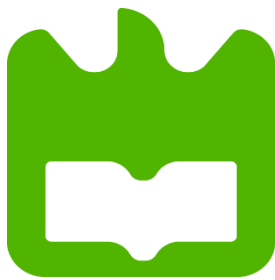


Universidade de Aveiro

# Arquiteturas de Alto Desempenho

Assignment nr.1



universidade de aveiro

Guilherme Duarte (107766), Guilherme Andrade (107696)

Departamento de Eletrónica, Telecomunicações e Informática

November 30, 2024

Introdução .....	5
1.1 Convert the AVX code to AVX2 code .....	6
2.1 Search function AVX.....	8
2.2 Search function AVX2.....	9
2.3 Search function CUDA .....	10
<b>Search with AVX and SIMD instructions .....</b>	<b>14</b>
Definições e Inclusões .....	14
Gerador de Números Aleatórios (PRNG) xorshift32 com AVX.....	15
Geração de Caracteres ASCII Imprimíveis Aleatórios.....	15
Função Principal de Pesquisa de "DETI coins" .....	16
<b>Search with AVX and SIMD implemented in OPENMP .....</b>	<b>18</b>
Inicialização do PRNG xorshift32 com AVX e OpenMP .....	18
Geração de Caracteres ASCII Imprimíveis Aleatórios.....	19
Função Principal de Pesquisa com OpenMP .....	20
Finalização e Saída .....	21
<b>Search with AVX2 and SIMD instructions .....</b>	<b>22</b>
Constantes Definidas.....	22
Inicialização do PRNG xorshift32 com AVX2.....	22
Geração de Caracteres ASCII Imprimíveis Aleatórios.....	23
Função Principal de Pesquisa.....	23
Finalização e Saída .....	25
<b>Search with AVX2 and SIMD instructions using OPENMP .....</b>	<b>26</b>
Constantes Definidas.....	26
Inicialização do PRNG xorshift32 com AVX2 e OpenMP .....	26
Geração de Caracteres ASCII Imprimíveis Aleatórios.....	27
Função Principal de Pesquisa com OpenMP .....	28
<b>Search with AVX on a Server with many clients.....</b>	<b>29</b>
Estrutura Geral do Código .....	29
Inclusões e Definições .....	30
Variáveis Globais .....	31
Funções Principais.....	31
Funções de Comunicação .....	34
Sincronização e Paralelismo .....	34
Detalhes Adicionais .....	35
Fluxo de Execução do Servidor .....	35
<b>deti_coins_client0 ( AVX Search ) .....</b>	<b>35</b>
Estrutura Geral do Código .....	36

Inclusões e Definições .....	36
Funções Principais.....	37
Funções de Comunicação .....	39
Funções Auxiliares .....	40
Fluxo de Execução do Cliente.....	40
<b>Search with AVX2 on a Server with many clients .....</b>	<b>42</b>
Explicação do Código.....	42
Estrutura Geral do Código .....	42
Inclusões e Definições .....	43
Variáveis Globais .....	44
Funções Principais.....	44
Funções de Computação .....	46
Funções de Comunicação .....	46
Fluxo de Execução do Servidor .....	47
<b>deti_coins_client_avx20 .....</b>	<b>47</b>
Estrutura Geral do Código .....	48
Inclusões e Definições .....	48
Funções Principais.....	49
Funções de Comunicação .....	51
Funções Auxiliares .....	52
Fluxo de Execução do Cliente.....	52
<b>3.1 Search using Web Assembly .....</b>	<b>54</b>
Instalação web assembly.....	54
Alteração no código de pesquisa .....	54
<b>Search with an OpenCL implementation.....</b>	<b>55</b>
Estrutura Geral do Código .....	55
Detalhes do Código.....	55
Detalhamento das Etapas Importantes.....	58
Considerações Adicionais.....	59
Otimizações e Desafios .....	60
<b>4.1 Number of attempts in one hour on a GTX 1660 Ti grafics card</b>	<b>61</b>
<b><math>6,0 \times 10^{14}</math> .....</b>	<b>61</b>
Number of attempts in one hour on a GeForce RTX 3080 Lite Hash Rate	
grafics card $1,6 \times 10^{14}$ .....	61
<b>4.2 Computing devices, performance, number of attemps per 180s .....</b>	<b>62</b>
<b>4.3 Search DETI coins with a special form our name .....</b>	<b>68</b>
<b>4.4 Histograms about computing .....</b>	<b>69</b>

# List of Figures

Figura 1: Comparação entre AVX e AVX2 .....	6
Figura 2: Comparação entre AVX e AVX2 2 .....	7
Figura 3: deti_coins_avx_search .....	8
Figura 4: deti_coins_avx2_search .....	9
Figura 5: deti_coins_intel_cuda .....	10
Figura 6: deti_coins_intel_avx_simd .....	14
Figura 7: deti_coins_simd_openmp_search .....	18
Figura 8: deti_coins_cpu_avx2_simd .....	22
Figura 9: deti_coins_cpu_avx2_simd_openmp_search .....	26
Figura 10: deti_coins_client e deti_coins_server with avx instructions .....	29
Figura 11: deti_coins_client e deti_coins_server with avx2 instructions .....	42
Figura 12: deti_coins_search with web Assembly to 10000000000 attempts .....	54
Figura 13: deti_coins_cpu_search with web Assembly to 2 bilhões de tentativas 205.4s ...	54
Figura 14: deti_coins_opengl_search .....	55
Figura 15: GTX 1660 Ti 1 hour searching for coins .....	61
Figura 16: GeForce RTX 3080 1 hour searching for coins .....	61
Figura 17: Resultado avx PC1 .....	62
Figura 18: Resultado avx2 PC1 .....	62
Figura 19: Resultado avx openmp PC1 .....	62
Figura 20: Resultado avx2 openmp PC1 .....	62
Figura 21: Resultado CUDA PC1 .....	62
Figura 22: Resultado avx PC2 .....	62
Figura 23: Resultado avx2 PC2 .....	62
Figura 24: Resultado avx SIMD PC2 .....	62
Figura 25: Resultado avx2 SIMD PC2 .....	62
Figura 26: Resultado CUDA PC2 .....	63
Figura 27: Resultado avx openmp PC2 .....	63
Figura 28: Resultado avx2 openmp PC2 .....	63
Figura 29: Resultado Server/ 3 Client avx PC2 .....	63
Figura 30: Resultado Server/ 3 Client avx2 PC2 .....	64
Figura 31: Resultado web assembly para 4 bilhões de tentativas .....	64
Figura 32: Resultado opengl PC2 .....	65
Figura 33: Resultado avx PC3 .....	65
Figura 34: Resultado avx2 PC3 .....	65
Figura 35: Resultado avx SIMD PC3 .....	65
Figura 36: Resultado avx2 SIMD PC3 .....	65
Figura 37: Resultado avx openmp PC3 .....	65

Figura 38:Resultado Server/ 1 Client avx PC3.....	66
Figura 39:Resultado Server/ 1 Client avx2 PC3.....	66
Figura 40:Resultado opengl PC3.....	66
Figura 41:Resultado avx PC4 .....	66
Figura 42:Resultado avx2 PC4 .....	66
Figura 43:Resultado avx SIMD PC4.....	66
Figura 44:Resultado avx2 SIMD PC4.....	66
Figura 45:Resultado avx openmp PC4.....	67
Figura 46:Resultado avx2 openmp PC4.....	67
Figura 47:Resultado CUDA PC4.....	67
Figura 48:Resultado avx2 openmp PC3.....	67
Figura 49:Pesquisa especial com os nossos nomes .....	68
Figura 50:Histograma power value.....	69
Figura 51:Histograma kernel execution times .....	69

# Introdução

Neste trabalho prático 1, “DETI COINS”, fomos desafiados a explorarmos conceitos avançados de arquitetura de computadores e computação de alto desempenho. Alguns dos objetivos que nos foram propostos neste trabalho foram explorar tecnologias como AVX, NEON, CUDA, OpenMP e webAssembly, com o objetivo de acelerar o número de tentativas(attempts) e consequentemente o número de moedas mineradas.

Comparar a eficiência das diversas implementações em diferentes plataformas e diferentes dispositivos.

As “DETI Coins” são arquivos de 52 bytes cujo digest MD5 apresenta pelo menos 32 bits finais iguais a zero. A dificuldade do desafio cresce exponencialmente com o número de bits zero desejados, destacando a importância de otimizações para maximizar o número de tentativas possíveis.

Os testes realizados para cada uma das search iniciais foram realizados por:

PC1

- 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz
- GP107M [GeForce MX350]

PC2

- AMD Ryzen 7 5800X3D 8-Core Processor
- NVIDIA Corporation GA102 [GeForce RTX 3080 Lite Hash Rate]

PC3

- Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz

PC4

- AMD Ryzen 9 3900X 12-Core Processor
- NVIDIA Corporation TU116 [GeForce GTX 1660 Ti] (rev a1)

# Chapter 1

## Task 1

### 1.1 Convert the AVX code to AVX2 code

// Tomás Oliveira e Silva, October 2024	// Tomás Oliveira e Silva, October 2024
//	//
// Arquiteturas de Alto Desempenho 2024/2025	// Arquiteturas de Alto Desempenho 2024/2025
// MDS hash CPU code using AVX instructions (Intel/AMD)	→ ← // MDS hash CPU code using AVX2 instructions (Intel/AMD)
// md5_cpu_avx() ----- compute the MD5 hash of a message	→ ← // md5_cpu_avx2() ----- compute the MD5 hash of a message
// test_md5_cpu_avx() --- test the correctness of md5_cpu() and measure its execution time	→ ← // test_md5_cpu_avx2() --- test the correctness of md5_cpu() and measure its execution time
//	//
#if defined(__GNUC__) && defined(__AVX__)	→ ← #if defined(__GNUC__) && defined(__AVX2__)
#ifndef MDS_CPU_AVX	→ ← #ifndef MDS_CPU_AVX2
#define MDS_CPU_AVX	→ ← #define MDS_CPU_AVX2
//	//
// CPU-only implementation using AVX instructions (assumes a little-endian CPU)	→ ← // CPU-only implementation using AVX2 instructions (assumes a little-endian CPU)
//	//
typedef int v4si __attribute__((vector_size(16)));	→ ← typedef int v8si __attribute__((vector_size(32)));
static void md5_cpu_avx(v4si *interleaved4_data, v4si *interleaved4_hash)	→ ← static void md5_cpu_avx2(v8si *interleaved8_data, v8si *interleaved8_hash)
{ // four interleaved messages -> four interleaved MD5 hashes	{ // eight interleaved messages -> eight interleaved MD5 hashes
v4si a,b,c,d,interleaved4_state[4],interleaved4_x[16];	v8si a,b,c,d,interleaved8_state[4],interleaved8_x[16];
# define C(c) (v4si){ (int)(c),(int)(c),(int)(c),(int)(c) }	# define C(c) (v8si){ (int)(c),(int)(c),(int)(c),(int)(c),(int)(c),(int)(c),(int)(c),(int)(c) }
# define ROTATE(x,n) (__builtin_ia32_pslldi128(x,n)   __builtin_ia32_psrlldi128(x,32 - (n)))	# define ROTATE(x,n) ((v8si)_mm256_or_si256(_mm256_slli_epi32((__m256i)(x), (n)), _mm256_srli_
# define DATA(idx) interleaved4_data[idx]	# define DATA(idx) interleaved8_data[idx]
# define HASH(idx) interleaved4_hash[idx]	# define HASH(idx) interleaved8_hash[idx]
# define STATE(idx) interleaved4_state[idx]	# define STATE(idx) interleaved8_state[idx]
# define X(idx) interleaved4_x[idx]	# define X(idx) interleaved8_x[idx]
CUSTOM_MD5_CODE();	CUSTOM_MD5_CODE_V8SI();
# undef C	# undef C
# undef ROTATE	# undef ROTATE
# undef DATA	# undef DATA
# undef HASH	# undef HASH
# undef STATE	# undef STATE
# undef X	# undef X
}	}
//	//
// correctness test of md5_cpu_avx() --- test_md5_cpu() must be called first!	→ ← // correctness test of md5_cpu_avx2() --- test_md5_cpu() must be called first!
//	//
static void test_md5_cpu_avx(void)	→ ← static void test_md5_cpu_avx2(void)
{	{
# define N_TIMING_TESTS 10000000u	# define N_TIMING_TESTS 10000000u
static u32 t_interleaved_test_data[13u * 4u] __attribute__((aligned(16)));	static u32 t_interleaved_test_data[13u * 4u] __attribute__((aligned(16)));

Figura 1 Comparação entre AVX e AVX2

static u32_t interleaved_test_data[13u * 4u] __attribute__((aligned(16)));	→	←	static u32_t interleaved_test_data[13u * 8u] __attribute__((aligned(32)));
static u32_t interleaved_test_hash[ 4u * 4u] __attribute__((aligned(16)));		←	static u32_t interleaved_test_hash[ 4u * 8u] __attribute__((aligned(32)));
u32_t n, lane, idx, *htd, *hth;			u32_t n, lane, idx, *htd, *hth;
if(N_MESSAGES % 4u != 0u)	→	←	if(N_MESSAGES % 8u != 0u)
{		←	{
fprintf(stderr, "test_md5_cpu_avx: N_MESSAGES is not a multiple of 4\n");	→	←	fprintf(stderr, "test_md5_cpu_avx2: N_MESSAGES is not a multiple of 8\n");
exit(1);		←	exit(1);
hth = &host_md5_test_hash[0u];		←	hth = &host_md5_test_hash[0u];
for(n = 0u; n < N_MESSAGES; n += 4u)	→	←	for(n = 0u; n < N_MESSAGES; n += 8u)
{		←	{
// interleave data		←	// interleave data
for(lane = 0u; lane < 4u; lane++)	→	←	for(lane = 0u; lane < 8u; lane++)
for(idx = 0u; idx < 13u; idx++)	→	←	for(idx = 0u; idx < 13u; idx++)
interleaved_test_data[4u * idx + lane] = htd[13u * lane + idx];	→	←	interleaved_test_data[8u * idx + lane] = htd[13u * lane + idx];
// compute MD5 hashes		←	// compute MD5 hashes
md5_cpu_avx((v4si *)interleaved_test_data, (v4si *)interleaved_test_hash);	→	←	md5_cpu_avx2((v8si *)interleaved_test_data, (v8si *)interleaved_test_hash);
// compare with the test_md5_cpu() data		←	// compare with the test_md5_cpu() data
for(lane = 0u; lane < 4u; lane++)	→	←	for(lane = 0u; lane < 8u; lane++)
for(idx = 0u; idx < 4u; idx++)	→	←	for(idx = 0u; idx < 4u; idx++)
if(interleaved_test_hash[4u * idx + lane] != hth[4u * lane + idx])	→	←	if(interleaved_test_hash[8u * idx + lane] != hth[4u * lane + idx])
{		←	{
fprintf(stderr, "test_md5_cpu_avx: MD5 hash error for message %u\n", 4u * n + lane);	→	←	fprintf(stderr, "test_md5_cpu_avx2: MD5 hash error for message %u\n", 8u * n + lane);
exit(1);		←	exit(1);
}		←	}
// advance to the next 4 messages	→	←	// advance to the next 8 messages
hth = &hth[13u * 4u];	→	←	hth = &hth[13u * 8u];
hth = &hth[ 4u * 4u];	→	←	hth = &hth[ 4u * 8u];
}		←	}
// measure the execution time of mp5_cpu_avx()	→	←	// measure the execution time of mp5_cpu_avx2()
//		←	//
# if N_TIMING_TESTS > 0u		←	# if N_TIMING_TESTS > 0u
time_measurement();		←	time_measurement();
for(n = 0u; n < N_TIMING_TESTS; n++)		←	for(n = 0u; n < N_TIMING_TESTS; n++)
md5_cpu_avx((v4si *)interleaved_test_data, (v4si *)interleaved_test_hash);	→	←	md5_cpu_avx2((v8si *)interleaved_test_data, (v8si *)interleaved_test_hash);
time_measurement();		←	time_measurement();

Figura 2: Comparação entre AVX e AVX2 2

As diferenças entre AVX (Advanced Vector Extensions) e AVX2 (Advanced Vector Extensions 2) estão relacionadas à capacidade de processamento SIMD (Single Instruction Multiple Data) e às instruções disponíveis.

- O código AVX manipula **4 lanes** usando registros vetoriais de 128 bits para inteiros e floats. ( `_mm128i*` ).
- O código AVX2 aumenta para **8 lanes** usando registros vetoriais de 256 bits, o que requer o uso de `_mm256_*` intrinsics para processar mais mensagens em paralelo.

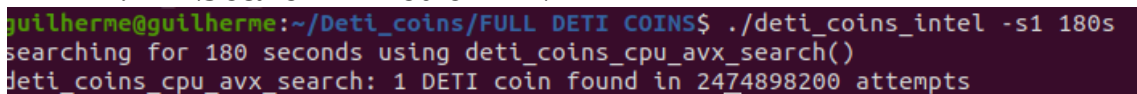
Estas melhorias tornam o AVX2 mais eficiente para tarefas que envolvem grandes quantidades de dados inteiros e cálculos simultâneos, como na procura por DETI coins.



# Chapter 2

## Task 2

### 2.1 Search function AVX



```
guilherme@guilherme:~/Deti_coins/FULL DETI COINS$ ./deti_coins_intel -s1 180s
searching for 180 seconds using deti_coins_cpu_avx_search()
deti_coins_cpu_avx_search: 1 DETI coin found in 2474898200 attempts
```

Figura 3:deti\_coins\_avx\_search

#### Variáveis Locais importantes:

- `data[13][4]`: Buffer para armazenar 13 palavras de 32 bits (52 bytes no total), processadas em 4 lanes paralelas.
- `hash[4][4]`: Buffer para armazenar os hashes MD5 calculados para 4 lanes.
- `v1, v2, v3, v4, v5, v6`: Vetores de 4 lanes para armazenar palavras aleatórias.

#### Inicialização:

- **Semente do Gerador Aleatório:** `srand(time(NULL))` garante valores pseudoaleatórios diferentes a cada execução.
- **Buffers de Dados (`data`):**
  - As primeiras palavras são configuradas como prefixos fixos: `"DETI", "coi", "n n"`.
  - Os valores seguintes (`v1, v2, ..., v6`) são inicializados com valores aleatórios.
  - A última palavra termina com o caractere `'\n'`.

#### For Principal:

- **Geração de Dados Aleatórios:**
  - `v1` e `v2` são atualizados com novos valores aleatórios para cada iteração.
  - Essas palavras atualizam os campos relevantes no buffer `data`.
- **Cálculo do MD5:**
  - A função `md5_cpu_avx` calcula os hashes MD5 para os dados em 4 lanes paralelamente.
  - A estrutura AVX permite processar até 4 mensagens simultaneamente.
- **Verificação de DETI Coins:**
  - Para cada lane, verifica-se se o hash termina em zero (`hash[3][lane] == 0`).
  - Quando uma DETI coin é encontrada, a mensagem correspondente é armazenada no buffer `coin` e guarda a moeda usando `save_deti_coin`.

## 2.2 Search function AVX2

```
gullherme@gullherme:~/Deti_coins/FULL DETI COINS$ ./deti_coins_intel -s2 180s
searching for 180 seconds using deti_coins_cpu_avx2_search()
deti_coins_cpu_avx2_search: 2 DETI coins found in 3076937360 attempts
```

Figura 4:deti\_coins\_avx2\_search

### Variáveis Locais importantes:

- `data[13][8]`: Buffer para armazenar 13 palavras de 32 bits (52 bytes no total), processadas em 8 lanes paralelas.
- `hash[4][8]`: Buffer para armazenar os hashes MD5 calculados para 8 lanes.
- `v1, v2, v3, v4, v5, v6`: Vetores de 8 lanes para armazenar palavras aleatórias.

### Inicialização:

- **Semente do Gerador Aleatório:** `srand(time(NULL))` garante valores pseudoaleatórios diferentes a cada execução.
- **Buffers de Dados (`data`):**
  - As primeiras palavras são configuradas como prefixos fixos: `"DETI", "coi", "n n"`.
  - Os valores seguintes (`v1, v2, ..., v6`) são inicializados com valores aleatórios.
  - A última palavra termina com o caractere `'\n'`.

### For Principal:

- **Geração de Dados Aleatórios:**
  - `v1` e `v2` são atualizados com novos valores aleatórios para cada iteração.
  - Essas palavras atualizam os campos relevantes no buffer `data`.
- **Cálculo do MD5:**
  - A função `md5_cpu_avx` calcula os hashes MD5 para os dados em 8 lanes paralelamente.
  - A estrutura AVX2 permite processar até 8 mensagens simultaneamente.
- **Verificação de DETI Coins:**
  - Para cada lane, verifica-se se o hash termina em zero (`hash[3][lane] == 0`).
  - Quando uma DETI coin é encontrada, a mensagem correspondente é armazenada no buffer `coin` e guarda a moeda usando `save_deti_coin`.

### Diferenças em Relação ao AVX:

- Enquanto o código AVX processa 4 lanes simultaneamente, o AVX2 processa 8 lanes, dobrando o paralelismo.
- O AVX2 permite manipular diretamente registros de 256 bits para inteiros (`v8si`), enquanto o AVX usa `v4si` com registros de 128 bits

## 2.3 Search function CUDA

```
guilherme@guilherme:~/Deti_coins/FULL DETI COINS$ ./deti_coins_intel_cuda -s4 180s
searching for 180 seconds using deti_coins_cuda_search()
initialize_cuda: CUDA code running on a NVIDIA GeForce MX350 (device 0)
deti_coins_cpu_search: 168 DETI coins found in 683436670976 attempts (expected 159.12 coins)
```

Figura 5: *deti\_coins\_intel\_cuda*

### **deti\_coins\_cuda\_kernel\_search**

#### **Parâmetros da Função:**

- **u32\_t \*deti\_coins\_storage\_area:**
  - Área de memória global na GPU para armazenar as DETI coins encontradas pelas threads.
  - As threads usam operações atômicas para garantir que as DETI coins sejam armazenadas corretamente.
- **u32\_t custom\_word\_1 e u32\_t custom\_word\_2:**
  - Palavras personalizadas que são incluídas no conteúdo da DETI coin. Estas são usadas para gerar variações específicas.

#### **Identificação da Thread**

- Cada thread CUDA possui um ID global (**n**) baseado na sua posição dentro de um bloco (**threadIdx.x**) e no índice do bloco (**blockIdx.x**).
- O ID da thread é usado para gerar valores únicos para as DETI coins.

#### **Inicialização da DETI Coin:**

- A DETI coin é representada como um array de 13 palavras de 32 bits (**coin[13]**).
- Os valores iniciais são configurados com um template fixo:  
`coin[0] = 0x49544544; // "DETI" em little-endian`  
`coin[1] = 0x696F6320; // "coi "`  
`coin[2] = 0x6E20206E; // "n n"`  
`coin[3] = custom_word_1; // Palavra personalizada`  
`coin[4] = custom_word_2; // Palavra personalizada`
- **Campos variáveis:**
  - **coin[5]:** Inclui valores derivados do ID da thread para criar variações únicas em cada thread.
  - **coin[6] a coin[11]:** Preenchidos com espaços (0x20202020) como padding.
  - **coin[12]:** Termina com um caractere de nova linha (0x0A202020).

**For Principal:**

- processa 64 iterações para cada thread;
- Em cada iteração, a palavra **coin[7]** é incrementada, criando variações nas 64 iterações.
- Cálculo do Hash MD5
- Verificação de Critérios(O ultimo valor do hash deve ser 0).

**Armazenamento da DETI Coin:**

Se uma DETI coin é encontrada, ela é armazenada na área compartilhada `deti_coins_storage_area` usando uma operação atômica:

- **atomicAdd**: Garante que múltiplas threads não sobrescrevam o mesmo local na memória.
- Apenas as DETI coins que atendem ao critério são armazenadas.

**Recursos Cuda:****Paralelismo:**

- **Threads**: Cada thread trabalha de forma independente, gerando variações de DETI coins e verificando os hashes.
- **Blocos**: Vários blocos de threads CUDA são usados, permitindo que milhares de threads sejam executadas simultaneamente.

**Controlo de lançamento:**

**`__launch_bounds__(128, 1)`**

- Define limites no número de threads por bloco e configura a eficiência do uso de recursos da GPU.

## deti\_coins\_cuda\_search

### Variáveis:

- **random\_word:**
  - Valor inicial aleatório para gerar DETI coins. Se `n_random_words` for 0, é inicializado como `0x20202020`.
- **custom\_word\_1 e custom\_word\_2:**
  - Palavras personalizadas para configurar a DETI coin. Inicialmente definidas como `0x20202020`.

### Inicialização da GPU:

- A função **initialize\_cuda** carrega o kernel CUDA `deti_coins_cuda_kernel_search` a partir de um arquivo binário (.cubin).
- Configura os recursos da GPU para usar 1024 bytes de memória global para armazenar os resultados.

### For principal:

#### 1. Inicialização dos Dados no Host:

- `host_data[0]` é configurado para armazenar o índice da próxima posição livre onde as DETI coins serão armazenadas.
- Os dados são copiados para a GPU (`CU_CALL(cuMemcpyHtoD, (device_data, (void *)host_data, 1024 * sizeof(u32_t))))`).

#### 2. Configuração dos parâmetros do Kernel:

- Um array `cu_params` é usado para passar os parâmetros para o kernel:
  - i. **device\_data:** Área de armazenamento na GPU para as DETI coins encontradas.
  - ii. **random\_word, custom\_word\_1, custom\_word\_2:** Palavras personalizadas para a configuração das DETI coins.

#### 3. Lançamento do Kernel:

`CU_CALL(cuLaunchKernel, (cu_kernel, (1u << 20)/128u, 1u, 1u, 128u, 1u, 1u, , (CUSTREAM)0, &cu_params[0], NULL));`

- **Grid dimensions:** 1 milhão de threads ((`1u << 20`)/128u blocos com 128 threads cada).
- **Block dimensions:** 128 threads por bloco.

#### 4. Cópia de Resultados para o host e processamento dos resultados:

- Após a execução do kernel, os dados da GPU são copiados de volta para a CPU.
- Verifica se existem DETI coins armazenadas em `host_data`.
- DETI coins válidas são guardadas e o contador `n_coins` é incrementado.

#### 5. Atualização de Palavras Personalizadas e Store:

- Se `custom_word_1` atingir o limite (0x7E7E7E7Eu), ele é reiniciado e `custom_word_2` é incrementado.
- As DETI coins encontradas são guardadas permanentemente.

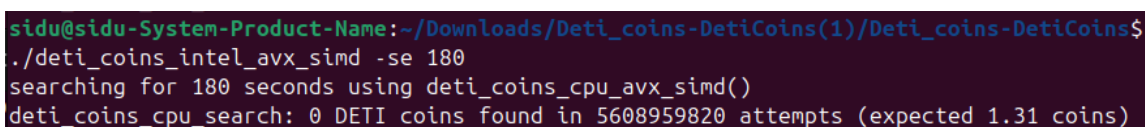
#### Recursos do CUDA Utilizados:

- **Lançamento do Kernel**
  - Configuração da grade e bloco permite que milhões de threads sejam executadas simultaneamente, com cada thread gerando e verificando DETI coins.
- **Memória Global**
  - `device_data` é usado como um buffer compartilhado entre as threads para armazenar as DETI coins encontradas.
- **Operações no Host**
  - O código usa funções como `cuMemcpyHtoD` e `cuMemcpyDtoH` para copiar dados entre o host (CPU) e o device (GPU).

## Chapter 3

### Task 3

#### Search with AVX and SIMD instructions



```
sidu@sidu-System-Product-Name:~/Downloads/Deti_coins-DetiCoins(1)/Deti_coins-DetiCoins$  
./deti_coins_intel_avx_simd -se 180  
searching for 180 seconds using deti_coins_cpu_avx_simd()  
deti_coins_cpu_search: 0 DETI coins found in 5608959820 attempts (expected 1.31 coins)
```

*Figura 6: deti\_coins\_intel\_avx\_simd*

Este código implementa uma pesquisa de "DETI coins" utilizando processamento paralelo com instruções **SIMD** (Single Instruction Multiple Data) da tecnologia **AVX** (Advanced Vector Extensions) em **CPU**. O objetivo é gerar cadeias de caracteres aleatórias, calcular os seus hashes **MD5** e verificar se cumprem um critério específico para serem consideradas uma "DETI coin".

#### Definições e Inclusões

##### Definição de Tipos Personalizados:

- **u32\_t**: Alias para **uint32\_t**, representa um inteiro sem sinal de 32 bits.
- **u64\_t**: Alias para **uint64\_t**, representa um inteiro sem sinal de 64 bits.
- **u08\_t**: Alias para **uint8\_t**, representa um inteiro sem sinal de 8 bits.

##### Variável Externa:

- **extern volatile int stop\_request**: Variável que sinaliza quando a pesquisa deve ser interrompida. Marcada como **volatile** para evitar otimizações indesejadas pelo compilador em ambientes multi-thread.

##### Constantes Definidas:

- **NUM\_LANES**: Definida como 4, representa o número de vias de processamento paralelo (lanes) nas instruções SIMD.
- **NUM\_VARIABLES**: Também definida como 4, indica o número de estados diferentes do PRNG (gerador de números pseudoaleatórios) a serem utilizados.

## Gerador de Números Aleatórios (PRNG) xorshift32 com AVX

### Estado Global do PRNG:

- **static \_\_m128i xorshift32\_state\_avx1[NUM\_VARIABLES]:** Vetor que armazena os estados iniciais do PRNG para cada variável, permitindo gerar números aleatórios independentes em paralelo.

### Inicialização do Estado do PRNG:

- A função **initialize\_xorshift32\_state\_avx1()** inicializa o estado do PRNG para cada variável com sementes diferentes. Utiliza o tempo atual (**time(NULL)**) combinado com deslocamentos baseados no índice da variável e do lane, garantindo diversidade nos números gerados.

### Implementação do xorshift32 com AVX:

- A função **xorshift32\_avx1(\_\_m128i \*state)** implementa o algoritmo **xorshift32** para gerar números pseudoaleatórios.
- Utiliza intrínsecos AVX para realizar operações lógicas (**XOR**) e deslocamentos de bits (**shift**) em paralelo sobre vetores de 128 bits, processando quatro inteiros de 32 bits simultaneamente.
- Atualiza o estado do PRNG com o novo valor gerado.

## Geração de Caracteres ASCII Imprimíveis Aleatórios

### Função **random\_printable\_u32\_avx1\_simd(u32\_t \*v, \_\_m128i \*state):**

- Gera quatro inteiros de 32 bits, onde cada byte representa um carácter ASCII imprimível.

### Processo de Geração:

1. **Geração Inicial:** Chama o PRNG para obter números aleatórios de 32 bits.
2. **Extração de Bytes:** Utiliza **\_mm\_and\_si128** para manter apenas os 8 bits menos significativos de cada inteiro, resultando em valores entre 0 e 255.
3. **Mapeamento para Caracteres Imprimíveis:**
  - a. Multiplica os valores por 95 usando **\_mm\_mullo\_epi16**, expandindo o intervalo para facilitar a distribuição uniforme.
  - b. Desloca os bits à direita em 8 posições com **\_mm\_srli\_epi16** para normalizar os valores no intervalo **[0, 94]**.
  - c. Soma **0x20** (32 em decimal) usando **\_mm\_add\_epi16** para ajustar os valores ao intervalo ASCII imprimível (32 a 126).
4. **Empacotamento:** Converte os valores de 16 bits de volta para 8 bits com **\_mm\_packus\_epi16**.
5. **Armazenamento:** Usa **\_mm\_storeu\_si128** para armazenar os resultados no array **v**.



## Função Principal de Pesquisa de "DETI coins"

### Declaração da Função:

- **static void deti\_coins\_cpu\_avx\_search\_simd(uint32\_t n\_random\_words):** Inicia a pesquisa das "DETI coins".

### Inicialização:

- **Contadores:** `u64_t n_attempts = 0, n_coins = 0;` para contar o número de tentativas e o número de moedas encontradas.
- **Arrays de Dados:**
  - **u32\_t data[13][NUM\_LANES]:** Armazena os dados a serem processados. Alinhado em 32 bytes para otimização de memória com AVX.
  - **u32\_t hash[4][NUM\_LANES]:** Armazena os hashes MD5 calculados.
  - **u32\_t v1[NUM\_LANES], v2[NUM\_LANES], v3[NUM\_LANES], v4[NUM\_LANES]:** Armazena os valores aleatórios gerados para cada lane.

### Preparação dos Dados:

- **Inicialização do PRNG:** Chama `initialize_xorshift32_state_avx1()` para configurar o estado inicial.
- **Configuração Inicial do Array data:**
  - Preenche o array com espaços (`0x20202020`, representando 4 espaços em little-endian).
  - Define valores fixos em posições específicas (e.g., "DETI", "coi ", "n n").

### Loop Principal de Busca:

- **Condição do Loop:** Continua enquanto `stop_request` for zero.
- **Processo em Cada Iteração:**
  - **Geração de Valores Aleatórios:** Chama `random_printable_u32_avx1_simd` para gerar os valores aleatórios.
  - **Atualização dos Dados:** Insere os valores gerados em posições específicas do array `data`.
  - **Cálculo do Hash MD5:** Chama `md5_cpu_avx` para calcular os hashes de forma paralela.
  - **Verificação de "DETI coin":**
    - Extrai o hash correspondente a cada `lane`.
    - Calcula o "poder" da coin com `deti_coin_power`, verificando se cumpre o critério.
    - Se o critério for atendido, salva a coin e incrementa `n_coins`.

**Finalização:**

- Após o término do loop, chama **STORE\_DETI\_COINS()** para salvar as coins encontradas.
- Exibe um resumo da busca, incluindo o número de coins encontradas e a estatística da pesquisa.

## Search with AVX and SIMD implemented in OPENMP

```
gullherme@gullherme:~/Deti_coins/FULL DETI COINS$ ./deti_coins_intel_openmp -s5 180s
searching for 180 seconds using deti_coins_cpu_avx_simd_openmp_search()
deti_coins_cpu_avx_simd_openmp_search: 2 DETI coins found in 26247257776 attempts (expected 6.11 coins)
```

Figura 7: *deti\_coins\_simd\_openmp\_search*

## Inicialização do PRNG xorshift32 com AVX e OpenMP

### Função `initialize_xorshift32_state_avx1`:

- **Propósito:** Inicializa o estado do PRNG para cada thread, garantindo que cada uma tenha uma sequência de números aleatórios diferente.
- **Parâmetros:** Recebe um ponteiro para um array `__m128i` que irá armazenar o estado inicial do PRNG para cada variável.
- **Implementação:**
  - Gera uma semente (seed) única para cada thread, combinando:
    - O tempo atual (`time(NULL)`).
    - O número da thread (`omp_get_thread_num()`).
    - O endereço da variável `state` (para variar entre threads).
    - O tempo de CPU (`clock()`).
    - O tempo de wall-clock (`omp_get_wtime()`).
  - Inicializa o estado do PRNG para cada variável com valores derivados da semente.

### Função `xorshift32_avx1`:

- **Propósito:** Implementa o algoritmo xorshift32 para gerar números pseudoaleatórios utilizando instruções AVX.
- **Parâmetros:** Recebe um ponteiro para o estado atual do PRNG.
- **Implementação:**
  - Executa operações de deslocamento e XOR em paralelo nos quatro inteiros de 32 bits contidos em `__m128i`.

## Geração de Caracteres ASCII Imprimíveis Aleatórios

### Função `random_printable_u32_avx1`:

- **Propósito:** Gera quatro inteiros de 32 bits, onde cada byte representa um carácter ASCII imprimível.
- **Parâmetros:**
  - `u32_t *v`: Ponteiro para o array onde serão armazenados os valores gerados.
  - `__m128i *state`: Estado atual do PRNG.

### Processo de Geração:

1. **Geração Inicial:** Obtém números aleatórios de 32 bits utilizando o PRNG `xorshift32_avx1`.
2. **Extração de Bytes:** Mantém apenas os 8 bits menos significativos de cada inteiro para obter valores entre 0 e 255.
3. **Mapeamento para Caracteres Imprimíveis:**
  - a. Multiplica os valores por 95 para expandir o intervalo.
  - b. Desloca os bits à direita em 8 posições para normalizar os valores no intervalo `[0, 94]`.
  - c. Adiciona 32 (`0x20`) para ajustar os valores ao intervalo ASCII imprimível (32 a 126).
4. **Empacotamento:** Converte os valores de 16 bits de volta para 8 bits.
5. **Armazenamento:** Guarda os resultados no array `v`.

## Função Principal de Pesquisa com OpenMP

### Função `deti_coins_cpu_avx_simd_openmp_search`:

- **Propósito:** Realiza a pesquisa de "DETI coins" utilizando paralelismo a nível de threads com OpenMP e paralelismo de dados com AVX.
- **Parâmetros:** `uint32_t n_random_words` (não é utilizado na função apresentada, mas pode ser relevante em versões completas).

### Implementação Geral:

- Utiliza a diretiva `#pragma omp parallel` para distribuir o trabalho entre múltiplas threads.
- Variáveis como `n_attempts` e `n_coins` são partilhadas entre threads e são protegidas com `reduction(+:n_attempts, n_coins)` para somar os valores de todas as threads no final.

### Dentro da Região Paralela:

- **Variáveis Locais por Thread:**
  - Arrays para dados, hashes e valores aleatórios.
  - Estado do PRNG específico para cada thread (`xorshift32_state_avx1`).
  - `coin[13]`: Para armazenar uma "DETI coin" encontrada.

### Inicialização dos Dados:

- Preenche o array `data` com espaços (`0x20202020`).
- Define valores fixos nas primeiras posições (e.g., "DETI", "coi ", "n n").
- Estes dados formam a estrutura base da mensagem a ser hashada.

### Loop Principal:

- Continua enquanto `stop_request` for zero.
- Em cada iteração:
  - **Geração de Valores Aleatórios:**
    - Chama `random_printable_u32_avx1` para gerar valores aleatórios para `v1`, `v2`, `v3` e `v4`.
  - **Atualização dos Dados:**
    - Insere os valores gerados nas posições `data[3]` a `data[6]`.
  - **Cálculo do Hash MD5:**
    - Utiliza a função `md5_cpu_avx` para calcular os hashes em paralelo.

- **Verificação e Armazenamento de "DETI coins":**
  - Para cada lane, verifica se o hash cumpre o critério (e.g., número de zeros iniciais).
  - Se cumprir, salva a coin utilizando uma secção crítica (`#pragma omp critical`) para evitar condições de corrida.
- **Incremento dos Contadores:**
  - `n_attempts` é incrementado pelo número de lanes.
  - `n_coins` é incrementado quando uma coin é encontrada.

#### Notas:

- **Secção Crítica (`#pragma omp critical`):**
  - Utilizada ao chamar `save_deti_coin(coin)` para garantir que apenas uma thread acesse a função de cada vez, evitando corrupção de dados.
- **Redução (`reduction(+:n_attempts, n_coins)`):**
  - Garante que os contadores `n_attempts` e `n_coins` sejam corretamente somados a partir dos valores de todas as threads no final da execução.

## Finalização e Saída

- Após a conclusão do loop (quando `stop_request` é diferente de zero), chama `STORE_DETI_COINS()` para salvar permanentemente as "DETI coins" encontradas.
- Exibe uma mensagem resumo com o número de coins encontradas, o número de tentativas realizadas e a expectativa estatística baseada no número de tentativas.

## Search with AVX2 and SIMD instructions

```
sidu@sidu-System-Product-Name:~/Downloads/Deti_coins-DetiCoins(1)/Deti_coins-DetiCoins$  
./deti_coins_intel_avx2_simd -sf 180  
searching for 180 seconds using deti_coins_cpu_avx2_simd()  
deti_coins_cpu_search: 4 DETI coins found in 10559594928 attempts (expected 2.46 coins)
```

*Figura 8: deti\_coins\_cpu\_avx2\_simd*

## Constantes Definidas

- **NUM\_LANES:** Definida como 8, representa o número de vias de processamento paralelo (lanes) nas instruções SIMD AVX2.
- **NUM\_VARIABLES:** Definida como 4, indica o número de estados diferentes do PRNG (Gerador de Números Pseudoaleatórios) a serem utilizados.

## Inicialização do PRNG xorshift32 com AVX2

### Estado Global do PRNG:

- `static __m256i xorshift32_state[NUM_VARIABLES];`: Vetor que armazena os estados iniciais do PRNG para cada variável, permitindo gerar números aleatórios independentes em paralelo.

### Função `initialize_xorshift32_state`:

- **Propósito:** Inicializa o estado do PRNG com sementes diferentes para cada variável e lane, utilizando o tempo atual como base.
- **Implementação:**
  - Para cada variável, define o estado inicial com valores baseados em `time(NULL)`, incrementados por um offset que depende de `var` e `NUM_LANES`.
  - Utiliza `_mm256_set_epi32` para criar um vetor de 256 bits com oito inteiros de 32 bits.

### Função `xorshift32_avx2`:

- **Propósito:** Implementa o algoritmo xorshift32 para gerar números pseudoaleatórios utilizando instruções AVX2.
- **Parâmetros:** Recebe um ponteiro para o estado atual do PRNG (`__m256i *state`).
- **Implementação:**
  - Executa operações de deslocamento e XOR em paralelo nos oito inteiros de 32 bits contidos em `__m256i`.
  - Atualiza o estado do PRNG com o novo valor gerado.

## Geração de Caracteres ASCII Imprimíveis Aleatórios

### Função `random_printable_u32_avx2_simd`:

- **Propósito:** Gera oito inteiros de 32 bits, onde cada byte representa um carácter ASCII imprimível.
- **Parâmetros:**
  - `u32_t *v`: Ponteiro para o array onde serão armazenados os valores gerados.
  - `__m256i *state`: Estado atual do PRNG.

### Processo de Geração:

1. **Geração Inicial:** Obtém números aleatórios de 32 bits utilizando o PRNG `xorshift32_avx2`.
2. **Divisão do Vetor de 256 bits:**
  - a. Utiliza `_mm256_extracti128_si256` para extrair as partes baixa e alta (128 bits cada) do vetor de 256 bits.
3. **Conversão para 16 bits:**
  - a. Converte os 16 valores `uint8_t` para `uint16_t` utilizando `_mm256_cvtepu8_epi16` para evitar overflow na multiplicação.
4. **Mapeamento para Caracteres Imprimíveis:**
  - a. Multiplica os valores por 95 para expandir o intervalo e facilitar a distribuição uniforme.
  - b. Desloca os bits à direita em 8 posições para normalizar os valores no intervalo `[0, 94]`.
  - c. Adiciona 32 (`0x20`) para ajustar os valores ao intervalo ASCII imprimível (32 a 126).
5. **Empacotamento:**
  - a. Converte os valores de 16 bits de volta para 8 bits com `_mm256_packus_epi16`.
6. **Armazenamento:**
  - a. Utiliza `_mm256_storeu_si256` para armazenar os resultados no array `v`.

## Função Principal de Pesquisa

### Função `deti_coins_cpu_avx2_search_simd`:

- **Propósito:** Realiza a pesquisa de "DETI coins" utilizando paralelismo de dados com instruções SIMD AVX2.
- **Parâmetros:** `uint32_t n_random_words` (não é utilizado diretamente na função apresentada, mas pode ser relevante em versões completas).



### Implementação Geral:

- Inicializa variáveis para contar o número de tentativas (`n_attempts`) e o número de coins encontradas (`n_coins`).
- Declara arrays para armazenar os dados, hashes e valores aleatórios.

### Inicialização:

- **Estado do PRNG:**
  - Chama `initialize_xorshift32_state()` para configurar o estado inicial do PRNG.
- **Array data:**
  - Preenche o array `data` com espaços (`0x20202020`), que representam quatro espaços em little-endian.
  - Define valores fixos em posições específicas para formar a estrutura base da mensagem (e.g., "DETI", "coi ", "n n").

### Loop Principal de Pesquisa:

**Condição do Loop:** Continua enquanto `stop_request` for zero.

### Processo em Cada Iteração:

1. **Geração de Valores Aleatórios:**
  - a. Gera valores aleatórios para `v1`, `v2`, `v3` e `v4` utilizando `random_printable_u32_avx2_simd` e os estados do PRNG correspondentes.
2. **Atualização dos Dados:**
  - a. Insere os valores gerados nas posições `data[3]` a `data[6]` para cada lane.
3. **Cálculo do Hash MD5:**
  - a. Chama `md5_cpu_avx2` para calcular os hashes MD5 em paralelo para todos os lanes.
4. **Verificação de "DETI coins":**
  - a. Para cada lane, verifica se o quarto elemento do hash (`hash[3][lane]`) é igual a zero, indicando que cumpre o critério para ser uma "DETI coin".
  - b. Se o critério for satisfeito, copia os dados correspondentes para o array `coin` e chama `save_deti_coin(coin)`.
  - c. Incrementa `n_coins`.
5. **Incremento do Contador de Tentativas:**
  - a. Incrementa `n_attempts` pelo número de lanes (`NUM_LANES`).

## Notas Importantes:

- **Critério de Verificação:**
  - O critério utilizado para identificar uma "DETI coin" é que `hash[3][lane] == 0`. Isto implica que a última parte do hash é zero, o que é estatisticamente raro e indica uma hash com certas propriedades desejadas.
- **Paralelismo:**
  - O código tira partido do paralelismo a nível de dados, processando oito lanes simultaneamente graças às instruções AVX2.
- **Eficiência:**
  - O alinhamento dos arrays com `__attribute__((aligned(32)))` otimiza o acesso à memória para as instruções AVX2, que operam em vetores de 256 bits.

## Finalização e Saída

- Após a conclusão do loop (quando `stop_request` é diferente de zero), chama `STORE_DETI_COINS()` para salvar permanentemente as "DETI coins" encontradas.
- Utiliza `printf` para exibir um resumo da pesquisa, incluindo:
  - Número de coins encontradas (`n_coins`).
  - Número de tentativas realizadas (`n_attempts`).
  - Expectativa estatística de encontrar coins com base no número de tentativas.

## Search with AVX2 and SIMD instructions using OPENMP

```
sidu@sidu-System-Product-Name:~/Downloads/Deti_coins-DetiCoins$ ./deti_coins_intel_openmp -s6 180s
searching for 180 seconds using deti_coins_cpu_avx2_simd_openmp_search()
deti_coins_cpu_avx2_simd_openmp_search: 39 DETI coins found in 121209091200 attempts (expected 28.22 coins)
```

Figura 9: *deti\_coins\_cpu\_avx2\_simd\_openmp\_search*

### Constantes Definidas

- **NUM\_LANES:** Definida como 8, representa o número de vias de processamento paralelo (lanes) nas instruções SIMD AVX2.
- **NUM\_VARIABLES:** Definida como 4, indica o número de estados diferentes do PRNG (Gerador de Números Pseudoaleatórios) a serem utilizados.

### Inicialização do PRNG xorshift32 com AVX2 e OpenMP

#### Função `initialize_xorshift32_state`:

- **Propósito:** Inicializa o estado do PRNG com sementes diferentes para cada thread, garantindo que cada uma tenha uma sequência de números aleatórios distinta.
- **Implementação:**
  - Gera uma semente (seed) única para cada thread, combinando:
    - O tempo atual (`time(NULL)`).
    - O número da thread (`omp_get_thread_num()`).
    - O endereço da variável `state` (para variação adicional).
    - O tempo de CPU (`clock()`).
    - O tempo de wall-clock (`omp_get_wtime()`) multiplicado por  $10^9$  para converter em nanosegundos).
  - Inicializa o estado do PRNG para cada variável utilizando `__mm256_set_epi32`, que cria um vetor de 256 bits contendo oito inteiros de 32 bits.

#### Função `xorshift32_avx2`:

- **Propósito:** Implementa o algoritmo xorshift32 para gerar números pseudoaleatórios utilizando instruções AVX2.
- **Parâmetros:** Recebe um ponteiro para o estado atual do PRNG.
- **Implementação:**
  - Realiza operações de deslocamento e XOR em paralelo nos oito inteiros de 32 bits contidos em `__m256i`.
  - Atualiza o estado do PRNG com o novo valor gerado.

## Geração de Caracteres ASCII Imprimíveis Aleatórios

Função `random_printable_u32_avx2_openmp`:

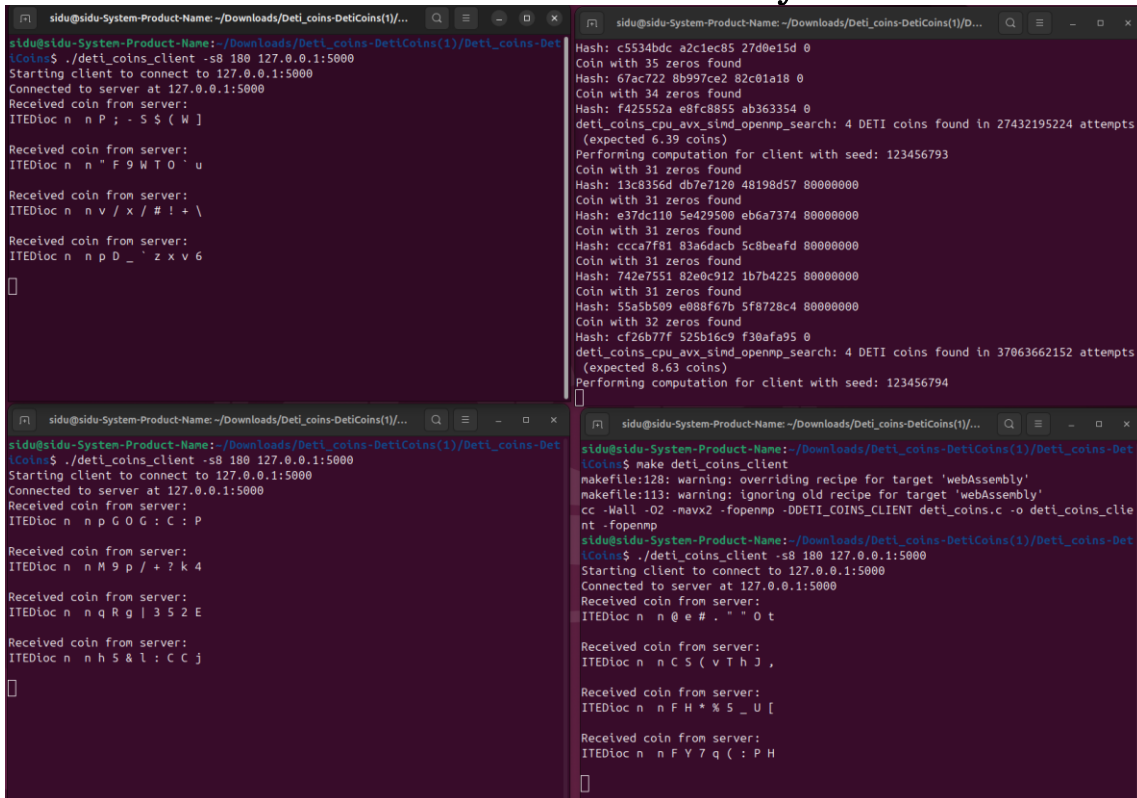
- **Propósito:** Gera oito inteiros de 32 bits, onde cada byte representa um carácter ASCII imprimível.
- **Processo de Geração:**
  - **Geração Inicial:** Obtém números aleatórios de 32 bits utilizando o PRNG `xorshift32_avx2`.
  - **Divisão do Vetor de 256 bits:** Utiliza `_mm256_extracti128_si256` para extrair as partes baixa e alta (128 bits cada) do vetor de 256 bits.
  - **Conversão para 16 bits:** Converte os bytes para `uint16_t` utilizando `_mm256_cvtepu8_epi16` para evitar overflow durante a multiplicação.
  - **Mapeamento para Caracteres Imprimíveis:**
    - Multiplica os valores por 95 para expandir o intervalo e facilitar a distribuição uniforme.
    - Desloca os bits à direita em 8 posições com `_mm256_srli_epi16` para normalizar os valores no intervalo `[0, 94]`.
    - Soma 32 (`0x20`) para ajustar os valores ao intervalo ASCII imprimível (32 a 126).
  - **Empacotamento:** Converte os valores de 16 bits de volta para 8 bits com `_mm256_packus_epi16`.
  - **Armazenamento:** Utiliza `_mm256_storeu_si256` para armazenar os resultados no array `v`.

## Função Principal de Pesquisa com OpenMP

### Função `deti_coins_cpu_avx2_simd_openmp_search`:

- **Propósito:** Realiza a pesquisa de "DETI coins" utilizando paralelismo a nível de threads com OpenMP e paralelismo de dados com instruções SIMD AVX2.
- **Implementação Geral:**
  - Utiliza `#pragma omp parallel` para criar múltiplas threads que executam em paralelo.
  - Variáveis partilhadas como `n_attempts` e `n_coins` são protegidas com `reduction(+:n_attempts, n_coins)` para acumular os valores de todas as threads.
  - **Dentro da Região Paralela:**
    - **Inicialização:**
      - Cada thread inicializa o seu próprio estado do PRNG com `initialize_xorshift32_state`.
      - Preenche o array `data` com espaços (`0x20202020`) e insere valores fixos em posições específicas para formar a estrutura base da mensagem (e.g., "DETI", "coi ", "n n").
    - **Loop Principal:**
      - Enquanto `stop_request` for zero:
        - **Geração de Valores Aleatórios:** Gera valores aleatórios para `v1`, `v2`, `v3` e `v4`.
        - **Atualização dos Dados:** Insere os valores gerados nas posições correspondentes do array `data`.
        - **Cálculo do Hash MD5:** Utiliza `md5_cpu_avx2` para calcular os hashes em paralelo.
        - **Verificação de "DETI coins":**
          - Para cada lane, verifica se o hash cumpre o critério (`hash[3][lane] == 0`).
          - Se cumprir, utiliza `#pragma omp critical` para proteger o acesso à função `save_deti_coin`, evitando condições de corrida.
          - Incrementa `n_coins`.
      - **Incremento dos Contadores:** Incrementa `n_attempts` pelo número de lanes.

## Search with AVX on a Server with many clients



```
sidu@sidu-System-Product-Name: ~/Downloads/Deti_coins-DetiCoins(1)/...
sidu@sidu-System-Product-Name: ~/Downloads/Deti_coins-DetiCoins(1)/Deti_coins-DetiCoins(1)$ ./deti_coins_client -s8 180 127.0.0.1:5000
Starting client to connect to 127.0.0.1:5000
Connected to server at 127.0.0.1:5000
Received coin from server:
ITEDioc n n P ; - S $ ( W ]

Received coin from server:
ITEDioc n n " F 9 W T O ` u

Received coin from server:
ITEDioc n n v / x / # ! + \

Received coin from server:
ITEDioc n n p 0 _ ` z x v 6

[]

sidu@sidu-System-Product-Name: ~/Downloads/Deti_coins-DetiCoins(1)/...
sidu@sidu-System-Product-Name: ~/Downloads/Deti_coins-DetiCoins(1)/Deti_coins-DetiCoins(1)$ ./deti_coins_client -s8 180 127.0.0.1:5000
Starting client to connect to 127.0.0.1:5000
Connected to server at 127.0.0.1:5000
Received coin from server:
ITEDioc n n p G O G : C : P

Received coin from server:
ITEDioc n n M 9 p / + ? k 4

Received coin from server:
ITEDioc n n q R g | 3 5 2 E

Received coin from server:
ITEDioc n n h S & l : C C j

[]

sidu@sidu-System-Product-Name: ~/Downloads/Deti_coins-DetiCoins(1)/...
Hash: c5534bdc a2c1ec85 27d0e15d 0
Coin with 35 zeros found
Hash: 67ac722 8b99/ce2 82c01a18 0
Coin with 34 zeros found
Hash: f425552a e8fc8855 ab363354 0
deti_coins_cpu_avx_slnu_openmp_search: 4 DETI coins found in 27432195224 attempts
(expected 6.39 coins)
Performing computation for client with seed: 123456793
Coin with 31 zeros found
Hash: 13c8356d db7e7120 48198d57 80000000
Coin with 31 zeros found
Hash: e37dc110 5e429500 eb6a7374 80000000
Coin with 31 zeros found
Hash: ccca7f81 83a6dacb 5c8beafd 80000000
Coin with 31 zeros found
Hash: 742e7551 82e0c912 1b7b4225 80000000
Coin with 31 zeros found
Hash: 55a5b509 e088f67b 5f8728c4 80000000
Coin with 32 zeros found
Hash: cf26b77f 525b16c9 f30afa95 0
deti_coins_cpu_avx_slnu_openmp_search: 4 DETI coins found in 37063662152 attempts
(expected 8.63 coins)
Performing computation for client with seed: 123456794

[]

sidu@sidu-System-Product-Name: ~/Downloads/Deti_coins-DetiCoins(1)/...
sidu@sidu-System-Product-Name: ~/Downloads/Deti_coins-DetiCoins(1)/Deti_coins-DetiCoins(1)$ make deti_coins_client
makefile:128: warning: overriding recipe for target 'webAssembly'
makefile:113: warning: ignoring old recipe for target 'webAssembly'
cc -Wall -O2 -mavx2 -fopenmp -DDETI_COINS_CLIENT deti_coins.c -o deti_coins_client -fopenmp
sidu@sidu-System-Product-Name: ~/Downloads/Deti_coins-DetiCoins(1)/Deti_coins-DetiCoins(1)$ ./deti_coins_client -s8 180 127.0.0.1:5000
Starting client to connect to 127.0.0.1:5000
Connected to server at 127.0.0.1:5000
Received coin from server:
ITEDioc n n 0 e # . " 0 t

Received coin from server:
ITEDioc n n C S ( v T h J ,

Received coin from server:
ITEDioc n n F H * % 5 _ U [

Received coin from server:
ITEDioc n n F Y 7 q ( : P H

[]
```

Figura 10: deti\_coins\_client e deti\_coins\_server with avx instructions

Este código implementa um servidor para a pesquisa de "DETI coins", utilizando programação paralela com OpenMP e sockets para comunicação em rede. O servidor aceita conexões de clientes, realiza computações intensivas para encontrar as "DETI coins" e retorna os resultados aos clientes. A implementação aproveita o processamento paralelo a nível de threads e dados para maximizar a eficiência computacional.

## Estrutura Geral do Código

- **Arquivo de Cabeçalho:** deti\_coins\_server.h
- **Objetivo Principal:** Implementar um servidor que aceita pedidos de trabalho de clientes, realiza computações para encontrar "DETI coins" e envia os resultados de volta aos clientes.
- **Principais Componentes:**
  - **Funções de Configuração do Servidor:** Configuram o socket do servidor e gerenciam conexões de clientes.
  - **Funções de Comunicação:** Envia e recebem mensagens entre o servidor e os clientes.
  - **Funções de Computação:** Realizam a pesquisa das "DETI coins" utilizando processamento paralelo com OpenMP e instruções SIMD.
  - **Variáveis Globais e Definições:** Gerenciam o estado do servidor e configuram parâmetros importantes.

## Inclusões e Definições

### Bibliotecas Incluídas:

1. **Bibliotecas Padrão C:**
  - a. `stdio.h`, `stdlib.h`, `string.h`, `unistd.h`, `signal.h`, `pthread.h`, `stdint.h`, `time.h`
2. **Bibliotecas para Processamento Paralelo e SIMD:**
  - a. `immintrin.h`: Intrínsecos para instruções SIMD AVX.
  - b. `omp.h`: Biblioteca OpenMP para programação paralela.
3. **Bibliotecas de Rede:**
  - a. `sys/types.h`, `sys/socket.h`, `netinet/in.h`, `arpa/inet.h`
4. **Arquivos de Cabeçalho Personalizados:**
  - a. `"md5_cpu_avx.h"`, `"cpu_utilities.h"`, `"deti_coins_vault.h"`
  - b. `"deti_coins_cpu_avx2_SIMD_OPENMP.h"`,  
`"deti_coins_cpu_avx_SIMD_OPENMP.h"`

### Definições de Constantes e Tipos:

- **Tipos Personalizados:**
  - `uint32_t`, `uint64_t`, `uint8_t` são usados para definir tamanhos específicos de inteiros.
- **Porta e Configurações de Rede:**
  - `PORT_NUMBER`: Porta em que o servidor escuta (definida como 5000).
  - `BACKLOG`: Número máximo de conexões pendentes na fila (definido como 10).
- **Parâmetros de Computação:**
  - `INITIAL_SEED`: Semente inicial para o gerador de números pseudoaleatórios (definida como 123456789).
  - `MAX_COINS_PER_REQUEST`: Número máximo de "DETI coins" a serem encontradas por pedido (definido como 4 isto porque se o cliente sair a meio parte das coins já foram guardadas, havendo menos desperdícios de coins).
- **Tipos de Mensagens:**
  - Definições de constantes para identificar diferentes tipos de mensagens trocadas entre cliente e servidor (e.g., `MSG_WORK_REQUEST`, `MSG_RESULTS`).

## Estruturas de Dados:

- **message\_t:**
  - Estrutura para representar mensagens enviadas e recebidas pelo servidor e clientes.
  - Contém o tipo da mensagem (`msg_type`), o tamanho do payload (`payload_size`) e um ponteiro para o payload (`payload`).

## Variáveis Globais

- **volatile int stop\_request:**
  - Variável utilizada para sinalizar quando o servidor deve ser interrompido.
  - Declarada como `extern` para ser acessível em outros arquivos.
- **pthread\_mutex\_t seed\_mutex:**
  - Mutex para sincronizar o acesso à variável `current_seed`.
- **uint32\_t current\_seed:**
  - Variável que mantém a semente atual para o gerador de números pseudoaleatórios.

## Funções Principais

### 1. `deti_coins_server()`

- **Descrição:**
  - Função principal que inicia o servidor.
  - Configura o socket do servidor, escuta por conexões e cria threads para lidar com clientes.
- **Implementação:**
  - Chama `setup_server(PORT_NUMBER)` para configurar o socket.
  - Entra em um loop que aceita novas conexões usando `get_connection()`.
  - Para cada nova conexão, aloca memória para o socket do cliente e cria uma nova thread `handle_client()` para lidar com a comunicação.
  - Utiliza `pthread_detach()` para permitir que os recursos da thread sejam liberados automaticamente após a conclusão.



## 2. `setup_server(int port_number)`

- **Descrição:**
  - Configura o socket do servidor para escutar em uma porta específica.
- **Implementação:**
  - Cria um socket TCP usando `socket(AF_INET, SOCK_STREAM, 0)`.
  - Configura opções de socket para reutilizar endereço e porta (`SO_REUSEADDR`).
  - Define a estrutura `sockaddr_in` com a família de endereços (`AF_INET`), endereço (`INADDR_ANY`) e porta (`htons(port_number)`).
  - Liga o socket ao endereço e porta usando `bind()`.
  - Começa a escutar por conexões com `listen()`.

## 3. `get_connection(int listen_fd, char connection_ipv4_address[32])`

- **Descrição:**
  - Aceita uma nova conexão de cliente.
- **Implementação:**
  - Chama `accept()` para aceitar uma conexão pendente.
  - Armazena o endereço IP do cliente se `connection_ipv4_address` não for NULL.

## 4. `handle_client(void *arg)`

- **Descrição:**
  - Função executada por cada thread para lidar com um cliente específico.
- **Implementação:**
  - Recebe o descritor de socket do cliente.
  - Entra em um loop para receber mensagens do cliente usando `receive_message()`.
  - Processa as mensagens com base no tipo:
    - **MSG\_WORK\_REQUEST:**
      - Gera uma nova semente usando `get_next_seed()`.
      - Chama `perform_computation()` para realizar a computação e encontrar "DETI coins".
      - Envia os resultados de volta ao cliente usando `send_message()`.
    - Outros tipos de mensagens são tratados conforme necessário.
  - Limpa os recursos alocados e fecha o socket ao terminar.

5. *perform\_computation(uint32\_t seed, uint32\_t coins\_found[][13], int \*num\_coins\_found)*

- **Descrição:**
  - Realiza a computação para encontrar "DETI coins" usando uma semente específica.
- **Implementação:**
  - Define um limite máximo de coins a serem encontradas (MAX\_COINS\_PER\_REQUEST).
  - Chama `deti_coins_cpu_avx_search_server()` para realizar a pesquisa.
  - Após a computação, o array `coins_found` e o contador `num_coins_found` são atualizados.

6. *deti\_coins\_cpu\_avx\_search\_server(...)*

- **Descrição:**
  - Realiza a pesquisa de "DETI coins" utilizando processamento paralelo com OpenMP e instruções SIMD AVX.
- **Implementação:**
  - Utiliza uma região paralela OpenMP com redução para contar o número total de tentativas (`n_attempts`).
  - Cada thread inicializa o seu próprio estado do PRNG com sementes únicas.
  - Entra em um loop que gera valores aleatórios, atualiza os dados, calcula o hash MD5 e verifica se uma "DETI coin" foi encontrada.
  - Utiliza `#pragma omp critical` para proteger o acesso à variável compartilhada `num_coins_found` e ao array `coins_found`.
  - O loop termina quando o número máximo de coins é encontrado ou se uma sinalização interna (`local_stop`) é ativada.
  - Após a conclusão, chama `STORE_DETI_COINS()` para armazenar as coins encontradas.

## Funções de Comunicação

### 1. `send_message(int socket_fd, message_t *m)`

- **Descrição:**
  - Envia uma mensagem para o cliente através do socket.
- **Implementação:**
  - Converte o tipo de mensagem e o tamanho do payload para ordem de bytes de rede (big-endian) usando `htonl()`.
  - Envia o cabeçalho contendo o tipo e o tamanho.
  - Se houver um payload, envia-o após o cabeçalho.

### 2. `receive_message(int socket_fd, message_t *m)`

- **Descrição:**
  - Recebe uma mensagem do cliente através do socket.
- **Implementação:**
  - Recebe o cabeçalho contendo o tipo de mensagem e o tamanho do payload.
  - Converte os valores para ordem de bytes do host (little-endian) usando `ntohl()`.
  - Se houver um payload, aloca memória para ele e recebe os dados.

## Sincronização e Paralelismo

- **Mutexes:**
  - `seed_mutex` é utilizado para sincronizar o acesso à variável `current_seed` na função `get_next_seed()`, garantindo que cada thread obtenha uma semente única.
- **OpenMP:**
  - Utiliza diretivas OpenMP para paralelismo a nível de threads.
  - A cláusula `reduction(+:n_attempts)` soma o número de tentativas de todas as threads.
  - A cláusula `#pragma omp critical` protege seções críticas onde múltiplas threads podem acessar e modificar recursos compartilhados.
- **PRNG Thread-Local:**
  - Cada thread inicializa o seu próprio estado do PRNG (`xorshift32_state_avx1`) com sementes únicas, garantindo independência nos números aleatórios gerados.

## Detalhes Adicionais

- **Gerador de Números Pseudoaleatórios (PRNG):**
  - O PRNG xorshift32 é utilizado para gerar números aleatórios de forma eficiente.
  - As sementes são inicializadas com uma combinação da semente fornecida, o tempo atual, o número da thread e um valor aleatório obtido com `rand_r()`.
- **Estrutura dos Dados:**
  - O array `data` armazena os dados que serão hashados. Contém partes fixas ("DETI", "coi ", "n n") e partes aleatórias.
  - O array `hash` armazena os resultados do cálculo MD5.
- **Critério para "DETI coin":**
  - Uma "DETI coin" é considerada encontrada se o valor retornado por `deti_coin_power(hash_lane)` for maior ou igual a 32.
  - Isso geralmente significa que o hash possui um certo número de zeros iniciais, indicando uma hash com propriedades específicas.

## Fluxo de Execução do Servidor

1. **Inicialização:**
  - a. O servidor é inicializado e começa a escutar na porta definida.
2. **Aceitação de Conexões:**
  - a. O servidor aceita conexões de clientes e cria uma thread para cada um.
3. **Comunicação com o Cliente:**
  - a. O cliente envia um pedido de trabalho (`MSG_WORK_REQUEST`).
  - b. O servidor gera uma semente única e realiza a computação.
4. **Computação:**
  - a. Utiliza processamento paralelo para encontrar "DETI coins".
  - b. As coins encontradas são armazenadas e enviadas de volta ao cliente.
5. **Encerramento:**
  - a. Após a conclusão ou se ocorrer algum erro, a conexão com o cliente é fechada.
  - b. O servidor continua a aceitar novas conexões até que `stop_request` seja sinalizado.

## `deti_coins_client()` ( AVX Search )

Este código implementa um cliente para o sistema de pesquisa de "DETI coins". O cliente conecta-se a um servidor, solicita trabalho (pesquisa de "DETI coins"), recebe os resultados e processa-os. O código utiliza sockets para comunicação em rede e é estruturado para permitir que o cliente funcione durante um período de tempo especificado.

## Estrutura Geral do Código

- **Arquivo de Cabeçalho:** `deti_coins_client.h`
- **Objetivo Principal:** Implementar um cliente que se conecta a um servidor, solicita trabalho para encontrar "DETI coins", recebe os resultados e os processa.
- **Principais Componentes:**
  - **Definições e Tipos:** Definição de constantes e estruturas de dados para comunicação.
  - **Funções de Comunicação:** Funções para enviar e receber mensagens através de sockets.
  - **Função Principal do Cliente:** `deti_coins_client` que controla o fluxo principal do programa.
  - **Funções Auxiliares:** Funções para lidar com os resultados recebidos e para estabelecer a conexão com o servidor.

## Inclusões e Definições

### Bibliotecas Incluídas:

1. **Bibliotecas Padrão C:**
  - a. `stdio.h`, `stdlib.h`, `string.h`, `unistd.h`, `stdint.h`, `time.h`
2. **Bibliotecas de Rede:**
  - a. `arpa/inet.h`, `sys/socket.h`, `sys/types.h`
3. **Arquivos de Cabeçalho Personalizados:**
  - a. `"cpu_utilities.h"`, `"deti_coins_vault.h"`

### Definições de Constantes e Tipos:

- **Tipos de Mensagens:**
  - `MSG_WORK_REQUEST`: Pedido de trabalho ao servidor (valor 1).
  - `MSG_RESULTS`: Mensagem contendo resultados (valor 3).
  - `MSG_NO_COINS_FOUND`: Indica que nenhuma "DETI coin" foi encontrada (valor 6).
  - `MSG_ACK`: Acknowledgment (confirmação) enviado pelo cliente (valor 5).
- **Estrutura `message_t`:**
  - Representa uma mensagem a ser enviada ou recebida.
  - Campos:
    - `uint32_t msg_type`;: Tipo da mensagem.
    - `uint32_t payload_size`;: Tamanho do payload (dados adicionais).
    - `void *payload`;: Ponteiro para os dados reais da mensagem.

## Funções Principais

*1. `deti_coins_client(char *server_address, int port_number, uint32_t seconds)`*

- **Descrição:**
  - Função principal que controla o fluxo do cliente.
  - Conecta-se ao servidor, envia pedidos de trabalho, recebe e processa os resultados.
  - Funciona durante um número especificado de segundos.
- **Parâmetros:**
  - `char *server_address`: Endereço IP do servidor.
  - `int port_number`: Número da porta do servidor.
  - `uint32_t seconds`: Duração em segundos que o cliente deve executar.
- **Implementação:**
  - Estabelece a conexão com o servidor usando `connect_to_server()`.
  - Regista o tempo de início com `time(NULL)`.
  - Entra num loop que continua enquanto o tempo especificado não for excedido.
    - Envia um pedido de trabalho (`MSG_WORK_REQUEST`) ao servidor usando `send_message()`.
    - Recebe a resposta do servidor usando `receive_message()`.
    - Processa a resposta com base no tipo de mensagem recebida:
      - **MSG\_RESULTS:**
        - Calcula o número de coins recebidas.
        - Converte as coins da ordem de bytes de rede para a do host (little-endian).
        - Chama `handle_received_coins()` para processar as coins.
        - Envia um acknowledgment (`MSG_ACK`) ao servidor.
      - **MSG\_NO\_COINS\_FOUND:**
        - Indica que o servidor não encontrou nenhuma coin nesta iteração.
      - **Outro Tipo de Mensagem:**
        - Notifica sobre o recebimento de um tipo de mensagem desconhecido.
    - Limpa a memória alocada para o payload, se aplicável.
  - Fecha a conexão com o servidor ao sair do loop.

## 2. `connect_to_server(char *ip_address, int port_number)`

- **Descrição:**
  - Estabelece uma conexão TCP com o servidor especificado.
- **Parâmetros:**
  - `char *ip_address`: Endereço IP do servidor.
  - `int port_number`: Número da porta do servidor.
- **Implementação:**
  - Cria um socket TCP usando `socket(AF_INET, SOCK_STREAM, 0)`.
  - Configura a estrutura `sockaddr_in` com a família de endereços (`AF_INET`), porta (`htons(port_number)`) e endereço IP convertido com `inet_pton()`.
  - Estabelece a conexão com `connect()`.
  - Imprime uma mensagem confirmando a conexão bem-sucedida.

## 3. `handle_received_coins(uint32_t (*coins)[13], uint32_t num_coins)`

- **Descrição:**
  - Processa as "DETI coins" recebidas do servidor.
- **Parâmetros:**
  - `uint32_t (*coins)[13]`: Ponteiro para um array de coins, cada uma composta por 13 palavras de 32 bits.
  - `uint32_t num_coins`: Número de coins recebidas.
- **Implementação:**
  - Itera sobre cada coin recebida.
    - Converte cada coin num texto legível:
      - Cada palavra de 32 bits é decomposta em 4 bytes.
      - Os bytes são colocados na ordem correta para formar a string original.
    - Imprime a coin convertida.
    - Opcionalmente, chama `save_deti_coin()` para armazenar a coin.

## Funções de Comunicação

### 1. *send\_message(int socket\_fd, message\_t \*m)*

- **Descrição:**
  - Envia uma mensagem através do socket especificado.
- **Parâmetros:**
  - `int socket_fd`: Descritor do socket através do qual enviar a mensagem.
  - `message_t *m`: Ponteiro para a mensagem a ser enviada.
- **Implementação:**
  - Prepara o cabeçalho da mensagem:
    - Converte `msg_type` e `payload_size` para ordem de bytes de rede usando `htonl()`.
  - Envia o cabeçalho usando `send()`.
  - Se existir um payload, envia-o também.

### 2. *receive\_message(int socket\_fd, message\_t \*m)*

- **Descrição:**
  - Recebe uma mensagem através do socket especificado.
- **Parâmetros:**
  - `int socket_fd`: Descritor do socket de onde receber a mensagem.
  - `message_t *m`: Ponteiro para onde armazenar a mensagem recebida.
- **Implementação:**
  - Recebe o cabeçalho da mensagem usando `recv()` com a flag `MSG_WAITALL`.
  - Converte `msg_type` e `payload_size` para ordem de bytes do host usando `ntohl()`.
  - Se existir um payload, aloca memória suficiente e recebe-o.
  - Em caso de erro, libera a memória alocada e retorna um código de erro.



## Funções Auxiliares

### 1. `close_socket(int socket_fd)`

- **Descrição:**
  - Fecha o socket especificado.
- **Parâmetros:**
  - `int socket_fd`: Descritor do socket a ser fechado.
- **Implementação:**
  - Usa a função `close()` para fechar o socket.
  - Em caso de erro, imprime uma mensagem de erro e termina o programa.

## Fluxo de Execução do Cliente

### 1. Inicialização:

- a. O cliente é iniciado com o endereço IP do servidor, o número da porta e a duração em segundos.
- b. Exemplo: `deti_coins_client("127.0.0.1", 5000, 60);` para executar durante 60 segundos.

### 2. Conexão ao Servidor:

- a. Estabelece uma conexão TCP com o servidor usando `connect_to_server()`.

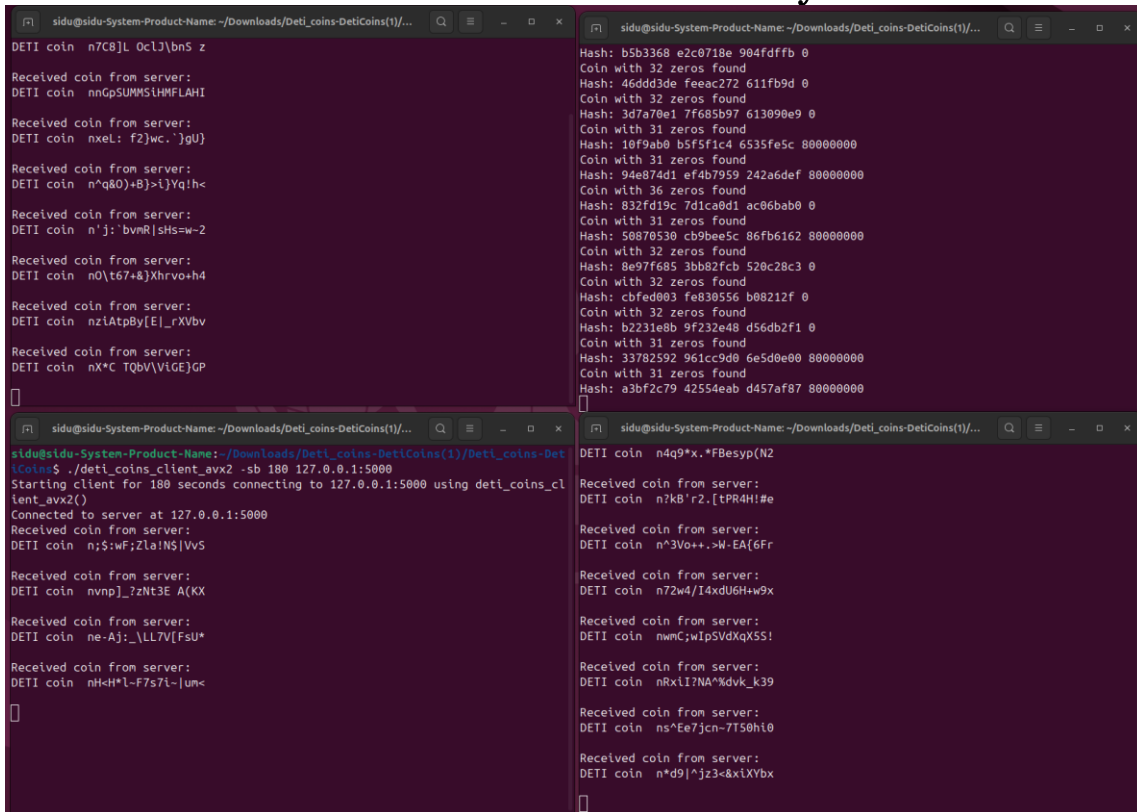
### 3. Loop Principal:

- a. Enquanto o tempo especificado não for excedido:
  - i. Envia um pedido de trabalho ao servidor (`MSG_WORK_REQUEST`).
  - ii. Recebe a resposta do servidor:
    1. Se for `MSG_RESULTS`, processa as coins recebidas com `handle_received_coins()` e envia um `acknowledgment`.
    2. Se for `MSG_NO_COINS_FOUND`, notifica que nenhuma coin foi encontrada nesta iteração.
    3. Se for um tipo de mensagem desconhecido, imprime uma mensagem de aviso.
  - iii. Limpa a memória alocada para o payload, se aplicável.

#### **4. Encerramento:**

- a. Após exceder o tempo especificado ou em caso de erro, fecha a conexão com o servidor usando `close_socket()`.

## Search with AVX2 on a Server with many clients



```

[1] sidu@sidu-System-Product-Name: ~/Downloads/DeTi_coins-DeTiCoins(1)/...
DETI coin  n7C8]L 0cl3}bn5 z
Received coin from server:
DETI coin  nnGpSUMMSLHMFLAHI
Received coin from server:
DETI coin  nxeL: f2}wc.}gUJ
Received coin from server:
DETI coin  n^q&0)+B>}>i}Yq!h<
Received coin from server:
DETI coin  n'j:'bvmR|sHs=w-2
Received coin from server:
DETI coin  n0\t67+&}Xhrvo+h4
Received coin from server:
DETI coin  nziAtpBy[E]_rXVbv
Received coin from server:
DETI coin  nX^C TQbV\ViGE]GP
[

[2] sidu@sidu-System-Product-Name: ~/Downloads/DeTi_coins-DeTiCoins(1)/...
Hash: b5b3368 e2c0718e 904fdffb 0
Coin with 32 zeros found
Hash: 46ddd3de feeac272 611fb9d 0
Coin with 32 zeros found
Hash: 3d7a70e1 7f685b97 613090e9 0
Coin with 31 zeros found
Hash: 10f9ab0 b5f5f1c4 6535fe5c 80000000
Coin with 31 zeros found
Hash: 94e874d1 ef4b7959 242a6def 80000000
Coin with 36 zeros found
Hash: 832fd19c 7d1ca0d1 ac06bab0 0
Coin with 31 zeros found
Hash: 50870530 cb9bee5c 86fb6162 80000000
Coin with 32 zeros found
Hash: 8e97f685 3bb02fcb 520c20c3 0
Coin with 32 zeros found
Hash: cbfed003 fe830556 b00212f 0
Coin with 32 zeros found
Hash: b2231e0b 9f232e48 d56db2f1 0
Coin with 31 zeros found
Hash: 33782592 961cc9d0 6e5d0e00 80000000
Coin with 31 zeros found
Hash: a3bf2c79 42554eab d457af07 80000000
[

[3] sidu@sidu-System-Product-Name: ~/Downloads/DeTi_coins-DeTiCoins(1)/DeTi_coins-DeTiCoins(1)
sidu$ ./deti_coins_client_avx2 -sb 180 127.0.0.1:5000
Starting client for 180 seconds connecting to 127.0.0.1:5000 using deti_coins_client_avx2()
Connected to server at 127.0.0.1:5000
Received coin from server:
DETI coin  n;5wF;Zla!N$|VvS
Received coin from server:
DETI coin  nvpj_?zNt3E A(KX
Received coin from server:
DETI coin  ne-Aj:_\LL7V[FsU*
Received coin from server:
DETI coin  nHkH*L-F7s7i-jum<
[

[4] sidu@sidu-System-Product-Name: ~/Downloads/DeTi_coins-DeTiCoins(1)/...
DETI coin  n4q9*x.*FBesyp(N2
Received coin from server:
DETI coin  n?kB'r2.[tPR4H!#e
Received coin from server:
DETI coin  n^3Vo+>.>N-EA{6Fr
Received coin from server:
DETI coin  n72w4/14xdU0H+w9x
Received coin from server:
DETI coin  nwmC;wlp5VdxqX5S!
Received coin from server:
DETI coin  nRxlI?NA^%dvk_k39
Received coin from server:
DETI coin  ns^Ee7jcn-7T50h10
Received coin from server:
DETI coin  n*d9|&jz3<&xiXVbx
[
```

Figura 11: deti\_coins\_client e deti\_coins\_server with avx2 instructions

## Explicação do Código

Este código implementa um servidor para a pesquisa de "DETI coins" utilizando instruções SIMD AVX2 e processamento paralelo com OpenMP. O servidor aceita conexões de clientes, realiza computações intensivas para encontrar as "DETI coins" e retorna os resultados aos clientes. A implementação foca-se em otimizar o desempenho através do uso de instruções AVX2 e programação paralela.

## Estrutura Geral do Código

- **Arquivo de Cabeçalho:** deti\_coins\_server\_avx2.h
- **Objetivo Principal:** Implementar um servidor que utiliza instruções AVX2 para encontrar "DETI coins" e comunica com clientes através de sockets TCP.
- **Principais Componentes:**
  - **Funções de Configuração do Servidor:** Configuram o socket do servidor e gerem conexões de clientes.
  - **Funções de Comunicação:** Envia e recebem mensagens entre o servidor e os clientes.
  - **Funções de Computação:** Realizam a pesquisa das "DETI coins"

- utilizando instruções AVX2 e processamento paralelo com OpenMP.
- **Variáveis Globais e Definições:** Gerem o estado do servidor e configuram parâmetros importantes.

## Inclusões e Definições

### Bibliotecas Incluídas:

- 1. Bibliotecas Padrão C:**
  - a. `stdio.h`, `stdlib.h`, `string.h`, `unistd.h`, `signal.h`, `pthread.h`, `stdint.h`, `time.h`, `errno.h`
- 2. Bibliotecas para Processamento Paralelo e SIMD:**
  - a. `immintrin.h`: Intrínsecos para instruções SIMD AVX2.
  - b. `omp.h`: Biblioteca OpenMP para programação paralela.
- 3. Bibliotecas de Rede:**
  - a. `sys/types.h`, `sys/socket.h`, `netinet/in.h`, `arpa/inet.h`
  - b. `sys/select.h`: Utilizada para operações de multiplexação de I/O (e.g., `select()`).
- 4. Arquivos de Cabeçalho Personalizados:**
  - a. `"md5_cpu_avx2.h"`, `"cpu_utilities.h"`, `"deti_coins_vault.h"`, `"deti_coins_cpu_avx2_SIMD_OPENMP.h"`

### Definições de Constantes e Tipos:

- **Parâmetros de Rede:**
  - `PORT_NUMBER`: Porta em que o servidor escuta (definida como 5000).
  - `BACKLOG`: Número máximo de conexões pendentes na fila (definido como 10).
- **Parâmetros de Computação:**
  - `INITIAL_SEED`: Semente inicial para o gerador de números pseudoaleatórios (definida como 123456789).
  - `MAX_COINS_PER_REQUEST`: Número máximo de "DETI coins" a serem encontradas por pedido (definido como 4 isto porque se o cliente sair a meio parte das coins já foram guardadas, havendo menos desperdícios de coins).
- **Tipos de Mensagens:**
  - `MSG_WORK_REQUEST`, `MSG_RESULTS`, `MSG_NO_COINS_FOUND`, `MSG_ACK`: Constantes para identificar diferentes tipos de mensagens trocadas entre cliente e servidor.

## Estruturas de Dados:

- **message\_t:**
  - Estrutura para representar mensagens enviadas e recebidas pelo servidor e clientes.
  - Campos:
    - `uint32_t msg_type`;: Tipo da mensagem.
    - `uint32_t payload_size`;: Tamanho do payload (dados adicionais).
    - `void *payload`;: Ponteiro para os dados reais da mensagem.

## Variáveis Globais

- **pthread\_mutex\_t seed\_mutex:**
  - Mutex para sincronizar o acesso à variável `current_seed`.
- **uint32\_t current\_seed:**
  - Variável que mantém a semente atual para o gerador de números pseudoaleatórios.

## Funções Principais

### 1. *deti\_coins\_server\_avx2(uint32\_t seconds)*

- **Descrição:**
  - Função principal que inicia o servidor e o mantém em execução por um número especificado de segundos.
- **Parâmetros:**
  - `uint32_t seconds`: Duração em segundos que o servidor deve executar.
- **Implementação:**
  - Configura o socket do servidor usando `setup_server(PORT_NUMBER)`.
  - Regista o tempo de início com `time(NULL)`.
  - Entra num loop que aceita novas conexões utilizando `get_connection()`, que tem em conta o tempo restante.
  - Para cada nova conexão, cria uma thread `handle_client()` para lidar com o cliente.
  - Utiliza `pthread_detach()` para permitir que os recursos da thread sejam liberados automaticamente após a conclusão.
  - O loop termina quando o tempo especificado é excedido.

## 2. *setup\_server(int port\_number)*

- **Descrição:**
  - Configura o socket do servidor para escutar em uma porta específica.
- **Implementação:**
  - Cria um socket TCP usando `socket(AF_INET, SOCK_STREAM, 0)`.
  - Configura opções de socket para reutilizar endereço e porta (`SO_REUSEADDR`).
  - Define a estrutura `sockaddr_in` com a família de endereços (`AF_INET`), endereço (`INADDR_ANY`) e porta (`htons(port_number)`).
  - Liga o socket ao endereço e porta usando `bind()`.
  - Começa a escutar por conexões com `listen()`.

## 3. *get\_connection(int listen\_fd, char connection\_ipv4\_address[32], time\_t start\_time, uint32\_t seconds)*

- **Descrição:**
  - Aceita uma nova conexão de cliente, considerando o tempo restante de execução do servidor.
- **Implementação:**
  - Utiliza `select()` para esperar por conexões com um timeout baseado no tempo restante.
  - Se o tempo limite for atingido, retorna `-1`.
  - Caso contrário, aceita a conexão com `accept()`.

## 4. *handle\_client(void \*arg)*

- **Descrição:**
  - Função executada por cada thread para lidar com um cliente específico.
- **Implementação:**
  - Recebe o descritor de socket do cliente.
  - Entra em um loop para receber mensagens do cliente usando `receive_message()`.
  - Processa as mensagens com base no tipo:
    - **MSG\_WORK\_REQUEST:**
      - Gera uma nova semente usando `get_next_seed()`.
      - Chama `perform_computation()` para realizar a computação e encontrar "DETI coins".
      - Envia os resultados de volta ao cliente usando `send_message()`.
      - Aguarda um acknowledgment (`MSG_ACK`) do cliente.
    - Outros tipos de mensagens são tratados conforme necessário.
  - Limpa os recursos alocados e fecha o socket ao terminar.

## Funções de Computação

### 1. `deti_coins_cpu_avx2_server(...)`

- **Descrição:**
  - Realiza a pesquisa de "DETI coins" utilizando instruções AVX2 e processamento paralelo com OpenMP.
- **Parâmetros:**
  - `uint32_t seed`: Semente para o gerador de números pseudoaleatórios.
  - `uint32_t coins_found[][13]`: Array para armazenar as coins encontradas.
  - `int *num_coins_found`: Ponteiro para o número de coins encontradas.
  - `int max_coins`: Número máximo de coins a serem encontradas.
- **Implementação:**
  - Utiliza uma região paralela OpenMP com redução para contar o número total de tentativas (`n_attempts`).
  - Cada thread inicializa o seu próprio estado do PRNG com sementes únicas.
  - Entra em um loop que gera valores aleatórios, atualiza os dados, calcula o hash MD5 e verifica se uma "DETI coin" foi encontrada.
  - Utiliza `#pragma omp critical` para proteger o acesso à variável compartilhada `num_coins_found` e ao array `coins_found`.
  - O loop termina quando o número máximo de coins é encontrado ou se uma sinalização interna (`local_stop`) é ativada.
  - Após a conclusão, chama `STORE_DETI_COINS()` para armazenar as coins encontradas.

### 2. `random_printable_u32_avx2_server(uint32_t *v, __m256i *state)`

- **Descrição:**
  - Gera oito inteiros de 32 bits, onde cada byte representa um carácter ASCII imprimível.
- **Implementação:**
  - Utiliza o PRNG `xorshift32_avx2_server` para gerar números aleatórios.
  - Realiza operações SIMD para mapear os números aleatórios para o intervalo de caracteres ASCII imprimíveis (32 a 126).

## Funções de Comunicação

As funções de comunicação `send_message()` e `receive_message()` são semelhantes às apresentadas anteriormente, gerenciando o envio e recebimento de mensagens através de sockets, incluindo o cabeçalho e o payload.

## Fluxo de Execução do Servidor

### 1. Inicialização:

- a. O servidor é iniciado com uma duração específica em segundos.
- b. Configura o socket e começa a escutar por conexões de clientes.

### 2. Aceitação de Conexões:

- a. Aceita conexões de clientes utilizando `select()` para respeitar o tempo restante de execução.
- b. Para cada cliente, cria uma nova thread para lidar com a comunicação.

### 3. Comunicação com o Cliente:

- a. A thread do cliente espera por mensagens do tipo `MSG_WORK_REQUEST`.
- b. Quando recebe um pedido, gera uma nova semente e realiza a computação chamando `perform_computation()`.
- c. Envia os resultados ao cliente ou uma mensagem indicando que nenhuma coin foi encontrada.

### 4. Computação:

- a. A computação é realizada utilizando instruções AVX2 e processamento paralelo com OpenMP.
- b. O objetivo é encontrar "DETI coins" que satisfaçam um determinado critério (e.g., número de zeros no hash).

### 5. Encerramento:

- a. O servidor verifica periodicamente se o tempo especificado foi excedido.
- b. Ao atingir o tempo limite, encerra novas conexões e fecha o socket de escuta.
- c. As threads em execução continuarão até completar a comunicação com os clientes.

## deti\_coins\_client\_avx2()

Este código implementa um cliente para o sistema de pesquisa de "DETI coins" utilizando instruções SIMD AVX2. O cliente conecta-se a um servidor, solicita trabalho (pesquisa de "DETI coins"), recebe os resultados e processa-os. A utilização de instruções AVX2 indica que o código está otimizado para funcionar em CPUs modernas que suportam essas extensões, permitindo operações vetorizadas e processamento paralelo de dados.



## Estrutura Geral do Código

- **Arquivo de Cabeçalho:** `deti_coins_client_avx2.h`
- **Objetivo Principal:** Implementar um cliente que se conecta a um servidor, solicita trabalho para encontrar "DETI coins" usando AVX2, recebe os resultados e os processa.
- **Principais Componentes:**
  - **Definições e Tipos:** Definição de constantes e estruturas de dados para comunicação.
  - **Funções de Comunicação:** Funções para enviar e receber mensagens através de sockets.
  - **Função Principal do Cliente:** `deti_coins_client_avx2` que controla o fluxo principal do programa.
  - **Funções Auxiliares:** Funções para lidar com os resultados recebidos e para estabelecer a conexão com o servidor.

## Inclusões e Definições

### Bibliotecas Incluídas:

1. **Bibliotecas Padrão C:**
  - a. `stdlib.h`, `string.h`, `unistd.h`, `stdint.h`, `stdio.h`
2. **Bibliotecas de Rede:**
  - a. `arpa/inet.h`, `sys/socket.h`, `sys/types.h`
3. **Arquivos de Cabeçalho Personalizados:**
  - a. `"cpu_utilities.h"`, `"deti_coins_vault.h"`

### Definições de Constantes e Tipos:

- **Tipos de Mensagens:**
  - `MSG_WORK_REQUEST`: Pedido de trabalho ao servidor (valor 1).
  - `MSG_RESULTS`: Mensagem contendo resultados (valor 3).
  - `MSG_NO_COINS_FOUND`: Indica que nenhuma "DETI coin" foi encontrada (valor 6).
  - `MSG_ACK`: Acknowledgment (confirmação) enviado pelo cliente (valor 5).
- **Estrutura `message_t`:**
  - Representa uma mensagem a ser enviada ou recebida.
  - Campos:
    - `uint32_t msg_type`;: Tipo da mensagem.
    - `uint32_t payload_size`;: Tamanho do payload (dados adicionais).
    - `void *payload`;: Ponteiro para os dados reais da mensagem.

## Funções Principais

*1. deti\_coins\_client\_avx2(char \*server\_address, int port\_number, uint32\_t seconds)*

- **Descrição:**
  - Função principal que controla o fluxo do cliente.
  - Conecta-se ao servidor, envia pedidos de trabalho, recebe e processa os resultados.
  - Funciona durante um número especificado de segundos.
- **Parâmetros:**
  - char \*server\_address: Endereço IP do servidor.
  - int port\_number: Número da porta do servidor.
  - uint32\_t seconds: Duração em segundos que o cliente deve executar.
- **Implementação:**
  - Estabelece a conexão com o servidor usando connect\_to\_server().
  - Regista o tempo de início com time(NULL).
  - Entra num loop que continua enquanto o tempo especificado não for excedido.
    - Envia um pedido de trabalho (MSG\_WORK\_REQUEST) ao servidor usando send\_message().
    - Recebe a resposta do servidor usando receive\_message().
    - Processa a resposta com base no tipo de mensagem recebida:
      - **MSG\_RESULTS:**
        - Calcula o número de coins recebidas.
        - Converte as coins da ordem de bytes de rede para a do host (little-endian).
        - Chama handle\_received\_coins() para processar as coins.
        - Envia um acknowledgment (MSG\_ACK) ao servidor.
      - **MSG\_NO\_COINS\_FOUND:**
        - Indica que o servidor não encontrou nenhuma coin nesta iteração.
      - **Outro Tipo de Mensagem:**
        - Notifica sobre o recebimento de um tipo de mensagem desconhecido.
    - Limpa a memória alocada para o payload, se aplicável.
  - Fecha a conexão com o servidor ao sair do loop.

## 2. `connect_to_server(char *ip_address, int port_number)`

- **Descrição:**
  - Estabelece uma conexão TCP com o servidor especificado.
- **Parâmetros:**
  - `char *ip_address`: Endereço IP do servidor.
  - `int port_number`: Número da porta do servidor.
- **Implementação:**
  - Cria um socket TCP usando `socket(AF_INET, SOCK_STREAM, 0)`.
  - Configura a estrutura `sockaddr_in` com a família de endereços (`AF_INET`), porta (`htons(port_number)`) e endereço IP convertido com `inet_pton()`.
  - Estabelece a conexão com `connect()`.
  - Imprime uma mensagem confirmando a conexão bem-sucedida.

## 3. `handle_received_coins(uint32_t (*coins)[13], uint32_t num_coins)`

- **Descrição:**
  - Processa as "DETI coins" recebidas do servidor.
- **Parâmetros:**
  - `uint32_t (*coins)[13]`: Ponteiro para um array de coins, cada uma composta por 13 palavras de 32 bits.
  - `uint32_t num_coins`: Número de coins recebidas.
- **Implementação:**
  - Itera sobre cada coin recebida.
    - Converte cada coin num texto legível:
      - Cada palavra de 32 bits é decomposta em 4 bytes.
      - Os bytes são colocados na ordem correta para formar a string original.
    - Imprime a coin convertida.
    - Opcionalmente, chama `save_deti_coin()` para armazenar a coin.

## Funções de Comunicação

### 1. *send\_message(int socket\_fd, message\_t \*m)*

- **Descrição:**
  - Envia uma mensagem através do socket especificado.
- **Parâmetros:**
  - `int socket_fd`: Descritor do socket através do qual enviar a mensagem.
  - `message_t *m`: Ponteiro para a mensagem a ser enviada.
- **Implementação:**
  - Prepara o cabeçalho da mensagem:
    - Converte `msg_type` e `payload_size` para ordem de bytes de rede usando `htonl()`.
  - Envia o cabeçalho usando `send()`.
  - Se existir um payload, envia-o também.

### 2. *receive\_message(int socket\_fd, message\_t \*m)*

- **Descrição:**
  - Recebe uma mensagem através do socket especificado.
- **Parâmetros:**
  - `int socket_fd`: Descritor do socket de onde receber a mensagem.
  - `message_t *m`: Ponteiro para onde armazenar a mensagem recebida.
- **Implementação:**
  - Recebe o cabeçalho da mensagem usando `recv()` com a flag `MSG_WAITALL`.
  - Converte `msg_type` e `payload_size` para ordem de bytes do host usando `ntohl()`.
  - Se existir um payload, aloca memória suficiente e recebe-o.
  - Em caso de erro, libera a memória alocada e retorna um código de erro.

## Funções Auxiliares

### 1. `close_socket(int socket_fd)`

- **Descrição:**
  - Fecha o socket especificado.
- **Parâmetros:**
  - `int socket_fd`: Descritor do socket a ser fechado.
- **Implementação:**
  - Usa a função `close()` para fechar o socket.
  - Em caso de erro, imprime uma mensagem de erro e termina o programa.

## Fluxo de Execução do Cliente

### 1. Inicialização:

- a. O cliente é iniciado com o endereço IP do servidor, o número da porta e a duração em segundos.
- b. Exemplo: `deti_coins_client_avx2("127.0.0.1", 5000, 60);` para executar durante 60 segundos.

### 2. Conexão ao Servidor:

- a. Estabelece uma conexão TCP com o servidor usando `connect_to_server()`.

### 3. Loop Principal:

- a. Enquanto o tempo especificado não for excedido:
  - i. Envia um pedido de trabalho ao servidor (`MSG_WORK_REQUEST`).
  - ii. Recebe a resposta do servidor:
    1. Se for `MSG_RESULTS`, processa as coins recebidas com `handle_received_coins()` e envia um `acknowledgment`.
    2. Se for `MSG_NO_COINS_FOUND`, notifica que nenhuma coin foi encontrada nesta iteração.
    3. Se for um tipo de mensagem desconhecido, imprime uma mensagem de aviso.
  - iii. Limpa a memória alocada para o payload, se aplicável.

#### **4. Encerramento:**

- a. Após exceder o tempo especificado ou em caso de erro, fecha a conexão com o servidor usando `close_socket()`.

### 3.1 Search using Web Assembly

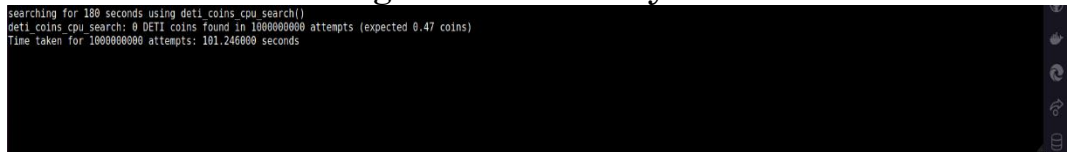


Figura 12: deti\_coins\_search with web Assembly to 100000000000 attempts

### Instalação web assembly

- Clonei o repositório do Emscripten SDK
- Instalei e Ativei o SDK
- Carreguei as Variáveis de Ambiente
- Verificação se o Emscripten foi Instalado Corretamente (“ emcc -version”)

### Alteração no código de pesquisa

- Definição de um número máximo de tentativas.
- Controle do Tempo de Execução.
- Fiz exatamente **2 bilhões de tentativas**.
- Mostrará o número de DETI coins encontradas, as tentativas realizadas e o tempo total em segundos.



Figura 13: deti\_cois\_cpu\_search with web Assembly to 2 bilhões de tentativas 205.4s

## Search with an OpenCL implementation

```
sidu@sidu-System-Product-Name:~/Downloads/Deti_coins-DetiCoins$ ./deti_coins_opencl_search -sd 180s
searching for 180 seconds using deti_coins_opencl_search()
deti_coins_opencl_search: 1557 DETI coins found in 6801349148672 attempts (expected 1583.56 coins)
```

Figura 14: *deti\_coins\_opencl\_search*

Este código implementa uma busca por "DETI coins" utilizando OpenCL para aproveitar o processamento paralelo em dispositivos como GPUs e CPUs multicore. O objetivo é acelerar o processo de mineração dessas coins, explorando a capacidade de computação massivamente paralela oferecida por dispositivos compatíveis com OpenCL.

### Estrutura Geral do Código

- **Arquivo de Cabeçalho:** `deti_coins_opencl_search.h`
- **Objetivo Principal:** Implementar uma função que realiza a busca de "DETI coins" utilizando OpenCL, permitindo que a computação seja executada em dispositivos paralelos como GPUs.
- **Principais Componentes:**
  - **Funções Auxiliares:**
    - `random_value_to_try_ascii()`: Gera um valor aleatório de 32 bits composto por caracteres ASCII imprimíveis.
    - `md5_compute()`: Função placeholder para o cálculo do hash MD5, referenciando um código personalizado (`CUSTOM_MD5_CODE()`).
  - **Função Principal:**
    - `deti_coins_opencl_search(uint32_t n_random_words)`: Implementa a lógica de busca das "DETI coins" utilizando OpenCL.

### Detalhes do Código

#### 1. Inclusões e Definições

- **Definição da Versão do OpenCL:**
  - `#define CL_TARGET_OPENCL_VERSION 200`: Define a versão alvo do



OpenCL para evitar warnings durante a compilação.

- **Inclusão de Bibliotecas:**
  - `CL/cl.h`: Biblioteca principal do OpenCL.
  - Bibliotecas padrão C: `stdio.h`, `stdlib.h`, `time.h`, `unistd.h`.
- **Variáveis Externas:**
  - `extern volatile int stop_request;`: Variável global usada para controlar a interrupção da busca, permitindo que seja sinalizada de fora da função principal.

## 2. Funções Auxiliares

### a) `random_value_to_try_ascii()`

- **Descrição:**
  - Gera um valor aleatório de 32 bits, onde cada byte representa um caractere ASCII imprimível (valores entre 0x20 e 0x7E).
- **Implementação:**
  - Utiliza um loop para gerar quatro caracteres aleatórios.
  - Cada caractere é obtido adicionando 0x20 a um valor aleatório entre 0 e 0x7F - 0x20.
  - Os caracteres são combinados em um único `uint32_t` através de deslocamentos de bits.

### b) `md5_compute()`

- **Descrição:**
  - Função placeholder para o cálculo do hash MD5.
  - Usa macros para definir constantes e operações necessárias.
  - Chama `CUSTOM_MD5_CODE()`, que presumivelmente contém a implementação específica do algoritmo MD5.

## 3. Função Principal: `deti_coins_opencl_search(uint32_t n_random_words)`

- **Propósito:**
  - Realiza a busca de "DETI coins" utilizando processamento paralelo com OpenCL.
  - Configura o ambiente OpenCL, compila o kernel, e executa o loop de

busca até que seja sinalizado para parar.

- **Variáveis Principais:**

- `uint32_t random_word`: Palavra aleatória usada no kernel.
- `uint32_t custom_word_1, custom_word_2`: Palavras customizadas que variam ao longo da busca.
- `uint64_t n_attempts, n_coins`: Contadores para o número de tentativas e coins encontradas.
- `size_t global_work_size, local_work_size`: Tamanhos de trabalho para o OpenCL, definindo o número total de work-items e o tamanho dos work-groups.

- **Inicialização do OpenCL:**

- Obtém plataformas e dispositivos disponíveis.
- Seleciona a plataforma e o dispositivo padrão.
- Cria o contexto OpenCL e a fila de comandos.
- Carrega e compila o código do kernel a partir do arquivo `deti_coins_openc1_kernel.cl`.
- Cria o kernel e o buffer de memória no dispositivo.

- **Loop de Busca:**

- Enquanto `stop_request` for zero:
  - Atualiza `random_word` com um novo valor aleatório.
  - Atualiza `custom_word_1` e `custom_word_2` utilizando `next_value_to_try()` (presumivelmente definida em `"search_utilities.h"`).
  - Define os argumentos do kernel com as palavras atualizadas e o buffer de dados.
  - Enfileira o kernel para execução no dispositivo com `clEnqueueNDRangeKernel()`.
  - Aguarda a conclusão da execução com `clFinish()`.
  - Lê os resultados de volta para o host usando `clEnqueueReadBuffer()`.
  - Processa os resultados, extraindo as coins encontradas e incrementando os contadores.
  - Reseta o índice de armazenamento no buffer do dispositivo para reutilização.

- **Finalização:**

- Chama `STORE_DETI_COINS()` para armazenar permanentemente as coins encontradas.
- Imprime um resumo das coins encontradas e das tentativas realizadas.
- Libera todos os recursos alocados no OpenCL, como buffers, kernels,

programas, filas de comandos e contexto.

## Detalhamento das Etapas Importantes

### *a) Carregamento e Compilação do Kernel OpenCL*

- **Leitura do Arquivo do Kernel:**
  - O código do kernel é lido a partir do arquivo `deti_coins_opencl_kernel.cl`.
  - O conteúdo é armazenado em `kernel_source`, que é então utilizado para criar o programa OpenCL.
- **Criação e Compilação do Programa:**
  - `clCreateProgramWithSource()` é usado para criar o programa a partir do código-fonte.
  - `clBuildProgram()` compila o programa para o dispositivo especificado.
  - Em caso de erros de compilação, o código obtém e imprime o log de compilação para facilitar a depuração.

### *b) Configuração e Execução do Kernel*

- **Definição dos Argumentos do Kernel:**
  - O kernel recebe como argumentos:
    - O buffer de dados (`device_data`).
    - As palavras `random_word`, `custom_word_1` e `custom_word_2`.
- **Execução do Kernel:**
  - `clEnqueueNDRangeKernel()` enfileira o kernel para execução.
  - `global_work_size` e `local_work_size` definem a distribuição dos work-items.
  - `clFinish()` é chamado para garantir que a execução do kernel seja concluída antes de prosseguir.

### *c) Processamento dos Resultados*

- **Leitura dos Dados do Dispositivo:**
  - Os resultados são copiados do dispositivo para o host usando

`clEnqueueReadBuffer()`.

- **Análise dos Resultados:**
  - O índice `host_data[0]` indica a próxima posição livre no buffer, ou seja, quantas coins foram encontradas.
  - O código itera sobre as posições do buffer, extraíndo cada coin encontrada.
  - Cada coin é composta por 13 palavras de 32 bits.
  - Os dados das coins são convertidos em caracteres ASCII e impressos.
  - A função `save_deti_coin()` é chamada para armazenar a coin encontrada.
- **Reset do Buffer:**
  - `host_data[0]` é resetado para 1u, indicando que o buffer está vazio e pronto para a próxima iteração.
  - O valor atualizado é copiado de volta para o dispositivo.

## Considerações Adicionais

### 1. Parâmetros de Trabalho

- **global\_work\_size:** Define o número total de work-items que serão executados. No código, está definido como `1u << 20`, o que equivale a 1.048.576 work-items.
- **local\_work\_size:** Define o tamanho dos work-groups. Está definido como 128, o que significa que cada work-group terá 128 work-items.

### 2. Controle de Parada

- **Variável stop\_request:**
  - Declarada como `extern volatile int`, permitindo que seja modificada por outros componentes do programa, sinalizando quando a busca deve parar.
  - O loop principal verifica `while (stop_request == 0)` para continuar ou interromper a execução.

### 3. Contadores e Estatísticas

- **n\_attempts:** Conta o número total de tentativas realizadas. Atualizado em cada iteração com o cálculo `n_attempts += ((uint64_t)global_work_size * 256u);`.
  - O multiplicador 256u provavelmente está relacionado ao número de

- valores testados por cada work-item.
- **n\_coins:** Conta o número total de "DETI coins" encontradas.

## Otimizações e Desafios

### *1. Paralelismo com OpenCL*

- **Aproveitamento de GPUs:**
  - O OpenCL permite que o código seja executado em GPUs, que são altamente paralelas e podem acelerar significativamente a busca.
- **Desafios:**
  - Garantir que o kernel seja otimizado para o hardware alvo.
  - Gerenciar corretamente a memória entre o host e o dispositivo para evitar gargalos.

### *2. Gerenciamento de Memória*

- **Buffers de Memória:**
  - O uso de buffers tanto no host quanto no dispositivo requer cuidado para evitar corrupção de dados.
  - É importante sincronizar as operações de leitura e escrita.

### *3. Tratamento de Erros*

- **Verificação de Erros OpenCL:**
  - Após cada chamada de função OpenCL, o código verifica se houve erros e imprime mensagens apropriadas.
  - Em caso de erro, os recursos são liberados e o programa é encerrado.

## Chapter 4

### Task 4

#### 4.1 Number of attempts in one hour on a GTX 1660 Ti graphics card $6,0 \times 10^{14}$

```
aad23@banana:~/Deti_coins$ ./deti_coins_intel_cuda -s4 3600
searching for 3600 seconds using deti_coins_cuda_search()
initialize_cuda: CUDA code running on a GeForce GTX 1660 Ti (device 0)
deti_coins_cpu_search: 13868 DETI coins found in 60018342756352 attempts (expected 13974.11 coins)
```

Figura 15: GTX 1660 Ti 1 hour searching for coins

#### Number of attempts in one hour on a GeForce RTX 3080 Lite Hash Rate graphics card $1,6 \times 10^{14}$

```
sidu@sidu-System-Product-Name:~/Downloads/Deti_coins-alpha$ ./deti_coins_intel_cuda -s4 3600s
searching for 3600 seconds using deti_coins_cuda_search()
initialize_cuda: CUDA code running on a NVIDIA GeForce RTX 3080 (device 0)
deti_coins_cpu_search: 37908 DETI coins found in 162291446185984 attempts (expected 37786.42 coins)
sidu@sidu-System-Product-Name:~/Downloads/Deti_coins-alpha$
```

Figura 16: GeForce RTX 3080 1 hour searching for coins

## 4.2 Computing devices, performance, number of attempts per 180s

Testes realizados no PC1:

```
guilherme@guilherme:~/Deti_coins/FULL DETI COINS$ ./deti_coins_intel -s1 180s
searching for 180 seconds using deti_coins_cpu_avx_search()
deti_coins_cpu_avx_search: 1 DETI coin found in 2474898200 attempts
```

Figura 17: Resultado avx PC1

```
guilherme@guilherme:~/Deti_coins/FULL DETI COINS$ ./deti_coins_intel -s2 180s
searching for 180 seconds using deti_coins_cpu_avx2_search()
deti_coins_cpu_avx2_search: 2 DETI coins found in 3076937360 attempts
```

Figura 18: Resultado avx2 PC1

```
guilherme@guilherme:~/Deti_coins/FULL DETI COINS$ ./deti_coins_intel_openmp -s5 180s
searching for 180 seconds using deti_coins_cpu_avx_simd_openmp_search()
deti_coins_cpu_avx_simd_openmp_search: 2 DETI coins found in 26247257776 attempts (expected 6.11 coins)
```

Figura 19: Resultado avx openmp PC1

```
guilherme@guilherme:~/Deti_coins/FULL DETI COINS$ ./deti_coins_intel_openmp -s5 180s
searching for 180 seconds using deti_coins_cpu_avx_simd_openmp_search()
deti_coins_cpu_avx_simd_openmp_search: 2 DETI coins found in 26247257776 attempts (expected 6.11 coins)
```

Figura 20: Resultado avx2 openmp PC1

```
guilherme@guilherme:~/Deti_coins/FULL DETI COINS$ ./deti_coins_intel_cuda -s4 180s
searching for 180 seconds using deti_coins_cuda_search()
initialize_cuda: CUDA code running on a NVIDIA GeForce MX350 (device 0)
deti_coins_cpu_search: 168 DETI coins found in 683436670976 attempts (expected 159.12 coins)
```

Figura 21: Resultado CUDA PC1

Testes realizados no PC2:

```
sidu@sidu-System-Product-Name:~/Downloads/Deti_coins-DetiCoins$ ./deti_coins_intel -s1 180s
searching for 180 seconds using deti_coins_cpu_avx_search()
deti_coins_cpu_avx_search: 0 DETI coins found in 2629787552 attempts
```

Figura 22: Resultado avx PC2

```
sidu@sidu-System-Product-Name:~/Downloads/Deti_coins-DetiCoins$ ./deti_coins_intel -s2 180s
searching for 180 seconds using deti_coins_cpu_avx2_search()
deti_coins_cpu_avx2_search: 3 DETI coins found in 3306456096 attempts
```

Figura 23: Resultado avx2 PC2

```
sidu@sidu-System-Product-Name:~/Downloads/Deti_coins-DetiCoins(1)/Deti_coins-DetiCoins$
./deti_coins_intel_avx_simd -se 180
searching for 180 seconds using deti_coins_cpu_avx_simd()
deti_coins_cpu_search: 0 DETI coins found in 5608959820 attempts (expected 1.31 coins)
```

Figura 24: Resultado avx SIMD PC2

```
sidu@sidu-System-Product-Name:~/Downloads/Deti_coins-DetiCoins(1)/Deti_coins-DetiCoins$
./deti_coins_intel_avx2_simd -sf 180
searching for 180 seconds using deti_coins_cpu_avx2_simd()
deti_coins_cpu_search: 4 DETI coins found in 10559594928 attempts (expected 2.46 coins)
```

Figura 25: Resultado avx2 SIMD PC2

```
sidu@sidu-System-Product-Name:~/Downloads/Deti_coins-DetiCoins$ ./deti_coins_intel_cuda -s4 180s
searching for 180 seconds using deti_coins_cuda_search()
initialize_cuda: CUDA code running on a NVIDIA GeForce RTX 3080 (device 0)
deti_coins_cpu_search: 1866 DETI coins found in 8317942366208 attempts (expected 1936.67 coins)
```

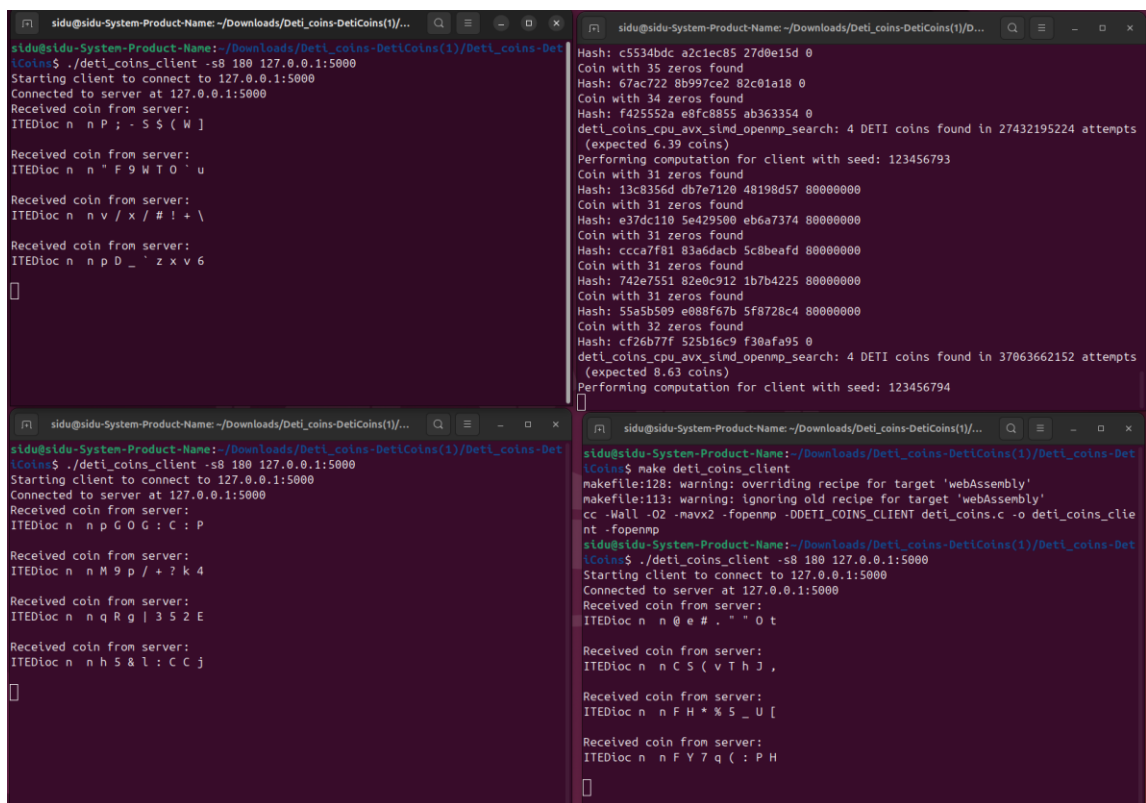
Figura 26:Resultado CUDA PC2

```
sidu@sidu-System-Product-Name:~/Downloads/Deti_coins-DetiCoins$ ./deti_coins_intel_openmp -s5 180s
searching for 180 seconds using deti_coins_cpu_avx_simd_openmp_search()
deti_coins_cpu_avx_simd_openmp_search: 8 DETI coins found in 65823150336 attempts (expected 15.33 coins)
```

Figura 27:Resultado avx openmp PC2

```
sidu@sidu-System-Product-Name:~/Downloads/Deti_coins-DetiCoins$ ./deti_coins_intel_openmp -s6 180s
searching for 180 seconds using deti_coins_cpu_avx2_simd_openmp_search()
deti_coins_cpu_avx2_simd_openmp_search: 39 DETI coins found in 121209091200 attempts (expected 28.22 coins)
```

Figura 28:Resultado avx2 openmp PC2



```
sidu@sidu-System-Product-Name:~/Downloads/Deti_coins-DetiCoins(t1)/...$ ./deti_coins_client -s8 180 127.0.0.1:5000
Starting client to connect to 127.0.0.1:5000
Connected to server at 127.0.0.1:5000
Received coin from server:
ITEDioc n n P ; - S $ ( W ]

Received coin from server:
ITEDioc n n " F 9 W T O ` u

Received coin from server:
ITEDioc n n v / x / # ! + \

Received coin from server:
ITEDioc n n p D _ ` z x v 6

[]

sidu@sidu-System-Product-Name:~/Downloads/Deti_coins-DetiCoins(t1)/...$ ./deti_coins_client -s8 180 127.0.0.1:5000
Starting client to connect to 127.0.0.1:5000
Connected to server at 127.0.0.1:5000
Received coin from server:
ITEDioc n n p G O G : C : P

Received coin from server:
ITEDioc n n M 9 p / + ? k 4

Received coin from server:
ITEDioc n n q R g | 3 5 2 E

Received coin from server:
ITEDioc n n h S & l : C C j

[]

Hash: c5534bdc a2c1ec85 27d0e15d 0
Coin with 35 zeros found
Hash: 67ac722 8b997ce2 82c01a18 0
Coin with 34 zeros found
Hash: f425552a e8fc8855 ab363354 0
deti_coins_cpu_avx_simd_openmp_search: 4 DETI coins found in 27432195224 attempts
(expected 6.39 coins)
Performing computation for client with seed: 123456793
Coin with 31 zeros found
Hash: 13c8356d db7e7120 48198d57 80000000
Coin with 31 zeros found
Hash: e37dc110 5e429500 eb6a7374 80000000
Coin with 31 zeros found
Hash: ccca7f81 83a6dacb 5c8beafd 80000000
Coin with 31 zeros found
Hash: 742e7551 82e0c912 1b7b4225 80000000
Coin with 31 zeros found
Hash: 55a5b509 e080f67b 5f8728c4 80000000
Coin with 32 zeros found
Hash: cf26b77f 525b16c9 f30afa95 0
deti_coins_cpu_avx_simd_openmp_search: 4 DETI coins found in 37063662152 attempts
(expected 8.63 coins)
Performing computation for client with seed: 123456794

[]

sidu@sidu-System-Product-Name:~/Downloads/Deti_coins-DetiCoins(t1)/...$ make deti_coins_client
makefile:128: warning: overriding recipe for target 'webAssembly'
makefile:113: warning: ignoring old recipe for target 'webAssembly'
cc -Wall -O2 -mavx2 -fopenmp -DDETI_COINS_CLIENT deti_coins.c -o deti_coins_client -fopenmp

sidu@sidu-System-Product-Name:~/Downloads/Deti_coins-DetiCoins(t1)/Deti_coins-DetiCoins$ ./deti_coins_client -s8 180 127.0.0.1:5000
Starting client to connect to 127.0.0.1:5000
Connected to server at 127.0.0.1:5000
Received coin from server:
ITEDioc n n 0 e # . " 0 t

Received coin from server:
ITEDioc n n C S ( v T h J ,

Received coin from server:
ITEDioc n n F H * % 5 _ U [

Received coin from server:
ITEDioc n n F Y 7 q ( : P H

[]
```

Figura 29:Resultado Server/ 3 Client avx PC2



```
sidu@sidu-System-Product-Name: ~/Downloads/Deti_coins-DetiCoins(1)/...
Received coin from server:
DETI coin n0\t67+8jXhrvo+h4

Received coin from server:
DETI coin nziAtpBy[E]_rXVbv

Received coin from server:
DETI coin nX*C TQbV\ViGE)GP

Received coin from server:
DETI coin n)hLLpWu=u?s8\2c

Received coin from server:
DETI coin n19Lgy#Q9^CG it//

Received coin from server:
DETI coin nM>uU_RP.ZA/\tN=$

Received coin from server:
DETI coin nZXL-R\%r|n|H[%N

Client has run for 180 seconds. Shutting down.
sidu@sidu-System-Product-Name: ~/Downloads/Deti_coins-DetiCoins(1)/Deti_coins-DetiCoins$

sidu@sidu-System-Product-Name: ~/Downloads/Deti_coins-DetiCoins(1)/...
Hash: cbfed003 fe830556 b08212f 0
Coin with 32 zeros found
Hash: b2231e8b 9f232e48 d56db2f1 0
Coin with 31 zeros found
Hash: 33782592 961cc9d0 6e5d0e00 80000000
Coin with 31 zeros found
Hash: a3bf2c79 42554eab d457af87 80000000
Coin with 32 zeros found
Hash: 4590e5b1 b1e1acd0 de7bc409 0
deti_coins_cpu_avx2_sind_openmp_search: 4 DETI coins found in 18676314504 attempts (expected 4.35 coins)
Coin with 32 zeros found
Hash: 14f96474 8b56c2a0 85880da3 0
deti_coins_cpu_avx2_sind_openmp_search: 4 DETI coins found in 16944402280 attempts (expected 3.95 coins)
Coin with 32 zeros found
Hash: 9f027e06 a254fba 65c3682f 0
Coin with 31 zeros found
Hash: f71237 8a5406b f86ec9f9 80000000
Coin with 32 zeros found
Hash: acde574a 17803380 85470aa1 0
deti_coins_cpu_avx2_sind_openmp_search: 4 DETI coins found in 28812416544 attempts (expected 6.71 coins)

sidu@sidu-System-Product-Name: ~/Downloads/Deti_coins-DetiCoins(1)/...
Received coin from server:
DETI coin nvnp]_?zNT3E ACKX

Received coin from server:
DETI coin ne-Aj+_\\LL7V[FsU*

Received coin from server:
DETI coin nH<H*L-F7s7i~|um<

Received coin from server:
DETI coin nw#PV+nYj(+7XbL0v

Received coin from server:
DETI coin nLRzKpD%nl$:veVal

Received coin from server:
DETI coin n'.S1RSU""R-\\B-Av

Received coin from server:
DETI coin n7qvgcl=il:w<9X&S

Client has run for 180 seconds. Shutting down.
sidu@sidu-System-Product-Name: ~/Downloads/Deti_coins-DetiCoins(1)/Deti_coins-DetiCoins$

sidu@sidu-System-Product-Name: ~/Downloads/Deti_coins-DetiCoins(1)/...
Received coin from server:
DETI coin nRxLI7NA^kdvk_k39

Received coin from server:
DETI coin ns^Ee7jcn~7T50hi0

Received coin from server:
DETI coin n*d9|^jz3<8xiXVbx

Received coin from server:
DETI coin n$C-jVI6ddVz>_ABQ

Received coin from server:
DETI coin nrpzB.BW#B17m,|KO

Received coin from server:
DETI coin nv5.0]-XU|LQr P\\

Received coin from server:
DETI coin nEBbC.LA/8D-#8iLj

Client has run for 180 seconds. Shutting down.
sidu@sidu-System-Product-Name: ~/Downloads/Deti_coins-DetiCoins(1)/Deti_coins-DetiCoins$
```

Figura 30:Resultado Server/ 3 Client avx2 PC2

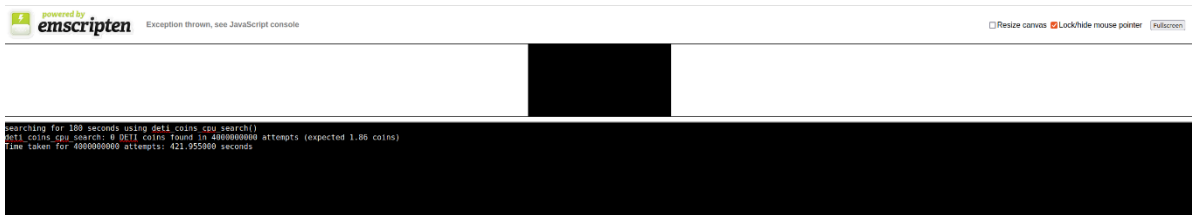


Figura 31:Resultado web assembly para 4 bilioes de tentativas

```
sidu@sidu-System-Product-Name:~/Downloads/Deti_coins-DetiCoins$ ./deti_coins_openc1_search -sd 180s
searching for 180 seconds using deti_coins_openc1_search()
deti_coins_openc1_search: 1557 DETI coins found in 6801349148672 attempts (expected 1583.56 coins)
```

*Figura 32:Resultado openc1 PC2*

Testes realizados no PC3:

```
guilherme@guilherme-VivoBook-S15-X530UF:~/UNIVERSIDADE/Deti_coins$ ./deti_coins_cpu_avx_search -s1 180
searching for 180 seconds using deti_coins_cpu_avx_search()
deti_coins_cpu_avx_search: 0 DETI coins found in 1061026688 attempts
```

*Figura 33:Resultado avx PC3*

```
guilherme@guilherme-VivoBook-S15-X530UF:~/UNIVERSIDADE/Deti_coins$ ./deti_coins_cpu_avx2_search -s2 180
searching for 180 seconds using deti_coins_cpu_avx2_search()
deti_coins_cpu_avx2_search: 0 DETI coins found in 1198714008 attempts
```

*Figura 34:Resultado avx2 PC3*

```
guilherme@guilherme-VivoBook-S15-X530UF:~/UNIVERSIDADE/Deti_coins$ ./deti_coins_cpu_avx_simd_search -se 180
searching for 180 seconds using deti_coins_cpu_avx_simd()
deti_coins_cpu_search: 1 DETI coin found in 4678461872 attempts (expected 1.09 coins)
```

*Figura 35:Resultado avx SIMD PC3*

```
guilherme@guilherme-VivoBook-S15-X530UF:~/UNIVERSIDADE/Deti_coins$ ./deti_coins_cpu_avx2_simd_search -sf 180
searching for 180 seconds using deti_coins_cpu_avx2_simd()
deti_coins_cpu_search: 4 DETI coins found in 8703234160 attempts (expected 2.03 coins)
```

*Figura 36:Resultado avx2 SIMD PC3*

```
guilherme@guilherme-VivoBook-S15-X530UF:~/UNIVERSIDADE/Deti_coins$ ./deti_coins_cpu_avx_simd_openmp_search -s5 180
searching for 180 seconds using deti_coins_cpu_avx_simd_openmp_search()
deti_coins_cpu_avx_simd_openmp_search: 5 DETI coins found in 29316969536 attempts (expected 6.83 coins)
```

*Figura 37:Resultado avx openmp PC3*

```

Coin with 39 zeros found
Hash: aa10f4ae 6baa9ebf 70edbe80 0
Coin with 31 zeros found
Hash: b76d1432 7b287631 e245ab7 80000000
Coin with 31 zeros found
Hash: d809f3b1 b7407aac 9e3972c0 80000000
Coin with 32 zeros found
Hash: f706444c 1170541d 190c8907 0
Coin with 31 zeros found
Hash: 7517ce47 4afale4a 2c58f251 80000000
Coin with 31 zeros found
Hash: 7647bd08 a3138aff b71a9e0d 80000000
Coin with 32 zeros found
Hash: 3ef87ed8 679c099d d7fb479 0
Coin with 31 zeros found
Hash: a2c2e455 6a3639a1 50226fc2 80000000
Coin with 31 zeros found
Hash: 8bb6dd1f afa86f9c c2cc8775 80000000
Coin with 33 zeros found
Hash: 59fa8df f9eb0540 466c14a6 0
deti_coins_cpu_avx_simd_openmp_search: 4 DETI coins found in 36219322392 attempts
s (expected 8.43 coins)
c

connect to server
# seconds is the amount of time spe
nt in the search
# n_random_words is the number of 4
-byte words to use
guilherme@guilherme-VivoBook-S15-X530UF:~/UNIVERSIDADE/Deti_coins$ ./deti_coins_
client -sb 180 127.0.0.1:5000
Starting client for 180 seconds connecting to 127.0.0.1:5000 using deti_coins_cl
ient()
Connected to server at 127.0.0.1:5000
Received coin from server:
ITEDioc n n 5 < t l s ; a 2

Received coin from server:
ITEDioc n n + o x Z % / , |

Received coin from server:
ITEDioc n n 4 4 s - T + ~ i

Received coin from server:
ITEDioc n n G j ; z B ( ? F

Client has run for 180 seconds. Shutting down.

```

Figura 38:Resultado Server/1 Client avx PC3

```

Coin with 33 zeros found
Hash: ad8a03cc 238c08c1 2ed04bde 0
Coin with 31 zeros found
Hash: e7525cb4 7f800eeb 7ed5ac89 80000000
Coin with 31 zeros found
Hash: 12a25c41 434133ae 46f515c2 80000000
Coin with 34 zeros found
Hash: 4e8ea4d0 b9991115 ba1341fc 0
Coin with 33 zeros found
Hash: 87c22a6c 7bf39248 a3ace97a 0
deti_coins_cpu_avx2_simd_openmp_search: 4 DETI coins found in 14110505344 attemp
ts (expected 3.29 coins)
Performing computation for client with seed: 123456790
Coin with 31 zeros found
Hash: 7183c410 5eedc5d0 fbeb0234 80000000
Coin with 32 zeros found
Hash: 56f997ed 7a1e6ef6 e544c047 0
Coin with 31 zeros found
Hash: d2c205cf e115928f 7ced90b8 80000000
Coin with 32 zeros found
Hash: 29b2d08d d55eafad 70fac767 0

guilherme@guilherme-VivoBook-S15-X530UF:~/UNIVERSIDADE/Deti_coins$ ./deti_coins_
client_avx2 -sb 127.0.0.1:5000 180
main: Bad number of seconds --- format [Nd][Nh][Nm][N[s]], where each N is a num
ber and [] means whats inside it is optional
guilherme@guilherme-VivoBook-S15-X530UF:~/UNIVERSIDADE/Deti_coins$ ./deti_coins_
client_avx2 -sb 180 127.0.0.1:5000
Starting client for 180 seconds connecting to 127.0.0.1:5000 using deti_coins_cl
ient_avx2()
Connected to server at 127.0.0.1:5000
Received coin from server:
DETI coin n+FA dhU"o(25z/oR

Received coin from server:
DETI coin ncBqL+/Zc'Ui-:le

Received coin from server:
DETI coin nfl- sH4$]Y%KNCNO

Received coin from server:
DETI coin nN1ED0./)zlc-(9rN

```

Figura 39:Resultado Server/1 Client avx2 PC3

```

Coin found: DETI coin n`>11F(      =s+!_8-!G! G!

deti_coins_openc1_search: 59 DETI coins found in 214748364800 attempts (expected
50.00 coins)

```

Figura 40:Resultado openc1 PC3

Testes realizados no PC4:

```

aad23@banana:~/Deti_coins$ ./deti_coins_intel -s1 180
searching for 180 seconds using deti_coins_cpu_avx_search()
deti_coins_cpu_avx_search: 0 DETI coins found in 2167937724 attempts

```

Figura 41:Resultado avx PC4

```

aad23@banana:~/Deti_coins$ ./deti_coins_intel -s2 180
searching for 180 seconds using deti_coins_cpu_avx2_search()
deti_coins_cpu_avx2_search: 3 DETI coins found in 2658944520 attempts

```

Figura 42:Resultado avx2 PC4

```

aad23@banana:~/Deti_coins$ ./deti_coins_intel_avx_simd -se 180
searching for 180 seconds using deti_coins_cpu_avx_simd()
deti_coins_cpu_search: 0 DETI coins found in 5342383152 attempts (expected 1.24
coins)

```

Figura 43:Resultado avx SIMD PC4

```

aad23@banana:~/Deti_coins$ ./deti_coins_intel_avx2_simd -sf 180
searching for 180 seconds using deti_coins_cpu_avx2_simd()
deti_coins_cpu_search: 1 DETI coin found in 9980703288 attempts (expected 2.32 c
oins)

```

Figura 44:Resultado avx2 SIMD PC4

```
aad23@banana:~/Deti_coins$ ./deti_coins_intel_openmp -s5 180
searching for 180 seconds using deti_coins_cpu_avx_simd_openmp_search()
deti_coins_cpu_avx_simd_openmp_search: 5 DETI coins found in 98575252836 attempts (expected 22.95 coins)
```

*Figura 45:Resultado avx openmp PC4*

```
aad23@banana:~/Deti_coins$ ./deti_coins_intel_openmp -s6 180
searching for 180 seconds using deti_coins_cpu_avx2_simd_openmp_search()
deti_coins_cpu_avx2_simd_openmp_search: 52 DETI coins found in 177829903264 attempts (expected 41.40 coins)
```

*Figura 46:Resultado avx2 openmp PC4*

```
aad23@banana:~/DetiCoins$ ./deti_coins_intel_cuda -s4 180
searching for 180 seconds using deti_coins_cuda_search()
initialize_cuda: CUDA code running on a GeForce GTX 1660 Ti (device 0)
deti_coins_cpu_search: 712 DETI coins found in 3068351479808 attempts (expected 714.41 coins)
```

*Figura 47:Resultado CUDA PC4*

```
guilherme@guilherme-VivoBook-S15-X530UF:~/UNIVERSIDADE/Deti_coins$ ./deti_coins_intel_openmp -s6 180
searching for 180 seconds using deti_coins_cpu_avx2_simd_openmp_search()
deti_coins_cpu_avx2_simd_openmp_search: 17 DETI coins found in 50420458552 attempts (expected 11.74 coins)
```

*Figura 48:Resultado avx2 openmp PC3*

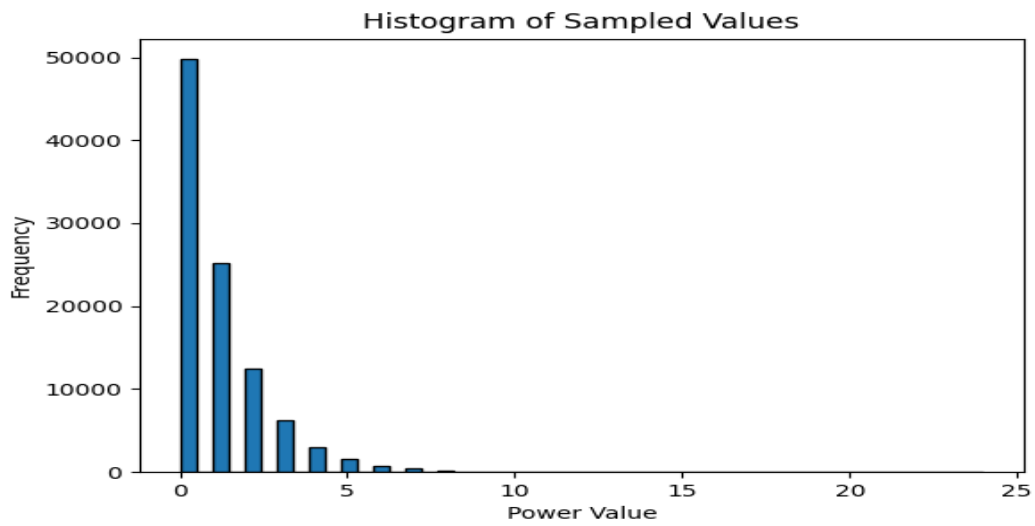
### 4.3 Search DETI coins with a special form our name

```
V00:DETI coin njc(u.F+B8v0c`3)%guilherme_duarte😄
V00:DETI coin n8)v]G@\\!FmB;G!#4guilherme_duarte😄
V01:DETI coin nGyyg`J^vEE)M[u8>guilherme_duarte😄
V02:DETI coin n."?RerGBH*HL}i=[guilherme_duarte😄
V01:DETI coin nNxIR%j"sbq4p)T'%guilherme_duarte😄
V09:DETI coin nVtb7[#z0/#eV(u}nguilhermeChapelo😄
V00:DETI coin nJ,ddC>?"?k`L8UZ(guilhermeChapelo😄
V02:DETI coin nzFCTGWl RAs;+/?guilhermeChapelo😄
V00:DETI coin nXe<"g#~4of66<QUdguilhermeChapelo😄
V01:DETI coin n,*Pv`*HXRjCsgl"guilhermeChapelo😄
V01:DETI coin nY6Q;}/(H)i':Nm0CguilhermeChapelo😄
```

*Figura 49: Pesquisa especial com os nossos nomes*

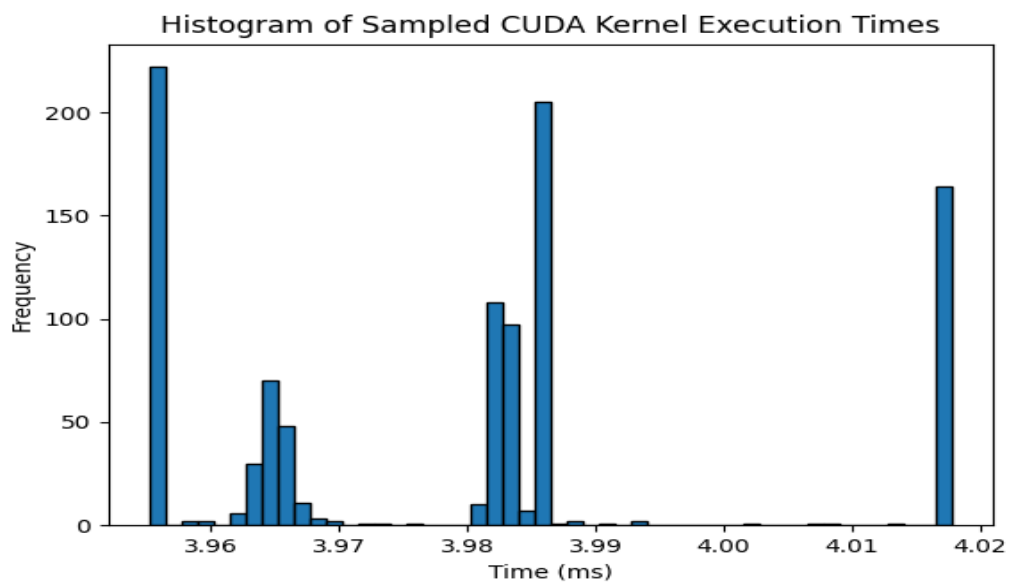
## 4.4 Histograms about computing

*The values returned by `deti_coin_power()` during a AVX2 search*



*Figura 50:Histograma power value*

*The wall time of each call to the CUDA search kernel*



*Figura 51:Histograma kernel execution times*