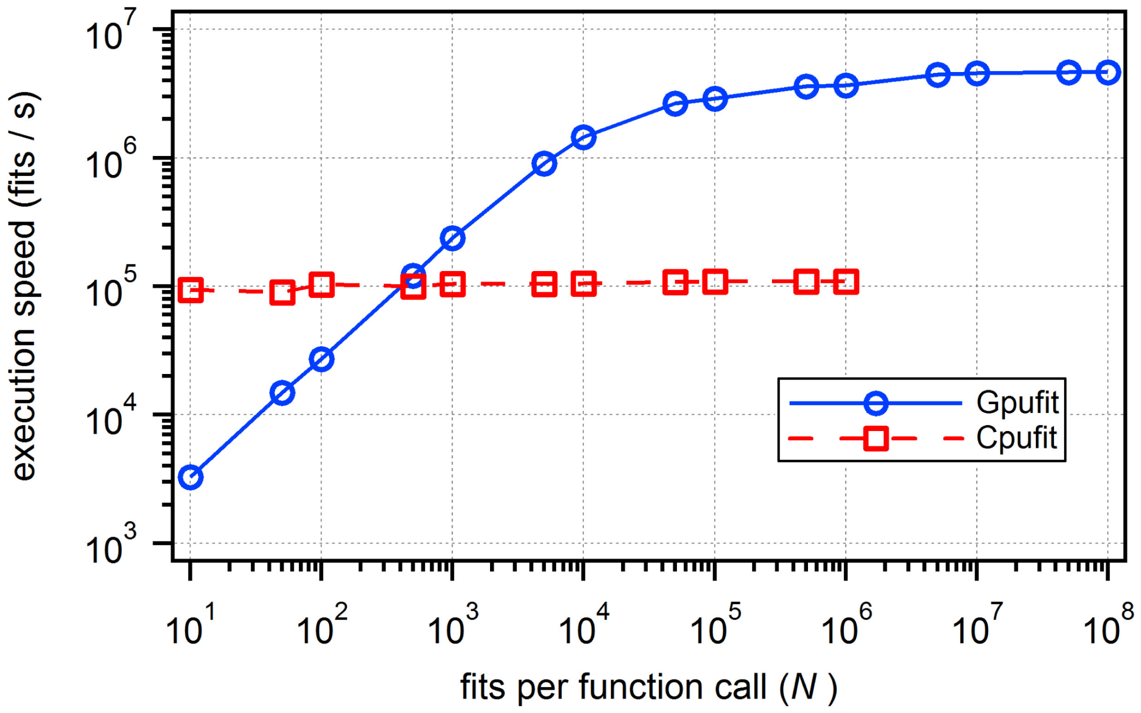


Figure 1: Execution speed of the Levenberg Marquardt algorithm on the CPU or the GPU, as a function of the number of fits processed. For small numbers of fits, GPU fitting is slower than on the CPU, due to the extra time spent copying data between the CPU and the GPU. For larger numbers of fits, however, the GPU significantly out-performs the CPU, as it can better take advantage of its parallel architecture. In this example, the maximum speed achieved was 4.65 x 10^6 fits per second (dependent on the specifics of the hardware and the fit data). For detailed test conditions, see the Supplementary Information.

Przybylski et al., Figure 2

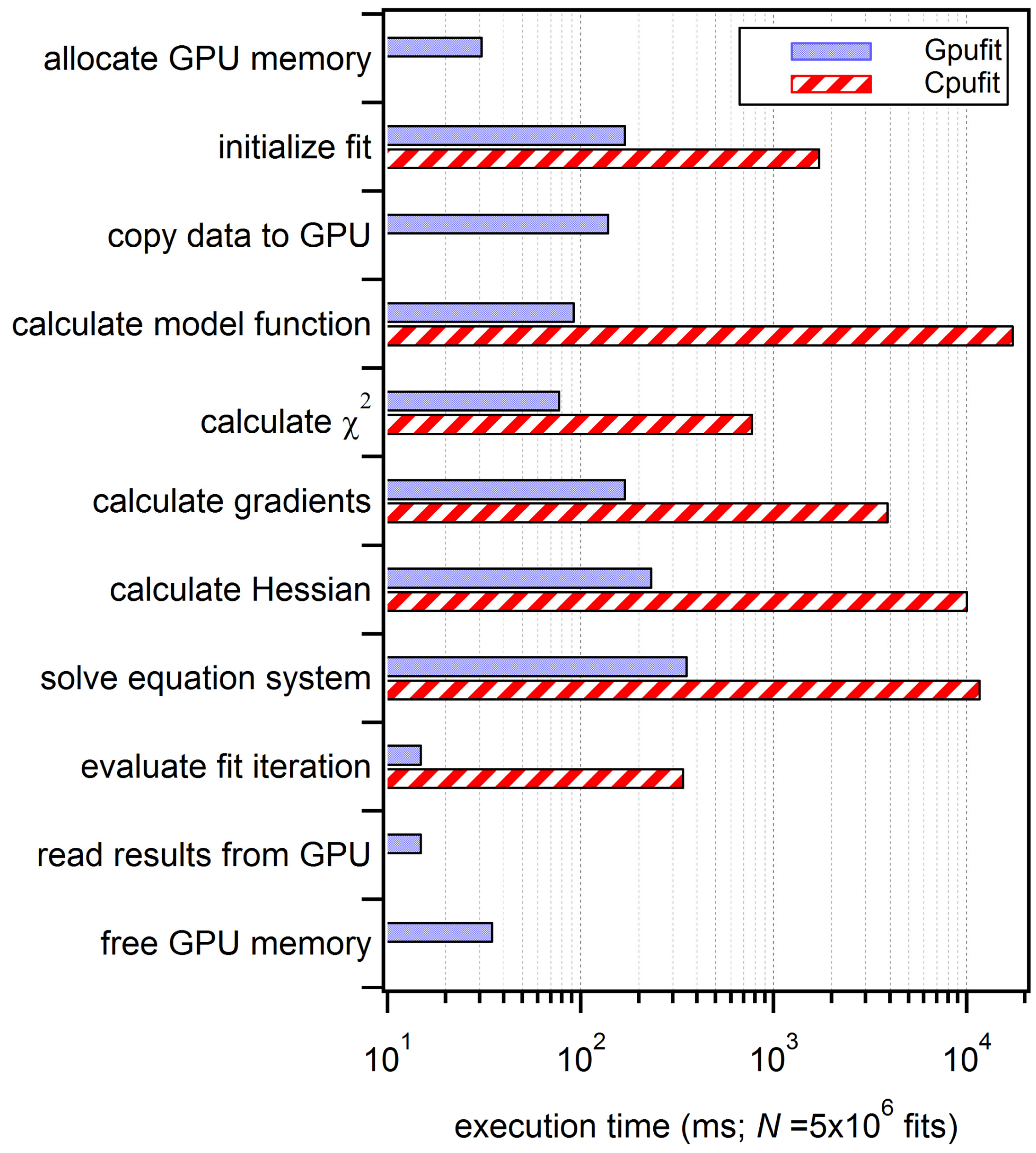
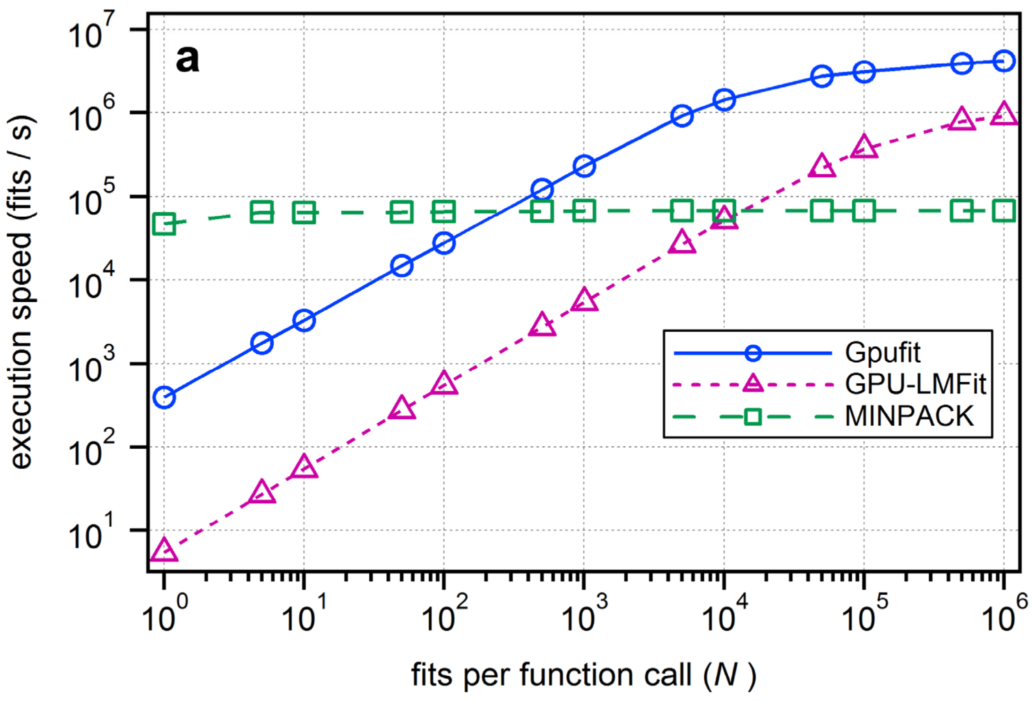


Figure 2: Comparison of execution times for each code section of Cpufit and Gpufit. The bars correspond to the execution time for processing a dataset consisting of five million fits. Four additional steps are required for Gpufit: GPU memory allocation, data transfer to and from the GPU, and GPU memory de-allocation. All code sections require less time when executed on the GPU, with the largest differences corresponding to the bottlenecks of the Cpufit algorithm: calculation of the model function, the gradients, the Hessian, and the solution of the equation system. For detailed test conditions, see the Supplementary Information.

Przybylski et al., Figure 3



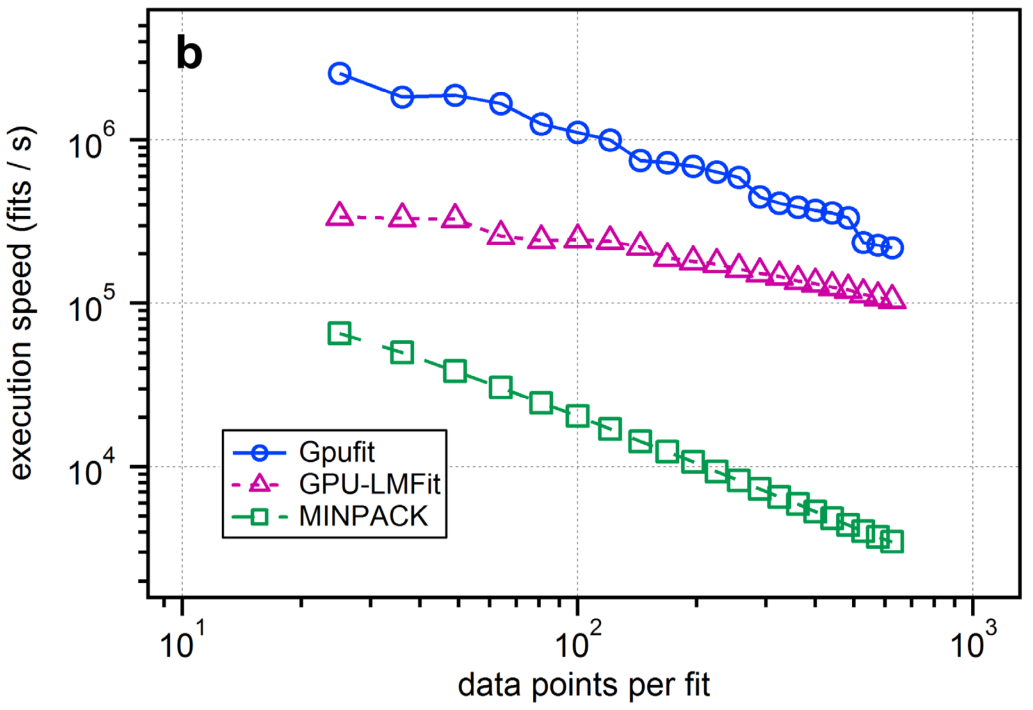


Figure 3: Processing speed comparison between three fitting libraries: Gpufit, MINPACK, and GPU-LMFit. (**a**) The execution speed as a function of the number of fits processed per call. (**b**) Execution speed as a function of the data size (number of data points) per fit. For smaller tasks (smaller numbers of fits per call or smaller data sizes) the Gpufit library makes more efficient use of the GPU processing resources, outperforming GPU-LMFit by more than one order of magnitude.

Przybylski et al., Figure 4

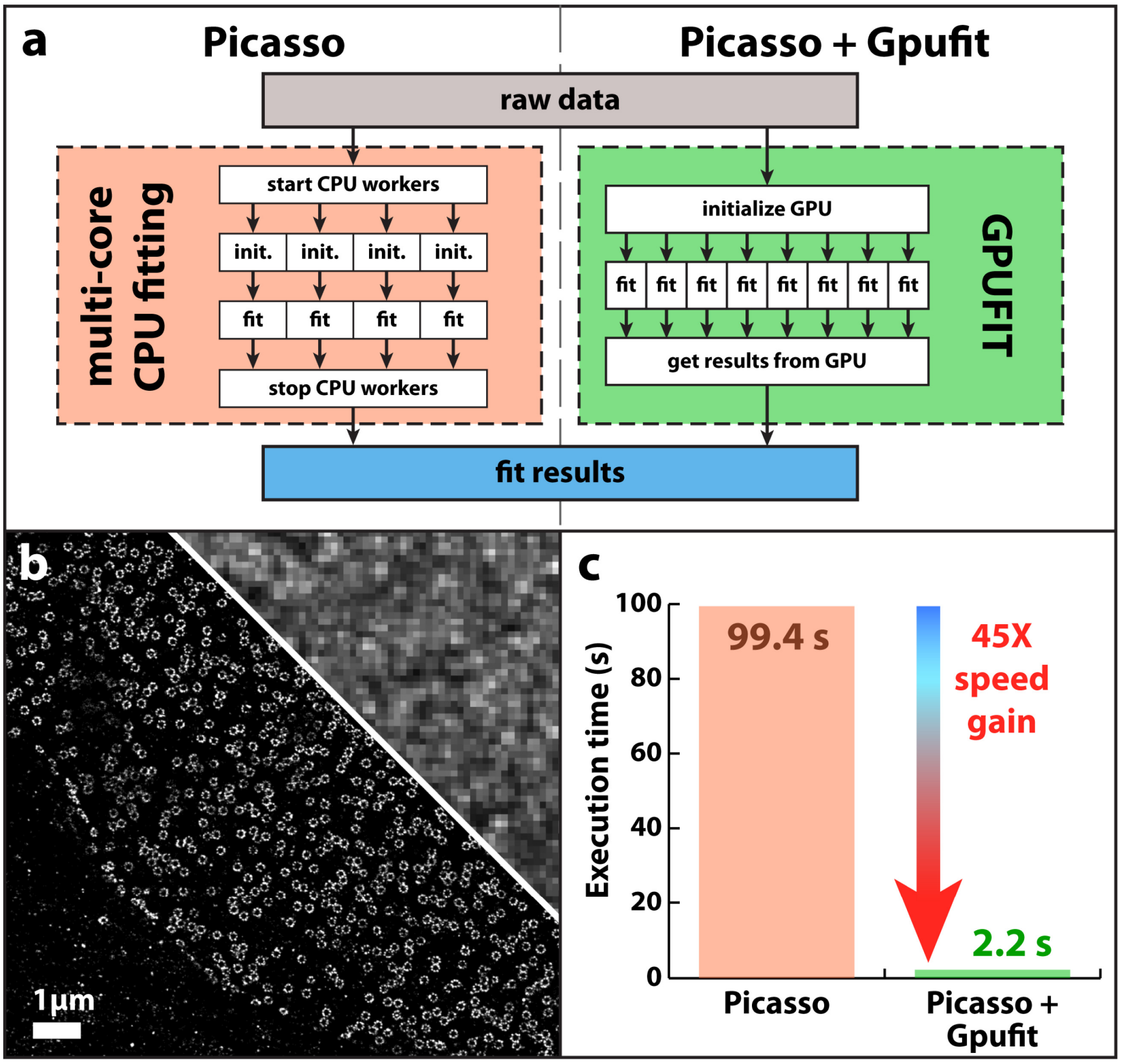


Figure 4: Accelerated STORM analysis in Picasso. (**a**) Schematic diagram comparing the multi-core CPU based fitting of Picasso (left) vs. parallelized GPU-based fitting (right). Picasso uses multiple cores ("workers") of the CPU to simultaneously carry out the fits. We replaced this code with Gpufit, which has a much greater capability for parallelization on the GPU. (**b**) STORM image of GP210, a component of the nuclear pore complex in eukaryotic cells. The conventional epi-fluorescence image is shown in the top right corner, to provide a resolution comparison. Data processing was performed in Picasso, and required 3.6 million individual curve fitting operations. (**c**) Execution time of the curve fitting process using the published Picasso software and the modified Picasso software including Gpufit. The time required for curve fitting was reduced by a factor of 45 after including Gpufit in the Picasso application.