A Complexity Effective Communication Model for Behavioral Modeling of Signal Processing Applications

Satya Kiran M.N.V., Jayram M.N. [*]
Dept. of Computer Science & Engg.
Indian Institute of Technology Delhi
New Delhi, India
skiran@cse.iitd.ernet.in

Pradeep Rao, S.K. Nandy
CAD Laboratory, SERC
Indian Institute of Science
Bangalore, India
{pradeep, nandy}@cadl.iisc.ernet.in

ABSTRACT

In this paper, we argue that the address space of memory regions that participate in inter task communication is over-specified by the traditional communication models used in behavioral modeling, resulting in sub-optimal implementations. We propose *shared messaging* communication model and the associated channels for efficient inter task communication of high bandwidth data streams in behavioral models of signal processing applications. In shared messaging model, tasks communicate data through special memory regions whose address space is unspecified by the model without introducing non determinism. Address space to these regions can be assigned during mapping of application to specific architecture, by exploring feasible alternatives. We present experimental results to show that this flexibility reduces the complexity (e.g., communication latency, memory usage) of implementations significantly (up to an order of magnitude).

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming—*Parallel programming*

General Terms

Performance, Design, Languages

1. INTRODUCTION

The complexity and heterogeneity of modern signal processing systems coupled with rapid advances in VLSI design and technology offer a vast design space and render efficient system design as a major challenge. Behavioral (or application) modeling is an important phase in the recently proposed system design methodologies, which advocate design reuse and orthogonalization of concerns to manage the complexity the systems[1, 5]. Behavioral modeling can be viewed as the process of capturing specifications of the system to be designed with a chosen model of computation (MOC)[2]. One approach to behavioral modeling is to represent an application as a network of communicating tasks/processes. Behavioral models enable reuse and serve as executable specifications. They are also useful for system analysis, functional verification and design space exploration. In order to enable exploration of the vast design space, behavioral models (and their underlying MOC) should be implementation independent, i.e., they should not restrict the architectures to which they can be mapped. It is important to note that systems that are over-specified with respect to the designer's intent, result in sub-optimal implementations[2, 5]. In the rest of this section, we show that `i)` the address space of memory regions that participate in the communication between concurrent tasks of behavioral model, is over-specified by conventionally used message passing and shared memory communication models, and `ii)` this over-specification either imposes unnecessary restrictions on implementation (shared memory model), or is unable to exploit the advantages of architectural features (message passing model), and both result in sub-optimal designs.

[*]Jayram currently works with Philips Research, Eindhoven, Netherlands. Email: jayram.nageswaran@philips.com

1.1 Motivation

In message passing communication model, each task operates on its private address space and communicates with other tasks by *explicitly* exchanging messages using `send` and `receive` interface. A message passing channel implements the `send` and `receive` interface by buffering data and provides asynchronous communication between processes. The following advantages of message passing render variants of process networks with message passing channels (e.g., Kahn process networks[3], Dataflow process networks[4]) as important MOCs for the behavioral modeling: `i)` Message passing improves the modularity and reusability of behavioral models by separating computation and communication, which helps reduce the design time of new products and services[8]. `ii)` Message passing models can be mapped onto a wide variety of micro-architectures and hence useful in exploring the vast design space of modern systems. `iii)` The explicit communication semantics of message passing allows static program analysis, which assists (formal) verification[3] and enhances the performance by static partitioning and/or scheduling[4]. `iv)` Tuning the buffer size of message passing channels offer different memory-performance tradeoffs[8]. `v)` With additional restrictions on the behavior of tasks, message passing offers deterministic and compositional task networks[3].

Message passing, however, requires data replication incase of broadcast and incurs high communication overhead because of the data copying between tasks and channel buffers, which causes redundant communication and excess memory accesses. This has a significant impact on memory capacity and performance, communication latencies and power consumption, especially with high bandwidth data streams. The problem is exacerbated with tasks that perform in-place computation and shuffling of data (e.g., BBP to PBB frame shuffling in the MPEG encoder), which are typical in signal processing applications[6, 7]. In summary, message passing model insists that no two tasks should have shared address space, which result in sub-optimal designs when the micro-architecture has an efficient shared address space between the communicating tasks. This disadvantage with the message passing model can be mitigated if the tasks use shared address space for communication, as in the shared memory communication model.

Though shared memory communication model effectively eliminates data replication and data copying on micro-architectures with implicit inter task communication[1] (e.g, multi-threaded uniprocessor, multi-tasking uniprocessor, small scale multiprocessor with shared bus), it has its own issues: `i)` It is expensive to implement on message passing micro-architectures due to the additional shared memory emulation layer needed, either on hardware or in software[9]. `ii)` The model provides implicit communication, which hinders static program analysis and reusability. This also makes it hard to reason about some of the critical properties such as safety and liveness. In summary, shared memory communication model insists that communicating tasks should have shared address space, which result in sub-optimal designs on micro-architectures that do not inherently provide a shared address space.

From the above discussion, it is apparent that: `i)` tasks can communicate data irrespective of their address space, `ii)` restricting the address space to either shared or private is an over-specification that results in sub-optimal designs, `iii)` the address space of memory regions that participate in inter task communication should be flexible, and `iv)` there is a need for a communication model that retains the advantages of message passing, while addressing its disadvantage with regard to communication overhead. We propose *shared messaging* communication model to address these issues.

2. SHARED MESSAGING

2.1 Communication Model

Shared messaging communication model integrates message passing and shared memory communication paradigms. In this model each task operates on its private address space as well as on special memory regions, which must be used for inter task communication. Tasks communicate data in blocks of predefined size, through tokens. A token can be viewed as a pointer (or a reference) to a memory region along with attributes such as size of the region and privilege. A token can be associated with either read-only (RO) or read-write (RW) privileges. The privilege associated with the token indicates the valid accesses (read and/or write) that the task holding this token can perform on the associated memory region. To access a memory region, the task must possess a token pointing to that memory region. It is the responsibility of the task (and hence programmer) to make sure that every access to special memory region conforms to the privilege attribute of the associated token and hence is a valid access. The tasks are not allowed to copy tokens or modify its attributes. Tokens not only communicate data but also synchronize tasks, thus unify communication and synchronization as in the message passing model. Shared messaging model provides the `get_unused_mem`, `send_token_rc`[2], `send_token`, `receive_token` and `usage_over` interface for tasks to obtain memory regions and to transfer tokens between them.

[1]unlike shared memory architectures which provide shared memory communication primitives to the programmer irrespective of the underlying micro-architecture

2.2 Communication Channels

We propose two shared messaging channels, SMS and SMM [3], which implement the shared messaging interface. These two variants are motivated by two distinct communication patterns (pipeline communication and broadcast) that are typical in signal processing applications and have an efficient implementation on micro-architectures with implicit inter task communication. Shared messaging channels are unidirectional and lossless communication channels with FIFO ordered token passing. The tokens that are sent but not received are buffered in the token buffer to provide asynchronous communication. An SMS channel has one sender, one receiver and it maintains a token buffer. An SMM channel has one sender, multiple receivers and it maintains a token buffer per receiver. The token (but not the data) sent by the sender is replicated and buffered in the token buffers of all receivers, i.e., an SMM channel broadcasts the token to all its receivers thereby acheiving data broadcast. Each receiver can receive tokens from its associated token buffer only and hence, multiple receivers of the SMM channel do not introduce non-determinism as no two tasks are allowed to read tokens from the same token buffer[4]. The size of token buffer is limited for bounded shared messaging channels and is unlimited for unbounded shared messaging channels. The limited size of token buffers regulate the data rate of sender if it is faster than its receivers and is required for bounded execution of task network[4].

In a task network with shared messaging communication, tasks are interconnected by shared messaging channels and use the access methods provided by the channel. We use pre-condition and post-condition notation for terse representation of semantics of channel access methods (Table 1). Each method is associated with a pre-condition and a post-condition to be satisfied. The post-condition is guaranteed by the channel and it is the responsibility of the task (and hence programmer) to make sure that the pre-condition is satisfied before the invocation of a channel access method. Both pre-condition and privilege violations can be detected by static (compile-time) analysis on individual task. The conformance of the tasks to pre-conditions and privileges of shared messaging model does not allow implicit communication, though memory regions are shared among tasks. From the pre-conditions and post-conditions specified in Table 1, one can infer that (1) a token received from a SMS channel can be sent to another SMS channel, (2) a token received from an SMM channel can be sent to one or more other SMM channels and (3) a token received from an SMS channel can be sent to other SMM channels, but the converse is not true. These indicate that data can be transferred from one shared messaging channel to another without copying of data, unlike message passing channels.

2.3 A Simple Example

We illustrate the usage of channel access methods with a simple example. Consider the modeling of a task network with broadcast communication. To send a block of data to the receivers, the sender should compose the data in memory regions provided by shared messaging channels. In order to do so, the sender should first obtain an unused memory region through a token using `get_unused_mem`. The sender then composes the data in the memory region associated with the obtained token and sends the token to the channel us-

[2]`rc` indicates retain copy

[3]SMS is an acronym for shared messaging channel with single receiver and SMM is an acronym for shared messaging channel with multiple receivers

Table 1: Semantics of access methods of SMS channel (upper half) and SMM channel (lower half). token_handle is a data structure that holds a token

Access method	Pre-condition on token_handle	Post-condition of token_handle	Invoked by	Operation
`get_unused_mem` (token_handle)	should not hold any token	holds a token with RW privilege	sender	blocks if an unused memory region is not available, else a token pointing to an unused memory region is created
`send_token` (token_handle)	should hold a token with RW privilege	does not hold any token	sender	blocks if the token buffer is full, else token is transferred from token_handle to token buffer
`receive_token` (token_handle)	should not hold any token	holds a token with RW privilege	receiver	blocks if the token buffer is empty, else earliest token from token buffer is transferred to token_handle
`usage_over` (token_handle)	should hold a token with RW privilege	does not hold any token	receiver	token is destroyed and the associated memory region is marked unused
`get_unused_mem` (token_handle)	should not hold any token	holds a token with RW privilege	sender	blocks if an unused memory region is not available, else a token pointing to an unused memory region is created
`send_token` (token_handle)	should hold a token with RW/RO privilege	does not hold any token	sender	blocks if any of the token buffers is full, else privilege of token is changed to RO, replicated in all token buffers, and then destroyed
`send_token_rc` (token_handle)	should hold a token with RW/RO privilege	holds the same token but with RO privilege	sender	blocks if any of the receiver buffers is full, else privilege of token is changed to RO and replicated in all token buffers
`receive_token` (token_handle)	should not hold any token	holds a token with RO privilege	receiver	blocks if the associated token buffer is empty, else earliest token from its token buffer is transferred to token_handle
`usage_over` (token_handle)	should hold a token with RO privilege	does not hold any token	sender/ receiver	token is destroyed and the associated memory region is marked unused if no token pointing to that region exists

ing `send_token`. `send_token_rc` is used in cases where the sender needs to retain a copy of token for further use. Each receiver receives the pointer to the memory region sent by the sender using `receive_token` and then uses (reads) the data present in the memory region. The receiver is not allowed to modify the contents of obtained memory region, as the `receive_token` of the SMM channel only provides RO privilege. When the receiver/sender no longer needs the data, it acknowledges to the channel with `usage_over`. This helps in marking unused memory regions[4] so that they can be used for other communication. It is to be noted that, unlike the message passing model, shared messaging does not involve data replication and redundant copying of data, as both the sender and the receiver use the same memory region for reading and writing when shared address space is assigned to the memory region. This results in memory savings, reduced memory accesses and reduced pressure on communication resources (quantifed in Section 3).

2.4 Implementation Issues and Advantages

Some of the advantages and issues in the implementation of shared messaging model are qualitatively discussed here: **A)** Shared messaging model inherits the advantages of message passing model due to its explicit communication, FIFO ordered token passing and blocking semantics of channel access methods. **B)** As shared messaging does not allow implicit communication, the `send_token`, `receive_token`, and `send_token_rc` methods (which directly participate in the inter task communication) of shared messaging channels can be implemented with the `send` and `receive` methods of message passing channels. This implies that shared messaging channels are mutable to message passing channels. **C)** The address space to memory regions that participate in inter task communication is assigned during mapping of the task network onto the selected micro-architecture, by exploring feasible alternatives. In message passing micro-architectures, memory regions are assigned a private address space and the shared messaging channel is replaced by the message passing channel. Thus, unlike shared memory communication model, our shared messaging model does not require the use of expensive emulation layers. **D)** As shared messaging model allows only explicit communication, it can be used in most of the MOCs that use the message passing model while retaining the properties and advantages of the corresponding MOC; e.g., (i) a network of monotonic processes[3] communicating through unbounded shared messaging channels is still deterministic and compositional, (ii) network of data flow actors[4] communicating with unbounded shared messaging channels are still amenable to static program analysis. **E)** Shared messaging model does not specify the memory organization and management[7], which allows the designer to write models without being concerned about it. The model support different memory management schemes at the channel level[6], processing element level, processing node level[7], or a combination of these, with different memory-performance tradeoffs. The variety of memory management schemes supported by the shared messaging model allows for a wider design space exploration. **F)** The `receive_token` method indicates the memory region (and its size) that may be used by the task (that invokes the method) in near future. This information can be used to assist both compiler supported and dynamic pre-fetching[9]. **G)** The `usage_over` method indicates that the memory region associated with the token is no longer used by the task. This information can be used for early invalidation of data cache. **H)** With shared messaging model, the shared locations can be distinguished from those that are not. This allows for an increased relaxation in the ordering of memory operations and better latency hiding techniques[9] in shared memory multiprocessors.

[4]A memory region is marked unused if there exists no token pointing to that region

3. EXPERIMENTAL RESULTS

In order to compare the communication latencies involved with shared messaging and message passing communication, we developed a C++ runtime library which executes a task network on a workstation as a single thread using the data driven execution model[8]. We use the `clock` function of standard C library to measure the execution time. We define the communication latency as the total time spent by all the tasks in executing the communication primitives, excluding the waiting time for communication resources.

We have modeled the typical producer-consumer example, in which the producer sends data chunks (each of size block_size) to the consumer, in message passing and shared messaging paradigms. In order to compare the communication latencies, we set the computation time of both producer and consumer to zero, while measuring the communication latency for varying block sizes. Second row of Table 2 shows the ratio of communication latency of producer and consumer communicating with message passing channel to communication latency of producer and consumer communicating with SMS channel (these numbers can be interpreted as factor of reduction in communication latency with the use of shared messaging channels). We have repeated the above procedure for a broadcast example, in which the sender broadcasts data chunks to 4 receivers. The third row of Table 2 corresponds to this example. In this case SMM channel is used in shared messaging model.

From the Table 2 it can be observed that `i)` message passing channels outperform shared messaging channels at lower block sizes due to the overhead associated with obtaining memory regions and token passing, `ii)` more overhead is associated with SMM channels (compared to SMS) as they need to keep track of tasks that are operating on same memory region (this is not required incase of SMS, as the semantics of SMS allows only one task to operate on a given memory region), `iii)` at higher block sizes shared messaging channels outperform message passing channels as the time saved due to the elimination of redundant data copying dominates the overhead associated with them. It can be analyzed that `a)` SMS channels do not affect the normalized memory (buffer) requirement with channel level memory management, while other schemes may result in improvements, `b)` SMM channels eliminate data replication and hence reduce memory (buffer) requirements, which is in proportion to the number of receivers, even with channel level memory management.

Our results indicate that for small block sizes, message passing channels perform better than shared messaging channels. As shared messaging channels coexist with message passing channels, we suggest that they be used appropriately to result in efficient behavioral models, i.e., one could use message passing channels for small data transfers (typically control information) and shared messaging channels for high bandwidth data transfers. As the communication patterns analyzed do not account for the computation time, the speedup factors presented should not be interpreted as the speedup in the execution time of the entire application. The overall speedup depends on the ratio of computation to communication within the application. We modeled the JPEG encoder to quantify the overall speedup. The baseline JPEG encoder[10] application is partitioned into tasks that perform in-place DCT, quantization and Huffman encoding. Shared messaging model has shown an overall execution speedup of **1.77** over the message passing model.

4. RELATED WORK

Synchronous message passing eliminate intermediate buffering by directly copying the data from the sender's private address space to the receiver's private address space. But, it reduces the effective parallelism in the application and still involves some redundant data transfers. To reduce the communication overhead associated with message passing, the ARACHNE protocol[7] explicitly exchanges pointers to shared memory regions. This protocol eliminates data replication and redundant communication, but it is not clear if this protocol is portable to message passing micro-architectures. It allows different memory management schemes but they are not transparent to the programmer, i.e., each scheme may require a modification in the program. It is unclear whether this protocol is deterministic.

Table 2: Normalized (to shared messaging) communication latency of message passing for two communication patterns with varying block size (shown in bytes)

	1	10	100	1000
prod-cons (SMS)	0.634	2.670	24.677	247.110
broadcast (SMM)	0.316	1.358	11.630	112.262

The C-HEAP communication protocol[6] provides channels with one or more receivers with `claim/release/query space/data` interface. Though the protocol eliminates data replication and data copying in some cases, it still suffers in case of in-place computation and shuffling. The C-HEAP protocol requires an in-order release of the space claimed, whereas our model does not. Only channel level buffer (or memory) management is allowed and the schemes at other levels are not addressed. Also the protocol introduces non-determinism due to the provision of `query_data/space`.

5. CONCLUSIONS

This paper has motivated the need for memory regions with flexible address space for inter task communication in implementation independent behavioral modeling. We have proposed the shared messaging communication model and its associated channels and shown that shared messaging models could provide up to an order of magnitude improvement in the communication latency over the message passing model; this results in an overall execution speedup of 1.77 for the baseline JPEG encoder. Shared messaging channels do not introduce non-determinism, coexist with message passing channels and their judicious use results in behavioral models that reduce the complexity of implementations significantly. Though the shared messaging model is implementation independent, it carries the information that is required for efficient implementation on a wide variety of architectures. Currently we are working on, the formal underpinnings of the proposed model and its rigorous evaluation on various architectures and applications.

6. REFERENCES

[1] Kurt Keutzer et. al. System level design: Orthogonalization of concerns and platform-based design. *IEEE Tr. on CAD of ICs and Systems*, 19(12), December 2000.

[2] M. Sgori et. al. Formal models for embedded system design. *IEEE Design & Test of Computers*, April 2000.

[3] G. Kahn. The semantics of a simple language for parallel programming. In *Proc. of the IFIP Congress 74*, pages 471–175. North-Holland Publishing Co., 1974.

[4] E. A. Lee and T. M. Parks. Dataflow process networks. In *Proc. IEEE*, pages 773–801, May 1995.

[5] E. A. de Kock et al. YAPI: Application modeling for signal processing systems. In *DAC*, June 2000.

[6] Nieuwland et. al. C-HEAP: A heterogeneous multiprocessor architecture template and scalable and flexible protocol for the design of embedded signal processing systems. In *Design automation for Embedded Systems*. Kluwer, 2002.

[7] K. G. W. Goossens. A protocol and memory manager for on-chip communication. In *ISCAS*, 2001.

[8] Twan Basten et. al. Efficient execution of process networks. In *Communicating Process Architectures*, 2001.

[9] Culler and Singh. *Parallel Computer Architecture - A Hardware/Software Approach*. Morgan Kaufmann, 1999.

[10] W. Gregory. The jpeg still picture compression standard. *Communications of the ACM*, 34(4):30–44, April 1991.