Multithreads em JavaScript e GPU.js

Adriano Luís de Almeida
Programação Paralela - elc139-2019a

Menino JavaScript

Conhecido como “Aquele que dá vida” aos seus companheiros HTML e CSS.

Mas....

JavaScript é um moleque sapeca e logo que começou a ganhar novos recursos, resolveu “explorar novos horizontes”...

HTML

JAVASCRIPT

CSS

A Grande Dúvida do Menino JavaScript

“Como executar threads simultâneas e acabar com as janelas travadas esperando resposta do servidor ??”

Novo companheiro - Web Worker

Principal objetivo: é executar, em uma thread separada, scripts com tarefas que serão custosas, demoradas, como por exemplo cálculos complicados e pesados.

Os workers trabalham com trocas de mensagens para terem paralelismo, permitindo que a aplicação continue rodando na thread principal de forma fluída sem travar ou quebrar a experiência do usuários.

Sem usar web worker

```
var i = 0;

function timedCount() {
    i = i + 1;
    setTimeout(timedCount(),500);
}

timedCount();
```

Código
worker.js

```
Uncaught RangeError: Maximum call stack size exceeded
    at timedCount (teste.js:3)
    at timedCount (teste.js:5)
    at timedCount (teste.js:5)
    at timedCount (teste.js:5)
    at timedCount (teste.js:5)
    at timedCount (teste.js:5)
    at timedCount (teste.js:5)
    at timedCount (teste.js:5)
    at timedCount (teste.js:5)
    at timedCount (teste.js:5)
> |
```

Resultado da execução

Exemplo - Com a utilização do web worker

```
function startWorker(){
    var worker = new Worker('worker.js');

    worker.onmessage = (e)=>{
        console.log(e.data);
    }
}
```

Novo Código que faz o load do arquivo anterior

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

Resultado

Resultados obtidos

Observações

Chrome não executa de forma nativa por questões de segurança. Foi necessário a utilização de um servidor embutido, nesse caso foi utilizado a lib local-web-server obtido através do npm.

Resposta da chamada, em algum momento o browser vai encerrar a criação de threads, no exemplo acima o **chrome** permitiu a criação de **6345 threads** e o **firefox** permitiu a criação de **28038 threads**.

Exemplo 2 - Troca de mensagem entre main e worker

```
1  var worker = new Worker('worker.js');
2
3  worker.addEventListener('message', function(e) {
4    console.log(e.data);
5  });
6
7  worker.postMessage('Happy Birthday');
```

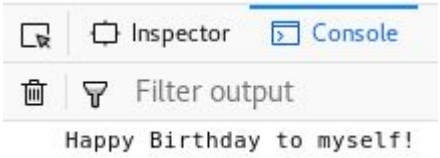
main.js

```
1  self.addEventListener('message', function(e) {
2      var message = e.data + ' to myself!';
3      self.postMessage(message);
4  });
```

worker.js

Passo a Passo do script

- O aplicativo criou um worker no main.js, que executa o código de worker.js
- Envia ao trabalhador uma mensagem com a string 'Happy Birthday'
- O trabalhador, que tinha um ouvinte de evento para 'mensagem', recebeu a mensagem e executou o código.
- O trabalhador acrescentou "a mim mesmo!" para os dados da mensagem criando "Feliz Aniversário para mim mesmo!" e envia isso como dados dentro de uma mensagem de volta para main.js
- Main.js, que também tinha um ouvinte de evento para a mensagem, do que console.log 'Happy Birthday to myself!'.



Resultado

Novo companheiro - GPU.JS

O gpu.js é uma biblioteca **GPGPU** (Programação para fins gerais em unidades de processamento gráfico) que permite entregar cálculos pesados à GPU para operação e saídas super rápidas.

Exemplo - Multiplicação entre matrizes 512x512

O método **createKernel**, cria um “Kernel” (um termo abstrato para o que poderia ser, na verdade *function*) que pode chamar de JS. “Por trás dos panos”, o código é compilado para os shaders **GLSL** usando um analisador AST e baseado em **jison**. Isso garante que o código escrito dentro do kernel será executado em uma GPU.

Importantes

shaders é um conjunto de instruções que definem o comportamento da superfície dos objetos. Basicamente são utilizados para aplicar efeitos como reflexos do ambiente na lataria de um carro em um game ou a movimentação da água enquanto um personagem está nadando.

Outras “nomenclaturas”

GLSL é uma linguagem de shading shaders de alto nível baseada na linguagem C. Foi criada com o objetivo de dar mais controle do pipeline de gráficos sem precisar usar assembly ou outras linguagens específicas de hardware (baixo nível).

AST(*Abstract Syntax Tree*). Podemos pensar no AST como um mapa para o código — é uma forma de entender como um pedaço de código é estruturado.

jison Uma API parser para criar analisadores em JavaScript. O Script gerado é usada para analisar entradas, aceitar, rejeitar ou executar ações com base na entrada.

Código sem utilização do GPU.JS

```
function cpuMatMult(m, n) {
    var result = [];
    for (var i = 0; i < m.length; i++) {
        result[i] = [];
        for (var j = 0; j < n[0].length; j++) {
            var sum = 0;
            for (var k = 0; k < m[0].length; k++) {
                sum += m[i][k] * n[k][j];
            }
            result[i][j] = sum;
        }
    }
    return result;
}
```

Código com utilização do GPU.JS

```
const gpuMatMult = gpu.createKernel(function (A, B) {
        var sum = 0;
        for (var i = 0; i < 512; i++) {
            sum += A[this.thread.y][i] * B[i][this.thread.x];
        }
        return sum;
    })
    .setDimensions([SIZE_A, SIZE_B])
    .setOutputToTexture(true);
```

Métodos utilizados do GPU.JS

Usando o método .**setDimensions** é possível ter acesso as dimensões da thread, podemos imaginar como comprimentos de um for quando usado na CPU por exemplo.

Um problema inerente é a questão da transferência dos comandos enviados da CPU para a caixa preta GPU, essa questão acaba se tornando um grande gargalo, principalmente quando envolve a realização de diversas operações matemáticas na GPU. No entanto no GPU.js podemos reverter esse problema deixando valores na GPU. Ao definir o **outputToTexture** como True, podemos garantir que não teremos penalidades de transferência.

Resultados

Resultados do teste gpu.js browsers chrome e firefox USANDO **.outputToTexture(true)**

```
Matriz A tam. 512  Matriz B tam. 512
CPU: 1728ms
GPU: 75ms
 22.04 vezes mais rápido!!!
```

Firefox

```
Matriz A tam. 512  Matriz B tam. 512
CPU: 641.1000000007334ms
GPU: 73.90000000305008ms
 7.675236806147135 vezes mais rápido!!!
>
```

Chrome

```
Matriz A tam. 512  Matriz B tam. 512
CPU: 688.0999999993946ms
GPU: 273.7000000051921ms
 1.5140664961137789 vezes mais rápido!!!
```

Firefox

```
Matriz A tam. 512  Matriz B tam. 512
CPU: 1908ms
GPU: 267ms
 6.146067415730337 vezes mais rápido!!!
```

Chrome

Fontes

- Web Worker: O jeito JS de fazer multithread. https://bit.ly/2JwUvp9
- Shaders: O que são e para que servem? Felipe Demartini https://bit.ly/2JaAXIv
- GLSL Language Specification, Version 4.10.6. Khronos Group. Acessado em 3 de julho de 2019. https://bit.ly/2XkfLnf
- An API for creating parsers in JavaScript https://zaa.ch/jison/
- Introducing gpu.js: GPU Accelerated JavaScript. Acessado em 3 de Julho de 2019. https://bit.ly/2MAkGeO

Perguntas?

Adriano Almeida alalmeida@inf.ufsm.br

https://github.com/adrianoluisalmeida/elc139-2019a

HTML

JAVASCRIPT

CSS

WEB WORKER

GPU.JS