Multiprocessor System Development for High Performance Signal Processing Applications

Eric K. Pauer
Sanders, a Lockheed Martin Company
Signal Processing Center
Nashua, NH 03061-0868
(603) 885-8358, Fax (603) 885-0631
pauer@sanders.com

Abstract

Developing multiprocessor systems to implement high performance signal processing algorithms can be a formidable undertaking. A process for designing multiprocessor systems, using primarily programmable processors, is proposed here. This design process starts with algorithm entry and analysis, continues with functional decomposition and architecture entry, which are used to drive the algorithm to architecture mapping. The development process then proceeds with a combination of hardware profiling and performance modeling, followed by the visualization of the results. Since the process is iterative, feedback from later steps are often used to make better design choices in earlier steps, allowing the impact of these decisions to be re-evaluated. Ultimately, the user finishes with an implemented system. Each of these steps are described in detail, as well as some of the tools which support them.

1. Introduction

The development of a signal processing system typically starts with system requirements, in terms of what functionally the system must implement. These requirements usually include a description of the type of system function to be implemented and the rate at which this processing must be accomplished as well as various environmental and physical restrictions such as weight, volume, power consumption, and reliability. For many applications, the high computational needs can be met with a multiprocessor system. The development of multiprocessor systems is inherently complex, as architectural decisions must be made, resources must be appropriately allocated and utilized, and the flow of data and information managed. As the number of processors increases, this complexity also increases and requires a disciplined development process and set of supporting tools for that process.

*PREPARED THROUGH COLLABORATIVE PARTICIPATION IN THE ADVANCED SENSORS CONSORTIUM SPONSORED BY THE U.S. ARMY RESEARCH LABORATORIES UNDER COOPERATIVE AGREEMENT DAAL01-96-2-0001.

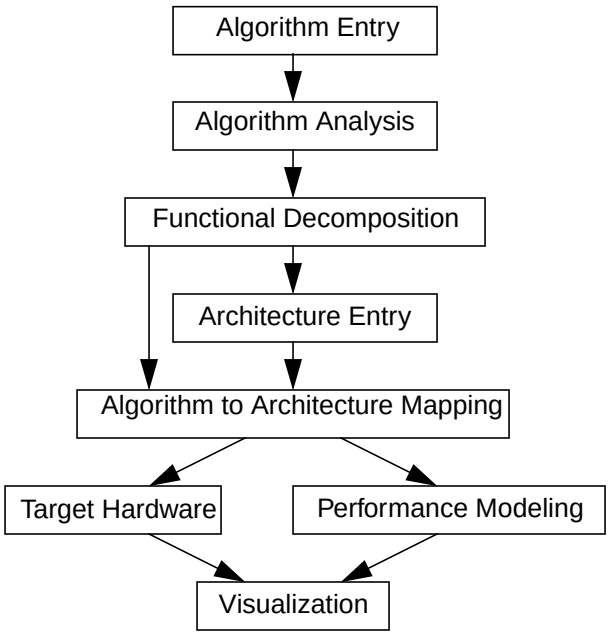

Figure 1: Multiprocessor System Development

The purpose of this document is to describe a process for developing multiprocessor systems that has been put together by various members of the Army Sensors Federated Labs Program. The process for developing these systems is shown in Figure 1. The process begins with system algorithm entry and analysis, which is followed by the functional decomposition of the algorithm. At this point, algorithmic blocks are matched to library elements and hardware architectures are defined. The results of these steps recombine as the user performs the system algorithm mapping, assigning the algorithmic blocks to processors or other hardware resources in the architecture. This mapped architecture is now available for performance simulation or execution on a target hardware system. Various performance metrics are collected during the performance simulation or target execution, which can be viewed in various ways using the visualization tools. Since this process is iterative, feedback from later steps are often used to make better design choices in earlier steps, allowing the impact of these decisions to be re-evaluated. Ultimately, the user fin-

ishes with an implemented system via the target execution. Each of these steps will be described in more detail in the following sections along with the tools that support these steps. Of specific concern are those areas in which tool support is weak. The transition from algorithm analysis and decomposition into mapping as well as the automatic mapping of decomposed algorithms onto architectures are not well supported.

2. System Algorithm Entry and Analysis

The first step in realizing an implementation of a multiprocessor signal processing system is designing and developing an algorithm that functionally satisfies the system requirements. This representation and analysis to verify the algorithm is the primary goal of these first steps in the process. In many cases, an algorithm or portions of the algorithm are available, but need some modification or adaptation for the specific problem at hand. There are several tools, each of which has their strengths. Each tool takes a slightly different approach to the design problem.

2.1. MATLAB

A popular tool for algorithm development, from the point of view of ASFL, is MATLAB* [1], by the Mathworks, Inc. MATLAB provides an easy to use computing environment that combines high performance computation with visualization capabilities. Within MATLAB, the user expresses their algorithms using a natural mathematical syntax, specific to MATLAB, in a command line environment. MATLAB, which is short for MATrix-LABoratory, uses matrices and vectors as its basic data entity for data representation. It is an especially good environment for quickly prototyping and testing algorithms when the complexity of the algorithms are relatively low and can be expressed easily as matrix operations. The integrated visual capability is especially amenable to facilitating algorithm refinement. MATLAB provides many built-in functions which are organized into comprehensive collections called toolboxes. There is also the ability to create custom functional blocks to create your own toolboxes. MATLAB can also call routines written in C or Fortran from its environment, which adds to its flexibility to utilize legacy code. The user must follow the certain conventions to enable this to work.

MATLAB itself does not provide strong support for hierarchial modeling. Besides the built-in matrix and vector data structures, it is not easy to represent data at higher levels or different levels of abstraction. The Mathworks appears to addressing this need and hierarchial modeling will likely be well supported in future releases of MATLAB. Similarly, functions suffer from similar limitations in abstracting different types of computations; however, the toolbox capability does help overcome this limitation to some extent. Since MATLAB is an interpreted environment, it gives quick feedback to the user when used interactively. However, when large simulations are run with MATLAB using scripts, they tend to suffer long simulation times because the interpreted environment slows down the computation. In short, MATLAB is an excellent tool for the development of algorithms as long as the size and scope of the simulations are not significantly impacted by the interpreted environment.

*MATLAB is a registered trademark of The Mathworks, Inc.

2.2. Ptolemy

Another tool useful for the prototyping of signal processing algorithms is Ptolemy [2], a software environment developed at the University of California, Berkeley, that supports heterogeneous system simulation and design using several different models of computation, each implemented in a separate domain. Ptolemy has been developed in C++ using an object-oriented software architecture, and is a freely distributable development environment, including all of its source code. This greatly facilitates extending and customizing Ptolemy to meet various needs. In the synchronous dataflow (SDF) domain of Ptolemy, algorithms, represented using data flow semantics, are comprised of functional blocks, also called *stars*. The SDF domain handles a class of algorithms in which the schedule, or order of execution of the stars, is deterministic and can be predicted at compile time. The Ptolemy distribution provides a rich library of SDF stars for algorithm development, and it is straightforward to create new stars. The Boolean dataflow (BDF) and dynamic dataflow (DDF) domains relax the determinancy requirements on the schedule, and can be useful in modeling dynamic systems. These domains also have many built-in stars for algorithm development.

Ptolemy has both a graphical interface and a command line interface. Algorithms can be expressed graphically using the extensible block diagram environment, and then simulated. There are many built-in display stars that enable the results of the simulation to be shown in any of a variety of graphical or textual formats. Ptolemy directly supports hierarchial modeling, as groups of connected stars can be combined into a galaxy, capturing the appropriate computations into a single entity. The hierarchial modeling also supports heterogeneous modeling, which allows different domains (models of computation) to be part of the same simulation model.

Currently, Ptolemy tends to require more set up to establish a simulation than MATLAB, especially for smaller simulations. However, members of the Ptolemy project are actively working on upgrading Ptolemy's graphical interface, making this less of an issue. In addition, Ptolemy's strong support for hierarchial modeling and the fact that the its simulation is performed by compiling and then executing greatly enhances its efficiency and usefulness as the problem size increases in size and complexity.

2.3. Khoros

Khoros [3], a tool by Khoral Research, Inc., is geared for the development of applications that involve image processing. It is especially strong in information processing, data exploration, and data visualization. As a visual programming and simulation environment, Khoros enables algorithms consisting of functions, or operators, to be graphically expressed using the data flow visual language to rapidly prototype new solutions. A command line capability is also available. Khoros, like MATLAB, includes toolboxes consisting of a library of closely related functions, which may be used in building and simulating applications, and there is a capability to define and store new operators as well.

2.4. Other Tools

There are a number of other commercial tools available that can be used for algorithm development, including products from the Alta Group (Signal Processing Workstation (SPW)*) [4], DSP Development Corporation (DADiSP*) [5], Hyperception (Hypersignal*) [6], Mathsoft (Mathcad*) [7], Mentor Graphics Corporation (DSP Station* and DSP Architect*) [8], Axiom (DataFlo*) [9], Orincon (RIPPEN*) [10], Synopsys (COSSAP*) [11], and Momentum Data Systems (DSPworks*) [12] to name a few. A comprehensive coverage of the spectrum of tools has for these steps have been compiled by Berkeley Design Technologies [13].

3. System Functional Decomposition

The system functional decomposition is accomplished by many of the same tools as the system algorithm analysis described in section 2. The goal of this step is to partition the algorithm into fundamental functional blocks, which can then be assigned to processors. The level of granularity for these blocks must be fine enough so that a single processor or computational element can meet the real-time throughput needs. The granularity can be such that multiple blocks are assigned to a processor or hardware element. However, the granularity of the blocks must be coarse enough so that the overhead in passing data between processors or computational elements does not become objectionable. The decomposition of the algorithm may be accomplished by pipelining the algorithm by breaking a functional block into a series chain of more fundamental blocks. The decomposition may also be accomplished by parallelizing the computations, often by having each processor in a group of N processors handle every Nth block of data. Commonly, a combination of pipelining and parallelization in implementing the algorithm is effective in meeting the throughput requirements of the system. A variety of combinations and topologies may need to be investigated using subsequent steps in the development process. In both cases, it is important to keep in mind that the overall latency requirements of the system in addition to the throughput requirements.

To facilitate implementation of the system later in the process, it is desirable to partition or decompose the algorithm into blocks that will match the available resources. In other words, it is important for the functional blocks from this step to have implementation counterparts in a vendor or third party software library for the target processors, or FPGA/ASIC designs that implement the functional block in hardware. This correspondence will probably not always be possible, but adherence to this general rule will greatly reduce the risk and schedule of prototyping the system. Most of the design tools discussed in section 2 facilitate this correspondence by imposing restrictions on how the algorithms are created or specified. In addition, performance models of the existing blocks are more likely to exist by following this guideline, making this process much easier and more accurate.

To facilitate steps later in the process, it useful to be able to capture the results of the functional simulation at various points in the algorithm. The input data, along with intermediate and final results, are necessary for verifying that the functionality of the algorithm is preserved when implementing the algorithm on target system hardware and/or software. A variety of data sets, exercising the full range of the algorithms computations, are a prudent measure to insure that the functionality is not significantly perturbed in the translation. Of course, there are always problems that emerge when implementing an algorithm, so the intermediate results are essential in tracking down where the implementation has broken down. Ideally, these results are stored in a central database that can be easily retrieved by a variety of tools at various steps in the process.

With some algorithms, this decomposition step has already been satisfactorily accomplished with the system algorithm analysis. In most cases, the system functional decomposition is best arrived at in an iterative fashion after the algorithm mapping and performance simulation steps in the process have been exercised.

4. Architecture Entry

The system functional decomposition has laid the groundwork for this next step in the development process, where the prototype architectures are defined. An architecture editor is used to define prototype architectures, typically at a high level of abstraction. The architectures consist of entities such as processors, memories, busses, interconnections, data sources, and programmable hardware. In the ideal situation, these architectural entities exist in a component hardware library or database. If they do not exist, new models must be created as necessary to represent its architecture, resources, and connectivity options.

*The name in parentheses is the trademark of the listed company.

Unfortunately, moving from the algorithm analysis and decomposition steps into the architecture entry and mapping is not a well supported translation from a tools perspective. This is the case because the algorithm analysis is typically a functional simulation while the architecture entry and mapping is a performance simulation, representing two different paradigms of modeling. The tools that do work with algorithms, architectures, and mappings are typically involved with performance modeling, and specific tools are discussed in the next section.

If the system functional decomposition is done properly, the functional blocks in the algorithm will, for the most part, match up with components of the signal processing library. This library can contain either software components or hardware components or both. The software routines are coded and optimized in assembly for one or more target processors, or may simply be well written routines in C. Software library routines that follow a standard application programming interface (API) for standard interprocessor communications may provide a good framework for implementation later, which can be a big advantage. Examples of the low overhead communication standards include Message Passing Interface (MPI) [14], Parallel Virtual Machine (PVM) [15], Active Messages [16], and Fast Messages [17]. The hardware components could include hardware designs for FPGAs or ASICs, implementing signal processing functions. The hardware implementations can take on a variety of forms and may be detailed designs, structural or behavioral VHDL, high-level performance models, or a combination of the above. Whether they are implemented in software or hardware, if functional blocks in the algorithm do not exist in the available libraries, an appropriate performance model must be created. Section 5.6 describes different ways of accomplishing this. While a detailed implementation will eventually be needed in subsequent iterations of the development process for profiling on target hardware, it is not necessary at this point. Mapping of the functional blocks onto the architectural resources will now take place, and this is described in more detail in section 5.

At this point, after defining the architectures and performing the mapping, the tools used may emphasize either performance modeling via simulation or preparing target hardware for performance analysis. During the early stages of the design, the next step is typically performance simulation, followed in later iterations by hardware profiling. In the ideal situation, the same tools or toolsets could be used as a front end for both purposes. The advantages of this type of approach are that the architectural trade-off decisions from performance simulations, including design information, can be easily shared between simulations and target implementations. It also provides a convenient mechanism for validating performance simulations against actual hardware performance measurements. The performance simulation helps in estimating answers to questions such as is there enough compute power and enough connectivity bandwidth to support the application. Lastly, it makes the process easier for the designer as they have to become familiar with only one set of tools versus two sets.

5. Mapping and Performance Modeling

The bottom line is that performance modeling information is needed for both the architecutural components as well as the algorithmic blocks in the functional decomposition to do system performance modeling. The architectural and algorithmic performance modeling information is needed for the system performance simulation while the implementation (or design) of the components are needed for profiling on the target hardware. Performance models can usually be created from detailed designs in a straightforward fashion, depending on the performance simulation tool and the level of abstraction used. The performance of software components (routines) can be obtained from the vendor that created them or by profiling them on an instruction set simulator (ISS) or even an evaluation board for the processor, depending on availability. Detailed hardware designs can be simulated, and performance numbers can be obtained. The need for performance information up front works out well since the system performance modeling is usually done before the target hardware profiling, and is in general easier to obtain than the detailed implementations.

With the algorithm and architectures appropriately modeled, the user now assigns the execution (in the case of software) or implementation (in the case of hardware) to the various entities in the architecture. For example, one or more algorithmic blocks may be mapped to a processor or each may be implemented in an FPGA. The execution time of the blocks on the architectural entity as well as memory and other processor resource usage can then be assigned. The flow of data from one entity to another, over a bus or interconnect, must also be assigned either automatically or explicitly. Different tools approach this modeling task differently.

Performance modeling can be done at different levels of abstraction. At a very high level of abstraction for a fairly simple system, a spreadsheet can be effective in determining loading and connectivity. Results can be obtained quickly and many alternatives can be examined. However, as the complexity of the problem increases, the accuracy of this approach decreases as the user must make increasingly larger assumptions about performance behavior. At the other end of the spectrum, performance modeling can be done a very fine level of abstraction where the hardware (and often software) can be described in VHDL [18] or Verilog [19]. The advantage here is that the accuracy of the models is very high because the models are (or are close to being) virtual prototypes of the system. The disadvantages are that it usually takes significant effort to set up these models, and the simulation times can be long when an entire system is modeled. However, for limited scope simulations, this type of modeling can be very effective.

An intermediate level of abstraction may be a good compromise between the high level spreadsheet approach and low level VHDL/Verilog modeling approach. This level of modeling uses a discrete event simulation approach. It offers reasonable accuracy with less set up and simulation time than for the low level modeling. The discrete event modeling may take advantage of the results of low level performance modeling for individual components and subsystems, and the preliminary results from a spreadsheet approach may serve as a starting point for the exploration of options. Three tools that use discrete event modeling (Ptolemy, RAMP, and NetSyn*) are described in the following sections. A fourth tool, PMW**, is based on VHDL modeling and is also described.

5.1. Ptolemy

Performance modeling work done at Sanders as part of the Rapid Prototyping of Application Specific Signal Processors (RASSP) program provides the capability to define architectures at a high level of abstraction and map blocks from a Ptolemy SDF algorithm onto the architecture [20]. The tool's graphical user interface consists of two windows, one for the algorithm and one for the architecture. The user then maps the execution of the algorithmic blocks onto the architectural entities. This tool then uses the algorithm, architecture, and mapping information to create a Ptolemy Discrete Event (DE) domain model, which is passed to the Ptolemy kernel and simulated. The DE domain uses a model of computation in which tokens with time stamps, called *particles*, representing events are among the architectural blocks, called *stars*. Extensions to the DE domain, in the form of new stars and particles, have been created so that performance models can be defined and simulated. The execution time of algorithmic blocks are represented by cost functions, which have an overhead component and a component proportional to the data size.

Another performance modeling effort at Sanders, under the High Performance Scalable Computing (HPSC) program, uses Ptolemy in a similar manner [21]. The focus of this effort was modeling the networking and connectivity aspects of the architecture. The HPSC architecture, which is implemented with the MYRINET* protocol [22], consists of nodes containing one or more processors interconnected via a network of MYRINET multi-port switches. Additional extensions to the Ptolemy DE domain were created to model the MYRINET protocol. As with the capability developed on RASSP, the Ptolemy kernel is at the heart of the simulation. Data and control packets flow between the nodes over the network of switches using routing information contained in each node's network interface, called a LANai. This tool, which uses the Ptolemy graphical user interface, allows the user to define the loads on each of the nodes to simulate the execution of algorithmic blocks, set up an architecture of nodes and switches, and simulate the performance. The primary goal of this capability to help address questions of network topology and routing in terms of how many switches, what type of switches, how should they connected, and what is the best routing to most efficiently utilize the network. Thus, it does not provide as much fidelity in modeling the execution of the algorithms on the architecture as other tools, but it does answer the complex questions of data flow within the architecture.

5.2. RAMP

A tool developed by a team at General Electric Corporate Research and Development, under the funding of Lockheed Martin, has developed a tool called Real-time Algorithm Mapper and Performance-analyzer, also known as RAMP [23]. RAMP is a graphical tool for designing multi-processor based systems, which helps in evaluating the suitability of architectures for implementing algorithms. It has a graphical interface consisting of an algorithm window and an architecture window. Much like the RASSP architectural trade capability, the user maps the algorithmic blocks onto the architecture. It provides an automatic routing capability, using a shortest route assignment, for the initial flow of data on the architecture. It provides the capability to import algorithm topologies exported from Alta's Signal Processing Workstation (SPW). One disadvantage of RAMP is that it does not easily allow the addition of models or modeling at higher or lower levels of abstraction. The cost functions are also a little constrained as they are not proportioned to the amount of data being processed. A built-in discrete event simulator is integrated into the tool and provides the simulation capability for the mapped architecture.

RAMP has been used as a front end for the HPSC modeling work. Although RAMP cannot model the MYRINET protocol at the desired level of abstraction, it can be used to determine the initial routing information needed to implement a particular mapping on an architecture. RAMP exports this information, which can then be imported into the HPSC modeling capability and simulated using the Ptolemy MYRINET models.

5.3. NetSyn

JRS Research Laboratories has developed their solution to this problem with their Network Synthesis System, called NetSyn [24]. NetSyn has similar capabilities as the other tools, but one significant difference that it bases its algorithm development on the Processing Graph Methodology (PGM) [25]. All algorithms are expressed and implemented in PGM, which is a standard means of expressing portable, dynamic dataflow applications. The PGM graph, as it is called, representing the algorithm is then mapped onto the target architecture, which NetSyn can do automatically (parallel scheduling) or allow the user to do manually. A standard library of PGM functions are available for express-

*NetSyn is a trademark of JRS Research Laboratories, Inc.

**PMW is a trademark of Omniview, Inc.

ing the algorithms, and an architecture library of processors and interconnects is also provided. NetSyn then performs a performance simulation using its own simulation engine. There is an integrated capability to view the results of the simulation using various forms of a Gantt chart.

5.4. PMW

Omniview has developed a product called the Performance Modeling Workbench, or PMW [26]. Although currently in the alpha release, it has been used to do some performance modeling on RASSP. PMW allows the user to create performance models at a varying degrees of fidelity, in both hardware and software. The basis for modeling is VHDL, which enables PMW, in general, to have a finer level of granularity in its models compared to Ptolemy, NetSyn, or RAMP. There are a number of tunable parameters which can be used to adjust the granularity of the model at the expense of longer simulation times. The user defines the software and hardware with block diagrams using their graphical interface, with parameters which model the processing/communication requirements and capabilities. There is no true mapping of an algorithm onto the architecture, but there is a mapping of software onto the hardware. Thus, if the algorithm is implemented completely in software, then the algorithm to architecture mapping is essentially done as with the other tools. Implementation of algorithmic blocks in hardware requires a little more work, but is fully supported by the modeling environment. PMW generates a VHDL model of the mapped architecture, which is then simulated on any VHDL simulator. Results are captured as the VHDL simulation is performed, and PMW provides the ability to view the results in a variety of forms and formats. A rich modeling library is included with the tools, which is based on the work done by Honeywell in developing their Performance Modeling Library. A limitation of PMW is that it is not easy to add additional models, only to change the parameters of the existing models. PMW also requires knowledge on how to use VHDL.

5.5. Other tools

The Alta Group has a tool called BONeS* [4], which is a discrete event simulator. Other performance modeling tools that use slightly different paradigms are Statemate* [27] by i-Logix and OPNET* [28] by Mil 3, Inc. Each of these tools emphasizes performance modeling of the interconnects between the processor and processing nodes.

5.6. Instruction Set Simulators

Creating accurate performance models of new algorithmic blocks can accomplished by several different means. Algorithmic functions implemented in vendor or third party libraries for target processors often provide performance metrics for their routines, and these can be used in creating the performance models. It may also be possible to assess the performance of commercial or custom routines by profiling them on an instruction set simulator (ISS) or on a target board. An ISS is often available ahead of the new processor that it simulates, and give a convenient means of profiling individual routines. By their nature, an ISS can become cumbersome and slow if the size of the simulation is allowed to grow too large. ISSs are becoming more and more sophisticated, and often can provide additional insight into the utilization of a processor's CPU and supporting devices like DMA channels, caches, registers, and internal memory. One excellent example of this is the Texas Instruments TMS320C80 simulator [29]. Target boards are also a good choice for profiling software routines, but are often less available and require more setup to use. Nonetheless, they provide a reliable and convenient method of profiling code. In the absence of an ISS and a target board, some profiling can be done on a host, such as a workstation or PC with a debugger, especially when the software is implemented in C or other high level language. While not ideal, this can provide some good initial estimates of the computational needs of the algorithmic blocks.

A tool to help in the process of finding acceptable mappings would be very useful. Often called multiprocessor schedulers, these tools would help in determining a reasonable load balance. As mentioned previously, NetSyn has such a capability. Parallel schedulers are usually coupled with code generation capabilities, which are discussed in section 6.1.

6. Target Hardware Configuration

This step involves configuring target hardware to implement a particular algorithm. This problem is difficult to solve in general, and is an area of relative weakness in the development process. The goal of the performance analysis is to make this step easier by providing candidate mappings and architectural configurations that show great promise for meeting the system needs. This step enables these options to be evaluated more carefully by profiling on hardware. As the design progresses, larger and large portions of the system are prototyped on the target hardware. The prototyping continues until the entire system has been implemented.

Several tools solve this problem for specific target hardware architectures with varying degrees of success. Some of examples of this capability are Rippen** [10] by Orincon Technologies, DataFlo MP** [9] by Axiom, and Ptolemy by the University of California, Berkeley as well as tools from Hyperception, Visual Solutions [30], and Real Time Signal Processing [31].

The goal of these tools to enable the rapid configuration and prototyping of algorithms executing on an architecture, capturing performance information. The performance information allows the configuration to be evaluated and determine where performance problems exist. The suite of tools for

*The name is the registered trademark of the listed company.

visualizing the performance information is discussed in the next section.

6.1. Code Generation

A class of tools supporting target hardware configuration involves those for code generation. Code generation can be implemented in a variety of ways. Most commonly, the user creates an algorithm from a built-in library of functional blocks, creates an architecture, and maps the execution of the algorithm onto various architectural entities, as discussed in section 5. The role of the code generator is to create the "glue code" needed to handle the passing of data (transmission and reception) between algorithmic blocks, both on the same processor or between processors, as needed. The "glue code" also includes any set up or initialization required for the processor and it also takes care of necessary resource allocation and insures that the application software routines (library functions) are called properly and in the right order. The distinction between code generation and target hardware configuration is not well defined, but in general, the later encompasses the first. Some code generation schemes are general purpose and create C code for the application software routines as well as for the glue code. Others generate assembly code for the target processor. Code generation is very powerful and efficient in situations where it takes advantage of optimized application libraries and glue code. Adherence to commercial or standardized APIs, as mentioned in section 4, is necessary for code generation, but is good design practice anyway. These APIs provide a standard set of routines needed to implement various resource allocations and pass data, which is especially important in a heterogeneous multiprocessor system.

Some code generation capabilities include the means to automatically suggest a reasonable mapping of algorithmic blocks onto the architecture. This automated mapping capability is called a multiprocessor scheduler or a parallel scheduler. The goal of these tools is to search many possibilities for mapping the algorithm onto the architecture, in order to find one that is optimal in terms of speed, size, and/or resource consumption. NetSyn has such a capability for the processors and algorithmic blocks it supports. Ptolemy has several code generation domains which have parallelizing schedulers which can find optimal schedules for certain classes of situations. Its suite of code generation domains can be used to generate C code or assembly code for several common DSPs, namely the Motorola 56K and 96K families and Texas Instruments TMS320C50.

6.2. Multiprocessor Debuggers

Another important group of tools for hardware profiling are multiprocessor debuggers. In a multiprocessor system, it is important to be able to have control over its execution, whether it is a single processor, a cluster of closely coupled processors, or a distributed system. Emulators are useful for controlling the execution of processors, so that the processors can be commanded to run, single step, stop at breakpoints, and view the status of system registers and memory. With the advent of JTAG emulators that use serial scan techniques, this control and observability can be extended to clusters of processors. Data at different points in the algorithm, stored in the decomposition step, can be used here to compare results and determine where implementation problems exist. This technology is also useful in controlling the individual processors on a multichip module or on a complex processor like the TMS320C80 (five processors) or new quad-SHARC AD14060 MCM [32] from Analog Devices, single chips with multiple processors. In order to debug a multiprocessor system, you need control over the processors so that each may be observed and controlled independently so that interprocessor communication can be debugged. Examples of this are White Mountain DSP's emulator JTAG emulator products [33] and accompanying debuggers for the TMS320 family of processors and the Analog Devices SHARC. At the system level, a need to control large numbers of distributed processors is also essential for similar reasons, where JTAG becomes too slow to handle the volume of data. BBN is developing a product called Total View* [34] which will allow distributed clusters of multiprocessors to be debugged in a fairly non-intrusive fashion over the MYRINET.

7. Visualization of Performance and Goals of Feedback

Many of the tools already described have an integrated capability to display the results of the performance simulations or the results of target hardware trials. A flexible capability to view the results in various ways is quite valuable. However, the interpretation and visualization of the results depends on the type and size of system being modeled.

Depending on the specific goals of the performance simulations, it may be useful to be able to examine the behavior exactly as it is modeled by the simulation. At this level, the user sees great detail about what is going on at the level of abstraction implemented by the models. This level of abstraction is useful for synchronous systems where the behavior of the system is periodic over some finite interval. Here, optimizing performance over one interval automatically provides the same performance over all time. Thus, by definition, simulation over one interval (or a small number of intervals for pipelined cases) is sufficient to characterize the system and its performance. A Gantt chart, showing activity or utilization as a function of time, provides enough visualization for most needs. This type of capability is available with the architecture trade and MYRINET modeling capabilities developed at Sanders.

---

*Total View is a trademark of BBN, Inc.

In many cases, the user may want to limit their view to a particular subsystem or component in the system, or only view "problem areas" that satisfy some criterion. The ability to "filter" the results to a limited scope can be very useful, especially as the system increases in size. A hierarchial approach to system optimization may be effective in some applications, where subsystems optimized separately in the system and a filtering capability on visualization can be quite useful. Examples of this type of capability are a Gantt chart that shows only selected activities, or a Gantt chart that displays only those activities where contention or wait time exceed a threshold.

As the system exhibits more and more dynamic behavior, the system no longer satisfies the synchronous model, and the repeat interval is no longer finite. Data reduction of the collected performance metrics may be more effective than viewing specific instances of behavior. Statistics on the performance, such as average utilization or peak usage, may be more useful. A distribution function or histogram, along with mean and variances of various metrics may provide the insight or criterion to determine whether the performance is acceptable or not. Statistical interpretation of performance metrics are more useful as the system model is exercised over longer periods of time, as is done with Monte Carlo approaches.

8. Summary

This paper has provided a narrative description of a process for designing multiprocessor systems for signal processing applications. Each step has been described, and a few tools which help with each step have been introduced. Other tools have been mentioned, but were not investigated in depth. Additional details are available from the vendors' web sites. There are clearly some areas of strength and weakness in terms of tools support for this process, and these have been summarized.

REFERENCES

1. Mathworks, 24 Prime Park Way, Natick, MA 01760 (http://www.mathworks.com)
2. E. A. Lee, et al., University of California, Berkeley, The Almagest, Volumes 1-4, Regents of the University of California, 1996. (http://ptolemy.eecs.berkeley.edu)
3. Khoral Research, Inc., 6001 Indian School Road, NE, Suite 200, Albuquerque, NM 87110 (http://www.khoral.com)
4. Alta Group, 555 N. Mathilda Avenue, Sunnyvale, CA 94086 (http://www.altagroup.com)
5. DSP Development Corporation, One Kendall Square, Cambridge, MA 02139 (http://www.dadisp.com)
6. Hyperception, 9550 Skillman LB125, Dallas, TX 75243 (http://www.hyperception.com)
7. Mathsoft, Inc., 101 Main Street, Cambridge, MA 02142 (http://www.mathsoft.com)
8. Mentor Graphics Corporation, 8005 SW Boeckman Road, Wilsonville, Oregon 97070 (http://www.mentorg.com)
9. Axiom Technology, Inc., 115 Murry Drive, Madison, AL 35758 (http://www.axiom.org)
10. Orincon Technologies, Inc., 9363 Towne Centre Drive, San Diego, CA 92121 (http://www.orincon.com)
11. Synopsys, Inc., 700 East Middlefield Road, Mountain View, CA 94043 (http://www.synopsys.com)
12. Momentum Data Systems, Inc., 1520 Nutmeg Place #108, Costa Mesa, CA 92626 (http://www.mds.com)
13. Berkeley Design Technology, Inc., DSP Design Tools and Methodologies, Volumes I and II, Fremont, California, 1995. (http://www.bdti.com)
14. Message Passing Interface Standard (http://www.mcs.anl.gov/mpi/index.html)
15. Parallel Virtual Machine (http://www.epm.ornl.gov/pvm)
16. Active Message Interface, University of California, Berkeley (http://now.cs.berkeley.edu/AM/active_messages.html)
17. MPI Fast Messages, University of Illinois at Urbana-Champaign (http://www-csag.cs.uiuc.edu/projects/comm/mpi-fm.html)
18. "IEEE Standard VHDL Language Reference Manual," IEEE/ANSI Standard 1076-1993, 1994.
19. "IEEE Standard Description Language Based on the Verilog(TM) Hardware Description Language," IEEE Standard 1364-1995, 1995.
20. E. K. Pauer and J. B. Prime, "An Architectural Trade Capability Using the Ptolemy Kernel," IEEE International Conference on Acoustics, Speech, and Signal Processing, 1996. (http://www.sanders.com/spard/publish.html)
21. E. K. Pauer, "High Performance Scalable Computing Performance Modeling Using Ptolemy", IASTED International Conference on Modelling and Simulation, 1997. (http://www.sanders.com/spard/publish.html)
22. Myricom, Inc., "Myrinet Link Specification," Arcadia, California, 1995. (http://www.myri.com)
23. RAMP, General Electric Corporate Research and Development, P.O. Box 8, Schenectady, NY 12301 (http://www.crd.ge.com, moitrad@crd.ge.com)
24. JRS Research Laboratories, 1036 West Taft Avenue, Orange, California, 92665
25. Processing Graph Method Standard (http://www.ait.nrl.navy.mil/pgmt/pgm2.html)
26. Omniview, Inc., 100 High Tower Blvd., Suite 201, Pittsburgh, PA 15205
27. i-Logix, Inc., 3 Riverside Drive, Andover, MA 01810 (http://www.ilogix.com)
28. MIL 3, Inc., 3400 International Drive, NW, Washington, DC 20008 (http://www.mil3.com)
29. Texas Instrument TMS320C80 MVP Simulator (http://www.ti.com/sc/docs/dsps/tools/c8x/sim.html)
30. Visual Solutions, Inc., 487 Groton Road, Westford, MA 01886 (http://www.vissim.com)
31. Real Time Signal Processing, Inc., 1715 - 27 Avenue N.E. #1, Calgary, Alberta, T2E 7E1, Canada
32. Analog Devices, 7910 Triad Center Drive, Greensboro, NC 27409 (http://www.analog.com)
33. White Mountain DSP, Inc., 410 Amherst Street, Suite 325, Nashua, NH 03063 (http://www.mwmedia.com/tpvs/whitemtn/dsp/mtn-510.htm)
34. BBN Corporation, 150 Cambridge Park Drive, Cambridge, MA 02140 (http://www.bbn.com:80/tv/TINDEX.html)