SSLShader: Cheap SSL Acceleration with Commodity Processors

Keon Jang\textsuperscript{+}, Sangjin Han\textsuperscript{+}, Seungyeop Han\textsuperscript{*}, Sue Moon\textsuperscript{+}, and KyoungSoo Park\textsuperscript{+}

KAIST\textsuperscript{+} and University of Washington\textsuperscript{*}
Security of Paper Submission Websites

NDSS 2011 Sign in

Sunday 27 Mar 2011 9:46:58pm EDT
Your local time: Monday 28 Mar 2011 11:52:03am

http://hotcrp.com/
Transfer Protocol
data: application/x-www-form-urlencoded
email=foobar@an.kaist.ac.kr&password=thisispassword

---

papers were accepted out of
Security Threats in the Internet

- Public WiFi without encryption
  - Easy target that requires almost no effort

- Deep packet inspection by governments
  - Used for censorship
  - In the name of national security

- NebuAd’s targeted advertisement
  - Modify user’s Web traffic in the middle
Secure Sockets Layer (SSL)

- A de-facto standard for secure communication
  - Authentication, Confidentiality, Content integrity

Client

TCP handshake

Key exchange using public key algorithm (e.g., RSA)

Server identification

Server

Encrypted data
SSL Deployment Status

- Most of Web-sites are not SSL-protected
  - Less than 0.5%
    - [NETCRAFT Survey Jan ‘09]

- Why is SSL not ubiquitous?
  - Small sites: lack of recognition, manageability, etc.
  - Large sites: cost
    - SSL requires lots of computation power
SSL Computation Overhead

- Performance overhead (HTTPS vs. HTTP)
  - Connection setup: 22x
  - Data transfer: 50x

- Good privacy is expensive
  - More servers
  - H/W SSL accelerators

- Our suggestion:
  - Offload SSL computation to GPU
SSLShader

- SSL-accelerator leveraging GPU
  - High-performance
  - Cost-effective

- SSL reverse proxy
  - No modification on existing servers
Our Contributions

- **GPU cryptography optimization**
  - The fastest RSA on GPU
  - Superior to high-end hardware accelerators
  - Low latency

- **SSLShader**
  - Complete system exploiting GPU for SSL processing
    - Batch processing
    - Pipelining
    - Opportunistic offloading
    - Scaling with multiple cores and NUMA nodes
CRYPTOGRAPHIC PROCESSING WITH GPU
How GPU Differs From CPU?

Intel Xeon 5650 CPU:
6 cores

NVIDIA GTX580 GPU:
512 cores

Instructions / sec

$62 \times 10^9 < 870 \times 10^9$
Example code: vector addition ($C = A + B$)

**CPU code**

```c
void VecAdd(
    int *A, int *B, int *C, int N)
{
    //iterate over N elements
    for(int i = 0; i < N; i++)
        C[i] = A[i] + B[i]
}
VecAdd(A, B, C, N);
```

**GPU code**

```c
__global__ void VecAdd(
    int *A, int *B, int *C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i]
}
//Launch N threads
VecAdd<<<1, N>>>(A, B, C);
```
Parallelism in SSL Processing

1. Independent Sessions

2. Independent SSL Record

3. Parallelism in Cryptographic Operations
Our GPU Implementation

- Choices of cipher-suite
  - Encryption: **AES**
  - Message Authentication: **SHA1**
  - Key exchange: **RSA**

- Optimization of GPU algorithms
  - Exploiting massive parallel processing
    - Parallelization of algorithms
    - Batch processing
  - Data copy overhead is significant
    - Concurrent copy and execution
Basic RSA Operations

- $M$: plain-text, $C$: cipher-text
- $(e, n)$: public key, $(d, n)$: private key

- Encryption: $C = M^e \mod n$
- Decryption: $M = C^d \mod n$

Small number: 3, 17, 65537

1024/2048 bits integer (300 ~ 600 digits)

Exponentiation $\rightarrow$ many multiplications
Breakdown of Large Integer Multiplication

Schoolbook multiplication

\[
\begin{array}{c}
649 \\
\times 627 \\
\hline
63 \\
280 \\
4200 \\
180 \\
800 \\
12000 \\
5400 \\
32000 \\
+ 360000 \\
\hline
406923
\end{array}
\]

Accumulation is difficult to parallelize due to

“overlapping digits”

“carry propagation”

3 x 3 = 9 multiplications
9 addition of 6-digits integers
$O(s)$ Parallel Multiplications

$s = \# \text{ of words in a large integer} \quad \text{(E.g., 1024-bits = 16 x 64 bits word)}$

Example of
$649 \times 627 = 406,923$

2s steps

1 or 2 steps
(s – 1 worst case)
More Optimizations on RSA

- Common optimizations for RSA
  - Chinese Remainder Theorem (CRT)
  - Montgomery Multiplication

- Parallelization of serial algorithms
  - Faster Calculation of $M \times n$
  - Interleaving of $T + M \times n$
  - Mixed-Radix Conversion Offloading

- GPU specific optimizations
  - Warp Utilization
  - Loop Unrolling
  - Elimination of Divergence
  - Avoiding Bank Conflicts
  - Instruction-Level Optimization

Read our paper for details 😊
Parallelism in SSL Processing

1. Independent Sessions

2. Independent SSL Record

3. Parallelism in Cryptographic Operations

Batch Processing
GTX580 Throughput w/o Batching

Throughput relative to a “single CPU core”

Intel Nehalem single core (2.66Ghz)

- RSA: 0.08x
- AES-ENC: 0.02x
- AES-DEC: 1.57x
- SHA1: 0.02x
Batch size: **32~4096** depending on the algorithm

Throughput relative to a "single CPU core"

Difference: ratio of computation to copy
Copy Overhead in GPU Cryptography

- GPU processing works by
  - Data copy: CPU → GPU
  - Execution in GPU
  - Data copy: GPU → CPU

Throughput (Gbps)

- AES-ENC: 2.4x
- AES-DEC: 3.3x
- HMAC-SHA1: 4x

GPU with copy vs. without copy
Hiding Copy Overhead

Synchronous Execution

Data copy: CPU -> GPU
Execution in GPU
Data copy: GPU -> CPU
Processing time : $3t$

Pipelining

Data copy: CPU -> GPU
Execution in GPU
Data copy: GPU -> CPU
Amortized processing time : $t$
GTX580 Performance w/ Pipelining

Throughput relative to a single core

- AES-ENC: $9 \times$ synchronous, $9 \times$ pipelining, $36\%$ increase
- AES-DEC: $9 \times$ synchronous, $9 \times$ pipelining, $36\%$ increase
- SHA1: $14 \times$ synchronous, $51\%$ increase
Summary of GPU Cryptography

- **Performance gain from GTX580**
  - GPU performs as fast as $9 \sim 28$ CPU cores
  - Superior to high-end hardware accelerators

<table>
<thead>
<tr>
<th></th>
<th>RSA-1024 (ops/sec)</th>
<th>AES-ENC (Gbps)</th>
<th>AES-DEC (Gbps)</th>
<th>SHA1 (Gbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX580</td>
<td>91.9K</td>
<td>11.5</td>
<td>12.5</td>
<td>47.1</td>
</tr>
<tr>
<td>CPU core</td>
<td>3.3K</td>
<td>1.3</td>
<td>1.3</td>
<td>3.3</td>
</tr>
</tbody>
</table>

- **Lessons**
  - Batch processing is essential to fully utilize a GPU
  - AES and SHA1 are bottlenecked by data copy
    - PCIe 3.0
    - Integrated GPU and CPU
BUILDING SSL-PROXY THAT LEVERAGES GPU
SSLShader Design Goals

- Use existing application without modification
  - SSL reverse proxy

- Effectively leverage GPU
  - Batching cryptographic operations
  - Load balancing between CPU and GPU

- Scale performance with architecture evolution
  - Multi-core CPUs
  - Multiple NUMA nodes
Batching Crypto Operations

- Network workloads vary over time
  - Waiting for fixed batch size doesn’t work

- Batch size is dynamically adjusted to queue length
Balancing Load Between CPU and GPU

- For small batch, CPU is faster than GPU
  - Opportunistic offloading

Diagram:

- Input queue
- Output queue
- CPU
- GPU
- GPU queue

- CPU processing
- GPU processing
  when input queue length > threshold

- RSA (1024-bit) 16 512
- AES
  - Decryption 32 2048
  - Encryption 128 2048
- HMAC - SHA1 128 2048
Scaling with Multiple Cores

- **Per-core worker threads**
  - Network I/O, cryptographic operation
- **Sharing a GPU with multiple cores**
  - More parallelism with larger batch size
A process = worker threads + a GPU thread
  - Separate process per NUMA node
  - Minimizes data sharing across NUMA nodes
Evaluation

- Experimental configurations

<table>
<thead>
<tr>
<th>Model</th>
<th>Spec</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel X5650</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2.66Ghz x 6 cores</td>
<td></td>
</tr>
<tr>
<td>GPU</td>
<td>NVIDIA GTX580</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1.5Ghz x 512 cores</td>
<td></td>
</tr>
<tr>
<td>NIC</td>
<td>Intel X520-DA2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>10GbE x 2</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation Metrics

- HTTPS connection handling performance
  - Use small content size
  - Stress on RSA computation

- Latency distribution at different loads
  - Test opportunistic offloading

- Data transfer rate at various content size
HTTPS Connection Rate

Connections / sec

<table>
<thead>
<tr>
<th>RSA Key Size</th>
<th>SSLShader</th>
<th>lighttpd</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024 bits</td>
<td>29K</td>
<td>11K</td>
</tr>
<tr>
<td>2048 bits</td>
<td>21K</td>
<td>3.6K</td>
</tr>
</tbody>
</table>

- 2.5x improvement for 1024 bits SSLShader compared to lighttpd
- 6x improvement for 2048 bits SSLShader compared to lighttpd
CPU Usage Breakdown (RSA 1024)

Current Bottleneck

Kernel (Including TCP/IP stack), 60.35

Libc, 9.88

SSLShader, 5.31

IPP + libcrypto, 12.89

lighttpd, 4.9

Kernel NIC device driver, 2.32

others, 4.35
Latency at Light Load

CDF (%)

Latency (ms)

Lighttpd at 1k connections / sec

SSLShader at 1k connections / sec

Similar latency at light load
Lower latency and higher throughput at heavy load.
Data Transfer Performance

Typical web content size is under 100KB

SSLShader: 13 Gbps

Lighttpd performance

Relative Performance

Content Size

4KB 16KB 64KB 256KB 1MB 4MB 16MB 64MB
CONCLUSIONS
Summary

- **Cryptographic algorithms in GPU**
  - Fast RSA, AES, and SHA1
  - Superior to high-end hardware accelerators

- **SSLShader**
  - Transparent integration
  - Effective utilization of GPU for SSL processing
    - Up to 6x connections / sec
    - 13 Gbps throughput

Linux network stack performance
Copy overhead
For more details
https://shader.kaist.edu/sslshader

QUESTIONS?

THANK YOU!