High Throughput Low Latency Online Image Processing by GPU/FPGA Data Coprocessors using RDMA Techniques

Raphael Ponsard

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : **SIGNAL IMAGE PAROLE TELECOMS**

Arrêté ministériel : 25 mai 2016

Présentée par

RAPHAEL PONSARD

Thèse dirigée par **DOMINIQUE HOUZET**, Université Grenoble Alpes
et codirigée par **Vincent FRISTOT**, Université Grenoble Alpes**,**
et **Nicolas JANVIER,** ESRF, Grenoble

préparée au sein du **Laboratoire Grenoble Images Parole Signal Automatiques**
dans **l'École Doctorale Electronique, Electrotechnique, Automatique, Traitement du Signal (EEATS)**

Traitement en temps réel, haut débit et faible latence d'images par coprocesseurs GPU & FPGA utilisant les techniques d'accès direct à la mémoire distante

High Throughput, Low Latency Online Image Processing by GPU/FPGA Coprocessors using RDMA Techniques

Thèse soutenue publiquement le **10/12/2020,**
devant le jury composé de :

Madame LAURENCE PIERRE
PROFESSEUR DES UNIVERSITÉS, UNIVERSITÉ GRENOBLE ALPES,
Présidente du jury
Monsieur DOMINIQUE HOUZET
PROFESSEUR DES UNIVERSITÉS, GRENOBLE INP,
Directeur de thèse
Monsieur BRICE GOGLIN
DIRECTEUR DE RECHERCHE, INRIA BORDEAUX – SUD - OUEST,
Rapporteur
Monsieur EMMANUEL FARHI
INGENIEUR HDR, SYNCHROTRON SOLEIL- GIF-SUR-YVETTE,
Rapporteur
Monsieur NICOLAS JANVIER
Ingénieur, ESRF GRENOBLE,
Invité
Monsieur VINCENT FRISTOT
MAITRE DE CONFÉRENCE, UNIVERSITÉ GRENOBLE ALPES,
Invité

High Throughput Low Latency Online Image Processing by GPU / FPGA Data Coprocessors using RDMA Techniques

Raphaël Ponsard

Abstract— The constant evolution of X-ray photon sources associated to the increasing performance of high-end X-ray detectors allows cutting-edge experiments that can produce very high throughput data streams and generate large volumes of data that are challenging to manage and store.

In this context, it becomes fundamental to optimize processing architectures that allow real-time image processing such as raw data pre-treatment, data reduction, data compression, fast-feedback. These data management challenges have still yet not been addressed in a fully satisfactory way as of today, and in any case, not in a generic manner.

This thesis is part of the ESRF RASHPA project that aims at developing a RDMA-based Acquisition System for High Performance Applications. One of the main characteristics of this framework is the direct data placement, straight from the detector head (data producer) to the processing computing infrastructure (data receiver), at the highest acceptable throughput, using Remote Direct Memory Access (RDMA) and zero-copy techniques with minimal Central Processing Unit (CPU) interventions.

The work carried out in this thesis is a contribution to the RASHPA framework, enabling data transfer directly to the internal memory of accelerator boards. A low-latency synchronisation mechanism between the RDMA network interface cards (RNIC) and the processing unit is proposed to trigger data processing while keeping pace with detector. Thus, a comprehensive solution fulfilling the online data analysis challenges is proposed on standard computer and massively parallel coprocessors as well.

Scalability and versatility of the proposed approach is exemplified by detector emulators, leveraging RoCEv2 (RDMA over Converged Ethernet) or PCI-Express links and RASHPA Processing Units (RPUs) such as Graphic Processor Units (GPUs) and Field Gate Programmable Arrays (FPGAs).

Real-time data processing on FPGA, seldom adopted in X ray science, is evaluated and the benefits of high level synthesis are exhibited.

The framework is supplemented with an allocator of large contiguous memory chunk in main memory and an address translation system for accelerators, both geared towards DMA transfer.

The assessment of the proposed pipeline was performed with online data analysis as found in serial diffraction experiments. This includes raw data pre-treatment as foreseen with adaptive gain detectors, image rejection using Bragg's peaks counting and data compression to sparse matrix format.

Traitement en temps réel, haut débit et faible latence, d'images par coprocesseurs GPU & FPGA utilisant une technique d'accès direct à la mémoire distante

Résumé— L'amélioration permanente des sources de rayonnement X, ainsi que les gains en performances des détecteurs de dernière génération rendent possibles des expériences qui peuvent produire des quantités énormes de données à haut débit, aussi difficiles à gérer qu'à stocker.

Dans ce contexte, il devient indispensable d'améliorer les systèmes de calculs et de permettre le pré-traitement en temps réel des données brutes, la réjection de celles qui sont inutiles, la compression voire la supervision en temps réel de l'expérience. Ces problématiques de gestion des flux de données n'ont pas encore reçu de réponse générique pleinement satisfaisante.

Cette thèse fait partie d'un projet plus vaste, le projet RASHPA de l'ESRF, visant à développer un système d'acquisition haute performance basé sur le RDMA (Remote Direct Memory Access). Une des caractéristiques essentielles de ce projet est la capacité à transférer directement des données de la tête du détecteur vers la mémoire de l'unité de calcul, au plus haut débit possible, en utilisant les techniques d'accès direct à la mémoire, sans copies inutiles, et minimisant le recours à un processeur.

Le travail réalisé pendant cette thèse est une contribution au système RASHPA, qui rend possible, non seulement le transfert de données dans la mémoire du système de calcul, mais aussi directement dans la mémoire interne de cartes accélératrices dans le cas de système à l'architecture hétérogène.

Un mécanisme de synchronisation à faible latence entre carte réseau et unité de calcul est proposé, déclenchant le traitement des données au rythme du détecteur.

Cela permet de fournir une solution globale au traitement de données en temps réel, tant sur ordinateurs classiques que sur accélérateurs massivement parallèles.

Pour illustrer la versatilité de l'approche proposée, plusieurs simulateurs de détecteurs ont été réalisés, s'appuyant sur les protocoles RoCEv2 ou PCI Express pour la partie transport, ainsi que diverses unités de calcul compatibles RASHPA à base de cartes graphiques (GPU) ou de circuits reconfigurables (FPGA).

Le traitement de données en temps réel sur FPGA, encore peu pratiqué dans les sciences du rayon X, est évalué en s'appuyant sur les récentes avancées de la synthèse de haut niveau (HLS).

La qualification du pipeline de calcul proposé a été faite en s'inspirant d'expériences de cristallographie en série (SSX). Il comprend un pré-traitement des données brutes comme prévu pour un détecteur à gain adaptatif, la réjection d'images en fonction du nombre de pics de Bragg, et la compression des données au format matrice creuse.

To Arlette and Nathalie,

Cette thèse n'aurait pas été possible sans l'esprit d'initiative de Pablo et Nicolas à la tête de ISDD et de EU. Ils m'ont fait confiance et ont bien voulu me confier un projet aussi important pour le rayonnement scientifique de l'ESRF. Je leur en suis profondément reconnaissant. Merci aussi à Marie qui gère le DDP avec brio pour son support constant.

Dominique et Vincent ont assuré l'encadrement universitaire, tâche peu commode à distance et leurs conseils et pressions m'ont permis d'aboutir dans les temps.

Je veux aussi saluer ici Andy et Petri grâce à qui j'ai pu faire mes premiers pas à l'ESRF dès 1999 et bien sûr l'équipe TANGO avec qui j'ai tout découvert à l'époque du fonctionnement de cette grandiose machine.

Pendant ces trois années de thèse passées si vite, Jérôme n'a pas compté son temps pour m'introduire aux subtilités de la programmation des GPUs, domaine dans lequel il excelle, mais aussi pour me faire découvrir la cristallographie ou les arcanes du calcul scientifique. Merci aussi à tous les collègues du DAU.

Pendant mon séjour, l'équipe RASHPA s'est progressivement étoffée et j'ai beaucoup appris avec Wassim, grand maître du FPGA et Aurélien qui m'a beaucoup impressionné avec son esprit de méthode. Je salue toute l'équipe EU pour son accueil sympathique.

Alejandro et Samuel m'ont aussi énormément apporté.

Laura a été très courageuse quand elle a accepté de relire et de corriger le manuscrit en anglais.

Je profite aussi de cette occasion pour saluer aussi tous mes anciens collègues de l'E.N. et bien sûr tous mes camarades que j'ai un peu laissé tomber pendant cette période un peu frénétique.

Pardon à tous ceux que je ne cite pas nommément, vous êtes bien trop nombreux !

Cela a été un grand bonheur.

Contents

List of Figures

List of Tables

Listings

Acronyms

AXI Advanced eXtensible Interface. 74

CPU Central Processor Unit. 12

CUDA Compute Unified Device Architecture. 17, 65

DAQ Data Acquisition System. 11

ESRF European Synchrotron Radiation Facility. 15, 19, 51

FPGA Field Programmable Gate Array. 12, 14, 27, 40, 72

GPU Graphics Processor Unit. 12, 14

HLS High Level Synthesis. 41

HPC High Performance Computing. 15, 31

I/O Input and Output. 22

MMU Memory Management Unit. 22

PCI-e Peripheral Component Interconnect Express. 17, 23, 49, 66

RASHPA RDMA-based Acquisition System for High Performance Applications. 15

RDMA Remote Direct Memory Access. 12

REMU RASHPA detector EMulator. 53

RNIC RDMA Network Interface Card. 12, 48

RoCEv2 RDMA Over Converged Ethernet. 17, 19, 93

ROMULU RASHPA back end receiver. 68

Glossary

Data Transfer Process is an ongoing data transfer Process between two RASHPA nodes. A RASHPA module can produce multiple concurrent DTPs from different data slices (temporal sampling) or from different Region of Interest (spatial sampling). 42

DMA stands for Direct Memory Access and describes an inner computer component in charge of data transfer from one memory region to another. DMA can be of two forms: Memory Mapped or Streamed. 24

High Level Synthesis is a development flow aiming at FPGA design using High Level language and concepts instead of convoluted low level, error prone, highly specialized techniques.. 74

RASHPA Buffer is the memory region allocated in the destination computer for the data transfers. It must be compatible with DMA operations. 42

RASHPA Manager is a software application in charge of configuring and monitoring a RASHPA system. It can perform capabilities retrieval of RASHPA nodes prior configuration, start/stop the acquisition and manage the errors. It uses XML telegram for communication on a dedicated control link. 42

RASHPA Processing Unit is an hardware entity that can ingest incoming data flow from an RNIC in its internal memory and that can process those data when triggered properly.. 66

RDMA Remote DMA is the generalization of the DMA concept to remotely connected computers. In this memoir, we focused on gigabit Ethernet links and RoCE protocol. 29

Acknowledgments

This work is part of the T2.a technology program proposed in the frame of ESRF-EBS Detector Development Plan (DDP).

Chapter 1

Introduction

Contents

Exploiting the full performance of the new generation X-ray 2D image detectors in terms of throughput, is challenging when using existing hardware and software solutions. This thesis aims at proposing contributive techniques to mitigate these difficulties.

This first chapter will give a broad overview of photon science and the minimal prerequisites on detector hardware. Some X-ray imaging techniques demanding in processing power are presented as well. Then we will describe the problem in detail and present the original intentions for this project. The significance of the thesis and our contributions are then stated before finally specifying the outcome of our work.

1.1 Making Science using X-ray Radiation

Please note that this section provides merely a brief introduction to the long and rich history of X-rays.

X-rays have a history parallel to that of Modern Physics and to the general understanding of fundamental laws of electromagnetism in the late 19^{th} century. This leads to the unification of formerly unrelated topics, such as light or electricity and mysterious artefacts in electrostatic or magnetic phenomenons.

X-rays were discovered by chance in 1895 by W. Roengten, who noticed a faint shimmering from a fluorescent screen at a remote location while he was studying the effect of high voltage in vacuum tubes.

Shortly, he discovered some of the interesting properties of the invisible and yet unknown radiation and took the first radiography in history presented in Figure 1.1. Because he did not know their origin he named them X-rays after the x in mathematics, the symbol of an *unknown* value.

The first X-ray sources in the late 19^{th} were evacuated sealed tubes. Last ones are the kilometer long linear accelerators called XFELs (XFree Electron Laser), built since late 20^{th} century, and producing extremely short (femtosecond range) and intense X-ray flashes with the properties of laser light. X-rays produced in a sealed tube, XFELs or at a synchrotron facility are fundamentally the same form of electromagnetic radiations and differ only by their respective energy and by the process of how they are produced.

Charged particles (e.g., electrons and ions) moving at high speeds lose some energy when their acceleration is changed. This is the case when they hit the cathode in Crook's tube or when their trajectory is curved by a bending magnet. This lost energy is the source of the X-rays. Relativistic particles at high speed (close to the speed of light) are extremely energetic and produce intense X-ray beam.

There are no natural source of synchrotron radiation on Earth. But there are such natural sources in the Universe, e.g. in some rotating stars. Synchrotron Radiation (SR) is one of the most important emission processes in astrophysics.

Synchrotron radiation was first observed in year 1947 as a byproduct during the first high-energy-physics (HEP) experiments conducted in circular accelerators where particles get accelerated periodically when they go through an RF cavity. Thus, they acquire during each revolution, synchronously, more and more energy. This is the origin for the name. The first synchrotron accelerator, that could stand on a table, is shown in

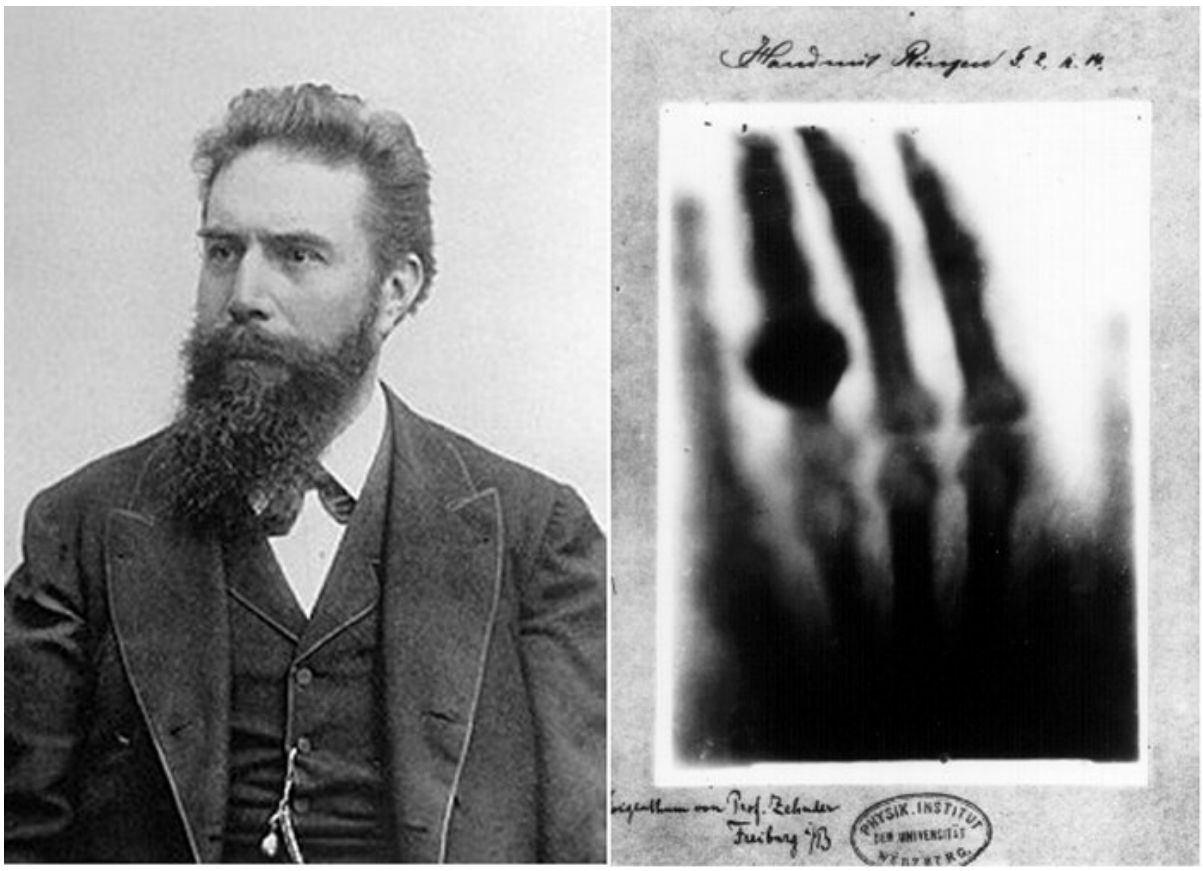


Figure 1.1: Left: picture of W. Röngten. Right: The first medical radiography taken in history by W. Röngten, the hand of his wife Anna Bertha Ludwig and her ring. (Source: Wikipedia)

Figure 1.4. During these early discoveries, SR was mainly a kind of a nuisance for the experimentalists.

As the particle energy is higher, the produced X-rays are much more intense than those produced by Crooks-like tubes. A SR beam is also pulsed, highly collimated and has a narrow spectral range [44]. Indeed they have many interesting properties, that shortly attracted the attention of many researchers looking for powerful light sources.

Synchrotron radiation scientists became kind of parasites on nuclear physics experiments, during the so-called 1^{st} generation synchrotron era, before they built their own facilities, those being the 2^{nd} generation synchrotron facilities.

There has been an amazing increase in brilliance of X-ray sources, see Figure 1.3, and a successful new science and technology has emerged to fruitfully employ SR in multiple research fields. This has boosted new usages and applications in industry, structural biology, condensed materials investigation, anthropology or cultural heritage studies, etc. X rays are mainly used for observation purpose and may be used in combination with heated or cooling enclosure, high pressure diamond anvil, laser beam, etc.

SR facilities are essential to science today and are in use all around the world. The

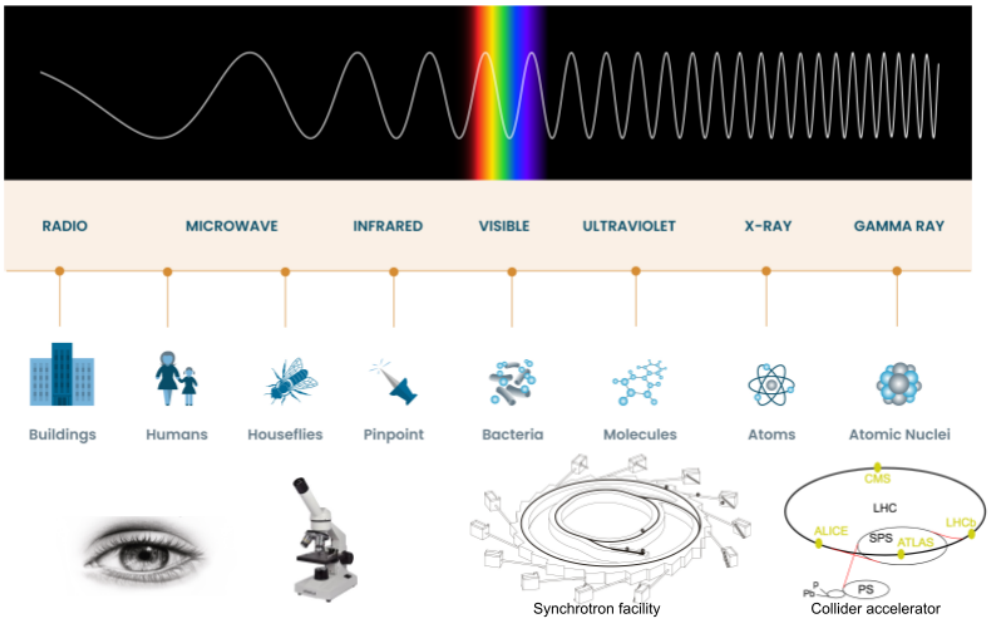


Figure 1.2: The electromagnetic spectrum comparing the size of objects that can be studied with various techniques. X-rays have the perfect wavelength necessary to perform investigations in material sciences.

fact is that X-rays have the right characteristics in terms of wavelength for many research fields as shown in Figure 1.2.

X-rays are not limited to 2D imaging based experiments. Many other techniques, fluorescence, spectroscopy, scalpel in brain surgery and uncountable other techniques are foreseen. However, in the context of this thesis, the focus will only be on the 2D image topic.

2D imaging techniques are not limited to collect digitized density variations of a sample, as done in the first X-ray image or modern scanners. It can also record scattering images containing information on the 3D atomic arrangement in a crystal. A high flux beam is likely to burn fragile samples during the data collection. Many techniques are under scrutiny in order to decrease the amount of radiation dose needed. This enables an increased acquisition rate in order to observe the kinetics of chemical reactions.

All this contribute to the success of X-ray science and explains why there are numerous SR sources operating all around the world, see Figure 1.5 and why so many scientists are competing for beam time.

1.1.1 ESRF-EBS Grenoble, The European Synchrotron

The european synchrotron project started in the 1980s, and construction work was completed 10 years later in 1995 see Figure 1.6. The ESRF accelerator ring was the first to implement Insertion Devices that boost brilliance by the mean of additive interference in

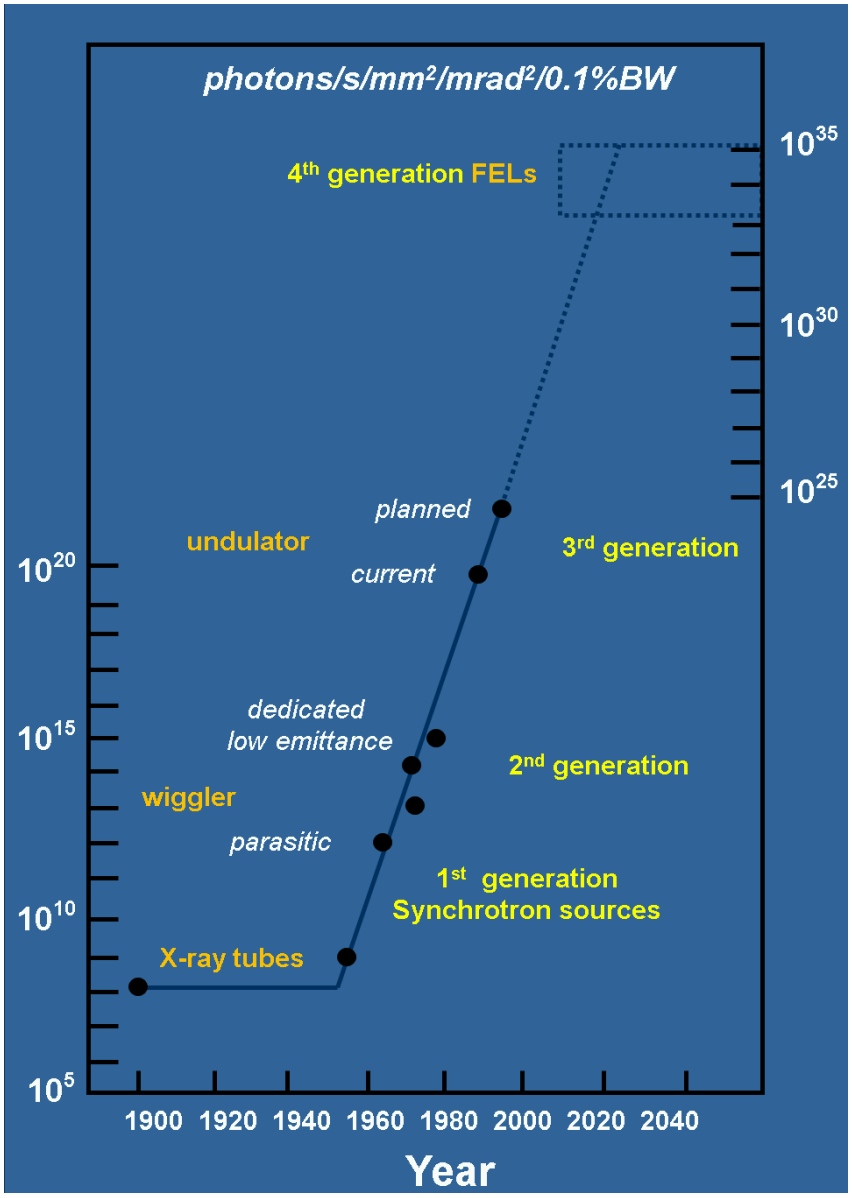


Figure 1.3: The brilliance measures the number of photons by unit of time, solid angle, square section. As no optical technique can improve it, it is considered as the best measure of X-ray flux. (Sources: J. Susini, ESRF)

Figure 1.4: From left to right: Langmuir, Elder, Gurewitsch, Charlton and Pollock made the first observation of the SR light around the vacuum chamber of a 1947 general electric synchrotron. (Source: NSLS, Brookhaven)

produced X-rays. Thanks to continuous innovations, the ESRF has since its early days been continuously in the top rank of the third generation synchrotrons.

A synchrotron radiation facility is a complex piece of engineering sections, including the core components shortly described below:

- a *linear accelerator* (linac), to produce the electrons at a given rate and give them a first acceleration, using a series of oscillating electric potentials.
- a *booster synchrotron*, to give them the equivalent of 6 GeV energy, in a fraction of a second, a level at which they behave as relativistic particles.
- a *storage ring*, a 850 m polygon, were electrons move straight in a high vacuum tube, focused by multi-pole magnets and bent by bending magnets at 32 edges.
- and multiple *beamlines*, tangent to the bending magnets, collecting the X-ray flux, that will be applied in the experimental hutches. Each beam line is 3 folded: optic hutch, experimental hutch, and control room.
- and there are also radiofrequency cavities (RF), vacuum pumps, beam alignment devices, timing system, computing and networking infrastructure, etc that we will not describe here.

Figure 1.5: World map of Synchrotron Radiation sources in operation or under construction. Proof of their invaluable service to science community, many SR facilities are in operation all around and counting. (Source: ESRF)

The ESRF is a world-class research center, with 700 staff members and a vast community of users. To keep its place in the world leading group of synchrotron, continuous upgrades and improvements are ongoing in all domains.

The ESRF resumed just recently to user-mode after one year and a half of shutdown. During this time the Extremely Brilliant Source (ESRF-EBS) upgrade program took place, which were mainly replacement of the storage ring. The ESRF is as of August 2020, the most brilliant fourth generation source!

Using recently invented HMBA (Hybrid Multi Bend Achromat) lattice by [51] in place of the standard bending magnets in the storage ring, a physicists dream enabling new science became a reality. Brilliance (x100) and focus (/30) have been considerably improved opening up new fields of investigation for fundamental research. It will also permit improved understanding of materials with the possibility to reach spatial resolution at the nanometer level, enabling unprecedented characterisation and understanding of materials and living matter.

Most of the 43 beamlines were kept as is with minor upgrades but eight are completely being redesigned.

Throughout this thesis, the ID29-EBSL8 beamline devoted to serial x-ray diffraction experiments, one of the most demanding in photon science, will serve us to exemplify the

(a)

(b)

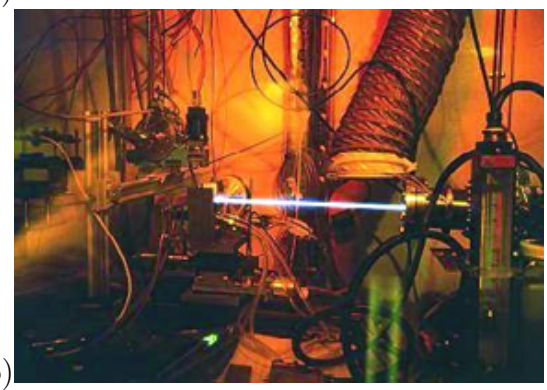


Figure 1.6: The European Synchrotron, Grenoble, France. From construction work (a) in 1986 to the first light (b) in 1995: 10 years of hard work (Source: ESRF)

(a)

(b)

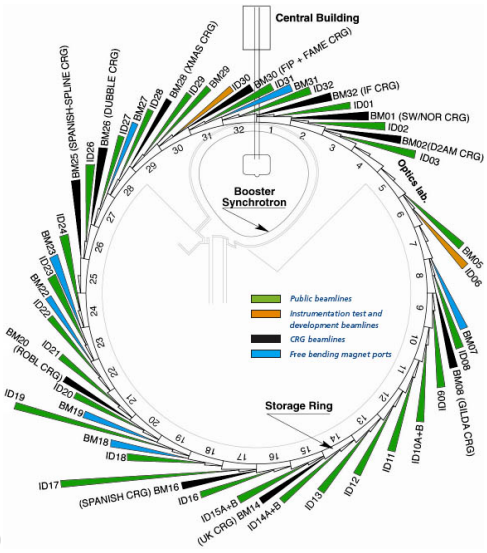


Figure 1.7: (a) The European Synchrotron Radiation Facility (ESRF) today in the European Photon and Neutron science campus (EPN) and (b) a map of the ESRF beamlines. (Source: ESRF).

data processing challenges.

1.1.2 X-ray Detectors and Experimental Setup Overview

X-ray science and SR are a vast research field and it is quite impossible to give a comprehensive overview of the topic. The size of the accelerator community to which SR contributes as a large part, the growing and counting number of facilities of any size all around the world, is a vivid proof that these light sources provide invaluable services in a broad spectrum of research topics. The work of [66] provides meaningful general insights on this topic.

For sure, not all X-ray experiments are producing large amounts of data and do not rely on online data processing. However a growing number does.

A selection of innovative demanding X-ray experimental setups is given below:

- In Serial Synchrotron X ray crystallography (SSX), i.e. diffraction experiments on proteins, a flow of a liquid containing small crystals is sent through a pulsed beam. The idea is to collect the signal of one unique crystal at a time. One of the major characteristics of serial diffraction is the amount of raw data produced. Most of the captured frames do not contain information, hence are useless. Up to 90% of all the collected signal is empty because no crystal is hit by the beam. The main data treatment is then to sort out frames to be saved and those which can be discarded. This selection is called the veto algorithm.

- X-ray Computed Tomography (XRCT) i.e. a scanner infers the density of the matter by measuring the image contrast. The first application of GPU computing for synchrotrons exploited the abilities of GPU to perform Fast Fourier Transform efficiently and ported the filtered backprojection algorithm for tomographic reconstruction [8]. This kind of processing is performed offline by PyHST2 software because the data collection needed to be completed before initiating the processing[1].

- Coherent Imaging have appeared in the last decade at X Free Electron Lasers (XFELs) that were much brighter than SR and have a much more coherent X-ray beam. These sources are so bright that they blast the sample in femto-seconds but the electron cloud of the sample diffracts before the sample is destroyed. Collecting interference patterns from samples has paved the way to powerful microscopy techniques: Coherent Diffraction Imaging (CDI) and Ptychography[2] nowadays. They have become available on the most brilliant synchrotron sources.

  Ptychography and other coherent imaging techniques rely on Fourier transform. Those analysis match perfectly with GPU computing as stated by [11] and is illustrated by the success of the PyNX suite[3].

[1] https://gitlab.esrf.fr/mirone/pyhst2

[2] image reconstruction technique using many interference patterns obtained as the sample is displaced to various positions

[3] https://github.com/nx-python/PyNX

Not all computations are possible online but a lot of low-level image pretreatment are: Bragg's peak detection, background normalization, mean and standard deviation calculation, thresholding, outliers removal, masking, rejection, ...

The new generation of X-ray detectors features cutting edge capabilities and benefits from the improvement in different fields of technology. Some of the most promising improvements are listed below:

- Larger sensor area and modular design for a wider acquisition area. From photons to digital value, there is a complex process, starting with photon to electron transformation by various means. Large sensors are difficult to manufacture. Therefore large detectors are comprised of assembly of multiple modules side by side.
- Higher pixel number by surface unit, higher acquisition rate and lower relaxation time. Last decades has seen the emergence of pixel detectors where every pixel embeds enough electronics to count each photons individually, benefiting from the better integration of ASIC chips. Therefore, a larger set of counting capabilities is possible, such as adaptive gain, pixel binning (merging in a single point the energy collected from neighboring pixels), etc. They are now collecting images routinely at hundreds of Hz and even a kHz. This produces a larger amount of raw data, in more complex formats and creates compelling needs for online data processing.

The SR community has identified some key topics to drive productivity and user experience in X-ray science [62]. Beside resolving high-throughput data transfer challenges, three different classes of crucially urgent problems are emerging. They could be worked out by online data processing. These challenges are sorted by related latency and are listed below:

- At human rate ($fps < 1$), it would be desirable to provide a minimum feed-back to the user, in order to monitor the proper functioning of the current experiment
- At control system rate ($fps < 100$), it would be interesting to enable remote control of experiments, automatically survey of sample alignment and similar control loop.
- At full acquisition rate ($fps > 1000$), performing low level data processing prior data rejection or compression. This includes raw data pretreatment needed prior any other processing, geometry reconstruction (offset, rotation), outliers removal (bad pixel) or masking (beam stop), binning...

In this context, the support of new computing hardware such as GPU and FPGA boards is clearly a way to explore. But this is not yet the end of standard CPU computing that are also evolving. By example, an attempt to tackle the challenges of sieving out in real time the data produced by one of the most challenging detector available as of today (Jungfrau 32M) is presented in [29] and this work evaluates the vectorized instruction set of a standard CPU.

All these problems are mostly addressed by RASHPA framework, demonstrating the strengths of the concepts of the work carried out.

1.2 Full Throughput X-ray 2D Imaging Experiments

Today the throughput of advanced detectors and the needs in associated online data processing are evolving faster than the capabilities of standard computers. Some of the encountered issues are related to the data throughput and the afferent bottlenecks along the data path. The others are related to the processing power required for the data processing itself and will be discussed later. Of course, these problems of transfer and processing of data are not specific to X-ray detectors. The following dissertation and proposal might apply to many other Data Acquisition System (DAQ) systems.

High throughput data transfer itself is a creative research topic and, in this field, innovative technologies emerge every day. However, there is yet no *one-size-fits-all* solution and probably never will be.

Targeting specifically 2D image detectors, we are somewhat narrowing the problem and are more prone to propose specific solutions, taking into account their peculiarities. Some of these particularities might be considered as extraneous constraints to the problem, e.g. the limited processing power of the detector-embedded readout-electronics. We could instead take benefit of other particularities, e.g. assuming that data transfer will be lossless in a lab environment. Thus, with properly configured and correctly sized networking devices, no packet losses should happen. That let us choose a lightweight transfer protocol.

Since detector developments are long processes that commonly take 5 to 10 years and detectors usually stay in operation around 10-20 additional years, the generality of the proposed system should be strongly considered. Most components of the framework should ideally be hardware agnostic and able to cope with the most disruptive future innovations. Of course, hardware implementations demonstrates the advantages the solution can offer to the user and are essential to verify that the targeted specifications are realistic. Due to the lack of available real detectors featuring all the advanced functionalities, we have developed simulators in order to benchmark the various alternatives of communication protocols.

The developed framework is intended to be scalable, i.e. possibly address the most demanding DAQ systems, but also more cost effective solution deprived of the latest innovative and expensive hardware. It is designed with X-ray detectors in mind, producing high throughput data streams, but the framework should also match the low-latency requirements of fast-feedback and control system.

The other significant characteristics of 2D imaging is the huge amount of data which is generated.

New generation pixel detector, as the PSI Jungfrau shown Figure 1.8, embed more and more processing power at the pixel level. Consequently the detector readout electronics that is collecting the data can generate huge throughput.

In many serial experiments, the setup comprises an automatic sample changing system, such as robotic arm, rotating disc or liquid jet carrying samples into the X-ray beam. Experiments like these producing large images (sizes in the MB range), acquired at high repetition rates (frames per second in the kHz range), are summing up in Tera Bytes of

data per minute. Obviously, merely storing the raw data would put a considerable strain on the storage file system.

As a consequence, end user will have to be ready to provide an online rejection algorithm at the time of the acquisition.

Nevertheless, the ubiquitous workflow: acquisition, transfer, storage and eventually batch processing, is no more the most effective. It now seems mandatory to perform online processing and proceed to some form of data compression or even rejection. These tasks require a considerable computing power and a high throughput that overwhelm traditional hardware architectures.

This challenge is not specific to photon science but it is tightly coupled to the previously mentioned data transfer bottlenecks. Many research fields focus on this topic and the impending end of Moore's Law has started a rethinking of the way to solve Big data analysis (see Figure 1.9). Certainly, the peak processing power of a single processor core does not increase the way it did, but hopefully new paradigms are emerging such as parallel processing and specialized hardware. The advent of high performance computing accelerators during the last decade, mainly based on Graphics Processor Unit (GPU) and Field Programmable Gate Array (FPGA), offer new opportunities to solve the computationally intensive tasks. One outcome of this work is the assessment of data transfer directly into the accelerator's memory, without neither Central Processor Unit (CPU) intervention or staged memory.

1.3 Research Questions

As the prerequisite for the work, we have identified the main causes of data transfer bottlenecks from detector to the standard computing unit. They are of multiple and mixed origins, rooted in both hardware and software.

1.3.1 Data Transfer Issues and RDMA Mitigation Techniques

In the early times of computing hardware the Central Processing Units (CPUs) where operating much faster than Input/Output (IO) devices. Real-time data processing was possible but this has not been true for a long time. The most annoying issues are linked to the complexity of the software stack used in modern architecture. Multiple data copies are also occurring between supervisor code executing in kernel memory space and user application in user space. In addition, the trivial task of using a CPU only for data transfer purposes, from input device to destination memory, wastes CPU cycles that should be more beneficial to user applications. The CPU must also acknowledge and handle interruptions issued by the communication device, using kernel context switches that are highly inefficient when using high-throughput links.

All these issues might be mitigated using Remote Direct Memory Access (RDMA) techniques and specialized hardware such as RDMA Network Interface Card (RNIC). DMA engines are specialized components embedded in computer devices such as RNICs and GPUs that can autonomously handle data transfer from PCI-e devices to the main memory or the reverse. CPU is thereby kept available for other tasks after having per-

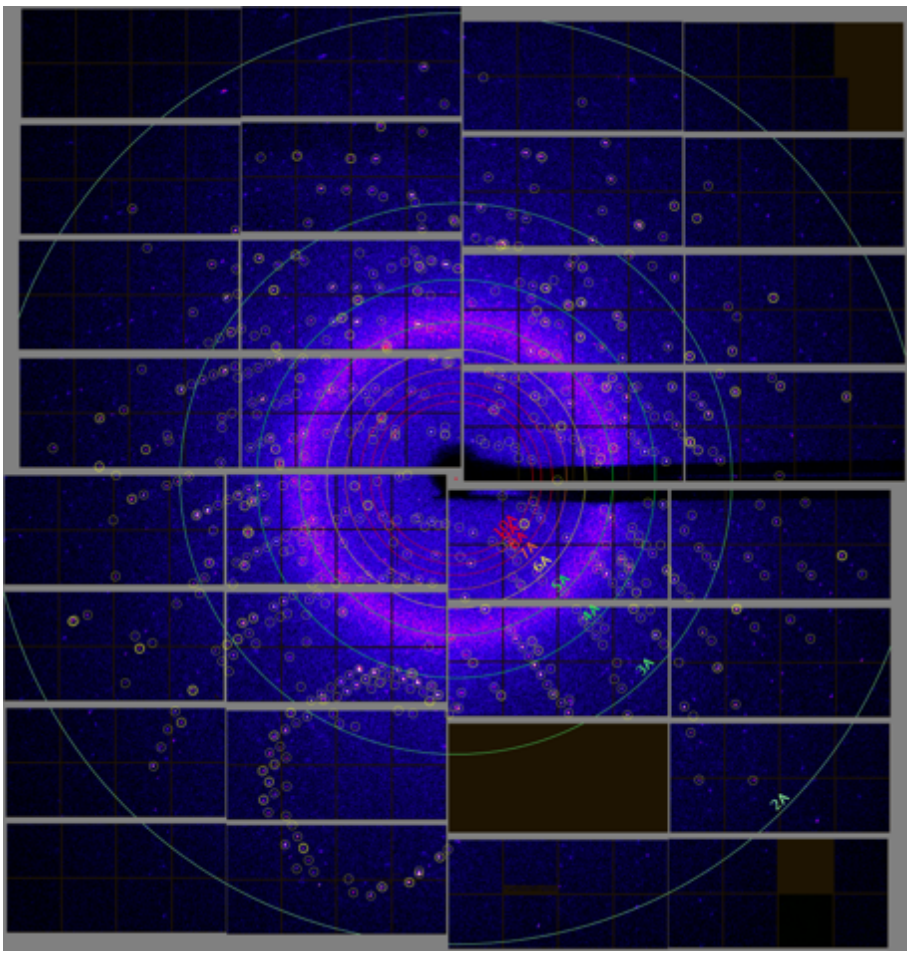


Figure 1.8: A scattering image produced by a PSI Jungfrau 16M detector featuring 32 modules, each of 1024 x 512 pixels, 16 bit raw data, operating up to 2.2 kHz. Final image reconstruction from independent modules requires geometric transformations. (Sources: Shibom Basu, EMBL)

formed the data transfer configuration. It is even possible to transfer data from one PCI-e device to another one, using peer to peer (P2P) PCI-e DMA, completely bypassing intermediary staged CPU memory. To operate properly, all that machinery is using a dedicated programming model that is slightly different from the conventional one. The abstraction layer fulfilled by this project will perform the tricky details of the low-level implementation without the end user even noticing.

RDMA is the generalization of this mechanism to the remote computer. Instead of a single DMA engine, there are two in consideration: one at the source and a second at the destination. They are inter-connected by network links and the data are streamed between both ends. We have performed an evaluation benchmark of the alternative solutions available and selected those compatible with the requirements of the detector electronics.

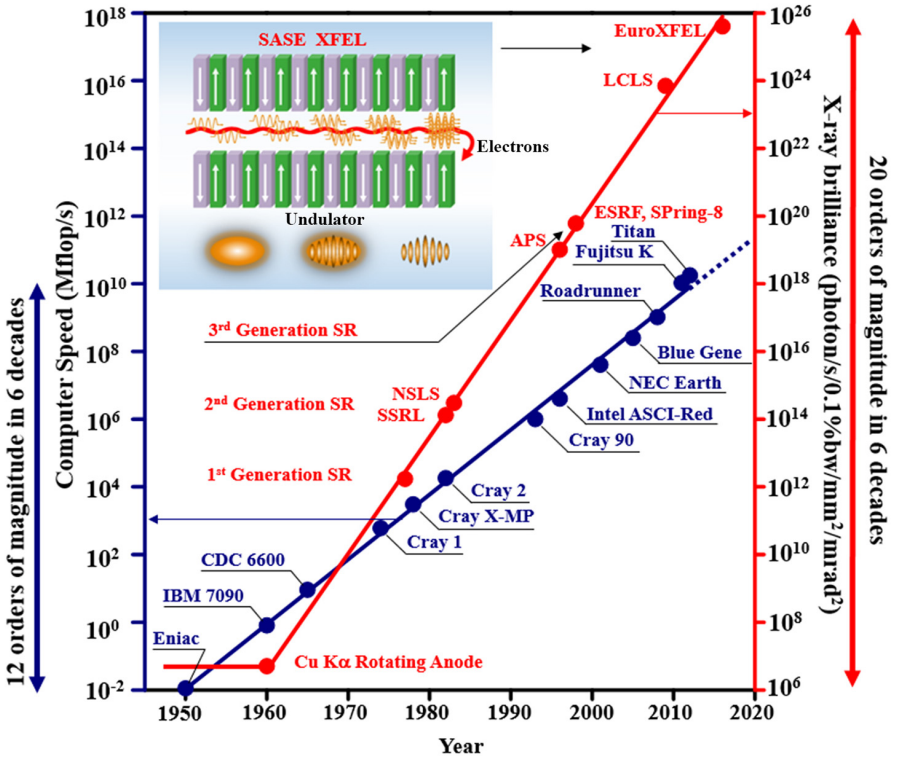


Figure 1.9: (a) Moore's Law represented for the computer speed of CPU chips compared to the X-ray brilliance of coherent light sources. While the computer speed has increased 12 orders of magnitude in 6 decades, the X-ray brilliance has improved by 20 orders of magnitude in 6 decades. The inset shows the extremely intense XFEL pulses. (Sources: UCLA Coherent Imaging Group)

1.3.2 Real-time Data Analysis with GPU/FPGA Accelerators

With the advent of the end of Moore's law, the peak processing power of a single CPU appears to be bounded. This enforces the use of heterogeneous computing systems, featuring new highly parallel hardware accelerators that have become more easily available the past 10 years. Among these are the well known Graphics Processor Unit (GPU), which is already successfully implemented in many demanding fields. But also Field Programmable Gate Array (FPGA), which has a still more limited adoption, but is geared towards critically low-latency scenario.

However, if those devices are efficient number crunchers, they are not fully autonomous black boxes, and thereby remain tightly coupled to the host computer. The main CPU must issue sequences of commands on a regular basis and must monitor the accelerator status as well. These commands include: i) explicit data movement from main CPU to accelerator memory, in both directions: Host to Device (H2D) and Device to Host (D2H), ii) launching the code processing on the accelerator synchronously with the DMA engine at data transfer completion.

This whole work of configuration cannot be done once, at the initial startup of FPGA/GPU data processing. It definitely must be performed on the fly during the whole process, at least for GPUs.

1.3.3 Contributions to RASHPA Data Acquisition Framework

This work is part of a larger project, namely the RDMA-based Acquisition System for High Performance Applications (RASHPA) framework initiated by the Instrumentation Service and Development Division (ISDD) at the European Synchrotron Radiation Facility (ESRF) (ESRF). The initial ideas of the RASHPA framework date from year 2013 [37], but remained for a long time at the stage of draft specifications. In the frame of the ESRF Detector Development Plan (2017 DDP), it was decided to restart the project and to continue until the achievements of the work would be satisfying.

In addition to the assessment of the RDMA over Converged Ethernet (RoCE) as a sustainable and robust transfer solution in the frame of RASHPA, our contribution is an extension to this framework called RASHPA Processing Units (RPUs). We have added support for data processing by GPU & FPGA accelerated code, thereafter called *kernels*, thus offering a comprehensive solution to the online data processing challenges in the context of X-ray techniques.

This extension includes a low-latency synchronization mechanism between RDMA network interface cards (RNIC) and the processing hardware, either CPUs, GPUs or FPGAs accelerator in charge of the online data processing. The transfer of the detector images onto the accelerator is made to overlap with the computation carried out, potentially hiding the transfer latency.

By applying a solution which is rarely reported in the literature, we managed to get the GPU kernels pre-launched and put them on hold in advance, in order to start them as soon as possible after being triggered by the RDMA engine at the end of the data transfer. The persistent kernel in the GPU, a continuously spinning kernel, despite being somewhat orthogonal to the underlying GPU's hardware, is investigated to limit the overhead due to kernel launch time. In addition we attempted to provide a programming methodology.

Another key contribution aiming at low-latency use, such as seen in alignment or control systems, is the successful deployment of the RASHPA paradigms with FPGA based accelerators. FPGAs have formerly been confined to very low level I/O. However, design of customized systems with demanding timing requirements are getting attention in High Performance Computing (HPC) fields for their highly configurable model and latency performances. In the same way as for other RPU instances, data are moved directly from the RNIC to the FPGA board internal memory, bypassing CPU and main memory, to be processed by user-defined IP blocks. These IP blocks usually designed using convoluted low level HDL, are evaluated in this work using high level synthesis language applying standard math expressions.

We supplemented the RASHPA framework with a collection of kernel modules, addressing the challenge of memory allocation suitable for DMA. This includes large (multi GB) contiguous memory allocation in Linux host. We also address memory allocation on

internal memory of GPU and FPGA devices. This is pertaining to PCI-e peer to peer compliant devices which memory can be accessed directly from network card.

Scalability and versatility of the proposed system is exemplified with several implementations of detector simulators. These are either full fledged software or FPGA-based implementations and hardware accelerated RPUs. They feature rejection and compression pipelines which are suitable for serial diffraction experiments conducted on synchrotrons on different hardware platforms and links.

The significance of this thesis it that the actual situation hampers the use at full capacity of existing new generation X-ray detectors and is detrimental to foreseen experimental setups. The data transfer architectures used today for data copying and dispatching are very CPU intensive. They implicitly attain performance limitations that impose the reduction of the maximum achievable frame rate or duty cycle of the detectors, consequently never exploiting their full capability. As a detrimental result, online data processing at their maximum throughput is simply not achievable.

Thanks to the approach proposed in this thesis, the RASHPA framework will hopefully help the scientists to get their result, or at least a preliminary feedback on them, in real time. This was a compelling need long hoped for, but only achievable by building the missing bridge between data flow and processing system. This is now becoming reality thanks to recent breakthrough in RDMA techniques and the tremendous processing power, low-latency of GPUs or other massively parallel architectures, like FPGA accelerators.

1.4 Organization of the Thesis

This thesis work emphasises the detailed specifications of efficient data transfer and processing framework for X-ray detectors, leveraging zerocopy / RDMA techniques.

The initial RASHPA draft, limited to transfer and storage into CPU memory problematic has been extended to data processing itself and generalized to support hardware accelerators. It takes benefit from GPUs and FPGAs, fully bypassing main memory and the CPU bottlenecks encountered in traditional processing flow.

The main contributions of this work can be outlined according to the chapters. After this introduction, the remainder of the memoir is structured as follows:

- The chapter 2 *State of the Art: Data Transfer & RASHPA*, gives a general overview of the key technologies involved in high throughput data transfers. We present the flaws of standard hardware and software and expose RDMA alternative capable to perform direct data movement from detector head to computing unit.

  Then we present today available massively parallel processing hardware such as GPUs that are ubiquitous in High performance Computing (HPC) and FPGAs as well. These are emerging as a new competitive technology in the field of ultra low latency data processing.

  The key concepts and philosophy behind RASHPA in its primitive stage are presented. Then we introduce our main contribution, the concept of RASHPA Pro-

cessing Unit (RPU), that supplements the already existing RASHPA Receiver (RR) with data processing capabilities.

- It is followed by chapter 3 *RASHPA Data Source Simulators*, where we perform the assessment of RDMA Over Converged Ethernet (RoCEv2) as a RASHPA-compliant protocol and describe the chosen approach to integrate a processing unit into a RASHPA system. We propose to add two new core components to the existing framework: i) a scheduler application in charge of event handling and recurring configuration, ii) an address translation system geared towards DMA engine of the data transfer in the RNIC of the receiver.

  Adequacy study and performance measurements lead us to the development of detector simulators as proof of concept. First version is a full fledged RASHPA compliant mockup. It is targeting mainly the development of a code base hiding as much as possible the programming details related to RDMA transfer.

  In addition, two FPGA-based RASHPA simulators are also presented as a feasibility study: i) a REMU-PCIE version, leveraging data transfer over Peripheral Component Interconnect Express (PCI-e) long distance links and performing its assessment, i) a REMU-ERNIC version, evaluating a recently available Xilinx IP (ERNIC) featuring RoCE protocol, and comparing its performance with an ESRF custom IP.

- Then chapter 4 *Online Accelerated Data Processing using RASHPA*, presents several versions of RASHPA data processing units: i) multi-threaded/vectorized application running on standard CPU that serves as a reference platform, ii) GPU accelerator, using Compute Unified Device Architecture (CUDA) on NVIDIA board or OpenCL kernel on AMD board, iii) configurable FPGA device, featuring ultra low latency data processing by programmable IP block developed by applying High Level Synthesis approach.

  The synchronization of data processing at the end of RDMA transfer is discussed and implemented accordingly.

  The assessment of a comprehensive acquisition chain is exemplified by online data analysis such as found in serial crystallography setup, featuring raw data pretreatment, rejection and compression algorithms.

- Finally chapter 4 concludes our work, summarizes our contributions and provides an outlook:

  i) Acceptance by the end-user of a new and complex framework might be improved with a better integration to more widely recognized parallel computing framework such as ESRF internally developed Lima [47], or Python Numpy scripting language which is very popular in the scientific community.

  ii) The direct data transfer to remote SSD, so called disaggregated storage, has gained momentum recently, as NVMEoF and RDMA technologies are well fitting together. The RASHPA philosophy being sufficiently generic, it will apply naturally to the crucial challenges of data persistence.

Chapter 2

State of the Art: Data Transfer & RASHPA

Contents

This chapter outlines what we learned from previous contributors to the field. As illustrated in Figure 2.1, this work is related to multiple and rather independent topics. In each domain, it is often dealing with unusual or corner case developments. We had to dig into cumbersome or not well documented features of API, encountered numerous issues caused by utilization of not yet mature technologies and have been struggling to get the advertised features working. Our main challenge has been devising generic components without relying on specific features of proprietary technologies.

These topics are mainly related to computer sciences, high performance networking and parallel processing, especially leveraging GPU coprocessors.

At first, an overview is given of the RASHPA Framework in its initial state at the beginning of this work. Then we present the methodology applied to perform the assessment of RDMA Over Converged Ethernet (RoCEv2) as a satisfactory RASHPA protocol and subsequently how has been implemented data processing support into the framework.

We then present some considerations on Linux operating system internals related to memory allocation, real time scheduling and high performance computing relevant with this work.

In this dissertation, the review of parallel algorithms is kept to the minimum. The algorithms are related to X-ray crystallography but the proposed implementations are simplistic as we are focusing on data transfer and synchronization mechanisms, rather than on data analysis itself. At this stage of the project, the only thing that matters, is the timing requirements which must remain compatible with the acquisition frame rate.

The rest of this chapter provides a presentation of the general concepts & philosophy behind the RASHPA framework that is the foundation of this thesis. Here is defined the working principle of a RASHPA Processing Unit.

2.1 High-throughput Networking

This section comprises discussions on the origin of the multiple bottlenecks encountered during data transfer. Figure 2.2 give an overview of the different devices and relative throughputs of the interconnects implemented in a typical acquisition system.

Data transfer bottlenecks may appear in multiple locations along the data path. They might be related to capacity of the links, to the hardware device in charge of the data transfer itself or to the software in use to perform the data movement. The affected data transfer throughput will obviously be capped by the slowest one.

The potential issues in the list presented below will be described and it will be specified which one are addressed by our contributions:

- The bottlenecks inside the detector electronics are not discussed in this work. This is the purpose of the readout electronics embedded in the detector side to solve these issues and for the rest of this project, we assume that the detector is able to push data on the network links at its full capacity. The actual work performed to design suitable RASHPA compliant detector electronics, performed by other contributors of the European Synchrotron Radiation Facility (ESRF), will not be described in this thesis [32].

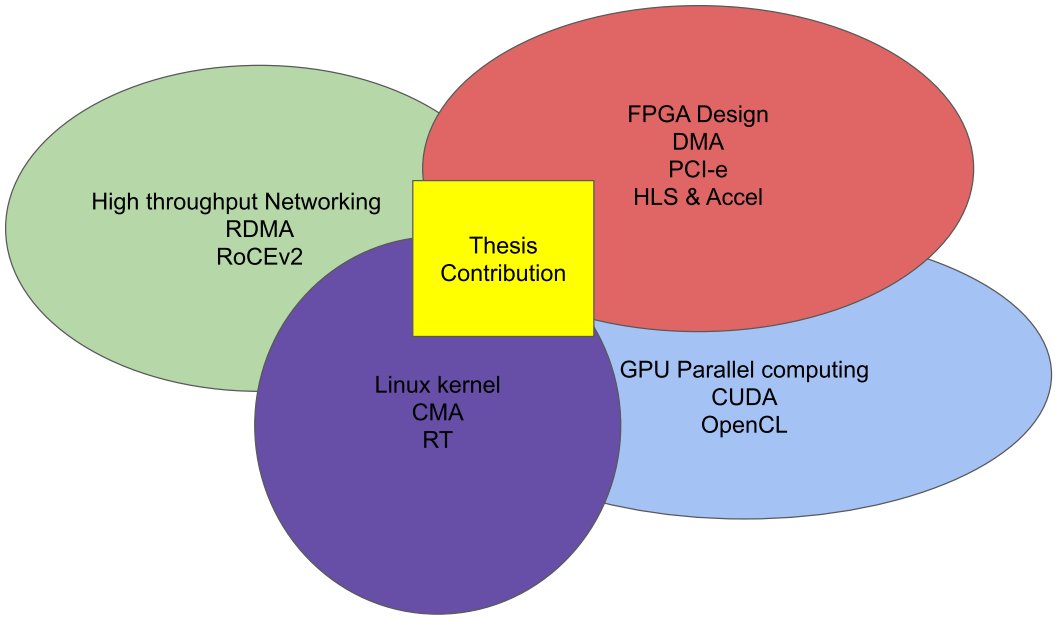


Figure 2.1: This thesis embraces multiple research fields and topics of diverse technologies.

- Contention in network links from detector to the remote computing unit is also out of scope for this work. A RASHPA system is intended to have its own network infrastructure. If network infrastructure is shared with different applications and computers, it is possible to tune the network devices (switches) and reserve the required bandwidth for the data acquisition system. Using modern switches, Quality of Service (QoS) is configurable and neither packet loss nor contention will occur in a lab environment. However a lossless network is mandatory for proper RDMA transfer[1].

- The transfer from a Network card to the CPU memory by user application is the main bottleneck. Historically, due to the complexity of operating systems (OS), multiple data copies were required between driver running in kernel space and application in user space [33]. Handling the interrupts coming from the NIC board by the CPU was inefficient as well. These issues are addressed by RDMA techniques that are used extensively in this work. The highest bandwidth network cards commercially available are using this techniques. However, it is worth to note that many progress have been done in the Linux kernel design and networking driver as well.

- A CPU application fetches data and code from main memory. This uses the memory channels of the processor, which have various performances depending on the access pattern. Accesses to subsequent and properly aligned data (called coalescent) are

[1] https://community.mellanox.com/s/article/understanding-qos-configuration-for-roce

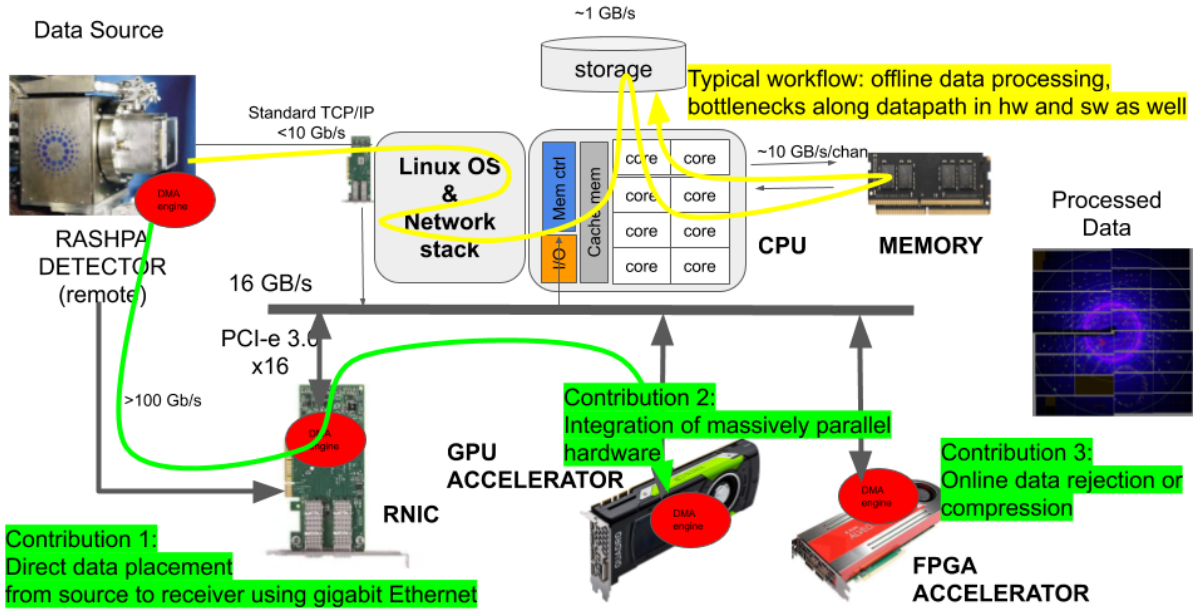


Figure 2.2: An overview of hardware in use and the general working of the investigated use case. A remotely connected X-ray detector by Gigabit Ethernet links produce data to a computing unit. On top is presented the standard solution: offline data processing by CPU. The proposed solution, below, features RDMA network card that offload CPU from data transfer and bypass software bottlenecks. Massively parallel hardware accelerators perform the online data analysis.

most efficient than random accesses to scattered data. It also greatly depends on the hardware topology in Non Uniform Memory Access (NUMA) systems as presented in [43]. Good CPU affinity and coalescent accesses are the key points of an efficient data processing.

2.1.1 Bottlenecks Related to Memory Management

We will at first consider the bottleneck issues inside the computer. Subsequently, we will deal with the network connected devices.

There is many different memory hardware in modern CPU design. The Dynamic Random Access Memory (DRAM) is used as main storage for data and instructions. It features high density storage but requires periodic refresh that limits its performances. Static Random Access Memory (SRAM) are found in Level 1 or 2 CPU caches. They are much more efficient in term of speed than DDR but more costly in term of power consumption and they require more transistors in the silicon chip. They are therefore available in more limited quantity[2].

[2]one DDR cell: one transistor by bit, one SRAM cell: 6 transistors or 4 + 2 resistors

All these kinds of memory are used in the computer architecture. There is an access hierarchy, based on their relative speed and capacity as shown in Figure 2.3. Most frequently accessed data are stored in *cache line* of 64 bytes in high speed *memory cache*. These caches are embedded in each processor core (Level 1) or shared by multiple cores (Level 2). Caches are automatically flushed/invalidated when they are full or when the associated physical memory changes as in case of DMA transfer. Maintaining cache coherency takes time.

In addition, for robustness and security reasons, Linux operating system implements a sophisticated virtualization machinery to perform memory addressing. Data and instructions are referenced by *virtual addresses* that are translated to *physical addresses* by a Memory Management Unit (MMU) on the basis of four look-up tables, and cached in a Translation Lookaside Buffer (TLB). All processes are virtually executing at the same address, i.e. 0x80000000 in Linux system, but obviously, code and data are actually stored at different physical memory locations. A process has only access to its code and data segment, but not directly to other processes nor to kernel memory space. The mapping between virtual address and physical memory is made on the basis of *pages*. These pages are usually 4 KB long. The page location might be transparently changed by the Memory Management Unit. Memory pages are reputed *movable* when they are not explicitly *pinned*. Thus the memory management system can mitigate memory fragmentation by the aggregation of large chunks of contiguous free space.

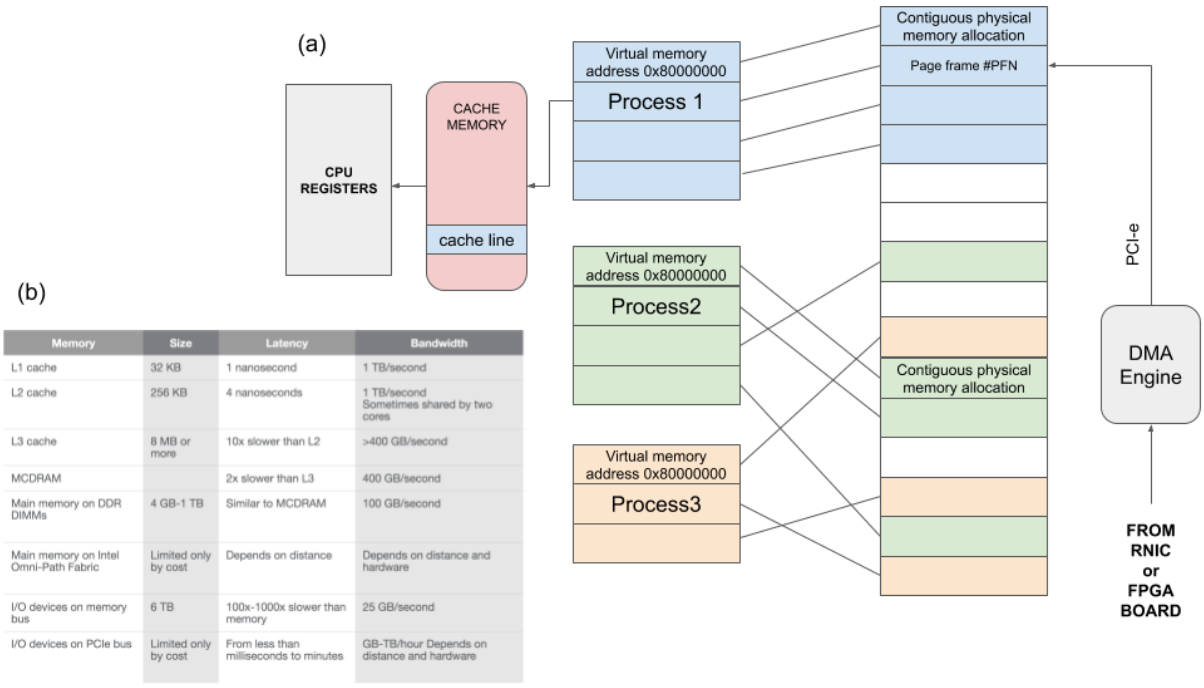
Processors have at least two modes of operation referred to as rings, *user* and *supervisor*, which change the available instruction set in a given mode. It is allowed only in supervisor mode to perform Input and Output (I/O) operations or MMU configuration. One can go from user to supervisor mode only in a controlled way using interrupts or system-call. To ensure robustness of the system, user applications are denied direct access to supervisor space. Data buffers must be copied by kernel code to and from memory in *application user space* to and from *kernel module or driver code* memory space. Going from one run level to the other, called context switching, has a cost in time as it is storing register contents in the stack, and is restoring them later.

All this clever machinery, added to multiple translations involved in virtualization to translate from virtual to physical address, has a time budget that may not be negligible at high throughput. Figure 2.4 shows the different bandwidth measured during data transfer in memory.

Modern processors also feature rich sets of *vectored instruction* or Single Instruction Multiple Data (SIMD) applying once the same operation on multiple data such as Intel Intrinsics[3]. Application performances benefit automatically from these instructions generated by a properly configured compiler. Effective Vectorization with OpenMP SIMD is presented in [20].

The proposed system has to perform both smoothly and efficiently the foreseen data transfer while preserving virtualization, cache coherency and user / kernel space isolation. Bypassing the bottlenecks will require new approaches presented throughout this memoir.

[3]https://software.intel.com/sites/landingpage/IntrinsicsGuide/



Memory	Size	Latency	Bandwidth
L1 cache	32 KB	1 nanosecond	1 TB/second
L2 cache	256 KB	4 nanoseconds	1 TB/second Sometimes shared by two cores
L3 cache	8 MB or more	10x slower than L2	>400 GB/second
MCDRAM		2x slower than L3	400 GB/second
Main memory on DDR DIMMs	4 GB-1 TB	Similar to MCDRAM	100 GB/second
Main memory on Intel Omni-Path Fabric	Limited only by cost	Depends on distance	Depends on distance and hardware
I/O devices on memory bus	6 TB	100x-1000x slower than memory	25 GB/second
I/O devices on PCIe bus	Limited only by cost	From less than milliseconds to minutes	GB-TB/hour Depends on distance and hardware

Figure 2.3: (a) Memory Hierarchy in modern computers includes several levels of caches. Data migration and coherency is performed by specialized hardware. Memory Management Unit performed the address translation related to virtualization. Table (b) shows some numbers on bandwidth and latency.

2.1.2 PCI-e Interconnect

Peripheral boards and devices are connected to CPU and main memory by Peripheral Component Interconnect Express (PCI-e) which has its own specific data transfer limits.

The today standard, PCI-e gen 3.0, can carry up to 8 Gb/s per *lanes*. 1, 4, 8 or 16 lanes are generally available on a given device, depending on the processor capabilities. Hence a throughput of 128 GT/s could be expected for a PCIe gen. 3 device with a x16 width. A host processor (server versions) has generally 48 (Intel) or 128 (Amd) lanes.

Thus, PCIe gen3 x16 maximum bandwidth is approximately 126 Gb/s taking into account diverse overhead and data encoding. However, the observed throughput on commercially available hardware such as GPUs or FPGAs is capped around 12GB/s when using standard driver[4].

PCIe Root Complex ensures transfer from/to device to/from main memory or from device to another device (peer to peer PCI-e (P2P) transfer). PCI-e switches can route data transfer from one PCI-e device to another on the same switch, completely bypassing the CPU.

The inter-node transfer performance vary mostly according to CPU affinity as shown Figure 2.6 and this is especially important for RNIC as explained in [36]. In servers

[4]https://www.xilinx.com/support/answers/68049.html

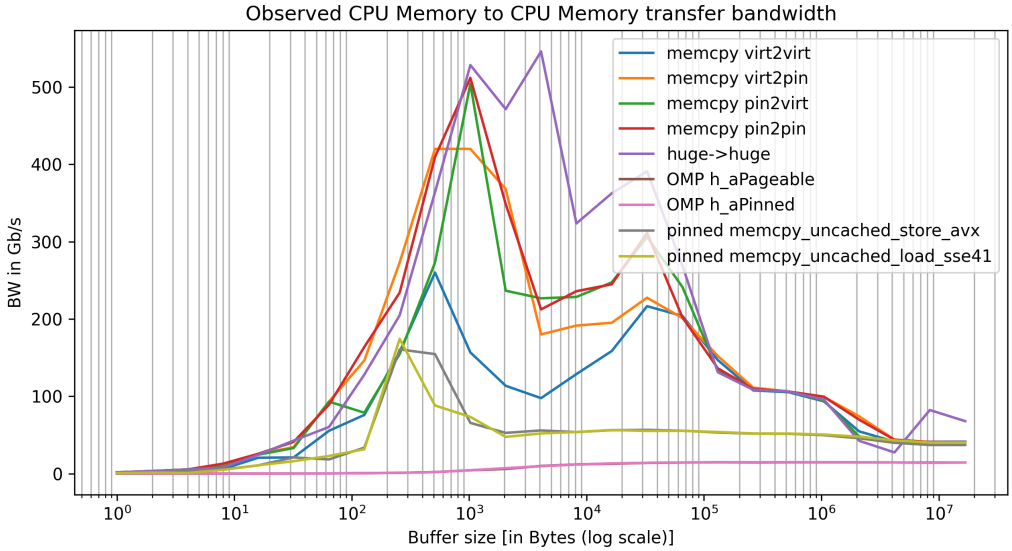


Figure 2.4: Observed CPU to CPU bandwidth by several techniques and memory type. Due to cache size effect, small size transfer from pinned memory are the fastest. The numbers greatly depend on the chosen hardware, the cache hit ratio, etc and are given only for comparison purpose on the same server.

with multiple processor sets (or even on some high end CPU), there are multiple PCIe interconnects, at least one for each CPU. As the inter-CPU set interconnect[5] is a severe bottleneck, it is wise to allocate memory on the same interconnect as the CPU core in use and the PCI-e device.

2.1.3 Direct Memory Access Overview

Direct Memory Access (DMA) is a key feature to perform high bandwidth transfer for PCIe based applications. DMA frees up CPU resources from being wasted for data movement and helps improving the overall system performances. A CPU performs data transfer staged in its internal registers, requiring two operations: loading from memory to register, then storing. It can generally perform 8, 16, 32 or 64 bit wide (aligned) memory access. However, using instructions from the vectorized instruction sets when available, the CPU could transfer packed data, up to 512 bits (Intel SSE, AVX512) and saturate the available bandwidth of PCIe interconnect.

But in modern computers, data transfer are mainly the task of DMA controllers (also called DMA engines). Those controllers are located in peripheral devices (PCI-e end point) and handle data transfer to and from the main memory without CPU intervention. The DMA engine might also be synthesized in custom FPGA design using standard IP

[5]Intel QuickPath Interconnect (QPI) or Amd Infinity Fabric

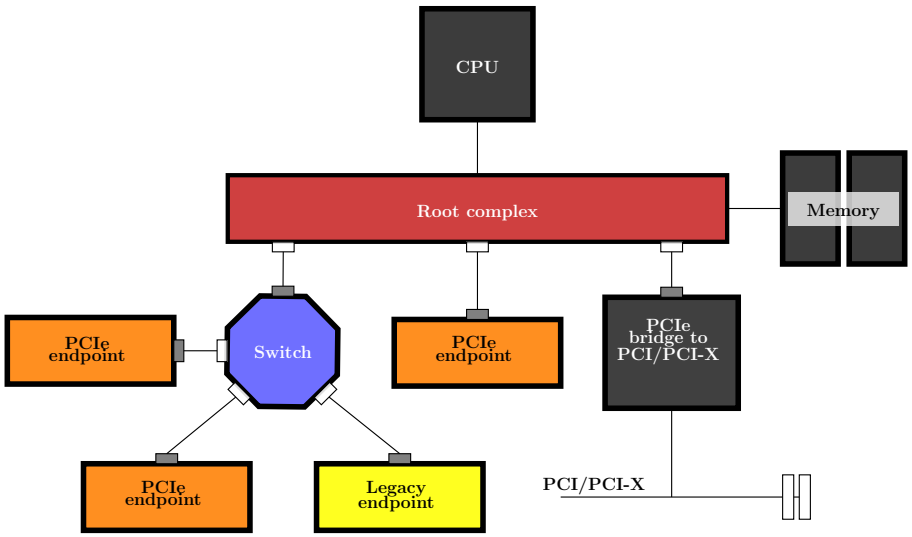


Figure 2.5: An overview of Peripheral Component Interconnect Express (PCI-e) topology showing PCI-e Root Complex and bridges (Source: Wikipedia).

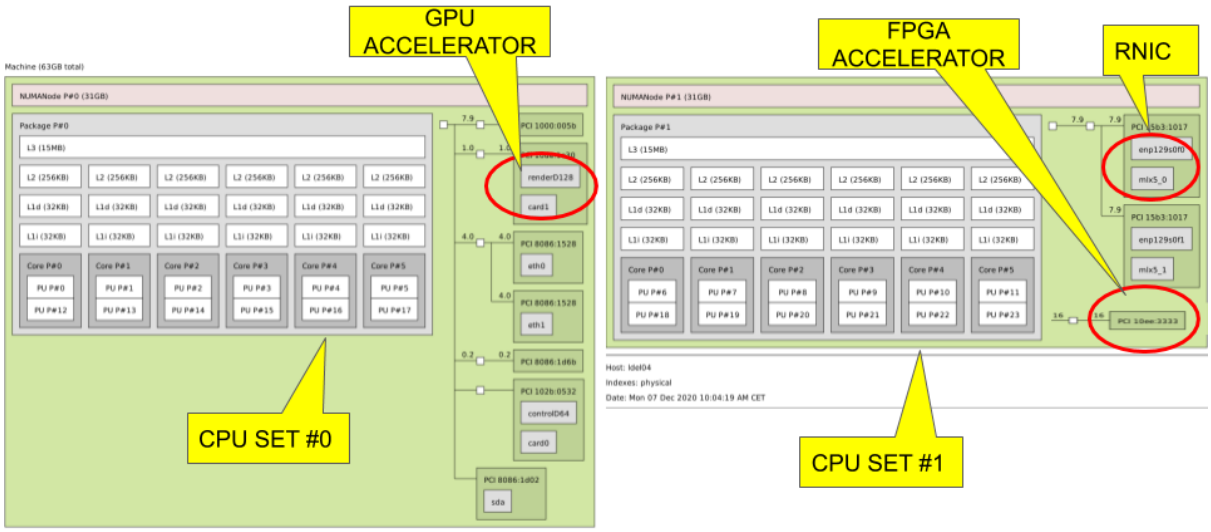


Figure 2.6: A snapshot from lstopo graphic user interface, showing a two CPU set computer and both PCIe-interconnects. Best throughput are achieved for data transfer from device to memory in the same interconnect and from PCI-e card to card with good affinity.

such as the Xilinx IPs CDMA[6] or the XDMA[7].

The software application must configure the DMA controller with a list of data transfer commands, mainly Buffer Descriptors (BDs), i.e. addresses and length of the memory regions, both at source and destination. BD processing proceeds in three successive steps:

i) The memory buffer must be pinned to prevent the memory management unit from moving it during a DMA transfer. This can take some time. ii) Then the virtual addresses of BDs given by the main application must be translated into their respective physical addresses used by the DMA engine. iii) And then, the data movement operations are performed autonomously through the PCI-e interconnect.

Physical addresses of recently used BDs might be cached internally for later reuse and performance reasons, .

Taking all this in consideration, it is important to optimize the duration of the data transfer. An efficient online data analysis essentially depends on a judicious memory allocation strategy. How to do this depends on the size of the data, their actual location, the targeted destination, the cache states, the capabilities of the DMA engine involved.

2.1.4 Memory Allocation Challenges

In the process of this work, we had to solve the problem of large memory buffer allocation suitable with RASHPA concepts and compatible with the Linux underlying operating system. Large memory allocation has always been a challenge for operating system designers and developers as explained in [42].

Here are some existing techniques on Linux based systems which include:

- The Reserved Memory regions are not visible from the Linux Operating System. By definition, this memory region is non-movable and does therefore work well with DMA. Then, the issue is to make the data from this memory region accessible from a normal user-application. With earlier Linux kernel, it was possible to make it accessible by performing a memory mapping of the pseudo-device */dev/mem*. This was the method foreseen in the initial RASHPA design. For security reasons this is not possible anymore when using recent Linux kernel version.

- The regular on-demand user-memory allocation procedure by *malloc*. Those memory regions are scattered in 4 KB pages in physical memory that can be migrated during memory management operations to different physical locations. Therefore, they must be pinned to be suitable for DMA. This takes time and requires a huge number of different descriptors in case of multi-GB allocations. There is an upper limit at 4 GB (set by OS rlimit).

- Allocation of memory backed by 2 MB or 1 GB pages, called *Huges pages* instead of regular 4 KB pages is possible on high-performance workstations. But a single

[6]https://www.xilinx.com/support/documentation/ip_documentation/axi_cdma/v4_1/pg034-axi-cdma.pdf

[7]https://www.xilinx.com/support/documentation/ip_documentation/xdma/v4_1/pg195-pcie-dma.pdf

multi gigabyte allocation in Huges pages is not guaranteed to be contiguous. That memory is pinned by the operating system during allocation.

The last two methods are performing memory allocation from user space. But it is not possible to get the physical addresses required for the DMA operations from user space. A kernel module must be implemented on purposely, performing a call to the function *get_user_pages*.

- Virtual memory might be allocated from kernel space using *kmalloc*. But it is reserved for small size allocation that does not match our use case.
- Contiguous Memory Allocation (CMA) from kernel space was introduced in the Linux kernel 3.12 precisely for this purpose [61]. A large pool of memory specified at boot time is eligible to CMA allocation but when it is not in use, it remains available by other non-CMA application. Such an allocation is done by a kernel module using *dma_alloc_coherent* system call. It provides also the physical addresses needed by the DMA engine. The proposed method is leveraging the Contiguous Memory Allocator (CMA) which is present on purposely in the Linux kernel.

2.1.5 Limitations of Conventional Networking Techniques

The Internet Protocol (IP) dates from the launch of the internet in the early 1970's [17] and was continuously upgraded until now. This library stack worked great and still does today on standard hardware. It has been constantly optimized: interrupt coalescing technique consolidates multiple hardware event in a single one, large buffers may be passed by reference between kernel and user application, parallel or vectorized I/O such as *recvmmsg* are allowing to receive multiple messages from a socket using a single system call, receive side scaling dispatch data transfer on multiple processor cores, etc...

But the demanding requirements of ultra-fast data links prevent to use it as is as stated in [16] that reports 33Gb/s maximum achieved when using UDP on a 100 gigabit Ethernet links.

Our use case in high-performance detectors has a non standard characteristics. A typical readout electronics monitors the photon sensors and transfer acquisition data at high speed into its internal memory. Such an embedded system uses Field Programmable Gate Array (FPGA) with a limited power budget. This is in particular the case for the FPGA of the family endorsed for this purpose. Such FPGAs are not well tailored nor have the sufficient memory resources to implement high-level network protocols such as the transmission control protocol (TCP). Indeed, TCP requires complex software stacks, features of sophisticated error handling and automatic data re-transmission as shown in Figure 2.7.

Therefore, detector electronics generally implements less sophisticated, single-sided data transfer to the computing unit, such as the User Datagram Protocol (UDP). We will restrict our approach to UDP data transfer in the rest of this thesis.

However, even using UDP with the traditional software library, the well known *Socket API*, packet losses might be experienced. Those dropped packets are not lost during the transmission phase but along the path within the operating system. This is a well known

flaw [50], related to the inner complexity of the network stack and many time-consuming tasks executed under the hood: data copies from/to the user and kernel memory space, complex interrupt handling and processor context-switches, etc. Fine-tuning of the operating system, e.g. increasing internal buffers, carefully allocating interrupted handling and disabling the Linux kernel from pre-emptively scheduling tasks onto the core dedicated to the receiving task, enables higher rates to be achieved [33]. However, the result does not scale well above 10 Gb/s.

Such a setup has been implemented in the field of X-ray science for the SLS detector software [18] and is successfully tested with a PSI Eiger 500K detector featuring two 10 Gb/s Ethernet links. It successfully aggregated 5 Gb/s on each link.

The data plane development kit (DPDK) [23] is an interesting framework which consistently implements these principles The whole application code runs in user space to achieve low latency. It is kept busy polling the status of the NIC. This works well with standard network cards and even with RNIC adaptors, at the price of a fully dedicated core devoted to the network traffic.

Below is presented a review of network solutions encountered on commercially available detectors:

- Detector with proprietary hardware/protocol: Some vendors provide proprietary solutions featuring a *black box* computer as interface. The vast majority of available solutions leverages Ethernet links with custom protocol. For reference, the theoretical maximum speed is 148 Mpps using 100 Gigabit Ethernet. The main drawbacks is that packets cannot be routed by standard switches.

- Detector producing UDP data flows such presented in [40]: Datagram are sent without reception acknowledgement from the receiver. The protocol does not ensure neither packet ordering nor lost detection that must then be handled separately and explicitly. The data flow can be routed using standard switches.

- Detector producing TCP data flows: This is a connected protocol where sent packets are acknowledged at the destination. It generates an extra load on the links. In addition to that, to ensure the possibility to re-transmit lost packets, a copy of the sent data must be stored for some time. That requires memory and complex processing algorithms. TCP is therefore rarely used with real detectors.

- ZeroMQ Sockets: ZeroMQ, [8], is an optimized socket library originally developed for high-frequency trading applications when microseconds matter. It has since the beginning largely been reused by the scientific community. It relies on TCP sockets at low level and therefore seems hardly suitable to the embedded electronics of a detector. However, detector vendors sometimes provide a black box computer that produce ZeroMQ data flow as Eiger Dectris presented in [34].

  ZeroMQ defines several transfer patterns optimizing the data communication processes between the stakeholders. With REP/REQ peer to peer pattern, a single

[8] https://zguide.zeromq.org/

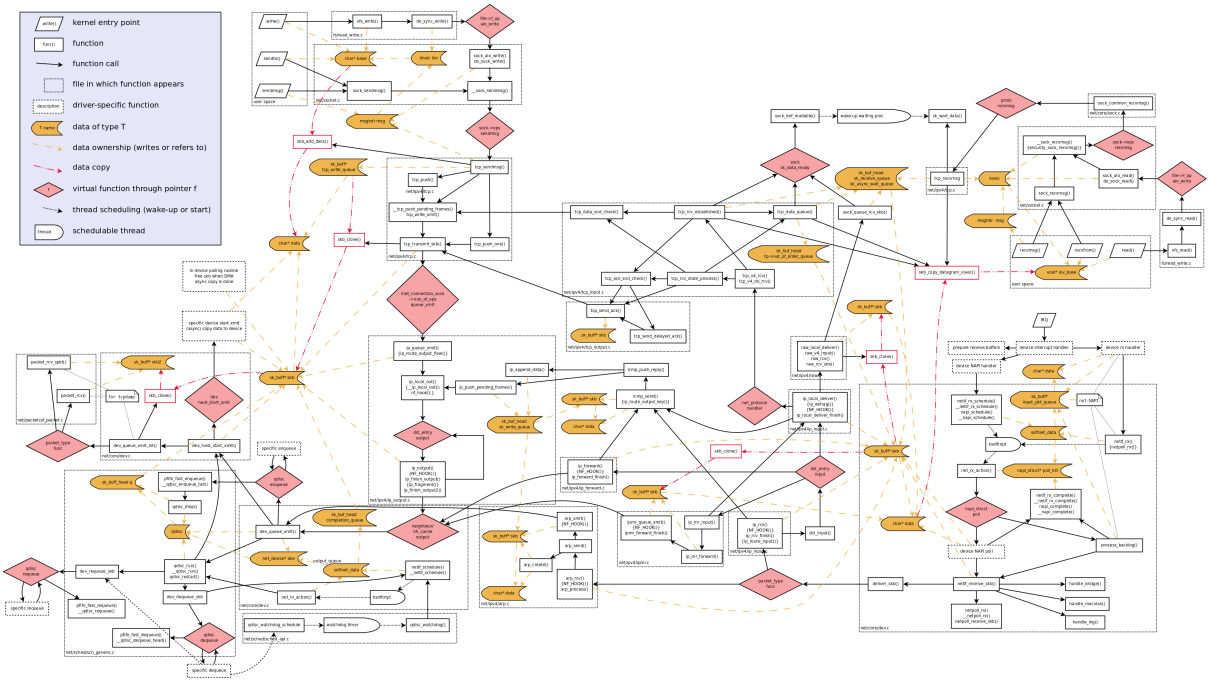


Figure 2.7: Packet journey through Linux kernel shown here to exhibit its internal complexity. It is only on an overview of the Transport Communication Protocol (TCP) State Machine and error handling mechanism. This techniques does not scale much at high throughput and therefore TCP becomes inefficient. (Source: Linuxfoundation.org)

response is given by the peer to the requester. PUB/SUB implements the publish/-subscribe model while PUSH/PULL performs a round robin amongst the connected sockets. The last pattern was found slightly more efficient in our measurements. Interestingly, a development of an UDP version of ZeroMQ, that was in a early draft stage at the beginning of this project, is in progress. It operates with the RADIO/DISH pattern, i.e. one way broadcasting[9]. It has not yet been evaluated if this protocol could be supported in a detector with the aforementioned constraints.

As far as networking is concerned, the rest of our work will relies on data transfer produced by a FPGA based design, producing UDP/IP datagram and received in data processing server by commercially available RNIC.

2.1.6 Overview of RDMA Techniques

RDMA is the generalization of DMA between remotely connected computers equipped with dedicated NIC such as shown Figure 2.8 or FPGA boards embedding high speed NIC. Dating from mid 1990, one of the first working implementation used in a Top 500 HPC computer is presented in [7]. The distinct advantage of this data-transfer solution

[9]http://api.zeromq.org/master:zmq-udp

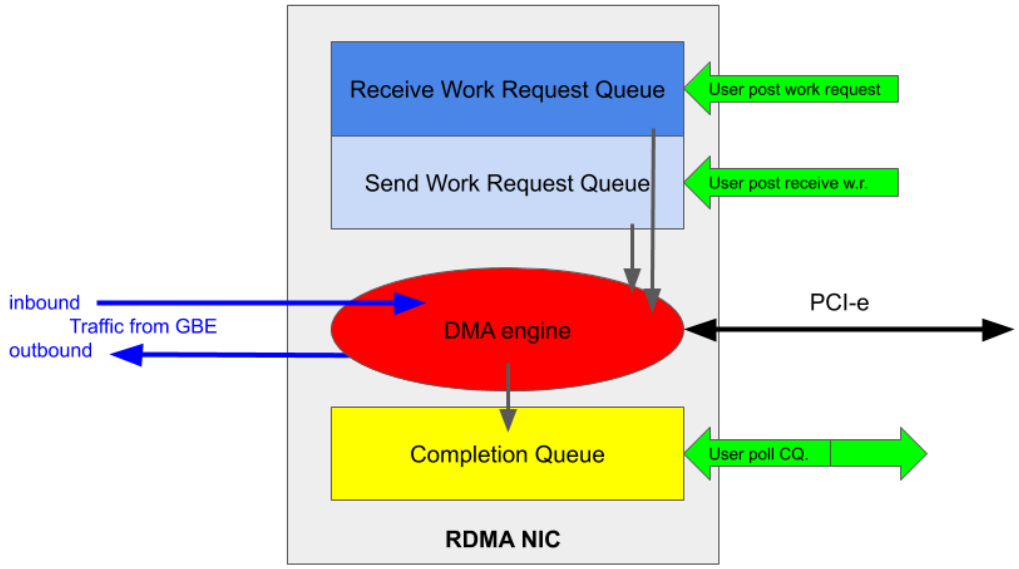


Figure 2.8: General procedures of an RDMA network Interface Card (RNIC). The user application posts asynchronously Work Requests for Receive Or Send operation. Those requests are processed one by one by the RNIC hardware: for Receive operation, when data arrives, for Send operation immediately. Work Completion Queue is updated with the status of the last operation.

is to bypass the CPU and operating system and move ingress data directly to their final destination. Several RDMA variants are available but only a few of them are compatible with the requirements for high performance over long-distance communication.

Infiniband (IB) from Open Fabrics Alliance is promoted mainly by Mellanox Technologies (now Nvidia). It was the first implementation of RDMA commercially successful and widely used in HPC field up to now. It has a cost as it uses proprietary hardware: RNIC, switches and cables featuring built-in loss less transfer capabilities.

Other solutions are available using commodity Ethernet. The internet Wide Area RDMA Protocol (iWARP)[10] is built on top of the TCP/IP stack to ensure lossless transmissions on the Ethernet. We did not evaluate this solution as it relies on the TCP stack, which seemed incompatible to us with the capacities of detector electronics that were being developed at that time. Interestingly, there are preliminary studies of Grant et al. and Lenkiewicz et al. of a UDP/iWARP implementation which deserve further investigation [14] [28] .

[10]https://www.chelsio.com/nic/rdma-iwarp/

Table 2.1: RDMA variants and compliance with the RASHPA framework.

Variant	**iWarp**	**Infiniband**	**RoCEv1**	**RoCEv2**	**custom**[1]
protocol	TCP/IP	IB	Ethernet	UDP/IP	Ethernet FPGA + MAC IP
RASHPA compliance	TCP tricky to embed	requires special hardware and links	not routable	OK	cost lack of software resources

[1] requires dedicated FPGA board at destination

2.1.6.1 RoCEv2 Assessment in the Frame of the RASHPA project

RoCE is an alternate solution that makes IB compatible with the already deployed Ethernet infrastructure. There are two implementations of RoCE: RoCEv1 is nonroutable by standard Ethernet switches, while RoCEv2, which encapsulates the RDMA payload in a UDP/IP datagram, is routable at the price of a slight overhead. The vendor roadmaps favor a wider and general adoption of RoCE. This technology received considerable interest, because it is promising RDMA performances and seamless integration into existing infrastructures as stated by Eitan in [10]. Miao et al. proposed SoftRDMA, which is a software implementation of RoCE on a standard NIC that may be valuable for low-cost solutions or for research and development purposes in [38]. An overview of RDMA can be found in the work of Romanow et al.[54]. Performances and best-practice studies can be found in the work of MacArtur et al.[31]. Tsai et al or Wang et al. have investigated RDMA performance in the High Performance Computing (HPC) field [63] [64] at research facilities. Mohr et al. performed a comprehensive evaluation of the upgrade of the fast-acquisition system at CERN and showed the key advantages of the technology [39].

RoCE technology matured very recently and this has caused some delay in the course of our work, since at the beginning of this work, the Mellanox OFED stack was not yet available for the Linux distribution/kernel version used at the ESRF. Mellanox developers were providing patches and help porting the driver to the kernel in use. This improved greatly over time as the Mellanox code was integrated in the upstream Linux kernel. This kind of issue is a major drawback for long term projects as are detector developments. Engineering teams always fear the potential obsolescence of not well established API and often prefer solutions based on proven building blocks.

Table 2.1 describes the different protocols and API available for the developer.

Our work is based on RDMA programming using the *rdma-core* libraries[11] that are the userspace components for the Linux Kernel's drivers/infiniband subsystem. The code is split in several libraries but we used only the one called *ibverbs*. IB is for Infiniband, but RoCE and iWarp are also using the same verbs. We did not use the *rdmacm* library for Communication Management that provides low level configuration mechanism of the RDMA transfer. The RASHPA framework has indeed its own communication manager.

[11] https://github.com/linux-rdma/rdma-core

We did neither not use higher level libraries such as *libFabrics*[12] or *OpenUCX*[13] largely used in HPC application. As a reminder, at detector side, there is an FPGA based design, featuring an unidirectional data link that prevent the implementation of sophisticated communication stack.

The main programming concepts and definitions from the developer point of view are presented below:

- i) The Queue Pair is a data structure describing the transport protocol. There are three main types of Queue Pair: Unreliable Datagram (UD), Unreliable Connected (UC) and Reliable Connected (RC). Park et al. present a review of transport and their distinctive characteristics in [45]. *Unreliable* means a single-sided transfer without guarantee of the data delivery. *Connected* means peer to peer (P2P) communication, always from the same source to the same destination during the life of the queue pair.

  A *Reliable* QP produces acknowledgement of packet reception and re-transmission of lost packets.

  Only the UD queue pair supports multi-casting to beforehand registered receivers. This technique could be possibly used in a future extension of our work to support a generic event concept that could be pushed to multiple receivers without overhead.

  In normal configurations UD queue pairs are P2P as the other types. Consequently, when it is desirable to send data to multiple destinations as required by our project, we would have to send data multiples times

- ii) The operations on Queue Pairs are called Verbs, because only their functional aspects are defined. There is not any precisely defined Application Programmer Interface (API). In practice, the *rdma-core* code implementation serves as reference API. Several verbs exist: WRITE and SEND, and also Atomic operations not used nor described here.

  WRITE and SEND differ only in the way destination address is handled: when using WRITE operation, the destination address is embedded in the datagram. While using SEND operation, it is up to the receiver to set the destination address.

Up to now, to the best of our knowledge, there are not yet commercially available detectors implementing any kind of zero copy techniques. The Brazilian synchrotron presented a project for the PI mega detector in [3]. It relies on RoCEv1 protocol. With this version of RoCE, the Ethernet datagram are produced in a custom format that is not routable by standard switches.

However, we are confident it is feasible to exploit RoCE features in the context of a data acquisition system compliant with the RASHPA Framework.

On the receiver side, it would be possible to implement a dedicated FPGA board, performing tunneling between Gigabit Ethernet RoCE and PCI-e interconnect. But,

[12]https://ofiwg.github.io/libfabric/
[13]https://ofiwg.github.io/libfabric/

for economic reasons regarding hardware and man power costs involved in the development of such a card, a dedicated RNIC such as provided by Mellanox/Broadcom/Marvell companies is the easiest solution.

On a detector module deprived of PCI-e support, it would not be possible to implement such a NIC. However, an intellectual property core (IP) implementing a subset of the RoCEv2 core in FPGA was designed by [32]. It is not exactly a versatile RNIC as shown in Figure 2.8 because it has been purposely designed to perform detector data transfer foreseen in RASHPA specifications. Performance evaluations are very good as it can reach peak bandwidth. It implements a parallel calculation of the RoCE invariant check redundancy code that would be costly otherwise. To our knowledge, it was the first RoCEv2 IP publicly available until the recently announced Xilinx embedded RDMA-enabled NIC (ERNIC IP[14]). A system on a chip or a dedicated integrated circuit may also be considered to implement the RoCEv2 protocol as in the work of [15] or such as the commercially available BlueField chip from Mellanox Technologies [35].

The Figure 2.9 show several datagrams of verb and their distinctive features.

2.1.6.2 Messaging Accelerator Library

The *libVMA* is a software library developed by Mellanox Technologies for their RNICs. This library is able to intercept and preempt any system calls from the application software to the IP stack and transparently replace them by DMA transfer as shown in Figure 2.10.

It is more or less a transparent substitution of the legacy socket API called by RDMA socket using the rsocket API [59]. The *LD_PRELOAD* Linux environment variable is used by dynamic library loader of the operating system to change the library in use. Thus, without rewriting legacy code, one can partly benefit of the hardware acceleration capabilities.

But to take benefits from all the features of RDMA, such as fully autonomous data placement at destination location, it is mandatory to develop a dedicated application. That contribution will be detailed in the chapter 3.

2.1.7 Prospects beyond DMA Techniques

Standard DMA engines are performing very efficiently data movements to main memory. But it is possible to further optimize the process by moving the data directly to the cache memory of the destination processor. This requires the management of the cache coherency between the different processors to work well and a special communication protocol for this purpose. A first implementation for Intel processor, called Direct Cache Access (DCA), was presented in [21]. Such optimized DMA might be more widely accessible through standardized approach such as the Cache Coherent Interconnect for Accelerators[15]. CCIX is already available on some ARM processors or ZynqMP SoC.

An extension of this concept is the Gen-Z interconnect, currently under development by major firms, including AMD, ARM, Broadcom, Cray, Dell EMC, Hewlett Packard

[14]https://www.xilinx.com/products/intellectual-property/ef-di-ernic.html#documentation
[15]https://www.ccixconsortium.com/

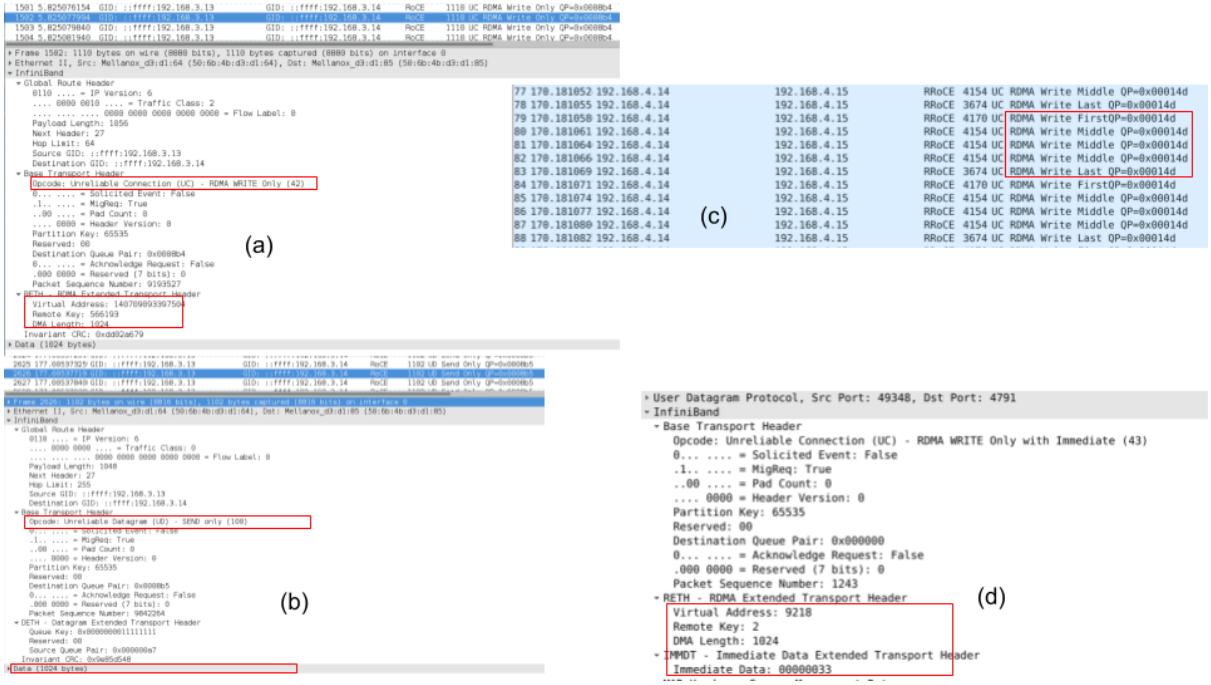


Figure 2.9: Snapshot of several RDMA verbs and their distinctive features. (a) WRITE Datagram on Unreliable Connected (UC) Queue Pair. The destination address is embedded in the payload with the data. The proposed RASHPA implementation uses this verb. (b) SEND Datagram on Unreliable Datagram (UD) Queue Pair. There is no destination address and it is up to the data receiver to set the destination address. (c) A large data transfer can automatically be split in multiple datagrams. Not used here. (d) WRITE_WITH_IMM Datagram on Unreliable Connected (UC) Queue Pair. The 32 bit immediate value is used to implement events in RASHPA system.

Enterprise, Huawei, IDT, Micron, Samsung, SK hynix, and Xilinx. This protocol is devised to attach remotely connected devices as shown in [19]. System on Chips (SoCs), memory, SSD, network cards, GPU accelerators, FPGA boards might be directly inter connected to the main System. For the sake of efficiency, the interconnect is done at the level of the cache-coherency system of the CPUs. With Gen-Z interconnect, the CPUs would access any of a pool of network-attached memory using the usual READ & WRITE semantics. The first hardware development kits are available but commodity devices are not yet.

A RASHPA system could be nicely implemented using such Gen-Z components.

2.2 Hardware Accelerators Overview

The advent of Moore's law in electronics is highlighted by the diversity and the success of the so-called hardware accelerators. These devices are similar to mainstream processors, but integrated in the same silicon chip in a ever growing number. There are many

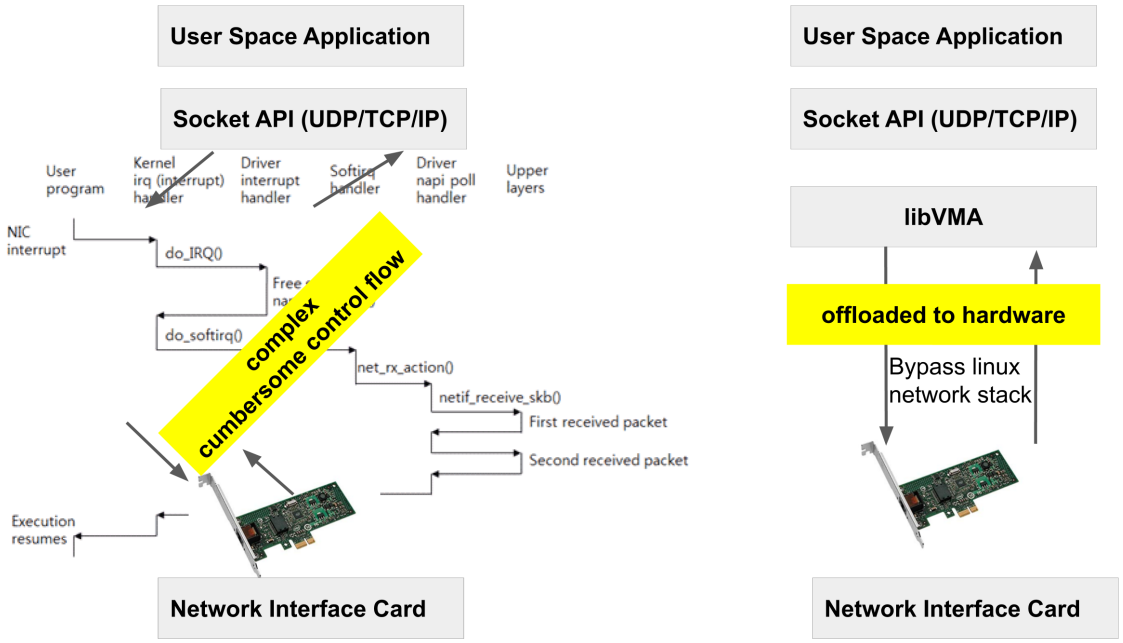


Figure 2.10: An application using LibVMA can bypass IP stack at run time. The executable code is kept the same: system calls to the Socket API are intercepted by libVMA. The actual data transfer are offloaded to the RNIC hardware accelerator. There are obviously some limitations on the performances.

kind of this devices, from many-core CPUs to GPUs or TPUs. We will not discuss exhaustively here all thes architecture and restrict to the well established solutions (GPUs and FPGAs).

The computationally intensive task of converting the raw data produced by images detectors and the subsequent online processing imposes the use of such massively parallel computing engines.

2.2.1 GPU accelerators

Graphic processing units (GPUs) are now routinely used for some application in high-performance and scientific computing computing due to their superior raw computing performances and ease of use.

These devices, originally designed for the gaming industry and 2D/3D image synthesis, took benefit from the mass market and now offer amazing processing power at low price. They have been quickly diverted from their initial target and exploited for highly intensive arithmetic data processing.

A GPU application is split in two parts: i) a host application executing on main CPU, performing sequential execution of the non parallel section of the algorithm, ii) the GPU accelerated kernels, performing the main data processing. Data movements must be scheduled between both devices at the right time.

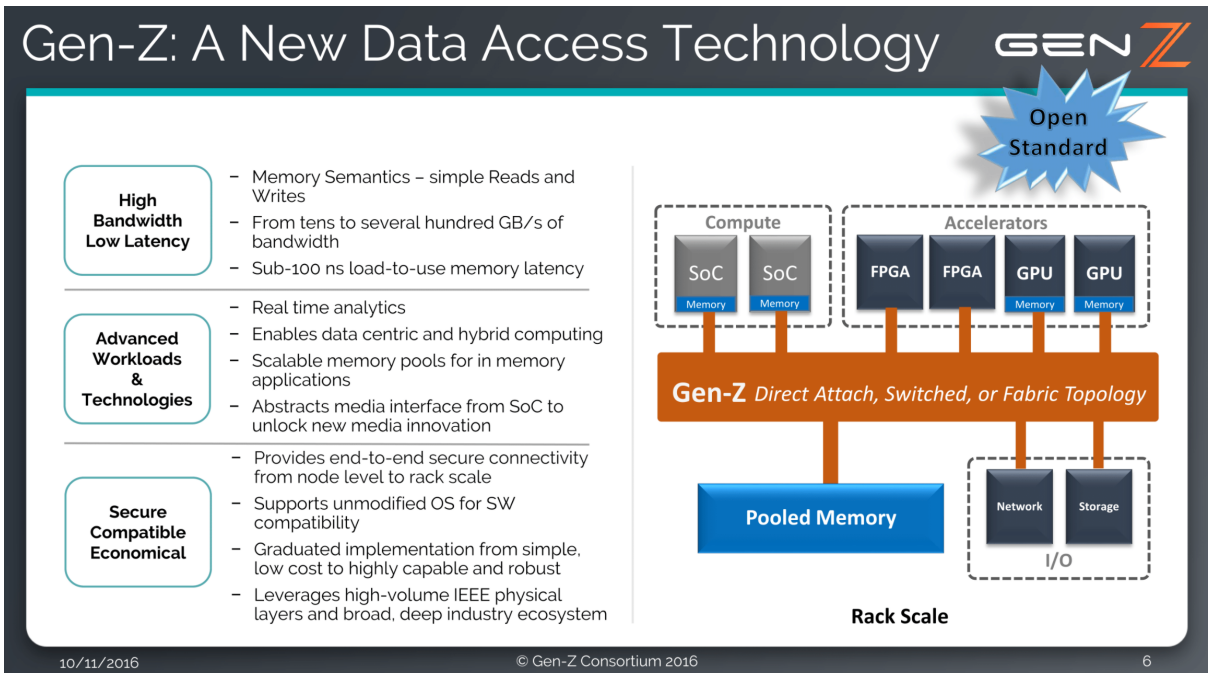


Figure 2.11: Gen-Z is a promising interconnect. It is based on a memory centric model and aims at attaching directly the remote resource to the cache controller of the processor. (Source: Genzconsortium slide)

A GPU board includes a PCI-e bridge interconnect, a fast GDDR memory, called global memory, two DMA engines, and several data processing engine called Streaming Multiprocessors (SM) with their own shared (cache) memory. Each of them is executing the GPU program, called kernel according to the Single Instruction, Multiple Thread model (SIMT). SIMT meaning that each instruction of a kernel is executed in groups of 32 parallel threads (called a Warp). Each thread of a given warp is executing the same instruction once at a time on different data. In case of conditional execution (divergence), only a part of the threads are kept activated. If there is a resource contention in a block, the scheduler can instruct the warp to execute another block. The overall computation is performed by multiple concurrent warps on the available resources of the SM. The data are arranged on a grid of 1, 2 or 3 dimensions. The grid is split in equal size blocks of up to 1024 threads.

The instruction set is more reduced than the one available on a processor geared towards desktop applications, but features generally integer and floating point arithmetic on 16/32/ or 64 bits. Commonly available GPUs feature tens of SM and tens of GB of DDR[16]. Thus, even if the GPUs are clocked relatively low compared to CPUs, they exhibit astonishing GFlops capability. GPUs are today already used for offline data processing. They are naturally the best candidates to manage data streams produced by demanding photon science encountered in many synchrotron radiation experiments.

[16]Nvidia Quadro RTX6000 24GB DDR6, 4608 Threads in 72 SM, 96KB shared memory per SM

However, to sustain this type of data treatment for a long duration, it is essential to transfer the detector data into the GPU memory continuously. The events required to synchronize computation to data flow must be triggered at the lowest possible latency.

Low-latency and zero-copy data transfer are exactly the aims of the RASHPA Framework. This work demonstrated its actual implementation on the GPU devices. We will show in chapter 4 how we have defined and implemented the generic concept of RASHPA Processing Units (RPUs) in the RASHPA ecosystem.

NVIDIA GPU hardware and CUDA programming tools are counting as 80% of the High Performance Computing (HPC) market and are driven by the actual growth of the Artificial Intelligence (AI) market. For this reason, focus has been put on Nvidia hardware. But AMD board has been also implemented in the experimental setup to check the adequacy of the proposed solution to multiple technical hardware and software platforms.

The main criteria for choosing the GPU programming model was the availability of fine control over the data movement between host and GPU.

High level approaches such as OpenACC (Open ACCelerators) and OpenMP (Open Multi-Processing) are programming standards for parallel computing. Both have support for GPUs. They supports C-like language programming driven by high level #pragma directives. The developer does not explicitly take care of the convoluted data placements between host and accelerator, that are managed transparently. But this is precisely such control over data movement that is needed in this project. Therefore our approach in not geared in its actual implementation, towards such high level programming pattern.

2.2.2 PCI-e P2P Transfer into GPU/FPGA Device Memory

A RNIC, capable of directly accessing the GPU memory, bypassing staged main memory, would be a key advantage in order to properly address the low latency challenges. This feature is called PCI-e DMA Peer to Peer (DMA P2P). Quadro and Tesla NVIDIA GPUs feature this technology called GPUDirect RDMA in NVIDIA jargon, which was developed conjointly by NVIDIA and Mellanox Technologies [60]). There are several version of GPUDirect, interlinked

It is available as a Linux kernel module called *nv_peer_mem*[17] and is presented in [2]. The process of integrating this glue code in a standardized way in the Linux kernel is undergoing but not yet completed. It is worth noting that it is only available with some classes of devices built for professional usage and closely tied to the CUDA runtime. Even on a compliant board, access to this technology is not possible from an OpenCL application. Gillert et al. explained that the closed source GPU driver and OpenCL runtime handle differently the memory allocation from CUDA or OpenCL, preventing the latter to use GPUDirect [13].

GPUDirect technology is only one among others features related to PCI-e interconnect. Hereafter are summarized the other findings in the literature about some promising capabilities:

[17] https://github.com/Mellanox/nv_peer_memory

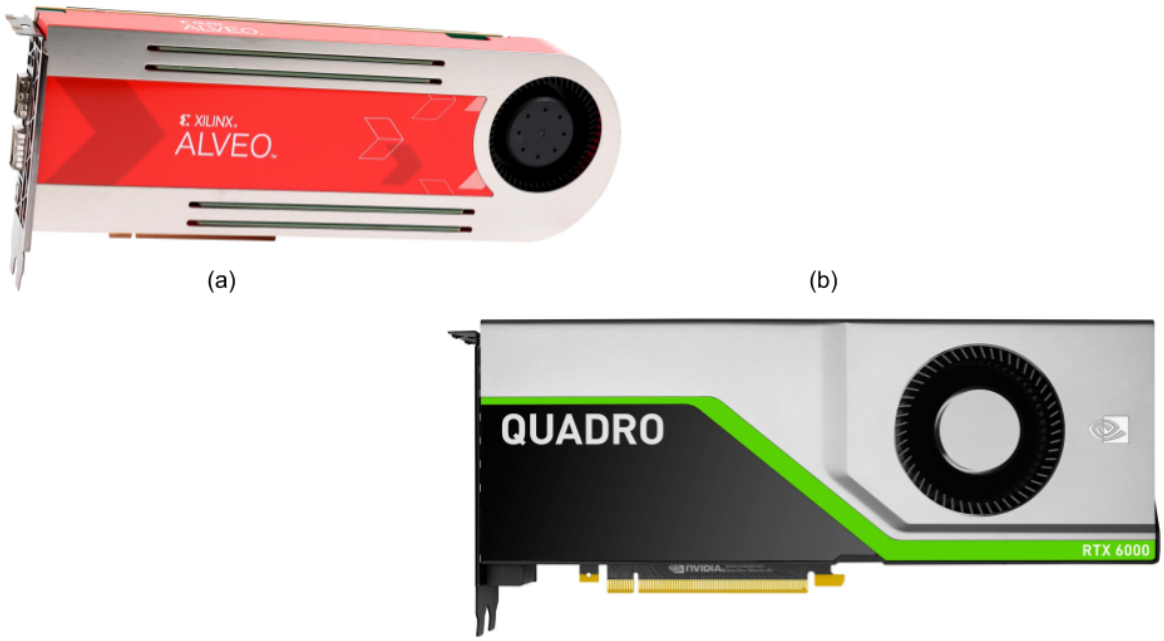


Figure 2.12: FPGA and GPU boards used as accelerators in our project: (a) Xilinx Alveo U200 64 GB DDR4, Virtex Ultrascale+, 2x 100 GBE, PCI-e gen3 x 16, (b) Nvidia Quadro RTX6000, 24 GB DDR5, PCI-e gen3 x16

- In [55] is presented the *gdrcopy* library and is shown how a host application can map and then perform read/write access to the GPU internal memory at high speed using Intel Intrinsics extended instruction sets (MMX, SSE or AVX).
- DirectGMA is the equivalent of GPUDirect for AMD Firepro GPUs. There exist slightly different flavors of this technology for more recent GPU boards. DirectGMA works quite the same way than the Nvidia counterpart and the software support is nowadays integrated in the RoCM (AMD open source) driver[18].
- D. Rossetti at al. have developed the GDASync library (GPUDirect Asynchronous) presented in [56] and [57]. That library enables a GPU to directly trigger data transfer in an RNIC. It is a promising feature but it cannot yet fulfill the requirements of this project. In this current implementation, the GPU can only trigger an RNIC in order to send egress data, but not in order to receive ingress data. In addition, the possibility for the RNIC to trigger directly the GPU is lacking at the time of writing.
- StorageDirect stands for direct GPU to SSD data transfer. It is the NVIDIA umbrella for the latter technology supplemented with RDMA NVMEoF techniques

[18]https://rocmdocs.amd.com/en/latest/Remote_Device_Programming/Remote-Device-Programming.html

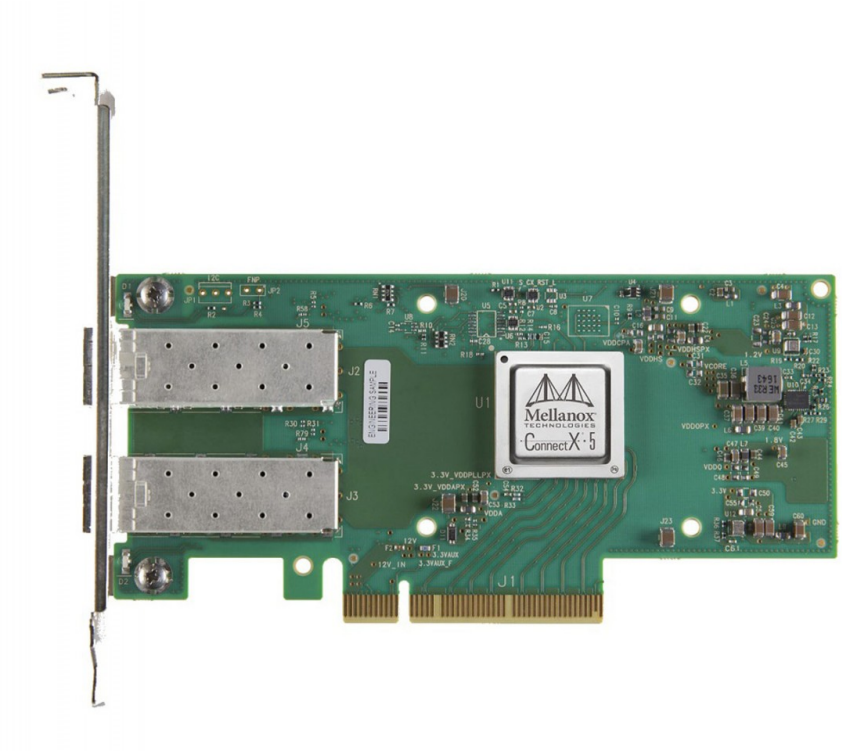


Figure 2.13: The Connectx-5 EN is a dual port 100 Gigabit Ethernet Remote DMA compatible Network card (RNIC) from Mellanox Technologies (now Nvidia).

and file system management. It is not yet publicly available. This could be used to extend the RASHPA framework to persistent storage unit. This point received considerable momentum recently and it will be mentioned in the outlook subsection 5.2.1 of the thesis.

2.2.3 Parallel Algorithms

As stated before, this thesis does not explore new solutions to mathematical or computational problems encountered in X ray imaging. The proposed algorithms are using standard approaches and they are presented in chapter 4.

In this section are exposed only the implementation details related to the RASHPA system. Kernels were written from scratch using CUDA language. Nvidia's offer for developers includes standard parallel code libraries such as Fast Fourier Transform (cuFFT), Linear Algebra Solver (cuBLAS), Sparse Matrix (cuSPARSE), etc The cuSPARSE library is intended for sparse matrix calculation but could not be implemented in the proposed code. A side issue was encountered during its evaluation: it has not been possible to obtain good performances using the built-in *dense-to-sparse* matrix conversion function. Actually, this library was highly efficient when used in the standard way. But it appeared that its internal code contains hidden memory buffer operations. Such operations

prevent concurrent CUDA streams to overlap when used concurrently in the processing pipeline. A memory allocation or deallocation force thread synchronizations, i.e. the code execution is serialized instead of being parallelized. As emphasized in the work of Yang et al., there are many other hidden synchronization disseminated in the libraries or programming patterns [67]. This is a real concern for those who want to perform real-time control using GPUs .

The proposed online data analysis relies on iterative launches of CUDA kernel. Decreasing task-launch latency was of crucial importance.

Dynamic Parallelism is the possibility to launch a kernel from another CUDA kernel, bypassing the usual host invocation. But unfortunately, it does not decrease the kernel launch time as stated by [9].

CUDA Streams were extensively used in this work. Streams are sequences of CUDA kernel or data transfer between host and GPU that can be executed in parallel. This way, data transfers can overlap computations. CUDA Asynchronous Task Graphs, a new approach to perform more complex kernel scheduling than the standard CUDA Streams, are described in [52]. It would be worth investigating the possibilities even if in the literature, it is mentioned that the kernel launch time is decreased on host side but not GPU side [1].

Seldom adopted in the literature, a spinning kernel, also called persistent kernel, is a continuously running kernel, suppressing drastically the tasks-launch overhead. This method can not solve generically all the ultra-low latency problems, but on suitable workload, it could be very efficient. Perret et al. proposed a control system for the adaptive optic of a large telescope performed this way [46].

Building scalable accelerated applications requires some form of automatic task management (dispatching the kernel execution on free nodes, transferring data between nodes, etc). Framework such StarPU presented in [5] aims to unify development process on heterogeneous architecture and to configure the scheduling of parallel tasks.

NCCL[19], the Nvidia Collective Communication Library for GPU, or Message Passing Interface (MPI) implementation such as MVAPICH2-GDR[20] are already GPU compliant. Such framework are widely used at the root of scientific distributed applications and one major challenge foreseen for the adoption of the RASHPA project by the community will probably be its fluent integration into largely widespread and adopted framework.

2.2.4 FPGA Accelerators

Field Programmable Gate Array (FPGA) are made of low level hardware cells, such as LUT Look up table, BRAM block RAM, multiplier, rich set of configurable I/O etc, that can be individually configured and interconnected. FPGA programming is performed by a versatile and sophisticated system acting like a memory that can be erased and reprogrammed at will.

In a CPU system, with say N processing cores available (N in one to a few tens), at a given time one core is executing only one instruction while at the same moment, N are

[19]https://github.com/NVIDIA/nccl

[20]http://mvapich.cse.ohio-state.edu/userguide/gdr/

executing in the system. With a GPU accelerator, there are a few thousands of threads available. They run slightly slower than those available on CPUs, but all in all, a GPU can offer more parallelism for the same number of transistors integrated in silicon as in a CPU. To get full occupancy of those cores, a massively parallel workload is required, each core performing roughly the same task at the same time. GPU cores are all the same, and somehow specialized, e.g. to compute 32/64 bits floats or 8, 16 bit integer, but they do not handle efficiently less common types, e.g. 14 bit width integers. With a FPGA design, this is no longer the case. Almost nothing is pre-configured in the silicon die. Even the processing cores are synthetisable and could be much more heterogeneous than those on CPU or GPU.

The main drawback of the FPGAs remains their price and also the complexity of the deployment of such solutions. A parallel software program is required but also a kind of *hardware design* on which the program will execute.

Recently the FPGA vendors have made a lot of efforts to provide better development tool for High Level Synthesis (HLS). Engineering and man power have been mobilized to assist the developer and decrease the learning curve. With the new generation tools such as Vitis HLS[21] from Xilinx, the Time to Market is significantly decreased compared to bare HDL code.

Despite being more slowly clocked than a GPU, in the range of a few hundreds MHz, an FPGA design can possibly perform data processing at very low latency. It is possible to implement a pipeline without task launch latency and no overhead involved.

2.3 The RASHPA Framework

The data transfer challenge from X ray image detector to computing unit has not been well addressed for a long time. Traditionally, efforts on detector development for photon sources have mainly focused on the properties and performances of the detection front-ends, sensors, ASICs, etc. In many cases the data acquisition chain and the data processing as well, are treated as a complementary component of the detector system and added at a later stage of the project.

The compelling need of a global approach of the entire acquisition chain has lead the ESRF to initiate a few years ago the development of an entirely new design pattern addressing the aforementioned issues from the ground. This new framework is called *RASHPA: RDMA-based Acquisition System for High Performance Applications.*

2.3.1 Paradigms

The main paradigm of the RASHPA framework is that a detector can produce multiple concurrent data flows to multiple data receivers. Thus a detector can always work at its full acquisition rate. When the ingress capacity of a data receiver is limited, only the throughput of the data flow directed to this specific receiver has to be decreased.

This can be done either by sub-sampling the transmitted images either by time slicing (one of N images acquired) or by spatial slicing (Region of Interest) or both.

[21] https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html

The treatment of the whole acquired data might then be achieved by dispatching concurrently the data flow to multiple receivers in charge of the data processing.

As shown in Figure 2.14 a typical RASHPA system would rely on the following hardware components:

- A peer to peer, unidirectional link for high throughput data transfer. RASHPA handles single sided data transfer between detector, source node and receiver, destination node for storage or online processing. Data transfers are peer to peer, but the system manage multiple transfers simultaneously, enabling one-to-many as well as many-to-one transfers.

- An additional bi-directional control link that serves for configuration purposes and the high level management of the tasks.

- Two DMA engines are located at both ends, operating in Memory Mapped mode to Stream (MM2S) at the source node and to Stream to Memory Mapped (S2MM) at the destination node. Those DMA engines might be implemented in a custom FPGA board or at convenience, by commercially available solutions such as RDMA Network Interface Cards (RNIC). It is foreseen the possibility to send a sub-image, called a *Region of Interest* (RoI) as shown in Figure 2.15. As a consequence, the data to be transferred are located in non-contiguous memory regions of a possibly larger RASHPA Buffer (RB). The scatter/gather capabilities of DMA engines are highly convenient in this context to send data, line by line. Buffer descriptors, i.e. size and addresses of data to be sent at source and destination, are just-in-time computed. For the sake of performance, it is performed by a highly efficient subsystem in the readout electronics of each detector module. The configuration of this parametrizable core is performed beforehand by the management system.

Thus, three subsequent operations take place: i) Data serialization at the source memory, possibly into multiple data flows ii) Data stream transfer over fast links and iii) De-serialization at the destination node and direct placement of the data in their final location.

A detector module can produce multiple different data flow simultaneously called Data Transfer Process (DTP). In addition to this data fan-out capabilities, a RASHPA receiver can also be used to consolidate multiple ingress data flows. Thus, a single high-performance computing unit might process the whole data issued from multiple detectors modules, while, in the same time, another computer may process a subset of the data to provide user feedback or automatic control purpose.

These Fan-in & Fan-out capabilities are key functional aspects of RASHPA systems.

The RASHPA Manager (RM) is a third party application software geared towards configuration and monitoring of a whole RASHPA system. Therefore, each RASHPA node features capability retrieval: data receiver and detector embedded controller as well are on-demand publishing their functional characteristics, status, etc. The RM is responsible of the configuration process of the relevant nodes according to the user requests and to start the acquisition process. This includes the configuration of the parametrizable core in charge of the preparation of the buffer descriptors to be consumed

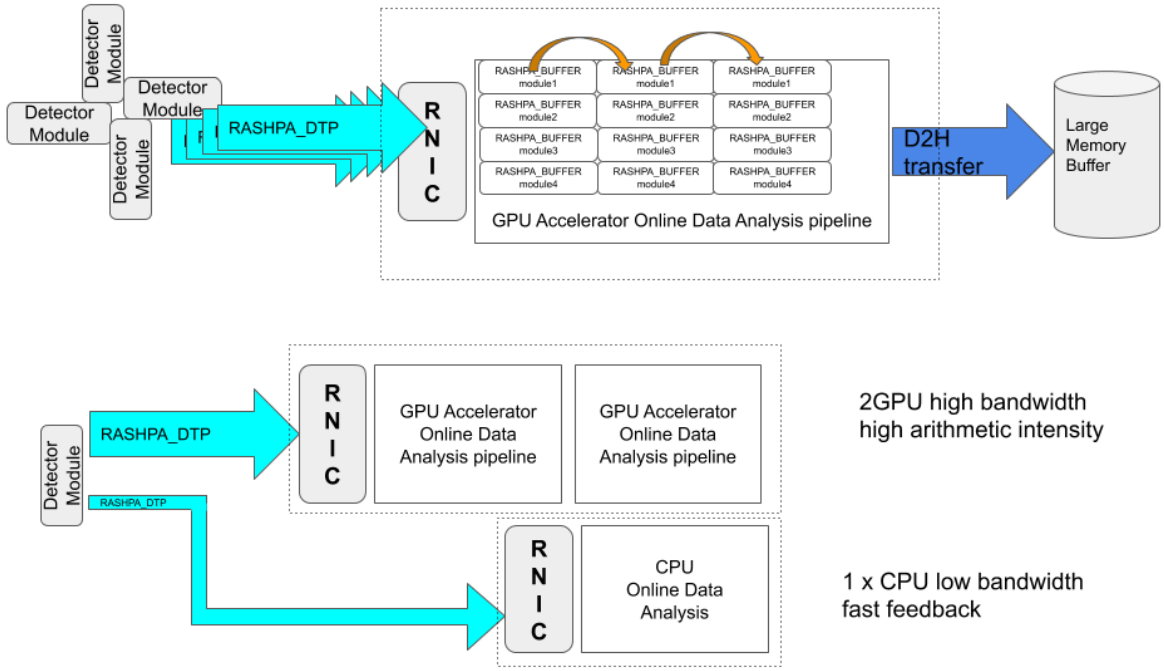


Figure 2.14: Overview of a typical RASHPA system. The multiple and concurrent data transfer are performed by DMA engines at source and destination.

by the DMA engine in the detector electronics. The RM has a specific API that defines messages to the nodes by telegram in XML format.

There is no handshake mechanism in the present RASHPA definition and there is merely no need for it. The RM knows in advance the capabilities of detectors and receiver as well, and consequently configure all the data transfers at an acceptable throughput.

The assessment of RoCE protocol as RASHPA compliant link is studied in section 3.1 and an event mechanism is also proposed.

2.3.2 Overview of the Frameworks in HEP or Astronomy

We have conducted a survey of the frameworks and technologies used in similar field of science facing big data challenges: High Energy Physics and Astronomy. All major facilities and new projects as well have upgrade plan of their IT infrastructure on regular basis to face the foreseen problem.

The Large Hadron Collider (CERN/LHC) data acquisition chain is presented in [30] or in [58]. The challenges are partly similar to ours but exacerbated. There are huge amount of data but the main difference is in the short burst nature of the data flow. Low latency startup time is very disruptive for computers that typically host simulations that run for days. The proposed DAQ system includes multiple cascaded trigger system.

The specific computing challenges of the Square Kilometer Area (SKA) telescope are in the remote location of the antennas, the scarce power supply and synchronization challenges. They are presented in [28].

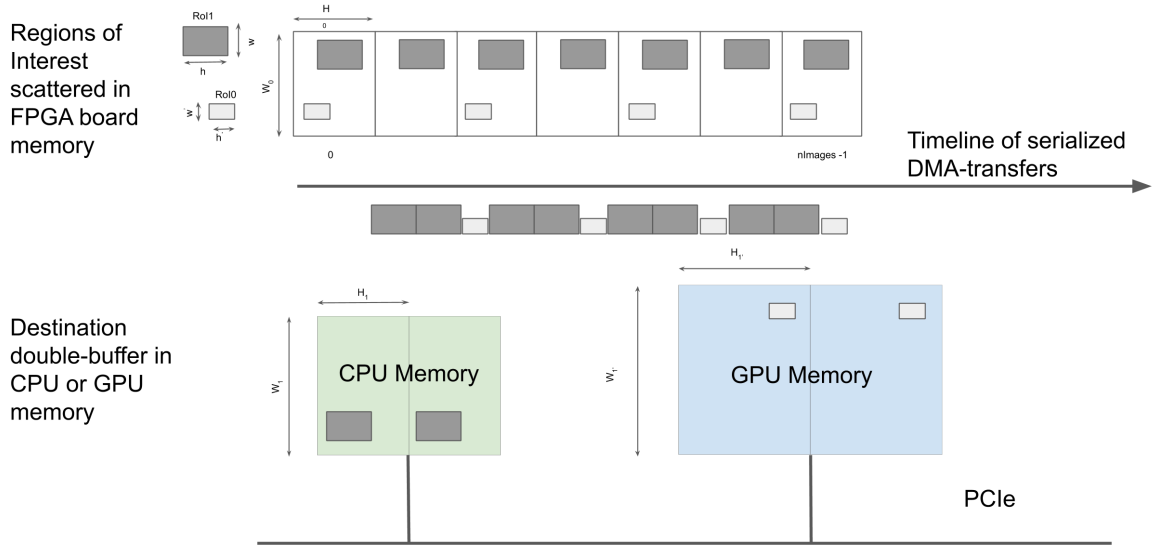


Figure 2.15: A View of Region of Interests (RoI). The end-user might be interested only by some fraction of the image. Only the related data will then be sent.

Very similar to our approach is the Adaptive Optics controlled by GPU presented in [46] but the proposed solution is limited to a particular application.

2.3.3 Contribution to RASHPA Processing Unit Specifications

In the early draft of RASHPA specifications, the topic of data processing was not addressed. It was laid under the sole responsibility of the developer of user applications.

The methodology we followed in extending the RASHPA project comprises the following aspects: i) Keeping compatibility with the previously defined concepts, ii) Establishing the RASHPA Processing Unit (RPU) concept, embracing different embodiments of hardware accelerators.

In our approach, a compliant RASHPA-processing system should behave as presented in Figure 2.16: i) The acquired data are remotely transferred using long haul links into the host computer or directly into the accelerator internal memory, as done in a classical RASHPA receiver. ii) Then data enter a staged data processing pipeline and processed at the pace of the detector. iii) Eventually, results are stored in a burst cache memory in the host computer.

The proposed extension features a RASHPA Scheduler daemon (RS) in charge of triggering the data processing pipeline at each RDMA transfer completion. This scheduler is a software application running on main CPU. It is monitoring the multiple *Image events* occurring in the RNIC devices at the end of a transfer on each Data Transfer Process. It

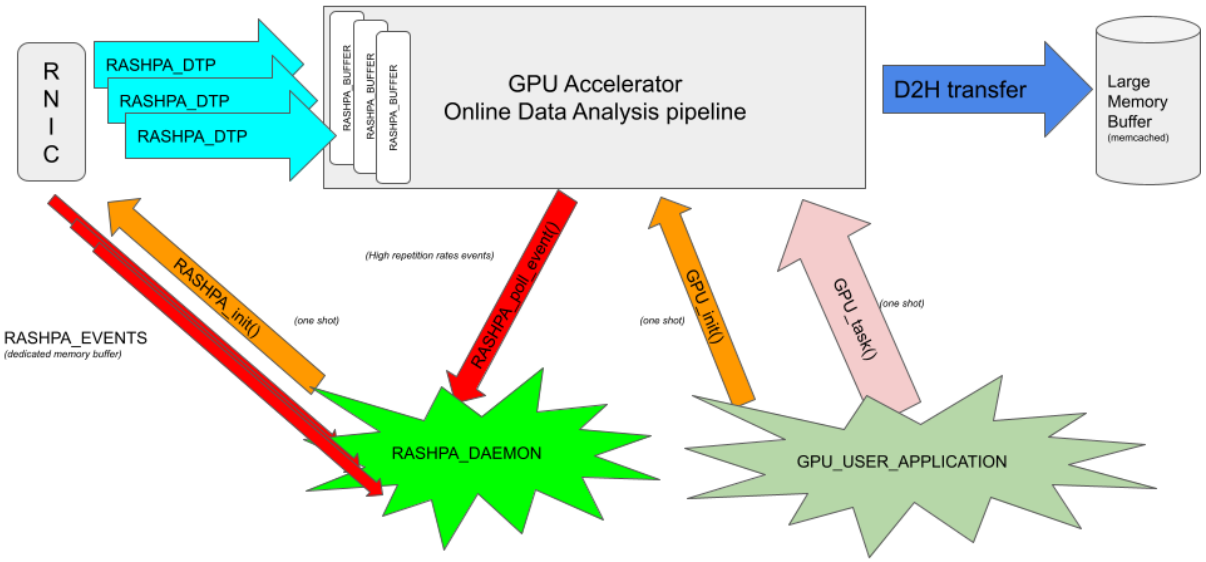


Figure 2.16: A RASHPA System implementing a GPU accelerator. The multiple modules from a detector might transfer sub-images to the same GPU for data processing performed on the consolidate image.

consolidates some of them, at the end of an image stack, into a single *global event* that triggers the accelerator device when present or the CPU in charge of the data processing.

The second proposed extension is the Address Translation System (ATS). The RASHPA ATS is used by the DMA engine at destination and performs the translation of the received addresses, from virtual to physical addresses. Indeed, the addresses of memory buffer in destination computer are set at detector side. But for robustness and security reasons, these are virtual addresses in the remote computer or accelerator. And therefore they must be translated as the DMA engine is using physical addresses. In the case of RoCE protocol and GPU this task is performed by mellanox drivers. But in the case of custom FPGA accelerator or in the case of PCI-e ove long distance, no such solution exists.

A typical RASHPA system providing the aforementioned extensions is presented in the schematics shown in Figure 2.16.

High throughput data transfer and accelerated GPU processing are well studied topics. However, in the particular frame of embedded readout electronics as envisioned in the RASHPA project, the most established and proven solutions based on high level code libraries or frameworks such as those adopted in HPC or Data Center are not fully satisfactory and can not be implemented as is. They either require some complex hand shake between the detector and the data receiver using bidirectional link or either a dedicated processor with a lot of processing power in the detector electronics. These

solutions can not be applied in our case. Other interesting solutions work well but are limited to a very particular problem and we are looking for a somewhat more generic solution.

Therefore we have built our own communication protocol from scratch, with 2D X ray imaging in mind, selecting an already existing lightweight RDMA data transfer protocol suitable to the embedded electronics found in detector head. The genericity and scalability of the proposed solution is demonstrated with the implementation of computing unit featuring heterogeneous accelerators. Its integration with existing framework has just started.

Chapter 3

RASHPA Data Source Simulators

Contents

This chapter presents the work carried out in view of the assessment of the RoCEv2 protocol as a sustainable data transfer solution in a RASHPA ecosystem. Only a few hardware alternatives were worth considering for the reasons presented in chapter 1. However, it was not known if the RoCEv2 protocol had all the required characteristics either in the programmability or either in a performance level suitable for a RASHPA implementation.

Some of the material comprised in this chapter has been published in the Journal of Synchrotron Radiation (IUCr), issue (Vol 27, 5) [49].

In this chapter, neither data processing nor the strategy to allocate memory on accelerator devices will be presented, as it will be extensively discussed in chapter 4. Instead, the focus is on the implementation details of the data transfer following RASHPA concepts. We propose several contribution in the form of detector mock-up in several versions on different hardware and links.

3.0.1 Methods

Our test were mainly done on two different hardware:

- Concerning RoCEv2 assessment, our benchmarks were run on a Dell T730 computer with two Intel Xeon CPUs E5-2643 (Sandy Bridge) v3 @ 3.40 GHz and 128 GB memory used as a detector simulator. The 100 Gb/s network card was a Mellanox ConnectX-5 EN in a PCIe generation 3 x16 slot, which is directly connected by an optical fiber to the analysis computer. This network card behaves similarly to a standard NIC for TCP-UDP/IP but decodes RoCE datagrams and transparently offloads the Linux kernel (4.16.0-0.bpo.2-amd64) and the CPU. The Mellanox Technologies stack in use is OFED-4.4-1.0.0

  The data-receiver computer is a Dell T740 (4.16.0-0.bpo.2-amd64) with two Intel(R) Xeon(R) Gold 6134 (Skylake) CPUs @ 3.20 GHz and 192 GB memory, and a ConnectX-5 EN card and OFED-4.6-1.0.1 stack.

- In the case of the RASHPA-PCIe proof of concept, a XILINX ALVEO U200 FPGA board is installed in a PCIE gen3 x16 slot. Workstation is a Supermicro server featuring two Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz (Ivy Bridge EP) 6 cores and 64GB RAM. Linux kernel 4.19.75 is configured with the CMA_DMA option and our cdma (Contiguous DMA allocator) and our gpua (GPU internal memory address translator) kernel modules.

3.1 REMU Detector Emulator

In this section, it is discussed how we have implemented the basic functionalities of the RASHPA Simulator and Receiver using the RoCEv2 protocol. The evaluated hardware is a standard workstation equipped with a 100Gb/s Connectx-5 EN dual port RDMA RDMA Network Interface Card (RNIC) from Mellanox Technologies. These RNICs can also be used as standard network cards in the traditional way, using the standard approach leveraging the TCP/IP stack. Although the target experiments seem possible at this

speed, the performed simulations highlight potential bottlenecks when using the standard Socket API as shown below. Therefore, a micro benchmark has been developed on purpose to evaluate RDMA over converged Ethernet (RoCE) techniques. It is concluded that the available transfer throughput from the RNIC to the compute accelerator inside the computer, the Peripheral Component Interconnect Express (PCI-e), is at present the major bottleneck in online processing.

3.1.1 Micro-benchmark of Network Protocols

The throughputs and packet losses when using the *Socket* API at 100Gb/s were measured by a testing application (leveraging on *recvfrom/sendto* call) that also checks the consistency of the received data.

Table 3.1 presents the maximum throughput achieved without packet loss using the standard techniques. A comparison of UDP performance is shown using classic socket API, without (column 1: UDP socket), or with Linux kernel optimization (column 2: UDP + kernel tuning), and the same code accelerated by the VMA library (Mellanox/libvma) (column 3: UDP-VMA). The DAQ framework under evaluation aimed at transferring non contiguous memory segments (mapping user-defined Region of Interest in an image). Therefore mainly small size transfer are measured. The results are more reliable when the data are always sent to the same destination address than when the data are sent to 100 000 different memory addresses. This is due to the limited size of the RNIC cache memory. However, there are anyway always packet losses if the transfer lasts.

Dramatic improvement can be observed when using UDP + libVMA, reaching up to 80 Gb/s, but only when the numbers of destination descriptors is limited. LibVMA is the Mellanox Messaging Accelerator Library presented in chapter 2

The measurement results when using ZeroMQ are added as reference as it is indeed one of the most efficient protocols available using the standard Socket API.

No improvement was observed when using libVMA with ZeroMQ, based on TCP transfers. That is due to the fact that ZeroMQ is already efficiently implemented and that even by using a hardware accelerator there is no further possible improvement. The implementation of RDMA into ZeroMQ was attempted in [27].

As expected, it appears that none of the pure software solutions can cope with 100 Gb/s network. The specialized hardware of the RNIC has hence been experimented to offload the host system and has shown its interest.

3.1.2 Programming with Verbs

The basic facts on RDMA in subsubsection 2.1.6.1 have already been presented.

As a remainder, using the Direct Memory Access (DMA) engine located in the RNIC, it is possible to bypass the Linux stack and put ingress data directly to its destination buffer in the CPU memory. The matrix presented in Table 3.3 summarizes which verbs and which features are available on a given queue pair type. Table 3.2 presents a comparison of the maximum bandwidth observed using different verbs. One can see that the maximum theoretical bandwidth and packets per seconds are achievable with all programming techniques. These performances are capped by the hardware. During our tests

Table 3.1: Observed bandwidth (in Gb/s) causing packet drops with the usual (non-RDMA) protocol and the hybrid solution.

	Standard UDP						**ZeroMQ**			
	regular Socket		**+ kernel tuning**[2]		**VMA**		**push/pull**		**rep/req**	
# of desc[1]	1	10^5	1	10^5	1	10^5	1	10^5	1	10^5
size in bytes										
2048	7.2	6.0	10.9	11	26.7	26.1	8	6.6	0.16	0.15
4096	9.1	6.4	22.8	23.1	51.2	45	9.5	8.5	0.30	0.29
8246[3]	9.8	5.2	36.7	27.4	77.0	39.7	11	10	0.57	0.54

Packets are not lost in the link during transfer but during Linux kernel UDP/IP stack processing. Images were send line by line for a fair comparison within protocols. As stated in the requirements of RASHPA, a detector can send only the data from a specific Region of Interest (RoI). This data is possibly in non contiguous memory if there is a stride between the lines of the RoI. ZeroMQ might be more efficient on a larger datagrams. Then, when sending several whole images at once, 40 Gb/s can be achieved

[1] Number of descriptors at destination, either one or 10^5 when transmitting 10^7 packets using Mellanox Technologies ConnectX-5 EN RNIC.

[2] increased size of system buffers

[3] PSI Jungfrau header size (4xlines of 1024pix (16b) + 54 Byte header)

using the RDMA techniques, we did not notice neither packet drops nor reordering for any of the variants of RoCE transfer.

At first, a loosely coupled approach has been evaluated between detector and data receiver using the SEND verb. This verb behaves like data streams or a standard sockets: there is no data destination addresses set in the datagram and this is up to the receiver to select where to store the data at its final location.

This implementation leads to halve the computations of buffer descriptors addresses: i) first half of computations is done in the detector head, setting the source addresses where to fetch the data in the detector internal memory ii) second half of computations is done in receiver unit, setting the data destination addresses. Both computations are performed on the fly according to rules fixed beforehand by RASHPA manager.

Sustainability of this mechanism relies on lossless transmission preserving packet order, precisely the meaning of Coherent in RoCE acronym. However, with properly configured and correctly sized networking devices, no packet losses should happen. It was confirmed by our tests that there is no issue preserving the packet order if the detector and receiver are connected back to back.

However, packets travelling through Local Area Network (LAN) may arrive out of ordered when using network switches with aggregation links. But it is possible to properly configure the switches, statically associating a given RDMA traffic to a specific link and thus suppressing this issue.

Remains a synchronization challenge between a receiver and a sender if the latter starts sending data before the receiver is ready. At the moment the receiver will start receiving and processing the ingress flow, it can not anticipate where the data has to be stored. An extra synchronization events is required. Such a mechanism has been successfully implemented. It is a bit convoluted taking into account the asynchronicity of the verb operations.

Table 3.2: Observed maximum bandwidths from RNIC to CPU memory using RDMA.

size (in bytes)	**RoCE-v2 SEND**	**RoCE-v2 WRITE**	**RDMA RAW**
2048	94.5	95.2	33.3
4096	97.6	97.5	60.5
8246	N/A[1]	N/A[1]	99.1

Results in Gb/s. Maximum achieved greatly depends on the memory allocation scheme (fragmented memory or not).
[1] RoCEv2 MTU max = 4096

Table 3.3: Distinctive characteristics of the main Verbs.

Verb/QP type compatibility	**SEND**	**WRITE**	**RAW**[1]
Unreliable Datagram qp	yes	no	no
Unreliable Connected qp	yes	yes	no
Reliable Connected qp	yes	yes	no
RAW qp	no	no	yes
RASHPA requirements			
unidirectional	on UD/UC qp	on UC qp	yes
multi-cast	on UD qp only	P2P only	yes
data placement from source	no	yes	no
event mechanism	IMMEDIATE	IMMEDIATE	no
max payload	4096	4096	9000
caveats[2]	40 bytes lost		

[1] method evaluated for the Jungfrau detector
[2] 40 bytes are consumed by GRH header (compatibility with IB)

But halving the number of descriptors to process at source has the advantage of decreasing the overall strain on the detector electronics. That could be a key point when the processing power of the FPGA is severely limited. The second half of descriptors are processed on a data receiver running on much more powerful computer.

In the present implementation of RASHPA systems, the WRITE verb on UC queue pair has been chosen. In this operation mode, the detector device must compute on the fly two lists of memory addresses: one list for the addresses at the source, and a second one for the addresses at the destination. In the real detector hardware, the whole process of generating RoCE datagram is performed by custom IPs, engineered at the European Synchrotron Radiation Facility (ESRF).

For testing purposes, we have developed micro applications based on *libibverbs* that serves either as source or data sink. Low level RDMA programming as done in this work might be delicate and must follow some rules. The main programming concepts are described below.

- The RDMA source application has to pre-compute a so-called list of *Work Requests* (WR). This is a linked list of descriptors (address, length) identifying source and destination addresses of the data buffer to be transmitted in the respective memories of the detector and the data receiver.

It is *posted* to the RNIC *send queue* and processed by the hardware as soon as possible (when it has completed the previous one). The *Completion Queue* can be potentially polled until an event by the main application in order to monitor the transfer progress. Posting WR is executed fully in the user space. It is in this context far less time consuming than standard network operations because none of the costly tasks such as CPU context switches are required. By the way, some optimised network cards used in HPC are also capable of executing applications fully in user mode.

Design of an efficient RDMA application is challenging as it should generate a list of work requests at the same pace that they are consumed by the RNIC hardware. Posting a list of work requests after the full consumption of an earlier batch by the hardware of the RNIC, leads to an inefficient application. Posting work requests is indeed a relatively long process and the hardware will stay idle until the WR are ready to be processed. Peak performance is reached following the proposed implementation: i) at the beginning of the application, post an half of the work request ii) then enter in the main loop, iterating over the four following operations: 1) post the top half of the work request, 2) poll for completion, 3) post the bottom half of the work request, 4) poll for completion.

This way, there will be always outstanding work requests in the RNIC.

- The RDMA receiver application is quite similar in principles. Using WRITE verb, there is no need to post WR. The hardware is fully autonomous while processing RDMA ingress traffic.

  When using the SEND verb, a list of receive WR must be posted on the *Receive Queue* and is asynchronously processed by the hardware. One must post receive WR of at least the same size as the corresponding send WR. Posting at the same pace than the source is therefore challenging. If there is no more *outstanding* work requests, any incoming packets will be silently dropped by the hardware and will be missed by the application. Posting the work request has a significant time budget. It is cautious to en-queue enough WR in the receive queue well in advance so that it never gets empty. Thus the RNIC has always outstanding work requests ready for any incoming RDMA datagram.

As a reminder, the buffer descriptors in the work requests have to be computed on the fly as the addresses of the data to transfer are continuously changing. A list of work requests can count a few thousands of items (precise number is depending of the RNIC resources) and must be computed during the processing of the previous. The similar process takes place in the receiver application posting work request in the *Receive Queue*.

SoftRoCE[1], a soft implementation of the RoCE protocol working on standard network card was evaluated in the course of this work that could serve for low budget solutions or testing purpose.

[1]https://github.com/SoftRoCE

3.1.3 Event Implementation

The initial draft of RASHPA stated the necessity of an event mechanism but nothing was precisely defined.

In order to monitor the data transfer in a RASHPA receiver or to process the data in a RASHPA processing unit, an information on image or line number is needed as it is progressing. Although there is a Packet Sequence Number (PSN) embedded in each RoCEv2 datagram, there is unfortunately no way to make it available to the user application. The proposed RASHPA event implementation is leveraging a special form of the WRITE verb, *IBV_SEND_WITH_IMM verbs*. This verb embed in the transmitted datagram an extra 32 bit value called *immediate*. At destination, this immediate data is not stored in memory with the data, but stored in the *Work Completion* queue, which is accessible by the receiver control application .

On the receiver side, the CPU is notified by an interrupt handler or it must be busy polling the RNIC completion queue (CQ). In our tests, one CPU core was permanently running at 100% to perform this task. When the interrupt handler is taking care of this task, it is effectively notably decreasing the CPU usage. Polling increases the CPU utilization, because the system polls the received rings for incoming packets; however, it may increase the network bandwidth since the incoming packet is handled faster. We have observed packet drops when using such an interrupt handler in harsh conditions when high rate events are occurring. This is likely to happen for ingress flow made of small stacks of images of small height.

Fulfillment of an event mechanism sufficient to trigger data processing has been achieved with a custom image number, inserted in the last datagram of each stack of images.

3.1.4 Proof of Concept

As a first proof of concept of RASHPA over Gigabit Ethernet, we have developed the REMU, a detector simulator. Micro-benchmark had already demonstrated the RoCE feasibility. For the reasons stated above, it leverages WRITE verbs and *libibverbs*. Events are transmitted using the aforementioned method. The general working is shown Figure 3.1.

REMU has been designed having in mind the other distinctive RASHPA features and their assessment with RoCE protocol:

- REMU can put on the network links the raw data of any detector format. The data are previously uploaded in memory. They are prepared by Python scripts from existing data-sets or fake experimental data, converted to the specific format of the targeted detector.
- The simulator is controlled by RASHPA telegrams issued by the RASHPA Manager. The RM is also made of Python scripts. The messages and commands exchanged between the nodes and RASHPA manager are XML telegrams.
- A Region of Interest (RoI) in the dataset, i.e. a rectangular sub region, can be defined and selected for data transmission.

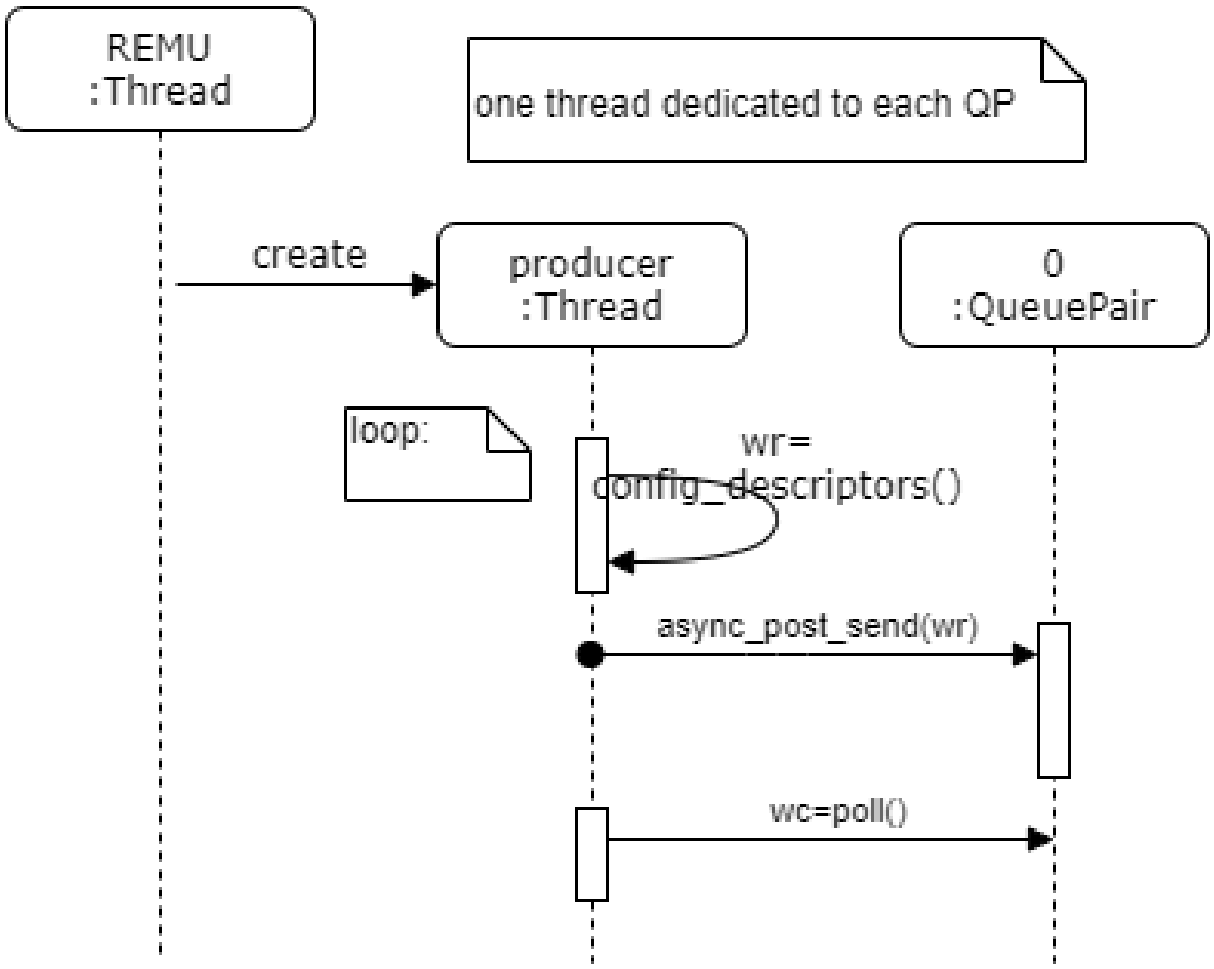


Figure 3.1: A typical sequence diagram of REMU simulator. It implements one thread for each ongoing data transfers. Multiple data transfer might be activated concurrently to fan-out the data to multiple receivers.

- The simulator supports multiple concurrent Data Transfer Process (DTPs). The DTPs are implemented by a different threads and they are associated to a different queue pair.
- It is built using a library internally developed at the ESRF, called *DaNCE*[2], a collection of reusable library for embedded system.

3.2 RASHPA PCI-e Implementation

Some materials of this chapter have been presented to the Euromicro Digital System Design DSD20 Conference. The proceedings paper can be found in [48].

[2]for ESRF internal use only

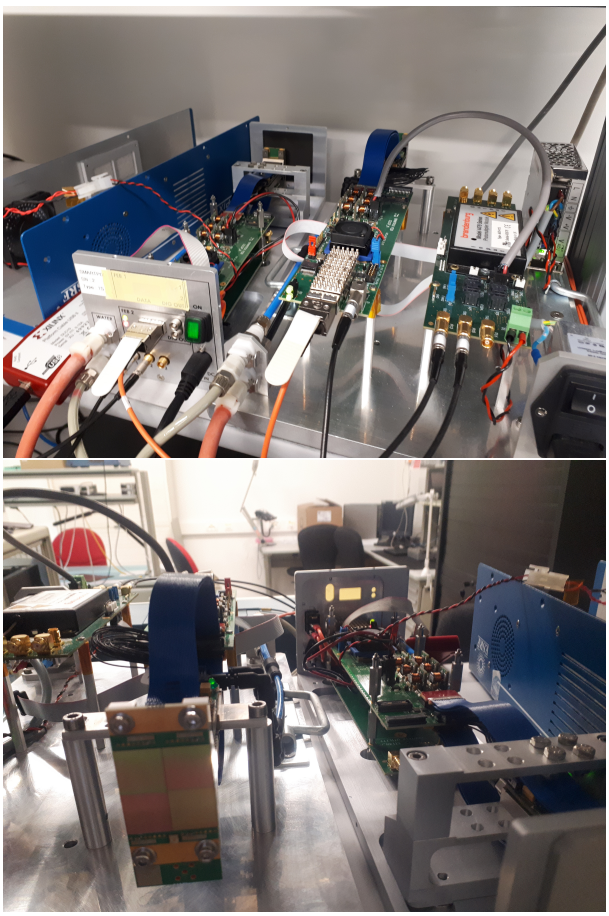


Figure 3.2: View of the Smartpix Detector, which is the first RASHPA compliant system. The featured links is PCI-e long distance with hardware from One Stop System Company.

The RASHPA specifications are links agnostics. While Gigabit Ethernet solution has competitive advantages and has been thoroughly presented in the previous section, other transport solutions are worth considering such as PCI-e over long distance[3]. This alternative solution is promoted by company such as One Stop System. It is still interesting as it offers very good performances. Latency might even be better than these available with RoCE (150ns vs 700ns). The main issues are the requirement of specialized hardware offered by a very small number of vendors and the limited routing capabilities as well.

An already existing detector prototype Smartpix shown Figure 3.2, designed at a time Gigabit Ethernet was not ready for the expected throughput (around 20 Gb/s), is leveraging PCI-e. This detector should be the first actual hardware detector to be RASHPA compliant.

This section presents an FPGA-based RASHPA PCI-e detector simulator. It has been used to validate a RASHPA compliant allocation of memory in Linux host and an event mechanism as well.

[3]https://www.onestopsystems.com/product/pcie-x16-gen-4-cable-adapter

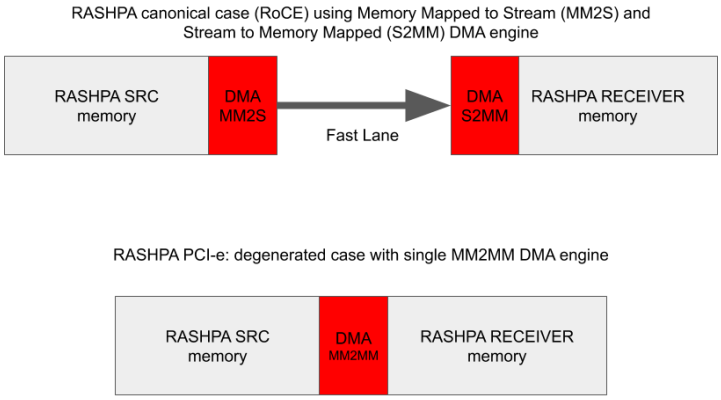


Figure 3.3: An Overview of the RASHPA system DMA engines. The typical system includes both a Memory Mapped to Stream (MM2S) and a Stream to Memory Mapped (S2MM) DMA engines. The PCI-e version is a reduced form of the latter, with a single MM2MM DMA engine.

3.2.1 Reduced RASHPA

While Gigabit Ethernet and PCI-e seems totally strangers to each other, it appears that the RASHPA PCIe version is a simplified version of the generic RASHPA concept. On can consider it is as a kind of degenerate form of the canonical one. This limited implementation features only a single DMA engine, transferring data from a memory mapped region to another memory mapped region as shown Figure 3.3.

As a contribution to the PCI-e version of RASHPA, an FPGA based detector emulator has been developed. It is four folded:

- implementing an FPGA design as detector proof of concept. It is using an embedded soft-core (Xilinx Microblaze) to perform on-the-fly configuration of a DMA controller (Xilinx CDMA). It is featuring a PCIe bridge, that is mapping the host memory into the FPGA internal Advanced eXtensible Interface (AXI) interconnect.
- proposing a firmware application in charge of computing the descriptors to be processed by the DMA engine,
- defining an event mechanisms compatible with PCI-e, in order to trigger the data processing at the end of a DMA data transfer,
- developing an allocator for large contiguous block of central memory. Allocated memory region are suitable for DMA operations,

3.2.2 FPGA Design

We have designed the proposed RASHPA mock-up over PCI-e long distance using a FPGA board that is presented in Figure 3.5. In addition to the before-mentioned soft-core Intellectual Property block, it is based on the following standard IP:

- DMA engine (Xilinx CDMA IP) This is the DMA engine operating in scatter / gather mode: it can transfer a list of buffers, predefined by a list of buffer descriptors (BD). The idea is to evaluate if the calculation of the next list of BD can be hidden during the transfer of the previous list.

  The system supports multiple simultaneous DMA transfers to multiple destinations as shown Figure 3.4. As only one single DMA controller is implemented, data transfers will be serialized by the PCIe-bridge and consequently, a slow PCIe device will slow-down the whole transfer.

- PCIe Bridge (Xilinx DMA Bridge subsystem IP) This IP handles data transfer to/from FPGA from/to CPU performing address translation between soft-core addresses (AXI bus) and host CPU memory map (physical addresses).

- Microblaze soft-processor is clocked at 300MHz by the DDR4 controller. The 512 bit data-width CDMA is transparent in the AXI bus and full throughput can be achieved over PCIe Gen3 x16 as well. The software was written in C language.

Such a design might not be so efficient than a full HDL based RASHPA system, that could compute BDs at each clock pulse, quicker than the pace at which they are consumed. But our design is much faster to design, maintain and synthesize than a pure HDL solution. For this reason, at the time of writing, the RASHPA PCI-e implementation does not generate events due to the complexity of the process and the changing specifications.

3.2.3 Firmware

We have developed concomitantly the firmware of the Microblaze processor which configures, controls and monitors the data transfers. Image datasets are uploaded during initialization using DMA engine from host to FPGA DDR. The firmware is in charge of the elaboration of the list of buffer descriptors. It builds a new list during the DMA transfer of the previous one. For a fair evaluation, one has to merge multiple concurrent data transfers. It as been evaluated if it is beneficial to make overlap computation of the buffer descriptors with data transfer using a pipeline.

The DMA configuration requires calculation of two memory descriptors: addresses and size, both at source and destination. This calculation must be done in real-time: this can not be fully pre-processed in advance as we are handling very large datasets and storing pre-calculated descriptors would require a huge amount of memory. The proposed design performs those buffer calculations by software using a soft processor.

To achieve maximum performances, the proposed design implements some performance tuning features: the Microblaze soft-core enables data cache, hard multiplier but does not implements MMU. Timing where met to run all the IPs at 300MHz. Firmware

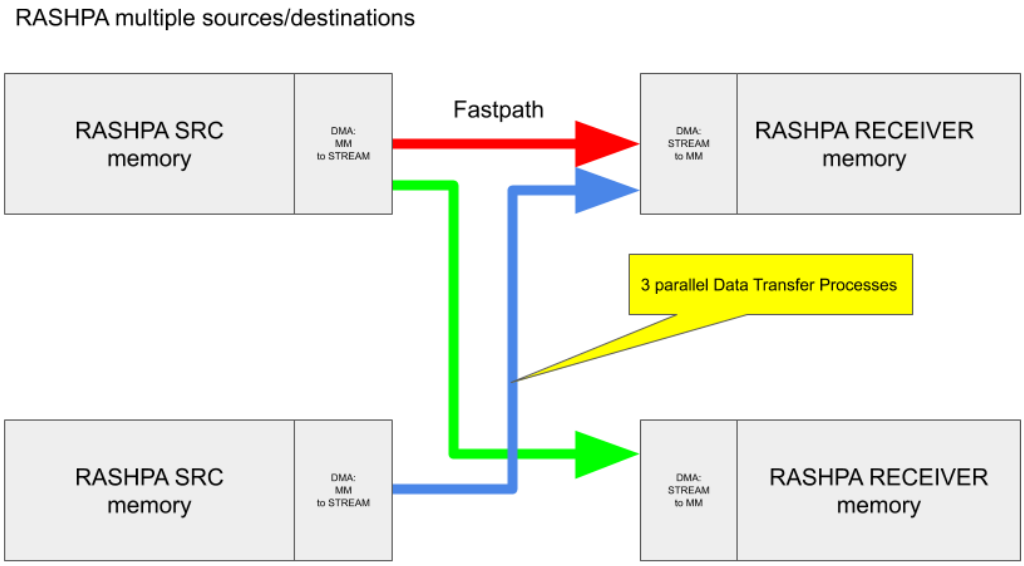


Figure 3.4: Overview of distinctive characteristics of the RASHPA framework. The system manages simultaneously multiple data transfers from multiple modules to multiple receivers. Memory Mapped to Stream (MM2S) DMA engine produces the data flow consumed by the Stream to MM (S2MM) DMA engine.

benefits from compiler optimizations: function in-lining, variables set in processor registers,...

The buffer descriptor processing is a very simple algorithm but many parameters are required to fully specify a RoI in a given dataset at source and at destination.

For each buffer, the configuration of the DMA engine is performed by functions similar to the one presented in Listing 3.1. The algorithmic complexity of the computation depends of the number of parameters and is tied to the user requirements. The same rule apply to calculation of destination, taking into account that there is also a double buffer.

3.2.4 Events

RASHPA events must be produced after each stack of images to trigger the data processing on the host computer or accelerator. The proposed PCI-e implementation of a RASHPA event is a PCI-e transaction generated by the firmware, using a memory write on a designated address that serves as memory lock.

3.2.5 Allocation of Large Memory Buffers

For proper operations, the RASHPA system require contiguous memory regions. This is mandatory to decrease the number of descriptors required to access the whole memory

Listing 3.1: This snippet performs the buffer descriptors (addresses at source and destination memory) in real-time. These addresses are used by the DMA engine.

```
1
2  /* Parameters at the source are:
3  FPGA_DDR:        AXI address in FPGA board
4  width, height:   RoI size in pixels
5  nImages:         number of Images
6  w0, h0:          image size in pixels at source
7  x0, y0:          RoI location
8  Parameters at destination are:
9  DDR_HOST:        physical address in host memory
10 OFFSET:          used to perform the alignment
11                  between AXI and host addresses
12 b:               boolean for ping pong buffer
13 bunch:           stack of image number
14 w1, h1:          image size at destination
15 x1, y1:          RoI location
16 */
17
18 //source address calculation
19 BD[i].addr_src  = FPGA_DDR + OFFSET
20     + n * w0 * h0
21     + x0
22     + y0 * w0
23     + i  * w0
24 //destination address is changing
25 //to a double buffer
26 BD[i].addr_dest = DDR_HOST
27     + i * w1
28     + (n % (2 * bunch)) * w1 * h1 + x1 + y1 * w1
29     + (!b) ? 0 : w1 * h1 * 2 * bunch
```

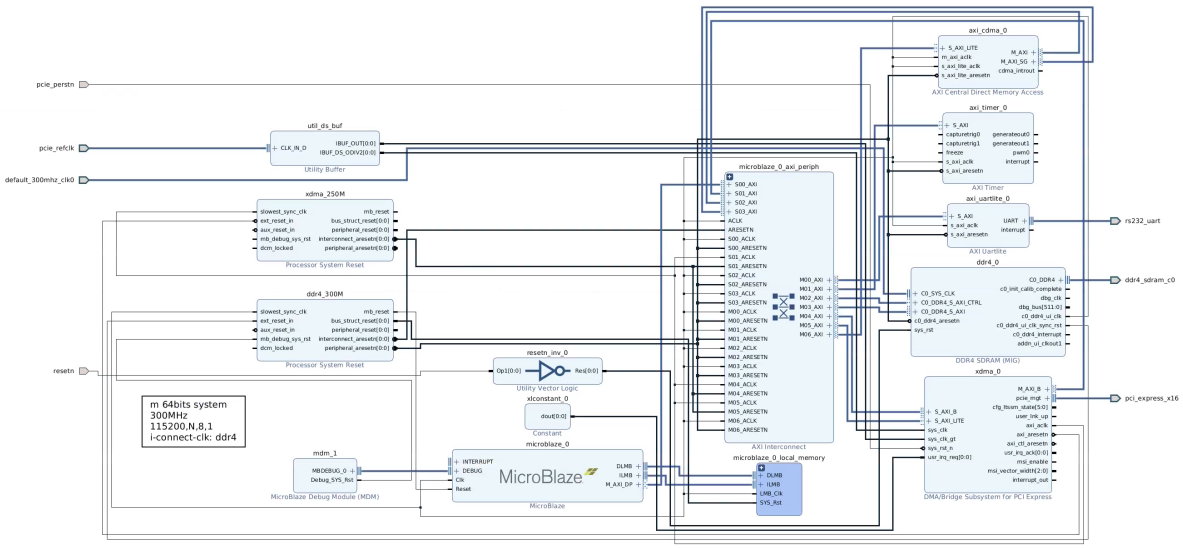


Figure 3.5: Overview of REMU PCI-e block design in Vivado. It features a DMA bridge, a CDMA controller and a Microblaze softcore in charge of the online configuration of the buffer descriptors.

region. As X ray detectors may produce huge amount of data, the size of allocation is expected to be in tens of GB. Such range in sizes would required hundreds of thousand of descriptors, a number challenging to manage.

A Linux kernel module leveraging the Contiguous Memory Allocator called *cdma* has been implemented. This module and a wrapper library are enabling a user land application to request large contiguous buffer in main memory. It is possible to allocate all the available memory of a given NUMA set. The physical memory address returned has to be transmitted one way or another to the FPGA system in charge of the data transfer taking into account the offset due to the misaligned addresses as shown Figure 3.6. Cache coherency (RDMA engine writes to physical memory, CPU cores read from their respective caches) is managed transparently by the system. The CMA_DMA feature is not enabled by default in the Linux kernel provided by standard Linux distribution and kernel has to be recompiled.

A second kernel module was developed for test purpose to get physical addresses of user allocated memory using huge-pages techniques. Huge pages size are 1GB in large (or 2MB depending on processor model). Larger allocations are backed by multiple pages that are not guaranteed to be contiguous.

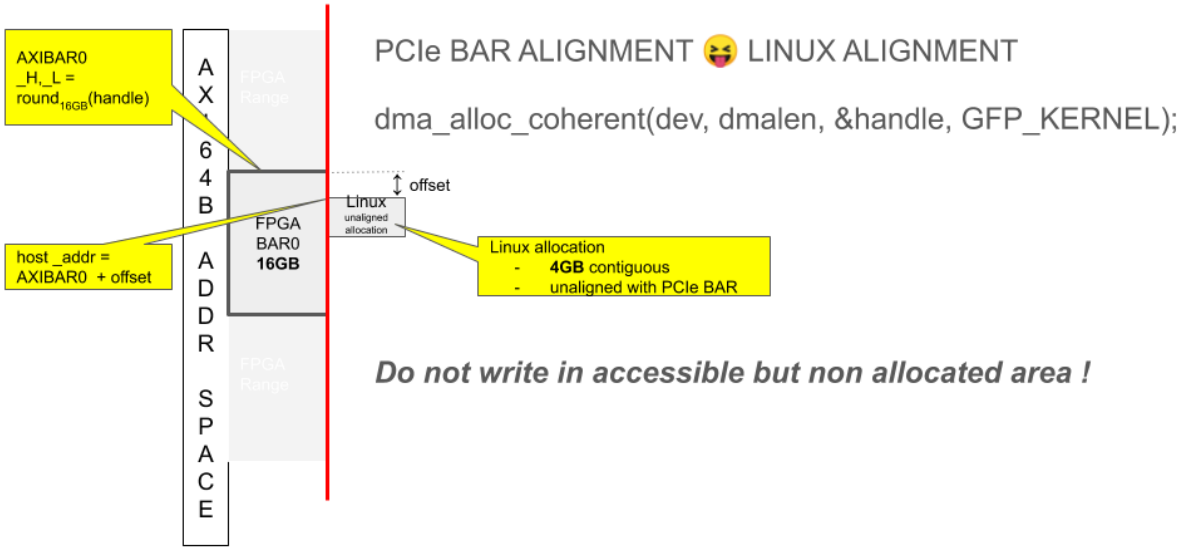


Figure 3.6: The memory region returned by the Linux CMA is not properly aligned with the PCI-e bridge Base Address Register (BAR) in the FPGA.

3.3 RASHPA RoCE using Xilinx IP

Recently FPGAs company Xilinx announced the availability of the ERNIC IP, a RoCEv2 implementation[4]. This IP was evaluated and compared to the ESRF development of [32] started before the announcement of its availability. ERNIC IP leverages Reliable Connected (RC) queue pairs that do not perfectly match our design pattern as we considered only uni-directional links. However, with a properly sized system, we can expect neither packet drop, congestion in switches, nor contention at destination memory. Thus, all acknowledgements would be true. Assuming this, the data initiator has nothing particular to do and it would be possible to ignore them.

We did some performance evaluations using the reference design proposed by Xilinx. The proposed design features a Microblaze soft-core to perform ERNIC configuration and interaction with the user. Thanks to an embedded Linux system, it makes also standard the IP stack available and thus greatly facilitates remote development process. However the performances suffers from the time taken to process the list of buffer descriptors when used as initiator (in this mode, ERNIC starts the data flow). The measured throughput in this circumstance was not better than 8Gb/s.

When ERNIC was used as receiver, performances are better and one can reach the maximum throughput. Our measurement where limited by the fact that our RDMA network card was in a 8x PCI-e gen3 slot, halving the available bandwidth.

[4]https://www.xilinx.com/products/intellectual-property/ef-di-ernic.html

3.4 Outcome

In this chapter we have focused on the data transfer capabilities of the RASHPA framework and respective implementation on RoCE and PCIe. We have designed Detector simulators and RASHPA Receivers without processing capabilities.

3.4.1 Results of the RoCEv2 version

We have evaluated the feasibility of RDMA data transfer in the frame of high-performance X-ray detector development using gigabit Ethernet data links after confirming the flaws of the standard socket API at 100 Gb/s. It appears that one way communication, a major challenge to stay compatible with detector embedded electronics, is possible using RoCEv2 Unreliable Connected queue pair and WRITE verb API and commercially available RNICs in data-receiver computers. The expected throughput is effectively achieved without usage of the destination CPU, without packet drop, and is solely limited by the RNIC packet per second message rates for small-size payloads.

Based on the outcome of our benchmarks, one can draw some numerical conclusions, as far as the data transfer is concerned: in the UDP-based design, 9 Gb/s data transfer is the limit, with unavoidable packet drops in continuous operation. In a first step of improvement, with a RDMA-compatible NIC, using the VMA library and without changing the application software, one can reach almost 26 Gb/s. At the price of the rewrite of the application code with RDMA API, it is definitively possible to saturate a 100 Gb/s link without a packet drop. The use of RDMA appears to be a very good solution to optimize the data transfer from the detector to a data receiver, especially the RoCEv2 technology preserving existing ethernet infrastructure.

We have also tested the RoCEv2 routable properties in a restricted[5] test bench featuring three workstations and a 100Gb/s switch. We noticed that as expected, Ethernet Global Pause Forwarding by the switch is required if the RASHPA receiver is slower than the source. We have verified it was possible to fan out detector data to multiple data receivers from the same computer and to use multiple endpoints at the same time without causing packet drops. We also checked that mixing RDMA datagram with standard IP traffic did not cause no packet drops. We tested the reverse functionality, gathering in a single data receiver, the data flows from multiple detector modules.

In order to do so, we developed a custom benchmark suite including a RoCEv2 sender and a data sink: rcwrite and rcwritesink. Thanks to the Pause Frame Control, the hand-shake protocol handled by Ethernet devices, the throughput is automatically shared between the concurrent data transfers.

3.4.2 Results of the PCIe version

DMA throughput does not suffer limitation in the FPGA design. In our setup with PCIe gen3 x16, 16GT/s (Giga Transfer) is the theoretical maximum. As expected when taking into account the diverse overheads, and PCIe Maximum Payload Size, the observed

[5]it was allowed inside the datacenter during a short period of time

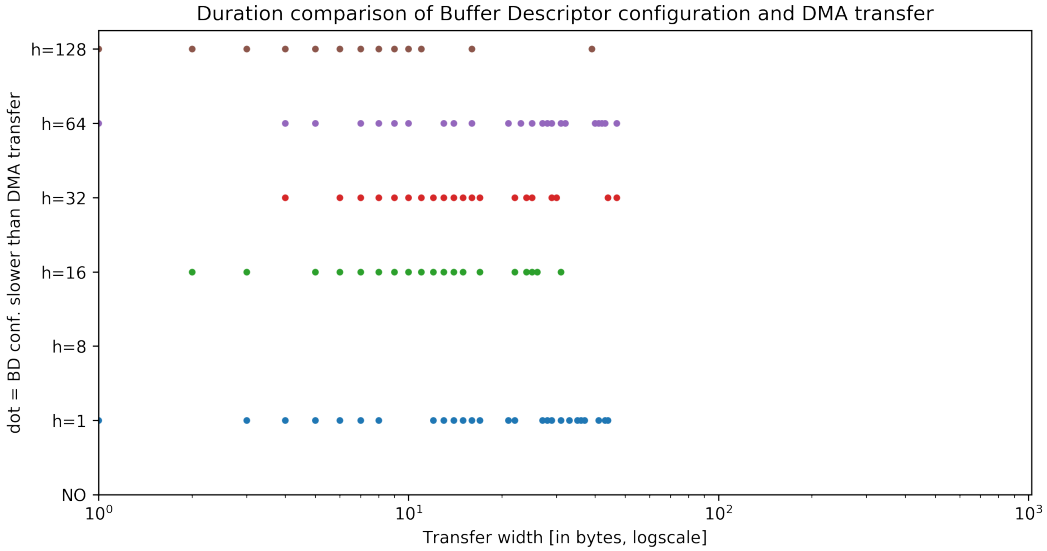


Figure 3.7: Efficiency of BDs soft-processing: a dot indicates when Buffer Descriptors processing by the embedded application has taken more time than the DMA data transfer.

maximum is around 12.8 GB/s. Write operation are more efficient (READ need an extra step sending address before receiving requested value). Better throughput is achieved when it is possible to hide the BD processing during ongoing data transfer.

During DMA transfer handled by CDMA engine, the Microblaze soft-core remains free for other tasks. The Figure 3.7 shows the number of free CPU cycles before DMA completion, in worst cases (small buffer size). This free time might be used to process the configuration of the next list of BDs. The DMA transfers take less time than the BDs preprocessing except, as expected, for small size transfers.

The work carried on the assessment of RoCEv2 and PCI-e long distance gave us good hope these protocols could be satisfactory implemented in real RASHPA compliant detectors.

Chapter 4

Online Accelerated Data Processing using RASHPA

Contents

This chapter is dedicated to the presentation of our approach of data processing in the frame of RASHPA-compliant X-ray detectors. The main characteristics of these detectors have been presented in the previous chapter as well as simulator implementations, engineered due to the unavailability of existing detectors.

Massively parallel coprocessors are already successfully used to perform *offline data processing*, i.e. with data previously acquired and stored in a storage system. The drawback of this approach is the space required on disks and limited bandwidth of read/write accesses, hampering efficient data processing.

The other potential issue with offline processing lies in the late detection of potential data acquisition problems. That is a serious risk considering the scarce accessibility of a Synchrotron X ray Source and the cost of beam time.

Our contribution to the RASHPA project as far as computing is concerned is four folded:

- The definition of a generic data processing mechanism that is suitable with the existing RASHPA framework. It includes a synchronization method between ongoing data transfer and the computing unit.
- The implementation and assessment of this new feature. We will dwell into the details of different implementations leveraging on multi-core CPU and GPU and FPGA based accelerators.
- The optimization of the task-launch mechanism for GPU accelerators. In order to decrease the latency overhead of kernels, they are pre-launched asynchronously on Compute Unified Device Architecture (CUDA) streams.
- The overall assessment of the system by several data processing algorithms from SSX science.

In the scope of this thesis, the focus has been be put on data processing performed in real time, i.e. at the speed of the data transfer. In its primitive design, a RASHPA Receiver (RR) only addressed data transfer and direct data storage challenges at the final destination in a large pre-registred memory region. The standard RASHPA receiver did not handle any data processing as is.

However, it was already foreseen to perform some form of data processing at the source of the data stream, in the detector electronics. Such data processing had to remain compatible with the scarce processing power available in a single module electronics. Low level pre-processing could be considered such as geometry reconstruction, offsetting, rotation, flip, binning, or other operations staying compatible with the FPGA computational resources.

An annoying limitation of the data processing at source, is that a module electronics has access only to its internal memory. It will therefore not be possible to perform any algorithm requiring the consolidate data from a whole image.

4.1 RASHPA Processing Units

A new concept as been introduced that extends the philosophy of RASHPA behind its primitive design. The so-called RASHPA Processing Unit (RPU hereafter) has been defined as shown Figure 4.1. The core framework has been supplemented with the components that were lacking to perform the foreseen data processing.

In this work, a hardware accelerated device is a Peripheral Component Interconnect Express (PCI-e) extension board in a standard host computer, beside the RNIC involved in RASHPA networking.

An accelerator board features the following functional items: i) a PCI-e bridge in case of an extension board, ii) an high speed memory (DDR4/5 or HBM2), in the range of tens of GB, iii) multiple parallel processors in an integrated circuit (IC), iv) ultra-fast memory in the silicon die, in the range of tens KB.

The simplest accelerator could be the host computer itself and in this case there is no PCI-e interconnect.

The general working principle of a RPU is the following: i) the peer device RNIC DMA engine transfers data to the internal memory on the RPU, ii) the data are then fetched by the hardware accelerator into its internal registers or shared memory for the data processing itself, iii) the results are then pushed back to the host main memory. It has also been considered staged data transfer in central memory, as many low end accelerators did not support direct data transfer in internal memory.

Only the real-time data processing has been investigating in this research work. Thus, the available time slot might be very short in case of images of small size, acquired and transferred at high frame rate. But that does not necessarily imply that the data processing is performed after each image. It could be more efficient to process larger chunks of data, i.e. by stacks of images processed as a whole. This mechanism enables to decrease diverse overheads and the number of task launches and transfers. The overall performance might also be improved because it increases the parallelism of the computation. Thus, the accelerator could process some algorithms possibly more efficiently on stacks of images than on single image.

Such workflow would on the other hand increases the overall latency of the process, up to the time required for the processing of a stack of image.

4.1.1 General Working

First of all, an event mechanism is required to trigger computation in RPU as soon as a chunk of data have been fully transferred in the accelerator memory. The implementation depends on the transfer protocol and its implementation in the detector side as already been presented in the previous chapter for RoCE and PCI-e as well.

While not being aware of the status of the RDMA transfer, by the very nature of the RDMA mechanism, the CPU must explicitly be notified by the RNIC of the transfer completion of a given data chunk. In the proposed system, a CPU application or *software daemon*, hereafter called the RASHPA Sequencer (RS) shown Figure 4.2, shall be busy polling the RNIC completion queue. It is kept on waiting for new events and shall do so

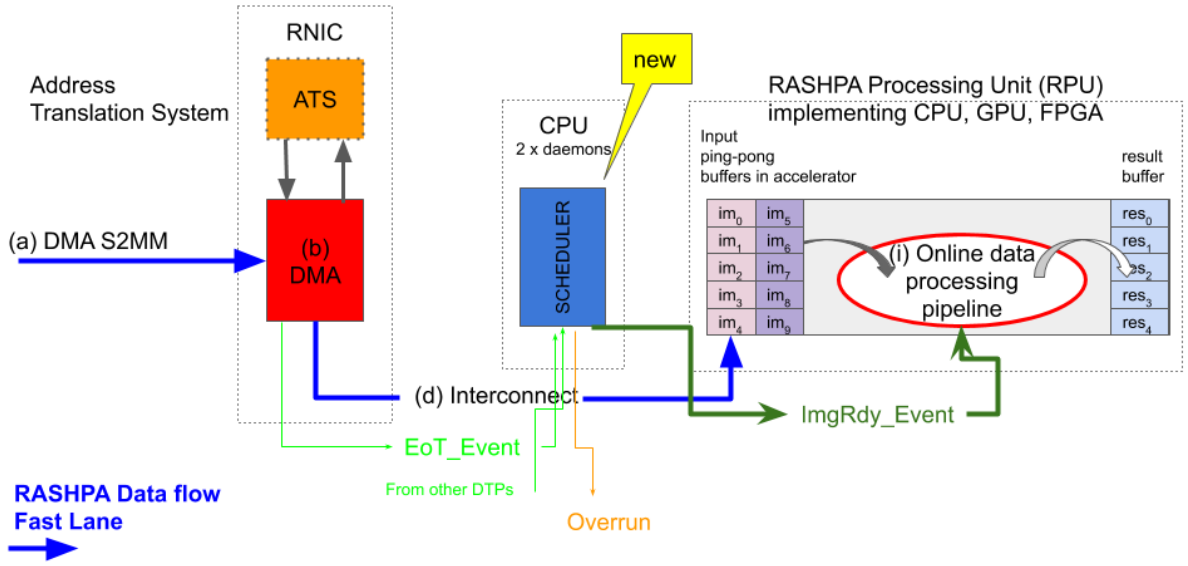


Figure 4.1: General overview of a RASHPA Processing Units (RPU). This is a three-tier system: RNIC, GPU, CPU. Major contribution of the work, the RASHPA Scheduler daemon manages the RNIC and consolidates ingress events. The user application controls the data processing and movements on the GPU. The Address Translation System (ATS) performs the virtual to physical address translation.

for any associated RASHPA sources. Thus the multiple module events are consolidated into a global one.

In addition, the RS is also entitled to enqueue asynchronously commands both to the RNIC. This point is implementation dependent, but existing RNICs cannot be configured only once at startup, and cannot remain autonomous for the total duration of the data processing. A periodic monitoring and reconfiguration of these devices must be performed.

The RS must also detect *data processing overrun*, when a data transfer is completed before the end of the previous data processing and thus overwhelms the coprocessor. However, unidirectional transfer, from detector to RPU, is a key paradigm of the RASHPA design. Therefore there is no foreseen way to mitigate such overrun issue. But the main monitoring system, the RASHPA manager (RM), must be notified of the incident and will pause the acquisition process.

A user application is in charge of the data processing itself, only the scheduling and the data transfer is on the responsibility of RASHPA.

This user application launches from the host computer a sequence of commands towards the accelerator. Depending on the hardware programming model, these commands are executed immediately or asynchronously. Four types of commands are identified: i)

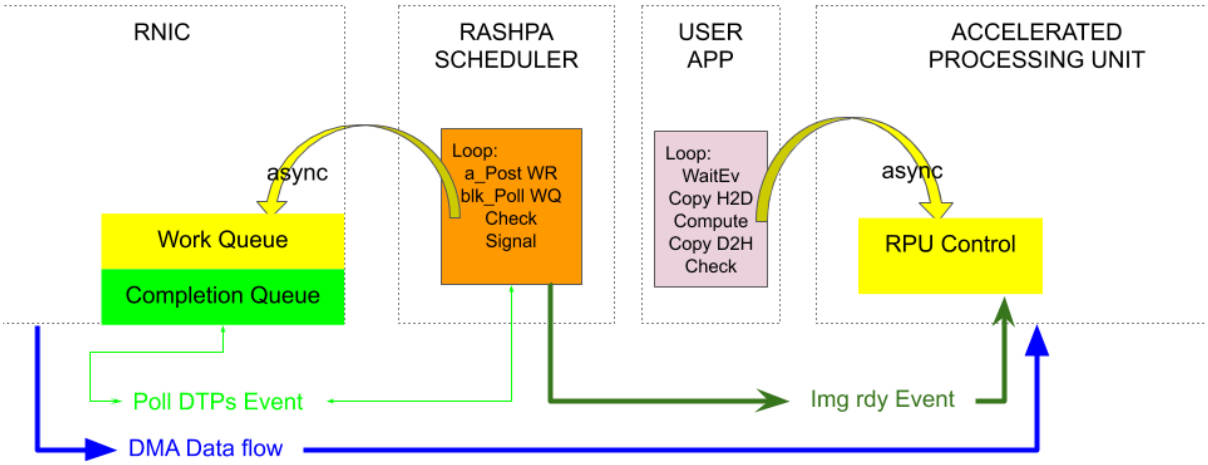


Figure 4.2: Details of the workflow for the RASHPA Scheduler (RS). It has several functions: configures and handles the events of the RNIC, triggers the accelerator when data are ready.

data transfer from the host memory to the RPU, when required ii) data processing on the RPU, iii) data transfer of the results from the RPU back to the host, iv) synchronization point. On compliant devices, the asynchronous commands are en-queued until execution. For the sake of efficiency, on the accelerator side, this sequence of operations might be turned into a multiple stage pipeline if data transfer and data computing can overlap.

Mitigating data transfer costs this way requires to allocate ping pong buffers along the data path and in the internal memory of the accelerator. Such bounce buffer must also be DMA compliant (pinned host memory).

RASHPA back end receiver (ROMULU)[1], has been implemented as a proof of concept of a data processing application. It is designed to receive multiple data transfers, consolidate events and trigger data processing. During an initialisation phase, it also performs the allocation of all the bounce buffers. The implementation details vary according to the link and the coprocessor hardware. This is a multithreaded application: one thread is dedicated to data transfer, a second one is controlling the actual data processing. A third one is used for monitoring purpose by the RASHPA manager. The RASHPA PCI-e version is a variant of this generic scheme.

[1]so called because it is the REMU brother and companion

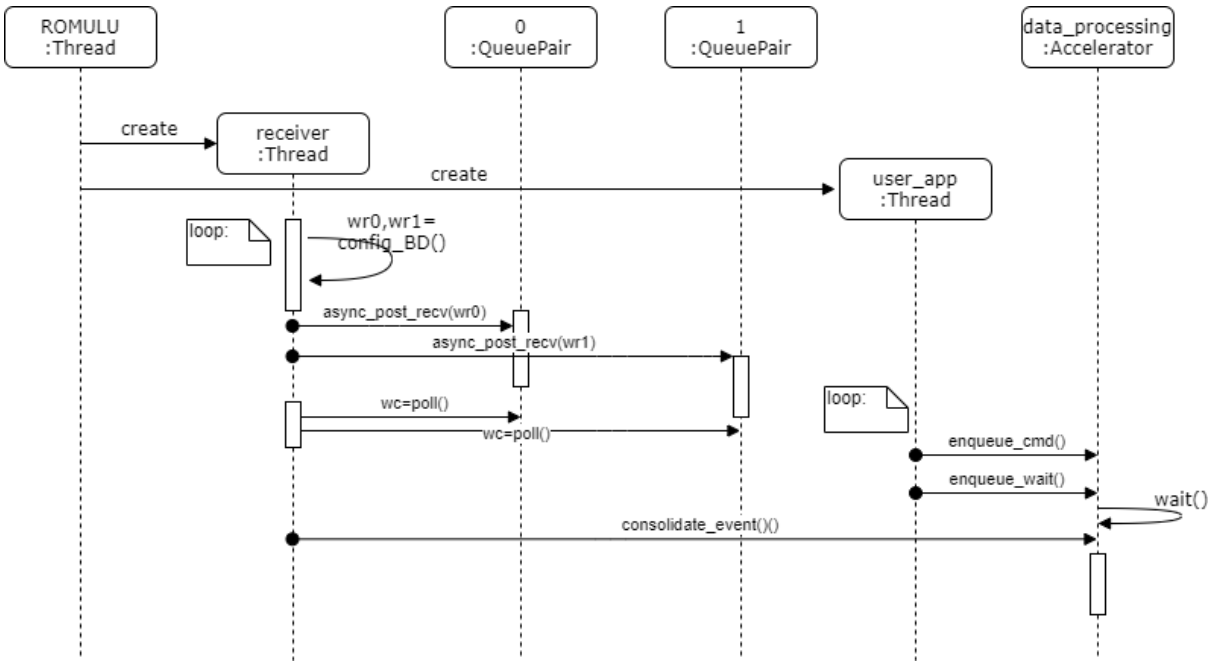


Figure 4.3: UML Sequence diagram of ROMULU, a typical RASHPA Processing Unit. The application controller creates two threads: one is managing the queue pairs associated to the RPU and consolidating the end-of-transfer events. The second one is dedicated to the configuration and scheduling of the data processing.

4.1.2 Address Translation System

The proposed Address Translation System (ATS) aims to provide the physical address of the internal memory used in an accelerator and used during DMA transfer. Such operation requires the execution of special system calls from kernel space. Therefore several kernel modules for the Linux operating system where developed.

- In the case of RoCE protocol, the physical addresses are requested by the RDMA driver *mlx_ib* using the PeerDirect technology[2]. This operation takes place during the memory registration by the *ibv_reg_mr* function from the *ibverbs* library and automatically searches for a kernel module specific to the accelerator device and capable to perform the address translation. These drivers are registering to the RDMA driver mlx_ib during the call to *ib_register_peer_memory_client*. When an application tries to register a new memory address, the RDMA driver iterates over the peer drivers and requests if the given address is in their memory range. In this case, the owner *acquires* the memory region and does the necessary work.

  The open-source *nv_peer_mem*[3] module is dedicated to Nvidia Quadro and Tesla GPUs. AMD Pro GPUs are supported natively by the open-source Radeon Open

[2]https://community.mellanox.com/s/article/howto-implement-peerdirect-client-using-mlnx-ofed
[3]https://github.com/Mellanox/nv_peer_memory

Compute platforM (ROCm) driver[4]. A custom kernel module called *fpga_peer_mem*, for FPGA boards used as computing accelerator has been implemented based on the model of the latter.

- In the case of PCI-e interconnect, the physical addresses are obtained in a similar way thanks to proposed modules. The *libcdma* module is a memory allocator for large contiguous memory buffer in central memory leveraging Linux CMA and using *dma_alloc_coherent* system call presented in chapter 2. It provides the requested physical address at the time of the memory allocation.

  The proposed *libgdma* module enables to get the physical address of a memory allocation performed in a Nvidia GPU memory by the CUDA driver. It is leveraging *nvidia_p2p_get_pages* and is inspired from gdrcopy driver[5]. Similar technique is available for AMD GPUs using *amdkfd_query_rdma_interface*.

  Then, those physical addresses are passed by the user application to the DMA controller.

  By the way, it is worth to note that in the case of a RASHPA processing unit in central memory, the addresses in use by a RoCEv2 RASHPA detector are the user land virtual addresses. It is technically possible to use directly the physical addresses in host with RoCE but it is considered as a security concern. Therefore, it is not available as is in the standard software stack. A recompilation of the code source is required to enable the features.

4.1.3 CPU Accelerated RPU

A bare-bone workstation or server can also serve as RASHPA Processing Unit. In this case no extra movement are needed from host to accelerator, but data processing still requires scheduling and double buffering and the generic principles apply.

Modern CPUs have multiple processing cores available for data processing and movements, especially processors used in servers. Each core can take benefit also from Single Instruction Multiple Data (SIMD) instruction and multithreading. The throughput and computational performances are tightly related to connectivity: number and width of data memory channels and contention on locking mechanism as well.

Such CPU-RPU is put on hold by a shared memory lock that is triggered by the RASHPA Scheduler when a consolidate event occur.

A benchmark application performing element wise data processing on image data has been developed using an OpenMP compiler. Its evaluation is presented in subsection 4.3.2.

[4]https://rocmdocs.amd.com/en/latest/Remote_Device_Programming/Remote-Device-Programming.html

[5]https://github.com/NVIDIA/gdrcopy

4.1.4 GPU Accelerated RPU

A GPU accelerator is an OEM black box that can only be used using the vendor driver and API. Its internal operating system, task scheduler, memory management system, etc are not exposed directly and partially configurable by user.

The general working principle of an RPU applies also to a GPU accelerator. With GPUs that do not support GPUDirect technology and the like, the RASHPA source will transfer the data into CPU memory. Once in CPU memory, the data will then be transferred into the GPU internal memory. Data transfers between CPU and GPU memory might be inefficient from virtual memory. *Pinned host memory* exhibits better performances but it might be a scarce resource and it must be allocated during initialisation phase as in takes time.

The GPU device memory may also be directly accessed by the RNIC (peer-to-peer DMA), removing the transfer step from CPU to GPU memory and decreasing latency.

One key point for efficient GPU-RPU implementation, is the optimization of the data-transfers from CPU to GPU memory. The proposed implementations leverages *CUDA Streams* and *Stream Memory Operation*. CUDA streams are sequences of operations, either data transfer from host computer to GPU device or kernels. These commands are asynchronously en-queued in the stream FIFO (until it is full) and executed one by one. Different streams can possibly execute in parallel.

It is proposed to create three streams in the GPU:

- H2D: data transfer from CPU to GPU. This transfer can be skipped if GPUDirect is enabled.
- Kernel computation, using the standard double buffering scheme
- D2H: data transfer from GPU to CPU for the results. This data transfer can overlap with kernel execution if performed on pinned memory. H2D and D2H might be concurrent as there are two DMA engines, when there are a enough PCI-e lanes left free by other devices.

This transforms the previous sequence into a three-stage pipeline in which transfers overlap with computations as shown in 4.4. The GPU application implements several double buffers so that the ongoing transfer is carried out in one buffer while computation is alternatively carried out in the second buffer.

The proposed synchronization mechanism aims at decreasing launch task overhead performed by using the CUDA driver. The processing tasks or data transfers are en-queued in multiple streams, so they are ready to be scheduled by the GPU Streaming Multiprocessors. But the execution of the stream is blocked, put on hold until being triggered by the end of RDMA data transfer. The proposed memory lock mechanism is relying on the CUDA stream memory operation called *cuStreamWaitValue32* and is seldom described in the literature.

This waiting operation is en-queued in the streams after each GPU tasks. Upon notification by the RASHPA event system, the CPU control application in turn triggers the concurrent execution of GPU streams using the *cuStreamWriteValue32* operation.

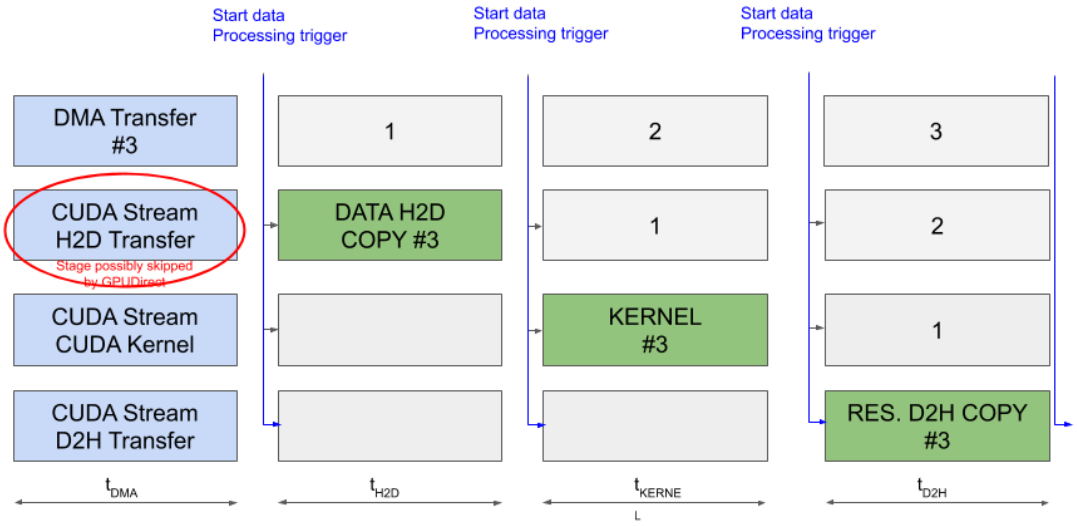


Figure 4.4: The GPU processing pipeline: from top to bottom: DMA transfer, CPU to GPU transfer, data processing, GPU to CPU transfer. The pipeline involves three heterogeneous systems: RNIC, CPU and GPU. The RNIC cannot directly synchronize the GPU. Each stage in the execution pipeline is synchronized by an RDMA event or conditionally by the veto kernel result

The memory lock is shared both by CPU and GPU. The lock is allocated in pinned CPU memory by *cudaHostRegister* CUDA function, then using *cuMemHostGetDevicePointer*, it is made accessible from the GPU device.

This mechanism removes the launch overhead detrimental to real-time data processing, as shown in Figure 4.5. Hence we measured a total kernel launch time of $4\mu s$ in place of $40\mu s$.

The code snippet presented in Listing 4.1 exhibits the proposed CUDA pipeline and synchronization mechanism between CUDA streams:

4.1.5 FPGA Accelerated RPU

Field Programmable Gate Array (FPGA) are the most configurable devices available as of today.

FPGA boards for data processing include a PCI-e interconnect and gigabit Ethernet as well. Ingress data flow may used one or the other. In our approach, we have used FPGA accelerators the same way we were using GPU accelerators. The processing model already applied to CPU and GPU is again applied to FPGA coprocessors. The data pushed by the detector are received by a RNIC attached to the workstation and then are offloaded by the internal DMA engine to the accelerator internal memory through the PCI-e bus.

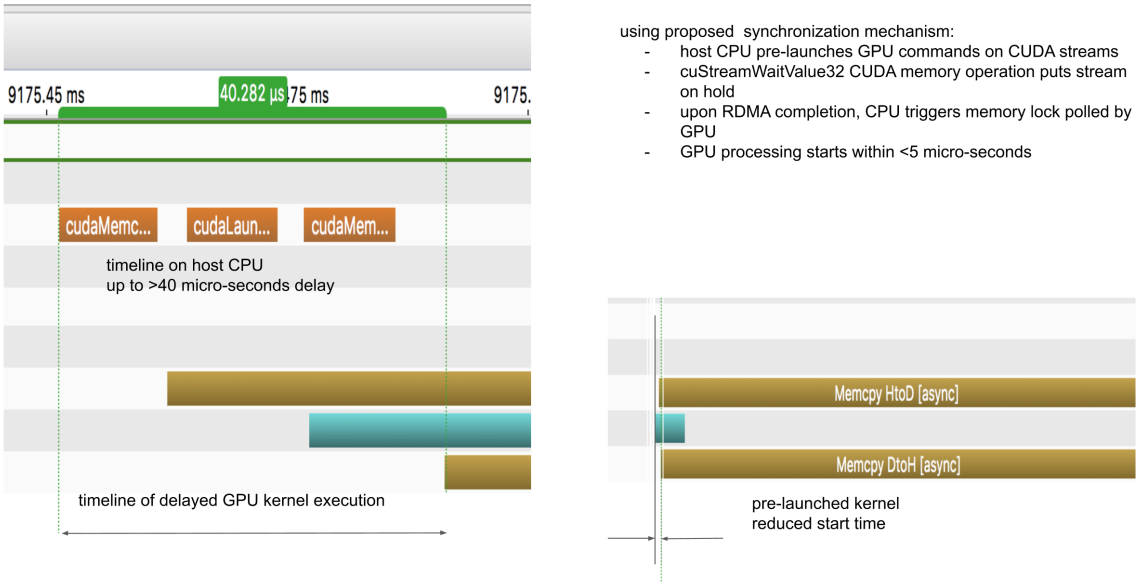


Figure 4.5: NVIDIA visual profiler snapshot of tasks launch times. Implementing standard launches from the host upon completion (left) causes driver overhead in the range of tens of microseconds per operation. Using CUDA stream memory operation (right) decreases latency to a few microseconds.

A user defined application is in charge of the orchestration on the host computer of the data processing. It is triggered by the RASHPA scheduler daemon that is consolidating the end of transfer events occurring at the end of each stack of images.

Some high-end FPGA boards such as the Xilinx Alveo U200 feature also 100 Gigabit Ethernet connectivity. With such boards, it would be possible to process directly the data flow inside the FPGA, without any intervention of the CPU system. The host would be merely used as a storage system.

Such design would be perfectly valid and efficient. It differs slightly of our implementation choices as the RNIC RoCEv2 compliant engine should be synthesised inside the FPGA. This block could be implemented using a XILINX ERNIC IP. We did not investigate further the Vitis Development Kit promoted by Xilinx. This is a full-fledged solution mitigating all the burden of low level development and interface implementation between host and accelerator. The host code is written in C or C++ language using the standard OpenCL and the accelerator code is seamlessly deployed on the reconfigurable portion of PCIe based Xilinx accelerator cards.

In order to process to the evaluation of the integration of RASPHA in such FPGA accelerators, a custom design has been proposed and is described in the next section.

Listing 4.1: This host application en-queues tasks on GPU streams, put on on hold until being triggered by a RASHPA global event. d_statusFlag is the shared memory lock.

```
1
2  for (i=0; i < nImages; i+=2) {
3      //H2D data transfer on stream s3
4      cuStreamWaitValue32(s3, d_statusFlag, i, ...
5      cudaMemcpyAsync(d_InAbis, ...
6      cuStreamWriteValue32(s3, d_statusFlag, -1, ...
7      cuStreamWaitValue32(s3, d_statusFlag, i + 1, ...
8      cudaMemcpyAsync(d_InBbis, dest_addr + buffer_size/2),
9      cuStreamWriteValue32(s3, d_statusFlag, -1, ...
10
11     //kernel data processing on stream s2
12     cuStreamWaitValue32(s2, d_statusFlag, i, ...
13     applyGainPedestalandCount<<<...>>>(d_InB, d_OutB, ...
14     cuStreamWaitValue32(s2, d_statusFlag, i + 1, ...
15     applyGainPedestalandCount<<<...>>>(d_InA, d_OutA, ...
16
17     //D2H data transfer on stream s1
18     cuStreamWaitValue32(s1, d_statusFlag, i, ...
19     cudaMemcpyAsync(h_output1 + offset, d_OutA, ...
20     cuStreamWaitValue32(s1, d_statusFlag, i + 1, ...
21     cudaMemcpyAsync(h_output2 + offset, d_OutB, ...
22     }
```

4.1.5.1 Proposed FPGA design

The proposed design of a RASHPA Processing Unit based on FPGA is shown Figure 4.6. It is built using Xilinx IP and the Advanced eXtensible Interface (AXI) interconnect. It includes a Xilinx XDMA IP, a DMA controller tightly coupled to a PCI-e Bridge. DDR4 internal memory bank is interconnected by an Advanced eXtensible Interface (AXI) that is the standard interconnect of Xilinx IPs. The host controlled DMA engine is used to output the results from the accelerator to the host. The PCI-e bridge is configured as a *bypass* to the internal AXI bus. Thus, a third party DMA engine, the one in the RNIC, can push the ingress data stream into the accelerator. A Microblaze soft-processor has been added, solely used for debugging purposes.

The host computer is in charge of the configuration of the control registers of the data processing IP by the AXI-Lite interface. It is in charge of triggering data processing in the same way.

4.1.5.2 HLS Kernel and Host Application

A custom IP block performing some complex raw data processing for variable gain detector is proposed. It is an High Level Synthesis (HLS) engineered IP using Xilinx Vitis_hls

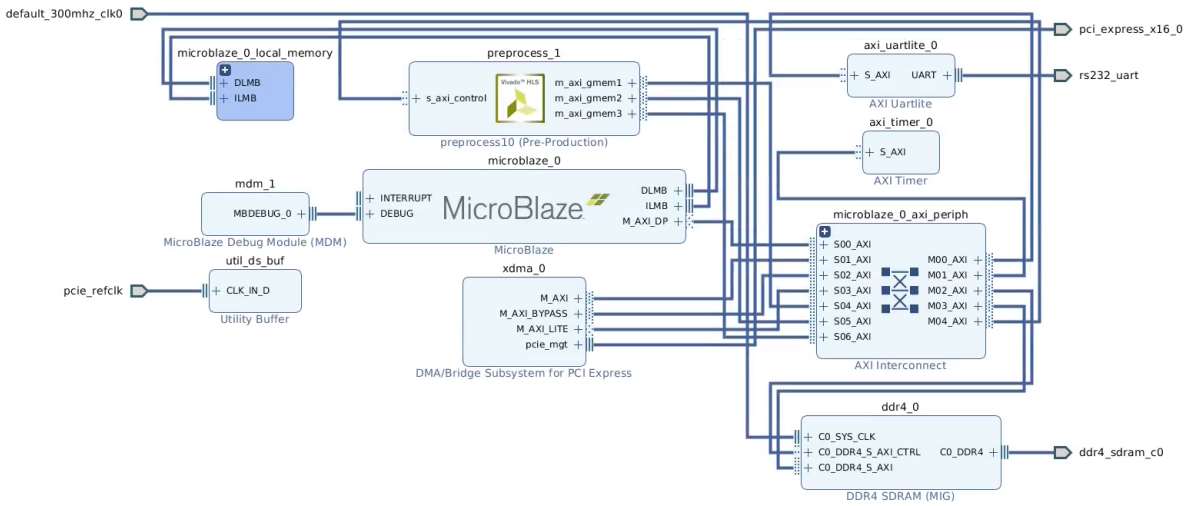


Figure 4.6: Overview of a RASHPA Processing Unit based on FPGA design. It includes an XDMA engine, a PCI-e Bridge, 16GB of DDR4 internal memory, and an HLS engineered data processing IP. The Microblaze soft-processor is used only for debugging purposes and for envisioned automatic data processing.

tool. This IP has access to the data in the internal DDR4 by AXI bus as Master. It is presented in Listing 4.2.

The RASHPA subsystem has previously filled this memory region using RNIC peer to peer DMA. The data processing is then performed by a host application, which schedules the three subsequent steps of the HLS block.

Listing 4.2: Snippet of a data processing IP with AXI-Master interface.

```
1  void data_processing(u16 raw[N], float res[N])
2  {
3  //creates an interface with control register
4  //used by the host user application to set
5  //the address of the raw data and to trigger the data processing
6  #pragma HLS INTERFACE s_axilite port=raw          bundle=control
7
8  //the raw argument uses AXI-Master interconnect to the DDR.
9  #pragma HLS INTERFACE m_axi port=raw     depth=N u16 _raw[N];
10
11 //memcpy instruction proceeds to a burst copy of data
12 //from DDR4 in the DDR board to BRAM inside the FPGA.
13 memcpy((void*)_raw,      (void*)raw,      N*sizeof(u16));
14
15 //... data processing here...
16
17 //burst memcpy of the result back to the internal DDR4.
```

```
18 //Then it will be transferred by the XDMA back to host memory.
19 memcpy((void *)res,(void*)_res,N*sizeof(float));
20 }
```

The host CPU application, which a code snippet is presented in Listing 4.3, uses the Linux XDMA driver to access the FPGA board resources.

Listing 4.3: Snippet of the host application triggering the data processing on FPGA.

```
1
2 //access AXI-Lite to set arg value
3 //using control registers of the HLS IP
4 int fd1 = open("/dev/xdma0_user",...
5 void *hls_ip_base = mmap(fd1,...
6 *(u_int32_t*)(hls_ip +0x10) = 0;
7 *(u_int32_t*)(hls_ip +0x14) = 4;
8
9 //mmap internal DDR
10 //perform a mapping of the FPGA internal memory
11 //in host user-land application
12 int fd2 = open("/dev/xdma0_bypass", ...
13 addr = mmap(memorySize,fd2,...
14
15 //trigger IP start
16 //the bit 0 of the CR register
17 //is set to start the data processing
18 //by the HLS IP block
19 *(u_int32_t*)(hls_ip +0x00) = 1;
```

4.2 Image Processing for SSX Experiments

In this chapter, we present how we propose to address the challenges encountered by EBSL8 staff in the new experimental facility performing SSX experiments. The foreseen setup shown Figure 4.7, due to late 2021, comprise a automatic sampling and a new generation detector. EBSL8 setup is taken as an example in order to compare realistic figures. It will use a high-end PSI Jungfrau 4M detector, described in [41], and generating up to 16 GB of data per second, operating continuously during several minutes.

While limiting the scope and the complexity of the GPU algorithm in use, we were able to check many important high-throughput data transfer issues seldom addressed in the literature as kernel launch time without the distractions of numerical algorithms problems.

Synchrotron serial crystallography (SSX) is among the most-demanding cases of photon science in terms of data throughput. In a typical SSX experiment, a liquid crystalline polymer (LCP) jet propels micro-crystal samples in a pulsed X-ray beam. In the case of EBSL8, a rotating chopper produces X-ray pulses synchronous to the data-acquisition

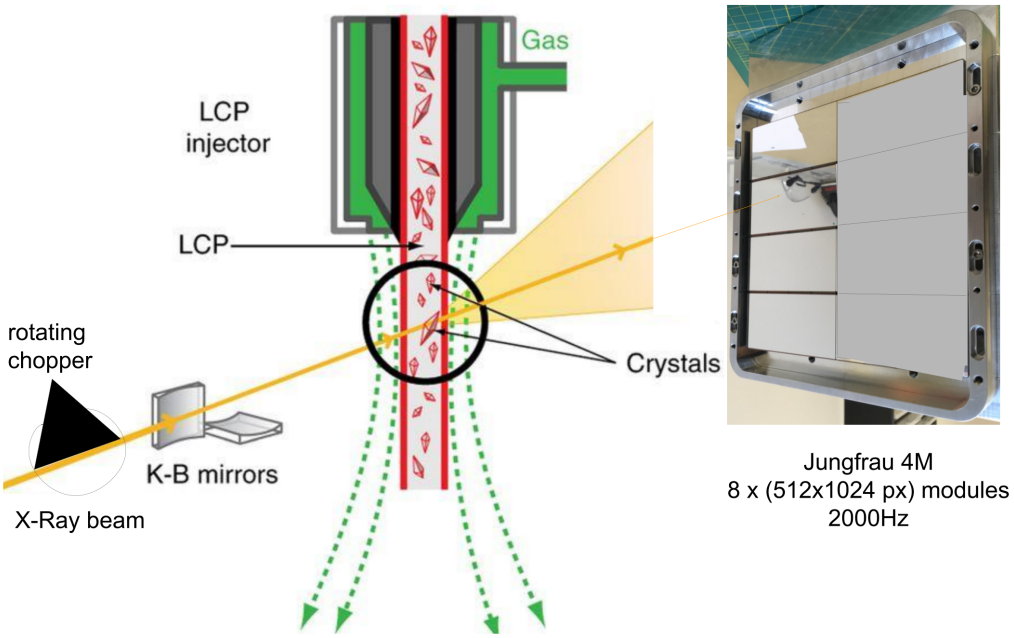


Figure 4.7: An example of Serial Synchrotron X ray crystallography (SSX) setup. The main components are the LCP crystal jet, the JUNGFRAU 4M detector and the rotating chopper (Sources: Daniele De Sanctis, ESRF and Aldo Mozzanica, PSI)

system. This enables the collection of alternate dark and signaling images at a maximum rate of 2000 images/s. When acquiring 4M pixel 16-bit images, such a high repetition rate will result in a 128 Gb/s data stream. This will produce nearly 1 TB of raw data in 1 min. Continuous operation requires an efficient online data-reduction scheme.

This cutting edge setup is foreseen to use a new generation pixel detector called JUNGFRAU. This modular detector will be the reference in the frame of this evaluation. A rear view of the modules showing the connectivity (two 10 Gb/s optic fiber links per module) is presented in Figure 4.8.

The JUNGFRAU detector was developed at the Paul Scherrer Institut (PSI, Switzerland). It was initially designed for X free-electron laser experiments (XFELs) but its characteristics are well adapted to other applications with pulsed beams, such as the SSX experiments. A specific characteristic of the JUNGFRAU detector is automatic selection of the gain level for each individual pixel, depending on the signal detected. The photon count is derived from the energy deposited in each pixel, which has to be computed from the raw digital value by subtracting a pedestal offset of a previous dark image and dividing by a gain factor. Three different pedestal values and gain-factor values are needed for each pixel. Thus, this adds up to 24 million different correction coefficients for a 4M detector. Hence, the processing of a single JUNGFRAU 4M raw-data frame to produce a final image requires 4 million 16-bit integer subtractions and 4 million 32-bit floating-point divisions.

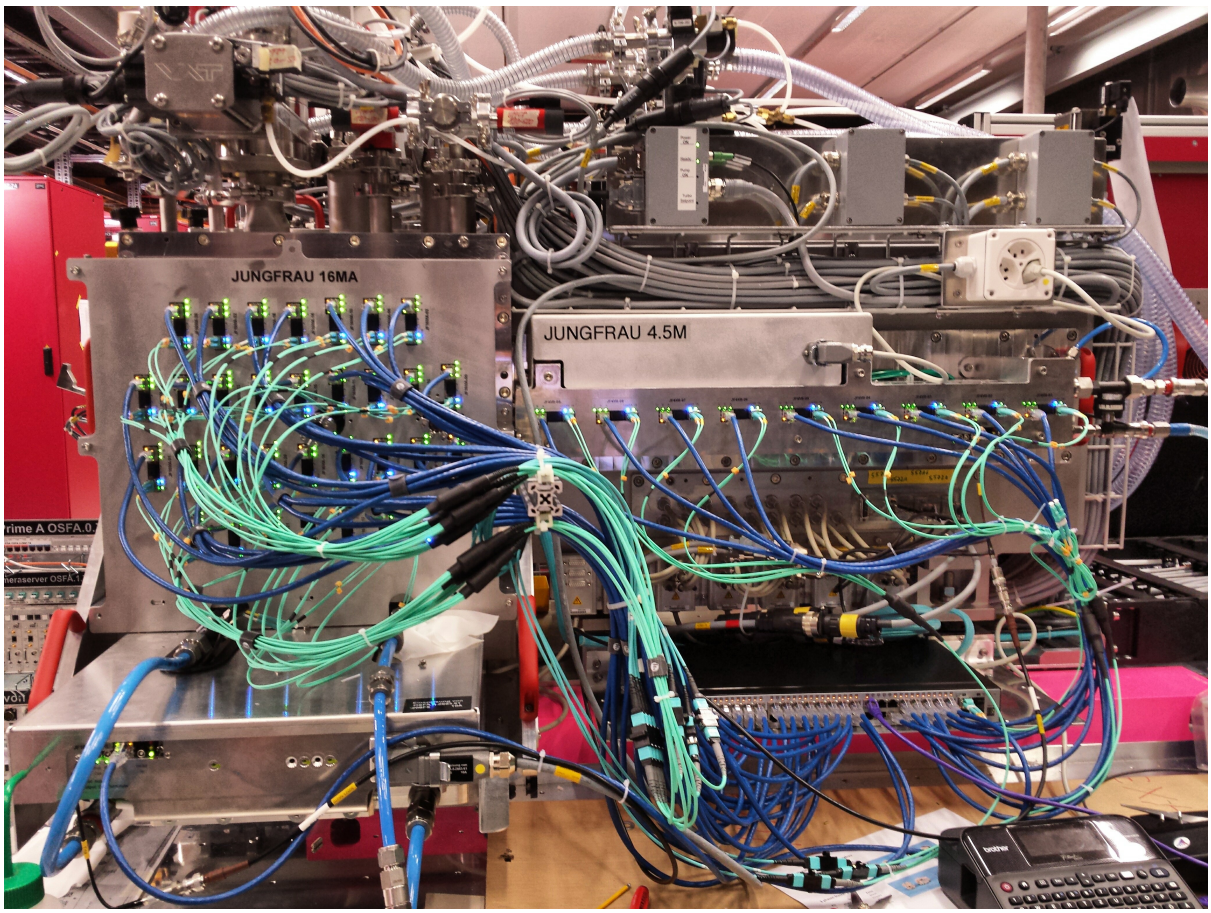


Figure 4.8: Rear view of the Jungfrau 16M - 32 modules on the left (and JF 4.5M - 5 modules right) showing the connectivity including two 10Gb/s optic fiber per modules (aqua blue) + one 1 Gb/s control link (blue). (Source: Fillip Leonarsky, PSI)

4.2.1 Raw-data Pre-treatment

The JUNGFRAU detector has three gain levels for each pixel, stored in two of the most significant bits of the raw data. The pixel value is stored in the 14 remaining bits as an integer. Therefore, the conversion to the integrated charge is given by the formula:

$$Pixel_{i,j}\,[keV] = \frac{(Raw_{i,j}\,[ADU] - Pede_{i,j}\,[ADU])}{Gain_{k,i\,,j}\,[\frac{ADU}{keV}]}$$

where $Pixel$ is the actual energy that was deposited in the pixel during the exposure, i, j are the pixel coordinates, $Gain$ the constant coefficient matrix, k is the gain level, Raw is the raw data and $Pede$ the pedestal, dark image transmitted just after the image with signal, both expressed in Arbitrary Detector Units (ADU).

This computation is inherently parallel. The gain factors are stored in a large constant dataset, i.e. they remain in the GPU memory, and only require one transfer from the CPU memory at the beginning of the process. Pedestal data are acquired interleaved

with image data when the chopper is cutting the beam. These dark images are used to update the pedestal value for the highest gain. For the two other possible gain regimes, their pedestal value is expected to be constant.

4.2.2 Data Rejection

One simplistic rejection algorithm counts the intense pixels (considered as Bragg's peaks) in each image, without noise correction or outlier removal. A first threshold defines the signal needed for a pixel to be counted as a Bragg's peak, while a second threshold defines the minimum number of such pixels in an image to be accepted. Such counting has been implemented using atomic addition and shared memory. When a non-pertinent image is rejected, the third step does not happen. Therefore, the result of the veto kernel algorithm should conditionally trigger the transfer. As the condition is not evaluated prior to the launch, we use another stream memory operation lock triggered by the veto function. A credit-based algorithm controls the number of queued transfers, since the queue size is limited.

The expected compression factor of 100x would allow to input the data using 100Gb/s links and saving the data via a simple gigabit links on a distributed file system.

4.2.3 Compression to Sparse Matrix

While the value provided by compression is widely recognized, its application is often limited because of the high processing cost and the resulting low throughput and high elapsed time for compression intense workloads. For demonstration purpose, we have implemented in the online processing pipeline a rudimentary loss-less compression algorithm. Like the previous one, it is counting pixels over a specific threshold, assuming it is a Bragg's peak.

The compression to the Compressed Sparse Raw (CSR) matrix format is well adapted to scattering images, benefiting of their high level of sparsity.

Our code benefits from the seminal work of [6] who performed a comprehensive review of the main algorithmic patterns encountered in parallel data processing. The *cumulative summation* also called *scan*[6] has been used in the proposed sparse matrix conversion.

The meta-programming capabilities of PyCUDA [26] were used to generate the CUDA code of the cumulative-sum algorithm used in CSR matrix compression. The storage size of the result matrix was chosen a priori given the expected dataset sparsity.

4.2.4 Azimuthal Integration

The first attempt to use a GPU at the ESRF was related to diffraction tomography where millions of detector 2D frames are acquired and subsequently reduced to a 1D spectra. This reduction is called Azimuthal Integration (AI) and it is roughly computing the average value of all pixels on a given radius, called *Debye Scherrer cones*. This gave later

[6]https://developer.nvidia.com/gpugems/gpugems3/part-vi-gpu-computing/chapter-39-parallel-prefix-sum-scan-cuda

birth to the pyFAI project. Today, AI for powder diffraction and small angle scattering experiments is in production on several beam-lines at the ESRF. PyFAI provides the same numerical results on CPU and on GPU and the latter is about 20x faster. This advance of GPUs still holds today and pyFAI is now the reference implementation for azimuthal integration and is used in all X-ray radiation facilities.

The proposed algorithm[7] has been implemented porting the work of the ESRF Data Analysis Unit [4] written in OpenCL language to CUDA. It implements atomic additions and a cumulative sum in shared memory and Kahan double summation to mitigate error propagation. We will not develop this subject further outside the scope of this memoir.

4.2.5 Ultra Low Latency Control

Two techniques are considered here to reach ultra-low latency required in foreseen control application. The first one applies to GPUs even if they have a hardware design more adapted to bandwidth performances than to low latency challenges. The second mockup is geared towards FPGA systems. The latency is the time take from an image acquisition to the completion of the data processing in the RPU. Our system being by design unidirectional, it would be tricky to measure this value. Thus, we are discussing here only the general implementations guidelines

A *Persistent Kernel* is infinitely running on the GPU. Therefore, there are no task launch overhead during the online processing of long lasting experiments. It is working with a double buffer and triggering mechanism as in the previously mentioned mechanisms. The proposed implementation is inspired by the work of R. Crovella [8]. To achieve good performances, there are application specific trad-off to mitigate between the size of the block and the grid of the kernel in use and the available GPU resources. This could lead to sub optimal implementations in case of subsequent kernels with different block sizes. Another drawback is that the GPU has 100% utilization even when the application is idle waiting for data transfer completion.

The possibility and convenience of using FPGAs for certain tasks has been demonstrated for various types of low latency of on-line image correction as well as data analysis algorithms such as time autocorrelators for instance, but the lack of suitable development and implementation environments has prevented its use for synchrotron radiation applications.

A test bench is foreseen using an FPGA design.

4.3 Outcome

We have developed many pieces of software to demonstrate the feasibility of our approach in implementing RPU. We did not had performance evaluation in mind at the time of this work. This task was foreseen in a later step and has been delayed due to the current pandemic.

[7] https://github.com/silx-kit/pyFAI/blob/master/pyFAI/resources/openCL/ocl_azim_CSR.cl

[8] https://stackoverflow.com/questions/33150040/doubling-buffering-in-cuda-so-the-cpu-can-operate-on-data-produced-by-a-persiste

4.3.1 Methods

The previously mentioned testbenches (subsection 3.0.1) are completed with GPU and FPGA:

- a NVIDIA QUADRO P6000 GPU accelerator is added to the first system for the assessment of RoCE and GPU processing using GPUDirect.
- The second test bench is dedicated to PCI-e RASHPA. A XILINX ALVEO U200 FPGA board was already installed in the x16 slot. A NVIDIA QUADRO RTX6000 GPU is added and a Connectx-4 as well. Unfortunately, both GPU an RNIC are installled in a PCIE gen3 x4 slot . Consequently, throughput performances are capped below 2.8 GB/s from CPU to GPU side. In addition, both cards are not installed in the same PCIe interconnect, which is detrimental to GPUDirect performances as data must go through the QPI inter-processor link. Xilinx hardware server is used to remotely program the FPGA board with the implementation bitstream.

4.3.2 Results with CPU / OpenMP

For comparison purpose, we present the results obtained using an OpenMP multicores application performing the Jungfrau raw data pre-processing computations. A 12 cores workstation is not sufficient to process the *embarrassingly* parallel workload of detector raw data pre-processing and the throughput is limited to 2.8 GB/s before overwhelming the computer.

The Listing 4.4 presents the OpenMP code skeleton.

We did not yet evaluate programming neither the Intrinsics instruction set of the Intel processor that implements the processor vectorized instructions (same instruction on multiple data) nor the Intel *icc* compiler.

Line 3 the directive *#pragma omp for* unroll the data processing loop on multiples cores for concurrent processing.

Line 5 the *#pragma omp simd* has been also evaluated but without significant improvement.

4.3.3 Results on GPUs

A RASHPA test bench compatible with high-throughput detectors has been upgraded with a GPU accelerator. Results are shown in Figure 4.9. The kernel execution overlapping data transfer is shown Figure 4.10.

4.3.3.1 NVIDIA CUDA Evaluation

The Table 4.1 presents some timing observed with the proposed system. It must be noticed that these numerical results depend greatly on the sparcity of the data set used for the test and merely rely on the RASHPA system performances.

Listing 4.4: Snippet of an OpenMP CPU application performing raw data pre-treatment foreseen for an adaptive detector (Jungfrau).

```
1  void cpuapplyGainPedestalandCount(...)
2  {
3  #pragma omp parallel for
4  for (size_t j = 0; j < height; j++)
5      #pragma omp simd
6      for (size_t i = 0; i < 1024; i++)
7      {
8          float value = 0.f;
9          uint16_t p = 0.f;
10         uint16_t raw = data[i+j*width+2*(N%b)*width* height];
11         uint16_t gn = raw >> 14;
12         if (gn == 1)
13         {
14             p = data[i+j*width+(2*(N%b)+1)*width*height];
15         }
16         else
17             value = ((float)((raw & 0b0011111111111111) - p))
18                     / gain[i+j*width+gn*width*height];
19         outputImage[i+j*width+N*width*height] = round(value);
20     }
21 }
```

It is possible with specific high-end GPUs (Quadro and Tesla series) to decrease the transfer latency by directly transferring data from the FPGA card to the accelerator memory (Peer to Peer PCIe DMA transfer).

It appears that GPUDirect throughput might be limited on some hardware by PCIe root complex poorly handling peerto-peer transfer [55]. This explains why NVIDIA is purposely integrating PCIe-switch chips into their high performance DGX computers as a workaround for this issue.

The maximum possible speed for data processing is defined by the minimum of: (i) RNIC to CPU speed, which is related to the length of the packet on the network (the width of the region of interest in the images that are sent), (ii) CPU to GPU speed, which depends on the size and number of images processed in the GPU. It is wise to transfer either a large image or a bunch of small images.

For comparison purposes, Figure 4.11 also exposes the performance of the ZeroMQ protocol, even if it relies on TCP/IP and therefore might not be easily embedded in a detector FPGA. The latest version of the CrystFEL software suite for SSX experiments has recently been upgraded with a ZeroMQ/MessagePack interface [65]. It appears that ZeroMQ is not well adapted to RDMA hardware and performances stay below 6 Gb/s depending on the selected messaging pattern: request/reply as in CrystFEL.

For good performances in workstation with multiple processors and attached PCIe

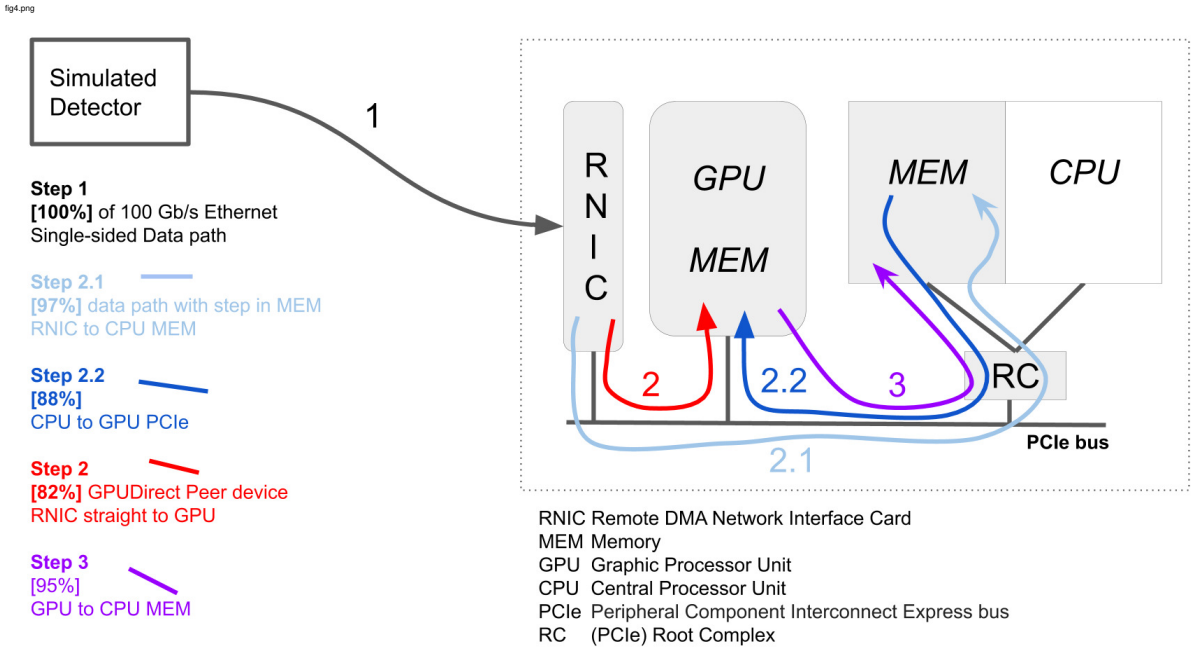


Figure 4.9: An overview of a GPU RASHPA processing Unit. The maximum achieved throughput is shown along data paths. Because of the lack of existing RDMA-compatible detectors, the detector was simulated by a workstation. The RDMA network card is a Connectx-5 from Mellanox Technologies. The processing computer embeds a NVIDIA QUADRO P6000 GPU accelerator. The GPUDirect technology (step 2) allows bypassing the extra copy in the central memory (steps 2.1 and 2.2).

interfaces, RNIC and GPU must be connected in the same PCIe bus (and controlled by a CPU core with the good affinity).

4.3.3.2 AMD OpenCL Evaluation

A validation of OpenCL code in the frame of RASHPA was evaluated. A code snippet of the synchronization between data transfer and GPU data processing using a *mutex* is shown in Listing 4.5.

To the best of our knowledge, there is no synchronization mechanism available with OpenCL queues similar to that was used in the CUDA version.

4.3.3.3 REMU PCI-e / GPU RPU evaluation

The proposed design for RASHPA over PCI-e long distance is a FPGA based design described in section 3.2. Observed measurement from FPGA to CPU an GPU data transfer results are shown in Figure 4.12. The poor bandwidth results using GPUDirect technology might be explained by several hardware flaws in our test bench: i) FPGA and GPU not in the same NUMA domain, ii) GPU is in PCI-e gen3 x4 slot, iii) the server has

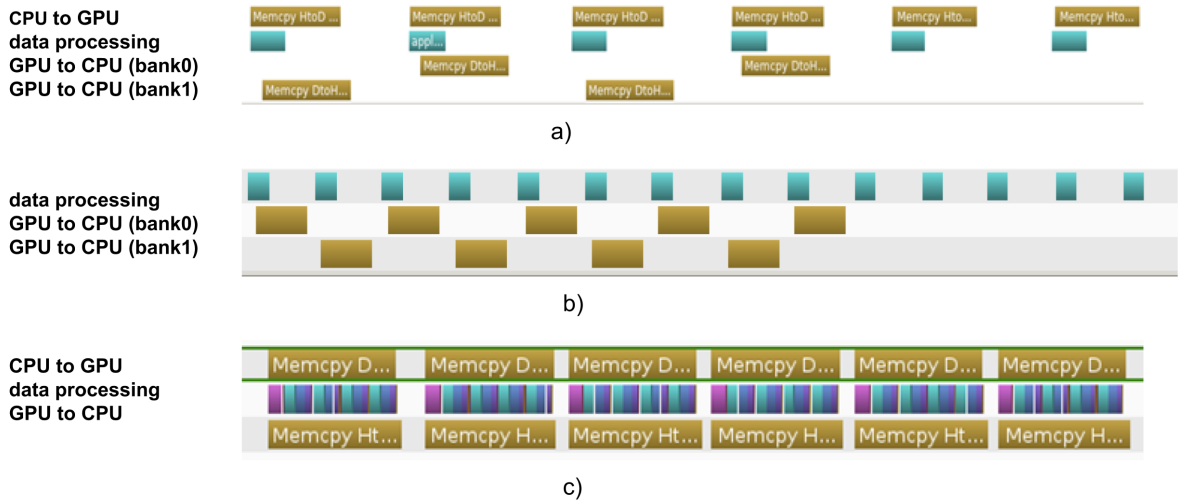


Figure 4.10: Snapshot of NVIDIA Profiler of CUDA Stream execution. In a) Four streams are overlapping and conditional storage is exhibited. Bank0 & 1 are ping-pong buffers used for storage. b) Thanks to GPUDirect technology, H2D step could be skipped. c) CSR compression performed sequentially on a stack of four images. All images are compressed and stored.

	Kernel execution time [microseconds per image]			
data chunk	Jungfrau raw-data pre-processing	Pre-processing + 'simplistic' Bragg's peaks count	Pre-processing + CSR matrix compression	Peak_finder pyFAI
10 x 500K pixels	28	31	147	N/A
10 x 4M pixels	219	240	469	2167
10 x 2070 x 2167[1]	212	240	492	N/A

Results in microseconds for one image of the given size.
[1] size of an image from JUNGFRAU 4M detector counting intermodule gaps.

Table 4.1: GPU kernel processing timing on NVIDIA QUADRO P6000 of the proposed online algorithms, as measured using the NVIDIA profiler.

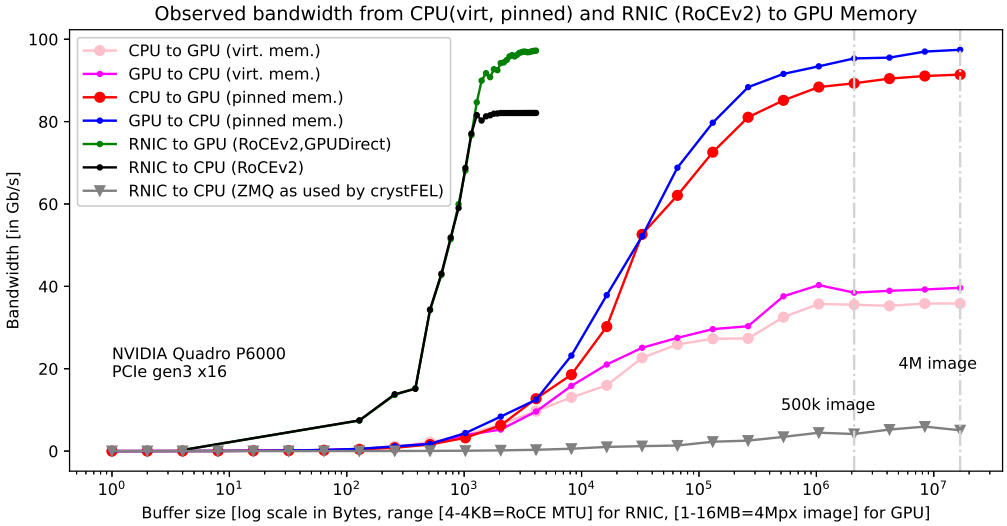


Figure 4.11: Observed transfer throughput from CPU and RNIC to GPU memory. The transfers from pinned memory are more efficient than from virtual memory. Using RDMA technology, full bandwidth was achieved from 1 KB. ZeroMQ transfers, which are used by the CrystFEL online reduction tool, are capped under 10 Gb/s.

Listing 4.5: Snippet of an OpenCL kernel and synchronization mechanism.

```
1
2  //If the buffer is allocated in shared virtual memory
3  //withclSVMAlloc, a direct data transfer from the RNIC
4  //is handled by the RoCE stack and ROCm driver,
5  //bypassing the staged memory.
6  void * sharedptr = clSVMAlloc (...);
7
8  //The shared sig variable is a memory lock
9  //triggered at the end of the data transfer.
10 pthread_cond_wait(&sig, &mutex);
11
12 //main loop
13 clSetKernelArgSVMPointer(...);
14 clSetKernelArg(&d_output);
15 clEnqueueNDRangeKernel(...);
16 clFinish(queue);
```

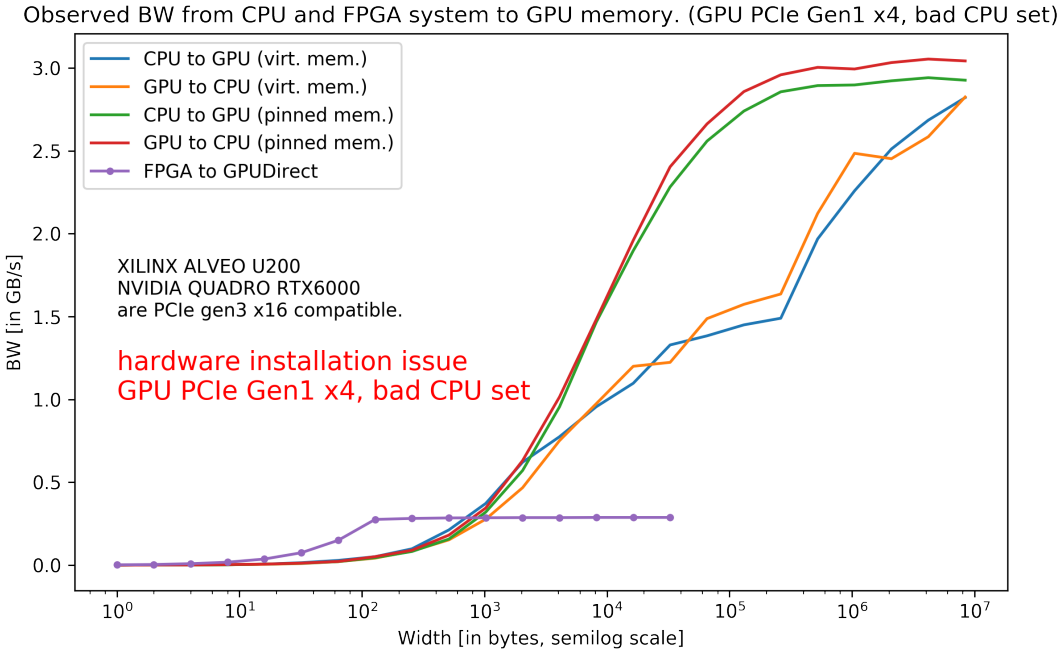


Figure 4.12: Transfer throughput from FPGA to CPU and GPU memory through the PCI-e interconnect.

an old motherboard, which PCIe Root Complex (RC) is badly handling PCI-e peer to peer requests. The first two issues might be addressed with a better hardware. The latter is related to the hardware implementation of the PCI-e RC which is known to change from CPU to CPU as mentioned in subsubsection 4.3.3.1.

For real-time data processing, a low execution jitter is desirable. This jitter was measured during the transfer of one buffer of 4 bytes. Round-trip is evaluated by a read operation from the FPGA (the proposed system is one way for now). Based on these measurements, the distributions shown in Figure 4.13 have been calculated.

4.3.4 Results on FPGA

The assessment of the FPGA RPU was done performing the already discussed raw data pre-processing for the Jungfrau detector.

The processing loop shown Listing 4.6 is unrolled and pipelined by the compiler.

Each RDMA transfer completion must end-up triggering the data processing: the host application uses the control registers of the data processing IP (AXI-lite interface exposed by the PCI-e bridge).

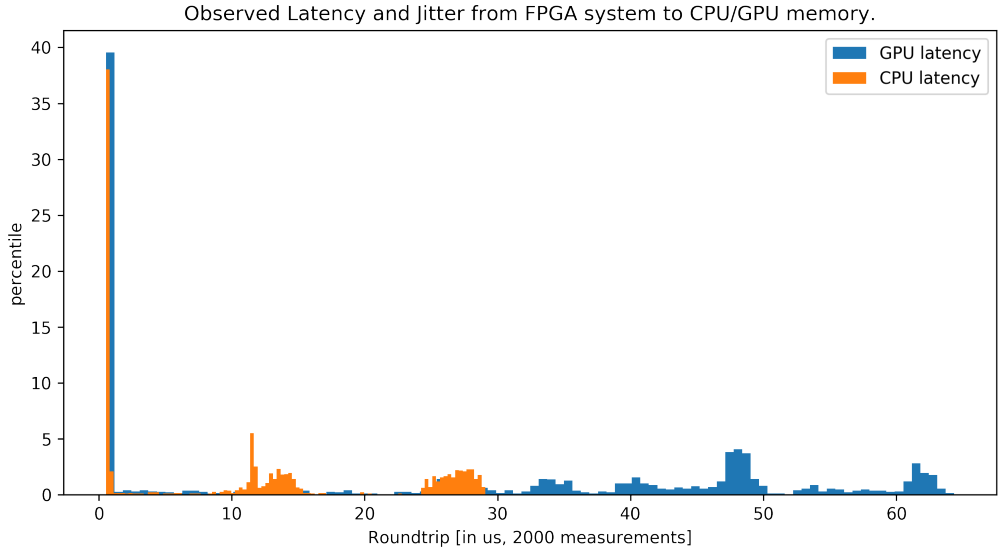


Figure 4.13: PCI-e latency estimations. As the RASHPA system is single sided, the latency is measured in a convoluted way.

4.3.5 Results on POWER9 Computer

The IBM AC922 is a powerful server that is employed as building brick in several supercomputers of the top 5 rank, such as Sierra or Summit operated by Oak Ridge National Laboratory (ORNL). Its architecture is described in [53] and connectivity is shown in Figure 4.14. As distinctive characteristics, the AC922 benefits from the IBM POWER9 processor and from an impressive PCI-e gen4 interconnect shown Figure 4.15. Its has been evaluated in the frame of SR experiments in [25].

It features up to 3 GPUs per CPU connected to the main processor by NVLink2 interconnect. An NVLink2 Processor Unit (NPU) handles data transfer to the GPU Nvidia Tesla as presented in [22]. On such heterogeneous computer, applications with various compute characteristics and requirements are executed on the most performance-efficient hardware.

RNICs are plugged in a specific slot of the AC922 which has 2 x8 links in a x16 (known as bifurcated slot) and shared between the two processors. Thus both PCI-e can benefit of the full RNIC throughput.

Such a computer seemed appealing as a RASHPA Processing Unit. We had the opportunity to perform some preliminary testing on this hardware, but only during a limited period of time. We have observed the expected maximum throughput with a pipeline featuring staged data in main memory.

But it appeared that using the low latency approach, GPUDirect, with direct placement into the NVIDIA GPU Tesla, was capped below 40 Gb/s. This deceptive figure seems related to the the connectivity used in this server. Actually the GPU is connected to the CPU only by a by PCI-e x2 lanes and the the bulk of the traffic, including GPUDi-

Listing 4.6: High level definition of a data processing IP. It performs the element-wise raw data processing as foreseen for an adaptive gain detector.

```
1  void jf_process(...)
2  {
3  memcpy(_raw,raw,k*sizeof(in_data_t));
4  for(int i=0; i < N; i++)
5  {
6  //The \#pragma pipeline is a directive
7  //for the synthesis tools. At the price of
8  //some hardware cell in the FPGA, the
9  //sequential loop is transformed in
10 //concurrent operation. This is possible
11 //when the operations are not competing
12 to access the same addresses.
13     #pragma HLS pipeline
14     int l=(_raw[i] >> 14) & 0x03;
15
16 //adder, floating point divider are synthesised
17 //and instanced to perform the computation.
18     _res[i] = ((_raw[i] & 0x3fff) - _pede[i]) / _g[i+l*N];
19 }
20 ...
21 }
```

rect RDMA, runs over the CPU-GPU NVLink2 bus. The PCI-e interconnect serves only for configuration purpose.

Better performances were expected using the Address Translation System (ATS). This subsystem is depicted and benchmarked by [57]. This is an IBM development recently integrated to the Linux kernel. It ensures the cache coherency between CPU and GPU memory, transparently migrating data when needed and overcoming the shortcomings of existing approaches. This mode of operation is activated when using Unified Memory programming model. The same data pointer is used either on CPU or GPU memory in order to simplify development tasks.

The data movements are handled automatically by the operating system by a special kernel module called *nv_rsync_mem*.

We have performed preliminary studies in order to perform RDMA transfer into Unified Memory allocation region. This is possible when implementing *On Demand Paging* (ODP) memory registration[9].

This technique requires the implementation, as transport queue pair, of the Reliable Connected (RC) type. Despite the RASHPA specification request a single sided communication, it should be doable if all the acknowledgement are positive.

[9]https://docs.mellanox.com/display/MLNXOFEDv461000/Optimized+Memory+Access

It is possible to optimize ATS efficiency giving explicitly *hints* to the system on the best possible location for the data. But it appeared that the observed throughput stayed limited to 40Gb/s. As the AC922 server used for testing was not running the officially supported Linux version, we cannot tell for sure if the RNIC are plugged in the *bifurcated* PCI-e slot as it is mandatory to activate ODP features.

The proposed RASHPA Processing Unit is a major milestone in the realization of a full-fledged RASHPA proof of concepts. It has been successfully implemented on diverse parallel processing hardware such as GPUs and FPGAs. The event mechanism is able to trigger data processing at the end of the transfer of a stack of images. We now have all the components available to make the online data processing, long time a scientist dream, a reality. Some work has yet to be done to refactor the code, improve the interfaces and documentation to improve re-usability and support.

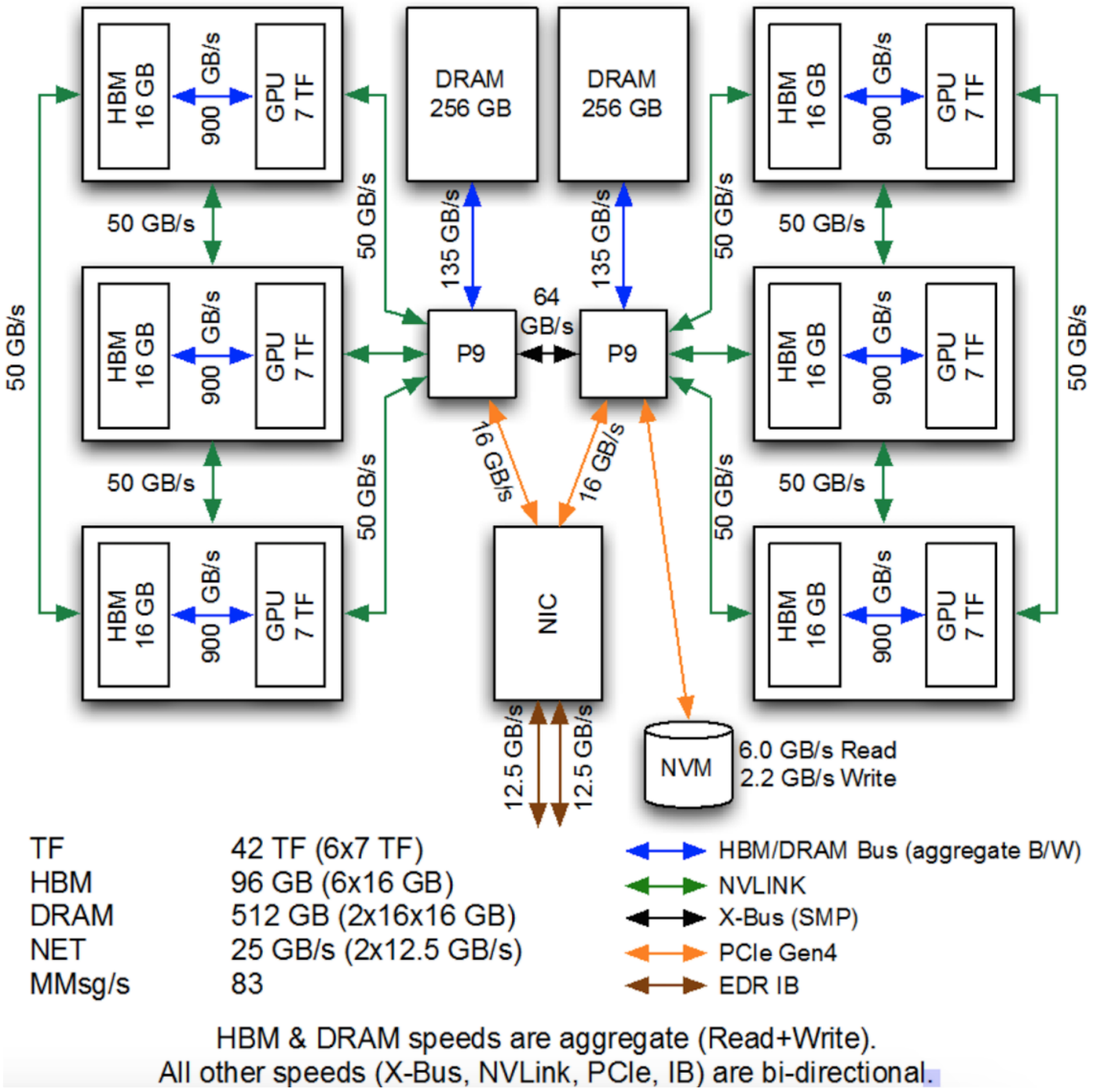


Figure 4.14: Overview of the IBM AC922 POWER9 Architecture and connectivity. The PCI-e gen4 and NVLink2 are a key advantage for high demanding experiments. (Source: POWER9 Redbook)

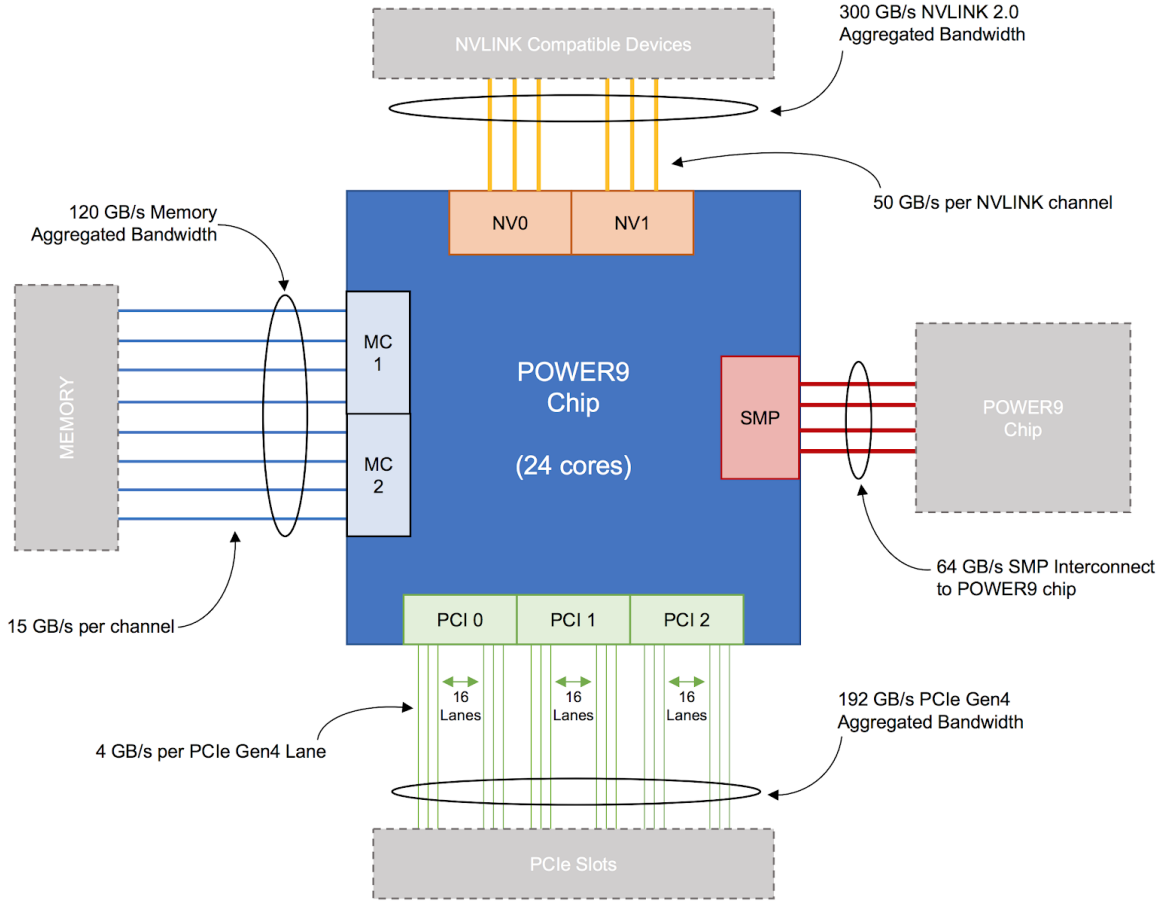


Figure 4.15: Overview of the POWER9 Processor PCI-e and NVLink2 interconnect (Source: POWER9 Redbook).

Chapter 5

Conclusion

Contents

This thesis aimed at investigating how RDMA techniques and massively parallel hardware applied conjointly can help to address ultra-fast data acquisitions challenges. The work conducted consists in diverse contributions to a previously existing framework called RASHPA, that was in an early draft stage at the beginning of this thesis.

5.1 Outcome

We conducted the assessment of RDMA Over Converged Ethernet (RoCEv2) in the perspective of its integration in the readout electronics of high throughput X-ray detectors. It is both sustainable and fulfill the requirements of a RASHPA compliant network protocol. The write verbs from the RDMA API and the Unreliable Connected (UC) queue pair type was selected as datagram format. It can both fit with a single-sided connection and with the constraints of embedded detector electronics.

As proof of concept, detector simulators were developed, either as pure software solutions or hardware-based FPGA boards capable of pushing multiple data transfers over RoCEv2 protocol into the memory of RASHPA receivers.

Similar assessments was performed in the case of data transfer over PCI-e long haul.

A low latency event mechanism is required to notify the receiver when the data transfers, silently offloaded to the DMA engine, are completed. A lightweight implementation has been proposed on both foreseen communication links. When using the RoCE protocol, it is based on the IMMEDIATE version of the WRITE verb.

We also brought contributions to another aspect of the challenge, supplementing the existing framework with the definition and implementation of what we refer to as RASHPA Processing Units (RPUs). While a RASHPA receiver is a remotely accessible pool of memory, a RPU is in addition of being a data sink, a busy processing system of the ingress data. RPUs have been successfully implemented using the standard CPU architecture or hardware accelerators such as GPU or FPGA.

In addition to managing the access to the internal memory of the accelerator device from the RNIC, the proposed system is defining a RASHPA Scheduler (RS) that aims at orchestrating the data processing. This subsystem manages the underlying hardware for both devices involved in the tasks. It is split in two functional parts: the first one managing the RNIC, the second one managing the data processing hardware.

In the first subsystem, a software daemon, an interrupt handler is in charge of handling possibly multiple events from multiple on going data transfer. These events are then consolidated in a global event, signaling that a whole new stack of images is ready in the processing pipeline. The second subsystem is a component of the user application. It is in charge of managing the workload on the processing hardware and triggering the accelerator unit when the data are available.

Our implementation of RPUs is supplemented by an Address Translation System (ATS) in charge of translating virtual to physical addresses as required by the DMA engine. The same mechanism can be applied to the main memory in the host system and to the internal memory of the accelerator devices as well. This ATS is plugable into the driver of the network cards.

In all, the result of our work is an end-to-end solution, encompassing hardware diversity. A variety of long lasting experiments could benefit from this system, when the amount of produced data exceeds the capacity or throughput of the disk storage, and therefore requires online rejection.

The only requirement remains the data processing time, which must be less for a given chunk of data, than the time spent on the data transfer of this given chunk.

In the context of this project, we have defined concepts, developed and validated softwares and FPGA designs with 2D X-ray image detectors in mind. Hopefully, this work should be generic enough to be applicable without modifications to a large variety of data analysis problems and algorithms.

The framework exploits the high throughput capabilities of the RDMA transfer mechanisms and the processing power of the hardware accelerators. But its implementation is not restricted to very high end configurations. It is indeed possible to scale down the data processing platform to relatively simple and inexpensive configurations when it would be convenient for economic or practical reasons. One way to achieve this might be by applying SoftRoCE, an implementation of RoCEv2 which is compatible with standard Ethernet network cards. The same code base and concepts still apply, the sole difference being the throughput performance, as the transfer would not be offloaded by a dedicated hardware on the card.

5.2 Outlook

5.2.1 Disaggregated Storage

In addition to the data processing, there is another challenge which has not yet been fully addressed by the RASHPA Framework: the data storage in a file system, not only in a memory buffer. Indeed, even a very large memory buffer could not be sufficient for a long lasting experiment. A backup strategy on disks has to be planned. For the sake of performance, it seems essential to skip any data storage operation on disk in the early stages of the acquisition process [24] but that cannot be completely avoided. Final storage remains necessary but has to be considered as the last stage of the pipeline.

There are other technical difficulties to overcome when storing scientific data sets, such as the large number of files to handle, taking into account that each image generates a file and its associated meta-data. Multiple simultaneous writers and readers support is also a challenge.

Large storage in the memory buffer could be improved to enable multiple concurrent access to the data. Various synchrotron facilities are working together on the Bluesky project [12] to use a so called *key-value store* (KVS) and leverage some of this issues. Such a KVS database could be used to absorb bursts of data as long as the central memory is still much faster than the transport. Cache system like *memcached*[1] or *ceph*[2] could serve as sharing facilities, with a lightweight put/get API.

[1] https://github.com/memcached/memcached
[2] https://github.com/ceph/ceph

The storage issue might be tackled from another approach. Despite the existing distributed file systems cannot cope with the expected throughput, remote storage seems possible using the new generation flash disk. Non-Volatile Memory Express (NVMe) is a specification for storage devices on Solid State Disks (SSD). These devices are gaining momentum due to their competitive price per Tera Byte as well as their small form factor, high density and possible throughput. The NVME over Fabric (NVMEoF[3]) shown Figure 5.1 is the RDMA extension of the NVME standard, making a remote disk acting like a local one. They possibly can be stacked to improve overall performances up to multi GB per second, even saturating a 100 gigabit Ethernet link. It is hence possible to share storage resources amid multiple computers (disagregated storage).

A new solution from NVIDIA called Cuda Storage[4], is due to integrate flash disk arrays (NVMEoF), RDMA techniques and GPU devices. This technology is enabling data transfer directly from the GPU to the remote SSD, like GPUDirect enables direct transfer from an RNIC to a GPU device. It has recently been upgraded with the *cuFile* library including a full support for standard file systems. It is even possible to access simultaneously to the same file from several GPU.

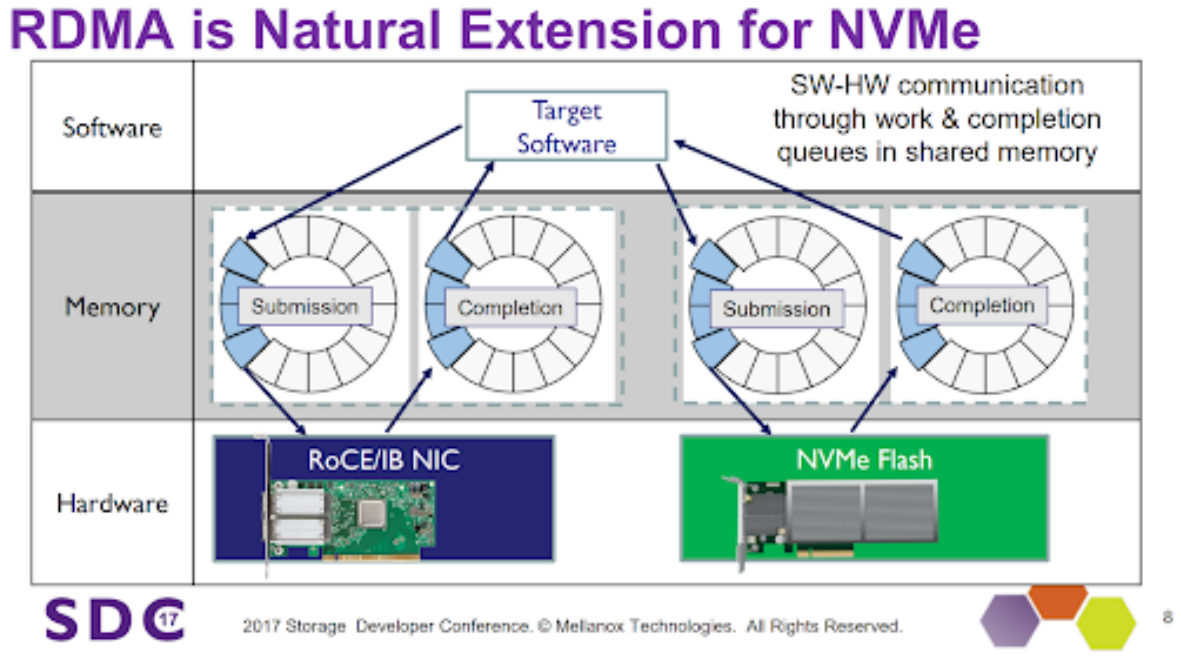


Figure 5.1: Non-Volatile Memory Express over Fabric (NVMEoF) is the specification for storage devices, mainly SSD leveraging RDMA techniques. Remote Flash disks appears in the system like a local PCI-e resource. (Source: SNIA slide)

[3]https://nvmexpress.org/wp-content/uploads/NVMe_Over_Fabrics.pdf
[4]https://developer.nvidia.com/blog/gpudirect-storage/

5.2.2 Integration Challenges

An other key point for the success of the RASHPA project, would be the seamless integration to already existing parallel computing frameworks. Some preliminary efforts have been done, e.g. the integration into Python programming language, which is largely adopted by the scientific community. Here a Numpy array can be used as a RASHPA buffer and directly filled by the detector data.

Some work has been initiated to implement RASHPA into the ESRF LIMA project[5]. This is an umbrella for all the detectors in use at the ESRF (as of today, more than 20 different kinds of detectors). Lima is featuring a data processing pipeline and the work is ongoing to make it RASHPA compliant.

A real-life project is foreseen with ID13, an already existing beamline of the ESRF. This would be a definitive demonstration of the work carried-on. The foreseen experiment is to scan quickly, at low resolution and low flux a sample to prevent its destruction by radiation damages. As soon as the interesting site in the sample is detected, the acquisition should go to high resolution mode, at low speed with high flux of X rays. This automatic detection will be done thanks to the processing power of a GPU accelerator by a not yet fully determined imaging algorithm.

The experimental setup deserves further feasibility studies and the implementation with the mechanical interface, but this should be an interesting demonstration of the RASHPA capabilities. To the best of our knowledge, such control system has not yet been demonstrated in the frame of Synchrotron Radiation.

[5]https://gitlab.esrf.fr/limagroup/lima

Bibliography

[1] Steve Abbott. "HIGHLIGHTS OF CUDA 10 FOR SUMMIT". en. In: (Mar. 2019), p. 30.

[2] E. Agostini, D. Rossetti, and S. Potluri. "GPUDirect Async: Exploring GPU synchronous communication techniques for InfiniBand clusters". In: *Journal of Parallel and Distributed Computing* 114 (Apr. 2018), pp. 28–45. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2017.12.007. URL: http://www.sciencedirect.com/science/article/pii/S0743731517303386 (visited on 03/11/2019).

[3] Henrique de Almeida et al. "The Back-End Computer System for the Medipix Based PI-MEGA X-Ray Camera". In: (Jan. 2018), THBPA03. DOI: 10.18429/JACoW-ICALEPCS2017-THBPA03.

[4] G. Ashiotis et al. "The fast azimuthal integration Python library: pyFAI". en. In: *Journal of Applied Crystallography* 48.2 (Apr. 2015). Number: 2 Publisher: International Union of Crystallography, pp. 510–519. ISSN: 1600-5767. DOI: 10.1107/S1600576715004306. URL: //scripts.iucr.org/cgi-bin/paper?fv5028 (visited on 05/10/2020).

[5] Cédric Augonnet et al. "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures". en. In: *Concurrency and Computation: Practice and Experience* 23.2 (2011). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.1631, pp. 187–198. ISSN: 1532-0634. DOI: https://doi.org/10.1002/cpe.1631. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.1631 (visited on 12/07/2020).

[6] Guy E. Blelloch. *Prefix Sums and Their Applications*. Tech. rep. Synthesis of Parallel Algorithms, 1990.

[7] N. J. Boden et al. "Myrinet: a gigabit-per-second local area network". In: *IEEE Micro* 15.1 (Feb. 1995). Conference Name: IEEE Micro, pp. 29–36. ISSN: 1937-4143. DOI: 10.1109/40.342015.

[8] Suren Chilingaryan et al. "A GPU-Based Architecture for Real-Time Data Assessment at Synchrotron Experiments". In: *IEEE Transactions on Nuclear Science* 58.4 (Aug. 2011). Conference Name: IEEE Transactions on Nuclear Science, pp. 1447–1455. ISSN: 1558-1578. DOI: 10.1109/TNS.2011.2141686.

[9] robert Crovella. *A question on nested parallelism*. en-US. Apr. 2019. URL: https://forums.developer.nvidia.com/t/a-question-on-nested-parallelism/72751 (visited on 10/12/2020).

[10] Zahavi Eitan. *Mellanox keynote Sigcomm 2018*. Aug. 2018. URL: http://conferences2.sigcomm.org/sigcomm/2018/files/slides/kbnet/keynote_2.pdf (visited on 11/08/2018).

[11] V. Favre-Nicolin et al. "PyNX: high-performance computing toolkit for coherent X-ray imaging based on operators". en. In: *Journal of Applied Crystallography* 53.5 (Oct. 2020). Number: 5 Publisher: International Union of Crystallography, pp. 1404–1413. ISSN: 1600-5767. DOI: 10.1107/S1600576720010985. URL: https://journals.iucr.org/j/issues/2020/05/00/zy5006/ (visited on 10/04/2020).

[12] *Full article: Bluesky's Ahead: A Multi-Facility Collaboration for an a la Carte Software Project for Data Acquisition and Management*. URL: https://www.tandfonline.com/doi/full/10.1080/08940886.2019.1608121 (visited on 09/30/2020).

[13] Alexander Gillert. "Direct GPU-FPGA Communication". en. In: *Thesis* (2015), p. 71.

[14] Ryan E. Grant et al. "Scalable connectionless RDMA over unreliable datagrams". en. In: *Parallel Computing* 48 (Oct. 2015), pp. 15–39. ISSN: 01678191. DOI: 10.1016/j.parco.2015.03.009. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167819115000617 (visited on 11/10/2018).

[15] Chuanxiong Guo et al. "RDMA over Commodity Ethernet at Scale". en. In: *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference - SIGCOMM '16*. Florianopolis, Brazil: ACM Press, 2016, pp. 202–215. ISBN: 978-1-4503-4193-6. DOI: 10.1145/2934872.2934908. URL: http://dl.acm.org/citation.cfm?doid=2934872.2934908 (visited on 03/18/2019).

[16] Nate Hanford and Brian Tierney. "Recent Linux TCP Updates, and how to tune your 100G host". en. In: (2016), p. 43.

[17] *History of the Internet*. en. Page Version ID: 975002821. Aug. 2020. URL: https://en.wikipedia.org/w/index.php?title=History_of_the_Internet&oldid=975002821 (visited on 09/07/2020).

[18] A Homs. *About SLSDetector UDP receiver, private communication*. Feb. 2019.

[19] Seokbin Hong, Won-Ok Kwon, and Myeong-Hoon Oh. "Hardware Implementation and Analysis of Gen-Z Protocol for Memory-Centric Architecture". In: *IEEE Access* 8 (2020). Conference Name: IEEE Access, pp. 127244–127253. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3008227.

[20] Joseph N. Huber, Oscar R. Hernandez, and Matthew Graham Lopez. *Effective Vectorization with OpenMP 4.5*. en. Tech. rep. ORNL/TM–2016/391, 1351758. Mar. 2017, ORNL/TM–2016/391, 1351758. DOI: 10.2172/1351758. URL: http://www.osti.gov/servlets/purl/1351758/ (visited on 10/13/2020).

[21] R Huggahalli, R Iyer, and S Tetrick. "Direct Cache Access for High Bandwidth Network I/O". en. In: (2005), p. 10.

[22] IBM POWER9 NPU team. "Functionality and performance of NVLink with IBM POWER9 processors". In: *IBM Journal of Research and Development* 62.4/5 (July 2018), 9:1–9:10. ISSN: 0018-8646. DOI: 10.1147/JRD.2018.2846978.

[23] Intel. *DPDK data plane developement Kit*. en-US. 2015. URL: https://www.dpdk.org/ (visited on 11/08/2018).

[24] J. Kieffer, S. Petitdemange, and T. Vincent. "Real-time diffraction computed tomography data reduction". en. In: *Journal of Synchrotron Radiation* 25.2 (Mar. 2018), pp. 612–617. ISSN: 1600-5775. DOI: 10.1107/S1600577518000607. URL: http://scripts.iucr.org/cgi-bin/paper?co5098 (visited on 02/25/2019).

[25] Jerome Kieffer. *Investigation of hardware compression on IBM Power9 - Dr. Jerome Kieffer et al.* Sept. 2019. URL: https://www.youtube.com/watch?v=C_OMmLDGc-I&list=PLPyhR4PdEeGaiiZ_iEi3rexxx830OpgwH&index=4&ab_channel=hdf5 (visited on 10/12/2020).

[26] Andreas Klöckner et al. "GPU Scripting and Code Generation with PyCUDA". In: *arXiv:1304.5553 [cs]* (Apr. 2013). arXiv: 1304.5553. URL: http://arxiv.org/abs/1304.5553 (visited on 07/18/2019).

[27] Alexopoulos Konstantinos S. *Extending an asynchronous messaging library using an RDMA-enabled interconnec*. thesis. Oct. 2017. URL: https://openlab.cern/sites/openlab.web.cern.ch/files/2018-03/thesis%5B1%5D.pdf (visited on 10/03/2020).

[28] Przemyslaw Lenkiewicz, P. Chris Broekema, and Bernard Metzler. "Energy-Efficient Data Transfers in Radio Astronomy with Software UDP RDMA". In: *Future Generation Computer Systems* 79 (Feb. 2018). tex.ids: lenkiewicz_energy-efficient_2018 arXiv: 1703.07626, pp. 215–224. ISSN: 0167739X. DOI: 10.1016/j.future.2017.03.027. URL: http://arxiv.org/abs/1703.07626 (visited on 02/25/2019).

[29] Filip Leonarski et al. "JUNGFRAU detector for brighter x-ray sources: Solutions for IT and data science challenges in macromolecular crystallography". In: *Structural Dynamics* 7.1 (Jan. 2020), p. 014305. DOI: 10.1063/1.5143480. URL: https://aca.scitation.org/doi/10.1063/1.5143480 (visited on 04/20/2020).

[30] Alessandro Lonardo et al. "A FPGA-based Network Interface Card with *GPUDirect enabling realtime GPU computing in HEP experiments.." en. In: *Proc. of GPUHEP 2014* (2015). Medium: PDF Publisher: Deutsches Elektronen-Synchrotron, DESY, Hamburg, 8691, DESY. DOI: 10.3204/DESY-PROC-2014-05/16. URL: http://www-library.desy.de/preparch/desy/proc/proc14-05/16.pdf (visited on 09/21/2020).

[31] P. MacArthur and R. D. Russell. "A Performance Study to Guide RDMA Programming Decisions". In: *2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems*. June 2012, pp. 778–785. DOI: 10.1109/HPCC.2012.110.

[32] Wassim Mansour, Nicolas Janvier, and Pablo Fajardo. "FPGA Implementation of RDMA-Based Data Acquisition System Over 100 GbE". In: *arXiv:1806.08939 [physics]* (June 2018). arXiv: 1806.08939. URL: http://arxiv.org/abs/1806.08939 (visited on 02/25/2019).

[33] Majkowski Marek. *How to receive a million packets per second*. en. June 2015. URL: https://blog.cloudflare.com/how-to-receive-a-million-packets/ (visited on 02/25/2019).

[34] Bruno Martins. "Drinking from the firehose: the ADEiger driver". en. In: (2017), p. 16.

[35] Mellanox. *PB_Bluefield_SoC*. June 2018. URL: http://www.mellanox.com/related-docs/npu-multicore-processors/PB_Bluefield_SoC.pdf (visited on 06/18/2019).

[36] Mellanox. *VMA Performance Tuning Guide — Mellanox Interconnect Community*. Aug. 2018. URL: https://community.mellanox.com/docs/DOC-2797 (visited on 11/08/2018).

[37] F Le Mentec et al. "RASHPA: A DATA ACQUISITION FRAMEWORK FOR 2D X-RAY DETECTORS". en. In: (2014), p. 4.

[38] Mao Miao et al. "SoftRDMA: Rekindling High Performance Software RDMA over Commodity Ethernet". en. In: *Proceedings of the First Asia-Pacific Workshop on Networking - APNet'17*. Hong Kong, China: ACM Press, 2017, pp. 43–49. ISBN: 978-1-4503-5244-4. DOI: 10.1145/3106989.3106995. URL: http://dl.acm.org/citation.cfm?doid=3106989.3106995 (visited on 11/10/2018).

[39] Hannes Mohr. "Evaluation of GPU-based track-triggering for the CMS detector at CERN's HL-LHC". en. In: *Report* (Oct. 2016), p. 103.

[40] Rajmund Mokso et al. "GigaFRoST: the gigabit fast readout system for tomography". eng. In: *Journal of Synchrotron Radiation* 24.Pt 6 (Nov. 2017), pp. 1250–1259. ISSN: 1600-5775. DOI: 10.1107/S1600577517013522.

[41] A. Mozzanica et al. "The JUNGFRAU Detector for Applications at Synchrotron Light Sources and XFELs". In: *Synchrotron Radiation News* 31.6 (Nov. 2018), pp. 16–20. ISSN: 0894-0886. DOI: 10.1080/08940886.2018.1528429. URL: https://doi.org/10.1080/08940886.2018.1528429 (visited on 07/17/2019).

[42] Michal Nazarewicz. *A deep dive into CMA [LWN.net]*. Mar. 2012. URL: https://lwn.net/Articles/486301/ (visited on 05/07/2020).

[43] Rolf Neugebauer et al. "Understanding PCIe performance for end host networking". en. In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication - SIGCOMM '18*. Budapest, Hungary: ACM Press, 2018, pp. 327–341. ISBN: 978-1-4503-5567-4. DOI: 10.1145/3230543.3230560. URL: http://dl.acm.org/citation.cfm?doid=3230543.3230560 (visited on 02/18/2019).

[44] Miroslav Z. Papiz and Graeme Winter. "X-Ray Crystallography, Biomolecular Structure Determination Methods". en. In: *Encyclopedia of Spectroscopy and Spectrometry (Third Edition)*. Ed. by John C. Lindon, George E. Tranter, and David W. Koppenaal. Oxford: Academic Press, Jan. 2017, pp. 640–647. ISBN: 978-0-12-803224-4. DOI: 10.1016/B978-0-12-803224-4.00050-9. URL: http://www.sciencedirect.com/science/article/pii/B9780128032244000509 (visited on 10/11/2020).

[45] Jiwoong Park et al. "SoftDC: software-based dynamically connected transport". en. In: *Cluster Computing* 23.1 (Mar. 2020), pp. 347–357. ISSN: 1386-7857, 1573-7543. DOI: 10.1007/s10586-019-02926-0. URL: http://link.springer.com/10.1007/s10586-019-02926-0 (visited on 06/13/2020).

[46] Denis Perret et al. "Bridging FPGA and GPU technologies for AO real-time control". In: *Adaptive Optics Systems V*. Vol. 9909. International Society for Optics and Photonics, July 2016, p. 99094M. DOI: 10.1117/12.2232858. URL: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/9909/99094M/Bridging-FPGA-and-GPU-technologies-for-AO-real-time-control/10.1117/12.2232858.short (visited on 11/28/2019).

[47] S Petitdemange et al. "The LIMA Project Update". en. In: (2014), p. 4.

[48] R. Ponsard et al. "Online GPUAnalysis using Adaptive DMA Controlled by Softcore for 2D Detectors". In: *2020 23rd Euromicro Conference on Digital System Design (DSD)*. Aug. 2020, pp. 436–439. DOI: 10.1109/DSD51259.2020.00075.

[49] R. Ponsard et al. "RDMA data transfer and GPU acceleration methods for high-throughput online processing of serial crystallography images". en. In: *Journal of Synchrotron Radiation* 27.5 (Sept. 2020). Number: 5 Publisher: International Union of Crystallography. ISSN: 1600-5775. DOI: 10.1107/S1600577520008140. URL: https://journals.iucr.org/s/issues/2020/05/00/il5050/ (visited on 08/05/2020).

[50] Danny C. Price. "Real-time stream processing in radio astronomy". In: *arXiv:1912.09041 [astro-ph]* (Dec. 2019). arXiv: 1912.09041. URL: http://arxiv.org/abs/1912.09041 (visited on 06/11/2020).

[51] Pantaleo Raimondi. "ESRF-EBS: The Extremely Brilliant Source Project". en. In: *Synchrotron Radiation News* (Dec. 2016). ISSN: 10.1080/08940886.2016.1244462. URL: https://www.tandfonline.com/doi/pdf/10.1080/08940886.2016.1244462 (visited on 07/17/2019).

[52] Pramod Ramarao. *CUDA 10 Features Revealed: Turing, CUDA Graphs, and More*. en-US. Sept. 2018. URL: https://devblogs.nvidia.com/cuda-10-features-revealed/ (visited on 01/22/2020).

[53] Steve Roberts, Pradeep Ramanna, and John Walthour. "AC922 Data Movement for CORAL". en. In: *2018 IEEE High Performance extreme Computing Conference (HPEC)*. Waltham, MA: IEEE, Sept. 2018, pp. 1–5. ISBN: 978-1-5386-5989-2. DOI: 10.1109/HPEC.2018.8547707. URL: https://ieeexplore.ieee.org/document/8547707/ (visited on 06/02/2020).

[54] Allyn Romanow. "An Overview of RDMA over IP". en. In: (2003), p. 22.

[55] Davide Rossetti. *Benchmarking GPUDirect RDMA on Modern Server Platforms*. en-US. Oct. 2014. URL: https://devblogs.nvidia.com/benchmarking-gpudirect-rdma-on-modern-server-platforms/ (visited on 02/25/2019).

[56] Davide Rossetti and Elena Agostini. "S7128 - HOW TO ENABLE NVIDIA CUDA STREAM SYNCHRONOUS COMMUNICATIONS USING GPUDIRECT". en. In: (2017), p. 75.

[57] Davide Rossetti and Elena Agostini. *s8474-gpudirect-life-in-the-fast-lane.pdf*. 2018. URL: https://on-demand.gputechconf.com/gtc/2018/presentation/s8474-gpudirect-life-in-the-fast-lane.pdf (visited on 12/07/2020).

[58] Jörn Schumacher, Christian Plessl, and Wainer Vandelli. "High-Throughput and Low-Latency Network Communication with NetIO". en. In: *Journal of Physics: Conference Series* 898 (Oct. 2017), p. 082003. ISSN: 1742-6596. DOI: 10.1088/1742-6596/898/8/082003. URL: https://doi.org/10.1088%2F1742-6596%2F898%2F8%2F082003 (visited on 06/17/2019).

[59] Hefty Sean. *2012_Workshop_Mon_Rsockets.pdf*. 2012. URL: http://www.smallake.kr/wp-content/uploads/2014/04/2012_Workshop_Mon_Rsockets.pdf (visited on 04/10/2019).

[60] Gilad Shainer et al. "The development of Mellanox/NVIDIA GPUDirect over InfiniBand—a new model for GPU to GPU communications". en. In: *Computer Science - Research and Development* 26.3 (June 2011), pp. 267–273. ISSN: 1865-2042. DOI: 10.1007/s00450-011-0157-1. URL: https://doi.org/10.1007/s00450-011-0157-1 (visited on 02/25/2019).

[61] Guy Shattah and Christoph Lameter. "Contiguous memory allocation in Linux user-space". en. In: (2016), p. 18.

[62] J. B. Thayer et al. "Building a Data System for LCLS-II". In: *2017 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC)*. ISSN: 2577-0829. Oct. 2017, pp. 1–4. DOI: 10.1109/NSSMIC.2017.8533033.

[63] Shin-Yeh Tsai and Yiying Zhang. "LITE Kernel RDMA Support for Datacenter Applications". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. SOSP '17. event-place: Shanghai, China. New York, NY, USA: ACM, 2017, pp. 306–324. ISBN: 978-1-4503-5085-3. DOI: 10.1145/3132747.3132762. URL: http://doi.acm.org/10.1145/3132747.3132762 (visited on 06/17/2019).

[64] Zhi Wang et al. "RDMAvisor: Toward Deploying Scalable and Simple RDMA as a Service in Datacenters". In: *arXiv:1802.01870 [cs]* (Feb. 2018). arXiv: 1802.01870. URL: http://arxiv.org/abs/1802.01870 (visited on 06/17/2019).

[65] T. A. White et al. "CrystFEL: a software suite for snapshot serial crystallography". en. In: *Journal of Applied Crystallography* 45.2 (Apr. 2012), pp. 335–341. ISSN: 0021-8898. DOI: 10.1107/S0021889812002312. URL: http://scripts.iucr.org/cgi-bin/paper?db5097 (visited on 01/07/2019).

[66] Philip Willmott. "Synchrotron Physics". en. In: *An Introduction to Synchrotron Radiation*. John Wiley & Sons, Ltd, 2019, pp. 51–106. ISBN: 978-1-119-28045-3. DOI: 10.1002/9781119280453.ch3. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119280453.ch3 (visited on 04/19/2020).

[67] Ming Yang et al. "Avoiding Pitfalls when Using NVIDIA GPUs for Real-Time Tasks in Autonomous Systems". In: *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Ed. by Sebastian Altmeyer. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 20:1–20:21. ISBN: 978-3-95977-075-0. DOI: 10.4230/LIPIcs.ECRTS.2018.20. URL: http://drops.dagstuhl.de/opus/volltexte/2018/8984 (visited on 11/27/2019).

Appendix A

Articles in Journals and Conferences

- Research paper for the Journal of Synchrotron Radiation, section Computer program Published in: J. Synchrotron Rad. (2020). 27, 1297-1306

  RDMA data transfer and GPU acceleration methods for high-throughput online processing of serial crystallography images

  R. Ponsard, N. Janvier, J. Kieffer, D. Houzet and V. Fristot

  https://doi.org/10.1107/S1600577520008140

- Paper for the DSD20/Euromicro Conference

  Published in: 2020 23rd Euromicro Conference on Digital System Design (DSD)

  Online GPU Analysis using Adaptive DMA Controlled by Softcore for 2D Detectors

  R. Ponsard; N. Janvier; D. Houzet; V. Fristot; W. Mansour

  https://doi.org/10.1109/DSD51259.2020.00075

- Paper for the RT2020 Conference (co-author)

  Oral presentation at the 22nd IEEE Real Time Conference

  https://indico.cern.ch/event/737461/contributions/4013042/

  FPGA Based Real-Time Image Manipulation and Advanced Data Acquisition For 2D-XRAY Detectors

  Wassim Mansour, Rattana Biv, Cyril Ponchut, Raphael Ponsard, Nicolas Janvier, Pablo Fajardo

- Research paper (co-author)

  Under review in IEEE Transactions on Nuclear Science

  FPGA Based Real-Time Image Manipulation and Advanced Data Acquisition For 2D-XRAY Detectors

  Wassim Mansour, Rattana Biv, Cyril Ponchut, Raphael Ponsard, Nicolas Janvier, Pablo Fajardo

  https://arxiv.org/abs/2010.15450

- Poster for the Supercomputing 2020 Conference (SC20)
  Doctoral Showcase Poster Display
  https://sc20.supercomputing.org/program/posters/doctoral-showcase/#schedule

Appendix B

Hardware and Software Contributions

The thesis work was mainly geared towards feasibility studies than optimization and performance challenges. We have designed many mock-up in this purpose, but a ready-to-use library is still under construction. The final specifications of the RASHPA framework are still not yet fully defined. The definition process has taken time because many considerations, external to this thesis work, where left unattended or had evolved over time: code maintainability over years, genericity, broader event concept, etc

We list here our main contributions hoping it would help anyone with similar aims.

Microbenchmark applications, REMU Simulator, ROMULU RPU and GPU kernels are available, including Python scripts to produce example datasets and plots of the observed measurements. FPGA softcore-based design (Xilinx Vivado 19.1) and firmware (C language source code) are available as well.

The code snippet are grouped by family, and are available on the ESRF gitlab: gitlab.esrf.fr/ponsard. The whole code will shortly be released on *github.com/rponsard*.

Python scripts are available on several jupyter notebook: RASHPA manager, initial dataset generation, performance measurements

- UDP micro benchmarks under the *testing/udp* directory

  udp_sendto with or without libVMA

  recvmmsg

  sendfile

- micro benchmarks under the *testing/zmq* directory

  zmq pull/pull

  rep/req

  radio/dish

- micro benchmarks of UC queue pairs under the *testing/rdma* directory

  ud_send

write using UC qp
multi-threaded data transfer using write
python ICRC calculator
rcwrite
UD send/recv synchronizer

- micro benchmarks of RC queue pairs under the *testing/rdma/rc* directory
  python/C RC source
  power9
- full fledged application software under the *rashpa_poc* directory
  remu software detector emulator
  romulu software data receiver and GPU data processing
  librashpa common code
- Xilinx U200 FPGA designs, firmware hand host code
  detector emulator remu PCI-e with event mechanism
  Xilinx ERNIC IP evaluation in the frame of RASHPA project
  data processing using romulu-hls
- Linux kernel modules for memory allocation or physical address resolution
  libcma: large contiguous memory allocator
  libgma: nvidia GPU memory physical memory provider (for RASHPA PCI-e)
  fpga_peer_mem
  AMD GPU memory physical memory provider (for RASHPA PCI-e)
  virtual memory with or without huge pages physical address provider
- GPU kernels
  CUDA kernels launch time overhead measurement
  data processing using multithread and CPU and OpenMP
  OpenCL - RoCE
  persistent kernel
  memory allocation

Appendix C

Résumé de la thèse en langue française

Traitement en temps réel, haut débit et faible latence d'images par coprocesseurs GPU ou FPGA utilisant les techniques d'accès direct à la mémoire distante

Introduction

Cette thèse porte sur le transfert à haut débit de données produites par des détecteurs à rayons X utilisés pour l'imagerie scientifique. Mais cette problématique n'est bien sûr pas spécifique au domaine précité et le travail réalisé pourra aisément être réutilisé dans d'autres domaines scientifiques ou industriels confrontés aux mêmes impératifs d'acquisition rapide.

En utilisant les réseaux et architectures d'ordinateurs traditionnels, on est en effet confronté à des goulots d'étranglement lors du transfert de ces données vers les unités de stockage ou de traitement qui empêchent l'utilisation à pleine performance des détecteurs. L'autre contribution de la thèse, étroitement liée à la précédente, porte sur le traitement en temps réel des données produites quand elles ont été acheminées, grâce à l'emploi des architectures massivement parallèles qu'on trouve dans des accélérateurs matériels de calcul de type GPU ou FPGA. L'objectif est de diminuer autant que possible la quantité de données à stocker avant leur analyse ultérieure.

Cette thèse s'est déroulée à l'ESRF, Installation Européenne de Rayonnement Synchrotron à Grenoble. Situé sur le Campus Européen de Photo-Neutronique (EPN Campus), l'ESRF est une des sources de rayonnement X les plus intenses existantes aujourd'hui. Après les récents travaux d'amélioration basé sur une innovation majeure au niveau des aimants de courbure, le projet EBS (Source Extrêmement Brillante), l'ESRF est maintenant le premier synchrotron complètement opérationnel de quatrième génération. La brillance, qui mesure l'intensité et la qualité du faisceau, a été multipliée par 100 et la focalisation grandement améliorée.

Le rayonnement synchrotron n'est pas de nature différente des rayons X utilisés depuis leur découverte par Roentgen au début du XX siècle. Ils ne diffèrent que par leur intensité

considérablement plus importante que celle rencontrée dans les autres sources.

Ces rayons X sont produits par des électrons relativistes portés à une énergie de 6 GeV dans un premier booster et qui dissipent ensuite une partie de leur énergie dans un anneau de stockage lorsque leur trajectoire est courbée par un champ magnétique. Ils circulent dans un tube à ultravide de 850 m de circonférence (un polygone régulier de 32 cotés) et sont guidés par des aimants (électro-aimants ou permanents) de focalisation et de courbure. Tangentiellement à chaque sommet de ce polygone se trouvent les lignes de lumière qui exploitent le RX produit. D'autres lignes sont situées dans le prolongement des arêtes pour exploiter le RX produit par des dispositifs magnétiques d'insertion qui font spiraler les électrons le long de leur trajectoire, produisant un faisceau très intense.

Le RX ainsi produit a de multiples propriétés intéressantes dans de nombreuses disciplines scientifiques d'exploration de la matière: très grande intensité, impulsion ultracourtes, cohérence du rayonnement comme dans un laser, etc... qui expliquent le succès de ce type d'installation et la diversité des expériences qui y ont cours.

L'amélioration permanente des sources de rayonnement X, s'accompagne aussi des gains en performances des détecteurs et sont ainsi réalisables des expériences qui peuvent produire des volumes énormes de données à très haut débit, quantités aussi difficiles à gérer qu'à stocker. Ainsi les pixel-détecteurs de nouvelle génération embarquent suffisamment d'électronique au niveau de chaque pixel qu'ils constituent en fait une source autonome de données. On peut de plus réaliser de très grands détecteurs avec un assemblage de plusieurs de ces modules autonomes.

Dans ce contexte, il devient indispensable de mettre à disposition des utilisateurs des systèmes de calculs plus performants afin de permettre le pré-traitement en temps réel des données brutes produites lors de l'acquisition. Parmi ces prétraitements qu'il est désirable de pratiquer le plus en amont possible dans la chaîne d'acquisition figurent : La conversion des données brutes issues de l'électronique de détection en données utiles La réjection de celles qui ne sont pas pertinentes ou vides, La compression des données La supervision en temps réel de l'expérience.

Pour le moment, ces problématiques de gestion des flux de données n'ont pas encore reçu de réponse générique pleinement satisfaisante.

Cette thèse fait partie d'un projet plus vaste, le projet RASHPA (RDMA-based Acquisition System for High Performance Applications) de l'ESRF, visant à développer un système d'acquisition haute performance basé sur le concept du RDMA (Remote Direct Memory Access) : transfert de donnée entre périphérique et mémoire sans recours au CPU.

Une des caractéristiques essentielles de ce projet est la capacité à transférer directement des données de la tête du détecteur vers la mémoire de l'unité de calcul, au plus haut débit possible, en utilisant les techniques d'accès direct à la mémoire (RDMA), sans copies inutiles entre les différents espaces mémoires (zéro copie), et éliminant le recours à un processeur.

Le travail réalisé pendant cette thèse est une contribution au système RASHPA, qui rend possible, non seulement le transfert de données à pleine vitesse dans la mémoire du système informatique à destination, mais aussi directement dans la mémoire interne de cartes accélératrices dans le cas de système à l'architecture hétérogène. Ce type d'architecture se rencontre quand il est fait recours à des calculateurs massivement par-

allèles type GPU et FPGA. Les cartes graphiques d'ordinateurs développées pour le marché de la synthèse d'image 2D/3D et du jeux vidéo ont été détournées de leur utilisation initiale en raison de leur capacité de calcul parallèle à un prix imbattable et sont maintenant massivement utilisées dans le domaine du calcul à haute performance. Nous avons étudié les différentes techniques existantes de transfert depuis une carte réseau vers la mémoire de ces dispositifs et les goulots d'étranglement afférents. Un mécanisme de synchronisation à faible latence entre carte réseau et unité de calcul est proposé, déclenchant le traitement des données au rythme du détecteur.

Cela permet de fournir une solution globale au traitement de données en temps réel, tant sur ordinateurs classiques que sur accélérateurs massivement parallèles.

État de l'Art

Après cette introduction, la thèse se poursuit par un état de l'art des principales technologies rencontrées. Ce projet se situe en effet à l'intersection de plusieurs disciplines des sciences informatiques : transfert de données à haute vitesse, architecture interne des ordinateurs, programmation de drivers pour le noyau Linux, techniques de calculs parallèles, en particulier sur GPU, développement de circuits pour FPGA... Pour chacun de ces sujets, nous avons eu bien souvent à gérer des détails d'implémentation mal ou peu documentés ou des modes de fonctionnement peu usités.

Dans cette revue, nous nous sommes limités aux techniques de transfert de données à haute vitesse et aux accélérateurs avant de présenter les enjeux du projet RASHPA en cours de développement à l'ESRF.

Pour l'implémentation de la partie réseau dans le cadre de ce projet, nous avons dû prendre en compte un certain nombre de contraintes relatives à la nature même des détecteurs à RX, qui sont des systèmes informatiques embarqués destinés à fonctionner dans des zones expérimentales. Il y a donc une limite à la puissance de calcul qu'on peut raisonnablement embarquer dans l'électronique d'un module de détecteur. Elle est limitée par l'encombrement, la consommation électrique, la sensibilité au RX, à la température, à la pression, etc... Cette électronique de détecteur est à base de circuits FPGA : aussi, il n'a pas été jugé réaliste d'utiliser un protocole réseau trop contraignant (comme TCP implémentant la réémission automatique de paquets potentiellement perdus, ce qui nécessite de les stocker temporairement). Pour la même raison, nous souhaitons utiliser un protocole unidirectionnel, du détecteur vers le dispositif informatique de stockage ou de traitement, sans acquittement obligatoire du bon acheminement. Il était aussi intéressant de pouvoir réutiliser les infrastructures réseau existantes de type Ethernet. Pour toutes ces raisons, le protocole réseau envisagé est RoCE (RDMA over Converged Ethernet). Mais nous étudions aussi une solution basée sur le bus PCI-e grande distance. Nous tenons à avoir une solution générique et "hardware agnostique".

Nous avons bien sûr présenté le principe de fonctionnement d'un contrôleur d'accès direct à la mémoire (DMA) et sa généralisation au RDMA à travers le bus PCI-e ou un lien réseau en utilisant des cartes réseaux dédiées (RNIC).

Nous avons ensuite dressé un panorama des innovations de ces dernières années visant à pallier la loi de Moore (la puissance crête d'un calculateur individuel atteint une limite liée à la physique des composants utilisés) : les approches "calculs parallèles" se sont

généralisées ainsi que le recours à de nouvelles architectures matérielles telles les GPU et les FPGA. Nous avons recensé les solutions possibles avec le matériel NVIDIA et le langage CUDA mais nous avons aussi évalué la faisabilité de notre approche avec le matériel AMD et le langage OpenCL. Nous avons comparé avec les approches multi processeurs basées sur un compilateur et la bibliothèque OpenMP.

Nous décrivons enfin succinctement le projet RASHPA en cours de développement à l'ESRF. Il s'agit de développer une série de bibliothèques de code qui cachent les détails d'implémentation matérielle et spécifient les interfaces nécessaires pour mettre en œuvre des transferts de données concurrents entre détecteurs multi module et des buffers mémoires distants à travers un réseau qui supporte le RDMA. L'objet de la thèse est de supplémenter ce projet encore en phase de définition avec comme récepteur de données les accélérateurs matériels précités.

Transferts de données RDMA

Nous avons dans une première étape évalué les performances atteintes en utilisant le protocole UDP/IP classique et vérifié que le débit maximum théorique de 100Gb/s sans pertes ne pouvait pas être atteint avec les bibliothèques traditionnelles (API Sockets TCP/IP). Les goulots d'étranglement sont multiples : complexité intrinsèque de l'API Socket elle-même, coûts induits par les commutations de contexte lors des appels systèmes, la gestion des avalanches d'interruptions, etc...

Nous avons qualifié parmi les variantes de protocoles RDMA disponibles aujourd'hui le choix de RoCEv2 et du verbe WRITE sur des paires de communications UD comme solution compatible avec les exigences du projet RASHPA (transfert unidirectionnel, placement complètement spécifié des données depuis la source, notification d'évènement tel que la fin de transfert d'une image au CPU). Nous nous sommes assuré que l'emploi de carte réseau RDMA de type Mellanox convenait pour nos objectifs et qu'il n'était pas nécessaire de développer nos propres cartes de réception de données à base de FPGA.

Pour illustrer la versatilité de l'approche proposée, plusieurs simulateurs de détecteurs ont été réalisés, s'appuyant sur les protocoles RoCEv2 ou PCI Express "grande distance" pour la partie transport.

Les premières contributions de ce travail apparaissent dans ce chapitre 3 et ont fait l'objet d'une publication dans le *Journal of Synchrotron Radiation* et d'une communication lors de la conférence *Euromicro/DSD20*. REMU est un émulateur de détecteur complètement logiciel qui suit le protocole RASHPA. Une seconde version matérielle de REMU basée sur FPGA a aussi été développée pour expérimenter la version PCI-e et explorer les techniques d'allocation mémoire requises (il faut en effet être capable d'allouer des buffers de grande taille avec de la mémoire physique contiguë, ce qui requiert un module noyau Linux dédié). Un récepteur de données ROMULU a été développé pour compléter le banc de test.

Nous avons aussi évalué le bloc de propriété intellectuelle (IP ERNIC de Xilinx) qui permet de synthétiser une source RoCEv2 dans un FPGA. Cet IP est devenu disponible tout récemment alors qu'un membre de l'équipe RASHPA assure le développement de notre propre IP et nous avons obtenu les premiers éléments de comparaison.

Traitement de données en temps réel par Accélérateurs

Dans le $4^{ème}$ chapitre dédié au traitement des données en temps réel dans un système RASHPA, nous présentons tout d'abord la définition du concept de RPU (RASHPA Processing Unit) avant d'en proposer différentes implémentations sur CPU, mais aussi sur GPU et FPGA. Cette définition est une contribution majeure au système RASHPA initial qui ne définissait que des buffers de stockage pour les données et ne précisait pas en détail comment le traitement des données pouvait se passer.

Un RPU est un système matériel composé de deux éléments principaux : une carte réseau rapide compatible RDMA et un dispositif de calcul dont la mémoire est accessible par le bus interne de l'ordinateur. Il est complété par un ordonnanceur (Scheduler), une application qui s'exécute sur la machine hôte, qui relève les événements arrivant sur la carte réseau à chaque fin de transfert d'image par un module, et qui se charge d'agglomérer ces multiples événements en un seul événement global qui déclenche le calcul sur toute une pile d'images par le dispositif accélérateur. Cet ordonnanceur doit aussi alimenter en commandes tant la carte réseau RNIC que le dispositif de calcul car ceux-ci ne sont pas complètement autonomes et doivent être alimentés en commandes tout au long du processus d'acquisition.

Un dispositif de translation des adresses mémoires (ATS) permet la conversion en temps réel des adresses virtuelles utilisées en adresses physiques requises par les contrôleurs RDMA.

Nous avons implémenté trois versions de ces RPU sur trois hardware différents : CPU, GPU et FPGA mais nous avons surtout étudié la version GPU et proposé un mécanisme de synchronisation très efficace avec le transfert RDMA. Nous avons même évalué la faisabilité d'une approche utilisant un seul lancement de code GPU persistant afin d'éliminer les temps de lancement qui sont non négligeables quand on fait du traitement en temps réel.

Le traitement de données en temps réel sur FPGA, encore peu pratiqué dans les sciences du rayon X, est évalué en s'appuyant sur les récentes avancées de la synthèse de haut niveau (HLS) qui permet le développement du code de calcul en utilisant les langages de haut niveau en place des HDL traditionnels. Le circuit à base de FPGA proposé, implémente le transfert PCI-e direct (PCIE P2P DMA) depuis la carte réseau Mellanox.

La qualification du pipeline de calcul a été faite en s'inspirant d'une expérience en cours de préparation sur la ligne ID29/EBSL8 de l'ESRF spécialisée en cristallographie série (SSX). Cette expérience va mettre en œuvre un détecteur Jungfrau de nouvelle génération particulièrement performant : il produira 1000 images par secondes, de 2048x2048 pixels en 16 bits. Chaque pixel doit être corrigé par l'application d'un coefficient de gain variable selon l'intensité mesurée et la soustraction d'un piédestal (une image intermédiaire obtenue quand le faisceau X est interrompu).

Le pipeline évalué comprend le pré-traitement des données brutes comme prévu pour un détecteur à gain adaptatif, la réjection d'images en fonction du nombre de pics de Bragg, ou la compression des données au format matrice creuse (CSR). Les algorithmes testés ont été choisis très simples à dessein car il s'agissait dans ce travail d'évaluer la faisabilité et la généralité du traitement du flot de données en temps réel, en masquant

le temps de transfert en parallèle des calculs.

Conclusion

Les principales contributions de la thèse consistent en la qualification de RoCEv2 comme protocole de transport compatible RASHPA, en la proposition d'un mécanisme de support des événements, en la définition d'une unité de calcul RASHPA (RPU) et son implémentation sur plusieurs architectures matérielles. Nous avons vérifié qu'il était possible de transférer les données sans copies extra numéraires vers les accélérateurs qui permettent un accès direct à leur mémoire interne. Nous avons développé un pipeline complet d'analyse de données sur GPU s'inspirant d'un cas réel de cristallographie série dans lequel les calculs se font en parallèle des transferts de données à vitesse maximum.

Nous présentons en conclusion quelques perspectives pour l'avenir de ce projet : nous avons identifié la problématique du stockage de données directement depuis le GPU vers un système de fichier distribué comme étant particulièrement intéressante. Dans la même veine, l'intégration dans RASHPA des dispositifs de stockage désagrégé comme les SSD avec le protocole NVMEoF particulièrement adapté au RDMA est un axe à ne pas négliger non plus. Un autre point clé de la réussite du projet RASHPA réside probablement dans son intégration sans soucis avec les bibliothèques applicatives de calculs distribués type MPI ou NCCL déjà utilisées par les développeurs d'applications scientifiques.