In,
International Conference on Acoustic, Speech and Signal Processing (ICASSP 2000).
pages 3354-3357, Istanbul, Turkey, June 2000.

A NEW APPROACH TO MODEL COMMUNICATION FOR MAPPING AND SCHEDULING DSP-APPLICATIONS

Claudia Mathis, Bernhard Rinner, Martin Schmid, Reinhard Schneider and Reinhold Weiss

Institute for Technical Informatics
Technical University Graz, AUSTRIA

ABSTRACT

We present a novel approach to model inter-processor communication in multi-DSP systems. In most multi-DSP systems, inter-processor communication is realized by transferring data over point-to-point links with hardware FIFO buffers. Direct memory access (DMA) is additionally used to concurrently transfer data to the FIFO buffers and perform computation. Our model accounts for the limited size of the communication buffers as well as concurrent DMA transfer.

This novel communication model is applied in our rapid prototyping environment for optimizing multi-DSP systems. Given an extended data flow graph of the DSP application and a description of the target multi-processor system, our rapid prototyping environment automatically maps the DSP application onto the multi-processor system and generates a schedule for each processor.

keywords: communication model; mapping and scheduling; multi-DSP; rapid prototyping

1. INTRODUCTION

Mapping and scheduling are key elements for rapid prototyping in embedded systems and digital signal processing (DSP) as well as codesign [8]. Mapping and scheduling of tasks onto multi-processor systems requires the estimation of computation and communication times. We propose a model for buffered inter-processor communication. This model accounts for the limited size of communication buffers as well as direct memory access (DMA) for inter-processor data transfer, all of which is important for mapping and scheduling DSP applications onto multi-DSP systems. This communication model results in a more accurate prediction of the inter-processor communication times and it is applied in our rapid prototyping environment for optimizing DSP systems [7].

Related research on design automation for distributed real-time systems uses different models and strategies to solve the mapping and scheduling problem. Tindell et al. [6] consider the most important parameters for hard real-time systems such as task period, worst-case execution time, memory requirement and replica tasks. However, their simple token-based communication model is not well suited for DSP systems. Beck and Siewiorek [1] refine this model for bus-based communication. Since they only consider synchronous communication the model is also difficult to apply to DSP. Burns et al. [3] use an asynchronous communication model based on dual-ported RAMs for a distributed system with a point-to-point communication structure.

In order to improve the accuracy of communication time prediction, we model inter-processor communication in more detail. In the remainder of this paper, we briefly sketch our rapid prototyping environment for optimized multi-DSP systems and focus on our model for buffered communication. A small example demonstrates the applicability of our communication model for mapping and scheduling in multi-DSP systems.

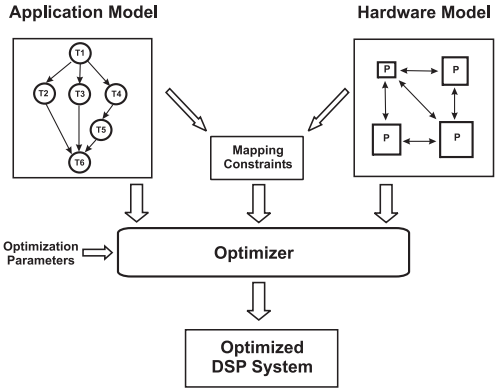

Figure 1: Overall architecture of our prototyping environment for optimized multi-DSP systems.

2. PROTOTYPING OPTIMIZED MULTI-DSP SYSTEMS

Figure 1 presents the overall architecture of our prototyping environment for optimized multi-DSP systems. The goal of this environment is to automatically map a DSP application onto a multi-processor system and to generate a schedule for each processor. This mapping and scheduling is approximated by a heuristic optimizer. Two models serve as the primary input to the optimizer. The application model describes the overall DSP application by means of tasks and dependencies between them. The hardware model describes the multi-processors system onto which the DSP application is mapped. Mapping constraints between application and hardware model may be specified and serve as an optional input to the optimizer.

Authors in alphabetical order

In the following, we present only parts of the prototyping environment relevant to the communication model.

2.1. Application Model

Our design tool is tailored for real-time DSP applications. Usually, a DSP application is decomposed into smaller tasks with dependencies. The dependencies between the tasks are due to data transfer. Most DSP real-time applications have the following characteristics. First, DSP applications are *cyclic*, i.e., their tasks have to be executed periodically. Second, tasks have *precedence relations*, i.e., tasks can only be initiated when all required input data are available. Finally, the tasks have to meet strict *timing constraints* (deadlines).

Our application model is based on a *dataflow graph* [2] – a representation frequently used to model DSP applications. The nodes of the graph represent the individual tasks, the arcs between nodes represent data transfer. Each node receives input data, performs some data processing, and sends output data to other tasks.

We add supplementary information to the simple data flow graph to better characterize DSP applications with limited resources. Thus, each node is assigned with a maximum task execution time C_T and the required memory needs m_T for code and data of that task. The bus usage f_{bu} represents the percentage of instructions requiring bus access of each task. The bus usage allows to estimate the effect of bus conflicts during DMA transfer. Data transfer between tasks is specified by a sender task T_i, a receiver task T_j and the amount of data d_{ij} transferred.

2.2. Hardware Model

Multi-processor systems with distributed memory are the target platforms for our design tool. Such multi-processor systems may consist of different processing elements with different communication links. Thus, the hardware model must be flexible enough to express these heterogeneous architectures.

In our hardware model, each processing element is characterized by its execution speed K_p and the amount of local memory m_p. Physical point-to-point connections are described by the features of the communication interfaces of the connected processing elements. A communication interface is represented by its transmission mode (uni- or bidirectional) c_m, the size of input and output buffer (B_r and B_s), as well as the initialization times (t_{ir} and t_{is}) and transfer rates (K_r and K_s) for reading (receiving) and writing (sending) from and to the corresponding hardware buffers.

Communication using DMA transfer is modeled by the initialization time of the DMA coprocessor t_{DMA} and the bus access priority p_{DMA} to resolve bus conflicts. Access priority for the common bus may be given permanently to either the DMA coprocessor or the processor or it may alternate between them.

2.3. Mapping Constraints

In general, the optimizer does not exclude any mapping of tasks onto processing elements a priori. Each task can be mapped onto each processing element. However, if a task requires dedicated resources, the mapping has to be restricted. This means that the mapping of individual tasks onto a (small) set of processing elements has to be enforced or avoided. Such restrictions on the mapping are expressed by mapping constraints. Thus, for each task a list of valid and invalid processing elements may be specified.

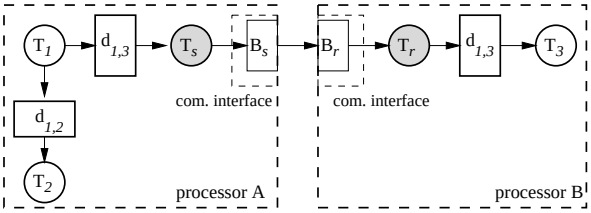

Figure 2: Realizing buffered intra- and inter-processor communication by introducing buffers (d_{ij}) and communication tasks (T_s and T_r). Inter-processor communication may result in synchronization of T_s and T_r.

2.4. Optimizer

The optimizer approximates an optimal mapping and schedule for all tasks given the application model, the hardware model and mapping constraints. For this approximation, the optimizer has to determine the memory usage as well as the execution and communication times.

Data transfer in our optimized DSP system is based on buffered communication (Figure 2). A task writes its output data into a communication buffer. The task(s) receiving these data read(s) from that communication buffer. If the buffer size is at least as large as the amount of data transferred, asynchronous communication is guaranteed and both sender and receiver task are decoupled. To realize buffered communication, the optimizer has therefore to allocate communication buffers between tasks transferring data. If both tasks T_i and T_j are mapped onto the same processing element, a buffer of size d_{ij} is allocated. If the tasks are mapped onto different processing elements, a buffer of size d_{ij} is required on both processing elements. In this case, the optimizer additionally introduces a sender task T_s on one processor and a receiver tasks T_r on the other processor (see Figure 2). These tasks read data from the buffer d_{ij} and write them to the corresponding hardware buffers of the communication interface and vice versa.

To reduce the number of communication buffers, the optimizer allocates only a single buffer among tasks receiving the same input data from an individual task. These tasks are identified by a group identifier g_T in the application model. Communication buffers are furthermore allocated dynamically, i.e., when all tasks receiving data from a single buffer have completed their read operation, the communication buffer is deallocated.

The memory usage for a processing element P_i is given by the sum of the required memory for each task located at P_i and the maximum memory need for the dynamically allocated buffers d_{ij} located at P_i.

The task execution times are specified in the application model. Time required for reading and writing data from and to buffers is included in the task execution times. The communication tasks T_s and T_r may not be decoupled

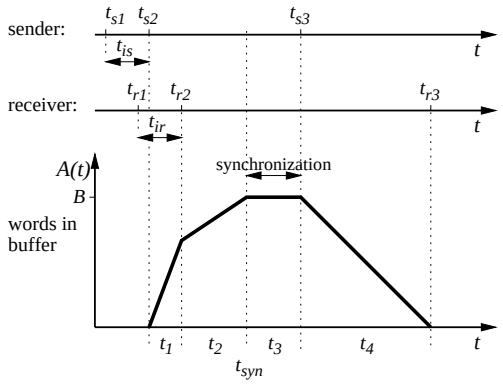

Figure 3: Timing diagram of a synchronized inter-processor communication.

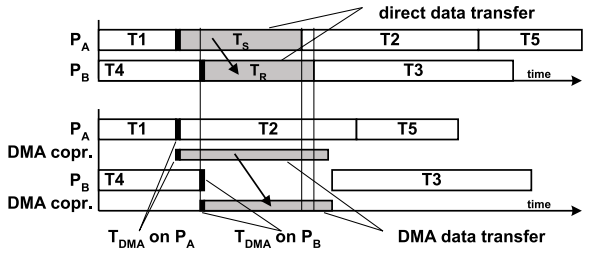

Figure 4: Comparison of direct inter-processor communication (above) and DMA data transfer (below). Inter-processor communication is indicated by an arc.

because the size of the hardware buffers in the communication interfaces B_s and B_r is limited. Thus, the overall execution time is determined by the individual task execution times, the task dependencies and the execution times of the communication tasks.

We apply Simulated Annealing (SA) [5][4] in our optimizer. SA minimizes a specified cost function which is composed by terms such as the overall completion time of the DSP application and the memory usage of the processing elements. These terms are weighted by the optimization parameters. By changing the cost function, e.g., by modifying the optimization parameters or introducing nonlinear functions, different optimization objectives can be achieved.

3. MODELING BUFFERED COMMUNICATION

3.1. Direct Inter-processor Communication

Due to the limited size of input and output hardware buffers (B_r and B_s) of the interfaces synchronization between sender and receiver tasks may occur in inter-processor communication. We model buffered inter-processor communication to determine the execution times of sender and receiver tasks transferring d data words over a buffered communication link of size $B = B_r + B_s$. K_s and K_r represent the transfer rates for writing to and reading from the buffers, respectively.

Figure 3 presents the timing diagram of the inter-processor data transfer. Three important time points can be identified for the sender as well as the receiver. At t_{s1} and t_{r1}, the sender and receiver tasks are initiated. After initialization (t_{is} and t_{ir}), the sender task starts writing data words into the communication buffer at t_{s2}, and the receiver task starts reading data words from the communication buffer at t_{r2}. Writing and reading data to and from the buffer is finished at t_{s3} and t_{r3}, respectively.

Data transfer over a buffered communication link can be separated into 4 phases. If we know the duration for each phase, the execution times for the sender and receiver tasks can be determined. In phase 1, only the sender writes data into the buffer. The duration is given as $t_1 = t_{r1} + t_{ir} - (t_{s1} + t_{is})$. At the end of the first phase (t_{r2}), $A(t_{r2}) = \min(B, d, \frac{t_1}{K_s})$ data words are stored in the buffer. In phase 2, both sender and receiver write/read asynchronously data to/from the buffer.[1] During this phase, the amount of data in the buffer is given by

$$A(t) = A(t_{r2}) + (\frac{1}{K_s} - \frac{1}{K_r})(t - t_{r2}). \quad (1)$$

Phase 2 ends when synchronization between sender and receiver is enforced. If the sender is faster than the receiver ($\frac{1}{K_s} > \frac{1}{K_r}$), synchronization is enforced when the buffer is completely filled ($A(t) = B$). Thus, by combining the synchronization condition with Equation 1, the synchronization time point can be determined

$$t_{syn} = \frac{B - A(t_{r2})}{\frac{1}{K_s} - \frac{1}{K_r}} + t_{r2}. \quad (2)$$

Synchronization does not occur if too less data are transmitted to fill the buffer ($d \leq \frac{1}{K_s}(t_{syn} - t_{s2})$). As a consequence, phase 3 is skipped and the duration of phase 2 is given as $t_2 = (d - A(t_{r2}))K_s$. Otherwise, the duration of phase 2 is $t_2 = t_{syn} - t_{r2}$. During phase 3 sender and receiver are synchronized. Data is written to and read from the buffer at the slower transfer rate. In the case described, the remaining data are written to the buffer at the speed of the receiver. Thus, the duration of phase 3 is given by $t_3 = (d - \frac{1}{K_s}(t_{syn} - t_{s2}))K_r$. In the final phase, the receiver only reads data from the buffer. The duration of phase 4 is $t_4 = A(t_{s3})K_r$.

For the other cases, i.e., if the sender is as fast as or slower than the receiver, the durations of phases 2 to 4 can be determined similarly. To summarize, the total execution time for the sender tasks T_s to write d words into the communication buffer is $t_{send} = t_{is} + t_1 + t_2 + t_3$ and the execution time for the receiver task T_r to read d words from the communication buffer is $t_{rec} = t_{ir} + t_2 + t_3 + t_4$.

3.2. DMA Transfer

In case of DMA inter-processor communication, data is transferred between memory and the hardware buffers of the communication interfaces (B_r and B_s) by dedicated

[1] Phase 2 is only entered if sufficient data has to be transmitted ($d > A(t_{r2})$).

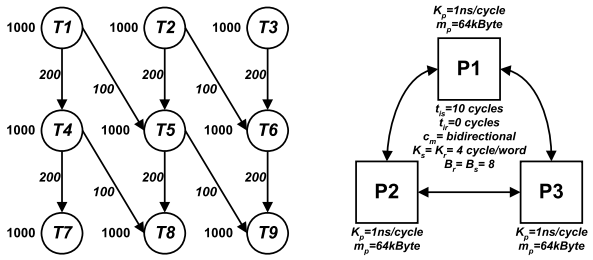

Figure 5: Design example with 9 tasks and 3 processors.

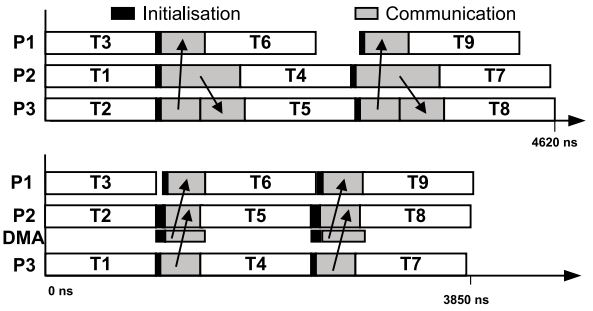

Figure 6: Optimized DSP system using direct inter-processor communication (above) and DMA transfer via a link of $P2$ (below).

DMA coprocessors. We model DMA transfer by replacing the communication tasks T_s and T_r with (short) DMA initialization tasks T_{DMAr} and T_{DMAs}. After DMA initialization data is transferred concurrently to CPU computation. Bus access conflicts between DMA and CPU may occur and delay task execution as well as data transfer depending on the assigned bus access priority (p_{DMA}). The factor for this delay is given by the relative number of bus conflicts c_b which is determined by the bus usage f_{bu} of all tasks during DMA transfer of time t:

$$c_b = \frac{1}{t} \sum_{\forall T_i \ in \ t} f_{bu}(T_i) t_{T_i} \qquad (3)$$

Figure 4 presents a comparison between direct inter-processor communication and DMA transfer. Inter-processor communication occurs between T_1 and T_3. Due to concurrent DMA transfer and CPU operation, the overall task completion time is shorter using DMA transfer. Note that the DMA transfer times are longer than the corresponding execution times of the communication tasks (T_s and T_r) due to bus access conflicts.

4. PROTOTYPING EXAMPLE

Mapping and scheduling using the communication model is demonstrated by the following example (Figure 5): the DSP-application consists of 9 tasks, each requiring an execution time of 1000 cycles and a data transfer to the lower and lower-right neighbor. The target system consists of three DSP processors, connected in a ring topology. In this example, the optimization objective is the overall task completion time only; memory requirements are not considered.

Figure 6 (above) shows a possible solution using direct inter-processor communication for all communication links. A better solution for this example is found, if DMA transfer is used for the communication between processors $P1$ and $P2$ (below).

5. CONCLUSION

The communication model presented in this paper is designed to meet the requirements of DSP applications. This model is successfully applied in our prototyping framework for optimizing multi-DSP systems. The main advantage of our modeling approach is the improved accuracy of its timing predictions which may be exploited to better utilize the hardware.

Future research will be focused on further refining the communication model and demonstrating our prototyping framework on more complex applications in the field of DSP and real-time systems. A mid-term goal of this research is to develop a framework for the codesign of multi-DSP systems.

6. REFERENCES

[1] J. E. Beck and D. P. Siewiorek. Simulated Annealing Applied to Multicomputer Task Allocation and Processor Specification. In *Proc. IEEE Symp. on Parallel and Distributed Processing*, pages 232–239, 1996.

[2] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee. Synthesis of embedded software from synchronous dataflow specifications. *J. of VLSI Signal Processing Systems*, 21(2), 1999.

[3] A. Burns, M. Nicholson, K. Tindell, and N. Zhang. Allocating and scheduling hard real-time tasks on a point-to-point distributed system. Technical report, University of York, UK.

[4] V. Ĉerny. Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *J. of Optimization Theory and Applications*, 45:41–51, 1985.

[5] S. Kirkpatrick, J. C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[6] K.W.Tindell, A. Burns, and A. Wellings. Allocating Hard Real-Time Tasks: An NP-Hard Problem Made Easy. *The Journal of Real-Time Systems*, 4:145–165, 1992.

[7] C. Mathis, M. Schmid, and R. Schneider. A Flexible Tool for Mapping and Scheduling Real-Time Applications onto Parallel Systems. In *Proc. Third Intern. Conference on Parallel Processing & Applied Mathematics*, pages 437–444, Kazimierz Dolny, Poland, 1999.

[8] W. Wolf. Hardware-Software Co-Design for Embedded Systems. *IEEE Proceedings*, 82(7):967–989, July 1994.