VHDL Token-based Performance Modeling for 2D and 3D Infrared Search and Track Processing

Eric K. Pauer[a], Mark N. Pettigrew[a], Cory S. Myers[a], Vijay K. Madisetti[b]

[a]Signal Processing Center, Sanders, a Lockheed Martin Company, Nashua, NH 03061
[b]VP Technologies, Inc., Marietta, GA 30067

ABSTRACT

The purpose of this study was to develop and evaluate a new VHDL-based performance modeling capability for multiprocessor systems.[1] The framework for this methodology involved modeling the following system aspects: processor characterization, task modeling, network characterization, and data set size. Initially, all aspects are specified at a high (or abstract) level, and eventually become specified at a detailed level through the process of verification and refinement of design assumptions. Processor characterization involves modeling the processor's speed, instruction set, and memory hierarchy. Task modeling is concerned with the execution time and instruction mix of software tasks within the system. Network characterization models bus protocols, topology, and bandwidths. Data set size refers to how much data is represented by the tokens used in the models. In this study, we applied and evaluated this methodology using both two-dimensional (2D) and three-dimensional (3D) infrared search and track (IRST) algorithms. Two different candidate processors were investigated: IBM PowerPC and Texas Instruments TMS320C80. For the 2D IRST algorithm, the abstract and detailed performance modeling results were obtained for both processors using partitioned data and pipelined algorithmic approaches. The PowerPC abstract performance modeling results were quite close compared to those for the TMS320C80. For the 3D IRST algorithm, abstract performance models for pipelined and parallelized implementations on the PowerPC were developed. These models examined the feasibility of the implementations, the potential risk areas, and laid the groundwork for detailed performance modeling. In short, this methodology provided a good approach to rapidly explore the performance of various system designs.

Keywords: performance modeling, VHDL, IRST, architectures, simulation

1. INTRODUCTION TO PERFORMANCE MODELING

Performance modeling is a method for evaluating alternate design approaches in a consistent and repeatable manner. More precisely, a performance model is an abstract simulation of a candidate architecture. From this model, information is obtained on a system's performance. Three metrics are commonly used to characterize system performance:

- Latency - the average time between the occurrence of events at a given point or between two points in a system.
- Throughput - the average number of events per unit time processed by the system from end-to-end.
- Utilization - the percentage of time a system component is busy.

A performance model includes mechanisms to estimate and apply performance-sensitive information such as software execution times, bus latencies, main memory access times, cache hit ratios, etc. With these system performance metrics, a designer can make informed decisions on architectural options (such as the number and type of processors, the network topology and bus protocol, the amount of custom hardware, the software architecture, and the amount of memory required) without the need for expensive and time-consuming laboratory testing.

Most of a system's cost is committed early in its development. Therefore, it is important to make the right decisions early in the design despite the lack of detailed design information. A performance model provides an abstract representation of the system during the conceptual phase of a product development cycle. These abstract models can be produced rapidly and represent candidate solutions for comparison. Models may reveal bottlenecks, overdesigns, or low-cost candidate solutions or

[1] 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 Project Agency, Electronic System Technology Office, 3701 North Fairfax Drive, Arlington, VA 22203-1714. The Sanders RASSP team consists of Sanders, Motorola, Hughes, and ISX.

provide early detection of system errors. Thus, performance modeling encourages a breadth-first, rather than depth-first, approach. Finally, each performance model provides documentation of the various assumptions made.

2. PERFORMANCE MODELING FRAMEWORK

In this framework, a performance model is defined in terms of four aspects: software task modeling, processor characterization, network characterization, and data set size. The framework describes a performance model by declaring each of the four aspects to be modeled abstractly or with detailed information. We assert that detailed models are more accurate, require more time and effort, improve confidence in system integrity, provide better documentation of the proposed architecture, and are more likely to identify system errors than abstract models. The purpose of this descriptive framework is to differentiate between performance models with varying degrees of system architecture details. This, in turn, will allows us to present a methodology for creating performance models of systems which both explores the design space and verifies the selected design solution. An aspect should be considered detailed if the increased effort improves the accuracy of the performance model and verifies assumptions made in the abstract performance model.

- Software Task Model. A task model represents the timing schedule and functional flow of an algorithm. Within a task model, each task may be realized as dedicated hardware or as software on a programmable processor. In performance modeling, the software task often includes a description of the number and type of instructions, which is required to estimate the execution time of the software on the target processor. An abstract software task model captures functional workloads at a coarse level and is a rough attempt to characterize the algorithm workload. The aim of a detailed software task model is to increase the accuracy of the estimated execution time of the software task on a programmable processor. This may be accomplished by accurately counting the number and type of instructions (e.g., an instruction set simulator) or benchmarking actual code on a target board.
- Processor Characterization. This aspect assumes a programmable processor model in the performance model to process software tasks. A processor characterization is identified as abstract or detailed by its ability to accurately represent cache hit ratios, processor instruction set, main memory access times, interrupt penalties, and context switches. One way to distinguish abstract from detailed processor characterization is to look at the source of the characterization. If a majority of the model is obtained from rule of thumb rather than from actual or specified values, then the characterization is abstract. A detailed processor characterization includes results from simulations or benchmarking [1]. Today, many processors have performance monitoring capabilities built into the CPU architectures themselves [2].
- Network Characterization. This aspect of performance modeling is a description of how well the model captures network considerations, such as topology and bus protocol. An abstract network characterization models the network as a common bus with no bus protocol considerations. The system models the transfer of data between hardware elements as instantaneous. From this configuration, basic I/O requirements between processors can be obtained. A detailed network characterization includes network topology, bus latency and protocol, including information on the arbitration scheme. With this information, bus utilization versus time can be extracted from the performance model.
- Data Set Size. Data set size refers to the amount of data that a token represents. In general, performance models with abstract data set sizes (tokens which represent large blocks of data) run faster than performance models with detailed data set sizes (tokens which represent small blocks of data) because token size is a good indicator of the number of events processed per unit time. Models with small tokens usually have more tokens to process per unit time. This creates much more work for the VHDL simulator, lengthening the run-time. In addition, token size may indicate the resolution of details in the performance model. Software tasks which consume small tokens may indicate a fine grain approach to the algorithm model, suggesting higher accuracy. System errors and bottlenecks may be easier to detect since the simulation operates at a greater level of detail. Although subjective, we considered data set sizes represented by tokens greater than 1% of the input samples transmitted to the system over one second abstract and less than 1% detailed.

3. CASE STUDY OVERVIEW

This case study explores the efficacy of performance modeling as an aid to architecture selection. To accomplish this, we examine system designs developed to process two different algorithms with this performance modeling framework. The two target examples are 2D and 3D infrared search and track (IRST) algorithms. The abstract models are verified by the detailed performance models as well as the results obtained from instruction set simulators and code profiling.

The performance models in this study were developed with Omniview's Cosmos[3] tool. This VHDL-based tool offers a library of system components, such as processors and buses, which can be used to build performance models. Each element

has tunable parameters to allow the components to accurately model the candidate architecture, allowing us to build various performance models quickly. All the work was conducted on a Sun Microsystems UltraSparc I on a networked system. The VHDL simulator was Mentor Graphics' QuickHDL. The subsequent discussion of performance modeling is general enough, however, to be applied to other performance modeling tools.

4. 2D IRST

The 2D IRST algorithm requires the real-time processing of 1.634 Mpixels/sec, with 16-bit pixels (shown in Figure 1). The algorithm is broken into four functional blocks: DC removal filtering, clutter estimation, spatial filtering, and background normalize and threshold processing. Data is transmitted to the candidate multiprocessor system one column at a time, with an input image size of 256 rows by 6383 columns. The prospective multiprocessor solutions investigated consisted of either Motorola PowerPC 604 [2] or Texas Instruments TMS320C80 [4] processing elements. In addition, the assumption was made that processor utilization should not exceed than 80%.

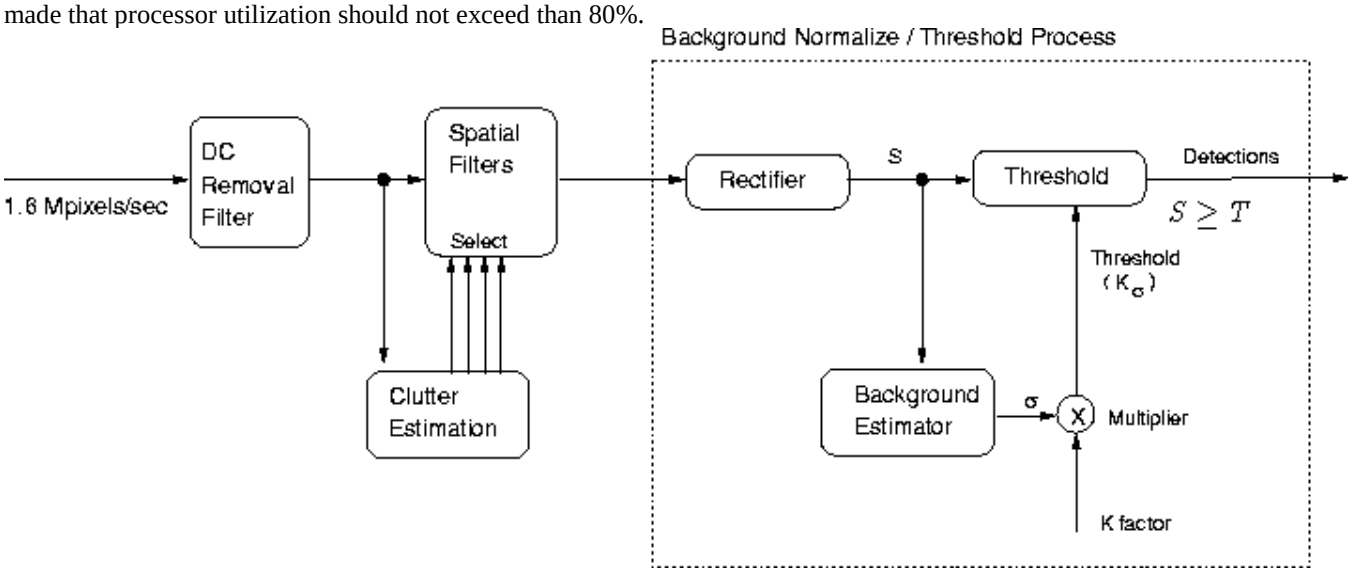

Figure 1: 2D IRST Algorithm

4.1 2D IRST ABSTRACT PERFORMANCE MODELING

Abstract performance modeling is a term used in this case study to describe the minimal amount of details needed to create the performance models for the 2D IRST algorithm. In such performance models, all aspects are normally resolved at the abstract level. In this case, however, the task model resolution and data set size were set to detailed. This change is due to the simplicity of the 2D IRST algorithm, where a detailed task model is easily produced. In this example, a software task model provides an estimate of the computational and memory access requirements of the target algorithm. The computational workload is represented by the number of instructions required for each software task: integer add, load, store, branch, negate, integer multiply, and, or, shift. Abstract network characterization assumed no barriers to interprocessor communication. Communications between processors used a common bus with instantaneous transfer so that the problem was characterized as compute-bound. Data set size varied throughout the examples to match the needs of the candidate implementation.

4.1.1 2D IRST ABSTRACT PERFORMANCE MODELING - POWERPC

For the first set of embedded IRST system designs, PowerPC 604 processors performed the signal processing workload [2]. Different multiprocessor configurations using the PowerPC were explored to arrive at a solution. In the process, performance bottlenecks, software-to-hardware mappings, data traffic rates and the number of processors were identified. The processor model includes a concise instruction set. This instruction set was the link between the processor model and the task model. For the PowerPC 604, the following instructions were selected: IADD, IMLT, LOAD, STORE, BRANCH, IOP where IADD and IMLT represent integer add and multiply instructions, respectively. The IOP instruction describes the class of ALU logical operations. Thus, the six instructions shown in Table 1 characterized the 2D IRST processing needs.

Instruction	IADD	IMLT	LOAD	STORE	BRANCH	IOP
DC Removal	5	1	4	3	0	0
Clutter Estimate	7	0	4	3	8	1
Spatial Filtering	48	15	49	2	1	2
Background Normalize	11	1	6	5	4	3
Total	71	17	63	13	13	6

Table 1: 2D IRST Processing Requirements

This investigation involved three groups of performance models. In the initial model, we simulated the 2D IRST algorithm running on a single PowerPC to compare processor computational power to the algorithm computational needs. Second, with this information, we created a model of the most straightforward and efficient solution available for this type of algorithm-processor combination: a partitioned data approach. In this approach, the 2D IRST algorithm was run on multiple processors for different partitions of the image. Collectively, they processed the entire image. Third, the obvious nature of this 2D IRST algorithm multiprocessor solution, i.e., the partitioned data approach, did not lend itself to extensive performance modeling. However, we include a discussion of the less efficient partitioned algorithm approach to illustrate why it is a less efficient approach. In addition, this approach is also useful in showing how performance models might be used to partition an algorithm for a custom hardware implementation. This approach reveals algorithm performance bottlenecks and suggests how algorithm segments should be pipelined and/or parallelized.

Single Processor Design. The initial design uses one PowerPC 604 to run the entire algorithm. This gives a lower bound on the number of processors required to solve the problem. In this model, a single PowerPC can process 138.5 Kpixels/sec, which is 8.5% of the total required processor power. Assuming linear speedup, a minimum of 12 PowerPC processors will be required to perform the task. This performance model provided a breakdown of processor resources consumed per task. It revealed that the spatial filtering task would require special attention in the partitioned algorithm approach because it represented a major performance bottleneck.

Partitioned Data Design. In the partitioned data approach, the input image is partitioned and processed on separate processors. Since the output of the processed image is detection reports, i.e., flags declaring a specific pixel judged to be a target, the partitioned image does not have to be reconstituted. The infrequency of detection reports means output communication bandwidth is low. Two data partition schemes emerge: row tiling and column tiling. Column tiling is much more efficient because the large number of columns compared to rows. It is necessary to add overlap regions to each tile, 12 columns in this case, so that window operations in the algorithm will be performed correctly. Given that there are 6383 columns in a frame, 12 PowerPCs were necessary to process in real-time. An additional PowerPC was allocated to distribute the image data to the other processors. Each processor contains a copy of the same 2D IRST algorithm code which it runs on its column tile of image data. Figure 2 depicts the utilization versus time for each processor in the system. It reveals a maximum processor utilization of 74.4%.

Paritioned Algorithm Design. In the partitioned algorithm approach, one strategy is to pipeline the four signal processing tasks over four PowerPCs, where each PowerPC performs one task. Here, the spatial filter is the computational bottleneck. The simple pipeline configuration produces a speedup of 1.59 over the single processor design with a throughput of 220.2 Kpixels/second. This is well below the optimal linear speed up of four due to poor load balancing across the system. In a variation of this design, we maintained the pipelined configuration but parallelize the spatial filter. It was partitioned into eight subtasks, where each subtask processed two of the filter product sums. With a ceiling of 80% utilization, a processing rate of 907.5 Kpixels/second was achieved, representing 55.5% of the total processing requirement. However parallelization increased traffic, generating an additional 25.6 MB/sec of data

The previous models illustrate an approach to meeting the performance requirements. To provide real-time processing, pipelining and parallelization techniques had to be extended. The main obstacle was properly partitioning the algorithm. The algorithm was broken into small enough pieces to run on each PowerPC while still meeting real-time performance requirements. The DC removal function was partitioned into a two-processor pipeline while the clutter estimation was partitioned into a three-processor pipeline. The parallelization of the Spatial Filter involved breaking the task into 16 components, each computing a product sum. These product sums were accumulated to generate the filter output. This

required five more processors, yielding 21 processors for spatial filtering. The background and normalization stage was pipelined onto three processors, completing the system. Thus, this solution used 29 processors, with utilizations ranging from 18.2% to 56.0%. A more efficient system was produced by mapping multiple tasks to each processor, staying below the 80% utilization limit. Processor utilizations were assumed to be additive when tasks were placed on a single processor.. With this modification, only 22 PowerPCs were required to meet the system requirements. The utilization versus time is shown in Figure 2.

This partitioned algorithm solution had several difficulties, however. First, it was impossible to partition the algorithm so that the load was evenly balanced across the system. Second, every connection between software tasks mapped to different processors represented sustained data traffic of 3.2 MB/sec, creating a communication requirement 70.4 MB/sec. In addition, this solution required a complex hardware and interconnect configuration, and the development of several versions of the software, each running a different portion of the algorithm.

Figure 2: Utilization Versus Time for Partitioned Data (left) and Partioned Algorithm Approaches Using the PowerPC

Both the partitioned algorithm and partitioned data solutions meet system requirements. In addition, the partitioned data solution had a number of advantages over the partitioned algorithm solution. First, bus traffic was low, 3.2 MB/sec, compared to 70.4 MB/sec for the partitioned algorithm solution. Second, the partitioned data approach used the same code for each processor. Third, fewer processors are needed in the data partitioned solution. This simplified the hardware interconnection and lowered the cost of the system. Fourth, the partitioned data solution resulted in lower latency since initial columns of the frame arriving at the first tile processor are processed immediately. In contrast, the partitioned algorithm solution had to fill up its 11-stage pipeline before detection reports could be produced. Finally, the partitioned data solution used tokens representing the data size of an entire column tile. This results in dramatically fewer tokens in the simulation, yielding faster simulation run times.

4.1.2 2D IRST ABSTRACT PERFORMANCE MODELING - TMS320C80

In this implementation of the 2D IRST algorithm, a Texas Instruments TMS320C80 performed the signal processing workload. Since the software task model is identical in the PowerPC 604 and C80 cases, the same instructions were identified. Similar fixed-point instruction sets allowed the same six high-level instructions to be selected: IADD, IMLT, LOAD, STORE, BRANCH, IOP. The difficulty in producing an accurate high-level characterization of the C80 is its unconventional design. As mentioned previously, the C80 architecture consists of four fixed-point parallel processors (PP) and one master processor (MP) [4]. The MP is a 32-bit RISC processor with a floating-point unit, and it generally

coordinates the operations of the PPs and communicates with external devices. Each PP is a 32-bit fixed-point DSP. The MP and PPs all access a common shared internal memory. To facilitate efficient memory traffic to and from the C80, an intelligent direct memory access (DMA) controller, called the transfer controller (TC), manages all memory transfers.

One approach to model the chip is to simply multiply the processor clock rate by four, to account for the four PPs. However, this scaling factor should be reduced for three related reasons. First, a clock rate increase by four implies linear speedup, which is very difficult to achieve in practice due to memory and synchronization overhead [5]. Second, the algorithm is largely sequential. Finally, the transfer controller provides all the access between main memory and the C80. With four separate processors running, it is not hard to imagine the transfer controller as a performance bottleneck. For these reasons, we increased the clock rate by a factor to three. This assumption is justified since near linear speedup is possible for a low number of processors[6], the increased processing required for data partitioning is relatively small, highly sequential tasks may have been partitioned within the C80 itself, and the TC can transfer at up to 400 MB/sec. Thus, the processor model clock rate was set to 120 MHz since a 40 MHz C80 was used. Recall that interconnections between processors are on an idealized common bus. The system design will, therefore, be characterized as a compute-bound problem. From these abstract performance models, insights into the communication requirements will be drawn.

Single Processor Design. The first performance model ran the 2D IRST algorithm on a single C80, which gave us an rough measure of the minimum number of processor needed for real-time operation. The performance model estimated a single C80 could process 209.2 Kpixels/sec, which is 12.8% of the total required processing power. Given linear speedup, a minimum of eight C80 processors were required to perform the task real-time. Figure 23 shows the utilization versus time for the single processor case. Here we again use the rule of thumb, no more than 80% utilization per processor.

Partitioned Data Design. Next, we applied the single processor information to the partitioned data approach. In the PowerPC approach to row tiling, we saw that row tiles were inefficient due to the large number of rows. Using column tiling, eight C80s are necessary to process the frame in real-time. An additional C80 was allocated to account for the data distribution. High processor utilization (74.2%) results from the balanced distribution of data across the multiprocessor system. The communication bandwidth of this C80 design solution is the same as the PowerPC case, 3.2 MB/sec.

Partitioned Algorithm Design. In the partitioned algorithm, we take advantage of the similar work done for the PowerPC. Here, we reuse the PowerPC partitioned algorithm solution and replace the processors with C80s. Since each C80 has more than twice the processing power of a PowerPC; we expect an overdesigned solution. The results reveal that processor utilizations range from 11.0% through 35.0%, well below the 80% maximum. By combining multiple tasks onto a single processor, the total number of required processors was reduced. In this case, the solution was a twelve C80 multiprocessor system. As in the PowerPC case, every communication between software tasks mapped on different processors produce the sustained data traffic of 3.2 MB/sec, for a total interprocessor communication of 41.6 MB/sec.

The partitioned data solution had a number of advantages over the partitioned algorithm solution. First, bus traffic is low: 3.2 MB/sec compared to 41.6 MB/sec. Second, the partitioned data approach used the same software on each processor. Third, fewer processors were used in the partitioned data approach. Fourth, the column tile approach resulted in lower latency since initial columns of the frame arriving at the first processor are immediately processed. Finally, this solution uses fewer tokens because the represented data size per token is larger, resulting in faster simulation run times.

4.1.3 SUMMARY OF 2D IRST ABSTRACT PERFORMANCE MODELING

In summary, the abstract performance models reveal several insights about the PowerPC 604 and C80 2D IRST multiprocessor solutions. Data partitioning, specifically column tiling, is more efficient than algorithm partitioning. The column tile approach offered a near linear speedup over the single processor performance model for both the PowerPC and C80 cases due to the well-balanced workload. The partitioned algorithm approach was more efficient in the C80 solution than in the PowerPC solution. The increased processing power of the C80 allowed the algorithm to be more effectively mapped. The partitioned data solution required significantly lower data traffic than the partitioned algorithm solution. In both the PowerPC and C80 systems, bus traffic was approximately 3.2 MB/sec compared to 70.4 MB/sec and 41.6 MB/sec for the PowerPC and C80 solutions, respectively. Our results showed that small tokens (such tokens represent small data sets) have a direct impact on model performance. Small tokens resulted in increased events in the simulator, increasing the simulation time.

Parameter	PowerPC Single Processor	PowerPC Partitioned Data	PowerPC Partitioned Algorithm	TMS320C80 Single Processor	TMS320C80 Partitioned Data	TMS320C80 Partitioned Algorithm
Throughput	138.8 Kpix/sec	1.76 Mpix/sec	1.69 Mpix/sec	209.2 Kpix/sec	1.76 Mpix/sec	1.75 Mpix/sec
Utilization	79.8%	74.3%	52.6%	79.9%	74.2%	62.8%
Latency	Not real-time			Not real-time		
Bus traffic	Not real-time	3.2 MB/sec	70.4 MB/sec	Not real-time	3.2 MB/sec	41.6 MB/sec
Processors	1	13	22	1	9	12

Table 2: Summary of Results from 2D IRST Abstract Performance Modeling

4.2 2D IRST DETAILED PERFORMANCE MODELING

Detailed processor performance modeling takes this initial model of the candidate system architecture and tests assumptions made about the processors during abstract performance modeling. The goal of this stage is to scrutinize and verify assumptions made in the programmable processor model and its corresponding software task. For example, the estimated cache hit ratio may have been too optimistic or the processor architecture may be poorly suited to the algorithm. Detailed processor performance modeling more finely tunes the processor model to reflect the algorithmic workload. This effort includes improving the software task model accuracy and usually requires software code to be written and benchmarked for the processors of interest. An instruction set simulator of the processor can be used to obtain a better estimate of the number and type of instructions executed. Alternatively, the algorithm itself or key portions of the algorithm can be benchmarked on actual hardware to obtain real values for the execution. In most cases, code is not efficiently tuned to the processor's architecture, and should be considered a conservative estimate of the amount of CPU resources the algorithm will consume. This discussion describes two detailed processor performance models for the partitioned data system architecture discussed in the previous sections.

4.2.1 2D IRST DETAILED PROCESSOR PERFORMANCE MODELING - POWERPC

From the algorithm specification, the 2D IRST algorithm was coded in C. Some high-level techniques were used to help make the code somewhat efficient, but the results from the software should be viewed as conservative. An instruction set simulator for the PowerPC, called PSIM[7], was used. PSIM has a number of performance monitoring capabilities such as instruction frequency counting, execution unit modeling, and instruction cache performance. This information was used to update the instruction cache hit ratio and refine the software task models. The software task was updated by taking the PSIM results and producing a small, representative instruction set. With this instruction set, an instruction frequency mix was computed to process a column of input data. This detailed processor model was applied to the single PowerPC case. The resulting performance model yielded a throughput of 146.3 Kpixels/sec, a 5.4% difference from the corresponding abstract performance model for a single PowerPC (138.8 Kpixels/sec). The conclusion drawn from this example is that the instruction set simulator verified the PowerPC processor model and software task assumptions.

The same software used for the instruction set simulator was also run on a target board, a Motorola MVME1604-011 processor module. It has a 100 MHz PowerPC 604 processor, 16K data cache, 16K instruction cache, and 8M of DRAM. The results of the benchmarking showed the PowerPC 604 was capable of processing 126.2 Kpixels/sec, adifference of -9.1% from the corresponding abstract performance model. Once again, the benchmarking of the 2D IRST algorithm on the PowerPC board verified the PowerPC processor model and software task assumptions.

The efficiency rating is defined as the ratio of algorithmic operations to total actual operations performed in the C version of the 2D IRST algorithm. Note that the theorectical operations per pixel count, 103 operations/pixel, assumes that load and store operations do not stall processing. Thus, the ideal execution time for a 256 x 320 image running on a 100 MHz processor is 87.65 ms, compared to the measured execution time of 649 ms. This yields an efficiency of 13.0%. Since efficiency ratings for well-written digital signal processing algorithms are typically 20% to 40%, this indicates that there is more room for improvement. The expectation of more efficient code can be incorporated into the performance model by including an efficiency rating parameter in the software task model.

The two tests, instruction set simulation and benchmarking, verified and refined the results from the PowerPC abstract performance models. The difference in the PowerPC 604 processing power estimates of the three approaches (shown in Table 3) did not warrant revisiting the abstract performance modeling stage. In addition, the refined processor models were used in our final stage of the performance modeling, Detailed Processor and Network Performance Modeling.

4.2.2 2D IRST DETAILED PROCESSOR PERFORMANCE MODELING - TMS320C80

Unlike PSIM for the PowerPC, the C80 instruction set simulator did not have the capacity of producing a summary of the aggregated instruction mix and frequency. Therefore, only code benchmarking on a target board was performed to verify the C80 processor and software task models. The target was an Ariel Griffin board with a single 40 MHz TMS320C80 processor with 8 Mbytes DRAM. The code used to perform the PowerPC benchmarking was modified to accommodate the parallel programming model of the TMS320C80. The major changes were driven by the fact that the C80 has the four PPs. Basically, the code was modified so that each PP could process about 25% of the data. The results of the benchmarking showed a single C80 was capable of processing 391.4 Kpixels/sec. The ideal execution time for the 256 x 320 image, using 103 operations/pixel, was calculated to be 27.4 ms, assuming the C80 performs eight operations per cycle. With a measured execution time of 209.3 ms, the efficiency rating was 12.6%. The relatively low efficiency rating points to the potential for more efficient code. As in the PowerPC case, this expectation of more efficient code can be incorporated into the performance model by including an efficiency rating parameter in the software task.

The benchmarking test for the single C80 case (391.4 Kpixels/second) revealed the abstract performance model (209.2 Kpixels/second) to be significantly in error, a difference of 87.1%. There are several reasons for this discrepancy. First, we underestimated the ability of the C80's TC to efficiently provide data to its four PPs. Also, we underestimated the number of operations the C80 could really do in a cycle. The conclusion to be drawn here is that verification techniques are especially helpful when one has no previous experience with the processor or when the processor architecture is very complex. Thus, the C80 benchmarking served to refine the processor model, which was used in the final stage of performance modeling.

4.2.3 2D IRST DETAILED PROCESSOR PERFORMANCE MODELING - SUMMARY

In both the PowerPC and C80 cases, the instruction set simulator and benchmarking experiments verified and refined the processor models. This was accomplished by developing a C code version of the 2D algorithm for the PowerPC and C80 cases. The total time to develop the C code for both the PowerPC and C80 cases was 450 hours. This is averaged out to roughly 19 lines of code per day. This amount of time represented work done on the coding, integration and test of the software, as well as the integration and test of the PowerPC and C80 single processor boards.

Processor	Abstract Model Throughput	Throughput from ISS	Throughput from target board	Software Efficiency
PowerPC	138.8 Kpix/sec	146.3 Kpix/sec (5.4%)	126.2 Kpix/sec (-9.1%)	13.5%
TMS320C80	209.2 Kpix/sec	Not available	391.4 Kpix/sec (87.1%)	13.1%

Table 3: Summary of Detail Performance Modeling - Results for Single Processor

The difference in accuracy between the abstract performance models and detailed processor performance models did not warrant looping back up to the abstract performance modeling space and re-exploring the design space. The partitioned data solution was clearly superior to the partitioned algorithm solution. The large difference between the C80 abstract and detailed processor models highlights the need to fully understand the candidate processor being evaluated. Performance models with processors that have complex architectures greatly benefit from the detailed processor performance modeling stage. In addition, this stage also represents part of the actual design phase, as real software is developed which can be used in subsequent stages of the design process.

4.3 2D IRST DETAILED PROCESSOR AND NETWORK PERFORMANCE MODELING

This detailed processor and network performance modeling stage picked up where the detailed processor performance modeling stage left off by utilizing the detailed processor and software task models developed for the PowerPC and C80. For better comparison, only the processor models developed using the benchmarked results with an efficiency rating parameter were employed. To develop the performance models with network information, we used the VME models from the network

model library available with Cosmos. Two performance models were developed in this stage: a partitioned data architecture using PowerPCs and a partitioned data architecture using C80s.

The software task models used in these performance models were identical to those developed in the detailed processor performance modeling stage for the PowerPC and C80. These performance models revealed average bus utilizations of 30.1% and 28.1% for the PowerPC and C80 cases, respectively. A low bus utilization was expected and found due to the implementation's low data traffic. The difference in bus utilization between the solutions was due to the different number of processors in each system. The PowerPC solution partitioned the data across more processors than the C80 solution. This required more overlap regions, increasing bus traffic.

The result of this stage was to establish the communication network's ability to support the interprocessor data traffic. Low bus utilizations increase the designer confidence that the network can support the data traffic. Clearly, this is a compute-bound application rather than an I/O bound application. Barring other considerations, a VME bus can be used in this system.

4.4 2D IRST SUMMARY

Several conclusions can be made specific to the 2D IRST algorithm example. First, this example appears too simple to fully test the performance modeling methodology. The algorithm itself was not conducive to true architectural tradeoffs. The partitioned data solution was the obvious and most efficient approach. Nevertheless, the example shows the flow of the methodology and how verification techniques were applied. Because the algorithm was simple, the verification techniques were not too difficult to apply. Despite its limitations, the abstract performance modeling stage showed an ability to produce good comparative results. The performance models that were developed illustrated the superiority of the partitioned data solution. In addition, these results were shown to be relatively immune to processor model inaccuracies, specifically the C80 case. On the other hand, the PowerPC case illustrated the potential for accurate performance models. Finally, the detailed processor performance modeling stage represented the initial stage of software development of an actual design.

5. 3D IRST

The next example in this study is a 3D IRST algorithm, which is much more complex and computationally intensive than the 2D IRST case. The intent was to follow the same performance modeling methodology as the 2D IRST case. The following discussion provides a background for the development and results of the two abstract performance models mapped to PowerPC multiprocessor systems. These models illustrated the risks and feasibility of the proposed architectures with sufficient clarity to highlight the important benefits of performance modeling. For the 3D IRST implementations, a maximum of ten 250 MHz PowerPC 603e processors were allowed due to existing power and space restrictions on the target platform. The throughput requirement was 1.56 Mpixels/sec (16-bit pixels), with processor utilization below 80%.

The 3D IRST performs detection processing in three dimensions: azimuth, elevation, and time. The diagram in Figure 3 represents the flow of the 3D IRST algorithm. Background registration removes apparent background motion of the two frames to a center reference frame. This alignment must be accurate to within a fraction of a pixel and significantly increases the computational complexity of the algorithm. The background registration is performed by generating a grid of match points which are shift parameters. Given two frames sampled in temporal succession, a match point is the amount of movement a given pixel experiences from one frame to the next. The shift parameters provide the basis for aligning the images. A hierarchical, recursive approach using decimated images is employed to determine match points. The spatial-temporal clutter suppression filter exploits the high degree of background correlation within and between frames to suppress clutter. Estimation of clutter in the background registered images drives the filter selection from a pre-designed bank of filters. The result is a locally adaptive clutter suppression filter which produces a registered clutter cancelled image. Target integration and thresholding sums target energy over an expected range of candidate trajectories and reduces the volume of data by thresholding. Threshold exceedances are considered detections. These detections are then fed into the track processing component of the IRST system where detections are used to generate tracks of temporally persistent detections. These tracks are evaluated to determine their validity as a target-based track.

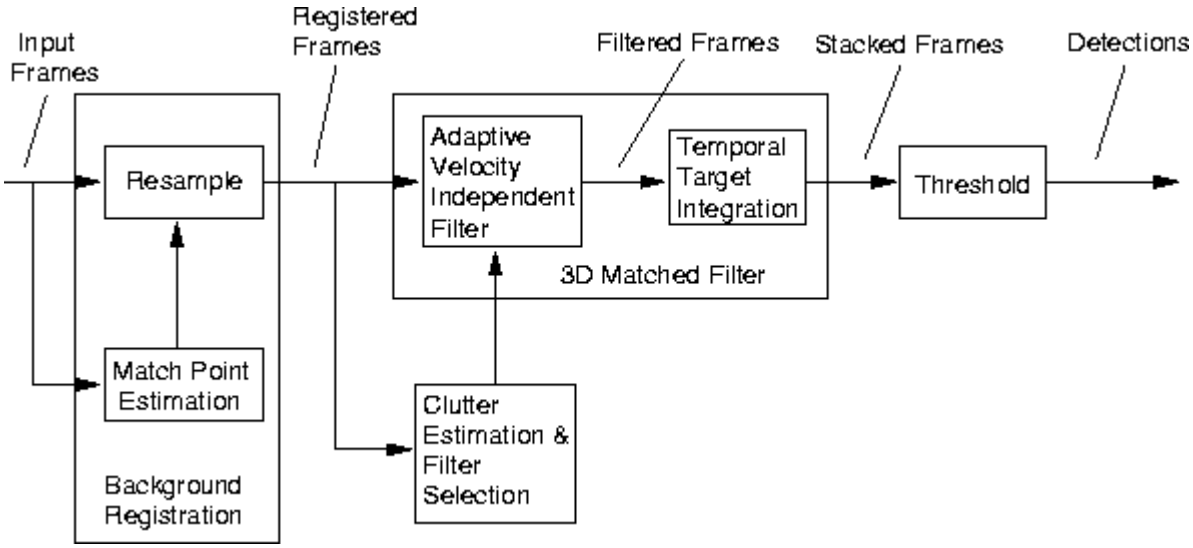

Figure 3: 3D IRST Algorithm

5.1 3D IRST ABSTRACT PERFORMANCE MODELING

A major influence on the development of these abstract performance models is the relaxed constraint on network characterization. The effect of this approach is to focus the design effort on meeting the throughput requirement while neglecting network latencies. This approach yields a first order approximation of the system architecture. In this 3D IRST example, performance modeling is used not only to verify a system architecture but also as an aid to help develop a system architecture which meets the design constraints. The PowerPC processor model developed during the 2D IRST abstract performance modeling stage was reused in this example, with the clock rate is changed to 250 MHz. The detailed PowerPC processor model could not be reused because it was specifically tuned to provide the instruction set for the software task of the 2D IRST algorithm.

The software architecture of the 3D IRST algorithm was modeled by breaking up the algorithm into software tasks, with each task assigned an estimated number of algorithmic operations based on the algorithm specification. The software efficiency rating, defined as the ratio of algorithmic operations to actual software operations, was estimated and used. As before, it was assumed that the PowerPC could maintain an instruction execution rate of one instruction per clock cycle.

The size of the 3D IRST algorithm prevents a comprehensive discussion of algorithmic operations count for the entire algorithm. Instead, we describe the algorithmic operations count for a single example. This will provide an idea of how we produced a software task model for an abstract performance model. The background registration function includes a reduced resolution image, produced by decimating the input image in azimuth and elevation. Given a filter length of 15 and a decimation ratio of two, operations needed for the horizontal decimation are D x W/2 x (15 mult + 14 adds) and for the vertical decimation are D x W/2 x (15 mult + 14 adds), where D and W are the image depth and width, respectively. The design space defined an input image size of 256 x 3800. Therefore, this task requires 21,158,400 operations, providing a rough cut of the amount of operations required to perform this task. Table 4 gives a breakdown of the algorithmic operations count for the major components of the 3D IRST algorithm. This information was used to help develop an initial partitioning of the algorithm on to the ten PowerPCs. Because the two alternatives both used data tiling, the operations count for all software tasks were recalculated to account for the smaller input image sizes and overlap regions.

	Registration	Filtering	Detection	Total
Operations/pixel	337.2	157.5	13.7	508.4
MOPs/second	524.9	245.1	21.3	791.3
Percentage of Load	66.3	31.0	2.7	100.0

Table 4: Computation loads of various portions of 3D IRST

Pipelined/Parallel. The first abstract performance model decomposed the 3D IRST algorithm into two major groupings. The first grouping was the background registration, and the second grouping was the 3D filter, clutter estimation, filter selection and thresholding. This was a natural partitioning since the background registration required more processing power than the rest of the functions combined (see Table 4). A two-stage pipeline for each of these groups was used, with parallelization via data tiling in each stage. The background registration function was mapped to six processors, and the remainder of the algorithm was mapped to four processors. Consequently, in the first stage the input image was divided into six column tiles while in the second stage four column tiles were used. In all, 122 software tasks were mapped to ten PowerPC processors. The results produced by the abstract performance model are illustrated in Figures 4. The steady state utilization of the ten processors is 79.6%. Thus, the processors do not exceed the 80% utilization design constraint and meet the other design constraints as well. However, to stay below the 80% utilization limit, the performance model was run with a global software efficiency rating of 58% which is difficult to achieve in practice. Bus traffic was found to be 8.68 Mbytes/second, and system latency was 812 ms.

Fully Parallel. In the second abstract performance model, the entire 3D IRST is implemented via parallelization via partitioned data (column tiling). All ten processors processed a section of data, except in a few places where nine processors where used and the tenth processor did some global computations. This model required 231 software tasks. From the results of the simulation, it turns out that the maximum utilization was 72.9%. The model runs in real time; however, a global software efficiency of 62.8% was needed to meet this requirement. The bus traffic was found to be 7.49 Mbytes/sec and the system latency was 428 ms. The results are summarized in Table 5.

Figure 4: Utilization Versus Time for Pipelined/Parallel (left) and Fully Parallel Approaches Using the PowerPC

The pipelined/parallel case needed a lower global software efficiency because it used less column tiles in each of its stages (six and four tiles) while the fully parallel case used ten tiles for the entire algorithm; the necessary overlaps cause a slight increasing in required operations. However, the fully parallel case has lower system latency, less bus traffic, and simplier software implementations than the pipelined/parallel case. Since software efficiencies over 40% are very difficult to achieve in practice, these results indicate that the 10 processor limit is probably too restrictive to implement this system with PowerPCs.

System Parameter	Pipelined/Parallel	Fully Parallel

Maximum Throughput (Mpix/sec)	1.564	1.708
Average Utilization	79.6%	72.9%
Number of Processors	10	10
Software Efficiency required	58.0%	62.8%
System Latency	812 ms	428 ms
Bus Traffic (Mb/sec)	8.68	7.49

Table 5: Summary of 3D IRST Performance Modeling Results

6. CONCLUSIONS

In this case study, we used performance models to understand the interaction between the three components of a system architecture: hardware, software and software-to-hardware mapping. The 2D and 3D IRST examples illustrated how these elements can be manipulated to quickly produce a number of performance models of different proposed architectures. A performance model's ability to highlight system performance and bottlenecks makes it an excellent aid in architecture selection. Despite abstract performance models with varying degrees of accuracy, the models allowed good comparisons between different architectures. Our experience in developing models for this performance modeling case study gave us greater insights into the exigencies of performance model development. Abstract performance modeling focuses on meeting the throughput requirements of a system design producing a first-order approximation of the system. Performance modeling provides good relative architecture comparison capability. Performance models do not have "expected results" in the ordinary sense. The verification of a performance model is less rigorous than results in other parts of the design process, and this makes performance models difficult to debug. In addition, too many details reduce the utility of a performance model as a long list of input parameters may produce incomprehensible or unreliable results. The accuracy of a performance model of an embedded multiprocessor system can be estimated but not established until the system itself is built. Despite the above mentioned qualifications, it should be stressed enough that performance models are very useful. Performance models encourage a more rigorous and thoughtful approach to system architectural design, provide documentation for the design approach, and can model the dynamic behavior of a system design. Complete results of this work is available on-line at http://rassp.sanders.com/legacy.

REFERENCES

1. Berkeley Design Technologies, 2107 Dwight Way, Berkeley, CA 94704 (http://www.bdti.com).
2. IBM Microelectronics and Motorola, Inc., PowerPC 604 RISC Microprocessor User's Manual, 1994.
3. Omniview, Inc., 100 High Tower Blvd., Suite 201, Pittsburgh, PA 15205 (http://www.omnivw.com).
4. Texas Instruments, Inc., TMS320C8x System-Level Synopsis, 1996 (http://www.ti.com).
5. Hennessy, J. and Patterson, D., Computer Architecture: A Quantitative Approach, 2nd edition, Morgan Kaufmann Publishers, Inc. San Francisco, 1996, p. 465.
6. Ibid, pp. 731-736.
7 PSIM, Vers. 1.1, Computer software, Andrew Cagney, 1996 (Unix), available at URL:ftp://ftp.ci.com.au/pub/psim/index.html.