Estimation of hardware requirements for data reduction with Mantid at ESS

Simon Heybrock

simon.heybrock@esss.se

August 8, 2018

Contents

1 Introduction

1.1 Overview

The aim of this document is to capture our understanding of what hardware resources will be required for reducing data with Mantid for ESS. While we attempt to give some estimates for, e.g., required core counts, it will become clear below there are too many uncertainties. In practice the main contribution of this document may thus be the description of how various parameters affect what hardware is required, allowing us the continuously update and improve our estimates.

Data reduction covers three main cases, live reduction, interactive reduction, and script-based reduction.

Live reduction Preliminary/simplified on-the-fly reduction of data and visualization is a key promise of ESS and will be required for all instruments, any time the beam is on, and at times also with beam off for other calibration work. It is critical that this is available at all times so a dedicated pool of hardware may be required.

Interactive reduction The general view is that data at ESS will be so large that users cannot do data reduction on their laptop or desktop PC. Therefore DMSC needs to provide resources for all interactive workflows using the Mantid GUI (MantidPlot or its successor) for data reduction and visualization. This is required during a user visit as well as a certain time period after the experiment. Furthermore, we need to provide access to such resources *before* an experiment such that users can familiarize themselves with the tools and provided software infrastructure, avoiding wasting beam time due to software problems. For interactive reduction, due to the interactive nature, high CPU use will alternate with shorter or longer idle breaks. However, generally there will be a rather large permanent memory requirement, i.e., clustering interactive user sessions on a single compute node is only possible to a limited extent, and will require nodes with a large amount of memory.

Script-based reduction Reduction of data based on a (Python) script. This covers automatic reduction and batch reduction. This is suitable for being run on a cluster using a standard queuing system with adequate means to ensure that queue lengths stay within limits. For example, one could imagine that jobs for data reduction of a running experiment need to be prioritized over modelling jobs of a experiment that happened a long time ago.

1.2 Threading and MPI support

The Mantid framework is multithreaded and algorithms run by a script or interactively by the GUI will typically run with multiple threads. To allow for scaling data reduction beyond

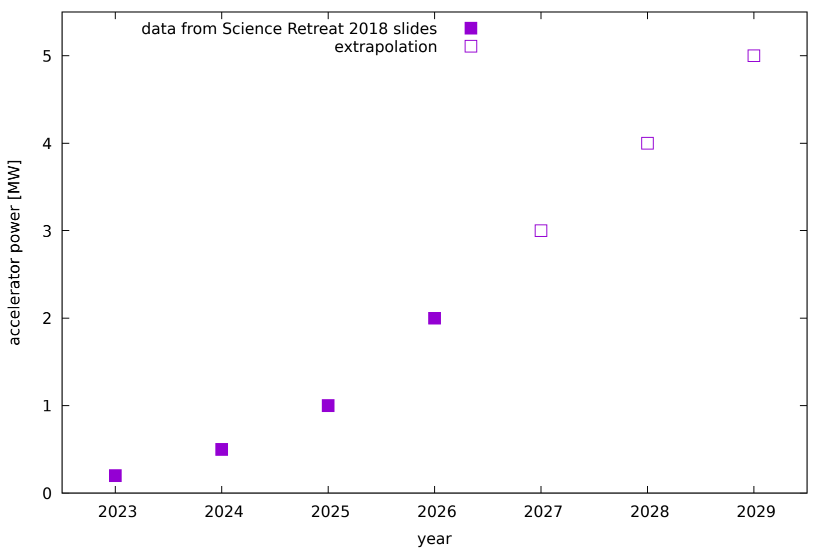

Figure 1: Estimate of accelerator power ramp up.

a single node, MPI support has been added to Mantid. MPI support is only available for script-based reduction. In principle it could also be used for live reduction, provided that it is only a passive way to observe raw and reduced data. There is no MPI support connected to the GUI and there are no plans for this either.[1].

In various benchmarks we observed that generally an equivalent threaded run of a Mantid reduction script takes longer than the same reduction done with MPI:[2] The scaling with the number of used cores is worse. The reason for this is not fully understood but it is probably related to the fork-join threading approach. Thus we cannot directly transfer results and scaling behaviors obtained for MPI to a threaded version of Mantid. For machines with a moderate number of cores of up to roughly 10 cores we can probably use the rule of thumb that the threaded reduction will be $2\times$ slower than the MPI reduction.

1.3 Ramp-up in hot commissioning and early operations

The initial rates and pixel counts will be lower than the design values due to the ramp-up of accelerator power and incomplete detector coverage. In Fig. 1 we show the estimate for the accelerator power ramp-up used in the following for scaling the expected event rates. ESS instrument scientists expect a linear scaling of the useable wavelength with the accelerator power. In Tab. 1 we list the pixel counts for the first instruments, i.e., effectively N_{spec}. We do not list the imaging beamline ODIN. It is not suitable for joint estimation with the other beamlines and it reduction hardware requirements should probably be considered jointly with its requirements for reconstruction and analysis. To obtain N_{bin} we need to combine N_{spec} with the number of bins per spectrum. This depends on the resolution of the instrument, which can be configurable. Details for this and other parameters are listed in Sec. 4.5.

[1]That is, there will be no equivalent to the client-server MPI mode supported by, e.g., ParaView.

[2]This is true even without the optimized MPI-based NeXus loader.

[3]We did not yet receive pixel counts from DREAM, but MAGIC uses same technology with a pixel size of 3 mm $\times$ 6 mm with 32 voxel layers. Full coverage is 5.11 sr with a sample-detector distance of 1.25 m. Day-1 coverage is 1.81 sr.

Instrument	Pixel count day 1	Pixel count final	Comment
BEER	200k	400k	not including SANS upgrade
BIFROST	?	5k	low pixel count but scanning
CSPEC	400k	750k	
DREAM	4000	12000k	estimated[3]
ESTIA	250k	500k	
LoKI	750k	1500k	
MAGIC	1440k	2880k	

Table 1: Pixel counts for the first instruments. Generally instruments to not have full detector coverage at day 1 and will add more pixels in an upgrade, typically a couple of years later.

Instrument	SANS2D	MERLIN	WISH
Pixel count	122888	286729	778245
10^5 events in 14 frames	1379	915	400
10^6 events in 14 frames	284	93	38
10^7 events in 14 frames	23	8	3

Table 2: Number of consumed frames per second. For ESS there are 14 frames (pulses) per second, i.e., we require a consumption rate of at least 14. The update timeout is set to 1 s.

2 Live reduction

The discussion in this section is based on live reduction benchmarks conducted by Lamar Moore. See the reference given in App. A for details. In Tab. 2 we show benchmark results for the event consumption rate of the Kafka event listener in Mantid without processing or visualization. This test is done using a default (non-MPI) build of Mantid. MPI support for the Kafka listener is not implemented currently.

With the exception of the SANS2D data point at 10^5 the scaling with the event rate is linear.[4] The benchmarks also show a (very approximately) linear scaling of the consumption rate with the number of pixels. This is not entirely expected. There are two aspects that can be used to explain the scaling: With more pixels the write behavior to the `EventWorkspace` becomes more random leading to a less and less efficient use of cache and the rest of the memory subsystem. This effect should be expected to reach a steady state above a certain number of pixels. In addition to that, a separate vector needs to be allocated for every pixel to store the events. This allocation cost could grow linearly with the pixel count.

By increasing the update timeout, which was set to one second in Tab. 2, the consumption rate can be improved moderately. A couple of other optimization also appear to be possible. On the other hand, live display of the raw data using the `InstrumentView`, (simplified) data reduction, and live display of the reduced data add extra overhead. Combining these opposite effects, which — according to preliminary benchmarks — all appear to be in the order of a $2\times$ change, leads us to conclude that the numbers given in Tab. 2 are a reasonable estimate. That is, without having access to much more detailed workflows and instrument models we have the crude estimate

[4]It is likely that for the high rate of more than 1000 frames per second other effects become relevant. Since we are not interested in such high frame rates we ignore this data point and assume linear scaling.

$$R_{\text{live}} = \frac{N_{\text{frame}}^{\text{max}}}{14} = \frac{3 \cdot 10^{13}\ \text{s}}{14 f_{\text{event}} N_{\text{spec}}} = \frac{2 \cdot 10^{12}\ \text{s}}{f_{\text{event}} N_{\text{spec}}}, \tag{1}$$

where $N_{\text{frame}}^{\text{max}}$ is the maximum number of consumed frames as in Tab. 2, 14 is the number of frames per second at ESS, and f_{event} is the event rate. We require $R_{\text{live}} > 1$ to be able to support live reduction without MPI. In Fig. 2 we show the implications for the first ESS instruments, depending on the accelerator power. With the exception of DREAM and possibly MAGIC, we seem to be able to handle most cases for most instruments even at higher accelerator power. However, we will definitely require MPI support in live reduction for DREAM from day one.

For experiments with very high throughput, i.e., quick changes of sample-environment parameters or sample, we have to deal with additional complexity in the live reduction workflow. This could include more advanced filtering, display and comparison of multiple results at the same time,[5] and overhead from frequent transitions to a new run. We have not yet started development of these features so benchmarks are not available. That being said, high throughput experiments are not likely to be required in the early days. Furthermore, UX is likely to be a bigger issue than performance in this case. We consider it unlikely that this scenario will cause a significant overall difference in the required hardware.

3 Interactive reduction

As discussed in Sec. 1.2 there is no MPI support for the GUI or interactive reduction in general, limiting scaling with the number of available cores. Generally we do not expect good scaling far beyond 10 cores. For reductions that are too slow in that case there are several options that can be used in practice:

- Use slow interactive reduction for a single run/sample, apply resulting workflow to other samples using script-based reduction on the cluster.
- Use interactive reduction using a reduced or compressed set of data, e.g., by loading only every N-th pulse or by compressing events, apply resulting workflow to other samples using script-based reduction on the cluster.
- Use slow interactive reduction and use different sessions to do something else in the meantime, such as reducing another sample.

With exception of the second option, there is a minimum amount of required total RAM. We can attempt to estimate the requirement as

$$M_{\text{interactive}} = 2\ \text{GByte} + N_{\text{spec}} \cdot 256\ \text{Byte} + N_{\text{workspace}}(24 N_{\text{bin}} + 16 N_{\text{event}})\ \text{Byte}, \tag{2}$$

with

- $N_{\text{workspace}}$ is the number of workspaces present in Mantid at the same time. For interactive reduction this can typically be larger than the absolute minimum required for a fixed script-based reduction. Therefore we use $N_{\text{workspace}} = 10$ here, which is higher than what we will require below for script-based reduction in Sec. 4.5.

[5]e.g., waterfall plots

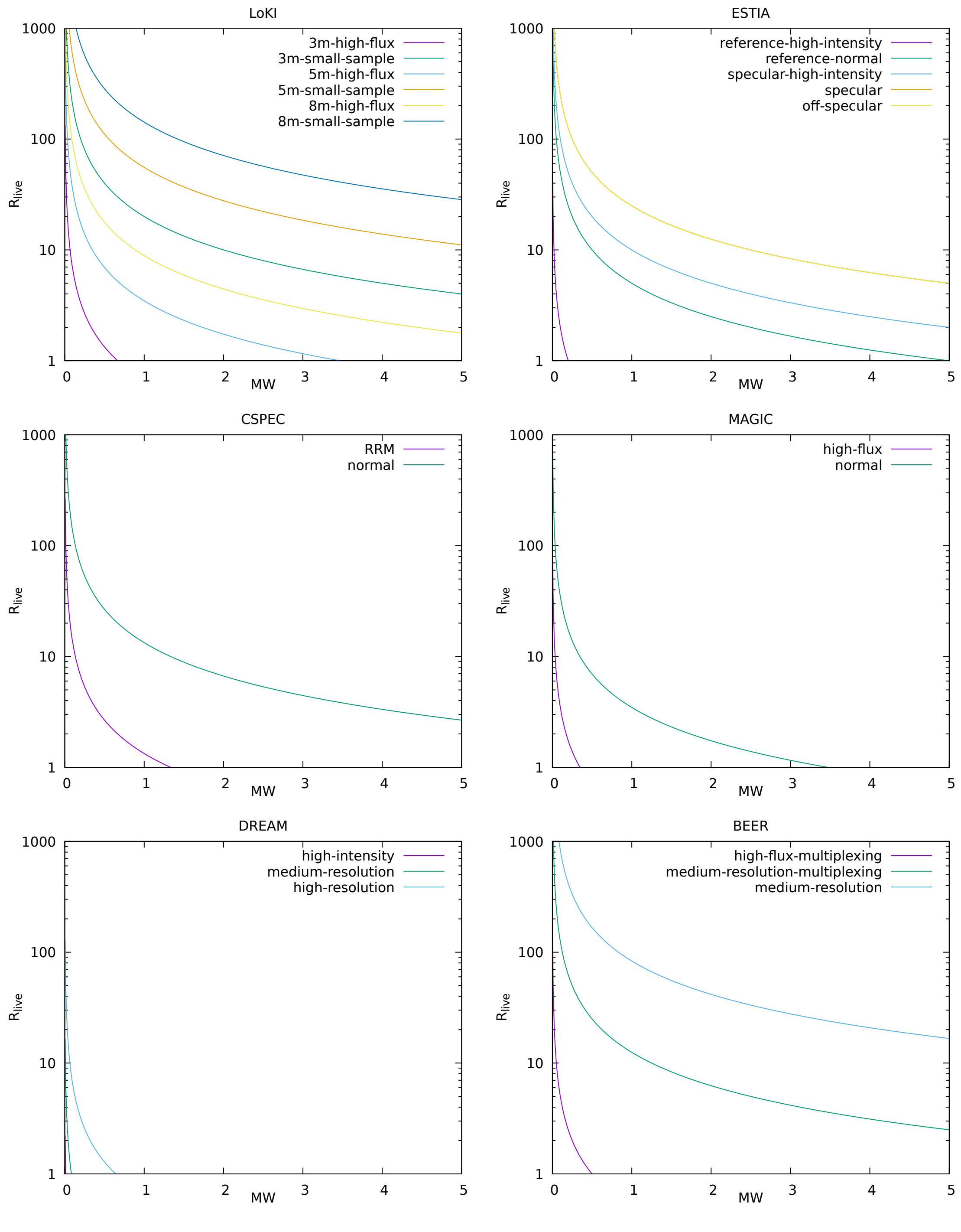

Figure 2: Reduction speed ratio R_{live} for the first ESS instruments. With increasing accelerator power, f_{event} increases and R_{live} decreases. We require $R_{\text{live}} > 1$ to be able to handle the data. The plot range has been chosen accordingly — a graph hitting the lower horizontal axis implies that the rate cannot be handled anymore. For BIFROST the pixel count is very low and we are not confident that we can extrapolate the scaling benchmarks from Tab. 2 this far. We have therefore excluded the panel for BIFROST but expect that the rate can be handled even at a power of 5 MW. The imaging beamline ODIN is a special case and has thus also been excluded from this figure.

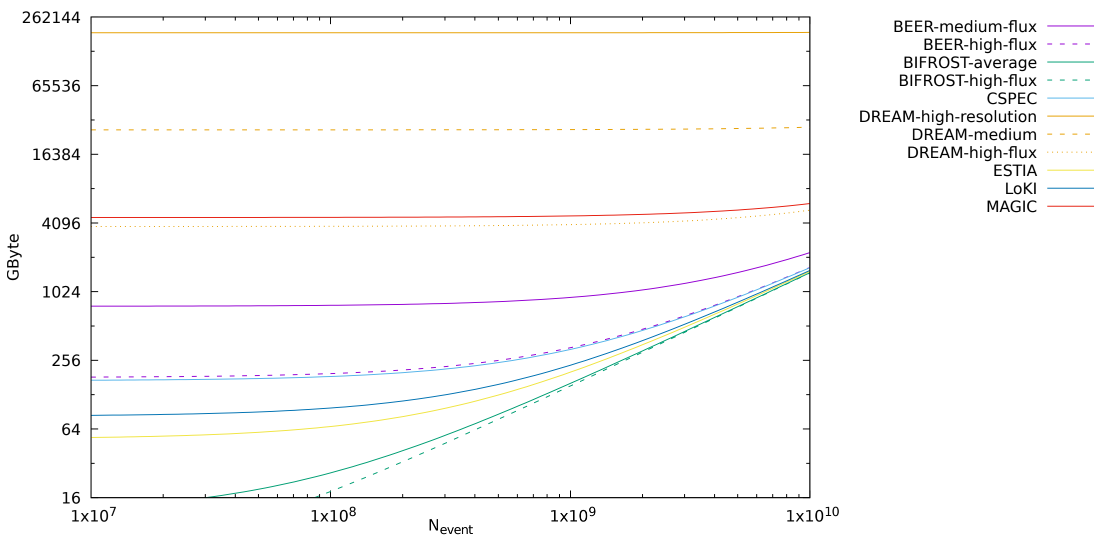

Figure 3: Scaling of memory requirements for interactive reduction with the event count per run. The plot uses the final pixel count from Tab. 1 and N_{bin} based on Tab. 4. The powder and single crystal diffractometers DREAM, MAGIC, and BEER show an exceedingly high memory requirement due to the high pixelation and time resolution. This is probably not a real and realistic requirement, see the discussion in Sec. 4.5. The worst-case assumption that a histogram-representation for all workspaces is required leads to a rather high baseline requirement — scaling with the total event count only becomes visible near 10^9 events and above. Excluding the special cases listed above, the required memory will mostly not exceed 256 GByte or up to 1 TByte for large runs with 10^{10} events. The requirement can be lowered further by reducing $N_{\mathrm{workspace}}$ and/or by avoiding binning of most workspaces, i.e., working purely in event mode.

- 2 GByte base memory. We assume a larger base requirement than in Sec. 4.5 for GUI requirements.
- 256 Byte of memory per pixel to represent the instrument geometry. This is assumed to be shared between workspaces.
- 24 Byte per bin per workspace for double-precision X, Y, and E.
- 16 Byte per event per workspace for double-precision time-of-flight and pulse-time.

Considering the projected duration of a single run t_{run} for design-power event rates, most experiments seem to use roughly 10^8 to 10^9 events. Based on this equation, we give example requirements for various instrument configurations in Fig. 3. Notable exceptions to this are the inelastic beamlines and the single-crystal beamlines, if we consider merging data from all sample rotation angles. Traditionally single crystal diffraction and inelastic scattering have the highest memory requirements but it is currently unclear whether this merge step would be done in Mantid.

Instrument scientists expect that typically 5 interactive sessions would be required per instrument at any given time. This figure is to be combined with requirements for data analysis since typically the interactive sessions would be used for both, data reduction and data analysis. For data reduction, a lot of the time Mantid will actually be idle or unused. Thus simply

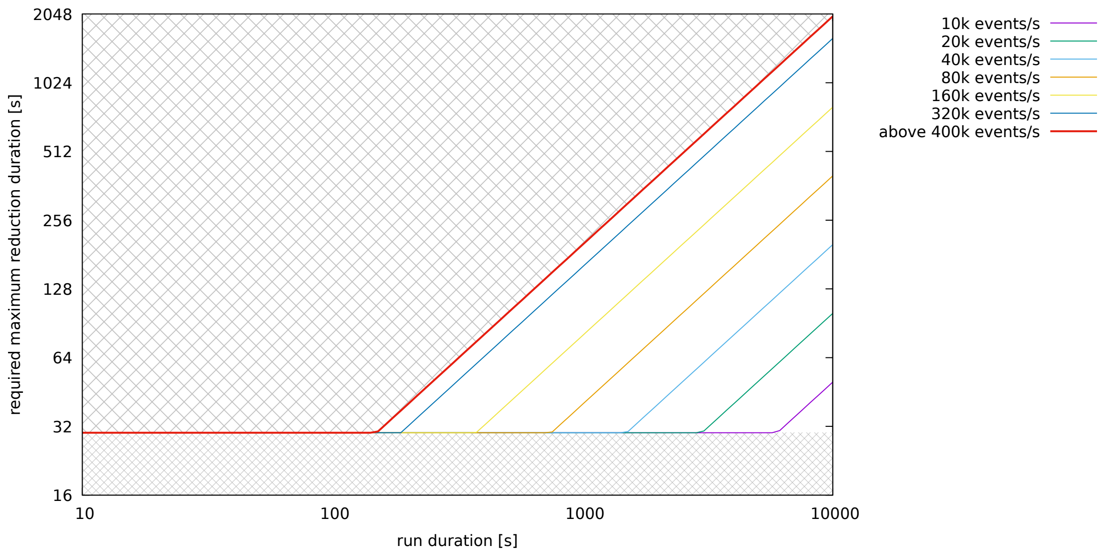

Figure 4: Imposed requirements for the maximum reduction time. The checked areas indicate regions that are forbidden by the defined limits. For runs that are short relative to the event rate we are governed by the 30-second limit. Above that limit, at low event rates the required reduction duration is low compared to the run duration. With increasing event rate we get closer to the limit of 5× speedup indicated by the bold red line.

adding the required 5 sessions to the required numbers for data analysis is likely to overestimate the required resources.

4 Script-based reduction

4.1 Scaling analysis

The number of cores required for MPI-based reduction depends on the required maximum runtime for a reduction. It is unclear what a good limit for the runtime $t_{\text{reduction}}$ is. For now we define:

- The reduction should be 5× faster than the experiment.
- To ensure that long-running experiments with low event rates are processed quickly we require that a minimum of $2 \cdot 10^6$ events/s are processed.
- We never require a runtime below 30 seconds.

A visualization of these requirements is given in Fig. 4.

With a couple of approximations (which are probably minor for this purpose and compared to other sources of uncertainty) we can describe the time required to reduce a set of data as

$$t_{\text{reduction}} = t_0 + \frac{N_{\text{bin}} t_{\text{bin}} + N_{\text{event}} t_{\text{event}}}{N_{\text{core}}} + \frac{N_{\text{event}}}{B_{\text{max}}}, \tag{3}$$

where

- N_{bin} is the number of bins in the workflow, i.e., the number of spectra N_{spec} multiplied by the number of bins per spectrum.[6] The number of bins per spectrum depends on the bandwidth and resolution of the instrument. As a rule of thumb, for a given energy resolution δE we require a bin size of $\delta E/10$.

- N_{event} is the total number of events that are being handled in the reduction workflow. This can include events from multiple files, e.g., for a sample run and a background run.

- N_{core} is the number of cores (MPI ranks) used in the reduction. We are not considering a hybrid threading+MPI approach.[7].

- t_0 is a constant time specific to the reduction workflow. It includes anything that does not depend and the number of spectra or number of events. Typically this includes small parts of the time spend in every algorithm, time for loading experiment logs from NeXus files, time for loading auxiliary files, and other overheads.

- t_{bin} is the (computed) time to run the workflow for a single bin.

- t_{event} is the (computed) time to run the workflow for a single event.

- B_{max} is the number of events that can be loaded from the file system per second.

The equation is motivated by an analysis of benchmark results — see Sec. 4.3 — and the computational structure of data reduction. N_{core} can be adjusted such that $t_{\mathrm{reduction}}$ fulfills the conditions listed above. There can be cases where the conditions are violated, e.g., when the event rate is high and B_{max} is too low. More details on this equation can be found in App. B.

Our benchmarks based on existing instruments and investigations of parameters for a series of ESS instruments show that any of the terms in Eq. (3) can be relevant or even dominant, depending on the instrument, the experiment, and the number of used cores:

- For instruments that produce many small files with few spectra and events, the t_0 term can become dominant. This can in theory be improved by processing multiple files in the same workflow, but at this point making such an assumption is not justified. It is thus important to capture the typical run duration for each instrument.

- Even if the t_0 term is not dominant, a run may contain relatively few events relative to the pixel count. In that case the N_{bin} term is important. Given that many ESS instruments have a high pixel count this could very well become a dominating factor.[8]

- For instruments with many events we may be using many cores to offset the reduction cost. In that limit the bandwidth-limiting term becomes relevant.

[6]Typically N_{spec} is the number of pixels of the instrument, but it can be different, e.g., when multiple adjacent pixels are grouped due to a lower resolution requirement or due to potential projections related to volumetric detectors.

[7]A few specialized algorithms are using threading internally, even in an MPI run.

[8]In many experiments the total number of events required for each measurement may not increase drastically over present-day experiments. However, the ESS event *rate* and pixel counts are often much higher, so the balance may shift to our disadvantage.

Based on the time for reducing a single run we can thus compute the average number of cores required for reducing data for the instrument. Note that this does *not* include live reduction and interactive sessions. We define

$$\langle N_{\text{core}} \rangle = p_{\text{use}} N_{\text{reduction}} N_{\text{core}} \frac{t_{\text{reduction}}}{t_{\text{run}}}, \tag{4}$$

where

- p_{use} is the probability that the instrument is in use in a specific operating mode. Typically there are multiple operating modes for an instrument so there will be multiple $\langle N_{\text{core}} \rangle$ values that need to be be summed.
- $N_{\text{reduction}}$ is the number of times data is reduced. Typically this should be small, e.g., 1, 2, or 3, but especially in the early days there will be exceptions.
- t_{run} is the duration of a single run.

For convenience, we can expand the master equation and obtain

$$\begin{aligned}
\langle N_{\text{core}} \rangle &= \frac{N_{\text{reduction}}}{t_{\text{run}}} \left[N_{\text{core}} \left(t_0 + \frac{N_{\text{event}}}{B_{\text{max}}} \right) + N_{\text{bin}} t_{\text{bin}} + N_{\text{event}} t_{\text{event}} \right] && (5) \\
&= N_{\text{reduction}} \left[N_{\text{core}} \left(\frac{t_0}{t_{\text{run}}} + \frac{f_{\text{event}}}{B_{\text{max}}} \right) + \frac{N_{\text{bin}} t_{\text{bin}}}{t_{\text{run}}} + f_{\text{event}} t_{\text{event}} \right] && (6) \\
&= N_{\text{reduction}} \left[N_{\text{core}} \left(\frac{t_0 f_{\text{event}}}{N_{\text{event}}} + \frac{f_{\text{event}}}{B_{\text{max}}} \right) + \frac{N_{\text{bin}} t_{\text{bin}} f_{\text{event}}}{N_{\text{event}}} + f_{\text{event}} t_{\text{event}} \right]. && (7)
\end{aligned}$$

All three lines are trivial variations of each other and are merely listed to highlight different aspects of the scaling behavior.

It is important to note that $\langle N_{\text{core}} \rangle$ depends on N_{core}, i.e., the more cores we use, the higher our overall hardware requirement. Reasons for using more cores are primarily (1) to reduce the time for a single reduction to something that is acceptable for users, and (2) to work around memory limitations on a single node. There are two terms that depend on $1/t_{\text{run}}$ (see Eq. (6)), i.e., shorter runs will increase the number of required cores even if all other parameters such as the event rate f_{event} are unchanged.

4.2 Event filtering

The discussion in this section is based on event filtering benchmarks conducted by Neil Vaytet. See the reference given in App. A for details. Many ESS instruments will require filtering events in one way or another. Benchmarks indicate that the contribution to the overall runtime is less than 10%, i.e., we do not need to handle it explicitly in our evaluations. While the cost of filtering itself is not significant, in many cases we are dealing with many more workspaces after the filtering process, e.g., one for every temperature interval in a temperature scan. If the majority of data reduction can be done *before* filtering this is not a problem. However, if a significant portion of data treatment can only happen after the filtering/splitting process the cost is increasing significantly since we are effectively dealing with many distinct runs. This problem is probably similar to the case of sample rotation angles discussed later in Tab. 11 for MAGIC and needs to be investigated closer in the future.

4.3 Baseline benchmarks

To establish an estimate for the parameters t_0, t_{bin}, t_{event}, and B_{max} in Eq. (3) a number of benchmarks have been undertaken.

1. Using an MPI-build of Mantid we measured run times of workflows for SANS, powder diffraction, and direct geometry spectroscopy (using the ISIS workflow for SANS2D, the `SNSPowderDiffraction` workflow for PG3, and the `DgsReduction` workflow for CNCS). We studied the scaling behavior with the number of used cores (MPI ranks), the number of events, and also the number of bins per spectrum. By looking at the various limits (few/many cores and few/many events) we can extract an approximation for t_0, t_{bin}, t_{event}. All these experiments were done on a single node, a dual-socket Xeon E5-2620 v3 running at 2.40GHz.

2. Scaling beyond a single node has been established in older benchmarks in 2015, again using the `SNSPowderDiffraction`. This benchmark showed that scaling beyond a single node does not lead to any significant slowdown and scaling works well up to a high number of MPI ranks, with the exception of algorithms loading or saving files.

3. For the special case of I/O we rely on benchmarks for the recently optimized parallel event loader for NeXus files. Currently only benchmarks with a local SSD are available and show scaling up to nearly 10^8 loaded events per second (slightly lower for compressed files) with 10 used cores. Benchmarks with a parallel file system are pending.

The experiments show a surprisingly consistent pattern, independent of the technique:

- t_0 is about 10 seconds.
- t_{bin} varies from 1/1.000.000 seconds for histogram-heavy reductions to about 1/10.000.000 seconds for reductions with near-ubiquitous use of event-mode.
- t_{event} is about 1/1.000.000 seconds, or slightly smaller.
- B_{max} is about 50.000.000 events/second for an SSD, tests with a parallel file system are pending.

4.4 Other factors

With low accelerator power and lower detector coverage in the early days it is to be expected that the type of experiments will be different from what is envisioned for full power. For example, larger samples will be used to compensate for the lower rates. The compensation could be of up to a order of magnitude in size, at least for certain experiments. While some instruments indicate they will do so, e.g., LoKI and DREAM, not all of them will, e.g., CSPEC. We have *not* included a scaling term to represent this effect in the numbers given in this document. Ensuring that the compute cluster size ramp up is sufficiently ahead of the accelerator power ramp up could help to deal with this uncertainty.

Parameter	Value	Comment
t_0	10 s	
t_{bin}	2×10^{-7} s	
t_{event}	10^{-6} s	
$B_{\max}$	10^8 s^{-1}	
scale factor for N_{event}	2	to approximate cost from need to reduce sample *and* background for every sample run TODO above we state the Nevent includes this, this is just internal based on count required for sample run (need it if we give that in a table?)
$N_{\text{workspace}}$	5	
$N_{\text{reduction}}$	2	
$N_{\text{core}}^{\max}$	512	set as arbitrary limit to reduce the number of nonsensical results from special parameter combinations in conflict with generic performance requirements from Sec. 4.1

Table 3: List of parameters used to generate Tabs. 5–11. See also Eq. (8) regarding memory requirements and related parameters.

4.5 Results

This section discusses results obtained for script-based reduction with the performance model introduced earlier. In the following we give example listings of reduction times, core counts, and memory consumption for various instruments and operating modes. We strongly emphasize that the resulting values heavily depend on parameters that we cannot estimate with good accuracy. Parameters used for generating the listings can be found in Tab. 3. Resolution options and the resulting bin counts are given in Tab. 4. Python scripts to generate the tables are described in App. D. The resulting data for the given parameters can be found in Tabs. 5–11.

We adopt a simplified model to estimate the amount of required main memory.[9] We define

$$M \equiv N_{\text{core}}(1\ \text{GByte} + N_{\text{spec}} \cdot 256\ \text{Byte}) + N_{\text{workspace}}(24N_{\text{bin}} + 16N_{\text{event}})\ \text{Byte}, \tag{8}$$

$$M_{\text{core}} \equiv \frac{M}{N_{\text{core}}} = 1\ \text{GByte} + N_{\text{spec}} \cdot 256\ \text{Byte} + \frac{N_{\text{workspace}}}{N_{\text{core}}}(24N_{\text{bin}} + 16N_{\text{event}})\ \text{Byte}. \tag{9}$$

with

- 1 GByte base memory per core.
- 256 Byte of memory per pixel per core to represent the instrument geometry. This is assumed to be shared between workspaces.
- 24 Byte per bin per workspace for double-precision X, Y, and E.
- 16 Byte per event per workspace for double-precision time-of-flight and pulse-time.

[9] This is similar to but different from Eq. (2) used for the case of interactive reduction.

Instrument	Mode	Bins per spectrum	Comment
BEER	medium-flux	8500	
	high-flux	2000	
BIFROST	average	850	
	high-flux	110	is highest flux always lowest resolution?
CSPEC	(all)	1000	no large resolution variation possible?
DREAM	high-resolution	71000	
	medium	10000	
	high-intensity	1420	
ESTIA	(all)	450	
LoKI	(all)	240	
MAGIC	(all)	7100	

Table 4: List of bin counts used to generate Tabs. 5–11. Bin counts have been estimated based on instrument proposals and input from instrument scientists. Details can be found in App. C.

This is a naive model that assumes that we require a histogram representation of the data at some point during the reduction. Histogram mode is used for visualization as well as data reduction steps that cannot be done in event mode, such as normalization. Accounting for memory for a full histogram representation can lead to very high memory requirements, see in particular the discussion on DREAM below.

Provided that the compute cluster is large enough, summing all $\langle N_{\text{core}} \rangle$ for all relevant instrument configurations from Tabs. 5–11 could yield the required size of the compute cluster. However, there are clearly cases with high peak-core requirements. Furthermore, in the early days with few operating instruments and a smaller cluster it needs to be ensured that an appropriate partition is always available within a to be defined time frame. Determining the resulting required size of the compute cluster is beyond the scope of this investigation.

Instrument with a very high time resolution and high pixel count such as DREAM suffer from an exceedingly high memory requirement. In reality there are probably ways to alleviate this issue, arising from a full histogram representation of the data. Working in event mode we have the following options:

1. For visualization purposes, on-the-fly binning of event data could be implemented.

2. The transition to histogram mode can be done histogram-by-histogram at the point of histogram summation, e.g., for powder diffraction workflows used for DREAM in the algorithm `DiffractionFoccusing`. In the case of powder diffraction this is already supported by the existing workflow. However, this is only possible provided that there are no issues arising from performing normalization in event mode, as described in `http://journals.iucr.org/j/issues/2016/02/00/fs5119/index.html`.

One of the parameters that is hardest to pin down in discussions with the instrument teams is t_{run}, which influences the hardware requirements in two ways:

- Shorter t_{run} implies *more* runs, i.e., more reductions.

- The t_0 term and the N_{bin} term in Eq. (3) become more dominant in relation to N_{event} since the former two terms stay constant while the latter scales with $1/t_{\text{run}}$. To fulfill the speedup conditions N_{core} needs to be increased when t_{run} increases.

Pixels	Mode	p-beam [MW]	Use [%]	f_{event} [s^{-1}]	t_{run} [s]	$t_{\text{reduction}}$ [s]	N_{core}	$\langle N_{\text{core}} \rangle$	M_{core} [GByte]
		0.2	70.0	$6 \cdot 10^3$	166.7	25	23	5	10
		0.5	70.0	$1.5 \cdot 10^4$	66.7	25	23	13	10
	medium-flux	1.0	70.0	$3 \cdot 10^4$	33.3	25	23	25	10
		2.0	70.0	$6 \cdot 10^4$	16.7	25	23	49	10
		5.0	70.0	$1.5 \cdot 10^5$	6.7	25	23	121	10
		0.2	20.0	$4 \cdot 10^4$	25.0	25	23	10	10
		0.5	20.0	$1 \cdot 10^5$	10.0	25	23	23	10
200000	medium-flux-multiplexing	1.0	20.0	$2 \cdot 10^5$	5.0	25	23	46	10
		2.0	20.0	$4 \cdot 10^5$	2.5	25	23	92	10
		5.0	20.0	$1 \cdot 10^6$	1.0	25	23	229	10
		0.2	10.0	$1 \cdot 10^6$	1.0	24	6	29	9
		0.5	10.0	$2.5 \cdot 10^6$	0.4	24	6	72	9
	high-flux-multiplexing	1.0	10.0	$5 \cdot 10^6$	0.2	24	6	143	9
		2.0	10.0	$1 \cdot 10^7$	0.1	24	6	285	9
		5.0	10.0	$2.5 \cdot 10^7$	0.0	24	6	711	9
		0.2	70.0	$1.2 \cdot 10^4$	83.3	25	45	20	10
		0.5	70.0	$3 \cdot 10^4$	33.3	25	45	48	10
	medium-flux	1.0	70.0	$6 \cdot 10^4$	16.7	25	45	96	10
		2.0	70.0	$1.2 \cdot 10^5$	8.3	25	45	191	10
		5.0	70.0	$3 \cdot 10^5$	3.3	25	45	476	10
		0.2	20.0	$8 \cdot 10^4$	12.5	25	45	37	10
		0.5	20.0	$2 \cdot 10^5$	5.0	25	45	91	10
400000	medium-flux-multiplexing	1.0	20.0	$4 \cdot 10^5$	2.5	25	45	182	10
		2.0	20.0	$8 \cdot 10^5$	1.2	25	45	363	10
		5.0	20.0	$2 \cdot 10^6$	0.5	25	45	907	10
		0.2	10.0	$2 \cdot 10^6$	0.5	25	11	109	10
		0.5	10.0	$5 \cdot 10^6$	0.2	25	11	273	10
	high-flux-multiplexing	1.0	10.0	$1 \cdot 10^7$	0.1	25	11	545	10
		2.0	10.0	$2 \cdot 10^7$	0.1	25	11	1089	10
		5.0	10.0	$5 \cdot 10^7$	0.0	25	11	2723	10

Table 5: Example estimate of hardware required for script-based reduction for BEER. In this and the following tables, the 'Pixels' column refers to the build-out phase. 'Use' is a guess of the fraction a particular instrument mode/configuration is used. The values are chosen to add up to 100% but could be adjusted to take into account time without beam and time for changing samples. If the obtainable $t_{\text{reduction}}$ violates the conditions such as a $5\times$ speedup stated in Sec. 4.1 the corresponding t_{run} in the table is highlighted in red.

For BEER the biggest uncertainty comes from the duration of a single run. The measurement time for each individual measurement point in the sample can be very low. If each such point is considered as an independent measurement we obtain a high hardware cost, especially in *high-flux-multiplexing* mode. If a combined treatment of the points in the sample scan is possible the requirement is likely to be much lower in the high-flux case. For the *high-flux-multiplexing* mode t_{run} looks unrealistic (below the pulse length), yielding an enormous core count. The rate given by the instrument team for this mode is two orders of magnitude above the stated average rate so our naive computation of t_{run} may not correspond to the actual use case, i.e., the result should be interpreted with care. For the *medium-flux-multiplexing* mode we do not have an actual statement of the event rate but we have chosen to include it as an estimate for an intermediate case.

Pixels	Mode	p-beam [MW]	Use [%]	f_{event} [s^{-1}]	t_{run} [s]	$t_{\mathrm{reduction}}$ [s]	N_{core}	$\langle N_{\mathrm{core}} \rangle$	M_{core} [GByte]
5000	high-flux	0.2	20.0	$4 \cdot 10^4$	25.0	12	1	1	2
		0.5	20.0	$1 \cdot 10^5$	10.0	12	1	1	2
		1.0	20.0	$2 \cdot 10^5$	5.0	12	1	1	2
		2.0	20.0	$4 \cdot 10^5$	2.5	12	1	2	2
		5.0	20.0	$1 \cdot 10^6$	1.0	12	1	5	2
	average	0.2	80.0	$4 \cdot 10^3$	250.0	13	1	1	2
		0.5	80.0	$1 \cdot 10^4$	100.0	13	1	1	2
		1.0	80.0	$2 \cdot 10^4$	50.0	13	1	1	2
		2.0	80.0	$4 \cdot 10^4$	25.0	13	1	1	2
		5.0	80.0	$1 \cdot 10^5$	10.0	13	1	3	2

Table 6: Example estimate of hardware required for script-based reduction for BIFROST. Our information on BIFROST is a bit dated so this table may be inaccurate. The pixel count for BIFROST is very low, yielding a comparatively low hardware requirement. However, the detector bank is scanned and data from multiple positions is to be included in the same reduction and analysis. This is likely connected with an overhead which has not been included here.

Pixels	Mode	p-beam [MW]	Use [%]	f_{event} [s^{-1}]	t_{run} [s]	$t_{\mathrm{reduction}}$ [s]	N_{core}	$\langle N_{\mathrm{core}} \rangle$	M_{core} [GByte]
400000	normal	0.2	20.0	$2.1 \cdot 10^4$	46875.0	377	6	1	21
		0.5	20.0	$5.3 \cdot 10^4$	18750.0	377	6	1	21
		1.0	20.0	$1.1 \cdot 10^5$	9375.0	377	6	1	21
		2.0	20.0	$2.1 \cdot 10^5$	4687.5	377	6	1	21
		5.0	20.0	$5.3 \cdot 10^5$	1875.0	290	8	1	16
	RRM	0.2	80.0	$2.1 \cdot 10^5$	4687.5	377	6	1	21
		0.5	80.0	$5.3 \cdot 10^5$	1875.0	290	8	2	16
		1.0	80.0	$1.1 \cdot 10^6$	937.5	160	16	5	9
		2.0	80.0	$2.1 \cdot 10^6$	468.7	76	45	12	4
		5.0	80.0	$5.3 \cdot 10^6$	187.5	36	362	111	2
750000	normal	0.2	20.0	$4 \cdot 10^4$	25000.0	388	6	1	28
		0.5	20.0	$1 \cdot 10^5$	10000.0	388	6	1	28
		1.0	20.0	$2 \cdot 10^5$	5000.0	388	6	1	28
		2.0	20.0	$4 \cdot 10^5$	2500.0	388	6	1	28
		5.0	20.0	$1 \cdot 10^6$	1000.0	164	16	2	12
	RRM	0.2	80.0	$4 \cdot 10^5$	2500.0	388	6	2	28
		0.5	80.0	$1 \cdot 10^6$	1000.0	164	16	5	12
		1.0	80.0	$2 \cdot 10^6$	500.0	97	32	10	7
		2.0	80.0	$4 \cdot 10^6$	250.0	47	128	39	3
		5.0	80.0	$1 \cdot 10^7$	100.0	34	512	281	2

Table 7: Example estimate of hardware required for script-based reduction for CSPEC. We assume that combining data from multiple runs (rotations) is *not* part of data reduction and is handled by the analysis software. If the transformation to $(\vec{Q}, \Delta E)$ and subsequent summation were to be done in Mantid the hardware requirement would increase significantly.

Pixels	Mode	p-beam [MW]	Use [%]	f_{event} [s^{-1}]	t_{run} [s]	$t_{\mathrm{reduction}}$ [s]	N_{core}	$\langle N_{\mathrm{core}} \rangle$	M_{core} [GByte]
4000000	high-resolution	0.2	33.0	$1.7 \cdot 10^4$	28846.2	246	256	2	127
		0.5	33.0	$4.3 \cdot 10^4$	11538.5	246	256	4	127
		1.0	33.0	$8.7 \cdot 10^4$	5769.2	246	256	8	127
		2.0	33.0	$1.7 \cdot 10^5$	2884.6	246	256	15	127
		5.0	33.0	$4.3 \cdot 10^5$	1153.8	180	362	38	90
	medium	0.2	33.0	$1.3 \cdot 10^5$	3750.0	220	45	2	103
		0.5	33.0	$3.3 \cdot 10^5$	1500.0	220	45	5	103
		1.0	33.0	$6.7 \cdot 10^5$	750.0	119	91	10	52
		2.0	33.0	$1.3 \cdot 10^6$	375.0	70	181	23	27
		5.0	33.0	$3.3 \cdot 10^6$	150.0	38	512	85	11
	high-intensity	0.2	33.0	$1 \cdot 10^6$	500.0	87	32	4	23
		0.5	33.0	$2.5 \cdot 10^6$	200.0	37	128	16	8
		1.0	33.0	$5 \cdot 10^6$	100.0	28	256	48	5
		2.0	33.0	$1 \cdot 10^7$	50.0	28	256	96	5
		5.0	33.0	$2.5 \cdot 10^7$	20.0	28	256	240	5
12000000	high-resolution	0.2	33.0	$5.2 \cdot 10^4$	9615.4	355	512	13	190
		0.5	33.0	$1.3 \cdot 10^5$	3846.2	355	512	32	190
		1.0	33.0	$2.6 \cdot 10^5$	1923.1	355	512	63	190
		2.0	33.0	$5.2 \cdot 10^5$	961.5	355	512	125	190
		5.0	33.0	$1.3 \cdot 10^6$	384.6	355	512	312	190
	medium	0.2	33.0	$4 \cdot 10^5$	1250.0	215	128	15	109
		0.5	33.0	$1 \cdot 10^6$	500.0	89	362	43	42
		1.0	33.0	$2 \cdot 10^6$	250.0	69	512	94	31
		2.0	33.0	$4 \cdot 10^6$	125.0	69	512	187	31
		5.0	33.0	$1 \cdot 10^7$	50.0	69	512	466	31
	high-intensity	0.2	33.0	$3 \cdot 10^6$	166.7	32	362	47	10
		0.5	33.0	$7.5 \cdot 10^6$	66.7	29	512	146	8
		1.0	33.0	$1.5 \cdot 10^7$	33.3	29	512	291	8
		2.0	33.0	$3 \cdot 10^7$	16.7	29	512	581	8
		5.0	33.0	$7.5 \cdot 10^7$	6.7	29	512	1451	8

Table 8: Example estimate of hardware required for script-based reduction for DREAM. See the discussion in the text regarding the extreme memory requirement in *high-resolution* mode. N_{core} is high and is dominated by the N_{bin} term of Eq. (3). Therefore N_{core} also scales badly with t_{run}. Our estimate of t_{run} is currently quite uncertain and this table needs to be updated once we have access to a better estimate. It should also be noted that while our benchmarks include the event-mode powder-diffraction workflow which will also be used for DREAM, the benchmarks are based on POWGEN with only 43121 pixels, i.e., we are dealing with an extrapolation over more than two orders of magnitude.

Pixels	Mode	p-beam [MW]	Use [%]	f_{event} [s^{-1}]	t_{run} [s]	$t_{\text{reduction}}$ [s]	N_{core}	$\langle N_{\text{core}} \rangle$	M_{core} [GByte]
250000	reference-high-intensity	0.2	0.1	$2 \cdot 10^6$	500.0	93	32	1	4
		0.5	0.1	$5 \cdot 10^6$	200.0	38	256	1	2
		1.0	0.1	$1 \cdot 10^7$	100.0	34	512	1	2
		2.0	0.1	$2 \cdot 10^7$	50.0	34	512	1	2
		5.0	0.1	$5 \cdot 10^7$	20.0	34	512	2	2
	reference-normal	0.2	1.0	$8 \cdot 10^4$	12500.0	367	6	1	16
		0.5	1.0	$2 \cdot 10^5$	5000.0	367	6	1	16
		1.0	1.0	$4 \cdot 10^5$	2500.0	367	6	1	16
		2.0	1.0	$8 \cdot 10^5$	1250.0	214	11	1	9
		5.0	1.0	$2 \cdot 10^6$	500.0	93	32	1	4
	specular-high-intensity	0.2	5.0	$4 \cdot 10^4$	250.0	24	3	1	6
		0.5	5.0	$1 \cdot 10^5$	100.0	24	3	1	6
		1.0	5.0	$2 \cdot 10^5$	50.0	24	3	1	6
		2.0	5.0	$4 \cdot 10^5$	25.0	24	3	1	6
		5.0	5.0	$1 \cdot 10^6$	10.0	24	3	1	6
	specular	0.2	50.0	$1.6 \cdot 10^4$	625.0	24	3	1	6
		0.5	50.0	$4 \cdot 10^4$	250.0	24	3	1	6
		1.0	50.0	$8 \cdot 10^4$	125.0	24	3	1	6
		2.0	50.0	$1.6 \cdot 10^5$	62.5	24	3	2	6
		5.0	50.0	$4 \cdot 10^5$	25.0	24	3	3	6
	off-specular	0.2	50.0	$1.6 \cdot 10^4$	62500.0	367	6	1	16
		0.5	50.0	$4 \cdot 10^4$	25000.0	367	6	1	16
		1.0	50.0	$8 \cdot 10^4$	12500.0	367	6	1	16
		2.0	50.0	$1.6 \cdot 10^5$	6250.0	367	6	1	16
		5.0	50.0	$4 \cdot 10^5$	2500.0	367	6	1	16
500000	reference-high-intensity	0.2	0.1	$4 \cdot 10^6$	250.0	46	128	1	2
		0.5	0.1	$1 \cdot 10^7$	100.0	34	512	1	2
		1.0	0.1	$2 \cdot 10^7$	50.0	34	512	1	2
		2.0	0.1	$4 \cdot 10^7$	25.0	34	512	2	2
		5.0	0.1	$1 \cdot 10^8$	10.0	34	512	4	2
	reference-normal	0.2	1.0	$1.6 \cdot 10^5$	6250.0	371	6	1	18
		0.5	1.0	$4 \cdot 10^5$	2500.0	371	6	1	18
		1.0	1.0	$8 \cdot 10^5$	1250.0	216	11	1	11
		2.0	1.0	$1.6 \cdot 10^6$	625.0	119	23	1	6
		5.0	1.0	$4 \cdot 10^6$	250.0	46	128	1	2
	specular-high-intensity	0.2	5.0	$8 \cdot 10^4$	125.0	26	4	1	8
		0.5	5.0	$2 \cdot 10^5$	50.0	26	4	1	8
		1.0	5.0	$4 \cdot 10^5$	25.0	26	4	1	8
		2.0	5.0	$8 \cdot 10^5$	12.5	26	4	1	8
		5.0	5.0	$2 \cdot 10^6$	5.0	26	4	3	8
	specular	0.2	50.0	$3.2 \cdot 10^4$	312.5	26	4	1	8
		0.5	50.0	$8 \cdot 10^4$	125.0	26	4	1	8
		1.0	50.0	$1.6 \cdot 10^5$	62.5	26	4	2	8
		2.0	50.0	$3.2 \cdot 10^5$	31.2	26	4	4	8
		5.0	50.0	$8 \cdot 10^5$	12.5	26	4	9	8
	off-specular	0.2	50.0	$3.2 \cdot 10^4$	31250.0	371	6	1	18
		0.5	50.0	$8 \cdot 10^4$	12500.0	371	6	1	18
		1.0	50.0	$1.6 \cdot 10^5$	6250.0	371	6	1	18
		2.0	50.0	$3.2 \cdot 10^5$	3125.0	371	6	1	18
		5.0	50.0	$8 \cdot 10^5$	1250.0	216	11	2	11

Table 9: Example estimate of hardware required for script-based reduction for ESTIA. The average required core count is low. With exception of the reference measurements in *high-intensity* mode the required hardware the peak core count is also low. The peak core count can be reduced by accepting a longer reduction time for the reference measurements, which are infrequent and only of short duration.

Pixels	Mode	p-beam [MW]	Use [%]	f_{event} [s^{-1}]	t_{run} [s]	$t_{\mathrm{reduction}}$ [s]	N_{core}	$\langle N_{\mathrm{core}} \rangle$	M_{core} [GByte]
750000	3m-high-flux	0.2	10.0	$2 \cdot 10^5$	500.0	42	8	1	5
		0.5	10.0	$5 \cdot 10^5$	200.0	33	11	1	4
		1.0	10.0	$1 \cdot 10^6$	100.0	27	16	1	3
		2.0	10.0	$2 \cdot 10^6$	50.0	27	16	2	3
		5.0	10.0	$5 \cdot 10^6$	20.0	27	16	5	3
	3m-small-sample	0.2	10.0	$6.7 \cdot 10^3$	15015.0	42	8	1	5
		0.5	10.0	$1.7 \cdot 10^4$	6006.0	42	8	1	5
		1.0	10.0	$3.3 \cdot 10^4$	3003.0	42	8	1	5
		2.0	10.0	$6.7 \cdot 10^4$	1501.5	42	8	1	5
		5.0	10.0	$1.7 \cdot 10^5$	600.6	42	8	1	5
	5m-high-flux	0.2	10.0	$3.8 \cdot 10^4$	2604.2	42	8	1	5
		0.5	10.0	$9.6 \cdot 10^4$	1041.7	42	8	1	5
		1.0	10.0	$1.9 \cdot 10^5$	520.8	42	8	1	5
		2.0	10.0	$3.8 \cdot 10^5$	260.4	42	8	1	5
		5.0	10.0	$9.6 \cdot 10^5$	104.2	27	16	1	3
	5m-small-sample	0.2	30.0	$2.4 \cdot 10^3$	41666.7	42	8	1	5
		0.5	30.0	$6 \cdot 10^3$	16666.7	42	8	1	5
		1.0	30.0	$1.2 \cdot 10^4$	8333.3	42	8	1	5
		2.0	30.0	$2.4 \cdot 10^4$	4166.7	42	8	1	5
		5.0	30.0	$6 \cdot 10^4$	1666.7	42	8	1	5
	8m-high-flux	0.2	10.0	$1.5 \cdot 10^4$	6666.7	42	8	1	5
		0.5	10.0	$3.8 \cdot 10^4$	2666.7	42	8	1	5
		1.0	10.0	$7.5 \cdot 10^4$	1333.3	42	8	1	5
		2.0	10.0	$1.5 \cdot 10^5$	666.7	42	8	1	5
		5.0	10.0	$3.8 \cdot 10^5$	266.7	42	8	1	5
	8m-small-sample	0.2	30.0	$9.4 \cdot 10^2$	106609.8	42	8	1	5
		0.5	30.0	$2.3 \cdot 10^3$	42643.9	42	8	1	5
		1.0	30.0	$4.7 \cdot 10^3$	21322.0	42	8	1	5
		2.0	30.0	$9.4 \cdot 10^3$	10661.0	42	8	1	5
		5.0	30.0	$2.3 \cdot 10^4$	4264.4	42	8	1	5
1500000	3m-high-flux	0.2	10.0	$4 \cdot 10^5$	250.0	46	8	1	8
		0.5	10.0	$1 \cdot 10^6$	100.0	29	16	1	5
		1.0	10.0	$2 \cdot 10^6$	50.0	29	16	2	5
		2.0	10.0	$4 \cdot 10^6$	25.0	29	16	4	5
		5.0	10.0	$1 \cdot 10^7$	10.0	29	16	10	5
	3m-small-sample	0.2	10.0	$1.3 \cdot 10^4$	7507.5	46	8	1	8
		0.5	10.0	$3.3 \cdot 10^4$	3003.0	46	8	1	8
		1.0	10.0	$6.7 \cdot 10^4$	1501.5	46	8	1	8
		2.0	10.0	$1.3 \cdot 10^5$	750.8	46	8	1	8
		5.0	10.0	$3.3 \cdot 10^5$	300.3	46	8	1	8
	5m-high-flux	0.2	10.0	$7.7 \cdot 10^4$	1302.1	46	8	1	8
		0.5	10.0	$1.9 \cdot 10^5$	520.8	46	8	1	8
		1.0	10.0	$3.8 \cdot 10^5$	260.4	46	8	1	8
		2.0	10.0	$7.7 \cdot 10^5$	130.2	29	16	1	5
		5.0	10.0	$1.9 \cdot 10^6$	52.1	29	16	2	5
	5m-small-sample	0.2	30.0	$4.8 \cdot 10^3$	20833.3	46	8	1	8
		0.5	30.0	$1.2 \cdot 10^4$	8333.3	46	8	1	8
		1.0	30.0	$2.4 \cdot 10^4$	4166.7	46	8	1	8
		2.0	30.0	$4.8 \cdot 10^4$	2083.3	46	8	1	8
		5.0	30.0	$1.2 \cdot 10^5$	833.3	46	8	1	8
	8m-high-flux	0.2	10.0	$3 \cdot 10^4$	3333.3	46	8	1	8
		0.5	10.0	$7.5 \cdot 10^4$	1333.3	46	8	1	8
		1.0	10.0	$1.5 \cdot 10^5$	666.7	46	8	1	8
		2.0	10.0	$3 \cdot 10^5$	333.3	46	8	1	8
		5.0	10.0	$7.5 \cdot 10^5$	133.3	29	16	1	5
	8m-small-sample	0.2	30.0	$1.9 \cdot 10^3$	53304.9	46	8	1	8
		0.5	30.0	$4.7 \cdot 10^3$	21322.0	46	8	1	8
		1.0	30.0	$9.4 \cdot 10^3$	10661.0	46	8	1	8
		2.0	30.0	$1.9 \cdot 10^4$	5330.5	46	8	1	8
		5.0	30.0	$4.7 \cdot 10^4$	2132.2	46	8	1	8

Table 10: Example estimate of hardware required for script-based reduction for LoKI. Peak as well as average core count requirements are low.

Pixels	Mode	p-beam [MW]	Use [%]	f_{event} [s^{-1}]	t_{run} [s]	$t_{\text{reduction}}$ [s]	N_{core}	$\langle N_{\text{core}} \rangle$	M_{core} [GByte]
1440000	normal	0.2	20.0	$2 \cdot 10^4$	25.0	26	128	54	11
		0.5	20.0	$5 \cdot 10^4$	10.0	26	128	134	11
		1.0	20.0	$1 \cdot 10^5$	5.0	26	128	267	11
		2.0	20.0	$2 \cdot 10^5$	2.5	26	128	533	11
		5.0	20.0	$5 \cdot 10^5$	1.0	26	128	1331	11
	high-flux	0.2	80.0	$2 \cdot 10^5$	2.5	26	128	2130	11
		0.5	80.0	$5 \cdot 10^5$	1.0	26	128	5324	11
		1.0	80.0	$1 \cdot 10^6$	0.5	26	128	10647	11
		2.0	80.0	$2 \cdot 10^6$	0.2	26	128	21294	11
		5.0	80.0	$5 \cdot 10^6$	0.1	26	128	53234	11
2880000	normal	0.2	20.0	$4 \cdot 10^4$	12.5	26	256	213	11
		0.5	20.0	$1 \cdot 10^5$	5.0	26	256	533	11
		1.0	20.0	$2 \cdot 10^5$	2.5	26	256	1065	11
		2.0	20.0	$4 \cdot 10^5$	1.2	26	256	2130	11
		5.0	20.0	$1 \cdot 10^6$	0.5	26	256	5323	11
	high-flux	0.2	80.0	$4 \cdot 10^5$	1.2	26	256	8517	11
		0.5	80.0	$1 \cdot 10^6$	0.5	26	256	21291	11
		1.0	80.0	$2 \cdot 10^6$	0.2	26	256	42581	11
		2.0	80.0	$4 \cdot 10^6$	0.1	26	256	85161	11
		5.0	80.0	$1 \cdot 10^7$	0.1	26	256	212902	11

Table 11: Example estimate of hardware required for script-based reduction for MAGIC. The numbers given here are very high — and most likely unrealistic — due to a combination of a high pixel count and a low event count per sample rotation angle, with only 10^5 to 5×10^5 events per rotation angle quoted by the instrument scientists. Assuming that each rotation is treated independently results in the figures given here. Probably this is not a good assumption and a joint treatment of sample rotation angles needs to be considered instead.

In combination this implies that in practice $\langle N_\text{core}\rangle$ shows *worse than linear* scaling with $1/t_\text{run}$.[10] Any uncertainty related to t_run will thus be amplified.

5 Hardware components

5.1 Main memory

Distinguishing between *default* and *high-mem* machine configurations probably makes sense. With the parameter settings used for Tabs. 5–11 we obtain a typical memory requirement of M_core up to $M_\text{core}^\text{default} = 10$ GByte. BIFROST is likely to fall in the $M_\text{core}^\text{default}$ class even when considering a detector scan. The are some major exceptions which we label $M_\text{core}^\text{high-mem}$ like CSPEC with M_core up to 30 GByte and DREAM exceeding 100 GByte per core in its highest-resolution mode. MAGIC is also in the $M_\text{core}^\text{high-mem}$ class, but the precise requirement depends on how the combination of rotation angles is handled.[11]

It is worth noting that a large part of the huge memory requirements for DREAM and MAGIC come from the large number of detector pixels resulting in a large N_spec. Both instruments use the same volumetric detector technology with 32 layers of voxels. This is similar for CSPEC. While there are long-term plans to use the depth information for ray-tracing, it is not absolutely necessary from the beginning. One could argue that we can thus project the voxel data onto a grid of pixels, gaining an order of magnitude reduction of N_spec. However, we cannot rely on that at this point since there are a couple of complications:

1. Voxels are not aligned with the secondary flight paths so any projection will need to rebin the data, splitting events into fractional events, unless done in histogram-mode.
2. Voxel-layer-dependent calibration may be necessary, preventing and early transition to a lower N_spec.

We currently do not have benchmarks relating the Mantid performance to RAM bandwidth. However, given the computational structure of Mantid, we expect that a higher RAM bandwidth is more important than a large number of cores per node.

5.2 Network

For data reduction, Mantid does not put particularly high demands on the interconnect between nodes. The most important factor is the speed of the connection to the file system. During file system reads and writes there will also be significant communication between nodes. Non-I/O algorithms in Mantid typically have either no communication or communication with only moderate latency and bandwidth requirements. While Ethernet is not sufficient, our current feeling is that any Infiniband network is, even if it is an older revision.

5.3 Filesystem I/O

The filesystem bandwidth is crucial. For Tabs. 5–11 we used $B_\text{max} = 10^8\ \text{s}^{-1}$ to model the maximum read bandwidth *for a particular reduction*, i.e., we assumed that the would be *no*

[10] Unless we fix N_core and drop our speedup conditions.

[11] The figures in Tab. 11 do *not* explicitly include handling of multiple angles or merging, unless it is covered by the generic factor $N_\text{workspace}$.

resource contention. The quoted $B_{\max}$ corresponds to an approximate bandwidth of 1 GByte/s, depending also on the NeXus compression level. Determining when conflicts could arise is beyond the scope of this report. If conflicts are likely $B_{\max}$ should be reduced to incorporate the contention into the model.

While all workflows need to read significant amounts of data, the same is not true for writing:

- BIFROST, CSPEC, and potentially MAGIC will write large files, whereas the output files for BEER, DREAM, ESTIA, and LoKI are small.
- For interactive reduction we need to take into account saving and loading of Mantid project files which often contain all workspaces and can thus become very large.

5.4 SSDs

Fast local SSDs could potentially be helpful for inelastic and single crystal, handling large multi-dimensional workspaces exceeding the available RAM. Mantid supports a file-backed mode for this case. This is something we considered as part of an investigation by Anton Piccardo-Selg, see `https://github.com/DMSC-Instrument-Data/documents/tree/master/investigations/MultiDimensionalInvestigation`, but due to the complexity and uncertainty of how useful it would be in practice we did not include it in any of the considerations in this documents. Nevertheless we should keep it in mind to increase our options in the future.

6 Discussion

(discussion will be added after initial review rounds)

We give an incomplete list of "action items" resulting from this report. With a couple of exceptions, most of the items in this list are not single action items but will instead involve a long-running process and effort:

- Progressively update parameter estimates.
- Ensure that estimates for ODIN are included either here or in a similar report for data analysis.
- Obtain updated specifications for BIFROST.
- Develop better understanding of how we need to handle data from measurements with parameters scans. This can be sample rotation scans for single crystal diffraction and inelastic scattering, scans of the gauge volume for engineering diffraction, and sample environment parameter scans such as temperature scans. There is a variety of options to deal with this, such as handling everything independently, filtering during event loading, and filtering in memory. Since there is no single solution that works for all cases it is difficult to provide a generic estimate without going into the details of every workflow.
- Benchmark the influence of memory bandwidth on the Mantid performance.

A Related documentation

- Repository containing scripts to produce data in this document as well as the source for this document:
  `https://github.com/DMSC-Instrument-Data/data-reduction-hardware-requirements`
- Initial thoughts and plan for estimating hardware requirements:
  `https://github.com/DMSC-Instrument-Data/documents/blob/master/mantid/hardware-requirements/plan.md`
- Notes from meetings with instrument teams:
  `https://confluence.esss.lu.se/display/DAM/Data+Reduction`
- Benchmarks for live reduction (Lamar Moore):
  `https://github.com/DMSC-Instrument-Data/documents/blob/master/investigations/Live%20Reduction/LiveReductionInvestigation.md`
- Benchmarks for event filtering (Neil Vaytet):
  `https://github.com/DMSC-Instrument-Data/data-reduction-hardware-requirements/blob/master/benchmarks/nvaytet/summary.md`
- ODIN data rate report:
  `https://confluence.esss.lu.se/download/attachments/274116159/report_ODIN_data_rate_volume.pdf?api=v2`

B Motivation of the performance master equation

The rationale for Eq. (3) is as follows:

- For the vast majority of algorithms used in data reduction, all spectra are treated independently. Thus there is a linear term in N_{spec} or N_{bin}, but no higher order terms, and there is perfect scaling with N_{core}.
- Events in Mantid are stored with their respective spectrum. Strictly speaking, we should thus include a term

$$t^{\text{nonlinear}}_{\text{reduction}} = \sum_{i=1}^{N_{\text{spec}}} (N_{event,i} t_{event,linear} + N_{event,i} log N_{event,i} t^{\text{NlogN}}_{\text{event}} + ...), \tag{10}$$

  i.e., a different term for each spectrum, depending on the number of events in that spectrum, and non-linear term, such as for sorting events. However, events are usually spread over many spectra, so we can approximate

$$N_{event,i} \approx N_{\text{event}}/N_{\text{spec}}. \tag{11}$$

We obtain

$$t_{\text{reduction}}^{\text{nonlinear}} = \sum_{i=1}^{N_{\text{spec}}} (N_{\text{event}}^{i} t_{\text{event}}^{\text{linear}} + N_{\text{event}}^{i} \log N_{\text{event}}^{i} t_{\text{event}}^{\text{NlogN}} + ...) \tag{12}$$

$$\approx \sum_{i=1}^{N_{\text{spec}}} (\frac{N_{\text{event}}}{N_{\text{spec}}} t_{\text{event}}^{\text{linear}} + N_{\text{event}}/N_{\text{spec}} \log \frac{N_{\text{event}}}{N_{\text{spec}}} t_{\text{event}}^{\text{NlogN}} + ...) \tag{13}$$

$$= N_{\text{event}} t_{\text{event}}^{\text{linear}} + \frac{N_{\text{event}}}{N_{\text{spec}}} \sum_{i=1}^{N_{\text{spec}}} (\log \frac{N_{\text{event}}}{N_{\text{spec}}} t_{\text{event}}^{\text{NlogN}} + ...) \tag{14}$$

$$= N_{\text{event}} t_{\text{event}}^{\text{linear}} + N_{\text{event}} (\log \frac{N_{\text{event}}}{N_{\text{spec}}} t_{\text{event}}^{\text{NlogN}} + ...). \tag{15}$$

The term in parenthesis depends on $\log(N_{\text{event}}/N_{\text{spec}})$ and *not* $\log N_{\text{event}}$ (similar for higher order terms). These terms are thus typically small and it seems reasonable to absorb them into the linear term

$$\approx N_{\text{event}} * t_{\text{event}}. \tag{16}$$

If instead events are peaked in a subset of spectra we approximate $N_{event,i}$ (0 or $N_{\text{event}}/N_{\text{peak}}$, the approximation works in a similar way.

- Loading large event files is a significant contribution to the overall reduction time. While the exact scaling of the new parallel event loader is unknown (no adequate parallel file system available), there is definitely a linearly scaling contribution that scales well with the number of cores and is thus already described by the $N_{\text{event}} * t_{\text{event}}$ term. In addition to that, the other major factor is given by the upper limit of file system bandwidth. Basically, this is a limit to the number of events that can be loaded per second. This will also depend on whether or not compression is used in NeXus files. We model this limit in the equation with the term $N_{\text{event}}/B_{\text{max}}$. In case a parallel file system provides an bandwidth that is much higher on average than what was benchmarked for a local SSD, we may need to include a different term that captures limited scaling of the parallel loader.
- The time for reducing a spectrum will often depend linearly on the bin count. Many instrument can adjust their resolution, usually by sacrificing brightness. Thus the bin count will be fixed for a given run but can vary between runs within a instrument-specific range.

C Instrument parameters

C.1 BEER

- According to the original proposal, resolution settings range from 0.1% to 1%. The wavelength band is 1.7 Å but there are chopper settings to double it using pulse skipping. For a wavelength band centered at 2 Å this yields bins counts of 10×1.7 Å$/(0.01 \cdot 2$ Å$) = 850$ for the low resolution setting and 10×1.7 Å$/(0.001 \cdot 2$ Å$) = 8500$ for the highest resolution.

C.2 BIFROST

- According to the original proposal, resolution settings range from 1% to 4%.
- The wavelength band is 1.7 Å. In the most extreme case of 1% resolution for a central wavelength of 2 Å this yields approximately $10 \times 1.7\ \text{Å}/(0.01 \cdot 2.0\ \text{Å}) = 850$. At 4 Å with 4% resolution we obtain $10 \times 1.7\ \text{Å}/(0.02 \cdot 4.0\ \text{Å}) = 110$.

C.3 DREAM

- The (useful) range of the pulse shaping is 10 μs to 500 μs. We thus obtain $10 \times 71\ \text{ms}/10\ \mu\text{s} = 71000$ bins for the highest resolution and $10 \times 71\ \text{ms}/500\ \mu\text{s} = 1420$ for the lowest resolution.

C.4 ESTIA

From the original proposal:

- Wavelength range is 5 Å to 9.4 Å with an intrinsic resolution of 2% and 4%, respectively. In the simplest case with constant bin width this yields $10 \times (9.4 - 5.0)/(0.02 \cdot 5.0) = 450$ bins, where the factor of 10 is the aforementioned rule of thumb to obtain a required bin count from a resolution.

C.5 LoKI

- Resolution is dominated by the pulse length. We thus obtain an approximate bin count of $10 \times 71\ \text{ms}/3\ \text{ms} = 240$.

C.6 MAGIC

- The shortest pulse from the pulse-shaping chopper is 100 μs yielding $10 \times 71\ \text{ms}/100\ \mu\text{s} = 7100$ bins, where 71 ms is the time to the next pulse. While there is a high-flux option, my understanding is that this depends on collimation while the pulse-shaping is unchanged, i.e., we always have this pulse length.

D Helper scripts

Use `generate_tables.py` to generate a table output on the command line as well as input files for this LaTex document. High-level parameters are currently set in `performance_model.py` and `beamline.py`. Instrument-specific settings can be found in `generate_tables.py`.