Schedule-Aware Performance Estimation of Communication Architecture for Efficient Design Space Exploration

Sungchan Kim   Chaeseok Im   Soonhoi Ha
The School of Electrical Engineering and Computer Science
Seoul National University
Seoul 151-744, Korea
{ynwie, csim, sha}@iris.snu.ac.kr

ABSTRACT

In this paper, we are concerned about the performance estimation of bus-based architectures assuming that the task partitioning on the processing components is already determined. Since the communication behavior is usually unpredictable due to dynamic bus requests of processing components, bus contention, and so on, simulation based approach seems inevitable for accurate performance estimation. But it is too time consuming to explore the wide design space. To overcome this serious drawback, we propose a static performance estimation method that is based on the queuing model and makes use of memory traces and task execution schedule information. We propose to use this static estimation approach to prune the design space drastically before applying a simulation-based approach. Comparison with trace-driven simulation results proves the validity of our static estimation technique.

Categories and Subject Descriptors

B.8.2 [**Hardware**]: Performance and Reliability – *Performance Analysis and Design Aids.*

General Terms

Design, Performance

Keywords

Performance estimation, design space exploration, communication architecture, queuing theory

1. INTRODUCTION

In system level design, selection of appropriate communication architecture is a critical design step. In this paper we assume that the system behavior is modeled as a composition of function blocks. Differentiation between functions and communication allows the system designer to explore the communication architecture independently of component selection. Then, communication architecture selection is performed after a decision is made on which processing components are used and which function blocks are mapped on which processing components. And a task is defined as a group of function blocks sequentially scheduled on a processing component.

The key tool for exploring the design space of communication architecture is to estimate the performance of communication architectures, which is the main theme of this paper. In particular, we are concerned about the bus-based architectures since the bus-based architecture is most widely adopted due to its simplicity and popularity. From the mapping result of function blocks to the processing elements, we obtain the bus requests from the processing elements and those bus requests are input data to the performance estimation tool. Unlike the previous work similar to ours [8], we take into account the bus requests for the local memory accesses as well as for the shared memory access of inter-component communication.

Since the communication behavior is usually unpredictable due to dynamic trace of memory accesses, bus contention, and so on, simulation based approach seems inevitable for accurate performance estimation. However, since the memory traces of processing elements are enormous, trace driven simulation needs a huge data storage for simulating only a few seconds of execution of application. Some commercial tools and research tools use execution-driven simulation where the simulator executes the tasks to produce the bus requests on-line which are passed to the module that accurately models the communication architecture. But the main drawback of a simulation-based approach is that it is too slow to be used for exploring the wide design space of architectures.

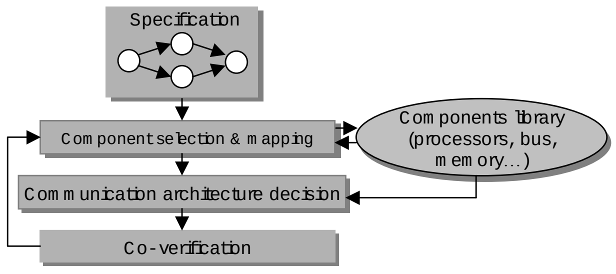
Even after a specific bus standard is chosen to be used, the design space can still be huge if a single bus segment cannot accommodate all memory requests within a given time budget. For example, we need to determine how many bus segments are used with what topology and which processing elements and memory banks are mapped to which bus segments. We also have to decide the memory types and memory system configurations. If we include the selection of bus operation frequency and arbitration policy, the design space explodes.

To explore the wide design space, we need a fast and accurate performance estimation method while the simulation-based approach is too slow. To overcome this difficulty, we propose a two-step communication architecture exploration scheme. We propose a static performance estimation method that is based on the queuing model and makes use of memory traces and task execution schedule information. We use this static estimation approach to prune the design space before applying a simulation-based estimation. As the static estimation gets more accurate, the design space to be explored by simulation would be narrower. The key contribution of this paper is to propose an accurate static estimation method taking into account of all dynamic behaviors due to bus contention and dynamic memory traces. We show the effectiveness of our method in terms of estimated time accuracy of entire execution compared with the results of trace-driven simulation.

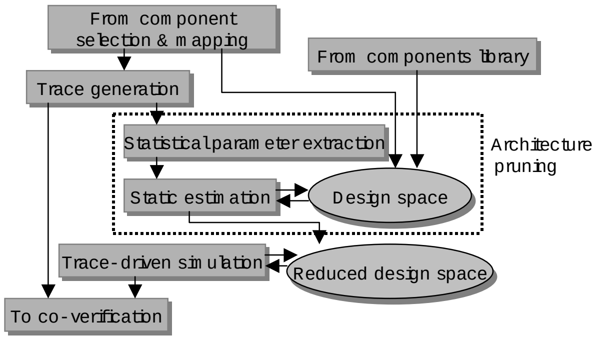
The remainder of this paper is organized as follows. In section 2 we state our proposed design space exploration framework and outline the performance estimation techniques. Section 3 reviews some related works. In section 4 the queuing model for a priority-based single bus is explained. Section 5 contains the detailed discussion on the estimation method considering a task schedule and memory traces and on the extension to multiple bus systems. In section 6 we give some experimental results and draw conclusions in section 7.

2. PROPOSED DESIGN SPACE EXPLORATION FRAMEWORK

The proposed design space exploration procedure, depicted in Figure 1(a), assumes that a system behavior is modeled as a composition of function blocks. We do not assume any specific computing model for block composition as long as the system behavior is defined by a non-preemptive execution sequence of function blocks. The only restriction is that the execution order of function blocks is known a priori. Dataflow specification for digital signal processing (DSP) application is a well-known example that meets the restriction.


Specification
Component selection & mapping
Components library (processors, bus, memory...)
Communication architecture decision
Co-verification

(a)



(b)

Figure 1. (a) The proposed design space exploration procedure and (b) the procedure of communication architecture exploration

From the functional specification of system behavior, we first perform platform and component selection and mapping of the function blocks to the processing components. Next, the design space of communication architectures is explored as shown in Figure 1(b). After mapping is completed, we obtain the memory access traces from all processing elements using component simulators: an instruction set simulator for a processor core, HDL simulator for ASIC parts, or IP simulators. In the trace information, we distinguish three different memory access types: code memory, local data memory, and shared memory accesses. We assume that the design space of communication architecture includes memory configurations and memory assignments. Therefore, we include the local memory accesses in the trace information. A memory trace consists of three fields: (processing component, memory access types, time stamp). When we obtain the time stamp, we assume that there is no resource contention, and all memory accesses are finished in a clock cycle.

From the traces, we extract some statistical parameter values to be used in the proposed static estimation method based on the queuing model. Using those parameter values, the static performance estimation of communication architectures is performed to prune the design space. How to prune the design space is beyond the scope of this paper. In the static estimation step, we utilize the task execution schedule on the processing components. The schedule information is used to determine which processing components participate in bus contention at a given instance of time. Finally, trace driven simulation is performed with a reduced set of candidate communication architecture.

3. RELATED WORK

Some researchers have considered communication architecture selection simultaneously during the synthesis of computation parts of system and mapping step. Since the communication overhead is needed for the mapping decision, static estimation of communication architecture has been investigated. A technique was proposed to estimate the communication delay using the worst-case response analysis of real-time scheduling [6]. Knudsen and Madsen estimated the communication overhead taking into account the data transfer rate variation depending on the protocol, configuration, and different operating frequencies of components [4][5]. However these techniques do not model the dynamic effects such as bus contention and explore only a limited configuration space.

While simulation-based performance estimation is proposed [2][3] and also widely adopted in the commercial tools at the transaction level [9][10] or at the pin-level [11], there have been proposed only a few static estimation techniques. One is to synthesize the bus architecture with a given and fixed bus request information [7]. But this approach is not applicable in the general case where memory request times are non-deterministic and bus contention exists. A hybrid approach between a static estimation and a simulation approach has been developed by Lahiri et. al. [8]. In their work, communication and computation segments are grouped to make the BSE(Bus and Synchronization Event) graph from the trace data obtained after system cosimulation. They focused on inter-component communication activities that are usually localized in time at the boundary of computation segment. The trace groups are scheduled on the communication media, being shifted by the estimated delays considering the resource contention. They use some static analysis to group the traces and apply a trace-driven simulation with the trace groups. Their approach is similar to ours in that they apply some static analysis to the memory traces to reduce the time complexity of trace-driven simulation. But the overall approach is different from ours. Moreover they do not consider the local memory accesses. If they consider the local memory accesses, the BSE graph size becomes prohibitively huge.

Our work is inspired by the work in [1] where a simple queuing model of SCSI bus is proposed and the model produces remarkable results comparing with the simulation results. Since the communication behavior of a processor-memory bus is quite different from an I/O bus, their approach can not be directly applicable. Instead, we made several extensions and improved the estimation accuracy significantly.

4. SIMPLE MODEL OF A SINGLE BUS

4.1 Base Queuing Model For A Single Bus

A base estimation technique using a queuing model for a single bus is explained in this section. The basic idea and notations used here are borrowed from [1]. There are N components, numbered 0 though N-1, competing for the use of the bus. It is assumed that the bus arbitration is based on the fixed priorities of components. Device 0 has the highest priority. The bus access is assumed non-preemptive. Figure 2 shows the queuing model of a single bus architecture. λ_i denotes the rate at which component i issues memory request. It is computed as the ratio between the memory access counts and the scheduled length of execution. If the execution time is lengthened due to bus contention, the effective arrival rate of request becomes smaller than λ_i. We denote the actual memory access rate by θ_i that is actually seen on the bus. The mean service time of a server for the request of component i is denoted by $1/\mu_i$. Let $\bar{k}_i$ be the expected number of requests from component i waiting for the bus. It is within the range of [0,1] if the component does not issue the next memory request until the current request is served. And we denote $\overline{w}_i$ as the expected waiting time of the stalled request. Then, we can obtain the following equation:

$$\theta_i = (1 - \bar{k}_i - u_i)\lambda_i \,, \tag{1}$$

where $u_i = \theta_i / \mu_i$ is the bus utilization factor by device i. Little's law shows

$$\overline{w}_i = \bar{k}_i / \theta_i \,. \tag{2}$$

From this equation, we want to obtain θ_i which indicates the delays incurred from contention on the bus. We can extract the parameters λ_i and μ_i from the memory traces. There is an unknown parameter $\bar{k}_i$. To obtain this parameter, we use the state transition diagram and steady-state probability.

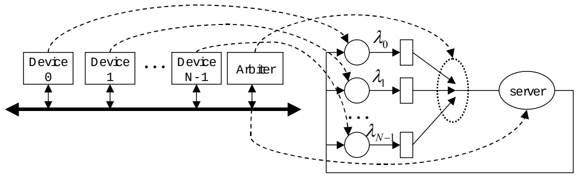


Figure 2. Queuing modeling of a single bus

We draw a state transition diagram from the viewpoint of component i (i=1,…,N-2). We define the system state as a quadruple and its steady state probability by $p(n_i,n_h,n_l,s)$, where n_i is the number of requests from device i (n_i=0,1), n_h(n_h=0,...,i-1) the number of requests from components of higher priority than i, n_l (n_l=i+1,…,N-1) the number of requests from components of lower priority than i, and s is the priority of component that is using the bus currently.

State (n_i,n_h,n_l,s) moves to (n_i,n_h+1,n_l,s) if another request arrives from a component of higher priority and its transition rate is approximated by α_h, $\alpha_h \approx (N_h - n_h)\gamma_h$, where $N_h = i$ is the total number of higher priority components and γ_h is the average rate of arrivals for higher priority requests. γ_h can be computed as follows by summing up the equation (1) from component 0 to component i-1.

$$\sum_{j=0}^{i-1}\theta_j = \left\{ N_h - \sum_{j=0}^{i-1}(\bar{k}_j + u_j) \right\}\gamma_h \,. \tag{3}$$

The approximate transition rate β_h to state (n_i,n_h-1,n_l,s) is approximated as follows:

$$\beta_h \approx \sum_{j=0}^{i-1}\left(\mu_j \cdot u_j \Big/ \sum_{k=0}^{i-1} u_k \right), \tag{4}$$

The similar formulations can be given to the lower priority requests. The readers are referred to [1] for more details about derivation of α_h, β_h, α_l, and β_l respectively. From the state transition diagram and the transition rates, we can compute the steady-state probability $p(n_i,n_h,n_l,s)$ using an additional requirement of $\sum_{n_i=0}^{l}\sum_{n_h=0}^{N_h}\sum_{n_l=0}^{N_l}\sum_{S} p(n_i,n_h,n_l,s) = 1$. After all steady-state probabilities are computed, we can compute the expected number of waiting requests $\bar{k}_i$ by summing up the probabilities of a certain set of states as follows:

$$\sum_{n_h=0}^{N_h}\sum_{n_l=0}^{N_l}\sum_{S} p(1,n_h,n_l,s) = u_i + \bar{k}_i \,. \tag{5}$$

Note that evaluation of $p(n_i,n_h,n_l,s)$ needs $\bar{k}_i$ by (3) and (4). Therefore, these equations can be solved by an iterative method until $\bar{k}_i$ becomes stable. And, this iterative procedure to get $\bar{k}_i$ is repeated for each priority level i (i=1,…,N-2). We used an ILP package to solve the equations and found that the solutions are converged within less than 10 iterations in all cases, which takes much less than trace-driven simulation.

4.2 Extension To Simultaneous Events

The base queuing model of the previous section models a continuous system where the bus request can be served at any instance of time. In reality, however, bus arbitration is performed at discrete sampling points (i.e. clock edges) among all bus requests accumulated so far. If we assume that there occurs only one event, either request arrival or service completion, in each clock period, the base queuing model may be used as an approximate model. This assumption is suitable for the I/O bus case where bus requests are infrequent and service time is relatively large. But in our case, several events are very likely to occur during a single clock period. Thus we modify the model to allow simultaneous events. And, the state transition rate of a transition arc should be replaced by the state transition probability within a clock period.

The number of possible transitions from a state grows exponentially as the number of simultaneous events increases. Thus, we make a compromise between the modeling complexity and the modeling accuracy. The compromised model allows up to two simultaneous events. And an event is defined by a single increment or decrement of a coordinate of a state (n_i,n_h,n_l,s). Table 1 shows all possible state transitions whose total number is 15. Type 1 transition indicates the case that only one event occurs while type 2 transition indicates that two events arrive during the next clock cycle. For the computation of the transition probabilities, we introduce two new parameters d_h and d_l as the probabilities that at least one component issues a new request among high priority components and low priority components respectively:

$$d_h = 1-(1-\gamma_h)^{N_h-n_h},\ d_l = 1-(1-\gamma_l)^{N_l-n_l}\ . \qquad (6)$$

And let S the service rate of component that is using the bus currently. For instance, the transition probability from state (n_i,n_h,n_l,s) to (n_i,n_h+1,n_l+1,s) becomes $(1-\lambda_i)d_hd_l(1-S)$. Note that this is a rough approximation of the state transition probability. Even though more than two requests may arrive from the higher-priority components, we only increase the number n_k by 1.

We performed a preliminary experiment to compare the accuracy of the base model and that of the modified model. In this experiment, the number of processing components is set to 10 and burst size of memory access is selected randomly. Memory access cycle is assumed one. About 36% of entire execution time is contributed by memory accesses when arrival rate λ is 0.1 and 67% when λ is 0.2. As shown in Table 2, the proposed modification improves the accuracy by 25.5% on average for maximum estimation error and 29.1% on average for average estimation error. We guess that more complicated modeling of state transitions would result in more accurate estimation.

Table 1. State transitions from state (n_i,n_h,n_l,s) to other states.

Type 1		Type 2	
Next state	Transition rate	Next state	Transition rate
$(n_i+1,n_h,n_l,\mathrm{s})$	$\lambda_i(1-d_h)(1-d_l)(1-S)$	$(n_i+1,n_h+1,n_l,\mathrm{s})$	$\lambda_id_h(1-d_l)(1-S)$
$(n_i,n_h+1,n_l,\mathrm{s})$	$(1-\lambda_i)d_h(1-d_l)(1-S)$	$(n_i,n_h+1,n_l+1,\mathrm{s})$	$(1-\lambda_i)d_hd_l(1-S)$
$(n_i,n_h,n_l+1,\mathrm{s})$	$(1-\lambda_i)(1-d_h)d_l(1-S)$	$(n_i+1,n_h,n_l+1,\mathrm{s})$	$\lambda_i(1-d_h)d_l(1-S)$
$(n_i-1,n_h,n_l,\mathrm{s})$	$\mu_i(1-d_h)(1-d_l)$	$(n_i-1,n_h+1,n_l,\mathrm{s})$	$\mu_id_h(1-d_l)$
$(n_i,n_h-1,n_l,\mathrm{s})$	$(1-\lambda_i)\beta_h(1-d_l)$	$(n_i-1,n_h,n_l+1,\mathrm{s})$	$\mu_i(1-d_h)d_l$
$(n_i,n_h,n_l-1,\mathrm{s})$	$(1-\lambda_i)(1-d_h)\beta_l$	$(n_i+1,n_h-1,n_l,\mathrm{s})$	$\lambda_i\beta_h(1-d_l)$
		$(n_i,n_h-1,n_l+1,\mathrm{s})$	$(1-\lambda_i)\beta_hd_l$
		$(n_i+1,n_h,n_l-1,\mathrm{s})$	$\lambda_i(1-d_h)\beta_l$
		$(n_i,n_h+1,n_l-1,\mathrm{s})$	$(1-\lambda_i)d_h\beta_l$

Table 2. Performance comparison of the base queuing model and the proposed queuing model

Arrival rate λ	0.1			0.2		
Error (%)	Base	Proposed	Improved	Base	Proposed	Improved
Minimum	0.73	0.21	71.22	2.40	2.41	-0.18
Maximum	16.02	12.14	24.19	16.73	12.25	26.75
Average	7.01	4.20	40.05	6.90	5.65	18.19

5. STATIC ESTIMATION USING SCHEDULE INFORMATION

5.1 Estimation Of A Single Bus

This section explains how task schedule information is used in our estimation method. A simple statistical modeling of bus requests from a processing component assumes that the bus requests are distributed throughout the whole execution duration of application. This assumption is one of the main sources of inaccuracy of simple static statistical modeling. The schedule information, on the other hand, discloses the active durations of bus requests as shown in Figure 3 (a): for example, it shows that during time 2 and 4, PE2 does not generate any bus request.

Therefore, we divide the time slots in such a way that all processing components maintain their bus request patterns during the time slot. And we apply the proposed queuing model analysis in each time slot starting from the beginning of the schedule. Figure 3 displays how the proposed scheme progresses. The shaded region indicates the remaining sections for static estimation. First, we perform the proposed queuing analysis during time 0 to 2 until PE2 finishes its execution (Figure 3(a)). Note that the schedule of Figure 3(b) is different from Figure 3(a) since the schedule of the first time slot is updated taking into account the bus contention. In the next time slot, we have to re-compute the statistical parameters for all processing components involved in bus contention.

The overall algorithm of the estimation technique for a single bus is described in Figure 4. The procedure ***Estimate_single_bus*** has 5 inputs. **B** represents a data structure of the single bus. **MT** and **ORI_SCHED** are the memory traces of all processing elements and the original schedule. **PL** is the set of processing components and **FL** is the set of tasks to be used for static estimation. The output, **EVAL_SCHED**, of this procedure is the updated schedule after considering all bus conflicts and other overheads if any.

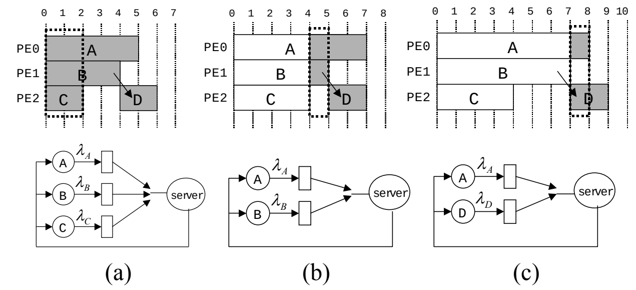


Figure 3. (a) Initial schedule of PE0, PE1, and PE2, and corresponding queuing system, (b) new queuing system of PE0 and PE1 after c finished, and (c) another queuing system of PE0 and PE2 in the last time slot

```
Estimate_single_bus
inputs: B, MT, ORI_SCHED, PL, FL
outputs: SCHEDULE EVAL_SCHED
begin
  EVAL_SCHED = ORI_SCHED;
  cur_time = 0;
  for each pe ¡ ôPL
    pe.cur_fb=NULL;
    pe.cur_fb_done=FALSE;
  end for
  while (FL not empty)
    for each pe ¡ ôPL
      if (pe.cur_fb_done==TRUE or pe.cur_fb==NULL)
        pe.cur_fb = get_current_fb(EVAL_SCHED, PL, FL);
        stat_params = get_stat_params(ORI_SCHED, PL, FL, MT);
        if (pe.fb_start_time < cur_time)
          cur_time = pe.fb_start_time;
        end if
      end if
    end for
    cur_time = estimate_end_time(B,PL,EVAL_SCHED,stat_params);
    update_schedule(EVAL_SCHED, cur_time);
  end while
end
```

Figure 4. Schedule-aware performance estimation algorithm for a single bus

The main procedure of queuing analysis is ***estimate_end_time.*** Before the queuing analysis, we first determine which processing components request the bus by calling the ***get_current_fb*** procedure and compute the statistical parameters by calling the ***get_stat_params*** procedure. We compute two statistical parameters of each task from the **MT** memory trace: they are the memory access rate, θ, and the mean service time, $\overline{S}$. The memory access rate $\theta_{FB,PE}$ is computed by the following formula:

$$\theta_{FB,PE} = M_{FB,PE} \;/\; EXE_{FB,PE} \;, \qquad (7)$$

where $M_{FB,PE}$ is a total memory access count during the execution of task FB and $EXE_{FB,PE}$ is the task execution time on PE. When computing the mean service time, we consider the different burst transfer size according to the memory access type. For example, code memory access is usually a burst access of which the size equals to the cache line size, whereas data memory access may have various burst lengths for write operation and the cache line size for read. Then the mean service time $\overline{S}_{FB,PE}$ is computed as

$$\overline{S}_{FB,PE} = \sum_{i \in \{all\_access\_type\}} \left(\sum_{j \in \{all\_burst\_type\}} S_{i,j,PE} \cdot \frac{M_{i,j,FB,PE}}{M_{FB,PE}} \right), \qquad (8)$$

where $S_{i,j,PE}$, $M_{i,j,FB,PE}$ are the service time (including the bus overhead as well as the memory access time) and the number of access counts of burst type j which belongs to memory access type i when task FB runs on processing component PE.

Final procedure ***update_schedule*** modifies the original schedule to obtain the updated schedule, **EVAL_SCHED,** after the current time slot. From the updated schedule, we define the next time slot and go back to the main iteration body until all tasks are considered.

5.2 Extension to Multiple Bus System

Communications across buses are achieved via bus bridge in a multiple bus system. A bus bridge plays both roles of a processing component and a memory system as displayed in Figure 5. We assume for simple analysis that no communication passes through more than 3 buses. Figure 5 shows how the bridge is modeled when a processing component on the *src* bus accesses a memory on the *dest* bus. The request rate of the component λ_{src} is reflected to the *dest* bus by the request rate λ_{dest} of the bridge. To compute λ_{dest} we have to know the mean waiting time, w_{src}, of the processing component on the *src* bus. At the same time, the bus bridge looks like a memory from the *src* bus point of view. The service rate μ_{src} of the bus bridge should be computed. Let w_{dest} be the mean waiting time of the bridge on the *dest* bus. Then, λ_{dest} and μ_{src} are computed as follows:

$$\frac{1}{\lambda_{dest}} = \frac{1}{\lambda_{src}} + w_{src} + O_{bridge} \;, \qquad \frac{1}{\mu_{src}} = w_{dest} + O_{bridge} + \frac{1}{\mu_{dest}} \;, \qquad (9)$$

where O_{bridge} is overhead time at bus bridge. On the other hand, w_{src} and w_{dest} are obtained from our static estimation technique after λ_{dest} and μ_{src} computation. Therefore the static estimation and bridge modeling are performed iteratively until all parameters become stable.

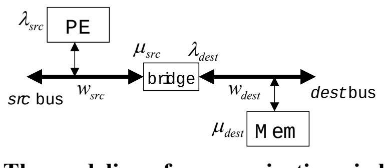


Figure 5. The modeling of communication via bus bridge

6. EXPREIMENTS

To investigate the accuracy of the proposed static estimation method over a wide variety of working conditions, we first perform the experiments on a single bus varying the number of processors, bus request rates, bus service rates (or memory access times), and burst lengths. We generate the memory traces from each processing component following a Poisson distribution (i.e. exponentially distributed inter-arrival times between bus requests). We compare the estimation results with those obtained from trace driven simulation. Our trace driven simulator adjusts the time stamps of trace data by accurately modeling the communication architecture that includes buses and memories. More precisely, in these experiments, our bus model consists of 4 phases: bus-arbitration, start-address-drive, sequential-burst-transfer, last-data-drive.

Figure 6 shows the accuracy results on the varying number of processing elements (PEs=2,4,6,10) according to the various bus request rates (λ) and the various memory access times. We display the range of estimation error with a bar graph, which indicates the minimum and the maximum errors of completion times of processing components. The scheduled execution time without bus contention is set to 10,000 cycles.

The burst size is randomly selected. In case that λ is 0.1 and memory access time is 1 cycle, the portion of the total memory access time over the entire execution time is about 0.36. This is similar to the case when we run an H.263 encoding algorithm, which we will discuss later.

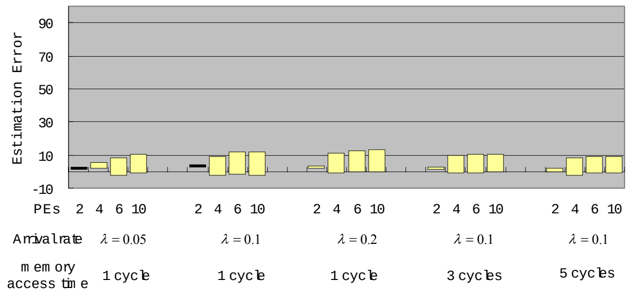


Figure 6. Estimation accuracy according to the various request rates and the various memory access times

In all cases, the estimation error of the proposed method does not exceed 13%. We also notice that the estimation results are greater than the simulation results: underestimation errors are below at most –3.0%. Three experiments show the robust performance of our estimation technique on various architecture configurations.

It is questionable whether there is a simpler static estimation method that is reasonably accurate. So, we devised an intuitive estimation method on the waiting time of processing elements i, w_i, by bus contention, for a single bus that has N processing elements, as shown in formula (10).

$$w_i = \frac{\sum_{j=0}^{i-1} m_j}{exe_i} \cdot \frac{ts_i}{2} \cdot (i-1) + \frac{\sum_{j=i+1}^{N-1} m_j}{exe_i} \cdot \frac{ts_i}{2} \;, \qquad (10)$$

where m_j, exe_j, and ts_j are the total memory access times, the total execution time, and the mean service time of component j respectively. At the right side of equation (10), the first term is the expected waiting time due to the bus requests from higher priority components and the second term is the waiting time due to the current outstanding request of a low priority component. Table 3 shows the accuracy comparison between this intuitive method and the proposed method assuming that λ is 0.2, the memory access time is 3 cycles, and the burst size is 4. The results of intuitive method get worse rapidly according to the increase of PEs whereas the proposed method keeps its error range of estimation within 10%.

Table 4 shows the run time of the proposed static estimation and the maximum and the minimum number of states varying the number of PEs on Xeon 1.8GHz. We used a commercial ILP package to solve the linear equations of the proposed method [12]. For each processing component i, the number of states $\{(n_i,n_h,n_p,s)\}$ is $O(N^2)$ where N is the total number of components. The number of states depends on the priority of the component of interest. Since the time complexity of the LP(linear program) solver is pseudo-polynomial, the overall time complexity is also pseudo-polynomial. Table 4 confirms this fact that the proposed technique has acceptable complexity for design space exploration.

Table 3. Comparison with intuitive estimation method

PEs	2		4		6		10	
Method	int	pro	int	pro	int	Pro	int	pro
Min. Error(%)	16.72	2.06	9.78	1.32	5.15	0.00	46.63	0.31
Max. Error(%)	17.32	2.66	18.91	9.79	84.31	9.90	232.98	9.95
Avg. Error(%)	17.02	2.36	15.16	4.86	43.75	4.07	121.71	3.47

Table 4. Complexity of the proposed estimation technique

PEs	CPU time (sec)	Min. Number of states	Max. Number of states
2	0.03	5	5
4	0.07	21	21
6	0.33	37	47
10	2.94	69	129

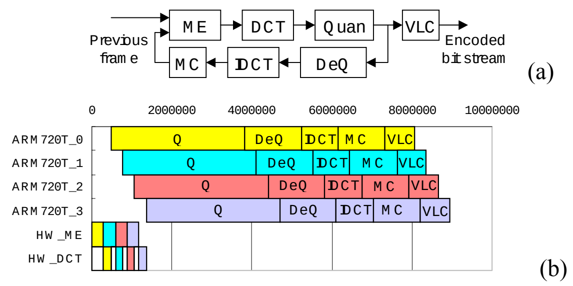


Figure 7. (a) Specification of an H.263 encoding algorithm, (b) the initial schedule of 4-channel digital video recorder (DVR)

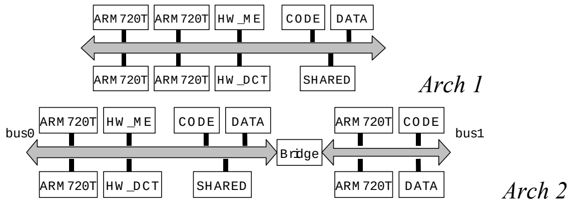


Figure 8. Alternative communication architectures

Finally, we validate our proposed technique by applying it to a practical example, 4-channel digital video recorder (DVR). DVR receives the raw bit streams from external 4 sources and encodes each stream separately using H.263 encoding algorithm. Figure 7(a) shows the specification of H.263 encoding algorithm. All function blocks of the H.263 encoder except ME and DCT blocks are mapped to one ARM720T. And all MEs and DCTs are mapped to a motion estimation (ME) hardware block and a discrete cosine transform (DCT) hardware block respectively so that four H.263 encoders share two hardware blocks. Therefore we construct the initial schedule of each function block like Figure 7(b). Also two communication architectures are chosen: a single bus architecture *Arch 1* and two bus, one bridge architecture *Arch 2* in Figure 8. In *Arch 2*, two ARM720Ts in bus1 access shared memory in bus0 to communicate with HW_ME and HW_DCT. A comparison of estimation results for two communication architectures with trace-driven simulation are shown in Table 5. Errors of overall execution time estimation are still within 10% in both of two architectures. Ideal execution time of *Arch 2* is larger than *Arch 1* due to bus bridge overhead. The real execution time gets larger in *Arch 2* by just 14% due to the concurrency of memory accesses in two buses whereas *Arch 1* extends its real execution time by 85% due to serious bus contentions. Consequently *Arch 2* is faster than *Arch 1* by about 30%.

Table 5. Experimental results about DVR

Architecture candidates	Arch 1	Arch 2
Schedule	8949897	8949897
Ideal	13427568	13541616
Simulation	24871433	15402311
Estimation	23559707	16473155
Error (%)	-5.27	6.95

7. CONCLUSION

In this paper, we presented an efficient static estimation technique of fixed-priority based communication architecture. It is based on the queuing model and makes use of the schedule information and memory traces. Since the estimation time is polynomial to the number of processing components, the number of tasks, and the number of bus segments, the proposed technique can be used to prune the large design space before trace-driven simulation. Experimental results show our proposed technique performs well providing suitable estimation error, at most 10% to 15%, compared with trace driven simulation in various communication architecture configurations.

Future researches will be focused on the development of design space exploration algorithm and the modeling of other bus arbitration schemes such as round robin and TDMA.

8. ACKNOLWEDGMENT

This work was supported by National Research Laboratory Program(number M1-0104-00-0015) and Brain Korea 21 Project. The RIACT at Seoul National University provided research facilities for this study.

9. REFERENCES

[1] A. Brandwajn, "A note on SCSI bus waits", ACM SIGMETRICS Performance Evaluation Review Vol.30, Issue 2, pp.41-47, Sept. 2002.

[2] P. Lieverse, P. V. Wolf, E. Deprettere and K. Vissers, "A methodology for architecture exploration of heterogeneous signal processing systems", IEEE Workshop on Signal Processing Systems, pp.181-190, 1999.

[3] J. A. Rowson and A. Sangiovanni-Vincentelli, "Interface based design", Design Automation Conf., pp.178-183, June. 1997.

[4] P. V. Knudsen and J. Madsen, "Communication estimation for hardware/software codesign", Int. Symp. on Hardware/Software Codesign, pp.55-59, 1998.

[5] P. Knudsen and J. Madsen, "Integrating communication protocol selection with partitioning in hardware/software codesign", Int. Symp. on System Level Synthesis, pp.150-155, Sept.1998.

[6] T. Yen and W. Wolf, "Communication synthesis for distributed embedded systems", Int. Conf. on Computer-Aided Design, pp.437-444, Nov.1995.

[7] M. Gasteier and M. Glesner, "Bus-based communication synthesis on system level", ACM Tran. on Design Automation Electronic Systems, pp.1-11, Jan.1999.

[8] K. Lahiri, A. Raghunathan, and S. Dey, "Fast performance analysis of bus-based system-on-chip communication architectures", Int. Conf. on Computer Aided Design, pp.590-597, Nov. 1999.

[9] Synopsys CoCentric System Studio, http://www.synopsys.com/products/ cocentric_studio/cocentric_studio.html

[10] CoWare N2C Design System, http://www.coware.com/cowareN2C.html

[11] Mentor Graphics SeamlessCVE, http://www.mentorg.com/seamless/

[12] ILOG CPLEX, http://www.ilog.com/products/cplex/