Automatic Mapping of Real Time Radio Astronomy Signal Processing Pipelines onto Heterogeneous Clusters

by

Terry Filiba

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor John Wawrzynek, Co-chair
Dan Werthimer, Co-chair
Professor Jan Rabaey
Associate Professor Aaron Parsons

Summer 2013

The dissertation of Terry Filiba, titled Automatic Mapping of Real Time Radio Astronomy Signal Processing Pipelines onto Heterogeneous Clusters, is approved:

Co-chair _____ Date _____

Co-chair _____ Date _____

_____ Date _____

_____ Date _____

University of California, Berkeley

Automatic Mapping of Real Time Radio Astronomy Signal Processing Pipelines onto Heterogeneous Clusters

Abstract

Automatic Mapping of Real Time Radio Astronomy Signal Processing Pipelines onto Heterogeneous Clusters

by

Terry Filiba

Doctor of Philosophy in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor John Wawrzynek, Co-chair

Dan Werthimer, Co-chair

Traditional radio astronomy instrumentation relies on custom built designs, specialized for each science application. Traditional high performance computing (HPC) uses general purpose clusters and tools to parallelize the each algorithm across a cluster. In real time radio astronomy processing, a simple CPU/GPU cluster alone is insufficient to process the data. Instead, digitizing and initial processing of high bandwidth data received from a single antenna is often done in FPGA as it is infeasible to get the data into a single server.

I propose to develop a universal architecture where each problem is partitioned across a heterogeneous cluster, taking advantage of the strengths different technologies have to offer. I propose we take an HPC approach to instrument development with a heterogeneous cluster that has both FPGAs and traditional servers. This cluster can be reprogrammed as necessary in the same way an HPC cluster is used to run many different applications on the same hardware.

The challenge in this heterogeneous of approach is partitioning the problem. Normal HPC uses a homogeneous cluster where the nodes are interchangeable. In a heterogeneous cluster, there is an additional issue of determining how to partition the problem across different types of hardware. I propose to design a tool that automatically determines how to partition instruments for radio astronomy. In order to do this work, I will need to model the platforms and based on a description of the final instrument, generate a processing pipeline. The partitioning needs to be done using a variety of techniques to assess the hardware. A static model of the hardware is useful to determine the amount of processing available in different types of hardware. Dynamic benchmarking would also be needed to deal with varying server architectures and determine how much processing and bandwidth the cpu/gpu servers can handle. Finally, to capture any overlooked subtleties or deal with things the tools cannot handle, the user will be able to input hints as to how the instrument should be generated.

The development of this tool will be driven by 2 instruments. First, the design of the GBT spectrometer, a spectrometer designed to support many different modes using the same cluster. By using the tool to design this spectrometer, additional modes can be easily added and if the cluster is expanded the each mode can be redesigned to do additional processing that takes advantage of the extra hardware. I will also be working on the LEDA correlator. This is a low bandwidth, large N correlator, which is an ideal application for heterogeneous clusters.

We can assess the performance of this automatic partitioning tool in a number of ways. First, this tool should significantly reduce NRE and time to science. By automatically generating the instrument the need for engineers who understand both science goals and programming is removed. However, this benefit should not come with a large increase in cost. The instruments produced by this tool will be compared to optimized implementations with the same parameters on the basis of hardware utilization and power consumption.

To TODO

TODO

Contents

List of Figures

List of Tables

Acknowledgments

TODO

Chapter 1

Introduction

1.1 Motivation

Chapter 2

Real Time Radio Astronomy Algorithms

Radio astronomy simply refers to the type of science that can be done by observing astronomical objects at radio wavelengths, rather than a specific scientific goal. There is a huge variety of different experiments, such as searching for gravity waves (TODO: ref nangrav), traces of the first stars (TODO: ref PAPER/LEDA), or aliens (TODO: ref SETI). But, despite this variety, the small number of algorithms detailed in this chapter serve as the first step in processing the data for many such projects.

Radio telescopes produce very high amounts of data (TODO: add estimates, how much data?). The reason for this high influx of data is twofold. First, to enable new science, new radio antenna observe increasingly higher bandwidths of data. Second, to sate the need for larger collecting area, rather than designing a large single dish, many new telescopes are being designed as antenna arrays, where the data from multiple antennas is combined to act as a single large dish. While it may be cheaper and easier to get a larger collecting area using small dishes rather than a single large dish, this adds additional complexity processing the data coming off the telescope. Rather than processing a single stream of data, now the instrument must process and combine multiple streams to make the array seem like a single large dish. As the size of the arrays and bandwidths for single dishes simultaneously increase, the data produced cannot be feasibly recorded in real-time. To cope with the progress in science and antenna technology, there is a constant need for new systems to process, rather than record, this data in real time. Each of these instruments begin processing the data immediately after it is digitized, and needs to reduce the data without loosing scientific information. Once the data is partially processed, and reduced to a manageable bandwidth, it can be stored and processed further offline.

There is a small number of real time algorithms commonly used to reduce the data. In this work, we specifically focus on spectroscopy, pulsar processing, beamforming and correlation. Spectroscopy and pulsar processing are both spectral methods of analyzing data from a single beam. They both can be used on antenna arrays but would either need to treat the array as separate dishes rather than one large dish, or the data from the dishes would need

to be combined into a single beam, which can be done using the third algorithm on the list, beamforming. The last two techniques, beamforming and correlation, are both ways of combining data from multiple antennas. Beamforming combines the data into a single beam by delaying and summing the data from each antenna. Correlation doesn't aim to form a single beam, but instead is the first step towards getting an image of the sky.

2.1 Spectroscopy

A spectrometer is simply an instrument that produces an integrated, or averaged, frequency domain spectrum from a time domain signal. A real time spectrometer works by constantly computing a spectrum over short windows of data (channelization), then each channel is summed for a predetermined amount of time to compute the average power in that channel (accumulation). Figure x shows a block diagram for a simple spectrometer design. After digitization, there is the channelization step, where the signal is processed by a digital filter bank, and then the channels are accumulated.

High resolution spectroscopy

Increasing resolution often requires an increase in complexity in the spectrometer design. Once the number of required channels is sufficiently high, it becomes infeasible to compute the spectrum using a single filter bank. To cope with this, the channelization is done in two steps as shown in figure (TODO: add hi res spectrometer block diagram). In the first step, the signal is divided into coarse channels using a filter bank. At this point the channels are much wider than intended and can't be accumulated yet. After coarse channelization, the spectrometer treats the data from a single channel as time domain data and passes it through a filter bank again. This step breaks up the wide channel into a number of smaller channels . At this point the data can be accumulated, since it has the desired resolution.

Applications

This high resolution spectroscopy technique is used in the SERENDIP V.v (Search for Extraterrestrial Radio Emissions from Nearby Developed Intelligent Populations) project. This project is part of the SETI (Search for Extraterrestrial Intelligence) effort to detect extraterrestrial intelligence. The SERENDIP V.v project is a commensal survey at the Arecibo observatory, meaning anytime the ALFA receiver is used for any observation, the SERENDIP spectrometer will also process and record data.

The project is focused on detecting strong narrow band radio signals, requiring a high resolution spectrometer installed at the observing telescope to analyze the data. The spectrometer should resolve channels of less than 2 Hz, so that natural astronomical signals that typically have a wider bandwidth can easily be distinguished from narrower, possibly

extraterrestrial, signals. The SERENDIP V.v spectrometer meets this by providing 128 million channels across 200MHz of bandwidth, for an resolution of 1.5 Hz per channel. Since it would be infeasible to channelize a 200MHz into 128 million channels using a single filter bank, the SERENDIP V.v spectrometer uses the high resolution architecture described in figure x.

2.2 Pulsar Processing

A pulsar processor is an instrument designed specifically to observe transient events, such as pulsars. A pulsar is a rotating neutron star that emits an electromagnetic beam. When the beam sweeps past Earth, due to the rotation of the star, it is observed as a wideband pulse. As the pulse travels through the interstellar medium (the matter filling interstellar space), the pulsar gets dispersed, meaning the low frequencies arrive before high frequencies, despite the fact that they were emitted at the same time.

A spectrometer, as described in Section 2.1, that accumulates the spectrum would smear the pulse, so there will need to be a few adjustments to the spectroscopy algorithm to make it suitable for processing transient events. In the case of pulsars, the algorithm starts with a high-resolution spectrometer without an accumulator. Instead, the algorithm becomes specialized to detect this type of quickly occurring event.

The high resolution data is then sent to a process called dedispersion, which undoes the dispersion caused by the ISM, realigning the pulse. There are 2 techniques to do this. First, the pulse can be dedispersed by shifting the frequency channels by different amounts to compensate for the different delays as shown in figure x, in a process called *incoherent dedispersion*. This process can't be used to reconstruct the original pulse, but due to its relatively low compute cost, is a useful algorithm to search for new pulsars. The second technique, *coherent dedispersion*, models the effect of the ISM as a convolving filter. To remove this effect, the signal is deconvolved with the model. This is more compute intensive than incoherent dedispersion, but can recover the original pulse.

After dedispersion, there is a still a lot of data and the pulse has very low SNR. The next step in processing is *folding*, or adds together many pulses, reducing the amount of data and improving the SNR. At this point, the data has been significantly reduced and can be recorded.

Applications

Some pulsars serve as a very accurate astronomical time keeper. There are millisecond pulsars with extremely stable periods, allowing any perturbations in the observed pulse period to be attributed to some external effect. This makes pulsars extremely useful for conducting relativity experiments. One such example is the North American Nanohertz Observatory for Gravitational Waves, or NANOGrav, Project, which uses pulsars in an attempt to make the first detection of gravitational waves. The project will try to detect gravitational waves by

observing an array of pulsars, measuring the effect of the waves passing between Earth and the pulsars as changes in observed pulse periods.

2.3 Beamforming

Beamforming is one technique for combining data from an array of antennas. The beamformer combines the data from multiple antennas into a single beam pointed at a single point in the sky. This is achieved by delaying the signal from each antenna by a different amount and then summing the delayed signals. As illustrated in figure x, the signal from the intended source will not arrive at all the antennas at the same time. The delay compensates for the disparity between the arrival times, and one the signals are summed it creates constructive interference in the direction of the source, and destructive interference in other directions. These delays can be changed to point the beam at a different source or to track a single source moving across the sky.

BF Beamforming

Since the signal is discrete, and the amount of delay might not be an integer multiple of the sample period s, the delay d is applied to the signal, $f[n]$, in 2 steps. The delay in clock cycles is represented as d/s and broken up into it's integer and fractional parts. First, the integer part is applied as a coarse delay, $n_f = \lfloor d/s \rfloor$. This can simply be implemented by buffering the signal. Applying the fractional delay $p \bmod s$ is more complex. Since there is no observed data at that time, the signal fractional samples are calculated by convolving the signal with an interpolation filter b. Applying these 2 steps, results in a new delayed signal $f[n] = f[n - n_f] * b_{fi}$

In practice, it is common to calculate the spectrum of the beamformed signal. A typical 2 element beamformer, with discrete input signals $f[n]$ and $g[n]$ is trying to calculate the spectrum of beam i, H_i, using coarse delays n_{fi} n_{gi} and delay filters b_{fi} and b_{gi}, as follows:

$$H_i(x) = FT(f[n - n_{fi}] * b_{fi} + g[n - n_{gi}] * b_{gi})$$

We describe this as BF beamforming because the delay operation (B), happens before the FT operation (F).

FB Beamforming

In the case where many beams are required, each beam needs a different FIR filter for every antenna, and a separate FT. This is called FB beamforming because the FT operations (F) occur before the delay (B). Using linearity of the fourier transform and the convolution theorem, the computation can be rearranged as follows:

$$H_i[x] = F[x] \cdot B'_{fi} + G[x] \cdot B'_{gi}$$

Where $F[x]$ and $G[x]$ are the fourier transforms of the signals f and g. The signal delay becomes a phase shift in frequency domain that is represented by B'_{fi} and B'_{gi}. In this case, there is a FT for each antenna, rather than one FT per beam. When forming multiple beams, there is no need to recalculate the Fourier transform of the signals. So, as the number of beams increases, it can be advantageous to use this algorithm rather than the BF algorithm described in the previous section.

Applications

2.4 Correlation

Aperture synthesis is another technique to combine that data from many antennas. The goal is to form an image of the sky, using 2 steps, correlation followed by imaging. In the correlation step, the cross-correlation of each pair of antennas is calculated. Once the cross-correlation is calculated, it is possible to accumulate the data, greatly reducing the amount of data that needs to be processed in the imaging step.

The uv plane represents the Fourier transform of the 2-dimensional sky image. The cross-correlation of an antenna pair represents a point in the uv plane, called a *visibility*. Since the visibilities are not evenly or continuously sampled, the imager must interpolate points on the uv plane in an evenly spaced grid so the FFT algorithm, which relies on even spacings, can be applied to the data. The imager must also account for the fact that the response of the telescope distorts the image, undoing the effect using iterative algorithms like CLEAN or Maximum Entropy. Once this is done, a two dimensional inverse Fourier transform can be applied to recover the sky image.

Typically, large telescope arrays do correlation in real time and finish the imaging offline, but there are a few notable exceptions. One of these is the VLBA, or Very Long Baseline Array. When designing antenna arrays, one important parameter is the distance between the antennas, or *baselines*. Arrays with longer baselines provide images with better angular resolution. The VLBA is an extremely high resolution array, achieving this resolution by using telescopes on opposite ends of the Earth. The distance between the antennas, while useful for science, creates a logistical issue for any real time processing that depends on data from different antennas. Instead of correlating in real time, the VLBA records the digitized data directly from the telescopes at each site, without any reduction. Later, the recorded data is flown to a central location, where it gets correlated.

The cross-correlation of two signals, f[n] and g[n] is defined as:

$$h[n] = f[n] \star g[n] = \sum_{i=0}^{m} f^*[i]g[n+i]$$

Like in beamforming, it is often desirable to calculate a spectrum, so the correlation step typically also calculates the spectrum of the cross-correlations. So, the final result of the

correlator produces the spectrum of the visibilities, H[x].

$$H[x] = FT(f[n] \star g[n])$$

XF Correlation

An XF, or lag, correlator calculates the spectrum by calculating the cross-correlation first (X), followed by a Fourier transform (F).

The calculation of the cross-correlation is very similar to a convolution, in fact the cross-correlation of 2 signals can be re-expressed as a convolution.

For an telescope with n antennas, this algorithm requires $O(n^2)$ cross-correlations, followed by $O(n^2)$ Fourier transform operations.

FX Correlation

The cross-correlation theorem further extends the parallel between correlation and convolution, relating the Fourier transform of the cross-correlation of two signals to the Fourier transforms of the original signals.

$$signal = otherstuff$$

$$H_i[x] = F^*[x] \cdot G[x]$$

Applications

Chapter 3

Related Work

3.1 Radio Astronomy

Digital Signal Processing for Radio Astronomy

DiFX

The DiFX Correlator is a scalable software implementation of an FX Correlator. The correlator was originally developed to do VLBI (Very Long Baseline interfereometery). DiFX was designed as a software correlator in order to maintain flexibility in the design, but that flexibility comes at a high hardware cost. To process 64 MHz of bandwidth and only 10 antennas, about 100 nodes are required to cross correlate all the data.

LOFAR

Like DiFX, the correlator for Low-Frequency Array for Radio Astronomy (or LOFAR) was also built using CPUs. This project used a Blue Gene BG/P to implement a 64 station correlator. The implementation got very high performance out of the cluster, 96% of the peak, but required an entire cluster and required finely tuned code to achieve that performance.

xGPU

CASPER

The need for high bandwidth spectroscopy manifests in many different radio astronomy applications. Keeping up with increasing computation demands has often resulted in the specialized design of spectrometers. At the Collaboration for Astronomy Signal Processing and Electronics Research (CASPER), we have developed a software package to automatically generate spectrometers for a variety of applications.

The CASPER FPGA libraries were developed to mitigate the need to redevelop common signal processing blocks for every new instrument [2]. Parameterized blocks such as FFTs and digital down-converters can easily be used to design many different instruments. Coupled with open source FPGA boards, such as the ROACH (Reconfigurable Open Architecture Computing Hardware), the CASPER libraries provide a useful toolbox for radio astronomy instrumentation development.

The goal of Collaboration for Astronomy Signal Processing and Electronics Research (CASPER) is to provide common libraries, tools, and hardware for radio astronomers developing instrumentation. This is achieved by identifying commonly used DSP blocks in radio astronomy instruments and developing parameterized implementations of these common blocks, making them useful for a variety of instruments. For example, the CASPER library provides FFTs, polyphase filter banks, accumulators, digital downconverters, digital mixers which can be linked together to make an instrument. Figure 3.1 shows the FFT library, which contains different types of FFT like blocks that can process multiple samples in parallel and blocks that only compute the FFT of a real signal.

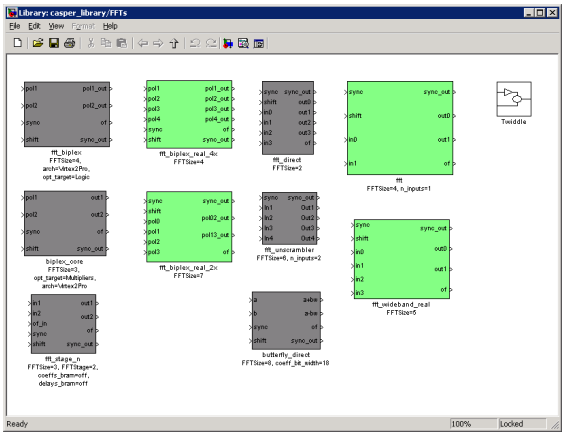


Figure 3.1: TODO

Figure 3.2 shows the options menu for one of the FFT blocks. In order to support a variety of instruments, the block can be reconfigured to support different FFT lengths. There are a number of other parameters provided like input bit width, which helps support a number of different ADCs or preprocessing algorithms, and FPGA-specific parameters like add latency, and multiply latency which have no effect on the result of the computation but change how the FFT gets mapped into hardware.

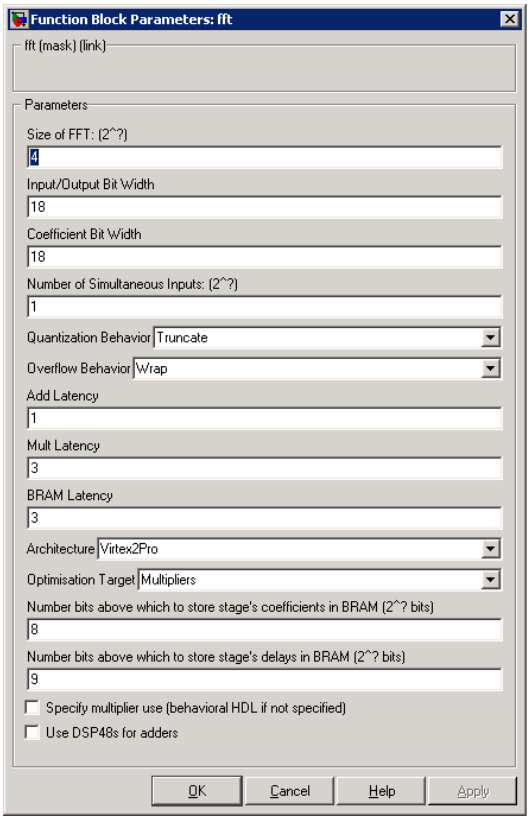


Figure 3.2: TODO

The library is implemented in Simulink, which allows for both simulation and, using Xilinx System Generator, compilation to FPGA code. In addition to software libraries, CASPER also develops general purpose FPGA boards designed for real time radio astronomy signal processing. These boards are designed to provide high bandwidth IO and dense computation. CASPER seeks to develop a small number of boards that can be deployed without a lot of effort and upgraded without doing a major redesign of the instrument. This is achieved in 2 steps. FIrst, the Simulink designs can be retargeted to different boards without changed the original implementation. Second, the boards are designed to communicate using common protocols, like 10GbE, so a board can be upgraded without modifying how it communicates with other boards. The use

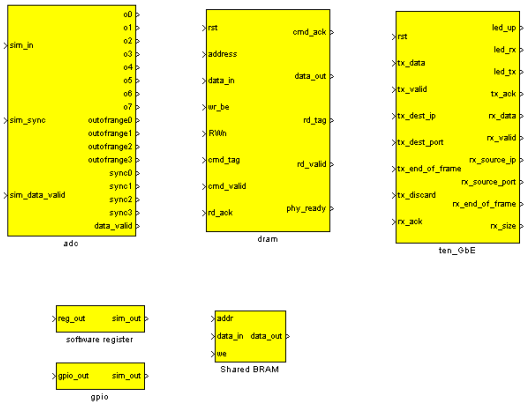


Figure 3.3: TODO

of 10GbE also makes communication to non-CASPER boards simple, allowing an FPGA to create UDP packets that eventually get processed on a CPU or GPU server, making these boards an ideal component of a heterogeneous cluster. This makes it simple to design a cluster, and allows continuous upgrades as technology improves, as the signal processing model and communication model do not change between boards.

To simplify the use of the FPGA further, the CASPER boards run linux directly on the board. Using this linux environment, called the Berkeley Os for ReProgrammable Hardware or BORPH, programming the FPGA is as simple as running an executable on the command line. Then, once the board has been programmed, BORPH can communicate with the chip using a interface where components on the FPGA like registers or memory appear as a file system in the operating system. These files can be accessed using normal file IO, making it simple to send control signals or monitor the status of the chip.

Figure 3.4: TODO

3.2 Tuning

Metropolis

The Metropolis project focused on mapping algorithms onto embedded system. The tool provides a framework for an abstract block based description of an embedded system. This description makes it easy to stitch algorithms together without specifying the eventual hardware implementation, providing a simple path to simulation and algorithm development that is separate from the implemented design. Then, the tool automatically maps the description onto an existing heterogeneous embedded system.

The mapping generated by the tool is simply a schedule, specifying where and when each part of the computation gets executed. The tool seeks to optimize performance of the algorithm, ensuring the generated schedule runs as fast as possible on the hardware provided. This technique requires a fixed hardware model and uses the existing hardware to optimize performance. While this work is useful when a hardware model exists, it does not provide any flexibility in the hardware model while mapping the algorithm. So, when it is necessary to design the hardware to begin with, this does not solve the problem.

Additionally, this type of solution is ill-suited to mapping the computation required to do real-time radio astronomy signal processing. This tool assumes it must schedule a discrete task onto a fixed piece of hardware and attempts maximize the performance of the task.

In the applications described in Chapter 2, the computation should always be running and needs to meet some minimum performance target. Once the performance target is met, it's better to have a tool that will other costs like power or amount of hardware, rather attempting to improve the runtime of the algorithm.

ILP for scheduling

An integer linear programming model for mapping applications on hybrid systems

Chapter 4

High Level Toolflow

Introduction - resummarize problem solved by this work overview of the steps in the process include images from dissertation talk

Instrument design is often done by building the instrument from scratch. This work extends the CASPER philosophy, demonstrating that entire instruments can be generated with minimal user input. Rather than designing a completely different instrument for every different specification, this software package is parameterized so a change in specification only requires a recompile.

4.1 Toolflow Goals

This tool is be accessible to both computer experts and radio astronomers, or other domain experts. For a domain expert, the tool should allows someone to choose the algorithm they need without worrying how it will ultimately map to hardware. The computer expert can to define a new algorithm, and has a way to providing optimization. For example, if an engineer wrote an optimized FFT algorithm, the tool will be able to incorporate that into the final optimized result. And in the end, regardless of who is using this tool, an optimal mapping of the algorithm gets produced.

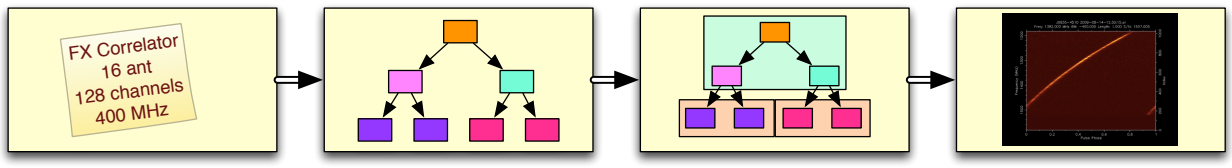


Figure 4.1: TODO Toolflow

These goals are achieved through a four stage toolflow. The rest of the chapter will describe each stage of the toolflow in more detail and explain how they are designed to meet the needs of these different users.

4.2 Instrument Definition

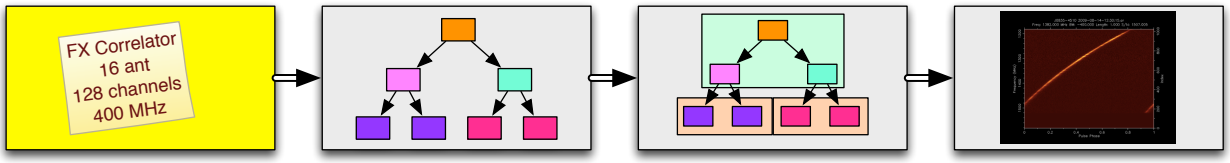


Figure 4.2: TODO Toolflow

At the first step in the toolflow, the user must describe the instrument using high level parameters. These parameters should all

4.3 Dataflow Model

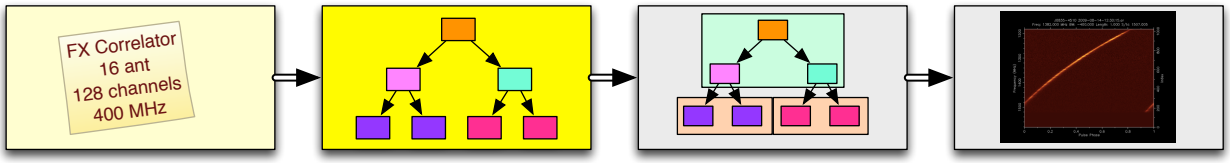


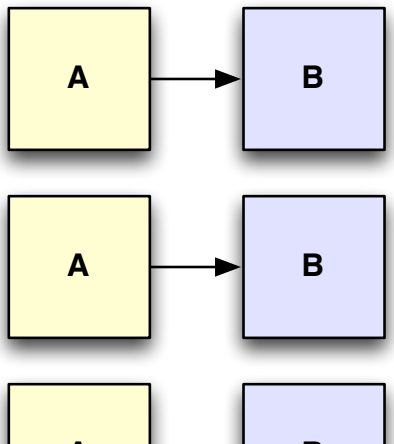
Figure 4.3: TODO Toolflow

Overview

Computational Blocks

Connection Types

Suppose we know we have 2 types of blocks: A, and B. Blocks of type A must send their output to blocks of type B. Now, we need to understand how blocks of type A send data to blocks of type B. This could happen in 2 ways, 'one-to-one' and 'all-to-all'.



A 'one-to-one' connection is where every block of type A communicates with exactly one block of type B, as shown in Figure 4.5. With this type of connection, the number of blocks of type A must be equal to the number of blocks of type B. The F-engines in a FX correlator are a good example of this type of connection. The correlator has an F-engine for each antenna, each containing the same blocks linked in the same way.

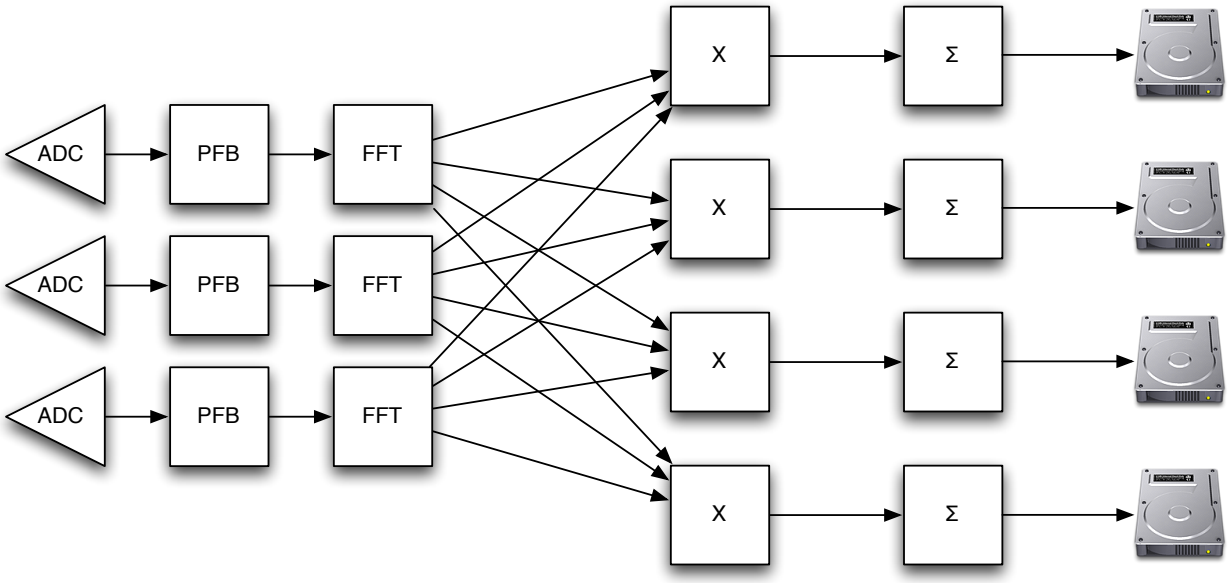


Figure 4.4: Example FX Correlator Dataflow Model

Within an F-engine, a PFB_FIR filter must communicate with a single FFT. In general, every PFB_FIR within an F-engine, blocktype 'A' must communicate with exactly one FFT, blocktype 'B'.

An 'all-to-all' connection occurs when every block of type A must send some data to ever block of type B. Figure 4.6 shows what an all-to-all connection between 3 blocks of type A, and 3 blocks of type B will look like. In this case, every block of type A must send some data to every block of type B. For example, the type of connection between the per-antenna FFTs and the per-channel X-engines in an FX correlator would be 'all-to-all'. Each X-engine needs a small amount of data from every F-engine to compute the cross-correlations from a single channel. In the 'all-to-all' case, there is no reason for the number of sending nodes needs to be the same as the number of receiving nodes.

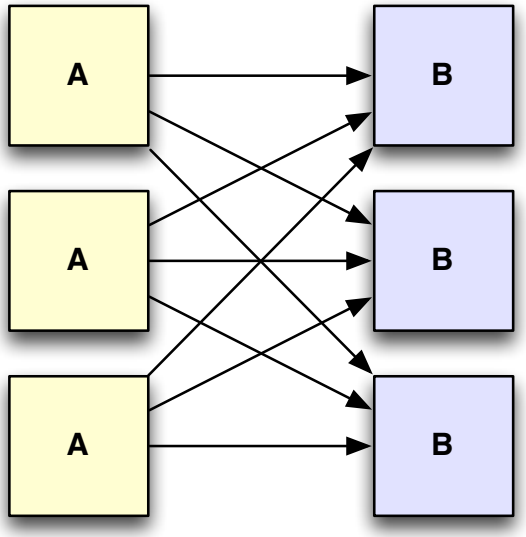
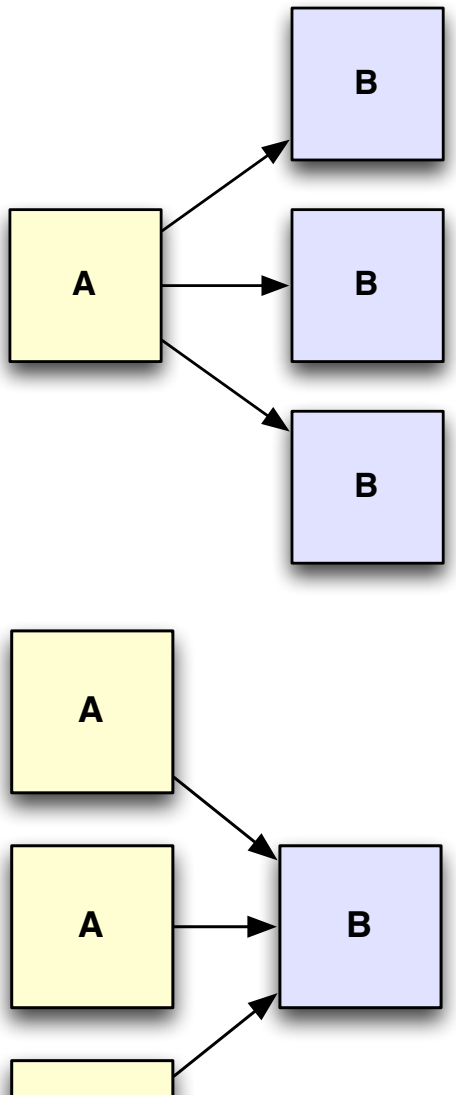


Figure 4.6: TODO

It might seem like there are two more possible types of connection, 'one-to-all' and 'all-to-one' . A data flow with a 'one-to-all' connection, shown in Figure x would have exactly one block of type A that needs to send data to many blocks of type B. This is exemplified in the dataflow for a high-resolution spectrometer. The coarse channelization is done in a single FFT block, which then needs to send the data to many other FFTs to do the fine channelization. The 'all-to-one' connection shown in Figure x is there reverse of the 'one-to-all' case. In this type of connection, there are many blocks of type A and they all need to send data to a single instance of a block of type B. An example of this arises when some processing is done in a distributed manner but the instrument needs to record the final result in a central place. The A blocks are responsible for the distributed processing, and then the B block needs to collect the results and combine them.

It turns out, these are both special instances of the 'all-to-all' connection. The 'one-to-all' connection is simply an 'all-to-all' where the



number of A blocks is fixed at 1. Similarly, the 'all-to-one' connection is also an 'all-to-all' where the number of B blocks is fixed at 1. Because of this, there is no need to include or support these cases as unique connection types.

While it may seem like additional link types exist like 'all-to-some' or 'one-to-some', this turns out to be impossible. Either a block of type A cannot send its data to only some blocks of type B because of the way blocktypes are defined. Any block of the same type should be interchangeable with another block of the same type. In an 'all-to-some' connection, blocks of type A would need to send data to B_1 but not send data to B_2. But that connection patterns implies that the blocks B_1 and B_2 are *not* interchangeable and therefore cannot have the same blocktype.

This does not preclude asymmetrical designs. Instead, asymmetry is supported by allowing blocks to define a list of blocktypes they must send data to or receive data from. The connection between any two blocktypes still must be described as above.

Instrument Dataflows

4.4 Mapping

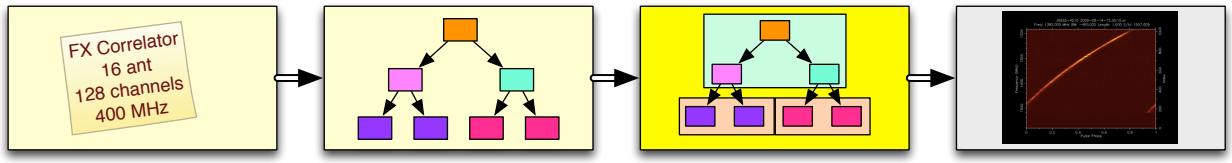


Figure 4.8: TODO Toolflow

4.5 Code Generation

Packetized Astronomy Signal Processor

This instrument has a wide variety of potential applications due to the flexibility of the server software. In this section, we describe a few specific applications than can make use of this package.

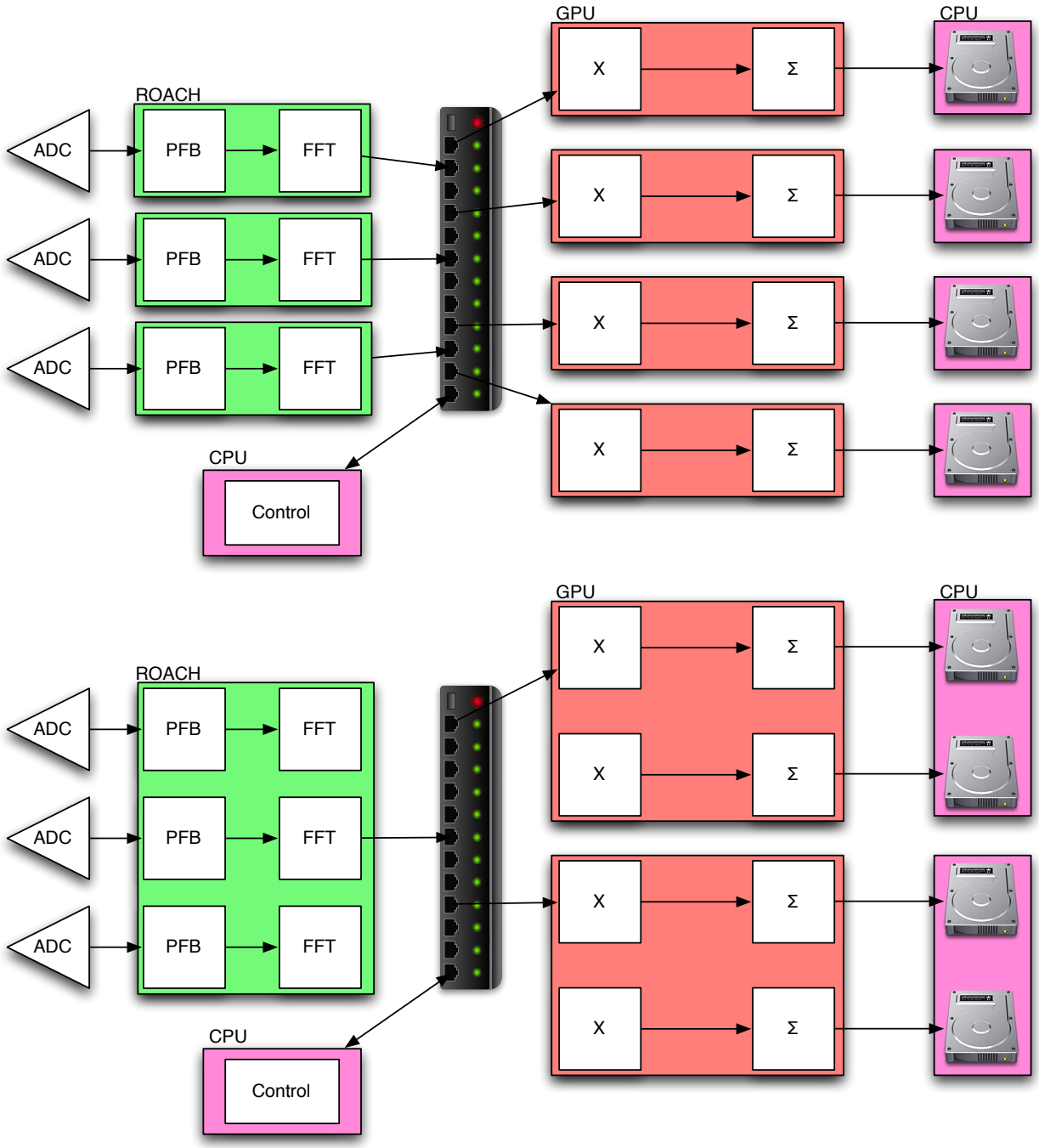


Figure 4.9: Two potential mappings for the FX Correlator

In the search for extraterrestrial intelligence (SETI), the ability to keep up with changes in technology allows searching instrumentation to stay on the leading edge of sensitivity. SETI aims to process the maximum bandwidth possible with very high resolution spec-

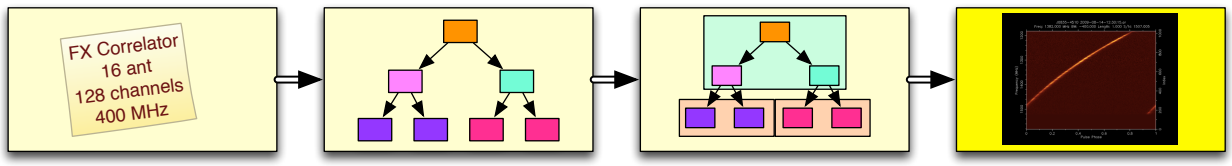


Figure 4.10: TODO Toolflow

troscopy. This instrument allows SETI projects to easily keep up with improvements on the telescope and increasing computational power. An increase in detector bandwidth, improving the breadth of the search, can be processed simply recompiling the FPGA design and distributing the extra subbands to new servers. As computation improves, the instrument can be reconfigured to send more bandwidth to each computer, reducing the required cluster size, or improve the resolution of the instrument by doing a larger FFT on the server.

This design also has applications in pulsar science. The fast channelization on the FPGA with no data reduction makes it an ideal pulsar spectrometer, since no information is lost before sending the data to the servers. GPUs provide a good platform for pulsar processing algorithms such as coherent dedispersion [3], which can easily be used as the processing function for the server software distributed in our package. Similar to SETI instruments, pulsar instruments designed using this package can also keep up with improvements in technology with a simple recompile.

Figure 4.11 gives an overview of the dataflow through the FPGA. The FPGA interfaces to a single ADC board that simultaneously digitizes 2 signals. Each signal can be sampled at a maximum rate of 1Msps. The samples are sent into a polyphase filter bank (PFB), consisting of an FIR filter and an FFT, which breaks up the entire bandwidth sampled by the ADC into smaller subbands. After dividing up the subbands, each band is rescaled. This step allows us to compensate for the shape of the analog filter feeding data into the ADC. After rescaling, the FPGA forms packets where each packet contains data from a single subband. The packets are sent out over CX4 ports to a 10 gigabit Ethernet switch.

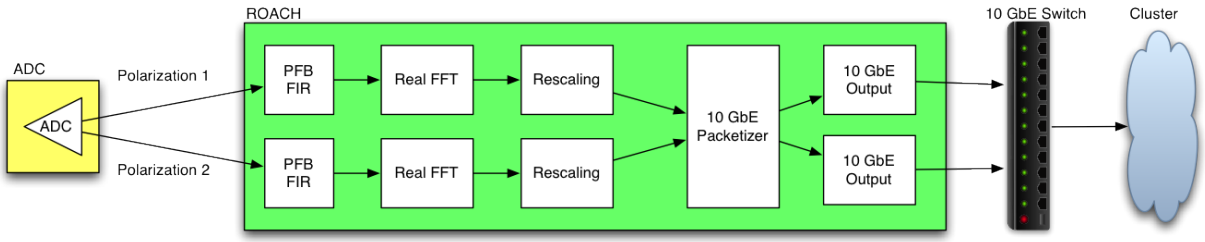


Figure 4.11: PASP Dataflow

PASP uses a PFB to split up the subbands. Figure 4.12 shows a comparison between the FFT and PFB response. The FFT response (on the left) has a lot of spectral leakage while

the PFB (on the right) has a much sharper filter shape and a better frequency response. The superior frequency response led us to use a PFB rather than an FFT to extract subbands, despite the additional FPGA resources required by the FIR filter before the FFT.

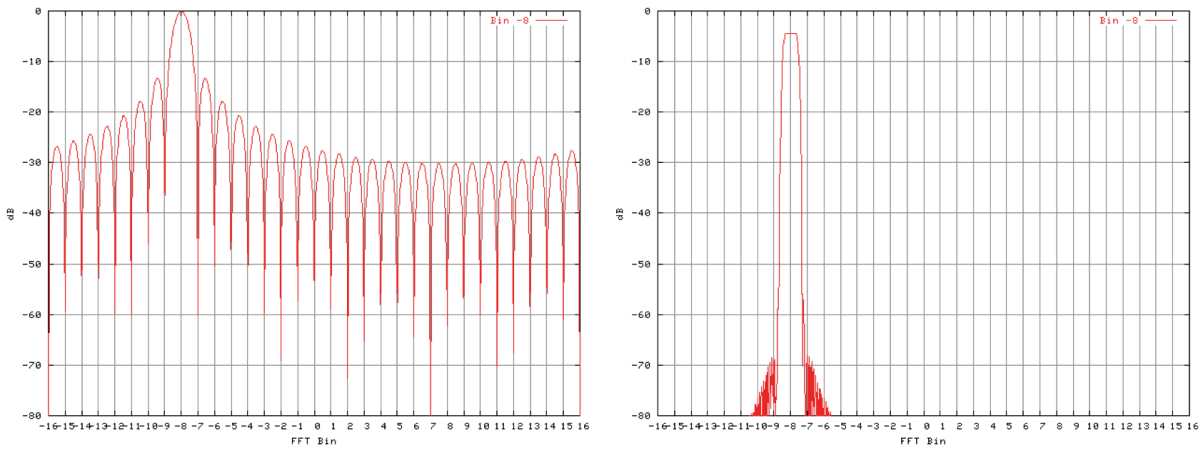


Figure 4.12: A comparison of FFT and PFB response

PASP is designed for flexibility. Building on the CASPER goal to automate the design of commonly used signal processing elements such as FFTs and digital downconverters, PASP automatically designs an entire FPGA instrument using only a few parameters. The user can input the desired number of subbands, CPU/GPU cluster size, and packet size and a new design is automatically generated in Simulink.

Heterogeneous Radio SETI Spectrometer

We have developed a software package to automatically generate spectrometers with minimal user input.

We have automated this design, creating a parameterized spectrometer that only requires a recompile to implement a change in specification. This spectrometer combines FPGAs and GPUs, doing coarse channelization on the FPGA and sending each subband to the GPUs for further processing. The server software is designed for flexibility, allowing astronomers to easily modify the processing algorithm run on the GPU and customize the instrument to fit their science goals.

The software package includes an FPGA design and server software to do spectroscopy, as well as server benchmarks used to determine an optimal instrument configuration. Both the FPGA and server software are parameterized, allowing for rapid deployment of a working spectrometer that is configured to take full advantage of available computing resources. We implement the instrument on a heterogeneous cluster consisting of both FPGAs and GPUs to take advantage of the benefits provided by both platforms. FPGAs provide high bandwidth processing but can be cumbersome to program. GPUs can't handle the same bandwidths

as FPGAs but they are easier to program. The CUDA language, for example, is a C-like language that can be used to develop software for many GPUs. The high level parameters in this package allow us to use FPGAs while abstracting away implementation details specific to the FPGA. To give the user control over their data processing algorithm, an application specific GPU program can be written and easily interfaced with the existing receive software in the package.

The instruments generated with this package use a heterogeneous design, allowing us to benefit from the strengths of FPGAs and GPUs. The FPGA board is able to sample and process very high bandwidths that a single CPU or GPU would not be able to manage; once the FPGA has split up the band the GPU provides a platform that is easier than an FPGA to program but still provides high compute power. A design called the Packetized Astronomy Signal Processor, or PASP, is run on the FPGA. PASP splits up the large band into smaller bands that can be processed using off the shelf servers. The subbands are put into packets on the FPGA and sent over a 10 gigabit Ethernet switch to a cluster of servers. The servers receive the data from the switch and process it using spectroscopy software provided in the software package or special purpose application software written by the user and linked into the provided packet processing infrastructure.

Figure 4.13 shows a high level view of a spectrometer that could be designed with this package. In this example, a ROACH board divides the input band into 64 subbands and sends them out to a 16 server cluster. An ADC is used to digitize data from the telescope and connects to the ROACH board via Z-DOK connectors. The digitized data is split into 64 subbands and sent through a 10 gigabit Ethernet switch. Each server in the cluster receives and processes 4 subbands.

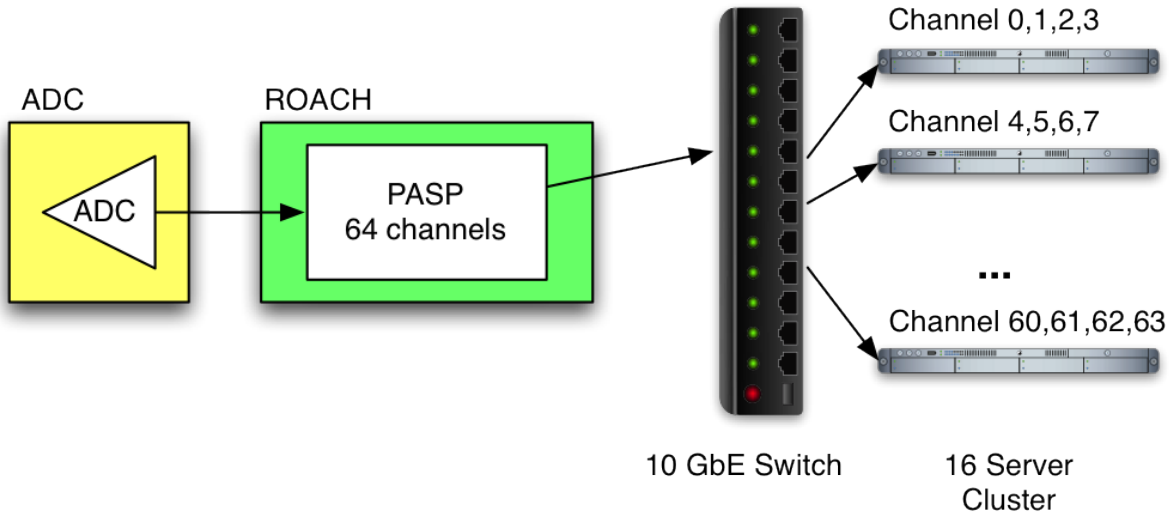


Figure 4.13: Example high level instrument architecture

Server Software

Our package includes spectroscopy software that interfaces with the PASP design. This software receives data over an Ethernet port and transfers it from the CPU to the GPU. The GPU runs an FFT and then sends the data back to the CPU to be recorded. The GPU software, like the GPU benchmark, uses the CUFFT library to run FFT. The FFT size depends on the desired resolution for a specific application and an efficient batch size can be determined by running the FFT benchmark to find the best batch size for the given FFT size.

The server software was designed so other applications could easily be implemented on the GPU without altering or rewriting the receive code that interprets the packet headers and transfers data to the GPU. Once the data is on the GPU, the software calls a process function and passes it a pointer to the GPU data. An initialization function is called before the data processing begins to do any setup needed by the processing function, and an corresponding destroy function cleans up once the processing is complete. In the spectroscopy software included in the package, the initialization function creates the FFT plan, the processing function calls CUFFT, and the destroy function deletes the FFT plan. Modifying the application run on the GPU simply requires a redefinition of these three functions. Using this interface, we successfully replaced the CUFFT processing with software developed for SETI searches designed by Kondo et al. [1].

In this paper, we describe a radio astronomy instrument that is easily reconfigured to suit a variety of applications. Figure **??** shows how this style of instrument design can be extended to a heterogeneous cluster running multiple processing algorithms at the same time. All of these algorithms require the data to be broken up into subbands before it can be processed by the server which can be done on the same FPGA. Using multicast packets, multiple servers can subscribe to the same subbands generated on by PASP and process them in different ways.

This style of instrument design greatly accelerates time to science for many projects. Separating the implementation of the instrument from the hardware specification has created a design that works well for a variety of computational resources and applications. As resources improve, the instrument can improve along with them, providing the opportunity to do new science that wasn't possible before.

Chapter 5

Algorithm Partitioning

After coming up with an instrument description, it is necessary to determine how that instrument will be implemented in hardware.

We use Integer Linear Programming (ILP) to model and solve this problem. As described in Section 3.2, ILP is a powerful technique

5.1 Variables

The variables in the ILP are used to infer the optimal mapping for the system, and the cluster architecture. Ultimately, the ILP needs to determine which boards should be used, and what part of the algorithm should get implemented on each board. This is achieved by having the ILP consider some platform, and assume it can instantiate at most n_p copies of that platform. We will call a copy of the platform a board. Each board must have some variables that determine which computation blocks get mapped to it. For some board i, the number of computation blocks of type b that get mapped to it is represented by the variable $n_{i,b}$.

A solution to the ILP, with each of the $n_{i,b}$ variables filled in, gives a complete specification of the optimal mapping for that instrument.

5.2 Constraints

The constraints serve 2 purposes. First, they ensure that no resource is overmapped, so that the amount of hardware the ILP generates will be sufficient to do the computation required. Second, they make sure that the correct design gets implemented.

Platform Resources

Any resource on some platform that gets used by mapping computation blocks to that platform must be accounted for in the linear program. This is abstracted in the ILP using

a single constraint for each resource.

Resource Limitations

For some resource, r, we use our performance model of each block to assess how much of the resource is used up by each block type. The percentage of the resource r required by some block type, or utilization, is represented by a constant (not a variable), $r_{p,b}$, where p represents the platform, and b represents the block type. Multiplying the resources required for a specific block type by the number of blocks needed on that specific board determines the total percent of resource r block type b will require on the board. Summing over all of the block types determines what percent of resource r is used in the final design, which gives us the final format of the constraint needed to ensure that some resource r is not overmapped on board i:

$$\sum_{b \in Blocks} n_{i,b} r_{p,b} \leq 1 \tag{5.1}$$

Each resource will require a separate constraint in the ILP of this form. By ensuring that the total utilization of each resource required is less than 100%, we guarantee that there are enough resources to allow all the blocks mapped to that board to complete their tasks.

Dataflow Model Constraints

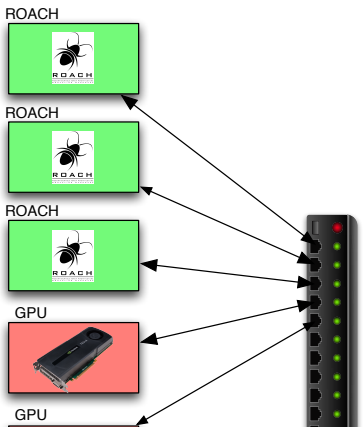
After getting assurance that no resources are overused, it is important to verify that the correct design was implemented. The resource utilization on the value of $n_{i,b}$, but there are is an additional constraint on these variables. Namely, that the correct number of blocks actually get implemented. This adds a simple constraint to each block type:

$$\sum_{board in boards} n_{board,blocktype} == n_{blocktype} \tag{5.2}$$

The total number of blocks of a certain type should be equal to the number of blocks of that type we actually need in the design.

Network Resources

The ILP does not design the network topology. While it would be possible to design the network using an ILP, this would add unnecessary complexity to the program (therefore increasing the runtime) with little gain. As described in Section 3.1, most radio astronomy applications require a full-crossbar interconnect at some point, because many



computational blocks have an all-to-all or one-to-all communication pattern. Rather than have the ILP redesign the same topology over and over, we simply assume this interconnect exists and every board can communicate with every other board directly.

Figure 5.1 shows an example of this topology. Each board gets connected to the same switch and can communicate with any other board on the switch, regardless of the platform type.

Bandwidth Limitations

But, there still are constraints on this communication and this must be taken into account in the ILP. While there might be a link available between each pair of boards, the bandwidth into and out of these boards is limited. Consider Figure 5.1 again. Suppose every other board in the cluster needed to stream 10Gbps of data to the CPU board, but it is only connected to the switch via a single 10Gbps link. There are additional constraints to ensure that the input and output bandwidths are not exceeded.

In order to write these constrains, we must first determine how many blocks need to send or receive data. We introduce a new variables $nr_{i,b}$ to represent the number of blocks of type b on board i that need to receive data from the cluster, and, similarly, $ns_{i,b}$ to represent the number of blocks of type b on board i that need to send data to the cluster. Given the amount of data some computational block type takes as input and the number of those blocks on the board, we can multiply them together to determine the amount of input bandwidth that computational block type will require. Summing over every block type determines the total amount of input bandwidth needed by all the computational blocks on the board, creating a constraint that the total required bandwidth must be less than or equal to the total available input bandwidth. The constraint on output bandwidth is calculated the same way, generating a pair of constraints for each board, one restricting the total amount of input bandwidth, and another restricting the total amount of output bandwidth.

$$\sum_{b \in Blocks} nr_{i,b} bw\_in_b \leq bw\_in_p \tag{5.3}$$

$$\sum_{b \in Blocks} ns_{i,b} bw\_out_b \leq bw\_out_p \tag{5.4}$$

Connection Constraints

First, we observe that the these variables must be bounded by 0 and $n_{i,b}$, since there cannot be a negative number of blocks that need to communicate, and the number of blocks of type b that need to communicate can't exceed the number of blocks physically on the board. Next, we must take into account the structure of the algorithm to determine whether or not a given block needs to communicate with a separate board.

While the constraints on the total input and output bandwidth might seem simple, ensuring the values for $nr_{i,b}$ and $ns_{i,b}$ are sane requires additional constraints. Suppose we know we have 2 types of blocks: A, and B. A is a source of data, meaning it does not data from any computation block. Similarly, block B is a sink, with no data to send to another block. Regardless of how many A and B blocks get placed on platform i, none of the A blocks will need to receive data and none of the B blocks will need to send data. Knowing that A is a source and B is a sink tells us that $nr_{i,A} = 0$ and $ns_{i,B} = 0$.

In order to appropriately define the linear program, it is first important to look at the different ways the computational blocks in a design may need to communicate, and create appropriate constraints. We must revisit the connection types introduced in Section 4.3, and determine how the different types of links affect the linear program.

When two blocktypes are linked via a 'one-to-one' connection, communication is required when the number of A blocks is different than the number of B blocks on a single board. When there are more A blocks then B blocks, $n_{i,A} > n_{i,B}$, the number of A blocks that need to send data to the cluster is $n_{i,A} - n_{i,B}$ and none of the B blocks on that board need to receive data from the cluster. In the opposite case, $n_{i,A} < n_{i,B}$, and the none of the A blocks need to send data to the cluster, but $n_{i,B} - n_{i,A}$ blocks of type B will need to receive data from the cluster. Both of these cases are captured by the same pair of constraints:

$$ns_{i,A} \geq n_{i,A} - n_{i,B} \tag{5.5}$$

$$nr_{i,B} \geq n_{i,B} - n_{i,A} \tag{5.6}$$

When $n_{i,A} - n_{i,B}$ is non-negative, we are guaranteed that we will not underestimate $ns_{i,A}$, and when $n_{i,A} - n_{i,B}$ is negative, $ns_{i,A}$ will be forced to at least 0 because of the lower limit on the variable.

Setting the ns and nr variable is a little more complicated for the 'all-to-all' case, and requires the implementation of some conditional logic in the linear program. When any of the B blocks are not on the board i, then every A block must send data to the cluster, and $ns_{i,A} = n_{i,A}$. Otherwise, none of the A blocks need to send and $ns_{i,A} = 0$. Similarly on the receive side, if any of the A blocks are not of board i, $nr_{i,B} = n_{i,B}$, otherwise $nr_{i,B} = 0$. The conditional logic is easily implemented in an ILP, as describe in X.

When a block must send to or receive data from multiple different block types, the is constraint the same way an 'all-to-all' connection gets constrained. If any of the blocks it must link to reside outside the board, it is assumed that all of the data must be sent over the link to the cluster.

Implementing the ILP this way results in an overestimation of the required bandwidth. In the case where $ns_{i,A} = n_{i,A}$, it's true that every block of type A will need to send some data. However, they might not need to send the full bandwidth bw_out_A to the switch, since some portion of the data sent by an A block may be required by B blocks residing on the same board. A more exact version of this calculation would also take into account the smaller bandwidth, but due to the complexity and rarity of this case the approximation is sufficient. As described in Section 3.1, many architectures have this type of connection but there very few cases where the A and B blocks connected this way reside on the same board.

5.3 Performance Modeling

5.4 Cost Modeling

5.5 Final Mapping

5.6 Optimization

While Integer Linear Programming has a number of desirable properties, the lack of efficient algorithms to solve it can constitute a significant obstacle in designing an ILP with reasonable performance. This section describes how the design of this ILP and the introduction of a few extra constraints serve to improve the performance and scalability of the program defined in this chapter.

Scaling

Symmetry

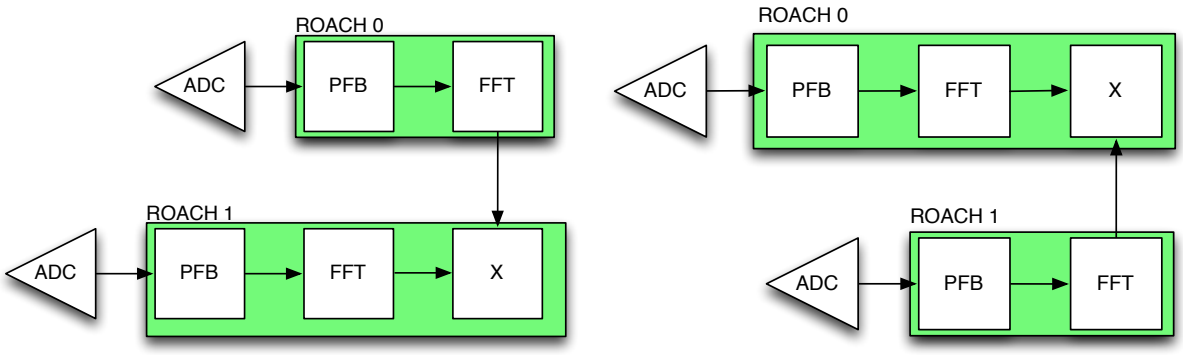


Figure 5.2: TODO

In this type of program, symmetry can often significantly increase the amount of time required to confirm the optimal solution. The boards that have the same platform type are interchangeable, so if board i implements some design and board j implements a different design in the optimal solution, there is another optimal solution where their designs are swapped. For example, suppose 2 ROACH boards are available to implement a FIR filter and an FFT. The ILP might observe that both blocks cannot fit on a single board and assigns the FIR to the first ROACH and the FFT to the second ROACH. This obviously seems like an optimal solution, but the ILP may also need to check the case where the FFT is placed on ROACH_0 and the FIR is on ROACH_1, only to find that it has the same cost as the previous result. In this simple example there was only one other solution to search, but as the ILP and the search space grows the number of solutions that are symmetric to the optimal case will also grow.

Searching symmetric solutions can become a major time sink, because the ILP solver will find an optimal solution early on, but will require a long time to confirm that it is actually the optimal result, spending time going over many other solutions that are isomorphic to the first one.

This returns the current best result the solver knows of, but it cannot guarantee that the solution is globally optimal or, in the case where it is not the a globally optimal mapping, determine if it is close to the optimal solution, because solving that problem would be analogous to solving the ILP. Early stopping works well when it's clear that symmetry is the cause of the long runtime and the amount of time it would take to get one of the isomorphic optimal solutions is short. Even so, the lack of predictable and repeatable results makes early stopping an unappealing solution.

Another solution relies on user aid to guide the mapping. The ILP may spend time going over solutions that may be obviously wrong to a human user. In this case, the user could intervene by setting some of the ILP variables manually and letting the ILP find a solution for the remaining variables. While this may be a feasible solution for a computer expert who might have some idea of what the optimal mapping should be, this not a useful technique for the domain specific experts who are as familiar with the hardware and computational block implementations. This violates one of the basic goals of this tool described in Section 4.1. The tool needs to be accessible and usable by domain specific experts as well as computer experts, and dealing with symmetry in this way will require a computer expert in the loop to generate a mapping and get a cost estimate. To maintain usability for domain experts, guided optimization will not be the solution used to solve this issue.

The solutions above outline ways to cope with the existing symmetry. Another way to reduce the runtime is to remove the symmetry altogether. In order to do this, the ILP must be modified so that only 1 of the isometric optimal solutions is a valid solution to the ILP. First, a variable lex_order_i is added for each board. This variable is meant to uniquely identifies the design running on the board; it is simply the concatenation of all the $n_{i,b}$ variables for that board. Note that the ordering of $n_{i,b}$ variables when the concatenation is done is irrelevant, the only thing that matters is that the order is consistent for every board. When $lex\_order_i = lex\_order_j$ we can infer that for all blocks b, $n_{i,b} = n_{j,b}$. Otherwise, there

must be some block b where $n_{i,b} \neq n_{j,b}$. Now that the designs can be identified, they can be ordered. They are simply ordered lexicographically, by adding the constraints in Equation 5.7 for every $i \geq 1$

$$lex\_order_{i-1} \geq lex\_order_i \tag{5.7}$$

While this make the ILP, adding both constraints and variables, it reduces the amount of time the solver takes to find a solution. This lexicographic ordering makes it impossible to swap designs between different blocks, resulting in unique and valid mappings.

Revisiting the symmetry example at the beginning of this section, the design with 1 FIR and no FFTs would be encoded with a $lex\_order = 10$, and the design the no FIRs and a single FFT would get the encoding $lex\_order = 01$. When the FIR is placed on ROACH_0, then $lex\_order_0 = 10 \geq lex\_order_0 = 01$, satisfying the new constraint. The solution where the blocks are swapped and FIR is on ROACH_1 violates the new constraint $lex\_order_0 = 01 \ngeq lex\_order_0 = 10$, and will not be considered by the ILP solver.

Generalizing this, it is impossible to take a valid solution (with the lexicographic constraint) and get another valid solution by swapping distinct designs between boards. Suppose, without loss of generality, board i has a design with $lex\_order_i = x$ and board j has a distinct design with $lex\_order_j = y$ and $i < j$. Knowing that the design is valid implies $x \geq y$. Another optimal mapping exists where the designs are swapped and $lex\_order_i = y$, $lex\_order_j = x$, but we are guaranteed that this is not a valid solution to the ILP because it violates the lexicographic ordering constraint.

These constraints have been implemented in the final ILP and drastically reduces the amount of time it takes to solve the ILP. The additional constraints do not change the cost of the optimal solution, instead they just reduce the number of valid optimal solutions. By modifying the ILP, the performance is greatly improved without sacrificing optimality or usability.

5.7 Mapped Dataflow

5.8 Performance

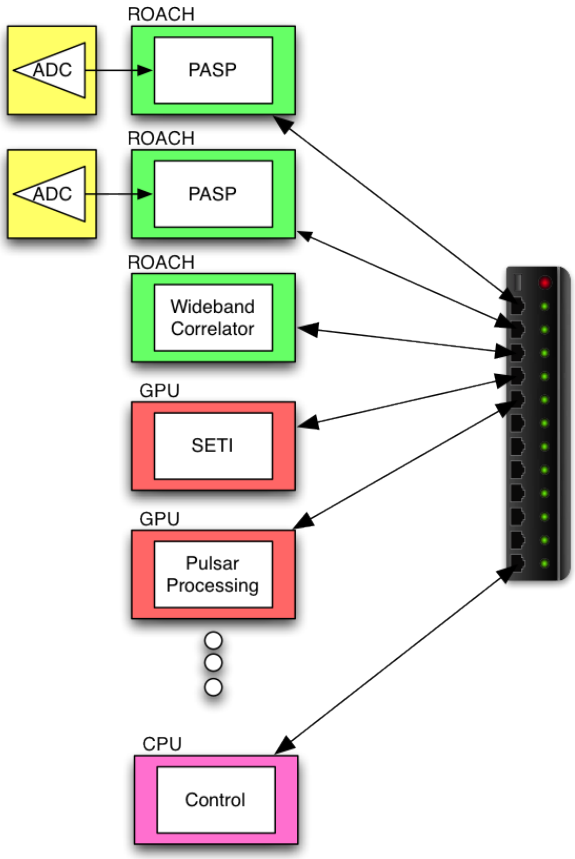


Figure 5.3: A potential architecture for multiple scientific instruments simultaneously processing data from the same telescope

Server Benchmarking

PASP has proven useful in many applications by itself, but the goal of automatically generating a spectrometer for any cluster requires more than just a reconfigurable FPGA design. It is difficult to determine what size the subbands or the cluster should be without knowing how much data the target servers can receive and process. Our benchmarking tools are designed to quickly determine how much bandwidth a server is capable of handling so the PASP parameters can be set appropriately.

We have developed a general purpose benchmark to test the networking capability of a server. This test uses an FPGA design to generate 10 gigabit Ethernet packets and transmits them to the server under test. The FPGA design has a runtime programable packet size and packet rate. The packet size is set to the largest size allowed by the server and the packet rate is initially set low and ramped up while the receive software running on the server checks for dropped packets. By searching for the highest bandwidth with no dropped packets, we

find the maximum allowable data rate where the server should reliably receive all the data.

While specific processing algorithms may vary between scientific applications, an FFT benchmark provides insight into possible processing requirements for a variety of radio astronomy applications. We developed an FFT benchmark using CUFFT, the CUDA FFT library, which supports FFT of arbitrary sizes and allows them to be run in batches on the GPU. Our benchmark tests a variety of FFT sizes and batch sizes. In general, we have found that running larger FFTs and batching many FFTs together is necessary to fully take advantage of the computing resources on GPU. Running this benchmark allows us to determine the maximum bandwidth that can be processed with the available resources.

Using these benchmarks, we can see how much compute power is provided by the server and determine the parameters that need to be entered into PASP. The benchmarks also allow us to identify potential bottlenecks by comparing the maximum bandwidth the server can receive to the maximum bandwidth the server can process. If the system is upgraded to reduce bottlenecks, it is easy to retest the server and recompile a PASP design that takes advantage of the new resources.

Chapter 6

Analysis/Results

6.1 Benchmarks

Discussion of benchmarks (show size vs resources) FFT, PFB,

6.2 Simple spectrometer

Simple spectrometer placement Include appropriate FFT or PFB benchmarks

Cost

Power

6.3 Hi res spectrometer

results for hi res spectrometer (gbt and seti) Same benchmarks as before, just discuss large bw, multi stage fft

Cost

Power

6.4 Pulsar processor

extend to pulsar processor design Need benchmarks for deconvolution algorithm (just cpu vs gpu? infeasible in fpga...) bug jonathon about this

Cost

Power

6.5 Cross-correlator

xgpu benchmarks, correlator placement Need benchmarks for xengine

Cost

Power

Chapter 7

Conclusions

Bibliography

[1] Hirofumi Kondo et al. “A Multi-GPU Spectrometer System for Real-Time Wide Bandwidth Radio Signal Analysis”. In: *ISPA ’10: Proceedings of the International Symposium on Parallel and Distributed Processing with Applications* (Sept. 2010).

[2] A Parsons et al. “Digital Instrumentation for the Radio Astronomy Community”. In: *Arxiv preprint arXiv: ...* (2009).

[3] S Ransom, P Demorest, and J Ford. “GUPPI: Green Bank Ultimate Pulsar Processing Instrument”. In: *American ...* (2009).