

A Framework for Fast Simulation of DSP Algorithms on Multiprocessors

Raghvendra Maloo
Coware India Private Ltd.
Noida Special Economic Zone (NSEZ)
Noida (UP) – 201305, India
Phone: +91 120-2462630
E-mail: rmaloo@coware.com

1 Abstract

This paper describes a framework to enhance the simulation speed of DSP algorithms using multiple processing units. The framework optimally and automatically partitions a DSP algorithm. It further distributes the different partitions to the available processors with the required communication calls. For partitioning the DSP algorithm, the trade-off between computation cost and communication cost is taken into account. While distributing the partitions, the framework considers constraints like the number of available processors, as well as other factors.

2 Introduction

Computer-aided simulation is widely used for the design and verification of DSP systems. For example, in a wireless communication system, the user needs to devise many DSP algorithms like the turbo decoder, convolution decoder and so on. To design such systems, the user needs to model various effects like propagation effects, interference effects, degradation effects, and so on. Designing and verification of such DSP algorithms and modeling the above stated effects needs immense software simulation power.

The complexity of DSP systems has increased significantly over the years. Hence, simulation speed requirements are becoming more and more stringent for such systems. Table 1 [1] shows a comparison of the simulation times for 2G and 3G wireless communication systems. The typical Bit Error Rate (BER) requirement for 2G systems is about 10^{-2} whereas, for a 3G system, it may be as low as 10^{-6} and hence more bits need to be simulated. Moreover, 3G systems need longer fade duration and hence, longer simulation times are required [1]. Even if we assume that the computer speed is improved 100 times, 3G systems require 1000 times faster simulation techniques.

2G Systems	3G Systems
1 hour	41 days
1 day	2.75 years

Table 1 Simulation Run Times

Table 1 depicts the magnitude of the increase in simulation time of DSP systems. The exponential growth in simulation time necessitates a faster simulation infrastructure. It is this multi-processor technology coupled with their ready availability, affordable cost, and dazzling speeds that come to our rescue

In this paper, the problem of increased simulation time of DSP systems is addressed and a framework is defined, which extends the dataflow simulation model to multi-processors. The paper also provides a short analysis of the other related techniques proposed in the past.

The Data-flow model is widely used to represent a DSP system. The approach works on an algorithm-level Data Flow Graph (DFG) representation of the DSP systems. In [section 5](#), Data Flow Graph (DFG) representation of a DSP algorithm is discussed in brief and an extension to DFG as needed by the framework is suggested.

The proposed framework automatically partitions the DFG into the optimal set of sub-graphs and communication edges. Each sub-graph is assigned to a different processor and communication edges are implemented as Message Passing Interface (MPI) [4] calls. In [section 6](#), various parts of the framework are discussed in detail.

The simulation model of the optimally partitioned DFG is explained in [section 9](#). Each processor carries out computation on the associated sub-graph and then, communicates required data to the processors (sub-graphs) connected to it and starts computing for the next iteration. Connected processor(s) receive required data and perform computation on the associated sub-graph. In other words, each processor computes for a part of the iteration concurrently. In this way, the simulation model emulates the software pipeline behavior.

Finally, the experimental data is presented in support of this methodology. More than 50% speed gain could be achieved using 2-3 processors for simulation models of physical layers of standard communication systems.

3 Related work

In the past, various techniques were proposed to distribute simulation of applications having multiple independent simulation runs [1]. In this approach, the problem of distributing single simulation run is addressed and hence it complements previous approaches.

Alamouti and Lundell [1] suggested parallel simulation of 3G wireless communications systems. In their approach, a single simulation with M samples was run as one simulation of $K=M/N$ samples on each N available processors concurrently. The approach was focused on 3G wireless communications systems. In order to obtain valid results from approach [1] a few conditions, for example, ergodicity or removal of initialization bias must hold. Also, the method relies on statistical assumptions.

The approach proposed in the present paper can further distribute the K samples single simulation run of [1] for parallel simulation. Hence, it complements the method suggested in [1].

In SPW MultiProx [3], the user creates a design (DSP algorithm) and partitions the design into subsections called regions. MultiProx, then simulates different regions on different processors. The inter-processor communication is based on the Architecture Description File (ADF).

Multiprox requires knowledge of platform architecture to define the ADF and it also depends on the user's partitioning skills. This makes the approach in [3] useful only for advanced users.

The present paper, discusses a framework which obviates the need to know the platform architecture and does not need manual partitioning of the systems. It uses robust MPI communication for inter-processor communication. This approach should work for all DSP algorithms. In other words, no conditions or statistical assumptions are required to hold for the DSP systems. Also, it requires minimum human intervention and hence less expertise.

4 Platform Architecture

Ideally, the framework should work on all sorts of Multiple Instruction Multiple Data (MIMD) machines. But the performance gain could vary depending upon the processor load and the communication cost.

To describe the concepts involved, a generic parallel processor architecture is assumed (see Figure 1).

In this architecture, processors communicate through message passing. Message passing is a widely used communication model for parallel machines. The Message Passing Interface (MPI) [4] effort defines a standard message passing layer that can be implemented efficiently on a variety of platforms.

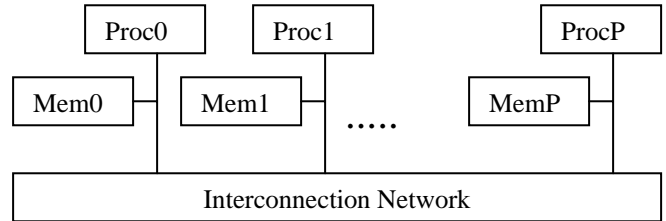
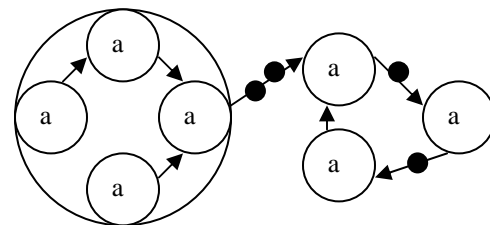


Figure 1 Block Diagram of a Generic Parallel Processor

5 Data Flow Graph (DFG)

Formal semantics for the dataflow model of DSP algorithms have centered on the version of data-flow known as Kahn Process Networks [5]. Various commercial tools use data-flow to specify signal processing systems [6-7]. The data-flow specification can be further partitioned for combined hardware and software implementation [8-9] and co-simulation. This idea is further utilized in this paper to partition and simulate the signal processing systems on multi-processors.

In dataflow, an algorithm is specified by a directed graph (see Figure 2) where the nodes represent computations (*actors*) and the arcs represent *streams* of data *tokens*. The graphs are often represented visually and are typically hierarchical, in that an actor in a graph may represent another directed graph [10].



Hierarchical Directed graph

Figure 2

In Dataflow Process Networks [10], which are a special case of the Kahn process networks, each process consists of repeated *firings* of a dataflow *actor*. By dividing processes into actor firings, the multitasking overhead of context switching incurred in direct implementations of Kahn process networks is avoided. In fact, in many of the signal processing

environments, a major objective is to statically (at compile time) schedule the actor firings (see [Figure 3](#)). The firings can be organized into a list (for one processor) or set of lists (for multiple processors). In [Figure 3](#), a dataflow graph is shown mapped into a single processor schedule.

The lower part of the figure represents a list of firings that can be repeated indefinitely. A basic requirement of such a schedule is that one cycle through the schedule should return the graph to its original *state* (defined as the number of tokens on each arc).

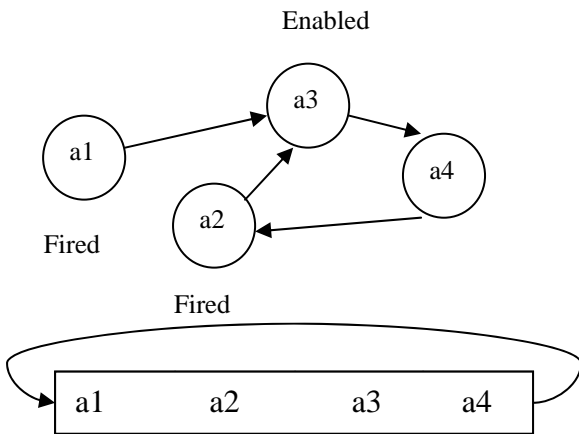


Figure 3

In [section 6](#), a framework is described which works on the Dataflow Process Network representation of the DSP algorithms. The framework optimally partitions the Data Flow Graph. In other words, it optimally divides the one schedule of the actors into a valid set of schedules, which can be performed on the different processors in succession.

This framework requires an extension to the Data Flow Graph representation of signal processing systems. In the extended DFG, each node which represents an actor is attributed a weight to show the quantum of computation on the node. Similarly, each edge that represents the streams of data tokens is annotated with a weight, which reflects the related communication overhead.

The framework requires a weight metric to compute the weight on the nodes and the edges. The weight metric can be a list of pairs of type $\langle \text{actor}, \text{weight} \rangle$ or $\langle \text{edge}, \text{weight} \rangle$. These pairs can be computed by profiling the system. The weight metric can also be expressed as a complex algorithm working on the behavioral specifications of the actors, and the data types and the data sizes of the edges.

The extended Data Flow Graph representation of a typical communication system is shown in [Figure 4](#).

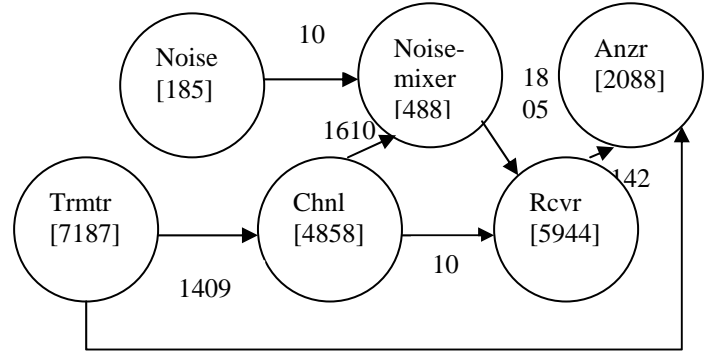


Figure 4

6 Framework Overview

In this section, we introduce a framework that optimally partitions the DFG. In other words, it optimally divides one schedule of the actors into a valid set of schedules which can be performed on the different processors in succession.

In [Figure 5](#), various phases of the tool flow are depicted. These phases work on the Dataflow Process Network representation of the DSP algorithms.

The *Graph Creator* works on a DSP algorithm to generate the extended Data Flow Graph of the system. It uses algorithm data. For example, a weight metric is used to calculate the weight of each actor in the graph or weight of the data stream connecting two actors.

The *Graph Partitioner* is the heart of the tool. It processes the extended DFG and partitions it optimally in sub-graphs. It considers the platform data to determine the optimal cuts in the graph. For example, communication cost and computation cost of the platform is used to resolve the computation and communication cost trade-off.

The *Constraint Manager* imposes the user constraint to the set of sub graphs such that optimality is affected minimally. For example, if number of the available processors is less than the required, it maps more than one sub graphs on the same processor in a way that the parallel computation time of the DFG increases minimally.

The *Code Generator* is the final phase of the framework. It generates code for each actor and tags this code with the corresponding processor ID. If this actor needs to receive/send data to a neighboring actor it appends the proper MPI code to

the generated code. The Code Generator picks the actor in the order specified by the valid schedule.

The Graph Partitioner and the Constraint Manager are discussed further in the following sections.

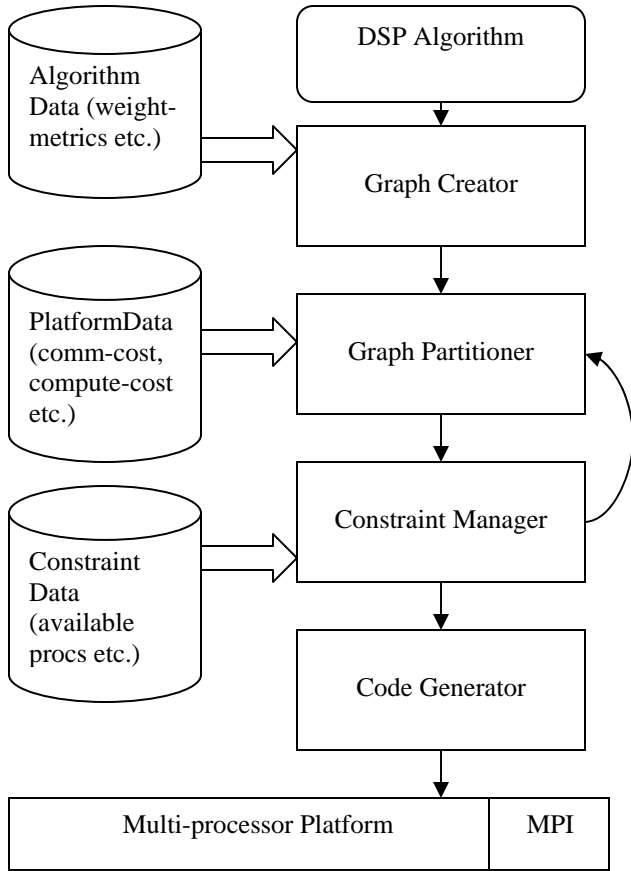


Figure 5

7 Graph Partitioner

The input to the Graph Partitioner is the extended Data Flow Graph (DFG). To partition the DFG optimally, estimation of parallel simulation time of DFG in a particular processor assignment to actors is required.

The *Cost Calculator* calculates the parallel computation time of the given extended DFG. The Cost Calculator virtually simulates the extended DFG according to the processor assignment of the actors. The Cost Calculator considers the various attributes of the node (actors) and edges (data). For example, weight on node, processor ID assigned to the node, communication cost of the connecting edge and so on. The Cost Calculator is heavily used by the Graph Partitioner.

Theoretically, the parallel computation cost is $\text{MAX}(C_0, C_1, \dots, C_{p-1})$, where C_i is time consumed at processor i for the assigned computation and communication.

The Graph Partitioner works as follows:

1. Initialize the DFG
2. For each remaining edge $e \langle a1, a2 \rangle$ in DFG Do Step 2a, 2b
 - a. Try mapping $a1$ and $a2$ on the same processor
 - b. If new cost is higher restore edge e
3. Order the processor assignment

Consequently, the communication system shown in [Figure 3](#) will be partitioned as shown in [Figure 6](#) by the Graph Partitioner.

8 Constraint Manager

The user can specify a few constraints to the tool. Typical constraints include:

- Number of available processors for the simulation
- A few actors pinned to a particular processor
- A few actors tied always together
- Limit on the maximum load on a processor

The Constraint Manager applies various techniques to satisfy user constraints.

Constraint type 1: Number of available processors (n). If the Graph Partitioner recommends the use of p processors and $p > n$, the Constraint Manager remaps few cuts of the DFG on the same processor such that increase in parallel computing cost is minimal.

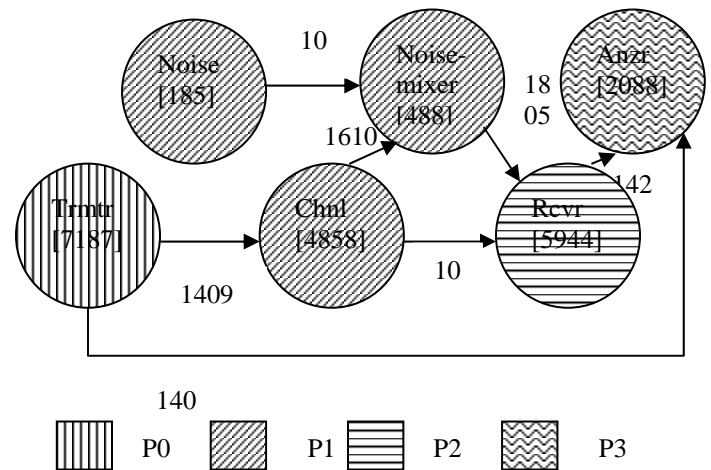


Figure 6

Constraint is matched as follows:

1. Find out procCommMap[][]
2. Calculate the parallel computing cost of the DFG
3. While $p > n$ Do Steps 3a, 3b, 3c
 - a. For each pair of alive cuts $\langle x1, x2 \rangle$
Find $\text{MIN}(\text{cost}(\text{merge}(x1, x2)))$
 - b. Merge x_{min1} and x_{min2}
 - c. Update procCommMap[][]
4. Order the processor assignment

procCommMap is a matrix, $m[i, j]^{\text{th}}$ element of this matrix represents the communication cost of the edge (if any) between DFG cut X_i and X_j .

For example, if the user only has 3 processors to simulate the system described in Figure 6, The Constraint Manager will map cut associated with P3 to P2.

Constraint type 2: A few actors pinned to a particular processor. The Constraint Manager annotates the actors with the processor ID and reruns the Graph Partitioner.

Constraint type 3: A few actors tied always together. The Constraint Manager merges the edges between given actors without considering the parallel computing cost and reruns the Graph Partitioner.

Constraint type 4: Limit on the maximum load on a processor. If the load on any of the processors exceeds the given limit, the Constraint Manager tries to ungroup the actors represented by hierarchical directed graph and reruns the Graph Partitioner. This constraint may not be satisfied for a few cases.

9 Simulation Model

This section describes how the methodology described in this paper provides a faster simulation of DSP algorithm.

DSP algorithms are simulated for multiple iterations. One iteration of the simulation is equivalent to one cycle through the schedule of the system. In this approach, one single schedule is partitioned into a set of schedules, which if run in succession ensures valid simulation results. Further, each element of the schedule set is performed on a different processor and the computed data is communicated to the processor performing computation for the next element of the schedule set.

In this way, the simulation model in the discussion emulates the software pipeline behavior. Each processor works on the part of the iteration and starts working for the next iteration. This increases the throughput of the

simulation platform and hence the speed-up. Figure 7 illustrates the above simulation model pictorially.

10 Practical Issues

The above approach is quite general and needs some refinement in a few cases to be practicable. In multi-rate designs, rates on different actors impose a constraint to tie the actors together in different groups. This leaves a little scope for the Graph Partitioner to cut the DFG.

In the DSP designs a large buffer is used to share data between different actors, but during one simulation iteration only a small chunk of data is modified. An issue with the current methodology is that it communicates the whole buffer, irrespective of the fact that big chunk of the buffer is unmodified. It calls for a smarter way of communication.

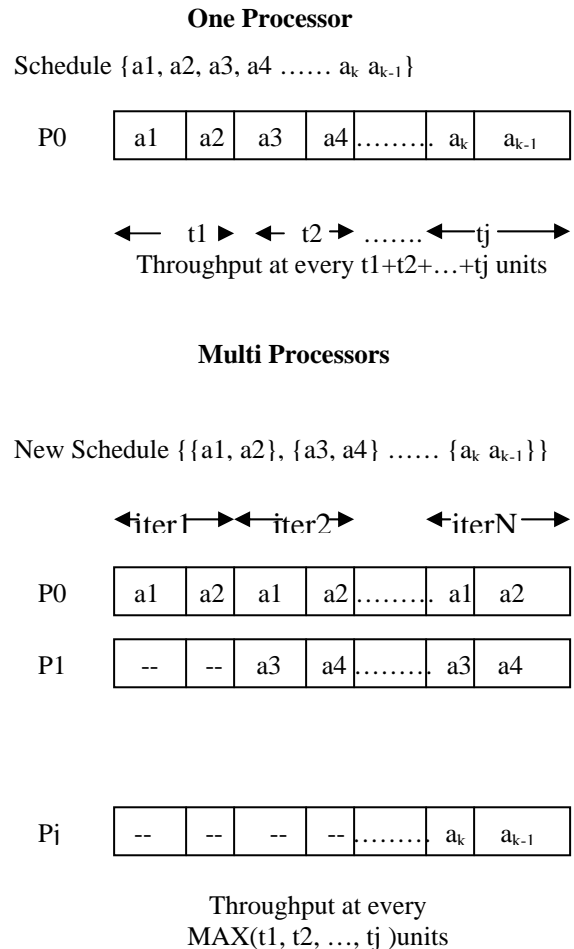


Figure 7

11 Experimental Results

The simulation model described in [section 9](#) requires a multi-processors platform. Ideally, there should be a cluster of high-speed processors with negligible communication cost and appropriate storage capabilities. But, if the simulation platforms are expensive machines, such an approach seems infeasible for widespread use. Hence, the methodology must be applicable to less expensive processors and operating systems as well.

This framework is implemented inside SPW [2]. For experimentation, a cluster of PCs running Linux (RH8) was used. To support a message passing communication model, MPICH2 [11] is installed on these machines. Table 2 shows the experimental data.

DSP Systems	#iterations	% gain	#Proc used
Digital Radio	100000	50%	2
Cable modem	100000	52%	2
802.11a*	200	30%	3
802.11a*	1000	56%	3

*system required manual modifications to achieve performance

Table 2

The data in Table 2 proves the methodology described in the paper. For the systems shown in the Table 2 more than 50% gain could be achieved by using 2-3 processors. The experimental data shows that the speed gain increases if the system is run for more number of iterations.

For a few other systems, like an MPEG-4 system, the Graph Partitioner is verified manually and the partitions are found to be close to optimal.

12 Conclusion

In this paper, the simulation speed issue for large DSP algorithms is addressed. A methodology to use multi-processors that can be a cluster of Linux PCs, for faster simulation of complex DSP systems, is proposed. This approach is quite automatic and requires a one time collection of the platform data. The user of the tool can specify constraints like the available number of processors and so on.

The framework first represents the DSP algorithm as a DFG. The Graph Partitioner phase of the framework optimally cuts the DFG into a set of sub-graphs, which can be processed on different processors. While partitioning the DFG, it considers the trade-off between

the cost of computation and the cost of data communication. The Constraint Manager, the next phase of the framework, works on the set of sub-graphs and imposes the constraints specified by the user. Finally, the Code Generator generates the code for simulation with embedded MPI calls for carrying out the required communication.

The experiments prove the methodology. More than 50% speed gain is achieved for standard designs like Digital Radio or Cable Modem by using 2-3 processors.

13 Future Work

A few practical issues related to multi-rate designs or if large buffers are used, are discussed in this paper. These issues can be resolved by enhancing the framework. For designs having large buffers communicating only the modified chunk of the data, must improve the gain significantly. The representation of multi-rate designs should be refined so that the constraint to tie many actors together can be slackened.

14 References

- [1] Siavash Alamouti, John Lundell, Ed Casas: "Reducing the Development Cycle for Third Generation Wireless Communications Systems with Parallel Simulations"; Cadence Design Systems
- [2] SPW: Design Automation Technology for Digital Signal Processing Applications; http://www.coware.com/products/spw4_techoverview.php
- [3] SPW MultiProx: SPW User's Guide, Product Version 4.7
- [4] <http://www-unix.mcs.anl.gov/mpi/>
- [5] G. Kahn: "The Semantics of a Simple Language for Parallel Programming," *Proc. of the IFIP Congress 74*, North-Holland Publishing Co., 1974.
- [6] J.L. Pino, S. Ha, E.A. Lee, and J.T. Buck: "Software synthesis for DSP using Ptolemy," *Journal on VLSI Signal Processing*, Vol. 9, No. 1, pp. 7-21, Jan. 1995. http://ptolemy.eecs.berkeley.edu/papers/jvsp_codegen
- [7] P. Zepter and T. Grötker: "Abstract multirate dynamic data -flowgraph specification for high throughput communication link ASICs," *IEEE VLSI DSP Workshop*, The Netherlands, 1993.
- [8] A. Kalavade: "System level codesign of mixed hardware software systems," Tech. Report UCB/ERL 95/88, Ph.D. Dissertation, Dept. of EECS, University of California, Berkeley, CA 94720, Sept. 1995.
- [9] A. Kalavade and E.A. Lee: "A hardware/software codesign methodology for DSP applications," *IEEE Design and Test*, Vol. 10, No. 3, pp. 16-28, Sept. 1993.
- [10] W.-T. Chang, S. Ha and E. A. Lee: "Heterogeneous Simulation - Mixing Discrete-Event Models with Dataflow"; RASSP special issue, *J. of VLSI Signal Processing*, 1996
- [11] <http://www.mcs.anl.gov/mpi/mpich2>