5th SASTech
Iran |Mashhad
May 12 - 17 | 2011
5th Symposium on Advances in Science & Technology

Comparing the structure and executive model of streamIt and Brook Languages

Fateme ghaffarian + f.ghaffarian@gmail.com
Sepide sheykholeslami +sheykholeslami.se@gmail.com
Name of the Presenter: fateme ghaffarian

Abstract

Stream languages are the new and distinct domain of parallel programs and are in a high degree of importance. These languages process the stream datas. There are different types of stream languages. StreamIt is a language for developing stream applications and have a hierachical structure that is designed for stream programming and increase the performance and stability of programs. Brook is another stream language that is designed inorder to correlate the parallel computing idea in data level and messive mathematical computations.The purpose of this paper is to survey the streamIt and Brook languages and computing the executive model of these languages.

Key words: Parallel processing, Stream processing, StreamIt,Brook

1. Introduction

Computing can be described as any activity of using and/or developing computer hardware and software. It includes everything that sits in the bottom layer, i.e. everything from raw compute power to storage capabilities. Recent years have seen the proliferation of applications that are based on some notion of "stream". Stream processing is a computer programming paradigm, related to SIMD that allows some applications to more easily exploit a limited form of parallel processing. The stream processing simplifies parallel software and hardware by restricting the parallel computation that can be performed. Given a set of data (a stream), a series of operations (kernel functions) are applied to each element in the stream. Stream programs represent an important class of high-performance computations. There is evidence that streaming media applications are already consuming most of the cycles on consumer machines, and their use is continuing to grow. Many applications, including media and signal processing, image compression, and scientific applications can be speeded up by several orders of magnitude by processing them with stream processors. Brook and StreamIt are examples of two languages that are used to write stream applications. StreamIt is a programming language and compilation infrastructure designed to facilitate the programming of streaming applications while at the same time maintain efficiency. The StreamIt language has a hierarchical structure that can be best expressed with a graphical environment that can capture and represent the stream graph structure. Brook is a general purpose streaming language that is capable of targeting any stream processor. Brook is very similar to C, making it easy to code and also cast

other existing applications in C-like language to Brook. The goal of this paper is to introduce stream processing and study the StreamIt and Brook languages and compare their execution model. Towards this end, this paper makes the following sections: section 2 describes stream concepts. Section 3 is about stream computing. in section 4 we talk about stream language and study StreamIt and Brook languages and their execution model. In section 5 we compare StreamIt with Brook. Section 6 is the conclution.

2. Stream concepts

Stream and Stream Elements: Streams provide a method for communicating data between memory, kernels, and stream operators. The purpose of declaring streams is to express a collection of objects which can be operated on in parallel. The collection of objects is referred to as a stream while each object is a stream element. The stream variable is a reference to a collection of stream elements. The type of the stream variable determines the stream element type but not the number of elements contained in the stream.

Filter: The basic building block in the StreamIt language is a filter. Each filter inputs some data, processes the data and outputs the new data. The values that the filter reads are taken from its input tape (pop) and the values it writes are placed on the output tape (push). A filter also has a work function which describes the filter's atomic execution step.

Channel: Filters communicate with neighboring blocks using typed FIFO channels. The channels support three operations:

- pop(): remove item from end of input channel
- peek(i): get value i spaces from end of input channel
- push(val): push value onto output channel

3. Stream Computing

Stream computing is a programming paradigm that models a computer program as a stream of data between several processing units, rather than as an implemented algorithm processing data. The stream processing paradigm simplifies parallel software and hardware by restricting the parallel computation that can be performed. Given a set of data (a stream), a series of operations (kernel functions) are applied to each element in the stream.

3.1. Characteristics of stream computing

- Enable new applications on new architectures
- Parallel problems other than graphics that map well on GPU[1] architecture .
- Transition from fixed function to programmable pipelines.
- Various proof points in research and industry under the name GPGPU .
- Data dependencies and parallelism.
- A great advantage of the stream programming model lies in the kernel

defining independent and local data usage. Kernel operations define the basic data unit, both as input and output. This allows the hardware to better allocate resources and schedule global I/O. Although usually not exposed in the programming model, the I/O operations seems to be much more advanced on stream processors (at least, on GPUs). I/O

[1] Graphics Processors

operations are also usually pipelined by themselves while chip structure can help hide latencies. Definition of the data unit is usually explicit in the kernel, which is expected to have well-defined inputs (possibly using structures, which is encouraged) and outputs. In some environments, output values are fixed (in GPUs for example, there is a fixed set of output attributes, unless this is relaxed). Having each computing block clearly independent and defined allows to schedule bulk read or write operations, greatly increasing cache and memory bus efficiency.

4. Stream Programming Language

Stream programming was originally developed for media and image processing applications with simple, regular data access patterns and statically predictable control. More recently it has grown into a more general-purpose programming model successfully applied to scientic applications and other irregular applications .Stream programming language targets to achieve two goals at the same time - efficiency and convenience in writing a streaming application. Since communication overhead dominates over that of computation in a streaming program, a stream processor exposes the communication explicitly into the upper layer so that software handles it to maximize performance. There are some examples of stream programming languages that have been developed and/or are being developed. Brook and StreamIt are examples of two languages that are used to write stream applications.

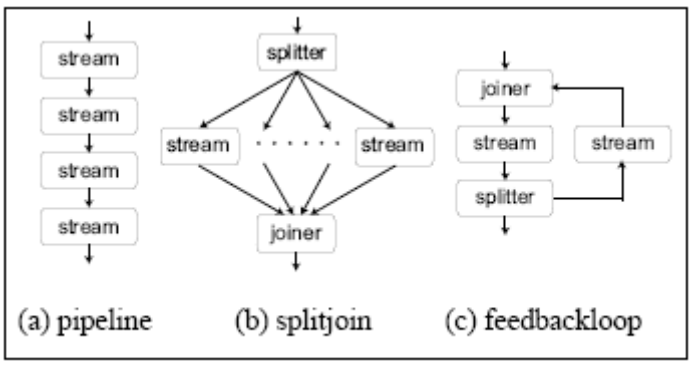

Fig 1. StreamIt containers

4.1. StreamIt Language

StreamIt is an architecture-independent language for streaming applications. It adopts the Cyclo-Static Dataflow model of computation which is a generalization of Synchronous Dataflow . StreamIt programs are represented as graphs where nodes represent computation and edges represent FIFO-ordered communication of data over tapes. The basic programmable unit in StreamIt is a filter. Each filter contains a work function that executes atomically, popping (i.e., reading) a fixed number of item from the filters input tape and pushing (i.e., writing) a fixed number of items to the filters output tape. A filter may also "peek" at a given index on its input tape without consuming the item, this makes it simple to rep-resent computation over a "sliding-window". The push, pop, and peek rates are declared as part of the work function, thereby enabling the compiler to construct a static schedule of filter firings StreamIt provides three hierarchical structures for composing filters into larger stream graphs.

StreamIt provides three hierarchical structures for composing filters into larger stream graphs (see Figure 1). The simplest composite stream is a pipeline. A pipeline contains a

number of child streams, the output of the first stream is connected to the input of the second, whose output is connected to the input of the third, and so on. The split join construct distributes data to a set of parallel streams, which are then joined together in around robin fashion. The feedback loop provides a mechanism for introducing cycles in the graph.

4.1.1. StreamIt applications

Stream processing is essentially a compromise, driven by a data-centric model that works very well for traditional DSP or GPU-type applications (such as image, video and digital signal processing) but less so for general purpose processing with more randomized data access (such as databases). By sacrificing some flexibility in the model, the implications allow easier, faster and more efficient execution. Depending on the context, processor design may be tuned for maximum efficiency or a trade-off for flexibility.

Stream processing is especially suitable for applications that exhibit three application characteristics. **Compute Intensity** the number of arithmetic operations per I/O or global memory reference. In many signal processing applications today it is well over 50:1 and increasing with algorithmic complexity .**Data Parallelism** exists in a kernel if the same function is applied to all records of an input stream and a number of records can be processed simultaneously without waiting for results from previous records. **Data Locality** is a specific type of temporal locality common in signal and media processing applications where data is produced once, read once or twice later in the application, and never read again. Intermediate streams passed between kernels as well as intermediate data within kernel functions can capture this locality directly using the stream processing programming model.

4.1.2. StramIt executing model

StreamIt does not explicitly assume sequential or parallel hardware. A component of the stream graph may fire, possibly in parallel with other firings, if its conditions to execute are met. The scheduler in the compiler chooses an order to execute filters such that the firing conditions are met. A particular back-end may place additional restrictions on firings, a uniprocessor back-end may require that only one stream object executes at a time, or a parallel back-end might add the constraint that two objects may not both be executing if the output of one is an input of another. These constraints are transparent to the programmer.

- A filter may fire when its input has at least as many items as its peek rate. It pops its pop rate from the input, and pushes its push rate onto its output.
- A joiner in a feedback loop or splitjoin may fire when the specified number of items are available on each of its inputs. It pushes the sum of the input weights onto its output.
- A round-robin splitter may fire when the sum of the number of output items is available on its input, and produces the specified number of items on each output.
- A duplicate splitter may fire when at least one item is available on its input. It pops one item from its input and pushes one item to each output.

The programmer should think of each work function as firing atomically. However, the compiler

may choose to run filters in parallel, or to interleave certain work functions (this is sometimes necessary in the case of unbounded I/O rates). Such transformations will not be observable unless the programmer uses print statements (the only function with side effects) within multiple filters.

4.2. Brook

Brook is an extension of standard ANSI C and is designed to incorporate the ideas of data parallel computing and arithmetic intensity into a familiar, efficient language. The general computational model, referred to as streaming, provides two main benefits over traditional conventional languages:

- Data Parallelism: Allows the programmer to specify how to perform the same operations in parallel on different data.
- Arithmetic Intensity: Encourages programmers to specify operations on data which minimize global communication and maximize localized computation.

4.2.1 Brook execution model

A Brook program consists of legal C code plus syntactic extensions to declare streams and denote given functions as kernels. An stream application in brook can be executed via these functions:

- **Streams:** A stream is a new data type addition which represents a collection of data which can be operated on in parallel. Streams are declared with angle brackets syntax similar to arrays.
- **Kernels**: Kernels are special functions that operate on streams. A kernel is a parallel function applied to every element of the input streams. Calling a kernel function on a set of input streams performs an implicit for loop over the elements of streams, invoking the body of the kernel for each element.
- **Reductions**: Brook provides support for parallel reductions on streams. A reduction is an operation from one stream to another stream of smaller dimensions or to a single value. The operation can be defined by a single, two-input operator that is both associative and commutative. Given this property, the elements of the stream can be treated as an unordered set and the operator is applied to combine those elements in any order until it yields the reduced values.
- **Scatter:** The stream Scatter operator performs an indirect write operation taking as input the stream s containing the data to scatter, the index stream containing the offsets within the array to write the data. The fourth argument is the operation to perform to combine the incoming data and the data already stored in the array.
- **Gather**: stream Gather operator performs an indirect read on the array using the index stream to produce a stream of fetch values (t). If the array is multidimensional, the index stream provides a linear offset into the array (based on C row-major array layout). The fourth argument can be any kernel function which consists of a single output stream or a pre-defined operator.

5. Differences between streamIt and Brook

- StreamIT think structured synchronous data flow, but Brook think pointer-less C, with embedded dataflow graphs instead of loop nested.
- StreamIT has single stream grag, but Brook has multiple stream graphs, surrounded by C-subset.
- In StreamIT streams are infinite length but in Brook streams are finite length.
- In StreamIT I/O rates in kernels are static, whereas in Brook I/O rates in kernels are dynamic.
- “Filters” can have state, may require sequential processing in StreamIT, but in Brook “Kernels” must be state-less, allow parallel processing.

- StreamIT is designed by compiler people, clean but more constrained, but Brook is designed by application and architecture people, rough but more expressive.
- Brook is designed for special applications and can not be used commonly, whereas streamIT can be used commonly.
- Brook programming model doesn't depend on the number of processors, but programming in streamIT depends on the number of processors.
- Kernels must have data parallelism in Brook, whereas in streamIT we can have parallelism in both levels (task and data levels).

6. Conclusion

In this paper we studied stream concepts and StreamIt and Brook languages. StreaMIT is not machine specific. The Streamit model helps identify task-level parallelism (eg. Different filters), data dependencies, as well as memory access patterns (eg. FIFOs). Thus, it makes the job of the compiler writer relatively easier. The catch, however, is that the programming model caters to a very limited set of applications. It requires fixed rate streams exhibiting fixed communication patterns. Variable output rates (eg. Data compression) cannot be handled very well by the StreamIt programming model. StreamIt is suited to a very limited class of applications. It scales very well, to as many filters as your application can support. It scales as long as the application data rates scale. Brook is designed to program the next version of Imagine. It has support for multiple dimension streams to allow for scientific computing. Broo**k** language based on the Imagine concept of streams, but It is machineindependent. In Brook, function calls to variable rate streams are allowed, thus ameliorating some of the drawbacks of StreamIt. The programming model is not tied to the number of processors in the underlying machine. Though Brook offers a high level of abstraction, there are quite a few restrictions, no loop carried dependencies are allowed in a kernel. The kernels need to be data-parallel. The usage of pointers is heavily restricted. Static storage classes are disallowed. Recursion is disallowed, and precise exceptions are not supported. There is no support for the integer data type or variable length streams.

Refrences

A. A. Lamb, W. Thies, and S. Amarasinghe. (2009).Linear Analysis and Optimization of Stream Programs, in Proc. of the SIGPLAN '03 Conf. on Programming, Language Design and Implementation, San Diego.

E.L. Waingold. SIFt.(2000). A Compiler for Streaming Applications. Master's thesis,Massachusetts Institute of Technology.

Lokhmotov, A., Gaster, B.R., Mycroft, A., Stuttard, D., Hickey, N.Revisiting.(2009).SIMD programming. In: 20th International Workshop on Languages and Compilers for Parallel Computing.

Miran Dylan.(2002).An Analysis of Stream processing Languages, Department of computing Macquarie University Sydney – Australia.

Prof. Bill Dally, Alex Solomatnikov , Jae-Wook Le Mattan Erez, (2006).Stream Programming Languages/Brook Tutorial. EE482C: Advanced Computer Organization.Stream Processor Architecture.Stanford University.

Saman Amarasinghe and William Thies.(2003).Stream Languages and Programming Models,Massachusetts Institute of Technology

W. Thies, M. Karczmarek, and S. Amarasinghe. (2002).StreamIt A Language for Streaming Applications, In Proceedings of the Internationa l Conference on Compiler Construction, to appear, Grenoble, France.