Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor Performance Tuning

Application Note

July 2004

Document Number: 253499-003

Contents

Figures

Tables

Revision History

Date	Revision	Description
July 2004	003	Updated Intel® product branding.
August 2003	002	Document edited for public release.
July 2003	001	Initial release of this document

This page intentionally left blank.

1.0 Introduction

1.1 Purpose

This document summarizes a number of performance-enhancement techniques. Some of the techniques are generally applicable and some are specific to networking applications, the Intel XScale® Core, or the Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor.

The techniques suggested in this document are suggested solutions to the problems proposed and are provided for informational purposes only. There can be no guarantee that these proposed solutions will be applicable to your application or that they will resolve the problems in all instances.

The performance tests and ratings mentioned in this document are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing.

Most of these techniques fall into one of four categories:

- Determining the cause of a performance problem
- Identifying the solution for a problem
- Information, tasks and ways of organizing performance work
- Organizing information, tasks, and performance work

Intel has published a significant amount of information on performance optimization for the processors based on the Intel XScale core. This information is available in many of the documents listed in "Related Documents" on page 8. This document summarizes and collects in one location some of the information in these different references.

For more information on performance tests and on the performance of Intel products, visit the Web site http:\\www.intel.com or call (U.S.) 1-800-628-8686.

1.2 Scope

This document does not repeat in detail materials which you can obtain easily elsewhere. We provide references to that information. We have tried to cover the big picture of optimization techniques. As a result, the document contains suggestions ranging from the obvious and general [see "Compiler Optimizations" on page 16] to the specific [see "Intel XScale® Core PLD Instruction" on page 35].

1.3 Related Documents

Table 1. Related Documentation

Document Name	Document Number
Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor Developer's Manual	252480
Intel® IXP400 Software Programmer's Guide	252539
Intel® XScale™ Microarchitecture Programmer's Reference Manual	273436
Performance Profiling Techniques on Intel® XScale™ Microarchitecture Processors Application Note	273661
Intel 80200 Processor based on Intel® XScale™ Microarchitecture, Developer's Manual	273411
Coding Tips for Developers Targeting I/O Processors Based on the Intel® XScale™ Microarchitecture Application Note	273618
Intel® PXA250 and PXA210 Processors Optimization Guide	http://www.intel.com/design/pca/applicationsprocessors/manuals/27855201.pdf 27387202
Red Hat* Intel® XScale™ implementations of libc	see http://www.redhat.com
ARM Architecture Reference Manual	N/A
ARM Application Note 34, Writing Efficient C for ARM	http://www.arm.com/arm/documentation?OpenDocument ARM DAI 0034A
The Art of Designing Embedded Systems, Jack Ganssle	ISBN 0-75-069869-1
C Programming FAQs	ISBN 0-20-184519-9
Code Complete: A Practical Handbook of Software Construction, Steve McConnell	ISBN 1-55-615484-4
Design Patterns: Elements of Reusable Object-Oriented Software; Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides	ISBN 0-20-163361-2
Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks and Patterns, Bruce Powel Douglass	ISBN 0-20-149837-5
Inner Loops: A Sourcebook for Fast 32-Bit Software Design, Rick Booth	ISBN 0-20-147960-5
The Timeless Way of Building, Christopher Alexander	ISBN 0-19-502402-8
Wikipedia definition of "optimization"	http://www.wikipedia.org
Writing Efficient Programs (Prentice-Hall Software Series), Jon Louis Bentley	ISBN 0-13-970244-X

1.4 Audience

Intel Software Engineers	Intel software engineers can use this document to optimize existing or new access layer software they are developing. They might also use these tips to improve the performance of application or demonstration software they are writing.
Intel Engineers	Engineers can use this document as a checklist of questions to help customers optimize their application software. In addition, they could take this document and use it in an application note for publication to customers.
Eco-System Partners	The information in this document will be useful to eco-system partners to help them optimize their applications and operating systems for the IXP42X product line and IXC1100 control plane processors.
Customer Engineers	The information in this document will be useful to customer engineers to help them optimize their applications for the IXP42X product line and IXC1100 control plane processors.

1.5 Assumptions

We assume the reader has some networking and software development experience.

1.6 Acronyms

AAL	ATM Adaptation Layers
AES	Advanced Encryption Standard
AHB	Advanced High-Performance Bus
APB	Advanced Peripheral Bus
API	Application Programming Interface
AQM	AHB Queue Manager
Assert	The logically active value of a signal or bit.
ATM-TC	Asynchronous Transmission Mode – Transmission Convergence
BTB	Branch Target Buffer
Clean	An operation that updates external memory with the contents of the specified line in the data/mini-data cache if any of the dirty bits are set and the line is valid. There are two dirty bits associated with each line in the cache so only the portion that is dirty will get written back to external memory. After this operation, the line is still valid and both dirty bits are deasserted.
CLZ	Count Leading Zeros
Coalescing	Bringing together a new store operation with an existing store operation already resident in the write buffer. The new store is placed in the same write buffer entry as an existing store when the address of the new store falls in the four-word, aligned address of the existing entry. This includes, in PCI terminology, write merging, write collapsing, and write combining.
CRC	Cyclical Redundancy Check
DUT	Device Under Test
Deassert	The logically inactive value of a signal or bit.
DMA	Direct Memory Access
DSP	Digital Signal Processor
E1	Euro 1 trunk line
FAQ	Frequently Asked Questions
FCS	Frame-Check Sequence
FIFO	First In, First Out
Flush	An operation that invalidates the location(s) in the cache by de-asserting the valid bit. Individual entries (lines) may be flushed or the entire cache may be flushed with one command. Once an entry is flushed in the cache it can no longer be used by the program.
GCI	General Circuit Interface

GPIO	General-purpose input/output
G.SHDSL	ITU G series specification for Single-Pair HDSL
HDLC	High-level Data Link Control
HDSL	High-Bit-Rate Digital Subscriber Line
HDSL2	High-Bit-Rate Digital Subscriber Line, Version 2
HEC	Head-Error Correction
HPI	(Texas Instrument) Host Port Interfaces
HSS	High-Speed Serial (port)
ICE	In Circuit Emulator
IOM	ISDN Orientated Modular
ISDN	Integrated Services Digital Network
ISR	Interrupt Service Routine
LFSR	Linear Feedback Shift Register
LSb	Least-Significant bit
LSB	Least-Significant Byte
LUT	Look-Up Table
MAC	*** Multiply Accumulate
MDIO	Management Data Input/Output
MIB	Management Information Base
MII	Media-Independent Interface
MMU	Memory Management Unit
MSb	Most-Significant bit
MSB	Most-Significant Byte
MVIP	Multi-Vendor Integration Protocol
NPE	Network Processing Engine
NRZI	Non-Return To Zero Inverted
PCI	Peripheral Component Interconnect
PEC	Programmable Event Counters
PLD	Preload (instruction)
PLM	Product Line Marketing
PMU	Performance Monitoring Unit
PHY	Physical Layer (Layer 1) Interface
Reserved	A field that may be used by an implementation. Software should not modify reserved fields or depend on any values in reserved fields.

RMII	Reduced Media Independent Interface
RTOS	Real Time Operating System
RX	Receive
SFD	Start of Frame Delimiter
T1	Type 1 trunk line
TDM	Time Division Multiplex
TLB	Translation Look-Aside Buffer
TX	Transmit
UART	Universal Asynchronous Receiver-Transmitter
WAN	Wide Area Network

1.7 Conventions

Each performance improvement suggestion is documented in the form of a pattern. (See *Design Patterns: Elements of Reusable Object-Oriented Software*.) A pattern is "a Solution to a Problem in a Context," a literary mechanism to share experience and impart *solutions* to commonly occurring *problems*. Each pattern has a number of elements:

- **Name** — Name by which we can reference this problem/solution pairing.
- **Context** — The circumstance in which we solve the problem that imposes constraints on the solution.
- **Problem** — The specific problem that we need to solve.
- **Solution** — The proposed solution to the problem. Many problems may have more than one solution and the "goodness" of a solution to a problem is affected by the context in which the problem occurs. Each solution takes certain forces into account. It resolves some forces at the expense of others. It may even ignore some forces.
- **Forces** — The often-contradictory considerations we must take into account when choosing a solution to a problem.

2.0 Gathering Information and Requirements

2.1 Defined Performance Requirement

2.1.1 Context

You are a software developer starting a performance improvement task work on an application or driver.

2.1.2 Problem

Performance improvement work can become a never-ending task. Without a goal, the activity can drag on longer than productive or necessary.

2.1.3 Solution

At an early stage of the project or customer engagement, define a relevant specific, realistic, and measurable performance requirement.

Document that performance requirement as a specific detailed application and setup with a numerical performance target.

- "Make it as fast as possible" is not a specific performance requirement.
- "The application must be capable of wire-speed routing of 64-byte packets" is not a realistic performance requirement.

2.1.4 Forces

- A performance target can be hard to define.
- Waiting to have a goal may affect your product's competitiveness.
- A performance target can be a moving target; competitors do not stand still. New competitors come along all the time.
- Without a goal the performance improvement work can drag on longer than productive.

2.2 Gathered Customer Information

2.2.1 Context

You are a software, TME, customer, or eco-system engineer and a customer informs you of a performance problem they have uncovered.

2.2.2 Problem

You need to gather some information to help you start diagnosing the problem or enlist the help of others in solving the problem.

2.2.3 Solution

Gather or request the information from the customer that will help you to diagnose the performance problem. This is the information you will need to apply the rest of the patterns in this document. It provides you with a good checklist for an initial engagement with a customer who reports a performance problem.

Here are the essential questions you should ask, at an early stage:

- What RTOS is the customer using, what version?
- What networking stack is the customer using, what version?
- What compiler is the customer using, what version?
- What optimization compiler flags is the customer using?
- What version of the Intel® IXP400 Software is the customer using?
- What other major software components is the customer using, drivers? What are their versions?

- What benchmark or performance measurement is the customer making? What performance do they expect or require?
- What is the test setup? Request a diagram.
- How much work have they done already to tune performance? Have they done any performance tuning for the Intel XScale core or IXP42X product line and IXC1100 control plane processors?
- What is a high-level description of the data path and software components involved.
- Are they using the data cache? Are they using it for packets?
- Have they read and tried the optimization suggestions in the IXP42X product line and IXC1100 control plane processors' documentation?
- Have they identified their current performance bottleneck (dependency stalls, computation, and memory bandwidth)?
- Do they have any explicit delays on their data path code?
- How do they manage packet buffers? Do they pre-allocate buffers for the access layers? What do they do when they are finished with a packet buffer?

Here are some other useful questions you may ask, or ask on a second pass:

- Have they done any measurements of the clock cycles used by the different software components of their data path?
- Are they doing non-pre-fetch PCI reads in their data path?
- How does their main packet processing work? Is it polled or interrupt driven? Are there context switches involved? If it is interrupt-driven, how many packets does each interrupt handle?
- Have they used the PMU?
- Have they turned off access layer statistics/counters and parameter checks?
- Have they enabled the BTB?
- Do all packets go through the IP stack?
- Do they have an ICE connected when they run the performance test?
- Are they using separate memory banks for packets and code?
- If they are using caches are they using read or read-write line allocation? Have they tried the other?
- If they are using caches, are they using write-through or write-back mode? Have they tried the opposite of what they are using?
- Have they enabled write coalescing?
- Are they using the mini data cache?
- Have they done any instruction or function profiling?
- Have they used any explicit pre-load instructions?
- Have they identified the main source of data-dependency stalls?

2.2.4 Forces

- You need enough information to help you start diagnosing the problem or enlist the help of others in solving the problem.
- Asking too many questions can irritate a customer who may be under time pressure to get a product into production.

3.0 General Optimization Approaches

You can apply some general approaches to improve the overall performance of your application or selected parts of it.

3.1 Best Compiler for Application

3.1.1 Context

You are writing an application for a processor based on the Intel XScale core.

3.1.2 Problem

A number of compilers are available that have different levels of code generation for the Intel XScale core and instruction set. You need to select the right one for your application and target platform.

3.1.3 Solution

Experiment with different compilers and select the best performing compiler for your application and environment.

Performance can vary between compilers. For example, when we ran the Dhrystone* MIPS Benchmark on a 533-MHz Intel® IXP425 Network Processor, the following compilers had the following relative performance, at the time of writing.

- Greenhills* v3.61 (Green Hills Software, Inc.*)
- ADS* v1.2 (ARM Ltd.*)
- Tornado* 2.2 Diab* (Wind River Systems*)
- Tornado 2.2 gcc

The Intel XScale core GNU-PRO compiler on http:\\developer.intel.com has been measured to be approximately 10% better than the open source gcc compiler when measuring the Java* CaffineMark* benchmark.

Intel is currently working to improve the GNU-PRO compiler to further optimize its code generation for the Intel XScale core.

3.1.4 Forces

- Some compilers are more expensive than others.

- Some compilers will not be able to generate code for some operating systems. For example, the ADS compiler generates ELF binaries but Tornado 2.1.1 requires COFF binaries.
- A particular compiler may optimize a particular benchmark better than another compiler, but that is no guarantee it will optimize your application better.
- You may need an open-source compiler.

3.2 Compiler Optimizations

3.2.1 Context

You are using a C-compiler and you have selected the compiler [see "Best Compiler for Application" on page 15].

3.2.2 Problem

You have not enabled all of the compiler optimizations.

3.2.3 Solution

Your compiler will have a number of optimization switches. Using these switches may increase global application performance for a small amount of effort. Read the documentation for your compiler and understand these switches.

In general, the highest-level optimization switch is the `-O` switch. In gcc, it takes a numeric parameter. Find out the maximum parameter for your compiler and use it. Typically, there are three levels of compiler optimization, try the highest — this is `-O3`. If you have problems at the highest level drop the level down one.

Moving from `-O2` to `-O3` made an improvement of approximately 15% in packet processing in one application tested. In another application, `-O3` was slower than `-O2`.

You can limit the use of compiler optimizations to individual C source files.

Introduce optimization flags, one by one, to discover the ones that give you benefit.

Other gcc optimization flags that may increase performance are:

- `-funroll-loops`
- `-fomit-frame-pointer -mapcs`
- `-align-labels=32`

3.2.4 Forces

- Optimizations increase generated code size.
- Some optimizations may not increase performance.
- There are a large number of switches and options for the compiler. The documentation is very large.
- Optimized code is difficult to debug.

- Some optimizations may reveal compiler bugs (`-fomit-frame-pointer -mapcs` reveals a post-increment bug in gcc 2.95.X for the Intel XScale core).
- Enabling optimization will change timings in your code. It may reveal latent undiscovered problems.

3.3 Performance Design

3.3.1 Context

You are a software developer designing a system. You have a measurable performance requirement [see "Defined Performance Requirement" on page 12].

3.3.2 Problem

The design of the system may not meet the performance requirement.

3.3.3 Solution

At design time, describe the main data path scenario. Walk through the data path in the design workshop and document it in the high level design.

When you partition the system into components allocate a portion of the clock cycles to the data path portion of each component. Have a target at design time for the clock cycle consumption of the whole data path. Reference *The Art of Designing Embedded Systems* contains notations and techniques for system design performance constraints.

During code-inspections, hold one code inspection that walks through the most critical data path. Code inspections are usually component-based. This code inspection should be different and follow the scenario of the data path.

In polled environments ensure that the CPU is shared appropriately.

It can also be useful to analyze the application's required bus bandwidth at design time to decide if the system will be CPU or memory bandwidth/latency limited.

Pay special attention to packet-buffer management, at design time. Are buffers being allocated on the application data path? Where in the design are packet buffers replenished into the IXP400 software access layer?

3.3.4 Forces

- It can be difficult to anticipate some system bottlenecks at design time.
- The design of the system may make it impossible to meet the performance requirement. If you find this out late in the project, it may be too difficult to do anything about it.

3.4 Early Performance Measurement

3.4.1 Context

You are a software engineer. You have a functioning system and you are in the integration phase of development. Or you have a performance requirement [see "Defined Performance Requirement" on page 12] that your system does not meet.

3.4.2 Problem

It is not clear where the processor cycles are being spent in the data path of your application.

3.4.3 Solution

Measure the performance of your system relative to the performance requirement [see "Defined Performance Requirement" on page 12].

Measure the performance of the data path as soon as possible during the integration testing. Check that the components of the data path have conformed to their cycle count budgets set during high-level design [see "Performance Design" on page 17].

Trace through the whole data path scenario using a debugger to understand the data path.

Sprinkle clock-counting measurements at different points around the code of your application. Run the system for a short period and work out the proportion of time spent in the different parts of your data path. Do the cycle counts add up? Is there a missing block of time that you cannot account for? Is the processor spending its clock cycles where you expected?

In one case, we found a particular application spent 60+% of its cycles managing packet buffers with only 12% in the IXP400 software and 22% in a complex routing algorithm. This result surprised us and focused our attention on the buffer management code, in which we then identified an approximate 20% packet processing performance improvement.

You should scrutinize all use of task delays, and relative task priorities.

Confirm the performance limitation identified in "Performance Design" on page 17. SDRAM Controller access figures and the IXP42X product line and IXC1100 control plane processors' PMU can help you identify the north- and south-bus occupancy.

3.4.4 Forces

- Measuring clock cycle counts can change the behavior and performance of the system.
- The highest priority activity during integration test is the testing of functionality rather than performance.
- Changes post-integration may change the performance characteristics of your application.

3.5 PMU Performance Measurement

3.5.1 Context

You have a functional system, based on the Intel XScale core, that does not meet its performance requirements.

3.5.2 Problem

Some general processor related issues might be affecting the performance of your applications in a number of software components.

3.5.3 Solution

Use the Intel XScale core's PMU to identify some macro performance characteristics of your whole application. Some of these performance characteristics include:

- Instruction cache efficiency
- Data cache efficiency
- Data/Bus Request Buffer Full
- Stall/Write-Back Statistics
- Instruction/Data TLB Efficiency

This kind of analysis may help you to identify a subtle bottleneck. (For more details on these characteristics, see the *Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor Developer's Manual*.)

For PMU sample code, see *Performance Profiling Techniques on Intel® XScale™ Microarchitecture Processors Application Note*. It is anticipated that a future release of the IXP400 software will provide a C-function API to simplify access to the PMU.

In most applications, this test will identify problems with data dependency stalls, but it is a good idea to test all of the characteristics as soon as possible to discount some possible factors early. For example, we have not seen instruction cache being an issue in any applications at the time of writing.

At higher clock speeds (400 MHz and 533 MHz), you may find a significant number of "cycles lost due to data-dependency stalls." If you measure this number on a 533-MHz IXP425 network processor and it is above 50%, this indicates an area for further investigation. Many of the later sections of this document focus on strategies to reduce lost cycles waiting for data.

A word of warning, it can be dangerous to try to optimize these characteristics individually. In one test case, an optimization improved packet processing performance, but made the "cycles lost due to data-dependency stalls" measurement worse.

3.5.4 Forces

- High-level PMU measurements can be misleading.
- Optimizing PMU performance indicators may not optimize networking performance.

3.6 Data Cache

3.6.1 Context

You are using a processor based on the Intel XScale core. The Intel XScale core is running faster than memory or peripherals.

3.6.2 Problem

The Intel XScale core is spending a significant amount of time stalled waiting on an external device. This affects the performance of the core. You have identified this problem using performance measurements [see "PMU Performance Measurement" on page 19] quantifying the number of stall cycles. In some applications, we have observed a significant number of cycles are lost to data-dependency stalls.

3.6.3 Solution

In general, the most efficient mechanism for accessing memory is to use the data cache. Core accesses to cached memory do not need to use the Internal Bus, leaving it free for other devices. In addition, accessing data in cache memory is faster than accessing it from the SDRAM.

The cache unit can make efficient use of the internal bus. The core fetches an entire cache line (32 bytes), making use of multiple data phases. This reduces the percentage of overhead cycles required for initiating and terminating bus cycles, when compared with issuing multiple bus cycles to read the same 32 bytes of data without using the cache.

The cache has several features to allow the system designer flexibility in tailoring the system to design needs. These features affect all applications to some degree; however the optimal settings are application-dependent. It is critical to understand the effects of these features and how to fine-tune them for the usage-model of a particular application. We cover a number of these cache features in later sections.

In one application, which was not caching mbufs and packet data, developers enabled caching and saw an approximate 25% improvement in packet-processing performance.

Tornado 2.1.1 netBufLib does not allocate mbufs and packet memory from cached memory. It is expected that Wind River Systems will provide a patch for Tornado 2.2 to fix this issue for the IXP42X product line and IXC1100 control plane processors.

In most of the applications we have seen, the instruction cache is very efficient. It is worth spending time optimizing the use of the data cache.

3.6.4 Forces

- If you cache data-memory that the core shares with another device, the programmer needs to manage cache flush/invalidation explicitly.
- If you use caching you need to ensure no two-packet buffers ever share the same cache line. This can lead to bugs that are difficult to diagnose.
- If you use the cache heavily, its round-robin replacement algorithm may cause performance sensitive data to be evicted by lower priority data.

3.7 ICE Disabled

3.7.1 Context

You are optimizing the performance of a network application. You are still using an ICE to download code to your DUT.

3.7.2 Problem

The ICE can in some circumstances, affect the performance of your application and processor

3.7.3 Solution

Get to the point where you can boot and download code to the DUT standalone without support of an ICE.

In one networking application, we observed an approximate 10% performance improvement when we disconnected the ICE.

3.7.4 Forces

An ICE may be required early in a project.

4.0 General Networking Performance

The following patterns can be applied to networking performance in general. They are not typically specific to the IXP42X product line and IXC1100 control plane processors.

4.1 Bottleneck Hunting

4.1.1 Context

You have a running functional system. You have a performance requirement [see "Defined Performance Requirement" on page 12]. A customer is measuring performance that is less than that requirement.

4.1.2 Problem

You may have a number of performance bottlenecks in the designed system but unless you identify the current limiting factor, you may optimize the wrong thing. One component of the system may be limiting the flow of network packets to the rest of the system.

4.1.3 Solution

Performance improvement really starts with bottleneck hunting. It is only when you find the performance-limiting bottleneck, that you can then work on optimizations to remove the bottleneck.

First look at the software components of the data path, these might include:

- Low-level device drivers specific to a piece of hardware. These device drivers may conform to an OS-specific interface
- A network interface service mechanism running on the Intel XScale core. This may be a number of ISRs or it may be a global polling loop
- Adaptor components or glue code that adapt the hardware-specific drivers or the underlying IXP400 software APIs to an RTOS or network stack
- Encapsulation layers of the networking stack
- The switching/bridging/routing engine of the networking stack or the RTOS
- IXP400 software access APIs. These functions provide an abstraction of the underlying firmware and silicon
- IXP42X product line and IXC1100 control plane processors' NPE firmware

If some algorithm in a low-level device driver is limiting the flow of data into the system, you will waste your time if you start tweaking compiler flags or optimize the routing algorithm.

It is best to look at the new or unique components to a particular system first. Typically these are the low level device drivers or the adaptor components that are unique to this system. Other projects may have already used the routing-algorithm, IXP400 software and NPE firmware. Concentrate on the unique components first especially if these components are on the edge of the system. In one wireless application we discovered the wireless device driver was a bottleneck that limited the flow of data into the system.

Many components of a data path may have packet buffers. Packet counters inserted in the code may help you identify queue overflows. Typically, the code that consumes the packet buffer is the bottleneck.

This is typically an iterative cycle. When you fix the current bottleneck, you then need to loop back and identify the next one.

4.1.4 Forces

- Most systems have multiple bottlenecks.
- Early bottleneck hunting — before you have a complete running system — increases the risk of misidentified bottlenecks and wasted tuning effort.

4.2 Evaluating Traffic Generator/Protocols

4.2.1 Context

You are using a network-traffic generator and protocols to measure the performance of a system.

4.2.2 Problem

The performance test environment can limit the measured performance of your application.

4.2.3 Solution

Identifying the first bottleneck is a challenge. First, you need to eliminate your traffic generators and protocols as bottlenecks, analyze the invariants.

Typical components in a complete test system might include:

- Traffic sources, sinks, and measurement equipment
- The device under test (DUT), part of which you are changing
- Physical connections and protocols between traffic sources and the DUT

There are a number of different types of traffic sources, sinks and measurement equipment. You need to first make sure they are not the bottleneck in your system.

Equipment like Smartbits* and Adtech* testers are not typically bottlenecks.

However, if you are using a PC with FTP software to measure performance, this may be a bottleneck. You need to test the PC and FTP software without the DUT to make sure your traffic sources can reach the performance you require.

Running this test may also flush out bottlenecks in the physical media or protocols, you are using.

In addition you need to make sure the overhead inherent in the protocols you are using make the performance you require feasible. For example:

- You cannot expect 100 Mbps over Ethernet with 64-byte packets, due to inter-frame gap and frame preamble. You will get at most 76 Mbps.
- You cannot expect to get 8 Mbps over an ADSL link; you will get at most 5.5 Mbps.
- You cannot expect to get 100 Mbps on FTP running over Ethernet. You have IP protocol overhead and TCP acknowledgements to take into account.
- You cannot expect 52 Mbps on 802.11a/g networks due to CTS/RTS overhead and protocol overhead.

Environmental factors may also be causing the bottleneck. When testing a wireless application you may have radio interference in the test environment. In this case, you may need a Faraday cage to radio-isolate your test equipment and DUT from the environment. Antenna configuration is also important. The antennas should not be too close (<1 meter). They should be standing up, not lying down. You may also need to make sure the DUT has some shielding to protect it from antenna interference.

The particular protocol or application could also be causing the bottleneck. For example, if the FTP performance is much lower (by a factor of 2) than the large packet performance with a traffic generator (Smartbits*), this may indicate a problem where the TCP acknowledgement packets are getting dropped which turned out to be another buffer management issue [see “Packet Buffer Management Analysis” on page 25].

FTP performance can also be significantly affected by the TCP window sizes on the FTP client and server machines.

Check all connectors and cables. If you are confident you are making improvements but the measurements are not giving the improvement you expect, try changing all the cables connecting your DUT to the test equipment. As a last resort try a replacement DUT.

4.2.4 Forces

Test equipment typically outperforms the DUT.

4.3 Throughput-Limiting Packet Loss

4.3.1 Context

You are testing the performance of a functioning system using a throughput test. You might be using the Spirent* SmartApplications* throughput test.

4.3.2 Problem

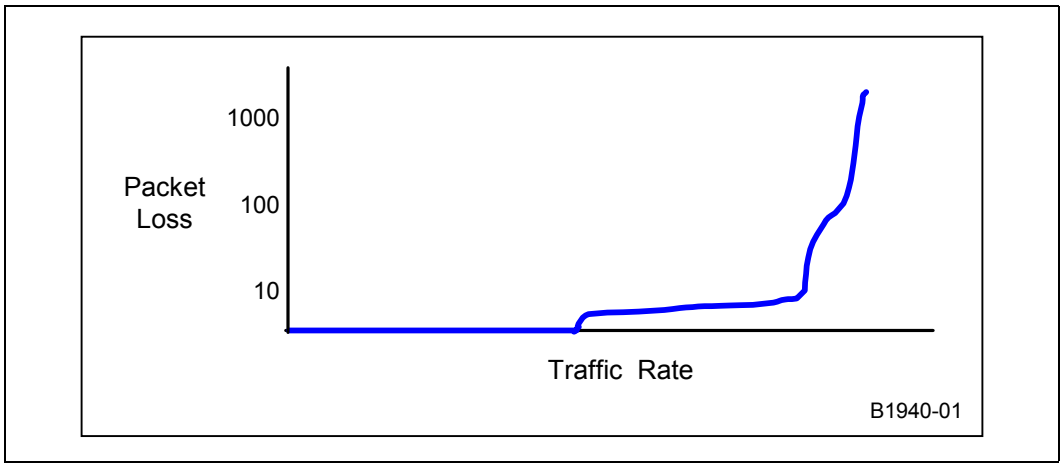
The throughput is much lower than you expect. Making code more efficient does not help increase throughput.

4.3.3 Solution

Graph packet loss over a longer test period at a number of rates leading up to your current throughput bottleneck.

When you draw this graph, you may — if your system has this problem — see a graph that looks approximately like the one shown in Figure 1 on page 24.

Figure 1. Throughput-Limiting Packet Loss



If you see this kind of a graph, it could indicate that a small amount of packet loss is occurring well before the system reaches its full processing capability. This small amount of packet loss will make the throughput look worse than it actually is.

This can help you look in the right place for the source of the problem. Optimizing code may not help; it may be an architectural problem.

Doing this test will help you identify the true cause of a throughput problem.

In one case where we applied this technique, we identified a cause of packet loss which when fixed increased the throughput measured by SmartApplications by approximately 60%.

4.3.4 Forces

- Throughput tests are typically sensitive to small amounts of packet loss.
- The packet processing mechanism of some applications can cause some small amounts of packet loss before the application reaches a throughput bottleneck.

4.4 Packet Buffer Management Analysis

4.4.1 Context

You are in the early stages of optimization of networking application. You have considered buffer management during performance design.

4.4.2 Problem

It can be computationally expensive to use system packet buffer pools. For example, putting packet buffers into the RTOS buffer library (i.e. netBufLib in VxWorks*) can incur interrupt locks. In other cases, the IXP400 software may not be replenished with RX buffers quickly enough.

4.4.3 Solution

Analyze the application's packet buffer management, especially on the performance critical data path code of your application.

If possible you should try to pre-allocate buffers to the driver layer above the access layer API. Storing the collection of these buffers in a stack rather than a linked list can minimize the number of operations that can potentially go to external SDRAM.

Packet buffer management has been the root cause of a significant number of performance issues we have analyzed to date.

The packet buffers you replenish to the access layer software are used by the NPEs to queue received packets. If the access layer runs low on replenished buffers, packets will be dropped. If your software does not handle Rx callbacks in a timely manner packets can also be dropped.

A delay in handling TxDone callbacks can also cause a bottleneck that can affect the receive path as the access layer may reject newly submitted packets because its internal queues are full. This means a number of buffers are unavailable for recycling to the packet Rx service.

In one particular application, when we analyzed buffer management, we identified improvements that yielded an approximate 20% packet processing performance improvement.

4.4.4 Forces

Sometimes you do want to drop packets to handle overload scenarios without wasting CPU cycles. For details, see "Edge Packet Throttle" on page 27.

4.5 Polled Packet Processor

4.5.1 Context

You are designing the fundamental mechanism that drives the servicing of network interfaces.

4.5.2 Problem

Some fundamental mechanisms can expose you to more overhead and wasted CPU cycles. These wasted cycles can come from interrupt preamble/dispatch and context switches.

4.5.3 Solution

You can categorize most applications as interrupt or polling driven or a combination of both.

When traffic overloads a system, it will run optimally if it is running in a tight loop, polling interfaces for which it knows there is traffic queued.

If the application driver is interrupt-based, you need to look to see how many packets you handle per interrupt. To get better packet processing performance handle more packets per interrupt (possibly using a polling approach in the interrupt handler).

Some systems put the packet on a queue from the interrupt handler and then do the packet processing in another thread. In this kind of a system, you need to understand how many packets the system handles per context switch. To improve performance increase the number of packets handled per context switch.

Other systems may drive packet processing, triggered from a timer interrupt. In this case, you need to make sure the timer frequency and number of packets handled per interrupt is not limiting the networking performance of your system. In addition, this system is not optimally efficient when the system is in overload.

Applying these techniques may increase the latency in handling some packets.

Systems based on Linux* are usually interrupt-based.

4.5.4 Forces

- Reducing wasted CPU cycles may complicate the overall architecture or design of an application.
- Some IP stacks or operating systems can restrict the options in how you design these fundamental mechanisms.
- You may need to throttle the amount of CPU given to packet processing to allow other processing to happen even when the system is in overload.
- You want to minimize the latency in processing packets.

4.6 Fast Path

4.6.1 Context

You are writing the glue code above the IXP400 software access layer API to send packets in and out of a network stack. You have found the network stack is using a significant proportion of the CPU cycles [see "Early Performance Measurement" on page 18].

4.6.2 Problem

Third-party and operating-system stacks can take a long time to decide where to send a packet and they may use a number of interrupt locks or context switches.

4.6.3 Solution

If possible, avoid sending all packets through the full stack. If the packet is destined for another interface on the processor, you may be able to modify and transmit the packet without sending it to the main data stack. You can translate the main packet flow into a simple pattern match (classifier) and change (modifier) algorithm. In one application we observed an approximate 650% performance gain using this technique. In this application, we had found the IP stack on a particular RTOS was consuming 70+% of the CPU cycles.

There are two types of fast path, one that runs on the Intel XScale core and the other is offloaded to the NPE engines.

Fast paths are usually application-specific and as such, customers and third-party partners are in the ideal position to construct core processor fast-paths.

The IXP42X product line and IXC1100 control plane processors have an NPE accelerated fast-path that can classify and modify packets from the UTOPIA interface directly to one of the Ethernet ports. This facility is customizable and configurable with the IXP400 software access-layer API. An NPE fast-path can give you an even bigger performance gain than a fast-path running on the Intel XScale core. The current existing NPE fast-path is restricted to the above interfaces.

Some of the access components (such as Ethernet) only allow one piece of client code. This may mean you have to add a layer of software on top of the access driver to support more than one client.

4.6.4 Forces

- Many systems and stacks have a common driver model that has a different implementation for each specific interface.
- Developing a new fast-path will take less effort on the Intel XScale core than on the NPE, but it will not have the same performance.

4.7 Edge Packet Throttle

4.7.1 Context

The bottleneck of your system is now the IP forwarding or transport parts of the IP stack.

4.7.2 Problem

You may be wasting CPU cycles processing packets to later drop them when a queue later in the data path fills.

4.7.3 Solution

When a system goes into overload, it is better to leave the frames back up in the RX queue and let the edges of your system (NPE, PHY devices) throttle reception. You can avoid wasting core cycles by checking a bottleneck indicator (i.e. queue full) early in the data path code.

On VxWorks, you can make netTask the highest priority task. This is one easy way to implement a "self-throttling" system. Alternatively, you could make the buffer replenish code a low-priority task which would ensure free buffers are only supplied when you have available CPU.

4.7.4 Forces

- Checking a bottleneck indicator may weaken the encapsulation of an internal detail of the IP stack.
- Implementing an early check will waste some CPU cycles when the system is in overload.

4.8 Packet Buffer Headroom

4.8.1 Context

You are allocating packet buffers for a networking application.

4.8.2 Problem

After the packet is received, it is typically modified before being forwarded to another address. Some packet modifications will add new data at the start or end of the existing packet. If you have no room in your packet buffer, the stack may allocate a new packet buffer and chain the packet, or it may copy the packet to larger storage. In either case valuable CPU cycles will be consumed.

4.8.3 Solution

All packet buffers allocated for network traffic should be 2048 bytes in size and should have store the packet data initially at a 128-byte offset into the buffer. (In mbuf terminology, m_data is offset.) This will allow the insertion of encapsulation or headers without chaining.

4.8.4 Forces

Allocating headroom will waste some memory.

4.9 Detecting Resource Collisions

4.9.1 Context

You make a change and performance drops unexpectedly.

4.9.2 Problem

A resource collision effect could be causing a pronounced performance bottleneck. Examples of such effects we have previously encountered are:

- TX traffic is being generated from RX traffic, Ethernet is running in half-duplex mode. The time it takes to generate the TX frame from an RX frame corresponds to the inter-frame gap. When the TX frame is sent it collides with the next RX frame.
- The Ethernet interface is running full-duplex, but traffic is being generated in a loop and frame transmission collides with times that the MAC is busy receiving frames.

4.9.3 Solution

These kinds of bottlenecks are difficult to find and can only be checked by looking at counters drivers, IXP400 software and underlying PHY devices. Error counters on test equipment may also help.

4.9.4 Forces

Counters may not be available or easily accessible.

5.0 Intel XScale® Core and Device-Specific Tuning

The following sections are specific to the Intel XScale core and the IXP42X product line and IXC1100 control plane processors.

5.1 Devices' Silicon Features

5.1.1 Context

You have identified the processing power of the Intel XScale core as the bottleneck in your application.

5.1.2 Problem

You are using cycles on the Intel XScale core that could be offloaded by the NPEs.

5.1.3 Solution

Get to know the IXP42X product line and IXC1100 control plane processors and firmware feature set. IXP42X product line and IXC1100 control plane processors have a number of processor elements that can operate in parallel to the Intel XScale core. Three NPEs and a DSP coprocessor can offload some processing from the Intel XScale core and may increase the performance of your application. Some of these features include:

- NPE Ethernet learning/filtering
- NPE crypto algorithm offload
- NPE fast-path from AAL-5 to Ethernet
- DSP coprocessor MAC instructions
- DSP voice CODEC implementations
- NPE Software DMA
- NPE ATM packet SAR
- NPE HDLC processing

If you believe an application may require some features offloaded to the NPE identify this potential feature request early in the development cycle.

For new applications that are bound to the Intel XScale core's CPU, it may be feasible for Intel to design new offload functionality into the NPE. This might include features such as IP checksum and TTL update (estimated to improve routing performance by 4%), 802.11 frame conversion. This kind of work needs some time to evaluate and implement.

5.1.4 Forces

- Integrating offloaded features may take some effort. Typically you need to replace or glue in the offloaded feature to an existing code base or software stack.
- New NPE features can only be implemented by Intel or some eco-system partners.

5.2 Understanding the Devices

5.2.1 Context

You are starting to work with the IXP42X product line and IXC1100 control plane processors.

5.2.2 Problem

It is difficult to optimize an application for a processor you do not understand.

5.2.3 Solution

You must understand the powerful device you are using. Extensive documentation is available for the IXP42X product line and IXC1100 control plane processors and software.

Read *Intel® IXP425 Network Processor Based on Intel® XScale™ Microarchitecture, Technical Specification*, May 2002, *ARM Architecture Reference Manual* (2nd Edition), and *Intel® IXP425 Network Processor Based on Intel® XScale™ Microarchitecture Developer's Manual*.

5.2.4 Forces

The documentation is “extensive.” Concentrate initially on the introductory sections.

5.3 Branch Target Buffer

5.3.1 Context

You are developing software for a processor based on the Intel XScale core.

5.3.2 Problem

The Intel XScale core disables the BTB by default after reset and it is invalidated when software invalidates the instruction cache.

5.3.3 Solution

The BTB needs to be explicitly enabled when the IXP42X product line and IXC1100 control plane processors come out of reset.

The Intel XScale core uses dynamic branch prediction to reduce the penalties associated with changing the flow of program execution. The Intel XScale core features a branch target buffer that provides the instruction cache with the target address of branch type instructions.

Many implementations of board support packages for IXP42X product line and IXC1100 control plane processors do not enable the BTB.

This change was tried in two applications and we observed a 4-5% performance improvement.

For more information on the BTB, see *Intel® IXP425 Network Processor Based on Intel® XScale™ Microarchitecture, Technical Specification*, May 2002.

5.3.4 Forces

You do not have to manage the BTB explicitly by software but you do need to enable it during initialization.

5.4 Latest Intel® IXP400 Software Access Layer

5.4.1 Context

You have a functional system that does not meet its performance requirements.

5.4.2 Problem

Your latest performance data points to the IXP400 software access layer as the performance bottleneck.

5.4.3 Solution

Check to make sure you have the latest IXP400 software access layer release. Each IXP400 software release has a number of performance improvements. For example, a future release will have a polling Ethernet access layer interface that consumes less than 500 clock cycles per packet on a 533-MHz IXP425 network processor.

You can download the latest release from this Web site:

http://www.intel.com/design/network/products/npfamily/ixp425swr1.htm

5.4.4 Forces

Upgrading to the latest IXP400 software release can take some engineering and validation effort.

5.5 Disabled Counters/Statistics

5.5.1 Context

The IIXP400 software access layer is part of the data path. You have completed integration testing of your application.

5.5.2 Problem

The IXP400 software access layer keeps a number of counters and statistics to facilitate integrating and debugging of both the access layer and the system as a whole. These counters will usually incur a read and write or increment in main (possibly cached) memory.

5.5.3 Solution

You can disable many of the internal IXP400 software counters and statistics by un-defining some macros.

For Ethernet, turn off `IX_ETH_ACC_DATA_PLANE_STATS_ON` in ethAcc/include/IxEthAccDataPlane_p.h.

For the QMGR, set `IX_QMGR_STATS_UPDATE_ENABLED` to zero in qmgr/IxQMgrDefines_p.h.

Other components will have their own #defines to enable/disable statistics.

In one application, use of this pattern increased packet processing throughput by up to 3%.

Future releases will simplify the global disabling of counters and statistics in the access-layer software.

5.5.4 Forces

Disabling counters and statistics will remove useful debugging information.

5.6 Disabled Parameter Checks

5.6.1 Context

The IXP400 software access layer is part of the data path. You have completed integration testing of your application.

5.6.2 Problem

The access layer, by default, has code that checks parameters for legal values. This facilitates the integration and debugging of both IXP400 software and the system as a whole. These checks will usually check conditions that never occur once the system and customer code has been fully tested and integrated.

5.6.3 Solution

You can disable many of these parameter checks by un-defining some macros.

For Ethernet data path, turn off `IX_ETH_ACC_DATA_PLANE_FUNC_ARG_CHECKS` in ethAcc/include/IxEthAccDataPlane_p.h.

For the QMGR, set `IX_QMGR_PARM_CHECKS_ENABLED` to zero in qmgr/IxQMgrDefines_p.h.

Other components will have their own #defines to enable/disable parameter checking.

Application of this pattern increased packet processing throughput by ~1% in one application. This pattern does not provide as much improvement because the parameters being checked are usually in registers [see “Understanding the Devices” on page 30].

Future releases will simplify the global disabling of parameter checks in the access-layer software.

5.6.4 Forces

Removing these checks may obfuscate an issue making it harder to detect incorrect parameter checks in customer code.

5.7 Stall Instructions

5.7.1 Context

You have run some tests using performance measurements [see “PMU Performance Measurement” on page 19] that indicate a large number of Intel XScale core cycles are being lost due to data dependency stalls.

5.7.2 Problem

You may find over 50% of the cycles are lost to stalls on a 533-MHz processor. You need to identify the pieces of code that are causing these stalls.

5.7.3 Solution

One simple way to identify "hot instructions" is to use a program counter sampler. The sampler would run at a regular interval and count the number of times each instruction is executed while running the networking performance test.

If you run the test for a significant period of time you should see a large number of samples on the instructions that stall most often.

The reference *Performance Profiling Techniques on Intel® XScale™ Microarchitecture Processors Application Note*, Aug. 2002 contains significant PMU example code which includes a PC sampler. It is anticipated that a future release of the IXP400 software will provide a C-function API to simplify access to the PMU.

You can then use other patterns to reduce the impact of these stalls. See the following sections for examples.

- "Intel XScale® Core PLD Instruction" on page 35
- "Stall-Filling Code" on page 46
- "Queue Look-Ahead" on page 44

5.7.4 Forces

Adding sampling code can affect the behavior of the system under test.

5.8 Profiling Tools

5.8.1 Context

You are at an early stage of performance improvement and have not identified a specific bottleneck but you have proven the current bottleneck is the speed of execution of the code on the Intel XScale core.

5.8.2 Problem

You have a working system that is not meeting a performance requirement. You suspect raw algorithmic processing power is the current bottleneck; you need to identify the bottleneck code.

5.8.3 Solution

A number of profiling tools exist to help you identify code hotspots. Typically, they identify the percent of time spent in each C-function in your code base.

- Rational Quantify* contains an excellent performance profiler (not to mention Purify*, the memory corruption/leak checker). It is a useful tool for finding where application bottlenecks are. The usage model is very similar to that used to gather code coverage: You instrument your code and then execute that code on the board.
- Gprof is available for many Linux-based systems.
- Some ICEs have profiling tools (i.e., visionClick*).

5.8.4 Forces

- Profiling tools can affect the performance of the system.
- Some tools will not be available for your RTOS.
- Some profiling tools cost money.
- Each of these tools have a learning curve but could pay back the time and money investment.

5.9 Intel XScale® Core PLD Instruction

5.9.1 Context

You have identified a stall instruction [see “Stall Instructions” on page 33].

5.9.2 Problem

You want to reduce the time the processor spends stalled due to a data dependency.

5.9.3 Solution

The IXP425 network processor has a true prefetch load instruction (PLD). The purpose of this instruction is to preload data into the data and mini-data caches. If the application is bounded by the memory latency, prefetch can effectively hide the memory latency.

Data prefetching allows hiding of memory transfer latency while the processor continues to execute instructions. The judicious use of the prefetch instruction can enormously improve throughput performance of the IXP425 network processor.

Look at the line of C-code that generates the stall instruction [see "Stall Instructions" on page 33]. Insert an explicit assembly language PLD instruction some time before the stall instruction [see Stall Instructions]. On a 533-MHz IXP425 network processor, you could issue the PLD 70 to 90 clock cycles before the stall.

Data prefetch can be applied not only to loops but also to any data references within a block of code. Prefetch also applies to data writing when the memory type is enabled as write-allocate.

The IXP425 network processor prefetch load instruction is a true prefetch instruction because the load destination is the data or mini-data cache and not a register. The prefetch load is a hint instruction and does not guarantee that the data will be loaded.

Using pre-fetches requires careful experimentation. In some cases we did find performance improvements and in others performance degraded.

Overuse of pre-fetches can use shared resources and degrade performance. This can happen if the bus traffic requests exceed the system resource capacity, the processor stalls. The IXP425 network processor data transfer resources are:

- Four fill buffers
- Four pending buffers
- Eight half-cache line write buffer

SDRAM resources are typically:

- Four memory banks
- One page buffer per bank referencing a 4-K address range
- Four transfer request buffers

Spread prefetch operations over calculations so as to allow bus traffic to free flow and to minimize the number of necessary pre-fetches.

5.9.4 Forces

- Overuse of pre-fetches can use shared resources and degrade performance.
- The pre-fetch is an instruction specific to the Intel XScale core. To make the code portable the instruction needs to be implemented by a `#define`.

5.10 Separate SDRAM Memory Banks

5.10.1 Context

You have completed a performance measurement [see "PMU Performance Measurement" on page 19]. This data has identified a significant percentage of cycles have been lost due to data dependency stalls.

5.10.2 Problem

SDRAMs are typically divided into four banks. Thrashing occurs when subsequent memory accesses within the same memory bank access different pages. The memory page change adds three to four bus-clock cycles to memory latency.

5.10.3 Solution

This type of thrashing can be resolved by placing the conflicting data structures into different memory banks or by paralleling the data structures such that the data resides within the same memory page. This can reduce the latency reading data from memory and reduce the extent of many stalls.

Allocate packet buffers in their own bank. The SDRAM controller can keep a page partially open in four different memory banks. You could also split packet buffers across two banks.

It is also important to ensure that instruction and data sections are in different memory banks, or they will continually thrash the memory page selection.

In one networking application this technique increased packet processing performance by approximately 10%. In another it had no effect.

5.11 Line-Allocation Policy

5.11.1 Context

You are using data cache for data or packet memory. The cache is enabled.

5.11.2 Problem

The cache line-allocation policy can affect the performance of your application.

5.11.3 Solution

The Intel XScale core makes a decision about placing new data into the cache based on the “line-allocation policy.”

If the line-allocation policy is read-allocate, all load operations that miss the cache, request a 32-byte cache line from external memory and allocate it into either the data cache or mini-data cache. Store operations that miss the cache will not cause a line to be allocated.

With a read/write-allocate policy, load or store operations that miss the cache will request a 32-byte cache line from external memory if the cache is enabled.

Most of the regular data and the stack for your application should be allocated to a read-write allocate region. Most applications will often write and read this data.

Write-only data (or data that is written and subsequently not used for a long time) should be placed in a read allocate region. Under the read-allocate policy, if a cache write miss occurs a new cache line will not be allocated, and hence will not evict critical data from the data cache.

In general we have found read-allocate to be the best performing policy for packet data. In one application, we saw an improvement of approximately 10% when packet memory was set up read-allocate.

5.11.4 Forces

The appropriate cache line-allocation policy can be application-dependent. It is worth experimenting with both types of line-allocation policies.

5.12 Cache Write Policy

5.12.1 Context

You are using data cache for data or packet memory. The cache is enabled.

5.12.2 Problem

The cache write policy can affect the performance of your application.

5.12.3 Solution

Cached memory also has an associated write policy. A write-through policy instructs the data cache to keep external memory coherent by performing stores to both external memory and the cache. A write-back policy only updates external memory when a line in the cache is cleaned or needs to be replaced with a new line.

Generally, write-back provides higher performance because it generates less data traffic to external memory.

In a multiple-bus/master environment, it may be necessary to use a write-through policy or explicit cache flushes, if data is shared across multiple masters.

5.12.4 Forces

The appropriate cache write policy can be application-dependent. It is worth experimenting with both types of write policies.

5.13 Write Coalescing

5.13.1 Context

You are optimizing performance for a processor based on the Intel XScale core. A performance critical part of the application does multiple writes to memory locations close together.

5.13.2 Problem

Multiple writes consume clock cycles.

5.13.3 Solution

Write coalescing allows you to bring together a new store operation with an existing store operation already resident in the write buffer. The new store is placed in the same write buffer entry as an existing store when the address of the new store falls in the four-word, aligned address of the existing entry.

The K bit in the Auxiliary Control Register (CP15, register 1) is a global enable/disable for allowing coalescing in the write buffer. When this bit disables coalescing, no coalescing will occur regardless the value of the page attributes. If this bit enables coalescing, the page attributes X, C, and B are examined to see if coalescing is enabled for each region of memory.

Write coalescing can only be used on memory that is free of side effects.

5.13.4 Forces

If coalescing is enabled in the write buffer, writes may occur out of program order to external memory. You may need to perform explicit drain operations of the write buffer and fill buffer to maintain consistency between the store requests and the fill buffer.

5.14 Faster Memory

5.14.1 Context

You are designing a board and have an option to use CAS 2 or 3 SDRAM.

5.14.2 Problem

Memory latency affects the performance of applications.

5.14.3 Solution

The IXP42X product line and IXC1100 control plane processors can work with memory with CAS latency 2 or 3 memory. CAS 2 memory will provide better general performance by reducing the latency in accessing SDRAM.

In one application, the overall packet processing improvement was estimated at approximately 8%, when CAS 2 memory is used.

5.14.4 Forces

CAS-2 memory is generally more expensive and it may be difficult to source extended temperature variants.

5.15 Cache-Aligned Packet Buffers

5.15.1 Context

Your application/driver caches packet buffers and buffer descriptors.

5.15.2 Problem

You need to use the cache as effectively as possible. On some systems, the descriptors may be larger than a cache line.

5.15.3 Solution

Allocate IX_MBUFs, clBlks (BSD) and packet data on cache line boundaries. This will maximize the use of cache when accessing these data structures.

Care must be taken to make sure the descriptors and packet storage for different packets do not share the same cache line. If they do, and they are cacheable, this can be the cause of subtle difficult-to-find bugs.

Using this pattern can also simplify the addition of an instruction [see "Intel XScale® Core PLD Instruction" on page 35].

5.15.4 Forces

You may waste some memory if the size of these data structures in your operating system is not divisible by the cache line size. Typically this memory wastage is worth the increase in performance.

5.16 On-Chip Memory

5.16.1 Context

You have a piece of data that is frequently used by the code on your data path.

5.16.2 Problem

Accesses to this data are causing stalls because the cache is heavily used and the data is being frequently evicted. Due to the Intel XScale core's round-robin replacement cache policy, all cache data that is not locked is eventually evicted.

5.16.3 Solution

You can lock tags associated with 32-byte lines in the data cache, thus creating the appearance of data RAM. Any subsequent access to this line will always hit the cache unless it is invalidated.

There are two methods for locking tags into the data cache; the method of choice depends on the application.

One method is used to lock data that resides in external memory into the data cache and the other method is used to re-configure lines in the data cache as data RAM.

Locking data from external memory into the data cache is useful for lookup tables, constants, and any other data that is frequently accessed. Re-configuring a portion of the data cache as data RAM is useful when an application needs scratch memory (bigger than the register file can provide) for frequently used variables. These variables may be strewn across memory, making it advantageous for software to pack them into data RAM memory.

In order to reduce cache pollution between two processes and avoid frequent cache flushing during context switch, the OS could potentially lock critical data sections in the cache.

You can also lock blocks of instructions into the instruction cache. However a case has not been found yet where the instruction cache was being overloaded to this extent.

5.16.4 Forces

Locking data into the cache reduces the amount of cache available to the processor for general processing.

5.17 Mini-DCache

5.17.1 Context

You are optimizing performance on a processor based on the Intel XScale core. The data cache is being used heavily.

5.17.2 Problem

Frequently used data is being constantly evicted by temporarily cached memory such as packet data.

5.17.3 Solution

Use the mini-dcache. It is a 2-K block of fast memory. You can use it to reduce the pressure on the main dcache.

You can map the following kinds of data to mini-dcache:

- Put your stack into the mini-dcache. One application saw an approximate 10% performance improvement using this technique. Another application saw no improvement.
- Put packet data into the mini-dcache. One application saw an approximately 5% performance improvement using this technique.
- Use the mini-dcache for frequently used tables.

5.17.4 Forces

The mini-dcache may not be available in future iterations of the Intel XScale core.

5.18 Optimized Libraries

5.18.1 Context

You are optimizing an application for the Intel XScale core. Your data path uses some of the standard C library functions (for example, memcpy, memcmp).

5.18.2 Problem

Some compilers come with default C implementations of the standard C libraries. These may not run optimally on the Intel XScale core.

5.18.3 Solution

Intel XScale core-optimized implementations of some of the standard C libraries are available both from Intel and the Internet.

For example, Red Hat* has a number of libc functions, specific to the Intel XScale core, on their Web site. [See *Red Hat* Intel® XScale™ implementations of libc*.] A memcpy optimized for the little-endian PXA250 and PXA210 processors, based on the Intel XScale core, is detailed in the appendix of the *Intel® PXA250 and PXA210 Processors Optimization Guide*, Oct. 2002.

An efficient memcpy is something that is processor-specific (sometimes even target-specific — depending on factors such as memory widths), so it's most sensible for RTOS infrastructure to supply a generic memcpy that's adequate and let the compiler/processor/target override it with its own if necessary.

Another good example is a count-leading-zeros operation. The Intel XScale core has a CLZ instruction but many systems implement it as a C-function.

On a related note, make sure you do not re-implement functions that are already available in the standard C library. The reference *Intel® IXP425 Network Processor Based on Intel® XScale™ Microarchitecture, Technical Specification*, May 2002 shows the performance difference between a C-implementation of memset and an one specific to the Intel XScale core, can be a factor of 30.

5.19 Aligned/Grouped Literal Pools

5.19.1 Context

You are using some literals or global variables on the data path.

5.19.2 Problem

Use of global variables and literals can be time consuming on the data path. These are usually accesses to SDRAM and if cached will pull in a cache line and evict another.

5.19.3 Solution

The Intel XScale core does not have a single instruction that can move all literals (a constant or address) to a register. Most compilers for the Intel XScale core load the literal from a memory location that is initialized with the constant or address. These blocks of constants are referred to as literal pools. These data blocks are located in the text or code address space so that they can be loaded using PC-relative addressing.

References to the literal pool area load the data into the data cache instead of the instruction cache. Therefore, the literal might be present in both the data and instruction caches, resulting in waste of space.

For maximum efficiency, the compiler should align all literal pools on cache boundaries and size each pool to a multiple of 32 bytes (the size of a cache line).

In addition, you could group highly used literal pool references into the same cache line. When one of the literals has been loaded, the other seven are available immediately from the data cache.

Use of this technique increased packet processing of one application by less than 1%.

5.20 Modulo/Divide Avoided

5.20.1 Context

You are writing code for a processor based on the Intel XScale core.

5.20.2 Problem

The processor does not directly support modulo or divide instructions and will require a call to a library support function.

5.20.3 Solution

You can translate some modulo or divide calculations into bit masks or shifts.

For example, modulo for dimensions that are a power of 2 can use a mask, e.g., instead of (var % 8) use (var & 7). Likewise, some divisions by constants can be converted into shift and add instructions.

Most compilers should be capable of generating this optimization but it may be worth examining generated code for any modulo or divide on your data path.

5.20.4 Forces

Bit masks/shifts are less readable code than division or modulo.

5.21 Minimal Cache Flush/Invalidation

5.21.1 Context

You have enabled caching for packet data memory.

5.21.2 Problem

Cached data exchanged with an NPE interface must be flushed when written and invalidated when read by the Intel XScale core.

5.21.3 Solution

Customer code must explicitly handle flush/invalidation of any of the packet memory they read/modify.

When transmitting data you should only flush packet memory cache lines you have written. When receiving data you should only invalidate the packet cache lines that you will read.

The IXP400 software handles the flush/invalidation of the IX_MBUF.

Flush/invalidate code needs to be carefully analyzed to ensure unnecessary flush/invalidates are not being initiated.

5.22 Endian Analysis

5.22.1 Context

The IXP42X product line and IXC1100 control plane processors can run as a big- or little-endian processor.

5.22.2 Problem

The performance of some applications can be affected by the processor's endian mode.

5.22.3 Solution

Networking protocols typically require big-endian data format. If the main processing requirement of your application involves reading or writing big-endian data it may be more efficient when running the processor in big-endian mode.

PCI devices typically work in little-endian data format. If the main processing requirement of your application involves interacting with PCI devices it may be more efficient when running the processor in little-endian mode.

5.22.4 Forces

A little-endian implementation may not be available for all RTOS.

5.23 Queue Look-Ahead

5.23.1 Context

You are developing access layer software for the IXP42X product line and IXC1100 control plane processors. You are reading and processing values from a hardware queue in a loop.

5.23.2 Problem

Reading a hardware queue will cause the processor to stall when you depend on the data read from the queue. This stall is in the order of 40 clock cycles on a 533-MHz IXP425 network processor.

5.23.3 Solution

Read and process the queue values with a look-ahead loop. Try to launch the next queue read when the previous value read from the queue is being processed.

This will not help the first stall on the first queue value read but if there are a number of queue values handled in a loop the subsequent values will not incur the same stall.

In one application this optimization increased packet processing throughput by approximately 3%.

5.24 Queue Status-Check Removed

5.24.1 Context

You are developing access layer software for the IXP42X product line and IXC1100 control plane processors.

5.24.2 Problem

Using the hardware queue status information will cause the processor to stall.

5.24.3 Solution

Avoid using the hardware queue status information if possible. If a queue empties when you read it you will read a value of 0. This means you do not need to read the queue status to detect underflow. Likewise, you may be able to avoid reading queue status to detect overflow on queue writes. This may, however, complicate your code.

Understand the underlying Qmgr functions you are using.

The removal of unnecessary queue status checks, in one application increased packet-processing performance by approximately 10%.

6.0 Code and Design Level

This section covers some general code tuning guidelines that are applicable to most processors. In many cases, these optimizations may decrease the readability, maintainability, or portability of your code. Be sure you are optimizing code that needs optimization [see “Avoiding Premature Code Tuning” on page 58].

6.1 Reordered Struct

6.1.1 Context

You have identified a bottleneck segment of code on your application data path. The code uses a large struct.

6.1.2 Problem

The struct spans a number of cache lines.

6.1.3 Solution

Reorder the fields in a struct to group the frequently accessed fields together. If all of the accessed fields fit on a cache line, the first access will pull them all into cache, potentially avoiding data-dependency stalls when accessing the other fields.

Organize all frequently written fields into the same half-cache-line.

6.1.4 Forces

Re-ordering structs may not be feasible if the struct is mapped to a packet or configuration information stored in flash.

6.2 Supersonic ISR

6.2.1 Context

Your application uses multiple interrupt service routines to signal the availability of data on an interface and trigger the processing of that data.

6.2.2 Problem

Interrupt service routines can interfere with other ISR and packet processing code.

6.2.3 Solution

Keep ISRs short. Design them to be re-entrant.

An ISR should just give a semaphore, set a flag or en-queue a packet. You should de-queue and process the data outside the ISR. This obviates the need for interrupt locks around data in an ISR.

Interrupt locks in a frequent ISR can have hard-to-measure effects on the overall system.

See reference *Doing Hard Time* for more detailed interrupt design guidelines.

6.3 Stall-Filling Code

6.3.1 Context

You have identified a stall instruction [see “Stall Instructions” on page 33] and traced it back to the line of C-code that stalls the processor.

6.3.2 Problem

The processor stalls due to a data dependency.

6.3.3 Solution

Try to insert code from later in your algorithm between the request for the data (the code that generates the load instruction) and the code that eventually uses the data (result of the load). If you can perform some other code that is independent of the data that can cause the stall your application can use the clock cycles that would otherwise be lost to a stall.

6.3.4 Forces

- Application of this pattern can sometimes make code less readable.
- Carefully comment the reason for moving the code to ensure other engineers do not undo this kind of optimization.
- The compiler may undo some of the changes that you are trying to accomplish by moving C code. Check the generated assembly.

6.4 Assembly-Language-Critical Functions

6.4.1 Context

You have identified a C function that consumes a significant portion of the data path.

6.4.2 Problem

The code generated for this function may not be optimal for your processor.

6.4.3 Solution

Re-implement the critical function directly in assembly language.

Use the best compiler for the application [see “Best Compiler for Application” on page 15] to generate initial assembly code, then hand-optimize it.

6.4.4 Forces

- Modern compiler technology is beginning to out perform the ability of humans to optimize assembly language for sophisticated processors.
- Assembly language is more difficult to read and maintain.
- Assembly language is more difficult to port to other processors.

6.5 Inline Functions

6.5.1 Context

You have identified a small C function that is called frequently on the data path.

6.5.2 Problem

The overhead associated with the entry and exit to the function may become significant in a small function, frequently called on the application data path.

6.5.3 Solution

Declare the function inline. This will mean the function will be inserted directly into the code of the calling function.

6.5.4 Forces

- Inline functions can increase the code size of your application and add stress to the instruction cache.
- Inline functions make debugging more difficult.
- A function call itself can limit the compiler's ability to optimize register usage in the calling function.
- A function call may cause a data dependency stall when a register waiting for an SDRAM access is still in flight.

6.6 Cache-Optimizing Loop

6.6.1 Context

You have identified a critical loop that is a significant part of the data-path performance.

6.6.2 Problem

The structure of the loop or the data on which it operates, could be "trashing" the data cache.

6.6.3 Solution

You can consider a number of loop/data optimizations:

- Array merging – the loop uses two or more arrays, merge them into a single array of a struct
- Induction variable interchange
- Loop fusion

For more details, see the *Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor Developer's Manual*.

6.7 Minimizing Local Variables

6.7.1 Context

You have identified a function that needs optimization. It contains a large number of local variables.

6.7.2 Problem

A large number of local variables will incur the overhead of storing them on the stack. The compiler may incur the overhead of setting up and restoring the frame pointer.

6.7.3 Solution

Minimize the number of local variables. This may mean the compiler can store all the locals and parameters in registers.

6.7.4 Forces

Removing local variables can decrease the readability of code or require extra calculations during the execution of the function.

6.8 Explicit Registers

6.8.1 Context

You have identified a function that needs optimization. A local variable or a piece of data is frequently used in the function.

6.8.2 Problem

Sometimes the compiler does not identify a register optimization.

6.8.3 Solution

It is worth trying explicit register hints to local variables that are frequently used in a function.

It may also be useful to copy a frequently used part of a packet that is used frequently in a data path algorithm into a local variable declared register. An optimization of this kind made a performance improvement of approximately 20% in one customer application.

6.8.4 Forces

The register keyword is only a hint to the compiler.

6.9 Removing Unnecessary Counters

6.9.1 Context

Your application code maintains counters for packets or memory on the data path. In some networking applications, there are separate counters at every layer of the networking stack.

6.9.2 Problem

Updating counters on the data path may cause accesses to SDRAM or add strain to the data cache. A counter update is usually a read/increment/write operation and could incur a data dependency stall.

6.9.3 Solution

Search for and remove all unnecessary counter and statistic updates on the critical data path code.

6.9.4 Forces

Counters can be useful when debugging or integrating an application.

6.10 Duff's Device

6.10.1 Context

You are writing a possible odd number of bytes to another location in memory or a memory-mapped output register.

6.10.2 Problem

In a loop with a small amount of processing, the loop boundary checks can take a significant amount of the processing of the loop.

6.10.3 Solution

Duff's device is a devious, unrolled, byte-copying loop. It is also a generic technique for unrolling loops.

In a conventionally controlled loop with one statement in the loop body, two statements will be executed per iteration. With a mod 8 Duff's device, you get the equivalent of 9/8 (1.125) statements per iteration, 44% fewer statements.

For more information, see question 20.35 in the C-FAQ at the following URL:

http://www.eskimo.com/~scs/C-faq/top.html

6.10.4 Forces

- The code is hard to understand; comment it well.
- Loops with a fixed number of iterations may be unrolled automatically by the compiler. This compiler feature may require the use of a compiler switch.

6.11 Optimized Hardware Register Write

6.11.1 Context

The data path code does multiple writes to one or more hardware registers.

6.11.2 Problem

Read-operation-writes on hardware registers can cause the processor to stall.

6.11.3 Solution

Read-operation-write statements can be broken up to hide some of the latencies when dealing with hardware registers. For example:

```
*reg1ptr |= 0x0400;
*reg2ptr &= ~0x80;
```

Would execute faster as:

```
reg1 = *reg1ptr
reg2 = *reg2ptr
reg1 |= 0x0400;
reg2 &= ~0x80;
*reg1ptr = reg1;
*reg2ptr = reg2;
```

One of the read dependency stalls is eliminated.

Secondly, search the data path code for multiple writes to the same hardware register. Combine all the separate writes to a single write to the actual register. For example, some applications disable hardware interrupts using multiple set/resets of bits in the interrupt enable register.

In one such application when we manually combined these write instructions we saw an approximate 4% performance improvement.

6.12 Avoiding the OS Packet-Buffer Pool

6.12.1 Context

The application uses a system packet buffer pools.

6.12.2 Problem

Memory allocation or calls to packet buffer pool libraries may be slow. In some operating systems, these functions lock interrupts and use local semaphores to protect simultaneous access to shared-heaps.

6.12.3 Solution

Avoid allocating or interacting with the RTOS packet buffer pool on the data path. Pre-allocate packet buffers outside the data path and store them in lightweight s/w pools/queues.

Stacks or arrays are typically faster than linked lists for packet buffer pool collections, as they require less memory accesses to add and remove buffers.

6.12.4 Forces

OS packet-buffer pools implement buffer collections. Writing another light collection duplicates functionality.

6.13 C-Language Optimizations

6.13.1 Context

You have identified a function or segment of code that is consuming a significant portion of the Intel XScale core's clock cycles on the data path. You may have identified this code using profiling tools [see "Profiling Tools" on page 35] or a performance measurement [see "PMU Performance Measurement" on page 19].

6.13.2 Problem

A function or segment of C-code needs optimization.

6.13.3 Solution

You can try a number of C-language level optimizations:

- Pass large function parameters by reference, never by value. They take time to copy and use registers.
- Avoid array indexing. Use pointers.
- Minimize loops by collecting multiple operations into a single loop body. You may not want to do this sometimes to preserve data cache.
- Avoid long if-then-else chains. Use a switch statement or a state machine.

- Use int (natural word-size of the processor) to store flags rather than char or short.
- Avoid floating point calculations on the data path.
- Use decrementing loop variables e.g., “for (i=10; i--;) {do something}” or even better “do { something } while (i--)”. Look at the code generated by your compiler in this case. The Intel XScale core’s processor has the ability to modify the processor condition codes (using adds and subs rather than add and sub) saving a “cmp” instruction in loops.

  For more details, see the *Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor Developer’s Manual*.
- Adjust structure sizes to power of two
- Place the most frequently true statement first in if-else statements
- Place frequent case labels first
- Write small functions. The compiler likes to reuse registers as much as possible and cannot do it in complex nested code.

For other similar tips, see chapter 29 in *Code Complete*.

6.13.4 Forces

- Good compilers will make a number of these optimizations.
- Avoid the trap of premature optimization [see “Avoiding Premature Code Tuning” on page 58].

6.14 Pre-Computed Data

6.14.1 Context

You have identified a code bottleneck in a function using a performance measurement [see “PMU Performance Measurement” on page 19] or profiling tools [see “Profiling Tools” on page 35].

6.14.2 Problem

A computation on the data path is taking significant time.

6.14.3 Solution

Do the computation at initialization time. Store the results for fast indexing by the data path.

A pre-computed array with two input variables in the range 0-255 and a word size result requires only 64-K words storage.

6.14.4 Forces

Storage of the results of computation will use memory.

7.0 VxWorks*-Specific Improvements

The following techniques are specific to VxWorks. Typically, these techniques can be applied in the board-support package (BSP) — when configuring the cache attributes of the memory map — or the END driver.

7.1 Aligned Mbufs and Clusters

7.1.1 Context

You are allocating VxWorks* Mbufs and clusters.

7.1.2 Problem

At time of writing VxWorks Mbufs are an inconvenient size, 36 bytes (32 bytes for the mbuf, 4 bytes for pool id). If you allocate them in the conventional manner you will get seven out of every eight mbufs straddling cache lines. This will cause invalidates or flushes to operate on two cache lines instead of one causing serious functional problems.

7.1.3 Solution

Use the Intel-supplied IxOsBuffMgt and IxOsBuffPoolMgt APIs to allocate mbufs for the IXP400 software. This API will allocate VxWorks* compatible, cache friendly mbufs at the cost of some memory padding.

7.1.4 Forces

- The Intel-supplied APIs are different from the standard netBufLib. A small amount of effort will be required to get familiar with them.
- They will cost some extra memory usage but typically you do not allocate more than 128 mbufs per receive interface.

7.2 Avoiding Separate Packet Buffer Pools

7.2.1 Context

You are using multiple packet buffer pools.

7.2.2 Problem

Multiple, separate packet buffer pools can add extra inefficiencies to the buffer management code.

7.2.3 Solution

Increase the size of the network data buffer stack instead of using multiple buffer pools.

7.2.4 Forces

- You need to work to size the network data buffer stack to account for multiple sources using it.
- You no longer have a guaranteed source of packet buffers for a particular driver.

7.3 Avoiding System Packet-Buffer Pool

7.3.1 Context

You are using netBufLib on the data path.

7.3.2 Problem

You cannot avoid allocating packet buffers on the data path [see "Packet Buffer Management Analysis" on page 25]. However, netBufLib is not efficient for some operations.

7.3.3 Solution

Write wrappers for netTupleGet and clChainFree. Call the wrappers from all END drivers (use the Intel-supplied IxOsBuffPoolMgt). The free function should add the packet buffer to its own chain free and only return to the system pool when the system pool is lower than a configured threshold or the internal chain is full. The alloc function should allocate from the internal chain and call netTupleGet (a very heavyweight function) only if the internal chain is empty. The alloc should reset data pointers, lengths and fields.

This is a VxWorks*-specific pattern for the implementation of some of the general concepts in Packet Buffer management Analysis.

7.3.4 Forces

This adds another library for application developers to learn.

7.4 Avoiding Unnecessary Packet-Buffer Allocations

7.4.1 Context

You have code that allocates a packet buffer then tries to replenish it to an IXP400 software API.

7.4.2 Problem

The algorithm relies on a failure from the IXP400 software API then frees the allocated packet buffer.

7.4.3 Solution

Some of the IXP400 software's APIs will help in this regard. The ATM API knows how many new replenish buffers it can handle, but the Ethernet API will just provide a failure indication when it can no longer take replenish buffers.

Avoid unnecessary allocations. Rewrite the packet buffer management code to keep track of outstanding buffers. Better still, pre-allocate them to a fixed size collection.

In one implementation of the VxWorks* Ethernet END driver it keeps calling netTupleGet until it fails or the replenish call fails, in which case it has to free the packet buffer it allocated. This is wasteful.

7.4.4 Forces

Some moderate complexity may be required to optimize replenishment of packet buffers for particular interfaces.

7.5 Batch Packets Handler

7.5.1 Context

You have an application or drivers that use netJobAdd.

7.5.2 Problem

Submitting single packets to netJobAdd could cause a context switch per packet in polled mode. In interrupt mode, there is a probability of a ring packet buffer overflow under high load.

7.5.3 Solution

Search for all usage of netJobAdd in your application and the networking stack software you are using. Inspect the code to determine if it calls netJobAdd for every packet.

If it does change the code to de-queue a number of packets, chain them together and submit them all. This kind of packet batching can also make better use of cache by ensuring the functions stay in instruction cache.

7.6 Avoiding Chaining

7.6.1 Context

You are implementing an END driver.

7.6.2 Problem

END driver examples typically have a large number of checks for chained packet buffers. These checks are on the data path and can incur a stall.

7.6.3 Solution

Allocate your packet buffers to avoid chaining [see “Packet Buffer Headroom” on page 28]. During integration enable code to detect chained buffers and generate a warning or increment a statistic to assist in optimizations.

When you are sure you have no chained packet buffers remove the END driver chaining check.

7.7 Disabled Functionality

7.7.1 Context

You are using VxWorks.

7.7.2 Problem

Some non-critical features use CPU cycles.

7.7.3 Solution

Disable non-critical features. For example, turn off netStatLib from config.h to increase IP stack performance.

7.8 Platform NE*

Platform NE* is a complementary software package to the VxWorks* Developer Toolkit that currently ships with the Intel® IXDP425 / IXCDP1100 Development Platform.

7.8.1 Context

You are using VxWorks.

7.8.2 Problem

A more efficient VxWorks-routing algorithm is now available.

7.8.3 Solution

Wind River* claims the Platform NE software improves the performance of the standard VxWorks* IP-routing capability by 10%-15%.

From a technical point of view, WRS gave assurances that the software shipped as the PNE product is 100% compatible with the IXDP425 / IXCDP1100 platform's board-support package (BSP). The software replaces or extends existing VxWorks* functionality at an application level.

This software adds extra networking functionality to the standard VxWorks* offering, including IPSEC, SNMP, PPP and OSPF features.

For more information see the following Web page:

http://www.windriver.com/platforms/platformne/

8.0 Development Strategies

The following patterns propose organization and procedures for performance tuning work.

8.1 Pair Team

8.1.1 Context

You are starting to work on performance improvement.

8.1.2 Problem

Optimization work is difficult; a lone engineer can get lost and chase numerous red herrings.

8.1.3 Solution

Implement optimization work in teams of two people. Two heads are better than one for this kind of work.

One of the pair could focus on strategic thinking while the other focuses on a tactical next-test approach.

It is also good to have another engineer available who is already spun up on the context of the work to provide consultation and act as a sounding board.

8.2 Avoiding Premature Code Tuning

8.2.1 Context

You are implementing a system and you are in the coding phase of the project. You do have a good system-level understanding of the performance requirements and the allocation of performance targets to different parts of the system because you have a performance design [see "Performance Design" on page 17].

8.2.2 Problem

It is difficult to know how much time or effort to spend thinking about performance or efficiency when initially writing the code.

8.2.3 Solution

"We should forget about small efficiencies, say about 97% of the time; premature optimization is the root of all evil." – Donald Knuth.

It is important to find the right balance between performance, functionality and maintainability.

Some studies have found 20% of the code consumes 80% of the execution time others have found less than 4% of the code accounts for 50% of the time [see *Code Complete*].

KISS — Keep it simple. Until you have measured and can prove a piece of code is a system-wide bottleneck, do not optimize it. Simple design is easier to optimize.

If you are working on a component of a system, you should have a performance budget for your part of the data path [see "Performance Design" on page 17].

In the unit test, you could have a performance test for your part of the data path. At integration time, the team could perform a performance test for the complete assembled data path.

"The best is the enemy of the good. Working toward perfection may prevent completion. Complete it first, then perfect it. The part that needs to be perfect is usually small." — Steve McConnell.

For further information, see chapters 28 and 29 in *Code Complete* and question 20.13 in comp.lang.c FAQ at the following URL:

http://www.eskimo.com/~scs/C-faq/top.html

8.2.4 Forces

- Efficient code is not necessarily "better" code. It may be difficult to understand.
- It is almost impossible to identify performance bottlenecks before you have a working system.
- If you spend too much time doing micro-optimization during initial coding, you might miss important global optimizations.
- If you look at performance, too late in a project it can be too late to do anything about it.

8.3 Step-by-Step Records

8.3.1 Context

You are trying a number of optimizations to fix a particular bottleneck. The system has a number of other bottlenecks.

8.3.2 Problem

Sometimes it is difficult when working at pace, to remember optimizations made even only a few days earlier.

8.3.3 Solution

Take good notes of each experiment you have tried to identify bottlenecks and each optimization you have tried to increase performance. These notes can be invaluable later. You may find you are stuck at a performance level with an invisible bottleneck. Reviewing recent optimization notes in a pair team [see "Pair Team" on page 58] may help you identify incorrect paths taken, or diversionary assumptions.

When a performance improvement effort is complete, it can be very useful to have notes on the optimization techniques that worked, to incorporate that learning into documents like this one to help other engineers benefit from your experience.

8.3.4 Forces

Making notes can sometimes break flow.

8.4 Quick- Run Traffic Test

8.4.1 Context

You are starting work to optimize the performance of your application. You are using a traffic test that can take some time to give you a result.

8.4.2 Problem

You may need to run a number of tests to first measure the performance of your application and analyze different parts of the data path. Over time, you will run a large number of iterations.

8.4.3 Solution

At an early stage, optimize the traffic test itself and the compile link cycle. If your traffic test takes five minutes to run it may be worth your time to choose a simpler traffic test for initial optimizing work and come back to the more extensive tests at the end of the optimization effort.

For example, the test you are trying to optimize may be a SmartApplications* throughput test or an FTP over a long period. For initial testing, use Smart Windows* instead of SmartApplications to measure a packet rate using the port counters. This will not consider packet loss but will be much faster to run. Later when you are almost done with performance tuning you can return to the real SmartApplications test.

8.5 Nightly Traffic Test

8.5.1 Context

You have reached your performance target. You are in the development phase of a project. Developers are checking new code into a source code repository daily.

8.5.2 Problem

Someone may check in new code or a bug fix that affects performance and pulls performance back below your requirement.

8.5.3 Solution

Set up an automated performance test to run nightly. This will flag an issue in new code soon after the developer introduces the problem and will facilitate the identification and resolution of the problem in a timely manner.

8.5.4 Forces

- This may require automation of test equipment which can be difficult to implement.

- This pattern will use a test bench, a DUT and performance measurement equipment.

8.6 Slam-Dunk Optimization

8.6.1 Context

You have made a number of improvements that have increased the efficiency of code running on the Intel XScale core.

8.6.2 Problem

The latest optimizations have not increased performance. You have hit some unidentified performance-limiting factor.

8.6.3 Solution

You may have improved performance to a point where environmental factors, protocols or test equipment are now the bottleneck.

It is useful to have a code modification identified which you know should improve performance, for example:

- An algorithm on the data path that can be removed temporarily (IP checksum)
- Increasing the Intel XScale core clock speed.

In one application, we implemented a number of optimizations that should have improved performance but did not. We then removed the IP checksum calculation and performance still did not increase. This pointed to a hidden limiting factor, an unknown bottleneck. When we followed this line of investigation, we found a problem in the way we configured a physical layer device and when we fixed this hidden limiting factor, we got an immediate performance improvement of approximately 25%. We retraced our steps and reapplied the earlier changes to identify the components of that performance improvement.

This page intentionally left blank.