Tarefa 1: Processos e memória compartilhada

Eduardo Dantas Luna, 2111484

Ricardo Bastos Leta Vieira, 2110526

Código Fonte

ℹ️ Para gerar os arquivos executáveis de cada arquivo `.c` basta compilá-los normalmente: todas as funções auxiliares utilizadas estão implementadas em cada arquivo

Versão sequencial (sequencial.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <unistd.h>

typedef struct timeval Timer;

// bota valor de cada elemento do vetor de tamanho tam como a int valor
void preenche_array(int valor, int tam, int *arr)
{
  int i;
```

```
  if (arr == NULL)
  {
    exit(-1);
  }

  for (i = 0; i < tam; i++)
  {
    arr[i] = valor;
  }
}

//calcula a diferenca de tempo entre dois Timers
float timediff(Timer t0, Timer t1)
{
  return (t1.tv_sec - t0.tv_sec) * 1000.0f + (t1.tv_usec - t0.tv_usec) / 1000.0f;
}

int main(void)
{
  int tam = 0;
  int i = 0;
  Timer comeco, fim;
  int status;
  int *vetA, *vetB, *vetC;

  printf("Tamanho dos vetores: \n");
  scanf("%d", &tam);

  vetA = (int *) malloc(sizeof(int) * tam);
  vetB = (int *) malloc(sizeof(int) * tam);
  vetC =  (int *) malloc(sizeof(int) * tam);

  preenche_array(1, tam, vetA);
  preenche_array(2, tam, vetB);

  gettimeofday(&comeco, NULL); // incio

  for (i = 0; i < tam; i++)
  {
    vetC[i] = vetA[i] + vetB[i];
  }

  gettimeofday(&fim, NULL);

  printf("\nTempo : %f ms\n", timediff(comeco, fim)); // fim

  free(vetA);
  free(vetB);
  free(vetC);

  return 0;
}
```

Versão com processos paralelos (1 pai + 8 filhos) → paralelo.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
```

```c
#include <sys/wait.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <unistd.h>

typedef struct timeval Timer;

// bota valor de cada elemento do vetor de tamanho tam como a int valor
void preenche_array(int valor, int tam, int *arr)
{
  int i;

  if (arr == NULL)
  {
    exit(-1);
  }

  for (i = 0; i < tam; i++)
  {
    arr[i] = valor;
  }
}

//calcula a diferenca de tempo entre dois Timers
float timediff(Timer t0, Timer t1)
{
  return (t1.tv_sec - t0.tv_sec) * 1000.0f + (t1.tv_usec - t0.tv_usec) / 1000.0f;
}


int main(void)
{
  int tam = 0;
  int num_filhos = 8;
  int tam_p = tam / num_filhos;
  int i = 1;
  int j = 0;
  int status = 0;
  int pid, segmento;
  int *vetA, *vetB, *vetC;
  Timer comeco, fim;

  printf("Tamanho dos vetores: \n");
  scanf("%d", &tam);

  vetA = (int *)malloc(sizeof(int) * tam);
  vetB = (int *)malloc(sizeof(int) * tam);

  preenche_array(1, tam, vetA);
  preenche_array(2, tam, vetB);

  segmento = shmget(IPC_PRIVATE, sizeof(int) * tam, IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
  vetC = (int *)shmat(segmento, 0, 0);

  gettimeofday(&comeco, NULL); // incio

  for (i = 1; i <= num_filhos; i++)
  {
    pid = fork();

    if(pid == 0)
```

```
    {
      for (j = tam_p * (i - 1); j < i * tam_p; j++)
      {
        vetC[j] = vetA[j] + vetB[j];
      }

      exit(0);
    }
  }

  for (int k = 0; k < num_filhos; k++)
  {
    waitpid(-1, &status, 0);
  }

  gettimeofday(&fim, NULL);

  printf("\nTempo : %f ms\n", timediff(comeco, fim)); // fim

  free(vetA);
  free(vetB);

  shmdt(vetC);
  // libera a memória compartilhada
  shmctl(segmento, IPC_RMID, 0);

  return 0;
}
```

Gráfico com os tempos

> 💡 OBS:
>
> - Eixo x é o tamanho dos vetores usados
> - Eixo y é o tempo levado em milissegundos (ms)
> - Usamos vetores com 1000, 10 000, 100 000, 1 000 000 e 10 000 000 de elementos

Gráfico para versão sequencial

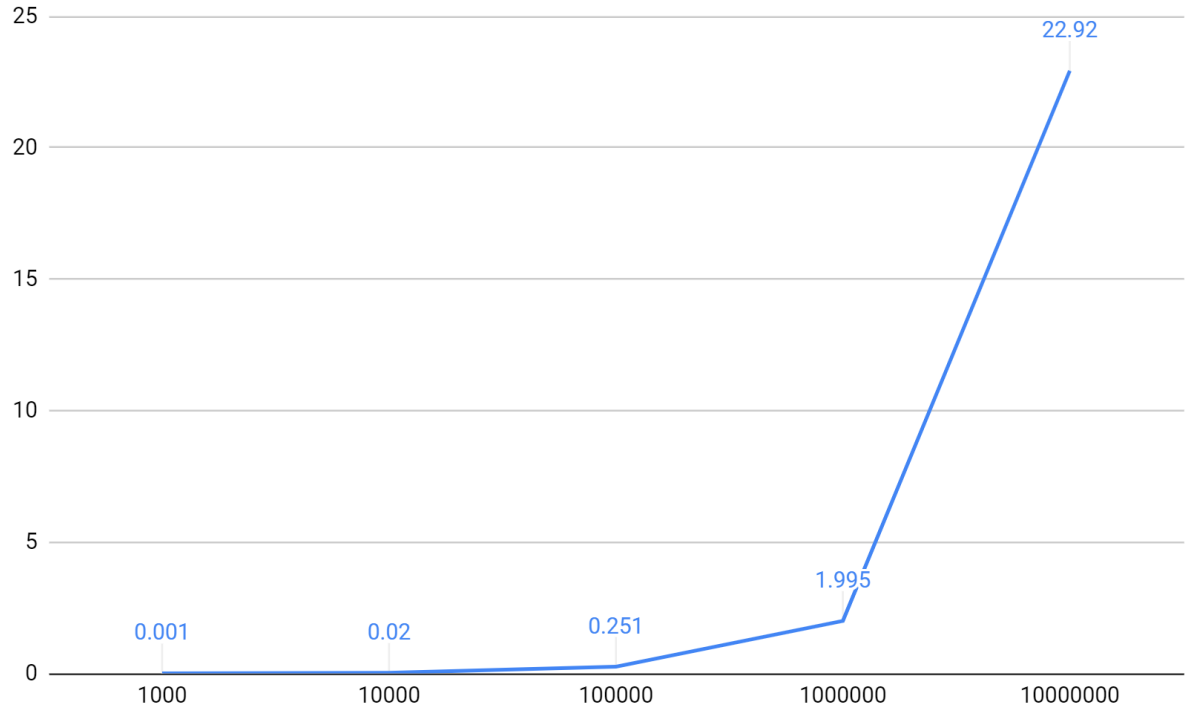
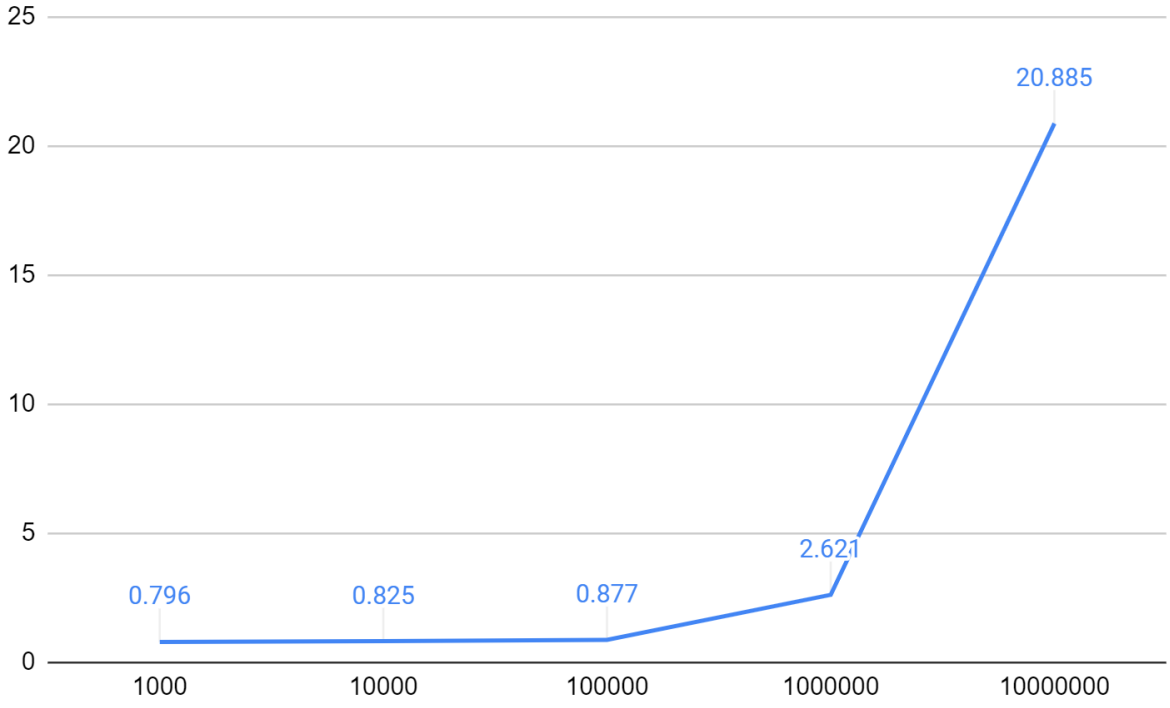


Gráfico para versão com processos paralelos



Análise dos resultados

- Ao contrário do previsto, a versão sequencial foi mais rápida que a com vários processos

- Isso provavelmente se dá porque leva tempo para criar cada um dos 8 processos, o que infla o tempo de execução da versão com vários processos

> ❓ E se incluirmos o mesmo número de processos na versão sequencial para tornar a medição mais justa?

Código fonte da versão sequencial justa (justo.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <unistd.h>

typedef struct timeval Timer;

// bota valor de cada elemento do vetor de tamanho tam como a int valor
void preenche_array(int valor, int tam, int *arr)
{
  int i;

  if (arr == NULL)
  {
    exit(-1);
  }

  for (i = 0; i < tam; i++)
  {
    arr[i] = valor;
  }
}

//calcula a diferenca de tempo entre dois Timers
float timediff(Timer t0, Timer t1)
{
  return (t1.tv_sec - t0.tv_sec) * 1000.0f + (t1.tv_usec - t0.tv_usec) / 1000.0f;
}

int main(void)
{
  int tam = 0;
  int i = 0;
  Timer comeco, fim;
  int status;
  int *vetA, *vetB, *vetC;

  printf("Tamanho dos vetores: \n");
  scanf("%d", &tam);

  vetA = (int *) malloc(sizeof(int) * tam);
  vetB = (int *) malloc(sizeof(int) * tam);
  vetC =  (int *) malloc(sizeof(int) * tam);
```

```
  preenche_array(1, tam, vetA);
  preenche_array(2, tam, vetB);

  gettimeofday(&comeco, NULL); // incio

  for (int j = 0; j < 8; j++)
  {
    if (fork() == 0)
    {
      waitpid(-1, &status, 0);
      exit(0);
    }
  }

  for (i = 0; i < tam; i++)
  {
    vetC[i] = vetA[i] + vetB[i];
  }

  gettimeofday(&fim, NULL);

  printf("\nTempo : %f ms\n", timediff(comeco, fim)); // fim

  free(vetA);
  free(vetB);
  free(vetC);

  return 0;
}
```

Comparação de tempos 2

Gráfico da versão sequencial justa

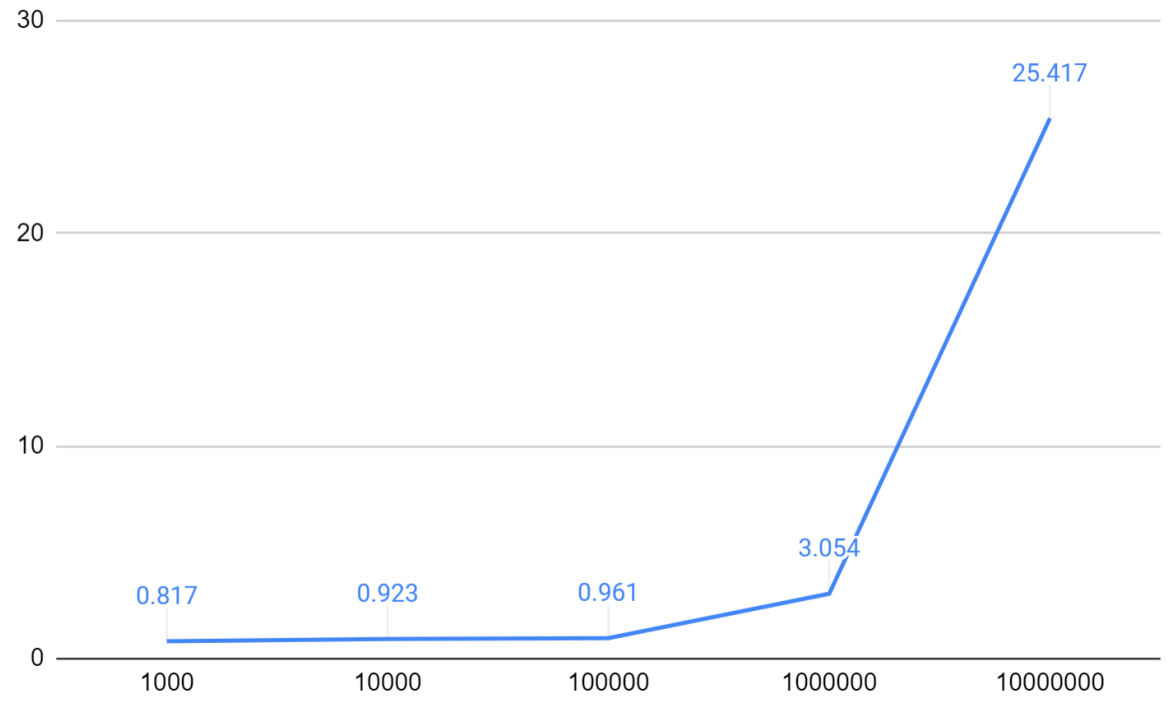
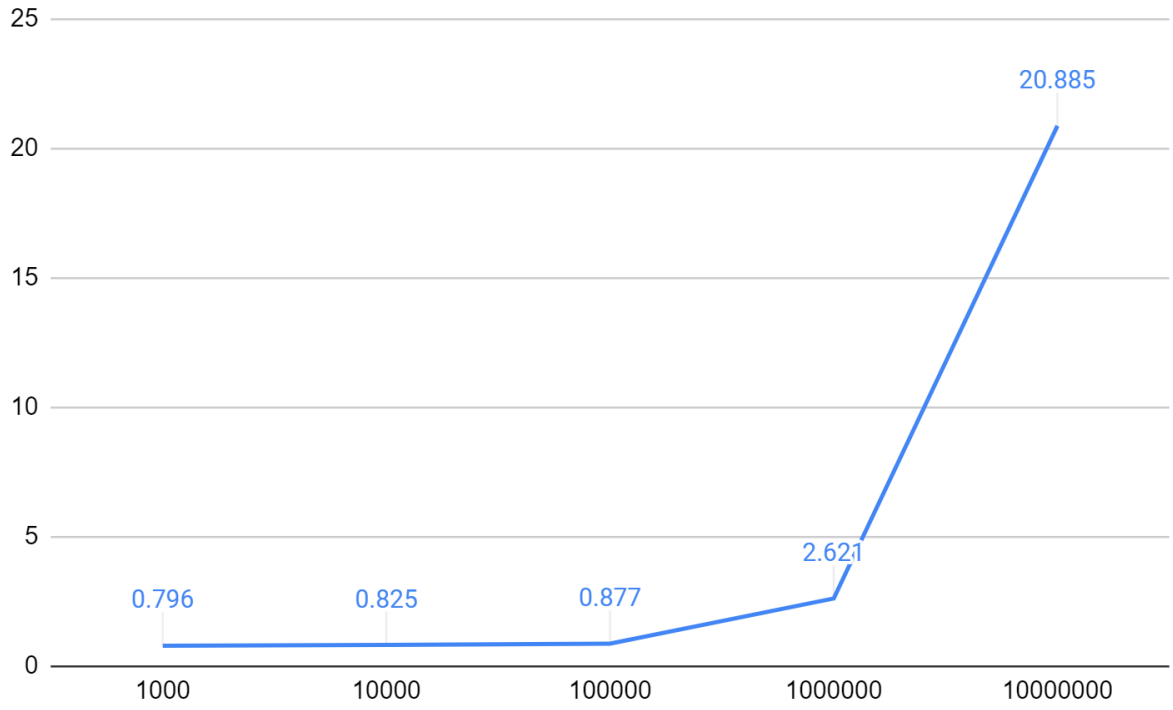


Gráfico da versão com vários processos (mesmo que o usado antes)



Análise dos resultados 2

- Como pode ser visto nos gráficos acima, quando se leva em conta o tempo de criação dos processos (no caso, ao criar processos que não fazem nada no

sequencial), a versão que usa vários processos é realmente mais rápida