DISTRIBUTED SIGNAL PROCESSING [*]

Li Lee
Alan V. Oppenheim

MIT Research Laboratory of Electronics
77 Massachusetts Ave.
Cambridge, MA 02139

ABSTRACT

This paper presents our recent work on a computational model for performing digital signal processing in dynamically-varying, heterogeneous computing environments. It defines a representation of signal processing tasks in which concurrency and equivalency are naturally expressed. The ability to express equivalency facilitates the system's objective of dynamically adapting implementations of algorithms to the current available resources. Furthermore, we present a simulation program in which our ideas can be demonstrated and verified. The paper presents an experiment showing how execution paths of algorithms can be selected in response to variations in load in the system.

INTRODUCTION

With the recent growth in internets and intranets, it becomes increasingly interesting and important to design new computation models for computing on dynamically-varying, heterogeneous network environments. In [3] we proposed a representation of signal processing algorithms in which alternatives in the implementation can be expressed, and then described a methodology for dynamically choosing the implementation. We found that by interpreting algorithmic expressions as similar to data networks, existing techniques for packet routing in data networks can be extended to dynamically adapt the algorithm to the computing resources. In this paper we start with a detailed description of our model of the distributed network environment. We then present an improved algorithm representation and describe a simulation program demonstrating our ideas and algorithms.

COMPUTATIONAL MODEL

The processing network consists of a collection of specialized virtual processors. One can conceptualize them as servers capable of executing only specific sets of processing instructions or algorithms. In the context of signal processing, for example, these algorithms may include radix-2 FFT, convolution, and sequence addition. The physical manifestation of the virtual processors is unspecified. Hence a virtual processor may reside on one or multiple physical processors, and each physical processor may behave as several virtual processors.

The processing network is assumed to change dynamically and unpredictably due to the random failure and recovery of communications links and network nodes. The function of any particular physical processor may also change. This is modeled by assigning multiple virtual processors to reside within the same physical processor, but allowing only one of them to be working at a time. For simplicity, however, we assume that no catastrophic failure occurs. In other words, every working node is connected to a set of working network nodes with full functionality at all times. Furthermore, we assume that data is not lost due to the failures of the links and processors. This can be accomplished through failure recovery mechanisms within the communications networks itself.

Within the network, data blocks requiring processing are tagged with descriptions of the desired algorithm. As they get processed, the algorithmic descriptions are updated to reflect the current processing needs of the data block. The focus of our research is to adapt the execution paths applied to data blocks to current system resources efficiently and optimally.

[*]This research was supported in part through collaborative participation in the Advanced Sensors Consortium sponsored by the U.S. Army Research Laboratory under Cooperative Agreement DAAL01-96-2-0001. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

PROCESSING GRAPHS

Description

Signal processing algorithms are often graphically represented by dataflow networks, which are directed graphs in which arcs represent data streams and nodes represent operations on the data[2]. A dataflow network easily shows the dependencies among data streams, and naturally expresses concurrency among operations.

In the context of distributed computation, we propose a new form of graph in which concurrency and equivalency in the task implementation are simultaneously expressible. To clarify, operations which can take place simultaneously are called *concurrent*, while those which would have the same effect on the same data are called *equivalent*. For example, different filters in a filter bank can be operated concurrently, while the time or frequency domain implementations of each filter are equivalent. The proposed task graphs express concurrency to facilitate the full use of the massive parallelism that is available on a processor network, and express equivalency to facilitate the dynamic adaptation of the implementations to the current resources of the network.

The new task graphs use directed graphs with nodes representing data blocks and modified arcs representing operations on the data blocks. Notice that the "unit" of data is blocks rather than streams (as in the case of data flow networks). Each block is assumed to be self-contained and processed and transmitted as a unit, even though they may be divided and combined with other blocks as a result of processing. In the task graph, nodes with no incoming arcs are the inputs of the algorithms, and those with no outgoing arcs are the outputs.

The arcs of the task graph, shown in Figure 1 can have single or multiple head and tail nodes corresponding to the number of inputs or outputs in the processing they represent. An task graph composed of nodes and arcs as we describe is therefore hierachical in the sense that each arc can be further represented by another task graph.

Multiple outgoing arcs originating from a single node represent alternatives in how computation on the data block can proceed from that node. It means that the data block represented by the node can be processed by any one of the outgoing arcs. Consequently, multiple paths between nodes imply that there are multiple implementations accomplishing the same processing objectives.

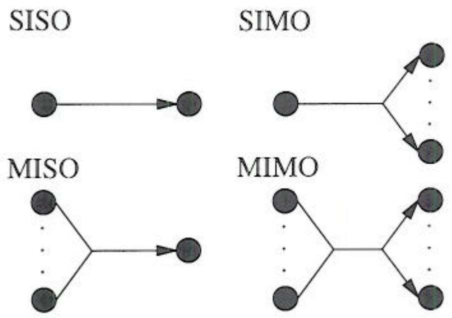


Figure 1: Arcs in task graphs

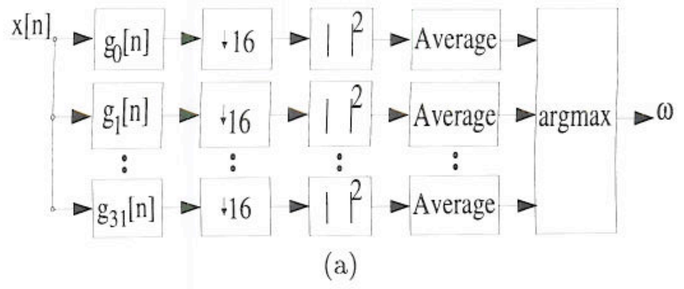
Example

These concepts are perhaps most easily clarified with the example shown in Figure 2. Figure 2(a), shows a block diagram of a modulated filter bank implementation of periodogram averaging to find the peak frequency of a spectrum, while Figure 2(b) shows the corresponding task graph.

In the depicted algorithm, the signal is first passed through a modulated filter bank. The filter outputs are downsampled, and then the average is taken of the magnuitdes of the sequences. The location of the peak in the spectrum is found finding the channel with the highest average magnitude. In the task graph of Figure 2(b), the signal, represented by the leftmost node, is first duplicated. Each of the replications is filtered independently via one of two methods: either through direct convolution, or through multiplication in the frequency domain. This is graphically represented by the fact that the "Convolution" arc connects the same two nodes as the "FFT-multiply-IFFT" path. The next two operations on the filter outputs are downsampling and finding the magnitudes of the samples. Since these two operations are commutative, there are parallel paths showing the two orders of execution. Finally, the channel with highest average output magnitude is found as the peak of the spectrum.

Validity

It is important to point out here that not all interconnections of nodes and arcs represent valid, computable processing task graphs. For example, it is important to verify that all inputs of a multi-input operation are indeed guaranteed to be computed in every possible realization of the actual implementation. For example, in figure 3, we show a simple graph in which MISO operator C operates on nodes 2 and 3. However, in any particular realization of the implementation, either



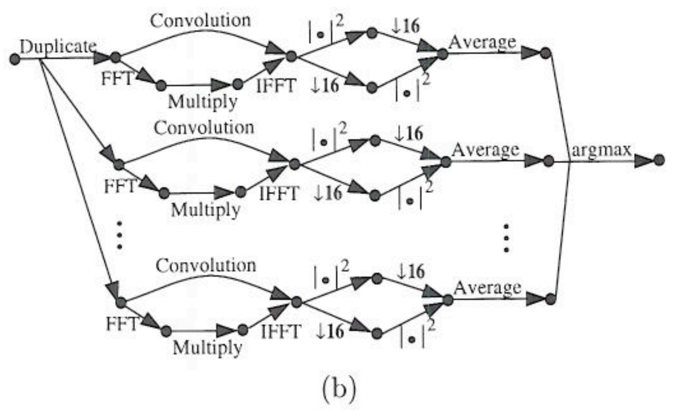


Figure 2: (a) Block diagram of a modulated filter bank implementation of periodogram averaging to find the peak frequency of a spectrum. (b) Corresponding task graph

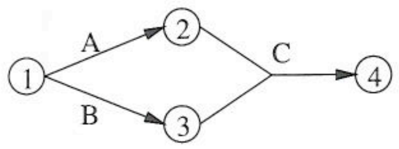


Figure 3: An example of an invalid task graph

node 2 or node 3 (but not both) is computed. Hence the task graph is not computable.

Furthermore, a valid task graph allows us to propagate an initial data block size at the input, and consistently calculate the data block sizes at all of the nodes in the graph. That is, it will not be found that an operation represented by an arc in the graph cannot be used because the size of some input data block is too small or too large. (For example, the inputs to a matrix multiplication operator must insure that the inner dimensions of the inputs are equal.)

While much theory can be developed regarding the verification of task graphs, it is not the focus of our research. Instead, we assume that any graph presented to the system is valid and computable.

SIMULATION

A simulation program has been written in Java which demonstrates the basic ideas discussed here. The program is an object-oriented implementation, and in this section, we describe the program and present the results from one simple experiment.

Input

The simulation program takes as input a description of the characteristics of the computing environment and the processing tasks, including the following:

- The function, processing rate, failure rate, and recovery rate of each processor.
- The data generation rate and processing needs of each sensor. It is assumed that all of the data blocks generated by a sensor require processing by the same algorithm.
- The transmission rates of communications links between sensors and processors.

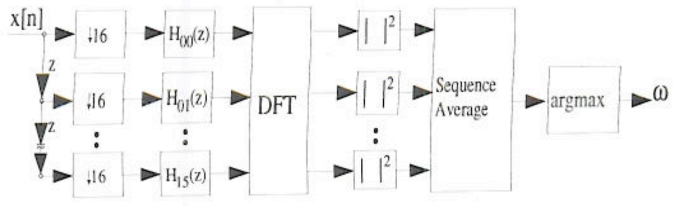
All random processes in the simulation program are Poisson processes. Hence, the time between successive generations of data packets, the time for processors to process each data packet, as well as the failure and recovery time intervals of processors are all taken as exponentially distributed random variables with the prespecified expected values.

As we describe earlier, the algorithm expressions contain alternatives in execution paths so that they can be adapted to the system conditions. The method of adaptation is as described in [3]. Briefly speaking, the adaptation method is formulated to minimize the expected system congestion in the network [1]. More specifically, the adaptation method attempts to minimize the expected number of packets occupying the system by choosing the execution paths appropriately.

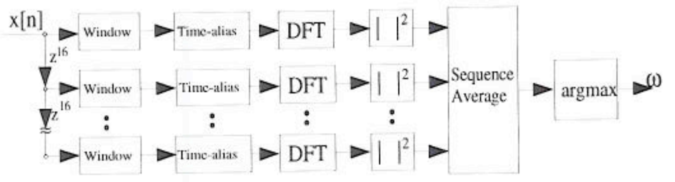
Experimental Setup

The experiment described here concerns the task of spectral analysis by periodogram averaging. In addition to the direct modulated filter bank implementation shown in Figure 2, there are two other equivalent implementations, shown in Figure 4 [4]. Figure 4(a) represents a polyphase implementation of the filter bank, and Figure 4(b) represents the short-time Fourier analysis interpretation of modulated filter banks.

In the experiment, there is a single sensor in the simulated environment, generating data blocks tagged for



(a)



(b)

Figure 4: (a) Polyphase implementation of modulated filter bank. (b) Short-Time Fourier Transform implementation of modulated filter bank.

Convolution	12000	FFT	12000
Multiplication	9500	IFFT	6000
Time-Aliasing	3000	Average	9000
Rect. Window	3000	Argmax	9000
Hamming Win.	3000	DownSample	15000
Autocorrelation	8000	Duplicate	3000
Serial-Parallel	3000	Magnitude	9000
Average Seq.	3000		

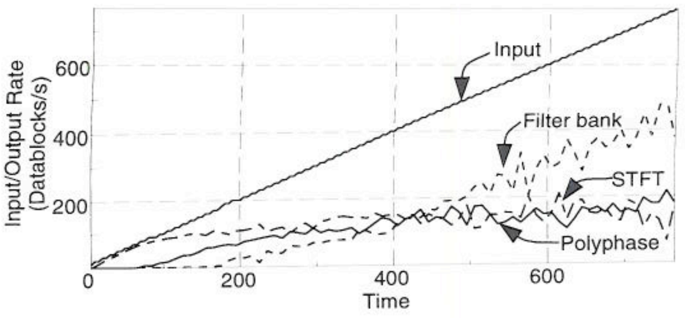
Figure 5: Primitive operations and processing rates in simulation experiment.

spectral analysis by periodogram averaging. The collection of primitive processors, along with their processing rates are shown Figure 5. The experiments tracks the execution path selections of the data blocks as the input rate is increased linearly in time.

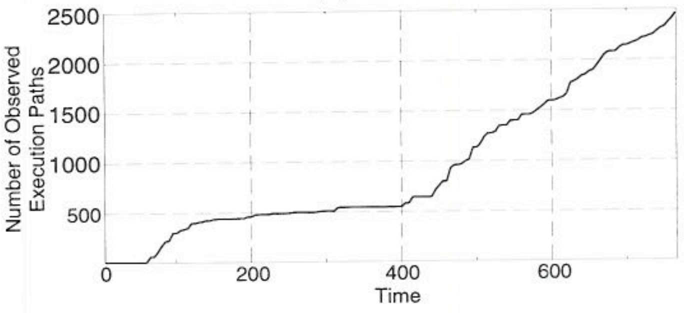
Experimental Results

Figure 6 shows the results of our experiment. In Figure 6(a), we plot the input and output rates of the data blocks versus time. As we can see, under low input rates, the system exclusively chose to use only the STFT and the polyphase implementations. But the capacity of these implemenations is reached quickly, resulting in data output rates remaining rather constant even though the input rate is increasing. The extra load is mostly placed on the filter bank implementation.

Figure 6(b) shows the number of observed execution paths as a function of time. As we expcted, the availability of choices throughout the task graph gave rise to many execution paths. It is interesting to note that this increase occurs non-uniformly in time.



(a)



(b)

Figure 6: Simulation Experiment Results

CONCLUSION

This paper presented an improved representation of signal processing tasks in which equivalency and concurrency are naturally expressed. Furthermore, a simulation program is described and our initial experiments show that the specification of alternative implementations via the proposed task graphs allows the system to accomplish processing objectives without creating bottlenecks in the system.

REFERENCES

[1] D.P. Bertsekas and R.G. Gallager. *Data Networks, 2nd Edition*, Prentice-Hall, 1992.

[2] E. A. Lee and T. M. Parks, "Dataflow Process Networks,", Proceedings of the IEEE, vol. 83, no. 5, pp. 773-801, May, 1995.

[3] L. Lee and A. V. Oppenheim. "Distributed Signal Processing," *ICASSP98* pp. 1749-1752

[4] P.P. Vaidyanathan. *Multirate Systems and Filter Banks* Prentice Hall, 1993.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied of the Army Research Laboratory or the U.S. Government.