A PERFORMANCE MODELING FRAMEWORK APPLIED TO REAL TIME INFRARED SEARCH AND TRACK PROCESSING

Eric K. Pauer, Mark N. Pettigrew, Cory S. Myers
Sanders, a Lockheed Martin Company
Signal Processing Center
P.O. Box 868, Nashua, NH 03061-0868
(603) 885-8358, Fax (603) 885-0631
{pauer, mpettigr, cory}@sanders.com

Vijay K. Madisetti
VP Technologies, Inc.
Marietta, GA 30067
(404) 894-4696, Fax (770) 578-1576
vkm@vptinc.com

Abstract

The purpose and goals of performance modeling for multiprocessor systems using a token-based methodology in VHDL are discussed. Following this motivation, a framework for performance modeling is described, which involves modeling hardware and software at different levels of abstraction; the scope of this paper primarily addresses the high profile performance model. A commercial tool supporting this modeling framework is then introduced. The discussion continues with an overview of the real time infrared search and track algorithm, and our system design problem. Preliminary results of our performance modeling efforts and validation via code profiling is summarized, and future plans are described.

1. Introduction

Performance modeling plays a key role in the RASSP-based top-down design methodology [1, 2]. It immediately follows the initial design steps which include system requirements capture, algorithm and functional design, and data/control flow design. The results of the performance modeling step can be used to refine the design by optionally revisiting the preceding steps. Nonetheless, performance modeling does drive the succeeding steps in the design methodology, including behavioral virtual prototyping, detailed hardware and software designs, and final prototyping. A primary benefit of performance modeling is that the increased effort it calls for in the conceptual phase of the design greatly lowers the risk and hence the cost in the development of a system, which has also been observed [3]. Throughout this methodology, VHDL [4] plays a key role in the system specification, modeling, synthesis, detailed design, and documentation because of its inherent support for representing systems at different levels of abstraction.

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.

2. Performance Modeling Goals

There are four primary goals of performance modeling in this methodology. First, system sizing can be accomplished including the number and type of processors, memories, and buffer elements. Second, network architecture considerations delve into the system's network topology (e.g. shared bus, ring, cube). Link bandwidths and protocol requirements are also explored under network architecture selection. Third, hardware/software mapping includes application partitioning and allocation, task scheduling and flow control, and assessing microprocessor communications. Fourth, concept verification deals with processor and link throughput and utilization. Note that performance modeling itself is a tool for satisfying these goals, and as such may also be useful in other steps of the top-down design methodology.

At the performance level, interaction between hardware and software is analyzed. Software tasks are represented as the execution of primitives on a processor. This is accomplished by modeling each primitive as a simple delay rather than a sequence of assembly instructions. Performance models are not concerned with the actual data but rather with the flow of data through the system. A data type, known as a token [5], abstractly represents this flow of data. Performance models allow the designer to explore the what (software task), where (processor element) and when (token) of a system design at a high abstraction level. The output of performance modeling is an efficient system architecture solution, which drives behavioral virtual prototyping, the next step in the design methodology. To more easily assess this performance modeling approach, this effort has limited its scope to the performance modeling of deterministic, synchronous algorithms.

3. Levels of Modeling Abstraction

The general structure of performance modeling is illustrated in Figure 1. The algorithms, implemented in software, are decomposed into separate cooperating tasks. Processor elements are characterized by descriptions of their instruction sets, memory hierarchies and token interfaces. Tokens model the data communication between processors. Communication mechanisms between processors are represented as abstract blocks, but can be realized using a variety of bus protocols and topologies. Finally, func-

Table 1: Aspects of performance modeling for each abstraction level

Major Aspect	High	Medium	Low
Software Task Modeling	time delay estimate	partial VHDL implementation of software tasks with time delay estimates (HW/SW Codesign)	full VHDL implementation of software tasks with time delay estimates (HW/SW Codesign)
Processor Characterization	high level DSP primitives, small set of simplified instructions	low level DSP primitives, medium to large instruction set	detailed instruction set, context switch support, detailed cache modeling
Network Communication	generic point-to-point communications	bus topology token I/O bus I/O instructions	bus protocol (VME, Myrinet, PCI, Raceway, SCI, etc.)
Token Resolution	10% to 100% of input sample rate	1% to 10% of input sample rate	less than 1% of input sample rate

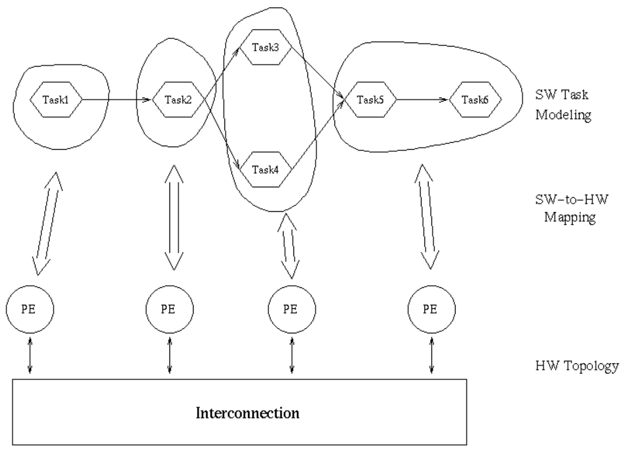


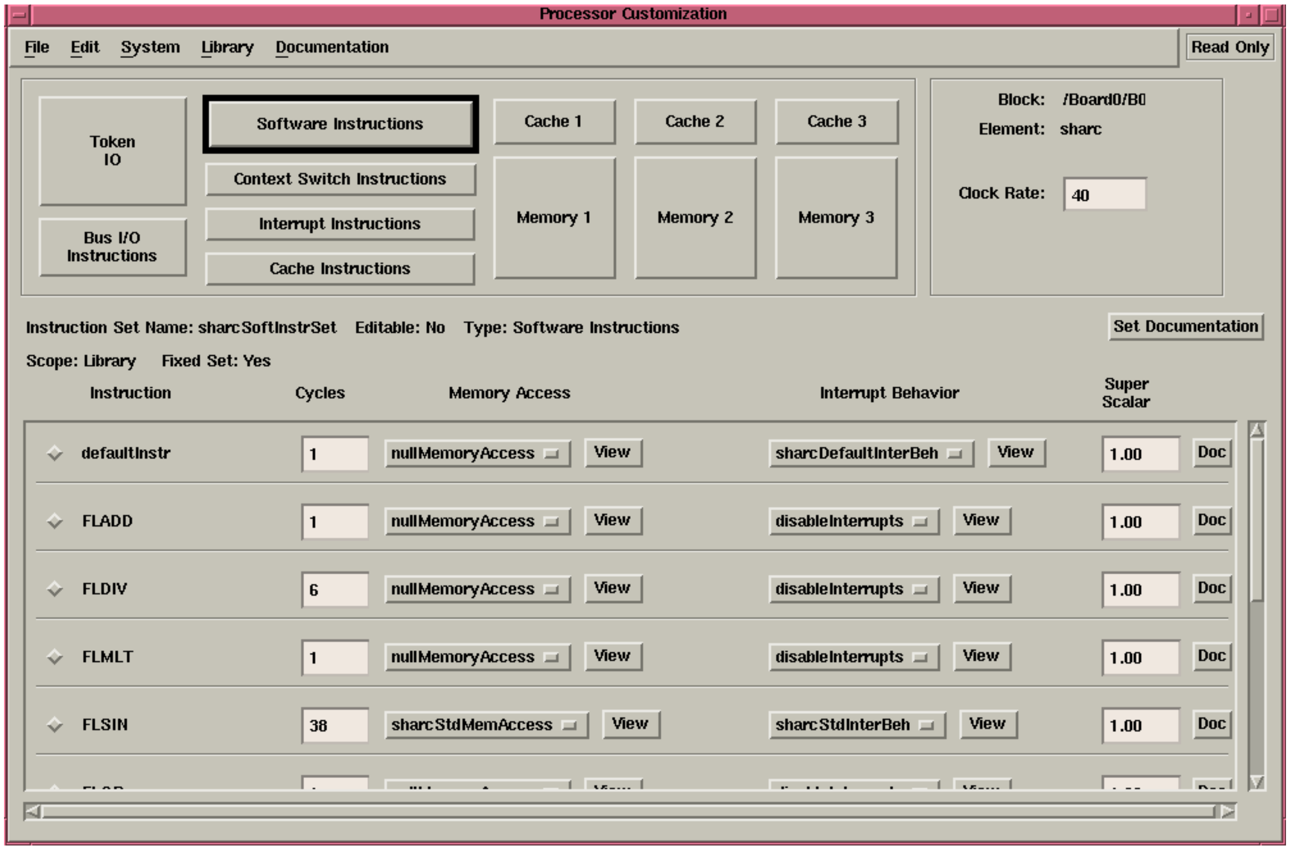
Figure 1: General structure of performance modeling

tional blocks of the target algorithm are mapped onto specific processors.

In this study, a flexible performance modeling framework to accurately and rapidly explore the architectural design space is used. It consists of the following four major aspects: software task modeling, processor characterization, network communication, and token resolution. Using high, medium, and low levels of abstraction for each of these aspects, three corresponding performance modeling profiles (high, medium, and low) have been developed, as shown in Table 1. The low level of abstraction requires an increased model development and simulation time, with the benefit of more accurate models and a commensurate higher confidence in the integrity of the proposed architectural design. The high level of abstraction yields quicker results with less effort, but typically does sacrifice the accuracy of the results. In an ideal situation, one would start with a high level profile to explore many options identify candidate architectures, explore promising candidates with more refined models at the medium level, and investigate a handful of models at the lowest level. However, for a particular project, it may be appropriate to use just one profile level, or combine different levels of abstraction from these major aspects for a customized performance model. The right approach depends largely on the time, budget, availability of modeling resources, and purpose of the study. This framework is intended as a set of flexible guidelines to raise issues of importance in developing performance models. The scope of this paper discusses primarily our results of applying the high level profile performance modeling to our system design problem.

4. Environment for Performance Modeling

Through RASSP-sponsored research, a generic, parameterizable library of VHDL performance models, called the Performance Model Library (PML) [5, 6, 7], was developed. PML serves as the foundation for a new modeling environment called Cosmos (formerly the Performance Modeling Workbench (PMW)) [8], developed by Omniview, Inc. [9]. Cosmos uses elements from PML to model processors, communication elements, I/O devices, and software tasks. This library contains many commonly used models, and new customized models can be created by modifying the parameters of these library elements. Beneath each model or component is VHDL code or a flow chart modeling the behavior, which is tuned with its available parameters. For example, a processor is modeled by specifying its various attributes including information about its instruction set, memory and cache architecture, and operating system, as shown in the Cosmos graphical user interface (GUI) in Figure 2. Hardware architecture topologies are composed by creating instances of the hardware components and by setting or modifying their parameters as necessary. Finally, the hardware components are connected together to complete the hardware model.



Block: /Board0/B0
Element: sharc
Clock Rate: 40

Instruction Set Name: sharcSoftInstrSet Editable: No Type: Software Instructions
Scope: Library Fixed Set: Yes

Instruction	Cycles	Memory Access		Interrupt Behavior		Super Scalar	
defaultInstr	1	nullMemoryAccess	View	sharcDefaultInterBeh	View	1.00	Doc
FLADD	1	nullMemoryAccess	View	disableInterrupts	View	1.00	Doc
FLDIV	6	nullMemoryAccess	View	disableInterrupts	View	1.00	Doc
FLMLT	1	nullMemoryAccess	View	disableInterrupts	View	1.00	Doc
FLSIN	38	sharcStdMemAccess	View	sharcStdInterBeh	View	1.00	Doc

Figure 2: Processor modeling parameters

Software is modeled separately from the hardware. The software tasks implementing the algorithm are specified graphically to establish their precedence and dependence relationships. The software design editor captures the target algorithm graphically. Within Cosmos, software tasks are individually modeled using VHDL or a flow chart. Once both hardware architectures and software tasks are specified, the software tasks are mapped onto the processor elements. After the mapping is complete, Cosmos generates a VHDL model of the system, which is then simulated by any IEEE-compliant VHDL simulator. As the model is simulated, a transcript file containing the results of the simulation is produced. The simulation times in this effort ranged from one to eighteen minutes. The results are then read, interpreted, and displayed by Cosmos. The Cosmos GUI supports various system and component level views of the results, facilitating the analysis of system performance (throughput, latency, utilization, etc.), identification of bottlenecks and over-designs, and comparisons of designs. In short, Cosmos provides the GUI, for creating the models and interpreting the simulation results, as well as the library containing the hardware and software models. A separate VHDL simulator provides the execution environment which performs out the system simulation. This design process is summarized in Figure 3.

5. IRST Algorithm and the Modeling Problem

Infrared Search and Track (IRST) systems are a class of passive military infrared systems. The goal of such a system is to reliably detect, locate and track infrared-emitting objects. The presence of background radiation and other disturbances increases the difficulty of this task. Valid targets are modeled as point-source objects in highly structured background at ranges beyond five kilometers. IRST systems are considered an alternative to radar in certain cases due to their passive nature and anti-stealth capability. The characteristics of an IRST system include a wide field of view, with coverage up to 360 degrees in azimuth and up to 90 degrees in elevation, for an airborne system. As a result, typical frame times on the order of one to ten seconds and typical pixel rates of one million pixels per second are encountered due to the large images these systems generate. Due to the high volume of pixels and the priority of detecting small targets, IRST systems depend on automation to screen false

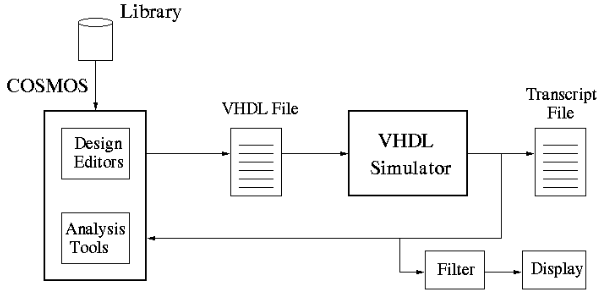


Figure 3: Cosmos design process

alarms. Most current research is focused on signal processing to detect targets in the presence of severe clutter [10].

The signal processing for IRST systems typically consist of two components: a detection processor and a track processor. The detection processor performs spatial processing on the input image at the required high data rate and thresholds the result to pass a manageable number of threshold exceedances (detections) to the track processing function. The track processor operates on the detections from each frame to generate tracks of temporally persistent detections and evaluates the validity of a particular track being target based. This performance modeling effort focused on modeling the implementation of the detection algorithms using 2D and 3D spatial processing to reliably detect infrared-emitting objects. As the subsequent discussion will reveal, these algorithms require a large amount of computational power (on the order of 100-1000 operations per pixel). Given this requirement, a multiprocessor system is necessary to process the data in real-time. Performance modeling will be shown to be a powerful aid in exploring the architectural design space of a high performance multiprocessor system.

As a vehicle for demonstrating this performance modeling approach, the detection portion of the 2D IRST algorithm is being investigated for implementation on two candidate processors, the PowerPC 604 [11] and the Texas Instruments TMS320C80 [12]. The scope of this paper is limited to high profile performance modeling. As depicted in Figure 4, the 2D IRST consists of three main stages: a DC

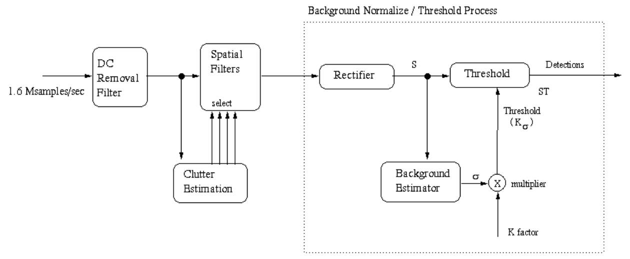


Figure 4: 2D IRST detection block diagram

removal front end, a clutter estimation and spatial filter section, and a background normalize/threshold stage where detections are made. The algorithm was prototyped and functionally verified using the Synchronous Dataflow domain of Ptolemy [13], which is a C++ based simulation environment. Our system problem was to evaluate candidate implementations using high profile performance models. These implementations included information on the number of processors, hardware topology, and algorithmic mapping needed to handle the throughput of 1.6 Mpixels/second, using architectures consisting of either the PowerPC or the TMS320C80 processors.

6. Software Task Modeling

In high profile performance modeling, the performance model is assumed to be derived from a paper specification. Hence, the computational and memory access requirements of each functional block were estimated from this specification. The same software task models were used for the PowerPC and the C80. At this level of performance modeling, the differing processor performance is captured by their respective processor characterizations, discussed further in section 7.

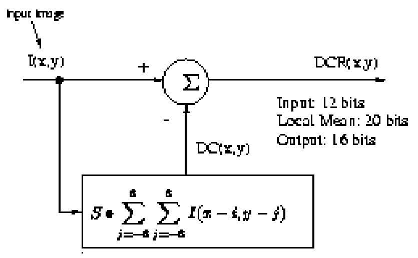


Figure 5: DC removal block diagram

6.1. DC Removal

The first functional block in the 2D IRST algorithm is the DC removal block. This function uses a 2D finite impulse response (FIR) filter with unity coefficients to eliminate the DC component from the image. Figure 5 shows a block diagram of this filter. Since the filter is really just a 13 x 13 summation window, it can be implemented very efficiently in a systolic fashion. The systolic implementation slides a summation window down the columns of the image. Pixels within the filter window are summed along the rows, and each row sum is kept in storage. Adding these row sums produces the filter output. The systolic nature of this approach has two consequences. First, the computational and memory access workloads are constant regardless of the summation window size. It is the size of the storage area that varies with window size. Second, there is a region around the edge of the image where the filter output is invalid because a part of the filter does not overlap the image. The following computational and memory access workload per pixel for the DC Removal filter is shown in Table 2.

Table 2: DC removal filter workload

Workload Estimate
5 adds
4 loads
3 stores

6.2. Clutter Estimator

The next software task is the clutter estimator and spatial filter. The clutter estimator performs a local standard deviation estimate and a filter select operation. The local standard deviation estimate is identical to the summation window except it processes the absolute value of the input image. The filter select uses this estimate to categorize the clutter into one of four clutter categories (none, low, medium, high) using three thresholds. This selection may be implemented with just two branches per pixel and needs the following resources, shown in Table 3.

Table 3: Filter select workload

Workload Estimate
3 adds
6 branches
1 load
1 store

Table 4 has the total clutter estimate workload requirements, which is the sum of the local standard deviation estimate and filter select.workloads.

Table 4: Clutter estimate total workload

Workload Estimate
7 adds
4 loads
8 branches
3 stores
1 negate

6.3. Spatial Filter

The output of the filter select drives the spatial filter, which applies one of four 7 x 7 filter kernels. The adaptive spatial filter performs two tasks: (i) spatial filtering and (ii) a scale/clip operation. Since the kernels possess quadrature-mirror symmetry, each kernel has just sixteen unique coefficients. By pre-adding the spatial data and then multiplying by these coefficients, the computational load can be reduced to 48 adds and 15 multiplies. The total workload requirement for the spatial filter per pixel is listed in Table 5.

Table 5: Spatial filter total workload

Workload Estimate
48 adds
49 loads
15 multiplies
2 stores
1 shift
1 AND/OR
1 branch

6.4. Background Normalizer/Threshold

The final functional block, the Background Normalizer/Threshold Process, as shown in Figure 6. It includes two

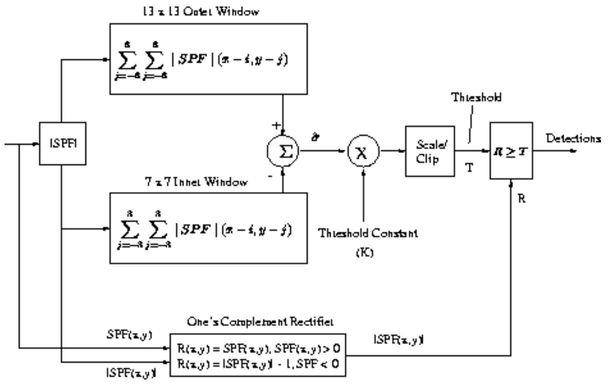

13 x 13 Outer Window
$\sum_{j=-6}^{6}\sum_{j=-6}^{6} |SPF|(x-i, y-j)$
Threshold
|SPF|
+
Σ
$\hat{\sigma}$
X
Scale/ Clip
T
$R \geq T$
Detections
-
7 x 7 Inner Window
$\sum_{j=-3}^{3}\sum_{j=-3}^{3} |SPF|(x-i, y-j)$
Threshold Constant (K)
R
One's Complement Rectifier
SPF(x,y)
R(x,y) = SPF(x,y), SPF(x,y) > 0
R(x,y) = |SPF(x,y)| - 1, SPF < 0
|SPF(x,y)|
|SPF(x,y)|

Figure 6: Background normalizer/threshold

summation windows which each have the same loads as the DC removal block. A total breakdown of computational and memory access workloads for each component of the Background Normalize/Threshold Process per pixel is shown in Table 6.

Table 6: Background normalize / threshold total workload

Workload Estimate
11 adds
6 loads
5 stores
4 branches
1 multiply
1 negate
1 shift
1 AND/OR

6.5. Total Workload

The above approximations of the 2D IRST algorithmic workload were used to specify the software tasking models, and have been totaled in Table 7. Note that a generic instruction set has been employed at this level, e.g. load, add, multiply. At lower levels of abstraction, performance models may contain sufficient instruction set detail to allow modeling the software tasks more accurately. It is important to try to keep the high-level software task models as generic as possible so they can be easily translated to different target architectures. This aspect of high-level software task models enable the architectural design space to be explored rapidly.

Table 7: 2D IRST algorithm total workload

Workload Estimate
71 adds
63 loads
16 multiplies
13 stores
13 branches
2 negates
2 shifts
2 AND/OR

7. Processor Characterizations

In this section, the various instructions used to model both the algorithm and the instruction sets of the processors are discussed and explained. Due to the relative simplicity of the algorithm and the limited number of ways it can be partitioned on the PowerPC and C80, no operating system overhead was considered directly. However, a 10% software overhead factor was used to represent initialization and loop setup stages of the software tasks in both cases.

7.1. PowerPC Processor characterization

For this first case, PowerPC 604 processors are the elements which perform the 2D IRST processing workload. Different multiprocessor configurations were explored to arrive at a potential solution. In the process, performance bottlenecks, software to hardware mappings, data traffic rates and the number of processors were identified. It is important at this level to use a simple model of the processor to simplify the replacement of different processors as well as to easily modify the software task models. For instance, if parallelization becomes necessary, it should be easy to break a large task into smaller parallelized portions. This is accomplished with a concise instruction set for the processor. A high-level instruction set is selected only after the task modeling phase is completed.

In the 2D IRST algorithm, the following unique instructions were identified: integer add, load, store, branch, negate, integer multiply, and, or, shift. For the PowerPC 604, the following instructions were selected: IADD, IMLT, LOAD, STORE, BRANCH, IOP. The mnemonics IADD and IMLT represent integer add and multiply instructions, respectively. The IOP instruction describes the class of ALU logical operations. Thus, six instructions were used to characterize the IRST algorithm processing needs. Instruction execution performance information is shown in Table 8 [11]. This information was necessary to calculate the time delays for each function processed on a PowerPC.

In our examples, the cache hit rates were set at 90%. This value was judged a reasonable starting point. This assumption was drawn from the cache hit rate analysis performed on the SPEC92 benchmarks [14]. The potential dangers in generalizing the cache hit rate must be stressed, however. Hit rates are extremely dependent on cache size, data dependencies and algorithm implementation. In high profile performance modeling, this is a unavoidable since no software code is assumed to exist from which cache hit rates can be determined. Main memory access times are determined in terms of the number of clock cycles. In this example, it takes six clock cycles to access a byte in main memory. Of course, different bus speeds are available with a corresponding impact on system cost. Bus speed should be considered an important variable in assessing the performance of a processor since efficient memory hierarchy often translates directly into improved processor performance.

Table 8: PowerPC high level processor instruction set

Instruction	Cycles	Super-scalar	Memory Access*
IADD	1	2	fetch 4 I bytes
IMLT	1	1	fetch 4 I bytes
LOAD	1	1	fetch 4 I bytes, fetch 4 D bytes
STORE	1	1	fetch 4 I bytes, store 4 D bytes
BRANCH	1	1	fetch 4 I bytes
IOP	1	1	fetch 4 I bytes

7.2. TMS320C80 Processor characterization

In this second implementation case for the 2D IRST algorithm, Texas Instruments TMS320C80 processors perform the signal processing workload. Since the software task model is identical with the PowerPC software task model, the same unique instructions were identified. Similar fixed-point instruction sets in the C80 allowed the same six high-level instructions as in the PowerPC to also be selected. Performance information on the instruction set execution is shown in Table 9. As in the PowerPC case, the cache hit rate was set at 90%.

The difficulty in producing an accurate high-level characterization of the C80 is its unconventional configuration. The C80 consists of one master processor (MP) and four parallel processors (PP), with 38 Kbytes of internal shared memory. The MP is a 32-bit RISC floating-point processor, and is used to coordinate the operation of the four PPs and to communicate with external devices. Each PP is a high perfor-

* I and D bytes refer to instruction and data memories respectively. Cycles represent the number of clock cycles required to execute the instruction. Superscalar refers to the number of functional units available to execute the operation.

Table 9: TMS320C80 high level processor instruction set

Instruction	Cycles	Super-scalar	Memory Access*
IADD	1	1	fetch 8 I bytes
IMLT	1	1	fetch 8 I bytes
LOAD	1	2	fetch 8 I bytes, fetch 8 D bytes
STORE	1	2	fetch 8 I bytes, store 4 D bytes
BRANCH	1	1	fetch 8 I bytes
IOP	1	1	fetch 8 I bytes

mance 32-bit fixed point DSP. To facilitate efficient memory traffic, the C80 also has an intelligent DMA controller, called the transfer controller (TC), which manages all memory transfers to and from internal memory, including cache servicing.

A reasonable first order approach to model the chip is to multiply the C80 clock rate by four, corresponding to the number of PPs. Since the algorithm is fixed-point, the floating-point MP, which traditionally acts as a task coordinator anyway, is not taken into account as contributing any usable processing power. However, a more reasonable assumption is to adjust the factor of four downward for three reasons. First, a clock rate increase of four implies a linear speed up which is very difficult to achieve in practice due to problems such as memory and synchronization overhead [15]. Second, the algorithm is largely sequential in nature, and not entirely parallelizable. Finally, the TC may become a performance bottleneck as the amount of data transfer increases. For these reasons, the clock rate was conservatively increased by a factor of three rather than four.

The C80 architecture in this case study consists of a 40 MHz C80 with a 20 MHz 64-bit bus. Thus, the processor performance model sets the clock rate at 120 MHz. Main memory access times were determined in terms of clock cycles. Eight bytes can be transferred from main memory every other processor clock cycle. This rate assumes that the transfer controller moves large blocks of data (128 bytes). This is a reasonable assumption for this image processing application.

8. Network Communication and Token Resolution

In the high profile performance modeling, the network communication between processors is modeled as generic point-to-point link with unlimited bandwidths. With this approach, there are no barriers to interprocessor communication. Thus, the problem is characterized as compute-bound. By not imposing a ceiling on the communication bandwidth within the models, the amount of communication needed to support a particular architecture and mapping can be estimated. These estimates can then be judged as to their feasibility in being realized in hardware. More importantly, these estimates are useful in transitioning to the medium and low profile performance models where communication bandwidth limits are part of the models.

Consistent with the discussion in section 3, a relatively coarse token resolution was used. Here, tokens were used to represent data blocks ranging from a column of 256 samples up to 800 columns (204,800 samples). This approach is appropriate at this level of abstraction as it is consistent with the expected accuracy of the results, in addition to keeping simulation times relatively short.

9. Design Space Exploration

9.1. Performance modeling using PowerPC-based designs

In this section, the progression of performance models to arrive at a viable solution using PowerPCs in a multiprocessor system is discussed. Various hardware configurations were modeled by adding additional PowerPCs until a viable multiprocessor design solution was reached. From this final performance model, communication traffic issues were estimated.

The initial design uses one PowerPC 604 to run the entire algorithm, providing a lower bound on the number of processors to needed to solve the problem at the 1.6 Mpixels/second rate. Using the algorithmic and processor modeling numbers developed in sections 5 and 6, a single PowerPC was estimated to implement the 2D IRST at 140.8 Kpixels/second, just 8.6% of the required rate. Assuming a linear speedup, at least eighteen PowerPC processors would be required to perform the task. This calculation also assumes that utilization on each processor does not exceed 80%. As expected, the spatial filtering task was the major performance bottleneck, and its alternative implementations were the focus of subsequent designs.

A second design assumed a pipelined configuration for all functions, with the spatial filter additionally being parallelized. The filter was partitioned into eight subtasks, where each subtask processed two of the sixteen product sums. Each of these subtasks was mapped to a single processor, and a ninth task to collect and add the results was assigned to a ninth processor. The other 2D IRST functions were pipelined and assigned to three other processors. With a ceiling of 80% utilization, this model estimated a processing rate of 608.7 Kpixels/second, 37.2% of the total processing requirement. In this design, the background normalize/threshold task became the limiting factor for higher performance as it was assigned to run on just one processor. In terms of communication bandwidth, the eight-way partitioning increased the peak point-to-point bandwidth to 12.8 Mwords/second.

A third PowerPC design parallelized the spatial filter into sixteen separate subtasks (one for each unique filter coefficient), with five additional subtasks to sum the results. The DC Removal task was pipelined into two subtasks, the clutter estimation was pipelined into three subtasks, and the background normalization/threshold was pipelined into three subtasks, for a total of 29 processors. The utilization versus time diagram illustrates the difficulty in balancing the algorithmic workload across the system (see Figure 7).

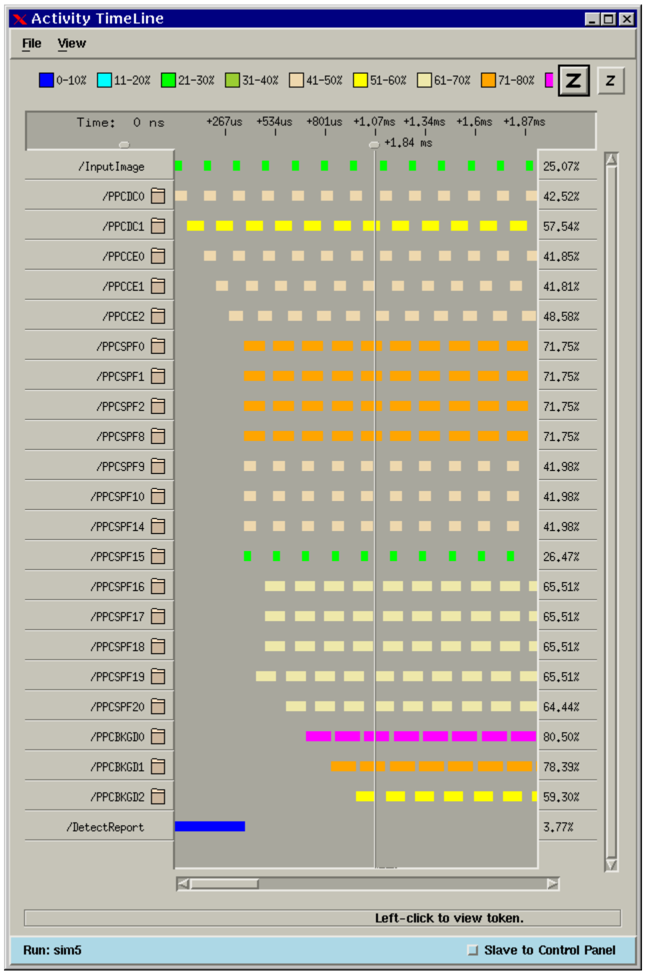


Figure 7: Performance profile of PowerPC partitioned algorithm implementation model

However, this partitioned algorithm solution has several difficulties from a communication perspective. Given that every connection between software tasks represents the sustained data traffic of 1.6 Mwords/second, then the total interprocessor communication requirement (including the input data) is 46.4 Mwords/second. If too many processors were placed on a single board, bus data traffic may be prohibitively high. Thus, this solution requires a relatively complex hardware architecture with less complex software development.

Another approach to a potential solution is through data partitioning. Instead of decomposing the algorithm, 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 straightforward partition schemes will be discussed: row tiling and column tiling.

In row tiling, each tile (section of the image) is composed of a specified number of rows. Column tiling partitions the image along columns. Because each processor will run the entire algorithm on the row (or column) tile, it is necessary to add overlap regions to each tile so that spatial filter and summing window operations will be performed correctly. Since the largest window size involved in these operations is 13 x 13, the row and column tile implementations must have an overlap of six on each side of the tile. Since input images are 256 rows by 6383 columns, row tiling is not feasible; the model provides an estimate of 128 processors to maintain the 1.6 Mpixels/second rate. In contrast, the column tiled model estimated that just eighteen PowerPC processors were required. Software development would be straightforward since each processor runs the same 2D IRST algorithm code to process its own column tile. The only modification to the algorithm is to disallow valid detection reports to come from the overlapping regions of input data. Figure 8 depicts the

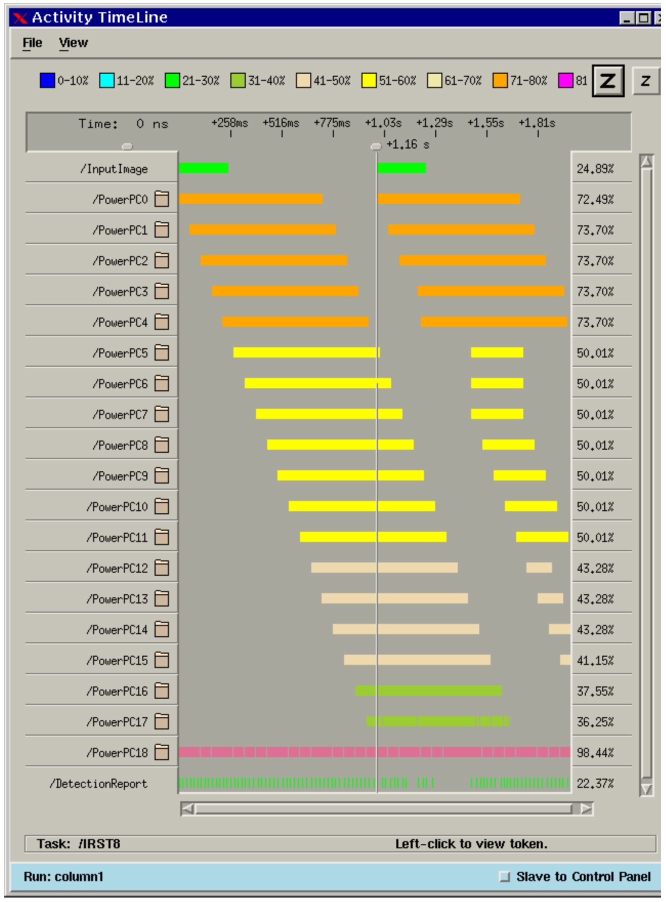


Figure 8: Performance profile of PowerPC partitioned data implementation model

utilization versus time for each processor in the system, revealing a maximum processor utilization of 74%.

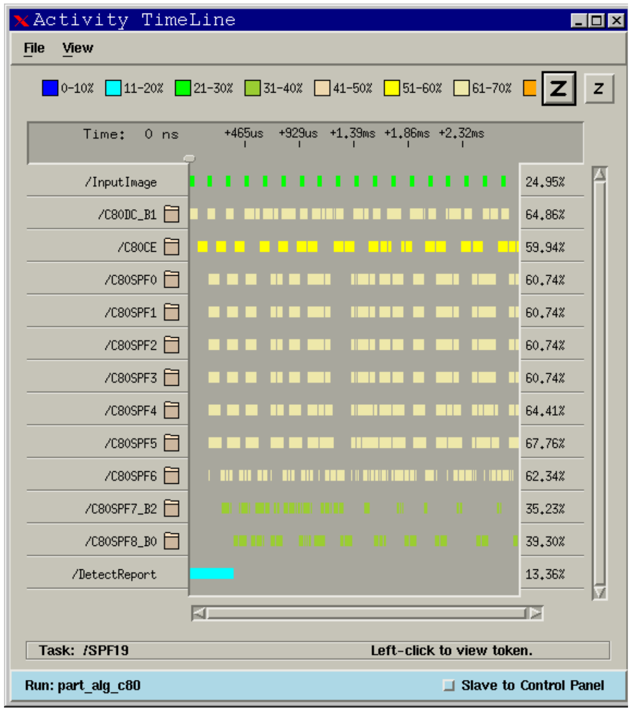
This column tile solution has a number of advantages over the partitioned algorithm solution. First, its bus traffic is low, approximately 1.6 Mwords/second versus 46.4 Mwords/second. Second, software development is easier since the one version of the code is needed instead of portions of the algorithm being scattered over 29 processors in the partitioned algorithm case. Third, this solution uses fewer processors. This will simplify the hardware interconnection and lower the cost of the system. Fourth, this solution results in lower latency since initial columns of the frame arriving at the first tile processor are processed immediately. The partitioned algorithm solution must fill up its eleven stage pipeline before detection reports will be produced. Finally, this solution uses tokens representing the data size of an entire column tile. This results in dramatically fewer tokens in the simulation yielding faster simulation run times.

The comparison between the partitioned algorithm solution and the partitioned data solution seems relatively immune to performance model inaccuracies for two reasons. First, improvements in performance model accuracy will effect both solutions fairly evenly. If a more accurate performance model establishes faster processing times then this will reduce the number of processors necessary in both solutions. Second, interprocessor data traffic for the partitioned algorithm solution is 29 times higher than the partitioned data solution. It is difficult to imagine inaccuracies in this performance model so great as to remove this disadvantage. Medium level performance modeling will more rigorously establish the validity of this 19 PowerPC multiprocessor system solution.

9.2. Performance modeling using TMS320C80-based designs

Because the software task modeling of the 2D IRST algorithm is the same for the PowerPC 604 and C80 cases, the design space exploration will closely mirror that for the PowerPC. The first performance model ran the 2D IRST algorithm on a single C80, indicating that a single C80 can process 209.2 Kpixels/second, which is 12.8% of the total required processing power. Assuming a linear speedup, a minimum of eight C80 processors would be required to perform the task in real-time. As with the PowerPC, the spatial filter task was also the major performance bottleneck.

The second approach attacked this performance bottleneck by partitioning the algorithm. At this point, the similar work done for the PowerPC was leveraged. Since each C80 has more than twice the processing power of the PowerPC, the expectation is that many of the processors would be significantly underutilized. The results confirmed this by revealing processor utilizations ranging from 11.0% through 35.0%, well below the 80% maximum. Two or more tasks were combined onto a single processor where possible, reducing the total number of required processors. In doing this, it was assumed that performing multiple tasks on a single processor that the single task processor utilizations were approximately additive, as long as the total of 80% utilization was not exceeded. In this case, the solution turned out to consist of eleven C80s, as shown in Figure 9. Similar to the PowerPC case, every connection between grouped software tasks (interprocessor) represented the sustained data traffic



/InputImage	24.95%
/C80DC_B1	64.86%
/C80CE	59.94%
/C80SPF0	60.74%
/C80SPF1	60.74%
/C80SPF2	60.74%
/C80SPF3	60.74%
/C80SPF4	64.41%
/C80SPF5	67.76%
/C80SPF6	62.34%
/C80SPF7_B2	35.23%
/C80SPF8_B0	39.30%
/DetectReport	13.36%

Task: /SPF19

Run: part_alg_c80

Figure 9: Performance profile of TMS320C80 partitioned algorithm implementation model

of 1.6 Mwords/second. Thus, the total interprocessor communication was 20.8 Mwords/second.

The column tile approach (data partitioning) yielded a much more efficient solution. The problem is set up identically to the PowerPC solution, but in this case eight processors were necessary to process the frame in real-time. An additional C80 processor was allocated to account for the data distribution handling necessary to distributed the column tiles. The utilization versus time for this system is depicted in Figure 10. The high processor utilizations result from the well balanced distribution of data across the multiprocessor system. The column tile solution had a number of advantages over the partitioned algorithm solution, as in the PowerPC case. However, the advantage in interprocessor communication bandwidth was less over the algorithm partitioned case for the C80 (1.6 Mwords/second versus 20.8 Mwords/second).

10. Verification of Performance Modeling

As a means of validating and verifying the models, the 2D IRST algorithm was implemented in C and targeted to the PowerPC and the C80. In coding the algorithm, only obvious and easy optimizations were made, and no assembly language was used. Individual routines as well as the entire 2D IRST algorithm were timed and profiled. The PowerPC target, using a 100 MHz processor, was measured to process images at 126.2 Kpixels/second, which is somewhat close to the 140.8 Kpixels/second rate predicted by the high profile model. For the TMS320C80, a 40 MHz processor was used, and was found to process data at 391.4 Kpixels/second, compared with 209.2 Kpixels/second for the model. These pre-

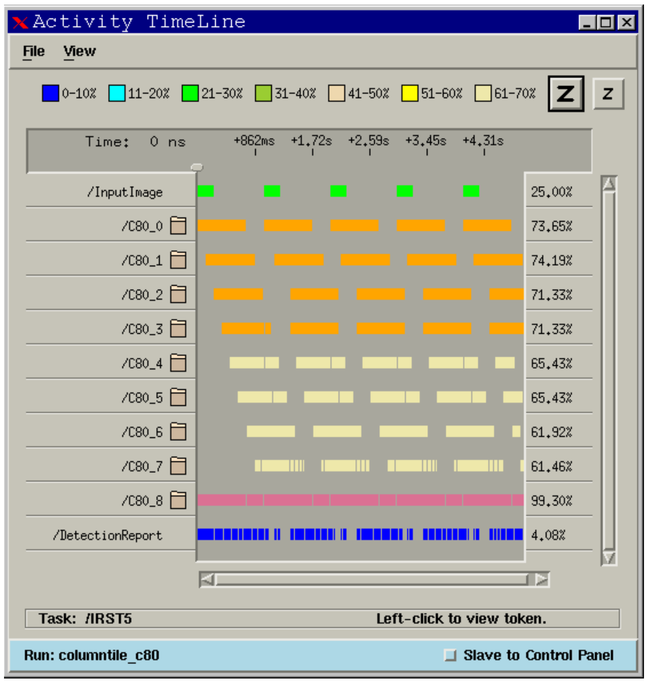


Figure 10: Performance profile of TMS320C80 partitioned data implementation model

dictions are reasonably close for the purpose of the high profile modeling. It is important to keep in mind that the primary goal of high profile modeling is for relative comparison among alternatives, which are investigated more closely in medium and low profile performance modeling, not to obtain absolute predictions. The detailed timing information obtained from these measurements will be useful inputs in the medium and low level modeling, where more accurate predictions of absolute performance are sought.

11. Conclusions and Further Work

In summary, there are several conclusions which the high-level performance models revealed about the PowerPC and C80 2D IRST multiprocessor solutions. Data partitioning via column tiling was more efficient than algorithm partitioning. The column tile models estimated a near linear speed up over the single processor performance models for both the PowerPC and C80 because the partitioned data approach provided a well-balanced workload across the system. Algorithm partitioning did not work as effectively since the algorithm could not be evenly partitioned for distributed execution. Furthermore, 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 1.6 Mwords/second, compared to 46.4 and 20.8 Mwords/second for the PowerPC and C80 solutions respectively. There was some agreement between the high profile performance modeling and the profiling of the C code implementation. The value of the high profile modeling will be more accurately quantified and conclusions on its advantages and drawbacks will be more clear once the medium and low profile performance modeling is complete during the fall, 1997. During that time, the methodology will be tried out on a more complex 3D IRST algorithm which will present a much more challenging modeling problem. Complete results on this performance modeling effort is available on line at http://rassp.sanders.com/legacy.

References

1. T. Egolf, M. Pettigrew, et al., “VHDL-Based Rapid System Prototyping,” *Journal of VLSI Signal Processing*, Vol. 14, 1996, pp. 125-156.
2. C. Hein, T. Carpenter, P. Kalutkiewicz, and V. Madisetti, “RASSP VHDL Modeling Terminology and Taxonomy--Revision 1.0,” *Proceedings of the Second Annual RASSP Conference*, 1995 (http://www.rassp.scra.org)
3. V. Madisetti and J. Debardelaben, “A RASSP Approach to HW/SW Codesign,” *The RASSP Digest*, December, 1995.
4. “IEEE Standard VHDL Language Reference Manual,” IEEE/ANSI Standard 1076-1993, 1994.
5. F. Rose, T. Steeves, and T. Carpenter, “VHDL Performance Models,” *Proceedings of the First Annual RASSP Conference*, 1994 (http://www.rassp.scra.org)
6. T. Carpenter and F. Rose, “A VHDL Performance Modeling Environment,” *Proceedings of VHDL International Users’ Forum*, 1991 (http://www.rassp.scra.org)
7. T. Steeves, F. Rose, et al., “Evaluating Distributed Multiprocessor Designs,” *Proceedings of the Second Annual RASSP Conference*, 1995 (http://www.rassp.scra.org)
8. C. Buenzli and J. Runkel, “Performance Modeling Workbench - A VHDL-Based Hardware/Software Codesign Tool,” *RASSP Digest*, September, 1996.
9. Omniview, Inc., 100 High Tower Blvd., Suite 201, Pittsburgh, PA 15205
10. Campana, S. B., ed., “The Infrared Search and Track Systems,” The Infrared and Electro-Optical Systems Handbook, 8 volumes, Ann Arbor: Infrared Information Analysis Center and SPIE Optical Engineering Press, 5:209-349, 1993.
11. IBM and Motorola, Inc. *PowerPC 604 RISC Microprocessor User’s Manual*, 1994 (http://www.ibm.com or http://www.mot.com)
12. Texas Instruments, Inc., *TMS320C8x System-Level Synopsis*, 1996 (http://www.ti.com).
13. 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)
14. Hennessy, J. and Patterson, D., *Computer Architecture: A Quantitative Approach*, 2nd edition, Morgan Kaufmann Publishers, Inc., San Francisco, 1996, p. 465.
15. Ibid, pp. 731-736.

Acknowledgments

The authors wish to thank Eric Rundquist (Sanders) for his guidance and insight on the 2D IRST algorithm and also Allan Anderson and Gary Shaw (MIT Lincoln Labs) for their useful inputs on the performance modeling methodology and results.