19th International Congress on Modelling and Simulation, Perth, Australia, 12–16 December 2011
http://mssanz.org.au/modsim2011

Heterogeneous parallel 3D image deconvolution on a cluster of GPUs and CPUs

L. Domanski [a], **T. Bednarz** [b], **P. Vallotton** [b], **J. Taylor** [b]

[a]*Advance Scientific Computing, CSIRO, Australia*
[b]*Mathematics, Informatics, & Statistics, CSIRO, Australia*
Email: Luke.Domanski@csiro.au

Abstract: This paper presents a heterogeneous computing algorithm for 3D Richardson-Lucy image deconvolution applicable for use on single heterogeneous workstations, all the way up to large distributed memory clusters consisting of many heterogeneous nodes. We demonstrate our solution on a cluster of nodes containing multiple CPU cores and GPUs. The algorithm uses a combination of message passing and massively-multicore programming technologies to achieve nested levels of parallelism, ranging from course grained domain decomposition across worker processes to more fine grain parallelism within worker processes utilising GPUs. The work distribution and worker framework is abstracted from the type of processor architecture used for core algorithm calculation by different worker processes. Allocation of computational resources (different processors or cores) to workers is handled collaboratively by the worker processes on each cluster node using efficient Operating System level counting semaphores, avoiding the need to manage computational resources centrally on the cluster.

The tested implementation utilises MPI (Message Passing Interface) for parallelisation across the cluster, CUFFT and custom written kernels for parallelisation of algorithm components on the GPU, and the highly tuned MKL math library for computations on the CPU. Result show that utilising a collection of different processor types on available nodes can provided performance benefits over the use of a single type alone. It is common to find heterogeneous workstations with a smaller number of high performance accelerator processors than general purpose processor cores. In these cases, when considering the number of cluster nodes utilised versus performance, using all available processors on a node generally provides a performance gain whilst using the same number of nodes, or allows us to achieve similar performance using fewer nodes. We discuss situations where using multiple processor types at once can inhibit performance, and make recommendations on when such an approach would or would not be advantageous.

Keywords: image deconvolution, image restoration, heterogeneous computing, GPU, high performance computing, parallel programming

1 INTRODUCTION

Deconvolution is an important operation in many areas of science, including astronomy (Hanisch and White, 1993), microscopy (McNally et al., 1999; Wallace et al., 2001), and medical imaging. It reduces the effects of blurring introduced during image capture, revealing objects and details that may not have been visible in the raw image. The blurring is characterised by the image of a point light or impulse source in an imaging device (the point spread function, PSF). Each point of an observed object produces one of these point spread functions (Fig. 1), hence, the observed image g can be modeled as a convolution of the true object f with the PSF p plus a noise value n at each point, taken from a random distribution characterizing the devices counting statistics:

$$g(x) = \int f(\xi)p(x,\xi)d\xi + n(x) \quad (1)$$

Algorithm 1 Richardson-Lucy(g,P,k)

1: **return** $f = f_r$
2: $f_r = g$
3: **for** 1 **to** k **do**
4: {apply imaging model to estimate}
5: $F_r = \text{fft}(f_r)$
6: $G_r = F_r \times P$
7: $g_r = \text{ifft}(G_r)$
8: {calculate correction vector}
9: $d = g/g_r$
10: $D = \text{fft}(d)$
11: $C = D \times P^*$
12: $c = \text{ifft}(C)$
13: {apply correct to get new estimate}
14: $f_r = f_r \times c$
15: **end for**

When the PSF is assumed to be identical at every location in the sample, a spatially invariant (SI) PSF $p(x,\xi) = p_{si}(x-\xi)$ can be used. In some systems the PSF changes based on location in the sample space, and a spatially variant (SV) function $p(x,\xi) = p_{sv}(\xi, x-\xi)$ provides a more accurate imaging model,giving the intensity of light produced at image point x by a point light source at point ξ in object space (Preza and Conchello, 2003). Given these imaging models, the process of deconvolution seeks the unknown true object f by inverting Equation 1.

The Convolution Theorem for Fourier Transforms leads to a direct inversion of a noise free Equation 1 through an element-wise division in the Fourier domain, of the observed image by the PSF

$$F(\omega) = G(\omega)/P(\omega) \quad (2)$$

where capital letters denote the Fourier transform of the associated lower case functions. Direct methods such as these do not account well for the random noise component n in Equation 1 and can amplify the noise (Lucy, 1994), rendering the output "deprived of any physical meaning" (Bertero and Boccacci, 2002). Iterative methods that try to account for the statistical noise, or regularise its effects, while converging towards an appropriate "solution" are, therefore, often favored despite their increased computational load (see Bertero and Boccacci, 2002, for an overview).

A popular iterative algorithm for achieving this is the Richardson-Lucy (RL) algorithm (Richardson, 1972; Lucy, 1974) which provides the Maximum Likelihood Estimator for f in the presence of Poisson noise, by iteratively re-applying a noiseless imaging model (Eq. 1) to an improving estimate of f. The algorithm is defined in the discrete case (replacing integrals with summation operators) by

$$f_{r+1}(\xi) = f_r(\xi)\sum_x \frac{g(x)}{g_r(x)}p(x,\xi), \quad \text{where} \quad (3)$$

$$g_r(x) = \sum_\xi f_r(\xi)p(x,\xi) \quad (4)$$

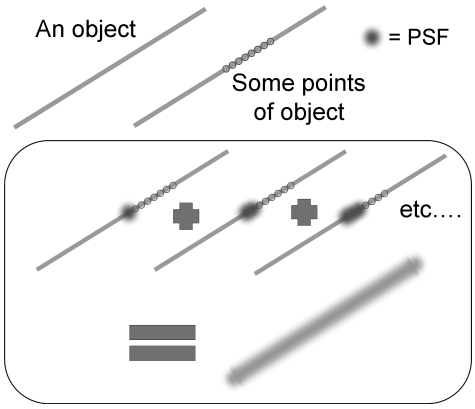

Figure 1. Image formation.

Given an initial estimate $f_r = f_0$ the algorithm iteratively:
a) applies the noise free imaging model to f_r producing a blurred estimate g_r, b) creates a correction vector by convolving the ratio of the observed image g to blurred estimate g_r by the transpose of the PSF (note domains of summation in Eq. 3 and 4), c) multiplies the current estimate f_r by the correction to

get a new estimate. Computation stops after a defined number of iterations or some stopping criteria is reached. The discrete Convolution Theorem can be applied here to perform convolutions "efficiently" by multiplication in the Fourier domain, through the use of Fast Fourier Transforms (FFTs); a class of algorithms for performing Discrete Fourier Transforms efficiently.

The RL algorithm can become computationally intensive when a large number of iterations are required or in the case of large images. In this paper we describe a heterogeneous computing algorithm that accelerates the Richardson-Lucy algorithm on a distributed memory cluster of heterogeneous workstations containing multiple GPUs and CPUs. As in the work of Domanski et al., (2009a) the implementation is capable of taking advantage of all compute resources on a node. We discuss the pros and cons of using a heterogeneous algorithm, and when utilising only GPUs and not CPUs is appropriate. Many of our discussions and findings are generally applicable to heterogeneous parallel computing, and we generalise discussions where appropriate.

2 BACKGROUND

A common approach to implementing the Richardson-Lucy algorithm in parallel on a cluster of PCs (Boden et al., 1996; Shearer et al., 2001; Pawliczek et al., 2010) is based on the sectioned method of Trussell and Hunt (1978b). In such an approach the image can be segmented into a number of abutting sub-tiles and the RL algorithm is performed on each tile individually. A guard-band of at least half the PSF diameter is included around each tile to allow for the additional information required during convolution at tile boundaries. Contents of the guard-bands can be exchanged between neighbouring tiles at each RL iteration to increase accuracy, however, this increases algorithm complexity and synchronisation barriers in a parallel setting. We find that a course grained approach with no guard-band exchange provides sufficient quality for our use in 3D microscopy. On completion the guard-bands are discarded and the tiles are recombined to form the final deconvolved image. Since the processing of each tile is performed in isolation, the tiles can easily be distributed to multiple PCs or processors and processed in parallel. The sectioned method also provides a convenient way of incorporating a SV model (Trussell and Hunt, 1978a). By processing each tile using a SI model, but using a different PSF for each tile based on its location, spatially variability is achieved at the tile level. Tile size can be adjusted to accommodate greater or less extents of variability, thus minimising deconvolution artifacts between adjacent tiles.

One can also parallelise the FFT algorithm used to facilitate frequency domain convolutions. Recent studies (Govindaraju et al., 2008; Nukada et al., 2008; Volkov and Kazian, 2008) describe implementations utilising modern massively multithreaded programming languages, and have provided significant performance improvements over popular optimised CPU libraries (Frigo and Johnson, 2005; Intel Corporation, 2011). This type of acceleration can be used for both sectioned methods (FFT on individual tiles) and non-sectioned methods (FFT on whole image).

Fung and Mann (2008) provide brief results for a non-sectioned GPU accelerated RL algorithm using CUFFT (NVIDIA Corporation, 2011) and custom kernels. They show speedups over the RL implementation in Matlab (MathWorks, 2011), but do not provide discussions of the algorithm. More comprehensive studies (Domanski et al., 2009b; Quammen et al., 2009) have explored the use of GPU FFT libraries combined with custom written GPU kernels for the acceleration of the non-sectioned RL algorithm. Domanski et al., (2009a) describe a heterogeneous implementation of the sectioned RL algorithm that utilises all CPUs and GPUs on a single workstation to perform RL processing, achieving speedups of up to 10x over a serial CPU algorithm based on MKL (Intel Corporation, 2011) and up to 5x over a multicore MKL version. While a non-sectioned approach using a single GPU out performs a sectioned method executed across all workstation processors, utilising a sectioned methods supports spatially variant deconvolution as mentioned above. We extend this previous work to distributed memory cluster environments (i.e. multiple workstations).

3 ALGORITHM

3.1 Heterogeneous parallel program framework

The work is distributed using MPI and a master-worker approach. The master process runs on a single general purpose processing resource (CPU core in our case) of the cluster and is responsible for segment-

ing the input image into tiles (work units), allocating work units to different MPI workers, and collating the results back into the final image. Each worker process also runs on a single general purpose processing resource and is responsible for requesting work units from the master, performing RL processing on each allocated tile, and sending the results back to the master.

Simple load balancing is achieved using a FIFO queue within the master process, which receives and responds to worker requests for work units as they are arrive from the MPI message subsystem. Non-blocking (asynchronous) MPI transfers are used to send and receive tiles so that the master can continue servicing work unit requests while the MPI subsystem concurrently transfers data. MPI implementations providing hardware assisted buffer communication (e.g. Infiniband) can then be exploited where available so that the CPU is free to continue other tasks. Of course, requests for work must also come through the MPI subsystem, and hence, the level of concurrency achieved between the master's work unit service loop and the transfer of tile data is dependent on the MPI vendor's implementation.

3.2 Hardware specific implementations and heterogeneity

To allow the RL algorithm to run on a range of computational resources (in our case CPUs and GPUs) we consider tile processing in two parts; 1) the RL algorithm's *outer loop*, which repetitively calls a single iteration of the core RL operations (increasing r in Eq. 3), 2) the RL *core operations* themselves (Eq. 3 and 4). Both CPU and GPU versions of the the core operation code are implemented on our system. At run time, each MPI worker decides independently what type of processing resource (processor hardware) they will execute their core RL operations on, and selects the appropriate implementation for use within the outer loop. The RL algorithm's outer loop is executed on the processing resource on which the MPI worker is running (generally a host's CPU core) and calls the core operation implementation chosen above. This can be managed easily in languages like C and C++ using function pointers, otherwise, one can use flags and conditional statements to call the appropriate core operation code.

Both the CPU and GPU core operation code follow the same general structure shown in Algorithm 1. In both cases we utilise optimised third party FFT libraries as black boxes to perform the Discrete Fourier Transforms; Intel MKL for the CPU and NVIDIA CUFFT for the GPU. Operations responsible for the element wise division and multiplication of complex and real matrix values are performed using in house code. Emphasis was placed on simplicity of implementation of these operations, thus, more efficient implementations might be available, particularly in the case of the CPU where vector instruction code and libraries could be used.

For the CPU core operations we utilise a single thread, because running an N thread FFT algorithm is usually less efficient than using N independent workers running single thread FFTs for sectioned deconvolution Domanski et al., (2009a) due to the overhead and imperfect scaling of multi-threaded FFT algorithms. For the GPU code, elementwise matrix operations are performed using custom written GPU kernels. Working data is left in GPU memory between custom kernel and FFT library calls and also between algorithm iterations. This avoids unnecessary transfers between host and device memory.

3.3 Compute resource management

With recent increases in the number and type of processors and commodity systems with heterogeneous processing architectures, workstations and distributed memory clusters containing a range of processor types are becoming more prevalent. It is often desirable to make use of different processor architecture for different tasks, or to utilise all available processing resources simultaneously to perform and single larger task. Our heterogeneous RL implementation ensures workers take full advantage of available processing resources on the cluster, while avoiding resource contention and over subscription. To achieve this we have implemented a simple resource managagement and allocation system that does not require the master process to have knowledge of the available computer resources on the cluster, nor for the workers to know explicitly what host they are running on, or to communicate explicitly what they are doing to other workers.

Workers use a resource discovery function and operating system level counting semaphores to lock available resources. The discovery function simply returns a count of each type of processing resource on the worker's host node. Semaphores guarding each type of resource are created, and workers try to claim

processing resources by inspecting and waiting on the appropriate semaphores, starting with resources that have the most efficient RL implementations and progressing to slower implementations if resources are unavailable. The value of the semaphore at the time of the wait success indicates to the worker which enumerated resource of a particular type it should use. This approach ensures that the resources on each node are used in a combination that leads to the best performance, as the most efficient resources are consumed first.

All workers in the MPI application use semaphores with the same name. Because semaphores are publicly accessible locks at the scope of a host operative system, this effectively implements resource management at the cluster node (host) level without workers being aware of what host they run on, or what workers on other hosts are doing. They only try to get the best processor available based on the semaphores on their host. The creation and initialisation of the semaphores are handled by the first worker to check on the semaphore on each host. If the semaphore does not exist yet on the host, the worker assumes it should be created and initialises it with the values returned by the discovery function.

Note that a worker utilising an non-host processor (GPU) for RL core operations also requires a general purpose host resource (CPU core) to run the MPI worker and RL algorithm's outer loop (Sec. 3.1). Our resource management scheme can be modified easily to consider this processor usage. This is usefully to ensuring host processing resources (cores) are not over subscribed if worker's are using multiple host cores to implement their RL core operations (e.g a multi-threaded CPU algorithm). Since we have chosen to use a single thread CPU RL implementation (Sec. 3.2), we rely on the cluster's batch system and MPI parameters to allocate no more workers to a node than there are CPU cores. Thus, only GPUs are tracked in our tests, with workers defaulting to CPU RL operations when GPUs are exhausted.

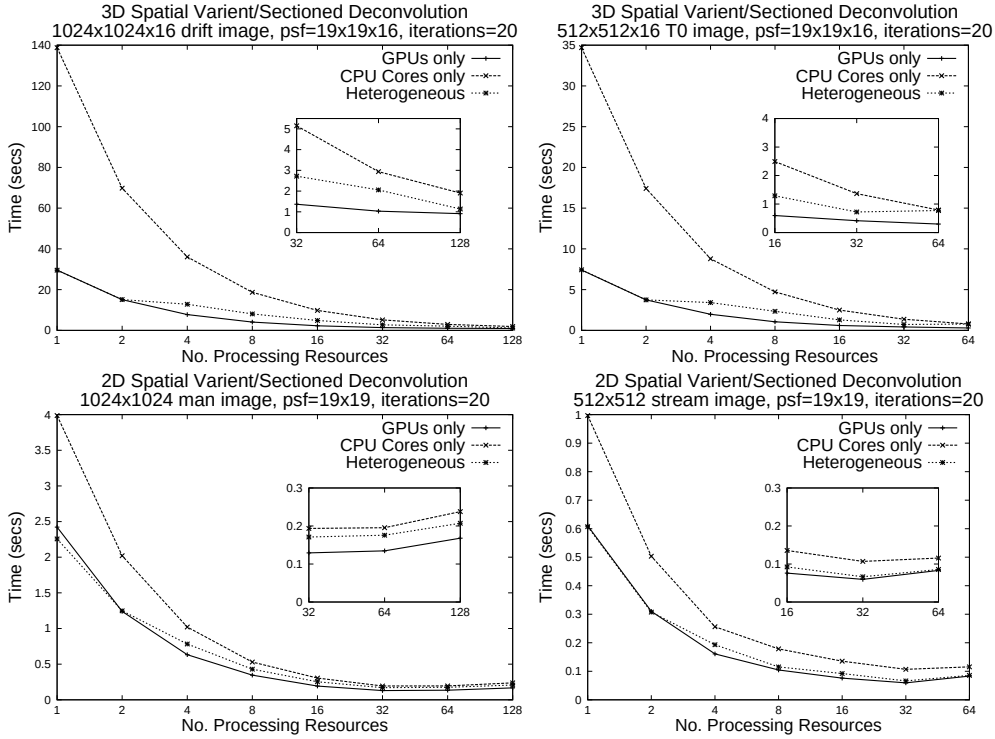

Figure 2. Deconvolution performance results for heterogeneous and homogeneous processing of core algorithm operations on 2D and 3D images. Number of algorithm iterations = 20.

4 RESULTS

Test were performed on a cluster of heterogeneous compute nodes, each containing 32GB of RAM, dual-socket quad-core Intel Xeon E5462 host CPUs, and two NVIDIA M2050 GPUs attached via PCI-Express bus. An Infiniband network interconnect was utilised with accelerated MPI library support. In

addition to the heterogeneous mode presented, we tested CPU and GPU worker only implementations for comparison. Each 2D and 3D image tested was broken into 64×64 tiles in the xy plane. Image loading was taken as constant overhead and excluded from the timings.

Figure 2 shows the performance of the various algorithm modes with increasing number of available processing resources. The minimum number of cluster nodes were selected to provide the required number of processing resources in each case, and differs between modes. In the case of 8 processing resources for example, 1 node produces 2 GPU plus 6 CPU workers for a heterogeneous run and 8 CPU workers for a CPU only run, while 4 nodes are required to produce 8 GPUs workers. Worker for worker, resource for resource, the GPU implementation significantly outperformed the CPU version for 3D images, but at the disadvantage of requiring four times more cluster nodes. The heterogeneous implementation's performance curve, however, tracked closer to the GPU only curve than the CPU only curve, but used the same number of nodes as the CPU only run. This emphasizes the importance of utilising all available processing resources on a node, and not just the fastest processors.

The performance improvement of the GPU and heterogeneous implementations over the CPU only version in the case of the 2D datasets is far less impressive than the 3D results. This is probably related to the selected title size of 64×64 in 2D compared to $64\times64\times16$ in 3D, since GPU FFT algorithms tend to provide greater speedups for larger problem sizes. Indeed, the performance improvement of a single GPU implementation over a single CPU for a *non-sectioned* FFT-based RL algorithm on the tested 2D images (not graphed) was between 8 and 13x. Therefore, better performance improvements are expected by tuning tile sizes when the image size and PSF variation permits. Figure 3 provides scaling results for the 1024×1024 and $1024\times1024\times16$ images. For the 3D image the scaling of the CPU and GPU only versions in fairly good up to between 16 and 32 processing resources (<4 nodes for CPU only, <16 for GPU only), but then starts to drop away from perfect scaling. Since this is an embarrassingly parallel problem, it is expected that this is due to communication congestion at the master node, which is sending work units to all workers as well as receiving their results. The heterogeneous algorithms scaling results are difficult to understand and provide little insight before 8 resources, as the number of GPU and CPUs only doubles simultaneously from 8 onwards. After this point the scaling is fairly good as above until 32 resources. Very poor to negative scaling past 16 resources in the 2D case indicates that the communication limitations are far outweigh the computational advantages. We plan to address this scaling issue in Future work.

A disadvantage of utilising a simple FIFO queuing mechanism ignorant of worker processing speed for work unit distribution (Sec 3.1 and 3.3) is that it cannot discriminate or prioritise the allocation of work to faster processor types. This can result in situations where the final few work units might be allocated to slower workers before faster workers request them, leaving a group of faster workers unutilised. This can have a negative impact on overall execution time when the performance ratio between processor types is high. A processor type t can potentially hinder execution time if there is less than $n + \sum_i R_i m_i$ work units to process, where n is the number of t processors available, m_i is the number of the i^{th} faster processors than t available, and R_i is the speed ratio between m_i and t. In such cases only the faster processor types should be used.

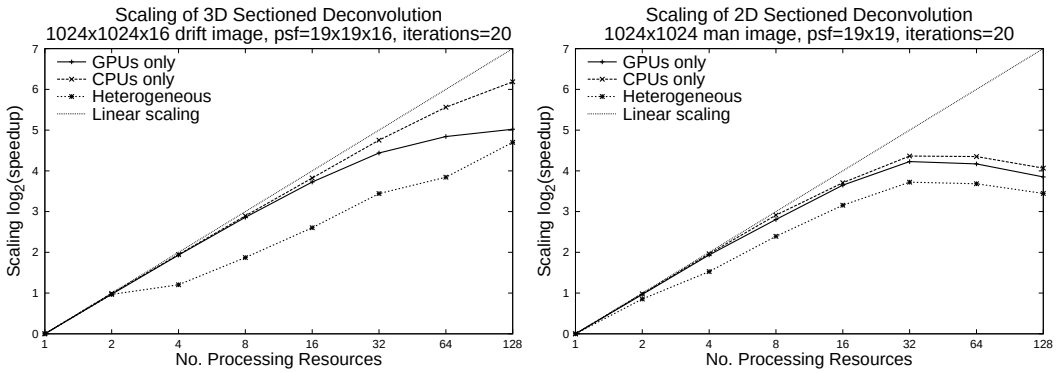

Figure 3. Scaling results for heterogeneous and homogeneous processing of core algorithm operations on 2D and 3D images.

5 CONCLUSIONS

A heterogeneous implementation of the Richardson-Lucy deconvolution algorithm which runs across a cluster of workstations containing CPU and GPU processors was presented. Results showed that utilising all available processor resources on a workstation, both CPU cores and GPUs, leads to greater performance than utilising CPU cores alone. Where there is a high ratio of CPU cores to accelerator processors in each node, the heterogeneous implementation produces performance close to the GPU (accelerator) only version, while utilising fewer overall nodes. This is useful in small cluster installations, where availability of accelerators is limited, or in high load shared systems, where economical use of resources and nodes becomes important for overall throughput.

REFERENCES

Bertero, M. and P. Boccacci (2002). *Introduction to Inverse Problems in Imaging*. IOP Publishing.

Boden, A. F., D. Redding, R. J. Hanisch, and J. Mo (1996). Massively parallel spatially variant maximum-likelihood restoration of hubble space telescope imagery. *J. Opt. Soc. Am. A 13*(7), 1537–1545.

Domanski, L., P. Vallotton, and J. Taylor (2009a). 3d image deconvolution on heterogeneous compute stations. NVIDIA GPU Technology Conference Research Summit.

Domanski, L., P. Vallotton, and D. Wang (2009b). Two and three-dimensional image deconvolution on graphics hardware. In R. Anderssen, R. Braddock, and L. Newham (Eds.), *18th World IMACS Congress and MODSIM09 International Congress on Modelling and Simulation*. Modelling and Simulation Society of Australia and New Zealand and International Association for Mathematics and Computers in Simulation.

Frigo, M. and S. G. Johnson (2005). The design and implementation of FFTW3. *Proceedings of the IEEE 93*(2), 216–231. special issue on Program Generation, Optimization, and Platform Adaptation.

Fung, J. and S. Mann (2008, June 23-April 26). Using graphics devices in reverse: Gpu-based image processing and computer vision. In *Proceedings of the 2008 IEEE International Conference on Multimedia and Expo*, pp. 9–12.

Govindaraju, N. K., B. Lloyd, Y. Dotsenko, B. Smith, and J. Manferdelli (2008). High performance discrete fourier transforms on graphics processors. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, Piscataway, NJ, USA, pp. 1–12. IEEE Press.

Hanisch, R. J. and R. L. White (Eds.) (1993). *The restoration of HST images and spectra–II: proceedings*. Advanced Systems Group, Science Computing and Research Support Division, Space Telescope Science Institute.

Intel Corporation (2011). Intel math kernel library (intel mkl). `http://software.intel.com/en-us/articles/intel-mkl/`, Retrieved September 6 2011.

Lucy, L. B. (1974). An iterative technique for the rectification of observed distributions. *The Astronomical Journal 79*(6), 745–754.

Lucy, L. B. (1994). Astronomical inverse problems. *Reviews in Modern Astronomy 7*, 31–50.

MathWorks (2011). Matlab - the language of technical computing. `http://www.mathworks.com/products/matlab/index.html`, Retrieved September 6 2011.

McNally, J. G., T. Karpova, J. Cooper, and J. A. Conchello (1999, November). Three-dimensional imaging by deconvolution microscopy. *Methods: A Companion to Methods in Enzymology 19*(3), 373–385(13).

Nukada, A., Y. Ogata, T. Endo, and S. Matsuoka (2008). Bandwidth intensive 3-d fft kernel for gpus using cuda. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, Piscataway, NJ, USA, pp. 1–11. IEEE Press.

NVIDIA Corporation (2011). *CUDA CUFFT Library*. `http://developer.download.nvidia.com/compute/DevZone/docs/html/CUDALibraries/doc/CUFFT_Library.pdf`.

Pawliczek, P., A. Romanowska-Pawliczek, and Z. Soltys (2010). Parallel deconvolution of large 3d images obtained by confocal laser scanning microscopy. *Microscopy Research and Technique 73*(3), 187–194.

Preza, C. and J.-A. Conchello (2003). Image estimation accounting for point-spread function depth variation in three-dimensional fluorescence microscopy. In J.-A. Conchello, C. J. Cogswell, and T. Wilson (Eds.), *Three-Dimensional and Multidimensional Microscopy: Image Acquisition and Processing X, Proceedings of the SPIE*, Volume 4964, pp. 135–142. SPIE.

Quammen, C., D. Feng, and R. Taylor, II (2009). Performance of 3d deconvolution algorithms on multi-core and many-core architectures. Technical Report Technical Report TR09-001, University of North Carolina at Chapel Hill Department of Computer Science.

Richardson, W. H. (1972). Bayesian-based iterative method of image restoration. *J. Opt. Soc. Am. 62*(1), 55.

Shearer, A., G. Gorman, T. O'Doherty, W. J. van der Putten, P. McCarthy, and L. Jelen (2001). Parallel image restoration with spatially variant point spread function: description and first clinical results. In M. Sonka and K. M. Hanson (Eds.), *Medical Imaging 2001: Image Processing, Proceedings of SPIE*, Volume 4322, pp. 787–795. SPIE.

Trussell, H. and B. Hunt (1978a, Dec). Image restoration of space-variant blurs by sectioned methods. *Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on 26*(6), 608–609.

Trussell, H. and B. Hunt (1978b, April). Sectioned methods for image restoration. *Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on 26*(2), 157–164.

Volkov, V. and B. Kazian (2008, May). Fitting fft onto the g80 architecture. Technical Report CS258 project report, UC Berkeley, http://www.cs.berkeley.edu/ kubitron/courses/cs258-S08/projects/reports/project6_report.pdf. Retrieved on 01/21/2008.

Wallace, W., L. H. Schaefer, and J. R. Swedlow (2001, November). A workingperson's guide to deconvolution in light microscopy. *BioTechniques 31*(5), 1076–1097.