Graph Programming Model

An Efficient Approach For Sensor Signal Processing

Mr. Steve Kirsch
Raytheon Space and Airborne Systems
El Segundo, California USA
skirsch@Raytheon.com

***Abstract*—The HPC community has struggled to find a parallel programming model or language that can efficiently expose algorithmic parallelism in a sequential program and automate the implementation of a highly efficient parallel program. A plethora of parallel programming languages have been developed along with sophisticated compilers and runtimes, but none of these approaches have been successful enough to became a defacto standard. Graph Programming Model has the capability and efficiencies to become that ubiquitous standard for the signal processing domain.**

Parallel Programming, Graph, Parallelism Extraction, Data Flow Method, Portability, Scalabilitiy

I. INTRODUCTION

Data flow architectures have been used for more than two decades to model signal processing designs. A Directed Acyclic Graph (DAG) is a form of a data flow architecture that has been shown to be very useful in the design and implementation of sensor signal processing parallel programs. The Graph Programming Model is a formal specification for a DAG that has been developed to specifically achieve a set of important goals: efficiency, portability, scalability, and productivity. Recent developments have shown that all of these goals are successfully achieved by the Graph Programming Model.

II. KEY GRAPH MODEL CONCEPT OF OPERATION

A. Data Flow and Signal Processing Implementation Independently Described

The Graph Programming Model defines a method and constructs for describing the data flow totally independent of the signal processing algorithm. This is a unique property not found in any of the myriad of other parallel programming languages. This feature is quite powerful because the most complex and difficult part of integrating any large parallel program is the data movement and data reorganization that must occur between processing. The process for developing a parallel program in the Graph Programming Model involves first implementing and then validating the data flow. The data flow becomes an executable program and can be exercised on the target hardware in dependent of the signal processing code. This has two primary advantages: correctness of the data flow can be validated with simple data patterns flowing through the graph, and the timing associated with the data movement can be assessed early in the design process.

It is often very difficult to validate data flow correctness at the same time as validating the processing of sensor data. Without simple data patterns it becomes an onerous task to assess whether data reorganization or data corner turns have been correctly implemented across all of the parallel processing nodes in the system. The Graph Programming Model Method provides a natural and consistent process for using simple data patterns for validating data flow without modifying or hacking the signal processing code in order to gain visibility into the data flow correctness.

B. Explicit constructs for expressing TLP and DLP

The sensor signal processing domain is blessed with an "embarrassing" amount of algorithmic parallelism. Many parallel programming languages have focused on using various forms of fork and merge constructs to express data level parallelism (DLP) in a section of sequential code. Creation of threads is often used to consume this DLP. However expressing and consuming thread level parallelism (TLP) is often more complex and difficult to automatically abstract from a sequential program. Therefore many parallel programming languages utilize DLP and ignore TLP. However TLP represents a rich area of parallelism to automatically exploit. The Graph Programming Model has explicit constructs for both forms of parallelism.

TLP within a graph is expressed by dividing a graph into Subgraphs. Subgraphs are independently allocable processing code segments. Multiple graphs can also be spawned simultaneously and represent a more coarse form of TLP

DLP is expressed by the use of a Single Program Multiple Data (SPMD) machine. This construct is called a Jobclass. Program instances of a Jobclass are called Jobs. Jobs are independent threads that consume a coarse form of DLP. Fine grain DLP is typically consumed by using a Single Instruction Multiple Data (SIMD) style machine found in many modern processors and is supported by the Graph Programming Model.

C. Dataset Tiling and Job Sequencing

Inputs and outputs from a Jobclass are referred to as datasets in the Graph Programming Model. The data in these datasets are subdivided into tiles. The data within a tile may have processing dependencies (e.g. a vector of N data points is required to perform a N point FFT.) However there are no data dependences between tiles. Therefore a tile denotes DLP boundaries. Jobclass views in the Graph description are used to define the dataset tiles and partitioning across Jobs. The Graph Programming Model allows as many Jobs as there are processing nodes assigned to the Jobclass to execute in parallel thus consuming the available DLP to the maximum extent allowed by the target hardware. The signal processing

simply is designed to operate on a single tile. The parallelism expressed in the Graph description defines the sequencing of tiles which is implemented by the runtime thus greatly simplifying the complexity of the signal processing code. The signal processing is implemented using any common sequential language (e.g. C++) and libraries for accessing the fine grain SIMD capability of a target.

D. Heterogeneous Processing Support

In Space Weight and Power (SWaP) constrained environments, performance/SWAP become the important metric. It has been shown that different algorithms are well suited to different style processors. GPGPUs, FPGAs, arrays of SIMD machines or more generic targets can be the optimal choice for any particular algorithm. In sensor processing as well as other domains a set of algorithms are used to implement the system requirements where more than one of these targets will be optimal. Therefore heterogeneous support in the programming model is key. The Graph Programming Model has capability specifically to support any style of processing (e.g. Streaming, Pipelined, SIMD, offload). The Graph Programming Model achieves this by using a consistent data flow model and a simple mechanism for denoting the target type for each Jobclass. Since the data communication model is consistent across all processor targets in the system a seamless integration of these diverse processor types is easy to achieve.

III. GRAPH PROGRAMMING MODEL EFFICIENCY

The primary goal of any parallel programming approach of course is efficiency. There are multiple aspects of the Graph Programming Model that provide for a highly efficient implementation. A few of the key features are: automated burying of data movement, enforced data alignment, efficient use of memory hierarchy, ability to control the data locality and efficient use of small caches and small local store, and efficient real time load balancing of processing resources

A very important aspect of performance in any parallel architecture is the ability to simultaneously move data and perform useful computation on the data. The dataset tiling feature provides the opportunity to double buffer tiles allowing for simultaneously compute cycles and data movement

Today many real-time signal processing problems are too large for a single SMP machine thus requiring a hybrid SMP and message passing architecture. The Graph Programming Model constructs are used by the runtime middleware to select an optimal transport thus achieving the best performance.

The Graph Programming Model provides a unique opportunity to control data locality because of the data flow aspect of the architecture. As tiles are sequentially processed they are sent to a memory local to the consuming process thus optimizing the data locality of the consuming process while burying the transport time.

Computational load balancing is a key to efficient use of a parallel processor. The Graph Programming Model allows for real-time resource management on a coarse scale when allocating graphs to processing resources and on a fine scale when allocating Jobs to processing nodes.

IV. GRAPH PROGRAMMING MODEL PORTABILITY

In today's environment of rapidly evolving hardware technology, processor chips may become obsolete within 5 years where as the life of the system is required to be 10-20 years. Maintaining a system containing obsolete technologies is very expensive requiring either a life time buy or an expensive porting of an existing code base to a new processor. Given the Graph Programming Model supports a wide range of architectures and abstracts away hardware details, the effort to port code are minimal and typically requires no source code modifications.

V. GRAPH PROGRAMMING MODEL SCABILITY

As hardware parallel computation capabilities increase it is desirable to have the performance of the existing code base also increase in performance with no modifications to the code. This is an attribute where the Graph Programming Model excels. Because both TLP and DLP have been explicitly expressed, more capable hardware can parallelize the exposed DLP and TLP that ran sequentially on the less capable hardware. Simply by adding more processing nodes more Jobs will run in parallel with no change to the source code and more subgraphs or graphs will execute in parallel.

VI. GRAPH PROGRAMMING MODEL PRODUCTIVITY

Finally and perhaps the most important feature of the Graph Programming Model is the increase in productivity and lowering of development costs. There are multiple reasons productivity is increased over other approaches: complexities of data movement are handled by the runtime environment, data flow functional and timing validation is completed early in the design cycle, separation and isolation of the Graph description from the signal processing functionality, the ability to achieve high performance with minimal effort, reuse due to the ability to efficiently scale a design, reuse due to the high level of target hardware abstraction, and a simple well defined and documented XML format for capturing a Graph description.

VII. CONCLUSION

The Graph Programming Model provides a unique approach for designing, developing and implementing an efficient parallel program. It offers the advantages of high performance and efficient parallel design while improving productivity and providing a high degree of scalability and portability.

ACKNOWLEDGMENT

Thanks to Mr. William Posey, Dr. Dwight Mellema, Dr. Troy Wood, Mr. Gerard Richardson, and Mr. Bruce Schmitz for their contributions to the Graph Programming model.