PROGRAMAÇÃO PARALELA

Eleição U.R.S.A.L

Urna Realmente Segura Amparada Legalmente

Flavio Vieira Leao

Moacir Mascarenha

Joao Cirqueira

Sumário

Principais Soluções

Sequencial

Paralela critical

Paralela atomic

Paralela matriz v1

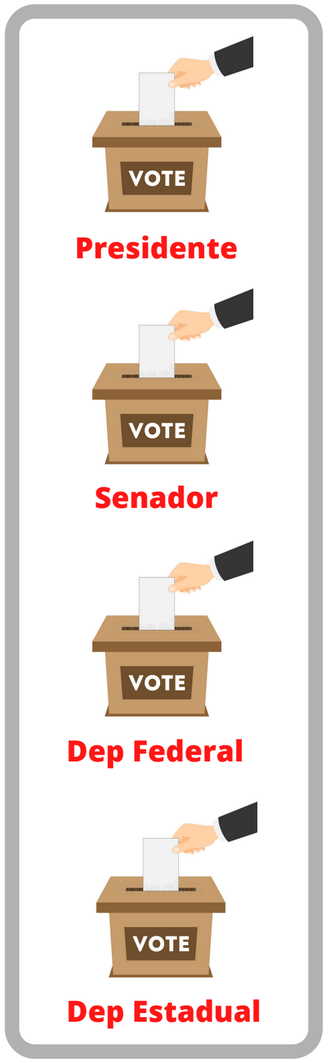
Paralela matriz strtok

Paralela matriz fastread

Contando votos em paralelo

https://moj.naquadah.com.br/contests/bcr-PSPD-2021_1/eleicao-ursal

SOLUÇÃO SEQUENCIAL

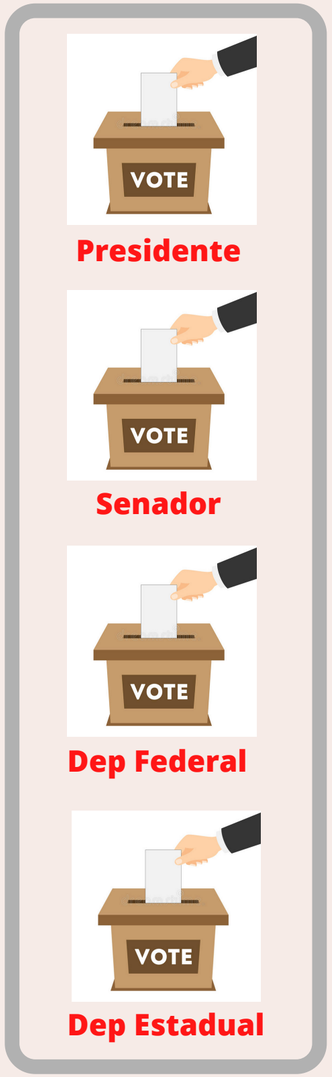


Ex: Fila com 200M linhas/votos

File-011-Big

LEITURA SEQUENCIAL ARQUIVO HD

SOLUÇÃO SEQUENCIAL

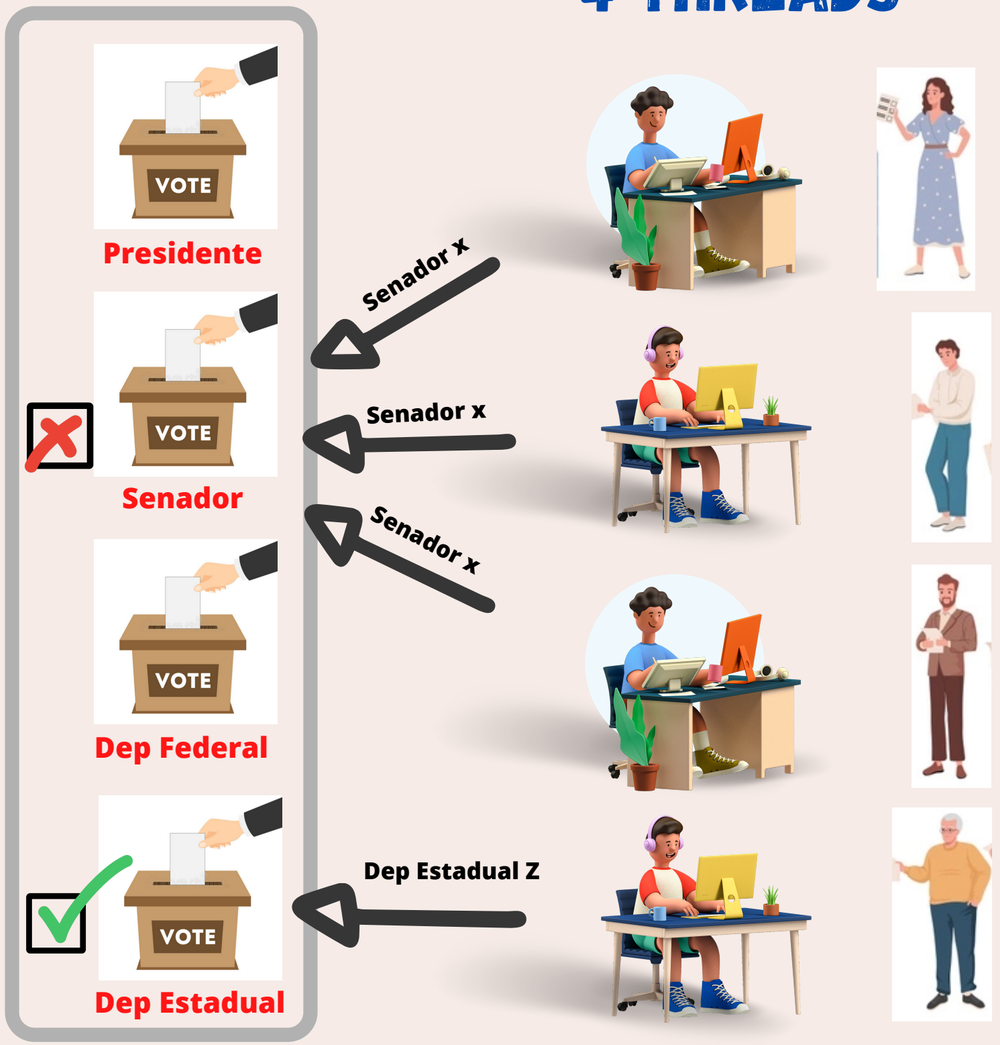


```c
int main(void)
{
    int i, j, voto, total_votos, votos_validos, votos_presidente;
    int sfe[3], sfe_offset[3];
    float p_percentual;

    Candidato registo_votos[MAX];
    memset(registo_votos, 0, sizeof(Candidato) * MAX);

    sfe_offset[0] = s_offset;
    sfe_offset[1] = f_offset;
    sfe_offset[2] = e_offset;

    for(i = 0; i < 3; i++)
        scanf("%d", &sfe[i]);


    total_votos = votos_validos = votos_presidente = 0;

    while (scanf("%d", &voto) != EOF)
    {
        if(voto > 0)
        {
            registo_votos[voto].id = voto;
            registo_votos[voto].cont_votos++;
            votos_validos++;
            if( voto < 100)
                votos_presidente++;
        }
        total_votos++;
    }
```

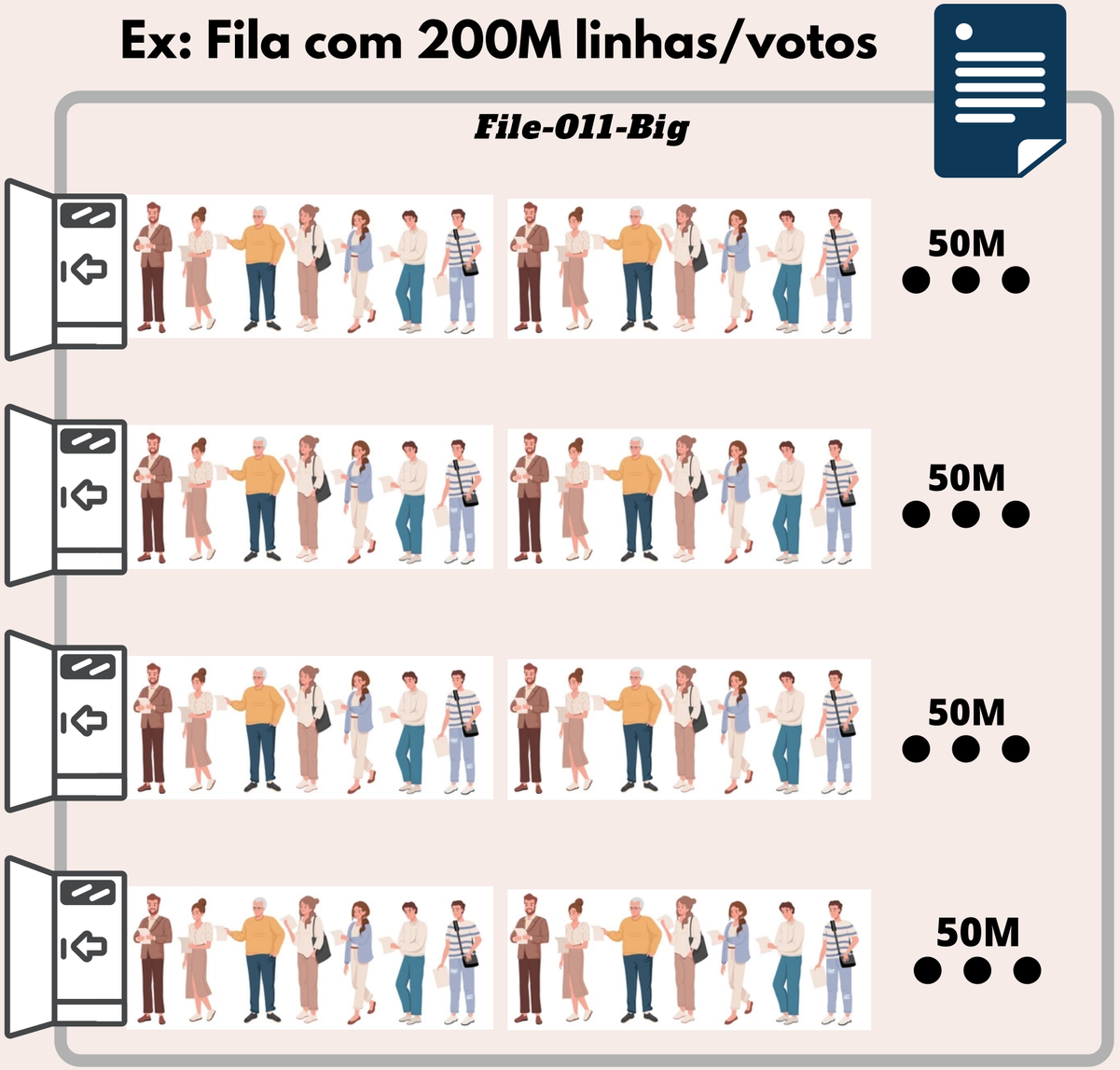
eleicao-ursal-serial.c

MEMÓRIA COMPARTILHADA

SOLUÇÃO PARALELA 4 THREADS



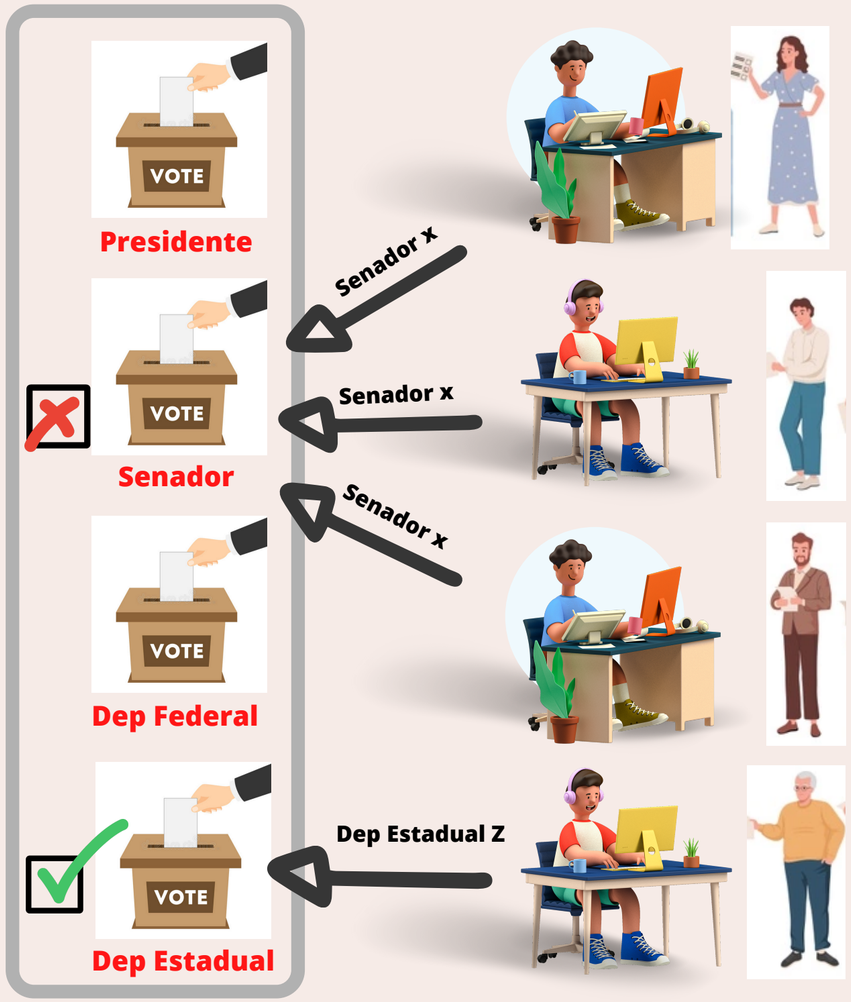
Ex: Fila com 200M linhas/votos



LEITURA PARALELA ARQUIVO HD

SOLUÇÃO PARALELA

MEMÓRIA COMPARTILHADA



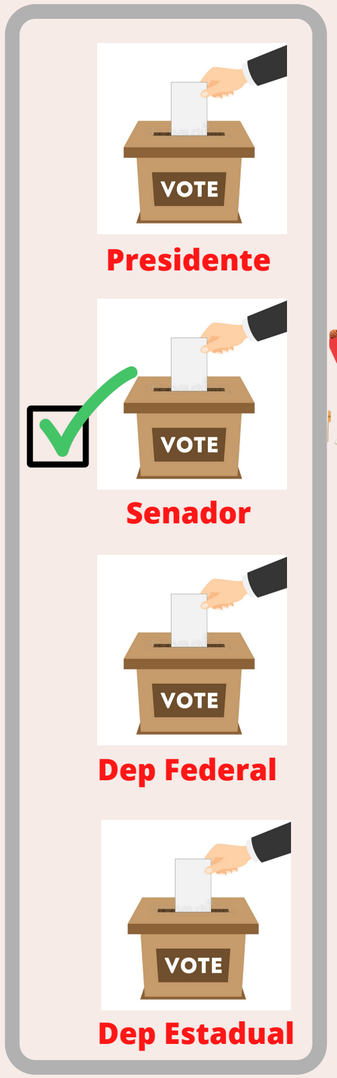

```c
# pragma omp parallel num_threads(max_threads)
{
    int id_th = omp_get_thread_num();
    int num_th = omp_get_num_threads();

    Candidato reg_votos_local[MAX];
    FILE *fp = fopen(argv[1], "r");

    int i, voto;
    int bloco, inicio, fim, resto;

    // definir blocos/chunks de leitura
    bloco = fsize / num_th;
    inicio = bloco * id_th;
    fim = inicio + bloco;

    fseek(fp, inicio, SEEK_SET);
    while(fgetc(fp) != '\n');

    while(inicio < fim)
    {
        fscanf(fp, "%d", &voto);
        if(voto > 0)
        {
            reg_votos_global[voto].cont_votos++;
            reg_votos_global[voto].id = voto;

            if( voto < 100)
                votos_pres++;

            votos_val++;
        }
        num_votos++;
        inicio = ftell(fp);
    }
```

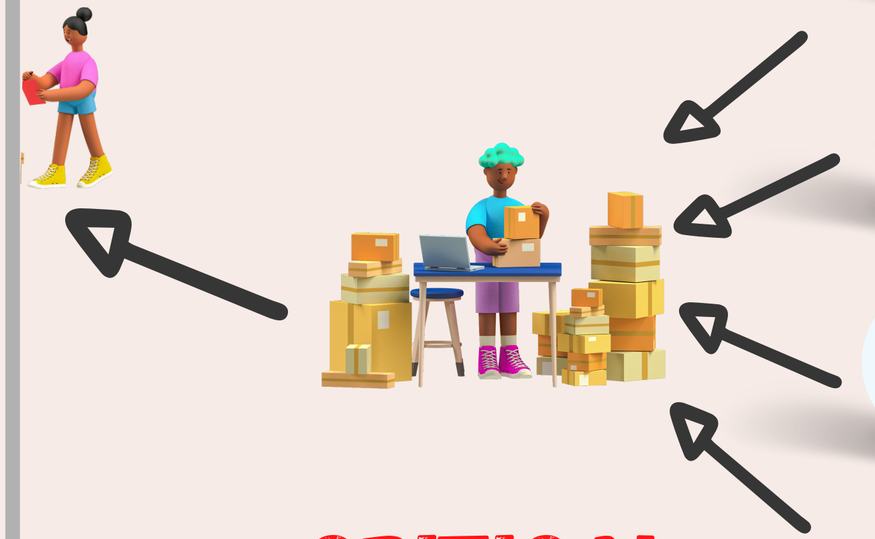
```c
typedef struct
{
    int id;
    int cont_votos;
}Candidato;

int comparaVotos (const void *x, const void *y);
void resultado(void);

// ================================  Main ================================
int main( int argc, char *argv[] )
{
    int i, j, n, fsize, num_reg;
    int sfe[3];
    int sfe_offset[3];
    int num_votos, votos_val, votos_pres;
    float percent;
    FILE *fp;
    Candidato reg_votos_global[MAX];

    memset(reg_votos_global, 0, sizeof(Candidato) * MAX);
    fp = fopen(argv[1], "r");

    sfe_offset[0] = s_offset;
    sfe_offset[1] = f_offset;
    sfe_offset[2] = e_offset;

    num_votos = votos_val = votos_pres = num_reg = 0;

    fseek(fp, 0, SEEK_END);
    fsize = ftell(fp) - 1;

    rewind(fp);
    for(i = 0; i < 3; i++)
        fscanf(fp, "%d", &sfe[i]);
```

eleicao-ursal-paralela.c

SOLUÇÃO PARALELA COM CRITICAL

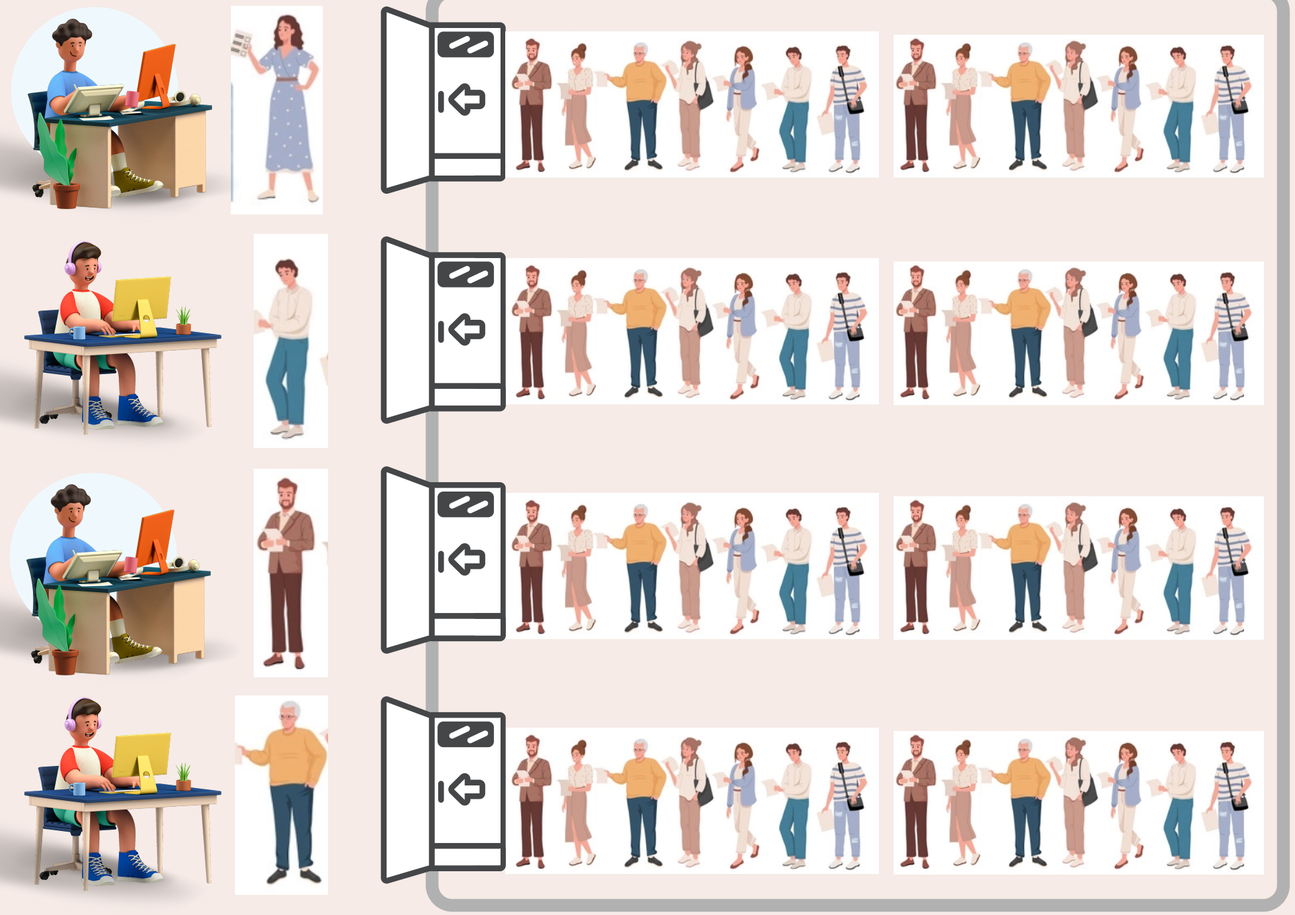
MEMÓRIA COMPARTILHADA





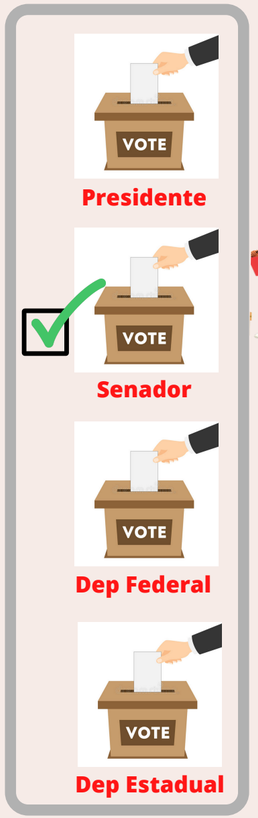
CRITICAL

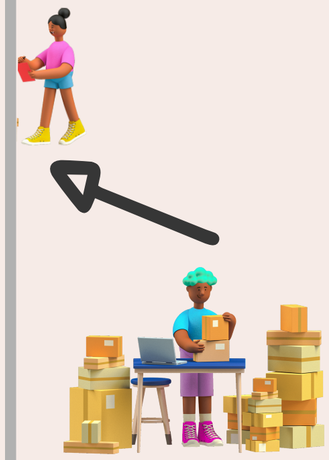
ÚNICA THREAD DE CADA VEZ



LEITURA PARALELA ARQUIVO HD

SOLUÇÃO PARALELA COM CRITICAL





CRITICAL

MEMÓRIA COMPARTILHADA

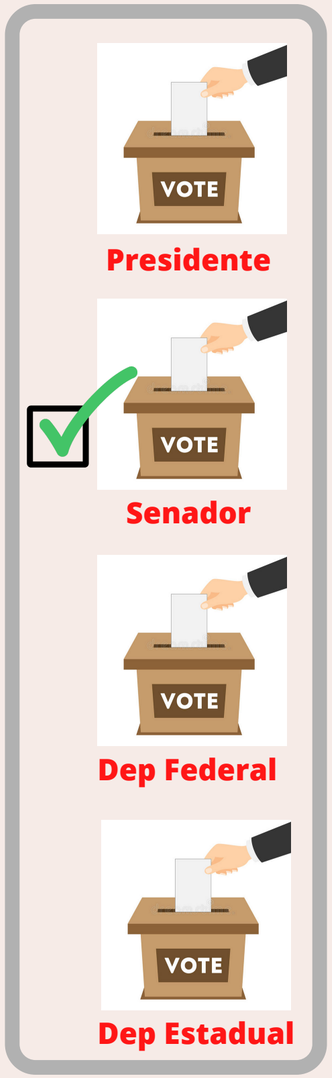
```c
while(inicio < fim)
{
    fscanf(fp, "%d", &voto);
    if(voto > 0)
    {
        #pragma omp critical
        {
            reg_votos_global[voto].cont_votos++;
            votos_val++;
        }
        reg_votos_global[voto].id = voto;


        if( voto < 100){
            #pragma omp critical
                votos_pres++;
        }

    }
    #pragma omp critical
        num_votos++;

    inicio = ftell(fp);


}
fclose(fp);
```

```c
# pragma omp parallel num_threads(max_threads)
{
    int id_th = omp_get_thread_num();
    int num_th = omp_get_num_threads();

    Candidato reg_votos_local[MAX];
    FILE *fp = fopen(argv[1], "r");

    int i, voto;
    int bloco, inicio, fim, resto;

    // definir blocos/chunks de leitura
    bloco = fsize / num_th;
    inicio = bloco * id_th;
    fim = inicio + bloco;

    if( id_th == num_th - 1)
    {
        resto = fsize % num_th;
        fim += resto;
    }

    fseek(fp, inicio, SEEK_SET);
    while(fgetc(fp) != '\n');
```

eleicao-ursal-critical.c

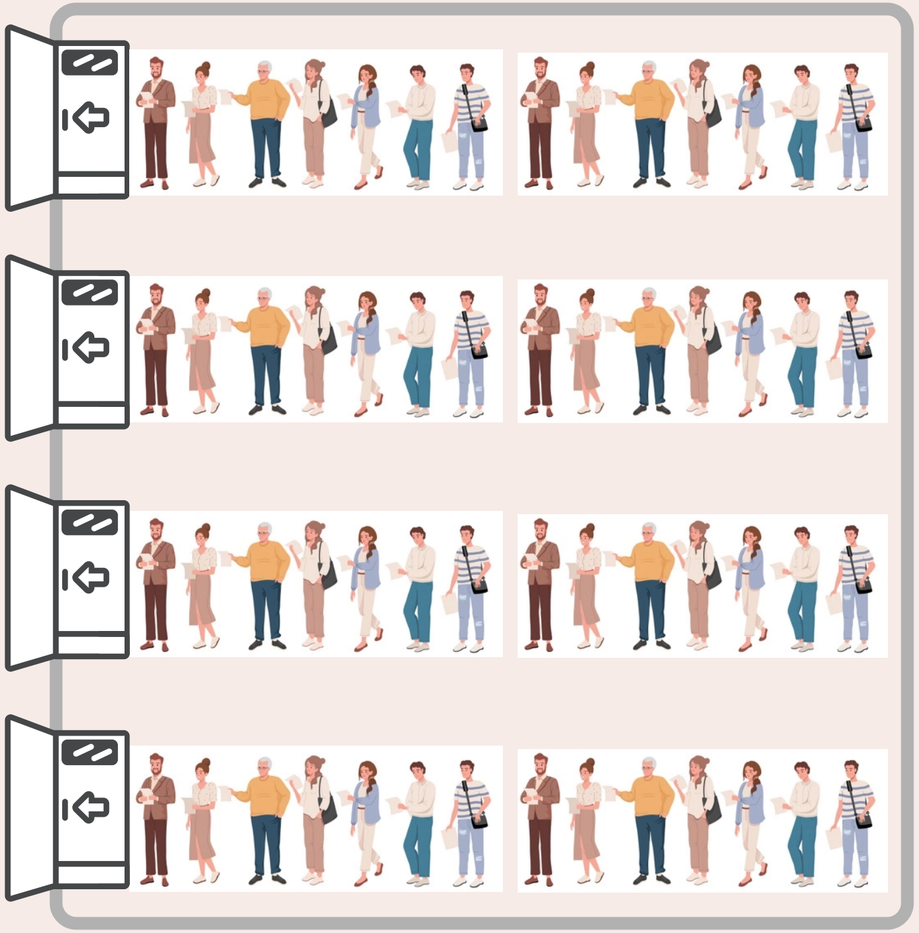
SOLUÇÃO PARALELA COM ATOMIC

MEMÓRIA COMPARTILHADA



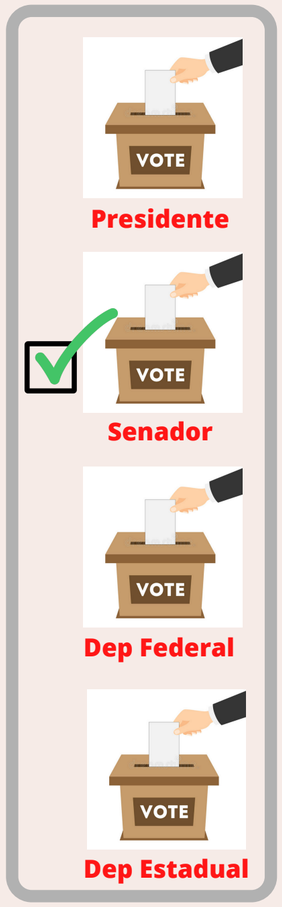
ATOMIC

ÚNICA THREAD DE CADA VEZ



LEITURA PARALELA ARQUIVO HD

SOLUÇÃO PARALELA COM ATOMIC



MEMÓRIA COMPARTILHADA

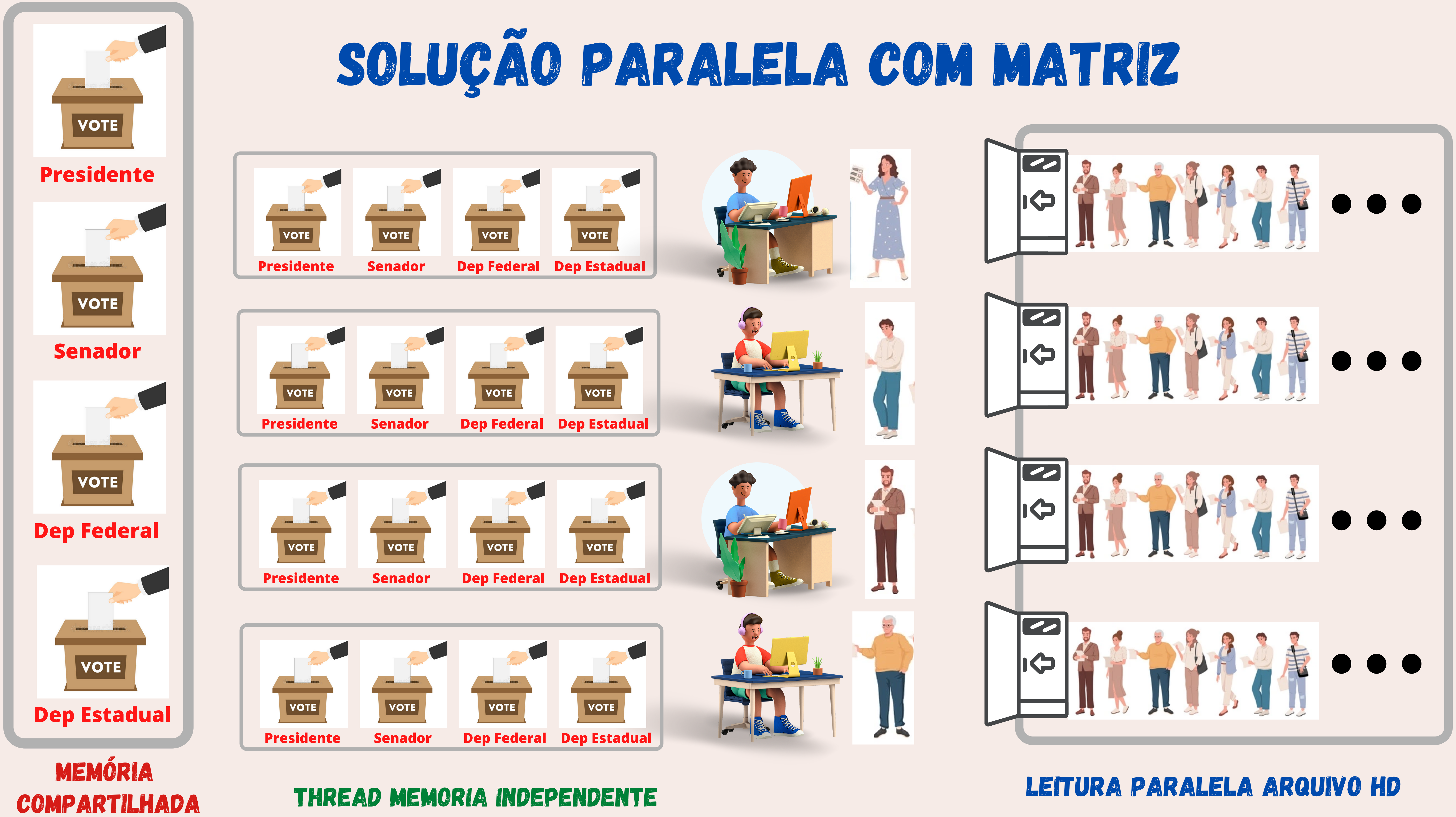
ATOMIC

```c
fseek(fp, inicio, SEEK_SET);
while(fgetc(fp) != '\n');

while(inicio < fim)
{
  fscanf(fp, "%d", &voto);
  if(voto > 0)
  {
    #pragma omp atomic
      reg_votos_global[voto].cont_voto
    reg_votos_global[voto].id = voto;

    if( voto < 100){
      #pragma omp atomic
        votos_pres++;
    }
    #pragma omp atomic
      votos_val++;
  }
  #pragma omp atomic
    num_votos++;

  inicio = ftell(fp);
}
fclose(fp);
```

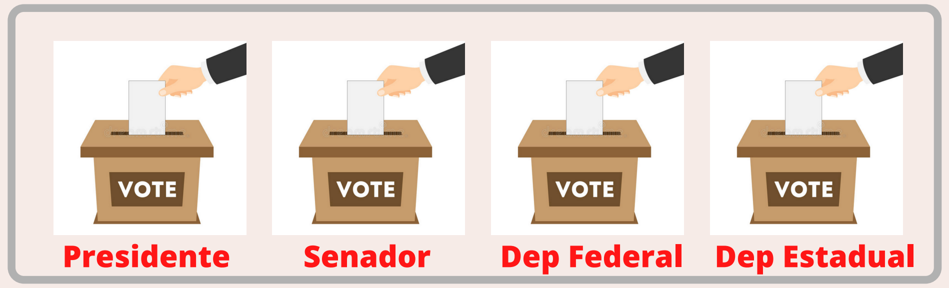
```c
# pragma omp parallel num_threads(max_threads)
{
    int id_th = omp_get_thread_num();
    int num_th = omp_get_num_threads();

    Candidato reg_votos_local[MAX];
    FILE *fp = fopen(argv[1], "r");

    int i, voto;
    int bloco, inicio, fim, resto;

    // definir blocos/chunks de leitura
    bloco = fsize / num_th;
    inicio = bloco * id_th;
    fim = inicio + bloco;

    if( id_th == num_th - 1)
    {
        resto = fsize % num_th;
        fim += resto;
    }

    fseek(fp, inicio, SEEK_SET);
    while(fgetc(fp) != '\n');
```

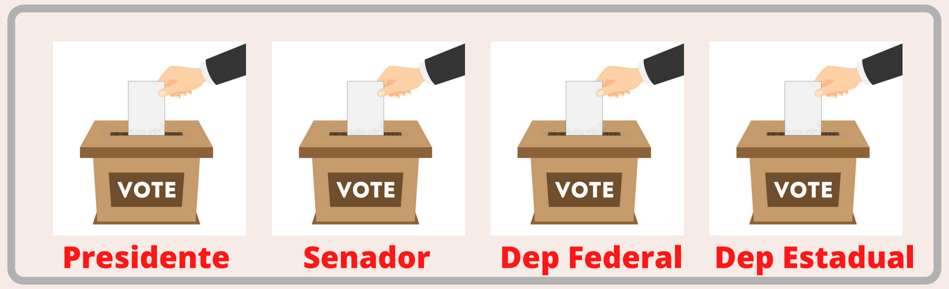
eleicao-ursal-atomic

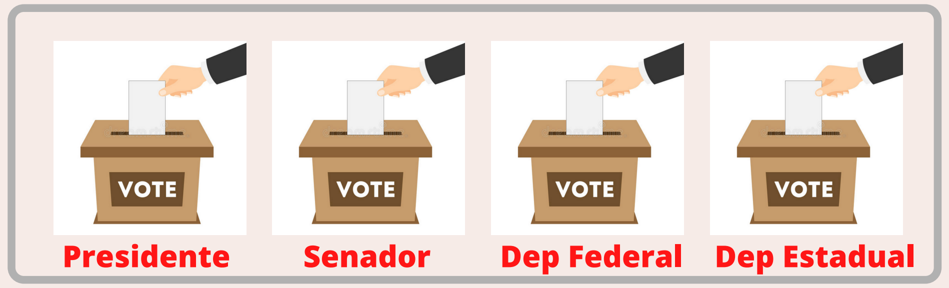
SOLUÇÃO PARALELA COM MATRIZ

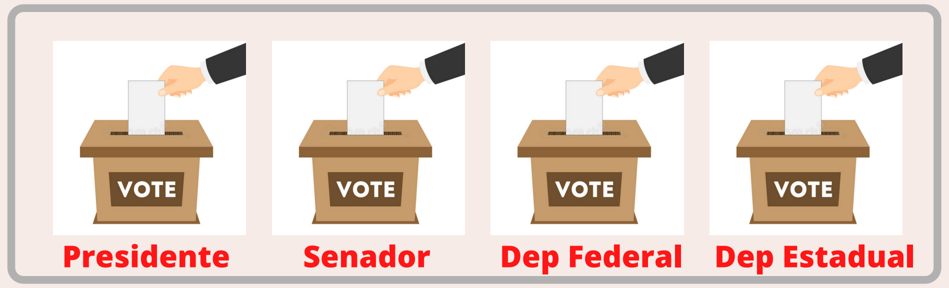


SOLUÇÃO PARALELA COM MATRIZ





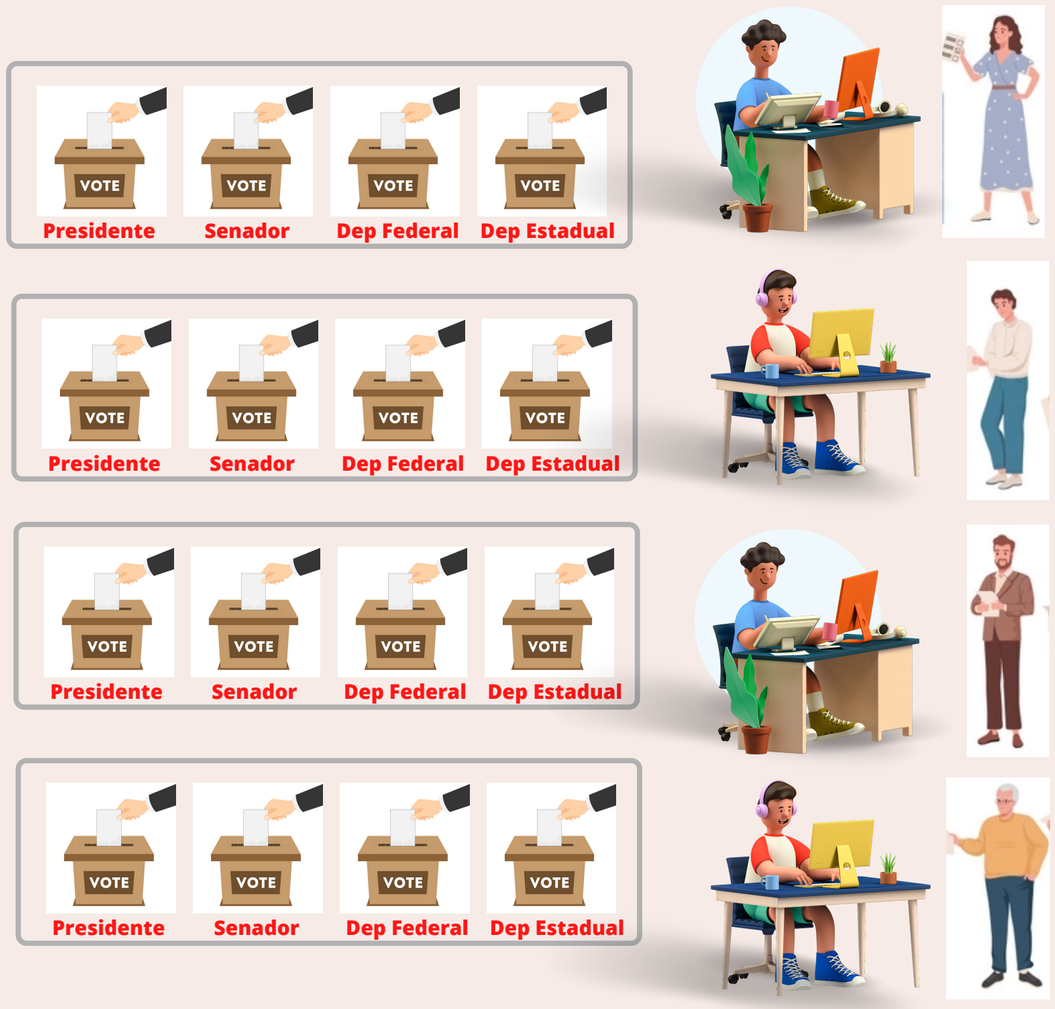





```c
// ============================================================================
typedef struct{
    int id;
    int cont_votos;
}Candidato;

typedef struct {
    int *votos_presidente;
    int *votos_validos;
    int *votos_total;
}Arg_votos;


// ============================================================================
    arg_voto.votos_total      = (int* ) calloc(num_th + 1, sizeof(int));
    arg_voto.votos_validos    = (int* ) calloc(num_th + 1, sizeof(int));
    arg_voto.votos_presidente = (int* ) calloc(num_th + 1, sizeof(int));

    matriz_votos_global       = (int**) calloc(num_th,     sizeof(int*));
    for(i = 0; i < num_th; i++)
        matriz_votos_global[i] = (int*) calloc(MAX, sizeof(int));

    votos_unidos = (Candidato*) calloc(MAX, sizeof(Candidato));

    fp = fopen(argv[1], "r");

    for(i = 0; i < 3; i++)
        fscanf(fp, "%d", &sfe[i]);

    fseek(fp, 0, SEEK_END);
    fsize = ftell(fp) - 1;

//============================================================================
```

eleicao-ursal-matriz.c

SOLUÇÃO PARALELA COM MATRIZ

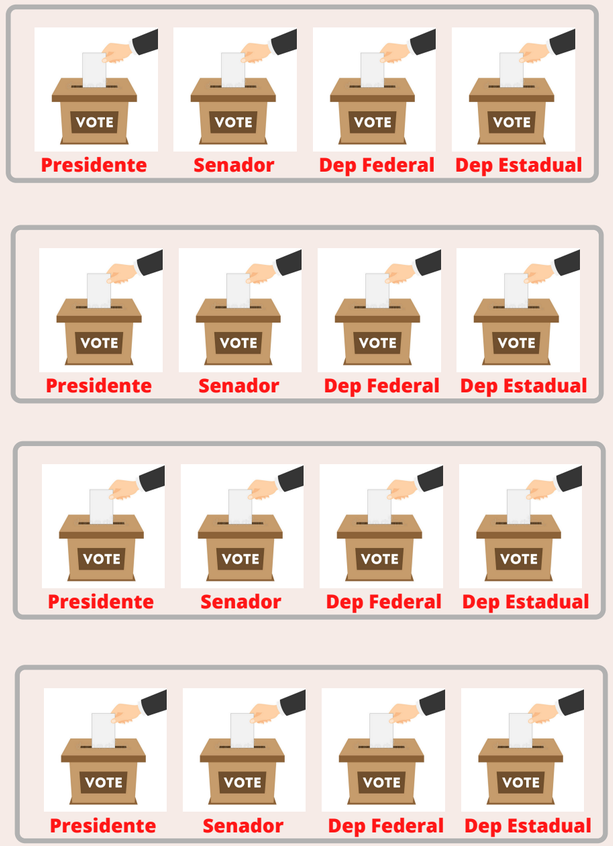



```c
void ler_arquivo(int file_size, char *arquivo)
{

    int i, id_th, voto, bloco, inicio, fim, resto;

    int votos_invalidos_local = 0,
        votos_validos_local = 0,
        votos_presidente_local = 0;

    id_th = omp_get_thread_num();
    FILE *fp = fopen(arquivo, "r");

    // definir blocos/chunks de leitura
    bloco = file_size / num_th;
    inicio = bloco * id_th;
    fim = inicio + bloco;

    fseek(fp, inicio, SEEK_SET);
    while(fgetc(fp) != '\n');

    while(inicio < fim)
    {
        fscanf(fp, "%d", &voto);
        if(voto > 0)
        {
            votos_validos_local++;
            matriz_votos_global[id_th][voto]++;

            if( voto < 100)
                votos_presidente_local++;
        }
        else
            votos_invalidos_local++;
        inicio = ftell(fp);
    }
    votos_invalidos[id_th]  = votos_invalidos_local;
    votos_presidente[id_th] = votos_presidente_local;
    votos_validos[id_th]    = votos_validos_local;
}
```

eleicao-ursal-matriz.c

SOLUÇÃO EM MEMÓRIA

MATRIZ + STRTOK()

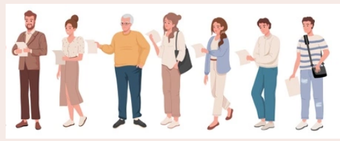


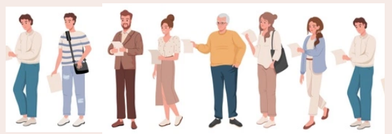
STRTOK()

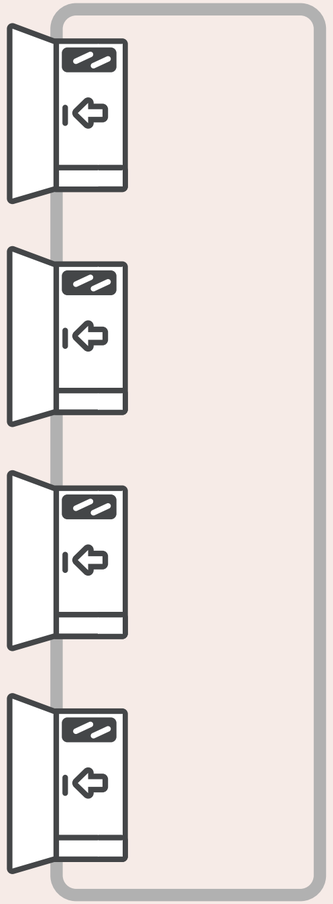
STRTOK()

STRTOK()

STRTOK()







SELEÇÃO E ORDENAÇÃO
BLOCOS EM MEMÓRIA

BLOCOS EM MEMORIA

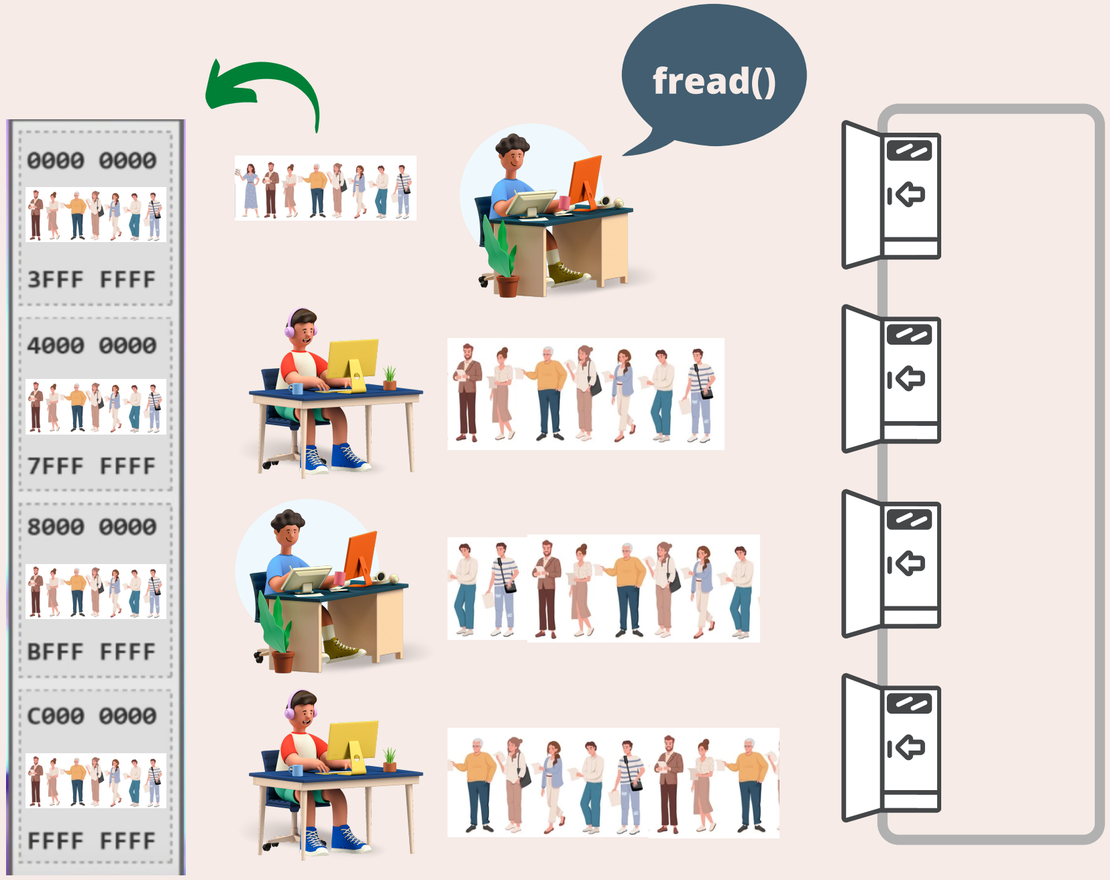
LEITURA PARALELA
BLOCOS ARQUIVO HD

```c
# pragma omp parallel num_threads(num_th)
void ler_arquivo(int file_size, int l1, char *arquivo)
{
    int i, id_th, voto, chunk, init, fim, buf_size, buf_size_th;
    char *p_split = NULL;
    char *p_save = NULL;
    char *buffer;       // buffer[fsize]

    int votos_invalidos_local = 0,
        votos_validos_local = 0,
        votos_presidente_local = 0;

    id_th = omp_get_thread_num();
    FILE *fp = fopen(arquivo, "r");

    // definir blocos/chunks de leitura
    chunk = (file_size - l1) / num_th;
    init = (chunk * id_th) + l1;

    fseek(fp, init+chunk, SEEK_SET);
    while(fgetc(fp) != '\n')
        chunk++;

    rewind(fp);
    fseek(fp, init, SEEK_SET);
    while(fgetc(fp) != '\n')
        chunk--;

    buffer = (char *) malloc(sizeof(int) * chunk);
    fread(buffer, chunk, 1, fp);
```

eleicao-ursal-strtok.c

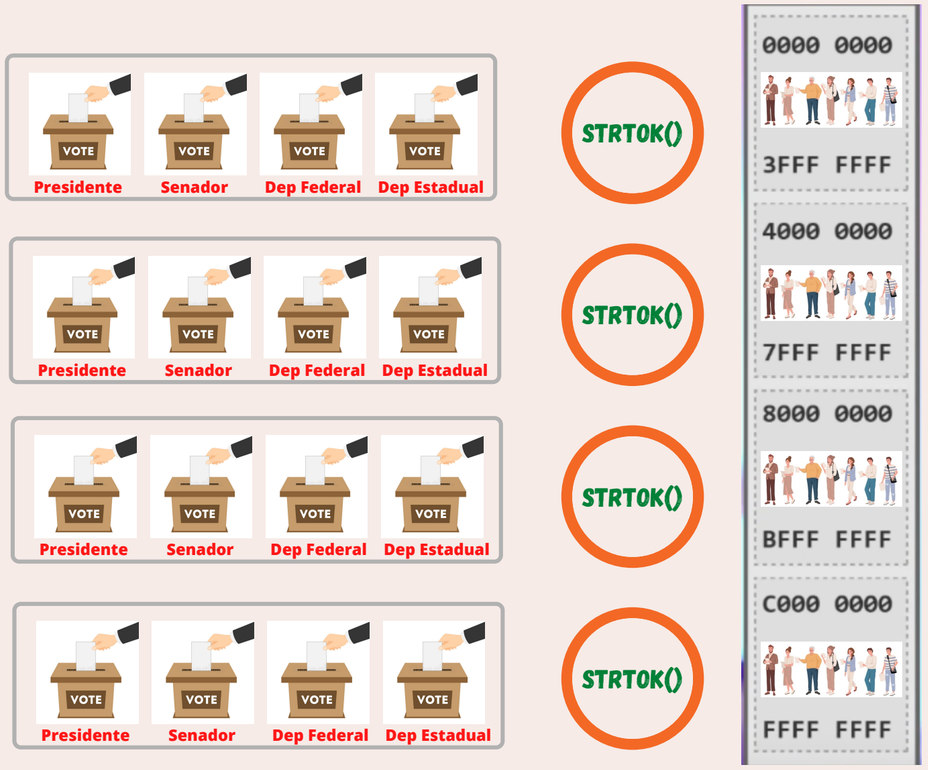
SOLUÇÃO EM MEMÓRIA

MATRIZ + STRTOK()



SOLUÇÃO EM MEMÓRIA

MATRIZ + STRTOK()

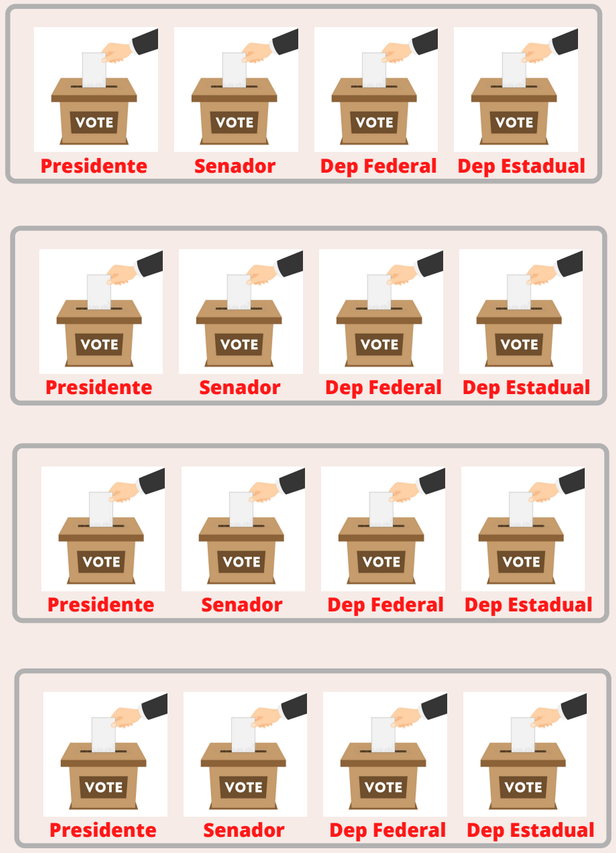



```c
    buffer = (char *) malloc(sizeof(int) * chunk);
    fread(buffer, chunk, 1, fp);

    for (p_split = strtok_r(buffer, "\n", &p_save);
         p_split != NULL;
         p_split = strtok_r(NULL, "\n", &p_save))
    {
        voto = atoi(p_split);
        if(voto > 0)
        {
            votos_validos_local++;
            matriz_votos_global[id_th][voto]++;

            if( voto < 100)
                votos_presidente_local++;
        }
        else
            votos_invalidos_local++;

    }
    votos_invalidos[id_th]  = votos_invalidos_local;
    votos_presidente[id_th] = votos_presidente_local;
    votos_validos[id_th]    = votos_validos_local;

    fclose(fp);
}
```

eleicao-ursal-strtok.c

SOLUÇÃO EM MEMÓRIA

MATRIZ + FAST_READ()



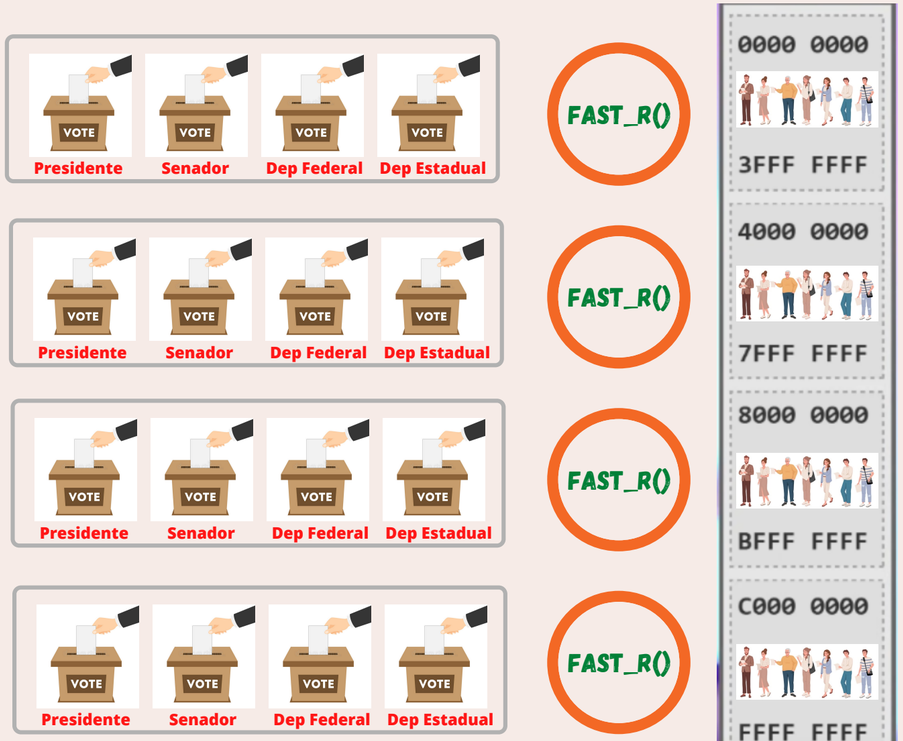
SELEÇÃO E ORDENAÇÃO
BLOCOS EM MEMÓRIA

BLOCOS EM MEMORIA

LEITURA PARALELA
BLOCOS ARQUIVO HD

SOLUÇÃO EM MEMÓRIA

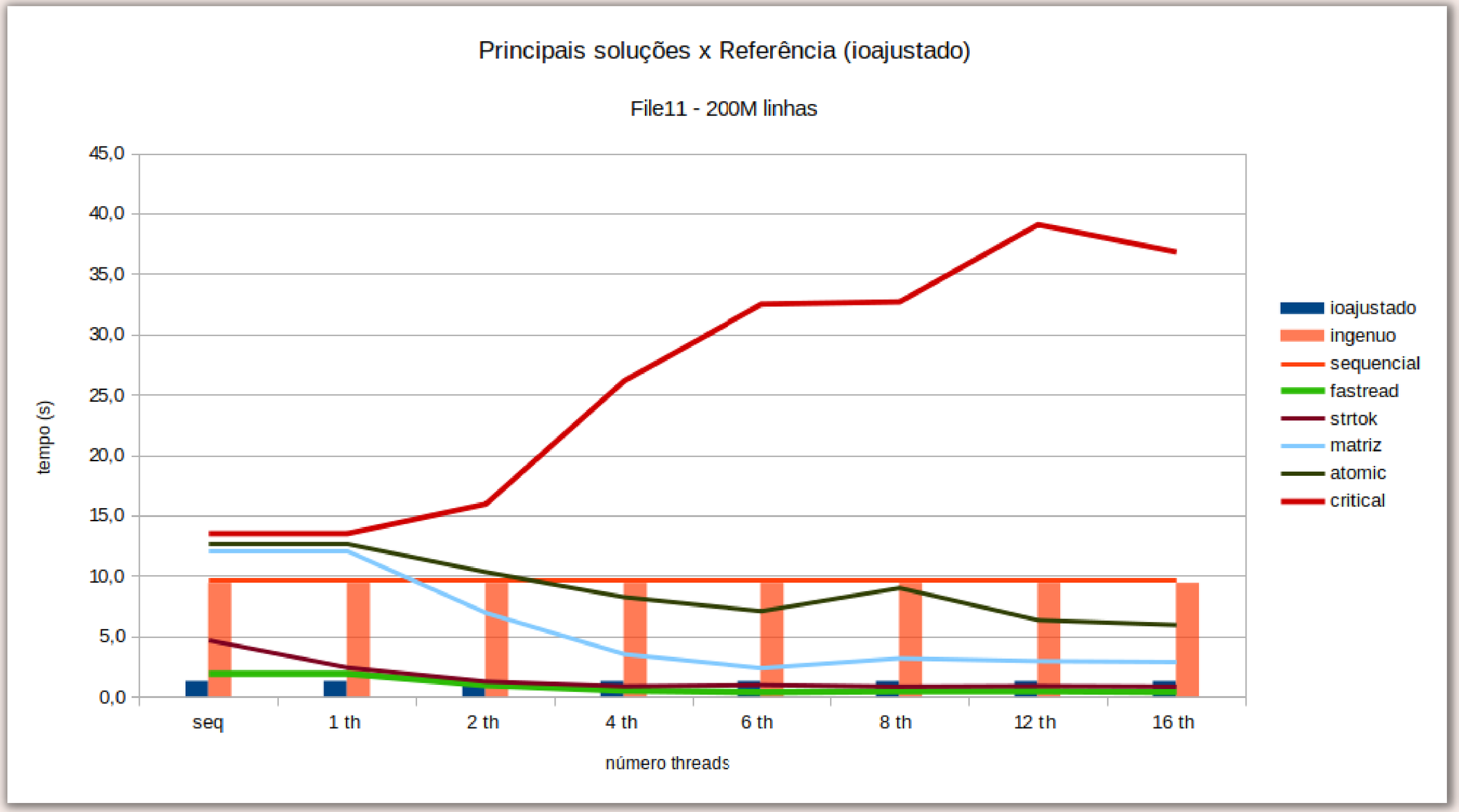
MATRIZ + FAST_READ()



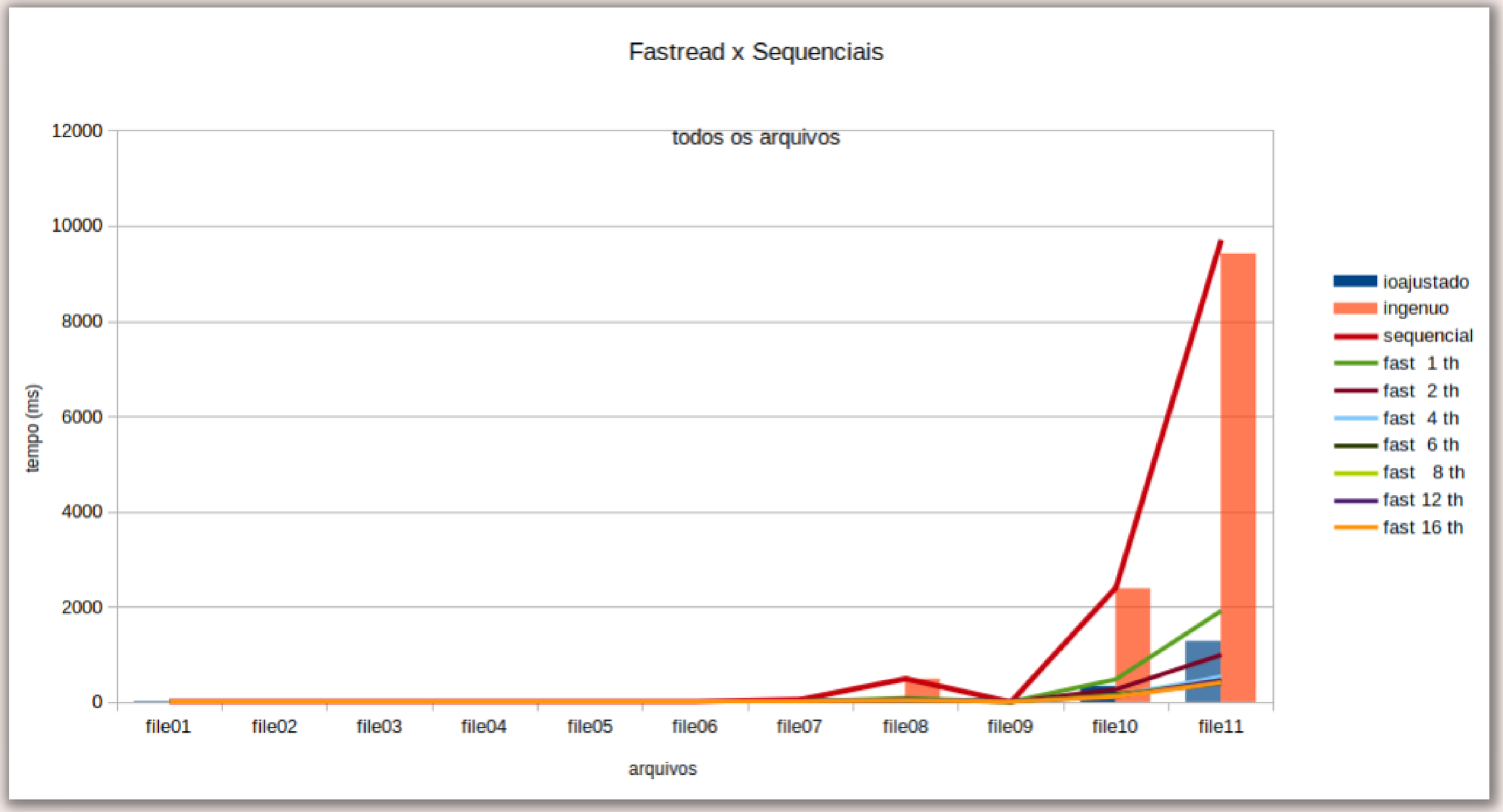

```c
buffer = (char *) malloc(sizeof(int) * buf_size_th);
fread(buffer, buf_size_th, 1, fp);

voto = i = 0;
while(i < buf_size_th)
{
    c = buffer[i++];
    if(c == '-')
    {
        while(buffer[i++] != '\n');
        votos_invalidos_local++;
    }
    else
    {
        if( c != '\n')
            voto = voto * 10 + c - 48;
        else
        {
            votos_validos_local++;
            matriz_votos_global[id_th][voto]++;

            if( voto < 100)
                votos_presidente_local++;
            voto = 0;
        }
    }
}

votos_invalidos[id_th]  = votos_invalidos_local;
votos_presidente[id_th] = votos_presidente_local;
votos_validos[id_th]    = votos_validos_local;

fclose(fp);
}
```

eleicao-ursal-readfast.c

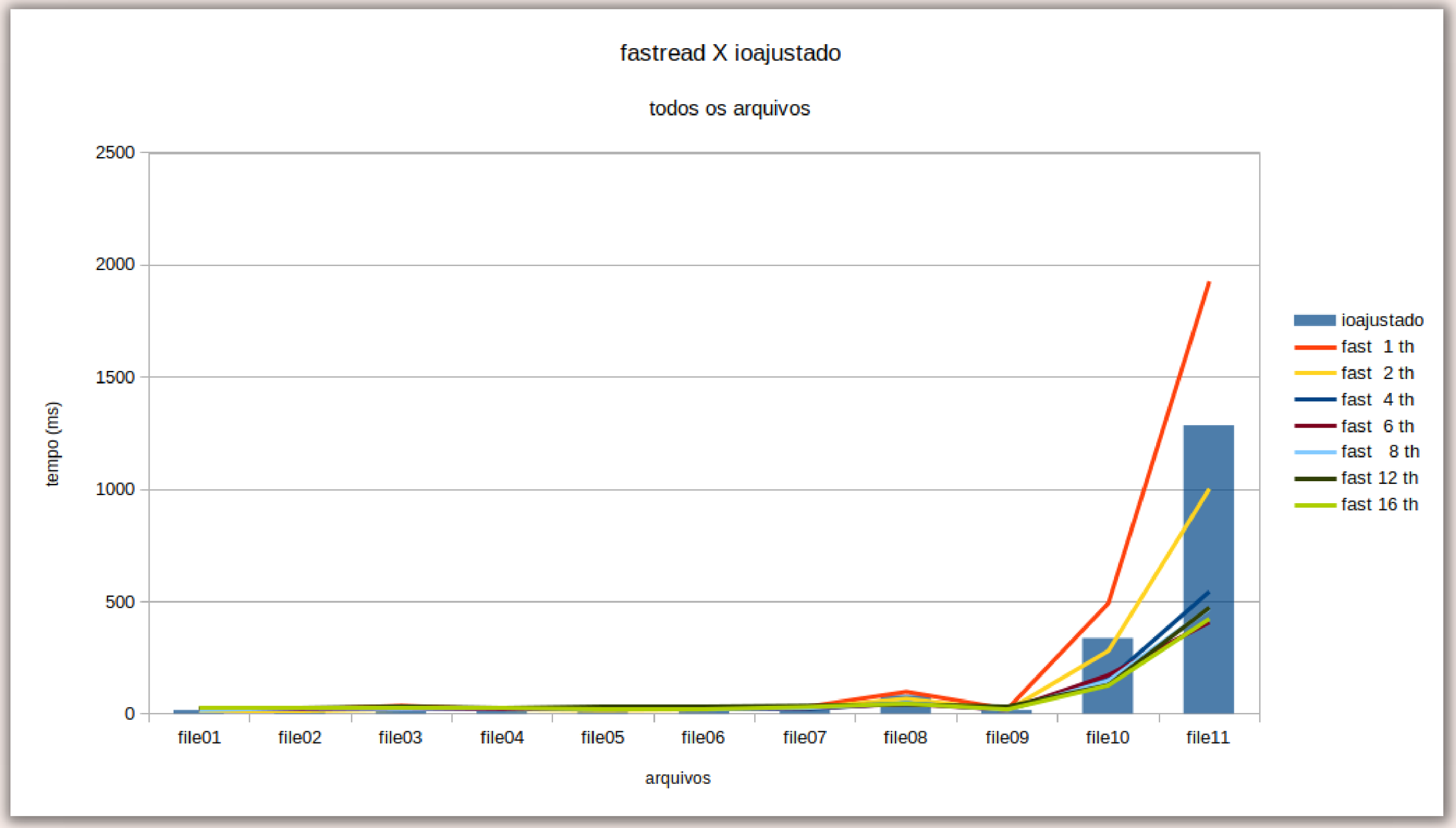
RESULTADOS


Principais soluções x Referência (ioajustado)
File11 - 200M linhas
tempo (s)
45,0
40,0
35,0
30,0
25,0
20,0
15,0
10,0
5,0
0,0
seq
1 th
2 th
4 th
6 th
8 th
12 th
16 th
número threads
ioajustado
ingenuo
sequencial
fastread
strtok
matriz
atomic
critical

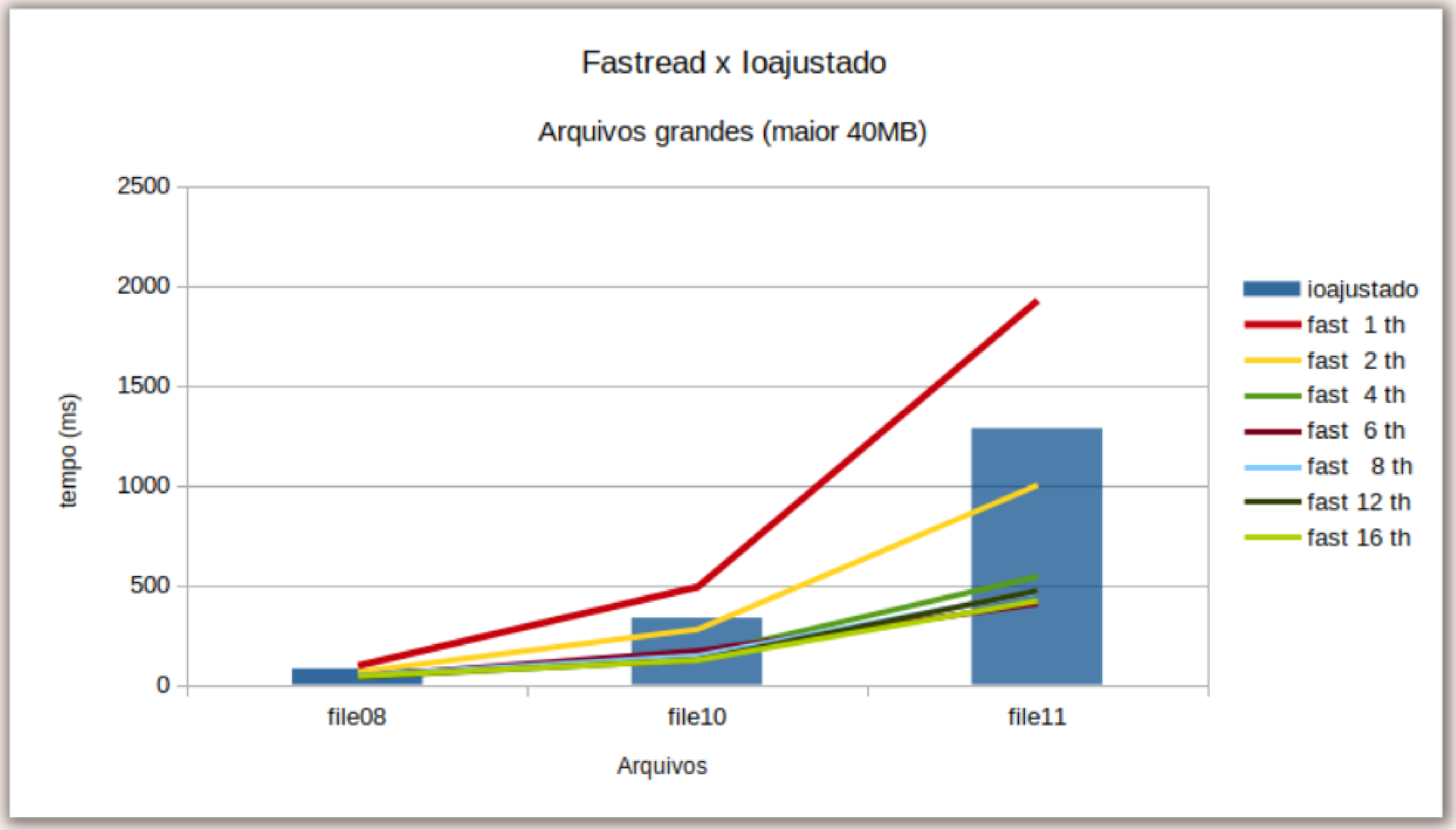
RESULTADOS


Fastread x Sequenciais
todos os arquivos
12000
10000
8000
6000
4000
2000
0
tempo (ms)
file01
file02
file03
file04
file05
file06
file07
file08
file09
file10
file11
arquivos
ioajustado
ingenuo
sequencial
fast 1 th
fast 2 th
fast 4 th
fast 6 th
fast 8 th
fast 12 th
fast 16 th

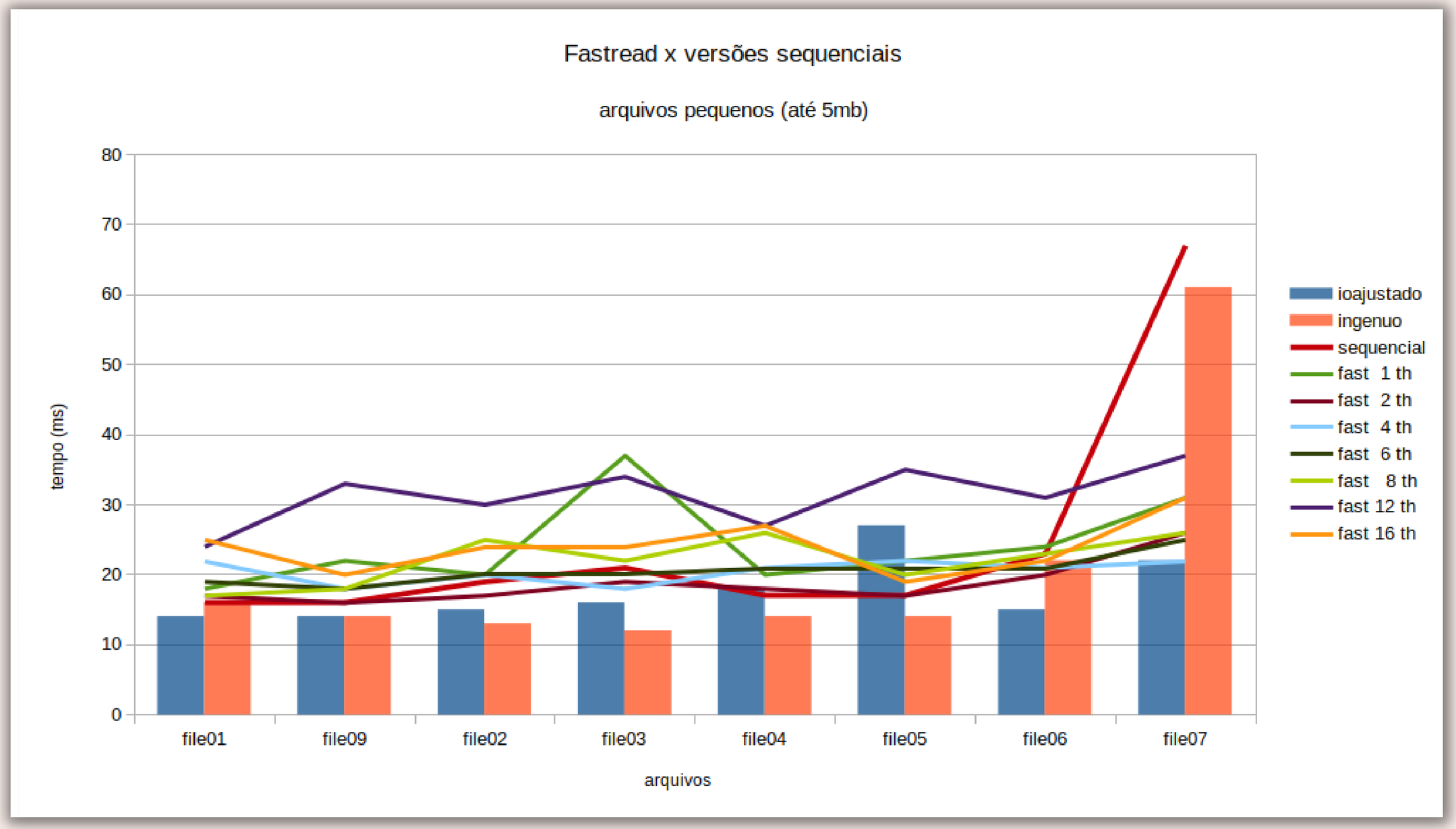
RESULTADOS


fastread X ioajustado
todos os arquivos
tempo (ms)
2500
2000
1500
1000
500
0
file01
file02
file03
file04
file05
file06
file07
file08
file09
file10
file11
arquivos
ioajustado
fast 1 th
fast 2 th
fast 4 th
fast 6 th
fast 8 th
fast 12 th
fast 16 th

RESULTADOS


Fastread x loajustado
Arquivos grandes (maior 40MB)
tempo (ms)
2500
2000
1500
1000
500
0
file08
file10
file11
Arquivos
ioajustado
fast 1 th
fast 2 th
fast 4 th
fast 6 th
fast 8 th
fast 12 th
fast 16 th

RESULTADOS


Fastread x versões sequenciais
arquivos pequenos (até 5mb)
tempo (ms)
0
10
20
30
40
50
60
70
80
file01
file09
file02
file03
file04
file05
file06
file07
arquivos
ioajustado
ingenuo
sequencial
fast 1 th
fast 2 th
fast 4 th
fast 6 th
fast 8 th
fast 12 th
fast 16 th