Just-in-time Javascript Acceleration Benchmarking

Aparna Tiwari
School of Informatics and Computing
Indiana University
Bloomington, In
tiwaria@cs.indiana.edu

Saurabh Malviya
School of Informatics and Computing
Indiana University
Bloomington, IN
malviyas@cs.indiana.edu

ABSTRACT
Browser has always been users' window to the digital world and Javascript is the key to making the user experience better. There has been substantial improvement in the hardware performance over last few years. But unfortunately Javascript can't make use of the parallel client hardware and still has limited resource available which seriously hinders the browser's capability to handle computation intensive applications like games.

In this technical report, we do the benchmarking of the proposed solution to increase the Javascript performance by offloading part of the Javascript that is computation intensive and can't be handled by browser to the underlying GPU.

Categories and Subject Descriptors
D.3.3 [**Programming Languages**]: Processors

General Terms
Performance

Keywords
Javascript

1. INTRODUCTION
Javascript performance has long been the concern among programmer community and many solutions have been proposed by the research community. Intel's River Trail project[1] and adoption of HTML5 by many modern browsers are few of the efforts in that direction. Our solution to tackle this problem is to make use of the clients underlying hardware i.e. GPU from within the Javascript. Unlike other solutions proposed so far *Just-in-time Javascript Acceleration* doesn't force programmer to learn new language or to be familiar with nuances of GPU programming but still allows parallelizing the computation intensive chunk.

The idea behind our approach is to segregate the part that can be parallelized and write a function which will contain that part. This function will be treated as the device kernel of the cuda program. This function will be *stringified* and passed as an argument to setupKernel() function which will do the magic. In this report, we will be comparing the execution time of this parallelized function with equivalent cuda's kernel execution and sequential javascript execution.

2. ARCHITECTURE
As per the progress made so far this feature is available to Javascript programmers via browser extension. Extension's shared object is exposed to the other html or Javascript files. Using this shared object the function which needs to be parallelized will be parsed and first converted to LLVM IR code and then to corresponding PTX. The only diversion from conventional Javascript programming is use of CTYPE arrays to store and pass data to the shared object. This will be just-in-time compilation of Javascript code to PTX which will be executed from the host cuda code that will be already present as part of the extension.

3. BENCHMARKING
3.1 N-Body simulation
The algorithm used for an N-Body simulation models the motion of N particles (or objects) subject to a particle-particle interaction. Each particle interacts with all other particles in the simulation, thus the computation takes $O(N^2)$ time. The interaction considered will be the gravitational force between the particles so that the simulation might represent the motion of stars in a galaxy. (The same algorithm can be used to model very different physics, for example, the motion of charged particles on a much smaller length scale.)[2].

The basic algorithm has two main steps. First, the total force on each particle resulting from the gravitational attraction to all other particles is calculated. Then each particles position and velocity is updated as result of this force using a simple integrator over some small timestep. Repeating this process results in a simulation of the motion of all of the particles (stars) within the system (galaxy).

For our purpose, we will be iterating only once over N particles to find their new position and velocity after one time step.

3.2 Algorithm
The pseudocode for the N-Body algorithm which we used to compare the performance of our *Just-in-time Javascript Acceleration* over the sequential javascript:

We assume that we have N bodies each with mass m_i at position (x_i, y_i) and moving with a velocity (vx_i, vy_i). The force acting on the particles is gravitational i.e. force between bodies i and j is $F(i,j) = G\ m_i\ m_j/r_{i,j}^3\ (x_i-x_j, y_i-y_j)$ where $r_{i,j} = sqrt(\ (x_i-x_j)^2+(y_i-y_j)^2)$. The objective is to simulate the movement of the bodies over a series of time with steps dt[3]. Assuming v(i) be the velocity of particle is at any time step. v(i) = ((vx(i), vy(i))

```
for i = 0:N-1:
    vx_new(i) = vy_new(i) = 0
    x_new(i)=y_new(i) = 0
    for j=0:N-1
        if(j != i)
          (x(i),y(i)) = (x(i),y(i)) + v(i)*dt
          (vx_new(i), vy_new(i)) = (vx(i),vy(i)) + F(i,j)/m(i) * dt

        endif
       (vx(i),vy(i)) = (vx_new(i),vy_new(i))
    endfor
endfor
```

Final project for CS B649, 23rd December 2012

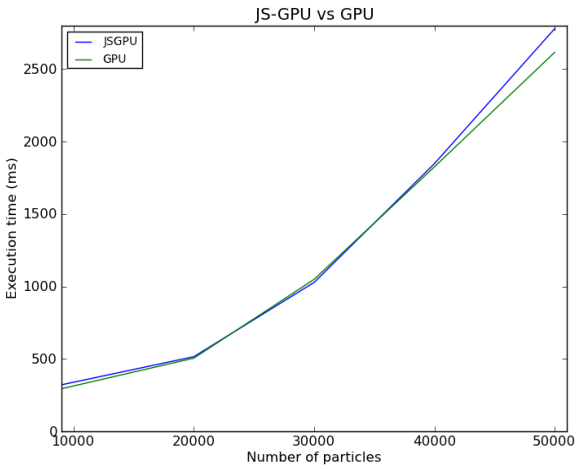

Figure 1: Comparison graph of execution time for N-Body algorithm. JS-GPU ~ *Just-in-time Javascript Acceleration*.

Updating the new position and velocity of each particle:

```
for i=0:N-1:
    (x(i), y(i)) = (x_new(i), y_new(i))
    (vx(i),vy(i)) = (vx_new(i),vy_new(i))
endfor
```

Each particle's distance with respect to other particles in the space is computed which is used to calculate the cumulative force acting upon that particle. Since, each particle's computation is independent of other particles it is suitable to be parallelized. For our performance comparison this is the chunk which we will be *stringifying* and offloading to GPU using *Just-in-time Javascript Acceleration.*

3.3 Comparison Results

Figure 1 shows the comparison graph of execution time of N-Body algorithm for different particle sizes on *Just-in-time Javascript Acceleration* which is the hybrid system of JS-GPU and on GPU. This comparison graph doesn't include the execution time plot for sequential javascript as browser is incapable of handling complexity of $O(N^2)$ for N>1000. The execution time of N-Body on GPU can be considered as the best performance that can be achieved. We can see that the execution time of N-Body over JS-GPU is very close to that of GPU.

4. Conclusion

Our results indicate that Javascript's performance can be significantly improved by offloading the computation intensive part of any algorithm to the GPU by making use of *Just-in-time Javascript Acceleration* technique proposed. In fact the performance is very much comparable to what can be achieved when the computation is performed over GPU. This outcome not only establishes *Just-in-time Javascript Acceleration* as one of the easy and effective way of improving the performance of Javascript but also serves as the purpose for Javascript programmers to think other areas or applications in which its features can be exploited.

5. REFERENCES

[1] Intel's River Trail Project https://github.com/rivertrail/rivertrail/wiki.

[2] OpenCl N-Body tutorial http://www.browndeertechnology.com/docs/BDT_OpenCL_Tutorial_NBody-rev3.html

[3] Parallel Computing Laboratory http://parlab.eecs.berkeley.edu/wiki/patterns/n-body_methods