Uso de Computação Paralela para Acelerar a Cripto-compressão de Dados

UFSC - Departamento de Informática e Estatística
Leandro Perin de Oliveira
2018

Introdução

- Comprimir dados é importante para diversos serviços
  - YouTube
  - Instagram
  - Facebook
- Tráfego de dados comprimidos é mais rápido
- Menor necessidade de espaço de armazenamento

Introdução

- Cifrar dados possui grande importância
- Tráfego de dados cifrados é mais seguro e confiável

O Problema

- Aeroportos do Reino Unido estão enviando fotografias 3D dos passageiros
- Aproximadamente 4TB de dados gerados diariamente
- Dados enviados para a polícia do destino e para a Polícia Metropolitana de Londres
- Problemas de espaço? Problemas de segurança?

Fundamentação Teórica

- OpenMP
- MPI
- OpenCL

Fundamentação Teórica

- OpenMP Schedulers
  - Static
    - Divide igualmente
  - Dynamic
    - Cada thread recebe um chunk quando fica ociosa
  - Guided
    - Redução exponencial do tamanho dos chunks

Fundamentação Teórica

- DCT - Transformada Discreta do Cosseno
  - Processo sem perdas e reversível;
  - Tem o objetivo de identificar redundâncias na imagem (Pixels muito semelhantes);
  - Agrupa a maior parte da informação no canto superior esquerdo da imagem, facilitando a otimização;
  - Usada por outros formatos de compressão, como o JPEG;

O ALgoritmo GMPR - Criptografia

- Geradas 3 chaves na etapa de cifragem
- Chaves são armazenadas no cabeçalho dos arquivos
- Cabeçalho cifrado com o algoritmo AES

O Algoritmo GMPR - Compressão de Textos

1. Obtém uma lista dos caracteres sem repetir nenhum;
2. K[0] ← rand() % 1001
3. K[1] ← (rand() % 11) * 2 * maxvalue
4. K[2] ← K[1] * maxvalue * (rand() % 11)
5. nCoded[i] ← K[0] ∗ nData[i] + K[1] ∗ nData[i + 1] + K[2] ∗ nData[i + 2]
6. Lista contendo a lista de caracteres, as 3 chaves e o vetor nCoded são convertidos para string e salvos em disco;

O Algoritmo GMPR - Descompressão de Textos

1. Texto é convertido para vetor e são extraídas a lista de caracteres, as chaves e a lista nCoded;

$$
\begin{array}{l}
ultimaPos \leftarrow data.tamanho() \\
\textbf{for } r \textbf{ from } 0 \textbf{ to } ultimaPos \textbf{ do} \\
\quad \textbf{for } i \textbf{ from } 0 \textbf{ to } nL \textbf{ do} \\
\quad\quad \textbf{for } j \textbf{ from } 0 \textbf{ to } nL \textbf{ do} \\
\quad\quad\quad \textbf{for } k \textbf{ from } 0 \textbf{ to } nL \textbf{ do} \\
\quad\quad\quad\quad \textbf{if } data[r] == K[0] * nLimited[i] + K[1] * nLimited[j] + K[2] * nLimited[k] \textbf{ then} \\
\quad\quad\quad\quad\quad nDecoded[3 * r + 0] \leftarrow nLimited[i] \\
\quad\quad\quad\quad\quad nDecoded[3 * r + 1] \leftarrow nLimited[j] \\
\quad\quad\quad\quad\quad nDecoded[3 * r + 2] \leftarrow nLimited[k]
\end{array}
$$

O ALgoritmo GMPR - Compressão de Imagens

1. Imagem é dividida em blocos de tamanho n;
2. É aplicada a DCT em cada bloco;
3. Vetor minimizado é produzido eliminando todos os zeros, que geralmente são muitos;
4. Intervalos onde os zeros estavam são salvos em um vetor;
5. K[0] ← 1
6. K[1] ← 2 + max_element(nonZeroData)
7. K[2] ← max_element(nonZeroData) * (1 + K[1])
8. vcoded[i] ← K[0] ∗ nonZeroData[i] + K[1] ∗ nonZeroData[i + 1] + K[2] ∗ nonZeroData[i + 2]

O Algoritmo GMPR - Descompressão de Imagens

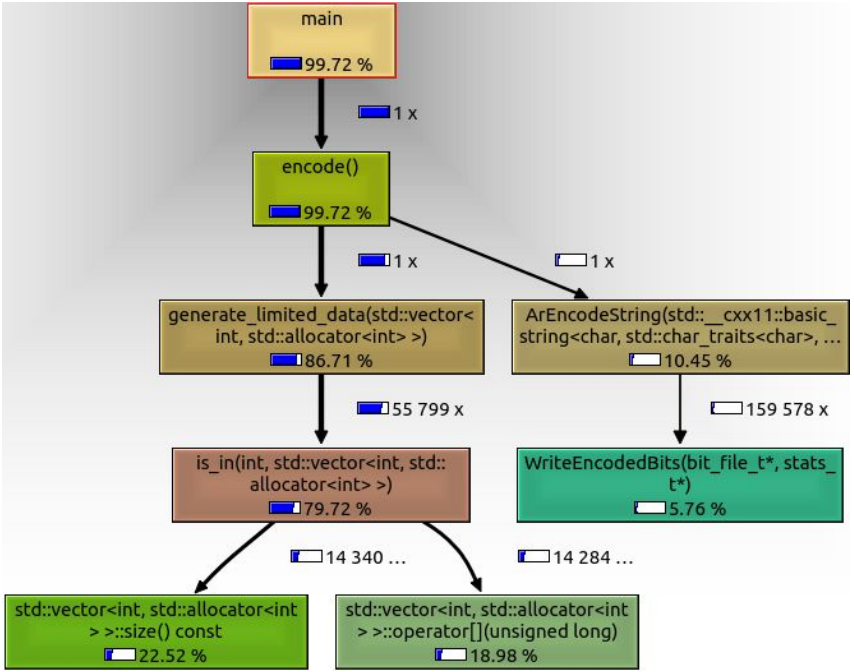
```
for r from 0 to vcoded.tamanho() do
    for i from 0 to nL do
        for j from 0 to nL do
            for k from 0 to nL do
                if vcoded[r] == K[0] * nLimited[i] + K[1] * nLimited[j] + K[2] * nLimited[k] then
                    nDecoded[3 * r + 0] ← nLimited[i]
                    nDecoded[3 * r + 1] ← nLimited[j]
                    nDecoded[3 * r + 2] ← nLimited[k]
```

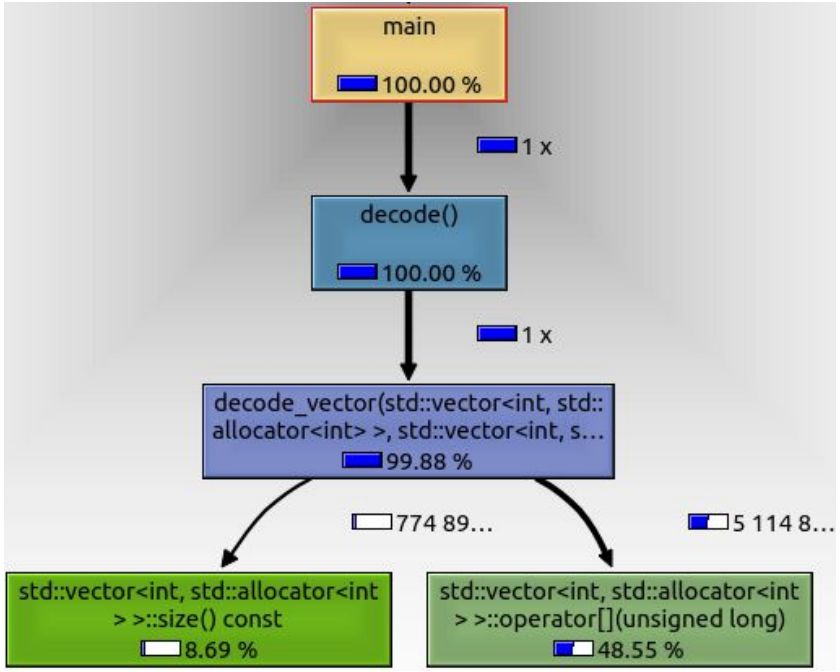
- Zeros são colocados nos intervalos indicados;
- É realizada a função inversa da DCT;

Proposta

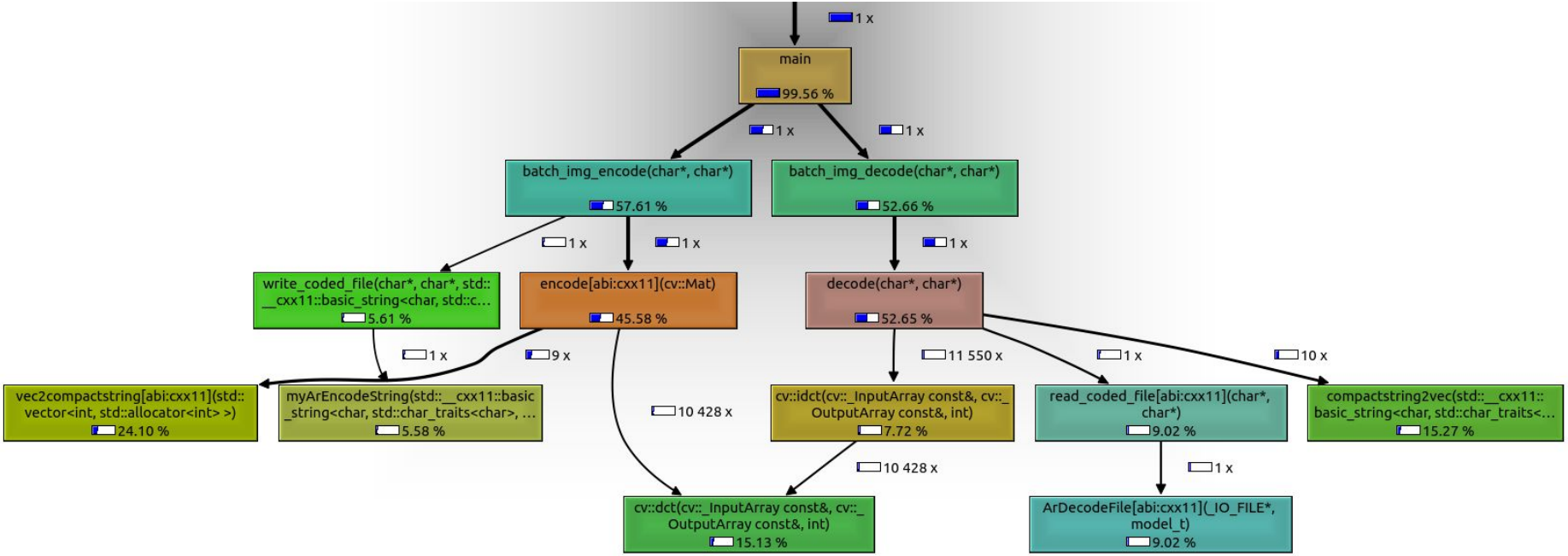
- Produção de um código limpo e otimizado em C++
- Propor uma solução paralela em OpenMP
- Realizar Experimentos

Identificação dos Trechos Mais Custosos





Identificação dos Trechos Mais Custosos



Otimizações - GMPR para Textos

```
1   bool is_in(int n, std::vector<int> vnData) {
2       bool bResult = false;
3
4       for (int i = 0; i < vnData.size(); i++) {
5           if (n == vnData[i])
6             bResult = true;
7       }
8
9       return bResult;
10  }
11
12  std::vector<int> generate_limited_data(std::vector<int> vnData) {
13      std::vector<int>tmp(256);
14      tmp[0] = vnData[0];
15      int k = 1;
16
17      for (int i = 1; i<vnData.size(); i++) {
18          if (!is_in(vnData[i], tmp)) {
19              tmp[k]=vnData[i];
20              k++;
21          }
22      }
23
24      std::vector<int>vnLimited(k);
25      for (int i = 0; i < k; i++)
26        vnLimited[i] = tmp[i];
27
28      return vnLimited;
29  }
```

```
1   bool is_in(int n, std::vector<int> vnData) {
2      return std::find(vnData.begin(), vnData.end(), n) != vnData.end();
3   }
4
5   std::vector<int> generate_limited_data(std::vector<int> vnData) {
6       std::vector<int>tmp(256);
7       tmp[0] = vnData[0];
8       int k = 1;
9
10      #pragma omp parallel for schedule(dynamic, 10)
11      for (int i = 1; i<vnData.size(); i++) {
12          if (!is_in(vnData[i], tmp)) {
13              tmp[k]=vnData[i];
14              k++;
15          }
16      }
17
18      std::vector<int>vnLimited(k);
19      for (int i = 0; i < k; i++)
20        vnLimited[i] = tmp[i];
21
22      return vnLimited;
23  }
```

Otimizações - GMPR para Textos

```
1   std::vector<int> decode_vector(std::vector<int> cdata,
2   std::vector<int> nLimited, std::vector<int> K) {
3     std::vector<int> nDecoded;
4     unsigned int nLastIndex = cdata.size();
5     unsigned int nLSize = nLimited.size();
6     nDecoded.resize(3*nLastIndex);
7
8     #pragma omp parallel for schedule(dynamic, 10)
9     for (unsigned int r = 0; r < nLastIndex; ++r) {
10      int cdatar = cdata[r];
11
12      for (unsigned int i = 0; i < nLSize; ++i) {
13        int nLimited_i = K[0] * nLimited[i];
14
15        for (unsigned int j = 0; j < nLSize; ++j) {
16          int nLimited_j = K[1] * nLimited[j];
17
18          for (unsigned int k = 0; k < nLSize; ++k) {
19              ...
20          }
21        }
22      }
23    }
24
25    return nDecoded;
26  }
```

Otimizações - GMPR para Imagens

```
1   std::string vec2compactstring(std::vector<int> vec) {
2       std::string s;
3       for (int i=0; i<vec.size(); i++)
4       {
5           int a=vec.at(i);
6           std::stringstream ss;
7           ss << a;
8           std::string str = ss.str();
9           s+=str;
10      }
11      return s;
12  }
```

```
1   std::string vec2compactstring(std::vector<int> vec) {
2       std::ostringstream vts;
3       std::copy(vec.begin(), vec.end(), std::ostream_iterator<int>(vts));
4       return vts.str();
5   }
```

Otimizações - GMPR para Imagens

```
1   cv::Mat generate_quantization_matrix(int Blocksize, int R) {
2     cv::Mat Q(Blocksize, Blocksize, CV_32F, cv::Scalar::all(0));
3
4     #pragma omp parallel for schedule(guided)
5     for (int r = 0; r < Blocksize; r++) {
6       for (int c = 0; c < Blocksize; c++) {
7         Q.at<float>(r,c) = (r == c == 0) ? 1 : (float)(r+c)*R;
8       }
9     }
10
11    return Q;
12  }
```

Otimizações - GMPR para Imagens

```
1       std::string encode(cv::Mat matImg) {
2         ...
3
4         #pragma omp parallel for schedule(guided)
5         for (int i = 0; i < planes.size(); i++) {
6           ...
7           cv::dct(roi, dst);
8           ...
9         }
10
11        ...
12
13        #pragma omp parallel for schedule(guided)
14        for (int i = 0; i < planesSize; i++) {
15          ...
16          cv::findNonZero(matBoolean, matThreeLocXY[i]);
17          ...
18          szCompactThreeData[i] = vec2compactstring(vnThreeData[i]);
19          ...
20        }
21
22        ...
23      }
```

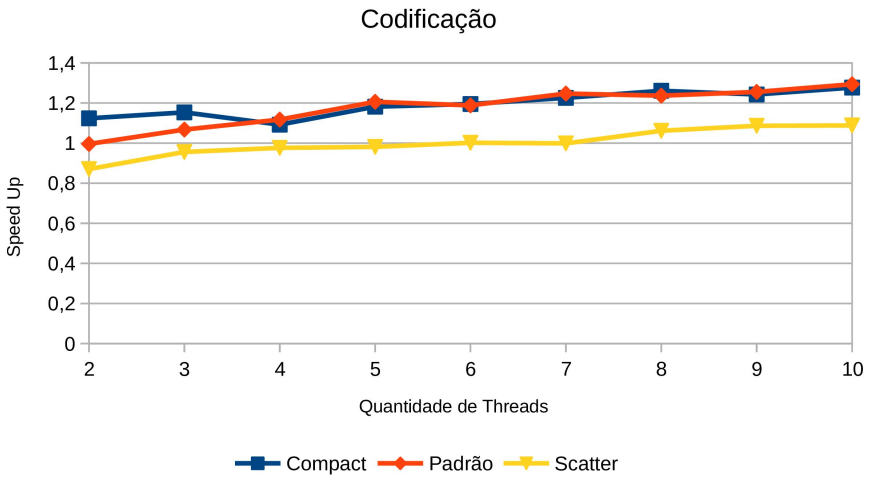
Otimizações - GMPR para Imagens

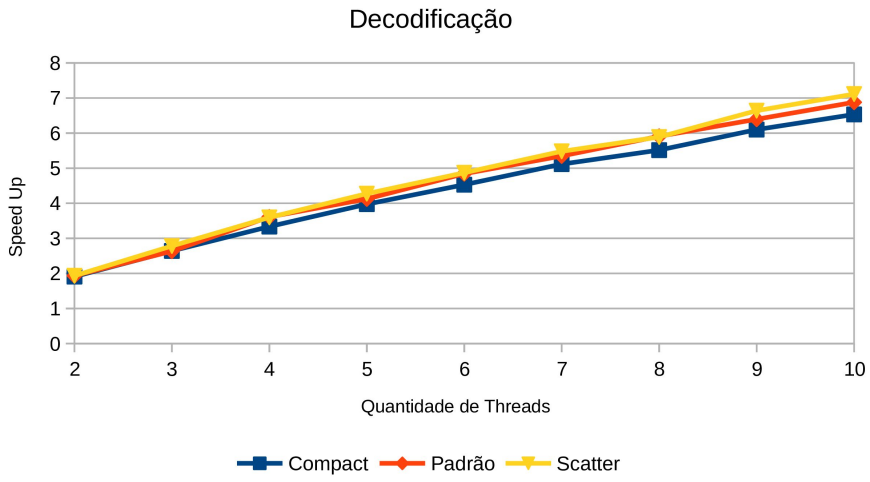
```
1   cv::Mat decode(char* filePathIn, char* fileName) {
2       ...
3
4       #pragma omp parallel for schedule(guided)
5       for (int i = 0; i < planes; i++) {
6           ...
7       }
8
9       #pragma omp parallel for schedule(guided)
10      for (int i = 0; i < planes; i++) {
11          ...
12          int index0 = 0;
13          for (int j = 0; j < vnDiffZeroLoc[i].size(); j++) {
14            index0 += vnDiffZeroLoc[i][j];
15            vec[index0] = 0;
16          }
17
18          int index = 0;
19          for (int j = 0; j < vec.size(); j++) {
20            if ( vec[j] == -1) {
21              vec[j] = vnNonZeroData[i][index];
22              index++;
23            }
24          }
25          ...
26      }
27
28      #pragma omp parallel for schedule(guided)
29      for (int i = 0; i < planes; i++) {
30          ...
31          cv::idct(roi2, dst);
32          ...
33      }
34
35      ...
36  }
```

Resultados

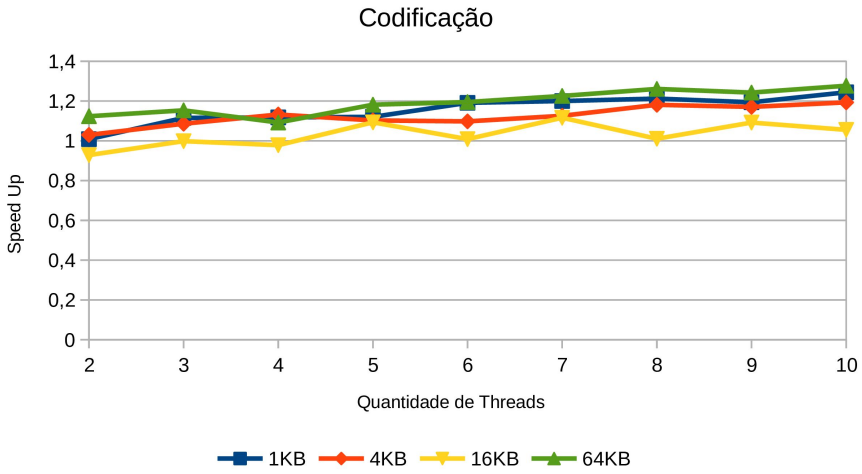
- Ambiente Experimental
  - 2 CPUs Intel(R) Xeon(R) E5-2640 (10 Núcleos Físicos e 10 Lógicos)
  - 128GB de RAM
  - NVIDIA Tesla K40c
  - Arquitetura NUMA
  - Ubuntu 16.04

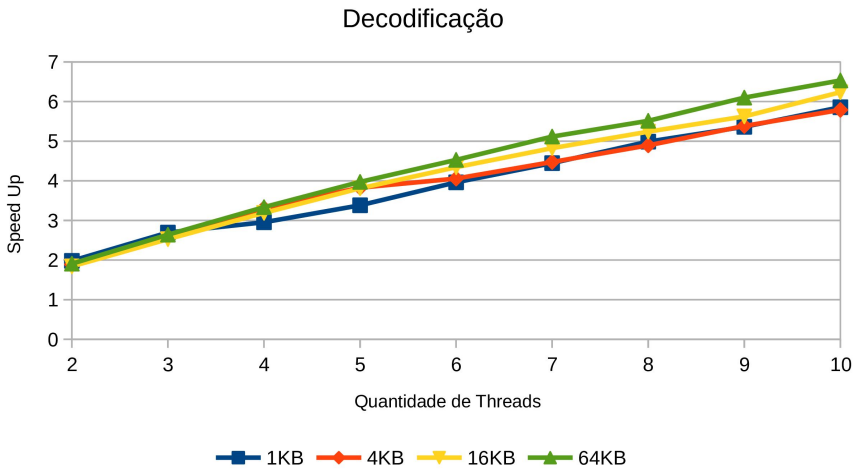
Resultados - Versão Texto



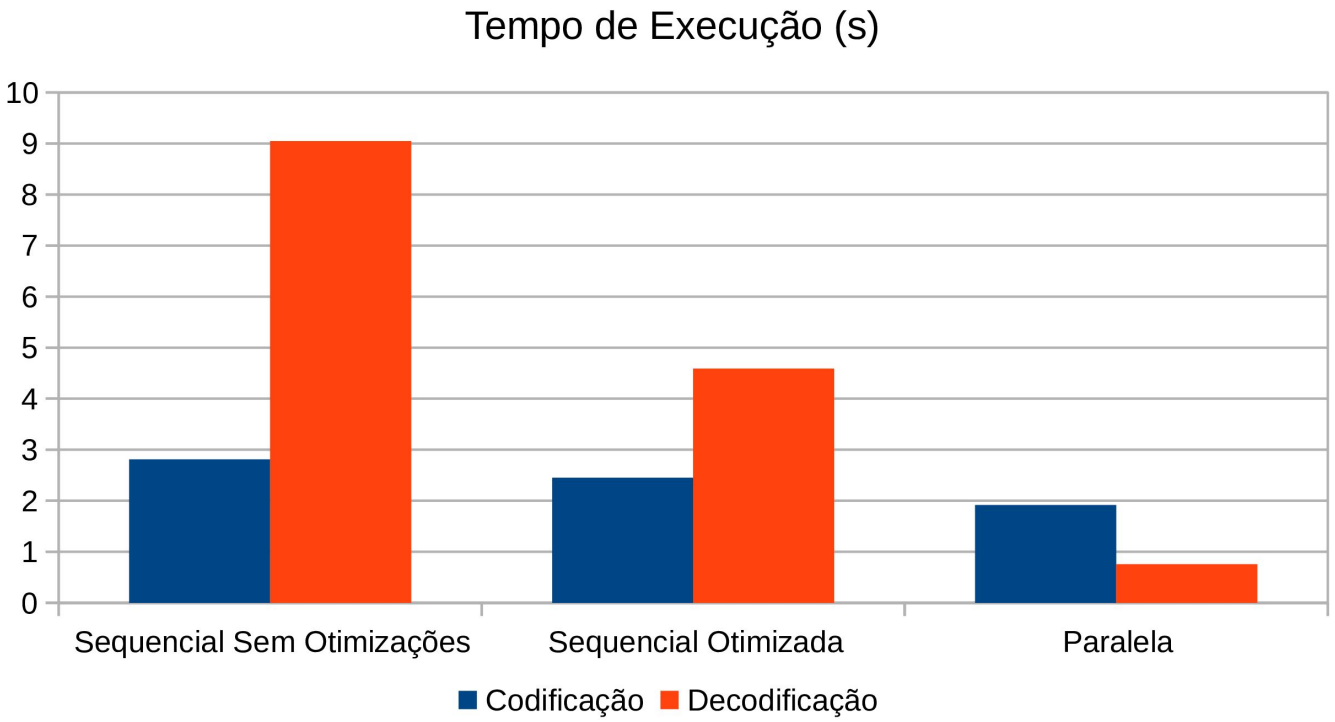


Resultados - Versão Texto

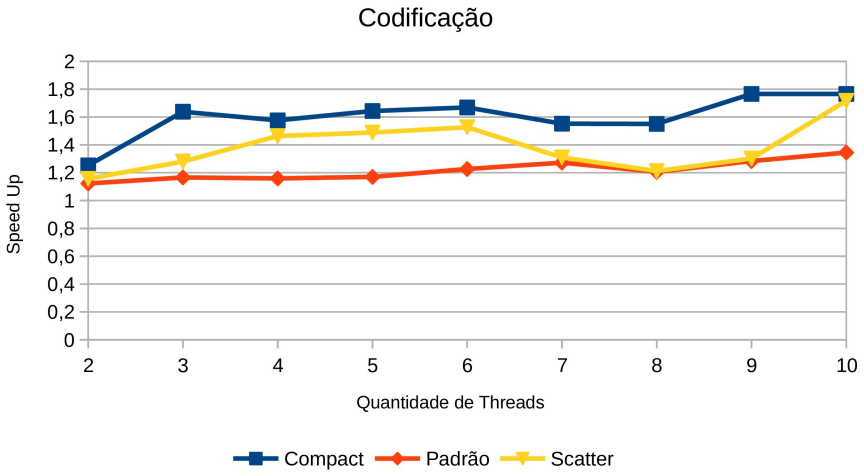


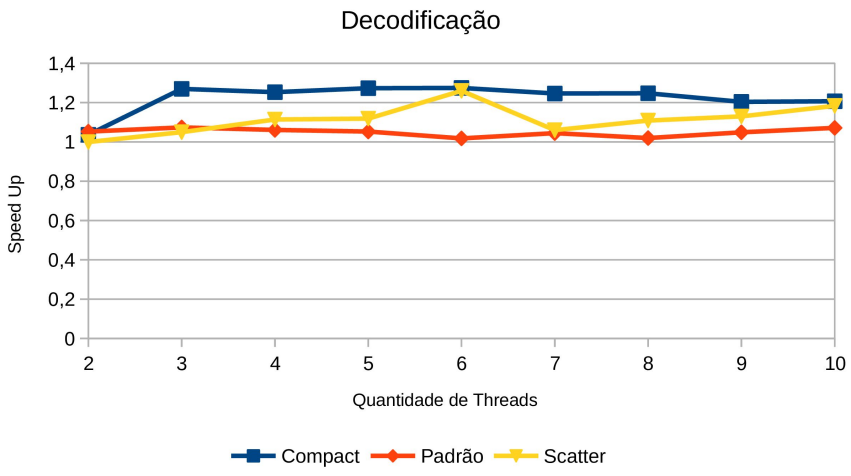


Resultados - Versão Texto

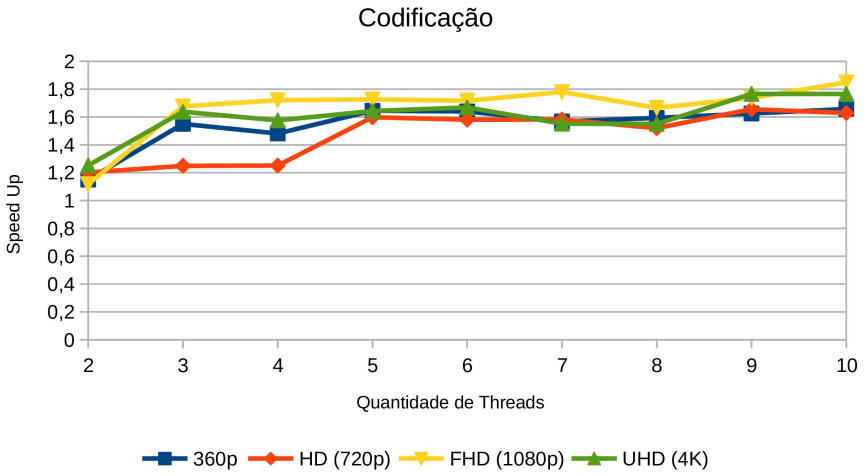


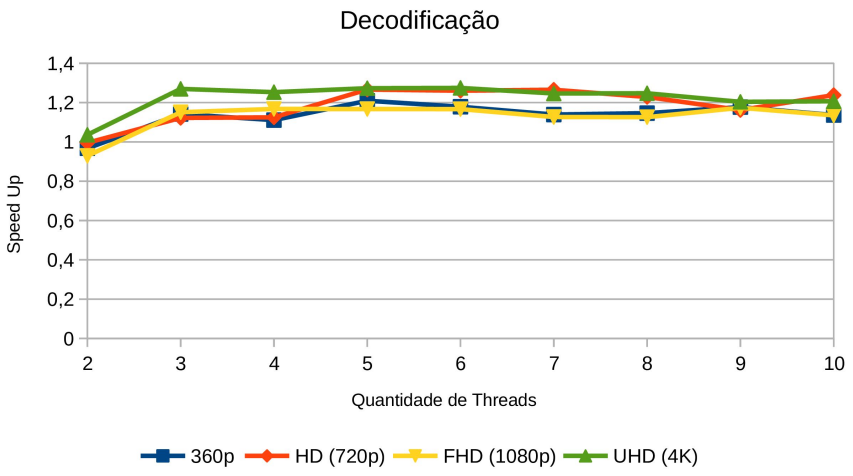
Resultados - Versão Imagens



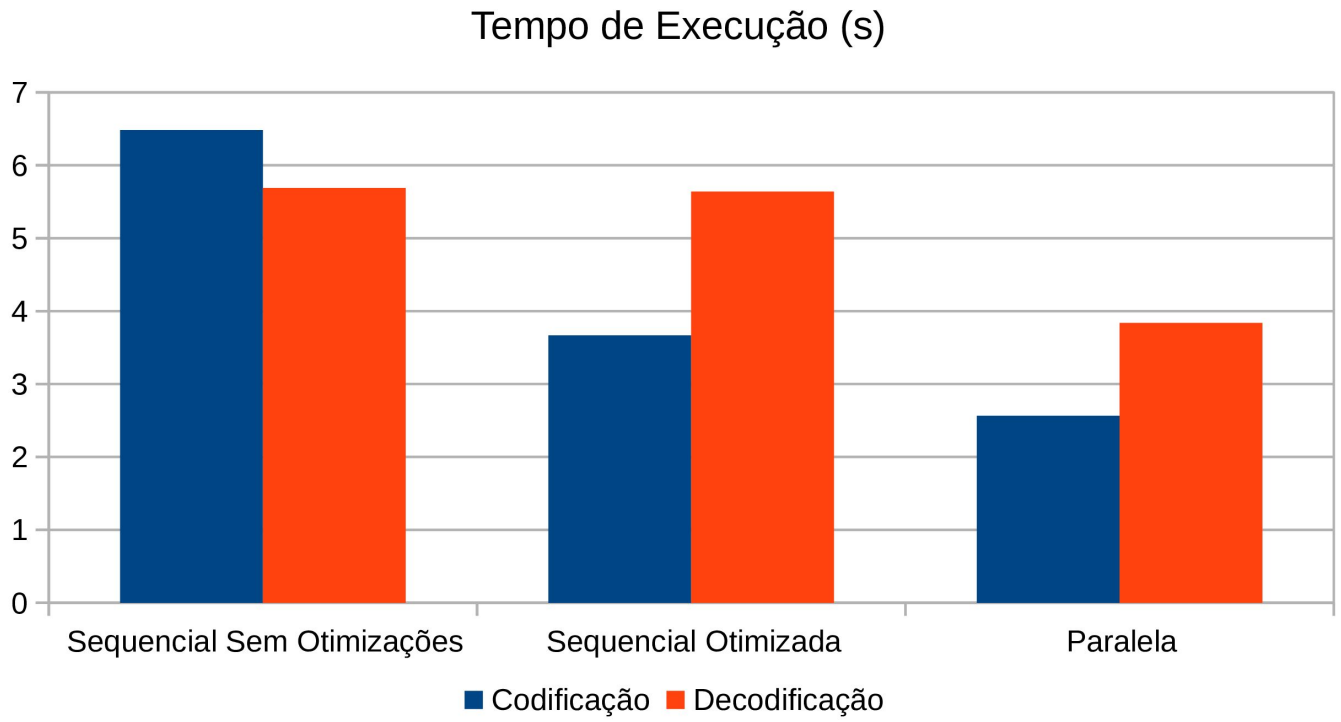


Resultados - Versão Imagens





Resultados - Versão Imagens



Considerações Finais

- Ganhos da Versão de Textos
  - 1,47x na Codificação
  - 11,9x na Decodificação
- Ganhos da Versão de Imagens
  - 2,53x na Codificação
  - 1,48x na Decodificação

- Algoritmo GMPR mais eficiente

Trabalhos Futuros

- Algoritmo pode ser adaptado para qualquer projeto com restrições de armazenamento ou segurança das informações
- Projeto pode ser aplicado em outras áreas, como a cripto-compressão de um banco de dados, dentre outras
- Podem ser estudados diferentes algoritmos de criptografia no arquivo todo e analisar os resultados em termos de desempenho

Obrigado