Next Generation Network Processor Technologies

Enabling Cost Effective Solutions for 2.5 Gbps to 40 Gbps Network Services

Network Processor Division
Intel® Corporation
October 2001

Next Generation Network Processor Technologies Enabling Cost Effective Solutions for 2.5 Gbps to 40 Gbps Network Services

Contents

Abstract 1

Problem Statement 1

Architecture Level Solution 3

Effective Pipelining & Network Processor Technologies 4

Intel's Next Generation Network Processor Technologies 5

Summary 7

For More Information 8

Abstract

Network processors continue to find widespread application in different market segments, covering a variety of network services and wide ranges of performance levels. Cost effectiveness and high flexibility are significant advantages that network processors offer to accelerate the development of Next Generation Network (NGN) equipment. As the NGN applications grow rapidly in complexity and performance level, the requirements on processing power and intelligence increase very quickly. The next generation network processors must stay ahead of the requirement curves for the various market segments, while continuing to offer excellent cost effectiveness and flexibility.

This white paper describes the key technical problems for network processing at 10 Gbps and higher line rates, and recommends an architecture level solution. Furthermore, the white paper highlights the important network processor technologies that enable effective implementations of this architecture level solution. More important, this paper reveals some of the new network processor technologies that Intel's next generation network processor families will offer, and illustrates their effectiveness.

In the short term, Intel will expand its network processor offerings with the introduction of three new network processor families, designed to meet the respective requirements of the core/metro, access/edge, and customer premises equipment market segments. These new network processor families will offer an impressive list of next generation network processor hardware technologies, optimized in terms of both performance and cost effectiveness for each of the respective market segments. In addition, these new network processors will employ the latest Intel semiconductor process technologies, advanced processor design techniques, and excellent economy-of-scale manufacturing capability. The ultimate goal is to enable network equipment vendors to offer their customers the best combination of customizable/upgradeable services and wire-speed performance, while minimizing development time and costs.

Problem Statement

At the microscopic perspective of a typical network processor, Layer-2 cells/packets arrive at a specified maximum line rate. For example, at 10 Gbps line rate, a new 40-byte minimum size IP packet may arrive every 35ns. At 40 Gbps line rate, this arrival rate tightens to only 8ns. The network processor must perform the necessary Layer-3 through Layer-7 applications on these cells/packets, and must transmit the processed cells/packets out at the desired sequence and rate. The network processor must be able to keep up with wire-speed so that no cell/packet gets dropped unintentionally.

Line rate		40-byte packet arrival rate
2.5 Gbps	OC-48	160ns
10 Gbps	OC-192	35ns
40 Gbps	OC-768	8ns

Table 1: Inter-packet arrival rates

At high line rates, for example 10 Gbps to 40 Gbps, this generic operation of a typical network processor reveals two classic problems: the *dependence problem* and the *independence problem*. The dependence problem is that cells/packets usually depend upon predecessor cells/packets, and processing of ordered cells/packets demands atomic and sequential accesses to shared data structures. For instance, in an ATM AAL5 application, CRC calculation for an ATM cell requires the CRC residual value that is generated from the CRC calculation for the predecessor cell. The independence problem refers to the fact that even unrelated cells/packets usually access common data structures, and accesses to these shared resources limit

performance by imposing long latency and serialization. For instance, cells/packets get enqueued to buffers and dequeued for transmit by a common transmit scheduler. These network processing problems become exponentially more complex as the line rate increases.

A closer look at the dependence problem reveals that there are actually three cases. First, since the arrival rate between sequential cells/packets that belong to the same context may not be deterministic, the network processor needs to buffer these contexts. As a new cell/packet arrives, the relevant context needs to be retrieved from the buffer. Depending on the number of contexts that the application supports and the internal storage capacity the network processor supplies, these contexts may need to be buffered in external memory, which incurs additional latency. Second, while a context is being updated for a cell/packet, a subsequent cell/packet may start to access the same context. These accesses must complete atomically and in the correct sequence. In other words, these accesses must not result in data corruption due to interference with each other. Third, in order to maintain line rate processing, there may not be sufficient time to allow the completion of a write to a context buffer, before the read to the same context buffer for a subsequent cell/packet must take place.

The independence problem can be generalized to be a linked list management problem. For instance, a pool of buffers is typically implemented as a linked list of buffers. Allocation and freeing up of buffers require traversing a linked list, which may be implemented as a sequence of dependent read operations. Dependent memory operations require long latency, and may force serialization on parallel accesses. In addition, at the application level, many Transmit Scheduler algorithms employ linked lists. At run-time, the execution of these algorithms leads to traversal of linked lists at wire-speed. More important, each linked list element must be added or removed atomically, while maintaining the correct sequence.

The following example application demonstrates the complexity of supporting linked list traversal at wire-speed. In the case of 10 Gbps line rate and 40-byte IP packets, a new packet arrives every 35ns. Suppose the application manages a large number of queues and each queue is implemented as a linked list. This application maintains the table of queues and the actual queues in SRAM for performance reasons. The application enqueues a packet into the proper transmit-queue, and dequeues it when the packet is scheduled for transmit. The processing of each packet requires at least one enqueue and one dequeue operation. Each enqueue and dequeue operation requires three dependent SRAM transactions. For instance, to dequeue the head element of a linked list, the following sequence of dependent operations happens: read the head pointer from the queue management table, read the next element pointer of the head element, and write this next pointer value to the queue management table as the new head pointer. The current SRAM technology can handle a dependent SRAM transaction in about 10ns. The enqueue and dequeue operations for each packet require $6 \times 10ns$. Even without taking into account other processing functions, this 60ns latency already far exceeds the 35ns packet processing time budget before a new packet arrives.

As a result of these problems, network processors must employ special schemes in order to handle wire-speed cell/packet processing at 10 Gbps and higher line rate, even with state-of-the-art semiconductor technology and advanced circuit design techniques. With ASIC and standard-cell hardware design methodologies, these problems become much more complicated, and may not be solvable efficiently and cost effectively starting at even 2.5 Gbps line rate.

Architecture Level Solution

Network processing requires extremely high speed update of state sequentially and coherently, while demanding exclusivity and atomicity. Intel's architecture solution to this classic problem is pipelining.

The idea behind pipelining is to break the entire processing for each cell/packet into sequential stages. As long as the dependencies and ordering of these stages are maintained, the execution of the stages that correspond to many cells/packets can be performed in parallel using multiple processing elements. Moreover, each processing element can execute multiple threads in a time-shared manner in order to maximize the overall processing throughput.

Network processing applications normally include multiple functions. For instance, a typical router application consists of the following functions: packet receive, route table look-up, packet classification, metering, congestion avoidance, transmit scheduling, and packet transmit. Each of these functions can be implemented as a pipeline or a pipeline stage. These pipelines connect together to form the entire application.

There are two basic pipelining approaches: *context pipelining* and *functional pipelining.*

A context pipeline is comprised of processing stages that each performs a specific function. The context of a cell/packet moves from stage to stage as the individual functions are sequentially performed on the context. Each context pipeline stage is represented by a processing element. If the processing element supports multi-threading with up to n threads, the processing element can apply the specific function to n contexts of cells/packets during one stage. As a result, the time budget for a stage can be n times the cell/packet arrival rate. In other words, one can afford to have n times the cell/packet arrival rate as time budget to perform only one function, which is just a portion of the entire processing. Another advantage of context pipelining is that each processing element only needs to perform its own function; consequently, the complexity and amount of software required for a processing element is minimized to the support of that particular function.

In functional pipelining, a single processing element performs different functions during different stages, on the same cell/packet context. Consider a processing element that performs m consecutive functions on a cell/packet context; during each stage, a processing element only performs one of the m functions. This processing element takes m stages to complete its processing on each cell/packet context. In order to avoid blocking the overall pipeline advancement as new cells/packets arrive, m processing elements work in parallel. These m processing elements form the functional pipeline. These m processing elements actually work in a staggered fashion, so that at any one stage in time, each processing element performs a different function out of the m functions. This staggering is needed because each function may demand exclusive ownership of some global state. In case each processing element supports n threads of multi-threading, it can process n different cell/packet contexts in a time shared manner in each stage. The advantage of functional pipelining is that the time budget each processing element has for operating on a cell/packet context is $m \times n \times$ the cell/packet arrival rate. This time budget is m times bigger than the time budget that a context pipeline offers. In other words, functional pipelining accommodates very long latency functions efficiently. The disadvantages of functional pipelining include the overhead for transferring state information between consecutive processing elements, and the relatively greater complexity or larger amount of software required in each processing element; each needs to perform m functions as opposed to one function for a context pipeline stage.

Effective Pipelining & Network Processor Technologies

The pipelining architecture approach enables complex network processing to keep up with extremely fast packet arrival rate. From the architecture perspective, pipelining clearly solves the throughput problem. Nevertheless, effective implementations of pipelining must also minimize the latency of network processing. The important network processor technologies that enable efficient pipelining implementation and minimize latency include a pool of intelligent and highly programmable processing elements, fast and flexible communication between processing elements, multi-threading, and hardware support for atomic and ordered data sharing.

Network processors need to include a pool of intelligent and highly programmable processing elements that excel in processing cells/packets at wire-speed. Network services are growing in both performance and complexity levels. Moreover, the applications are changing as standards evolve or get established. New applications emerge to support new usage patterns and to replace old applications that have become obsolete. At the macro level, network processors must provide ample processing power and intelligence to run the various network applications efficiently. In addition, network processors must offer high flexibility to accommodate changes in application and usage patterns. Optimal balance between customized functionality and general purpose programmability is significant. Hard-wired solutions perform well on the expected functionality and usage pattern, but may not be applicable and may become obsolete easily when the application or usage pattern changes. For instance, a network processor that is optimized to process ATM traffic may not be able to support the various Quality of Service (QoS) functions for IP packets, let alone execute at wire-speed. On the other hand, general purpose microprocessors are not cost effective, and lack hardware support on some common and basic networking functionality; consequently they may not be able to sustain wire-speed network processing. Moreover, the cost penalty associated with using general purpose microprocessors for wire-speed network processing worsens quickly as line rate increases.

At the micro level, each processing element owns the processing of a pipeline stage. Processing elements must offer high computing power, support basic networking functions in hardware, and directly enable efficient implementation as well as execution of a pipeline stage. Moreover, in order to optimize the implementation of network processing for high line rates, the workload for the pipeline stages and the topology of the various functional and context pipelines need to be adaptable with respect to the application and usage patterns. Flexibility in processing elements significantly improves the development efficiency for such optimization, or adjustments to changes in usage patterns.

In a pipeline, data and control information continuously flow between pipeline stages. Fast and flexible communication between processing elements is extremely important. Specifically, in a context pipeline, as the pipeline advances, the processed cell/packet context is pushed to the subsequent processing element in the pipeline. In a functional pipeline, processing elements pass global state information to the next processing element in the same functional pipeline. Each processing element should have direct paths to communicate with its neighbor processing elements. Ideally, processing elements have exclusive usage of the direct paths so that communication incurs minimum latency and happens deterministically in order to simplify performance tuning.

Multi-threading is a proven technique for managing memory latencies. Specifically, the same processing element executes multiple

threads in a time-shared manner. When a thread is waiting for a memory access to complete, the processing element can switch the execution to a different thread. When the earlier thread becomes ready again (once the memory access has completed), the processing element switches back to it and resumes execution. Computing capacity gets utilized efficiently even while memory accesses take place. Network processing involves many dependent memory accesses, which result in long latency and serialization. As a result, Intel's architecture and processing elements support multi-threading in order to maximize throughput and utilization. Moreover, as the performance gap between processors and memory technologies continues to widen, the architecture allows processing elements to scale upward in thread count, ensuring wire-speed performance at line rates of 10 Gbps and beyond.

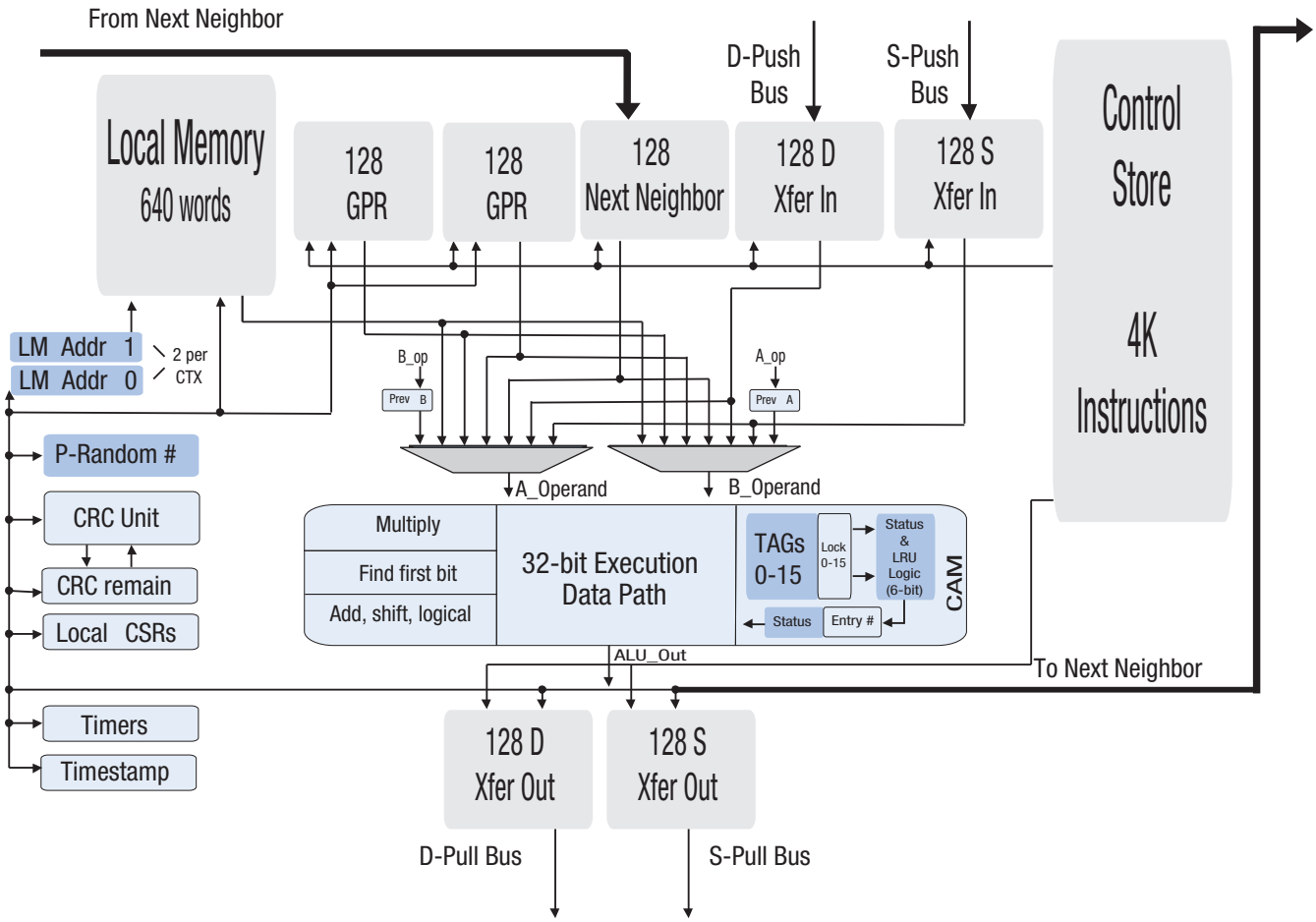
In an overall network processing pipeline, there exist many scenarios of data sharing. For instance, some of the threads on the same processing element may need to update the same entry of a common data structure in memory. Without dedicated hardware support, such scenarios pose extra performance overhead. This is because the processing element first needs to detect these data access collisions and then needs to serialize the accesses among the multiple threads while maintaining the sequence and atomicity of the accesses. If the shared data is stored in external RAM, this process will involve several long-latency accesses being performed sequentially. On the other hand, if each processing element has a cache, a simple cache look-up can detect whether an earlier thread has already accessed a certain shared data location. Furthermore, subsequent threads can pick up the shared data, which is still resident in the processing element, from an earlier thread directly instead of going through unnecessary memory accesses. For instance, consider a processing element whose function performs read-modify-write operations on some shared data structures. A thread can pick up the modified data from an earlier thread, modify the data per its own operation, and either pass the data to a subsequent thread or write it back to memory depending on whether a subsequent thread will need the same data.

The performance implications illustrate the advantages of having such distributed caches. In the above example, using distributed caches, the worst case scenario occurs when all the threads access different data locations. Without the cache support, the worst case would occur when all the threads need to access the same data, and these accesses must happen in a serial fashion. In addition, the cache support offers improved overall network processing performance by minimizing memory bandwidth consumption for multiple threads accessing the same data.

Intel's Next Generation Network Processor Technologies

Intel's upcoming network processor offerings include many exciting next generation network processor technologies. Of the complete list of next generation technologies, this white paper illustrates *next neighbor registers* and *distributed caches*.

Next neighbor registers are new registers that have been added to Intel's microengine architecture; microengines are the data plane processing elements in Intel's network processor offerings. Each microengine can directly and exclusively write to the next neighbor registers of its neighbor microengine. Moreover, the receiving microengine can use the content of the next neighbor registers just like General Purpose Registers from both the instruction set register usage and performance perspectives. Furthermore, there is additional hardware support to make the next neighbor registers form a FIFO ring between the producing microengine and the



consuming microengine. The functionality, performance, and deterministic nature of next neighbor registers minimize the communication overhead between adjacent microengines in a pipeline, and maximize the development efficiency for implementing and tuning pipelines.

The microengines in Intel's next generation network processors contain their own caches. These distributed caches allow parallel content-addressable look-up on all entries in one clock cycle. In addition to the typical look-up result of hit or miss, the distributed caches supply extra information. Specifically, when the look-up results in a miss, a distributed cache also returns the index to be replaced. When the look-up results in a hit, a distributed cache returns not only the index of the matching entry, but also 4 bits of user-defined information tagged with that particular entry. For instance, before a thread accesses a shared data location in memory for a read-modify-write operation, it first looks up the distributed cache on its microengine. A hit can indicate that a prior thread has already accessed the desired

data. In this case the 4-bit user defined information can convey this message as well as the thread number of the prior thread. Thus, the current thread can access the shared data from this prior thread at a later time, rather than performing another memory fetch. The functionality and performance of the distributed caches readily enable optimization in data sharing.

Furthermore, the fully associative content-addressable nature of distributed caches directly facilitates optimizations on multiple-outcome conditional branches, such as a C language switch statement. For instance, a distributed cache readily forms a software jump-table.

In addition to instruction set architecture level enhancements, Intel's next generation network processors employ state-of-the-art Intel semiconductor process technologies, advanced processor development techniques, and excellent economy-of-scale manufacturing capability. The net result is not only optimal circuit performance with minimal power consumption, but also extremely cost-effective single-chip network processors even at 10 Gbps performance level. These advantages directly translate into decreased time and costs for NGN equipment development for Intel's customers.

Summary

At 10 Gbps and higher line rates, network processing involves overcoming critical dependence and independence problems. In other words, network processing requires extremely high speed update of state sequentially and coherently, while demanding exclusivity and atomicity. Intel's solution to this architectural problem is pipelining. The basic pipelining approaches include context pipelining and functional pipelining. At the architecture level, Intel's pipelining approach for high speed network processing facilitates optimal combinations of functional and context pipelines, each of which represents a specific function of the entire network application. Moreover, Intel's approach enables the topology and mapping of functions to the individual pipeline stages to be precisely tuned, to maximize throughput as well as to minimize the overall latency for the target application and usage patterns. Intel's design incorporates important network processor technologies, including a pool of intelligent and highly programmable processing elements, fast and flexible communication between processing elements, multi-threading, and hardware support for atomic and ordered data sharing. In addition, Intel's network processors deliver advanced circuit level performance with minimal power and silicon area consumption.

In the short term, Intel will expand its network processor offerings. These next generation network processors will offer an impressive list of advanced network processor hardware technologies. Some of the new technologies include next neighbor registers and distributed cache. Moreover, Intel's state-of-the-art semiconductor technologies, advanced processor design techniques, and excellent economy-of-scale manufacturing capability will enable extremely cost effective single-chip network processors, even at 10 Gbps performance level. The ultimate goal is to enable network equipment vendors to offer their customers the best combination of customizable/upgradeable services and wire-speed performance, while minimizing development time and costs, in the core/metro, access/edge, and customer premises equipment market segments.

Intel Access

Developer's Site	developer.intel.com
Networking and Communications Building Blocks Site	www.intel.com/design/network
Intel® Internet Exchange Architecture Site	www.intel.com/IXA
Other Intel Support	developer.intel.com/design/litcentr
Intel Literature Center	(800) 548-4725 7 a.m. to 7 p.m. CST (U.S. and Canada) International locations please contact your local sales office.
General Information Hotline	(800) 628-8686 or (916) 356-3104 5 a.m. to 5 p.m. PST

For more information, visit the Intel Web site at: **developer.intel.com**

UNITED STATES AND CANADA
Intel Corporation
Robert Noyce Bldg.
2200 Mission College Blvd.
P.O. Box 58119
Santa Clara, CA 95052-8119
USA

EUROPE
Intel Corporation (UK) Ltd.
Pipers Way
Swindon
Wiltshire SN3 1RJ
UK

ASIA-PACIFIC
Intel Semiconductor Ltd.
32/F Two Pacific Place
88 Queensway, Central
Hong Kong, SAR

JAPAN
Intel Kabushiki Kaisha
P.O. Box 115 Tsukuba-gakuen
5-6 Tokodai, Tsukuba-shi
Ibaraki-ken 305
Japan

SOUTH AMERICA
Intel Semicondutores do Brazil
Rue Florida, 1703-2 and CJ22
CEP 04565-001 Sao Paulo-SP
Brazil

Order Number: 279050-001 Printed in USA/1001/3K/IL11497 CM