McXtrace anno 2020 - Complex sample geometries and GPU acceleration

Knudsen, Erik B.; Willendrup, Peter K.; Garde, Jacob; Bertelsen, Mads

Published in:
Advances in Computational Methods for X-Ray Optics V

Link to article, DOI:
10.1117/12.2568753

Publication date:
2020

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
McXtrace anno 2020 - complex sample geometries and GPU acceleration

Knudsen, Erik, Willendrup, Peter, Garde, Jacob, Bertelsen, Mads


Event: SPIE Optical Engineering + Applications, 2020, Online Only
ABSTRACT

We present two main developments within the ray tracing package McXtrace in the recent timespan; The Union concept for building complex sample geometries which may also include sample environments, and the next generation code generator (nicknamed 3.0) which includes the option for GPU-acceleration through the OpenACC programming standard.

Union is a concept which allows beamline simulation users to define enclosed regions in which the regular sequential nature of McXtrace simulation is replaced by a scattering network. Within the network any object can scatter towards any other object. Through a pre-analysis of the scattering the this may be done without excessive computational effort - i.e. it is still practical on a standard desktop computer without high-end specs. We will discuss our result results with this concept and how it can be used to, for instance, assess background contributions.

Using the OpenACC programming paradigm, the simulation code generated by the new code generator, may now harness the power of novel GPU-cards for faster ray tracing, with fairly non-invasive changes to the user facing code. We will present results on where GPUs may be benefited from and what the user is required to do, in order to enjoy significant speed-ups.

Keywords: Manuscript format, template, SPIE Proceedings, LaTeX

1. INTRODUCTION

The McXtrace X-ray tracing software package has been around for some time, and as such provides an extensible, fast, and open software solution for simulating X-ray beamlines\(^1\) and X-ray experiments. Among the several capable software packages for performing beamline simulations and design such as OASYS,\(^2\) SRW,\(^3\) McXtrace has had a focus on the back-end of beamline simulation, in including and focusing on standard types of samples in the simulation package. Due to its structure using compiler technology and code-generation as well as the extremely widely available code-language c, it allows for an efficient and portable code. Code is always inherently optimized for the platform it is supposed to run on. Yet, it is clear that code must evolve with the times, and we have identified areas which have thus far not been well covered by McXtrace.

1. The package includes models of standard samples, and such a sample could well be for instance a sample environment to assess it’s influence on the signal. Thus far, however a user has had the choice of either simulating the environment or the sample itself. To target this use-case the Union concept was invented, where all components in a specific region of a McXtrace simulation may scatter to any other component within the region.

2. For some years we have witnessed the rise of GPUs as a means to massively accelerate computations. The drawback has often been a significant threshold of rewriting code combined with a lack of high-level tools to target more than one GPU architecture with one code-base. Two popular paradigms have so far been CUDA and OpenCL, but a recent development has been the advent of the standard OpenACC, which
sports free availability of compilers that support it\textsuperscript{4,5} and which supports the major brands of cards. A major advantage of OpenACC is that one can largely rely on the same programming constructs as for OpenMP. (In fact the implementations of OpenACC in gcc relies on the much of the same offloading concepts as OpenMP.)

In the following we will present the concepts used to target these two rather different issues/developments as separate tracks. Each section will contain its own subsections on each of the two tracks.

### 2. MODELS

#### 2.1 The union system

The Union concept in essence defines a specific regions in which ray tracing may follow a non-sequential path instead of the strict sequence generally followed in McXtrace. There is an added benefit to the concept in that it allows to separate physics from geometry, which in turn also makes it simpler for developers and uses to extend the library of physics.

In general terms, the system works in the way that with one (or more) certain regions, we define a set of geometric primaries. At present, cylinders, spheres and rectangular blocks are allow. Using these primaries we may build complex geometries. Should we require more complicated shapes we can "cut" away bits of other objects using other objects, as they are allowed to overlap. Each object is assigned a unique priority number which determines which is active in overlapping regions. Each geometry is furthermore assigned a material or "Vacuum", and in turn each material is assigned a set of scattering processes that are active within the objects containing that material. A scattering process is a piece of code that determines the probability of scattering (and in which direction) as a function of incoming wavevector. At any time when propagating a ray through a geometry we may therefore compute the total scattering cross-section as:

\[
S(q, \mathbf{k}) = \sum S_i(q, \mathbf{k}) ; \quad \sigma = \sum \sigma_i
\]  

(1)

where the elements of the sum denote the separate scattering processes. The wavevector of the ray is known at all times, and so is the remaining distance to travel, $l$, within the object, owing to the unique priority number. From here to determine the ray’s trajectory through the remainder of the object becomes a matter of random Monte Carlo choices. Should the scattering branch be chosen, one of the material’s processes and a scattering point along $\mathbf{k}_i$ are chosen randomly, and a new wavevector, $\mathbf{k}_f$, is picked from the distribution associated with that process. If so - the total cross section, and the remaining flight path in this new direction is recomputed. Thus, multiple scattering is included by design.

Scattering networks with several objects quickly become cumbersome for the simple reason that the number of possible connection is an arithmetic series sum, as illustrated in fig. 2:

\[
N_{tot} = \frac{1}{2}(n + 1) = O(n^2)
\]

(2)

where $n$ is the number of objects. To fill the phase space spanned by the connections between a large number of objects, can often requires vast amount of statistics, leading to impractical computation times. To avoid this the union system includes a pre-analysis of the scattering network, and excludes impossible connections. As in fig. 2 (left), let the network be a a set of objects within a volume surrounded by a double walled cylinder, mimicking for instance a cryostat or other sample environment. It is obvious that there is no direct connection between the outer shell and the innermost objects, and so we may remove many connections, leaving us with a much simpler network (fig. 2 (bottom right)), which our algorithms are much more likely to handle comfortably.

#### 2.2 GPU acceleration of McXtrace through OpenACC

Traditionally McXtrace relies on global variables throughout the process of ray tracing an experiment. This is obviously contrary to generally established coding practices, but nonetheless has enabled fairly simple code-generator that needs very few assumptions about an experiment. This simplicity comes at a cost, in that the...
Figure 1. Block sketch of the structure that the union concept follows. Left: Each Material is assigned one or more physical processes; each geometrical object is assigned one (or more) materials. Right: Example where material 1 has significant contributions from absorption and Compton scattering, and material 2 has contributions from powder scattering and absorption. The scattering region consists of a cylinder and a sphere of material 1, and a rectangular block of material 2.

Figure 2. Sketch of a union scattering network. Left: The geometry of region. A double-walled cylinder (red and blue) contains a sample object which consists of three discrete, slightly overlapping, objects (green, magenta, orange). Top Right: The full scattering network where all geometries can scatter to each other. Bottom Right: The reduced network with only the actually possible connections.

Figure 3. Visual ray tracing of a union geometry with three absorbing objects inside a a double walled cylinder. Left: Using the 3D visualizing tool mxdisplay. Middle and Right: XZ- and YZ-plane projections.
generated code can be hard to read and debug, albeit this is hardly ever done by anyone but the development team. Further, and more serious from a user’s perspective, the code also can becomes slow to optimize for modern compilers, thus increasing the time used for compiling large experiments with many components (>1000). Even so, the concept has been successful and still allows for large-scale parallelism via the MPI-paradigm, where parallelism is on the process level.

In terms of GPU-acceleration with OpenACC global variables become a more serious problem, where a large set of asynchronous threads run simultaneously. Here parallelism works on loop-level and lead to race-conditions and/or low efficiency, due to necessary mutex locks.

To answer this challenge, we have rewritten the code-generator of McXtrace (and its neutron counterpart McStas\(^7\)), to become more in line with modern best practices. In the new concept, each component type now has actual initialization and trace functions that may be run independently. One upshot of this is that the generated code becomes smaller. For a small scale experiment the impact is small to nonexistent but for a large-scale end-to-end simulation it may be significant (table 1).

Thus, the main ray tracing loop gets transferred to the GPU device, before running. As present, we have implemented no synchronization between the loop threads, meaning that threads may exit at different times in the loop.

### 3. RESULTS

#### 3.1 Union simulations

To quantify our work with union samples and environments thus far, we start with the inner part of the scattering network from fig. 2 and assign each of the objects absorbing materials: A rectangular block of Ag, a sphere of Pb, and a cylinder of W. Let a collimated beam of 30keV X-rays impinge on the set and record the transmitted intensity. The result is shown in fig. 4, where the various geometries are easily discernible. The whole set may be rotated and translated as any other McXtrace component as their positions are set relative to a reference point. In this case it is the centre of the sphere. Rotating the set by 90° around the X-axis (horizontal and perpendicular to the optical axis, Z, we get the pattern shown in fig. 4 (right). From this we can also deduce the order of priority between the objects, as they overlap: The cylinder takes precedence over the sphere which takes precedence over the block.

We now take the simulation one step further and add the double wall shell around the set, and also add a Compton-like\(^8\) isotropic scattering function to all the elements. Also adding loggers to the union set allows to visualize scattering points within a set, in terms of heat maps. Figure 5 shows examples of such loggers as projections of scattering events. As is also seen in fig. 5, logger may be made conditional of e.g. which order of scattering they are supposed to log. As an example, fig. 5(bottom) shows only the 2nd order scattering within the union set, where the cylinder walls that are not illuminated by the direct beam are clearly visible. Being granted the possibility to specifically view a specific scattering order helps clarify which scattering paths may...
influence the signal in the main detectors, and may given handles on how to best shield against background in an experiment. In fig. 5 we can clearly see what we do also get scattering, albeit weak, from the double wall cylinder, even though it is not illuminated directly by the beam.

3.2 GPU acceleration and code-size.

A very positive side-effect of the new code-generation model employed mainly to enable GPU-acceleration, is that code is reused among components of the same type. This makes the size of the generated code only weakly dependent on the size (in terms of number of components) of the simulation. Table 1 shows the size of the generated code for a series of examples. The \texttt{Null} and \texttt{Trivial} are dummy instruments that in the former cases does nothing, in the latter generates a set of photons and measures them without manipulating them in any way. Unsurprisingly there’s nothing gained for these two. \texttt{MAX IV Danmax} on the other hand is a full beamline simulation complete with undulator source, optics, sample etc. which shows a significant reduction in code-size. As an extreme case, we have included a simulation of a proposed X-ray space telescope probe \texttt{Baby IAXO} which has an unusually large number of components of the same kind. Here the improvement is substantial.

One additional positive side-effect of the new code generation scheme is that the code now inherently becomes much easier to profile using standard tool. This in turn helps to identify and in extension remove bottlenecks. Figure 6 shows execution raw times for some different scenarios. We have performed 2 difference simulations on two different GPU-architectures available to us: NVIDIA V100 Tesla 16GB, and NVIDIA Quadro K1200. The former is a high-end card targeted or use in compute clusters, whereas the latter is a low-power (45 W) card that may be installed in a standard desktop unit with no necessary upgrades. Two different example simulations have be chosen:

1. A single crystal diffraction case: SX.
2. A Compound Refractive Lens scenario: CRL.
Table 1. Comparison of code size (Lines of Code, LoC) and compilation time for some different simulation cases. These test have been run on a standard desktop machine with GCC version 8 and the PGI c-compiler version 19.10. The new code-generation scheme has a slightly larger overhead, but clearly is superior when the number of components becomes large. The Null and Trivial simulations are mere dummy test instruments only useful for unit testing, whereas MAX IV Danmax and Baby IAXO are real life examples. Old denotes the code generated with a stock McXtrace 1.5 installation, whereas new refers to the next generation code generator nicknamed cogen 3.

<table>
<thead>
<tr>
<th>Simulation</th>
<th># components</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null old</td>
<td>1</td>
<td>5.6 k</td>
</tr>
<tr>
<td>Null new</td>
<td>1</td>
<td>6.3 k</td>
</tr>
<tr>
<td>Trivial old</td>
<td>3</td>
<td>8.1 k</td>
</tr>
<tr>
<td>Trivial new</td>
<td>3</td>
<td>8.7 k</td>
</tr>
<tr>
<td>MAX IV Danmax old</td>
<td>105</td>
<td>51 k</td>
</tr>
<tr>
<td>MAX IV Danmax new</td>
<td>105</td>
<td>25 k</td>
</tr>
<tr>
<td>Baby IAXO old</td>
<td>321</td>
<td>239 k</td>
</tr>
<tr>
<td>Baby IAXO new</td>
<td>321</td>
<td>49 k</td>
</tr>
</tbody>
</table>

In the former case the majority of the computation effort is spent searching lists of reflections for scattering vectors that satisfy Bragg’s law, where in the latter case the main effort is repeatedly solving 2nd order equations to find ray intersection with parabolic surfaces surfaces of the lenses.

From analyzing fig. 6 we find the picture to be rather complex. In most cases we get significant speed-up from offloading to GPU, with the single crystal scattering simulation on the Quadro K1200 card the exception, where performance is worse. We also find that code as-is is not particularly well suited for openMP-parallelism. Interestingly the 16 core CPU with 32 threads exhibits almost no speed gain with respect to the 4 core CPU. On a positive note, speed ups of 2 and 3 orders of magnitude the high-end GPU-card is promising.

3.3 Discussion and Outlook

First, we have presented a subsystem of McXtrace dubbed Union which allows specific regions of a simulation to have a non-sequential ray tracing flow, while retaining much of the computational efficiency of a strict sequential flow. This is most often relevant in cases where scattering angle can become fairly large, e.g. around and in the experimental sample. As an example we have shown a simplistic use of it elucidate the scattering interplay between a oddly shaped sample and a sample container. We speculate that a particularly interesting use-case is for RIXS-beamlines where many competing processes may be present in the signal. Each of the physical processes may be targeted on its own and then simply added to the material in question to provide a complete solution, as the system has been designed with this in mind. Certainly, detailed and accurate simulations of

Figure 6. Execution times (Left) and Normalized speed-up (Right) for runs using two different graphics cards (V100 and K1200) and host CPUs (16 and 4 cores, resp.) with acceleration through openMP (multicore). CRL refers to a simulations with a set of 100 Be-lenses, whereas SX denotes simulations with a single Mo-crystal as scattering element. The PGI-compiler was used. For speed-ups, execution times for equivalent runs with a single thread on CPU has been used.
RIXS instrumentation exists (see for example\textsuperscript{11}), yet non-direct beam scattering is rarely taken into account. We believe the union concept can provide instrument developers and beamlines scientists a unique handle to understand the sometimes complicated interplay between scattering elements in a beamline. It is clear that to really understand the full picture more detailed models of various scattering processes than what has been presented in the present study is imperative.

Second, we have a shown the next generation code-generator of McXtrace, specifically aimed at taking advantage of the large scale parallelism of modern-day GPUs through the OpenACC standard. We have found the approach to be successful, in a best case scenario providing orders of magnitude speed improvement. We do also find the speed gain to be application specific, in one studied case even slowing down simulations. As a positive side effect we have shown that size of the generated code does no longer depend on the number of components in a simulation. Instead it now depends only the number of types of components. A fact that is important for optics built from many small but similar pieces. The may be further exploited to split components into many simpler parts, thus reducing the code-complexity.

REFERENCES