УДК 681.518

РОЗПАРАЛЕЛЮВАННЯ ОБЧИСЛЕНЬ В СУЧАСНИХ ВЕБ-БРАУЗЕРАХ

Щербакова М. Є., Щербаков Є. В.

PARALLELIZING COMPUTING IN MODERN WEB BROWSERS

Shcherbakova M.E., Shcherbakov E.V.

Представлені можливості синхронних та асинхронних обчислень в веб-програмуванні з використанням мови JavaScript. Показано, що використання технології синхронних (векторних) обчислень SIMD для JavaScript може в декілька разів пришвидшити обробку великих масивів даних при наявності в комп'ютерному пристрої векторного розширювача. Додатковий виграш в часі рішення великих задач можна отримати на багатоядерних процесорах за рахунок асинхронного розпаралелювання задачі на багато потоків, використовуючи технологію веб-виконавців.

Ключові слова: *JavaScript, синхронні обчислення, SIMD API, асинхронні обчислення, веб-виконавці*

Вступ. Мова програмування JavaScript стала доступною майже всюди, в мережі Інтернет - вона використовується на серверах, вбудована в додатки за технологією веб-сторінок і, звичайно, працює в якості середовища виконання в веб-браузерах, які активно використовуються, як в мобільних пристроях, так і в настільних комп'ютерах.

JavaScript є стандартною мовою програмування веб-платформ, і все частіше традиційне програмне забезпечення стає веб-програмним забезпеченням. Це стосується мобільних додатків, ігор, програмного забезпечення вбудованих комп'ютерів, пристроїв IoT, хмарних сервісів. Все це програмне забезпечення потребує використання найбільш сучасних комп'ютерних технологій, які потребують багато обчислень: воно включає моделювання фізичних та інших процесів, воно користується алгоритмами штучного інтелекту, в ньому часто проводиться цифрова обробка сигналів, забезпечується візуалізація динамічних даних, а також виконуються багато інших функцій обробки інформації в режимі реального виміру часу.

Розробники програмного забезпечення все більше покладаються на JavaScript і все в більшій мірі потребують передбачуваної високої продуктивності виконання програм, написаних на цій мові, наприклад, коли для гри потрібні гарантовані 60 кадрів в секунду. Це потребує ефективного використання в програмах всіх засобів пришвидшення обчислень, в першу чергу таких, як синхронні обчислення з використанням розширювачів процесорів типу SIMD, а також асинхронні обчислення на сучасних багатоядерних процесорах.

Синхронні обчислення. При синхронних (або векторних) обчисленнях один потік інструкцій обробляє одночасно багато потоків даних (Single Instruction Multiple Data - SIMD), тобто в одній інструкції в якості кожного операнду використовується не одне скалярне значення, а відразу масив (вектор) значень. Нехай, наприклад, X1, X2 і Y - це три одновимірних масиви, що мають однакове число елементів, і є інструкція: Y = X1 + X2.

Векторний розширювач по одній інструкції процесора виконає попарні складання елементів масивів X1 і X2 і присвоїть отримані значення відповідним елементам масиву Y. На противагу цьому, звичайному послідовному процесору довелося багато разів виконувати операцію додавання елементів двох масивів, а також кожного разу виконувати кілька додаткових інструкцій, які організовують циклічну обробку масивів. Очевидно, що при вирішенні великих задач за рахунок паралельної векторної реалізації її алгоритму можна досягти більш високої швидкості обчислень.

В JavaScript інтерфейс прикладного програмування SIMD API [1] складається з декількох нових типів і операцій. Браузери забезпечують високо оптимізовану реалізацію цього API в залежності від базового апаратного забезпечення. В даний час реалізація SIMD API існує для платформи ARMv7 з розширенням NEON та платформи x86 з розширенням SSE.

На рис. 1 представлена ієрархія основних типів об'єктів модуля SIMD для мови програмування JavaScript.

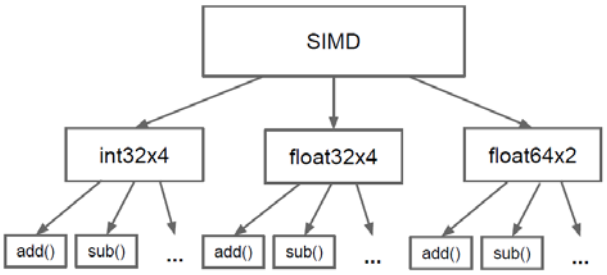

Рис. 1. Ієрархія об’єктів модуля SIMD для JavaScript

В цілому, реалізація SIMD API включає в себе наступні типи об’єктів: 2 - для роботи з числами з плаваючою точкою (float32x4, float64x2), 3 - для роботи з цілими числами без знаку (uint8x16, uint16x8, uint32x4), 3 - для роботи з цілими числами зі знаком (int8x16, int16x8, int32x4), 4 - для роботи з логічними значеннями (bool8x16, bool16x8, bool32x4, bool64x2). В назвах типів перше число позначає розрядність оброблюваних відповідним об’єктом величин, а друге число – кількість одночасно оброблюваних величин або, в термінології SIMD-технології, кількість одночасно оброблюваних смуг (lanes) в регістрах векторного розширювача процесора. Як можна переконатися, добуток першого і другого числа в імені типу кожного об'єкта завжди дорівнює 128, тобто, розрядності регістрів векторного розширювача.

Кожний з об’єктів має функціонально повний набір методів для створення та виконання обчислень над векторами значень відповідного типу та розмірності. Нижче наводиться приклад додавання двох векторів:

```
let a = SIMD.Float32x4(1, 2, 3, 4);
let b = SIMD.Float32x4(5, 6, 7, 8);
let c = SIMD.Float32x4.add(a,b);
// Float32x4[6,8,10,12]
```

Для поелементного розгалуження векторної обробки в модулі SIMD використовується технологія, подібна методології, запропонованій в роботі [2]. В якості прикладу векторного розгалуження нижче приведена функція отримання вектору max, елементи якого містять максимуми елементів вхідних векторів a та b:

```
max = function(a, b) {
let greaterThan = SIMD.Float32x4.greaterThan(a, b);
return SIMD.float32x4.select(greaterThan, a, b);
}
```

В якості демонстрації на рис. 2 приведені значення елементів векторів greaterThan та max при роботі функції після її виклику на виконання з вхідними векторами a та b, значення яких позначені різними кольорами.

Як можна побачити з цього прикладу, для проведення обчислень з 32-розрядними числами використовуються 4 смуги (lanes), тобто прискорення векторних обчислень в порівнянні зі скалярними обчисленнями при роботі з 32-розрядними числами не може бути більше 4-х.

Вектор a	1.0	2.0	3.0	4.0
Вектор b	0.0	3.0	5.0	2.0
Вектор маски greaterThan	true	false	false	true
Вектор max	1.0	3.0	5.0	4.0

Рис. 2. Значення елементів векторів a, b, greaterThan та max при виконанні векторної функції

Технологія синхронних обчислень SIMD для JavaScript є експериментальною і реалізовувалась в рамках бібліотеки SIMD.js. Передбачалось, що опис SIMD API ввійде в специфікацію ECMAScript 2016, але SIMD була виключена з цієї специфікації, оскільки було прийнято рішення розвивати цю технологію в рамках проекту WebAssembly – низькорівневої мови веб-програмування та віртуальної машини для неї, який підтримується всіма основними виробниками веб-браузерів.

Асинхронні обчислення. Доповнення до мови розмітки HTML 5 «Web Workers» [3] ввело в мову програмування JavaScript механізм веб-виконавців, за допомогою яких можна організувати ефективну паралельну обробку даних на всіх ядрах процесора так, щоб вона не впливала на час реакції візуального інтерфейсу користувача, який зазвичай реалізується браузером. Окрім того, специфікація ECMAScript 2015 [4] додала в мову JavaScript підтримку популярних шаблонів програмування. Одним з таких шаблонів є об'єкт Promise, який полегшує розробку легкого для сприйняття асинхронного коду.

Використання сучасних асинхронних програмних інтерфейсів мов HTML5 та JavaScript сприяють написанню більш швидких, надійних, коротких та зрозумілих програм веб-додатків та веб-сайтів.

Веб-виконавці. Веб-виконавці представляють собою ефективно працюючі потоки виконання. Веб-виконавці живуть в автономному середовищі виконання, проте, без доступу до об'єктів Window або Document, і можуть спілкуватися з головним потоком тільки через механізм асинхронної передачі повідомлень. Це означає, що паралельна модифікація об'єктної моделі документа (DOM) все ще не можлива, але в той же час веб-виконавці дають можливість використовувати синхронні API і писати занадто довгі по часу виконання функції, які не гальмують цикл обробки подій і не підвішують браузер. Створення нового виконавця не є важкою операцією, такою, як відкриття нового вікна браузера. Але виконавці не найпростіші потоки з усіх, і тому не має сенсу створення нових виконавців для виконання тривіальних операцій. Для складних веб-додатків може виявитися корисним створювати десятки виконавців, але малоймовірно, що додатки з сотнями або тисячами виконавців будуть зустрічатися на практиці.

Як і в будь-якому потоковому API, є дві частини визначення веб-виконавця. Першою частиною є об'єкт Worker: він визначає, як

виконавець представляється ззовні, з боку потоку, який його створює. Другою частиною є WorkerGlobalScope - глобальний об'єкт для нового виконавця, який визначає, як потік виконавця виглядає з внутрішньої сторони, тобто з коду виконавця.

Об'єкт Worker. Щоб створити новий виконавець, треба використати конструктор Worker(), передавши йому URL файлу з кодом JavaScript, який виконавець повинен виконати:

```
var loader = new Worker ("utils / loader.js");
```

Об'єкту Worker можна посилати дані за допомогою методу postMessage(). Значення, яке передається postMessage(), клонується і його копія доставляється виконавцю через подію message:

```
loader.postMessage ("file.txt");
```

Повідомлення від виконавця можна отримувати, прослуховуючи подію message об'єкта Worker:

```
worker.onmessage = function(e) {
var message = e.data;
    // Отримання повідомлення
console.log("URL contents: " + message);
   // Робота з ним
}
```

Подібно до всіх приймачів подій, об'єкти Worker визначають стандартні методи addEventListener() та removeEventListener(), і їх можна використовувати замість властивостей onmessage та onerror, якщо потрібно управляти кількома обробниками подій.

Об'єкт Worker має ще один метод, метод terminate(), який забезпечує завершення виконання потоку виконавця.

Об'єкт WorkerGlobalScope. Коли створюється новий виконавець за допомогою конструктора Worker(), задається URL файлу з кодом JavaScript. Цей код виконується в новому незайманому оточенні виконання JavaScript, повністю ізольованому від сценарію, який створив виконавця. Глобальним об'єктом для цього нового оточення виконання є об'єкт WorkerGlobalScope. WorkerGlobalScope - це щось більше ніж глобальний об'єкт базового JavaScript, але менше, ніж повномасштабний об'єкт Window клієнтського JavaScript.

Об'єкт WorkerGlobalScope має метод postMessage() та властивість обробника подій onmessage, які такі ж, як і у об'єкта Worker, але працюють в протилежному напрямку: виклик postMessage() зсередини виконавця генерує подію message поза виконавцем, а повідомлення, що посилаються ззовні виконавця, перетворюються в події і доставляються обробнику onmessage. Потрібно зауважити, що оскільки WorkerGlobalScope є глобальним об'єктом для виконавця, postMessage() та onmessage для коду виконавця виглядають як глобальна функція і глобальна змінна.

Функція close() дає можливість виконавцю завершити самого себе і працює подібно методу terminate() об'єкта Worker. Однак, слід зауважити, що немає API для об'єкта Worker, який би забезпечував перевірку, чи закрив себе виконавець, і немає властивості обробника подій onclose. Якщо викликати postMessage() для виконавця, який був закритий, повідомлення буде тихо скинуто і ніякої помилки не буде згенеровано. Загалом, якщо виконавець має намір викликати close() для закриття самого себе, хорошою ідеєю може бути попередня посилка якогось «закриваючого» повідомлення.

Приклад використання. Найбільш поширене застосування веб-виконавців - це вирішення задач, які потребують тривалих обчислень без гальмування візуального інтерфейсу користувача.

Прикладом такої задачі може служити відображення на екрані фракталів на основі множини Мандельброта (рис. 3). Фрактал Мандельброта визначається як множина точок на комплексній площині, для яких не йде в безкінечність ітеративна послідовність:

$Z_0 = 0$

$Z_{n+1} = Z_n^2 + C$

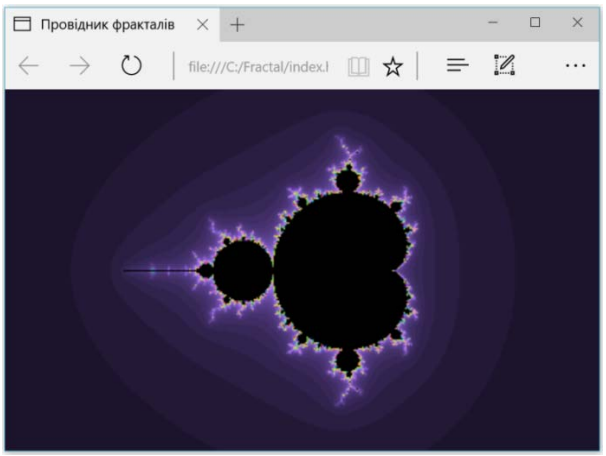

Рис. 3. Представлення в браузері початкового зображення множини Мандельброта

Якщо отримувати наведене на рисунку зображення послідовно за допомогою одного веб-виконавця, то на його відображення знадобиться близько 20 секунд. Оскільки ітерації для отримання зображення ведуться порядково, обчислення можна розподілити між групою паралельно працюючих веб-виконавців, ітеруючих сусідні рядки зображення. В результаті на комп'ютері з багатоядерним процесором час отримання зображення зменшується в кілька разів.

У програмі цього прикладу передбачено автоматичне масштабування зображення в залежності від розмірів вікна браузера. Крім того, після клацання мишею в якійсь точці зображення встановлюються нові межі множини Мандельброта з встановленим коефіцієнтом масштабування, рівним 8, і за допомогою тих же веб-виконавців обчислюється новий фрактал, що підтримує співвідношення сторін поточного розміру вікна. На рис. 4 представлений один з таких фракталів.

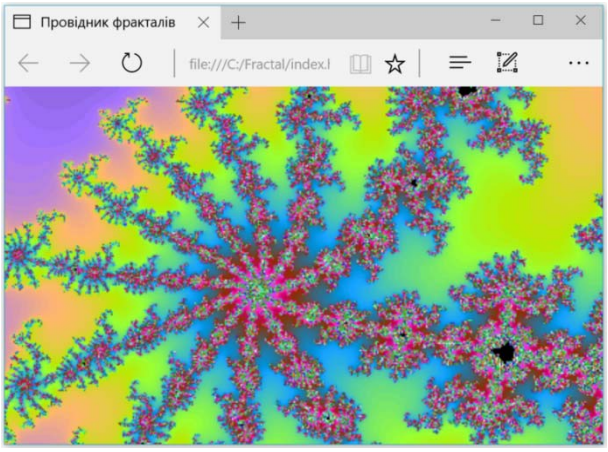

Рис. 4. Зображення одного з фракталів, отриманого масштабуванням однієї з прямокутних ділянок на кордоні множини Мандельброта

Висновок. Представлені засоби паралельних обчислень в веб-браузерах є дуже перспективними і при їх належній реалізації браузерами можуть істотно розширити сферу застосування веб-програмування. При наявності в комп'ютерному пристрої векторного розширювача процесора використання технології синхронних (векторних) обчислень SIMD для JavaScript може в декілька разів пришвидшити обробку великих масивів даних. Додатковий виграш в швидкості рішення великих задач, особливо їх ділянок, які не піддаються векторному розпаралелюванню, можна отримати на багатоядерних процесорах за рахунок асинхронного розпаралелювання задачі на багато потоків, використовуючи технологію веб-виконавців. В цілому, технології SIMD та веб-виконавців спільно з іншими сучасними засобами веб-розробки, такими як типізовані масиви JavaScript та WebAssembly, API для безпосереднього малювання (canvas) та інші просунуті можливості HTML5, можуть скласти серйозну конкуренцію таким технологіям, як SilverLight (Microsoft) і Adobe Flash, особливо, при програмуванні мобільних додатків, ігор, програмного забезпечення вбудованих комп'ютерів, пристроїв IoT.

Література

1. SIMD [Електронний ресурс]. - Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SIMD.
2. Щербакова М.Е. Балансирование загрузки конвейера процессора с помощью векторов ветвления / Щербакова М.Е., Щербаков Е. В., Рязанцев А.И. // Збірник наукових праць ДонНТУ. Серія «Інформатика, кібернетика та обчислювальна техніка», 2009. – С. 123-130
3. Workers [Електронний ресурс]. - Режим доступу: http://dev.w3.org/html5/workers/.
4. ECMAScript® 2015 Language Specification [Електронний ресурс]. - Режим доступу: http://www.ecma-international.org/ecma-262/6.0/index.html#sec-ecmascript-function-objects.

References

1. SIMD [Electronic resource]. - Access mode: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SIMD.
2. Shcherbakova M.E. Balancing of processor conveyor loading using branch vectors / Shcherbakova M.E., Shcherbakov E. V., Ryazantsev O.I. // Digest of scientific works of DonNTU. Series «Informatics, cybernetics and computer engineering», 2009. – P.123-130
3. Workers [Electronic resource]. - Access mode: http://dev.w3.org/html5/workers/.
4. ECMAScript® 2015 Language Specification [Electronic resource]. - Access mode: http://www.ecma-international.org/ecma-262/6.0/index.html#sec-ecmascript-function-objects.

Щербакова М.Е., Щербаков Е.В. Распараллеливание вычислений в современных веб-браузерах

Представлены возможности синхронных и асинхронных вычислений в веб-программировании с использованием языка JavaScript. Показано, что использование технологии синхронных (векторных) вычислений SIMD для JavaScript может в несколько раз ускорить обработку больших массивов данных при наличии в компьютерном устройстве векторного расширителя. Дополнительный выигрыш во времени решения больших задач можно получить на многоядерных процессорах за счет асинхронного распараллеливания задачи на много потоков, используя технологию веб-исполнителей.

Ключевые слова: *JavaScript, синхронные вычисления, SIMD API, асинхронные вычисления, веб-исполнители*

Shcherbakova M.E., Shcherbakov E.V. Parallelizing computing in modern web browsers

The possibilities of synchronous and asynchronous calculations in web programming using the JavaScript language are presented. It is shown that the use of synchronous (vector) SIMD technology for JavaScript can speed up processing of large data sets in a several times if there is a vector expander in the computer device. Additional time gain in solving of large tasks can be obtained on multi-core processors due to the asynchronous parallelization of the task to many threads using web workers technology.

Keywords: *JavaScript, synchronous calculations, SIMD API, asynchronous calculations, web workers*

Щербакова М. Є. – к.т.н., доцент, доцент кафедри комп'ютерної інженерії Східноукраїнського національного університету ім. В. Даля, e-mail: m.shcherbakova432@gmail.com
Щербаков Є.В. – к.т.н., доцент, доцент кафедри комп'ютерної інженерії Східноукраїнського національного університету ім. В. Даля, e-mail: gkvarc@gmail.com

Рецензент: д.т.н., професор ***Рязанцев О.І.***

Стаття подана 26.09.2017