ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΣΥΓΚΡΙΤΙΚΗ ΜΕΛΕΤΗ ΑΛΓΟΡΙΘΜΩΝ ΥΨΗΛΗΣ ΥΠΟΛΟΓΙΣΤΙΚΗΣ ΙΣΧΥΟΣ ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ WEB WORKERS, WEBASSEMBLY

Διπλωματική Εργασία

του

Στότογλου Αναστάσιου
Α.Μ.: 21054

ΣΥΓΚΡΙΤΙΚΉ ΜΕΛΕΤΗ ΑΛΓΟΡΙΘΜΩΝ ΥΨΗΛΗΣ ΥΠΟΛΟΓΙΣΤΙΚΗΣ ΙΣΧΥΟΣ ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ WEB WORKERS, WEBASSEMBLY

Στότογλου Αναστάσιος

Πτυχιούχος Μηχανικών Πληροφορικής, Τ.Ε πρώην ΤΕΙ Στερεάς Ελλάδας

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Κασκάλης Θεόδωρος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 21/9/2022

Κασκάλης Θεόδωρος	Μαργαρίτης Κωνσταντίνος	Σακελλαρίου Ηλίας
..…………………………	..…………………………	…………………………

Στότογλου Αναστάσιος
…………………………

Περίληψη

Η συγκεκριμένη έρευνα αφορά την πειραματική εφαρμογή και σύγκριση αλγορίθμων που απαιτούν υψηλή υπολογιστική ισχύ, σε μια νέα τεχνολογία του web, την WebAssembly. Καθώς αυξάνονται οι χρήστες του διαδικτύου και κατά συνέπεια η χρήση των περιηγητών του ιστού, ολοένα και περισσότερες απαιτητικές εφαρμογές εκτελούνται στα προγράμματα περιήγησης. Για να διεκπεραιωθούν αυτές οι εφαρμογές με μεγαλύτερη αποτελεσματικότητα και για να μεγαλώσει το οικοσύστημα του διαδικτύου απαιτούνται νέες τεχνολογικές υποδομές. Η WebAssembly μπορεί να δώσει την λύση στην επέκταση του οικοσυστήματος αλλά και στην βελτίωση των αποδόσεων, καθώς επιτρέπει την διεκπεραίωση εφαρμογών, υλοποιημένων σε γλώσσες χαμηλού επιπέδου (C, C++, Rust, κ.α.) μέσω εικονικής μηχανής του φυλλομετρητή. Αυτό βελτιώνει σε σημαντικό βαθμό το χρόνο εκτέλεσης μια εφαρμογής στο πρόγραμμα περιήγησης όπως και θεραπεύει μερικές από τις αδυναμίες που εμφανίζονται σε διαδικτυακές εφαρμογές υλοποιημένες από την γλώσσα JavaScript. Με τον συνδυασμό JavaScript και WebAssembly μπορούμε να έχουμε το δυνατότερο επιθυμητό αποτέλεσμα. Στη παρούσα εργασία συγκρίνουμε τις δύο γλώσσες προγραμματισμού που προαναφέρθηκαν ώστε να κατανοήσουμε σε ποιες περιπτώσεις αλγοριθμικών προβλημάτων υπερτερεί η μια από την άλλη καθώς και τα πλεονεκτήματα - μειονεκτήματα αυτών. Τα πειράματα της συγκριτικής μελέτης χωρίζονται σε τρεις κατηγορίες αλγορίθμων και εκτελούνται σε διαφορετικούς φυλλομετρητές, όπως και συσκευές. Μέσα από την παρούσα έρευνα θα αναδειχθεί ποιος φυλλομετρητής υπερτερεί και σε ποιες κατηγορίες προβλημάτων, ποια συσκευή παρουσιάζει πλεονεκτήματα και μειονεκτήματα, όπως και τις μεθόδους βελτιστοποίησης της WebAssembly ώστε να έχουμε καλύτερους χρόνους διεκπεραίωσης σε σχέση με την απλή JavaScript. Τα αποτελέσματα των συγκρίσεων παρουσιάζονται σε μορφή πίνακα και διαγραμμάτων.

Λέξεις Κλειδιά: WebAssembly, native Javascript, συγκριτική μελέτη, InstantiateStreaming, Native C, προγράμματα περιήγησης

Abstract

This research concerns the experimental application and comparison of algorithms that require high computing power, in a new Web technology, WebAssembly. As internet users and consequently the use of web browsers increase, more and more demanding applications are running on browsers. To handle these applications more efficiently and to grow the Internet ecosystem, new technological infrastructures are required. WebAssembly can provide the solution to the expansion of the ecosystem but also to the improvement of performance, as it allows the processing of applications, implemented in low-level languages (C, C++, Rust, etc.) through a virtual machine of the browser. This greatly improves the runtime of an application in the browser as well as cures some of the weaknesses that appear in web applications implemented by the JavaScript language. By combining JavaScript and WebAssembly we can get a desired result. In this paper we compare the two programming languages mentioned above in order to understand in which cases of algorithmic problems one is superior to the other as well as their advantages and disadvantages. The benchmarking experiments are divided into three categories of algorithms and run on different browsers as well as devices. Through this research to show which browser excels and in which categories of problems, which device presents advantages and disadvantages, as well as optimization methods of WebAssembly so that we have better processing times compared to simple JavaScript. The results of the comparisons are presented in table and diagram form.

Keywords: WebAssembly, native Javascript, comparative study, Native C, Browsers

Περιεχόμενα

1 Εισαγωγή 7
- 1.1 Πρόβλημα – Σημαντικότητα Θέματος 7
- 1.2 Σκοπός – Στόχοι 8
- 1.3 Συνεισφορά 8
- 1.4 Διάρθρωση της μελέτης 9

2 Θεωρητικό Υπόβαθρο 9
- 2.1 Η γλώσσα JavaScript 9
- 2.2 Εισαγωγή στους Μεταγλωττιστές και Διερμηνείς 10
  - 2.2.1 Διερμηνείς (Interpreter) 10
  - 2.2.2 Μεταγλωττιστές (Compilers) 11
  - 2.2.3 Φάσεις του μεταγλωττιστή 11
  - 2.2.4 Σύγκριση Μεταγλωττιστή και Διερμηνέα 12
- 2.3 Profile-based Optimization 13
- 2.4 Just-In-Time (JIT) 14
- 2.5 V8 Engine 15
- 2.6 Spidermonkey Engine 16
- 2.7 Native Client (NaCI) 16
- 2.8 asm.js 17

3 WebAssembly στην Πράξη 18
- 3.1 Το πρότυπο WebAssembly 18
- 3.2 Σημασιολογική Φάση
- 3.3 Ασφάλεια Προτύπου 20
- 3.4 JavaScript API και Web API 21
- 3.5 Βιβλιογραφική Επισκόπηση 24
- 3.6 Σύνοψη 25

4 Μεθοδολογία 27
- 4.1 Συγκριτική Αξιολόγηση και Συλλογή Αποτελεσμάτων 27
- 4.2 Περιβάλλον Εκτέλεσης Αλγορίθμων 28
  - 4.2.1 Προγράμματα και Εφαρμογές Περιβάλλοντος 29
  - 4.2.2 Υπολογιστικός Εξοπλισμός 29
- 4.3 Αλγόριθμοι Υψηλής Υπολογιστικής Ισχύος 30
  - 4.3.1 Αλγόριθμοι Αριθμητικών Υπολογισμών 30
    - 4.3.1.1 Ακολουθία Fibonacci 30
    - 4.3.1.2 Πολλαπλασιασμός Διπλής Ακρίβειας (DOUBLE) 31
    - 4.3.1.3 Πολλαπλασιασμός Ακεραίων Αριθμών (INT) 31
    - 4.3.1.4 Πολλαπλασιασμός Πινάκων 31
    - 4.3.1.5 Πρώτοι Αριθμοί 31
- 4.3.2 Αλγόριθμοι Ταξινόμησης 31

4.3.2.1 Αλγόριθμος Φυσαλίδας 32
4.3.2.2 Αλγόριθμος Ταχυταξινόμησης (Integers - Doubles) 32
4.3.2.3 Αλγόριθμος Αντιστροφής Πινάκων 32
4.3.3 Αλγόριθμοι Επεξεργασίας Εικόνας 32
4.3.3.1 Αλγόριθμος Ασπρόμαυρης Εικόνας (Grayscale) 33
4.3.3.2 Αλγόριθμος Συνέλιξης Εικόνας (Convolution) 33
4.3.3.3 Αλγόριθμος Προσαρμοστικού Κατωφλιού Εικόνας (Threshold) 33
4.4 Γλώσσες Προγραμματισμού Συγκριτικής Μελέτης 34
4.5 Αποτελέσματα 35
4.5.1 Εισαγωγή 35
4.5.2 Κατάσταση Cold - Warm - Hot 35
4.5.3 Αποτελέσματα Μετρήσεων σε Desktop 36
4.5.3.1 Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Desktop με φυλλομετρητή Chrome (hot) 36
4.5.3.2 Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Desktop με φυλλομετρητή Firefox (hot) 38
4.5.3.3 Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Desktop με φυλλομετρητή Opera (hot) 39
4.5.3.4 Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Desktop με φυλλομετρητή Chrome (hot) 41
4.5.3.5 Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Desktop με φυλλομετρητή Firefox (hot) 42
4.5.3.6 Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Desktop με φυλλομετρητή Opera (hot) 43
4.5.3.7 Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Desktop με φυλλομετρητή Chrome (hot) 44
4.5.3.8 Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Desktop με φυλλομετρητή Firefox (hot) 45
Εικόνα 4.8: Γράφημα συγκρίσεων χρόνων εκτέλεσης αλγορίθμων επεξεργασίας εικόνας σε Firefox 46
4.5.3.9 Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Desktop με φυλλομετρητή Opera (hot) 47
4.24.11 Εικόνα 4.9: Γράφημα συγκρίσεων χρόνων εκτέλεσης αλγορίθμων επεξεργασίας εικόνας σε Opera 47
4.5.4 Αποτελέσματα Μετρήσεων σε Laptop 48
4.5.5 Αποτελέσματα Μετρήσεων σε Mobile 49
5 Συζήτηση και Συμπεράσματα Αποτελεσμάτων **50**
5.1 JavaScript και WebAssembly στα Προγράμματα Περιήγησης 55
5.2 Συμπεράσματα και Πλεονεκτήματα - Μειονεκτήματα WebAssembly 56
5.3 Μελλοντική Επέκταση 57
Βιβλιογραφία 59
Παράρτημα Α.1 - Πίνακες Αποτελεσμάτων σε Laptop 63
Παράρτημα Α.2 - Διαγράμματα Αποτελεσμάτων σε Laptop 69

Παράρτημα Β.1 - Πίνακες και Διαγράμματα Mobile 74
Παράρτημα Β.2 - Διαγράμματα Αποτελεσμάτων σε Mobile 81

Κατάλογος Εικόνων

Εικόνα 2.1: Οι φάσεις του μεταγλωττιστή
Εικόνα 2.2: Μεταγλώττιση και εκτέλεση κώδικα σε μηχανή V8
Εικόνα 2.3: Μεταγλώττιση και εκτέλεση κώδικα σε μηχανή SpiderMonkey
Εικόνα 3.1: Πρόσβαση στην γραμμική μνήμη με εντολές και συναρτήσεις της WASM μέσω δεικτών.
Εικόνα 3.2: Τύποι μεταβλητών της wasm σε δεκαεξαδική μορφή και κειμένου
Εικόνα 3.3: Τυχαία διάταξη μνήμης
Εικόνα 3.4: Κώδικας γραμμένος σε C για την επιστροφή ακέραιου αριθμού
Εικόνα 3.5: Κώδικας γραμμένος σε WASM (σε μορφή WAT), για την επιστροφή ακέραιου αριθμού
Εικόνα 3.6: Κλήση της συνάρτησης InstantiateStreaming για την φόρτωση του module και την εκτύπωση της επιστρεφόμενης τιμής.
Εικόνα 3.7: Φόρτωση WASM μονάδας με την συνάρτηση Instantiate
Εικόνα 3.8: Διαχείριση μνήμης WebAssembly στην JavaScript
Εικόνα 3.9: Πίνακες της WebAssembly σε JavaScript
Εικόνα 3.10: Διαφορές και ομοιότητες μεταξύ τεχνολογιών
Εικόνα 4.1: JavaScript Performance API
Εικόνα 4.2: Καταγραφή χρόνου διεκπεραίωσης σε C

—--

Κατάλογος Πινάκων

Πίνακας 2.1: Πίνακας συγκρίσεων μεταγλωττιστή και διερμηνέα
Πίνακας 3.1: Ενσωμάτωση JavaScript σε wasm, οι πιο χρησιμοποιούμενες αναφορές
Πίνακας 4.1: Παρουσίαση διαφορών μεταξύ Hot και Cold κατάστασης μεταγλωττιστή JIT
Πίνακας 4.2: Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Desktop με φυλλομετρητή Chrome

Πίνακας 4.3: Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Desktop με φυλλομετρητή Firefox
Πίνακας 4.4: Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Desktop με φυλλομετρητή Opera
Πίνακας 4.5: Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Desktop με φυλλομετρητή Chrome
Πίνακας 4.6: Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Desktop με φυλλομετρητή Firefox
Πίνακας 4.7: Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Desktop με φυλλομετρητή Opera
Πίνακας 4.8: Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Desktop με φυλλομετρητή Chrome
Πίνακας 4.9: Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Desktop με φυλλομετρητή Firefox
Πίνακας 4.10: Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Desktop με φυλλομετρητή Opera
Πίνακας 4.11: Πίνακας αποτελεσμάτων με χρήση Laptop σε φυλλομετρητή Chrome
Πίνακας 4.12: Πίνακας αποτελεσμάτων με χρήση Mobile σε φυλλομετρητή Chrome
Πίνακας 5.1: Πίνακας διαφορών απόδοσης WASM και JavaScript στις περιπτώσεις βέλτιστης απόδοσης της εκάστοτε τεχνολογίας σε περιβάλλον Desktop
Πίνακας 5.2: Πίνακας αποτελεσμάτων γεωμετρικών μέσων στους τρεις φυλλομετρητές, για κάθε κατηγορία αλγορίθμων σε περιβάλλον Desktop
Πίνακας 5.3: Πίνακας διαφορών απόδοσης WASM και JavaScript στις περιπτώσεις βέλτιστης απόδοσης της εκάστοτε τεχνολογίας σε περιβάλλον Laptop
Πίνακας 5.4: Πίνακας αποτελεσμάτων γεωμετρικών μέσων στους τρεις φυλλομετρητές, για κάθε κατηγορία αλγορίθμων σε περιβάλλον Laptop
Πίνακας 5.5: Πίνακας διαφορών απόδοσης WASM και JavaScript στις περιπτώσεις βέλτιστης απόδοσης της εκάστοτε τεχνολογίας σε περιβάλλον Mobile
Πίνακας 5.6: Πίνακας αποτελεσμάτων γεωμετρικών μέσων στους τρεις φυλλομετρητές, για κάθε κατηγορία αλγορίθμων σε περιβάλλον Mobile
Πίνακας Α.1: Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Laptop με φυλλομετρητή Chrome
Πίνακας Α.2: Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Laptop με φυλλομετρητή Firefox
Πίνακας Α.3: Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Laptop με φυλλομετρητή Opera
Πίνακας Α.4: Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Laptop με φυλλομετρητή Chrome
Πίνακας Α.5: Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Laptop με φυλλομετρητή Firefox
Πίνακας Α.6: Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Laptop με φυλλομετρητή Opera
Πίνακας Α.7: Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Laptop με φυλλομετρητή Chrome

Πίνακας Α.8: Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Laptop με φυλλομετρητή Firefox
Πίνακας Α.9: Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Laptop με φυλλομετρητή Opera
Πίνακας Β.1: Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Mobile με φυλλομετρητή Chrome
Πίνακας Β.2: Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Mobile με φυλλομετρητή Firefox
Πίνακας Β.3: Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Mobile με φυλλομετρητή Opera
Πίνακας Β.4: Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Mobile με φυλλομετρητή Chrome
Πίνακας Β.5: Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Mobile με φυλλομετρητή Firefox
Πίνακας Β.6: Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Mobile με φυλλομετρητή Opera
Πίνακας Β.7: Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Mobile με φυλλομετρητή Chrome
Πίνακας Β.8: Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Mobile με φυλλομετρητή Firefox
Πίνακας Β.9: Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Mobile με φυλλομετρητή Opera

1 Εισαγωγή

1.1 Πρόβλημα – Σημαντικότητα Θέματος

Αν θεωρήσουμε ότι ζούμε στην εποχή της πληροφορικής, τότε αναμφισβήτητα ένα από τα σημαντικότερα κομμάτια της καθημερινότητας μας είναι ο παγκόσμιος ιστός και συγκεκριμένα η πλοήγηση στα προγράμματα περιήγησης του (browsers). Το γεγονός αυτό επιφέρει μεγάλο βάρος και σημαντικότητα στη χρήση της γλώσσας προγραμματισμού JavaScript, η οποία αποτελεί τη πρώτη επιλογή σε ότι σχετίζεται με τη δημιουργία διαδικτυακών εφαρμογών και ιστοσελίδων αλλά και τη διαχείριση αυτών, την αλληλεπίδραση με το χρήστη, το διακομιστή ιστού (web server), την αποθήκευση δεδομένων από την πλευρά του πελάτη (local storage) και διάφορες άλλες λειτουργίες. Τα τελευταία χρόνια η ωρίμανση του παγκόσμιου ιστού μας έχει οδηγήσει σε εξελιγμένες και απαιτητικές διαδικτυακές εφαρμογές, όπως η διαδραστική τρισδιάστατη απεικόνιση, το λογισμικό ήχου και βίντεο, τα παιχνίδια. Οι απαιτήσεις αυτές έχουν επιφέρει τη βελτίωση της γλώσσας JavaScript αλλά και των προγραμμάτων περιήγησης. Ωστόσο, η JavaScript ώς η μόνη ενσωματωμένη γλώσσα του ιστού δεν είναι καλά εξοπλισμένη ώστε να καλύψει τις παραπάνω απαιτήσεις.

Ένα άλλο σημαντικό πρόβλημα που εμφανίζεται με την αποκλειστικότητα της JavaScript στον παγκόσμιο ιστό, είναι η ανάγκη της επαναυλόποίησης ήδη υλοποιημένων εφαρμογών σε διαφορετικές γλώσσες προγραμματισμού (όπως, C, C++, Rust) ώστε να είναι συμβατές και εκτελέσιμες από τα προγράμματα περιήγησης. Για παράδειγμα αν υποθέσουμε ότι μια εφαρμογή έχει υλοποιηθεί σε γλώσσα C και αυτή η εφαρμογή είναι πολύ σημαντική για μια επιχείρηση ή έναν οργανισμό, τότε σε περίπτωση που θέλουμε να διεκπεραιώνεται σε ένα πρόγραμμα περιήγησης ώστε να είναι πιο εύκολα προσβάσιμο από τους χρήστες, θα πρέπει να την υλοποιήσουμε από την αρχή σε JavaScript χάνοντας χρόνο για την κατασκευή της αλλα και ταχύτητα εκτέλεσης από αυτή της αρχικής υλοποίησης. [44]

Αυτές οι ανάγκες και οι σύγχρονες απαιτήσεις του ιστού επέφεραν μια πρόκληση για τους μηχανικούς πληροφορικής των μεγαλύτερων προγραμμάτων περιήγησης, τη δημιουργία μιας συμβατής τεχνολογίας bytecode χαμηλού επιπέδου, την WebAssembly. Η WebAssembly προσφέρει μια αποτελεσματική λύση για κάποιες περιπτώσεις που ίσως μπορεί να αντικαταστήσει την γλώσσα JavaScript. Συγκεκριμένα, υπόσχεται καλύτερες επιδόσεις στην επίλυση ειδικευμένων υπολογισμών αλλά και μεγαλύτερη φορητότητα σε διάφορα περιβάλλοντα. Ακόμα μπορεί να ζωντανέψει εφαρμογές που έχουν υλοποιηθεί σε γλώσσα χαμηλού επιπέδου με το να τις εντάξει στον παγκόσμιο ιστό.

Το ερώτημα είναι ποιες εφαρμογές και υπολογιστικές πράξεις μπορεί να κάνει με μεγαλύτερη επιτυχία από την JavaScript. Είναι το ίδιο αποδοτική σε μικρότερες εφαρμογές από την άποψη μεγέθους κώδικα και πολυπλοκότητας; Είναι τα οφέλη περισσότερα από τον χρόνο που πρέπει να δαπανήσουμε για να μεταφερθούμε στην WebAssembly; Αυτά είναι κάποια από τα ερωτήματα που πρέπει απαντηθούν ώστε να γνωρίζουμε αν πρέπει και σε ποιες περιπτώσεις χρειάζεται να αντικαταστήσουμε την JavaScript, με το πρότυπο της WebAssembly.[44]

1.2 Σκοπός – Στόχοι

Η παρούσα μελέτη, στοχεύει αρχικά στην παρουσίαση του προτύπου WebAssembly αλλά και στην ανάδειξη των δυνατοτήτων της τεχνολογίας. Στη συνέχεια θα αναλυθούν τα επιμέρους τμήματα της και θα αναδειχθούν οι πιο πρόσφατες μέθοδοι μεταγλώττισης. Έμφαση θα δοθεί στην αναζήτηση απαιτητικών αλγορίθμων που χρήζουν υπολογιστική ισχύ, ώστε να πραγματοποιηθεί σύγκριση στην ταχύτητα εκτέλεσης ανάμεσα σε WebAssembly και native JavaScript, η σύγκριση θα γίνει στα τρία πιο γνωστά προγράμματα περιήγησης (Google Chrome, Mozilla Firefox, Opera) σε τρεις διαφορετικές συσκευές με διαφορετικούς επεξεργαστικούς πυρήνες (CPU). Οι υψηλής υπολογιστικής ισχύος αλγόριθμοι που θα εξεταστούν, θα είναι υλοποιημένοι σε γλώσσα C και JavaScript, και θα έχουν τον ίδιο βαθμό πολυπλοκότητας ενώ από τη μεριά της WebAssembly η υλοποίηση θα γίνει στο περιβάλλον της Emscripten αλλά και με hand coding με τη χρήση του API JavaScript WebAssembly.

1.3 Συνεισφορά

Η συνεισφορά της συγκεκριμένης μελέτης, θα λέγαμε ότι είναι η εξοικείωση των χρηστών με το πρότυπο WebAssembly, με σκοπό την ένταξη χρήσιμων εφαρμογών χαμηλού επιπέδου στο διαδίκτυο. Έπειτα, η βαθύτερη κατανόηση των δυνατοτήτων του προτύπου αλλά και των αδυναμιών του, με βάση των συγκρίσεων που θα πραγματοποιηθούν σε διαφορετικούς αλγορίθμους. Τέλος η ανάδειξη της υλοποίησης WebAssembly με hand coding και τρόποι βελτιστοποιήσεις του προτύπου ως προς τις ταχύτητες εκτέλεσης. Τα αποτελέσματα των συγκρίσεων και η επισκόπηση της βιβλιογραφίας θα αποτελούν βασικό βοήθημα για την επιλογή ανάμεσα σε native JavaScript και WebAssembly όταν κάποιος κληθεί να υλοποιήσει μια εφαρμογή.

1.4 Διάρθρωση της μελέτης

Η διάρθρωση της παρούσας μελέτης αρχίζει από το δεύτερο κεφάλαιο, όπου παρουσιάζεται το θεωρητικό υπόβαθρο για την κατανόηση των τεχνολογιών της μελέτης με σχετικές εργασίες και δημοσιεύσεις. Στη συνέχεια στο τρίτο κεφάλαιο θα εμβαθύνουμε στο πρότυπο της WebAssembly όπου θα αναλύσουμε τα τεχνικά χαρακτηριστικά της, τους τρόπους με τους όποιος μπορούμε να την αξιοποιήσουμε και τις τεχνολογίες γύρω από αυτή. Η βιβλιογραφική ανασκόπηση που θα πραγματοποιηθεί στο 2 και 3 κεφάλαιο έχει ως βασικό στόχο να παρέχει πληροφορίες σχετικά με τα χαρακτηριστικά της κάθε γλώσσας, τα πλεονεκτήματα και μειονεκτήματα των τεχνολογιών. Είναι απαραίτητο να ληφθούν πληροφορίες σχετικά με τη λειτουργικότητα των τεχνολογιών καθώς μπορεί να γίνουν πιο ξεκάθαρα και αντιληπτά τα αποτελέσματα την συγκριτικής μελέτης στον αναγνώστη. Στο τέταρτο κεφάλαιο θα παρουσιαστούν τα προγράμματα λογισμικού που χρησιμοποιήθηκαν καθώς και ο εξοπλισμός (υπολογιστές) όπου διεξήχθησαν τα πειράματα. Στη συνέχεια θα

εξηγηθούν οι τρεις κατηγορίες αλγορίθμων που επιλέχθηκαν αλλά και κάθε αλγόριθμος ξεχωριστά. Έπειτα θα αναλυθούν οι διάφοροι τρόποι βελτιστοποίησης των γλωσσών προγραμματισμού και οι συναρτήσεις με τις οποίες διεκπαιρεώθηκαν οι αλγόριθμοι. Τέλος θα παρουσιαστούν τα αποτελέσματα των πειραμάτων σε μορφή πίνακα και γραφημάτων. Το πέμπτο κεφάλαιο αποτελεί τον επίλογο της εργασίας, παρουσιάζονται και συζητούνται τα συμπεράσματα τα οποία προέκυψαν από τις συγκρίσεις του τέταρτου κεφαλαίου και έπειτα από την βιβλιογραφική επισκόπηση που πραγματοποιήθηκε. Θα παρουσιαστούν τα θετικά και αρνητικά της κάθε περίπτωσης και θα αναφερθούν κάποιες μελλοντικές ιδέες όπως και τρόποι αξιοποίησης του προτύπου WebAssembly.

2 Θεωρητικό Υπόβαθρο

2.1 Η γλώσσα JavaScript

Η γλώσσα προγραμματισμού JavaScript εντάχθηκε το 1994 - 1995 ως ένας νέος τρόπος δημιουργίας και ένταξης εφαρμογών στον παγκόσμιο ιστό και στα προγράμματα περιήγησης Netscape Navigator. Η JavaScript από την ημέρα δημιουργίας μέχρι σήμερα χρησιμοποιείται από όλα τα προγράμματα περιήγησης καθώς μέσω αυτής, έχουν υλοποιηθεί εφαρμογές διαφορετικού τύπου με μοντέρνο και φιλικό σχεδιασμό για τους χρήστες. Βασικό χαρακτηριστικό της είναι η αλληλεπίδραση που παρέχει στο χρήστη καθώς δε χρειάζεται να επαναφορτωθεί η ιστοσελίδα κατά την αίτηση νέας ενέργειας [27]. Η γλώσσα υποστηρίζει διάφορες μεθόδους - παραδείγματα (paradigms) προγραμματισμού όπως τον αντικειμενοστραφή, το δηλωτικό και το διαδικαστικό. Οι ιστοσελίδες που δημιουργούνται με JavaScript είναι δυναμικές. Λόγω της δυναμικής του φύσης χρησιμοποιεί τη μεταγλώττιση just-in-time JIT. Η Διαδικασία μεταγλώττισης κώδικα JavaScript σε κώδικα μηχανής για εκτέλεση ποικίλλει ανάλογα με το πρόγραμμα περιήγησης που εκτελείται ο κώδικας, με τη μηχανή SpiderMonkey να εκτελείται στον Firefox, την V8 να εκτελείται στον Chrome και την Carakan στον Opera. Ωστόσο, υπάρχει μια γενική προσέγγιση που εφαρμόζουν τα μεγάλα προγράμματα περιήγησης για να επιταχύνουν την απόδοση τους. Οι μηχανές της JavaScript υιοθετούν μια πολύ περίπλοκη αρχιτεκτονική πολλαπλών επιπέδων - φάσεων τα οποία εξηγούνται παρακάτω. Η αρχιτεκτονική αυτή συντίθεται από πολλά επίπεδα εκτέλεσης, συμπεριλαμβανομένου του διερμηνέα και του μεταγλωττιστή, αυτά τα επίπεδα λειτουργούν σε συνάρτηση προς συνάρτηση (function-by-function) ώστε να επιταχυνθεί η εκτέλεση του κώδικα.[46]

2.2 Εισαγωγή στους Μεταγλωττιστές και Διερμηνείς

Για να κατανοήσουμε σε βάθος την αναγκαιότητα και γενικότερα τη λειτουργικότητα της WebAssembly και άλλων γλωσσών προγραμματισμού, είναι επιτακτική ανάγκη να κατανοήσουμε πως εισάγεται και πως καταλήγει σε μορφή εκτέλεσης ο γραπτός κώδικας

στον υπολογιστή. Στο παρακάτω κεφάλαιο θα περιγράψουμε αυτή τη διαδικασία, καθώς και τις διαφορές μεταξύ μεταγλωττιστών και διερμηνέων, όπως και τους διαφορετικούς τους τύπους. Οι διερμηνείς και οι μεταγλωττιστές είναι προγράμματα ή μέθοδοι σχεδιασμένοι για να μεταφράζουν γλώσσες προγραμματισμού υψηλού επιπέδου, σε γλώσσες μηχανής. Η κάθε μέθοδος έχει διαφορετικό τρόπο στην εκτέλεση της. [3] [11] [47]

2.2.1 Διερμηνείς (Interpreter)

Η πρώτη εμφάνιση των διερμηνέων έγινε το 1952, όπου είχαν σαν στόχο την επίλυση του προβλήματος χώρου αποθήκευσης προγραμμάτων αλλά και την υποστήριξη των αριθμών κινητής υποδιαστολής. Η πρώτη διερμηνευμένη γλώσσα υψηλού επιπέδου ήταν η Lisp, η οποία χρησιμοποιήθηκε για πρώτη φορά το 1958 σε έναν υπολογιστή IBM 704 από τον Steve Russell. Η μέθοδος του διερμηνέα (Interpreter) πραγματοποιεί την μετατροπή του πηγαίου κώδικα γραμμή προς γραμμή (line-by-line) σε γλώσσα μηχανής. Ο διερμηνέας την αξιολογεί και τροποποιεί το πρόγραμμα κατά την εκτέλεση (Run Time). Ένας διερμηνέας μπορεί να χρειαστεί να επεξεργαστεί τις ίδιες εκφράσεις (expressions) και δηλώσεις (statements) αρκετές φορές σε σύγκριση με έναν μεταγλωττιστή, εξού και ο λόγος που θεωρείται πιο αργός. Ωστόσο, ένας διερμηνέας είναι πιο εύκολο να μετακινηθεί σε άλλο μηχάνημα και να είναι πιο γρήγορος ως προς την έναρξη της λειτουργικότητας του. Οι διερμηνείς είναι γρήγοροι στην εκκίνηση τους αλλά αργοί στην εκτέλεση.[3]

Παραδείγματα διερμηνέων είναι:

- Διερμηνέας Bytecode. Μπορεί να εκτελεστεί (executed) κατά τη διαδικασία του parsing με απευθείας εκτέλεση των εντολών. Μεταφράζει μια γλώσσα υψηλού επιπέδου σε bytecode
- Διερμηνέας Threaded code. Είναι μια μέθοδος που χρησιμοποιείται για την υλοποίηση διερμηνέων εικονικής μηχανής, ο κώδικας μου δημιουργείται από τον διερμηνέα περιέχει κυρίως κλήσεις (calls) σε υπορουτίνες (subroutines). Παρόμοιος με τον διερμηνέα bytecode με τη σημαντική διαφορά ότι χρησιμοποιεί δείκτες αντί για bytes.
- Διερμηνέας αφηρημένου συντακτικού δέντρου (Abstract syntax tree). Με το abstract syntax tree, ο διερμηνέας μπορεί να δημιουργήσει κώδικα μηχανής ή να αξιολογήσει μια εντολή και να μετατρέψει τον κώδικα σε ένα βελτιστοποιημένο αφηρημένο συντακτικό δέντρο.

2.2.2 Μεταγλωττιστές (Compilers)

Η δεύτερη μέθοδος γίνεται με τη χρήση του μεταγλωττιστή (Compiler). Ένας μεταγλωττιστής μεταφράζει τον κώδικα υψηλού επιπέδου σε κώδικα μηχανής χαμηλού επιπέδου ώστε να γίνει κατανοητό από τον υπολογιστή. Κατά τη διαδικασία της μεταγλώττισης του προγράμματος, ο μεταγλωττιστής εντοπίζει συντακτικά λάθη και τα αναφέρει μέσω του τερματικού (αφού τελειώσει - διακόψει τη διαδικασία μετάφρασης). Ένας

από τους λόγους χρήσης γλώσσας υψηλού επιπέδου είναι διότι είναι πιο κοντά στο τρόπο με τον οποίο σκέφτονται οι άνθρωποι όταν επιλύουν ένα πρόβλημα. Ο μεταγλωττιστής μπορεί να εντοπίσει αποτελεσματικά τυχόν προβλήματα με τα προγράμματα και να δώσει την επιλογή στον προγραμματιστή να μεταφέρει τον κώδικα σε πολλές διαφορετικές γλώσσες μηχανής. Οι μεταγλωττιστές, μεταγλωττίσουν τον κώδικα πριν την εκτέλεση του προγράμματος και αυτό απαιτεί χρόνο, ωστόσο μετά από αυτό το πρόγραμμα εκτελείται πιο αποτελεσματικά αφού έχει ήδη μεταφραστεί. Οι μεταγλωττιστές είναι αργοί στην εκκίνηση αλλά γρήγοροι στην εκτέλεση. [3] [11] [47]

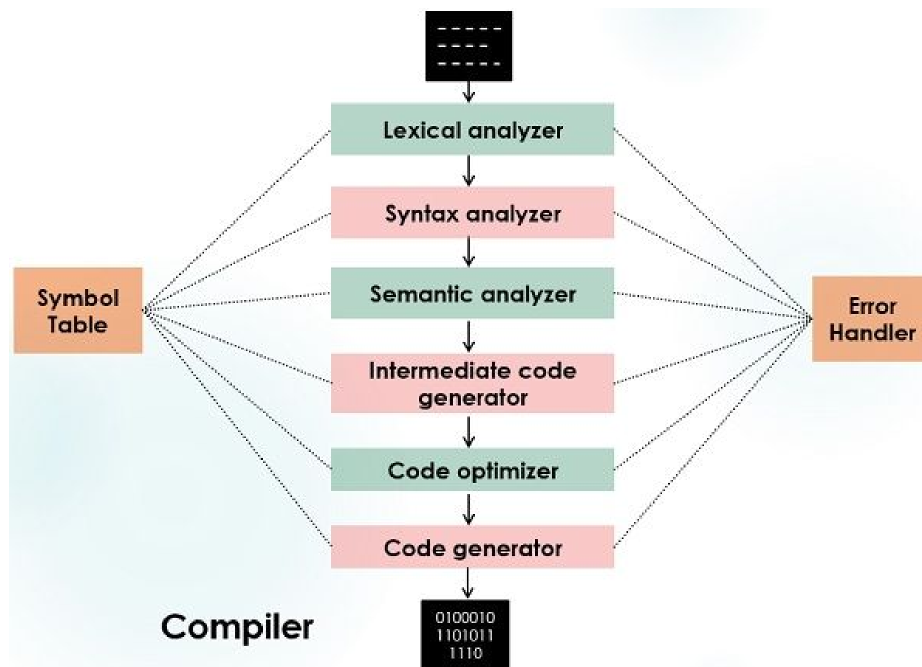
Παραδείγματα μεταγλωττιστών είναι:

- Μεταγλωττιστής από πηγή σε πηγή (source-to-source - transpiler). Μεταφράζει τον πηγαίο κώδικα γραμμένο σε μια γλώσσα προγραμματισμού και τον μετατρέπει σε μία που έχει παρόμοιο επίπεδο αφαιρετικότητας για παράδειγμα Emscripten (C/C++ σε JavaScript).
- Μεταγλωττιστής Bytecode. Μεταφράζει μια γλώσσα υψηλού επιπέδου σε μια ενδιάμεση γλώσσα (Intermediate code) που μπορεί να ερμηνευτεί από έναν διερμηνέα bytecode ή από μια εικονική μηχανή (Virtual Machine)
- Συμβολομεταφραστής (Assembler). Μεταφράζει γλώσσα assembly η οποία είναι αναγνώσιμη από τον άνθρωπο σε γλώσσα μηχανής.

2.2.3 Φάσεις του μεταγλωττιστή

Τα διάφορα στάδια των διαδικασιών μεταγλώττισης (compiling) και ερμηνείας (interpreting) αναφέρονται στο παρακάτω σχήμα. Οι δύο ομάδες στις οποίες θα μπορούσαμε να κατατάξουμε τις φάσεις είναι αυτή της ανάλυσης και της σύνθεσης. Η φάση της λεξικής ανάλυσης (Lexical analyzer) αποτελεί τη πρώτη φάση και επιτελεί τη σάρωση του πηγαίου κώδικα. Η Δεύτερη φάση είναι η ανάλυση της σύνταξης (Syntax analyzer) όπου αφορά τον έλεγχο της σωστής συγγραφής, δομής και σύνταξης του κώδικα, τέλος δημιουργείται ένα δέντρο σύνταξης με χρήσιμες εγγραφές. Η σημασιολογική ανάλυση (Semantic analyzer) είναι η τρίτη φάση όπου ελέγχεται η σημασιολογική συνέπεια του κώδικα, το δέντρο σύνταξης που δημιουργήθηκε στη προηγούμενη φάση, συμβάλλει στον έλεγχο της σημασιολογικής συνέπειας. Μετά τα 3 στάδια ανάλυσης και αφού ελεγχθεί ο κώδικας σε σχέση με τη σύνταξη και τους κανόνες της γλώσσας προγραμματισμού, δημιουργείται ο ενδιάμεσος κώδικας (Intermediate code generator) όπου περνάει στο στάδιο της σύνθεσης για να βελτιστοποιηθεί (Code optimizer) και στην τελική φάση να μεταφραστεί σε γλώσσα μηχανής. [3], [11], [47],[8],[49]

Εικόνα 2.1: Οι φάσεις του μεταγλωττιστή [49]



2.2.4 Σύγκριση Μεταγλωττιστή και Διερμηνέα

Όπως βλέπουμε παρακάτω στην εικόνα 2.1, οι διαφορές μεταξύ των δύο μεθόδων μετάφρασης είναι αρκετά σημαντικές και φαίνεται πως ο μεταγλωττιστής αποτελεί μια πιο αποτελεσματική λύση. Ωστόσο, ένας διερμηνέας είναι περισσότερο βολικός κατά την ανάπτυξη ενός προγράμματος, καθώς είναι γρηγορότερος στην εκτέλεση των εντολών αλλά και στη διαδικασία της αποσφαλμάτωσης (debugging), λόγω του ότι τα μηνύματα των σφαλμάτων, περιέχουν αναφορές στη γραμμή του κώδικα. Οι πρώτες μηχανές JavaScript που χρησιμοποιήθηκαν από προγράμματα περιήγησης ήταν διερμηνείς, μέχρις ότου να δημιουργηθούν οι μεταγλωττιστές JIT (Just-In-Time) οι οποίοι και τους αντικατέστησαν.[49]

Σύγκριση	Μεταγλωττιστής	Διερμηνέας
Είσοδος	Ολόκληρο το πρόγραμμα	Μία γραμμή του προγράμματος / μία εντολή
Έξοδος	Δημιουργεί ενδιάμεσο κώδικα αντικειμένου	Δεν δημιουργεί ενδιάμεσο κώδικα αντικειμένου
Μηχανισμός εργασίας	Το πρόγραμμα μεταφράζεται πριν την εκτέλεση	Το πρόγραμμα μεταφράζεται και εκτελείται ταυτόχρονα
Χρόνος εκτέλεσης	Γρηγορότερος	Πιο αργός

Μνήμη	Απαιτεί περισσότερη	Απαιτεί λιγότερη
Σφάλματα	Προβολή σφαλμάτων μετά την εκτέλεση	Προβολή σφαλμάτων με αναφορές για κάθε μια γραμμή του κώδικα
Γλώσσες Προγραμματισμού	C, C++, C#, Typescript	PHP, Python, Perl, Matlab

Πίνακας 2.1: Πίνακας συγκρίσεων μεταγλωττιστή και διερμηνέα [49]

2.3 Profile-based Optimization

Η JavaScript είναι μια δυναμική γλώσσα για τον λόγο του ό,τι έχει μερικές δυναμικές πτυχές, σχεδόν όλα είναι δυναμικά. Για παράδειγμα οι μεταβλητές που χρησιμοποιεί είναι δυναμικές ως προς τον τύπο αλλά και ως προς αποβελτιστοποίηση

την ύπαρξη τους, ακόμη και οι ιδιότητες μπορούν να εισάγονται και να διαγράφονται δυναμικά. Επίσης παρέχεται η δυνατότητα δημιουργίας μεταβλητής και ο καθορισμός του τύπου της κατά το χρόνο εκτέλεσης του προγράμματος. Εξαιτίας αυτού, ο μεταγλωττιστής χρειάζεται συνεχώς να αξιολογεί τον τύπο της μεταβλητής ή του αντικειμένου που κλίνεται να μεταφραστεί. Μετά από ένα χρονικό διάστημα και αριθμό επαναλήψεων, ο μεταγλωττιστής θα προσπαθήσει να προβλέψει την επόμενη εκτέλεση του κώδικα που ακολουθεί βασισμένος σε μοτίβα που συλλέχθηκαν, από προηγούμενες εκτελέσεις. Έτσι δημιουργούνται τα προφίλ (profiles) που αναφέρθηκαν προηγουμένως. Οι πληροφορίες των προφίλ σχετίζονται με τις δυναμικές συμπεριφορές της JavaScript κατά τη διάρκεια της εκτέλεσης του προγράμματος. Στη συνέχεια ο JIT προχωρά στην δημιουργία κώδικα για να ταιριάξει τα προφίλ, προσπαθώντας έτσι να προβλέψει ποια θα είναι η συμπεριφορά του κώδικα σε μελλοντικές εκτελέσεις.

Οι μηχανές JavaScript τείνουν να χρησιμοποιούν κρυφές κλάσεις, μια λίστα ιδιοτήτων και τις μετατοπίσεις τους εντός του αντικειμένου. Αυτό συμβαίνει για την αναπαράσταση του σχήματος ενός αντικειμένου, που σημαίνει ότι κάθε αντικείμενο έχει ένα δείκτη προς την κρυφή του κλάση. Η κρυφή κλάση χρησιμοποιείται για να αποκτήσουμε πρόσβαση σε μια συγκεκριμένη ιδιότητα ενός αντικειμένου. Ανάλογα με τον κώδικα και τι παρατηρεί (observe) ο JIT όταν προσπαθεί να πραγματοποιήσει την βελτιστοποίηση, αγνοείται η κρυφή συνάρτηση που ονομάζεται inline caching, επειδή έχει παρατηρήσει ότι η συνάρτηση καλείται πάντα με το ίδιο αντικείμενο και θα περιμένει το ίδιο πανομοιότυπο αντικείμενο στο κοντινό μέλλον. Το JIT προσπαθεί να επικυρώσει κάθε υπόθεση που κάνει εισάγοντας έναν κωδικό προστασίας. Η διεύθυνση της κρυφής κλάσης αποθηκεύεται στην κρυφή μνήμη και στη συνέχεια συνέχεια συγκρίνεται με τον τρέχον αντικείμενο. Αν διαφέρει από αυτό, τότε σηματοδοτεί τον μεταγλωττιστή ότι ένα άλλο αντικείμενο με διαφορετικό σχήμα έχει περάσει ως όρισμα. Έτσι ο JIT θα αρχίσει να ανακάμπτει, η διαδικασία αυτή ονομάζεται αποβελτιστοποίηση (deoptimization) και θα σταματήσει την εκτέλεση στο συγκεκριμένο σημείο του βελτιστοποιημένου κώδικα, επιστρέφοντας στην αντίστοιχη θέση του αρχικού κώδικα, αυτή τη φορά με απλή εκτέλεση. [22]

2.4 Just-In-Time (JIT)

Όπως παρατηρείτε στην προηγούμενη υποενότητα, η μέθοδος της μεταγλώττισης του κώδικα παρέχει μικρότερο χρόνο στην εκτέλεση του προγράμματος, ενώ ο διερμηνέας παρέχει γρηγορότερη εκτέλεση των εντολών κατά την εκκίνηση (faster startup). Αυτές οι δύο παροχές των δύο μεθόδων, οδήγησαν τους μηχανικούς των προγραμμάτων πλοήγησης στην δημιουργία των Just-In-Time compilers. Οι JITCs σχεδιάστηκαν για να μεταφράζουν τον κώδικα κατά τη διάρκεια της εκτέλεσης τους.

Οι JITCs τείνουν να χωρίζονται σε τρία επίπεδα. Ξεκινώντας από το πρώτο επίπεδο, έχουμε τη διαδικασία μετάφρασης του πηγαίου κώδικα από τον αναλυτή (parser) σε bytecode. Στη συνέχεια εκτελείται το bytecode που δημιουργήθηκε για να διασφαλιστεί η γρήγορη έναρξη του προγράμματος. Από αυτό το σημείο η εκτέλεση επαναλαμβάνεται πολλές φορές, ανάλογα με την μηχανή JavaScript (JavaScript Engines) που χρησιμοποιείται ορίζονται και οι επαναλήψεις. Οι μηχανές ορίζουν στις πόσες επαναλήψεις της εκτέλεσης θα θεωρηθεί σε 'ζεστή' (warm) η κατάσταση του bytecode. Με τον όρο warm εννοούμε τα τμήματα του κώδικα τα οποία επαναλαμβάνονται συχνά, αν τα τμήματα επαναλαμβάνονται πάρα πολλές φορές χαρακτηρίζονται και ως "ζεστό" (hot). Όταν λοιπόν το bytecode χαρακτηριστεί ως "ζεστό", τότε ο μεταγλωττιστής JIT θα ενεργοποιηθεί και θα αρχίσει την μετάφραση του bytecode σε γλώσσα μηχανής, εφαρμόζοντας μικρές βελτιστοποιήσεις. Στη συνέχεια ο JIT θα συλλέξει κάποιες πληροφορίες από τον κώδικα για να δημιουργήσει το προφίλ του, ώστε να εισαχθεί ο instrumentation code. [8]

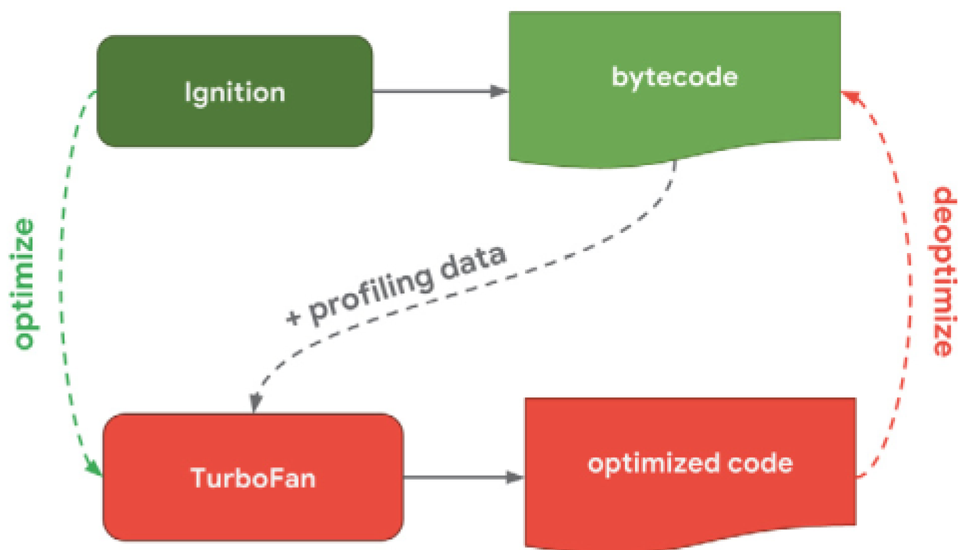
Χρειάζονται αρκετές εκτελέσεις έτσι ώστε να σχηματιστεί το προφίλ του κώδικα, οι οποίες εξαρτώνται από την μηχανή JavaScript που χρησιμοποιεί το πρόγραμμα περιήγησης. Οι εντολές - συναρτήσεις που θεωρήθηκαν "ζεστές" έχουν υποστεί μετάφραση και χρησιμοποιούνται ως μεταφρασμένες στις επόμενες εκτελέσεις, κερδίζοντας έτσι σημαντικό χρόνο. Αν μια εντολή - συνάρτηση επαναληφθεί πολλές φορές τότε χαρακτηρίζεται ώς "καυτή" και έτσι ξεκινάει η τρίτη φάση στον μεταγλωττιστή, όπου ο ήδη βελτιστοποιημένος JIT θα ξεκινήσει την επαναμετάφραση του bytecode χρησιμοποιώντας επιπλέον βελτιστοποιήσεις, μία από αυτές βασίζεται στο προφίλ (profile-based) που αναφέρθηκε προηγουμένως.[8]

2.5 V8 Engine

Η γλώσσα μηχανής V8, λειτουργεί λαμβάνοντας τον πηγαίο κώδικα της JavaScript και περνώντας τον μέσω του αναλυτή (parser) για να δημιουργήσει μια ενδιάμεση αναπαράσταση στη μορφή αφηρημένου συντακτικού δέντρου (AST). Ένα αφηρημένο συντακτικό δέντρο διατηρεί σε μια δενδρική μορφή, την αναπαράσταση της δομής του πηγαίου κώδικα. Κάθε κόμβος στο δέντρο διατηρεί μια κατασκευή (construct) μέσα στον πηγαίο κώδικα. Στη συνέχεια το AST εκτελείται από τον πρώτο διερμηνέα μου ονομάζεται

ignition (αναφλέκτης). Ο διερμηνέας ignition θα προχωρήσει στη δημιουργία μη βελτιστοποιημένου κώδικα μηχανής ή αλλιώς bytecode. Ο bytecode είναι μια αφαίρεση του κώδικα μηχανής.[13]

Εικόνα 2.2: Μεταγλώττιση και εκτέλεση κώδικα σε μηχανή V8[25]



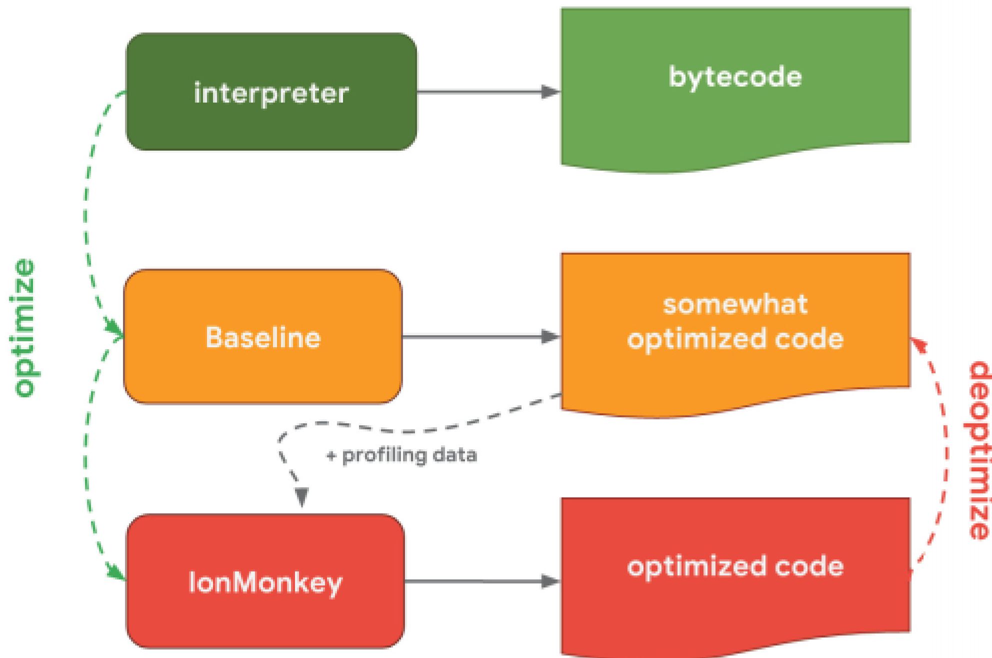
Ο διερμηνέας ignition αποτελείται από bytecode χειριστές όπου χειρίζονται συγκεκριμένα bytecodes και τους περνούν στο πρόγραμμα χειρισμού ώστε να επεξεργαστεί το επόμενο bytecode. Η αρχιτεκτονική τους είναι υλοποιημένη σε κώδικα assembly. Αυτό σημαίνει ότι ο διερμηνέας μπορεί να δημιουργηθεί μία φορά και στη συνέχεια να βασίζεται στον μεταγλωττιστή TurboFan για να δημιουργήσει εντολές μηχανής για κάθε αρχιτεκτονική που υποστηρίζεται από τη μηχανή V8.[25]

2.6 Spidermonkey Engine

Η γλώσσα μηχανής SpiderMonkey χρησιμοποιεί μια διαφορετική προσέγγιση για το JITC του. Ο πηγαίος κώδικας μεταβιβάζεται στον αναλυτή (parser) ο οποίος δημιουργεί ένα αφηρημένο συντακτικό δέντρο (AST). Το αφηρημένο δέντρο χρησιμοποιείται από τον διερμηνέα για τη δημιουργία bytecode, το οποίο μεταβιβάζεται στο μεταγλωττιστή βελτιστοποίησης με δεδομένα προφίλ (profiling data). Η διαφορά από τη μηχανή V8 είναι ότι το SpiderMonkey έχει δύο μεταγλωττιστές βελτιστοποίησης αντί για έναν. Ο πηγαίος κώδικας περνάει πρώτα από το διερμηνέα για να βελτιστοποιηθεί στο baseline μεταγλωττιστή (βασικής γραμμής). Ο μεταγλωττιστής baseline θα παράξει και θα αναλύσει

τον βελτιστοποιημένο κώδικα. Στη συνέχεια θα μεταβιβαστεί στον μεταγλωττιστή μεγάλης βελτιστοποίησης, IonMonkey.

Εικόνα 2.3: Μεταγλώττιση και εκτέλεση κώδικα σε μηχανή SpiderMonkey [25]



2.7 Native Client (NaCl)

Το 2011, η Google κυκλοφόρησε ένα νέο έργο ανοιχτού κώδικα που ονομάστηκε Native Client (NaCl). Η ιδέα ήταν να παρέχεται σχεδόν native ταχύτητα εκτέλεσης κώδικα στο φυλλομετρητή ενώ ταυτόχρονα η εκτέλεση να πραγματοποιείται σε ένα δοκιμαστικό περιβάλλον (sandbox) με περιορισμένα δικαιώματα, για λόγους ασφαλείας.

Η τεχνολογία κάλυψε ορισμένους από τους μεγαλύτερους στόχους της Google, όπως για παράδειγμα την υποστήριξη του Chrome OS και τη μεταφορά των desktop application σε web application. Οι περιπτώσεις χρήσης αφορούσαν κυρίως την υποστήριξη εφαρμογών υψηλής υπολογιστικής ισχύος σε περιβάλλον φυλλομετρητή. Οι εφαρμογές αυτές είναι βίντεο-παιχνίδια, επεξεργασία ήχου και βίντεο, επιστημονικές πράξεις, προσομοιώσεις. Η βασική εστίαση ήταν στις γλώσσες προγραμματισμού C και C++ ως γλώσσες πηγής, επειδή όμως ήταν βασισμένη στην αλυσίδα εργαλείων μεταγλωττιστή LLVM, θα ήταν δυνατή η υποστήριξη επιπρόσθετων γλωσσών προγραμματισμού που θα μπορούσαν να δημιουργήσουν το LLVM Intermediate representation (ενδιάμεση αναπαράσταση). [14]

Υπήρχαν δύο μορφές διανεμητέου κώδικα. Το πρώτο ήταν το ομώνυμο NaCl που οδήγησε σε μονάδες «nexe» που θα στόχευαν συγκεκριμένη αρχιτεκτονική υλικού (π.χ. ARM ή x86-64) και θα μπορούσε να διανέμεται μέσω του καταστήματος Google Play. Το άλλο ήταν μια φορητή φόρμα PNaCl που θα εκφραζόταν στη δημιουργία μορφής Bitcode

του LLVM. Αυτά ονομάζονται “pexe” modules και θα έπρεπε να μετατραπούν σε μία native αρχιτεκτονική στο περιβάλλον του πελάτη. Η τεχνολογία ήταν επιτυχημένη, με την έννοια ότι η απόδοση της ταχύτητας που δοκιμάστηκε στο πρόγραμμα περιήγησης ήταν ελάχιστα μικρότερη από αυτή της native. Χρησιμοποιώντας τεχνικές απομόνωσης σφαλμάτων λογισμικού (software fault isolation SFI), δημιουργήθηκε η δυνατότητα λήψης υψηλής απόδοσης κώδικα από τον ιστό αλλά και η εκτέλεση του σε προγράμματα περιήγησης. Αρκετά δημοφιλή παιχνίδια όπως το Quake και το Doom μεταγλωττίστηκαν με επιτυχία σε αυτήν τη μορφή. Το πρόβλημα ήταν ότι τα δυαδικά NaCl θα έπρεπε να δημιουργηθούν και να συντηρηθούν ξεχωριστά για κάθε φυλλομετρητή. Ενώ η εκτέλεση σε πειραματικά περιβάλλοντα (sandboxes) ήταν εφικτή, δημιουργήθηκε η απαίτηση για στατική επικύρωση των δυαδικών αρχείων ώστε να διασφαλιστεί ότι δε θα υπήρχε πρόσβαση στις υπηρεσίες του λογισμικού. Ο κώδικας που δημιουργήθηκε με αυτόματο τρόπο, έπρεπε να ακολουθήσει συγκεκριμένες διευθύνσεις και μοτίβα στοίχισης ορίων, ώστε να εξασφαλιστεί η μη παραβίαση εκχωρημένων χώρων μνήμης.

Η υποδομή LLVM θα μπορούσε να δημιουργήσει είτε τον native κώδικα NaCI είτε το φορητό Bitcode χωρίς να τροποποιήσει την αρχική πηγή. Αυτό είναι ένα χρήσιμο αποτέλεσμα. Παρόλα αυτά, υπάρχουν διαφορές μεταξύ φορητότητας κώδικα και εφαρμογής. Οι εφαρμογές απαιτούν τη διαθεσιμότητα των API για τη λειτουργικότητα τους. Η Google παρείχε μια δυαδική διεπαφή εφαρμογής (application binary interface ABI) που ονομάζονται Pepper API για υπηρεσίες χαμηλού επιπέδου όπως βιβλιοθήκες τρισδιάστατων γραφικών, αναπαραγωγής ήχου, πρόσβασης σε αρχεία (προσομοίωση μέσω IndexedDB ή LocalStorage) και πολλά άλλα. Ενώ οι μονάδες (modules) PNaCI θα μπορούσαν να τρέξουν σε φυλλομετρητές που είχαν την κατάλληλη υποδομή. [23]

2.8 asm.js

Ο πρωταρχικός λόγος που υποκινήθηκε το έργο asm.js, ήταν για να φέρει μια καλύτερη εμπειρία στα βίντεο παιχνίδια. Αυτό σύντομα επεκτάθηκε για να συμπεριλάβει την επιθυμία της μετάγγισης εφαρμογών σε πειραματικά περιβάλλοντα (sandboxes) του κάθε προγράμματος περιήγησης, ώστε να χρησιμοποιείται χωρίς να χρειάζεται ουσιαστική τροποποίηση του κώδικα της εφαρμογής, όλα αυτά στα πλαίσια ενός ασφαλούς περιβάλλοντος.

Η λειτουργία σε αυτό το περιβάλλον θα επέτρεπε στις εφαρμογές του να χρησιμοποιήσουν οποιαδήποτε χαρακτηριστικό της JavaScript. Οι μηχανές JavaScript ήταν αποτελεσματικές σε αυτό το περιβάλλον και είχαν υποβληθεί σε σημαντικούς ελέγχους ασφαλείας. Το πρόβλημα που παρέμεινε ήταν η αδυναμία βελτιστοποίησης της JavaScript ahead-of-time (AoT), ώστε η απόδοση χρόνου εκτέλεσης να βελτιωθεί περισσότερο. Λόγω της δυναμικής του φύσης και την έλλειψης κατάλληλης υποστήριξης ακέραιων αριθμών, υπήρχαν αρκετά εμπόδια στην βελτίωση της απόδοσης που δεν μπορούσαν να αντιμετωπιστούν ουσιαστικά μέχρι να φορτωθεί ο κώδικας στο πρόγραμμα περιήγησης. Όταν ο κώδικας φορτωνόταν, οι μεταγλωττιστές βελτιστοποίησης Just-in-Time (JIT) ήταν σε θέση να επιταχύνουν την απόδοση της εφαρμογής. Παρόλα αυτά εξακολουθούσαν να υπάρχουν εγγενή ζητήματα, όπως οι αναφορές πινάκων αργών ορίων (Array bounds). Η

JavaScript δεν μπορούσε να βελτιστοποιηθεί στο σύνολό της, παρά μόνο ένα μεγάλο υποσύνολο της. Το asm.js χρησιμοποιούσε τον μεταγλωττιστή Clang που είναι βασισμένος στο LLVM μέσω της αλυσίδας εργαλείων της Emscripten. [48]

Το LLVM αντιπροσωπεύει μια καθαρή αρχιτεκτονική (modular architecture) ώστε τμήματα αυτού να μπορούν να αντικατασταθούν όπως και η δημιουργία backend κώδικα μηχανής. Η Emscripten θα μπορούσε να επαναχρησιμοποιήσει τα δύο πρώτα στάδια της βελτιστοποίησης και ανάλυσης (optimization and parsing) και στη συνέχεια να εκπέμψει αυτό το υποσύνολο της JavaScript ως προσαρμοσμένο backend. Επειδή η έξοδος ήταν σε JavaScript, θα ήταν πολύ πιο φορητή από την προσέγγιση της NaCl/PNaCl. Η σημαντική απώλεια που εμφανίστηκε ήταν αυτή της απόδοσης. Συμπερασματικά, αποτελεί μια σημαντική βελτίωση σε σχέση με την απλή JavaScript, αλλά όχι τόσο αποτελεσματική όσο η προσέγγιση της Google. Πέρα από τη μέτρια απόδοση του προτύπου, ενδιαφέρον αποτελεί για τους προγραμματιστές η ικανότητα να μεταφέρουν ήδη υλοποιημένες εφαρμογές σε γλώσσα C και C++ σε περιβάλλον περιήγησης (browsers). [15]

3 WebAssembly στην Πράξη

3.1 Το πρότυπο WebAssembly και ο Σχεδιασμός του

Το πρότυπο WebAssembly είναι ένας νέος τύπος ενδιάμεσης γλώσσας, υλοποιημένος σε δυαδική μορφή για να εκτελείται στον ιστότοπο παράλληλα με την JavaScript, παρέχει ορισμένες δυνατότητες και βελτιώνει την απόδοση[4], [18], [51]. Προκειμένου οι διαφορετικές μηχανές JavaScript να είναι σε θέση να εκτελέσουν ενότητες (modules) της WebAssembly, οι εταιρείες περιήγησης (Google Chrome, Mozilla Firefox, Opera) έχουν επεκτείνει τις δυνατότητες της αντίστοιχης εικονικής μηχανής τους, επιτρέποντας την εκτέλεση του προτύπου στους φυλλομετρητές.[18] Ένας από τους λόγους για τον οποίο η γλώσσα είναι κατασκευασμένη σε δυαδική μορφή, είναι για να συμβάλει στην επιτάχυνση των υπολογιστικών εργασιών και να εξασφαλίσει σταθερή απόδοση σε αντίθεση με την απλή JavaScript [46]. Ωστόσο, η WebAssembly δεν προορίζεται για να γραφτεί με το χέρι, αντιθέτως έχει ως στόχο να μεταγλωττιστεί από άλλες γλώσσες προγραμματισμού. Ένα από τα μειονεκτήματα της τεχνολογίας είναι ότι δεν υποστηρίζει τη συλλογή σκουπιδιών, σύμφωνα όμως με τους κατασκευαστές, είναι ένας μελλοντικός στόχος καθώς θα την απλοποιήσει σε μεγάλο βαθμό. Προς το παρόν η διαχείριση των σκουπιδιών πραγματοποιείται από την γλώσσα που μεταγλωττίστηκε σε WebAssembly. Αυτό σημαίνει πως ο αρχικός κώδικας μπορεί να είναι γραμμένος σε πολλές διαφορετικές γλώσσες που προηγουμένως δεν είχαν τη δυνατότητα να εκτελεστούν στο διαδίκτυο. Έτσι δίνεται η ευκαιρία στου προγραμματιστές να μεταφέρουν υλοποιημένες εφαρμογές σε γλώσσες όπως C/C++, Rust στον ιστό.

Γενικά, οι μεταγλωττιστές λειτουργούν παίρνοντας μια γλώσσα πηγής (source language) και μετατρέποντάς την σε γλώσσα μηχανής (machine language), την οποία μπορεί στη συνέχεια να χρησιμοποιήσει ο κεντρικός επεξεργαστής για να εκτελέσει τις παρεχόμενες εντολές. Το πρότυπο WebAssembly είναι παρόμοιο με μια γλώσσα assembly, η βασική εξαίρεση είναι ότι κάθε γλώσσα assembly (x86, ARM) είναι ειδικά συνδεδεμένη με συγκεκριμένη αρχιτεκτονική μηχανής, ενώ η WebAssembly όχι. Ο λόγος πίσω από αυτό είναι ότι ποτέ δεν γνωρίζουμε τι είδους αρχιτεκτονική θα χρησιμοποιηθεί από τον τελικό χρήστη. Ως εκ τούτου, υπάρχει ανάγκη να παρέχουμε έναν τύπο αρχιτεκτονικής γλώσσας assembly [34]. Η τεχνολογία είναι ένα βήμα κοντά στον πραγματικό κώδικα μηχανής από τον πηγαίο κώδικα JavaScript, επειδή έχει πιο άμεση αντιστοίχιση με τον κώδικα μηχανής. Όταν ένα σενάριο εκτελείται, το πρόγραμμα περιήγησης θα κατεβάσει την WebAssembly και θα την μεταγλωττίσει απευθείας σε γλώσσα μηχανής για το υποκείμενο μηχάνημα (hardware). Αυτό την καθιστά αποτελεσματική με την εκτέλεση και ταχύτερη από την κανονική μεταγλώττιση Just-in-time της JavaScript, διότι παραβλέπεται η ανάγκη της ανάλυσης (parsing) και βελτιστοποίησης του πηγαίου κώδικα [34].

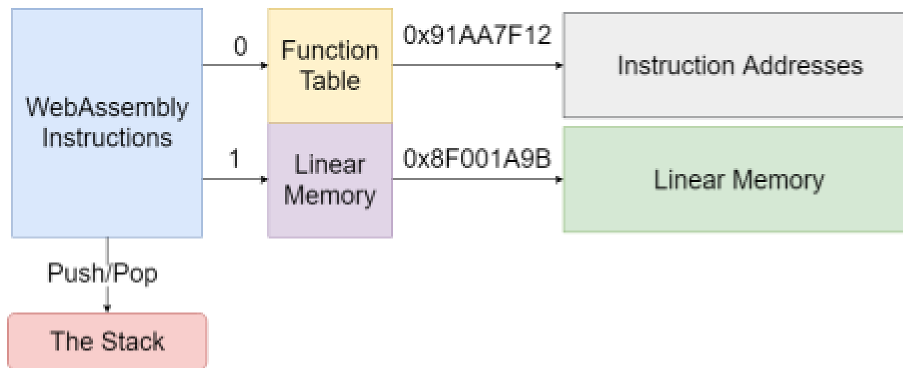
Η WebAssembly είναι υλοποιημένη γύρω από ορισμένες έννοιες προκειμένου να μπορεί να κωδικοποιήσει και να παρέχει δυαδική μορφή χαμηλού επιπέδου. Η τεχνολογία είναι λίγο περιορισμένη όσον αφορά τους τύπους μεταβλητών της, καθώς λειτουργεί μόνο με τέσσερις βασικούς τύπους. Όλοι αυτοί είναι τυπικοί αριθμοί κινητής υποδιαστολής IEEE. Καθένας από αυτούς τους τύπους μπορεί να έχει μέγεθος 32 ή 64 bit. Επίσης, δεν υπάρχει διάκριση μεταξύ προσημειωμένων και μη ακέραιων αριθμών [40]. Η τεχνολογία χειρίζεται τις εντολές βασίζοντας σε μια δομή δεδομένων στοίβας που λειτουργεί σύμφωνα με την αρχή της τελευταίας εισόδου (last-in, first-out).

Όταν γράφουμε κώδικα σε WebAssembly πολύ πιθανό ορισμένες εντολές να προκαλέσουν παγίδες (traps) οι οποίες διακόπτουν την εκτέλεση ακαριαία. Μια παγίδα μπορεί να είναι η πρόσβαση σε μνήμη που δεν έχει οριστεί. Όταν λοιπόν συμβεί αυτό η WebAssembly δε διαχειρίζεται την αντιμετώπιση τους. Αντιθέτως αναφέρονται στο εξωτερικό περιβάλλον.[40].

Οι συναρτήσεις λειτουργούν λαμβάνοντας μια ακολουθία τιμών ως παραμέτρους και μπορούν να καλούν η μία την άλλη, περιέχοντας έτσι την αναδρομή. Οι συναρτήσεις μπορούν επίσης να δηλώνουν τοπικές μεταβλητές που μπορούν να χρησιμοποιηθούν ως εικονικοί καταχωρητές. Στην τρέχουσα έκδοση του WebAssembly οι συναρτήσεις μπορούν να επιστρέψουν μόνο μία τιμή.[40]

Η WebAssembly χρησιμοποιεί γραμμική μνήμη, η οποία είναι μια σειρά από ακατέργαστα byte. Η μνήμη δημιουργείται με αρχικό μέγεθος που ορίζεται από τον προγραμματιστή και μπορεί να αυξηθεί με την μέθοδο memory.grow ανάλογα με τις απαιτήσεις του αλγόριθμου. Όταν εκτελείται ένα πρόγραμμα, μπορεί να αποθηκεύσει τιμές στη μνήμη, μπορεί επίσης να έχει πρόσβαση σε αυτό σε οποιαδήποτε διεύθυνση. [40]

Εικόνα 3.1: Πρόσβαση στην γραμμική μνήμη με εντολές και συναρτήσεις της WASM μέσω δεικτών. [40]



Οι πραγματικές διευθύνσεις της γραμμικής μνήμης και τα δεδομένα που υπάρχουν μέσα σε αυτήν είναι κρυμμένα από το WebAssembly. Με άλλα λόγια, η WebAssembly δεν μπορεί να έχει απευθείας πρόσβαση στα περιεχόμενα της μνήμης. Αντίθετα, θα ζητήσει τα δεδομένα από ένα ευρετήριο της γραμμικής μνήμης και έτσι το πρόγραμμα περιήγησης φροντίζει να βρει την πραγματική τους διεύθυνση. Αυτός είναι ένας λόγος για τον οποίο η WebAssembly είναι πιο ασφαλές από τον εγγενή κώδικα της C/C++ κλπ. Η μόνη μνήμη στην οποία έχει πρόσβαση η WebAssembly είναι ο γραμμικός buffer μνήμης. Οποιαδήποτε προσπάθεια πρόσβασης στη μνήμη εκτός αυτής, θα έχει απλώς ως αποτέλεσμα ένα σφάλμα “εκτός ορίων” (out of bounds) σε μορφή JavaScript.

Οι σχεδιαστικοί στόχοι του wasm όπως ορίζονται από την βιβλιογραφία είναι οι εξής: γρήγορη ταχύτητα, ασφαλές και φορητό περιβάλλον. Ο κώδικας wasm εκτελείται σχεδόν σε εγγενή ταχύτητα (native speed) σε ένα ασφαλές περιβάλλον (sandbox) για την μνήμη αλλά και για την αποφυγή της αλλοίωσης δεδομένων (data corruption). Η δύο μορφές αρχείων που αποτελούν την εξεταζόμενη τεχνολογία, είναι η δυαδική μορφή .wasm και η .wat όπου είναι μια μορφή κειμένου αναγνώσιμη από τον άνθρωπο. Οι δύο αυτές μορφές αντιστοιχούν σε μια κοινή δομή. Για παράδειγμα στη δυαδική μορφή (wasm) ο τύπος μίας τιμής i32 αναπαρίσταται ως δεκαεξαδικός 0x6E, όπου στην δυαδική αναπαράσταση είναι 0110 1110 (εικόνα 3.2)

Εικόνα 3.2: Τύποι μεταβλητών της wasm σε δεκαεξαδική μορφή και κειμένου

0x6E => i32
0x6F => i64
0x6C => f32
0x6D => i64

Η δυαδική μορφή μεταδίδεται γρήγορα λόγω της συμπιεσμένης μορφής της. Οι λειτουργικές μονάδες (modules) της wasm επιτρέπουν την ξεχωριστή μετάδοση αλλα και αποθήκευση στην κρυφή μνήμη (cache). Η ξεχωριστή επεξεργασία των μονάδων παρέχει μεγαλύτερη αποδοτικότητα στην τεχνολογίας μας, καθώς η διαδικασία της αποκωδικοποίησης,

επικύρωσης και μεταγλώττισης μπορούν να ξεκινήσουν προτού διαβαστούν όλα τα δεδομένα αλλά και να χωριστούν σε παράλληλες εργασίες.

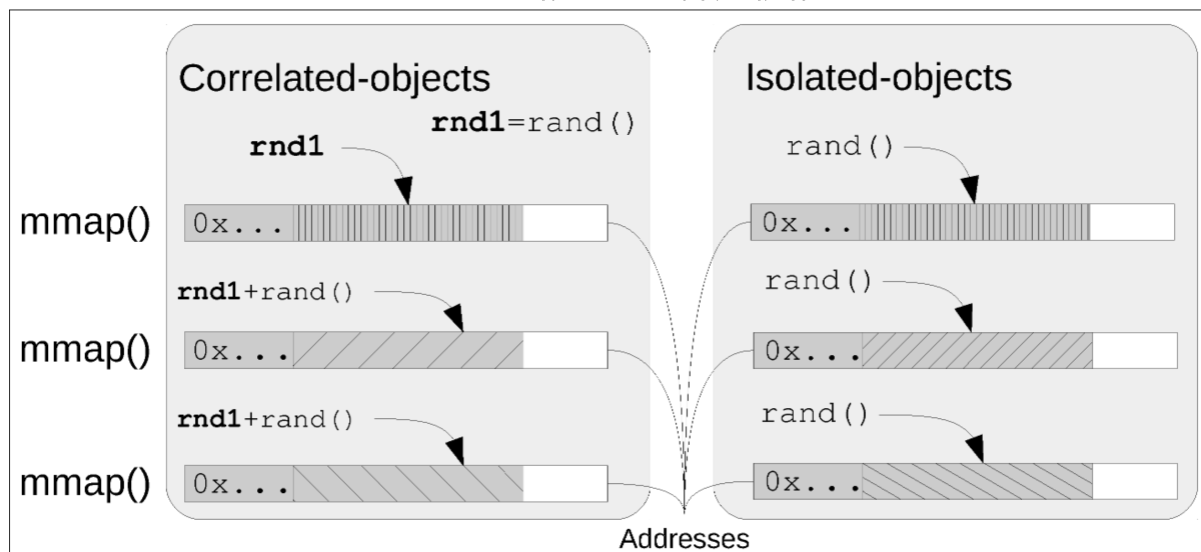
3.2 Σημασιολογική Φάση

Οι ενότητες (modules) διανέμονται σε δυαδική μορφή όπου αργότερα αποκωδικοποιούνται και μετατρέπονται σε εσωτερική αναπαράσταση μιας ενότητας. Σε μια πραγματική υλοποίηση, αυτό θα μεταγλωττιζόταν απευθείας σε κώδικα μηχανής. Για να λειτουργήσουν οι λειτουργικές μονάδες wasm, θα πρέπει να είναι έγκυρες. Η φάση της επικύρωσης (validation phase) πραγματοποιεί ελέγχους σε διάφορες συνθήκες ώστε να διασφαλιστεί, ότι η μονάδα (module) είναι ασφαλής και εκτελέσιμη. Για παράδειγμα μια ιδιαίτερα σημαντική προϋπόθεση είναι να πραγματοποιηθεί ο έλεγχος του τύπου της συνάρτησης. [21] Όταν η μονάδα (module) έχει αποκωδικοποιηθεί και επικυρωθεί, είναι έτοιμη προς εκτέλεση. Η εκτέλεση μιας λειτουργικής μονάδας μπορεί να θεωρηθεί ως δημιουργία ενός στιγμιότυπου της λειτουργικής μονάδας, που σημαίνει ότι όλα όσα απαιτούνται για την εκτέλεση πρώτα θα εγκατασταθούν , όπως για παράδειγμα η μνήμη. Η λειτουργική μονάδα μπορεί να εκτελεστεί με την κλήση μιας εξαγόμενης συνάρτησης. [40]

3.3 Ασφάλεια Προτύπου

Εφόσον η WebAssembly εκτελείται σε απομονωμένο περιβάλλον (sandbox) διαχωρίζεται από τον χρόνο εκτέλεσης τους κεντρικού υπολογιστή και δε μπορεί να χρησιμοποιηθεί χωρίς τη χρήση διεπαφής προγραμματισμού (API). Επιπλέον, κάθε λειτουργική μονάδα υπόκειται στις πολιτικές ασφαλείας της ενσωμάτωσής της. Σε ένα πρόγραμμα περιήγησης ιστού, αυτό περιλαμβάνει περιορισμούς στη ροή πληροφοριών μέσω της ίδιας πολιτικής. Σε μια πλατφόρμα εκτός ιστού, αυτό θα μπορούσε να περιλαμβάνει το μοντέλο ασφαλείας POSIX. Η μνήμη της WebAssembly είναι γραμμική, αυτό σημαίνει ότι το μπλοκ της μνήμης είναι πλήρης συνεχόμενο. Η μνήμη είναι διαθέσιμη από τον δείκτη με την θέση 0 έως του μέγιστου ορίου μνήμης. Αντιθέτως η μνήμη μιας εγγενούς υλοποίησης (π.χ γλώσσα C) έχει πολλά ενδιάμεσα κενά στις καταχωρήσεις των διευθύνσεων της. Σε περίπτωση που ένα κακόβουλο πρόγραμμα προσπαθήσει να διαβάσει και να γράψει σε μία μη αντιστοίχιση σελίδα μνήμης, τότε θα αποτύχει. Τα εγγενή προγράμματα χρησιμοποιούν επίσης τυχαιοποίηση στη διάταξη του χώρου διευθύνσεων για να κάνουν πιο δύσκολο σε έναν εισβολέα να στοχεύσει μια συγκεκριμένη διεύθυνση μνήμης, όπως φαίνεται στην Εικόνα 3.3 παρακάτω. Ενώ η διάταξη της μνήμης του WebAssembly μπορεί να συναχθεί από τον μεταγλωττιστή και το πρόγραμμα

Εικόνα 3.3: Τυχαία διάταξη μνήμης [56]



3.4 JavaScript API και Web API

Το API της JavaScript, αποτελεί μία από τις πιο υποσχόμενες μονάδες σύγκρισης που μελετήθηκε στην έρευνα. Το παρόν API της JavaScript επιτρέπει την πρόσβαση στις μονάδες της WebAssembly μέσω της JavaScript. Ακόμα ορίζει τις κλάσεις και τα αντικείμενα (Objects and Classes) της JavaScript, όπως και τις μεθόδους επικύρωσης, μεταγλώττισης και της δημιουργίας στιγμιότυπου (Validation, Compilation, Instantiation), όπως θα δούμε συγκεντρωτικά στο παρακάτω πίνακα. Η WebAssembly παρέχει το δικό της API το οποίο επεκτείνει αυτό της JavaScript και για παράδειγμα ορίζει την συνάρτηση που χρησιμοποιήθηκε στην πειραματική μας μελέτη InstantiateStreaming. Το Web API ενσωματώνει την WebAssembly στον αντίστοιχο φυλλομετρητή [52,28]

Αντικείμενο (object) ή Συνάρτηση	**Περιγραφή**
Καθολικές (Global)	Καθολικές μεταβλητές προσβάσιμες από το wasm και JavaScript
Μονάδα (Module)	Περιέχει κώδικα (stateless) σε μορφή wasm, έχει υποστεί μεταγλώττιση
Instance	Εκτελέσιμο στιγμιότυπο μονάδας (stateful)
InstantiateStreaming	Συνάρτηση για την μεταγλώττιση και στιγματοποίηση wasm κώδικα
Μνήμη	Μνήμη (ArrayBuffer) με δυνατότητα αλλαγής μεγέθους. Περιέχει raw bytes.
Πίνακες	Πίνακες με τη δυνατότητα αλλαγής μεγέθους (Typed Arrays). Για παράδειγμα uint8array, uint32array, float64array

Πίνακας 3.1: Ενσωμάτωση JavaScript σε wasm, οι πιο χρησιμοποιούμενες αναφορές

Παρακάτω εξηγείται με παράδειγμα, ο τρόπος με τον οποίο φορτώνεται η μονάδα (module) της WebAssembly. Οι κώδικες της παρακάτω εικόνας, πραγματοποιούν την επιστροφή ενός ακέραιου αριθμού. Ο πρώτος κώδικας περιέχει την συνάρτηση main όπου δεχεται σαν όρισμα την ακέραια μεταβλητή num και στη συνέχεια την επιστρέφει. Ο δεύτερος κώδικας είναι σε μορφή WAT, δηλαδή WebAssembly Text και είναι το αποτέλεσμα της μετατροπής του κώδικα C (όπως εξηγήθηκε στην ενότητα 3.1). Ο λόγος που το αρχείο είναι σε μορφή .wat είναι διότι μπορεί να αναγνωσθεί από τον προγραμματιστή σε αντίθεση με τα αρχεία .wasm τα οποία είναι σε δυαδικά μορφή και μπορούν να γίνουν αντιληπτά μόνο από τη γλώσσα μηχανής

Κώδικας γραμμένος σε C και WASM (σε μορφή WAT), για την επιστροφή ακέραιου αριθμού

Εικόνα 3.4: Κώδικας γραμμένος σε C για την επιστροφή ακέραιου αριθμού

```
int main(int num) {
return num;
}
```

Εικόνα 3.5: Κώδικας γραμμένος σε WASM (σε μορφή WAT), για την επιστροφή ακέραιου αριθμού

```
;;  WAT (WebAssembly Text)
(module
 (table 0 anyfunc)
 (memory $0 1)
 (export "memory" (memory $0))
 (export "main" (func $main))
 (func $main (param $0 i32) (result i32)
  (get_local $0)
 )
)
```

Η μονάδα (module) που δημιουργήθηκε παραπάνω, μπορεί να φορτωθεί στην JavaScript με την κλήση της συνάρτησης “WebAssembly.InstantiateStreaming()”. Αυτός είναι ο πιο αποδοτικός και βελτιστοποιημένος τρόπος φόρτωσης κώδικα σε μορφή wasm [28]. Στη παρούσα έρευνα θα μελετήσουμε περισσότερο την συγκεκριμένη συνάρτηση ώστε να πετύχουμε την βέλτιστη απόδοση. Το παράδειγμα της εικόνας 3.5, εκτυπώνει την τιμή που

δίνεται σαν όρισμα στην συνάρτηση και επιστρέφεται από την συνάρτηση της WebAssembly, στη περίπτωση αυτή ο αριθμός που επιστρέφεται είναι το 111

Εικόνα 3.6: Κλήση της συνάρτησης InstantiateStreaming για την φόρτωση του module και την εκτύπωση της επιστρεφόμενης τιμής.

```
WebAssembly.instantiateStreaming(fetch('simple.wasm'))
  .then(obj =>
  console.log(obj.instance.exports.main(111) // print 111
));
```

Μια μονάδα (module) της WebAssembly μεταγλωττίζεται και προβάλλεται απευθείας από την συνάρτηση InstantiateStreaming. Εάν δεν είναι εφικτό να χρησιμοποιηθεί η μέθοδος της ροής (streaming) τότε χρησιμοποιείται η συνάρτηση WebAssembly.compile ή η "WebAssembly.instantiate"

Εικόνα 3.7: Φόρτωση WASM μονάδας με την συνάρτηση Instantiate

```
fetch('simple.wasm').then(response =>
  response.arrayBuffer()
  ).then(bytes =>
  WebAssembly.instantiate(bytes)
  ).then(results => {
  console.log(results.instance.exports.main(111));
  });
```

Οι συναρτήσεις μη ροής (non-streaming) δεν έχουν άμεση πρόσβαση στον byte code. Αυτό σημαίνει ότι η απόκριση (response) θα πρέπει πρώτα να μετατραπεί σε έναν ArrayBuffer πριν από την μεταγλώττιση ή την εγκατάσταση (instantiating) της μονάδας WASM. Στην παρακάτω εικόνα βλέπουμε το παράδειγμα της συγκεκριμένης περίπτωσης.[24]
Η μνήμη έχει τη δυνατότητα της αλλαγής μεγέθους και είναι σε μορφή ArrayBuffer ή SharedArrayBuffer και δημιουργείται μέσω του κατασκευαστή WebAssembly.Memory. Οι παράμετροι του κατασκευαστή είναι το αρχικοποιημένο το μέγιστο μέγεθος. Η μνήμη χωρίζεται σε σελίδες και κάθε σελίδα αντιστοιχεί σε 64 KB [29]

Εικόνα 3.8: Διαχείριση μνήμης WebAssembly στην JavaScript

```
let memory = new WebAssembly.Memory({ initial: 10, maximum: 100});
WebAssembly.instantiateStreaming(fetch("simple.wasm"))
    .then(obj => {
        new Uint32Array(memory.buffer)[0] = 111;
        console.log(new Uint32Array(memory.buffer)[0]); //print 111
        memory.grow(1); // 1 x 64KB (page)
    });
```

Στο παραπάνω απόσπασμα του κώδικα, έχουν αρχικοποιηθεί 10 σελίδες στη μνήμη. Οπότε η μνήμη μας είναι 640KB. Ο ακέραιος αριθμός 111 γράφεται στη πρώτη θέση της γραμμικής μνήμης και μπορεί να διαβαστεί από τη θέση Uint32Array(memory.buffer)[0]. Για να επεκτείνουμε την χωρητικότητα της μνήμης χρησιμοποιούμε τη συνάρτηση "Memory.prototype.grow(parameter)" όπου σαν παράμετρο δίνεται ο αριθμός των σελίδων.

Οι πίνακες είναι ένα εργαλείο που χρησιμοποιεί η WebAssembly για την αποθήκευση αναφορών σε συναρτήσεις. Ο κώδικας της εικόνας 3.7 ορίζει έναν πίνακα με δύο στοιχεία. Τα στοιχεία αυτά είναι δύο αναφορές στις συναρτήσεις "ten" και "twenty". Όπου οι συναρτήσεις τυπώνουν τον αριθμό 10 και 20 αντίστοιχα και μπορούν να ανακτηθούν από την JavaScript με την εντολή tbl.get(0) όπως μπορούμε να παρατηρήσουμε στο δεύτερο μέρος του κώδικα. [30]

Εικόνα 3.9: Πίνακες της WebAssembly σε JavaScript

```
(module
    (func $ten (result i32) (i32.const 10))
    (func $twenty (result i32) (i32.const 20))
    (table (export "table1") anyfunc (elem $ten $twenty))
)
    let table1 = results.instance.exports.table1;
    console.log(table1.get(0)()); // 10
    console.log(table1.get(1)()); // 20
    let table2 = new WebAssembly.Table({initial:1, element:"anyfunc"});
    table2.set(0, table1.get(0)); //sets "ten" to table2
    console.log(table2.get(0)); // 10
```

Το WebAssembly.Table δημιουργείται με τα στοιχεία "element" και "initial". Το element αντιπροσωπεύει τον τύπο της τιμής που θα αποθηκευτεί στον πίνακα μας, όπου το "anyfunc" είναι η μόνη αποδεκτή τιμή και το initial ορίζει τον αριθμό των στοιχείων του πίνακα. [30]. Η συνάρτηση set του πίνακα, θέτει μια εξαγόμενη συνάρτηση στον table2.

3.5 Βιβλιογραφική Επισκόπηση

Η WebAssembly είναι σχετικά μια καινούργια τεχνολογία που εξελίσσεται συνεχώς. Τα τελευταία τρία χρόνια παρατηρείται όλο και μεγαλύτερο ενδιαφέρον από ερευνητές του χώρου, καθώς αποτελεί σημαντικό θεμέλιο για την επέκταση του οικοσυστήματος στο διαδίκτυο αλλά και για την βελτιστοποίηση ήδη υπάρχον υλοποιήσεων. Επίσης μεγάλο ενδιαφέρον εκδηλώνεται από τους προγραμματιστές ανά τον κόσμο, όπου εκμεταλλεύονται το γεγονός της υλοποίησης εφαρμογών σε διαφορετικές γλώσσες προγραμματισμού, μεταγλωττίζοντας τα σε αρχεία .wasm, κοντεύοντας πολλές φορές τις πρωταρχικές ταχύτητες (Native Speeds). Στην συγκεκριμένη υποενότητα θα αναδείξουμε παρόμοιες έρευνες και συγκριτικές μελέτες ώστε να αντλήσουμε τα χρήσιμα συμπεράσματα τους. Αναδεικνύοντας την βιβλιογραφία, θα προσπαθήσουμε να καλύψουμε ένα κενό που ίσως υπάρχει, το οποίο μας οδήγησε στη παρούσα έρευνα.

Στην έρευνα του Sandhu όπου παρουσιάστηκε το 2018, πραγματοποιήθηκε η σύγκριση της απόδοσης JavaScript και WebAssembly σε σχέση με τη γλώσσα C. Τα αποτελέσματα της έρευνας απέδειξαν ότι η native C είναι από 2.2x έως και 5.6x φορές πιο γρήγορη από την JavaScript. Όταν συγκρίθηκε η WebAssembly με την Native C αποδείχθηκε ότι οι χρόνο διεκπεραίωσης είναι πολύ κοντινοί μεταξύ τους, και σε κάποιες περιπτώσεις να είναι καλύτερη η wasm. Οι συγκριτικοί αλγόριθμοι ήταν πολλαπλασιασμοί διανυσματικών αραιών πινάκων, όπου εκτελέστηκαν σε περιβάλλον φυλλομετρητή και μεταγλωττίστηκαν από την Emscripten. Μέσω της Emscripten δημιουργήθηκαν οι μονάδες (modules) της WebAssembly. [41]

Πάλι οι Sandhu, P., Verbrugge, C., & Hendren, L. πραγματοποιήσαν μια έρευνα το (2020), όπου συγκρίνεται το πρότυπο WebAssembly με την native C, χρησιμοποιώντας πάλι αραιούς υπολογισμούς πινάκων. Ωστόσο, οι υλοποιήσεις της WebAssembly αναπτύχθηκαν με το χέρι και όχι χρησιμοποιώντας μεταγλωττιστή. Αυτή τη φορά ανακάλυψαν ότι όταν η WebAssembly εκτελείται στο πρόγραμμα περιήγησης Chrome έχει προβλήματα απόδοσης λόγω της διευθέτησης της μνήμης. Αυτό αφήνει ένα κενό στην έρευνα, αλλά θα ήταν σημαντικό να κατανοήσουμε εάν διάφοροι μεταγλωττιστές της WebAssembly θα μπορούσαν να αντιμετωπίσουν το πρόβλημα. [42]

Οι Herrera, D., Chen, H., Lavoie, E., & Hendren, L (2018) αξιολόγησαν την απόδοση της JavaScript και της WebAssembly σε σύγκριση με την native C. Χρησιμοποίησαν το εργαλείο Ostrich Benchmark Suite συγκριτικής αξιολόγησης (Khan et al., 2015).Οι αλγόριθμοι είχαν να κάνουν με αριθμητικούς υπολογισμούς και χρησιμοποιήθηκε το περιβάλλον της Emscripten για να δημιουργηθούν οι μονάδες (modules) της WebAssembly. Οι συγκρίσεις τους, έγιναν μεταξύ προγραμμάτων περιήγησης ιστού, συσκευών του διαδικτύου των πραγμάτων και το Node JS. Τα ευρήματά τους δείχνουν για άλλη μια φορά ότι το WebAssembly πλησιάζει τις native επιδόσεις της C, και είναι επίσης ταχύτερη από την JavaScript. [21]

Ο Denis Eleskovic για το Blekinge Institute of Technology (2020) πραγματοποίησε συγκριτική μελέτη ανάμεσα σε WebAssembly και JavaScript σε hot και warm φυλλομετρητές με διαφορετικά λειτουργικά συστήματα. Τα αποτελέσματα της σύγκρισης απέδειξαν ότι η JavaScript προσφέρει καλύτερη απόδοση στις πράξεις που έχουν να κάνουν με την επεξεργασία πινάκων, ενώ η WebAssembly στους υπολογιστικούς υπολογισμούς.

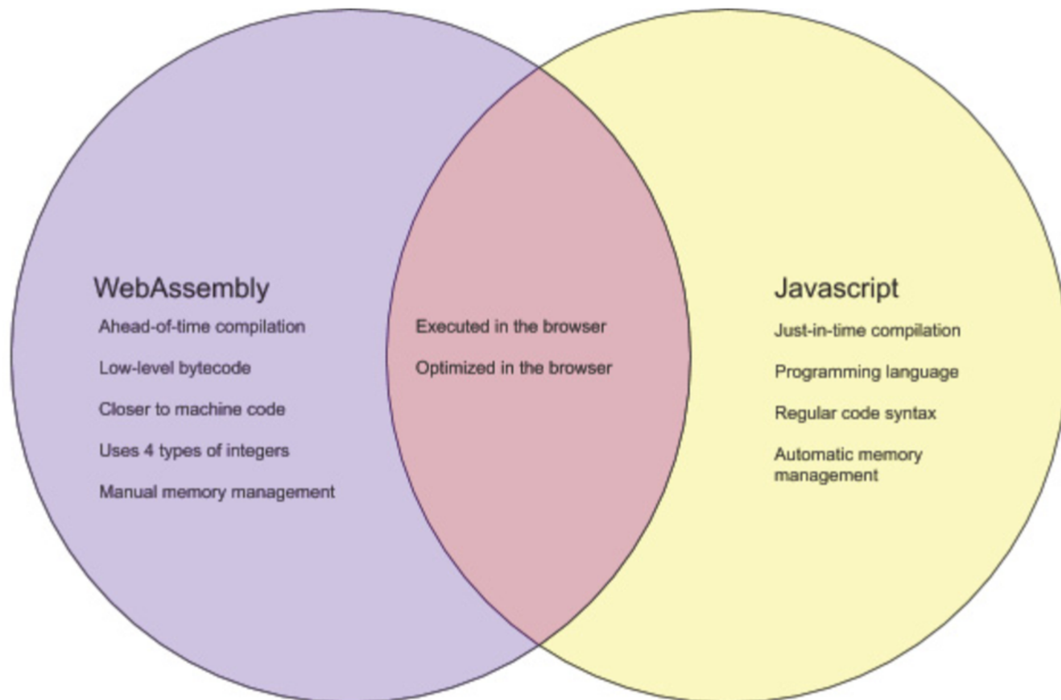
Κάθε φυλλομετρητής συμπεριφέρεται διαφορετικά στη βελτιστοποίηση του κώδικα, κάποιες μηχανές πλεονεκτούν σε συγκεκριμένες διεργασίες. Επίσης αποδείχθηκε ότι η WebAssembly παρέχει μεγαλύτερη σταθερότητα (consistency) για τις πράξεις με καθαρούς υπολογισμούς, με ορισμένες εξαιρέσεις ανάλογα με το πρόγραμμα περιήγησης, το λειτουργικό σύστημα και τον τύπο εκτέλεσης.

3.6 Σύνοψη

Η JavaScript έχει σχεδιαστεί κυρίως για να εκτελείται απευθείας σε πρόγραμμα περιήγησης. Για να γίνει όσο το δυνατόν πιο γρήγορη χρησιμοποιεί μια σύνθετη αρχιτεκτονική πολλαπλών επιπέδων, προκειμένου να αναλύσει και να βελτιστοποιήσει τον κώδικα που εκτελείται. Η αρχιτεκτονική εξαρτάται από το πρόγραμμα περιήγησης στο οποίο εκτελείται ο κώδικας JavaScript, όπως ο Chrome που χρησιμοποιεί τη μηχανή V8 για τη μεταγλώττιση του κώδικα, ο Firefox τη μηχανή SpiderMonkey και ο Opera την Caravan. Σε γενικές γραμμές χρησιμοποιούν και οι τρεις διερμηνέα και μεταγλωττιστή. Ο κώδικας θα εκτελεστεί γρήγορα μέσω του διερμηνέα και στη συνέχεια θα αναλυθεί και θα βελτιστοποιηθεί με βάση τα δεδομένα προφίλ (profiling data) από τον μεταγλωττιστή, όλα αυτά πριν μεταφραστεί σε γλώσσα μηχανής που είναι κατάλληλη για το υποκείμενο μηχάνημα. Τα παραπάνω βήματα πραγματοποιούνται ενώ εκτελείται ο κώδικας, γι' αυτό ονομάζεται μεταγλώττισης Just-in-time.

Η WebAssembly αναπαρίσταται σε δυαδική μορφή και μεταγλωττίζεται από άλλες γλώσσες. Ως εκ τούτου, μεταγλωτίζεται με τρόπο ahead-of-time και οι βελτιστοποιήσεις που υφίσταται γίνονται πριν από τη λήψη της στο πρόγραμμα περιήγησης. Το γεγονός ότι αναπαρίσταται σε δυαδική μορφή παρέχει μια πιο κοντινή αντιστοίχιση σε κώδικα μηχανής και δεν χρειάζεται να αναλυθεί (parsed) όπως η απλή JavaScript. Οι μονάδες (modules) τείνουν να είναι μικρότερες σε μέγεθος, αυτό καθιστά πιο γρήγορη τη λήψη. Η τεχνολογία χρησιμοποιεί κυρίως 4 τύπους ακέραιων αριθμών οι οποίοι είναι κινητής υποδιαστολής (floating point). Για το λόγο αυτό κρίνεται κατάλληλη για καθαρούς υπολογισμούς. Ωστόσο, ενώ η τεχνολογία είναι σε θέση να παρέχει κώδικα σε μορφή χαμηλού επιπέδου, δεν είναι τόσο βελτιστοποιημένη ώστε να διαχειρίζεται περίπλοκες δομές δεδομένων, ούτε να δημιουργεί τμήματα του ιστότοπου. Επιπλέον η τεχνολογία χρησιμοποιεί μία στοίβα μηχανής ώστε να χειρίζεται τις εντολές, όπως αναφέρθηκε προηγουμένως η στοίβα λειτουργεί με την μέθοδο last in, fist out όπου μέχρις ότου να επιστραφεί η τιμή στη στοίβα πραγματοποιούνται οι λειτουργίες των εντολών. Η μνήμη που χρησιμοποιείται είναι γραμμική (linear memory), όπου μπορεί να αποθηκεύσει τιμές σε οποιοδήποτε byte της διεύθυνση της. Αυτό επίσης χρησιμοποιείται όταν απαιτείται η επικοινωνία μεταξύ JavaScript και WebAssembly, μια τέτοια περίπτωση είναι η ανταλλαγή δομών δεδομένων μεταξύ των τεχνολογιών η οποία θα φέρει επιπλέον επιβάρυνση στην εκτέλεση. Στο παρακάτω σχήμα έχουν συλλεχθεί κάποιες από τις βασικές διαφορές και ομοιότητες των δύο τεχνολογιών

Εικόνα 3.10: Διαφορές και ομοιότητες μεταξύ τεχνολογιών



4 Μεθοδολογία

Στο συγκεκριμένο κεφάλαιο, παρουσιάζονται τα αποτελέσματα των συγκρίσεων μεταξύ των τεχνολογιών που επιλέχθηκαν. Για το σκοπό αυτό εξετάστηκαν 12 διαφορετικοί αλγόριθμοι χωρισμένοι σε 3 κατηγορίες, οι οποίοι εκτελέστηκαν σε διαφορετικούς φυλλομετρητές και συσκευές. Για να συλλέξουμε μια ακριβής μέτρηση της απόδοσης μιας δεδομένης τεχνολογίας, θα πρέπει να προσέξουμε πολλούς παράγοντες οι οποίοι θα αναλυθούν παρακάτω. Αφού συγκεντρώσουμε τα δεδομένα των μετρήσεων τότε θα πρέπει να τα εκτελέσουμε σε διαφορετικούς διερμηνείς, λειτουργικά συστήματα αλλά και συσκευές. Αυτό είναι εξαιρετικά χρονοβόρο έργο σε σχέση με το χρονοδιάγραμμα της διπλωματικής, γι αυτό το λόγο επιλέχθηκε ένα μικρότερο εύρος τεχνολογιών και γενικότερα παραμέτρων εκτέλεσης. Αναλυτική περιγραφή θα συναντήσουμε στις παρακάτω υποενότητες.

4.1 Συγκριτική Αξιολόγηση και Συλλογή Αποτελεσμάτων

Η υποενότητα αυτή αποτελεί μια από τις σημαντικότερες καθώς καθορίζει σε μεγάλο βαθμό τα αποτελέσματα των συγκρίσεων. Προκειμένου να ληφθούν με ακρίβεια τα αποτελέσματα από τα πειράματα μας προστέθηκαν επαναλήψεις στις εκτελέσεις, διότι έτσι θα μπορέσουμε να υπολογίσουμε το γεωμετρικό μέσο του κάθε αλγορίθμου ώστε να γίνει πιο ολοκληρωμένα η σύγκριση μας. Επίσης, οι επαναλήψεις παίζουν σημαντικό ρόλο στο

φυλλομετρητή και συγκεκριμένα στο διερμηνέα, διότι έτσι καθορίζεται η κατάσταση του. Όταν ο φυλλομετρητής εκτελεί για πρώτη φορά μια συνάρτηση τότε η κατάσταση του είναι "cold" αυτό σημαίνει ότι δεν υπάρχει καμία βελτιστοποίηση στον κώδικα που εκτελέστηκε, επομένως ο JIT δε χρησιμοποιήθηκε. Αντιθέτως όταν μια συνάρτηση εκτελείται επανειλημμένα τότε η κατάσταση του φυλλομετρητή θεωρείται "hot", που σημαίνει ότι ο JIT έχει βελτιστοποιήσει τον κώδικα μας και έχουμε σημαντικές μειώσεις στο χρόνο εκτέλεσης του αλγορίθμου. Στα πειράματα της παρούσας διπλωματικής εκμεταλευτήκαμε τον JIT και τρέξαμε όλους τους αλγορίθμους σε hot κατάσταση φυλλομετρητή, πράγμα που σημαίνει πως το πρόγραμμα περιήγησης διατηρήθηκε ανοιχτό σε κάθε πολλαπλή εκτέλεση αλγορίθμου. Για να καταλάβουμε όμως πόσο σημαντική είναι η διαφορά μεταξύ cold και hot περιηγητή και για να αντιληφθούμε αν επηρεάζεται η WebAssembly από την μηχανή της JavaScript, εφαρμόσαμε ενδεικτικά κάποιους αλγορίθμους σε cold περιβάλλον που θα δούμε παρακάτω. Οι φυλλομετρητές που χρησιμοποιήθηκαν είναι ο Chrome, Firefox, Opera προκειμένου να εντοπίσουμε τις διαφορές που οφείλονται ανάμεσα στις διαφορετικές μηχανές JavaScript.

Οι χρόνοι των αποτελεσμάτων για την JavaScript και WebAssembly λήφθηκαν μέσω της συνάρτησης performance.now(). Είναι μια μέθοδος του Performance API της JavaScript που επιστρέφει ενα DOMHighResTimeStamp ("Χρονική σφραγίδα υψηλής ανάλυσης") το οποίο με απλά λόγια είναι μια χρονική τιμή σε χιλιοστά του δευτερολέπτου στο κλασματικό μέρος μεταξύ δύο χρονικών σημείων. Η χρήση της συνάρτησης performance.now() έγινε με την οριστικοποίηση της, πριν και μετά από την υπό εξέταση συνάρτηση. Τέλος για να βρούμε το χρόνο διεκπεραίωσης της εξεταζόμενης συνάρτησης αφαιρούμε την τελική performance.now() από την αρχική (ενδιάμεσα βρίσκεται η υπό εξέταση συνάρτηση) ώστε να βρούμε το τελικό χρόνο εκτέλεσης. Στην εικόνα παρακάτω βλέπουμε αναλυτικά τη χρήση της performance.now().

Εικόνα 4.1: JavaScript Performance API

```
var startTime = performance.now();
var newImageDataBuffer = wsImageConvolute(
    array0,
    array2,
    width,
    height,
    weights,
    wWidth,
    wHeight);
var endTime = performance.now();
elapsedTime = (endTime - startTime);
```

Με την ίδια λογική συλλέξαμε και τους χρόνους στους αλγόριθμους της Native C. Συγκεκριμένα χρησιμοποιήθηκε η συνάρτηση clock() της βιβλιοθήκης <time.h> που έχει παρόμοιο τρόπο λειτουργίας μ' αυτόν της performance.now(). Στη παρακάτω εικόνα παρουσιάζεται ο τρόπος με τον οποίο υλοποιήθηκε..

Εικόνα 4.2: Καταγραφή χρόνου διεκπεραίωσης σε C

```c
clock_t begin = clock();
for (it = 0; it < 10; it++){
    bubbleSort(my_array, n);
}
clock_t end = clock();
time_spent += (double)(end - begin) / CLOCKS_PER_SEC;
printf("The elapsed time is %f seconds", time_spent);
```

4.2 Περιβάλλον Εκτέλεσης Αλγορίθμων

Στα πλαίσια της διπλωματικής εργασίας κρίθηκε απαραίτητο να στηθεί ένα περιβάλλον στο οποίο θα εκτελεστούν οι συγκεκριμένοι αλγόριθμοι, σε συγκεκριμένες παραμετροποιήσεις, από διάφορους υπολογιστές αλλά και μεταγλωττιστές, προκειμένου να εξάγουμε όσο περισσότερα χρήσιμα αποτελέσματα και στη συνέχεια συμπεράσματα για τη συγκριτική μελέτη των τεχνολογιών μας. Επομένως στο κεφάλαιο αυτό θα παρουσιαστούν τα απαραίτητα εργαλεία που χρησιμοποιήθηκαν, η προετοιμασία και τα προβλήματα που προέκυψαν, οι αλγόριθμοι υψηλής υπολογιστικής ισχύος, οι διαφορετικές γλώσσες και βελτιστοποιήσεις των αλγορίθμων. Τα αποτελέσματα παρουσιάζονται σε πίνακες αλλά και διαγράμματα.

4.2.1 Προγράμματα και Εφαρμογές Περιβάλλοντος

Τα προγράμματα και οι εφαρμογές που χρησιμοποιήθηκαν για την συγκριτική μελέτη είναι τα παρακάτω:

- Για τη σύνταξη και την ανακατασκευή των αλγορίθμων χρησιμοποιήθηκε ο συντάκτης κώδικα Visual Studio Code με τα extensions (wasmerio, CMake, WABT).

- Η εκτέλεση των αλγορίθμων δεν πραγματοποιήθηκε σε online διακομιστή (web server) αλλά σε τοπικ,ό ώστε να μειωθούν οι πιθανές καθυστερήσεις της σύνδεσης του ίντερνετ οι οποίες επηρεάζουν αρνητικά τα αποτελέσματα και κυρίως στη φόρτωση του αρχείου (wasm). Το πρόγραμμα που χρησιμοποιήθηκε για την τοπική εκτέλεση είναι το XAMPP όπου χρειάστηκε να γίνουν παραμετροποιήσεις στο configuration file της εφαρμογής για να εκτελεστεί ομαλά η WebAssembly.

- Για την εκτέλεση των αλγορίθμων υλοποιημένων σε γλώσσα C και τη καταγραφή των ταχυτήτων εκτέλεσης τους, χρησιμοποιήθηκαν τα προγράμματα Code::Blocks, Visual Studio 2019, και ο compiler gcc μέσω του MinGW-w64.

- Για την εκτέλεση των αλγορίθμων σε διαφορετικούς υπολογιστές (Desktop, Laptop, Mobile) χρησιμοποιήθηκε η σήραγγα ngrok σε συνδυασμό με το XAMPP.

- Για την δημιουργία των γραφημάτων (Bar Chart) χρησιμοποιήθηκε το Excel

- Οι Browsers είναι ο Google Chrome (v. 102.0.5005.63), Mozilla Firefox (v. 101.0.1), Opera (v. 87.0.4390.45)

4.2.2 Υπολογιστικός Εξοπλισμός

Οι υπολογιστές που χρησιμοποιήθηκαν παρουσιάζονται παρακάτω με τα εξείς χαρακτηριστικά:

1. Desktop (Windows 10 pro)
    α. CPU: AMD Ryzen 5 3600X - 3.8GHz
    β. RAM: 16,0GB - 1596 MHz
2. Laptop (Windows 10 pro)
    α. CPU: INTEL Core i5 7200U - 2.50GHz
    β. RAM: 8 GB - 1064 MHz
3. Mobile (IOS 15.5)
    α. CPU: A13 Bionic Chip - 2.6 GHz
    β. RAM: 6 GB

4.3 Αλγόριθμοι Υψηλής Υπολογιστικής Ισχύος

Οι αλγόριθμοι που χρησιμοποιήθηκαν για τη συγκριτική μελέτη της παρούσας διπλωματικής, συλλέχθηκαν από το διαδίκτυο. Οι περισσότεροι από αυτούς χρειάστηκε να υποστούν παραμετροποίηση και να υλοποιηθούν από την αρχή ώστε να βελτιστοποιηθούν και να υπάρξει ο ίδιο βαθμός πολυπλοκότητας σε όλες τις εκτελέσεις αλλά και για να αποφευχθούν σφάλματα μνήμης των διακομιστών. Οι αλγόριθμοι εκτελέστηκαν σε διαφορετικούς μεταγλωττιστές και διερμηνείς.

Η επιλογή τους χωρίστηκε σε τρεις κατηγορίες, οι οποίες είναι:

- Αλγόριθμοι αριθμητικών υπολογισμών
- Αλγόριθμοι ταξινόμησης
- Αλγόριθμοι επεξεργασία εικόνας

Στην πειραματική μας μελέτη, κάποιοι από τους αλγορίθμους έχουν διαφορετικό τύπο μεταβλητών (double, integer) ώστε να δούμε πως συμπεριφέρεται κάθε γλώσσα στην εκάστοτε μεταβλητή. Επίσης υπάρχουν πράξεις με πίνακες οι οποίοι έχουν πάνω από 100,000

στοιχεία. Για να καταφέρουμε να μεγαλώσουμε τις απαιτήσεις των αλγορίθμων ως προς τον επεξεργαστή έχουμε φροντίσει κάθε αλγόριθμος να εκτελείται πολλαπλές φορές, συνήθως ο αριθμός των επαναλήψεων ξεπερνάει τις 50. Ένας ακόμα λόγος για την προσθήκη επαναλήψεων είναι για την βελτιστοποίηση που μας παρέχει ο JIT compiler, καθώς όσο περισσότερες φορές εκτελείται μια συνάρτηση τόσο μεγαλύτερη πιθανότητα βελτιστοποίησης υπάρχει. Έτσι μετατρέπεται και η κατάσταση του περιηγητή, από cold σε warm και hot. Οι αλγόριθμοι που επιλέχθηκαν στην παρούσα μελέτη είναι οι κατάλληλοι για μια συγκριτική μελέτη, καθώς έχουν επιλεχθεί και σε άλλες παρόμοιες έρευνες.

4.3.1 Αλγόριθμοι Αριθμητικών Υπολογισμών

Οι αριθμητικοί αλγόριθμοι που χρησιμοποιήθηκαν για την συλλογή των δεδομένων είναι η ακολουθία Fibonacci, ο πολλαπλασιασμός διπλής ακρίβειας (DOUBLE), πολλαπλασιασμός ακεραίων, πολλαπλασιασμός πινάκων, υπολογισμός πρώτων αριθμών.

4.3.1.1 Ακολουθία Fibonacci

Η ακολουθία Fibonacci είναι μια πολύ γνωστή υπολογιστική πράξη η οποία χρησιμοποιείται συχνά σε συγκριτικές αξιολογήσεις. Η ακολουθία παίρνει έναν αριθμό ως παράμετρο εισόδου και προσθέτει τους δύο προηγούμενους αριθμούς για να παραχθεί η επόμενη αριθμητική ακολουθία όπου θα χρησιμοποιηθεί σαν νέα είσοδος. Η υλοποίηση έγινε με την αναδρομική (recursive) μέθοδο. Ο αλγόριθμος υπολογίζει όλους τους αριθμούς Fibonacci από το 3 μέχρι το 45. Ο λόγος που υπολογίζουμε από το 3 και όχι από το 0 είναι διότι ο Fibonacci του αριθμού 1 και 2 είναι το 1. Ο αλγόριθμος έχει την ίδια πολυπλοκότητα σε όλες τις γλώσσες που χρησιμοποιούμε στην συγκριτική αξιολόγηση.

4.3.1.2 Πολλαπλασιασμός Διπλής Ακρίβειας (DOUBLE)

Στον πολλαπλασιασμό αριθμών διπλής ακρίβειας υλοποιήθηκε μια συνάρτηση η οποία υπολογίζει το γινόμενο τριών αριθμών που αποτελούνται από 7 ψηφία. Η συνάρτηση αυτή καλείται 500 φορές και εκτελείται 3,000,000 φορές. Η συνάρτηση έχει την ίδια πολυπλοκότητα σε όλες τις γλώσσες που χρησιμοποιούμε στην συγκριτική αξιολόγηση.

4.3.1.3 Πολλαπλασιασμός Ακεραίων Αριθμών (INT)

Στον πολλαπλασιασμό ακεραίων αριθμών υλοποιήθηκε μια συνάρτηση η οποία υπολογίζει το γινόμενο τριών αριθμών που αποτελούνται από 7 ψηφία. Η συνάρτηση αυτή καλείται 500 φορές και εκτελείται 3,000,000 φορές. Η συνάρτηση έχει την ίδια πολυπλοκότητα σε όλες τις γλώσσες που χρησιμοποιούμε στην συγκριτική μας αξιολόγηση. 15 x 10^8

4.3.1.4 Πολλαπλασιασμός Πινάκων

Στον πολλαπλασιασμό πινάκων έχουμε δημιουργήσει μια συνάρτηση η οποία υπολογίζει το γινόμενο δύο πινάκων και το αποθηκεύει σε ένα τρίτο πίνακα. Οι Πίνακες αποτελούνται από 70000 στοιχεία με τυχαίους αριθμούς από το 0 έως το 10000, οι πράξεις επαναλαμβάνονται 9000 φορές. Η συνάρτηση έχει την ίδια πολυπλοκότητα σε όλες τις γλώσσες που χρησιμοποιούμε στην συγκριτική αξιολόγηση.

4.3.1.5 Πρώτοι Αριθμοί

Ένα σημαντικό θέμα στην επιστήμη των υπολογιστών και την κρυπτογραφία είναι η θεωρία των αριθμών, όπου ο υπολογισμός του πρώτου αριθμού παίζει σημαντικό ρόλο. Ένας πρώτος αριθμός είναι ο αριθμός που έχει μόνο δύο παράγοντες, το 1 και τον αριθμό εαυτό. Σε αυτό το πείραμα, ο αλγόριθμος θα υπολογίσει κάθε πρώτο αριθμό από το 2 έως το 200.000. Η συνάρτηση έχει την ίδια πολυπλοκότητα σε όλες τις γλώσσες που χρησιμοποιούμε στην συγκριτική αξιολόγηση.

4.3.2 Αλγόριθμοι Ταξινόμησης

Οι αλγόριθμοι ταξινόμησης αποτελούν αναμφισβήτητα σημαντικό κομμάτι στην επιστήμη των υπολογιστών, καθώς δίνουν λύση σε πολλούς κλάδους της πληροφορικής. Πολλές φορές χρησιμοποιούνται σε έρευνες και δημοσιεύσεις για την συγκριτική αξιολόγηση τεχνολογιών. Οι αλγόριθμοι που επιλέχθηκαν στην παρούσα έρευνα είναι ο αλγόριθμος φυσαλίδας (Bubble Sort), η ταχυταξινόμηση (Quick Sort) με μεταβλητές διπλής ακρίβειας αλλά και ακέραιους και τέλος η αντιστροφή πινάκων (Reverse array). Στους αλγορίθμους του πειράματος, πραγματοποιείται ταξινόμηση σε πίνακες μεγαλύτερους από 2000 στοιχεία, οι ταξινομήσεις επαναλαμβάνονται από 10 - 500 φορές. Κάθε αλγόριθμος έχει συγκεκριμένες τιμές σαν μεταβλητές εισόδου. Όταν αποπειράθηκε η εισαγωγή τυχαίων αριθμών μέσω της συνάρτησης random (γεννήτρια τυχαίων αριθμών) τα αποτελέσματα ήταν διαφορετικά σε κάθε εκτέλεση του αλγορίθμου διότι κάποιες φορές οι αριθμοί ήταν “ευνοϊκοί για την ταξινόμηση (best case scenario) και κάποιες καθόλου (worst case scenario).

4.3.2.1 Αλγόριθμος Φυσαλίδας

Ο αλγόριθμος φυσαλίδας συγκρίνει δύο γειτονικά στοιχεία του πίνακα και τα ανταλλάσσει (swap) θέσει αν δεν έχουν την απαιτούμενη σειρά. Στη συνάντηση μας δίνουμε σαν μεταβλητή εισόδου έναν πίνακα 5000 θέσεων, με συγκεκριμένα στοιχεία. Η ταξινόμηση πραγματοποιείται 10 φορές και έχει τον ίδιο βαθμό πολυπλοκότητας σε όλες της γλώσσες που συγκρίνεται.

4.3.2.2 Αλγόριθμος Ταχυταξινόμησης (Integers - Doubles)

Ο αλγόριθμος ταχυταξινόμησης είναι ένας αναδρομικός αλγόριθμος και χρησιμοποιεί την μέθοδο του «διαίρει και κυρίευε». Επιλέγει ένα στοιχείο ως άξονα (pivot) και χωρίζει τον δεδομένο πίνακα γύρω από τον επιλεγμένο άξονα. Υπάρχουν πολλές διαφορετικές εκδόσεις της ταχυταξινόμησης που επιλέγουν το pivot με διαφορετικούς τρόπους. Στη συνάρτηση μας δίνουμε μεταβλητή εισόδου ένα πίνακα 2000 θέσεων, με συγκεκριμένα στοιχεία διπλής ακρίβειας στη μια περίπτωση και ακέραιων στη δεύτερη. Η ταξινόμηση πραγματοποιείται 100 φορές και έχει τον ίδιο βαθμό πολυπλοκότητας σε όλες της γλώσσες που συγκρίνεται.

4.3.2.3 Αλγόριθμος Αντιστροφής Πινάκων

Ο αλγόριθμος αντιστροφής πινάκων χρησιμοποιείται για την αντιστροφή ενός πίνακα. Δηλαδή το στοιχείο που βρίσκεται στη τελευταία θέση του πίνακα θα ρθεί πρώτο και το στοιχείο που βρίσκεται στη πρώτη θέση θα πάει στην τελευταία, έτσι θα ακολουθήσουν και τα υπόλοιπα στοιχεία του πίνακα. Να διευκρινιστεί ότι η ταξινόμηση δε πραγματοποιήθηκε με την συνάρτηση που μας παρέχει η JavaScript, την reverse αλλά με custom κώδικα ώστε να μπορέσουμε να έχουμε τον ίδιο βαθμό πολυπλοκότητας σε όλες τις γλώσσες. Τα στοιχεία του πίνακα είναι 50000 και η ταξινόμηση πραγματοποιείται 500 φορές.

4.3.3 Αλγόριθμοι Επεξεργασίας Εικόνας

Οι αλγόριθμοι επεξεργασίας εικόνας απαιτούν μεγάλη υπολογιστική ισχύ λόγω της μετατροπής των πληροφοριών της εικόνας σε δισδιάστατους ή τρισδιάστατους πίνακες, τις πράξεις μεταξύ αυτών την ταξινόμηση στοιχείων αλλά και διαφόρων μαθηματικών πράξεων. Αυτοί είναι κάποιοι από τους λόγους που επιλέχθηκαν για την συγκεκριμένη έρευνα. Οι αλγόριθμοι που χρησιμοποιήθηκαν είναι ο αυτός της δημιουργίας ασπρόμαυρης εικόνας, της συνέλιξης και της επεξεργασίας εικόνας βάση κατωφλίου (threshold).

4.3.3.1 Αλγόριθμος Ασπρόμαυρης Εικόνας (Grayscale)

Ο αλγόριθμος Grayscale είναι από του πιο διαδεδομένους αλγορίθμους στο χώρο της επεξεργασίας εικόνων και έχει ώς αποτέλεσμα την δημιουργία μιας νέας ασπρόμαυρης φωτογραφίας. Κάθε pixel της εικόνας μετατρέπεται σε 8-bit unsigned integer όπου το κάθε ένα αντιπροσωπεύει τις πληροφορίες έντασης φωτός. Δηλαδή η εικόνα περιέχει μόνο μαύρο, λευκό και γκρι χρώμα σε διάφορες αποχρώσεις (0-255). Η εικόνα που επεξεργαζόμαστε στη μελέτη μας έχει διαστάσεις 512x512 το μέγεθος της είναι 112kb ο τύπος της JPG και η επεξεργασία επαναλαμβάνεται για 50 φορές. Ο Αλγόριθμος έχει την ίδια πολυπλοκότητα για όλες τις γλώσσες που υλοποιήθηκε.

4.3.3.2 Αλγόριθμος Συνέλιξης Εικόνας (Convolution)

Η συνέλιξη αποτελεί μια πράξη κοστοβόρα για τον επεξεργαστή του συστήματος καθώς περιέχει πράξεις με επιπλέον μάσκα, πολλαπλασιασμούς και διάφορες πολύπλοκες

συναρτήσεις. Πιο αναλυτικά, αν θεωρήσουμε ότι έχουμε μια εικόνα f(x,y) και μια μάσκα k(it, ij) 6x6 τότε ο υπολογισμός της συνέλιξης για το (x0, y0) θα πραγματοποιηθεί με τα παρακάτω βήματα:

- Το pixel που βρίσκεται στο κέντρο της μάσκας k θα τοποθετηθεί πάνω από τα pixel του σημείου (x0, y0) της εικόνας
- Πραγματοποιείται υπολογισμός στο άθροισμα των γινομένων και στις δυο διαστάσεις του πίνακα.
- Το από άθροισμα των γινομένων θα αποθηκευτεί σε νέα εικόνα fnew στο σημείο (pixel) του (x0, y0)
- Στη συνέχεια η μάσκα μετατοπίζεται ώστε το pixel που βρίσκεται στο κέντρο να βρεθεί πάνω από το επόμενο pixel της αρχικής εικόνας F.

Η εικόνα που επεξεργαζόμαστε στη μελέτη μας έχει διαστάσεις 512x512 το μέγεθος της είναι 48kb ο τύπος της JPG και η επεξεργασία επαναλαμβάνεται για 10 φορές. Ο Αλγόριθμος της συνέλιξης έχει την ίδια πολυπλοκότητα για όλες τις γλώσσες που υλοποιήθηκε.

4.3.3.3 Αλγόριθμος Προσαρμοστικού Κατωφλιού Εικόνας (Threshold)

Ο αλγόριθμος του κατωφλιού είναι ο απλός τρόπος της τμηματοποίησης των pixel της εικόνας (segmentation) σε σκοτεινά και φωτεινά. Ο αλγόριθμος εισάγει μια νέα εικόνα που αντιπροσωπεύει το segmentation. Για κάθε pixel στην εικόνα πρέπει να υπολογιστεί ένα κατώφλι. Εάν η τιμή του pixel είναι κάτω από το κατώφλι, τότε ορίζεται στην τιμή φόντου, διαφορετικά λαμβάνει την τιμή προσκηνίου. Η εικόνα που επεξεργαζόμαστε στη μελέτη μας έχει διαστάσεις 512x512 το μέγεθος της είναι 112kb ο τύπος της JPG και η επεξεργασία επαναλαμβάνεται για 20 φορές.

4.4 Γλώσσες Προγραμματισμού Συγκριτικής Μελέτης

Στην υποενότητα αυτή, αναφέρονται οι γλώσσες προγραμματισμού και οι παραμετροποιήσεις που πραγματοποιήθηκαν σ’ αυτές, ώστε να είναι αντιληπτός ο λόγος που χρησιμοποιήθηκαν:

1. **Native JavaScript**: Ο αλγόριθμος υλοποιήθηκε σε απλή JavaScript.
2. **WebAssembly Emscripten**: Ο αρχικός αλγόριθμος υλοποιήθηκε σε γλώσσα C και μεταγλωττίστηκε από την Emscripten ώστε να φτάσει στη μορφή wasm. Ο μεταγλωττιστής δημιούργησε το κώδικα σε JavaScript (glue code) που είναι απαραίτητος για την “επικοινωνία” της JavaScript με τον κώδικα wasm και στη συνέχεια με τις μονάδες (modules) του.
3. **WASM JavaScript API**: Ο αρχικός αλγόριθμος υλοποιήθηκε σε γλώσσα C και μετατράπηκε σε WASM με τη χρήση του online μετατροπέα WasmFiddle. Η συνάρτηση που χρησιμοποιείται είναι η InstantiateStreaming() που μεταγλωττίζει και

δημιουργεί το WebAssembly module ταυτόχρονα. Η συγκεκριμένη υλοποίηση γράφτηκε από την αρχή για κάθε αλγόριθμο ανάλογα με τις ανάγκες του (δέσμευση μνήμης, δημιουργία πινάκων).

4. **Native C**: O κώδικας είναι γραμμένος σε γλώσσα C και εκτελέστηκε μέσω του gcc compiler
5. **WebAssembly Emscripten (O3)**: Πρόκειται για την ίδια υλοποίηση με το νούμερο 2, με τη διαφορά ότι ο κώδικας βελτιστοποιείται και μειώνεται σε μέγεθος.
6. **WebAssembly Emscripten (Oz)**: Παρομοίως με το βήμα 5 με τη διαφορά ότι η ο κώδικας μειώνεται ακόμα περισσότερο σε μέγεθος και επηρεάζεται το glue code της JavaScript αλλά και το αρχείο WASM, με αυτή τη βελτιστοποίηση έχουμε μεγαλύτερο χρόνο στη μεταγλώττιση.
7. **Native JavaScript (WebWorker)**: O αλγόριθμος είναι γραμμένος σε απλή JavaScript και εκτελείται σε διαφορετικό νήμα του επεξεργαστή με τη χρήση WebWorker. Να διευκρινιστεί πως δεν υπάρχει παραλληλισμός εσωτερικά του αλγόριθμου και ότι το νήμα που χρησιμοποιείται εκτελεί εξ ολοκλήρου τον αλγόριθμο. Η χρήση του WebWorker γίνεται για μη επηρεάζεται το νήμα από άλλες διεργασίες του περιηγητή.

4.5 Αποτελέσματα

4.5.1 Εισαγωγή

Αφού έχει ολοκληρωθεί η βιβλιογραφική επισκόπηση, η αναφορά των απαραίτητων προγραμμάτων, όπως και οι παράμετροι που χρησιμοποιούμε για την συγκριτική μας μελέτη, θα παρουσιαστούν τα αποτελέσματα των συγκρίσεων σε πίνακες αλλά και γραφήματα (Bar chart). Τα αποτελέσματα που θα δούμε παρακάτω στους πίνακες περιέχουν το γεωμετρικό μέσο από 30 αποτελέσματα (εκτελέσεις) ανά αλγόριθμο, το ίδιο ισχύει και για τα διαγράμματα. Στην τελευταία σειρά του κάθε πίνακα αναγράφεται ο συνολικός χρόνος γεωμετρικός μέσος για κάθε φυλλομετρητή. Τα κελιά του πίνακα που είναι χρωματισμένα με πράσινο, δηλώνουν ότι το αποτέλεσμα έχει τον μικρότερο χρόνο διεκπεραίωσης. Οι αλγόριθμοι που εκτελέστηκαν σε C δεν χρωματίζονται με πράσινο χρώμα, καθώς η σύγκριση πραγματοποιείται ανάμεσα σε WASM και JavaScript.

4.5.2 Κατάσταση Cold - Warm - Hot

Όπως εξηγήσαμε στο δεύτερο κεφάλαιο, ο μεταγλωττιστής JIT (just in time), αναλαμβάνει τη μεταγλώττιση του κώδικα κατά τη διάρκεια της εκτέλεσης του. Ο μεταγλωττιστής JIT μπορεί να χωριστεί σε διαφορετικά επίπεδα βελτιστοποίησης. Μπορούμε να θεωρήσουμε αυτά τα τρια επίπεδα βελτιστοποίησης σε Cold, Warm και Hot, με το Cold να είναι το χαμηλότερο και το Hot το υψηλότερο επίπεδο. Η προκαθορισμένη κατάσταση του μεταγλωττιστή είναι η Cold και όσο εκτελείται μια συνάρτηση στον μεταγλωττιστή τότε

πραγματοποιείται επιπλέον βελτιστοποίηση και μεταβαίνει στο επίπεδο Warm, το ίδιο ισχύει και για το τελικό επίπεδο Hot. Στη συγκριτική μας μελέτη εκτελέσαμε τους αλγορίθμους σε Hot περιβάλλον, όμως για να αντιληφθούμε τη διαφορά μεταξύ επιπέδων δημιουργήθηκε ο παρακάτω πίνακας όπου αναδεικνύει τις διαφορές στις εκτελέσεις των αλγορίθμων: ακολουθίας Fibonacci, πολλαπλασιασμού διπλής ακρίβειας, ταξινόμηση φυσαλίδας και αντιστροφής πίνακα, στην πρώτη και στη τελευταία κατάσταση του φυλλομετρητή και πιο συγκεκριμένα του μεταγλωττιστή JIT.

Algorithm	Native JavaScript COLD	Native JavaScript HOT	WASM Emscripten COLD	WASM Emscripten HOT
Fibonacci Numbers	19.831	18.949	9.543	8.179
Multiply Doubles	2.317	2.129	5.210	4.998
BubbleSort	0.701	0.635	0.342	0.212
Reverse Array	0.710	0.630	0.102	0.086
Grayscale	0.048	0.040	0.071	0.069

Πίνακας 4.1: Παρουσίαση διαφορών μεταξύ Hot και Cold κατάστασης μεταγλωττιστή JIT

4.5.3 Αποτελέσματα Μετρήσεων σε Desktop

Στη παρακάτω ενότητα παρουσιάζονται τα αποτελέσματα της συγκριτικής μελέτης σε περιβάλλον Desktop.

4.5.3.1 Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Desktop με φυλλομετρητή Chrome (hot)

Ο γεωμετρικός μέσος (geometric mean) που υπολογίστηκε για κάθε αλγόριθμο της πρώτης κατηγορίας μετά τη πολλαπλή εκτέλεση των αλγορίθμων, υποδεικνύει ότι η WebAssembly υλοποιημένη με τη συνάρτηση InstantiateStreaming είναι πιο γρήγορη στις περισσότερες περιπτώσεις. Στο παράδειγμα της ακολουθίας Fibonacci παρατηρούμε πολύ μεγάλη διαφορά στη σύγκριση μεταξύ Native JavaScript και υλοποίησης με το χέρι WebAssembly (WASM API JavaScript), συγκεκριμένα η WASM είναι 6,4x φορές πιο γρήγορη από την JavaScript. Στο μόνο αριθμητικό αλγόριθμο που δείχνει καλύτερη η JavaScript, είναι αυτός του πολλαπλασιασμού πινάκων διπλής ακρίβειας (double), η διαφορά είναι μόλις 53μs, πάρα πολύ μικρή για να συμπεραίνουμε ότι έχει πλεονέκτημα στις συγκεκριμένες πράξεις, καθώς αυτή η διαφορά θα μπορούσε να οφείλεται σε παράπλευρες καθυστερήσεις που μπορεί να

συνέβησαν στον φυλλομετρητή κατά την εκτέλεση. Αυτό που παρατηρείται επίσης είναι η καθυστέρηση στις πράξεις με αριθμούς διπλής ακρίβειας (double) από τη μεριά της WASM, σε αντίθεση με την JavaScript που στη πραγματικότητα δεν έχει μεταβλητές διπλής ακρίβειας (DOUBLE). Οι WebWorkers δείχνουν να μην επηρεάζουν ιδιαίτερα τους χρόνους μας. Τέλος αξίζει να σημειωθεί ότι με τη χρήση της συνάρτησης που προαναφέρθηκε έχουμε καλύτερα αποτελέσματα από την Native C με εξαίρεση των υπολογισμό των πρώτων αριθμών, με την διαφορά 195μs.

Numerical Computing Algorithm	**Native JavaScript**	**WASM Emscripten**	**WASM JavaScript API**	**Native C**	**WASM Emscripten Optimised (O3)**	**WASM Emscripten Optimised (Oz)**	**Native JavaScript** (WebWorker)
Fibonacci Numbers	18.167	10.216	2.809	10.663	10.806	11.763	18.128
Multiply Doubles	2.152	4.872	2.039	5.121	2.164	2.154	2.142
Multiply Integers	2.056	3.089	1.082	4.679	1.094	1.074	2.146
Multiply 2 Double Arrays	0.629	2.173	0.682	2.342	1.046	1.026	0.629
Prime Numbers	5.945	6.096	5.925	5.647	5.981	5.988	5.842
Geometrical Mean	3.160	4.589	1.904	5.113	2.759	2.783	3.141
Geometrical Mean of Chrome	**2.955**						

Πίνακας 4.2 : Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Desktop με φυλλομετρητή Chrome

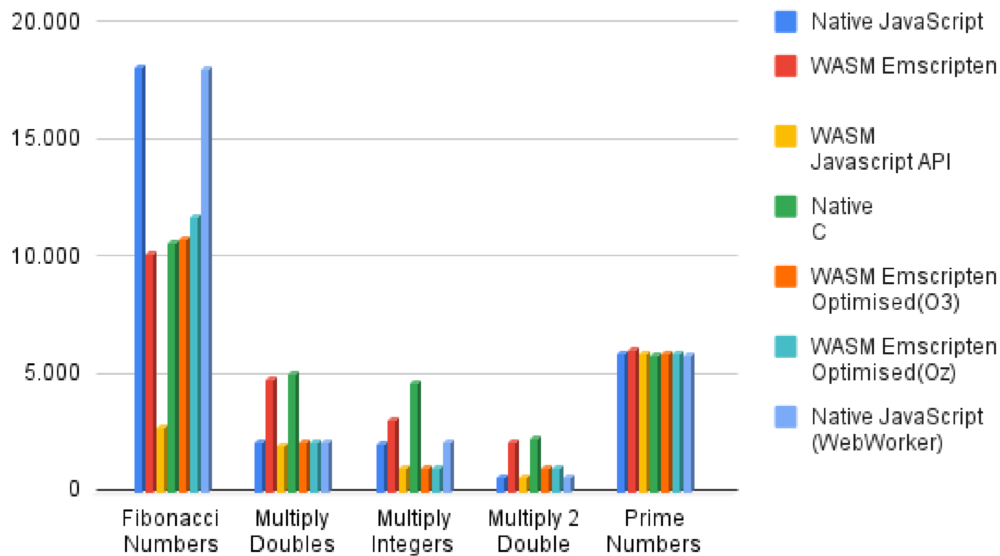
Browser**:** Chrome - State hot (V.102.0.5005.63)

CPU**:** AMD Ryzen 5 3600X - 3.8GHz

RAM**:** 16 GB - 1596MHz

Operating System: Windows 10 pro

Εικόνα 4-1: Γράφημα συγκρίσεων χρόνων εκτέλεσης αριθμητικών υπολογισμών σε Chrome



4.5.3.2 Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Desktop με φυλλομετρητή Firefox (hot)

Η συγκριτική ανάλυση που πραγματοποιήθηκε στο φυλλομετρητή Firefox έχει παρόμοια αποτελέσματα μ’ αυτά του Chrome. Το αξιοσημείωτο στο πείραμα μας είναι η διαφορά στο χρόνο διεκπεραίωσης της ακολουθίας Fibonacci ανάμεσα σε Chrome και Firefox κυρίως στη Native Javascript, όπου στη περίπτωση της μηχανής V8 είχαμε χρόνο 18.167 sec ενώ στη περίπτωση της μηχανής SpiderMonkey έχουμε 42.855. Επίσης διαφορές παρουσιάζονται και σε όλες τις εκτελέσεις της WebAssembly, με τη μηχανή V8 να παρουσιάζει καλύτερα αποτελέσματα.

Numerical Computing Algorithm	**Native JavaScript**	**WASM Emscripten**	**WASM JavaScript API**	**Native C**	**WASM Emscripten Optimised (O3)**	**WASM Emscripten Optimised (Oz)**	**Native JavaScript** (WebWorker)
Fibonacci Numbers	42.855	8.755	3.178	10.663	8.451	8.321	43.984
Multiply Doubles	2.147	5.036	2.151	5.121	2.175	2.166	2.129
Multiply Integers	2.167	3.307	1.120	4.679	1.054	1.078	2.182

Multiply 2 Double	0.779	2.065	0.693	2.343	1.019	1.009	0.789
Prime Numbers	5.971	6.035	5.902	5.847	5.991	5.992	5.998
Geometrical Mean	3.921	4.486	1.991	5.114	2.597	2.594	3.954
Geometrical Mean of Firefox	**3.125**						

Πίνακας 4.3: Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Desktop με φυλλομετρητή Firefox

Browser**:** Firefox- State hot (V.102.0.5005.63)

CPU**:** AMD Ryzen 5 3600X - 3.8GHz

RAM**:** 16 GB - 1596MHz

Operating System: Windows 10 pro

Εικόνα 4-2: Γράφημα συγκρίσεων χρόνων εκτέλεσης χρόνων εκτέλεσης αριθμητικών υπολογισμών σε Firefox

4.5.3.3 Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Desktop με φυλλομετρητή Opera (hot)

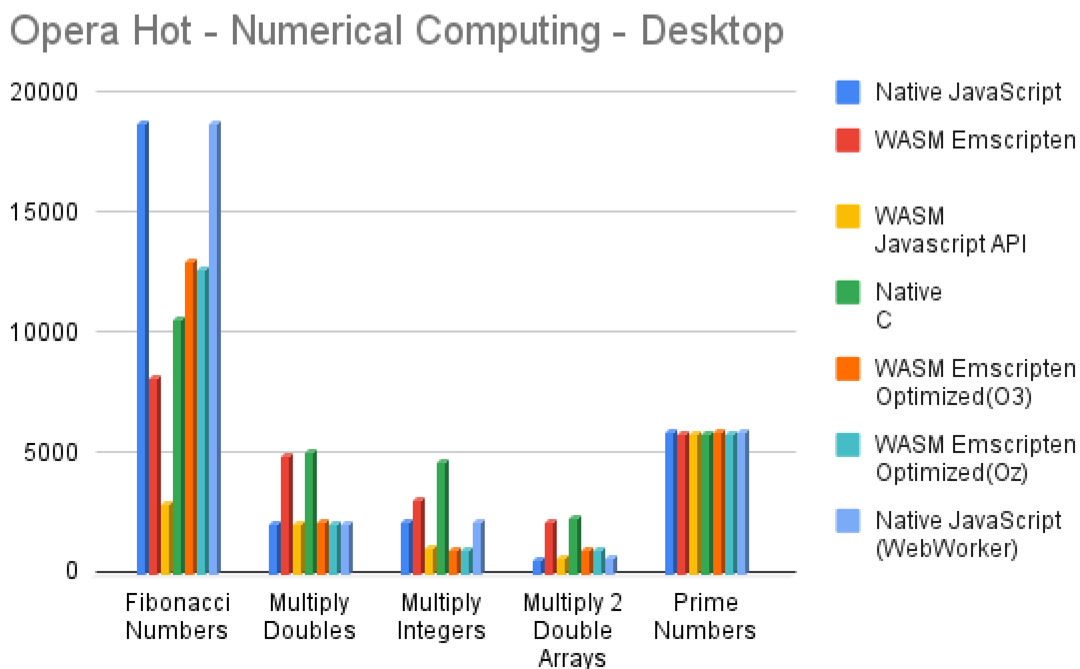
Ο τρίτος και τελευταίος φυλλομετρητής στον οποίο πραγματοποιήθηκε η συγκριτική μας μελέτη είναι ο Opera. O Opera χρησιμοποιεί την μηχανή (engine) Carakan ή οποία έχει πολλές ομοιότητες με την V8 του Chrome. Αυτός είναι και ο λόγος που δεν έχουμε μεγάλες διαφορές στην διεκπεραίωση των αλγορίθμων μας. Οι διαφορές μας είναι στα 10-100μs με καλύτερη απόδοση στην μηχανή V8.

Numerical Computing Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	Native C	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)	Native JavaScript (WebWorker)
Fibonacci Numbers	18.949	8.179	2.927	10.663	13.031	12.737	18.827
Multiply Doubles	2.129	4.998	2.149	5.121	2.201	2.133	2.156
Multiply Integers	2.171	3.107	1.097	4.679	1.064	1.075	2.187
Multiply 2 Double	0.628	2.178	0.699	2.341	1.045	1.024	0.672
Prime Numbers	5.954	5.922	5.870	5.647	5.960	5.902	5.991
Geometrical Mean	3.184	4.394	1.951	5.113	2.856	2.814	3.240
Geometrical Mean of Opera	**2.987**						

Πίνακας 4.4: Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Desktop με φυλλομετρητή Opera

Browser**:** Opera- State hot (V.102.0.5005.63)
CPU**:** AMD Ryzen 5 3600X - 3.8GHz
RAM**:** 16 GB - 1596MHz
Operating System: Windows 10 pro

Εικόνα 4.3: Γράφημα συγκρίσεων χρόνων εκτέλεσης αριθμητικών υπολογισμών σε Opera



4.5.3.4 Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Desktop με φυλλομετρητή Chrome (hot)

Τα αποτελέσματα της δεύτερης κατηγορίας περιέχουν τους αλγόριθμους ταξινόμησης. Σ' αυτή τη κατηγορία της πειραματικής μας μελέτης παρατηρούμε την WebAssembly να έχει καλύτερα αποτελέσματα από την JavaScript όταν εκτελείται μέσω της Emscripten (βελτιστοποιημένη Oz). Στο μόνο σημείο που μπορεί να γίνει ανταγωνιστική η JavaScript είναι στον αλγόριθμο της ταχυταξινόμησης, όπου προσφέρει καλύτερα αποτελέσματα συγκριτικά με την custom υλοποίηση της wasm (JavaScript API). Όταν όμως η wasm βελτιστοποιείται μέσω της Emscripten τότε η διαφορά είναι ορατή. Αξιοσημείωτο είναι επίσης, το γεγονός ότι σ' αυτή τη κατηγορία αλγορίθμων έχουμε καλύτερα αποτελέσματα στη σύγκριση wasm JavaScript API με την βελτιστοποιημένη Emscripten (Oz) με τη μόνη εξαίρεση την αντιστροφή πινάκων. Η τεχνολογία μας δεν έχει μικρότερους χρόνους διεκπεραίωσης από αυτούς της Native C, όπως είδαμε νωρίτερα σε κάποιες περιπτώσεις της πρώτης κατηγορίας.

Sorting Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	Native C	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)	JavaScript (WebWorker)
Bubble Sort	0.635	0.212	0.085	0.017	0.085	0.077	0.680
QuickSort Double	0.205	0.453	0.263	0.018	0.124	0.113	0.231
QuickSort Int	0.202	0.459	0.124	0.014	0.111	0.100	0.233
Reverse Array	0.630	0.086	0.025	0.016	0.045	0.032	0.625
Geometrical Mean	0.358	0.248	0.091	0.016	0.085	0.072	0.388
Geometrical Mean of Chrome	**0.163**						

Πίνακας 4.5: Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Desktop με φυλλομετρητή Chrome

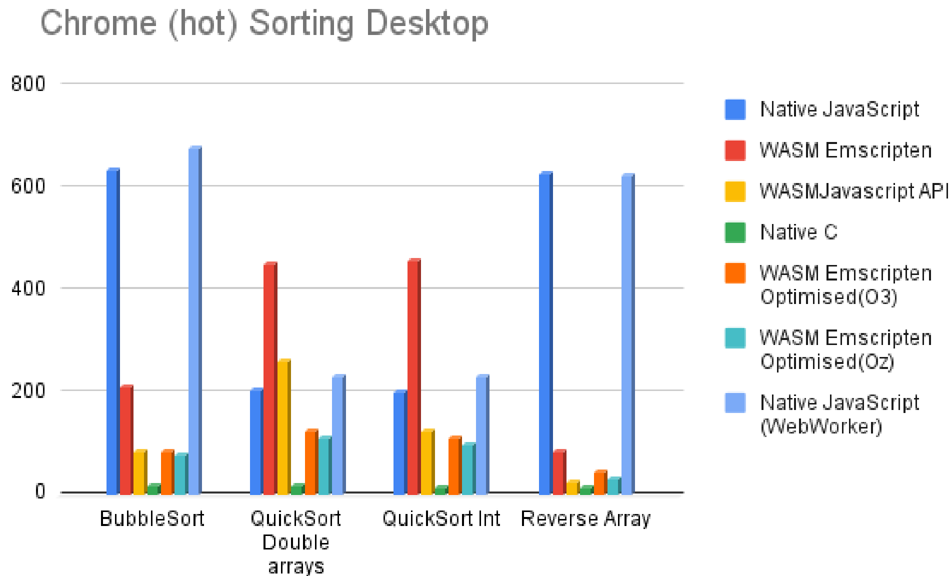
Browser**:** Chrome- State hot (V.102.0.5005.63)

CPU**:** AMD Ryzen 5 3600X - 3.8GHz

RAM**:** 16 GB - 1596MHz

Operating System: Windows 10 pro

Εικόνα 4.4: Γράφημα συγκρίσεων χρόνων εκτέλεσης αλγορίθμων ταξινόμησης σε Chrome



4.5.3.5 Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Desktop με φυλλομετρητή Firefox (hot)

Τα αποτελέσματα της δεύτερης κατηγορίας στον Firefox είναι παρόμοια με αυτά του Chrome με τη διαφορά ότι η μηχανή V8 έχει καλύτερους χρόνους. Πάλι η WASM είναι καλύτερη σε απόδοση από τη JavaScript αλλά αυτή τη φορά όταν εκτελείται από την Emscripten με τη βελτιστοποίηση Oz.

Sorting Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	Native C	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)	JavaScript (WebWorker)
BubbleSort	0.638	0.214	0.091	0.017	0.140	0.128	0.632
QuickSort Double	0.261	0.521	0.294	0.018	0.174	0.152	0.271
QuickSort Int	0.186	0.533	0.149	0.014	0.165	0.151	0.192
Reverse Array	0.477	0.087	0.026	0.016	0.037	0.037	0.472
Geometrical Mean	0.348	0.268	0.100	0.016	0.110	0.102	0.352
Geometrical Mean of Firefox	**0.324**						

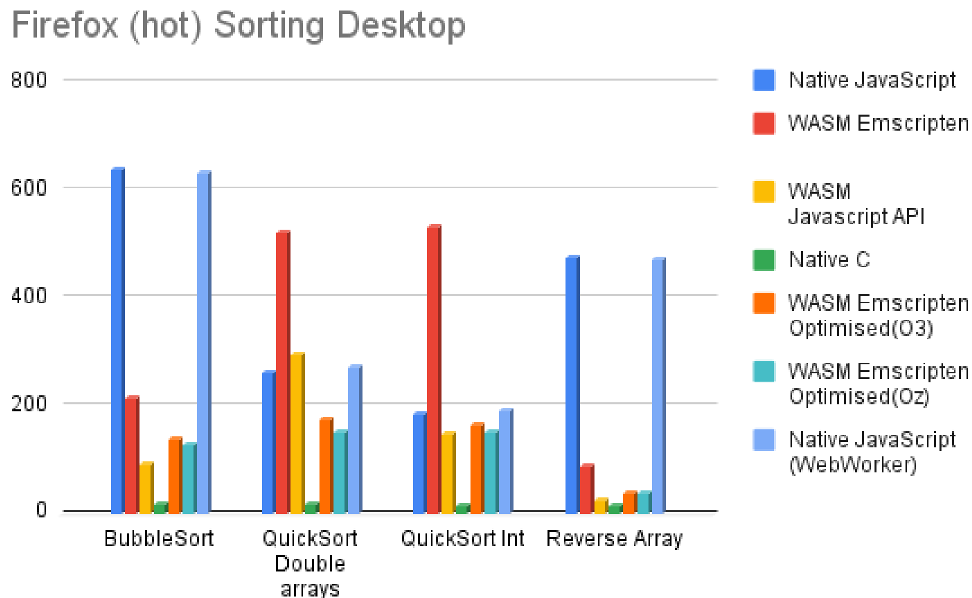
Πίνακας 4.6: Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Desktop με φυλλομετρητή Firefox

Browser**:** Firefox- State hot (V.102.0.5005.63)

CPU**:** AMD Ryzen 5 3600X - 3.8GHz

RAM: 16 GB - 1596MHz
Operating System: Windows 10 pro

Εικόνα 4.5: Γράφημα συγκρίσεων χρόνων εκτέλεσης αλγορίθμων ταξινόμησης σε Firefox



4.5.3.6 Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Desktop με φυλλομετρητή Opera (hot)

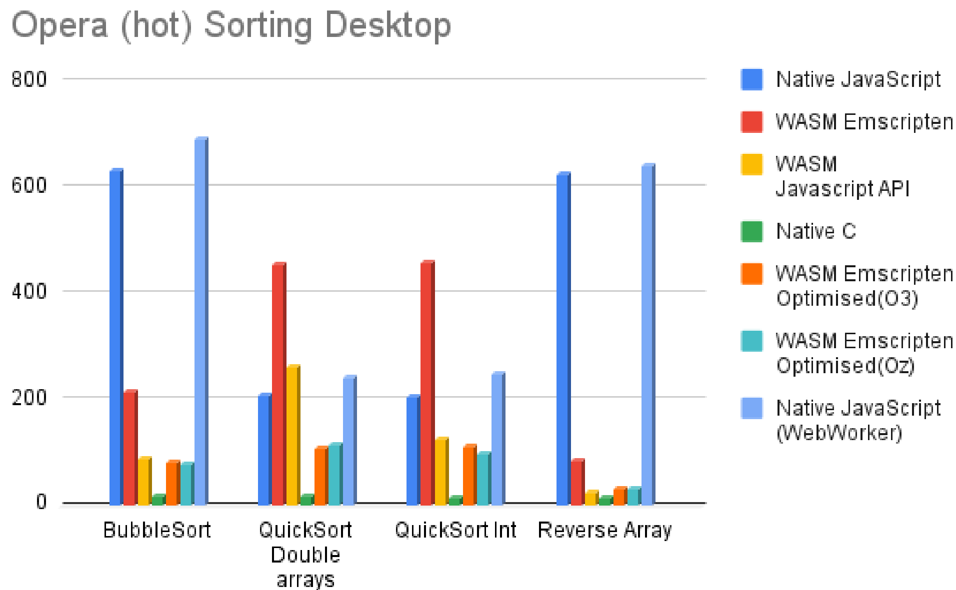
Τα αποτελέσματα στον φυλλομετρητή Opera είναι σχεδόν ίδια με αυτά του Chrome η διαφορές τους είναι από 0.001 - 0.004 μs. Όπως προαναφέρθηκε η μηχανή Carakan που χρησιμοποιείται στο Opera είναι παρόμοιος σε μεγάλο βαθμό με την V8.

Sorting Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	Native C	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)	JavaScript (WebWorker)
BubbleSort	0.634	0.216	0.087	0.017	0.082	0.077	0.694
QuickSort Double	0.207	0.455	0.261	0.018	0.130	0.116	0.243
QuickSort Int	0.204	0.460	0.124	0.014	0.115	0.098	0.249
Reverse Array	0.625	0.086	0.026	0.016	0.042	0.031	0.643
Geometrical Mean	0.359	0.249	0.092	0.016	0.084	0.074	0.405
Geometrical Mean of Opera	0.167						

Πίνακας 4.7: Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Desktop με φυλλομετρητή Opera

Browser: Opera- State hot (V.102.0.5005.63)
CPU: AMD Ryzen 5 3600X - 3.8GHz
RAM: 16 GB - 1596MHz
Operating System: Windows 10 pro

Εικόνα 4.6: Γράφημα συγκρίσεων χρόνων εκτέλεσης αλγορίθμων ταξινόμησης σε Opera



4.5.3.7 Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Desktop με φυλλομετρητή Chrome (hot)

Στη τρίτη και τελευταία κατηγορία των αλγορίθμων μας, εξετάζουμε την επεξεργασία εικόνας με τρεις διαφορετικούς αλγορίθμους. Τα αποτελέσματα της πειραματικής σύγκρισης έδειξαν για άλλη μια φορά την υπεροχή της WebAssembly (στο χρόνο διεκπεραίωσης) όταν αυτή χρησιμοποιείται με την συνάρτηση InstantiateStreaming (WASM JavaScript API) και υλοποιείται από την αρχή (χωρίς generate glue code) με τέτοιο τρόπο ώστε να είναι αποδοτική για τον κάθε αλγόριθμο. Η τεχνολογία που εξετάζεται φαίνεται να έχει περίπου 2 φορές καλύτερο χρόνο από την απλή JavaScript. Ακόμα η WASM φαίνεται πως διεκπεραιώνεται πιο γρήγορα από την JavaScript όταν έχει μεταφραστεί από το περιβάλλον της Emscripten με της λειτουργίες βελτιστοποιήσεις κώδικα (O3 - Oz). Η γλώσσα C έχει τους χαμηλότερους χρόνους από όλες τις γλώσσες που χρησιμοποιήθηκαν.

Image Processing Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	Native C	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)
Grayscale	0.040	0.069	0.023	0.014	0.033	0.038
Blurring (Convolution)	0.381	0.534	0.157	0.121	0.165	0.214
Threshold	0.101	0.117	0.051	0.023	0.058	0.060
Geometric Mean	0.115	0.162	0.056	0.033	0.068	0.078
Geometric Mean of Chrome	**0.075**					

Πίνακας 4.8: Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Desktop με φυλλομετρητή Chrome

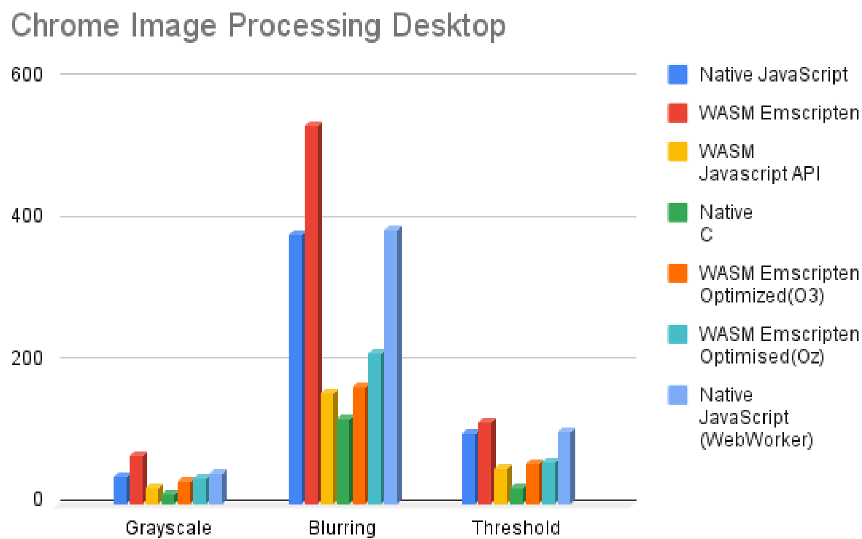
Browser: Chrome - State hot (V.102.0.5005.63)

CPU: AMD Ryzen 5 3600X - 3.8GHz

RAM: 16 GB - 1596MHz

Operating System: Windows 10 pro

Εικόνα 4.7: Γράφημα συγκρίσεων χρόνων εκτέλεσης αλγορίθμων επεξεργασίας εικόνας σε Chrome



4.5.3.8 Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Desktop με φυλλομετρητή Firefox (hot)

Στη περίπτωση εκτέλεσης των αλγορίθμων στο φυλλομετρητή Firefox δεν έχουμε ιδιαίτερες διαφορές. Η απλή JavaScript έχει μια βελτίωση στους χρόνους διεκπεραίωσης συγκριτικά με

αυτές του Chrome. Από την άλλη όταν έχουμε υψηλότερους χρόνους στον Firefox, στη περίπτωση της βελτιστοποιημένης Emscripten (O3 - Oz) με τη διαφορά να είναι κατά μέσο όρο στα 15μs.

Image Processing Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	Native C	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)	Native JavaScript (WebWorker)
Grayscale	0.048	0.061	0.018	0.014	0.031	0.035	0.051
Blurring (Convolution)	0.416	0.558	0.166	0.121	0.196	0.236	0.406
Threshold	0.114	0.171	0.059	0.023	0.085	0.098	0.116
Geometric Mean	0.131	0.179	0.056	0.033	0.080	0.093	0.133
Geometric Mean of Firefox	**0.104**						

Πίνακας 4.9: Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Desktop με φυλλομετρητή Firefox

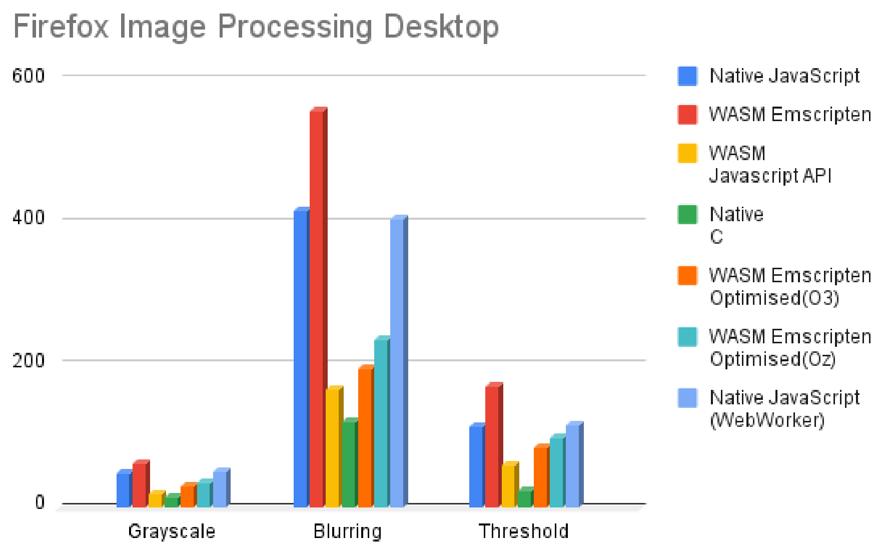
Browser: Firefox- State hot (V.102.0.5005.63)

CPU: AMD Ryzen 5 3600X - 3.8GHz

RAM: 16 GB - 1596MHz

Operating System: Windows 10 pro

Εικόνα 4.8: Γράφημα συγκρίσεων χρόνων εκτέλεσης αλγορίθμων επεξεργασίας εικόνας σε Firefox



4.5.3.9 Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Desktop με φυλλομετρητή Opera (hot)

Στη περίπτωση του φυλλομετρητή Opera δεν υπάρχουν αξιοσημείωτες διαφορές σε σχέση με τα αποτελέσματα του Chrome. Έχουμε σχεδόν τα ίδια αποτελέσματα.

Image Processing Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	Native C	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)	Native JavaScript (WebWorker)
Grayscale	0.038	0.068	0.022	0.014	0.031	0.037	0.041
Blurring (Convolution)	0.384	0.535	0.158	0.121	0.167	0.215	0.381
Threshold	0.101	0.113	0.053	0.023	0.058	0.063	0.105
Geometric Mean	0.113	0.160	0.056	0.033	0.066	0.079	0.117
Geometric Mean of Opera	**0.092**						

Πίνακας 4.10: Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Desktop με φυλλομετρητή Opera

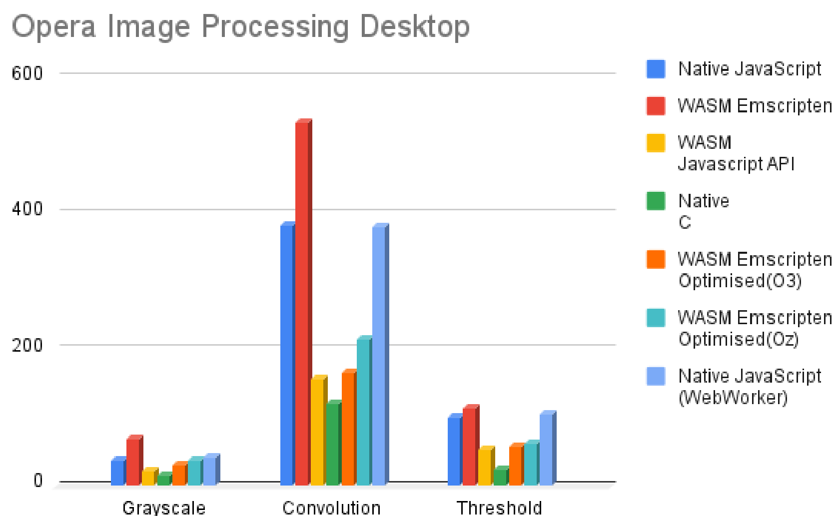
Browser**:** Opera- State hot (V.102.0.5005.63)

CPU**:** AMD Ryzen 5 3600X - 3.8GHz

RAM**:** 16 GB - 1596MHz

Operating System: Windows 10 pro

Εικόνα 4.9: Γράφημα συγκρίσεων χρόνων εκτέλεσης αλγορίθμων επεξεργασίας εικόνας σε Opera



4.5.4 Αποτελέσματα Μετρήσεων σε Laptop

Στην παρακάτω υποενότητα παρουσιάζονται τα αποτελέσματα της συγκριτικής μελέτης σε περιβάλλον Laptop σε φυλλομετρητή Chrome. Τα αποτελέσματα των τριών κατηγοριών έχουν συγκεντρωθεί σε ένα πίνακα. Τα διαγράμματα, οι πίνακες και οι φυλλομετρητές, παραθέτονται στα παραρτήματα.
Τα αποτελέσματα της μελέτης σε υπολογιστή Laptop δεν διαφέρουν πολύ από αυτά του Desktop με τη βασική εξαίρεση της αύξησης του χρόνου διεκπεραίωσης λόγω της χαμηλότερης ισχύος του επεξεργαστή. Η υπεροχή της τεχνολογίας WebAssembly είναι ορατή καθώς έχει καλύτερους χρόνους σε όλους τους αλγορίθμους που επιλέχθηκαν με την μόνη εξαίρεση τον πολλαπλασιασμό των πινάκων.

Algorithms Laptop Chrome	**Native JavaScript**	**WASM Emscripten**	**WASM JavaScript API**	**WASM Emscripten Optimised (O3)**	**WASM Emscripten Optimised (Oz)**
Fibonacci	27.159	30.487	6.185	20.154	18.254
Multiply Doubles	3.931	4.515	3.925	3.954	3.918
Multiply Integers	3.939	4.427	1.476	1.490	1.485
Multiply Double Arrays	1.087	2.846	1.301	1.820	1.925
Prime Numbers	6.956	5.624	5.024	5.547	5.514
BubbleSort	1.038	0.322	0.128	0.125	0.112
QuickSort Double	0.432	0.544	0.335	0.172	0.176
QuickSort Int	0.416	0.536	0.175	0.152	0.152
Reverse Array	1.081	1.254	0.065	0.067	0.054
Grayscale	0.075	0.106	0.045	0.062	0.076
Blurring	0.654	0.742	0.295	0.292	0.368
Threshold	0.242	0.234	0.154	0.149	0.159

Πίνακας 4.11: Πίνακας αποτελεσμάτων με χρήση Laptop σε φυλλομετρητή Chrome
Browser**:** Chrome- State hot
CPU**:** INTEL Core i5 7200U - 2.50GHz
RAM**:** 8 GB - 1064 MHz
Operating System: Windows 10 pro

4.5.5 Αποτελέσματα Μετρήσεων σε Mobile

Ο παρακάτω πίνακας περιέχει τα αποτελέσματα των κατηγοριών που επιλέχθηκαν για την παρούσα εργασία σε mobile περιβάλλον εκτελεσμένα από τον φυλλομετρητή Chrome. Τα αποτελέσματα παρουσιάζουν κάποιες διαφορές σε σχέση με το Desktop και Laptop. Συγκεκριμένα παρατηρούμε ότι η WASM υλοποιημένη με την συνάρτηση InstantiateStreaming προσφέρει 4.5 φορές καλύτερη απόδοση στην ακολουθία fibonacci από την απλή JavaScript όπως και 5.7 φορές στην αντιστροφή των πινάκων. Στις υπόλοιπες μετρήσεις έχει πάλι καλύτερη απόδοση η WASM αυτή τη φορά μεταγλωτισμένη και βελτιστοποιημένη από την Emscripten με την παράμετρο Oz. Εξαίρεση στα αποτελέσματα μας αποτελούν οι αλγόριθμοι του πολλαπλασιασμού πινάκων (DOUBLE), οι πρώτοι αριθμοί, η συνέλιξη εικόνας και ο αλγόριθμος προσαρμοστικού κατωφλιού όπου καλύτερη απόδοση έχει η απλή JavaScript.

Algorithms Laptop Chrome	**Native JavaScript**	**WASM Emscripten**	**WASM JavaScript API**	**WASM Emscripten Optimised (O3)**	**WASM Emscripten Optimised (Oz)**
Fibonacci	16.232	36.315	3.550	11152	10886
Multiply Doubles	4533	8483	4528	4648	4556
Multiply Integers	4526	6259	1724	1720	1702
Multiply Double Arrays	663	4550	1827	2425	2477
Prime Numbers	1486	3382	1937	3364	3317
BubbleSort	1326	4252	0.148	0.145	0.140
QuickSort Double	0.312	1041	0.435	0.170	0.164
QuickSort Int	0.482	1.083	0.361	0.265	0.241
Reverse Array	0.168	0.271	0.029	0.058	0.058
Grayscale	0.061	0.112	0.025	0.033	0.034
Convolution	0.211	0.454	0.231	0.243	0.271
Threshold	0.091	0.197	0.145	0.107	0.056

Πίνακας 4.12: Πίνακας αποτελεσμάτων με χρήση Mobile σε φυλλομετρητή Chrome
Browser**:** Chrome - State hot
CPU**:** A13 Bionic Chip - 2.6 GHz

RAM: 6 GB
Operating System: IOS 15.5

5 Συζήτηση και Συμπεράσματα Αποτελεσμάτων

Για να μπορέσουμε να έχουμε μια ολοκληρωμένη εικόνα των αποτελεσμάτων, δημιουργήσαμε έναν πίνακα που περιέχει τις μετρήσεις γεωμετρικών μέσων των αλγορίθμων που εκτελέστηκαν σε Chrome, Firefox, Opera σε περιβάλλον Desktop. Συγκεκριμένα στον πίνακα συγκρίνεται το καλύτερο αποτέλεσμα της JavaScript με αυτό της WebAssembly.

Desktop									
Algorithms Desktop	**JavaScript Chrome**	**WASM Chrome**	**Difference Chrome**	**JavaScript Firefox**	**WASM Firefox**	**Difference Firefox**	**JavaScript Opera**	**WASM Opera**	**Difference Opera**
Fibonacci Numbers	18.167	2.809 (API)	15.358	42.855	3.178 (API)	39.677	18.949	2.927 (API)	16.022
Multiply Doubles	2.152	2.039 (API)	113	2.129	2.151 (API)	-22	2.129	2.149 (API)	-20
Multiply Integers	2.056	1.074 Emc (Oz)	982	2.167	1.054 Emc (O3)	1.113	2.171	1.064 Emc (O3)	1.107
Multiply Arrays Double	629	682 (API)	-53	779	693 (API)	86	628	699 (API)	-71
Prime Numbers	5.945	5.925 (API)	20	5.971	5.902 (API)	69	5.954	5.870 (API)	84
BubbleSort	635	77 Emc (Oz)	558	638	91 (API)	558	634	77 Emc (Oz)	557
QuickSort Double	205	113 Emc (Oz)	92	261	152 Emc (Oz)	92	207	116 Emc (Oz)	91
QuickSort	202	100 Emc (Oz)	102	186	149 (API)	102	204	98 Emc (Oz)	106
Reverse Array	630	25 (API)	605	477	26 (API)	605	625	26 (API)	599
Grayscale	40	23 (API)	17	48	18 (API)	30	38	22 (API)	16
Convolutio n	381	157 (API)	224	416	166 (API)	250	384	158 (API)	226

Threshold	101	51 (API)	50	114	59 (API)	55	101	53 (API)	48

Πίνακας 5.1: Πίνακας διαφορών απόδοσης WASM και JavaScript στις περιπτώσεις βέλτιστης απόδοσης της εκάστοτε τεχνολογίας σε περιβάλλον Desktop

Από τα αποτελέσματα του παραπάνω πίνακα, διακρίνουμε την υπεροχή της βελτιστοποιημένης WebAssembly ως προς το χρόνο εκτέλεσης των αλγορίθμων. Η τεχνολογία που εξετάζουμε έχει καλύτερη απόδοση σε προβλήματα υπολογισμού, ταξινόμησης αλλά και πράξεων μεταξύ πινάκων. Η αδυναμία της WASM σε σχέση με την JavaScript είναι στις πράξεις των μεταβλητών διπλής ακρίβειας. Δηλαδή όταν μια μεταβλητή είναι δηλωμένη σαν Double στο κώδικα της C, τότε κατά την μετατροπή της σε wasm αντιστοιχεί σε έναν 64-bit unsigned integer που επιβαρύνει την απόδοση εκτέλεσης λόγω του μεγέθους του. Για το λόγο αυτό παρατηρούμε την υπεροχή της JavaScript στους αλγορίθμους "Multiply Doubles" και "Multiply Double Arrays". Στους αλγορίθμους επεξεργασίας εικόνας παρατηρούμε ότι η απόδοση της wasm είναι δύο φορές καλύτερη από αυτή της JavaScript

Η διαφορές που εντοπίζουμε στα αποτελέσματα μεταξύ των προγραμμάτων περιήγησης, αποδεικνύουν ότι ο κάθε μεταγλωττιστής - διερμηνέας έχει κατασκευαστεί έτσι ώστε να βελτιστοποιεί με διαφορετικό τρόπο τον τύπο του κάθε αλγόριθμου. Στο παρακάτω πινακάκι έχει υπολογιστεί ο γεωμετρικός μέσος για κάθε κατηγορία ανά φυλλομετρητή. Ο Chrome δείχνει να έχει καλύτερους χρόνους από τον Firefox, ενώ σε σχέση με των Opera δείχνει να είναι πολύ κοντά λόγω της παρόμοιας αρχιτεκτονικής των μηχανών τους.

Desktop			
Browser	Numerical	Sorting	Image Processing
Google Chrome (V8)	2.955	0.163	0.075
Mozilla Firefox (SpiderMonkey)	3.125	0.324	0.104
Opera (Caracan)	2.987	0.167	0.092

Πίνακας 5.2: Πίνακας αποτελεσμάτων γεωμετρικών μέσων στους τρεις φυλλομετρητές, για κάθε κατηγορία αλγορίθμων σε περιβάλλον Desktop

Στον πίνακα 5.2 παρουσιάζονται τα αποτελέσματα των γεωμετρικών μέσων για τις εκτελέσεις: Native JavaScript, WebAssembly Emscripten, WASM JavaScript API, WebAssembly Emscripten (O3), WebAssembly Emscripten (Oz), Native JavaScript (WebWorker). Για την κατηγορία των υπολογιστικών πράξεων παρατηρούμε την υπεροχή του Chrome (1.05x) σε σχέση με τον Firefox και μια μικρή διαφορά με τον Opera (1.01x). Στους αλγόριθμους ταξινόμησης ο Chrome αυξάνει την διαφορά καθώς είναι 1.98x γρηγορότερος από τον Firefox και 1.02x από τον Opera. Το ίδιο συμβαίνει και στους

αλγόριθμους επεξεργασίας εικόνας όπου είναι 1.38x από τον Firefox και 1.22x από τον Opera.

Laptop									
Algorithms Laptop	**JavaScript Chrome**	**WASM Chrome**	**Difference Chrome**	**JavaScript Firefox**	**WASM Firefox**	**Difference Firefox**	**JavaScript Opera**	**WASM Opera**	**Difference Opera**
Fibonacci Numbers	27.159	6.185 (API)	20.974	52.924	4.320	48.604	32.270	4.154	28.116
Multiply Doubles	3.931	3.925	6	3.923	3.918	5	3.999	3.929	70
Multiply Integers	3.939	1.476	2.463	3.939	1.475	2.464	3.924	1.477	2.447
Multiply Arrays Double	1.087	1.301	-214	1.346	1.396	-50	1.058	1.268	-210
Prime Numbers	6.956	5.024	1.932	7.662	6.229	1.433	6.964	5.089	1.875
BubbleSort	1.038	112	926	2.262	145	2.117	1.088	110	978
QuickSort Double arrays	432	172	260	412	255	157	454	156	298
QuickSort	416	152	264	312	247	65	442	152	290
Reverse Array	1.081	65	1.016	1.086	64	1.022	1.086	55	1.031
Grayscale	75	62	13	76	40	36	74	45	29
Convolution	654	292	362	719	314	405	708	288	420
Threshold	242	149	93	335	161	174	251	134	117

Πίνακας 5.3: Πίνακας διαφορών απόδοσης WASM και JavaScript στις περιπτώσεις βέλτιστης απόδοσης της εκάστοτε τεχνολογίας σε περιβάλλον Laptop

Τα αποτελέσματα του παραπάνω πίνακα, αφορούν τις πειραματικές εκτελέσεις που πραγματοποιήθηκαν σε συσκευή Laptop. Οι μετρήσεις είναι παρόμοιες με αυτές του Desktop, με την αναμενόμενη διαφορά ότι οι χρόνοι είναι από 1.5x εώς 2x φορές μεγαλύτεροι από αυτές του σταθερού υπολογιστή, αυτό οφείλεται στη διαφορά δυναμικής του επεξεργαστή του κάθε συστήματος. Αυτό που αξίζει να σημειωθεί είναι ότι παρατηρούμε ακόμα μεγαλύτερες διαφορές στους χρόνους της WebAssembly. Δηλαδή η wasm έχει μεγαλύτερο χρόνο αναλογικά από αυτόν της JavaScript, αυτό συμβαίνει διότι η WebAssembly απαιτεί σημαντικά περισσότερη μνήμη από ότι η JavaScript στους φυλλομετρητές που την εξετάζουμε. Η JavaScript χρησιμοποιεί συλλέκτη σκουπιδιών όπου διαχειρίζεται με δυναμικό τρόπο το χώρο της μνήμη, την δεσμεύει και την αποδεσμεύει. Η WebAssembly χρησιμοποιεί μοντέλο γραμμικής μνήμης που δεν ανακτά τη μνήμη αυτόματα.

Στην custom υλοποίηση της WebAssembly έχουμε φροντίσει να δεσμεύουμε τη μνήμη που χρειάζεται κάθε αλγόριθμος και στη συνέχεια να την αποδεσμεύουμε.

Laptop			
Browser	Numerical	Sorting	Image Processing
Google Chrome (V8)	4.474	0.239	0.176
Mozilla Firefox (SpiderMonkey)	4.721	0.311	0.194
Opera (Caracan)	4.374	0.250	0.179

Πίνακας 5.4: Πίνακας αποτελεσμάτων γεωμετρικών μέσων στους τρεις φυλλομετρητές, για κάθε κατηγορία αλγορίθμων σε περιβάλλον Laptop

Στον πίνακα 5.4 παρουσιάζονται τα αποτελέσματα των γεωμετρικών μέσων για τις εκτελέσεις: Native JavaScript, WebAssembly Emscripten, WASM JavaScript API, WebAssembly Emscripten (O3), WebAssembly Emscripten (Oz). Στις συγκρίσεις που πραγματοποιήθηκαν για την συσκευή Laptop παρατηρούμε καλύτερα αποτελέσματα στην κατηγορία υπολογιστικών πράξεων για τον φυλλομετρητή Opera με 1.02x από τον Chrome και 1.08x από τον Firefox. Στην κατηγορία ταξινόμησης προβάδισμα έχει ο Chrome με 1.30x από τον Firefox και 1.04x από τον Opera. Το ίδιο συμβαίνει και σ’ αυτούς της επεξεργασίας εικόνας όπου είναι γρηγορότερος κατά 1.10x του Firefox και 1,07x του Opera.

Mobile									
Algorithms	JavaScript	WASM	Difference	JavaScript	WASM	Difference	JavaScript	WASM	Difference

Mobile	Chrome	Chrome	Chrome	Firefox	Firefox	Firefox	Opera	Opera	Opera
Fibonacci Numbers	16.232	3.550	12.682	16.280	3.559	12.721	16.286	3.532	12.754
Multiply Doubles	4.533	4.528	5	4.547	4.535	12	4.534	4.540	-6
Multiply Integers	4.526	1.702	2.824	4.559	1.703	2.856	4.546	1.700	2.846
Multiply Arrays Double	663	1.827	-1.164	651	1.914	-1.263	664	1.820	-1.156
Prime Numbers	1.486	1.937	-451	1.463	1.939	-476	1.467	1.949	-482
BubbleSort	1.926	140	1.786	1.930	147	1.783	1.923	146	1.777
QuickSort Double	312	164	148	444	174	270	453	113	340
QuickSort	482	241	241	485	263	222	478	248	230
Reverse Array	168	29	139	192	57	135	172	24	148
Grayscale	61	25	36	51	26	25	54	26	28
Convolution	211	231	-20	235	226	9	228	237	-9
Threshold	91	56	35	107	56	51	93	55	38

Πίνακας 5.5: Πίνακας διαφορών απόδοσης WASM και JavaScript στις περιπτώσεις βέλτιστης απόδοσης της εκάστοτε τεχνολογίας σε περιβάλλον Mobile

Στον πίνακα 5.5 έχουμε καταγράψει τα αποτελέσματα των γεωμετρικών μέσων σε συσκευή κινητού για τους τρεις διαφορετικούς φυλλομετρητές. Τα αποτελέσματα αποτελούν έκπληξη για την παρούσα έρευνα, καθώς αναδεικνύεται η δυναμική του επεξεργαστή της κινητής συσκευής. Παρατηρείται ότι ο επεξεργαστής του Iphone παρόλο που έχει χαμηλότερα χαρακτηριστικά από αυτά του Desktop, καταφέρνει και έχει καλύτερα αποτελέσματα σε ορισμένους αλγόριθμος υλοποιημένους σε JavaScript, αυτό οφείλεται στην αρχιτεκτονική του επεξεργαστή A-13 Bionic. Κάποιοι από τους αλγορίθμους (σε JavaScript) όπου υπερτερεί το κινητό είναι: Fibonacci, Prime Numbers, Threshold. Όσον αφορά τους χρόνους εκτέλεσης της WebAssembly δε παρατηρείται κάποια βελτίωση στην ταχύτητα διεκπεραίωσης καθώς όπως εξηγήσαμε και στα αποτελέσματα του Laptop, το πρότυπο wasm χρειάζεται παραπάνω μνήμη από την JavaScript και η ram της κινητής συσκευής είναι σχετικά μικρή με αυτή του Desktop. Το πλεονέκτημα της συγκεκριμένης συσκευής να εκτελεί γρηγορότερα τους αλγορίθμους που έχουν υλοποιηθεί σε JavaScript αλλάζει τα τελικά αποτελέσματα.

Mobile

Browser	Numerical	Sorting	Image Processing
Google Chrome (V8)	3.737	0.242	0.110
Mozilla Firefox (SpiderMonkey)	3.763	0.250	0.111
Opera (Caracan)	3.820	0.243	0.110

Πίνακας 5.6: Πίνακας αποτελεσμάτων γεωμετρικών μέσων στους τρεις φυλλομετρητές, για κάθε κατηγορία αλγορίθμων σε περιβάλλον Mobile

Στον πίνακα 5.6 παρουσιάζονται τα αποτελέσματα των γεωμετρικών μέσων για τις εκτελέσεις: Native JavaScript, WebAssembly Emscripten, WASM JavaScript API, WebAssembly Emscripten (O3), WebAssembly Emscripten (Oz). Η σύγκριση των φυλλομετρητών σε περιβάλλον κινητής συσκευής αναδεικνύει τη υπεροχή του Chrome αλλά αυτή τη φορά σε μικρότερη διαφορά. Συγκεκριμένα ο Chrome είναι γρηγορότερος στην κατηγορία των πράξεων με 1.007x του Firefox και 1.02x του Opera. Το ίδιο παρατηρείται και στους αλγόριθμους ταξινόμησης με 1.003x επί του Firefox και την αμελητέα διαφορά επί του Opera. Οι διαφορές για την επεξεργασία εικόνας είναι επίσης αμελητέες.

5.1 JavaScript και WebAssembly στα Προγράμματα Περιήγησης

Οι πληροφορίες που αντλήθηκαν από την βιβλιογραφική επισκόπηση δείχνουν ό,τι η JavaScript μεταφράζεται σε κώδικα μηχανής μέσα από την πολυεπίπεδη αρχιτεκτονική του μεταγλωττιστή. Η αρχιτεκτονική εξαρτάται από το πρόγραμμα περιήγησης που χρησιμοποιείται. Ο προγραμματιστής θα υλοποιήσει τον κώδικα όπου αργότερα θα μεταβεί στη μηχανή και θα εκτελεστεί. Ο κώδικας διαχωρίζεται ώστε να δημιουργηθεί το αφηρημένο συντακτικό δέντρο που επιτυγχάνει την γρηγορότερη εκτέλεση. Το αφηρημένο συντακτικό δέντρο προορίζεται για να αναλύσει και να επαληθεύσει ότι τα στοιχεία της γλώσσας χρησιμοποιούνται σωστά. Από εκεί και πέρα ο profiler εξετάζει τους τρόπους βελτιστοποίησης του κώδικα. Μέσω του profiler, ο μη βελτιστοποιημένος κώδικας μεταβιβάζεται στο μεταγλωττιστή για να βελτιστοποιηθεί και να δημιουργηθεί ο κώδικας μηχανής, ο οποίος στη συνέχεια θα αντικαταστήσει την προηγούμενη έκδοση (μη βελτιστοποιημένη). Όσο αυτές οι αλλαγές συσσωρεύονται, η απόδοση στη εκτέλεση θα βελτιώνεται.

Η μορφή του αρχείου .wasm της WebAssembly, είναι πιο κοντά σε αυτή του κώδικα μηχανής συγκριτικά με την απλή JavaScript. Ως εκ τούτου χρειάζονται λιγότερα βήματα για την μεταγλώττιση του σε κώδικα μηχανής και δεν είναι απαραίτητη η βελτιστοποίηση του καθώς θα έχει ήδη βελτιστοποιηθεί (ahead of time). Επίσης η μορφή του αρχείου wasm είναι κατασκευασμένο με τέτοιο τρόπο ώστε να στοχεύει σε διαφορετικές μηχανές με πιο αποτελεσματικό τρόπο. Καθώς το αρχείο έχει μεταγλωττιστεί, το πρόγραμμα περιήγησης πρέπει απλά να κατεβάσει το αρχείο. Αυτό έχει ως αποτέλεσμα να γίνεται ένα μικρό άλμα στην όλη διαδικασία της εκτέλεσης, καθώς αποφεύγονται τα βήματα της ανάλυσης και της βελτιστοποίησης προτού μεταφερθεί στον μεταγλωττιστή ώστε να δημιουργηθεί ο κώδικας μηχανής.

Η WebAssembly δεν προσφέρει μεγάλο εύρος δομών δεδομένων όπως άλλες σύγχρονες γλώσσες προγραμματισμού. Χρησιμοποιεί κυρίως αριθμούς κινητής υποδιαστολής και είναι πιο κατάλληλη για τον προγραμματισμό που περιλαμβάνει φορολογικούς υπολογισμούς. Ακόμα όταν ο προγραμματιστής καλείται να υλοποιήσει μια εφαρμογή σε WebAssembly τότε θα πρέπει πρώτα να την υλοποιήσει στη γλώσσα που επιθυμεί (ανάλογα με τις ανάγκες του) και στη συνέχεια να την μεταγλωττίσει με έναν από τους πολλούς μεταγλωττιστές που υπάρχουν.

Όπως συζητήθηκε στο κεφάλαιο 3, τα σύγχρονα προγράμματα περιήγησης είναι πολύ πολύπλοκα στο τρόπο με τον οποίο εκτελούν τον κώδικα ώστε να παρέχουν τη βέλτιστη δυνατή απόδοση. Λόγω της just-in-time μεταγλώττισης παρέχεται η δυνατότητα στην JavaScript να βελτιστοποιηθεί σε μεγάλο βαθμό, η WebAssembly όμως έχει και εδώ το πλεονέκτημα καθώς οι βελτιστοποιήσεις πραγματοποιούνται πριν και όχι κατά τη διάρκεια της εκτέλεσης. Κάθε ιστότοπος έχει το δικό του τρόπο στη βελτιστοποίησης του κώδικα.

Όσον αφορά την απόδοση των φυλλομετρητών, παρατηρείται (πίνακες 5.2, 5,4. 5,6) πως ο Chrome συνολικά προσφέρει καλύτερη απόδοση στην ταχύτητα εκτέλεσης συγκριτικά με τους υπόλοιπους φυλλομετρητές της συγκριτικής μας μελέτης. Η κατάσταση του μεταγλωττιστή JIT επίσης βελτιώνει την εκτέλεση της εκάστοτε συνάρτησης στους αλγορίθμους μας (πίνακας 4.1)

5.2 Συμπεράσματα και Πλεονεκτήματα - Μειονεκτήματα WebAssembly

Μέσω της σχετικής μελέτης της βιβλιογραφίας και των πειραματικών αποτελεσμάτων, οδηγούμαστε στο συμπέρασμα πως η βελτιστοποιημένη WebAssembly υπερέχει σε θέματα ταχύτητας έναντι της JavaScript. Αυτό συμβαίνει και στις τρεις κατηγορίες αλγορίθμων που επιλέχθηκαν. Παρόμοιες έρευνες έχουν αποδείξει πως η wasm είναι γρηγορότερη σε συγκεκριμένες κατηγορίες, όπως αυτή των υπολογισμών πράξεων. Η διαφορά μας με τις αντίστοιχες έρευνες είναι ότι χρησιμοποιήσαμε διαφορετική μέθοδο για την μεταγλώττιση του κώδικα (InstantiateStream), αποφύγαμε την αυτοματοποιημένη δημιουργία κώδικα, προσαρμόσαμε τη μνήμη του κάθε αλγορίθμου στις ανάγκες μας και εφαρμόσαμε βελτιστοποιήσεις μέσω του compiler. Όλο αυτό ήταν αρκετά χρονοβόρο όσον αφορά το θέμα της υλοποίησής του προτύπου. Επίσης οι αλγόριθμοι μας μπορεί να απαιτούν υψηλή υπολογιστική ισχύ αλλά δεν αποτελούν εφαρμογές με μεγάλες ανάγκες (enterprise), ούτε πρόκειται για real time υλοποιήσεις όπου η κατασκευή τους σε WebAssembly θα απαιτούσε ακόμα περισσότερο χρόνο υλοποίησης. Οπότε αντιλαμβανόμαστε ότι πριν υλοποιήσουμε - μεταφέρουμε μια εφαρμογή στο custom πρότυπο της wasm θα πρέπει να μελετήσουμε τις απαιτήσεις της εφαρμογής και να κρίνουμε αν θα είναι άξιο υλοποίησης.

Η οριοθέτηση της μνήμης και η έλλειψη του συλλέκτη σκουπιδιών (garbage collector) αποτελεί αναμφισβήτητα ένα από τα σημαντικότερα ελαττώματα της τεχνολογίας. Η συγκεκριμένη αδυναμία έχει ως συνέπεια τα αρχεία που δημιουργούνται να έχουν μεγαλύτερο μέγεθος με αποτέλεσμα να χρειάζεται περισσότερος χρόνος για την λήψη τους σε επίπεδο δικτύου.

Η μορφή της WebAssembly έχει κάνει ένα σπουδαίο βήμα ως προς την επέκταση του οικοσυστήματος του διαδικτύου “ζωντανεύοντας” εφαρμογές και δίνοντας την επιλογή χρήσης διαφόρων γλωσσών προγραμματισμού σε συνδυασμό με την JavaScript. Οι γλώσσες C, C++ και Rust, πλέον αποτελούν κομμάτι του οικοσυστήματος και ακόμα καλύτερα το επεκτείνουν αποτελεσματικά. Αυτό δε σημαίνει ότι η wasm μπορεί να αντικαταστήσει την JavaScript αλλά να συνδυαστούν για ένα καλύτερο αποτέλεσμα σε περισσότερα επίπεδα.

Η WebAssembly δεν έχει τη δυνατότητα να επέμβει με άμεσο τρόπο στο DOM (Document Object Model). Αυτός είναι και ένας από τους πολλούς λόγους που δε μπορεί να σταθεί χωρίς την JavaScript στο οικοσύστημα του διαδικτύου. Η τεχνολογία εξελίσσεται συνεχώς και αυτό αποτελεί μια ελπίδα για την επέκταση των λειτουργιών της.

Σύμφωνα με την βιβλιογραφία η WebAssembly παρέχει μεγαλύτερη σταθερότητα στην απόδοση της εκτέλεσης και αυτό οφείλεται στην περιορισμένη σύνταξη της γλώσσας αλλά και την πρόωρη μεταγλώττιση (ahead of time) και βελτιστοποίηση του κώδικα. Αντιθέτως ο μεταγλωττιστής JIT προσφέρει μικρότερη σταθερότητα στα αποτελέσματα καθώς ενδέχεται κατα τη διάρκεια της βελτιστοποίησης να κάνει ξαφνική διακοπή της διαδικασίας και να επανεκκινηθεί η διαδικασία σπαταλώντας έτσι σημαντικό χρόνο.

Καταλήγοντας, συμπεραίνουμε ότι με την πάροδο του χρόνου η υπό εξέταση τεχνολογία WebAssembly βελτιώνεται συνεχώς από τους μηχανικούς πληροφορικής. Η συγκριτική μελέτη μπορεί να έδειξε την υπεροχή της wasm στα περισσότερα αποτελέσματα αυτό όμως δεν σημαίνει ότι μπορεί να την αντικαταστήσει την JavaScript. Ο προγραμματιστής που θα κληθεί να υλοποιήσει μια εφαρμογή για τον ιστότοπο θα πρέπει να χρησιμοποιήσει και τις δύο γλώσσες, την κάθε μια για συγκεκριμένο σκοπό. Είναι σημαντικό να σημειωθεί ότι παρόλα τα πλεονεκτήματα που προσφέρει η τεχνολογία, υπάρχουν κάποιες απώλειες που πρέπει να ληφθούν υπόψη για ορισμένες υλοποιήσεις. Ο χρόνος που απαιτείται για την λήψη και τη μεταγλώττιση μιας λειτουργικής μονάδας (module) είναι γρήγορος αλλά αν η βελτιστοποίηση που παρέχεται από την WebAssembly σε σχέση με την αρχική υλοποίηση είναι μικρής τάξης, τότε ενδέχεται να μην είναι αποδοτικότερη. Η σύσταση που γίνεται είναι να χρησιμοποιηθούν και οι δύο γλώσσες για διαφορετικά λειτουργικά κομμάτια μιας εφαρμογής. Έτσι ο προγραμματιστής θα μπορεί να έχει τα βέλτιστα αποτελέσματα.

5.3 Μελλοντική Επέκταση

Καθώς η παρούσα εργασία απαρτίζει ένα σχετικά μικρό σύνολο από αλγορίθμους, θα μπορούσε μελλοντικά να επεκταθεί με μεγαλύτερες real world εφαρμογές στις οποίες θα μπορούσαν να αναδειχθούν σε μεγαλύτερο βάθος τα μειονεκτήματα και τα πλεονεκτήματα της τεχνολογίας. Ακόμα θα μπορούσε να μελετηθεί η διαχείριση - κατανάλωση μνήμης από τους φυλλομετρητές καθώς παρατηρούμε ότι αυτό το θέμα επηρεάζει την απόδοση των αλγορίθμων σε διαφορετικές συσκευές. Άλλες πτυχές από τις οποίες θα μπορούσαν να αντληθούν χρήσιμα συμπεράσματα είναι αυτές των μεταγλωττιστών. Δηλαδή θα ήταν ωφέλιμο να πραγματοποιηθούν συγκριτικές μελέτες όταν χρησιμοποιούμε διαφορετικούς μεταγλωττιστές για να την δημιουργία της λειτουργικής μονάδας (module), καθώς

επηρεάζουν σε μεγάλο βαθμό την απόδοση αλλά και το μέγεθος της κάθε μονάδας. Επίσης σημαντικό, είναι να γνωρίζουμε πως συμπεριφέρεται κάθε γλώσσα όταν μετατρέπεται σε wasm. Για παράδειγμα όταν μετατρέπεται από C++, Rust. TypeScript κ.α, σε wasm. Ποια γλώσσα θεραπεύει τα προβλήματα που παρουσιάζονται αλλά και ποια δημιουργεί νέα. Η συγκριτική μελέτη της τεχνολογίας θα πρέπει να γίνεται με μεγάλη προσοχή καθώς μπορεί να επηρεαστεί από πολλούς παράγοντες που δεν έχουμε υπολογίσει εξ' αρχής, όπως για παράδειγμα οι εκδόσεις των φυλλομετρητών, η μνήμη cache, οι λειτουργίες που εκτελούνται στο παρασκήνιο, η εξοικονόμηση ενέργειας που μπορεί να είναι ενεργοποιημένη σε ένα μηχάνημα και να επηρεάζει την απόδοση του επεξεργαστή. Καθώς η τεχνολογία εξελίσσεται και βελτιώνεται συνεχώς είναι απαραίτητο να γίνονται τακτικά οι συγκρίσεις καθώς τα αποτελέσματα συνήθως αλλάζουν προς το καλύτερο.

6 Βιβλιογραφία

[1] Allan Sendagi. Jan 29, 2020,

https://medium.com/@allansendagi/inside-the-javascript-engine-compiler-and-interpreter-c8faa638b0d9

[2] Atapattu, Sachille. March 6, 2020 - By. “Bringing You Up to Speed on How Compiling WebAssembly is Faster.” *Cornell CS*,

https://www.cs.cornell.edu/courses/cs6120/2019fa/blog/wasm/..

[3] Awesome WASM. October 2020

https://github.com/mbasso/awesome-wasm#compilers

[4] Ben Sassi, Rakia 2021-. Compiler vs. Interpreter: Know the Difference and When to Use Each of Them. Medium.

https://betterprogramming.pub/compiler-vs-interpreter-d0a12ca1c1b6

[5] Basso, Matteo. July 22, 2019,. “Using WebAssembly with Web Workers.” *SitePen*, 22

https://www.sitepen.com/blog/using-webassembly-with-web-workers

[6] Betts, Anaïs. 28 October 2017. "Building Hybrid Applications with Electron - Slack Engineering." *Slack Engineering* -, https://slack.engineering/building-hybrid-applications-with-electron/.

[7] "Bubble Sort (With Code in Python/C++/Java/C)." *Programiz*, https://www.programiz.com/dsa/bubble-sort.

[8] Clark L., 2017. A crash course in just-in-time (JIT) compilers, Available: https://hacks.mozilla.org/2017/02/a-crash-course-in-just-in-time-jit-compilers/

[9] Emscripten. "Emscripting a C library to Wasm." *web.dev*, 5 March 2018, https://web.dev/emscripting-a-c-library/#bonus_content_running_something_simple_the_hard_way.

[10] Emscripten"Optimising Code — Emscripten 3.1.16-git (dev) documentation.", https://emscripten.org/docs/optimizing/Optimizing-Code.html.

[11] Franziska Hinkelmann. Aug 16, 2017. https://medium.com/dailyjs/understanding-v8s-bytecode-317d46c94775

[12] Flanagan, David. *JavaScript: The Definitive Guide*. O'Reilly Media, Incorporated, 2011.

[13] Floros, Orestis, et al. "Superoptimization of WebAssembly bytecode." March 2022, https://dl.acm.org/doi/abs/10.1145/3397537.3397567?casa_token=VMA5nK7f4NcAAAAA:hl4y1gcm45nvF4dvwBlhtyEPWg1O6qOgyAW95XWUktUX0F5YBVxd-_S2wTML8qtA7EV9Nh9bzDXbVA.

[14] "Francesco Rizzi's Journal | Algorithms in JavaScript: Bubble Sort, Quicksort, Mergesort." *Francesco Rizzi*, 17 September 2020, https://frarizzi.science/journal/web-engineering/algorithms-in-javascript-bubblesort-quicksort-mergesort..

[15] Gallant, Gerard. *WebAssembly in Action*. Manning, 2019.

"Get Started With WebAssembly In 5 Minutes (Step By Step Tutorial)." *Code Boxx*, 6 March 2022, https://code-boxx.com/get-started-webassembly-beginners/.

[16] Gurgone, Giuseppe. "Optimising WebAssembly Startup Time." *PSPDFKit*, https://pspdfkit.com/blog/2018/optimize-webassembly-startup-performance/

[17] Haas, Andreas, et al. *Bringing the Web up to Speed with WebAssembly*.

[18] Haverbeke, Marijn. "Eloquent JavaScript." *A Modern Introduction to Programming*, 2014.

[19] Herman D., Wagner L. and Zakai A., 10th October 2020 2014. http://asmjs.org/spec/latest/.

[20] Herrera, David, et al. "WebAssembly and JavaScript Challenge: Numerical program performance using modern browser technologies and devices." 2018.

[21] Herrera, D., Chen, H. Lavoie, E 2018. - WebAssembly and JavaScript Challenge: Numerical program performance using modern browser technologies and devices

[22] Hyunwoo Park , Sungkook Kim , Jung-Geun Park , Soo-Mook. December 2018. Reusing the Optimised Code for JavaScript Ahead-of-Time Compilation, https://doi.org/10.1145/3291056

[23] LLVM 24 June 2022 https://llvm.org/

[24] Macedo, Joao De, et al. "On the Runtime and Energy Performance of WebAssembly." *Is WebAssembly superior to JavaScript yet?*, 05 2021.

"mbasso/awesome-wasm: Curated list of awesome things regarding WebAssembly (wasm) ecosystem." *GitHub*, https://github.com/mbasso/awesome-wasm#compilers.

[25] Mathiasbynens. 14th June 2018

JavaScript engine fundamentals: Shapes and Inline Caches,

https://mathiasbynens.be/notes/shapes-ics

[26] McFadden, Brian, et al. "Security Chasms of WASM." 3 August 2018.

[27] MDN, Jul 6, 2022,

https://developer.mozilla.org/en-US/docs/Web/JavaScript

[28] MDN Web Docs. 27.02.2021 - WebAssembly.instantiateStreaming()..

https://developer.mozilla.org/enUS/docs/Web/JavaScript/Reference/Global_Objects/WebAssembly/instantiateStreaming .

[29] MDN Web Docs. 27.02.2021 - WebAssembly.Memory() constructor.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/

[30] MDN Web Docs 28.02.2021 - WebAssembly.Table() constructor. 2021.

https://developer.mozilla.org/enUS/docs/Web/JavaScript/Reference/Global_Objects/WebAssembly/Table/Table

[31] MDN Web Docs. Clark, Lin. "Memory in WebAssembly (and why it's safer than you think)." *Mozilla Hacks*, 19 July 2017,

https://hacks.mozilla.org/2017/07/memory-in-webassembly-and-why-its-safer-than-you-think/.

[32] MDN Web Docs. "Compiling a New C/C++ Module to WebAssembly - WebAssembly | 15 June 2022, https://developer.mozilla.org/en-US/docs/WebAssembly/C_to_wasm.

[33] Meurer, Benedikt. "JavaScript engine fundamentals: Shapes and Inline Caches · Mathias Bynens." *Mathias Bynens*, 14 June 2018, https://mathiasbynens.be/notes/shapes-ics.

[34] Parthasarathi, Ranjani. 20.02.2021 - Instruction Set Architecture. Computer Architecture

https://www.cs.umd.edu/~meesh/411/CAonline/chapter/instruction-set-architecture/index.html

[35] Peacock, Ashley. "How Fast is WebAssembly Versus JavaScript? | by Ashley Peacock." *Better Programming*, https://betterprogramming.pub/how-fast-is-webassembly-versus-javascript-bc0eca058a54.

[36] "Performance measurement of data exchange between webassembly and JavaScript." 前端知识, 31 January 2022, https://qdmana.com/2022/01/202201310430408801.html.

[37] Rakia Ben Sassi. Jan 19, 2021 https://betterprogramming.pub/compiler-vs-interpreter-d0a12ca1c1b6

[38] Reiser, Micha. "Accelerate JavaScript applications by cross-compiling to WebAssembly." https://dl.acm.org/doi/abs/10.1145/3141871.3141873?casa_token=imxt7eYwpDcAAAAA:BX_Zi9tcsgYMsKtQC8xNTRbWk99vuNSmsroaGctVciMOCq2JRSqSEyICUOVdtrJUKuZ3D_tZPB8FhA.

[39] Rourke, Mike. *Learn WebAssembly*. Packt Publishing, 2018.

[40] Rossberg, Andreas. 02.03.2021. WebAssembly Specification. https://webassembly.github.io/spec/core/_download/WebAssembly.pdf

[41] Sandhu, P., Herrera, D., & Hendren, L. Proceeding Series. 2020. https://doi.org/10.1145/3237009.3237020

[42] Sandhu, P. Verbrugge, C. 2020. . A fully structure-driven performance analysis of sparse matrix-vector multiplication. ICPE 2020 - Proceedings of the ACM/SPEC International Conference on Performance Engineering, 108{119. https://doi.org/10.1145/3358960.3379131

[43] Sassi, Rakia Ben. "Compiler vs. Interpreter in Programming." *Better Programming*, https://betterprogramming.pub/compiler-vs-interpreter-d0a12ca1c1b6. []Serdaru, Silviu. "WebAssembly vs JavaScript: A Performance Comparison." *OPTASY*, 7 December 2018, https://medium.com/@OPTASY.com/webassembly-vs-javascript-is-wasm-faster-than-js-when-does-javascript-perform-better-db86d2ecf2cc.

https://hackernoon.com/essential-guide-to-image-processing-with-webassembly-q11u33hq.

[44] Sletten, Brian. *WebAssembly: The Definitive Guide*. O'Reilly Media, Incorporated, 2021. "State of the Web: WebAssembly." *Byte Dev*, 21 February 2022, https://byteofdev.com/posts/webassembly/.

[45] Stack Overflow."webassembly.instantiate versus Module." *Stack Overflow*, 15 January 2019, https://stackoverflow.com/questions/54202840/webassembly-instantiate-versus-module.

[46] Shu-yu Guo, Michael Ficarra, Kevin Gibbons 13th edition, June 2022. https://262.ecma-international.org/

[47] 25 Shubhdwiv. Nov, 2021 https://www.geeksforgeeks.org/what-happens-inside-javascript-engine/

[48] Surma.dev. 2019/5/28 - By Compiling C to WebAssembly without Emscripten —. https://surma.dev/things/c-to-webassembly/.

[49] TechDifference. November 21, 2019 https://techdifferences.com/difference-between-compiler-and-assembler.html

[50] Tulka, Tomas. "Essential Guide to Image Processing with WebAssembly." *HackerNoon*, 21 February 2021,

[51] W3C. 19.03.2021 - World Wide Web Consortium brings a new language to the Web as

WebAssembly becomes a W3C Recommendation.

https://www.w3.org/2019/12/pressrelease-wasm-rec.html.en

[52] WebAssembly Web API. 27.02.2021 -. WebAssembly GitHub.

https://webassembly.github.io/spec/web-api/index.html.

[53] "WebAssembly.instantiate() - JavaScript | MDN." *MDN*, 11 July 2022,

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/WebAssembly/instantiate#parameters.

[54] "What Are The Names Of Other JavaScript Engines Besides V8? – Spritely.net."

Spritely.net, 11 June 2022,

https://www.spritely.net/what-are-the-names-of-other-javascript-engines-besides-v8/.

[55] Zakai, Alon. 2017 - Why WebAssembly Is Faster Than asm.js.. Mozilla Hacks.

https://hacks.mozilla.org/2017/03/why-webassembly-is-fasterthan-asm-js/

[56] Zhao, ZhengxU 2011- Protecting Against Address Space Layout Randomization (ASLR) Compromises and Return-to-Libc Attacks Using Network Intrusion Detection Systems

Παράρτημα A.1 - Πίνακες Αποτελεσμάτων σε Laptop

Numerical Computing Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)	Native JavaScript (WebWorker)
Fibonacci Numbers	27.159	30.487	6.185	20.154	18.254	27.238
Multiply Doubles	3.931	4.515	3.925	3.954	3.918	4.522
Multiply Integers	3.939	4.427	1.476	1.490	1.485	3.928
Multiply 2 Double Arrays	1.087	2.846	1.301	1.820	1.925	1.027
Prime Numbers From	6.956	5.624	5.024	5.547	5.514	6.847
Geometric Mean	5.017	6.278	2.977	4.128	4.077	5.085
Geometric Mean of Chrome	4.474					

Πίνακας A.1: Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Laptop με φυλλομετρητή Chrome

Browser: Chrome - hot (102.0.5005.63 - 64bit)

CPU: INTEL Core i5 7200U - 2.50GHz

RAM: 8 GB - 1064 MHz

Operating System: Windows 10 pro

Numerical Computing Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)	Native JavaScript (WebWorker)
Fibonacci Numbers	52.924	23.134	4.320	11.223	10.824	56.800

Multiply Doubles	3.923	6.387	3.955	3.947	3.918	3.929
Multiply Integers	3.939	4.535	1.524	1.478	1.475	4.070
Multiply 2 Double	1.346	2.825	1.396	2.101	2.122	1.331
Prime Numbers	7.662	6.493	6.229	6.484	6.440	7.719
Geometric Mean	6.098	6.575	2.957	3.891	3.858	6.222
Geometric Mean of Mozilla	4.721					

Πίνακας A.2: Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Laptop με φυλλομετρητή Firefox

Browser: Mozilla- hot (101.0.1 - 64bit)

CPU: INTEL Core i5 7200U - 2.50GHz

RAM: 8 GB - 1064 MHz

Numerical Computing Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)	Native JavaScript (WebWorker)
Fibonacci Numbers	32.270	29.428	4.154	11.485	16.521	26.985

Multiply Doubles	3.999	6.394	3.929	3.954	4.019	3.939
Multiply Integers	3.924	4.405	1.477	1.490	1.605	3.919
Multiply 2 Double	1.058	2.828	1.268	1.820	1.783	1.011
Prime Numbers	6.964	5.812	5.089	5.547	5.703	6.899
Geometric Mean	5.180	6.712	2.743	3.688	4.045	4.927
Geometric Mean of Opera	4.374					

Πίνακας A.3: Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Laptop με φυλλομετρητή Opera

Browser: Opera- hot (87.0.4390.45 - 64bit)

CPU: INTEL Core i5 7200U - 2.50GHz

RAM: 8 GB - 1064 MHz

Sorting Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)

BubbleSort array	1.038	0.322	0.128	0.125	0.112
QuickSort Double array	0.432	0.544	0.335	0.172	0.176
QuickSort Int array	0.416	.5036	0.175	0.152	0.152
Reverse Array	1.081	1.254	0.065	0.067	0.054
Geometric Mean	0.670	0.585	0.148	0.121	0.112
Geometric Mean of Chrome	0.239				

Πίνακας Α.4: Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Laptop με φυλλομετρητή Chrome

Browser: Chrome - hot (102.0.5005.63 - 64bit)

CPU: INTEL Core i5 7200U - 2.50GHz

RAM: 8 GB - 1064 MHz

Sorting Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)
BubbleSort array	2.262	0.398	0.177	0.165	0.145
QuickSort Double array	0.412	0.704	0.447	0.263	0.255
QuickSort Int array	0.312	0.689	0.345	0.242	0.247
Reverse Array	1.086	1.252	0.076	0.079	0.064
Geometric Mean	0.749	0.701	0.213	0.169	0.155
Geometric Mean of Chrome	0.311				

Πίνακας Α.5: Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Laptop με φυλλομετρητή Firefox

Browser: Mozilla- hot (101.0.1 - 64bit)

CPU: INTEL Core i5 7200U - 2.50GHz

RAM: 8 GB - 1064 MHz

Sorting Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)
BubbleSort array	1.088	0.321	0.110	0.123	0.115
QuickSort Double array	0.454	0.543	0.346	0.156	0.178
QuickSort Int array	0.442	0.542	0.322	0.174	0.152
Reverse Array	1.086	1.251	0.092	0.055	0.056
Geometric Mean	0.697	0.586	0.183	0.116	0.114
Geometric Mean of Chrome	0.250				

Πίνακας Α.6: Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Laptop με φυλλομετρητή Opera

Browser: Opera- hot (87.0.4390.45 - 64bit)

CPU: INTEL Core i5 7200U - 2.50GHz

RAM: 8 GB - 1064 MHz

Image Processing Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)
Grayscale	0.075	0.106	0.045	0.062	0.076
Blurring	0.654	0.742	0.295	0.292	0.368
Threshold	0.242	0.234	0.154	0.149	0.159
Geometric Mean	0.228	0.264	0.126	0.139	0.164
Geometric Mean of Chrome	0.176				

Πίνακας Α.7: Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Laptop με φυλλομετρητή Chrome

Browser: Chrome - hot (102.0.5005.63 - 64bit)

CPU: INTEL Core i5 7200U - 2.50GHz

RAM: 8 GB - 1064 MHz

Sorting Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)
Grayscale	0.076	0.122	0.040	0.062	0.077
Blurring	0.719	0.801	0.326	0.314	0.374
Threshold	0.335	0.274	0.161	0.192	0.197
Geometric Mean	0.263	0.299	0.128	0.155	0.178
Geometric Mean of Mozilla	0.194				

Πίνακας Α.8: Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Laptop με φυλλομετρητή Firefox

Browser: Mozilla- hot (101.0.1 - 64bit)

CPU: INTEL Core i5 7200U - 2.50GHz

RAM: 8 GB - 1064 MHz

Sorting Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)
Grayscale	0.074	0.108	0.045	0.068	0.074
Blurring	0.708	0.750	0.288	0.297	0.373
Threshold	0.251	0.249	0.134	0.152	0.164
Geometric Mean	0.236	0.272	0.120	0.145	0.165
Geometric Mean of Chrome	0.179				

Πίνακας Α.9: Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Laptop με φυλλομετρητή Opera

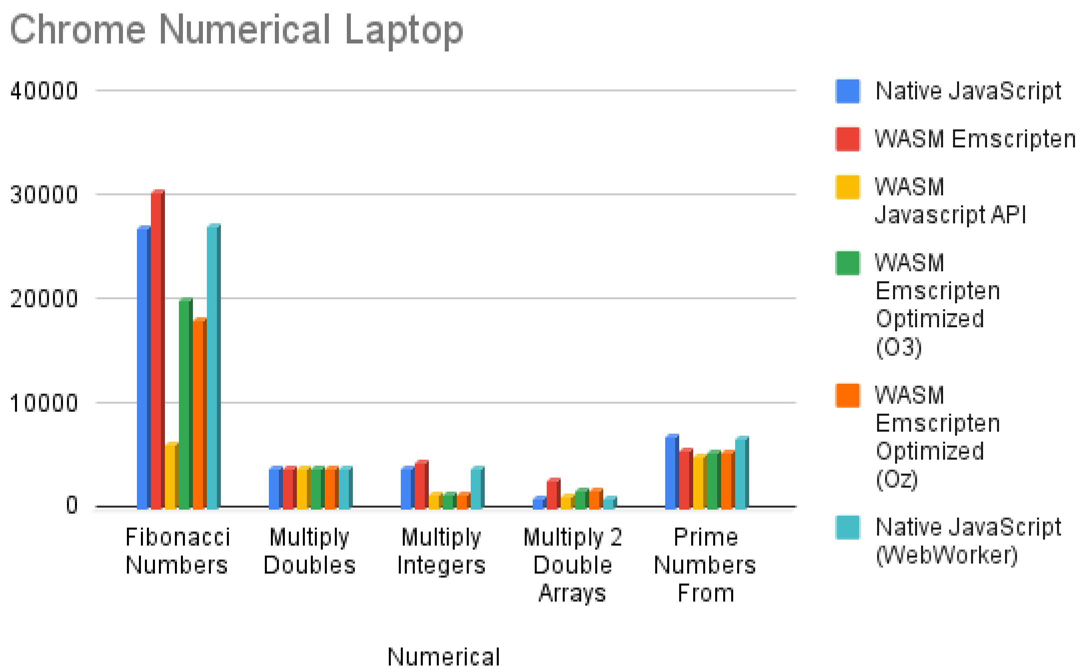
Browser: Opera- hot (87.0.4390.45 - 64bit)

CPU: INTEL Core i5 7200U - 2.50GHz

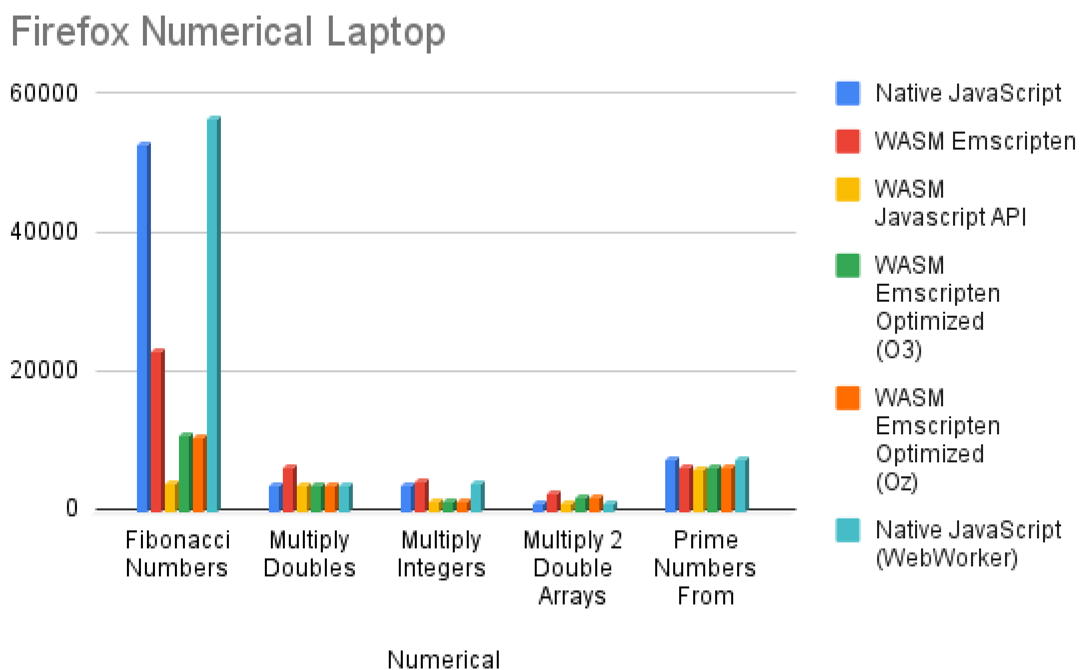
RAM: 8 GB - 1064 MHz

Παράρτημα Α.2 - Διαγράμματα Αποτελεσμάτων σε Laptop

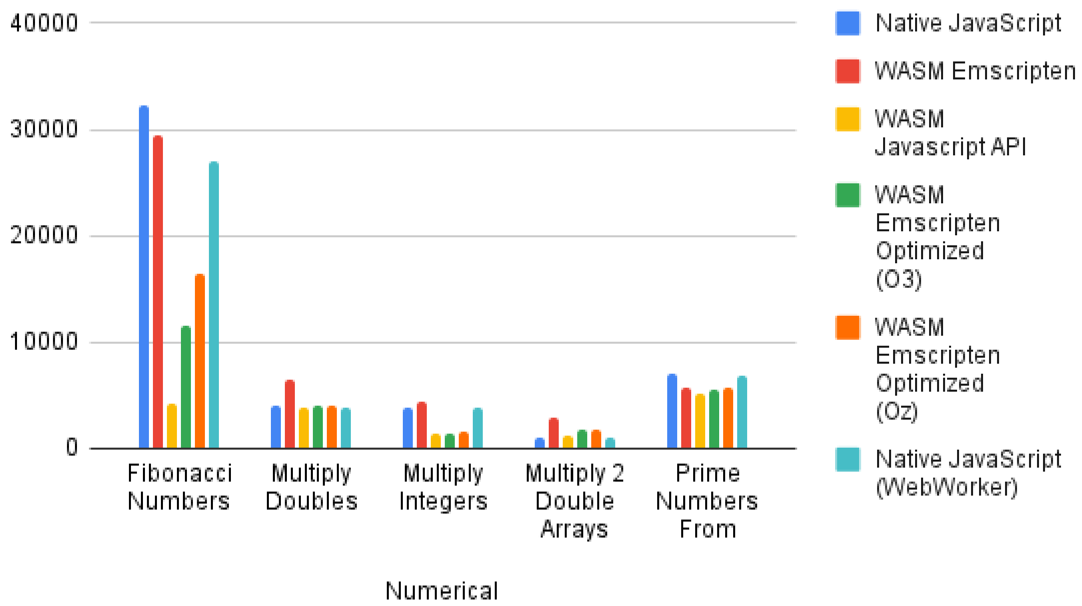
Εικόνα Α.1



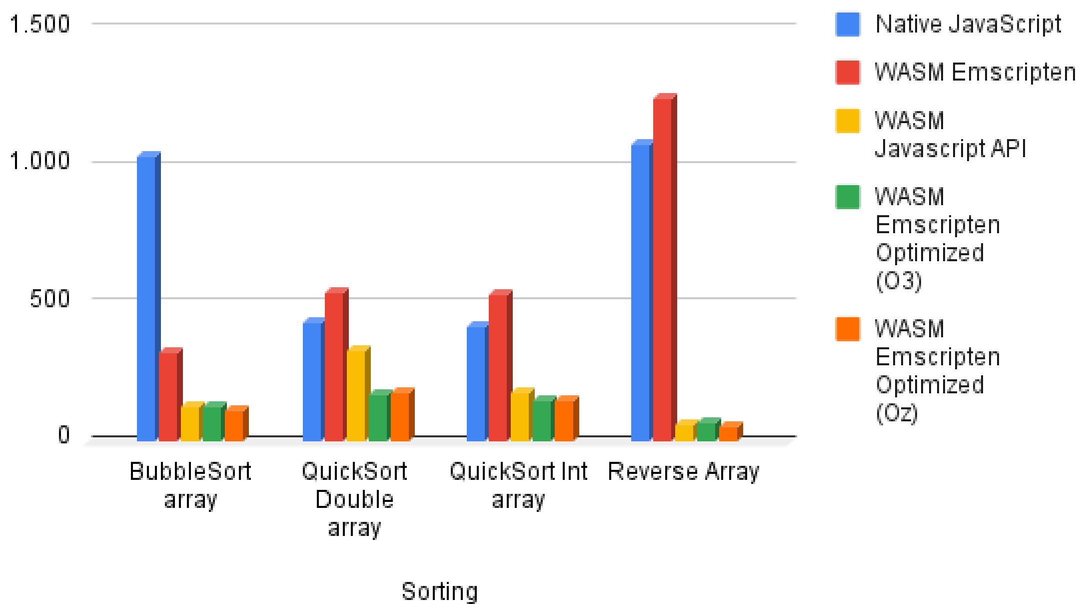
Εικόνα Α.2



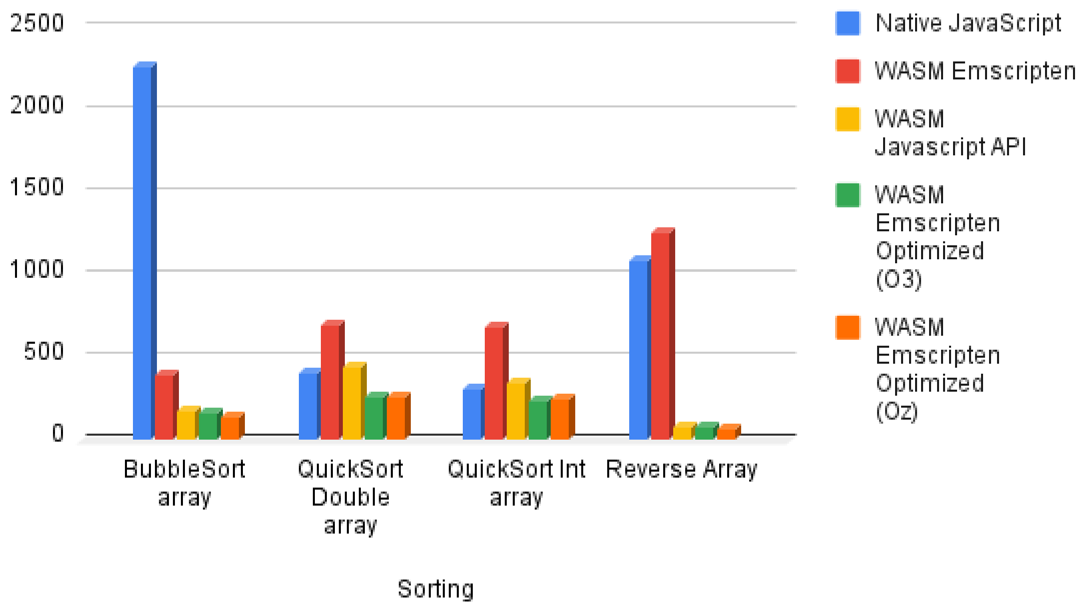
Εικόνα Α.3



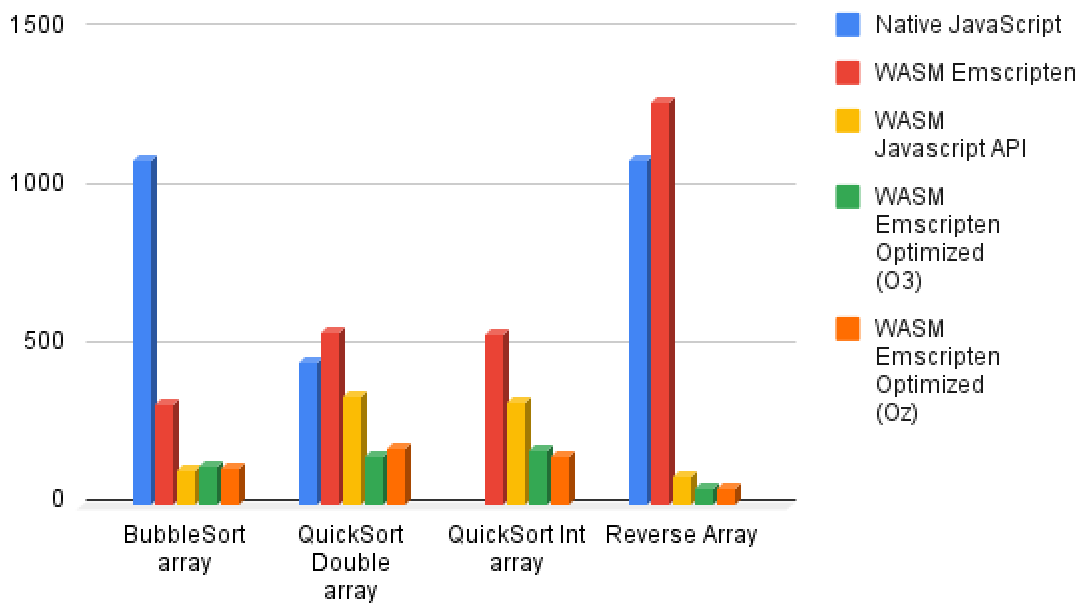
Εικόνα Α.4



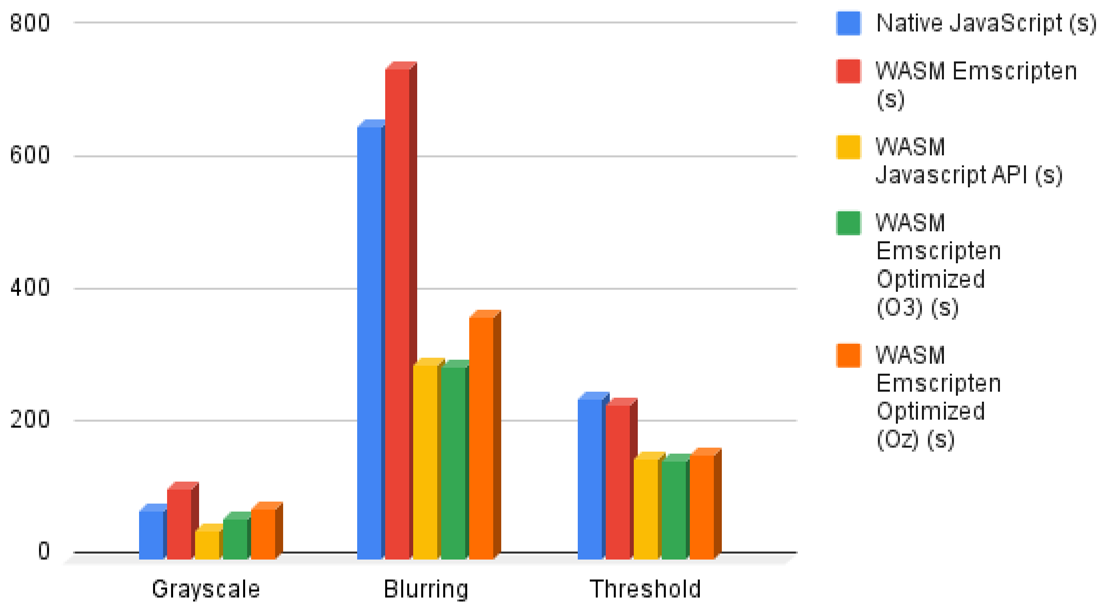
Εικόνα A.5



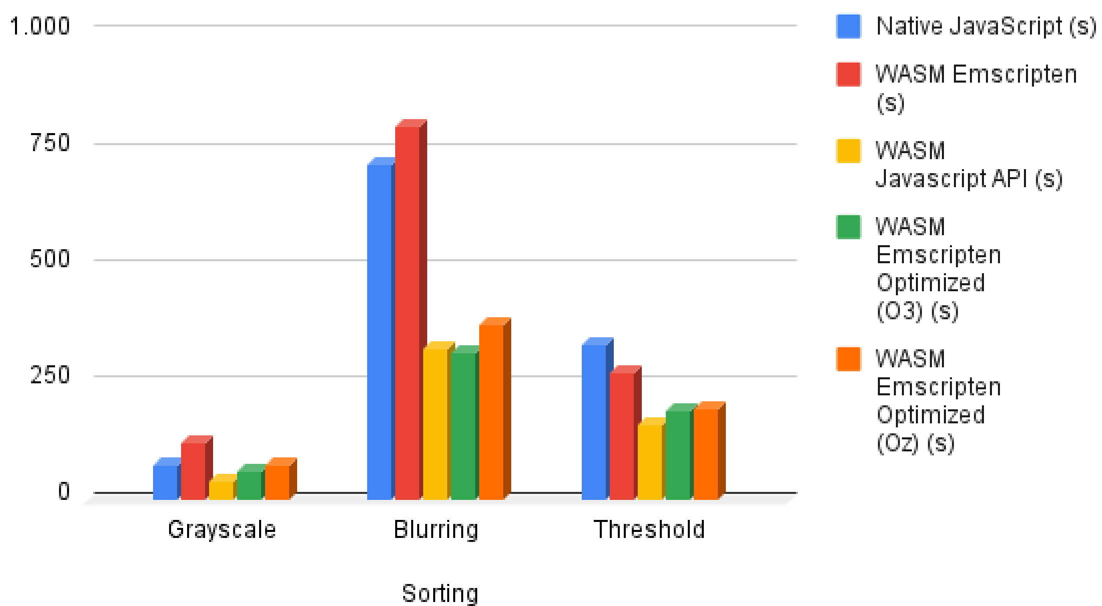
Εικόνα A.6



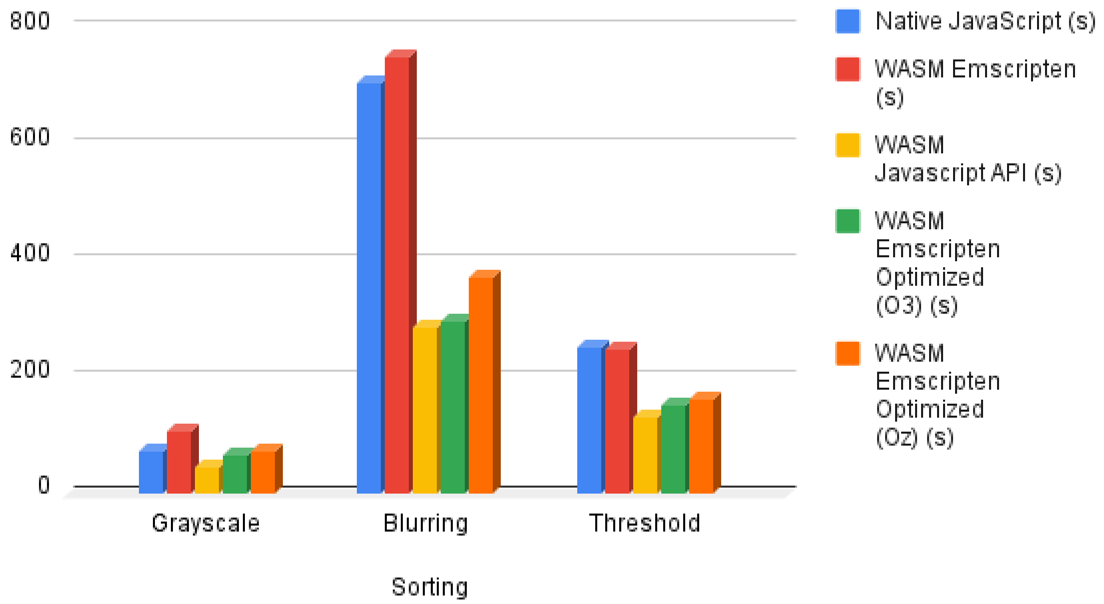
Εικόνα Α.7



Εικόνα Α.8



Εικόνα Α.9



Παράρτημα B.1 - Πίνακες και Διαγράμματα Mobile

Numerical Computing Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)	Native JavaScript (WebWorker)
Fibonacci Numbers	16.232	36.315	3.550	11.152	10.886	16.825
Multiply Doubles	4.533	8.483	4.528	4.648	4.556	4.565
Multiply Integers	4.526	6.259	1.724	1.720	1.702	4.556
Multiply 2 Double Arrays	0.663	4.550	1.827	2.425	2.477	0.614
Prime Numbers	1.486	3.382	1.937	3.364	3.317	1.458
Geometric Mean	3.185	7.842	2.502	3.735	3.700	3.156
Geometric Mean of Chrome	3.737					

Πίνακας B.1: Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Mobile με φυλλομετρητή Chrome

Browser: Chrome - hot (102.0.5005.63 - 64bit)

CPU: A13 Bionic Chip - 2650 MHz

RAM: 6GB RAM

Numerical Computing Algorithm	**Native JavaScript**	**WASM Emscripten**	**WASM JavaScript API**	**WASM Emscripten Optimised (O3)**	**WASM Emscripten Optimised (Oz)**	**Native JavaScript (**WebWorker**)**
Fibonacci Numbers	16.280	36.455	3.559	11.800	10.927	16.246
Multiply Doubles	4.547	8.443	4.535	4.648	4.546	4.555
Multiply Integers	4.559	6.254	1.733	1.720	1.703	4.567
Multiply 2 Double Arrays	0.651	4.544	1.914	2.425	2.499	0.701
Prime Numbers	1.463	3.477	1.939	3.364	3.354	1.420
Geometric Mean	3.172	7.88	2.53	3.777	3.716	3.201
Geometric Mean of Chrome	**3.763**					

Πίνακας B.2: Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Mobile με φυλλομετρητή Firefox

Browser: Mozilla- hot (101.0.1 - 64bit)

CPU: A13 Bionic Chip - 2650 MHz

RAM: 6GB RAM

Numerical Computing Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)	Native JavaScript (WebWorker)
Fibonacci Numbers	16.286	36.544	3.532	15.631	16.067	16.124
Multiply Doubles	4.534	8.462	4.537	4.556	4.540	4.525
Multiply Integers	4.546	6.260	1.703	1.702	1.700	4.588
Multiply 2 Double Arrays	0.664	4.586	1.820	2.577	2.499	0.624
Prime Numbers	1.467	3.372	1.949	3.210	3.245	1.455
Geometric Mean	3.183	7.856	2.495	3.983	3.985	3.138
Geometric Mean of Chrome	3.820					

Πίνακας B.3: Συγκριτική Ανάλυση Αριθμητικών Υπολογισμών σε Mobile με φυλλομετρητή Opera

Browser: Opera- hot (87.0.4390.45 - 64bit)
CPU: A13 Bionic Chip - 2650 MHz
RAM: 6GB RAM

Sorting Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)
BubbleSort array	1.926	0.425	0.148	0.145	0.140
QuickSort Double array	0.312	1.041	0.435	0.170	0.164

QuickSort Int array	0.482	1.083	0.361	0.265	0.241
Reverse Array	0.168	0.276	0.029	0.058	0.058
Geometric Mean	0.469	0.603	0.161	0.139	0.133
Geometric Mean of Chrome	0.242				

Πίνακας B.4: Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Mobile με φυλλομετρητή Chrome

Browser: Chrome - hot (102.0.5005.63 - 64bit)
CPU: A13 Bionic Chip - 2650 MHz
RAM: 6GB RAM

Sorting Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)
BubbleSort array	1930	0.425	0.147	0.152	0.147
QuickSort Double array	0.444	1.099	0.456	0.174	0.174
QuickSort Int array	0.485	1.058	0.364	0.269	0.263
Reverse Array	0.192	0.239	0.027	0.059	0.057
Geometric	0.531	0.586	0.160	0.143	0.139

Mean					
Geometric Mean of Chrome	0.250				

Πίνακας B.5: Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Mobile με φυλλομετρητή Firefox
Browser: Mozilla- hot (101.0.1 - 64bit)
CPU: A13 Bionic Chip - 2650 MHz
RAM: 6GB RAM

Sorting Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)
BubbleSort array	1.923	0.428	0.155	0.184	0.146
QuickSort Double array	0.453	1.113	0.431	0.142	0.113
QuickSort Int array	0.478	1.092	0.374	0.242	0.248
Reverse Array	0.172	0.252	0.024	0.068	0.057
Geometric Mean	0.517	0.601	0.156	0.143	0.123
Geometric Mean of Chrome	0.243				

Πίνακας B.6: Συγκριτική Ανάλυση Αλγορίθμων Ταξινόμησης σε Mobile με φυλλομετρητή Opera
Browser: Opera- hot (87.0.4390.45 - 64bit)
CPU: A13 Bionic Chip - 2650 MHz
RAM: 6GB RAM

Image Processing Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)
Grayscale	0.061	0.112	0.025	0.033	0.034
Blurring	0.211	0.454	0.231	0.243	0.271
Threshold	0.091	0.197	0.145	0.107	0.056
Geometric Mean	0.105	0.215	0.094	0.095	0.080
Geometric Mean of Chrome	0.110				

Πίνακας B.7: Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Mobile με φυλλομετρητή Chrome
Browser: Google Chrome
CPU: A13 Bionic Chip - 2650 MHz
RAM: 6GB RAM

Image Processing Algorithm	Native JavaScript	WASM Emscripten	WASM JavaScript API	WASM Emscripten Optimised (O3)	WASM Emscripten Optimised (Oz)
Grayscale	0.051	0.118	0.026	0.035	0.036
Blurring	0.235	0.425	0.226	0.254	0.264
Threshold	0.107	0.197	0.147	0.109	0.056
Geometric Mean	0.108	0.214	0.095	0.098	0.081
Geometric Mean of Chrome	**0.111**				

Πίνακας B.8: Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Mobile με φυλλομετρητή Firefox
Browser: Mozilla- hot (101.0.1 - 64bit)
CPU: A13 Bionic Chip - 2650 MHz
RAM: 6GB RAM

Image Processing Algorithm	**Native JavaScript**	**WASM Emscripten**	**WASM JavaScript API**	**WASM Emscripten Optimised (O3)**	**WASM Emscripten Optimised (Oz)**
Grayscale	0.054	0.104	0.026	0.035	0.036
Blurring	0.228	0.449	0.237	0.246	0.277
Threshold	0.093	0.198	0.149	0.109	0.055
Geometric Mean	0.104	0.209	0.097	0.097	0.081
Geometric Mean of Chrome	**0.110**				

Πίνακας B.9: Συγκριτική Ανάλυση Επεξεργασίας Εικόνας σε Mobile με φυλλομετρητή Opera

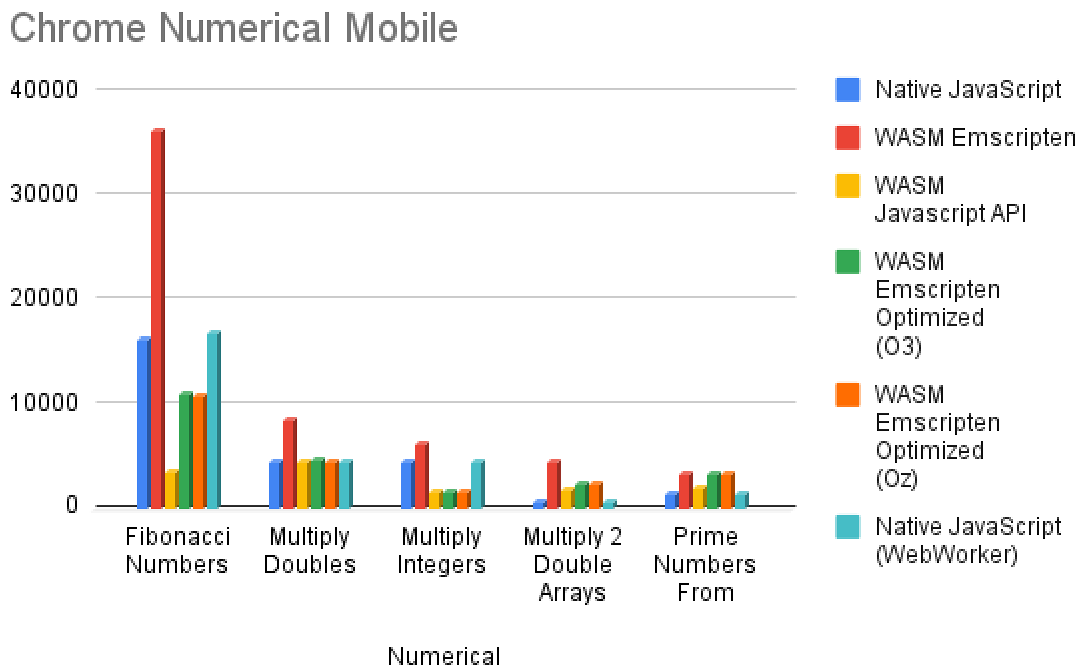
Browser: Opera- hot (87.0.4390.45 - 64bit)

CPU: A13 Bionic Chip - 2650 MHz

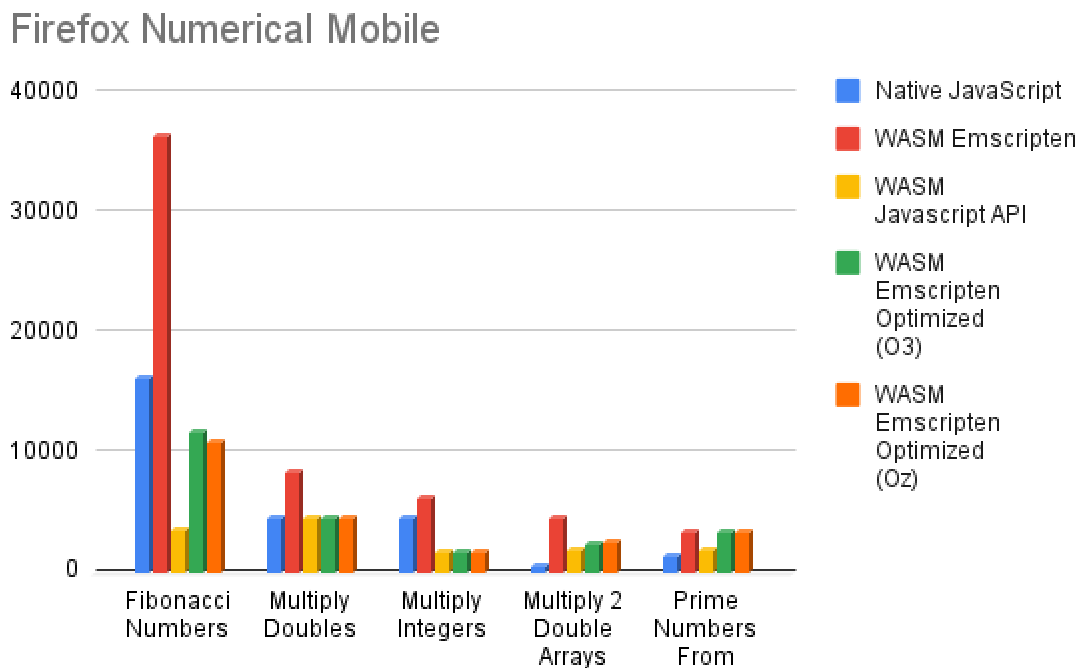
RAM: 6GB RAM

Παράρτημα Β.2 - Διαγράμματα Αποτελεσμάτων σε Mobile

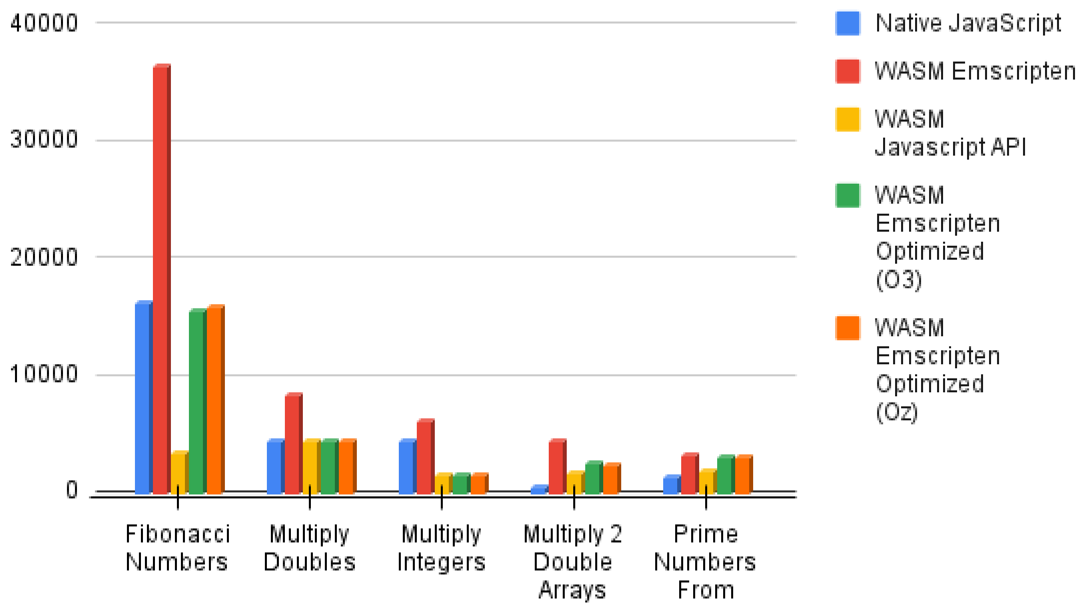
Εικόνα Β.1



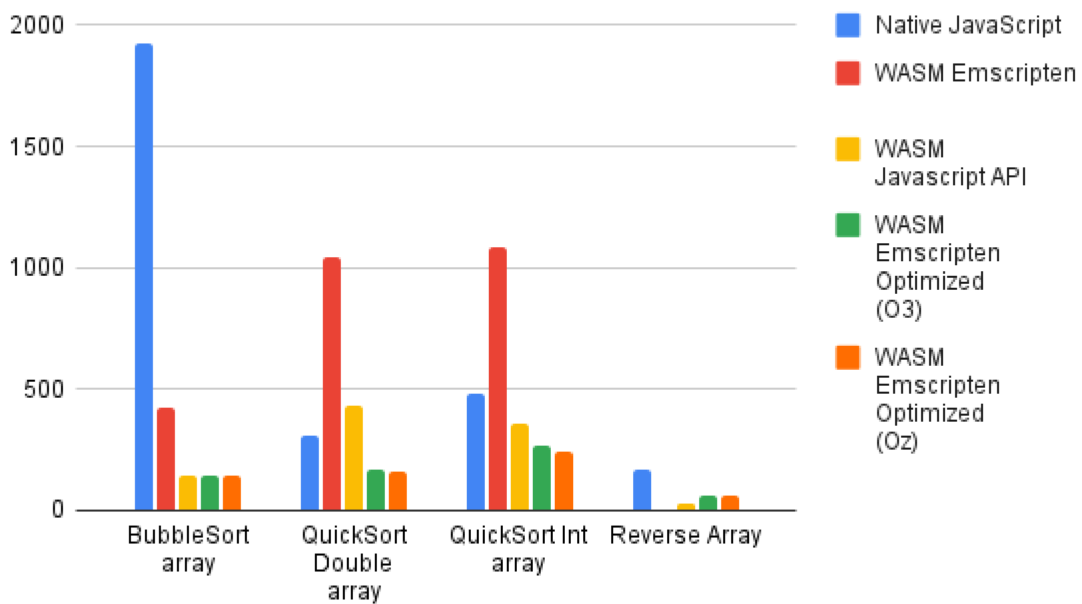
Εικόνα Β.2



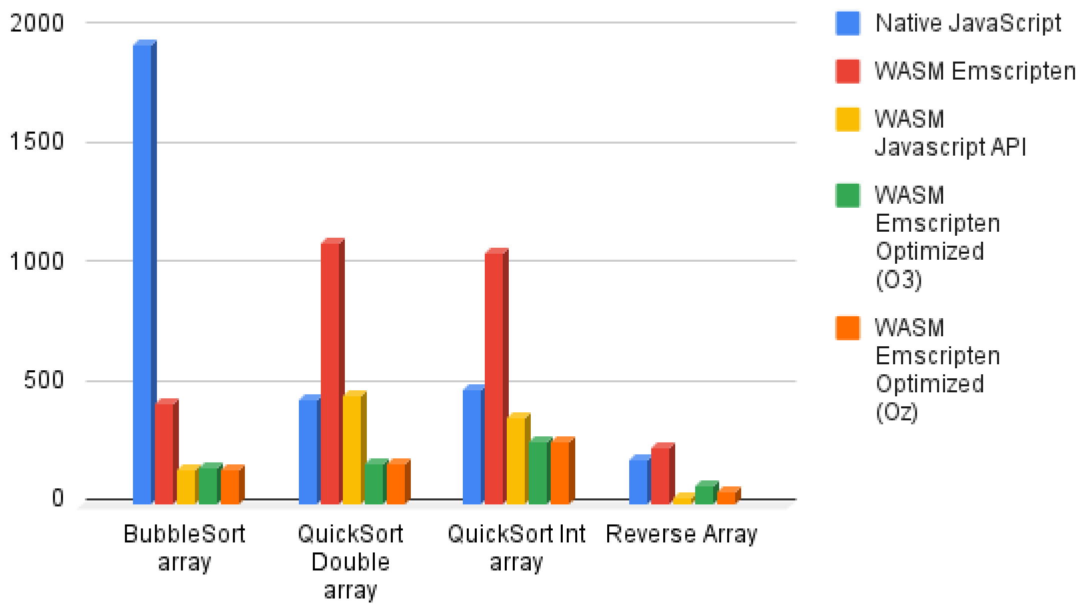
Εικόνα Β.3



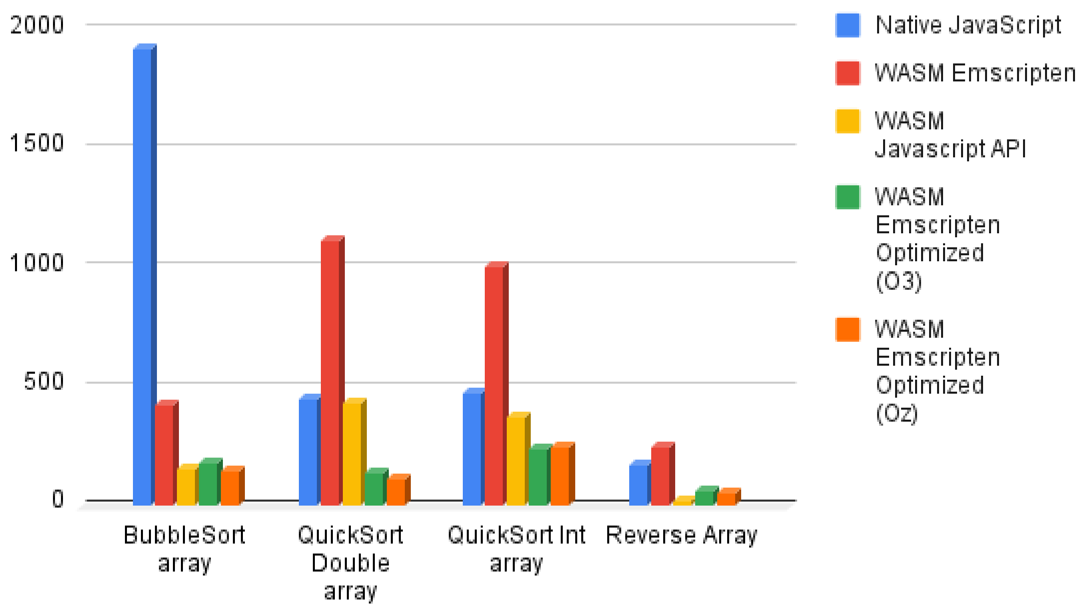
Εικόνα Β.4



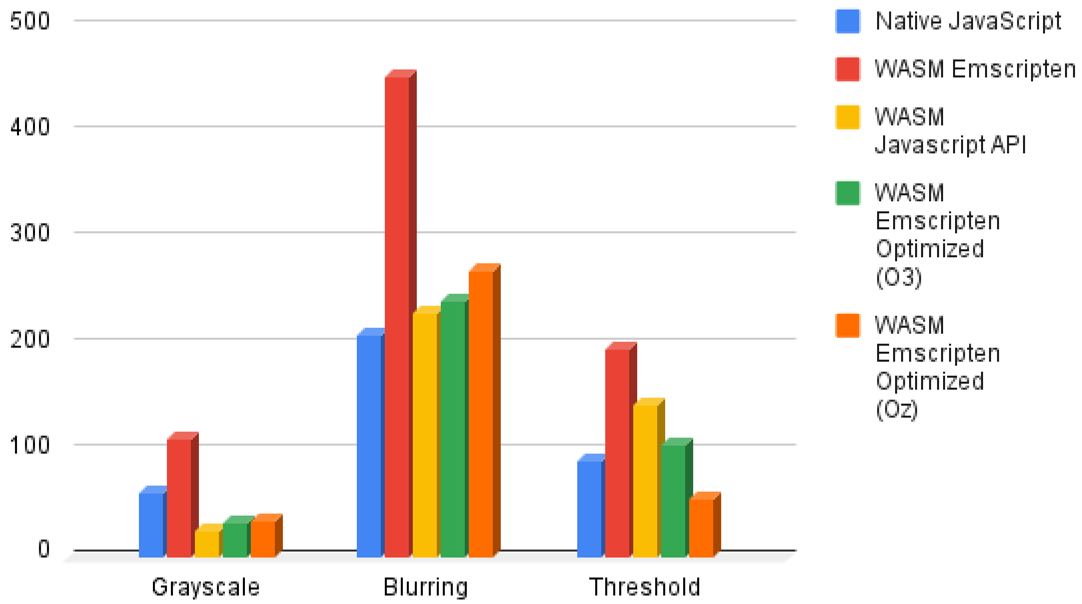
Εικόνα Β.5



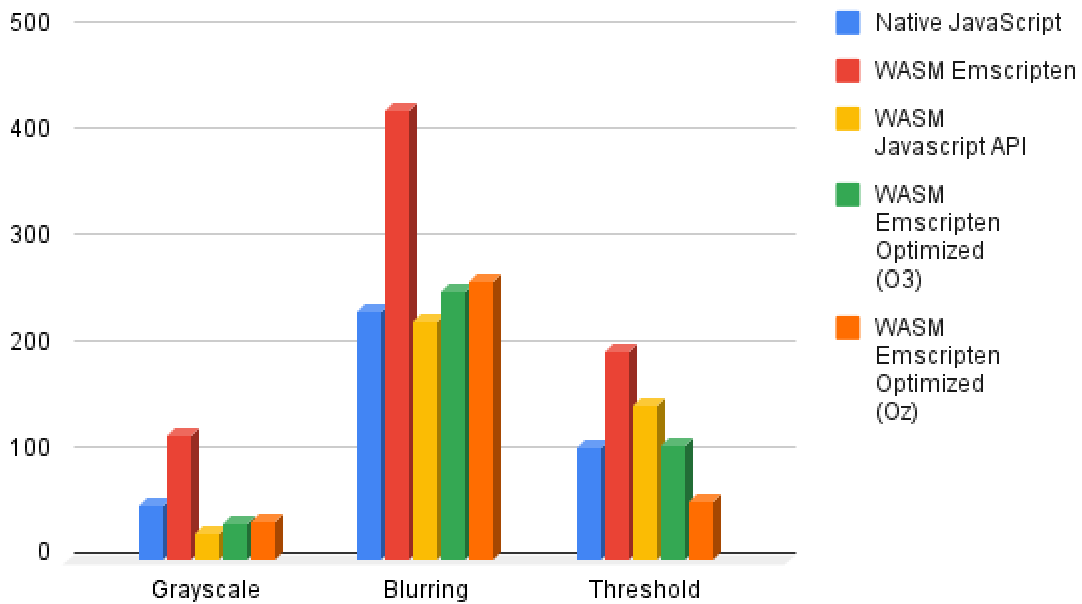
Εικόνα Β.6



Εικόνα Β.7



Εικόνα Β.8



Εικόνα Β.9