Machine learning for ultrafast X-ray diffraction patterns on large-scale GPU clusters

Tomas Ekeberg¹, Stefan Engblom² and Jing Liu¹,²

Abstract
The classical method of determining the atomic structure of complex molecules by analyzing diffraction patterns is currently undergoing drastic developments. Modern techniques for producing extremely bright and coherent X-ray lasers allow a beam of streaming particles to be intercepted and hit by an ultrashort high-energy X-ray beam. Through machine learning methods the data thus collected can be transformed into a three-dimensional volumetric intensity map of the particle itself. The computational complexity associated with this problem is very high such that clusters of data parallel accelerators are required. We have implemented a distributed and highly efficient algorithm for the inversion of large collections of diffraction patterns targeting clusters of hundreds of GPUs. With the expected enormous amount of diffraction data to be produced in the foreseeable future, this is the required scale to approach real-time processing of data at the beam site. Using both real and synthetic data we look at the scaling properties of the application and discuss the overall computational viability of this exciting and novel imaging technique.

Keywords
Expectation–maximization, X-ray laser diffraction, GPU cluster, single-molecule imaging

I Introduction
X-ray crystallography is currently the most successful method for protein structure determination. A limitation is that this method requires high-quality crystals of the sample protein. This is particularly problematic for membrane proteins which are notoriously hard to crystallize. This class of proteins contains about 20–30% of all proteins and are targeted by 50% of modern drugs; still they comprise less than 0.1% of the known protein structures.

The recent construction of free-electron lasers (FELs) has the potential to revolutionize structural biology by allowing structure determination without the need for crystallization. FEL pulses are intense enough that an interpretable diffraction signal can be recorded from single proteins or viruses. Also, the pulses are short enough to outrun the radiation damage to the particle and the scattered data thus represents the intact particle even though the extreme intensity will destroy the sample within picoseconds.

Since the diffraction data frames are collected one at a time and the extremely intense X-ray pulse destroys the samples, it is impossible to collect multiple exposures of the same particle. However, just like in crystallography, we can use the fact that many biological particles exist in identical copies. Data collected from many identical particles can thus be treated as if they come from the same particle.

In this scheme, particles are injected into the stream of X-ray pulses and intercepted in random orientation (Neutze et al., 2000). A diffraction pattern represents a curved two-dimensional slice through the modulus of the Fourier transform of the electron density of the particle. Since the particles are assumed identical, the patterns will correspond to different slices through the same Fourier density. If the unknown orientations can be recovered, the diffraction patterns can thus be assembled to the complete three-dimensional Fourier intensity of the particle.

¹Laboratory of Molecular Biophysics, Department of Cell and Molecular Biology, Uppsala University, Uppsala, Sweden
²Division of Scientific Computing, Department of Information Technology, Uppsala University, Uppsala, Sweden

Corresponding author:
Stefan Engblom, Division of Scientific Computing, Department of Information Technology, Uppsala University, Box 337, SE-751 05 Uppsala, Sweden.
Email: stefane@it.uu.se
As opposed to the situation in crystallography, the orientation of each particle is not directly measurable. Instead, the orientations are recovered by maximizing the fit between the individual diffraction patterns. Several algorithms for solving this problem have been proposed (Loh and Elser, 2009; Fung et al., 2008). The most successful of these is the expansion maximization compression (EMC) method (Loh and Elser, 2009) which has been verified experimentally using artificial samples (Loh et al., 2010), and which was recently used for the reconstruction of the giant Mimivirus (Ekeberg, 2012).

The LINAC Coherent Light Source (LCLS) (Emma et al., 2010) has a repetition rate of 120 Hz and a sustained hit ratio of 20% has been achieved reproducibly. This corresponds to 1 million diffraction patterns in a single 12-hour shift or about 4 TB of data. The European XFEL is becoming operational in 2016 and will have a repetition rate of 27,000 Hz (Schneidmiller and Yurkov, 2011).

The 3D-alignment algorithms are computationally very demanding, yet high data volumes are fundamental for achieving high resolution and to balance the low photon signal when studying smaller objects. In the light of the above developments there is an imminent need for a massively parallel implementation of the EMC algorithm to keep up with the increased data rates and the increasing problem size.

Based on previous experience with an implementation for smaller heterogeneous GPU-computers (Engblom and Liu, 2014), in this paper we present a working fully distributed implementation targeting large-scale clusters of hundreds of GPU computers. In an effort to prepare for the increasing data rates we ensure in our implementation that data can be effectively and flexibly distributed. We also devise a kind of adaptive iteration which allows computational resources to be used in proportion to the resolution of the final reconstruction. Similar techniques, we argue, will be required when handling streaming data at the beam site.

An overview of the XFEL imaging setup and the associated computational methodology is found in Section 2. Our data parallel and fully distributed implementation is discussed in some detail in Section 3. Performance results on clusters of up to 100 GPUs, reaching up to and beyond 4 TFLOPS, are presented in Section 4, and a concluding discussion is found in Section 5.

2 X-ray laser diffraction and maximum likelihood imaging

In this section we summarize the experimental setup and the principles behind 3D imaging with XFELs. The associated data analysis is formulated as a hidden variable maximum likelihood problem which can be handled by the expectation–maximization (EM) algorithm. We also describe the current ‘best practice’ in designing a working such algorithm. Although we certainly expect the methodology to develop further, it seems reasonable to believe that our implementation, or at least a very similar one, will be used extensively when modern XFEL facilities are increasingly being put to use.

2.1 Ultrafast X-ray diffraction patterns

The schematics of collecting data by XFELs is depicted graphically in Figure 1. An inflow of samples of biomolecules is intercepted by an X-ray laser pulse resulting in a collection of diffraction images. We denote the raw data output from this procedure by $K = (K_k)_{k=1}^M$; this is a collection of frames, each containing measured photon counts. The detector is discrete and hence for the $k$th frame, $K_k = (K_{ik})_{i=1}^{M_{pix}}$, where $M_{pix} = 2^{20} = 1024 \times 1024$ is a typical resolution. Some pixel counts near the center are missing or may reach saturation as a result of inherent physical limitations with the experimental procedure.

Assuming ideally that the stream of samples consists of identical copies of a single physical object with an electron density $O$, diffraction theory (Gaffney and Chapman, 2007) gives that each frame $K_k$ is given by a certain slice, an Ewald sphere, of the 3D Fourier transform $W$ of the real space object $O$.

With a sufficiently large collection of frames an estimate $\hat{W}$ of $W$ is first determined. Notably, this estimate lacks information about the Fourier phases since it is
based on photon count data only. The final step is therefore a phase retrieval procedure (Chapman et al., 2006; Shapiro et al., 2005), after which an estimate of the real space object can be obtained.

2.2 Maximum likelihood estimation via EM

With independent and identically distributed (i.i.d.) frames \( K = (K_k)_{k=1}^{M_{\text{data}}} \), the maximum likelihood estimator is given by

\[
\hat{W} = \arg \max \mathbb{P}_{\text{data}} \sum_{k=1}^{M_{\text{data}}} \log \mathbb{P}(K_k|W)
\]

that is, for some probabilistic intensity model, maximizing the likelihood of the obtained photon count data when presented with the recorded data. The problem is incomplete for two reasons. First, the true rotation \( R_k \) of the object in measurement \( K_k \) is unknown and consequently the frame cannot be directly associated with a definite Ewald sphere. Second, the energy of the X-ray pulse, the photon fluence \( \phi_k \), which hits the sample is also an unknown variable.

In addition to the problem of hidden data, the overall signal-to-noise ratio is very small and implies a grand computational challenge since to counteract this the processed data volume has to be large. With input data corrupted by high noise levels, measures have to be taken in order to design a robust and useful algorithm (Serkez et al., 2014).

The EM algorithm (Dempster et al., 1977) aims at producing likelihood estimates with hidden data in a constructive way. The basic procedure of alternating steps of (i) assigning probabilities to the hidden states, and (ii) maximum likelihood estimates of the parameters of the model can be shown under broad conditions to be at least a descent step of the full likelihood (Neal and Hinton, 1998). Notably, in step (i) the model parameters are kept fixed, while in (ii) the estimated probabilities of the hidden states (the responsibilities using EM terminology) from step (i) are assumed known.

We now introduce some notation. First, the rotational space is discretized by \( (R_j)_{j=1}^{M_{\text{rot}}} \). Since this is generally a non-uniform discretization we denote by \( w_j \) the prior weight for the \( j \)th rotation, normalized such that \( \sum_j w_j = 1 \). In other words, selecting \( R_j \) with probability \( w_j \) implies a practically uniform sampling of the rotational space. The intensity space is similarly discretized by \( \{q_s\}_{s=1}^{M_{\text{int}}} \) such that using this coordinate system the unknown Fourier intensity at position \( R_jq_s \) can be denoted by \( W_{jk} \). Finally, we denote by \( \phi_{jk} \) the intensity of the beam that produced data frame \( k \), given that the object was rotated according to \( R_j \).

An early EM algorithm for solving the problem was devised in (Loh and Elser, 2009) using the assumption that the signal is Poissonian. This assumption has the benefit of producing an essentially parameter-free algorithm but neglects other potential sources of noise. In practice a Gaussian model has been more successful. More precisely we assume that the measured intensity of the \( i \)th pixel in the \( k \)th measurement, when scaled with the photon fluence, is Gaussian around the unknown Fourier intensity \( W_{ij} \) (Loh et al., 2010; Ekeberg, 2012)

\[
\log \mathbb{P}(K_{ik} = \kappa | W_{ij}, R_j, \phi_{jk}) \sim -\frac{(\kappa / \phi_{jk} - W_{ij})^2}{2\sigma^2} =: Q_{ijk}(W, \phi)
\]

with \( \sigma \) a noise parameter which is kept at conservative values or decreases slightly as the iteration proceeds. Summing over \( i \) we get the joint log-likelihood function,

\[
Q_{jk}(W, \phi) := \sum_{i=1}^{M_{\text{pix}}} Q_{ijk}(W, \phi)
\]

that is, the logarithm of the probability of observing frame \( K_k \), given rotation \( R_j \) and fluence \( \phi_{jk} \). Integrating this over the space of rotations we get

\[
P^{(n+1)}_{jk} = P^{(n+1)}(W^{(n)}, \phi^{(n)}) := \mathbb{P}(R_j|K_k, W^{(n)}, \phi^{(n)})
\]

\[
= \frac{W_j T_{jk}(W^{(n)}, \phi^{(n)})}{\sum_{j'=1}^{M_{\text{rot}}} W_{j'} T_{j'k}(W^{(n)}, \phi^{(n)})}
\]

(“E-step”)

in terms of \( T_{jk}(W, \phi) = \exp(Q_{jk}(W, \phi)) \). Some care is required when evaluating (4) to avoid finite precision effects.

Although there is no explicit maximum likelihood formula for computing \( (W, \phi) \) given \( P \), the following fix-point iteration for the normal equations has been proposed (Loh et al., 2010),

\[
W^{(n+1)}_{ij} = \frac{\sum_{k=1}^{M_{\text{data}}} P^{(n+1)}_{jk} K_{ik} / \phi_{jk}}{\sum_{k=1}^{M_{\text{data}}} P^{(n+1)}_{jk}}
\]

(“M-step”)

When the EM iteration is understood as a descent step of the full likelihood function, this approach of using ‘partial steps’ can be justified (Neal and Hinton, 1998).

The rotations \( (R_j)_{j=1}^{M_{\text{rot}}} \) and the corresponding prior weights \( w_j \) must be found through some kind of discretization procedure. The suggestion of Loh and Elser (2009) is to use the fact that quaternions encode rotations; any rotation can be identified as a point on a 4D sphere which can hence be discretized. A suitable geometric object for this purpose is the 600-cell (or hexacosichoron) which is a 4D convex regular 4-polytope whose boundary is composed of 600 tetrahedra. At
even larger values of \( M_{\text{rot}} \) one further uses the fcc-cell in which each tetrahedron is uniformly divided \( d \) times into \{1, 4, 10, 20, 35, \ldots\} smaller tetrahedra. This implies the relation (Loh and Elser, 2009, Appendix C)

\[
M_{\text{rot}}(d) = 1 \cdot (5d^3 + d) \\
= 6300, 10860, 25680, 50100, 86520
\]  
for \( d = [5, 6, 8, 10, 12] \)  
(7)

In Section 3.4 below we make an active use of this discretization by increasing \( d \) adaptively whenever the increase of likelihood goes below some predefined threshold.

### 2.3 EM with compression steps: the EMC

A problem with the EM iteration defined by (4) and (5)–(6) is that averages are computed in discrete space while data is continuous. There are many pairs \( (i,j) \) such that \( R_{q_i} \) are very close, but in the M-step (5) they will be exchanging information with disjoint or nearly disjoint sets of frames. If the end result is to be understood as a continuous object some kind of smoothing procedure has to be devised.

A straightforward way to achieve this is to add expansion/compression steps. The purpose of the latter step is to compress (average/smooth) the representation into, say, a Cartesian representation with a uniform spatial resolution. The expansion step is the inverse of this operation and takes us back to the working description in \( R_{q_i} \) space. The combination of the average (5) in the M-step and a compression step then ensures that nearby pixels and rotations have exchanged information with overlapping sets of frames.

Let interpolation weights \( f \) and interpolation abscissas \( (p_l)_{l=1}^{M_{\text{rot}}} \) be defined such that for \( g \) some smooth function,

\[
g(q) \approx \sum_{l=1}^{M_{\text{rot}}} f(p_l - q) g(p_l).  
\]  
(8)

An expansion operator can now be defined,

\[
W_{ij} = \sum_{l=1}^{M_{\text{rot}}} f(p_l - R_{q_l}) \mathbb{V}_{ij} \quad (\text{‘e-step’}) \]  
(9)

which maps values from a grid \( \mathbb{V} := W(p_l) \) into the working description \( W_{ij} = W(R_{q_l}) \).

Similarly, a suggestion for the compression operator is given by (Loh and Elser, 2009),

\[
\mathbb{V}_{ij} = \frac{\sum_{i=1}^{M_{\text{fin}}} \sum_{j=1}^{M_{\text{rot}}} f(p_l - R_{q_l}) W_{ij}}{\sum_{i=1}^{M_{\text{fin}}} \sum_{j=1}^{M_{\text{rot}}} f(p_l - R_{q_l})} \quad (\text{‘c-step’})  
\]  
(10)

It should be noted that, whereas (9) is a consistent interpolation, (10) rather falls under the framework of inverse distance weighting. Furthermore, in the implementation discussed here, the M- and the c-steps are intertwined in that the normalization is deferred until after \( \mathbb{V} \) has been obtained. Hence, we compute (compare (5))

\[
W_{ij}^{(n+1)} = \sum_{k=1}^{M_{\text{fin}}} f_{jk}^{(n+1)} K_{jk} / \phi_{jk}^{(n)}  
\]  
(11)

The c-step is then computed as

\[
\mathbb{V}_{ij}^{(n+1)} = \frac{\sum_{i=1}^{M_{\text{fin}}} \sum_{j=1}^{M_{\text{rot}}} f(p_l - R_{q_l}) W_{ij}^{(n+1)}}{\sum_{i=1}^{M_{\text{fin}}} \sum_{j=1}^{M_{\text{rot}}} f(p_l - R_{q_l})} \sum_{k=1}^{M_{\text{fin}}} f_{jk}^{(n+1)}  
\]  
(12)

In Algorithm 1 a summary of the algorithm which will be considered here is given. As we shall next see, the algorithm has a distinct data parallel character and can be distributed efficiently.

### 3 Parallelization in GPU clusters

By inspection the algorithm under consideration is computationally intensive as it is composed mainly of blocks of nonlinear matrix operations. This implies that it can be expected to perform well on modern data parallel accelerators in general and on GPUs in particular. The most common approach to distributed GPU computing is to transfer data via an MPI layer and use CUDA at the computational nodes. This master-slave

---

**Algorithm 1** The EMC algorithm.

**Input:** Initial guess of the 3D intensity distribution \( \mathbb{W}^{(0)} \) of the object on the grid \( (p_l)_{l=1}^{M_{\text{rot}}} \), and an initial estimate of the rotational probabilities \( P^{(0)} \).

**Output:** Improved image \( \mathbb{W} \) and probabilities \( P \).

1: repeat
2: \( n = 0, 1, \ldots \).
3: \( \mathbb{W}^{(n)} \) := \( \mathbb{W}^{(n)} \).
4: \( P^{(n+1)} := E \cdot P^{(n)} \). \{Expansion step, \( (9) \)\}
5: \( \mathbb{W}^{(n+1)} := cM \cdot \mathbb{W}^{(n)} \). \{Expectation step, \( (4) \)\}
6: until change in either \( \mathbb{W} \) or likelihood is small enough

\{Combined Maximization and compression, \( (11) \) and \( (12) \)\}
approach has been employed successfully in other multi-GPU applications (Jacobsen et al., 2010; Pennycook et al., 2011), and is also our approach.

In this section we take a bottom-up approach and first discuss the single-node data parallelism and then extend the same scheme to a distributed environment. We next devise a fully distributed approach in which very large diffraction datasets may be considered, and we finally also develop a simple but efficient multiresolution type adaptivity.

3.1 Single-node data parallelism

In an implementation targeting a single GPU node, data must be shared between the cores of the GPU. Currently, the typical resolution is to reconstruct a $64 \times 64 \times 64$ or a $128 \times 128 \times 128$ intensity model using about 1000 diffraction patterns (see Table 1). Given the complexity of the steps of the algorithm, the E-step (4) stands out as the most expensive part. Recall that the E-step estimates likelihoods $P_{jk}$ for rotations $R_j$, given data frames $K_k$, and it therefore makes sense to distribute the resulting probability matrix $P$. With a usually quite large rotational space, the algorithm requires around 1.5 GB of memory to store the $M_{\text{data}} \times M_{\text{rot}}$ matrix $P$ even for a low-resolution reconstruction. Such a matrix can be distributed by dividing the rotational space or by distributing the images themselves. In the single-node implementation we choose not to distribute by images since $M_{\text{data}}$ is usually much smaller than $M_{\text{rot}}$.

Our single-node EMC was implemented using CUDA with C/C++ wrappers and the implementation closely follows the logic in Algorithm 1. Briefly, the CPU controls the overall procedure and streams all required data to the GPU as well as writes the output. Hence, the diffraction patterns are initially loaded and copied into GPU memory. In each EMC iteration, the compute intensive steps (9), (4), (11) and (12) are all evaluated on the GPU.

To discuss those steps, let a ‘chunk’ $C_c$ denote a contiguous set of rotations, writing $R_c = \{ R_{j} ; j \in C_c \}$, $C_c := \{ j ; f_{-} \leq j < f_{+} \}$. Let the number of rotations in the set $C_c$ be denoted by $|C_c|$, and the total number of chunks as indexed by $c$ by $M_{\text{chunks}}$. The computations in the e-step (9) and the M-step (11) are partitioned into chunks naturally. Neither of those steps involve a normalization over the rotational space, and therefore we can easily calculate and update partial slices $W_{jk}^c$ in a GPU kernel. Each kernel uses $|C_c|$ blocks, and each block takes care of the computations for one rotation.

The E-step (4) and the c-step (12) are more complex due to the normalization over the rotational space. For the latter kernel $M_{\text{data}} \times |C_c|$ blocks are used and each thread in the block calculates one value of the nominator in (4). The normalization is then performed in a separate final sweep. In the c-step the updated model is determined by averaging among chunks, rotations, and diffraction patterns. Here the numerator and denominator in (12) are calculated separately with the division as a final step. The associated GPU kernel uses $|C_c|$ blocks, and each block handles one slice in $R_c$.

3.2 Distributed implementation on a GPU cluster

As argued previously, partitioning the rotational space implies that the algorithm fits efficiently into memory. This scheme is also simple enough to be extended to GPU clusters under our preferred master–slave approach. In addition, our distributed EMC algorithm has the feature of not only dividing the computations of the most expensive step (the E-step, see Figure 5), but also that it localizes the computations of the corresponding photon fluence $\phi$, which is the second most expensive step.

In the implementation, we associate each GPU with one CPU at the same cluster node, hence uses one MPI process per GPU, and we designate one such CPU/GPU pair as the master node. For every EMC iteration, each pair takes care of local computations and synchronizes when necessary. The master CPU has the overall control of how EMC synchronizes over the nodes, and each CPU is in charge of the local GPU computations. Communications between the nodes are thus only needed in three places. First, diffraction patterns and algorithmic configurations must be broadcast before the algorithm starts. Second, for each chunk $C_c$, local estimated probabilities $P^c$ must be normalized globally over all nodes. Third, and finally, the local model $W^c$ must be merged (averaged) at the end of each iteration. The data flow among the nodes in our implementation is shown in Figure 2. This procedure is a special case of the fully distributed EMC, which we now proceed to discuss.

3.3 Fully distributed EMC

For large enough datasets, the diffraction patterns themselves also need to be distributed as they no longer fit on a single node. The chunks $C_c$ are now sets of two-dimensional indices, $C_c = \{ (j, k) ; k_- \leq k < k_+ , j_- \leq j < j_+ \}$. With this Cartesian grid-like topology, data is either communicated over the rotational space (along the $j$-direction), or over image space (along the $k$-direction).

The steps that do not involve the diffraction patterns in a global sense, namely the E-step (4) and the fluence calculation of the M-step (6), are not affected by this novel way of partitioning data, and can therefore be implemented as previously described. The remaining steps (11)–(12) require data to be broadcast in the $k$-
3.4 Adaptive EMC

Previously, we aimed for an efficient execution by distributing data and using an increasing number of compute nodes. In most actual experience with the method a fairly highly resolved rotational space is used and one typically observes a likelihood which slowly but steadily improves. The performance critical steps of the method, the E-step (4) and part of the combined M/c-step (12), scale directly with the discretization of the rotational space \(M_{\text{rot}}\). Hence, it seems reasonable not to waste computational time using a large value of \(M_{\text{rot}}\) before the likelihood has improved sufficiently. A simple way to achieve this is to start using some small value of \(M_{\text{rot}}\), \(M_{\text{rot}}(d_0)\) say, and increase \(d\) in (7) whenever the current value seems too small to improve the likelihood any further. In practice we increase \(d\) by 1 when

\[
|L^n - L^{n-1}| \leq 0.01|L^n|
\]

with \(L^n\) the likelihood at iteration \(n\).

4 Experiments

We now proceed to investigate the achievable performance of our implementation. Our datasets are either real ones and of similar quality to those currently being processed, or are synthetic and of considerably larger size to be able to assess future performance profiles. In particular, we will explore the possibility to obtain a highly resolved image from a very large diffraction dataset on a cluster consisting of 100 GPUs.

4.1 Setup and basic profiling

We ran our experiments using a 32-node homogeneous GPU cluster. Each node of the cluster is equipped with 4 six-core Intel Xeon E5-2620 CPUs and 4 Nvidia GeForce GTX 680 GPUs. All of the CPU cores operate at 2.0 GHz with 32K L1i and L1d caches. They are organized as 2 NUMA nodes with a total of 64 GB memory. Each GTX 680 GPU has 4 GB memory, and
Algorithm 2 Fully distributed data-parallel version of the EMC algorithm.

**Input:** Diffraction patterns $K$, initial guess of the 3D intensity distribution $W^{(0)}$.
**Distribute:** Divide all computational nodes into a Cartesian grid. Distribute diffraction patterns $K$ along the $k$-direction, and probabilities $P$ along the $j$-direction.

1: repeat
2: At each node copy local data from CPU to GPU;
3: Execute expansion step (9): at each GPU and for each partition $c$, expand $W^{(0)}$ into $W^c$, using the GPU kernel __global__ void expansion $<<W^c|256>>$;
4: Execute expectation step (4): compute the probabilities $P$ as in (4) (steps 1.1 and 1.2 in Figure 3). The GPU kernel is __global__ void calculate_probability $<<M_{data}|C_6|256>>$. Normalization is performed among nodes that share the same distribution of $K$, hence a MPI_AllReduce operation in the $j$-direction.
5: Execute the maximization step (11): update $W$ according to (12), the numerators and denominators are calculated via the GPU kernel __global__ void insert_slices $<<C_6|256>>$;
   For all nodes, use MPI_AllReduce to transfer and add the numerators and denominators separately, and then perform the final division by the GPU kernel __global__ void insert_slices_division $<<M_{data}|256>>$; Finally determine $W^{n-1}$ in the c-step as in (12) (step 2 in Figure 3).
6: Execute the compression step (12): update $W$ according to (12), the numerators and denominators are calculated via the GPU kernel __global__ void insert_slices $<<C_6|256>>$;
7: Use MPI_Bcast to update $W^{n-1}$ for every node (step 3 in Figure 3).
8: until change in either $W$ or likelihood is small enough

![Figure 3](image.png)

Figure 3. Communication pattern of the fully distributed EMC implementation. A node group is a set of GPU/CPU pairs which share the same partition of diffraction data $K$. Communications among node groups are illustrated by thin arrows, and communications within a node group are shown in wide arrows. The master node works as a normal node in node group 1 with the exception for steps 0 and 3. According to the notation in the figure, $W_c$ is the intensity model $W$. After the configuration is broadcast, every node fetches their own portion of $K$, and the same initial model $W$.

the reported nominal peak performance in single precision of matrix multiplication, matrix left division, and the Fast Fourier Transform are $[966,573,101]$ GFLOPS, respectively (GPU Bench). Cluster nodes are interconnected using a QDR Infiniband with a bandwidth of 32 Gbit/s. We measured the bandwidth of CPU/GPU connections by doing a host-to-device and device-to-host memory copy, and we found that the bandwidth in both direction is around 3 GB/s. For the compilers and libraries, we used GCC 4.4.6, CUDA 5.0, and Open MPI 1.5.4. By considering the inter-node MPI bandwidth, the intra-node CPU/GPU bandwidth and the data volume which needs to be transferred, we judge that our implementation is not bandwidth bound.

We used mainly two different diffraction datasets. A smaller case consisting of 198 images from an X-ray diffraction experiment with the giant Mimivirus (Ekeberg, 2012; Seibert et al., 2011) performed at the Linac Coherent Light Source (LCLS). A larger case was obtained through synthetic simulation for an icosahedral shape and consists of 1000 frames. Figure 4 displays samples of these diffraction patterns.

In Table 1 we list the sizes of all relevant data in our experiments. The experiments were configured to reconstruct a small ($64 \times 64 \times 64$), or, respectively, a large ($128 \times 128 \times 128$) 3D intensity model in single precision. In Section 4.2 we also ran a 'giant' case consisting of our synthetic dataset, but duplicated ten times (i.e. a total of 10,000 frames).

The result of profiling our single-GPU implementation provided a motivation for our approach to distribute data in our multi-GPU implementation. We profiled the single-GPU implementation by reconstructing at low resolution the Mimivirus dataset (Set #1 in Table 1) on one Nvidia GeForce GTX 680.
Figure 5 displays the computational statistics after averaging over the first 10 iterations. As expected, the E-step was the most expensive step, consuming more than 55% of the total time. This means that it is preferable to distribute data along the rotational space (along the \( j \)-direction), at least given this relatively small number of diffraction patterns. Since the probability \( P_{jk} \) has the same size as the photon fluence \( f_{jk} \), this also means that the second most expensive step, i.e. the computation of \( f \), parallelizes very well.

### 4.2 Performance analysis

In this section we report results from our distributed EMC from Section 3.2 for both the Mimivirus dataset and the synthetic dataset on up to 32 GPUs. We also look at the performance of the fully distributed EMC from Section 3.3 using the ‘giant’ synthetic dataset, consisting of 10,000 diffraction patterns. In contrast to these large-scale experiments, we will also explore the performance of adaptive EMC on a single GPU.

We first ran some experiments with the distributed EMC implementation as discussed in Section 3.2. We measured Amdahl’s efficiency,

\[
E = \frac{T(1)}{nT(n)} = \frac{T(1)}{T(1)(B + \frac{1}{n}(1-B))} = \frac{1}{nB + (1-B)}
\]

where \( n \) is the number of GPUs, and \( B \) the fraction of the algorithm that is serial. The results are shown in Figure 6. As can be seen we obtain a nearly perfect efficiency, at least up to 32 GPUs.

We next ran some really large-scale experiments for the fully distributed EMC implementation. In an attempt to follow the topology of our cluster, we distributed the rotational space (i.e. the \( j \)-direction) over a total of 25 nodes, and for the 4 GPUs belonging to the same node, we distributed the 10,000 diffraction patterns uniformly.

Defining one multiplication as 1 FLOP and one division as 8 FLOPs, we find by inspection that one EMC iteration requires about \( M_{\text{rot}} \times M_{\text{data}} \times M_{\text{pix}} \times 27 \times 10^{-9} \) GFLOPs. Table 2 lists the average execution time per iteration and the achieved GFLOPS per GPU. It is remarkable that we loose less than 4% floating point performance at 100 GPUs compared with 16 GPUs. In fact, the fully distributed EMC

---

**Table 1.** Sizes of the different datasets used in our experiments. Note that the value of \( M_{\text{pix}} \) is the result after binning the raw data \( 1024 \times 1024 \) into a coarser \( 64 \times 64 \) (or \( 128 \times 128 \)) format, and note also the relation \( M_{\text{grid}} = M_{\text{pix}}^3 \). In all of these experiments we used the value \( M_{\text{rot}} = 86,520 \).

| Set # | data     | \( M_{\text{data}} \) | \( M_{\text{pix}} \) |
|-------|-----------|------------------------|----------------------|
| 1     | Mimivirus | 198                    | 4096                 |
| 2     | Mimivirus | 198                    | 16,384               |
| 3     | synthetic | 1000                   | 4096                 |
| 4     | synthetic | 1000                   | 16,384               |
| 5     | synthetic | 10,000                 | 4096                 |
| 6     | synthetic | 10,000                 | 16,384               |
implementation achieves a higher floating point performance when compared with the single GPU implementation (32.9 and 39.4 GFLOPS, respectively, for \(M_{\text{pix}} = 4096\) and \(M_{\text{pix}} = 16,384\)). Indeed, these figures compare favorably with the online GPU benchmark (GPU Bench), where square matrix–matrix multiplication in single precision achieves \([4.4, 32.6, 181]\) GFLOPS, respectively, at the comparable matrix sizes \(N = [16,384, 65,536, 262,144]\).

Finally, we also performed some experiments with our adaptive EMC algorithm. For simplicity we used a single GPU only, and we reconstructed a small \(64 \times 64 \times 64\) intensity model. For the adaptivity we increased \(d\) in (7) from 5 to 12 according to the likelihood-based criterion (13). Figure 7 displays the relative difference of likelihood together with the execution time per iteration. As expected, the execution time increases as the resolution of the rotational space increases. Whenever \(d\) is increased, there is a sharp peak in the relative likelihood difference, indicating iterations that successfully increase the likelihood. The performance gain for the adaptive version is quite remarkable, 7341.6 seconds compared with 14,524 seconds for the original version, using 60 iterations for both runs. By the very simplicity of this approach, we expect that this gain of about 2 remains also for larger load cases.

### 4.3 Scalability

Instead of a scaling for a fixed problem-size, we study a metric of scalability that takes the data volume into account. Define the effective problem size as \(V(C) = M_{\text{rot}}(C) \times M_{\text{pix}}(C) \times M_{\text{data}}(C)\) for \(C\) a problem configuration. For two different such configurations \(C_1\) and \(C_2\), the scalability \(S(C_1, C_2)\) is then defined as

\[
S(C_1, C_2) = \frac{T(C_1)/T(C_2)}{V(C_1)/V(C_2)}
\]

where \(T(C)\) is the execution time for configuration \(C\). A superlinear scalability \(S(C_1, C_2) < 1\) means that the program works more efficiently with configuration \(C_2\), and also suggests that the program does not make full use of the computing power in \(C_1\). The situation \(S(C_1, C_2) > 1\) may happen for a problem which is compute-bound.

Figure 8 displays the scalabilities of different configurations. Notably, the cases \(S(2,1)\) and \(S(4,3)\) show superlinear scalability. Since only \(M_{\text{pix}}\) is changed in both cases, we judge that this is an effect of that the GPU kernels of the c- and e-step are not fully loaded. To the contrary, increasing \(M_{\text{data}}\) makes the scalabilities \(S(3,1)\) and \(S(4,2)\) larger than 1. For the latter case
we see that as we add more GPUs, we obtain a slightly better scalability. This indicates that as the size of the datasets increases, the fully distributed EMC becomes a favorable choice.

Finally, with $S(4,1)$, we compare a synthetic dataset to a real one, and we simultaneously increase $M_{\text{pix}}$ and $M_{\text{data}}$. However, the values in this group only slightly differ from, e.g., $S(2,4)$, which suggests that $M_{\text{data}}$ plays a more prominent role than $M_{\text{pix}}$ when measuring performance.

5 Conclusions

We have implemented the EMC algorithm for the assembly of randomly oriented diffraction patterns on GPUs using the CUDA framework and have extended the algorithm to run efficiently on multiple GPUs. We use two different partition schemes depending on the number of diffraction patterns. For a medium-sized dataset we partition the rotational space only, while for a large amount of data we also distribute the images themselves. We observe almost linear speedups for up to 100 GPUs and a parallel efficiency that thus compares very well with other MPI/CUDA applications (Jacobsen et al., 2010; Wang et al., 2013; Zaspel and Griebel, 2013). We also devised an adaptive technique by which the resolution is increased on par with the increase in likelihood. In our experiments this idea worked very well and we expect similar ideas to be useful in implementations on site where data is processed in a streaming fashion (Andreasson et al., 2014).

It seems likely that our implementation is going to develop and adapt further as larger datasets become available. Hopefully, the present software framework can substantially shorten the development cycle for novel algorithms targeting large and noisy datasets.

Hit ratios and data quality at the LCLS is steadily improving and in 2016 the European XFEL will become operational and provide a repetition rate of 27,000 Hz, a 200-fold increase compared with the LCLS. With this, data analysis will become a bottleneck and EMC in particular is the main computational step in this analysis. With this work we hope to ensure that data analysis can keep up with the rapid development of X-ray laser facilities, while simultaneously enabling the study of biological particles from single proteins to viruses.

Acknowledgements

Input and suggestions on an earlier draft of the paper from Filipe R. N. C. Maia and Janos Hajdu are hereby gratefully acknowledged.

Funding

This work was supported by the Swedish Research Council within the UPMARC Linnaeus center of Excellence (S. Engblom, J. Liu) and by the Swedish Research Council, the Knut och Alice Wallenberg Foundation, the European Research Council, the Röntgen-Ångström Cluster, and the Swedish Foundation for Strategic Research (T. Ekeberg, J. Liu).

References

Andreasson J, Martin AV, Liang M, et al. (2014) Automated identification and classification of single particle serial femtosecond X-ray diffraction data. Optics Express 22(3): 2497–2510. DOI:10.1364/OE.22.002497.

Chapman HN, Barty A, Marchesini S, et al. (2006) High-resolution ab initio three-dimensional X-ray diffraction microscopy. Journal of the Optics Society of America A 23(5): 1179–1200. DOI:10.1364/JOSAA.23.001179.

Dempster AP, Laird NM and Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society Series B Statistical Methodology 39: 1–38.

Ekeberg T (2012) Flash Diffractive Imaging in Three Dimensions. PhD Thesis, Uppsala university.

Emma P, Akreand R, Arthur J, et al. (2010) First lasing and operation of an Ångström-wavelength free-electron laser. Nature Photonics 4(9): 641–647. DOI:10.1038/nphoton.2010.176.

Engblom S and Liu J (2014) X-ray laser imaging of biomolecules using multiple GPUs. In: Wyrzykowski R, Dongarra J, Karczewski K and Was´niewski J (eds.) Parallel Processing and Applied Mathematics (Lecture Notes in Computer Science). Berlin: Springer, pp. 480–489. DOI: 10.1007/978-3-642-55224-3_45.

Fung R, Shneerson V, Saldin DK and Ourmazd A (2008) Structure from fleeting illumination of faint spinning objects in flight. Nature Physics 5(1): 64–67. DOI:10.1038/nphys1129.

Gaffney KJ and Chapman HN (2007) Imaging atomic structure and dynamics with ultrafast X-ray scattering. Science 316(5830): 1444–1448. DOI:10.1126/science.1135923.

GPU Bench (2012) GPU comparison report: Quadro K5000. Available at: http://folk.uio.no/jorgentr/GPUBenchmarkReport (accessed 11 April 2014).

Figure 8. The scalability $S(C_1, C_2)$ of configuration $C_1$ and $C_2$, according to (15), with $C_1$ and $C_2$ chosen among cases 1–4 in Table 1.
Jacobsen DA, Thibault JC and Senocak I (2010) An MPI-CUDA implementation for massively parallel incompressible flow computations on multi-GPU clusters. In: 48th AIAA aerospace sciences meeting and exhibit, volume 16. DOI:10.2514/6.2010-522.

Loh ND and Elser V (2009) Reconstruction algorithm for single-particle diffraction imaging experiments. Physical Review E 80(2): 026705. DOI:10.1103/PhysRevE.80.026705.

Loh ND, Bogan MJ, Elser V, et al. (2010) Cryptotomography: Reconstructing 3D Fourier intensities from randomly oriented single-shot diffraction patterns. Physical Review Letters 104: 225501. DOI:10.1103/PhysRevLett.104.225501.

Neal RM and Hinton GE (1998) A view of the EM algorithm that justifies incremental, sparse, and other variants. In: Learning in Graphical Models. Dodrecht: Kluwer, pp. 355–368.

Neutze R, Wouts R, van der Spoel D and Hajdu J (2000) Potential for biomolecular imaging with femtosecond X-ray pulses. Nature 406(6797): 752–757. DOI:10.1038/35021099.

Pennycook SJ, Hammond SD, Jarvis SA and Mudalige GR (2011) Performance analysis of a hybrid MPI/CUDA implementation of the NASLU benchmark. ACM SIGMETRICS Performance Evaluation Review 38(4): 23–29. DOI:10.1145/1964218.1964223.

Schneidmiller EA and Yurkov MV (2011) Photon beam properties at the European XFEL. December 2010 Revision.

Seibert MM, Ekeberg T, Maia FR, et al. (2011) Single mimic virus particles intercepted and imaged with an X-ray laser. Nature 470(7332): 78–81. DOI:10.1038/nature09748.

Serkez S, Kocharyan V, Saldin E, Zagorodnov I, Geloni G and Yefanov O (2014) Perspectives of imaging of single protein molecules with the present design of the European XFEL: part I - X-ray source, beamline optics and instrument simulations. Preprint http://arxiv.org/abs/1407.8450.

Shapiro D, Thibault P, Beetzand T, et al. (2005) Biological imaging by soft x-ray diffraction microscopy. Proceedings of the National Academy of Sciences USA 102(43): 15343–15346. DOI:10.1073/pnas.0503305102.

Wang Y, Dou Y, Guo S, Lei Y and Zou D (2013) CPU–GPU hybrid parallel strategy for cosmological simulations. Concurrency and Computation: Practice and Experience. DOI:10.1002/cpe.3046.

Zaspel P and Griebel M (2013) Solving incompressible two-phase flows on multi-GPU clusters. Computers and Fluids 80: 356–364. DOI:10.1016/j.compfluid.2012.01.021.

Author biographies

Tomas Ekeberg is a Researcher at the Department of Cell and Molecular Biology at Uppsala University. His research focuses on developing bioimaging methods using free-electron lasers. He received his PhD from Uppsala University in 2012.

Stefan Engblom is an Associate Professor at the Department of Information Technology, Uppsala University. He is associated with the Linnaeus Center of Excellence UPMARC, Uppsala Programming for Multicore Architectures Research Center. His research is centered around scientific computing and numerical analysis with a focus in the biosciences. He received his PhD from the Uppsala university in 2008, and became a Docent there in 2013.

Jing Liu is a PhD candidate jointly at the Departments of Cell and Molecular Biology and Information Technology, Uppsala University. Her research focuses on high-performance computing aspects of single-particle imaging methods. She received her MSc in Computer Science from Uppsala university in 2012.