AN ARCHITECTURAL TRADE CAPABILITY USING THE PTOLEMY KERNEL

Eric K. Pauer, Jonathan B. Prime
Sanders, a Lockheed Martin Company
Signal Processing Center of Technology
Nashua, NH 03061-0868
epauer@sanders.com, prime@sanders.com

ABSTRACT

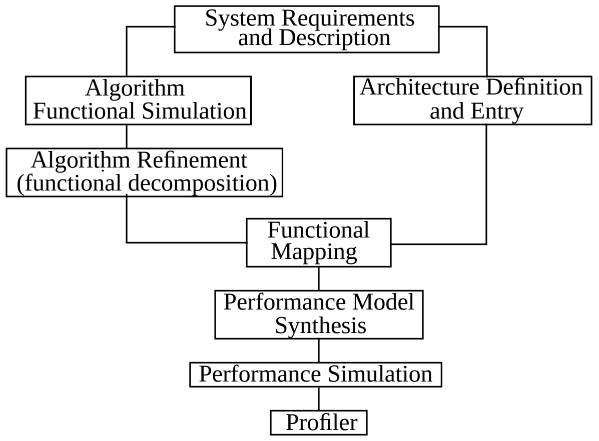
A perennial problem in the process of developing signal processing systems is identifying an architecture which meets the computational and memory needs of the algorithm yet is still affordable in terms of cost, size, and complexity. To aid in making this decision, an architecture trade tool, using the Ptolemy kernel, has been developed. The purpose of this tool is to provide the user with an easy mechanism of specifying a mapping of algorithmic functional blocks onto an architecture, and simulating its performance. The goal is to allow the user to quickly evaluate many architectures and mappings at a high-level to determine whether or not they satisfy basic criteria, and warrant further investigation. These criteria include computational and memory requirements as well as other system metrics (e.g. size, power, and reliability, etc.) which are often important in selecting an architecture.

1. INTRODUCTION

In a typical system design (figure 1), an algorithm is developed and simulated to verify that its functionality satisfies the system requirements. From this functional simulation of the algorithm, a functional decomposition is performed by refining the algorithm into more fundamental functional blocks, as necessary. Once the functionality has been successfully simulated and decomposed, a performance simulation of the refined algorithm running on one or more candidate architectures is needed. The user defines a functional mapping of the refined algorithm onto an architecture, creates and executes a performance model, and then interprets the results. This process is repeated until the simulation yields results which meet the system requirements.

In order to facilitate this process, an architectural trade tool was developed. This tool provides a capability to define an architecture, assign a functional mapping, synthesize and execute a performance model, and provide feedback on the results. The architectural trade tool starts with a functional representation of the algorithm using the Synchronous Dataflow (SDF) domain of Ptolemy[1]. Ptolemy is a software environment developed at the University of California at Berkeley that supports heterogeneous system simulation and design using a several different models of computation, each implemented in a separate domain. In the SDF domain, algorithms are represented using data flow semantics 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 determined at compile time. The Ptolemy distribution provides a rich library of SDF stars for algorithm development, and it is straightforward to create new stars.

This work was performed by Sanders, a Lockheed Martin Company, as part of the Sanders RASSP program under contract N00014-93-C-2172 to the Naval Research Laboratory, 4555 Overlook Avenue, SW, Washington, DC 20375-5326. The Sponsoring Agency is: Advanced Research Projects Agency, Electronic System Technology Office, 3701 North Fairfax Drive, Arlington, VA 22203-1714. The Sanders RASSP team consists of Sanders, Motorola, Hughes, and ISX.


System Requirements and Description
Algorithm Functional Simulation
Architecture Definition and Entry
Algorithm Refinement (functional decomposition)
Functional Mapping
Performance Model Synthesis
Performance Simulation
Profiler
Figure 1: Steps in a typical system design

2. ARCHITECTURE MODELING

Once the functionality of the algorithm has been successfully verified and a SDF domain representation is available, the process of investigating different architectures and mappings can begin. Because the class of algorithms has been limited to the SDF domain, it is possible to completely separate the functional simulation from the performance simulation because the schedule does not depend on the data. This assumption has the advantage of simplifying the complexity of the simulation and decreasing the time to complete a performance simulation. The architectural trade tool uses the Ptolemy Discrete Event (DE) domain as its engine

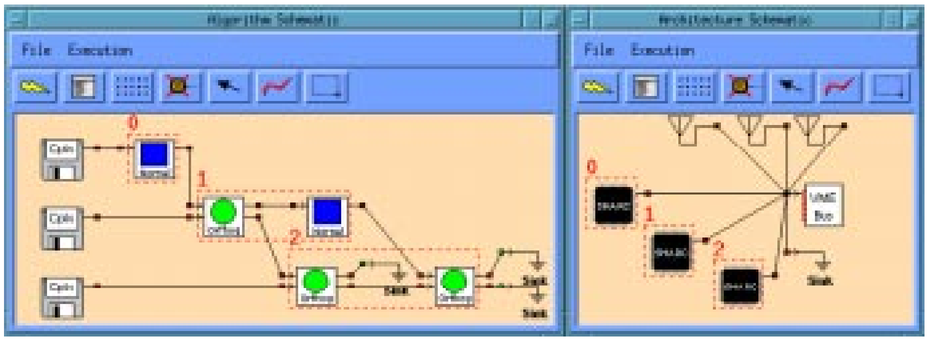


Figure 2: Assignment of functional mapping onto an architecture

for performance modeling. The DE domain uses a model of computation in which tokens with time stamps, called *particles*, representing events are among the 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.

Generic performance level models of various architectural entities have been implemented as DE stars: processors, busses, data sources, and data sinks. These stars are parameterizable and serve as the basis for models of specific hardware devices. The *Bus* star is characterized by the bus bandwidth and the number of simultaneous users that are allowed. It models bus contention, and currently implements a first-come first-serve approach to bus usage, without preemption. The *Processor* star is characterized by a clock rate, an input transfer rate, an output transfer rate, and a memory size. This star simulates the software running on the processor at a performance level and models the passing of data into and out of the processor. The *Source* star simulates the availability of data at a specified rate and block size, and is the source of the data in the simulation. This star can simulate the generation of any type of data (floating point, fixed point, complex, etc.). The *Sink* star acts as the destination for processed data, and can represent an interface to another system or possibly a user display.

As previously mentioned, these stars serve as generic templates which can be used to create models of specific hardware. For example, a model of digital signal processors, like the i860 and SHARC, have been created from the Processor star, and standard busses and interconnects, like the VME bus and Raceway, have been created from the Bus star. Additional processor, bus/interconnect, data source, and data sink hardware models can be easily created by instantiating these templates and filling in the necessary parameters.

3. SPECIFYING AN ARCHITECTURE AND A MAPPING

Using these architectural entities, the user can define an architecture and then specify a functional mapping onto the architecture. In order to provide an easy means of defining architectures and specifying mappings, a custom graphical user interface was developed as part of the architectural trade tool. As shown in figure 2, the left window depicts the SDF dataflow representation of the algorithm in a similar fashion as the Ptolemy interactive graphical interface. In this case, an implementation of the Gram-Schmidt orthogonalization process has been used. The right window shows the architecture consisting of processors, busses, data sources, and data sinks. The left window allows the user to select one or more SDF stars, and the right window lets the user assign the execution of the selected SDF stars to a processor. Once the user has assigned all SDF stars to the processors in the architecture, the mapping is complete.

4. PERFORMANCE MODEL SYNTHESIS AND SIMULATION

A performance-level simulation of the architecture with the specified functional mapping can now be performed. The architectural trade tool uses the architecture model, mapping, and the SDF representation of the algorithm to synthesize and simulate a performance model in the DE domain. This performance model includes Bus, Processor, Sink, and Source stars as well as one or more DE *Function* stars. One of these Function stars is created for each functional star in the SDF representation of the algorithm, and is mapped to its assigned Processor star accordingly.

The Function star is essentially a performance model of the SDF star that it represents, having the same number of input and outputs as its SDF counterpart. The Function star models the computational load (or execution time) of the SDF star, its memory usage, and data block sizes for inputs and outputs. Cost functions are used to represent the execution time and memory usage. These processing and memory cost functions consist of two terms: one which accounts for overhead and the other which is related to the amount data processed per execution of the block. The cost functions are expressed using arithmetic operations and transcendental functions, and are obtained by profiling the code on the host or target processor, using a specified value for library implementation of the function, estimation, or by other means. The accuracy of the performance simulation depends directly on the accuracy of the cost functions, and in general, obtaining these cost functions can be a difficult problem. The cost functions for an SDF star are typically

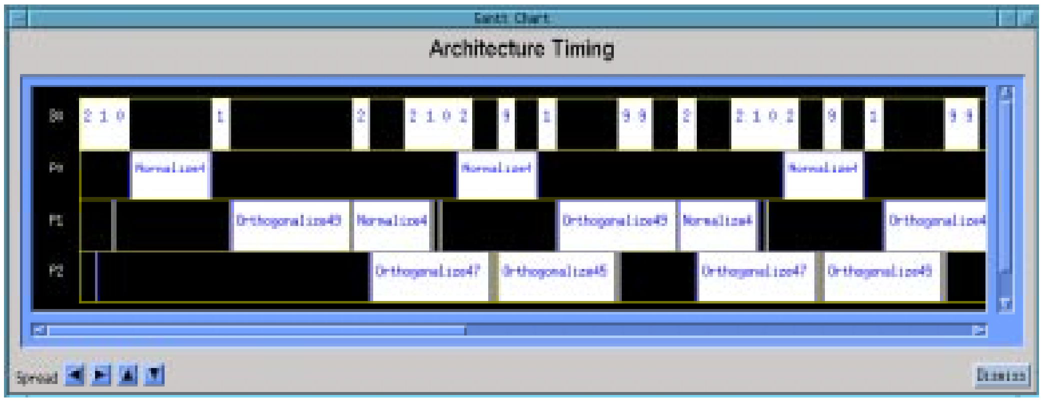


Figure 3: Gantt chart showing processor and bus utilization over time

unique for each different type of processor. If processor-specific cost functions are not available, default cost functions are used. Thus, when creating new Processor stars, more accurate simulations will result if processor-specific cost functions are provided for the SDF stars of interest.

In addition to the new DE stars, this performance simulation capability required the development of two new particles. A *DataBlock* particle represents a block of data in the simulation and is passed among Function, Processor, Source, and Sink stars. This particle contains information concerning the amount and type of data it represents as well as information on the destination of the data. A *ResourceBlock* particle is used by the Function stars to request a specified amount of processing cycles and memory from their Processor star. The DataBlock and ResouceBlock particles are essential for modeling data flow as well as resource contention.

The Function star executes once it has received all of the data that it needs for each of its inputs. Using the performance model cost functions, a Function star creates a ResourceBlock particle requesting a specific number of processing cycles and amount of memory to execute the SDF function that it represents. This ResourceBlock particle given a time stamp with the current simulation time and is passed to the corresponding Processor star. The Processor star, using a first-come first-serve approach, determines the earliest time at which the resources are available, and then uses this time to calculate when the requested execution will be completed. The Processor star then sends a ResourceBlock particle back to the Function star with a time stamp indicating the completion time of the execution. Once the Function star receives this ResourceBlock particle, it creates an appropriate DataBlock particle for each of its outputs using the same time stamp. The Function star also annotates each DataBlock particle with the ultimate destination of the data (another Function or Sink star), and sends it along to its Processor star.

Based on the mapping specified by the user, the Processor star has knowledge of which Function stars have been mapped to it. Whenever a DataBlock particle is received from one of its Function stars, the Processor star examines the particle to see if it needs to be sent to another Function star on the same processor or if it needs to be sent onward to another Processor or Sink star. If the DataBlock particle is sent outside a processor to another Processor or Sink star, it gets there via one or more Bus stars. A Bus star uses a simple model--it simply rebroadcasts any DataBlock particle it receives to all stars that connected to it at the earliest available time. Source stars discard all received DataBlock particles, Sink stars consume DataBlock particles addressed to them and discard all others, and other Bus stars rebroadcast the particle. Processor stars accept only DataBlock particles that are addressed to one of their mapped Function stars; all other DataBlock particles are ignored.

Thus, the architectural trade tool creates the entire performance model for the DE domain, creating all necessary Source, Processor, Sink, Bus, and Function stars, establishing the necessary dependency information among the stars, and initiating the simulation. As the model simulates, various profile information is collected: the execution start and stop times for all Function stars, the memory and I/O usage on all Processor stars, and the bus traffic on all Bus stars. At the end of the simulation, a Gantt chart (figure 3) displays the execution of functional blocks on each processor, memory and I/O usage, and the data being passed across the busses. This display allows the user to pictorially view the performance of the architecture and mapping to identify bottlenecks as well as underutilized resources.

5. SYSTEM METRICS

In selecting an architecture and a mapping, performance is usually very important. However, certain other system metrics must often be considered as well. As a result, in addition to the Gantt chart provided after each simulation run, the architectural trade tool also gives some feedback on the following system metrics[2]: function, environment, interfaces, schedule, cost, processor, interconnect, software, size, weight, power, reliability, testability, maintainability, fault tolerance, scalability, and standards. These system metrics are estimated using simple models with stored manufacturer specifications, historical data, and certain infor-

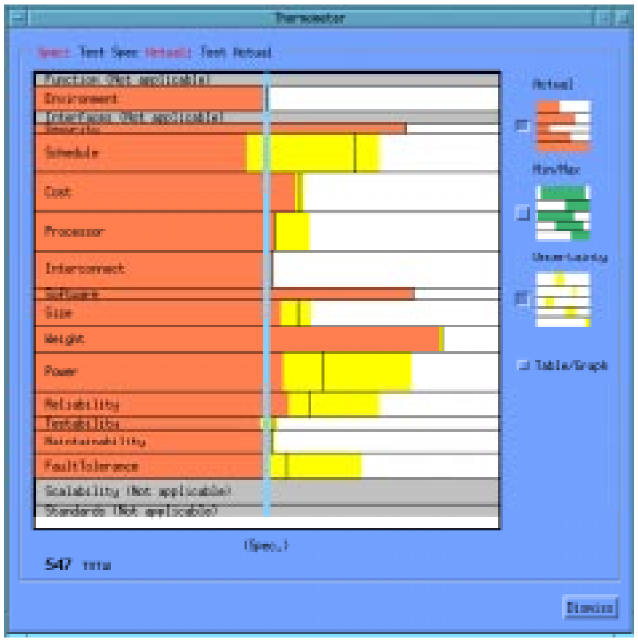


Figure 4: System Thermometer Display

mation from the mapping and performance simulation. The user provides system specifications for each of these metrics, in terms of minimum, nominal, and maximum values. In addition, the user also specifies the relative importance of the metrics to each other using a numeric weighting.

The estimated system metrics are graphed against the given system specifications using a thermometer bar graph display, as shown in figure 4. The height of the individual thermometer bar denotes the relative importance assigned to each metric. All thermometers are normalized so that the center matches the nominal specified value for each metric. A thermometer bar filled in to the left of center indicates that the system metric does not meet the nominal value while a bar reaching to the right of center shows that the nominal value has been satisfied. Some of the metrics are displayed using a reverse scale so that the thermometers are consistent in showing shortfalls--a value to the left of center always indicates a shortfall, regardless of whether the actual numeric value is lower or higher than the nominal value (e.g. power versus reliability). The display has the option of showing the minimum and maximum specification values for the metrics as a range around the nominal value. Because the reported metric values are estimates, a measure of the relative accuracy of the calculations can also be shown as a range around the reported metric. The purpose of these estimates is to provide a first level measure of system metrics to aid in selecting an architecture instead on concentrating entirely upon the performance results.

6. ARCHITECTURE AND MAPPING TRADES

The goal of this tools is to allow the user to quickly explore as many different combinations of mapping and architectures as possible in order to find that one that best meets their needs. The user can easily make some modifications to the mapping or the architecture and re-simulate. These changes may include mapping a function to a different processor, changing to a different type of processor or bus, or adding additional processors and busses. After re-simulation, in less than a minute a new Gantt chart and system thermometer is available. This capability allows the user to check out a number of possible configurations in a relatively short period of time.

7. FUTURE WORK AND SUMMARY

There are several areas that warrant further work or investigation to improve the capabilities of the architectural trade tool. An automated or semi-automated means for profiling functional blocks on different processors, to determine the cost functions, and for performing functional mappings would be very useful capabilities[3,4]. In addition, the models implemented by the Bus and Processor stars could be improved by allowing other types of resource usage besides first-come first-serve. Architectural modeling could also be enhanced by implementing a shared memory block as an architectural entity. The sophistication of the system metrics calculations could be increased by utilizing specialized tools for specific calculations (e.g. reliability, fault-tolerance, etc.). Lastly, a graphical mechanism to facilitate the comparison of the results of several performance simulations would also be helpful in finding the best architecture.

The architectural trade tool provides the user with a convenient means of performing high-level performance simulations of functional mappings onto candidate architectures. The user takes an SDF representation of their algorithm and defines a mapping onto an architecture, and the architecture trade tool creates and runs a DE domain performance model using the Ptolemy kernel. Since the results are available in a matter of seconds, the user can easily explore a number of architectures and mappings. Thus, this tool provides the user with the capability to explore many architectures and architectural mappings for their algorithm than previously possible.

REFERENCES

1. E. A. Lee, et. al., University of California at Berkeley, The Almagest, Volumes 1-4, Regents of the University of California, 1995.
2. F. Shirley and R. Bassett, “Architectures for a RASSP Signal Processor,” Second Annual RASSP Conference, 1995.
3. J. L. Pino and E. A. Lee, “Hierarchical Static Scheduling of Dataflow Graphs onto Multiple Processors,” International Conference on Acoustics, Speech, and Signal Processing, 1995.
4. J. L. Pino, T. M. Parks, and E. A. Lee, “Mapping multiple Independent Synchronous Dataflow Graphs onto Heterogeneous Multiprocessors,” IEEE Asilomar Conference on Signals, Systems, and Computers, 1995.