GPU acceleration towards real-time image reconstruction in 3D tomographic diffractive microscopy

J. Bailleul, B. Simon, M. Debailleul, H. Liu and O. Haeberlé

Laboratoire MIPS, Université de haute-Alsace
61 rue Albert Camus, Mulhouse, France

ABSTRACT

Phase microscopy techniques regained interest in allowing for the observation of unprepared specimens with excellent temporal resolution. Tomographic diffractive microscopy is an extension of holographic microscopy which permits 3D observations with a finer resolution than incoherent light microscopes. Specimens are imaged by a series of 2D holograms: their accumulation progressively fills the range of frequencies of the specimen in Fourier space. A 3D inverse FFT eventually provides a spatial image of the specimen.

Consequently, acquisition then reconstruction are mandatory to produce an image that could prelude real-time control of the observed specimen. The MIPS Laboratory has built a tomographic diffractive microscope with an unsurpassed 130nm resolution but a low imaging speed - no less than one minute. Afterwards, a high-end PC reconstructs the 3D image in 20 seconds. We now expect an interactive system providing preview images during the acquisition for monitoring purposes.

We first present a prototype implementing this solution on CPU: acquisition and reconstruction are tied in a producer-consumer scheme, sharing common data into CPU memory. Then we present a prototype dispatching some reconstruction tasks to GPU in order to take advantage of SIMD-parallelization for FFT and higher bandwidth for filtering operations. The CPU scheme takes 6 seconds for a 3D image update while the GPU scheme can go down to 2 or > 1 seconds depending on the GPU class. This opens opportunities for 4D imaging of living organisms or crystallization processes. We also consider the relevance of GPU for 3D image interaction in our specific conditions.

Keywords: Microscopy, Digital Holography, Diffractive Tomography, 3D Image Reconstruction, Phase Imaging, Synthetic Aperture, GPU acceleration

1. INTRODUCTION

Because of its unique abilities for imaging live specimens, the optical microscope is a key tool for biologists. Within this domain, specific labelling of sub-cellular structures make the fluorescence microscopy the most popular observation method. Moreover, due to its non-elasticity, the fluorescence is compatible with structured illumination techniques (including confocal microscopy) allowing for improved 3D resolution.

However, fluorescence techniques may suffer from unwanted effects such as photo-bleaching or phototoxicity. Furthermore, the labelling of observed structures can turn into a challenging task. This

Further author information: (B. Simon: bertrand.simon@uha.fr, Telephone: +33 (0)3 89 33 76 62)

could explain the regain of interest for techniques which allow for specimen observation without prior specific staining. Among those, we can quote Second or Third-Harmonic Generation (SHG, THG), Coherent Anti-Stokes Raman Spectroscopy (CARS) and even the conventional Transmission Microscopy and its derivatives. However, in techniques such as Phase Contrast or Differential Interference Contrast, the image results from a complex interaction between the incoherent illuminating light and the specimen. As a consequence, the recorded signal is not directly related to the optical properties of the specimen, which often restricts the purpose of the resulting image to morphological studies.

Using coherent light allows for recording holograms and, consequently, obtaining an information on both the amplitude and phase of the light diffracted by the specimen. Taking into account the diffraction effects through an adapted model, one can infer from these records the distribution of the index of refraction over the specimen.[1] This can be achieved under the first Born approximation,[2] considering the diffracted wave as a subset of the Fourier coefficients of the object wave function.[3]

Combining this technique with synthetic aperture methods (e.g variation of the illuminating wave's inclination, specimen rotation, wavelength variation) allows for recording a wider range of objects frequencies, resulting in a finer 3D resolution of the final image. The Tomographic Diffractive Microscopy (TDM) provides a 3-D information which directly depends on the specimen's physical properties, and also provides a finer resolution than the other classical microscopy techniques.

However, TDM output does not provide an interpretable image: a reconstruction of the 3-D data is required. The size and time complexity of a live 3D visualization concurrent to the acquisition process represents a challenging task. Since GPU have proven their efficiency in Computer Generated Holography (CGH),[4,5] we decided to assess their relevance in our specific TDM setup.

In this paper, we will first briefly present the theoretical foundations of the TDM. Then, we will describe the experimental set-up and the principles of the image reconstruction process. We will introduce a standard prototype CPU-based software architecture, updating the reconstructed image during the acquisition for visualization purposes. We will compare it to a mixed CPU and GPU software architecture, which significantly improves the visualization update rate. Using this architecture, and depending on the class of the GPU, one can achieve update rates under a second, which confers real-time monitoring of the acquisition process.

2. IMAGING SYSTEM PRINCIPLE

2.1 Theoretical background

In digital holography, the holograms, which can be considered as projections of an object, are reconstructed numerically. The recorded spatial frequencies span a limited support in the Fourier domain: as a consequence, the resolution of resulting images is quite poor. Synthetic aperture techniques improve this resolution in increasing the size of the aforementioned support (band pass). This can be achieved in several ways: specimen rotation, angular scanning of the illumination or wavelength variation.[3,6] However, for any synthetic aperture technique, the resulting resolution depends directly on assumptions made during the reconstruction process.[7,8] In this case, filtered back-projection is, by tomography convention, performed and allows for a resolution of about one wavelength.[9,10] Nevertheless, back-projection makes abstraction of the diffraction phenomenon, an assumption which is only valid for objects with a small refractive index span.[11] Another approach consists in considering the reconstruction as an inverse problem with a non-linear model.[12–15] In this case, the achieved resolution exceeds the one of a far-field microscope that do not use fluorescence.

In our case, the method we chose takes into account the diffraction phenomenon without performing inverse problem techniques: though efficient, those would imply unwanted complications. Our method is based upon the solving of the Helmholtz equation through the first Born approximation.[16–18] This solution provides the spatial frequencies of the object in the 3D Fourier space, and is given by:

$$A(\mathbf{k_d}) \propto \tilde{n_\delta}(\mathbf{k_d} - \mathbf{k_i}) \tag{1}$$

where A is the complex amplitude of the diffracted scalar field in the Fourier domain, $\tilde{n_\delta}$ is the Fourier transform of the index of refraction at the position $\mathbf{r}$ relative to average index n_0. Also, $\mathbf{k_d}$ and $\mathbf{k_i}$ are respectively the diffracted and illumination wave-vectors. Due to the elasticity condition, it is important to note that $\|\mathbf{k_d}\| = \|\mathbf{k_i}\| = 2\pi/\lambda$. As a consequence, the spatial frequencies are located on a cap of the so-called Ewald sphere, and can be interpreted as the 3D Fourier coefficients of the distribution of the refractive index within the object. With angular scanning, the object 3D frequencies are accumulated into the "butterfly"-shaped support (fig.1).

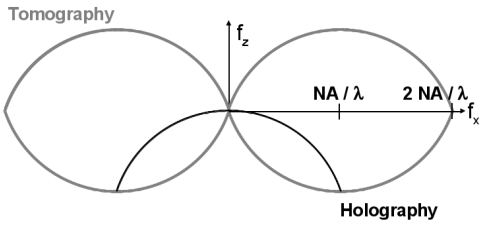

Figure 1. XZ section of the support of the object frequencies. Revolution over Z axis provides a 3D "donut".

2.2 Experimental setup and image reconstruction

Our TDM setup is based on a Mach-Zehnder interferometer, modified in order to perform phase-shifting holography and to record the complex amplitude of both illuminating and diffracted fields (fig.2). A tilting mirror in the illumination arm performs the angular scanning. For a 3D image made from 300 holograms, our first setup used to run acquisitions in 30 min:[17,19–21] this duration was dramatically decreased to 5 min in using more effective hardware components and re-designing the automation scheme. Ultra-fast acquisition has been demonstrated in[10,22–24] since holographic acquisition is not limited by low-level light emission, like fluorescence.

It is important to note that the span of the final frequencies support, which determines the resolution, is determined by the numerical apertures of the condenser and of the objective lens.[15,16,18] In our set-up, we use a standard oil immersion objective and condenser, featuring a numerical aperture of $NA = 1.4$. Using a wavelength of 633 nm, this aperture determines the extreme illumination angle to 67 $^\circ$ and the resulting experimental lateral resolution to 130 nm.

An hologram provides the complex amplitude of the diffracted field for one angle of the illumination beam. Using the Phase-shifting technique, each hologram is calculated from a series of 4 interferograms. Each interferogram is acquired by the CCD camera as a 2D image, one for each value of phase modulation. On the resulting hologram, we perform a 2D Discretized Fourier Transform (DFT) in order to compute the 2D Fourier coefficients of the diffracted field. Finally, these coefficients are projected into a 3D Fourier matrix in a cumulative manner (fig.3).

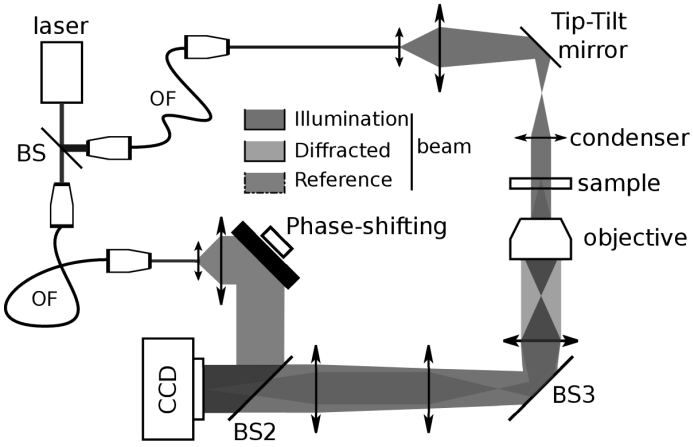

Figure 2. TDM hardware components schematic.

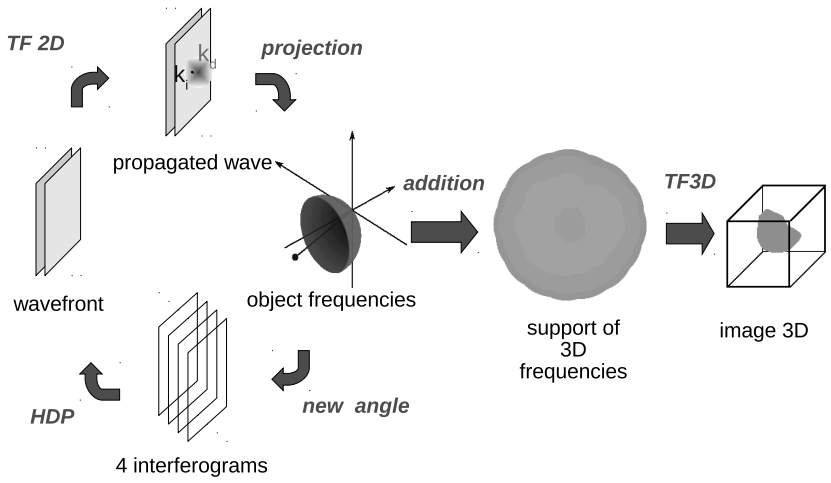

Figure 3. Image reconstruction from acquired interferograms to a complex 3D image of the specimen.

Once enough holograms have been obtained and projected, we compute an inverse FFT on the 3D matrix. The real and imaginary part of the result provide a pair of 3D images of the observed specimen, the real part quantifying the refraction index of the specimen, and the imaginary part the absorption (fig.4). Refraction and absorption can reveal different details on a given locus.

One can note that several projections may cover the same area in the Fourier domain: in order to maintain proper frequency values after their addition, we need to normalize in dividing the value by the number of additive contributions involved.[16, 20]

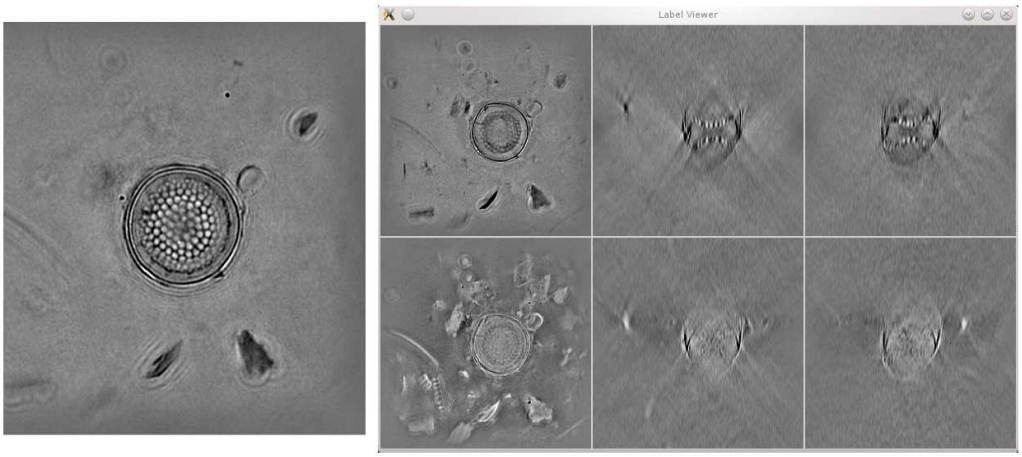

Figure 4. Left: focus on a reconstructed diatomea slice (XY plane). Right: reconstruction of a diatomea cell visualized by a double 3D slicer. Up: refraction volume. Down: absorption volume. Left: XY plane, Middle: XZ plane, Right: YZ plane. Due to the shape of the support, the resolution in Z is visibly inferior to the XY one.

3. PROGRESSIVE IMAGE RECONSTRUCTION

3.1 PC Hardware Setup for Image Acquisition

The PC in charge of the acquisition, reconstruction and visualization is a high-end mainstream model (2000$). It features a Core i7 2600K, a quad-core processor of average frequency 3.4GHz, and 16GB of 1.6GHz DDR3 dual-channel memory.

- In input, the holograms are acquired as grey-level images by a PhotonFocus camera. It is linked to the PC via a dedicated gigabit Ethernet (GigE) line, piloted in jumbo-frames mode by an Intel PCIe 1x controler conforming to the Camera manufacturer requirements. The camera has no embedded memory: snapshots must be sent out before a new one can be taken.
- In output, the PC sends snapshot orders to the Camera via the same GigE line. It also pilots the tilting mirror and the phase modulator via Labjack DAC devices plugged into fast-commutation USB ports.

3.2 Offline Image Acquisition and Reconstruction

In this framework, the different steps are processed sequentially by separate software programs which may not even run on the same PC. Their independent development follows the development time-line of the TDM itself, for which validation was of higher priority than speed optimizations.

- *The acquisition program* commands and synchronizes the camera and Labjack devices: it allows for selecting desired acquisition parameters and handles specific tasks such as auto calibration. When running, it continuously retrieves interferograms as separate image files at a regular pace, until the acquisition ends. Typically, we select a periodical function for the

angular scanning and a sampling parameter: for each function sample, we orient the mirror consequently, and retrieve 4 snapshots for different phase amplitude values.

Due to significant changes in the microscope hardware and in automation control, the acquisition time dropped from 30 min to durations between 1 and 5 min. At this pace, the interferogram images come too fast from the GigE line to be saved synchronously to every drive we could test (from SAS 15K to middle-range SSD) and the use of a ramdisk was necessary to avoid slower rates and stuttering.

- *The reconstruction program* loads the drive-saved interferograms, computes holograms and projects those incrementally into 3D Fourier space. Once all projected, an inverse FFT raises 2 3D images of the specimen (double float precision, 512^3 voxels) which can be saved to drive for later use.

  At the time it was developed, reconstruction took about 10 minutes to compute 300 holograms with the hardware and software available. Continuous evolution of processor computing capacity and of memory size and bandwidth dramatically accelerated the reconstruction process, alongside with progressive increases in the optimization of the FFTW[25] library. Unoptimized reconstruction now takes about 1 minute, and software optimizations decrease this duration to 15 seconds (for 300 holograms) from ramdisk interferograms to ramdisk specimen images.

- *The visualization program* used to be the ImageJ slice viewer. But since we have 2 reconstructed volumes showing different details on a given zone of the specimen, we had to develop a dedicated visualizer for combined viewing of the 2 volumes in the same slices of the XY, YZ and XZ planes. It is based upon the VTK library,[26] which was chosen for its hardware acceleration support and for its prominent status in the n-D data visualization and processing field.

At this stage of software development, we can visualize a 3D specimen a few minutes after the acquisition started, the minimal duration being 1 min 20 secs. Though very fast regarding to the durations we used to cope with, we are still far from the real-time visualization a biologist can expect when he examines a specimen with other imaging methods (often in a 2D fashion). Even from the TDM development point of view, 1 minute can be long when the reconstruction shows approximate selection of region of interest or unoptimal acquisition parameters. The acquisition often needs to be repeated several times until a satisfactory image can be finally produced.

We know that the level of detail of the image increases with a finer sampling of the angular scanning function, which means we can obtain an interpretable image with only a subset of all the interferograms produced by the acquisition (fig.5). In order to make steps towards real-time visualization, we will now present a CPU-based software architecture that will produce increasingly detailed specimen images in processing such subsets as fast as possible, for the least aiming at the same pace of the TDM fastest setup of 5 holograms per second.

3.3 Progressive Reconstruction on a CPU-based architecture

In this framework, a single computer has to proceed concurrently to the acquisition of interferograms and to the image reconstruction and visualization, assuming that during the 1-5 minutes interval an acquisition can last, as many updates as possible of the visualization volume will be provided.

A perfect basis for such an assumption would be to develop a single program instancing concurrent threads for each task: the memory allocated by the program would be shared by the 3 main threads

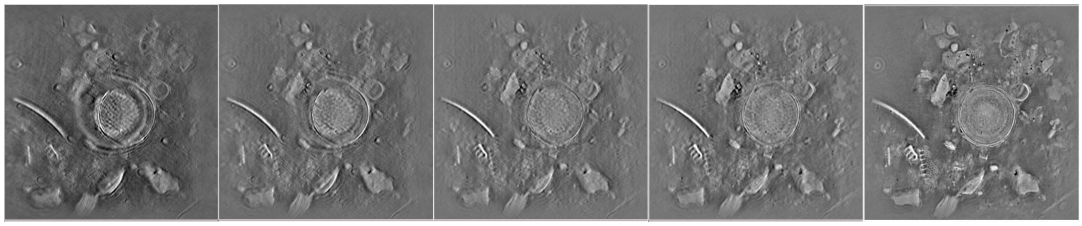

Figure 5. Diatomea slice 210, refraction part, reconstructed from 10, 25, 50, 75 and 500 holograms

and would be immediately accessible by any of these at maximum bandwidth. For several practical reasons, we chose an easier but less optimal solution where the 3 programs are kept as independent executables which share data on files in a ramdisk. Some time and space is then lost in copies from memory to ramdisk and conversely, but this time remains constant, predictable, and somewhat short. As a consequence, this disputable assumption lets us set up a benchmark architecture at a minimum software development time cost.

In order to spare some computation time and memory bandwidth, we downgraded the data precision from double to single float. With a resolution of $2*512^3$, this already represents 1GB of data, which is critical for a benchmark with GPUs, considering their embedded memory typically amounts to 2.3GB for an entry-level Nvidia Tesla. Furthermore, FFTW 3.3.x with maximum optimization level allows for a singularly very short FFT computation time of 0.7s (fig.8) while 3s is the minimum in double mode. We performed both float and double reconstruction and did neither notice nor measure any significant alteration of the biological specimens, so we assume this assumption makes sense, at least in order to provide fast previews.

3.3.1 Software architecture

The acquisition software remains unchanged: it just saves interferograms into the ramdisk. The visualization program did not change much either, it is just modified in order to update volume display every time the volume files on ramdisk are re-written by the reconstruction program. The reconstruction software concentrates most of the modifications: it is re-designed to iteratively process batches of interferograms arriving in the ramdisk. For each batch, it first updates the 3D Fourier volume with the resulting projected holograms, then it performs the inverse 3D FFT, and finally writes to ramdisk the resulting 3D Volumes. When an iteration is finished, it takes as a next batch every new interferogram who came in ramdisk during the former iteration and pursues until the whole acquisition stops.

3.3.2 Data organization for CPU iterative reconstruction

(fig.6) Let's name S_l^{-1} the un-normalized Fourier volume updated by iterative accumulation of projected holograms, and C_S the volume counting contributions for each value. It is allocated in unsigned short to provide relevant precision for counting projections overlaps. S_l^{-1} is made of $(S_r, S_i)^{-1}$, where real and imaginary parts are stored separately in linear sub-volumes in order to keep the same data-ordering as incoming interferograms, i.e linear images sent by the camera. The normalized volume $\overline{S_l^{-1}}$ must be kept separate in order to preserve S_l^{-1} values for next iteration. This data needs to be copied and reordered into $\overline{S_c^{-1}}$, a volume of pairs of complex values which is expected by any FFT computation methods we could examine (e.g[25, 27, 28]). Once the inverse FFT is done, we obtain the specimen S_c in complex form, which needs to be copied in separate linear volumes for visualization (S_r, S_i) due to graphics devices constrains.

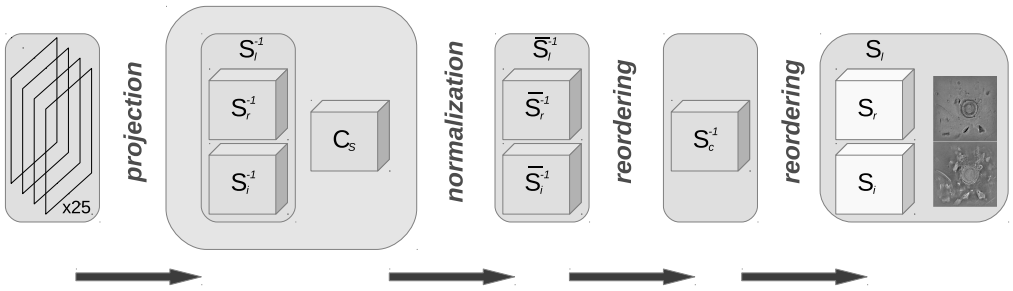

Figure 6. Volumes allocated for iterative reconstruction, CPU.

3.3.3 Timing reconstruction steps

Let's detail the measured duration of the computing steps during one reconstruction update for the processing of a single batch, set at 25*4 interferograms - the result of 5 seconds of acquisition in fast setting mode (fig.7). The batch size of 25 holograms is set so as to correspond to the quantity of interferograms which will arrive in ramdisk at constant rate while the reconstruction performs an iteration.

Figure 7. Duration of each step in one reconstruction update, CPU.

	projection (25)	normalization	reorder	FFT3D
total	0.77	1.42	2.20	0.72
FFT	0.16	0	0	0.72
origin shift	0.14	0	0.22	0
bandwidth	?	0.5	1.76	0
/	?	1	0	0
ramdisk read	0.13	0	0	0
other	0.33	0	0	0

- $\overline{S_l^{-1}}$ computation is surprisingly long with 1.5s of computation time, one-third of it being spent on sequential memory access and the rest in floating-point divisions.
- When $\overline{S_l^{-1}}$ is copied into $\overline{S_c^{-1}}$, we need to copy $S_r[i]$ and $S_i[i]$ into $S_c[i]$, which demands individual memory accesses and consumes 1s bandwidth. And before this, we need to reorder values since our Fourier projections needed a zero in the middle of images/volumes when FFT library conventions demand a zero in the lower-left corner. Fortunately, only 0.22s is used since we can work on the linear volumes for this and take advantage of burst rates with the use of *memcpy* for we don't compute any value and have same input and output memory ordering.
- We can finally perform the inverse FFT so as to obtain a reconstructed specimen in 2 dual images. FFTW 3.3.1 raises S_c in 0.7s with 3 threads used, making full use of 3 CPU cores, and using "wisdom" presets computed offline in exhaustive mode. In order to leave some computing for acquisition, visualization and OS, we should rely on 2 threads only (0.89s).

This is still very fast for a CPU FFT processing, and represents some kind of singularity compared to most computing times in other configurations (fig.8).

- The projection of the 25x4 interferograms into the Fourier volume S_l^{-1} takes 0.75s, including 0.13s in ramdisk image file reading and decode, 0.15s in 2D FFTs and 0.15s in data reordering. The remaining time is used for the projection of the Fourier coefficients into the volume, mixing numerical operations and a lot of data accesses, both random and contiguous.

One could notice that the 2D FFT performed in projection stage could be an opportunity to switch data ordering and avoid the $\overline{S_l^{-1}}$ to $\overline{S_c^{-1}}$ re-ordering, but this would break all the burst rates during projection and afterwards.

One very important point here is that projection time is linearly related to the size of the batch, although all the other steps always take exactly the same time since the amount of data and operations remains constant.

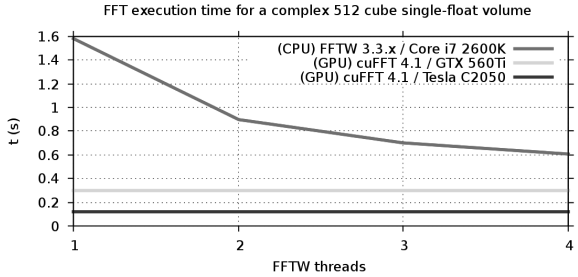

Figure 8. 3D FFT computing time for a 512^3 complex volume of single-floats, performed in FFTW 3.3 CPU library and cuFFT 4.1 GPU library. FFTW computations were performed with pre-optimizations in exhaustive mode.

3.3.4 Discussion

Using a fast setting, the acquisition software produces 20 interferograms per second, aiming at 300 holograms in a minute. The reconstruction program can process a batch in about 5 seconds, with a fixed part of approx. 4.5 seconds for the reconstruction and a variable part depending on the number of interferograms in the batch (3s for 4*100 interferograms). The visualization program hangs for 1-2 seconds for updating the 2 whole volumes due to our non-optimal overall implementation.

This new architecture lets us acquire data until the desired level of detail is obtained on the specimen image: at least, we are no longer forced to guess a priori the level of detail and to restart an acquisition until we realize a posteriori we had selected a proper one. Nevertheless, the update rate remains quite slow at 0.20 fps, which remains very far from the 15 fps one could ideally expect to observe live phenomenons. On its current setup, the reconstruction already produces holograms at 5fps, and several improvements in the current TDM setup will increase this rate in the monthes to come.

These figures are marginally influenced by the “ramdisk pool” assumption we made, since we need to process in full a series of 3D volumes before getting any spatial image as a selection of 2D stacks or as a whole rendered volume. Plus, most of this time is wasted in saving data to ramdisk and reading back from visualizer, a problem we can temporarily avoid in plugging a trivial volume slicer into the reconstruction program which would share the same volume memory.

One should note that some computations could be performed in less time via dispatching to several cores. OpenMP[29] could easily parallelize most array computations with a linear time decrease over cores, but no improvements on memory bandwidth. FFT libraries are already parallelized but, as can be seen on fig.8, the computation time decreases in a logarithmic manner over the number of cores, which was confirmed by tests performed on a hexa-core Xeon CPU. A bi-CPU PC could also make sense, but memory banks are separate for each CPU. The main problem is that our 4 cores are already quite busy, and getting extra cores or CPUs would increase the price tag of several orders of magnitude since such hardware is intended for the server market. In comparison, a high-end Tesla represents a single order of magnitude in price.

We assume that BLAS libraries would provide linear speedup on most computations and we eventually consider using it. Though, since BLAS ports are available both for CPU (GSL[30]) and GPU,[27] we assume it would not change the outcome of this comparison.

We apparently cannot achieve a satisfactory update rate with our PC setup. Most of the time is spent in copying gigabytes of data or in performing simple operations to data elements. Alas, memory bandwidth, CPU core frequency or cores number are the limiting factors here and those are known to evolve progressively over manufacturer's time-lines. Deeper code optimization would probably provide some speedup at the cost of heavy modifications which would make the code very specialized and difficult to maintain. Such a solution does not make sense in the context of microscopy development where many setups are tested over time. If we want a significant improvement in update rates shortly, we need to rely on a different hardware architecture.

4. PROGRESSIVE RECONSTRUCTION ON A GPU ARCHITECTURE

4.1 GPU computing as an alternative to CPU

The development of the video game industry, parallel to the Computer Graphics and Computer Aided Design fields, led into the consumer market some video game cards such as Nvidia's or ATI's, with specialized computing abilities aimed at very fast memory operations sustaining a 60fps display rate. High-bandwidth memory systems were developed on GDDR5 (dual-channel) on rather small amounts of memory, but progressively increased over time (896Mo to 1GB, 1.5, 2, and up to 3GB on latest single GPU boards). With the release of shader operations on single float texture memory, motivated programmers were left enough freedom for developing genuine general-purpose programs on a GPU (GPGPU), assuming they could cope with the dedicated organization scheme of each card model they work with.[31]

Nvidia took an edge in developing and releasing the Compute Unified Device Architecture (CUDA[31]) toolkit for any programmer's use. This system consists in a SDK, a dedicated video driver and a CUDA-enabled device, e.g. every Nvidia gaming board produced for a few years (GeForce) or dedicated computing devices (Tesla). By definition, the CUDA system allows for any CUDA program (C/C++ code) to run on any of aforementioned devices of same memory without code modifications (except for optimizations). De facto, CUDA provides any gaming PC a co-processor for performing appropriate operations faster, or at least in parallel with those of the CPU.

4.2 GPU programming constraints

Though, the programmer needs to handle the rather slow communications between CPU memory and GPU memory via the PCIe 2.0 bus which, in 16x mode, can't exceed 16GB/s of bi-directional bandwidth. Benchmark reported practical one-way data transfer rates between 1-3GB/s only,

with a limit to 5-6GB/s on Teslas for a theoretical limit of 8GB/s. Those figures stand for GB data transfers and rates decrease due to bus latency for smaller transfers. Even though PCIe 3.0 release is likely for the next release of workstations and servers, the PCIe remains a bottleneck any programmer must compose with since the CPU memory is the central hub for PCIe lines onto which drives and controlers are plugged in. In comparison, CPU architecture offers practical bandwidth above 20GB/s in DDR3 dual-channel and close to 40GB/s in quad-channel (recent servers). Inside GPUs, the bandwidth is variable due to a more hierarchized organization. On our mainstream GTX 560Ti, max. bandwidth is already 128GB/s and depending on optimization level, 50GB/s and higher can be reached.

Furthermore, the programmer needs to adapt the program, when applicable, to the SIMD* architecture of the GPU where data-oriented parallelism is key. The graphics memory is processed by shaders, renamed "cores", which can amount up to 448 per GPU, but work with a single speed of about 1.5GHz (4-6 cores at >3GHz on a CPU).

Therefore GPU programs will give maximum throughput on operations performed inside the GPU memory, and which can be decomposed into independent operations led concurrently by the CUDA cores on recursive subdivisions of the global amount of data to process. For instance, GPU architecture outperforms the CPU when it comes to an operation like the "trivial" normalization we had to run in Sec. 3.3.3. But conversely, transfering the data to the GPU and back may consume most of time we spared in SIMD. So, an optimal CPU / GPU architecture is a trade-off between bus transfers and SIMD acceleration.

Nvidia sells mainstream video and computing cards with 2-3GB for 250-600$, and dedicated computing cards of the Tesla series with 6GB for 2000$, from which we need to substract 700Mo for video memory and OpenGL. So, spending on the setup cost, we will have very different assumptions to make on data locality for developing our software.

4.3 Unit tests on a reconstruction loop

As far as possible, we migrated sub-parts of the progressive CPU computation into GPU and performed computing and bandwidth tests as-is (fig.9). We performed computing tests on a GTX 560Ti (200$) gaming card with 384 cores at 1.66GHz and 256 bits dual-channel memory bus set at 2GHz, which can be considered as the low-end for CUDA FFT-capable boards. FFT tests were also performed on deprecated Tesla C2050. We are expecting a 6GB C2075 Tesla in a few monthes, so we expect major changes in the presented figures.

Tests were performed on data already set inside the GPU memory, and results were kept on the GPU memory so as to elude the PCIe bandwidth problem as much as possible. The batch size was downsized to 7 holograms so as to account for the faster reconstruction rate, inferior to 1.50s.

- The 3D FFT part takes advantage of the SIMD architecture: for a 512^3 single float volume, processing time with cuFFT drops to 0.3s on the GTX560 and to 0.12s on the Tesla, in comparison with computing times over 0.7s on CPU (cf fig.8). This single step limits the whole setup at a 3fps rate (8fps on Tesla) while our microscope setup produces holograms at 5fps, so we put extreme attention to the performance of CUDA FFT libraries. At this date, the CUDA 4.1 version of cuFFT seems the best choice for a 2^n-sized matrix, though

*Single Instruction over Multiple Data

Figure 9. Duration of each step in one reconstruction update, GPU.

	projection (7)	normalization	reorder	FFT3D
total	0.23?	0.54	0.45	0.29
FFT	0.02	0	0	0.29
origin shift	0.02	0	0.05	0
bandwidth	?	0.12	0.40	0
/	?	0.42	0	0
ramdisk read	0.04	0	0	0
CPU to GPU	0.05	0	0	0
other	0.10?	0	0	0

third-party libraries[28] used to outperform cuFFT on this configuration and still have an edge on different sizings.

As we can see in fig.8, cuFFT does not offer parameterization like FFTW. We assume a wisdom was computed and provided with the SDK, and that the computing grid is already set so as to make optimal use of all CUDA cores and memory layers.

- The memory bandwidth is a key part here. Even though they vary a lot and are subject to change (i.e computing grid resizing), the experimental results we obtained are way superior to their CPU equivalents. We can also expect further improvement with the Tesla which is designed for maximum memory bandwidth.

- The normalization spares time from bandwidth but mainly from SIMD. A division on 2 volumes of same size 2^n remains a very basic operation for a GPU, for which each core can concurrently perform the very same operation on its own sub-cube. Depending on kernel settings, we obtained overall times under 0.40s on GTX (1 on CPU). We can note that GPUs were designed for fast floating point arithmetic in the context of texture memory processing. Double precision performance is subject to performance drops, but this cost is reduced in Tesla which is more precision-oriented.

- The projection algorithm is difficult to adapt to SIMD architecture, though it remains possible to produce a poor implementation for limited acceleration. Alas, this is still in development at the present time, so, the results presented at this step are based upon expectations from separate measurements.

  In overall, poor man's adaptations on GPU tend to perform at least as fast as in CPU. CUDA SDK is designed so as to compensate most design mistakes in automatic adjustments, which also means a good knowledge of the CUDA SDK and the hardware architecture is required for obtaining decisive speed improvements.

  Though, at this stage, we mainly benefit from improvements on other steps: since the microscope works at a steady rate, we get less interferograms to process at each iteration. If projection was made on CPU for this batch size, projection time would already be only of 0.21s. On GPU, we can benefit from bandwidth on reorderings and other, plus on 2D FFTs. Though, we need to account for the batch transfer on the PCIe bus (cf. sec. 4.4) under unoptimal conditions. Overall, we should not exceed 0.25s for processing a batch and may expect noticeable improvements over time.

So, at this stage, we can expect a computation time after projection in about 1.25s on GTX 560. Switching to Tesla would drop to 1.07 just for the FFTs, other improvements being difficult to assess for the moment. This lets us shape full reconstruction iterations in 1.50s on GTX 560. Though these last figures are estimated, we can shorten the update delay from 5s to 1.5s, which remains an important stride: updates rates rises from 0.20fps to 0.66fps with a modest GeForce.

Moreover, we know we can just afford a better GPU and recompile our code to obtain immediate improvements. We no longer need to buy another CPU or to change the whole setup in order to get noticeable acceleration.

4.4 Data locality and transfers

We measured the time needed for transfering 1GB S_l^{-1} from CPU to GPU, which is a relatively favorable case where data amount tends to overcome the cost of PCIe bus latency. We allocated pinned memory in CPU, i.e. memory allocated by CUDA SDK so as to be aligned in the purpose of a fast PCIe transfer. The time is 0.32s, and remains constant in either way. This amounts for a data transfer rate of 3.1GB/s, which is close to figures reported in similar cases. The Tesla setups are reported faster (up to 5-6GB/s) and seem to have been optimized for scientific data flows when gaming cards assume a game pre-loads NURBS-based geometry and compressed textures before launching a session.

Since the PCIe 2.0 bus is a bottleneck, one should perform as many operations inside the GPU provided we can afford the cost of the relevant GPU class. For a rich man's architecture, let's consider the hypothesis of a C2075 Tesla with 6GB fitted, i.e 5.3GB memory available for computation.

$S_l^{-1}, C_S, \overline{S_l^{-1}}, \overline{S_c^{-1}}, S_c$ fit on less than 4.3GB, which leaves enough room for FFT computation. All these volumes are allocated internally and are updated at each iteration from internal data, except for S_l^{-1} and C_S which depend on imported interferograms. So, in this case, we can expect to implement and run the solution we tested atomically.

The reconstructed volumes remains in GPU memory, which is a favourable case allowing for direct display of information. In our CPU implementation, the 6 images representing the visualization of a single plane have to be transfered from CPU memory to the embedded video memory handled by OpenGL. Though the overhead remains limited, such constraints may become too limiting if we expect a volumic visualization of the specimen from maximum intensity projection methods, which would let us represent the whole biological specimen without manually scanning back and forth through the slices of interest it intersects.

Since we don't have ported to GPU the hologram projection yet, we attempted to consider intermediate setups relevant for our GTX 560 2GB, or also for GTX 580 3GB which remain cheaper than the 6GB Tesla series. (cf. fig.6) We first noticed that with OpenGL graphics enabled, we can't fit S_l^{-1}, C_S altogether, which means a 3GB model would just lets us fit $\overline{S_c^{-1}}, C_S$, perform FFT an keep room for a subset of (S_r, S_i) to visualize. If we can perform normalization directly on $\overline{S_c^{-1}}$ with extra accesses, we can't copy linear volumes any longer and the final performance seems hard to predict, though we could expect an overall speedup anyway.

5. CONCLUSION

Even though we can't assess definitive results without the relevant GPU, we assume from atomic tests that a Tesla would reduce the iteration time of a visualization update from 5 second to

one or less, which represents a significant stride performed on the same PC machine. It opens opportunities for more advanced visualization systems than a classical 2D slicer and would be useful for the computations we expect to perform for improved resolution. Indeed, from fig.4, we can see that the Z-resolution of the TDM is way poorer than XY, a problem which will require specimen rotation techniques and consequently more computations.

ACKNOWLEDGMENTS

This work was supported by OSEO/ANVAR and by MRCT - CNRS.

REFERENCES

[1] Wolf, E., "Three-dimensional structure determination of semi-transparent objects from holographic data," *Opt. Comm.* **1**, 153–156 (September 1969).

[2] Born, M. and Wolf, E., [*Principles of Optics: Electromagnetic Theory of Propagation, Interference and Diffraction of Light*], Cambridge University Press, seventh ed. (1999). ISBN 0-521-64222-1.

[3] Kawata, S., Nakamura, O., and Minami, S., "Optical microscope tomography. I. support constraint," *J. Opt. Soc. Am. A* **4**, 292–297 (Jan 1987).

[4] Chen, R. H.-Y. and Wilkinson, T. D., "Computer generated hologram with geometric occlusion using gpu-accelerated depth buffer rasterization for three-dimensional display," *Appl. Opt.* **48**, 4246–4255 (Jul 2009).

[5] Shimobaba, T., Ito, T., Masuda, N., Ichihashi, Y., and Takada, N., "Fast calculation of computer-generated-hologram on amd hd5000 series gpu and opencl," *Opt. Express* **18**, 9955–9960 (May 2010).

[6] Vertu, S., Delaunay, J.-J., Yamada, I., and Haeberlé, O., "Diffraction microtomography with sample rotation: influence of a missing apple core in the recorded frequency space," *Centr. Eur. J. Phys.* **7**(22), 22–31 (2009).

[7] Gorski, W. and Osten, W., "Tomographic imaging of photonic crystal fibers," *Opt. Lett.* **32**, 1977–1979 (Jul 2007).

[8] Kak, A. C. and Slaney, M., [*Principles of Computerized Tomographic Imaging*] (1988).

[9] Charrière, F., Marian, A., Montfort, F., Kuehn, J., Colomb, T., Cuche, E., Marquet, P., and Depeursinge, C., "Cell refractive index tomography by digital holographic microscopy," *Opt. Lett.* **31**, 178–180 (Jan 2006).

[10] Choi, W., Fang-Yen, C., Badizadegan, K., Oh, S., Lue, N., Dasari, R. R., and Feld, M. S., "Tomographic phase microscopy," *Nat. Meth.* **4**(9), 717–719 (2007).

[11] Carter, W. H. and Ho, P.-C., "Reconstruction of inhomogeneous scattering objects from holograms," *Appl. Opt.* **13**, 162–172 (Jan 1974).

[12] Belkebir, K., Chaumet, P. C., and Sentenac, A., "Superresolution in total internal reflection tomography," *J. Opt. Soc. Am. A* **22**, 1889–1897 (Sep 2005).

[13] Maire, G., Drsek, F., Girard, J., Giovannini, H., Talneau, A., Konan, D., Belkebir, K., Chaumet, P. C., and Sentenac, A., "Experimental demonstration of quantitative imaging beyond abbe's limit with optical diffraction tomography," *Phys. Rev. Lett.* **102**, 213905 (May 2009).

[14] VanDenBerg, P. M. and Kleinman, R., "Large-scale optimization with applications. part. I: Optimization in inverse problems and design," **92**, 173, Springer Verlag, New York (1997).

[15] Chaumet, P. C., Belkebir, K., and Sentenac, A., "Three-dimensional subwavelength optical imaging using the coupled dipole method," *Phys. Rev. B* **69**, 245405 (Jun 2004).

[16] Lauer, V., "New approach to optical diffraction tomography yielding a vector equation of diffraction tomography and a novel tomographic microscope," *J. Microsc.* **205**, 165–176 (2002).

[17] Debailleul, M., Georges, V., Simon, B., Morin, R., and Haeberlé, O., "High-resolution three-dimensional tomographic diffractive microscopy of transparent inorganic and biological samples," *Opt. Lett.* **34**, 79–81 (Jan 2009).

[18] Haeberlé, O., Sentenac, A., and Giovannini, H., [*Modern research and educational topics in microscopy*], vol. II, ch. An introduction to diffractive tomographic microscopy, pp. 956–967 (2007). ISBN 978-84-611-9418-6.

[19] Simon, B., Debailleul, M., Georges, V., Lauer, V., and Haeberlé, O., "Tomographic diffractive microscopy of transparent samples," *Eur. Phys. J.-Appl Phys.* **44**(01), 29–35 (2008).

[20] Debailleul, M., Simon, B., Georges, V., Haeberlé, O., and Lauer, V., "Holographic microscopy and diffractive microtomography of transparent samples," *Meas. Sci. Technol.* **19**(7), 074009 (2008).

[21] Simon, B., Debailleul, M., Beghin, A., Tourneur, Y., and Haeberlé, O., "High-resolution tomographic diffractive microscopy of biological samples," *J. Biophotonics* **3**(7), 462–467 (2010).

[22] Park, Y., Diez-Silva, M., Popescu, G., Lykotrafitis, G., Choi, W., Feld, M., and Suresh, S., "Refractive index maps and membrane dynamics of human red blood cells parasitized by plasmodium falciparum," *PNAS* **105** (September 2008).

[23] Lue, N., Choi, W., Popescu, G., Badizadegan, K., Dasari, R. R., and Feld, M. S., "Synthetic aperture tomographic phase microscopy for 3D imaging of live cells in translational motion," *Opt. Exp.* **16**, 16240–16246 (Sep 2008).

[24] Sung, Y., Choi, W., Fang-Yen, C., Badizadegan, K., Dasari, R. R., and Feld, M. S., "Optical diffraction tomography for high resolution live cell imaging," *Opt. Exp.* **17**, 266–277 (Jan 2009).

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

[26] Schroeder, W., Martin, K., and Lorensen, B., [*The Visualization Toolkit, An Object-Oriented Approach To 3D Graphics*], Prentice Hall, fourth ed. (2006). ISBN 978-1930934191.

[27] Farber, R., [*CUDA Application Design and Development*], Morgan Kaufmann Publishers (2011). ISBN 978-0-12-388426-8.

[28] Nukada, A. and Matsuoka, S., "Auto-tuning 3-D FFT library for CUDA GPUs," in [*Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*], *SC '09*, 30:1–30:10, ACM, New York, NY, USA (2009).

[29] Gao, R., Sato, M., and Ayguade, E., "Special Issue on OpenMP," *Int. J. Parallel Prog.* **36** (June 2008).

[30] Galassi, M., Davies, J., and Theiler, J., [*GNU Scientific Library Reference Manual*], third ed. (January 2009). ISBN 0954612078.

[31] Kirk, D. and Hwu, W., [*Programming Massively Parallel Processors*], Morgan Kaufmann Publishers (2010). ISBN 978-0-12-381472-2.