Modelling the CPU load for the XRISM Resolve Instrument

Bachelorarbeit aus der Physik

Vorgelegt von
David Lochner
21. Februar 2024

Dr. Karl Remeis-Sternwarte Bamberg
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer: Prof. Dr. Jörn Wilms

Abstract

During the observation of bright X-ray sources with the XRISM Resolve instrument, the rate of events that can be processed by the CPU is limited. This affects the processing of the data and can result in unwanted event loss. To simulate this effect in the SIXTE end-to-end simulator, I design a model of the CPU, which tracks the unprocessed events. This model implements the fair scheduling of the pixels in queues based on a round-robin algorithm. Afterwards, I analyze the current output of SIXTE and the functionality of the algorithm by simulating sources at various fluxes. Using these results, I discuss methods to reduce the impact of the CPU limit on observations of bright sources and give an outlook on further improvements of the scheduling.

Contents

1. Introduction

The universe, viewed through human eyes, seems to be an overall cold and dark place. This is not the case in regards to the X-ray spectrum, where previously invisible structures and colors are revealed. Earth's atmosphere absorbs these X-rays at a high extent, which is why space missions are required. XRISM (X-ray Imaging and Spectroscopy Mission) is one of these spacecraft missions by the space agencies JAXA from Japan, NASA from the United States and the European ESA (XRISM Science Team, 2020).
"The primary purpose of the mission is to recover the science that was lost after the Astro-H/Hitomi mission failed in 2016, approximately one month after launch (XRISM Science Team, 2020)." The XRISM spacecraft, shown in Figure 1, launched on 2023 September 7 Japan Standard Time[1].
The two instruments onboard are the soft X-ray spectrometer (SXS) Resolve and the soft X-ray imager Xtend, whereby the focus in this thesis lies on Resolve.
Resolve enables non-dispersive X-ray spectroscopy and operates in the soft X-ray bandpass in the energy range of $\sim 0.3 - 12$ keV with an expected energy resolution of 7 eV (XRISM Science Team, 2020).

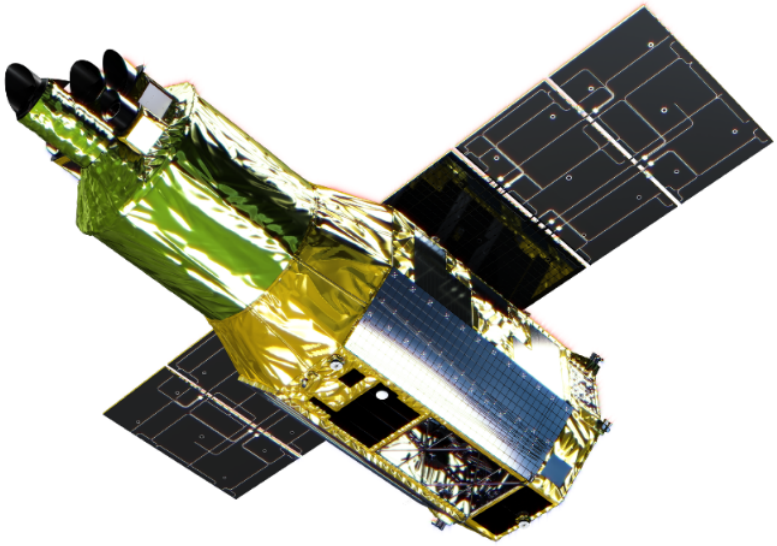

Figure 1: The XRISM satellite, where the two instruments (Resolve and Xtend) are located inside the lower-right area of the main corpus (XRISM Science Team, 2020).

The first data of XRISM published by NASA is shown in Figure 2. The supernova-remnant N132D in the Large Magellanic Cloud that has been created by an explosion of "a star roughly 15 times the Sun's mass" was observed[2].

[1] https://heasarc.gsfc.nasa.gov/docs/xrism/
[2] https://science.nasa.gov/missions/xrism/nasa-jaxa-xrism-mission-reveals-its-first-look-at-x-ray-cosmos/,

In the spectrum detected by Resolve, multiple elements can be identified through their emission lines. From these lines, physical properties like temperatures or densities, which provide more insight into the explosion process can be deduced. The first results promise an energy resolution of 5 eV representing an improvement over Hitomi. Further measurements are planned in 2024, which address a broad field of topics[2].

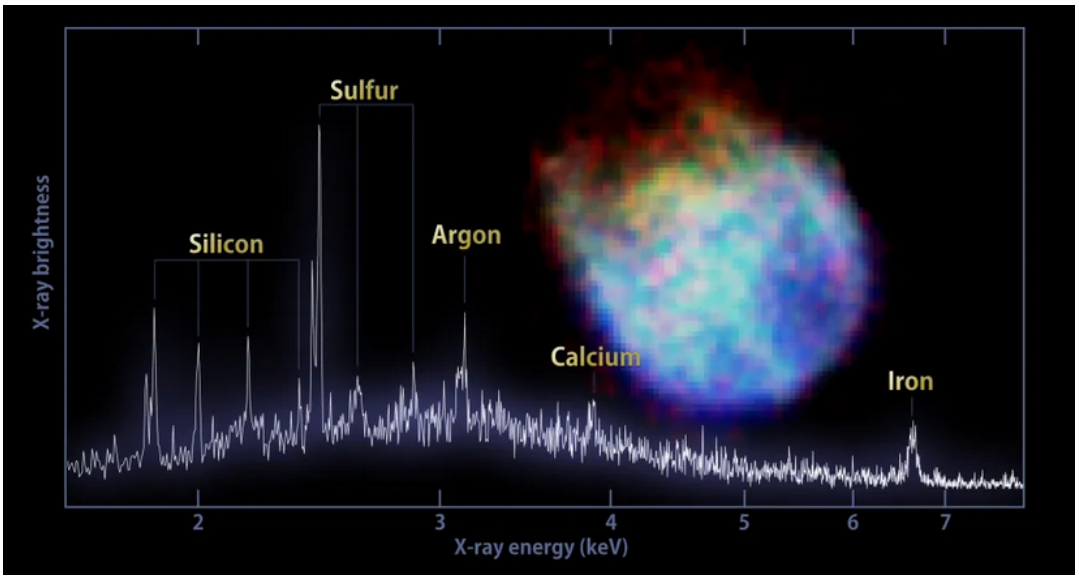

Figure 2: The spectrum for the supernova remnant N132D that was measured by Resolve. It shows peaks that are linked to specific elements. The image on the right was taken by Xtend (JAXA/NASA/XRISM Resolve and Xtend).

Potential topics in addition to the exploration of supernova remnants are black holes and galaxy clusters. Resolve enables studies of the surroundings of the event horizon and the black hole's growth through mass accretion. The vicinity of a black hole is shaped by its radiation field and outflowing winds. The study of these gas-containing winds from black holes can yield detailed insights into the mass outflow, the driving mechanism, and their impact on the accretion of black holes. Narrow emission and absorption lines can be detected due to Resolve's high resolution.
Furthermore, emission lines from hot gas in galaxy clusters can be measured and might reveal more details about the biggest objects in the universe. The turbulent gas can be heated by merging clusters, mass accretion or the influence of active galactic nuclei and flows characteristically. Therefore, the investigations made by Resolve might increase our understanding of the cluster's development (XRISM Science Team, 2022).
Brief and quickly fluctuating phenomena, for instance, the disk winds, can best be studied in bright sources with a sufficiently high count rate (Hodges-Kluck, 2023). One potential object is the Crab supernova remnant and nebula, which is a bright source with a X-ray flux of $2.4 \cdot 10^{-8}$ erg/s/cm^2 defined as 1 Crab[3].
An image of the Crab nebula is shown in Figure 3. The Crab nebula was created through

[3] `https://heasarc.gsfc.nasa.gov/docs/heasarc/headates/brightest.html`

the collapse of a massive star. The neutron star in its center rotates quickly and possesses a strong magnetic field. As a result, jets consisting of matter and anti-matter are powered in the pole directions and a wind propagates perpendicular to them[4].

Figure 3: A combined image of the Crab nebula, where the X-ray data from the Chandra mission is blue and white. The data from the Hubble space telescope in the optical range is purple and the data from the Spitzer mission in the infrared range is pink (X-ray: NASA/CXC/SAO; Optical: NASA/STScI; Infrared: NASA-JPL-Caltech).

Before these astrophysical observations with XRISM are executed, sufficient preparation and study of Resolve's performance is necessary. This is possible by a detailed simulation of an object and the instrument. One of the best software tools for this task is SIXTE by Dauser et al. (2019). The quality of the simulation depends greatly on the consideration of effects and disturbances on the instruments and their electronics.
One of the fundamental effects on the processing of events from bright sources, like the Crab nebula, is the load-limitation of the central processing unit (CPU). If this load limit is exceeded, a significant amount of unprocessed events is loss (Ishisaki et al., 2016; Mizumoto et al., 2022). This thesis is an attempt to understand the current output of SIXTE and to provide a first solution for the implementation of an algorithm into SIXTE, which tracks the unprocessed events. At the beginning, a deeper understanding of the detector and its functionality is necessary.

[4] `https://chandra.harvard.edu/photo/2018/crab/`

2. Resolve

The SXS, shown in Figure 4, is a HgTe-microcalorimeter and currently the only detector of that type operated in space (Hodges-Kluck, 2023).
A calorimeter consists of three fundamental elements. Firstly, an absorber takes in the incident photon and converts its energy into heat. Secondly, a thermometer measures the temperature variation in the absorber, which acts as the signal pulse. Finally, a weak thermal link to a heat sink is responsible for returning the absorber material to its initial state (McCammon, 2005). Due to the small temperature variations induced by the X-rays, in order to achieve a high resolution, the system stays at a low temperature of 50 mK and is cooled by liquid helium cryogen.
Each of the 36 pixels has a dimension of $30''$. The pixel array is of a 6×6-shape with a field of view of $2.9' \times 2.9'$ (XRISM Science Team, 2020).
One of the pixels is located outside of the array to record the gain drift of the detector through the illumination with a ^{55}Fe-source (Mizumoto et al., 2022).

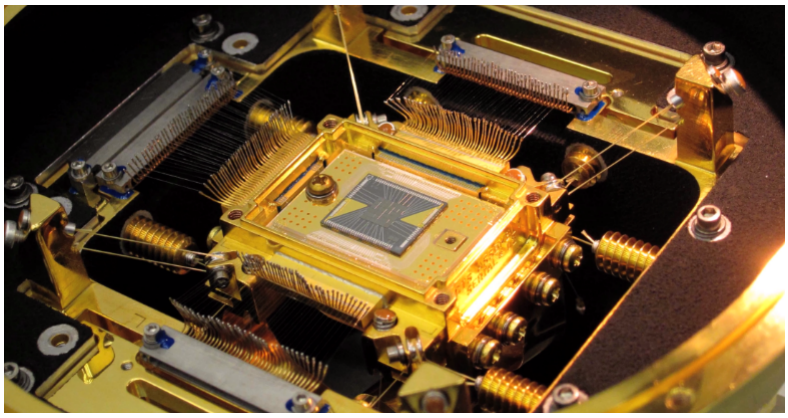

Figure 4: The SXS onboard of Hitomi photographed prior to launch. Resolve is alike (XRISM Science Team, 2020).

The incident photon induces a pulse in a pixel, whose amplitude ΔT is proportional to the photon's energy E_0 via

$$\Delta T = \frac{E_0}{C} \tag{1}$$

with the absorber's heat capacity C. An example pulse shape is displayed in Figure 5, where τ denotes the time for returning the absorber back to its initial state (McCammon, 2005).

The digital electronics for the SXS onboard is the pulse-shape-processor (PSP) with two units. Each unit possesses one field-programmable-gate-array (FPGA) and two CPUs, which is shown in Figure 6 (Tsujimoto et al., 2018).
An analog-digital converter (XBOX ADC) amplifies the signal received from the pixel and digitizes it. Afterwards, the FPGA triggers the digitized pulse of the event and sends it to the CPU for further processing.
The event dual buffer (EDB) stores the triggered events and distributes them to buffers,

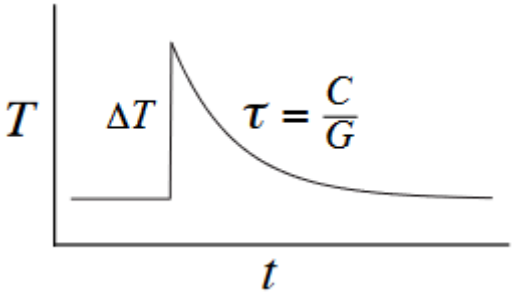

Figure 5: The shape of the pulse caused by the thermalization of the energy. The thermal link possesses the conductivity G (McCammon, 2005).

which work on a first-in-first-out basis (FIFO). There is one FIFO for each pixel, so no spatial information is lost. If it exceeds its capacity of 256 entries and needs to store another event, the associated FIFO is emptied and the new event fills the first spot (Hodges-Kluck, 2023).
Each CPU is responsible for the processing of nine FIFOs. The processing includes the inspection if it is a cosmic ray event using an anti-coincidence detector located below the pixel array, event grading, and optimal filtering (Mizumoto et al., 2022).
In optimal filtering, the CPU determines the photon's energy, which is proportional to its pulse height. The PSP "cross-correlates a normalized pulse template with each pulse", where the templates vary for different event types (Hodges-Kluck, 2023).

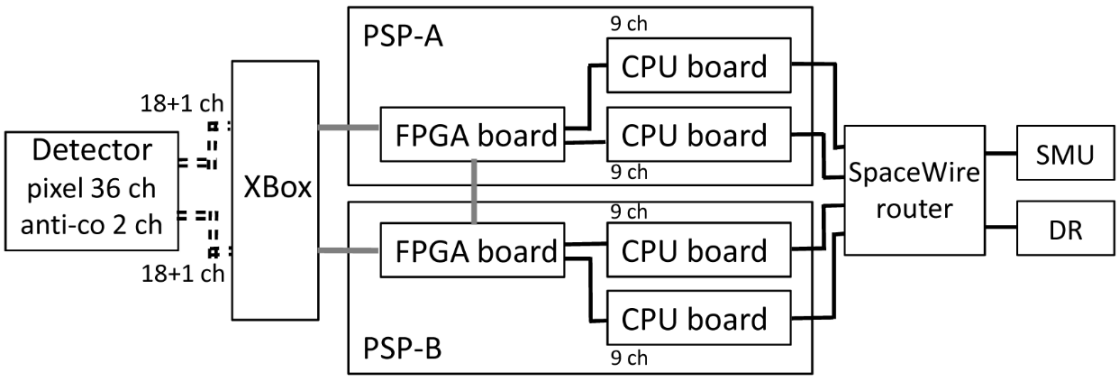

Figure 6: An overview of the main parts of the processing electronics. The continuous lines designate digital signals and the double lines designate analog signals. After the CPU, the data are transferred via SpaceWire to the satellite-management-unit (SMU) and the data-receiver (DR) (Ishisaki et al., 2016).

3. Event grading and CPU load

Event grading distinguishes the different event types. Each incoming event receives a grade, which contains information about its temporal position to other triggered pulses in this pixel. Figure 7 illustrates this process, where the minimum temporal separations for the different grades and where they are set are visible.

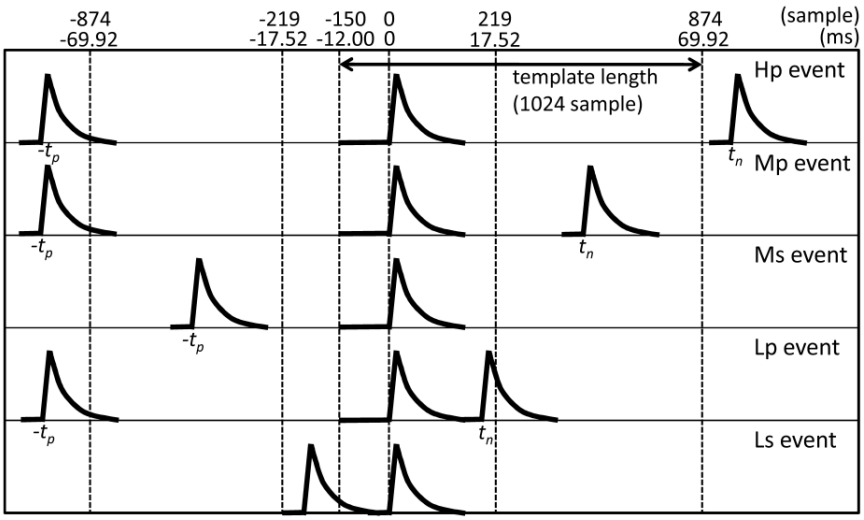

Figure 7: An event's grade depends on the temporal position of the pulse at $t_0 = 0$ to the previous pulse at $-t_p$ and to the next pulse at t_n. The decisive values for t_p and t_n are 17.52 ms and 69.92 ms (XRISM Science Team, 2022).

The differentiation between secondary and primary events is important to classify the position in an event pair. Thus, for high-resolution primary (Hp)-events, a high resolution-template is used to measure the amplitude, which has a record length of 81.92 ms or 1024 samples. For mid-resolution primary and secondary (Mp and Ms)-events a shorter template of 20.48 ms duration is used and for the low-resolution primary and secondary (Lp and Ls)-events merely the maximum height for the pulse is measured without any optimal filtering (Ishisaki et al., 2016).
The energy resolution of $\Delta E \leq 7$ eV is only ensured for Hp- and Mp-events and makes them the preferred event grades (XRISM Science Team, 2022).
The event processing consumes a minimal share of the CPU load for a single event depending on its grade, though it increases enormously for higher incoming rates. The maximal CPU load is finite and a limitation can be reached, which affects the measurement significantly starting at a flux of around 1 Crab according to Hodges-Kluck (2023). This limit at 100 % CPU load is reached for a countrate of ~ 200 counts/s for all four CPUs combined. Exceeding this limit, the FIFOs fill up faster than the CPU can process the events, which leads to FIFO dumping and a massive event loss (Hodges-Kluck, 2023). For the correctness of the measurement, it is crucial to know the events that are affected. Therefore, the processed and unprocessed events are tracked using an algorithm, which is the topic of the following section.

4. SIXTE and CPU model

SIXTE provides the input for the following algorithm, which means that a basic understanding of its architecture and functionality is beneficial.

4.1. Introduction to SIXTE

SIXTE (Simulation of X-ray Telescopes) by Dauser et al. (2019) is based on a Monte Carlo simulation with single photons and can be executed easily with Dauser et al. (2023). It takes a SIMPUT file as input, in which the observed source is specified. In the next step, SIXTE applies this input and details regarding the instrument's pointing for the photon production. The generated photons are directed through the specified optics of the spacecraft onto the detector so that an impact list of their properties, such as arrival time, incident pixel, and energy is created. At the end, SIXTE creates the final event list, where the detector model and its form of event processing are taken into account. Specific information about the vignetting, the point spread function (PSF), the ancillary response file (ARF), and the redistribution matrix file (RMF) is necessary to run SIXTE. These data are, for instance, from measurements or from simulations. An overview of SIXTE is provided in Figure 8, where the few specific parameters SIXTE requires at each step, are visible (Dauser et al., 2019). The algorithm for the CPU limit effect will be implemented into the detector model so that the unprocessed and processed events are separate in the final event list. With this knowledge, the next step is the development of the algorithm, which needs specific data on the events from SIXTE.

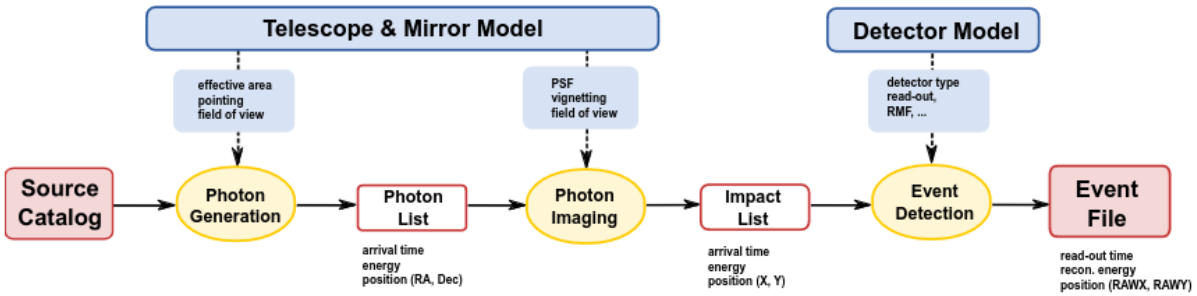

Figure 8: An overview of the functionality of SIXTE, where the three yellow symbols are the fundamental steps. The SIMPUT file defines the source catalog and the final event file is used for further operations (Dauser et al., 2019).

4.2. Modelling and algorithm

The CPU load for Resolve onboard of XRISM is simulated with an algorithm where the tasks of the CPUs to process events are distributed using a round-robin scheduling. The idea refers to Hodges-Kluck (2023). The original code is in Appendix A. Each CPU possesses one queue which is initialized in a strict rising order and contains FIFO 0 to

Table 1: The parameters are calculated as the mean values over all CPUs of Table 3 from Mizumoto et al. (2022).

a_{Hp}	a_{Mp}	a_{Ms}	a_{Lp}	a_{Ls}	c
0.0196	0.018475	0.016875	0.018525	0.015475	0.085875

8. This implies that no FIFO is prioritized. The rotation of this queue schedules the event processing of each FIFO because a single event is processed at a time. The queues and the FIFOs are implemented as double-ended queues, which provide insertion and deletion actions at the front and the back.
The important information about each event like the event grading, the arrival time, and the pixel of detection is computed by SIXTE. Using this information the time consumption of the CPU for the processing of a single event, is

$$(\text{CPU rate}) = \sum_{\text{pixel}} a_k \cdot (\text{pixel count rate of the event type k}) + c \tag{2}$$

(Mizumoto et al., 2022). The count rate denotes the number of triggered events in the pixel per second and grade. The parameters a_k are the associated values of the five grades and c is the baseload of each CPU, which considers other tasks that occupy the four CPUs. The parameters are the mean values over all CPUs of Table 3 from Mizumoto et al. (2022) and are shown in Table 1.

In the algorithm, the CPU load per event is understood as the processing time t_p of an event. To illustrate this, if an incident event is marked with a Hp-grade by SIXTE, t_p is

$$t_p = \frac{a_{Hp} \cdot 1}{(1 - c)} \approx 21.441 \text{ ms} \tag{3}$$

Furthermore, for a Mp-event $t_p \approx 20.211$ ms, for a Ms $t_p \approx 18.460$ ms, for a Lp $t_p \approx 20.265$ ms and for a Ls $t_p \approx 16.929$ ms. This confirms that the Hp-events consume the most time of the CPU for their processing and the Ls-events the least. This formula is part of the algorithm, of which an overview is shown in Figure 10. The symbol shapes used in the overviews are explained in the captions of Figure 9.
When an event enters the detector, the algorithm wakes up and needs initial information on the layout of the detector to find the associated CPU and queue. It processes events that are stored in the FIFOs until the arrival time t_0 of the incident event. The queue of each CPU schedules the FIFOs and so the order of processing. The queue processing, as indicated in Figure 11, is based on a loop, which processes stored events as long as a filled FIFO exists. Otherwise, the current time t_{CPU} will be set to t_0. If a filled FIFO exists, its first stored event is processed if there is sufficient time until t_0 and t_{CPU} is updated with the event processing time t_p. Otherwise, the processing will stop. After the processing of an event or the discovery of an empty FIFO, the queue rotates, which means that the FIFO in focus is added to the back of the line.
After the queue processing, the incident event is added to the associated FIFO as long as

it has storage left. Otherwise, the stored events are added to the not-processed events, and the FIFO gets dumped. In the last step, the main loop continues with the next event at t_1. After the last event of the simulation, no further events will arrive, so the FIFOs can not overflow and all stored events are added to the processed events. In case of processed or unprocessed events, they are added to the final output lists, which are referenced as arrows to the shapes according to Figure 9.
In order to gain a deeper understanding of the algorithm and its functionality beyond the overviews, the following sections contain various analyses of the algorithm with data simulated by SIXTE.

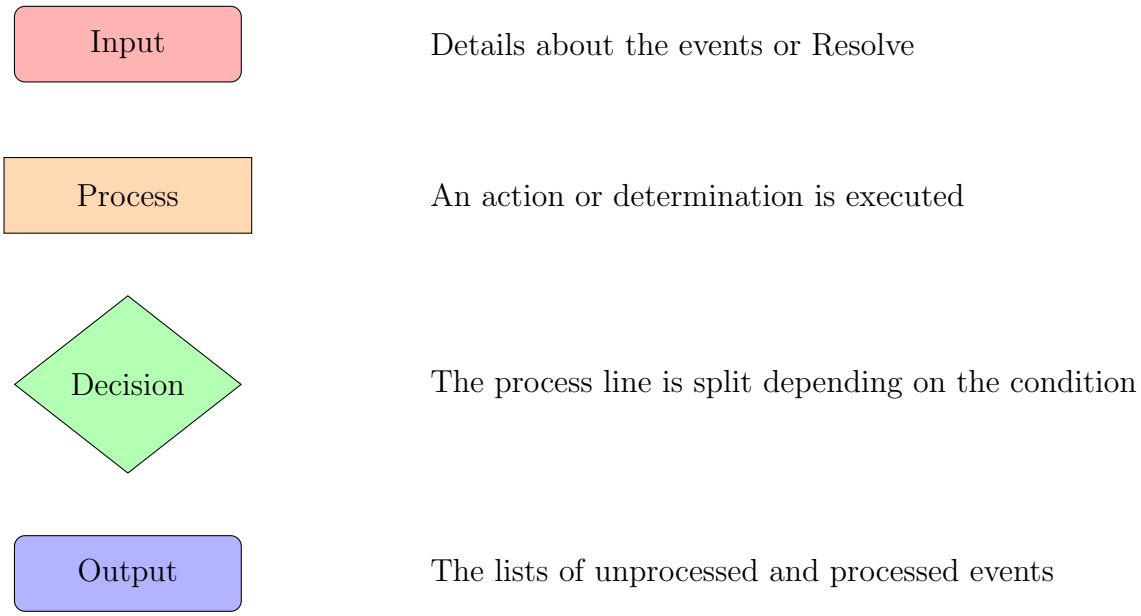

Figure 9: The symbols used in Figure 10 and Figure 11 on the left and their explanations next to them.

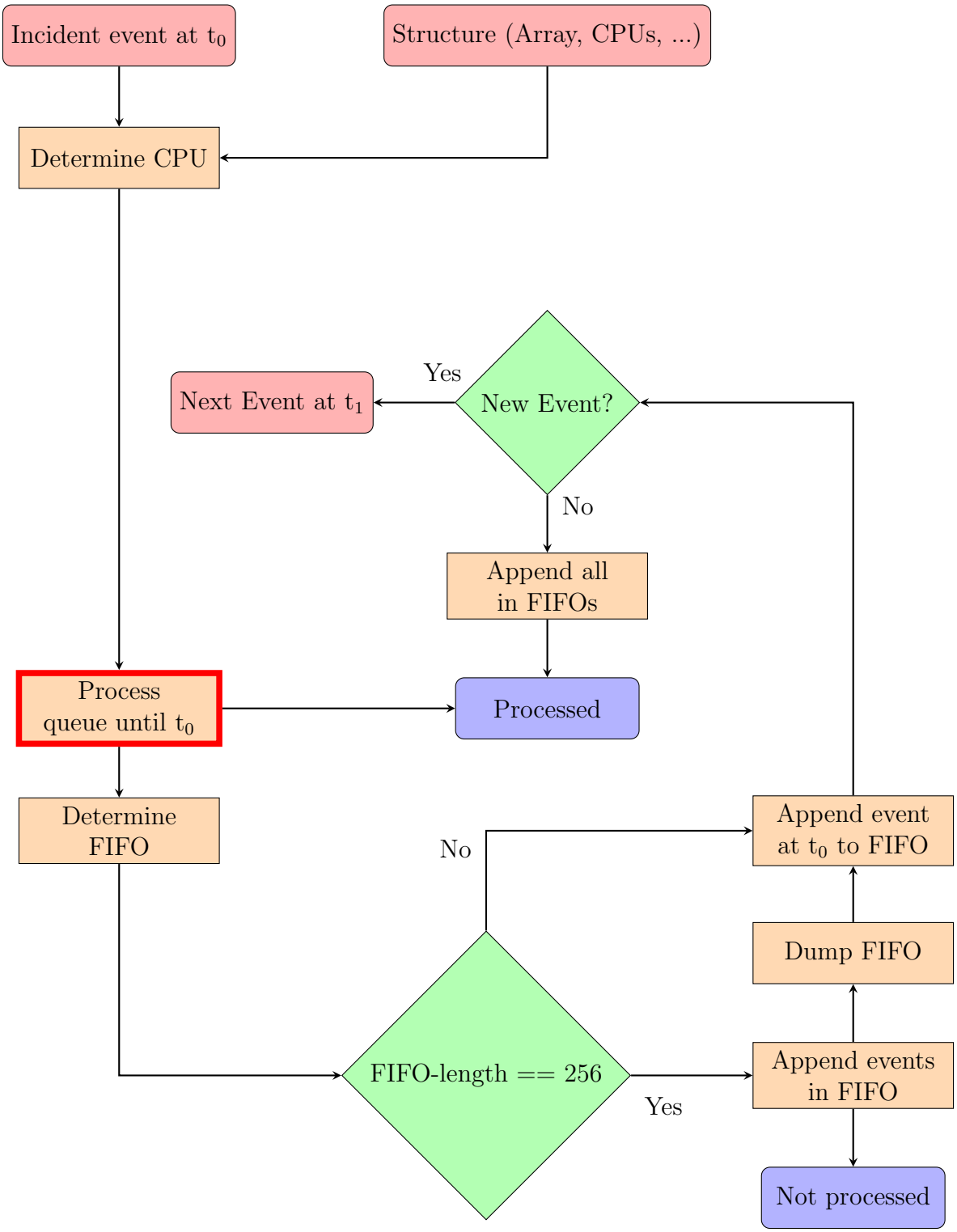

Figure 10: Schematic overview of the algorithm that was used to simulate the CPU load. The underlying scheduling was done by a round-robin algorithm. The highlighted step is shown more detailed in Figure 11.

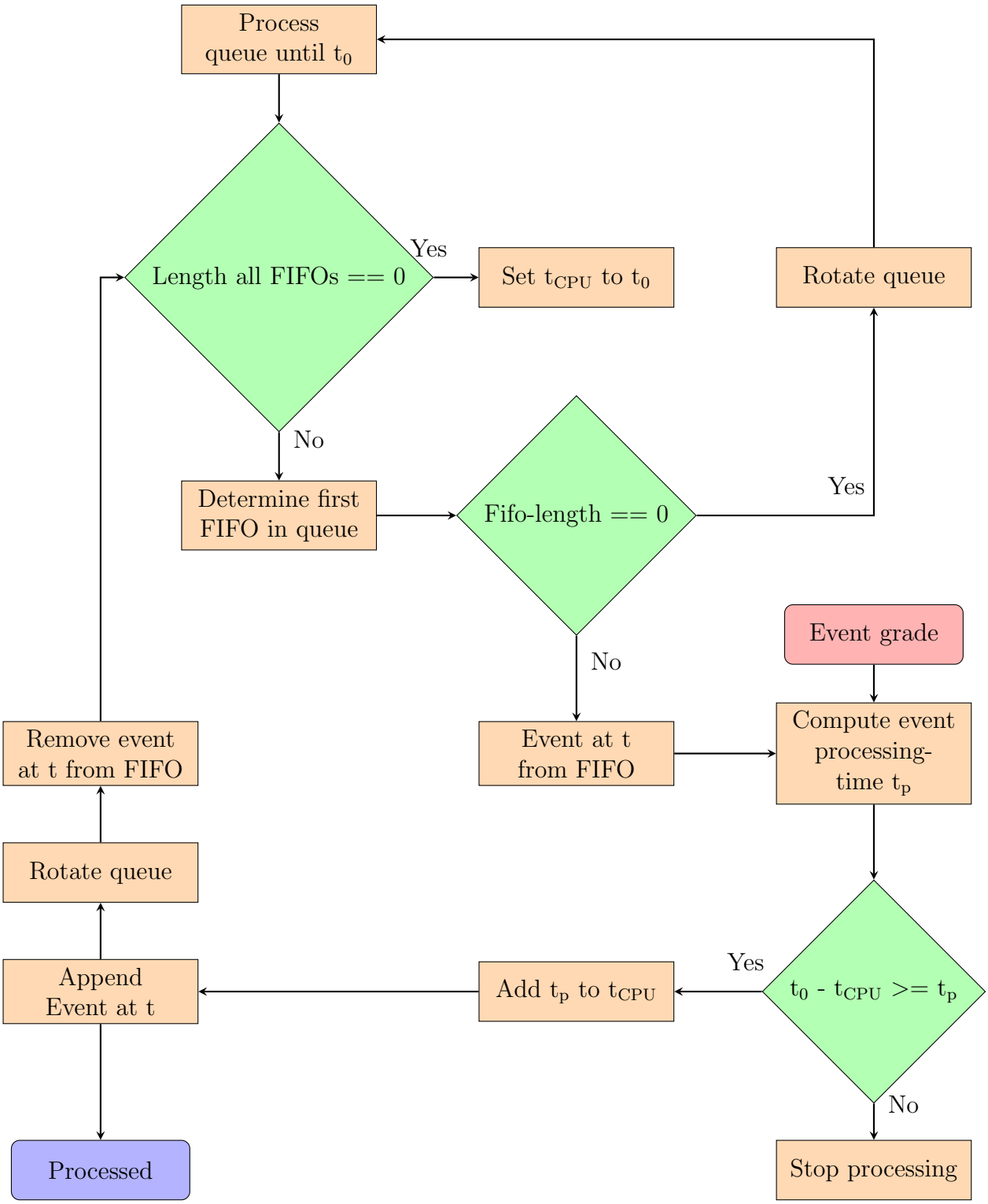

Figure 11: Schematic overview of the step, where the queue is processed until the time of the incoming event t_0. t_{CPU} tracks the current time of the CPU.

5. Analysis and testing

It is necessary to test the model and algorithm sufficiently before implementing it into SIXTE itself. Therefore, I analyze the current output of SIXTE in order to investigate the model for the CPU load and the algorithm in detail. Since the simulations are based on pre-launch assumptions, the Gate-Valve located in front of the detector is in its open state for the following simulations. This is a cover in front of the detector, which filters mainly low-energy photons in its closed state[2]. Since the behavior shown below depends only on the event rate, simulations with open or closed Gate-Valve would produce similar results, with only the conversion between flux and incoming rate changing.

5.1. Branching ratios of the event grades

The final event list of SIXTE contains the grade of each event. The branching ratios of the grades in the final event file depend on the source's flux. Based on SIXTE simulations of Resolve, the branching ratios of a point source for different fluxes are shown in Figure 12. The global behavior is that the resolution of the events decreases with higher fluxes as the time between the events in the pixels shrinks. As a result, the fraction of Ls-events rises, whereas the fraction of Hp- and Mp-events decreases with higher fluxes. For bright sources with a flux of more than 1 Crab, mainly low-resolution events enter the FIFOs. The branching ratios of Mp- and Lp-events are relatively low because they require a temporal separation of more than 69.92 ms to the previous pulse followed by a temporal closer subsequent pulse, which is generally improbable.

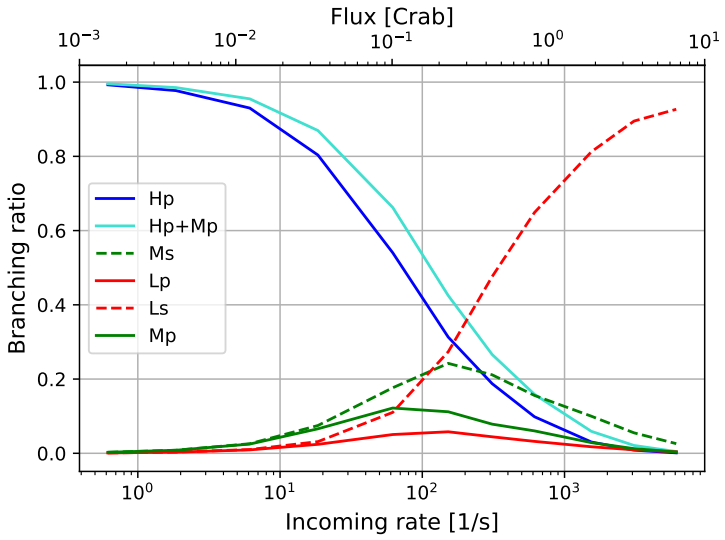

Figure 12: The branching ratios of the event grades at given fluxes of 1, 3, 10, 30, 100, 250, 500, 1000, 2500, 5000 and 10000 mCrab. The exposure time was scaled via an initial value of $1 \cdot 10^5$ s divided by the flux in mCrab.

5.2. CPU load per grade

To test Equation 2 separately from the algorithm, it is initially important to consider the CPU load in percentage per grade. Therefore, I apply Equation 2 with the parameters of Table 1 on the data of Figure 12. The result is seen in Figure 13. The exposure time was scaled via an initial value of $1 \cdot 10^5$ s divided by the flux in mCrab. In this plot the Hp- and Mp-line diverge with increasing incoming rates from the 'Total'-line and the Ls-line converges to the 'Total' line at high fluxes due to the branching ratios. The 'Total' line passes the limit for all four CPUs below a flux of 1 Crab at an incoming rate of $\sim$ 200 counts/s, which means that event loss is likely. In general, although the CPU load for all four CPUs is lower than 400 %, a single CPU with a high incoming rate could exceed its own limit of $\sim$ 50 counts/s and therefore lose events.

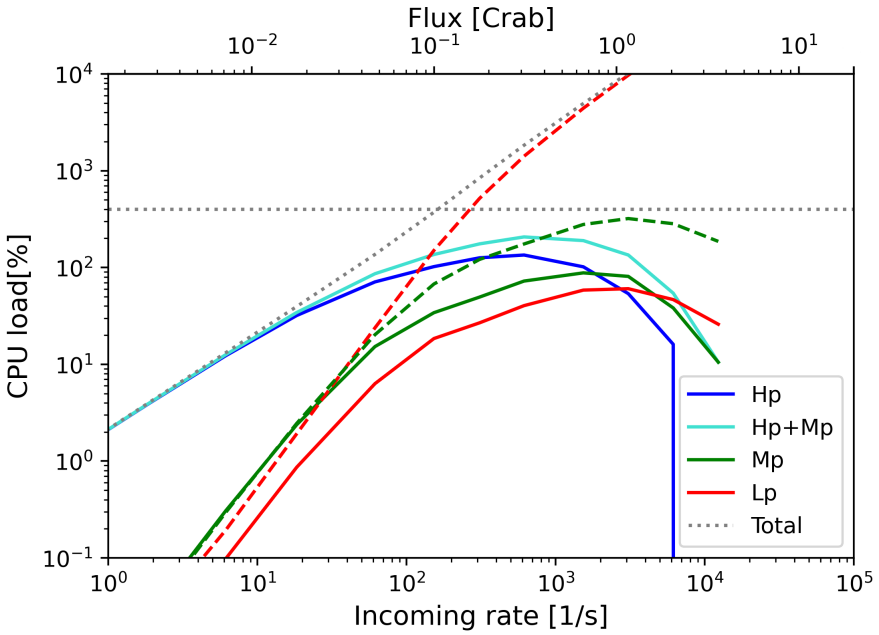

Figure 13: The CPU load as a function of the incoming rate for a point source. The limit of 400 % for all four CPUs is marked by the horizontal line. The dashed colored lines represent the associated secondary grades.

5.3. Application of the algorithm

As the next step, the output of SIXTE is given to the algorithm, which then returns a list of processed and unprocessed events. The algorithm's output varies for extended and point sources, because of the different distributions of events onto the pixel array. Analyses of all grades for different fluxes, as evident in Figure 14 and Figure 15, show that the loss of events starts at incoming rates of $\sim$ 200 counts/s for the two source types. The circular area of the extended source covers the entire pixel array. This distribution causes a lower incoming rate per pixel compared to the point source for the same flux due to the hits beyond the pixel array. With increasing flux more and more low-resolution events enter the detector, whereas the number of event grades with higher processing time decreases. This leads to a rise in the absolute number of processed events per second, whereas the fraction of processed events decreases. The two plane saddle points visible in both figures shortly after the beginning of event loss are affected by the transition from the predominantly high-resolution to the low-resolution events. Here, the divergence of the processed rate from the input rate is the highest.

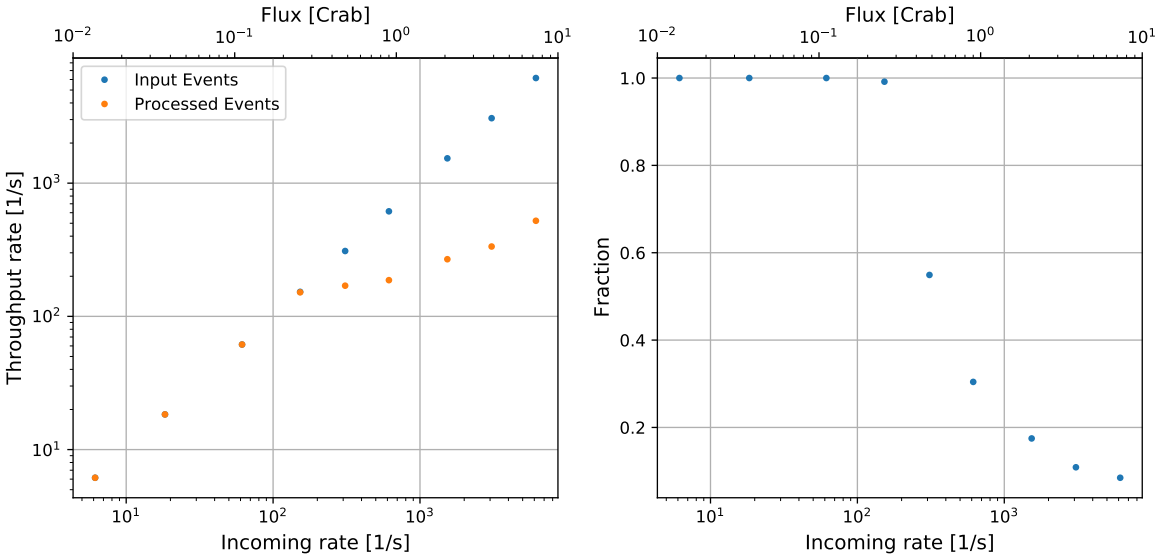

Figure 14: Simulation output showing the rate (left) and fraction (right) of processed events as a function of the incoming event rate for a point source with the same scaling of the exposure time as previously. In the left plots below a flux of 1 Crab, the dots for the input events are covered by the dots for the processed events due to the marginal loss of events.

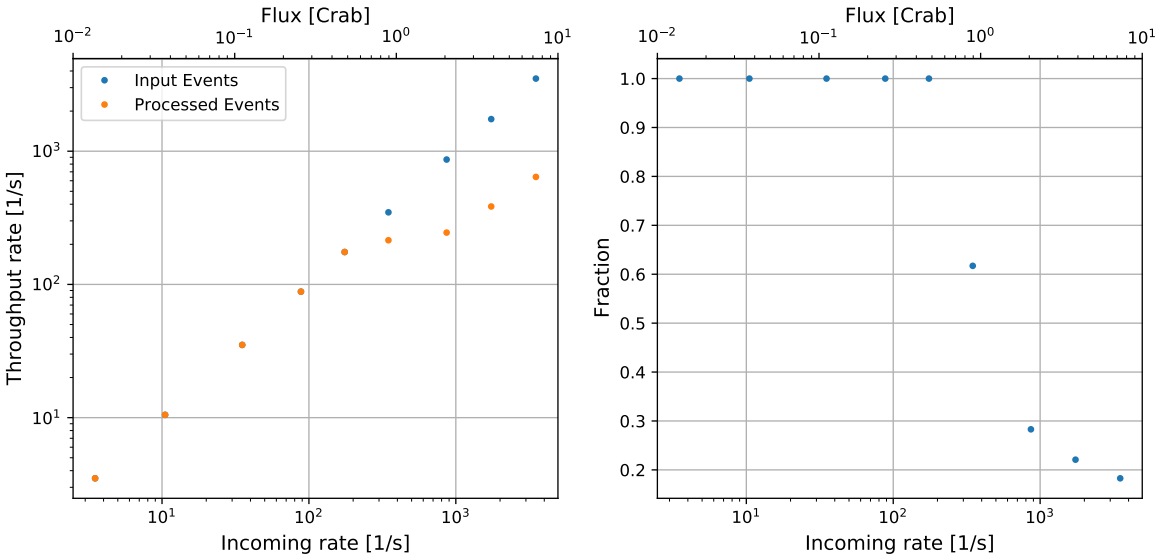

Figure 15: The same output as in Figure 14 for an extended source.

It is important to gain deeper insight into the event loss taking the event grades into account. As illustrated in Figure 16, the branching ratios result in a divergence of the Hp- and Mp-line from the total event line and a convergence of the Ms-, Lp- and Ls-line to the total event line with higher fluxes. The algorithm conserves events of all grades for low incoming rates due to the fair scheduling of the queue and the low CPU load. The throughput rate of Hp- and Mp-events for the extended source is higher for medium fluxes due to the equal distribution of events on the entire pixel array. On the other hand, the throughput rate of Ms-, Lp-, and Ls-events is initially higher for the point source in Figure 16. This behavior is caused by the concentration of the incident events onto a few pixels in the center of the pixel array, where the time intervals between the triggered events shrink early.
As the event loss begins at an incoming rate of ~ 200 counts/s, the extended source loses more processed Hp- and Mp-events caused by the uniform distribution of the different event grades among all pixels. With higher fluxes, the extended source produces fewer Hp- and Mp-events than the point source. The lower incoming rate to the edges of the pixel array for the point source guarantees more time between the triggered pulses in these edge pixels. As a result of the fair scheduling, high-resolution events are more likely for the point source and Ms-, Lp-, and Ls-events are less likely to be processed for the point than the extended source. Thus, the point source has a better resolution at high fluxes. Moreover, the consideration of focusing the point source on one CPU quadrant for higher resolution in the remaining quadrants at high incoming rates is plausible.

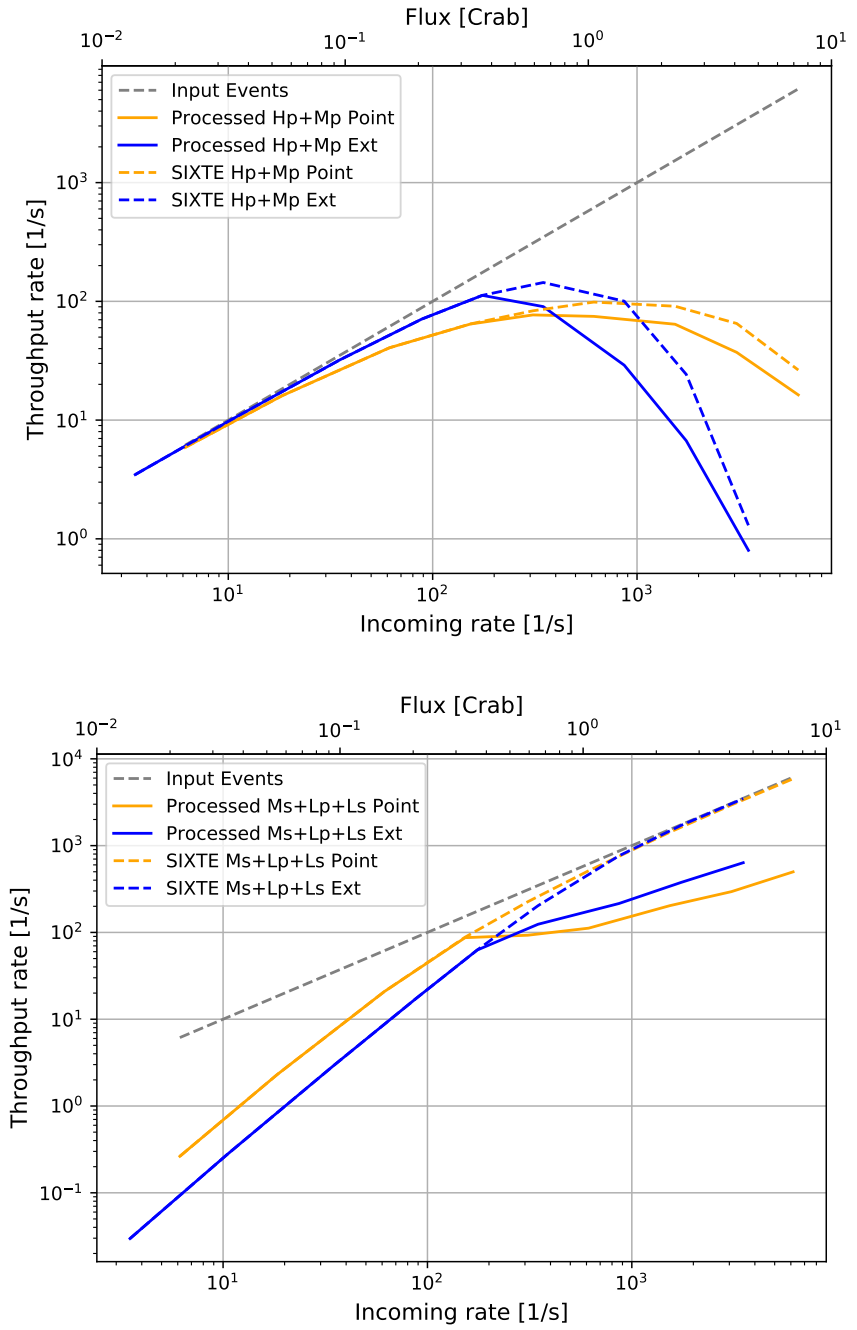

Figure 16: The throughput rate of Hp- and Mp-events (above) and of Ms-,Lp-, and Ls-events (below) as functions of the incoming rate and flux for a point and an extended source. The data of SIXTE are dashed lines and the outputs after the algorithm continuous lines.

It is useful to inspect the conservation of the spatial information by the detector and the algorithm to understand, which pixels lose events. I simulate a point source with

a flux of 1 Crab, so event loss is present. The source points to the center of the pixel array and the exposure time is 100 s. The result is visible in Figure 17. The pixels with the highest incoming rates lose the most events. Moreover, pixels with relatively low incoming rates provide the highest amount of processed events. In general, pixels with low incoming rates located at the edges of the pixel array do not lose events, although they are in CPU quadrants, where pixels lose many events. This is achieved by the fair scheduling of the queue with strict order, so these pixels with low incoming rates, which tend to produce high-resolution events, are not neglected. Moreover, fair scheduling achieves similar rates of processed events in pixels, although their incoming rates deviate enormously.

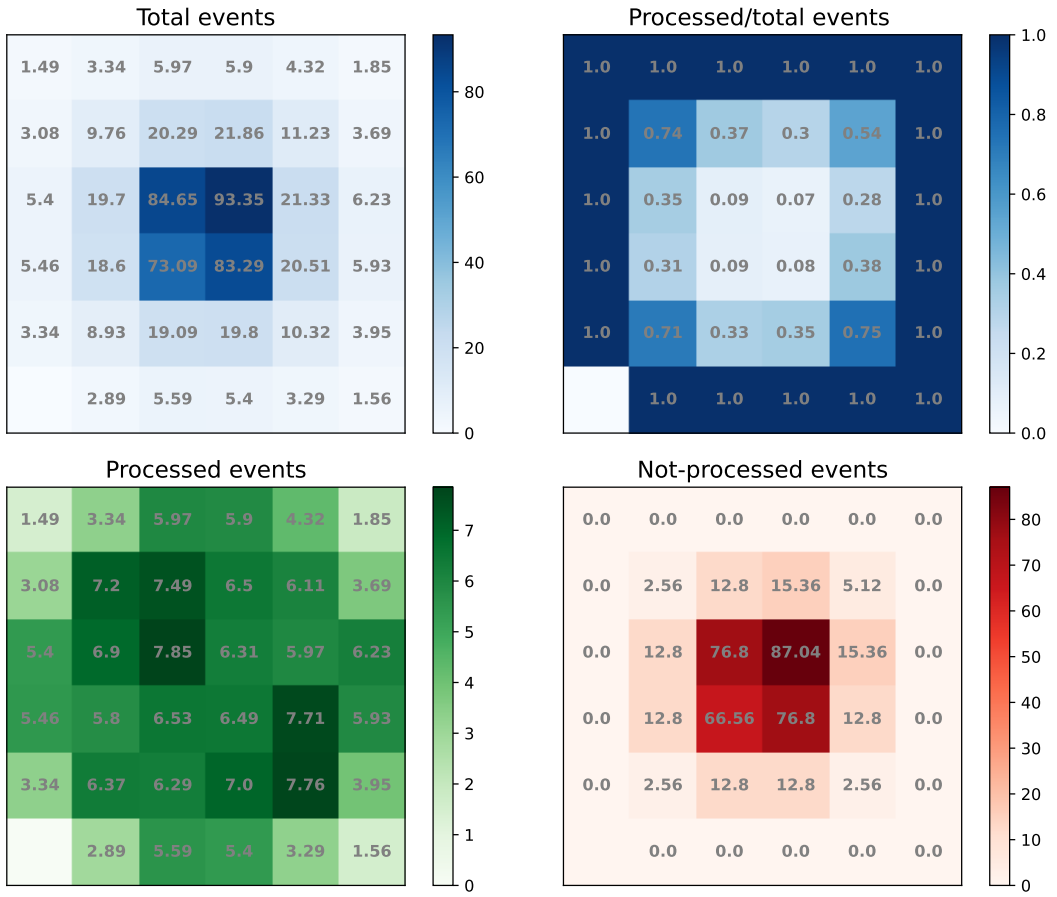

Figure 17: The pixel arrays for the ratio of processed to incoming events, processed events and not-processed events per second for a point source with a flux of 1 Crab. The pixel in the lower left corner is the mentioned calibration pixel that is located outside of the array. The color and the value in the pixel illustrate the event rate per second.

Other interesting information conserved by the algorithm are the arrival times of the incoming photons. The following is for the same source as in the spatial analysis. In

Figure 18, patterns are visible, which show periodic behavior. After high event loss, because of the FIFO-dumping, the number of processed events rises again. The bars of the unprocessed events complement the gaps between the bars of the processed events. At the beginning, the fraction of processed events is high because of the initial empty state of the FIFOs. This is not the case for the pixel with the highest incoming rate. The total incoming rate in its CPU quadrant is ~ 170 counts/s, so it exceeds the limit of ~ 50 counts/s for each CPU significantly and the FIFO fills up quickly resulting in an early event loss. At the end of the simulation, the fraction of processed events is the highest, because after the last event is detected, all events remaining in the FIFOs are processed events, since the FIFOs will not overflow anymore. It is clearly visible that the amount of unprocessed events over the exposure time is inefficient for bright sources. Thus, it is important to study methods for reaching a higher efficiency.

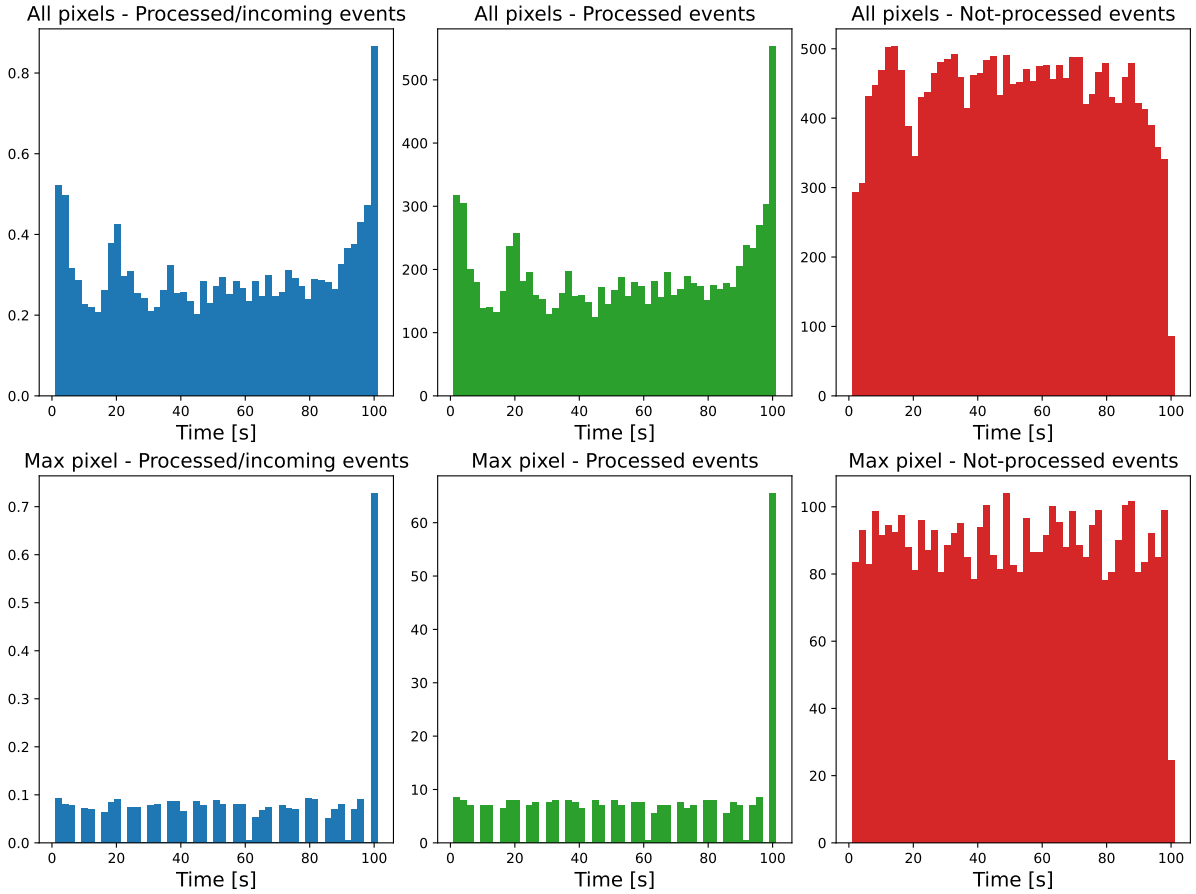

Figure 18: The ratios of processed to incoming events, processed events and not-processed events are divided into 50 time bins spanned over the exposure time. This is again for the point source with a flux of 1 Crab. The plots on the top are for all 36 pixels and the three on the bottom are for the pixel with the highest incoming rate, as observed in Figure 17.

5.4. Filter and offset pointing

Potential methods to decrease the CPU load for bright sources, in order to gain more Hp- and Mp-events, are the reduction of incoming events by filters or offset pointing. The filter wheel of XRISM is equipped with a beryllium, a neutral-density, and a polyimide filter (XRISM Science Team, 2022). In the following analyses, the Gate-Valve is again open, so that the effects of the filters are in focus.
The energy-dependent throughput of the beryllium and the neutral-density filters are shown in Figure 19. The effective area and so the number of detected photons decreases significantly at low energies of < 5 keV. The beryllium filter aims to conserve photons of higher energies, whereas the neutral density filter aims to balance and lower the effective area for the entire energy spectrum.

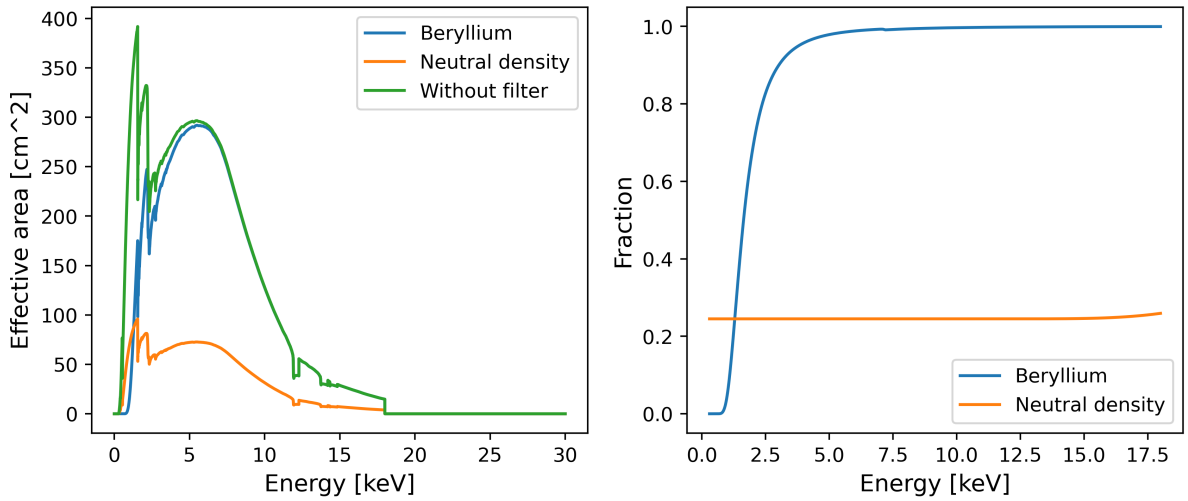

Figure 19: The energy dependence of the effective area for the beryllium filter, the neutral-density filter, and without filter on the left and the fractions of the filters from the clear state are in the right plot.

The other reduction method is to perform an offset pointing, as shown in Figure 20. This picture was created with the same point source as in Section 5.2 but with a 75″ offset in right ascension and declination, which puts the source in the outer corner of the detector array. The quadrants adjacent to the one on the top left are able to process all incident events. It is remarkable that the pixel with the sufficiently low incoming rate in the top left CPU quadrant does not lose events, despite the very large incoming rates in the other pixels. This behavior is due to the fair scheduling of the queue.

The effects of filters and the offset pointing on the throughput of high-resolution events are shown in Figure 21. For high fluxes of more than 3 Crab, the lower input photon rates due to the filters and the offset pointing actually lead to a higher Hp- and Mp-rate than for the regular pointing when both the branching ratios and CPU limits are taken into account. The rate for the neutral-density filter increases continuously to the highest value due to the strong filtering for all events, which is counterproductive at low fluxes. Moreover, the rate for the offset pointing increases continuously caused by the rise in

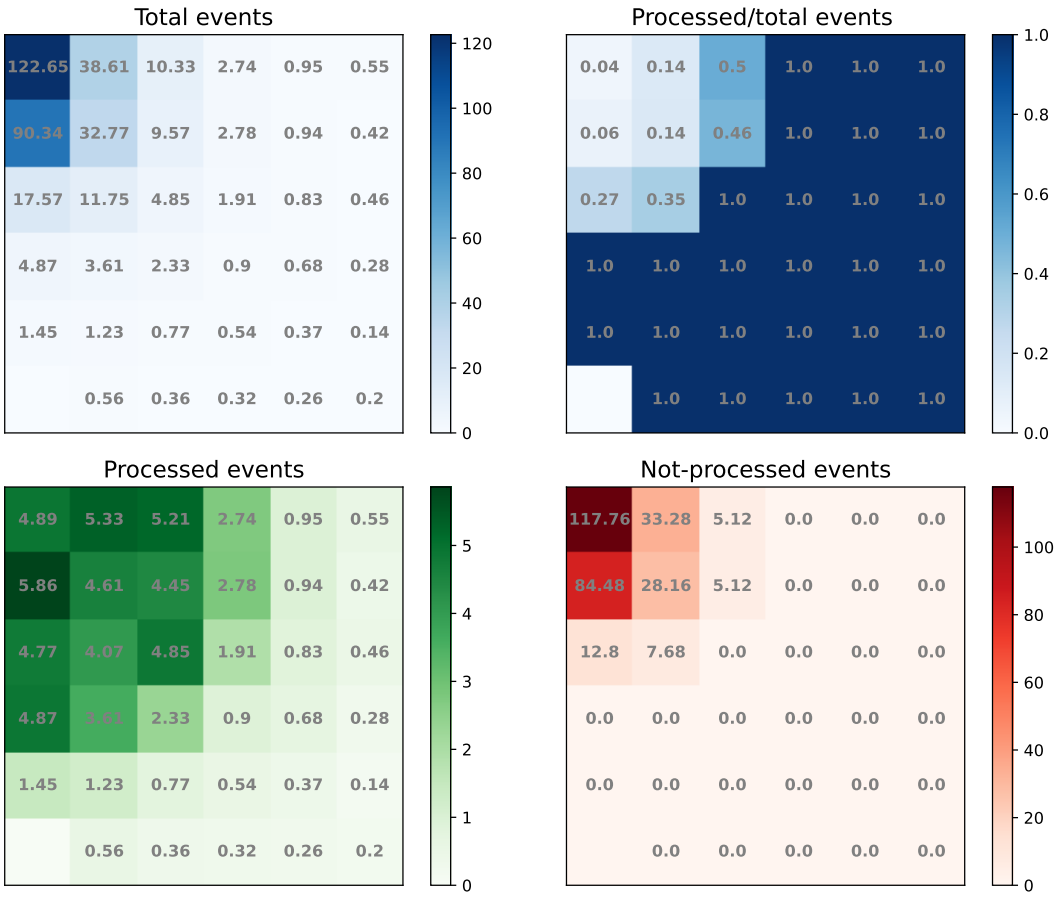

Figure 20: The same representation as in Figure 17 for the offset pointing of the point source.

events in the CPU quadrants that are adjacent to the CPU quadrant in focus. The high number of hits beyond the pixel array suppresses this at lower fluxes. SIXTE's output for the beryllium filter deviates slightly from the original state for low and high fluxes. The beryllium filter absorbs exclusively low energies, but nevertheless, fewer events enter the pixel array, so more Hp- and Mp-events are processed for high fluxes.
Taking the overall picture into account, the original state is the best choice to gain the most high-resolution events below a flux of $\sim$ 1.5 Crab and the neutral-density filter for higher fluxes. In general, the best choice depends strongly on the source. As an example, the impact of the beryllium filter would be much higher for a source that is very bright below 2 keV. Moreover, a desired spectrum can choose the filter, for instance, if the energies of interest are above 5 keV, the beryllium filter is more effective than the neutral-density filter.

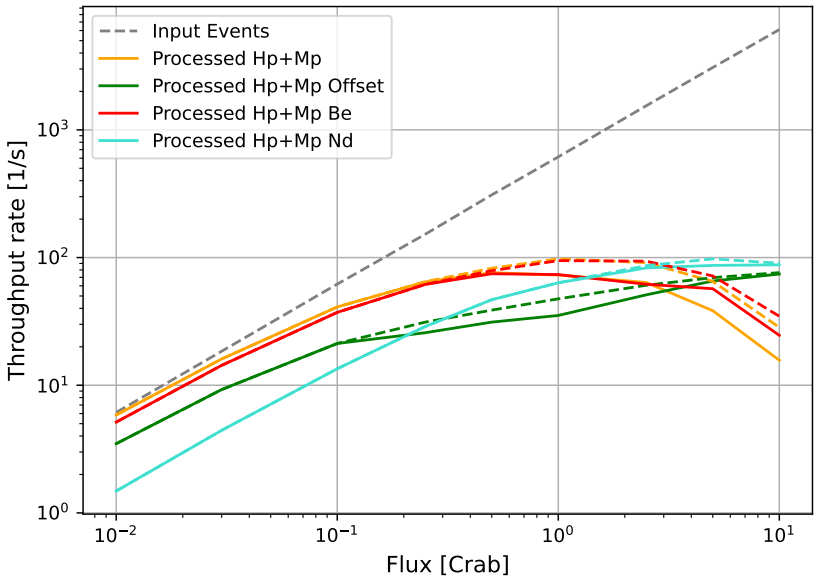

Figure 21: The Hp- and Mp-rates as functions of the flux for the original state, the two filters, and the offset pointing. The outputs of SIXTE are the dashed and the processed rates of the algorithm the continuous lines. The input event rate is for the data of the original state and the filter plots have no offset pointing.

6. Conclusion and outlook

In this thesis, I studied the event grades of the XRISM Resolve instrument and how the CPU limit affects their processing for bright sources. After designing a CPU model and a round-robin algorithm to track the affected events, I analyzed the functionality for real sources simulated by SIXTE.
The results and analyses show that the round robin algorithm is indeed a solution for SIXTE to schedule the processing of FIFOs for XRISM's Resolve fairly. The algorithm tracks the event loss during the observation of an object in the X-ray spectrum, which reaches the limit of the CPU load. The analyses of data simulated by SIXTE show that the effect is present and beginning at a flux of less than 1 Crab or an incoming rate of $\sim$ 200 counts/s, the algorithm is able to identify events that are not processed and returns them to the user. The branching ratios of the event grades are crucial to have a more accurate model. The study of an extended and a point source reveal that the amount and resolution of the processed events depend strongly on the coverage of the pixel array and the flux. Further testing shows that the algorithm conserves spatial and temporal information. As a result, it balances the processing among all pixels and damps pixels with very high incoming rates that tend to produce low-resolution events. The methods to reduce the CPU load for higher fluxes are filters and offset pointing, which increase the rate of Hp- and Mp-events for bright sources with a flux of more than 3 Crab. The impact of these reduction methods varies for different source types, their flux, and the desired spectrum.
Further steps are the implementation of the algorithm into SIXTE and the consideration of neglected factors like the difference in the baseload of each CPU due to the varying contact with the XBOX and the anti-coincidence detector and also the baseline (Mizumoto et al., 2022). These factors are neglected in this thesis due to the marginal difference and the initial aim to suggest a first-version solution for SIXTE. Furthermore, according to information from the XRISM team at GSFC, the concept of the locked queue needs more focus on the threads that can run simultaneously and the influence of secondary events, which could alter the order of processing.

7. Acknowledgements

I would like to dedicate these lines to my supervisors, who have supported me on the entire way leading to this work with their enormous knowledge and advice. I am grateful for the support and insights shared by Jörn and the SIXTE team including Lea, Maximilian, Ole, Thomas, and the other members of the Remeis observatory. I want to highlight Christian, who was so patient with me and took every single question I addressed to him seriously. His conviction and passion to do science intrigued me like no other tutor had done during my bachelor's studies. Christian and Jörn's wive Katja invited me to participate in a XRISM meeting with NASA scientists in which I gained a lot of experience. I am grateful for Edmund's help, a NASA scientist working on XRISM, who participated in this meeting and provided assistance regarding his work on Resolve's CPU load as far as he was allowed to. Finally, the working atmosphere at the observatory is very pleasant and the organization is professional, which enabled me to work efficiently and joyfully.

List of Figures

List of Tables

8. References

Dauser T., Falkner S., Lorenz M., et al., 2019, 630, A66

Dauser T., Wilms J., Peille P., et al., 2023, SIXTE MANUAL, `http://www.sternwarte.uni-erlangen.de/~sixte/data/simulator_manual.pdf` Accessed: 2024-02-20

Hodges-Kluck E., 2023, Observing Bright Sources with Resolve, `https://heasarc.gsfc.nasa.gov/docs/xrism/analysis/workshops/doc_feb23/Hodges-Kluck_Bright_Sources.pdf` Accessed: 2023-10-17

Ishisaki Y., Yamada S., Seta H., et al., 2016, In: den Herder J.W.A., Takahashi T., Bautz M. (eds.) Space Telescopes and Instrumentation 2016: Ultraviolet to Gamma Ray, Vol. 9905. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, p. 99053T

McCammon D., 2005, Thermal Equilibrium Calorimeters - An Introduction. In: Enss C. (ed.) Cryogenic Particle Detection, Vol. 99., p. 1

Mizumoto M., Tsujimoto M., Cumbee R.S., et al., 2022, In: den Herder J.W.A., Nikzad S., Nakazawa K. (eds.) Space Telescopes and Instrumentation 2022: Ultraviolet to Gamma Ray, Vol. 12181. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, p. 121815Z

Tsujimoto M., Tashiro M.S., Ishisaki Y., et al., 2018, Journal of Low Temperature Physics 193, 505

XRISM Science Team 2020, Science with the X-ray Imaging and Spectroscopy Mission (XRISM), White paper (arXiv:2003.04962)

XRISM Science Team 2022, arXiv e-prints arXiv:2202.05399

A. Algorithm in python code

```
from astropy.io import fits
import numpy as np
from collections import deque


def whichcpu(event, dictwhich):
    return (dictwhich[event["PIXID"]][1] - 1)


class CPU():

    def __init__(self, max_fifo_len, numfifos, dictwhich, a, bp, bs, cp
                                        , cs, e):
        self.fifos = []
        for i in range(numfifos): self.fifos.append(deque())
        self.max_fifo_len, self.dictwhich = max_fifo_len, dictwhich
        self.a, self.bp, self.bs, self.cp, self.cs, self.e = a, bp, bs,
                                            cp, cs, e
        self.queue = deque(np.arange(0, numfifos))
        self.numfifos = numfifos
        self.executed = []
        self.notexecuted = []
        self.currentTime = 0

    def gradtotime(self, event):
        hp, mp, ms, lp, ls = 0, 0, 0, 0, 0
        if event["GRADING"] == 0: hp = 1
        if event["GRADING"] == 1: mp = 1
        if event["GRADING"] == 2: ms = 1
        if event["GRADING"] == 3: lp = 1
        if event["GRADING"] == 4: ls = 1
        return (self.a*hp + self.bp*mp + self.bs*ms + self.bs*ls + self
                                            .cp*lp + self.cs*ls) / (1-
                                            self.e)

    def whichfifo(self, event):
        return self.dictwhich[event["PIXID"]][0] - 1

    def insert_event(self, event):
        self.propagate_queue(event["TIME"])
        fifoidx = self.whichfifo(event)

        if len(self.fifos[fifoidx]) >= self.max_fifo_len:
            self.notexecuted += self.fifos[fifoidx]
            self.fifos[fifoidx].clear()

        self.fifos[fifoidx].append(event)


    def propagate_queue(self, timing):
```

```
        num_empty_fifos = 0
        while num_empty_fifos < self.numfifos:
            ififo = self.queue[0]
            if len(self.fifos[ififo])!=0:
                num_empty_fifos = 0
                event = self.fifos[ififo][0]
                if timing - self.currentTime >= self.gradtotime(event):
                    self.currentTime += self.gradtotime(event)
                    self.executed.append(event)
                    self.queue.rotate(-1)
                    self.fifos[ififo].popleft()
                else:
                    break
            else:
                num_empty_fifos += 1
                self.queue.rotate(-1)

        if num_empty_fifos == self.numfifos:
                self.currentTime = timing


    def finnish_sim(self):
        self.propagate_queue(self.currentTime + 1000)


def build_cpus(num_cpus, max_fifo_len, numfifos, dictwhich, a, bp, bs,
                                    cp, cs, e):
    CPUs = []
    for i in range(num_cpus): CPUs.append(CPU(max_fifo_len, numfifos,
                                        dictwhich, a, bp, bs, cp, cs, e
                                        ))
    return CPUs


def processevents(cpus, dictwhich, infile):
    data = fits.open(infile)[1].data
    timesort = np.argsort(data["TIME"])
    data = data[timesort]
    dtype = data.dtype

    lenexec, lennotexec = 0, 0
    offsetexe, offsetnotexe = 0, 0

    for event in data:
        cpu = whichcpu(event, dictwhich)
        cpus[cpu].insert_event(event)

    for icpu in cpus:
        icpu.finnish_sim()
        lenexec += len(icpu.executed)
        lennotexec += len(icpu.notexecuted)
```

```
    processed, notprocessed = np.zeros(lenexec, dtype=dtype), np.zeros(
                                        lennotexec, dtype=dtype)

    for icpu in cpus:
        for ieve in range(len(icpu.executed)):
            for nam in processed.dtype.names:
                processed[ieve + offsetexe][nam] = icpu.executed[ieve][
                                                    nam]
        for ieve in range(len(icpu.notexecuted)):
            for nam in notprocessed.dtype.names:
                notprocessed[ieve+offsetnotexe][nam] = icpu.notexecuted
                                                    [ieve][nam]
        offsetexe += len(icpu.executed)
        offsetnotexe += len(icpu.notexecuted)

    return processed, notprocessed


def writeevents(out, outfitsname):
    primary_hdu = fits.PrimaryHDU()
    exec_hdu = fits.BinTableHDU(data = out[0], name = "Processed")
    notexec_hdu = fits.BinTableHDU(data = out[1], name = "NotProcessed"
                                    )
    hdul = fits.HDUList([primary_hdu, exec_hdu, notexec_hdu])
    hdul.writeto(outfitsname)
```

List of abbreviations

XRISM	X-ray Imaging and Spectroscopy Mission
JAXA	Japan Aerospace Exploration Agency
NASA	National Aeronautics and Space Administration
ESA	European Space Agency
SXS	Soft X-ray Spectrometer
SIXTE	Simulation of X-ray Telescopes
PSP	Pulse Shape Processor
FPGA	Field Programmable Gate Array
CPU	Central Processing Unit
EDB	Event Dual Buffer
FIFO	First-In-First-Out
SMU	Satellite Management Unit
DR	Data Receiver
Hp	High resolution primary
Mp	Medium resolution primary
Ms	Medium resolution secondary
Lp	Low resolution primary
Ls	Low resolution secondary
SIMPUT	Simulation Input

Erklärung

Hiermit bestätige ich, dass ich diese Arbeit selbstständig und nur unter Verwendung der angegebenen Hilfsmittel angefertigt habe.

Ort, Datum	David Lochner