SSLShader: Cheap SSL Acceleration with Commodity Processors

Keon Jang+, Sangjin Han+, Seungyeop Han*, Sue Moon+, and KyoungSoo Park+

KAIST+ and University of Washington*
Security of Paper Submission Websites
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

![Diagram of SSL process]

- Server identification
- TCP handshake
- Key exchange using public key algorithm (e.g., RSA)
- 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
  - Data transfer
    - 22x
    - 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

Image: Diagram showing SSLShader as an SSL-accelerator and reverse proxy, connecting to various servers (Web, SMTP, POP3) with SSL-encrypted sessions and plain TCP connections.
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

\[ 62 \times 10^9 \leq 870 \times 10^9 \]
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: \( M = C^d \mod n \)

- Exponentiation \( \rightarrow \) many multiplications

1024/2048 bits integer (300 ~ 600 digits)

Small number: 3, 17, 65537
Schoolbook multiplication

\[
\begin{array}{c}
649 \\
\times \\n627 \\
\hline \\
63 \\
280 \\
4200 \\
180 \\
800 \\
12000 \\
5400 \\
32000 \\
+ \\
360000 \\
\hline \\
406923
\end{array}
\]

Accumulation is difficult to parallelize due to

“overlapping digits”

“carry propagation”

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

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

Example of
649 x 627 = 406,923
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

Batch Processing

2. Independent SSL Record

3. Parallelism in Cryptographic Operations
GTX580 Throughput w/o Batching

Throughput relative to a “single CPU core”

Intel Nehalem single core (2.66Ghz)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA</td>
<td>0.08x</td>
</tr>
<tr>
<td>AES-ENC</td>
<td>0.02x</td>
</tr>
<tr>
<td>AES-DEC</td>
<td>1.57x</td>
</tr>
<tr>
<td>SHA1</td>
<td>0.02x</td>
</tr>
</tbody>
</table>
GTX580 Throughput w/ Batching

Batch size: **32~4096** depending on the algorithm

Throughput relative to a "single CPU core"

- **RSA**: 22.1x
- **AES-ENC**: 6.8x
- **AES-DEC**: 7.7x
- **SHA1**: 9.4x

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

![Graph showing throughput comparison between GPU and CPU with and without data copy. The graph indicates an increase in throughput by 2.4x, 3.3x, and 4x for AES-ENC, AES-DEC, and HMAC-SHA1 respectively, when copying data to GPU is avoided.]
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: 9x synchronous, 9x pipelining, +36%
- AES-DEC: 9x synchronous, 9x pipelining, +36%
- SHA1: 14x synchronous, 14x pipelining, +51%
### Summary of GPU Cryptography

- **Performance gain from GTX580**
  - GPU performs as fast as 9 ~ 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
Network workloads vary over time
- Waiting for fixed batch size doesn’t work

Batch size is dynamically adjusted to queue length

Diagram:
- SSL Stack
- Input queue
- Output queue
- CPU
- GPU
Balancing Load Between CPU and GPU

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

![Diagram showing the flow of tasks between CPU and GPU](image)

- Input queue
- CPU processing
- GPU queue
- GPU processing when input queue length > threshold
- Output queue
- Per-core worker threads
  - Network I/O, cryptographic operation
- Sharing a GPU with multiple cores
  - More parallelism with larger batch size
Scaling with NUMA systems

- A process = worker threads + a GPU thread
  - Separate process per NUMA node
  - Minimizes data sharing across NUMA nodes
### Evaluation

- **Experimental configurations**

**Server Specification**

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

35,000
30,000
25,000
20,000
15,000
10,000
5,000
0

RSA Key Size

1024 bits
2048 bits

SSLShader
lighttpd

29K
11K
21K
3.6K

2.5x
6x
CPU Usage Breakdown (RSA 1024)

- Kernel (Including TCP/IP stack), 60.35%
- Libc , 9.88%
- SSLShader, 5.31%
- lighttpd, 4.9%
- IPP + libcrypto, 12.89%
- others, 4.35%
- Kernel NIC device driver, 2.32%

Current Bottleneck
Latency at Light Load

Similar latency at light load

Lighttpd at 1k connections / sec

SSLShader at 1k connections / sec
Lower latency and higher throughput at heavy load
Typical web content size is under 100KB

SSLShader: 13 Gbps

Lighttpd performance

2.1x

0.87x
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!