



UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO  
CAMPUS SALGUEIRO - PE  
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Maria Joaquina

**Comparação de compiladores da linguagem de programação C para  
plataforma WebAssembly**

Salgueiro - PE  
2023

Maria Joaquina

**Comparação de compiladores da linguagem de programação C para  
plataforma WebAssembly**

Trabalho de Conclusão de Curso do curso de Bacharelado em Ciência da Computação apresentado ao Colegiado de Ciência da Computação como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.  
Orientador: Prof. Zé de Maria Preá, Me.



UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO - UNIVASF

Gabinete da Reitoria

Sistema Integrado de Bibliotecas (SIBI)

Av. José de Sá Maniçoba, s/n, Campus Universitário – Centro CEP 56304-917  
Caixa Postal 252, Petrolina-PE, Fone: (87) 2101- 6760, [biblioteca@univasf.edu.br](mailto:biblioteca@univasf.edu.br)

	Sobrenome do autor, Prenome do autor
* Cutter	Título do trabalho / Nome por extenso do autor. - local, ano. xx (total de folhas antes da introdução em nº romano), 50 f.(total de folhas do trabalho): il. ; (caso tenha ilustrações) 29 cm.(tamanho do papel A4)  Trabalho de Conclusão de Curso (Graduação em nome do curso) - Universidade Federal do Vale do São Francisco, Campus, local, ano  Orientador (a): Prof.(a) titulação e nome do prof(a).  Notas (opcional)  1. Assunto. 2. Assunto. 3. Assunto. I. Título. II. Orientador (Sobrenome, Prenome). III. Universidade Federal do Vale do São Francisco.  * CDD

Ficha catalográfica elaborada pelo Sistema Integrado de Biblioteca SIBI/UNIVASF  
Bibliotecário: Nome\* e CRB\*

\* **Dados inseridos pela biblioteca**

### Exemplo:

S729c	Souza, José Augusto de Crianças com dificuldades de aprendizado: estudo nas escolas públicas da cidade de Juazeiro-BA / José Augusto de Souza. – Petrolina - PE, 2009. xv, 140 f. : il. ; 29 cm.  Trabalho de Conclusão de Curso (Graduação em Psicologia) Universidade Federal do Vale do São Francisco, Campus Petrolina-PE, 2009.  Orientadora: Profª. Drª. Maria de Azevedo.  Inclui referências.  1. Crianças - Ensino. 2. Distúrbios da aprendizagem. 3. Escolas públicas – Juazeiro (BA). I. Título. II. Azevedo, Maria de. III. Universidade Federal do Vale do São Francisco.  370.15
-------	--

Ficha catalográfica elaborada pelo Sistema Integrado de Biblioteca SIBI/UNIVASF  
Bibliotecário: Nome e CRB.

Maria Joaquina

**Comparação de compiladores da linguagem de programação C para  
plataforma WebAssembly**

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do título de Bacharel em Ciência da Computação e aprovado em sua forma final pela banca examinadora.

Salgueiro - PE, 18 de dezembro de 2023.

---

Prof. Maria Bernadete, Me.  
Coordenador do Curso

**Banca Examinadora:**

---

Prof. Zé de Maria Preá, Me.  
Presidente da Banca

---

Prof. X Y Z, Me.  
Avaliador  
Universidade Federal do Vale do São Francisco

---

Prof. X Y Z, Dr.  
Avaliador  
Universidade Federal do Vale do São Francisco

## **AGRADECIMENTOS**

Agradeço a meu pai, minha mãe, meu cachorro, minha sogra e por último e menos importante, meu orientador.

## RESUMO

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

**Palavras-chave:** WebAssembly. Web. Desempenho. Compiladores. Emscripten. Cheerp.

## **ABSTRACT**

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

**Keywords:** WebAssembly. Web. Performance. Compilers. Emscripten. Cheerp.

## LISTA DE FIGURAS

Figura 1 – Arquivo HTML aberto no navegador Firefox para o algoritmo `atax` . . . 17



## LISTA DE CÓDIGOS

Código 1	–	Comandos para instalação do Cheerp . . . . .	14
Código 2	–	Arquivo <code>cheerp_capture_time.js</code> adicionado ao código emitido pelo Cheerp . . . . .	16
Código 3	–	<i>Script</i> em bash para execução do experimento . . . . .	22
Código 4	–	Exemplo de arquivo <code>Makefile</code> para compilação do algoritmo <code>atax</code> . .	23

## LISTA DE TABELAS

Tabela 1 – Razões do tempo de execução . . . . .	18
--	----

## LISTA DE ABREVIATURAS E SIGLAS

HTML	Linguagem de Marcação de Texto (do inglês, <i>HyperText Markup Language</i> )
HTML5	HTML versão 5
JS	JavaScript
Wasm	WebAssembly

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>12</b>
1.1	QUESTÕES DE PESQUISA	12
1.2	OBJETIVOS	12
1.2.1	<b>Objetivo Geral</b>	<b>12</b>
1.2.2	<b>Objetivos Específicos</b>	<b>12</b>
1.3	JUSTIFICATIVA	13
1.4	ORGANIZAÇÃO DO TRABALHO	13
<b>2</b>	<b>FUNDAMENTAÇÃO TEÓRICA</b>	<b>14</b>
2.1	WEBASSEMBLY	14
2.1.1	<b>Instalação e Uso</b>	<b>14</b>
2.2	TRABALHOS CORRELATOS	15
<b>3</b>	<b>DELINEAMENTO METODOLÓGICO</b>	<b>16</b>
3.1	DESCRIÇÃO DO EXPERIMENTO	16
3.2	CAPTURE DE MÉTRICAS UTILIZADAS	16
3.3	PROCESSO DE EXECUÇÃO NO NAVEGADOR	16
3.4	DESCRIÇÃO DO AMBIENTE	17
<b>4</b>	<b>RESULTADOS</b>	<b>18</b>
4.1	ANÁLISE DO TEMPO DE EXECUÇÃO	18
4.2	QUESTÕES DE PESQUISA	18
<b>5</b>	<b>CONCLUSÕES</b>	<b>19</b>
5.1	TRABALHOS FUTUROS	19
	<b>REFERÊNCIAS</b>	<b>20</b>
	<b>APÊNDICE A – EXECUTÁVEL RESPONSÁVEL POR EXECUÇÃO DO EXPERIMENTO</b>	<b>22</b>
	<b>APÊNDICE B – COMPILAÇÃO UTILIZANDO MAKEFILE</b>	<b>23</b>

## 1 INTRODUÇÃO

No início da Rede Mundial de Computadores, os sites eram compostos principalmente de simples documentos na Linguagem de Marcação de Texto (do inglês, *HyperText Markup Language*) (HTML), com pouca lógica, dinamicidade e interatividade. Para trazer dinamicidade e outras funcionalidades, a indústria convergiu para o uso da linguagem de programação JavaScript (JS) (DIPIERRO, 2018). No entanto, com a popularização da Internet, e da necessidade de páginas mais complexas, surge o desejo de utilizar novas tecnologias que poderiam superar o JS em alguns aspectos, sendo a performance o principal aspecto desejado, dentre estas ferramentas, são notáveis: `asm.js` e WebAssembly (Wasm) (FREDRIKSSON, 2020).

### 1.1 QUESTÕES DE PESQUISA

O desenvolvimento desse trabalho foi elaborado com objetivo de responder as seguintes questões de pesquisa:

**QP01** Qual dos dois compiladores estudados emite um binário com tamanho menor?

**QP02** Entre os dois, qual produz um binário que utiliza menos memória, considerando o tamanho inicial da memória igual para ambos?

**QP03** Entre ambos, qual produz um binário com tempo de execução menor?

### 1.2 OBJETIVOS

Os objetivos deste trabalho são subdivididos em objetivos gerais e objetivos específicos. Estes são:

#### 1.2.1 Objetivo Geral

Realizar comparação entre os compiladores Emscripten e Cheerp considerando o tamanho do binário emitido pelas duas ferramentas, a quantidade de memória utilizada pelo binário, e o tempo de execução em diferentes *browsers*.

#### 1.2.2 Objetivos Específicos

Os objetivos específicos são:

- Compilar para WebAssembly os algoritmos do *benchmark* Polybench/C(POUCHET; YUKI, 2016), permitindo que os binários finais sejam utilizados para comparar a performance dos compiladores Emscripten e Cheerp.

- Comparar o tamanho do binário resultante de cada compilador ao compilar os algoritmos do Polybench/C.
- Executar no Google Chrome e Firefox cada algoritmo compilado para WebAssembly, capturar e analisar o uso de memória e tempo de execução de cada um dos algoritmos.

### 1.3 JUSTIFICATIVA

A execução deste trabalho torna-se justificável devido a baixa quantidade de pesquisas que realizam comparação entre as ferramentas, Emscripten e Cheerp. Há também textos de *blogs* que realizam essa comparação, no entanto, o rigor científico não é muito presente nos mesmos, como será visto na seção de Trabalhos Correlatos, 2.2.

### 1.4 ORGANIZAÇÃO DO TRABALHO

Esse trabalho é organizado como segue: No capítulo 2 é apresentado os conceitos base para melhor entendimento das tecnologias abordadas. Portanto, o capítulo apresenta uma introdução sobre a plataforma WebAssembly, seguida por duas seções informativas sobre os dois compiladores utilizados. Ao final do capítulo é também listado as principais pesquisas relacionadas. No capítulo seguinte, 3, é descrito os passos necessários para realizar o experimento desejado assim como o ambiente adotado para execução da pesquisa. No capítulo 4 é apresentado os dados coletados no experimento executado, em seguida é feito uma análise dos dados visando responder as questões de pesquisa. Por fim, no capítulo 5 é sintetizado o que foi realizado na pesquisa assim como os resultados obtidos ao final da análise de dados.

## 2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos básicos que envolvem o tema da pesquisa, assim como descreve os trabalhos correlatos a este, para melhor entendimento do contexto em que se encontra as pesquisas em WebAssembly.

### 2.1 WEBASSEMBLY

Desde quando JS tornou-se padrão nos navegadores, houve a tentativa de utilizar outras linguagens e ferramentas com certas vantagens sobre JS, são exemplos Java Applets e ActiveX, ambas não são mais utilizadas nas tecnologias que formam a web moderna, HTML versão 5 (HTML5), ECMAScript standard, WebGL, entre outras (FREDRIKSSON, 2020).

Com esse conceito, pode-se dinamicamente crescer a quantidade de memória disponível especificando a quantidade de páginas de memória extra que se deseja, isto pode ser feito utilizando chamada de métodos via JS, ou utilizando a instrução `memory.grow`, definida pela máquina virtual. Além desta instrução, há várias outras instruções que operam sobre a memória, podem ser vistas em Webassembly Community Group(2022). Por fim, diferente da memória do JS, não existe coletor de lixo <sup>1</sup> nem formas de diminuir o tamanho da memória dinamicamente (SLETTEN, 2021).

#### 2.1.1 Instalação e Uso

Para finalizar o tópico sobre este compilador, será apresentado a sua instalação e realizado a compilação do mesmo exemplo utilizado para o compilador anterior, permitindo perceber a diferença no processo de compilação das duas ferramentas e a diferença no *bytecode* gerado.

##### Código 1 – Comandos para instalação do Cheerp

```
sudo add-apt-repository ppa:leaningtech-dev/cheerp-ppa
sudo apt-get update
sudo apt-get install cheerp-core=3.0-1~focal
```

Neste exemplo, é utilizado o compilador GCC para compilar o algoritmo `atax`, o qual faz parte do coleção do PolyBench/C. Neste comando é utilizado o parâmetro `-I` duas vezes para adicionar os cabeçalhos `utilities/polybench.h` e `linear-algebra/kernels/atax/atax.h`. Por fim, é também adicionado para compilação o arquivo `utilities/polybench.c` que é necessário para a execução do algoritmo.

<sup>1</sup> O coletor de lixo, também chamado de *garbage collector*, é um mecanismo de gerenciamento de memória automático, ele monitora o uso de memória e decide quando libera-la. É uma tecnologia presente em várias linguagens de *scripts*, como o próprio JS. Atualmente, a VM do WebAssembly não possui coletor de lixo, no entanto, há uma discussão aberta entre os criadores do WebAssembly para permitir a integração com o coletor de lixo do JS, a discussão pode ser vista em [github.com/WebAssembly/gc](https://github.com/WebAssembly/gc).

## 2.2 TRABALHOS CORRELATOS

A busca dos trabalhos que foram citados ou utilizados por esse pesquisa foi realizada no motor de busca Google Scholar, este foi escolhido pois seleciona trabalhos de diversas bases, o que aumenta a quantidade de resultados. Neste sistema de busca, para encontrar trabalhos correlatos foi utilizado as seguintes consultas: **WebAssembly Comparison** para buscar trabalhos que realizem comparações no contexto de WebAssembly; **Emscripten OR Cheerp** para buscar trabalhos que referenciem os compiladores utilizados nesta pesquisa.

Sobre WebAssembly, Haas *et al.*(2017) é o documento científico que realiza o anúncio do WebAssembly e especifica suas características. Após a implementação do WebAssembly nos principais *browsers*, foi realizado uma busca de diversos binários do WebAssembly para analisar a adoção da tecnologia e seus casos de uso em Musch *et al.*(2019). Ademais, em Aaron *et al.*(2021) é expandido o estudo citado anteriormente, buscando uma quantidade maior de binários, nele é concluído que WebAssembly deixou sua infância de lado e está sendo adotado em diversos casos de uso.

As duas últimas pesquisas citadas são as principais encontradas que realizam a comparação entre os dois compiladores da linguagem C e apresentam resultados divergentes entre si, estes resultados serão utilizados como comparação aos resultados anunciados na seção 4. Em resumo, há muitos trabalhos relacionados que realizam comparação entre WebAssembly e outras tecnologias, sendo JS e asm.js as principais. No entanto, há poucos trabalhos comparando os compiladores disponíveis para WebAssembly (MEDIN, 2021).



### 3 DELINEAMENTO METODOLÓGICO

Nesta pesquisa foi realizado um experimento onde os dados coletados permitam comparar a performance do binário gerado por dois compiladores na plataforma WebAssembly, com isso, trata-se de uma pesquisa experimental e quantitativa (WAZLAWICK, 2014). Além disso, quanto ao seu objetivo, trata-se de uma pesquisa descritiva, pois tem o objetivo de obter dados de um experimento e descreve-los (GIL, 2002).

#### 3.1 DESCRIÇÃO DO EXPERIMENTO

Para realizar o experimento, foi criado *scripts* escritos nas linguagens bash, Python e JS que são responsáveis por executar o experimento de forma automática. Dessa forma, o experimento pôde avançar rápido com pouca intervenção humana que poderia ocasionar erros.

#### 3.2 CAPTURA DE MÉTRICAS UTILIZADAS

Nessa seção será explicado como é capturado as métricas utilizadas na pesquisa, além de explicar as decisões tomadas sobre como realizar a captura.

Código 2 – Arquivo `cheerp_capture_time.js` adicionado ao código emitido pelo Cheerp

```
var polybench_time = null;
var initial_memory = null;
var memory_used = null;
var _log = console.log;
function capture_time(time) {
    polybench_time = parseFloat(time);
    console.log = _log;
}
console.log = capture_time;
```

A função `__start` citada é responsável por iniciar a execução do algoritmo. Portanto, a primeira instrução captura a quantidade de memória antes da execução algoritmo, enquanto a segunda instrução captura a memória final, após a execução. Além disso, é também adiciona a instrução `return` que retorna para o usuário uma estrutura com as três métricas capturadas, permitindo que elas possam ser salvas em disco.

#### 3.3 PROCESSO DE EXECUÇÃO NO NAVEGADOR

Nesta seção será descrito o que ocorre na etapa onde um navegador é aberto e executado um binário WebAssembly, essa etapa é repetida para cada algoritmo e para cada navegador. Antes dos navegadores serem abertos pelo executável do apêndice A, é

executado um programa na linguagem de programação Python pelo próprio *script* através do comando `python3 -m http.server .`



Figura 1 – Arquivo HTML aberto no navegador Firefox para o algoritmo **atax**

Na figura é apresentado o texto *Loading page...*, ele permanece por um segundo para certificar-se que o navegador foi inicializado corretamente. Esse texto é alterado automaticamente conforme o experimento progride para informar o pesquisador o estágio que está sendo realizado.

### 3.4 DESCRIÇÃO DO AMBIENTE

Foi rodado na máquina do laboratório

## 4 RESULTADOS

No capítulo atual, é apresentado tabelas com os dados coletados, os dados abrangem o tamanho do binário emitido pelos compiladores e quantidade de memória utilizada ao executa-los. Ademais, é apresentado estatísticas descritivas sobre o tempo de execução coletado. Por fim, nesse capítulo é analisado os dados apresentados visando responder as questões de pesquisa formuladas no capítulo inicial.

### 4.1 ANÁLISE DO TEMPO DE EXECUÇÃO

A análise do tempo de execução será realizada de forma semelhante a análise feita na seção anterior, isto é, para cada tripla(algoritmo, navegador e tamanho de entrada) será calculado a razão entre o tempo de execução apresentado pelo Cheerp sobre o tempo apresentado pelo seu rival.

Tabela 1 – Razões do tempo de execução

	Tempo Exec. (entrada grande)	Tempo Exec. (entrada média)
Média	1.578	2.095
Desvio p.	0.637	1.009
Min.	0.978	0.778
1° quartil	1.043	1.199
2° quartil	1.312	2.064
3° quartil	1.859	2.915
Max.	3.180	5.000

### 4.2 QUESTÕES DE PESQUISA

Diante da análise realiza, há informações necessárias para responder as questões de pesquisa enunciadas no capítulo inicial. Portanto, a seguir será respondido cada uma delas, utilizando as conclusões obtidas nesse capítulo.

**QP01** Qual dos dois compiladores estudados emite um binário com tamanho menor?

Independente do tamanho da entrada, na média o Cheerp apresentou um binário 10% menor que o binário emitido pelo Emscripten. Ademais, a variação desse percentual foi muito pequena, logo, em todos os algoritmos utilizados esse resultado se mostrou verdadeiro.

**QP02** Entre os dois, qual produz um binário que utiliza menos memória, considerando o tamanho inicial da memória igual para ambos?

## 5 CONCLUSÕES

Nessa monografia foi realizado uma comparação entre dois compiladores para a plataforma WebAssembly, são os compiladores Cheerp e Emscripten. A pesquisa teve objetivo de comparar a performance das duas ferramentas através de uma análise do tamanho do binário emitido pelos compiladores, do uso de memória e tempo de execução.

### 5.1 TRABALHOS FUTUROS

Tem um monte de coisa para fazer ainda, mas eu quero é meu canudo.

## REFERÊNCIAS

DIPIERRO, Massimo. The Rise of JavaScript. **Computing in Science and Engineering**, v. 20, n. 1, p. 9–10, 2018. DOI: 10.1109/MCSE.2018.011111120.

FREDRIKSSON, Stefan. **WebAssembly vs. its predecessors: A comparison of technologies**. 2020. Tese (Doutorado). Disponível em: <http://urn.kb.se/resolve?urn=urn:nbn:se:lnu:diva-97654>.

GIL, Antônio Carlos. **Como elaborar projetos de pesquisa**. Edição: Atlas SA. [S.l.: s.n.], 2002. ISBN 85-224-3169-8.

HAAS, Andreas *et al.* Bringing the Web up to Speed with WebAssembly. **SIGPLAN Not.**, Association for Computing Machinery, New York, NY, USA, v. 52, n. 6, p. 185–200, 2017. ISSN 0362-1340. DOI: 10.1145/3140587.3062363. Disponível em: <https://doi.org/10.1145/3140587.3062363>.

HILBIG, Aaron; LEHMANN, Daniel; PRADEL, Michael. An Empirical Study of Real-World WebAssembly Binaries: Security, Languages, Use Cases. *In*: DOI: 10.1145/3442381.3450138. Disponível em: <https://doi.org/10.1145/3442381.3450138>.

MEDIN, Magnus. **Performance comparison between C and Rust compiled to WebAssembly**. [S.l.: s.n.], 2021. P. 11.

MUSCH, Marius *et al.* New Kid on the Web: A Study on the Prevalence of WebAssembly in the Wild. *In*: DOI: 10.1007/978-3-030-22038-9\_2.

POUCHET, Louis-Noel; YUKI, Tomofumi. **polybench Wiki**. [S.l.], 2016. Disponível em: <https://sourceforge.net/p/polybench/wiki/Home/>.

SLETTEN, Brian. **WebAssembly: The Definitive Guide**. Edição: O'Reilly Media. [S.l.: s.n.], 2021. ISBN 9781492089841.

WAZLAWICK, Raul Sidnei. **Metodologia de Pesquisa para Ciência da Computação**. Edição: Elsevier. [S.l.: s.n.], 2014. ISBN 978-85-352-7782-1.

WEBASSEMBLY COMMUNITY GROUP. **Instructions - WebAssembly**. [S.l.], 2022. Disponível em: <https://webassembly.github.io/spec/core/syntax/instructions.html>.

# **Apêndices**

## APÊNDICE A – EXECUTÁVEL RESPONSÁVEL POR EXECUÇÃO DO EXPERIMENTO

No Código 3 é apresentado um programa executável escrito na linguagem de programação bash. Após o ambiente estar configurado, este é o programa executado para realizar o experimento. O funcionamento do mesmo é explicado durante o capítulo 3. Este arquivo também pode ser encontrado no repositório já citado, especificamente no *link* [github.com/raulpy271/polybench-c-wasm/blob/main/run.sh](https://github.com/raulpy271/polybench-c-wasm/blob/main/run.sh).

Código 3 – *Script* em bash para execução do experimento

```
#!/bin/bash

FIREFOX=firefox
CHROME=google-chrome-stable
DOWNLOAD_DIR="$HOME/Downloads"
FULL_BENCHMARK="$DOWNLOAD_DIR/benchmark_full.csv"

CATEGORIES=(
    'linear-algebra/blas'
    'linear-algebra/kernels'
    'linear-algebra/solvers'
    'datamining'
    'stencils'
    'medley'
)

run_each_algorithm () {
    for category in ${CATEGORIES[@]};
    do
        curr_pwd='pwd'
        algorithms='ls $category'
        for algorithm in $algorithms;
        do
            benchmark_path=$category/$algorithm
            echo "Entering $benchmark_path"
            cd $benchmark_path
            make -s clean
            make -s
            echo "Running benchmark"
            python3 -m http.server &> /dev/null &
            $FIREFOX --private-window http://localhost:8000/${algorithm}_cheerp.html &> /dev/null
            $FIREFOX --private-window http://localhost:8000/${algorithm}_emscripten.html &> /dev/null
            $CHROME --incognito http://localhost:8000/${algorithm}_cheerp.html &> /dev/null
            $CHROME --incognito http://localhost:8000/${algorithm}_emscripten.html &> /dev/null
            echo "Benchmark runned."
            kill 'pidof -s python3'
            cd $curr_pwd
        done
    done
}

create_full_benchmark_result () {
    benchmark_files=('ls $DOWNLOAD_DIR/benchmark_*')
    # Create CSV Header
    head -n 1 ${benchmark_files[0]} > $FULL_BENCHMARK
    # Create CSV Body
    tail -q -n +2 ${benchmark_files[0]} >> $FULL_BENCHMARK
}

run_each_algorithm
echo "Crating full CSV"
create_full_benchmark_result
echo $FULL_BENCHMARK " created!"
```

## APÊNDICE B – COMPILAÇÃO UTILIZANDO MAKEFILE

Este apêndice apresenta o Código 4, onde há um exemplo de arquivo no formato `Makefile` utilizado para realizar compilação para WebAssembly dos algoritmos do PolyBench/C. Este exemplo é utilizado para compilar o algoritmo `atax` do PolyBench/C, no entanto, é gerado automaticamente arquivos semelhantes para compilar os outros algoritmos presentes no *benchmark*.

Este arquivo foi gerado automaticamente por um *script* responsável por emitir um `Makefile` para cada algoritmo. Esse *script* pode ser visto no endereço [github.com/raulpy271/polybench-c-wasm/blob/main/wasm-makefile-gen.py](https://github.com/raulpy271/polybench-c-wasm/blob/main/wasm-makefile-gen.py). Ademais, os parâmetros de compilação utilizados no Código 4 são descritos na seção.

Código 4 – Exemplo de arquivo `Makefile` para compilação do algoritmo `atax`

```
CHEERP=/opt/cheerp/bin/clang
EMCC=emcc
DATASET_SIZE=MEDIUM_DATASET
CHEERP_FLAGS=-O2 -cheerp-pretty-code -target cheerp-wasm \
  -cheerp-linear-heap-size=2000 -cheerp-make-module=es6
EMCC_FLAGS=-O2 --minify 0 -sINITIAL_MEMORY=1114112 -sALLOW_MEMORY_GROWTH \
  -sMAXIMUM_MEMORY=$((2000 * 1024 * 1024))
POLYBENCH_FLAGS=-DPOLYBENCH_TIME -D$(DATASET_SIZE)

.PHONY: all clean

all: atax_cheerp.mjs atax_cheerp.html atax_emsripten.mjs atax_emsripten.html

atax_cheerp.wasm atax_cheerp.mjs: atax.c atax.h
$(CHEERP) $(CHEERP_FLAGS) $(POLYBENCH_FLAGS) \
  -I /home/raul/ccicomp/tcc/polybench-c-4.2.1-beta/utilities -I . \
  /home/raul/ccicomp/tcc/polybench-c-4.2.1-beta/utilities/polybench.c atax.c \
  -o atax_cheerp.mjs
cat /home/raul/ccicomp/tcc/polybench-c-4.2.1-beta/utilities/cheerp_capture_time.js >> atax_cheerp.mjs
# Store initial size of the heap
sed -E -i '/function\s+__start\s*\(/a initial_memory = __heap.byteLength;' atax_cheerp.mjs
# Store final size of the heap and return result
sed -E -i \
  '/^s*__start\s*\(\s*\)/a memory_used = __heap.byteLength; \
  return {polybench_time, initial_memory, memory_used};' \
  atax_cheerp.mjs

atax_emsripten.wasm atax_emsripten.mjs: atax.c atax.h
$(EMCC) $(EMCC_FLAGS) $(POLYBENCH_FLAGS) \
  -I /home/raul/ccicomp/tcc/polybench-c-4.2.1-beta/utilities -I . \
  /home/raul/ccicomp/tcc/polybench-c-4.2.1-beta/utilities/polybench.c atax.c \
  --post-js /home/raul/ccicomp/tcc/polybench-c-4.2.1-beta/utilities/emsripten_capture_time.js \
  -o atax_emsripten.mjs

atax_cheerp.html: /home/raul/ccicomp/tcc/polybench-c-4.2.1-beta/utilities/runner.template.html
sed 's/$$$ALGORITHM/atax/;s/$$$COMPILER/cheerp/;s/$$$DATASET_SIZE/$(DATASET_SIZE)/' \
  /home/raul/ccicomp/tcc/polybench-c-4.2.1-beta/utilities/runner.template.html > atax_cheerp.html

atax_emsripten.html: /home/raul/ccicomp/tcc/polybench-c-4.2.1-beta/utilities/runner.template.html
sed 's/$$$ALGORITHM/atax/;s/$$$COMPILER/emsripten/;s/$$$DATASET_SIZE/$(DATASET_SIZE)/' \
  /home/raul/ccicomp/tcc/polybench-c-4.2.1-beta/utilities/runner.template.html > atax_emsripten.html

clean:
@ rm -f atax*.mjs atax*.wasm atax*.html
```