



Transparent Multi-core Cryptographic Support on Niagara CMT Processors

Dr. Christoph Schuba

Christoph.Schuba@Sun.COM

<http://blogs.sun.com/schuba>



Joint Work

- James Hughes
- Gary Morton
- Jan Pechanec
- Christoph Schuba
- Lawrence Spracklen
- Bhargava Yenduri

Overview

- Hardware acceleration
- UltraSPARC T2 processor
- Solaris Cryptographic Framework
- Packet size issues
- SSH
- Conclusion

Hardware Acceleration

- Cryptographic protections becoming essential
 - > E.g., web servers, databases, filesystems, networking
- Security costly (wrt performance)
 - > Commonly 2x+ slowdown
 - > Hindering adoption
- Cryptographic processing offloading to accelerators
 - > Can significantly reduce the overhead
 - > Reduces cost of crypto processing by 20x+

Off-Chip Acceleration Problematic

- Higher cost for off-chip accelerator cards
- Lower CPU utilization
- Higher I/O bandwidth consumption
- Higher I/O latency overhead
- I.e., Problematic for
 - > bulk ciphers
 - > small/moderately sized packets

Hardware Acceleration (cont.)

- Accelerators promise near zero-cost security
 - > Going secure: negligible performance impact?
- Application access via software framework
 - > Efficient, scalable, transparent access to accelerator
 - > Multiple requesting threads
 - > Multiple accelerators likely in today's multi-core environment
 - > Additional complications with virtualization trends

UltraSPARC T2 (*Niagara*) processor

- Accelerators are on-chip in Sun's multi-core CMT
 - > 1 accelerator per core
 - > up to 8 accelerators per chip
 - > up to 32 accelerators per system
- Discrete accelerators with minimal use of core pipeline resources
 - Each accelerator performs concurrently an RSA operation and an AES operation
 - In addition to the 8 hardware threads per core
 - Hardware threads execute unimpeded

UltraSPARC T2 (*Niagara*) processor

- MAU - Modular Arithmetic Unit
- Accelerators support
 - > Bulk encryption (RC4, DES, 3DES, AES)
 - > Secure hash (MD5, SHA-1, SHA-256)
 - > Public key algorithms (RSA, DH, ECC)
- Light-weight accelerator interface
 - > Communication is via a memory-based control word queue
 - > Stateless
 - > No limit on number of simultaneous contexts

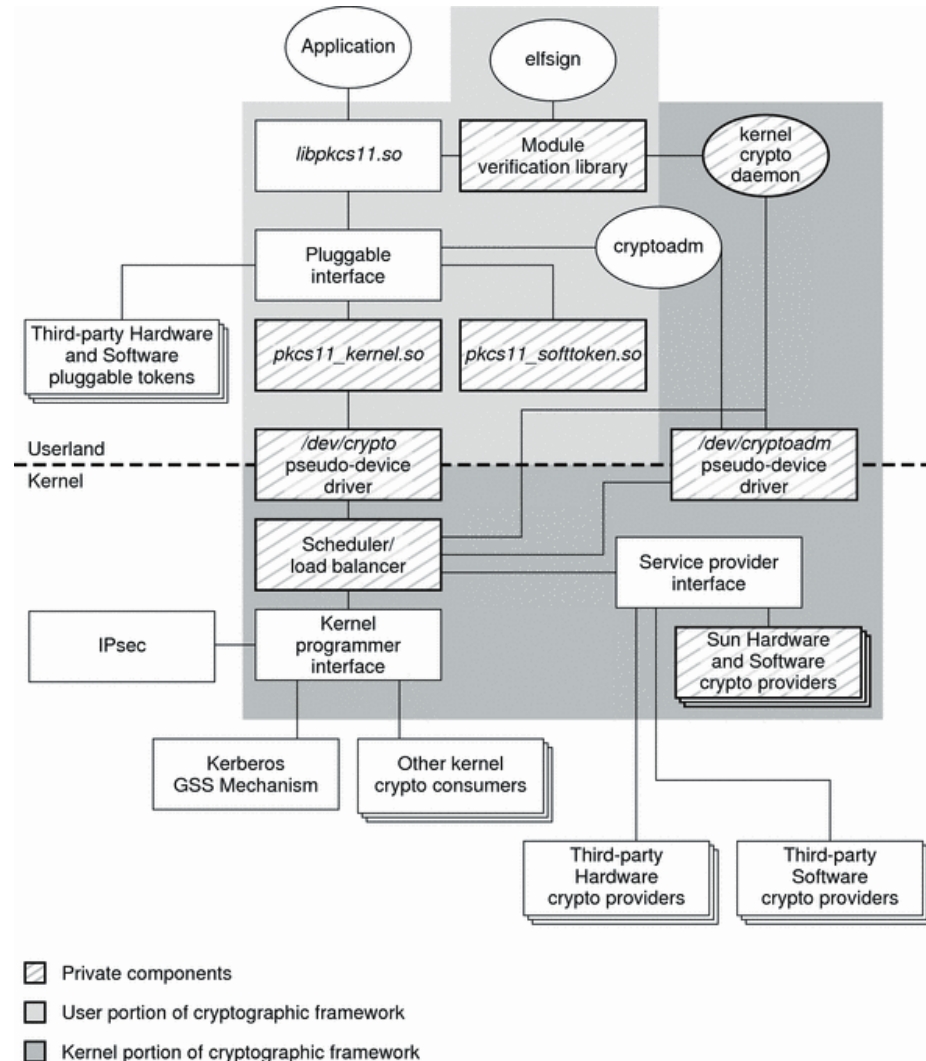
Cryptographic Framework

- Crypto used in lots of places:
 - > Userland: SSL, SSH, GSS, Kerberos, IKE, PAM, ...
 - > Kernel: IPsec (AH, ESP), WiFi drivers, lofi, KSSL, ...
- Problems with duplicate implementations/libs
 - > OpenSSL, Mozilla NSS,...
- Problems with versioning and software defects
- Administrative challenge to utilize hardware acceleration
- Need to take advantage of standard APIs
- Need for pluggable interface

Solaris Cryptographic Framework

- Introduced in Solaris 10
- New features added on ongoing basis
 - > E.g., Niagara2 crypto provider, ECC algorithms/modes
- User-level and kernel-level interfaces
- Administrative commands control the framework
- Consumer-producer architecture
- Benefits:
 - > No duplicate implementations
 - > Optimization for hardware
 - > Transparent use of sw providers & hw accelerators

Solaris Crypto Framework Architecture



User-Level Crypto Framework

- libpkcs11.so is your friend!
- API version 2.20
- Convenience functions. E.g.,
 - > SUNW_C_GetMechSession
 - > SUNW_C_KeyToObject
- Engine for OpenSSL library
- Digest library
 - > libmd.so

Kernel-Level Crypto Framework

- Similar programming logic to PKCS#11
- Kernel-Level software providers
 - > Loadable kernel modules
 - > Kernel-level software providers are synchronous
- Kernel-Level hardware providers
 - > Device nodes and drivers
 - > Kernel-level hardware providers are usually asynchronous
 - > Framework handles scheduling, callbacks, state, etc.

Hardware-accelerated Sw Stacks

- Special shared libraries
- Hardware provider
- Accelerator access
- Scheduling
- Load balancing

Special Shared Libraries

- `pkcs11_kernel.so` and `pkcs11_softtoken.so`
 - > Interact with the kernel-level framework to take advantage of hardware providers
 - > Use the `ioctl` interface of the pseudo device driver, `/dev/crypto`, to invoke operations in the kernel-level framework
 - > `pkcs11_softtoken.so` provides a software implementation of all standard cryptographic algorithms

Hardware provider

- The T1 and T2 cryptographic drivers plug into the Solaris Cryptographic Framework as hardware providers
 - > Makes them transparently accessible to any user of the CF, in both kernel and user space
- The CF selects (based on configuration)
 - > T2 cryptographic hardware for the algorithms which they support
 - > Another provider for those which the hardware does not handle

Accelerator Access

- Cryptographic hardware accessed via a queue interface
- Separate queues for each core
- Different queues for
 - > (A)symmetric cryptographic algorithms
- New requests enqueued on appropriate core's cryptographic queue
- Requests processed in FIFO order
- Requesting threads notified of job completion

Scheduling

- When cryptographic requests arrive the request is scheduled on a core by
 - i. Using the core that thread is currently running on (& binding to avoid migration) or
 - ii. Dispatching the request on a thread already bound on a particular core
- On larger, CMT systems, lock contention is an obvious issue
 - > However, locking is on a per core & queue basis, reducing contention to the 8 strands on a core

Load balancing

- Asymmetric cryptographic requests:
- Round robin policy across all cores
 - > The overhead of a RSA cryptographic operation is significantly larger than the overhead to schedule on a new core
 - > Improves performance by using the accelerators in parallel

Load balancing (cont.)

- Symmetric workload requests:
- Scheduled on the core that the submitting thread is running on
 - > Threads can only interact with the core local accelerator
 - > The overhead to migrate the submitter from one core to another was found to be larger than the time to process the requests!

Load balancing (cont.)

- Solaris thread scheduling provides reasonable load balancing for cryptographic workloads when assigning cryptographic requests to the accelerator of the executing core

Threading CF – async operation

- CF clients can submit multiple async cryptographic requests
- The CF internally maintains a pool of threads that handle these requests concurrently
- This architecture results in improved throughput
 - > Each thread is typically scheduled on a separate core by the Solaris scheduler
 - > Results in each request being handled by a separate cryptographic unit

Peak Crypto T2 Accelerator Performance

Bulk cipher

Algorithm	Gb/s/chip
RC4	83
DES	83
3DES	27
AES-128	44
AES-192	36
AES-256	31

Secure hash

Algorithm	Gb/s/chip
MD5	41
SHA-1	32
SHA-256	41

Public key

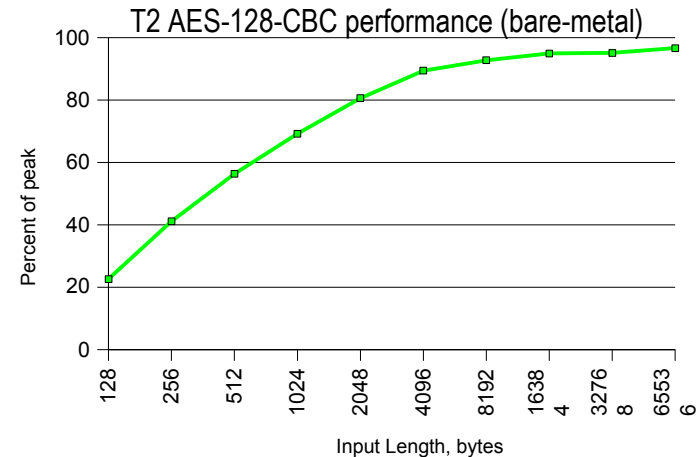
Algorithm	Ops/sec/chip
RSA-1024	37K
RSA-2048	6K
ECCp-160	52K
ECCb-163	92K

Peak Crypto T2 Accelerator Performance (cont.)

- UltraSPARC T2 Plus provides identical crypto support to UltraSPARC T2
 - > 5GB/s of AES-128 throughput achievable when using all 8 accelerators on T2
 - > 10GB/s of AES-128 throughput achievable when using all 16 accelerators in a 2-chip T2 Plus system
- Use of the HW accelerators thwarts many side-channel attacks
 - > Operations have fixed data-independent latency

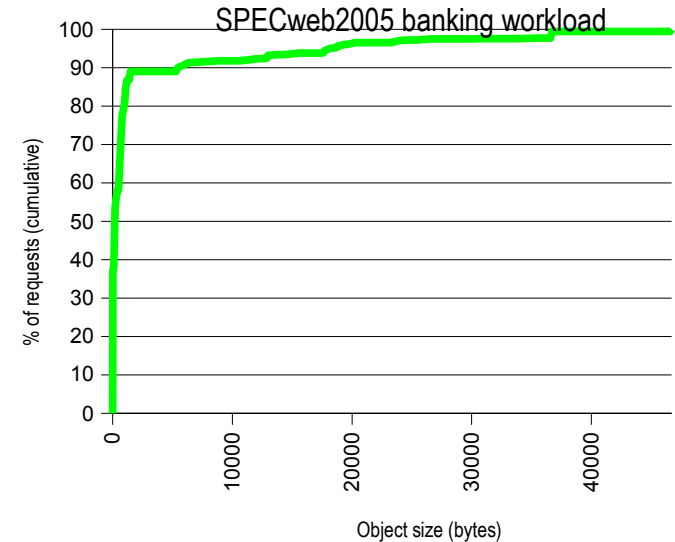
Packet size issues

- On-chip accelerators are capable of cost-effectively offloading even small objects
 - > Much more efficient than offchip cards
 - > Very efficient HW interface
- Handling of small packets is essential in many application spaces
 - > For example, most objects are pretty small in SPECweb05 banking



Packet size issues

- CF has own overhead before the data block transferred to accelerator
- There is a break-even point for every algorithm
- Determines when it is more efficient to offload to the hardware
 - > Can be fairly large for light-weight ciphers (e.g. RC4)



Packet size – breakeven point

```
$ openssl speed -evp aes-128-cbc
```

```
Doing aes-128-cbc for 3s on 16 size blocks: 2288268 aes-128-cbc's in 2.99s
Doing aes-128-cbc for 3s on 64 size blocks: 703001 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 256 size blocks: 186333 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 384 size blocks: 125058 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 512 size blocks: 94113 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 640 size blocks: 75442 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 768 size blocks: 62952 aes-128-cbc's in 3.00s
...
```

Without HW acceleration

Breakeven point

```
$ openssl speed -evp aes-128-cbc -engine pkcs11 -elapsed
```

```
Doing aes-128-cbc for 3s on 16 size blocks: 121829 aes-128-cbc's in 2.99s
Doing aes-128-cbc for 3s on 64 size blocks: 120523 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 256 size blocks: 116682 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 384 size blocks: 113612 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 512 size blocks: 109206 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 640 size blocks: 105578 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 768 size blocks: 102965 aes-128-cbc's in 3.00s
```

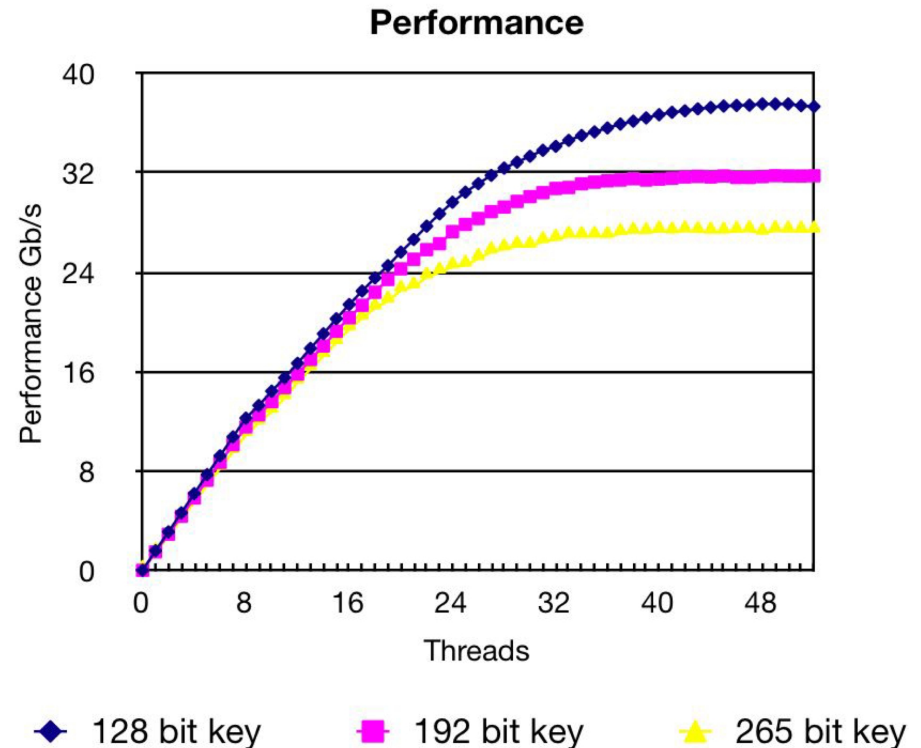
With HW acceleration

Real-world example – SSH

- SunSSH client and server use the OpenSSL PKCS#11 engine
 - > OpenSSL in Solaris uses the CF
- On the UltraSPARC T2 processor using the HW accelerators improved data throughput by 2.5x
- SunSSH uses AES counter mode
 - > Allows AES blocks to be processed in parallel
 - > Normal modes like CBC have inter-block data dependencies
- Further improvements with mult. threads

AES Performance

- Varying key size
- AES Counter mode
- T5120 w/ 8 cores
- 50 threads
- Peaks:
> 37, 32, 27 GB/s



Conclusions

- Transparent access to hardware accelerated functionality via a software stack
- Access from both kernel and user space
- Tremendous performance gains through on-chip accelerator hardware



Thank you!

Questions?

Dr. Christoph Schuba

Christoph.Schuba@Sun.COM

<http://blogs.sun.com/schuba>

