Telecommunication Application Modelling with Multi Writer Multi Reader Channels: a Case Study

Etienne Faure, Daniela Genius (Laboratoire LIP6, Université Pierre et Marie Curie)

Abstract: Telecommunication applications require an increasingly high throughput; their task graph often exhibits a high level of coarse grained parallelism. We extend the KPN model by multiple writers and readers and present the SystemC based co-design of a packet classification application on a multi processor system on chip.

1 Introduction

Telecommunication applications process packet streams, where essentially the same sequence of operations is performed on each packet, but the actual computing depends on the packet contents. Throughput requirements are variable: backbone equipments, such as routers, require high throughput and low processing per packet, while e g. traffic analysis requires less throughput but intensive computation per packet. For [5], this variable processing time is the main characteristic of network applications. Inter-task communications can be done through message passing like in STepNP [3], modeled in the form of data flow graphs like Ptolemy [1] or use the shared memory capabilities of the multi-processor hardware architecture. Kahn Process Networks (KPN [2]) propose a semantics of inter-task communication through point to point FIFO channels.

2 Application model

We focus on telecommunication applications written in the form of a set of coarse grain parallel threads communicating with each other. While the KPN formalism is well suited for video and multimedia applications, which can be modeled by a task graph where each communication channel has only one producer and one consumer, it is not convenient for telecommunications applications where several tasks access the same communication buffer and whose task graph exhibits a hybrid pipelined/task farm parallelism.

The application structure can be modeled as a directed bipartite graph, where the two types of nodes are the tasks and the communication channels. Each task can be implemented as a software task running on programmable processors, or as an hardware coprocessor. Each communication channel can have several producers, and several consumers, and is protected by a specific exclusive access lock.

Classification is an important and resource-consuming part of many telecommunication applications. Packet headers or contents are analyzed, depending on the chosen method, before sending on the packet to one of several priority queues. Figure 1 shows the task and communication graph (TCG) of a typical classification application. For a

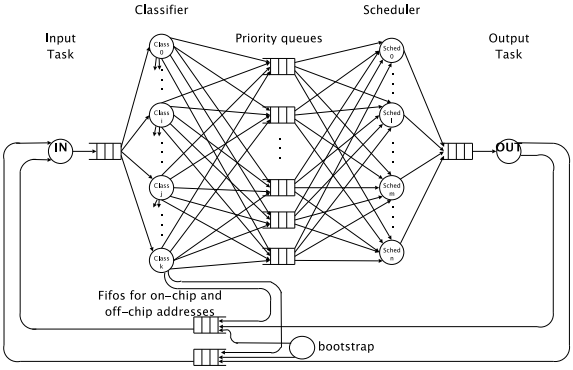

Figure 1. Classification application TCG

large majority of networking applications it is sufficient to consider the *beginning* of a packet, which is consequently stored in faster on-chip RAM. Assume that packets have already been cut into chunks of equal size; these so-called *slots* of 128 bytes are adapted to the size of the on-chip memory banks. Each slot has a small descriptor (usually eight byte) containing the address of the next slot and some necessary information to retrieve it. Only descriptors transit the MWMR channels, limiting thus the size of the memory allocated to the channels. The *input task* accepts on-chip and off-chip addresses from two separate channels. It reads Ethernet IP packets, computes the number of slots required, and copies these slots to on-chip and off-chip memory, respectively. A *classification task* tries to read one or more descriptors and then retrieves the packet from memory. The packet is deallocated if one of various checks fails. The classification task then writes the descriptor to one of twelve priority queues. The *scheduling tasks* ponder by the priority

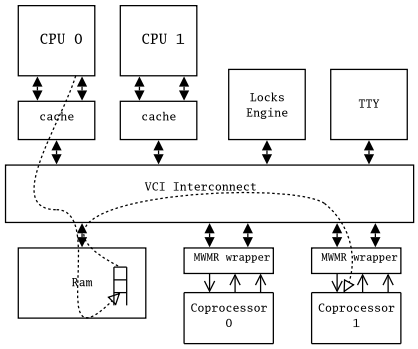

Figure 2. Generic hardware architecture

of the current queue and write the descriptor to the unique output queue if eligible. A *bootstrap task* organizes the system startup. The *output task* constantly reads the output queue. Each time a slot is read and sent to the buffer, its liberated address is sent to either of the two address channels. Classification, scheduling and bootstrap are software tasks, while input and output tasks are hardware coprocessors.

3 Target hardware architecture

All components are integrated on a single chip (Multi Processor System-on-Chip MPSoC). Our generic target MPSoC architecture contains a variable number of small programmable 32 bit RISC processors, a variable number of embedded RAM banks, other components such as lock engine, terminals, interrupt controllers, and several I/O coprocessors. These hardware components are chosen from a large selection of simulation models from a public domain library of simulation models written in SystemC [4]. Hardware component models are connected to an VCI/OCB compliant micro-network based on the shared memory paradigm. There are two types of components: *initiators* (typically a processor's cache or a coprocessor) and *targets* (most others like RAM and terminal). Our hardware descriptions are cycle-accurate bit-accurate. Figure 2 shows the hardware architecture with two processors and one memory bank. On the TTY the progress of the application can be observed, the Locks Engine manages the locks when more than one initiator is present. The MWMR channel is located in on-chip RAM; it implements a communication channel between a software task running on CPU0 and a hardware task executed by coprocessor 1. To connect a MWMR coprocessor to the VCI interconnect and thereby to the RAM, a hardware wrapper is required.

To reduce contention on the interconnect in the presence of many components, we use a clustered platform. The input cluster contains one input coprocessor, a controller of access to external memory and a memory bank for on-chip slot storage. It is connected to other clusters by a network-on-chip. The output cluster is symmetric to the input cluster. Processing clusters contain four processors and their caches, two memory banks, a timer and a locks engine. The platform contains eight clusters: two are dedicated to I/O, the remaining six are general purpose. In such a NUMA (Non Uniform Memory Access) architecture, memory access times differ depending on whether a processor accesses a memory bank local to the cluster, or on another cluster.

Tasks are statically mapped unto processors, one task per processor, to avoid context switching overhead the MIPS R3000 processor model does not support multiple contexts. For sake of efficiency, all MWMR channels are mapped to *cached* segments. To guarantee coherency in the presence of multiple readers and multiple writers, cache lines are selectively invalidated after each MWMR channel access.

Small packets of 40 bytes payload and 14 bytes Ethernet header constitute the worst case for classification, because the number of headers to verify is largest wrt. the data throughput. Mean latency is measured in SystemC simulation cycles, from reading a packet from the input stream to writing it to the output stream in *steady state*: bootstrap is terminated and buffers have filled up. We observe decreasing latencies with increasing packet number: the InputEngine starts to copy packets into memory while the classification tasks have not yet started their work; mean latency is 14.000 cycles. For our platform running 200MHz MIPS R3000, we achieve 1.67 bit/cycle i e. 334Mbit/sec.

4 Conclusion and perspectives

We designed a MPSoC platform using SoCLib components and added two specialized I/O coprocessors. A MWMR channel based task graph description has been integrated as an optional front end for SoCLib. We are currently exploring the design space more thoroughly.

References

[1] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: a framework for simulating and prototyping heterogeneous systems. *Readings in hardware/software co-design*, pages 527–543, 2002.

[2] G. Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Information Processing '74*, pages 471–475. North-Holland, NY, 1974.

[3] P. Paulin, C. Pilkington, and E. Bensoudane. StepNP: A system-level exploration platform for network processors. *IEEE Des. Test*, 19(6):17–26, 2002.

[4] SoCLib Consortium. The SoCLib project: An integrated system-on-chip modelling and simulation platform. Technical report, CNRS, 2003. http://www.soclib.fr.

[5] L. Thiele, S. Chakraborty, M. Gries, and S. Kuenzli. A framework for evaluating design tradeoffs in packet processing architectures. In *Proc. DAC*, pages 880–885, NY, USA, 2002. ACM Press.