QNX SOFTWARE SYSTEMS

Using Symmetric Multiprocessing (SMP) to Scale Data Plane and Control Plane Performance

Sebastien Marineau-Mes, Senior Networking Architect,
QNX Software Systems
sebastien@qnx.com

Table of Contents

Abstract

Symmetric multiprocessing (SMP) can offer enormous scaling benefits, particularly when applied to networking applications such IP and MPLS route calculation, Layer 3 packet forwarding, storage caching, and call-processing encryption. In this paper, we provide an overview of SMP, describing how it compares to other forms of multiprocessing, why it benefits networking applications, and what techniques developers can employ to take advantage of it. Then, to illustrate the power of SMP, we review a benchmark study that measures Layer 3 packet forwarding, using a test system based on the QNX® Neutrino® RTOS and Broadcom's SiByte BCM1250 dual-processor SOC.

The study demonstrates that SMP implemented on QNX Neutrino can offer dramatic performance gains — more than 100 per cent faster throughput for small packets — without modifications to software. In addition, the study shows how trace analysis tools such as the QNX Momentics® system profiler can uncover hidden bottlenecks in SMP systems, allowing developers to achieve even greater concurrency and performance.

The Demand for Scalability

Fortune 1000 companies and service providers share a common problem**:** large, unpredictable increases in their network traffic. In response, these organizations have stepped up demands for networking equipment that can handle growing workloads without sacrificing performance or requiring additional hardware. Nonetheless, this demand for scalability — combined with the continuous growth of network topologies — is placing enormous processing demands on both the control plane and the data plane.

Consider, for example, the control plane in a typical VPN router. To keep track of all nodes on a network and to update the most reliable path between any two points, the control plane typically uses Open Shortest Path First (OSPF), a link-state routing protocol with a large appetite for CPU cycles. When OSPF is combined with a protocol such as the Virtual Redundancy Routing Protocol (VRRP), the control plane must create and maintain huge routing tables that can cover hundreds of thousands of nodes. But that's only the beginning. The processing units on VPN routers must also help forward packets, since many routers, in an effort to reduce costly ASICs, handle some IP forwarding in software.

Together, these computational demands have reached the point where even the fastest host processor struggles to keep pace. Consequently, many networking equipment designers are looking at multiprocessing architectures, in particular, symmetric multiprocessing (SMP), to address the problem.

SMP Defined

Before continuing, let's define what we mean by multiprocessing and SMP.

Unlike multitasking, which is the simultaneous execution of multiple tasks by a single processor, multiprocessing is the simultaneous execution of multiple tasks by *N* processors. A variety of multiprocessing techniques exist — everything from parallel processing across multiple processors to instruction pipelining on multiple logic units within a single processor.[1]

In a multiprocessing system, processors are either loosely coupled or tightly coupled, depending on how the processors communicate and share resources. Loosely coupled designs, which have been the staple of embedded networking equipment, typically consist of multiple independent processing complexes, each with its own peripherals, memory, and software stack. Applications running on these processors communicate via messaging protocols implemented over peripheral interconnects such as PCI, switch fabrics, or Ethernet. An emerging trend in the industry is the advent of high-speed, switched, chip-to-chip interconnects that allow high-speed, low-latency processor-to-processor messaging. Examples include RapidIO, HyperTransport, and Advanced Switching.

[1] Of course, in a conventional multitasking system, tasks only appear to run simultaneously — in reality, the processor is simply switching from one task to another very quickly. In a multiprocessing system, on the other hand, multiple tasks really do run at the same time.

Tightly coupled processors, which form the basis of SMP systems, share global resources such as memories and data buses, and communicate through common hardware circuits; see Figure 1.

Tightly coupled processors can be asymmetric or symmetric:

- **Asymmetric Multiprocessing** — Specific processors can execute only certain task types. For example, processor *x* runs only kernel code, processor *y* runs only application code, and processor *z* is restricted to executing interrupt service routines. This asymmetry may arise from processor differences, interconnect differences, application design, or OS kernel architecture.

- **Symmetric Multiprocessing** — In a true SMP system, any processor can execute any thread, including kernel code, application code, and interrupt service code. This symmetry, which allows each processor to be exploited to the fullest, is achieved through identical processors and interconnects, combined with appropriate application and OS kernel design.

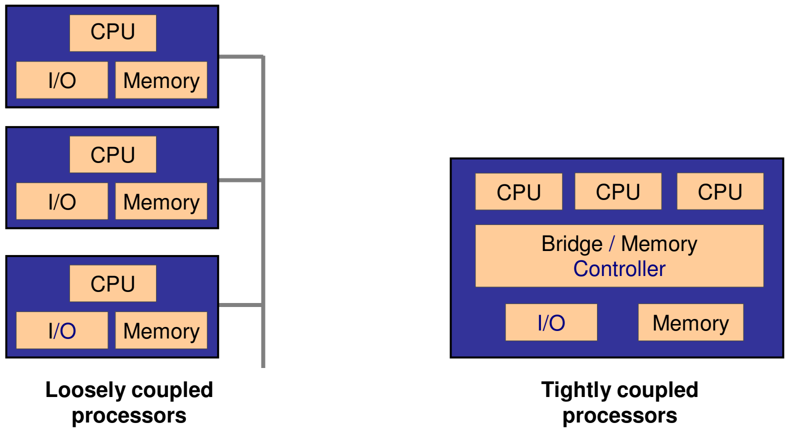

CPU
I/O
Memory
CPU
I/O
Memory
CPU
I/O
Memory
Loosely coupled processors
CPU
CPU
CPU
Bridge / Memory Controller
I/O
Memory
Tightly coupled processors

Figure 1 — Loosely vs. tightly coupled processors.

SMP variants

Recently, two variants of SMP have appeared in the communication space. The first consists of multicore processors that implement tightly coupled SMP on a single chip. Examples include the Broadcom BCM1250 and the PMC-Sierra RM9000x2, both of which are dual-core, highly integrated MIPS-based designs.

The second approach, simultaneous multithreading (SMT), allows multiple tasks or threads to execute concurrently on a single processor core.[2] The result is greater execution-unit utilization and higher overall instruction execution rate. An example is Intel's Hyper-Threading technology, implemented in Pentium 4 and Xeon processors. Intel's implementation of Hyper-Threading closely follows the SMP model, as the multiple threads on each processor appear as virtual processors to software.

[2] From this point on, we use the generic designation "SMP" to indicate both SMT and traditional SMP.

The Benefits of SMP

Implementing SMP in your system design can provide a number of advantages, including higher processing density, lower cost per MIPS, and increased scalability.

High density and lower cost per MIPS

Within the last several years, multiprocessing offerings from commercial embedded vendors have made SMP a cost-effective vehicle for achieving higher raw computing power at lower cost. Vendors such as Broadcom, IBM, Intel, Motorola, and PMC-Sierra now offer off-the-shelf SMP boards for a variety of MIPS, PowerPC, and x86 processors.

A major benefit of SMP is the reduced board and chip real estate needed to achieve higher processing power. In traditional loosely coupled systems, each processor complex has its own bridge chip, memory controller, memory, boot flash, and peripherals. With SMP, however, only the processor needs to be duplicated.

The simultaneous multithreading enabled by SMP also reduces costs by offering higher utilization of processor resources, with limited incremental cost or heat dissipation. For instance, typical performance gains for Hyper-Threading processors are in the order of 40 per cent for real-world, multithreaded applications.

Until recently, SMP entailed significant software development costs. Because engineers built SMP systems with proprietary hardware and had to code SMP awareness into their applications, an application implemented for a particular SMP system required substantial reengineering to migrate to new hardware. With the advent of SMP-capable operating systems like QNX Neutrino, these problems have been eliminated. As a software engineer, you no longer have to hardcode SMP awareness into each application, nor do you have to recode when migrating applications to different SMP platforms or back to uniprocessor platforms. In fact, if you’ve converted your existing uniprocessor code from a single-threaded to a multithreaded model, most of your work is already done.

Moreover, software doesn’t incur any overhead to maintain a coherent view of the data distributed among the processors of an SMP system; this coherency is handled transparently by modern SMP hardware.

Increased scalability

When an RTOS kernel such as the QNX Neutrino microkernel properly implements and exclusively manages SMP, applications can be SMP-capable without having to be SMP-aware. In effect, an application running on such a kernel is inherently scalable: the application’s independent threads will automatically multitask when running on a single processor, and those same threads can automatically migrate to the multiple processors when running on an SMP system. Application performance can increase without any need for code modifications or any need for algorithms that detect when additional processors are present.

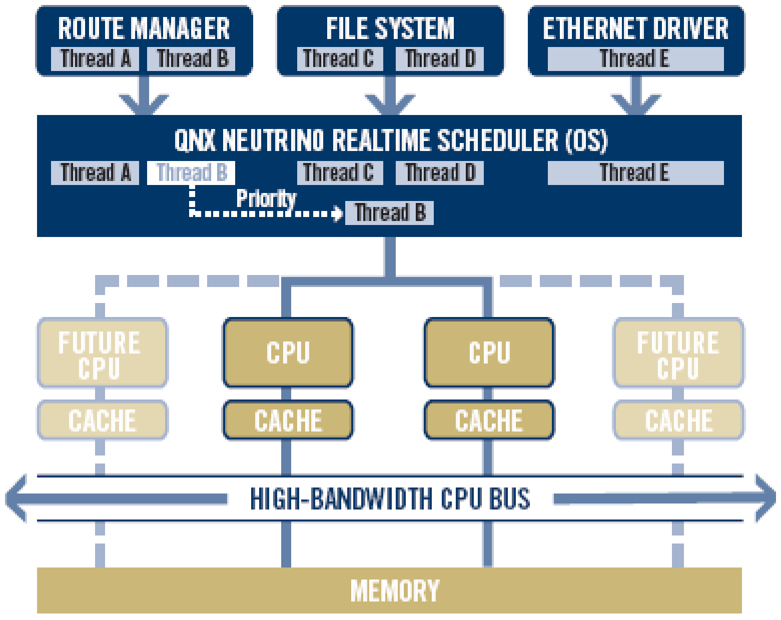


Figure 2 — While offering performance gains (by doing the same amount of work in less time), an SMP system also offers capacity scaling (by doing more work in the same amount of time).

QNX Neutrino and SMP

QNX Neutrino is a multithreading, SMP operating system that implements file systems, device drivers, and other system services as memory-protected user processes. These services communicate with each other, and with applications, by means of message-passing and synchronization primitives that are managed by the QNX Neutrino microkernel.

This microkernel architecture offers a fundamental advantage over the monolithic OSs traditionally used to implement SMP. In monolithic OSs, the kernel contains the bulk of OS services. As a result, adding support for SMP typically requires large numbers of performance-robbing modifications and spinlocks throughout the kernel code. In fact, since device drivers also run in kernel space, adding SMP support typically means modifying each driver as well. With QNX Neutrino, on the other hand, only the microkernel needs SMP awareness and supporting logic. As a result:

- Support for SMP requires only a few additional kilobytes of kernel code, incurring negligible overhead.
- Multithreaded applications, drivers, protocol stacks, and OS services can automatically take advantage of all available processors on an SMP system, since the microkernel handles the details of scheduling threads on each processor. There’s no need for additional application logic or recompilation.
- Most system services don’t grow in complexity as a result of the conditional logic traditionally required to support both uniprocessor and multiprocessor platforms.

Moreover, QNX Neutrino SMP is truly symmetric: Any kernel thread, any process (e.g. database, file system), and any thread within a process can be transparently scheduled for execution on any CPU.

Note that QNX Neutrino supports a variety of SMP boards based on MIPS, PowerPC, and x86 processors. Support is provided though board support packages (BSPs).

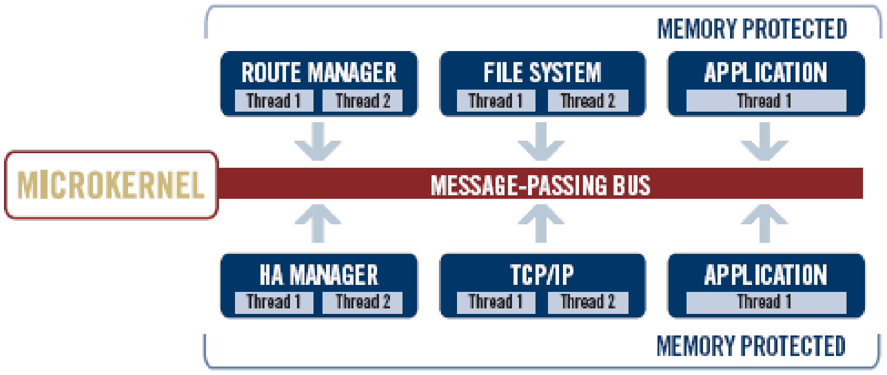


Figure 3 — As a microkernel OS, QNX Neutrino runs both applications and system services (e.g. drivers, file systems) as separate, user-space processes. This architecture enables optimal SMP performance by allowing a large number of independent components to run concurrently.

Identifying Applications that Benefit from SMP

In general, SMP can provide performance improvements for:

- compute-bound tasks[3] that wait for intermittent I/O, then process the input
- loosely coupled tasks that seldom communicate with each other
- independent computations
- distributed algorithms
- sets of independent applications

Which networking applications benefit from SMP?

Networking applications that successfully employ SMP include:

- IP and MPLS route calculations
- Layer 3 packet forwarding
- storage caching algorithms

[3] In this document, "task" refers to any arbitrary unit of computation. In practice, a task is assigned to either a thread or a process. (In some OSs, such as QNX Neutrino, a process may contain more than one thread.)

- call-processing encryption applications
- voice and data compression and encoding

Such applications are often compute bound, are either multithreaded or amenable to multitask decomposition, and contain tasks that seldom contend over shared resources.

Applications less likely to benefit

The following algorithms may benefit less from SMP:

- I/O-bound tasks that primarily stream I/O with little or no computation between task arrivals
- Tasks that frequently communicate with each other, contend for shared resources, or serialize access to a resource
- Tasks that implement a serial pipeline of dependent computations

Nonetheless, I/O-bound tasks can benefit from multithreading and Hyper-Threading; execution units that become idle because of stalls in communicating with I/O devices can execute instructions from other threads running in the processor core.

Case Study: Packet Forwarding Performance with SMP

To demonstrate the benefits of SMP, and to investigate techniques for optimizing its performance, QNX Software Systems conducted a study that examined Layer 3 packet forwarding. The following sections:

- describe the test setup
- examine packet forwarding performance in both single-processor and SMP configurations
- discuss the architecture of the forwarding code in QNX Neutrino
- explore an optimization that offers additional performance gains under SMP

Test setup

Since measurement of Layer 3 packet forwarding performance is a common benchmark for routers, the testing methodology has been standardized by the Internet Engineering Task Force (IETF) in RFC2544. We followed this methodology, using an Adtech AX/4000 Broadband Test System that included two Gigabit Ethernet port interfaces and two 1 Gbps mAX IP and mAX IPex routing generator/analyzers. We connected the Ethernet ports to the device under test (DUT), which was configured as a router. See Figure 4.

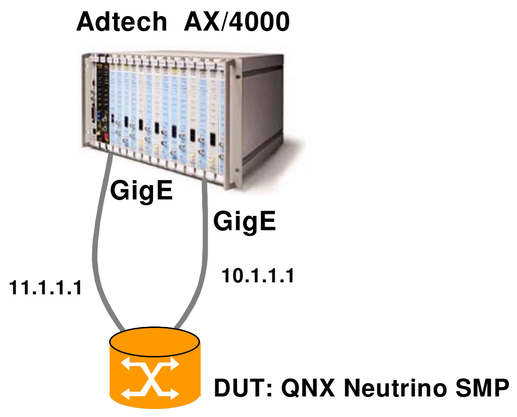


Figure 4 — Test setup for measuring packet forwarding performance. RFC2544 was used as the testing methodology.

The DUT consisted of the following:

- **Hardware** — The DUT used the Broadcom BCM91250A "SWARM" evaluation board, based on the SiByte BCM1250 dual-processor SOC (clocked at 600MHz for this study). The board includes two Gigabit Ethernet ports; both were used as the network interfaces in the packet forwarding tests.

- **Software** — The DUT software consisted of the QNX Neutrino RTOS v6.2.1B, including a TCP/IP stack, supporting software utilities, and an instrumented SMP kernel. QNX Neutrino's SWARM board support package (BSP) provided boot support, including UART and PCI drivers, as well as Ethernet network drivers for the board's Gigabit ports.

The RFC2544 testing methodology includes the measurement of throughput, latency, frame loss, and back-to-back packet handling. We focused on the first test, throughput, which RFC2544 defines as the highest rate at which packets of a given size can be sent to a DUT and forwarded back, without loss of a single frame.

The AX/4000 traffic generator/analyzers implement this benchmark by generating known packet streams through each interface and analyzing the forwarded packets to measure frame loss. The AX/4000 software automates measurement of throughput by iterating through different packet rates (via a binary search algorithm) to determine the packet rate at which no frames are dropped. The results are stored, and the test is repeated for all packet sizes.

We configured the two ports on the DUT to reside on different subnets (10.x.x.x and 11.x.x.x), and set up static route entries to allow forwarding between these two interfaces. We then set the AX/4000 to generate two traffic streams, in a fully meshed configuration; that is, traffic would be sent to both ports and the throughput would be the highest packet rate that both ports could sustain.

On the DUT, we used the following commands to start the QNX network manager (io-net), network driver, and TCP/IP stack:

```
Io-net -dbcm1250 receive=256,transmit=1024 -ptcpip-v6 fastforward -s &
     # Set number of RX and TX descriptor
     # Enable fastforward
     # Enable static io-net configuration
Ifconfig en0 10.1.1.1
Ifconfig en1 11.1.1.1
Ifconfig +ip4csum en0
     # Turn on HW checksum on port 0
Ifconfig +ip4csum en1
     # Turn on HW checksum on port 1
```

Packet forwarding results with a single processor

To obtain a baseline, we first measured packet forwarding for a single processor. The software configuration was identical to that used in our subsequent SMP test, with one exception: the QNX Neutrino startup component was passed the “-P1” option, forcing it to run in single-CPU mode. See Table 1 for the results.

Packet Size	Packet Rate (PPS per port)	Total Throughput (Mbit/sec)
64	167280	214
128	164666	421
256	166254	851
512	145056	1485
768	130208	2000
1024	97656	2000
1280	78125	2000
1514	65876	2000

Table 1 — Throughput results for single-processor case.

As you can see, the forwarding rate is fairly constant for small packet sizes — about 180,000 packets per second (PPS) for each port, or 360,000 forwarded PPS for the device — and quickly reaches the theoretical maximum rate for large packets.

We then used the QNX Momentics system profiler, a visualization tool that hooks directly into the instrumented SMP kernel, to capture system trace events as the test progressed. This trace information allowed us to determine exactly how the system was behaving. But before we review this information, let’s look at the architecture of the QNX networking stack to understand the path that packets take as they are being forwarded. This will provide a better understanding of the trace results.

How packets are forwarded under io-net

Under the QNX Neutrino networking stack architecture, io-net, packets can originate either within a device driver (received packets from interfaces) or from a protocol stack; see Figure 5. In the present case, packets are received by either of the Gigabit Ethernet ports. The driver removes the packets from the received queues, then sends them up through io-net to the next layer. To determine where a packet goes next, io-net looks up the packet type (for instance, "ethertype"), then chooses the appropriate upper module.

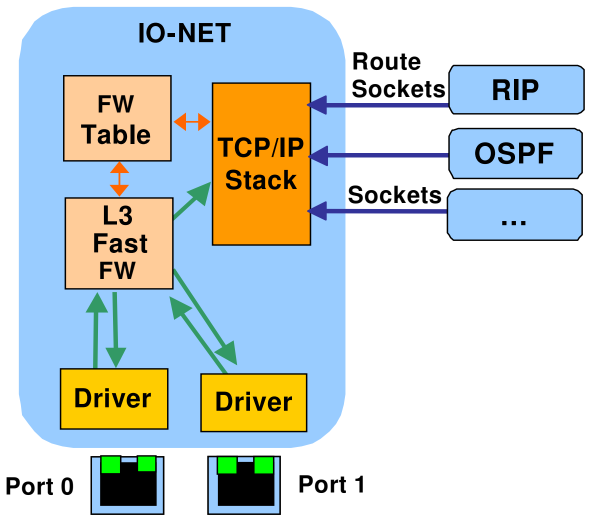


Figure 5 — QNX networking stack architecture (io-net).

In this particular scenario, the packets are IP packets, which get passed up to the TCP/IP stack. If packet forwarding is enabled, the IP stack will send any new packet it receives through the "fast-forwarding" path (see L3 Fast FW in Figure 5). The fast-forwarding code looks up the packet source and destination in the forwarding table, and, if it finds a match, updates the next-hop information and sends the packet down to the appropriate driver to be transmitted.

If no match is found in the forwarding table, the packet is queued for exception processing. Packets that require additional processing include packets destined for the local host (for instance, SNMP and RIP packets) and packets that need additional transformations before being forwarded (for instance, packets that match an IPSec rule and need to be encrypted).

Now let's look at the system profiler trace taken during the packet forwarding test in single-processor mode. As shown in Figure 6, threads 5 and 6 in io-net are alternatively active: these are the driver threads for the two Gigabit Ethernet ports. These threads share the processor and never actually block, each draining its receive queue and forwarding the packets to the other port's transmit queue. From this trace, we could conclude that the fast-forwarding path works as expected, and that our packet rate is limited mostly by processing power — we'll see later whether this conclusion holds up under further scrutiny.

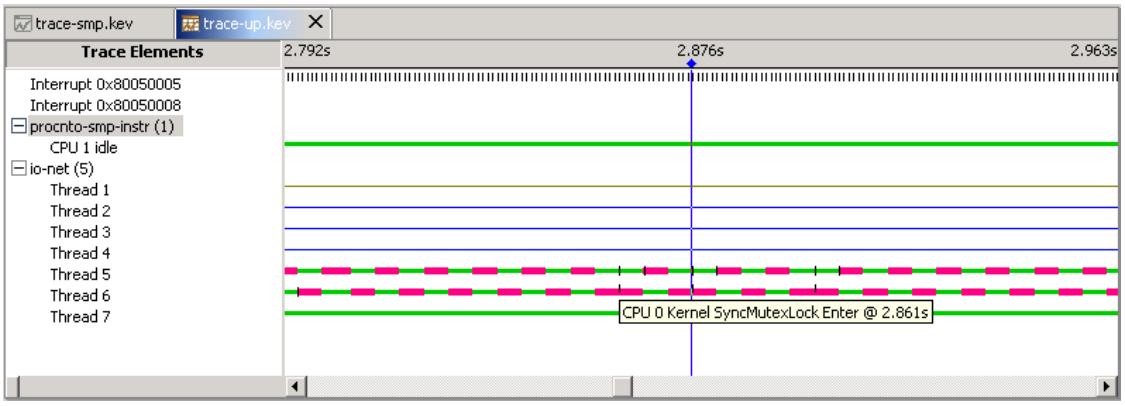


Figure 6 — System profiler results for single-processor case.

Packet forwarding results with symmetric multiprocessing

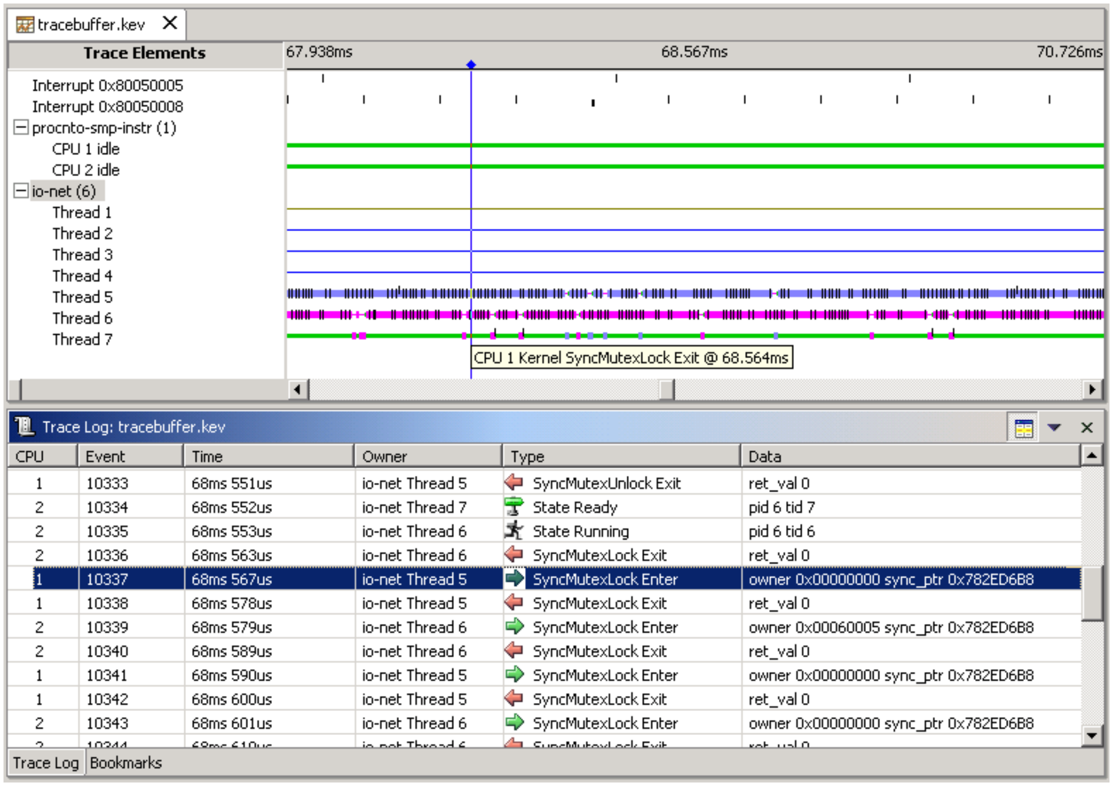
Next, we repeated the test, but with both processors enabled. See Table 2.

Packet Size	Packet Rate (PPS per port)	Total Throughput (Mbit/sec)
64	323429	413
128	315630	808
256	312545	1600
512	195312	2000
768	130208	2000
1024	97656	2000
1280	78125	2000
1514	65876	2000

Table 2 — Throughput results for dual-processor case. Throughput has improved almost 100 per cent for small packets.

As you can see, SMP provided clear performance gains; for small packets, total throughput has improved by almost 100 per cent. This performance gain was achieved without any modifications to the software or to the configuration of the device.

Figure 7 shows a system profiler trace taken during the benchmark — the running thread on CPU 0 is light blue, the running thread on CPU 1 is magenta. We can now see that the performance gain from the dual-processor system comes from its ability to run the two driver threads in parallel (threads 5 and 6 of io-net), each driver forwarding packets from a separate interface.



CPU	Event	Time	Owner	Type	Data
1	10333	68ms 551us	io-net Thread 5	SyncMutexUnlock Exit	ret_val 0
2	10334	68ms 552us	io-net Thread 7	State Ready	pid 6 tid 7
2	10335	68ms 553us	io-net Thread 6	State Running	pid 6 tid 6
2	10336	68ms 563us	io-net Thread 6	SyncMutexLock Exit	ret_val 0
1	10337	68ms 567us	io-net Thread 5	SyncMutexLock Enter	owner 0x00000000 sync_ptr 0x782ED6B8
1	10338	68ms 578us	io-net Thread 5	SyncMutexLock Exit	ret_val 0
2	10339	68ms 579us	io-net Thread 6	SyncMutexLock Enter	owner 0x00060005 sync_ptr 0x782ED6B8
2	10340	68ms 589us	io-net Thread 6	SyncMutexLock Exit	ret_val 0
1	10341	68ms 590us	io-net Thread 5	SyncMutexLock Enter	owner 0x00000000 sync_ptr 0x782ED6B8
1	10342	68ms 600us	io-net Thread 5	SyncMutexLock Exit	ret_val 0
2	10343	68ms 601us	io-net Thread 6	SyncMutexLock Enter	owner 0x00000000 sync_ptr 0x782ED6B8

Figure 7 — System profiler results for dual-processor case.

SMP optimizations for packet forwarding performance

Next, we investigated whether we could further speed up packet forwarding with SMP.

A common SMP bottleneck occurs when two or more threads contend for the same resource — for instance, two threads that share a data structure whose access is protected by a mutex or spinlock. If the two threads run in parallel on separate processors (SMP) and both access this data structure for significant amounts of time, they may end up spending considerable time contending for the lock, thereby serializing execution. This will diminish the benefits of parallel processing.

Returning to the system profiler trace in Figure 7, we can make some additional observations. For instance, the trace shows that each driver thread makes several calls to *mutex_lock*() and *mutex_unlock*(), a telltale sign of lock contention. This contention is for the same mutex object, which we can trace back to the mutex protecting the forwarding table in the TCP/IP fast-forwarding code. This lock is necessary, since the table has to be updated by the driver threads whenever packets are forwarded (to increment statistics, for instance). The table also has to be updated by the main TCP/IP stack thread when route table entries are added or deleted, usually by external routing protocols such as RIP and OSPF. While these updates are infrequent, the mutex protecting the table is necessary to prevent the data structures from becoming corrupted during forwarding-table modifications.

To reduce the lock contention and improve scalability and performance with SMP, we modified the fast-forwarding code to use a separate forwarding table for each processor. So, in a two-way SMP system, we now maintain two separate copies of the forwarding table, each with its own forwarding entries (which are duplicated for each CPU), and a separate mutex protecting each table. In principle, this should reduce lock contention, as each driver thread, running on a separate processor, would no longer contend for a shared forwarding table. It also has the benefit of improving cache utilization. The modified stack architecture is shown at right in Figure 8.

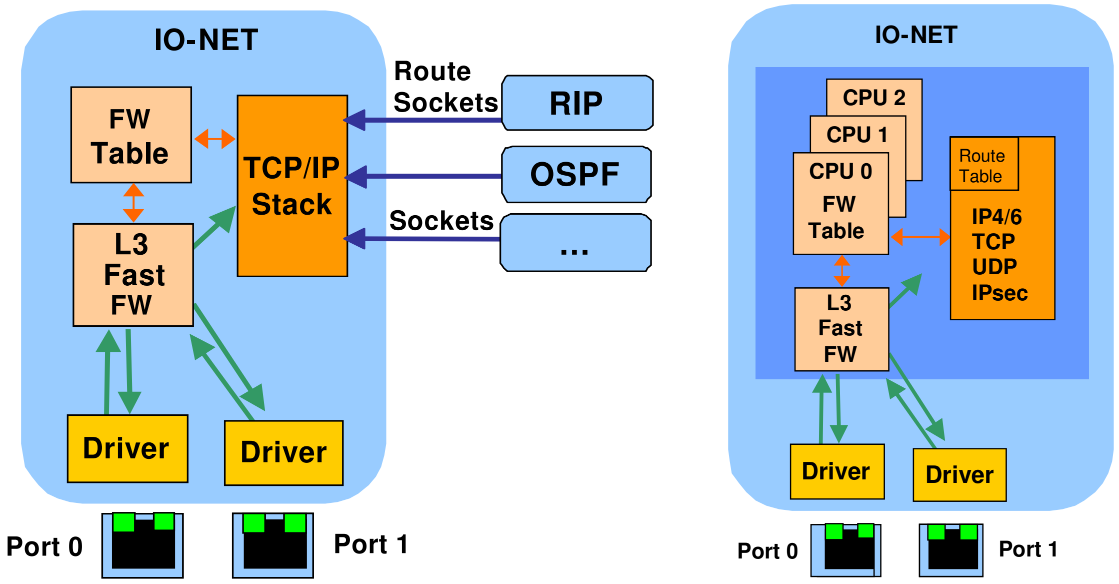


Figure 8 — Original io-stack architecture (left) and modified io-stack stack architecture (right).

We repeated our tests, this time with the new architecture. In Table 3 and Figure 9, you'll see that the forwarding rate for small packets is now about 20 per cent faster than the original SMP results. Furthermore, a system profiler trace, shown in Figure 10, reveals that resource contention has been virtually eliminated: threads 5 and 6 of io-net now run concurrently on each processor without any interruptions, rescheduling, or lock contention.

Packet Size	Packet Rate (PPS per port)	Total Throughput (Mbit/sec)
64	341390	436
128	339602	869
256	330858	1693
512	195312	2000
768	130208	2000
1024	97656	2000
1280	78125	2000
1514	65876	2000

Table 3 — Throughput results after lock contention was reduced. Forwarding rate for small packets is now about 10 per cent faster than original SMP results, and over 100 per cent faster than single-CPU results.

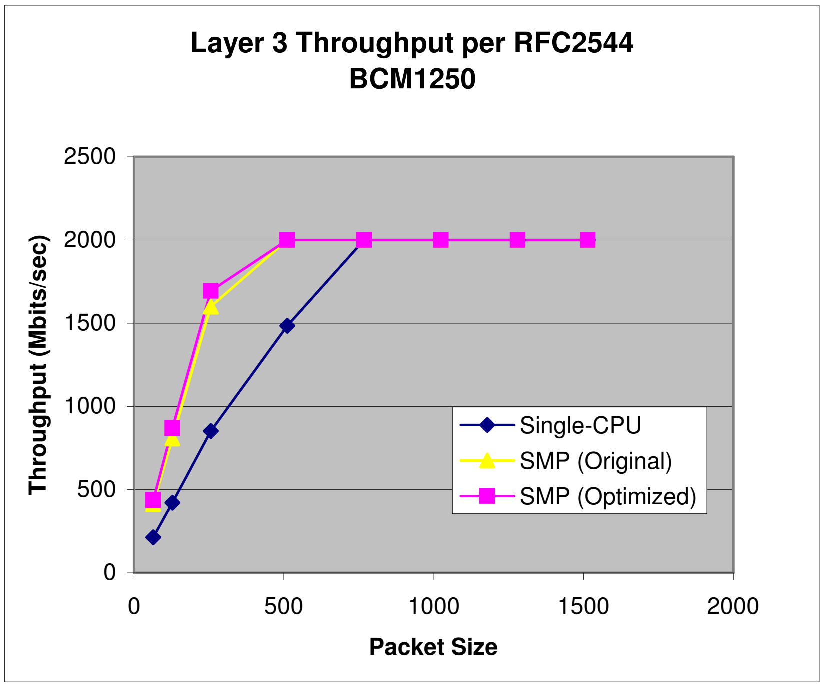


Figure 9 — Throughput results after lock contention was reduced. (See comments for Table 3.)

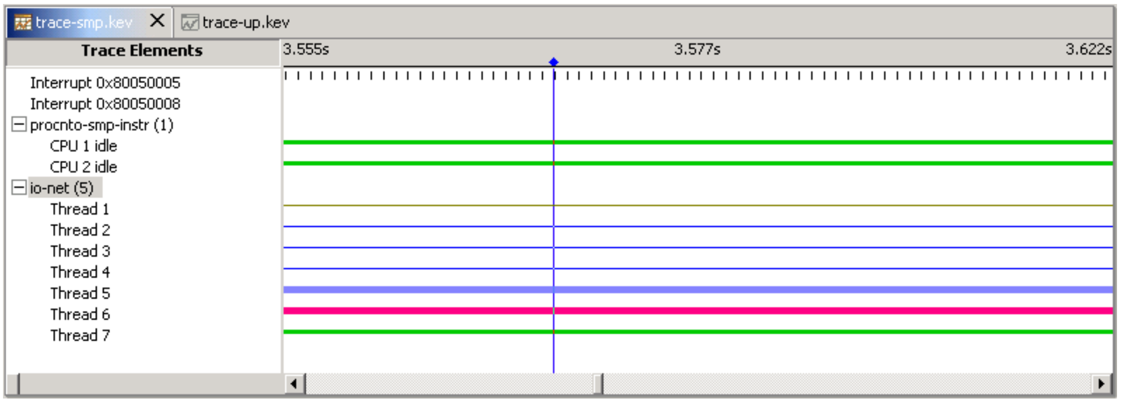


Figure 10 — Use of separate forwarding tables has virtually eliminated resource contention. Threads 5 and 6 of io-net now run concurrently without any interruptions or rescheduling.

Performance Gains Out of the Box

In the QNX Neutrino RTOS, only the microkernel needs to have SMP awareness; consequently, developing SMP-capable applications is no more difficult than developing multithreaded applications for a uniprocessor platform. This ease of development, combined with the high processing density and low cost per MIPS of SMP hardware, makes SMP a compelling alternative for any OEM building scalable, high-performance network elements. In fact, as the case study demonstrates, dramatic performance benefits can be obtained “out of the box” without any code changes or recompilation. And even greater performance can be achieved through a tool such as the QNX Momentics system profiler, which graphically pinpoints contention for shared resources and other potential SMP bottlenecks.

While the tests described here were performed on a MIPS dual-core processor, similar increases in performance can also be realized on other processor architectures (e.g. PowerPC, x86) that support SMP.

References

RFC 2544, “Benchmarking Methodology for Network Interconnect Devices”
http://www.faqs.org/rfcs/rfc2544.html

Adtech AX/4000 Broadband Test System
http://www.spirentcom.com/analysis/product_line.cfm?pl=1&WS=173&wt=2

BCM91250A (SWARM) evaluation board product brief
http://www.broadcom.com

QNX Neutrino RTOS product brief
http://www.qnx.com/resource/rs_pdf/rs_neutrino.pdf

QNX SOFTWARE SYSTEMS

About QNX Software Systems

QNX Software Systems is the leading global provider of innovative embedded technologies, including middleware, development tools, and operating systems. The component-based architectures of the QNX® Neutrino® RTOS, QNX Momentics® Tool Suite, and QNX Aviage® middleware family together provide the industry's most reliable and scalable framework for building high-performance embedded systems. Global leaders such as Cisco, Daimler, General Electric, Lockheed Martin, and Siemens depend on QNX technology for vehicle telematics and infotainment systems, industrial robotics, network routers, medical instruments, security and defense systems, and other mission- or life-critical applications. The company is headquartered in Ottawa, Canada, and distributes products in over 100 countries worldwide.

www.qnx.com