KIPS Tr. Comp. and Comm. Sys.
Vol.4, No.8 pp.239~244 pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2015.4.8.239

Profiler Design for Evaluating Performance of WebCL Applications

Cheolwon Kim† · Hyeonjoong Cho††

ABSTRACT

WebCL was proposed for high complex computing in Javascript. Since WebCL-based applications are distributed and executed on an unspecified number of general clients, it is important to profile their performances on different clients. Several profilers have been introduced to support various programming languages but WebCL profiler has not been developed yet. In this paper, we present a WebCL profiler to evaluate WebCL-based applications and monitor the status of GPU on which they run. This profiler helps developers know the execution time of applications, memory read/write time, GPU statues such as its power consumption, temperature, and clock speed.

Keywords : WebCL, Profiler, NVML

WebCL 기반 애플리케이션의 성능 평가를 위한 프로파일러 설계 및 구현

김 철 원† · 조 현 중††

요 약

자바스크립트 상에서 높은 연산량을 처리하기 위해 제안된 WebCL은 불특정 클라이언트 환경에서 실행되므로 개별 클라이언트에서 애플리케이션의 성능 평가 작업이 중요하다. 현재 다양한 프로파일러들이 서비스 되고 있지만 WebCL을 위한 프로파일러는 아직 개발되어있지 않다. 본 논문에서는 웹 이기종 병렬컴퓨팅 언어인 WebCL 기반으로 구현된 애플리케이션의 성능 평가 및 GPU 상태 정보를 모니터링하기 위한 프로파일러를 설계 및 구현하여 소개한다. 본 프로파일러를 통하여 사용자는 WebCL 기반 애플리케이션의 수행시간 및 메모리 읽기/쓰기 시간을 알 수 있고, GPU 디바이스의 소비 전력, 현재 온도, 클락 속도 등 현재 상태를 실시간 모니터링할 수 있다.

키워드 : WebCL, 프로파일러, NVML

1. 서 론

최근 웹 브라우저 기반의 대용량 멀티미디어 서비스, 고성능 그래픽스 처리, 소셜 게임과 같은 다양한 웹 애플리케이션이 서비스 되면서 클라이언트 웹 언어인 자바스크립트의 복잡한 계산이 요구되고 있다. 하지만 자바스크립트는 구조적으로 순차적으로 수행되므로 고성능 계산을 위한 병렬 처리에 어려움이 있다. 이를 해결하기 위한 대표적 방법으로 웹 이기종 병렬컴퓨팅을 지원하는 WebCL(Web Computing Language)이 제안되었다[1]. WebCL은 작성된 별도의 커널 함수 스크립트를 통해 OpenCL을 자바스크립트로 바인딩 하여 GPU(Graphics Processing Unit) 같은 이기종 디바이스나 멀티프로세서 자원을 이용하여 웹 애플리케이션을 가속화하기 위한 병렬 처리 기술이다.

이기종 병렬컴퓨팅은 CPU에서 다른 디바이스로 태스크 또는 데이터를 분할, 할당하고 계산을 분담하여 가속화하는 방법이기에 연산의 효율성, 메모리 읽기/쓰기로 인한 지연시간, 동기화 등의 문제가 발생되며, 성능 프로파일링을 통하여 시스템의 효율성을 평가하는 작업이 중요하다. 또한 WebCL은 개발자가 작성한 스크립트가 클라이언트의 브라우저와 디바이스에서 운영되기 때문에 상기 성능 프로파일 외에도 디바이스의 상태, 소비 전력, 누수, 온도를 모니터링 할 필요가 있다.

기존에 존재하던 다양한 언어들은 각각의 프로파일링 툴이 서비스 되고 있으나 현재까지 WebCL을 위한 프로파일링 툴은 제공되고 있지 않다. 또한 스크립트 기반으로 작성

※ 이 논문은 중소기업청에서 시행하는 산학연협력 기술개발사업(Q1430514) 지원을 받아 수행되었음.
† 비 회 원 : 고려대학교 컴퓨터정보학과 석사과정
†† 종신회원 : 고려대학교 컴퓨터정보학과 부교수
Manuscript Received : February 24, 2015
First Revision : May 20, 2015; Second Revision : August 12, 2015
Accepted : August 13, 2015
* Corresponding Author : Hyeonjoong Cho(raycho@korea.ac.kr)

된 웹 애플리케이션이 동작할 때 GPU 상태를 모니터링 해주는 프로파일러는 아직 존재하지 않는다. 따라서 본 논문에서는 WebCL로 작성된 스크립트를 프로파일링 해주는 WebCL 프로파일러를 제안하고 이를 구현한다. WebCL 프로파일러는 (1) 작성된 커널 함수들의 수행시간, 디바이스의 메모리 읽기/쓰기 시간과 (2) GPU 디바이스의 상태 모니터링 정보를 사용자에게 제공한다. 스크립트의 시간 정보를 프로파일링 하기 위하여, Record/replay 방식을 적용하여 WebCL API의 프로파일링 이벤트 정보를 활용하였다. 또한 개발자가 NVIDIA® Tesla® Kelper 아키텍처 GPU를 사용한다면 소스 코드가 실행되는 동안의 GPU의 소비 전력, 온도, GPU 클락 속도, 메모리 정보들을 NVML(NVIDIA Management Library)을 사용하여 실시간 모니터링도 지원한다. NVML[2]은 전력, 온도, 클락, 메모리, 효율 등 NVIDIA GPU의 상태를 관리해주는 C-기반의 라이브러리이다. 개발자는 이를 통하여 제작한 웹 애플리케이션의 전력 소비의 변화를 그래프로 확인하고 전력 누수를 방지할 수 있다. 본 논문의 WebCL Profiler는 오픈소스로 개발되었다[12].

2. 관련 연구

프로그램의 성능 평가, 코드 최적화, 디버깅을 위한 프로파일러들은 지속적으로 개발되어왔다. Unix/Linux 애플리케이션의 성능 평가를 위한 프로파일러 GNU gprof[3]는 1982년 L. Graham이 개발하였고 현재까지 많이 사용되고 있는 대표적인 프로파일러이다. gprof는 함수들의 수행시간을 측정하고 call-graph, 함수 호출과 프로시저 사이의 관계를 프로파일링 하여 화면에 보여준다.

대표적인 이기종 병렬컴퓨팅 언어 OpenCL과 CUDA는 프로파일링을 위한 툴로 AMD App Profiler[4], CUDA Visual profiler[5]가 서비스 되고 있다. 이 툴들은 프로그램의 커널 함수의 실행 시간, 메모리 읽기/쓰기 시간, 데이터들의 병목현상 등의 정보를 프로파일링 하여 개발자에게 제공한다. 하지만 이들은 시스템의 하위 레벨에서 프로파일 작업이 수행되므로 프로파일러 개발자가 툴의 동작을 정확히 이해하는 데 어려움이 있다[6]. 반면 Haptic[6]은 OpenCL의 Record/replay 분석 방식의 프로파일러이다. 상위 레벨에서 프로파일 작업이 이루어지며, OpenCL SDK의 이벤트 정보를 활용하여 디바이스의 효율성을 분석한다.

또한 웹 스크립트 언어의 프로파일러는 MSDN, Chrome, Mozilla 등의 웹 브라우저에서 제공하는 기본 프로파일러들과 Firebug, spy-js와 같이 오픈소스로 제공되는 다양한 프로파일러들이 서비스 되고 있다. 위 프로파일러들은 웹 스크립트 언어의 특징에 맞게 디버깅에 초점을 맞춘 기능을 제공하며 HTML, Javascript, CSS, DOM의 함수 실행 시간, 함수의 호출 횟수 등의 기능 또한 개발자에게 제공한다[7-11].

이들 중 Firebug 프로파일러는 오픈소스로 간편하게 기능을 확장할 수 있는 이점이 있어서 본 논문의 WebCL Profiler를 구현하기 위한 기초로 활용하였다. 본 논문에서 제안하는 WebCL Profiler는 WebCL을 위한 최초의 프로파일러이다. WebCL Profiler는 스크립트 기반의 프로파일러이지만, 다른 이기종 병렬컴퓨팅 언어들의 프로파일러들과 같이 디버깅보다는 성능 모니터링에 초점을 둔다. 또한 Haptic[6] 프로파일러와 같이 Record/replay 방식의 방법을 통하여 상위 레벨에서 성능 측정 프로파일 작업이 이루어진다. 특히, WebCL 기반의 웹 애플리케이션은 GPU 디바이스를 사용하는데, GPU 모니터링 정보를 WebCL Profiler를 통해서 사용자에게 제공함으로써 개발자는 편리하게 웹 애플리케이션의 성능을 측정하고 클라이언트는 보다 안정적인 서비스를 제공받을 수 있게 된다. 프로파일러의 GPU 모니터링을 위해 본 논문에서는 NVML을 사용하였다.

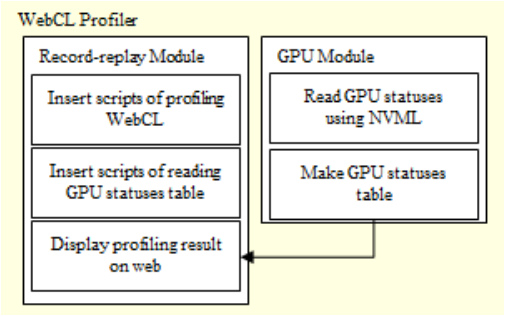

Fig. 1. System Architecture of WebCL Profiler

3. 본 론

Fig. 1을 보면, WebCL 프로파일러는 (1) 작성된 스크립트에 프로파일링을 위한 WebCL 이벤트 정보를 삽입하고, 실행하는 웹 브라우저의 Record-replay 모듈과 (2) 스크립트가 실행되는 동안의 GPU 상태 정보를 읽어 테이블로 만드는 GPU 모듈 등 두 부분으로 구성되어있다.

GPU의 상태를 모니터링하기 위한 NVML은 아직 웹 버전이 지원되고 있지 않음에 따라 별개의 모듈로 구성하였다. GPU 모듈은 시스템 백그라운드로 실행되며, GPU의 상태를 실시간으로 모니터링한다. 또한 웹 브라우저 모듈과는 XMLHttpRequest 통신을 통하여 GPU 상태 테이블을 제공한다.

3.1 WebCL 스크립트 프로파일링

Fig. 2(a)에 보이는 바와 같이 개발자가 작성한 WebCL 스크립트가 실행되면, WebCL 프로파일러는 이벤트 객체들을 삽입하고 GPU 상태 테이블을 읽기 위한 스크립트를 삽입한다. 이벤트 객체들은 스크립트가 실행되는 동안의 커널 함수, 메모리의 시간 정보와 상태 정보를 OpenCL SDK를 통해서 받아온다. WebCL 스크립트는 자바스크립트로 구성되어있어서 실험 파일을 생성하고, 수정하고, 실행할 수 있다.

Fig. 3는 이 과정을 설명하고 있는데, (1) 사전 작업으로 복사된 스크립트에 주석을 제거하고, (2) 커널 함수명을 찾고 정의된 함수 안에서 context, commandQueue 변수를 찾는다.

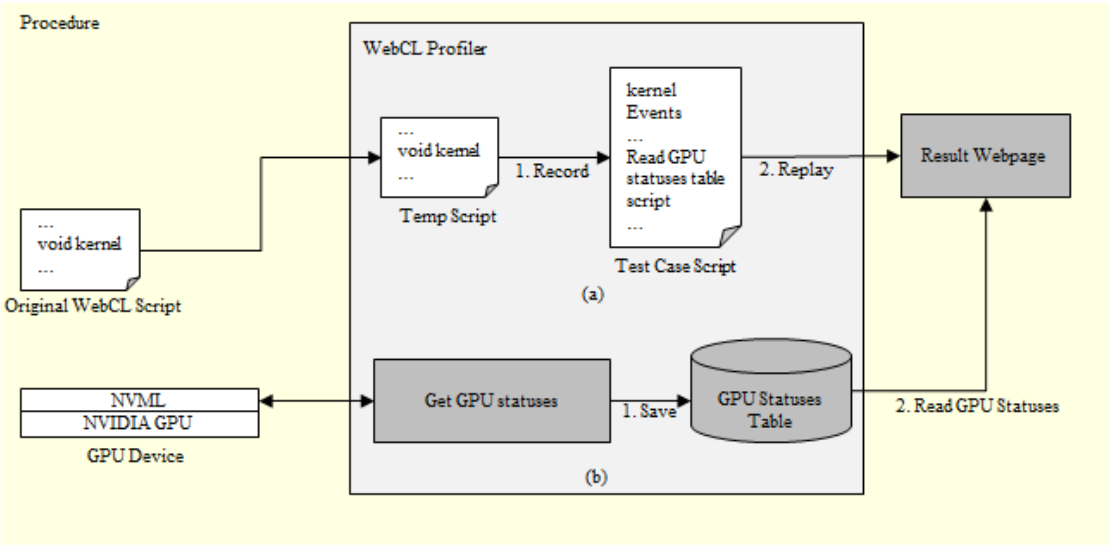

Fig. 2. Procedure of the WebCL Profiler (a) Part of Record (b) Part of Reading GPU Statues and Saving It

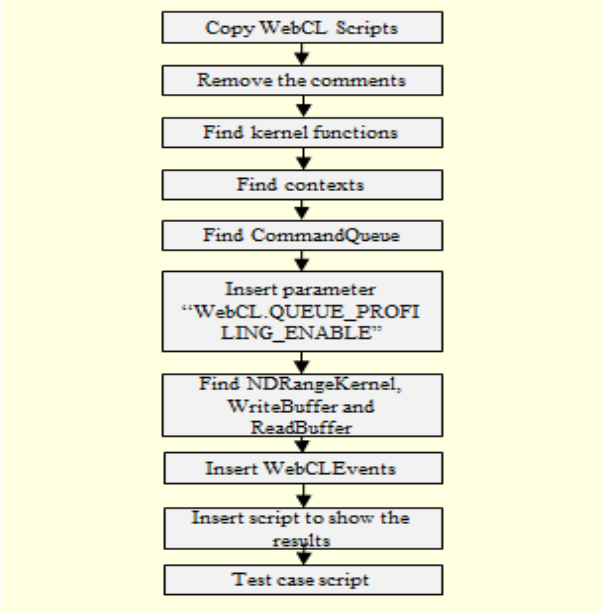

Fig. 3. Flow Chart of Recording Step

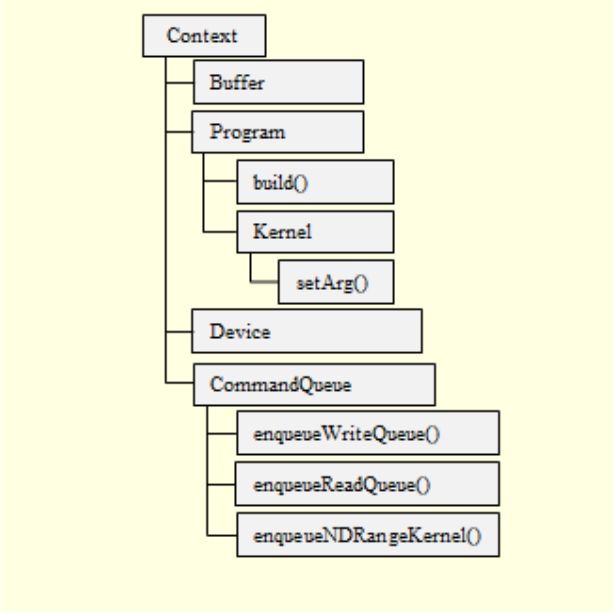

Fig. 4. Tree of Variable Degree in WebCL

Fig. 4를 보면, WebCL 요소의 뿌리는 context이며 WebCL의 buffer, program, kernel, device, commandQueue 변수와 관련된다. (3) WebCL 스크립트의 프로파일링을 위하여 commandQueue가 생성될 때 "QUEUE_PROFILING_ENABLE"가 파라미터 값으로 있어야 하므로 추가한다. (4) WebCL 이벤트 객체는 Fig. 4의 WebCL 요소 중 디바이스에 데이터를 보내는 enqueueWriteBuffer와 커널 함수를 실행하는 enqueueNDRangeKernel, 그리고 디바이스로부터 데이터를 받아오는 enqueueReadBuffer의 파라미터로 사용되므로, 위의 (2)에서 찾은 요소들을 기반으로 사용된 해당 함수들을 찾아서 이벤트 객체를 선언하고 파라미터에 추가한다. (5) 마지막으로 실행되면서 이벤트 객체에 기록된 프로파일링 결과를 보여주는 스크립트를 삽입한다. 위의 과정들을 통하여 WebCL 이벤트 객체가 삽입된 스크립트가 실행되고, 프로파일러는 사용자에게 결과를 보여주도록 구현되었다.

3.2 GPU 상태 모니터링

GPU 모듈은 C-CUDA로 구현되었으며 NVML을 사용하여 GPU의 디바이스 이름, 메모리 용량, 최대 사용할 수 있는 전력량, 클락, 메모리 클락과 현재 사용 중인 전력량, 온도를 받아와 테이블을 실시간으로 만든다. Fig. 2과 같이 실시간으로 만들어지는 테이블 정보를 WebCL Profiler가

Fig. 5. Snapshot of Profiling Results (a) Graphs Showing Power Consumption and Temperature (b) Execution Time of Kernel Function Named CkVectorMul (c) Uploading GPU Statues in Firebug

XMLHttpRequest 통신을 통하여 불러들여 모니터링할 수 있도록 구현하였다.

NVIDIA GPU를 사용하는 개발자, 클라이언트들은 대부분 단일 Geforce 계열 그래픽 카드를 사용한다. 하지만 GPU 모니터링에 강력한 기능을 제공하는 NVML은 Geforce 계열 그래픽 카드에서 사용하기에는 제약이 많다. WebCL 스크립트는 불특정 클라이언트에게 배포되는 만큼 개발자는 작성한 스크립트가 실행될 때, GPU의 상태를 모니터링 후 배포할 책임이 있다. NVIDIA Tesla Kepler 아키텍처 이상의 GPU를 사용하면 제약 없이 NVML을 이용하여 GPU의 상태를 모니터링하여 작성한 스크립트의 결함 유무를 판단할 수 있다.

4. 실 험

본 연구에서는 WebCL 애플리케이션의 성능 프로파일링과 GPU 상태를 모니터링하는 실험을 위하여, NVIDIA® GeForce® GTX 650과 NVIDIA Tesla® K20c GPU를 실험 환경으로 구성하였다. 또한 실험을 위한 스크립트는 병렬컴퓨팅의 성능을 보여주기 위하여 일반적으로 많이 사용되는 Table 1과 같은 행렬의 곱을 사용하였다.

Table 1의 스크립트를 대상으로 WebCL Profiler는 Fig. 4에 해당하는 스크립트에서 사용된 WebCL 요소들의 구성을 파악하고 WebCL 이벤트 객체들을 스크립트에 삽입한다. 이후 GPU의 상태 테이블을 읽는 스크립트와 결과를 보여주는 스크립트를 삽입함으로써 Fig. 2(a)의 Test Case Script를 만들어낸다. Table 2는 Table 1에서 WebCL 이벤트 객체들이 삽입되어 변경된 Test Case Script의 스크립트를 보여준다. 이와 동시에 Fig. 2(b)와 같이 GPU의 상태는 계속 모니터링 되고 테이블 형태로 저장되고 있으며 스크립트가 실행됨으로써 수행된 결과가 Fig. 5와 같이 웹 브라우저에 나타난다.

Fig. 5는 샘플 스크립트의 프로파일링 된 결과 화면을 보여준다. Fig. 5(a)를 통하여, WebCL 스크립트가 실행되면 개발자는 그래프를 통하여 전력 소비와 온도를 확인할 수 있다. 스크립트가 실행되고 디바이스가 사용된 순간부터 사용되는 전력량이 급증하고 사용이 끝나는 순간 급락하는 것을 확인할 수 있으며, 온도는 천천히 상승하는 것을 확인할 수 있다. 이와 같은 그래프를 그려주는 스크립트는 Table 2의 <html><body>의 사이에 추가된다. Fig. 5(b)는 행렬의 곱을 위한 두 행렬을 디바이스가 읽기 위한 시간과 연산에 사용된 시간, 디바이스에서 결과 값을 가져오는 시간을 파

Table 1. WebCL Scripts of Matrices Multiplication

Javascript

```
<html>
<body>
1. <script type="text/javascript">
2. function vectorMul () {
3.  var vectorLength = 4096*4096;
4.  var UIvector1 = new Uint32Array(vectorLength);
5.  var UIvector2 = new Uint32Array(vectorLength);
6.  var ctx = webcl.createContext();
7.  var bufSize = vectorLength * 4;
8.  var bufIn1 = ctx.createBuffer
          (WebCL.MEM_READ_ONLY, bufSize);
9.  var bufIn2 = ctx.createBuffer
          (WebCL.MEM_READ_ONLY, bufSize);
10.  var bufOut = ctx.createBuffer
          (WebCL.MEM_WRITE_ONLY, bufSize);
11.  var device = ctx.getInfo(
                        WebCL.CONTEXT_DEVICES)
[0];
12.  var cmdQueue = ctx.createCommandQueue (device);
13.  var kernelSrc = loadKernel("clProgramVectorMul");
14.  var program = ctx.createProgram(kernelSrc);
15.   try { program.build ([device], ""); }catch(e) {throw
e;}
16.  var kernel = program.createKernel ("clVectorMul");
17.  kernel.setArg (0, bufIn1);
18.  kernel.setArg (1, bufIn2);
19.  kernel.setArg (2, bufOut);
20.  kernel.setArg (3, new Uint32Array([2048]));
21.  cmdQueue.enqueueWriteBuffer(bufIn1,
                        false, 0, bufSize, UIvector
1);
22.  cmdQueue.enqueueWriteBuffer(bufIn2,
                        false, 0, bufSize, UIvector
2);
23.  var localWS = [16,16];
24.  var globalWS = [4096,4096];
25.  cmdQueue.enqueueNDRangeKernel(kernel,
      globalWS.length, null, globalWS, localWS, null, n
ull);
26.  outBuffer = new Uint32Array(vectorLength);
27.  cmdQueue.enqueueReadBuffer (bufOut,
                  true, 0, bufSize, outBuffer, null, nul
l);
28.  cmdQueue.finish();
29.  bufIn1.release();
30.  bufIn2.release();
31.  bufOut.release();
32.  program.release();
33.  kernel.release();
34.  cmdQueue.release();
35.  ctx.release(); }
36. </script>
</body>
</html>
```

Kernel script

```
1. <script id="clProgramVectorMul" type="text/x-opencl">
2. kernel void ckVectorMul(global uint* vectorIn1,
                        global uint* vectorIn2,
                        global uint* vectorOut,
                        uint uiVectorWidth) {
3. uint x = get_global_id(0);
4. uint y = get_global_id(1);
5. uint acc = 0;
6. for(int k = 0; k < uiVectorWidth; k++){
7.    acc += vectorIn1[y*uiVectorWidth+k]
          * vectorIn2[k*uiVectorWidth+x];

8. }
9. vectorOut[y*uiVectorWidth+x] = acc; }
10. </script>
```

Table 2. Modified WebCL Scripts of Matrices Multiplication

Changed Javascript

```
<html>
powerConsumptionChartDrawing();
temperatureChartDrawing();
<body>
1. <script type="text/javascript">
...
12.   var cmdQueue = ctx.createCommandQueue (device,
WebCL.QUEUE_PROFILING_ENABLE );
...
     var writeEvt1 = new WebCLEvent();
     var writeEvt2 = new WebCLEvent();
21.  cmdQueue.enqueueWriteBuffer (bufIn1,
          false, 0, bufSize, UIvector1, null, writeEvt1);
22.  cmdQueue.enqueueWriteBuffer (bufIn2,
          false, 0, bufSize, UIvector2, null, writeEvt2);
...
     var kerEvt = new WebCLEvent();
25.  cmdQueue.enqueueNDRangeKernel(kernel, globalWS.le
ngth, null, globalWS, localWS, null, kerEvt);
...
   var readEvt = new WebCLEvent();
27.  cmdQueue.enqueueReadBuffer (bufOut, true, 0, bufSiz
e, outBuffer, null, readEvt);
...
36. profilingResultPrint();
37. </script>
</body>
</html>
```

악할 수 있다. 해당 그림의 write time은 Table 2의 21, 22번째 줄을 보면 삽입된 WebCL 이벤트 객체로 호스트에서 디바이스로 데이터가 전송된 시간이 측정된 결과를 보여준다. 본 실험에서는 각각 약 11ms가 소요된 것을 확인할 수 있다. ckVectorMul은 디바이스에서 수행된 커널 함수명을 보여주며 Table 2의 25번째 줄에 삽입된 이벤트 객체로부터 측정되어 Table 1의 커널 스크립트 연산이 751.2ms가 소요되었음을 보여준다. read time은 디바이스에서 수행된 결과 데이터가 호스트로 전송된 시간을 보여주며 Table 2의 27번째 줄에 삽입된 이벤트 객체로부터 약 14.9ms가 소요되었음을 알 수 있다. 위의 프로파일링 결과를 출력해주는 스크립트는 마지막 줄 Table 2의 36번째 줄에 추가된다. Fig. 5(c)는 실시간으로 업데이트 되고 있는 GPU의 상태 테이블로부터 상세 정보를 받아오는 것을 확인할 수 있다.

Fig. 6은 WebCL 스크립트의 전력 누수와 디바이스의 온도 상승을 보여주기 위한 실험이다. 본 실험에서 Fig. 6(b)는 Table 1과 동일한 스크립트를 사용하였을 때의 전력 그래프를 보여주고, Fig. 6(a)는 Table 1의 자바스크립트 부분에서 29~35번째 줄의 메모리 해제 부분에서 하나라도 누락되었을 때의 전력 그래프를 보여준다. Fig. 4의 buffer, program, kernel, commandQueue, event 등 WebCL 요소들의 변수들이 제대로 메모리 해제되지 않는다면, Fig. 6(a)와 같이 디바이스가 다 사용된 후에도 전력 누수가 발생되는 것과 디

바이스의 온도가 상승하는 것을 알 수 있다. Fig. 6(b)에서는 위의 변수들이 제대로 메모리 해제가 되었을 때 디바이스가 사용되기 전의 전력으로 되돌아가는 것을 확인할 수 있다.

5. 결 론

본 논문에서는 WebCL 스크립트의 프로파일링을 위한 WebCL Profiler 프레임워크를 제안한다. WebCL Profiler의 주요 기능은 다음과 같다.

- 작성한 WebCL 스크립트가 사용하는 커널 함수 수행시간, 디바이스 메모리의 읽기/쓰기 시간 프로파일링
- NVIDIA Tesla Kepler 디바이스를 사용한다면 GPU 디바이스의 기본 정보, 전력 소비, 온도, 클락, 메모리 클락 정보의 모니터링
- 전력 소비, 온도 그래프를 통하여 디바이스가 다 사용된 후에도 전력 누수, 디바이스의 온도가 상승되는 현상의 파악

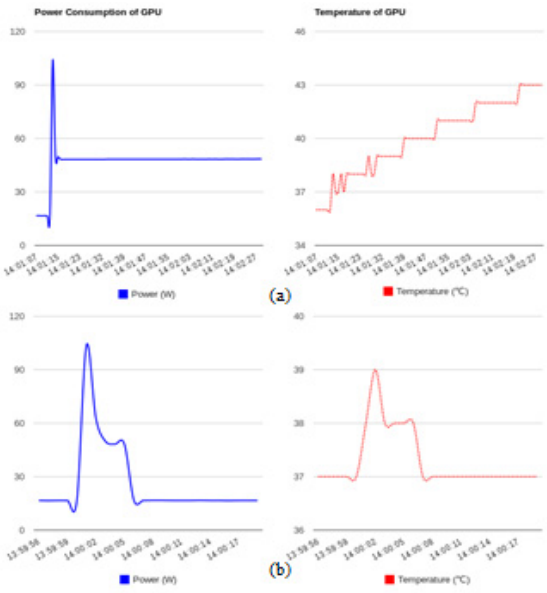

Fig. 6. (a) Power Leakage (b) No Power Leakage

WebCL 개발자는 본 프로파일러를 통하여 편리하게 웹 애플리케이션의 성능을 프로파일링 할 수 있고, 클라이언트에게 보다 안정적인 서비스를 제공할 수 있을 것으로 기대한다.

References

[1] Khronos, “WebCL: Heterogeneous parallel computing in HTML web browsers,” Khronos Group, 2012, [Internet], https://www.khronos.org-/webcl/.

[2] NVIDIA, “NVML API Reference Guide,” NVIDIA Corporation, 2014, [Internet], https://developer.nvidia.com/nvidia-management-library-nvml/.

[3] S. L. Graham, P. B. Kessler, and M. K. Mckusick. “Gprof: a Call Graph Execution Profiler,” *SIGPLAN Not.*, Vol.17, No.6, pp.120-126, Jun., 1982.

[4] AMD App Profiler [Internet], http://developer.amd.com/tools-and-sdks/archive/amd-app-profiler/.

[5] NVIDIA Visual Profiler [Internet], http://docs.nvidia.com/cuda/pdf/CUDA_Profiler_Users_Guide.pdf.

[6] Perhaad Mistry, Yash Ukidave, Dana Schaa, and David Kaeli, “A Framework for Profiling and Performance Monitoring of Heterogeneous Applications,” *MULTIPROG*, 2013.

[7] MSDN profiler [Internet], https://msdn.microsoft.com/en-us/library/ie/dn255005(v=vs.85).aspx.

[8] Chrome profiler [Internet], https://developer.chrome.com/devtools/docs/cpu-profiling.

[9] MDN profiler [Internet], https://developer.mozilla.org/en-US/docs/Tools/Profiler.

[10] Firebug profiler [Internet], http://getfirebug.com/faq/.

[11] spy-js profiler [Internet], http://spy-js.com/.

[12] WebCL profiler [Internet], http://esrc.korea.ac.kr.

김 철 원

e-mail : yohan3704@korea.ac.kr

2014년 고려대학교 컴퓨터정보학과(학사)

2014년~현 재 고려대학교 컴퓨터정보학과 석사과정

관심분야 : Heterogeneous Parallel Computing, Multiple Objects Tracking, Human-computer Interactions

조 현 중

e-mail : raycho@korea.ac.kr

1996년 경북대학교 전자공학부(학사)

1998년 포항공과대학교 전자전기공학과(석사)

2006년 9월 Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg VA(Ph.D)

2009년~현 재 고려대학교 컴퓨터정보학과 부교수

관심분야 : Real-time Computing for Embedded Systems, Human-computer Interactions for Smart Devices