

Application of Advanced Multi-Core Processor Technologies to Oceanographic Research

Mark R. Abbott
104 CEOAS Administration Building
College of Earth, Ocean, and Atmospheric Sciences
Oregon State University
Corvallis, OR 97331-5503
phone: (541) 737-5195 fax: (541) 737-2064 email: mark@coas.oregonstate.edu

Grant Number N000141110104
<http://www.ceoas.oregonstate.edu>

LONG-TERM GOALS

Improve our ability to sense and predict ocean processes, utilizing state-of-the-art information processing architectures.

OBJECTIVES

Next-generation processor architectures (multi-core, multi-threaded) hold the promise of delivering enormous amounts of compute power in a small form factor and with low power requirements. However, new programming models are required to realize this potential. Our objectives are to deploy signal processing algorithms onto a variety of “systems on a chip” (SOC) such as those being developed by Intel and NVidia, as well as the application of SOC architectures for other vehicle functions.

APPROACH

The overarching theme of this work relates to the application of advanced heterogeneous processors (both in an embedded environment and in a cluster) to high bandwidth signal processing. Our previous work included the development of a task dispatcher model for rapid development of signal processing applications on the IBM Cell/B.E. platform. This year, we completed several enhancements to this system; including the addition of scheduling tasks on a cluster, heterogeneous (GPU/CPU) computation, and a graphical programming language.

WORK COMPLETED

Our previous research explored how advancements in processor and system architecture influence software design and how they can be leveraged to accelerate signal processing tasks, specifically, the Conventional Beam Former (CBF). The IBM Cell processor was the initial development platform for this research, and we successfully ported the Oasis implementation of the CBF onto it. In addition to implementing the CBF on the Cell, we developed a client-server approach for executing beam forming

tasks. This research expanded into a general model for agile, high-bandwidth, computation inspired by lessons learned.

Porting applications between platforms, languages, and paradigms are each complex tasks. Over the course of our previous work, we ported the beam former across each of these. The most challenging was the difference in programming paradigm between Matlab's managed mathematical constructs and the basic C/C++ constructs required for the Cell/B.E. In a sense, they could not be further apart. Using Matlab, the developer does not need to consider memory management at all. In C, memory management is a constant concern. The Cell/B.E. takes memory management to a new level of complexity, as every transfer to or from the Synergistic Processing Elements (SPE) requires an explicit DMA (Direct Memory Access) operation. It became clear that we would need to develop a suite of tools to abstract the complexities of the underlying hardware to make efficient forward progress.

Our toolkit solves the most difficult problem we encountered while programming heterogeneous multiprocessor systems, which is related to scheduling compute tasks onto the disparate cores in the most efficient way possible. A common approach to multi-threaded programming is to divide an algorithm into a few well-understood parallel constructs. Examples of these include the *pipeline*, and the *parallel for* loop. These examples of parallel division of work are useful because they are distinct and complementary strategies. The *pipeline* executes different tasks on each thread, and the *parallel for* executes the same task on each thread, but with different data. In a *pipeline*, each work unit passes from thread to thread as it moves through the algorithm. In contrast, a *parallel for* executes a distinct invocation of the entire algorithm independently, and in parallel. In each case, the work units must be independent. Every model for dividing work up for parallel processing requires compromise. The *pipeline* approach can yield the best latency, but if its stages are unbalanced, throughput and efficiency will suffer. The *parallel for* is the most efficient, but latency is no better than a serial solution. It is the job of the programmer or architect to analyze the application and make the best decision. This task requires insight and experience, and can still yield less-than ideal results.

We thought that it should be possible to allow the system to make some of these decisions dynamically. Our solution was to develop a system for describing an algorithm in terms of its stages and their dependencies. The core of the system is a scheduling algorithm that executes work units as their dependencies are completed. This is a hybrid between the *pipeline* and *parallel for* approach. If each stage in the pipeline is given a higher scheduling priority, a latency value approaching the pipeline can be achieved. However, if the pipeline is unbalanced, efficiency can remain high by scheduling other work units to fill gaps in execution. The scheduling system is automatic and dynamic, and does not require human effort to achieve excellent results.

RESULTS

We decided that the best way forward was to begin again, and use the lessons learned from our previous work and design a new, enhanced, scheduling system. We are calling this system *Distributed OpenCL*. In the enhanced scheduling system, tasks are defined using OpenCL. Designed to provide an abstract and uniform programming language for CPUs, GPUs, and Accelerators (FPGAs, and the Cell/B.E., for example), OpenCL insulates its users from vendor lock-in that is common when developing for these specialized architectures. In addition to OpenCL, the scheduling system defines a higher level XML-based format for storing and communicating complete algorithms, made up of several OpenCL kernels (units of code in OpenCL parlance). The implementation files produced by this system can be evaluated on a vast array of target hardware. We have extensively tested the system

with a cluster of workstations with NVidia and AMD GPUs. We are actively developing the runtime system for embedded execution.

We have demonstrated the applicability of the Distributed OpenCL for high bandwidth signal processing by implementing the OASIS-developed conventional beam former (CBF). Figure 1 is a screen shot of the CBF implemented using the graphical programming language we developed, which is described in more detail below. Using a library of digital signal processing blocks, the implementation of a CBF could be completed in a matter of minutes. Compare this to the weeks of months that it could take using a traditional programming language. In addition, the traditionally programmed implementation would be highly dependent on the underlying architecture; it is likely that performance would suffer, if it works at all, if the code was moved to another architecture.

In addition to the CBF, we implemented an example software defined radio (SDR) application, which are known for having very high computational demands. The second figure is the implementation of a wide band FM receiver (broadcast FM radio). Its input is two channels of 2.048 million sample per second single-precision floating point data. This data is down- converted, filtered, demodulated, resampled across a cluster and output to speakers in real time.

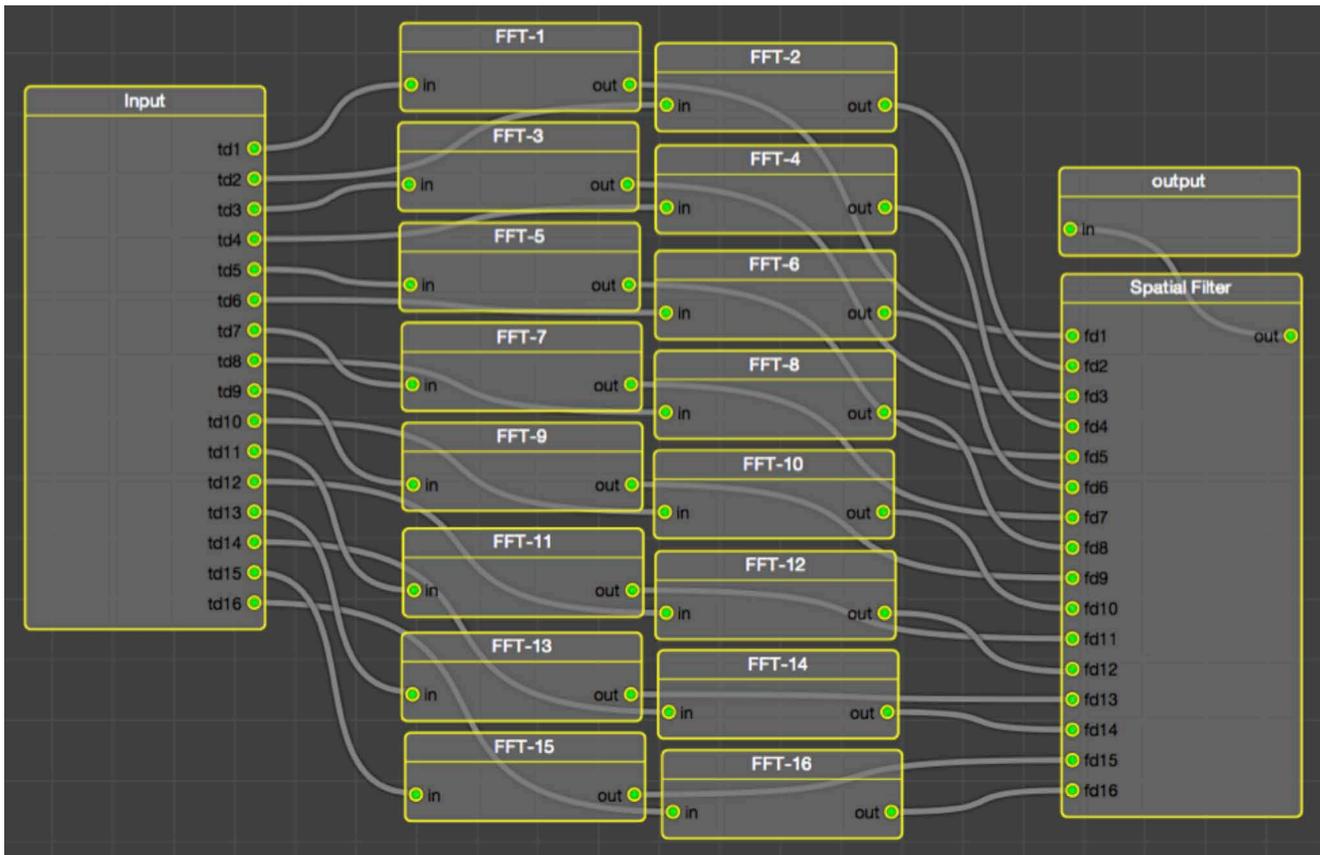


Figure 1. Schematic of a conventional beam former implemented under Distributed OpenCL.

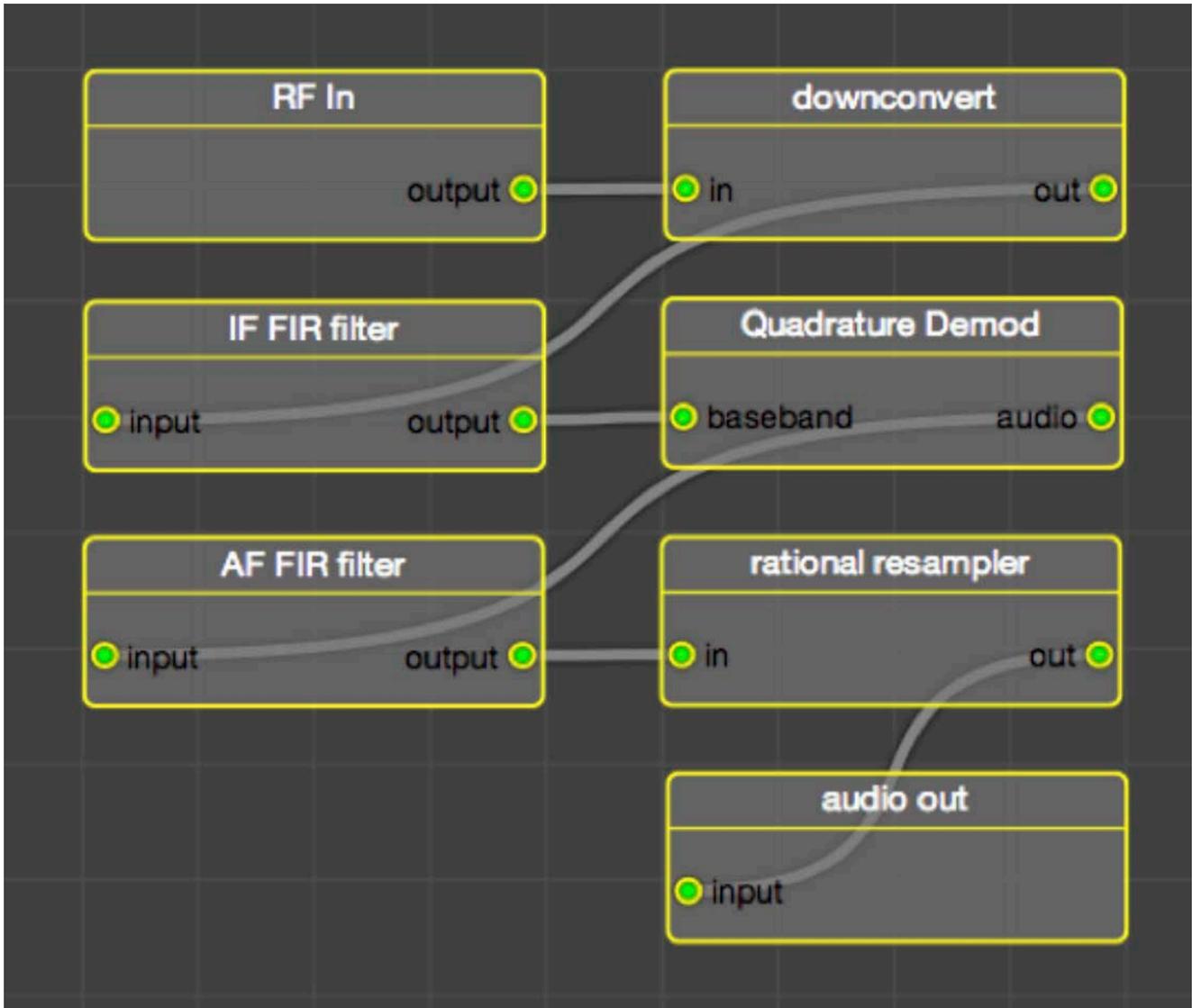


Figure 2. Implementation of a wide band FM receiver in Distributed OpenCL.

The figures provided above are screen captures of the graphical programming language we designed. As each block is implemented with OpenCL, they can be compiled in real time for CPUs, GPUs and Accelerators. The user designates data flow between blocks by dragging wires from their arguments, which are shown with small yellow circles. These wires are encoded into an XML file that defines the structure of the application. The XML file may be evaluated interactively, through the interface, or run in a standalone batch mode. The format of the XML file is clearly defined which allows for the development of many systems capable of executing the application.

We intended this system to be used by a signal processing and intelligence expert, not a programmer. It provides the flexibility necessary for the dynamic environment of modern operations. For example, if an unknown signature is discovered in the field, it is possible to refine the signal processing algorithms to detect and then act on this new information with minimal effort. Furthermore, it should be possible to transmit the algorithm description to an autonomous vehicle while deployed, further enhancing their capability in-flight.

During the early design phases, we decided to leverage standard network protocols for data ingress, intermediate transit, and egress. As a result of this decision, it is possible to use our system within existing network-based workflows. Furthermore, it is possible to insert specialized network functions within an algorithm implemented within the system.

IMPACT/APPLICATIONS

In the next year, we will extend the development of the platform to allow for asynchronous detection and notification of relevant events. It is possible to process a data stream in real time and determine whether a pre-defined condition has occurred, and notify users asynchronously. Leveraging the network support, these notifications can come in the form of database updates, web page changes, text messages, etc.

We are also considering the development of a passive/active situational awareness system that integrates with Distributed OpenCL. When idle, the system will display generic data that could be interesting or relevant to the people that generally occupy a space. Using the Kinect hardware from Microsoft, we could detect when a unique user has approached the device at which point the system will filter the information to match that individual's interests. Furthermore, using a Microsoft Surface, or related multi-touch device, it is possible to interact directly with the real time information feeds.

The goals of the future work are to provide the user tools to interrogate the massive amounts of data that are available today. In conjunction with our scheduling system, we can transform vague and overwhelming data into valuable information, and provide it in a format that respects the human attention span.

RELATED PROJECTS

None

PUBLICATIONS

Dillon, William H., 2012, Distributed OpenCL: a platform for distributed, heterogeneous computing for domain scientists, PhD dissertation, Oregon State University. [published]

HONORS/AWARDS/PRIZES

Mark R. Abbott, Oregon State University, Jim Gray eScience Award, Microsoft Research.