10. Apêndice A – Códigos Fonte das Aplicações Utilizadas

Neste Apêndice encontram-se os códigos fonte das aplicações utilizadas nesta tese tanto para fazer as avaliações de desempenho com o PVM e com o AMIGO, quanto do programa de simulação construído para avaliar os índices de carga e desempenho.

As listagens apresentam respectivamente, os códigos de: integralpvm.c e escravo.c (método do trapézio composto), ordena.c e quick.c (quicksort paralelo), disco_rw.c e escravo.c (aplicação "sintética" para uso de disco), parmatrix.c e slavematrix.c (multiplicação paralela de matrizes) e o simula_indice.c (programa de simulação de índices de carga e de desempenho).

- **Aplicação Integral (Método do trapézio composto)**

Programa Mestre (integralpvm.c)

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include "time.h"
4   #include <math.h>
5   #include "/home/amigo/pvm3/include/pvm3.h"
6   #include<sys/time.h>
7   #include<unistd.h>
8
9   #define ESCRAVO "/home/amigo/pvm3/examples/integral/escravo"
10  #define NOME_ARQ_TEMPO "/home/amigo/pvm3/examples/integral/tempo_pvm_integral.txt"
11
12  #define N 120000000  /*quantidade de divisoes*/
13  //#define N 10
14  /* a quantidade de divisoes deve ser um multiplo de (NPROCS+1), pois,
15     serao NPROCS escravos mais o mestre. */
16  #define NPROCS 19    /*quantidade de escravos*/
17
18  double   s = 0,sum = 0;
19  struct timeval tempo1, tempo2;
20  struct timezone tzp;
21  double tempo;
22  double a,b;
23  double limite_inferior,limite_superior;
24  double x;
25  double resultado,res,valor_retorno;
26  int info, infos;
27
28  float integral(double a, double b, int n)
29  {
30     double h,var,r,s = 0, aux;
31     int i;              // numero de termos
32     h = (double)(b-a)/n;
33     s = s + sin(a) + exp(a);
34     s = s + sin(b) + exp(b);
35     aux = a;
36     for(i=1;i<n-1;i++){
```

```
37          var = aux + h;
38          r = 2 * (sin(var) + exp(var));
39          s = s + r;
40          aux = var;
41       }
42       return((h/2) * s);
43    }
44
45
46    main(int argc, char **argv)
47    {
48    int mytid;           /* Identificacao do processo mestre */
49    int tids[NPROCS];    /* Identificacao dos processos escravos */
50    int cc;              /* Numero de processos criados */
51    int tid;             /* Identificacao do processo escravo */
52    int i;               /* Contador */
53    FILE *out;           /* Arquivo de saida do tempo de processamento */
54    float timef;         /* Tempo para a execucao da integral */
55
56
57       mytid = pvm_mytid ();
58       printf("Processo mestre: %x\n", mytid);
59    //   for (i=0; i < NPROCS; i++) {
60          cc=pvm_spawn(ESCRAVO, (char**)0, PvmTaskDefault, NULL, NPROCS, tids);
61          //   printf("Passou o pvm_spawn... \n");
62          //   fflush(stdout);
63          if (cc < NPROCS){
64              printf("Nao consegui gerar algum escravo...\n");
65              exit(0);
66          }
67    //   }
68       printf("Iniciando calculos...\n");
69       gettimeofday(&tempo1,&tzp);
70       a = 0.0;
71       b = 1.2;
72
73       x = (double)(b - a)/(NPROCS+1); //numtasks;
74
75       limite_inferior = a;
76       limite_superior = a + x;
77       for(i=0;i<NPROCS;i++){
78          pvm_initsend(PvmDataDefault);
79          pvm_pkdouble(&limite_inferior,1,1);
80          pvm_pkdouble(&limite_superior,1,1);
81          pvm_send (tids[i],0);
82          limite_inferior = limite_superior;
83          limite_superior = limite_superior + x;
84       }
85
86       res = integral(limite_inferior,limite_superior,N/(NPROCS+1)); //numtasks);
87
88       printf("\nAguardando escravos...\n");
89       for(i=0;i<NPROCS;i++){
90          pvm_recv(-1, -1);  //tids[i],-1);
91          pvm_upkdouble(&valor_retorno, 1, 1);
92          sum = sum + valor_retorno;
93          printf("\nRecebeu: %d",i+1);
94       }
95
96       res = res + sum;
97
98       gettimeofday(&tempo2,&tzp);
99
100      tempo = (double)(tempo2.tv_sec - tempo1.tv_sec) + (((double)(tempo2.tv_usec-
101   tempo1.tv_usec))/1000000);
102
103      /* Colocando o resultado no arquivo*/
104
105          /* Abrindo arquivo timeresults para escrita...*/
106          out=fopen(NOME_ARQ_TEMPO,"a");
107          if(out==NULL) //se arquivo ainda nao existe
108                out=fopen(NOME_ARQ_TEMPO,"w"); //criando o arquivo
109                if (out==NULL)
110                      { //nao abriu o arquivo
111                      printf("Problemas para criar o arquivo");
112                      }
113                else
114                { // gravando o tempo no arquivo...
```

```
115
116                     fprintf(out, "%f\n",tempo);
117
118
119                 }//end if
120
121         fclose(out);
122
123     printf("Tempo em segundos = %f\n",tempo);
124     printf("Quantidade de divisoes = %d\n",N);
125     printf("------------------------------------------------\n");
126     fflush(stdout);
127     pvm_exit ();
128 }
129
```

Programa Escravo (escravo.c)

```
 1
 2  #include <stdio.h>
 3  #include "pvm3.h"
 4  #include <stdlib.h>
 5  #include "time.h"
 6  #include <math.h>
 7
 8  #define N 120000000  /*quantidade de divisoes*/
 9  //#define N 10
10  /* a quantidade de divisoes deve ser um multiplo de (NPROCS+1), pois,
11     serao NPROCS escravos mais o mestre. */
12  #define NPROCS 19   /*numero total de escravos*/
13
14  float integral(double a, double b, int n)
15  {
16     double h,var,r,s = 0, aux;
17     int i;               // numero de termos
18     h = (double)(b-a)/n;
19     s = s + sin(a) + exp(a);
20     s = s + sin(b) + exp(b);
21     aux = a;
22     for(i=1;i<n-1;i++){
23        var = aux + h;
24        r = 2 * (sin(var) + exp(var));
25        s = s + r;
26        aux = var;
27     }
28     return((h/2) * s);
29  }
30
31  main()
32  {
33     int mytid;             /* Identificacao do processo escravo */
34     int master;            /* Identificacao do processo mestre */
35     double a,b,res;
36
37     mytid = pvm_mytid ();
38     master = pvm_parent ();
39
40     pvm_recv(-1,-1);
41     pvm_upkdouble(&a,1,1);
42     pvm_upkdouble(&b,1,1);
43
44     res = integral(a,b,(N/ (NPROCS+1) ));
45
46  //   while ( 1 );
```

- **Aplicação Sintética (Leitura e escrita em disco)**

Programa Mestre (disco_rw.c)

```
#include <stdio.h>
#include <stdlib.h>
#include "time.h"
#include <math.h>
#include "/home/amigo/amigo/pvm3311/include/pvm3.h"
#include<sys/time.h>
#include<unistd.h>
#include "bench.h"

#define ESCRAVO "/home/amigo/amigo/examples/diskrw/escravo"
#define NOME_ARQ_TEMPO "/home/amigo/amigo/examples/diskrw/tempo_amigo_disk.txt"

/* a quantidade de divisoes deve ser um multiplo de (NPROCS+1), pois,
   serao NPROCS escravos mais o mestre. */
#define NPROCS 9    /*quantidade de escravos*/


double   s = 0,sum = 0;
struct timeval tempo1, tempo2;
struct timezone tzp;
double tempo;
double a,b;
double limite_inferior,limite_superior;
double x;
double resultado,res,valor_retorno;
int info, infos;
int mytid;

main(int argc, char **argv)
{
//int mytid;           /* Identificacao do processo mestre */
int tids[NPROCS];    /* Identificacao dos processos escravos */
int cc;              /* Numero de processos criados */
int tid;             /* Identificacao do processo escravo */
int i;               /* Contador */
FILE *out;           /* Arquivo de saida do tempo de processamento */
float timef;         /* Tempo para a execucao da integral */


   mytid = pvm_mytid ();
   printf("Processo mestre: %x\n", mytid);
//   for (i=0; i < NPROCS; i++) {
      cc=pvm_spawn(ESCRAVO, (char**)0, PvmTaskDefault, NULL, NPROCS, tids);
      //   printf("Passou o pvm_spawn... \n");
      //   fflush(stdout);
      if (cc < NPROCS){
          printf("Nao consegui gerar algum escravo...\n");
          exit(0);
      }
//   }

   system("rm /tmp/diskio*");
   printf("Iniciando Disk IO...\n");

   disk_read(TIMESTOTEST);

   gettimeofday(&tempo1,&tzp);
   a = 0.0;
   for(i=0;i<NPROCS;i++){
      pvm_initsend(PvmDataDefault);
      pvm_pkdouble(&a,1,1);
      pvm_send (tids[i],0);
   }
 printf("\nAguardando escravos...\n");
   for(i=0;i<NPROCS;i++){
      pvm_recv(-1, -1);  //tids[i],-1);
      pvm_upkdouble(&valor_retorno, 1, 1);
   }

   gettimeofday(&tempo2,&tzp);

```

```
72      tempo = (double)(tempo2.tv_sec - tempo1.tv_sec) + (((double)(tempo2.tv_usec-
73   tempo1.tv_usec))/1000000);
74
75      /* Colocando o resultado no arquivo*/
76
77          /* Abrindo arquivo timeresults para escrita...*/
78          out=fopen(NOME_ARQ_TEMPO,"a");
79          if(out==NULL) //se arquivo ainda nao existe
80                 out=fopen(NOME_ARQ_TEMPO,"w"); //criando o arquivo
81                 if (out==NULL)
82                        { //nao abriu o arquivo
83                        printf("Problemas para criar o arquivo");
84                        }
85                 else
86                 { // gravando o tempo no arquivo...
87
88                        fprintf(out, "%f\n",tempo);
89
90
91                 }//end if
92
93          fclose(out);
94
95      printf("Tempo em segundos = %f\n",tempo);
96      printf("-----------------------------------------------\n");
97      fflush(stdout);
98      pvm_exit ();
99   }
100
101
102  int disk_read(int times)
103  {
104     FILE      *fp, *fp_copia;
105     int              i, j;
106     char              buff[MAXLENGTHBUFFER], temp[10];
107     char         saida[50];
108     //char    *data;
109     //float   *vetresult;
110     //struct timeval bef, aft, diff;
111
112     //data = malloc(NBYTES);
113     //memset(data, 'X', NBYTES-1);
114     //data[NBYTES] = '\0';
115     //vetresult = malloc(times * sizeof(float));
116
117     printf("\nLendo no disco: data.tmp\n");
118     for (i=0; i < times; ++i)
119     {
120       if ((fp = fopen("/tmp/data.tmp","r")) == NULL)
121       {
122          fprintf(stderr, "Error reading file data.tmp");
123          return(-1);
124       }
125
126       sprintf(saida,"/tmp/diskio.%d",mytid);
127       if ((fp_copia = fopen(saida,"w")) == NULL) {
128         fprintf(stderr, "Error creating file data_temp.tmp");
129         return(-1);
130       }
131
132       fflush(stdout);
133       //gettimeofday(&bef, NULL);
134       j = 1;
135       while (fgets(buff,MAXLENGTHBUFFER,fp) != NULL) {
136         fsync(fileno(fp));
137         sscanf(buff,"%s",temp);
138         //printf("\nLinha %d: %s",j,temp);
139         fprintf(fp_copia, temp);
140         //fsync(fileno(fp_copia));
141         j = j + 1;
142       }
143
144
145       fflush(stdout);
146     }
147     fclose(fp);
148     fclose(fp_copia);
149
```

```
150      printf("Copiando arquivo diskio.mytid para diskio_mytid.tmp...\n");
151      sprintf(saida,"cp -f /tmp/diskio.%d /tmp/diskio_%d.tmp",mytid,mytid);
152      system(saida);
153      printf("Apagando arquivo data_temp.tmp\n");
154      sprintf(saida,"rm /tmp/diskio.%d",mytid);
155      system(saida);
156
157      //gettimeofday(&aft, NULL);
158      //timeval_subtract(&diff, &aft, &bef);
159      //vetresult[0] = timeval_to_float(&diff);
160      //printf("\nTempo Master: %f", vetresult[0]);
161      //write_results(DISKRESULTS,vetresult, times);
162      //printf("\nRodou %d vezes\n", times);
163   }
```

Programa Escravo(escravo.c)

```
  1   #include <stdio.h>
  2   #include "pvm3.h"
  3   #include <stdlib.h>
  4   #include <sys/time.h>
  5   #include <math.h>
  6   #include "bench.h"
  7
  8   /* a quantidade de divisoes deve ser um multiplo de (NPROCS+1), pois,
  9      serao NPROCS escravos mais o mestre. */
 10   #define NPROCS 9   /*numero total de escravos*/
 11
 12   int mytid;
 13
 14   main()
 15   {
 16      //int mytid;            /* Identificacao do processo escravo */
 17      int master;           /* Identificacao do processo mestre */
 18      double a,b,res;
 19
 20      mytid = pvm_mytid ();
 21      master = pvm_parent ();
 22
 23      pvm_recv(-1,-1);
 24      pvm_upkdouble(&a,1,1);
 25
 26      disk_read(TIMESTOTEST);
 27
 28   //   while ( 1 );
 29
 30      pvm_initsend(PvmDataDefault);
 31      pvm_pkdouble(&a,1,1);
 32      pvm_send(master,4);
 33
 34      pvm_exit ();
 35   }
 36
 37
 38   int disk_read(int times)
 39   {
 40      FILE      *fp, *fp_copia;
 41      int              i, j;
 42      char              buff[MAXLENGTHBUFFER], temp[10];
 43      char         saida[50];
 44      //char    *data;
 45      //float   *vetresult;
 46      //struct timeval bef, aft, diff;
 47
 48      //data = malloc(NBYTES);
 49      //memset(data, 'X', NBYTES-1);
 50      //data[NBYTES] = '\0';
 51      //vetresult = malloc(times * sizeof(float));
 52
 53      system("rm /tmp/diskio*");
 54      printf("\nLendo no disco: data.tmp\n");
 55      for (i=0; i < times; ++i)
 56      {
 57        if ((fp = fopen("/tmp/data.tmp","r")) == NULL)
 58        {
 59           fprintf(stderr, "Error reading file data.tmp");
```

```
60          return(-1);
61        }
62
63        sprintf(saida,"/tmp/diskio.%d",mytid);
64        if ((fp_copia = fopen(saida,"w")) == NULL) {
65          fprintf(stderr, "Error creating file diskio.mytid");
66          return(-1);
67        }
68
69        fflush(stdout);
70        //gettimeofday(&bef, NULL);
71        j = 1;
72        while (fgets(buff,MAXLENGTHBUFFER,fp) != NULL) {
73          fsync(fileno(fp));
74          sscanf(buff,"%s",temp);
75          //printf("\nLinha %d: %s",j,temp);
76          fprintf(fp_copia, temp);
77          //fsync(fileno(fp_copia));
78          j = j + 1;
79        }
80
81
82        fflush(stdout);
83      }
84      fclose(fp);
85      fclose(fp_copia);
86
87      printf("Copiando arquivo diskio.mytid para diskio_mytid.tmp...\n");
88      sprintf(saida,"cp -f /tmp/diskio.%d /tmp/diskio_%d.tmp",mytid,mytid);
89      system(saida);
90      printf("Apagando arquivo diskio.mytid\n");
91      sprintf(saida,"rm /tmp/diskio.%d",mytid);
92      system(saida);
93
94      //gettimeofday(&aft, NULL);
95      //timeval_subtract(&diff, &aft, &bef);
96      //vetresult[0] = timeval_to_float(&diff);
97      //printf("\nTempo: %f", vetresult[0]);
98      //write_results(DISKRESULTS,vetresult, times);
99      //printf("\nRodou %d vezes\n", times);
100
```

- **Aplicação Ordena (Quicksort paralelo)**

Programa Mestre(quick.c)

```
1   #include "/home/amigo/amigo/pvm3311/include/pvm3.h"
2   #define MaxVet 30600  // numero de elementos para ordenar
3   #define NUM_PROC 9     // numero de escravos (quick.c)
4
5   typedef int indice;
6
7   void particao (int *a, indice esq, indice dir, indice *i, indice *j)    {
8         int x, w;
9
10        *i = esq;
11        *j = dir;
12        x = a[(*i + *j)/2];
13
14        do
15        {
16              while (a[*i] < x)   (*i)++;
17              while (a[*j] > x)   (*j)--;
18              if ( *i <= *j)
19              {
20                    w = a[*i];
21                    a[*i] = a[*j];
22                    a[*j] = w;
23                    (*i)++;
24                    (*j)--;
25              }
26        } while (*i <= *j);
27  }
28
29
30  void ordena (int *a, indice esq, indice dir)  {
31
32        indice i, j;
33
34        particao (a, esq, dir, &i, &j);
35        if (esq < j)
36              ordena (a, esq, j);
37        if (i < dir)
38              ordena (a, i, dir);
39  }
40
41
42  int main(void)            {
43
44        int tid_parent;
45        int tot_subvet;
46        int *a = (int *)malloc( (MaxVet/NUM_PROC) * sizeof(int) );
47        int buf_rcv;
48
49        tid_parent = pvm_parent();
50
51        tot_subvet = (MaxVet/NUM_PROC);
52
53        buf_rcv = pvm_recv (tid_parent, 1);
54        pvm_upkint (a, tot_subvet, 1);
55        pvm_freebuf(buf_rcv);
56
57        ordena (a, 0, (tot_subvet - 1));
58
59        pvm_initsend(PvmDataRaw);
60        pvm_pkint(a, tot_subvet, 1);
61        pvm_send(tid_parent, 2);
62
63        pvm_exit();
64        free(a);
65
66        return(0);
67  }
```

Programa Escravo(ordena.c)

```
1   #include <stdio.h>
2   #include <limits.h>
3   #include <sys/time.h>
4   #include "/home/amigo/amigo/pvm3311/include/pvm3.h"
5
6   #define TRUE 1
7   #define FALSE 0
8   #define MaxVet 30600  // numero de elementos para ordenar 30600
9   #define NUM_PROC 9     // numero de escravos (quick.c)
10
11  // aquivo onde serao gravados os tempos coletados
12  #define ARQ_TEMPO "/home/amigo/amigo/examples/Quicksort/quick_amigo.txt"
13
14  // arquivos escravos que irao ordenar partes do vetor de elementos
15  #define ESCRAVO "/home/amigo/amigo/examples/Quicksort/quick"
16
17
18  void grava(double dif_tot, double dif_spawn, double dif_sr, double dif_seq)    {
19
20        FILE *fd;
21
22        fd = fopen(ARQ_TEMPO, "a+");
23
24        if(fd == NULL)            {
25              fprintf(stderr, "Cannot open input file.\n");
26              return;
27        }
28
29        fprintf(fd, "tot:%f spawn:%f send/rcv:%f seq:%f \n", dif_tot, dif_spawn,
30  dif_sr, dif_seq);
31
32        fflush(fd);
33        fclose(fd);
34  }
35
36
37  int main()   {
38
39        int i, t, ind;
40        int tid_sort[NUM_PROC], indices[NUM_PROC];
41        int tot_subvet, pos_min;
42        int *vetor = (int *)malloc( MaxVet * sizeof(int) );
43        int *vet_ordenado = (int *) malloc( MaxVet * sizeof(int) );
44        unsigned long int val_min;
45        int block = 1, cc, my_tid;
46
47         struct timeval tmp1, tmp2, tmp_mandar, tmp_receber;
48         struct timezone tzp;
49
50         double dif_tot, dif_spawn, dif_sr, dif_seq;
51
52         my_tid = pvm_mytid();
53         printf("Ordena tid=[%x]\n", my_tid);
54
55        tot_subvet = MaxVet/NUM_PROC;
56
57        for (t = 0; t < MaxVet; t++) {  // gera vetor
58              vetor[t] = rand() % 30000;
59        }
60  //    for (t = (MaxVet-10) ; t < MaxVet; t++)   // mostra elementos
61  //          printf ("vetor gerado [%d] = %d \n", t, vetor[t]);
62
63        // determinar tempo inicial da execucao
64         gettimeofday(&tmp1,&tzp);
65
66         cc = pvm_spawn(ESCRAVO, (char **)0, PvmTaskDefault, NULL, NUM_PROC,
67  tid_sort);
68        if( cc < NUM_PROC) {
69              printf ("ERRO:pvm_spawn nao gerou todas as tasks[gerou %d]...\n", cc);
70  //              printf ("ESCRAVO:|%s| \n", ESCRAVO);
71  //              printf ("PvmTaskDefault:|%d| \n", PvmTaskDefault);
72  //              printf ("NUM_PROC:|%d| \n", NUM_PROC);
73  //              printf ("Erro retornado t1:|%d| t2:|%d| \n", tid_sort[0],
74  tid_sort[1]);
75                exit (-1);
76        }
```

```
77
78          // determinar tempo para enviar subvetores
79            gettimeofday(&tmp_mandar,&tzp);
80
81          // envia subvetores para os processos
82          for (t = 0; t < NUM_PROC; t++)
83          {
84                pvm_initsend (PvmDataRaw);
85                pvm_pkint (&vetor[(tot_subvet * t)], tot_subvet, 1);
86                pvm_send (tid_sort[t], 1);
87          }
88    //        printf("Enviou mensagens \n");
89
90          // recebe os subvetores ordenados
91          for (t = 0; t < NUM_PROC; t++)
92          {
93                pvm_recv (-1, 2);
94                pvm_upkint (&vetor[(tot_subvet * t)], tot_subvet, 1);
95                indices[t] = (tot_subvet * t);
96          }
97    //        printf("Recebeu mensagens \n");
98
99          // determinar tempo do recebimento dos subvetores
100           gettimeofday(&tmp_receber,&tzp);
101
102         // merge - busca o menor elemento dos subvetores
103         for (t = 0; t < MaxVet; t++)
104         {
105               val_min = ULONG_MAX;
106               pos_min = -1;
107               for (ind = 0; ind < NUM_PROC; ind++)
108               {
109                     if
110   ((indices[ind]<(tot_subvet*(ind+1)))&&(vetor[indices[ind]]<=val_min))
111                     {
112                           val_min = vetor[indices[ind]];
113                           pos_min = ind;
114                     }
115               }
116               if (pos_min > -1)    {
117                     vet_ordenado[t] = val_min;
118                     (indices[pos_min])++;
119               }
120               else
121                     printf ("Nao conseguiu achar o menor...\n");
122         }
123           // determina o tempo final da execucao
124           gettimeofday(&tmp2,&tzp);
125
126           dif_tot = (double)(tmp2.tv_sec - tmp1.tv_sec) + (((double)(tmp2.tv_usec-
127   tmp1.tv_usec))/1000000);
128         printf("Tempo total: %f\n", dif_tot);
129
130           dif_spawn = (double)(tmp_mandar.tv_sec - tmp1.tv_sec) +
131   (((double)(tmp_mandar.tv_usec-tmp1.tv_usec))/1000000);
132   //      printf("Tempo p/ spawn: %f \n", dif_spawn);
133
134           dif_sr = (double)(tmp_receber.tv_sec - tmp_mandar.tv_sec) +
135   (((double)(tmp_receber.tv_usec-tmp_mandar.tv_usec))/1000000);
136   //      printf("Tempo entre send e recv: %f\n", dif_sr);
137
138           dif_seq = (double)(tmp2.tv_sec - tmp_receber.tv_sec) +
139   (((double)(tmp2.tv_usec-tmp_receber.tv_usec))/1000000);
140   //      printf("Tempo parte seq.: %f\n", dif_seq);
141
142           grava(dif_tot, dif_spawn, dif_sr, dif_seq);
143
144         // mostra os elementos ordenados
145         for (t = (MaxVet-1); t < MaxVet; t++)
146               printf ("vet_ordenado [%d] = %d \n", t, vet_ordenado[t]);
147
148         free(vetor);
149         free(vet_ordenado);
150
151         pvm_exit();
152   //      printf("Ordena saindo com pvm_exit ! \n");
153   //      getchar();
154
```

```
155         return(0);
156     }
```

- **Aplicação Parmatrix (Multiplicação paralela de matrizes)**

Programa Mestre(parmatrix.c)

```
  1     // defines and prototypes for the PVM library
  2     #include <stdio.h>
  3     //#include <conio.h>
  4     #include <stdlib.h>
  5     #include <time.h>
  6     #include "/home/amigo/amigo/pvm3311/include/pvm3.h"
  7
  8     #define SLAVE "/home/amigo/amigo/examples/matrix/parmatrix/slavematrix"
  9
 10     /*     prototipos
 11      *******************************************************/
 12
 13     void InitBlock(long double *, long double *, long double *, int, int, int, int, int,
 14     int);
 15     int  main(int, char **);
 16
 17     /*******************************************************/
 18
 19     //
 20     //     Multiplicação Paralela de Matrizes
 21     //
 22
 23     #include "parmatrix.h"
 24
 25     void InitBlock(long double *a, long double *b, long double *c, int rA, int cA, int
 26     rB, int cB, int rC, int cC ) {
 27           int len, ind, i, j;
 28           time_t t;
 29
 30        srand((unsigned) time(&t));
 31
 32           len = rA * cA;
 33           for(ind = 0; ind < len; ind++)   {
 34                 a[ind] = (long double)0;
 35           }
 36
 37           len = rB * cB;
 38           for(ind = 0; ind < len; ind++)   {
 39                 b[ind] = (long double)ind;
 40     //          b[ind] = (long double)(rand()%10000)/7.0;
 41           }
 42
 43           len = rC * cC;
 44           for(ind = 0; ind < len; ind++)   {
 45                 c[ind] = (long double)2;
 46     //          c[ind] = (long double)(rand()%10000)/7.0;
 47           }
 48     /*
 49        printf("Mestre => rA=%d, cA=%d \n",rA, cA);
 50        for( i=0; i < rA; i++) {
 51           for( j=0; j < cA; j++)     {
 52           printf("A[%d,%d]=%Lf   ", i, j, a[i * cA + j]);
 53           }
 54           printf("\n");
 55        }
 56
 57        printf("Mestre => rB=%d, cB=%d \n",rB, cB);
 58        for( i=0; i < rB; i++) {
 59           for( j=0; j < cB; j++)     {
 60           printf("B[%d,%d]=%Lf   ", i, j, b[i * cB + j]);
 61           }
 62           printf("\n");
 63        }
 64
 65        printf("Mestre => rC=%d, cC=%d \n",rC, cC);
 66        for( i=0; i < rC; i++) {
 67           for( j=0; j < cC; j++)     {
 68           printf("C[%d,%d]=%Lf   ", i, j, c[i * cC + j]);
 69           }
```

```
70          printf("\n");
71       }
72
73       printf("Mestre => Inicialização de matrizes efetuada!!!!");
74    */
75    } // end of InitBlock
76
77    int main(int argc, char *argv[]) {
78          int i, j, result;
79          long double *a, *b, *c; // a = b * c
80       long double *line;            // linhe enviada para escravos
81          int rA, cA, rB, cB, rC, cC; // linhas e colunas
82       int amount_rows, num_tasks;
83       int first_task; // numero de linhas para primeira tarefa
84       int lenB, lenC;
85       int send_tag = 0, recv_tag = 1;
86       int size_data_type;
87       int *tid_slaves, slave_number;
88       struct timeval tempo1,tempo2, tempo3, tempo4;
89       double tempo_res1, tempo_res2;
90
91          if(argc > 1 && argc > 4)  {
92                rA = rB = atoi(argv[1]);
93                cB = atoi(argv[2]);
94                rC = atoi(argv[3]);
95                cA = cC = atoi(argv[4]);
96          amount_rows = atoi(argv[5]);
97          }
98          else {
99                fprintf(stderr, "for A = B * C, usage: rowB colB rowC colC
100   amount_of_rows_/_process \n");
101               return(-1);
102         }
103
104      lenB = rB * cB;
105      lenC = rC * cC;
106         size_data_type = sizeof(long double);
107
108         if(cB != rC || !lenB || !lenC || amount_rows <= 0)  {
109               fprintf(stderr, "Parametros invalidos. Saindo... \n");
110               return(-1);
111         }
112
113         // initial time
114         gettimeofday(&tempo1, NULL);
115
116         // aloca a memoria
117         a = (long double *)malloc(size_data_type * rA * cA);
118         b = (long double *)malloc(size_data_type * lenB);
119         c = (long double *)malloc(size_data_type * lenC);
120
121         // checa ponteiros validos
122         if (!(a && b && c) )      {
123               fprintf(stderr, "%s:out of memory ! \n", argv[0]);
124               free(a);
125               free(b);
126               free(c);
127               return(-1);
128         }
129
130      InitBlock(a, b, c, rA, cA, rB, cB, rC, cC);
131
132         num_tasks = (int)(rB / amount_rows);
133         if((num_tasks * amount_rows) < rB)
134         first_task = amount_rows + (rB - (num_tasks * amount_rows));
135      else
136         first_task = amount_rows;
137
138      tid_slaves = (int *) malloc(sizeof(int) * num_tasks);
139
140      (void) pvm_mytid();
141      pvm_setopt(PvmRoute, PvmRouteDirect);
142
143
144      gettimeofday(&tempo2,NULL);
145         // cria processos
146           if ((result = pvm_spawn(SLAVE, NULL, PvmTaskDefault, NULL, num_tasks,
147   tid_slaves)) < num_tasks) {
```

```
148               printf("Não criou todos os escravos.\n");
149         for(i=0;i < num_tasks; i++) {
150               printf("tid_slaves[%d]=%d (TID or error code) \n", i, tid_slaves[i]);
151         }
152         printf("num_tasks=%d, created=%d \n", num_tasks, result);
153         printf("SLAVE=%s \n", SLAVE);
154
155         return(-1);
156      }
157
158      // tempo para enviar e receber
159      gettimeofday(&tempo3,NULL);
160
161      // envia linhas da matriz b para o primeiro escravo
162      i = 0;
163      pvm_initsend( PvmDataRaw );          // cria buffer
164      pvm_pkint(&i, 1, 1);                 // numero escravo (0 == primeiro escravo)
165          pvm_pkint(&first_task, 1, 1);    // numero de linhas para primeiro escravo
166      pvm_pkint(&cB,1,1);
167          pvm_pkint(&rC,1,1);
168          pvm_pkint(&cC,1,1);
169          pvm_pkbyte((char *)b, (first_task*cB*size_data_type), 1 ); // empacota linhas
170   para envio
171      pvm_pkbyte((char *)c, (lenC*size_data_type),1); // matriz c
172
173          pvm_send(tid_slaves[0], send_tag );                        // envia linhas
174   para primeiro escravo
175
176          line = b + first_task*cB;
177
178   /*   printf("Tecle... line-b=%d\n", (line-b));
179      printf("b[1,0]=%Lf, (*line)=%Lf \n", b[1 * cB + 0], (*line));
180      printf("&b[1,0]=%x, line=%x \n", &b[1 * cB + 0], line);
181   */
182   //   getchar();
183
184      for(i = 1; i < num_tasks; i++)      {  // i = 1 => nao envia para primeiro
185   escravo
186                pvm_initsend( PvmDataRaw );            // cria buffer
187          pvm_pkint(&i,1,1);                 // slave numberle
188          pvm_pkint(&amount_rows, 1, 1);     // numero de linhas para escravo
189              pvm_pkint(&cB,1,1);
190                pvm_pkint(&rC,1,1);
191                pvm_pkint(&cC,1,1);
192           pvm_pkbyte((char *)line, (amount_rows*cB*size_data_type), 1 );    // empacota
193   linhas para envio
194          pvm_pkbyte((char *)c, (lenC*size_data_type),1); // matriz c
195
196                pvm_send(tid_slaves[i], send_tag );        // senvia linhas par
197   primeiro escravo
198
199                line += amount_rows*cB;
200      }
201
202      // recebe resultados matriz (a) dos escravos
203      for(i = 0; i < num_tasks; i++)      {  // i = 1 => nao envia para primeiro
204   escravo
205                pvm_recv( -1, recv_tag );              // recebe mensagem
206                pvm_upkint(&slave_number, 1, 1); // tidpack numero de linhas para
207   escravos
208                pvm_upkint(&amount_rows, 1, 1);  // numero de linhas para escravos
209                pvm_upkbyte((char *)(a + ((amount_rows*cA)*slave_number) ),
210   ((amount_rows*cA)*size_data_type), 1 );// empacota linhas para envio
211      }
212
213   /*   printf("\nMASTER => Resultados\n");
214      for( i=0/*rA-1; i < rA; i++) {
215          for( j=0; j < cA; j++)     {
216          printf("A[%d,%d]=%Lf   ", i, j, a[i * cA + j]);
217         }
218         printf("\n");
219      }*/
220
221      // ending time
222      gettimeofday(&tempo4,NULL);
223
224           tempo_res1 = (double)(tempo2.tv_sec - tempo1.tv_sec) +
225   (((double)(tempo2.tv_usec-tempo1.tv_usec))/1000000);
```

```
226         tempo_res2 = (double)(tempo4.tv_sec - tempo3.tv_sec) +
227   (((double)(tempo4.tv_usec-tempo3.tv_usec))/1000000);
228         tempo_res1 += tempo_res2;
229
230          printf("MASTER => Done! \n");
231
232          printf("\nTempo total(s): %.3f\n", tempo_res1);
233
234         free(a);
235         free(b);
236         free(c);
237
238         return(0);
239
240   } // end of main()
```

Programa Escravo (slavematrix.c)

```
  1   // defines and prototypes for the PVM library
  2   #include <stdio.h>
  3   #include <stdlib.h>
  4   #include <time.h>
  5
  6   #include "../../../pvm3311/include/pvm3.h"
  7
  8   ******************************************************/
  9
 10   void InitBlock(long double *, long double *, long double *, int, int, int, int, int,
 11   int);
 12   int  main(void);
 13
 14   /******************************************************/
 15
 16   //
 17   //     Multiplicação paralela de matrizes  - Processo Escravo
 18   //
 19
 20   #include "slavematrix.h"
 21
 22   int main(void)      {
 23         int i, j, k;
 24         long double *a, *b, *c; // a = b * c
 25         long double *temp;
 26         int rA, cA, rB, cB, rC, cC; // linhas e colunas
 27      int amount_rows;
 28      int lenB, lenC;
 29      int send_tag = 0, recv_tag = 1;
 30      int size_data_type;
 31      int parent_tid, slave_number;
 32
 33      size_data_type = sizeof(long double);
 34
 35      parent_tid = pvm_parent();
 36
 37   //     printf("************************************\n");
 38         pvm_recv(parent_tid, send_tag);
 39
 40      pvm_upkint(&slave_number,1,1);             // numero escravo
 41          pvm_upkint(&amount_rows, 1, 1);        // numero de linhas para escravos
 42      pvm_upkint(&cB,1,1);
 43          pvm_upkint(&rC,1,1);
 44          pvm_upkint(&cC,1,1);
 45
 46   //     printf("===> SLAVE[%d] amount_rows=%d, cB=%d, rC=%d, cC=%d  \n",
 47   slave_number, amount_rows, cB, rC, cC);
 48
 49      rA = rB = amount_rows;
 50      cA = cC;
 51
 52      lenB = rB * cB;
 53      lenC = rC * cC;
 54
 55          // aloca a memoria
```

```
56          a = (long double *)malloc(size_data_type * rA * cA);
57          b = (long double *)malloc(size_data_type * lenB);
58          c = (long double *)malloc(size_data_type * lenC);
59
60          // checa ponteiros validos
61          if (!(a && b && c) )      {
62                fprintf(stderr, "SLAVE[%d]: out of memory ! \n", slave_number);
63                free(a);
64                free(b);
65                free(c);
66                   pvm_exit();
67                return(-1);
68          }
69
70          pvm_upkbyte((char *)b, (amount_rows*cB*size_data_type), 1 );      // linhas de
71    b
72       pvm_upkbyte((char *)c, (lenC*size_data_type),1); // matriz c
73    /*
74       printf("SLAVE[%d]==> rA=%d, cA=%d \n", slave_number, rA, cA);
75       for( i=0; i < rA; i++) {
76           for( j=0; j < cA; j++)    {
77           printf("A[%d,%d]=%Lf   ", i, j, a[i * cA + j]);
78           }
79           printf("\n");
80       }
81       printf("SLAVE[%d]=> rB=%d, cB=%d \n", slave_number, rB, cB);
82       for( i=0; i < rB; i++) {
83           for( j=0; j < cB; j++)    {
84           printf("B[%d,%d]=%Lf   ", i, j, b[i * cB + j]);
85           }
86           printf("\n");
87       }
88
89       printf("SLAVE[%d]=> rC=%d, cC=%d \n", slave_number, rC, cC);
90       for( i=0; i < rC; i++) {
91           for( j=0; j < cC; j++)    {
92           printf("C[%d,%d]=%Lf   ", i, j, c[i * cC + j]);
93           }
94           printf("\n");
95       }
96       printf("SLAVE[%d]=> Recebido!!!! \n", slave_number);
97    */
98          // efetuando multiplicacao
99        for( i=0; i < rB; i++) {
100          for( j=0; j < cC; j++) {
101             a[i * cA + j] = 0;
102             for ( k=0; k < cB; k++)  {
103   //                printf(" A=> i * cA + j= %d \n ", (i * cA + j));
104   //                printf(" B=> i * cB + k=%d \n C=> k * cC + j=%d \n", (i * cB + k),
105   (k * cC + j) );
106                a[i * cA + j] += (long double) (b[i * cB + k] * c[k * cC + j]);
107             }
108   //          printf(" \n ");
109          }
110       }
111
112          temp = a;
113       // envia resultado matriz (a) para mestre
114          pvm_initsend( PvmDataRaw );            // cria buffer
115            pvm_pkint(&slave_number,1,1);        // numero escravo
116          pvm_pkint(&amount_rows, 1, 1);   // numero de linhas deste escravo
117          pvm_pkbyte((char *)temp, ((amount_rows*cA)*size_data_type), 1 );  // empacota
118   resultado matriz
119          pvm_send(parent_tid, recv_tag ); // envia mensagem usando recv_tag
120   /*
121      printf("\nSLAVE[%d]=> Result\n", slave_number);
122      for( i=0; i < rA; i++) {
123          for( j=0; j < cA; j++)    {
124          printf("A[%d,%d]=%Lf   ", i, j, a[i * cA + j]);
125          }
126          printf("\n");
127      }
128
129
130          printf("SLAVE[%d]=> Feito! \n", slave_number);
131   */
132          free(a);
133          free(b);
```

```
134         free(c);
135
136         pvm_exit();
137
138         return(0);
139
140     } // end of main()
```

- **Programa de Simulação de Índices de Carga e Desempenho (simula_indice.c)**

```
/* ------------------------------------------------------------------- */
// Programa de Simulação de Índices de Carga e Desempenho
/* ------------------------------------------------------------------- */
#include "smplx.h"
#include "randpar.h"
#define NUM_TASK 5000
#define NUM_MAQ 10
/* ESTRUTURAS E VARIÁVEIS GLOBAIS */
enum tipo_indice {cpu_livre,     // CPU
                  mem_livre,           // Memoria
                  num_escrita_leitura, // Disco
                  pacotes_in_out,      // Rede
                  VIP,                 // Ind. Desemp.
                  PVIP,                //variacao indice desempenho
                  PVM};                // PVM
int indice_id; // escolha do índice de carga a ser avaliado na simulação
char nomearqsaida[30];
 char *recursos;
struct TipoAplicacao_t{
        float proc,
            disco,
            rede;
         float Qmemoria;  //quantidade que a aplicação necessita de memória
 } Tipo_aplicacao;
int NumSemente;
float TAppProc;
float TAppDisco;
float TAppRede;
float TAppMemoria;
struct Aplicacao_t{
        float TempoProcessamento;  // acumula o tempo de Processador que a aplicacao
já utilizou
        float acabou;           // 1 - já acabou                  0 - não acabou
        float AcessouDisco;     // 1 - já passou pelo disco       0 - não passou
        float AcabouSwap;     // 1 - já acabou swap         0 - caso contrário
        float AcessouRede;      // 1 - já passou pela rede        0 - não passou
        float PrimeiraVezProc;  // 1 - primeira vez no processador 0 - já passou
pelo menos uma vez pelo processador
        float TempoServicoProc;    // tempo de serviço do processador
        float tempo_disco;      // auxiliar para quando utilizar disco
        float tempo_rede;      // auxiliar para quando utilizar rede
        int TirouMemoria;      // se a aplicacao tira memoria == 1, cc, == 0
        int PrimeiraVezProc_com_disco_rede;
 } Aplicacao[NUM_TASK+1];
struct Aplicacao_t *Cliente;
float quantum[NUM_MAQ+1];
int melhor;          // melhor maquina com relação à carga existente
float tabela[NUM_MAQ+1];  // tabela de hosts
float bench_cpu[NUM_MAQ+1],          // vetores para cálculo do benchmark
      bench_disco[NUM_MAQ+1],
      bench_memoria[NUM_MAQ+1],
      bench_rede[NUM_MAQ+1];
float cpu[NUM_MAQ+1],     // recursos a serem avaliados
      disco[NUM_MAQ+1], rede[NUM_MAQ+1];
float cpu_norm[NUM_MAQ+1],     // recursos a serem avaliados
      disco_norm[NUM_MAQ+1], rede_norm[NUM_MAQ+1];
float memoria_livre[NUM_MAQ+1], // quantidade de memória livre
      memoria_norm[NUM_MAQ+1], // quantidade de memoria normalizada
      memoria_total[NUM_MAQ+1];  // quantidade de memória total incial das máquinas
float tempo_atualizacao;  // tempo de atualização da tabela, peridiocidade
float servidor_escolhido; //retorna servidor do qual será obtida a utilização
real  Tsp[NUM_MAQ+1],
             Tsd[NUM_MAQ+1],  // tempos de servicos de cada recurso
             Tsr[NUM_MAQ+1];
int Event = 1, Atualiza_tabela = 100, Aleatorio, i, m_i;
 int   flag_atualizou_tabela = 0;   // verifica se a tabela foi atualizada
                                        // 0 - não foi, 1 - foi
 int num_tours=NUM_TASK;
 int num_task;   // contador do numero de tarefas no sistema
 long aux;       // armazena o tempo da Poisson
 int ultimo_indice = 1;   // para escalonamento do PVM
struct Aplic_Maq_t{
```

```
73          int Maquina;
74          float Tempo_restante;
75          } Aplic_Maq[NUM_TASK +1];
76    real tmp_aux = 0;
77    real Tat = 0.01,      // tempo gasto para efetuar cálculo do índice
78          Tat2 = 0.01;     // tempo de gasto para atualização do vetor e da matriz
79    (indice desempenho)
80    real Ta1 = 0.1;         // media da distribuicao de chegada
81    real Ts1 = 0.02;    // tempo do escalonador
82    int maquina = 1;
83    FILE *p, *saida;
84    void Imprime_aplicacao()
85    {
86          for (i = 1; i <= NUM_TASK; i++)
87                {
88                      printf("\nnumero aplicacao %d", i);
89                      printf("\tprocessamento %d", Aplicacao[i].TempoProcessamento);
90                      printf("\tacabou %f",Aplicacao[i].acabou);
91                      printf("\tdisco? %f",Aplicacao[i].AcessouDisco);
92                      printf("\tprimeira proc? %f",Aplicacao[i].PrimeiraVezProc);
93                      printf("\ttservproc %d",Aplicacao[i].TempoServicoProc);
94
95                }
96    }
97    // Retorna o indice da tabela que possui o melhor valor para o índice utilizado
98    // no caso, será sempre o maior valor da tabela de hosts ou 0 caso ele exista.
99    float Verifica_melhor_indice(float tabela[NUM_MAQ+1])
100   {
101         float menor = tabela[1];
102         float menor_indice = 1;
103         int continua =1;
104
105      for (m_i=1; m_i <= NUM_MAQ; m_i++)
106      {
107         //fprintf(saida, "Tabela: %f\n", tabela[m_i]);
108      }
109         if (indice_id == 8)  // indice 8 igual a pvm
110         {
111            if (ultimo_indice == NUM_MAQ+1)
112            {
113              menor_indice = 1;
114              ultimo_indice = 2;
115            }
116            else
117            {
118              menor_indice = ultimo_indice;
119              ultimo_indice++;
120            }
121         }
122         else
123         {
124           if (indice_id == 1)
125           {
126            if (tabela[m_i] > menor)
127            {
128               menor = tabela[m_i];
129               menor_indice = m_i;
130            }
131           }
132           else
133           {
134            m_i = 1;
135              while (m_i<= NUM_MAQ && continua == 1)
136              {
137                    if (tabela[m_i] < menor)
138                    {
139                            menor = tabela[m_i];
140                            menor_indice = m_i;
141                    }
142                m_i++;
143              }
144           }
145          }
146         return (menor_indice);
147   }
148   // Calcula benchmark das máquinas que serão utilizadas pelo escalonador, de modo que
149   a melhor máquina tenha seu valor igual a 1.
150   void Benchmark(void)
```

```
{
     float maior_disco, maior_rede, maior_cpu, maior_memoria;
     int j;
     for (j = 1; j <= NUM_MAQ; j++)
     {
        bench_disco[j] = (Tsd[j]);  // equivalente ao benchmark
        bench_rede[j] = (Tsr[j]);
        bench_cpu[j] = (Tsp[j]);
        bench_memoria[j] = 1/memoria_livre[j];
     }
     maior_disco = bench_disco[1];
     maior_rede = bench_rede[1];
     maior_cpu = bench_cpu[1];
     maior_memoria = bench_memoria[1];
     for (j=2; j <= NUM_MAQ; j++)
     {
         if (bench_disco[j] > maior_disco)
              maior_disco = bench_disco[j];
         if (bench_rede[j] > maior_rede)
              maior_rede = bench_rede[j];
         if (bench_cpu[j] > maior_cpu)
              maior_cpu = bench_cpu[j];
       if (bench_memoria[j] > maior_memoria)
              maior_memoria = bench_memoria[j];
     }
     for (j=1; j <= NUM_MAQ; j++)
     {
         bench_disco[j] = bench_disco[j]/maior_disco;
         bench_rede[j] = bench_rede[j]/maior_rede;
         bench_cpu[j] = bench_cpu[j]/maior_cpu;
         bench_memoria[j] = bench_memoria[j]/maior_memoria;
     }
}
//Função para ininialização da tabela de índices de carga (com benchmark referente a
cada índice)
void Inicializacao_Tabela(void)
{
      int j;
      for (j=1; j<=NUM_MAQ; j++)
      {
        if (indice_id == 0)
        {
            tabela[j] = bench_cpu[j];
        }
        if (indice_id == 1)
        {
            tabela[j] = memoria_livre[j];
        }
        if (indice_id == 2)
        {
            tabela[j] = bench_disco[j];
        }
        if (indice_id == 3)
        {
            tabela[j] = bench_rede[j];
        }
        if (indice_id == 4)
        {
            tabela[j] = ((bench_cpu[j] + bench_disco[j] + bench_rede[j] +
bench_memoria[j])/4);
        }
        if (indice_id == 5)
        {
            tabela[j] = ((bench_cpu[j] + bench_disco[j] + bench_rede[j] +
bench_memoria[j])/4);
        }
        if (indice_id == 6)
        {
            tabela[j] = ((bench_cpu[j] + bench_disco[j] + bench_rede[j] +
bench_memoria[j])/4);
        }
      }
}
// função que vai normalizar todos os valores de cpu, memória, disco e rede - que
será utilizado no índice de desempenho
void Normaliza(void)
{
     float maior_memoria, maior_cpu, maior_disco, maior_rede;
```

```
229       int j;
230       maior_memoria = memoria_livre[1];
231       maior_disco = disco[1];
232       maior_rede = rede[1];
233       maior_cpu = cpu[1];
234       for (j=2; j <= NUM_MAQ; j++)
235       {
236           if (memoria_livre[j] < maior_memoria)
237               maior_memoria = memoria_livre[j];
238
239           if (disco[j] > maior_disco)
240               maior_disco = disco[j];
241           if (rede[j] > maior_rede)
242               maior_rede = rede[j];
243           if (cpu[j] > maior_cpu)
244               maior_cpu = cpu[j];
245        }
246       for (j=1; j <= NUM_MAQ; j++)
247       {
248         memoria_norm[j] = maior_memoria/memoria_livre[j];
249         if (maior_cpu != 0.0)
250         {
251               cpu_norm[j] = cpu[j]/maior_cpu;
252         }
253         if (maior_disco != 0.0)
254         {
255            disco_norm[j] = disco[j]/maior_disco;
256         }
257         if (maior_rede != 0.0)
258         {
259              rede_norm[j] = rede[j]/maior_rede;
260         }
261          // após os valores serem normalizados para que possam ser utilizados para o
262   cálculo do índice de carga
263          // as variáveis cpu, rede e disco [indice] são zeradas para poderem acumular
264   novamente tempo
265       }
266   }
267   void Inicializa_variaveis(void)
268   {
269      tempo_atualizacao = 0.07; //tempos entre atualizações
270       // tempos de serviços de disco, rede e processador
271         Tsp[1] = 0.30;              // Processador 1
272         Tsd[1] = 7.84;              // Disco 1
273         Tsr[1] = 25.6;              // Rede 1
274         Tsp[2] = 0.30;              // Processador 2
275         Tsd[2] = 7.84;              // Disco 2
276         Tsr[2] = 25.6;              // Rede 2
277         Tsp[3] = 0.30;              // Processador 3
278         Tsd[3] = 7.84;              // Disco 3
279         Tsr[3] = 25.6;              // Rede 3
280         Tsp[4] = 0.30;              // Processador 4
281         Tsd[4] = 7.84;              // Disco 4
282         Tsr[4] = 25.6;              // Rede 4
283         Tsp[5] = 0.30;              // Processador 5
284         Tsd[5] = 7.84;              // Disco 5
285         Tsr[5] = 25.6;              // Rede 5
286         Tsp[6] = 0.60;              // Processador 6
287         Tsd[6] = 15.68;              // Disco 6
288         Tsr[6] = 51.2;              // Rede 6
289         Tsp[7] = 0.60;              // Processador 7
290         Tsd[7] = 15.68;              // Disco 7
291         Tsr[7] = 51.2;              // Rede 7
292         Tsp[8] = 0.60;              // Processador 8
293         Tsd[8] = 15.68;              // Disco 8
294         Tsr[8] = 51.2;              // Rede 8
295         Tsp[9] = 0.60;              // Processador 9
296         Tsd[9] = 15.68;              // Disco 9
297         Tsr[9] = 51.2;              // Rede 9
298         Tsp[10] = 0.60;              // Processador 10
299         Tsd[10] = 15.68;              // Disco 10
300         Tsr[10] = 51.2;              // Rede 10
301        /* memoria definida no sistema */
302        memoria_total [1] = 256;
303        memoria_total [2] = 256;
304        memoria_total [3] = 256;
305        memoria_total [4] = 256;
306        memoria_total [5] = 256;
```

```
307       memoria_total [6] = 128;
308       memoria_total [7] = 128;
309       memoria_total [8] = 128;
310       memoria_total [9] = 128;
311       memoria_total [10] = 128;
312       for (m_i = 1; m_i <= NUM_MAQ; m_i++)
313           memoria_livre[m_i] = (memoria_total[m_i] *0.8);
314       /* quantidade transferida por segundo */
315       disco[1] = 0;
316       disco[2] = 0;
317       disco[3] = 0;
318       disco[4] = 0;
319       disco[5] = 0;
320       disco[6] = 0;
321       disco[7] = 0;
322       disco[8] = 0;
323       disco[9] = 0;
324       disco[10] = 0;
325       /* quantidade transferidas por segundo */
326       rede[1] = 0;
327       rede[2] = 0;
328       rede[3] = 0;
329       rede[4] = 0;
330       rede[5] = 0;
331       rede[6] = 0;
332       rede[7] = 0;
333       rede[8] = 0;
334       rede[9] = 0;
335       rede[10] = 0;
336       cpu[1] = 0;
337       cpu[2] = 0;
338       cpu[3] = 0;
339       cpu[4] = 0;
340       cpu[5] = 0;
341       cpu[6] = 0;
342       cpu[7] = 0;
343       cpu[8] = 0;
344       cpu[9] = 0;
345       cpu[10] = 0;
346       quantum[1] = 0.01;
347       quantum[2] = 0.01;
348       quantum[3] = 0.01;
349       quantum[4] = 0.01;
350       quantum[5] = 0.01;
351       quantum[6] = 0.01;
352       quantum[7] = 0.01;
353       quantum[8] = 0.01;
354       quantum[9] = 0.01;
355       quantum[10] = 0.01;
356       /* Atribuicao do tipo da aplicacao */
357       Tipo_aplicacao.proc = TAppProc;  // a aplicacao é x% cpu-bound
358       Tipo_aplicacao.disco = TAppDisco;  // a aplicacao é x% disk-bound
359       Tipo_aplicacao.rede = TAppRede;   // a aplicacao é x% network-bound
360       Tipo_aplicacao.Qmemoria = TAppMemoria;   // ocupa quantidade de memoria
361          for (m_i = 1; m_i <= NUM_TASK; m_i++) // o token (nesse caso a aplicação)
362   não deve ser 0
363          {
364               // possuir valor igual a zero
365                     Aplicacao[m_i].TempoProcessamento = 0;
366                     Aplicacao[m_i].acabou = 0;
367                     Aplicacao[m_i].AcessouDisco = 0;
368                     Aplicacao[m_i].AcabouSwap = 0;
369                     Aplicacao[m_i].AcessouRede = 0;
370                     Aplicacao[m_i].PrimeiraVezProc = 1;
371                     Aplicacao[m_i].TempoServicoProc = 0;
372                     Aplicacao[m_i].TirouMemoria = 1;
373                     Aplicacao[m_i].PrimeiraVezProc_com_disco_rede = 1;
374               }
375   }
376   void Atualizar_tabela(float tabela[NUM_MAQ+1])      // função de atualização da
377   tabela de índices de carga
378   {
379                     // pode ser um vetor ou uma tabela dependendo do índice
380                     // a ser utilizado
381    switch (indice_id)
382    {
383      case cpu_livre: //cpu_livre:
384           for (i = 1; i <= NUM_MAQ; i++)
```

```
385          {
386              tabela[i] = ((cpu[i]+bench_cpu[i]) * bench_cpu[i]);
387          }
388          break;
389     case mem_livre: //mem_livre:
390            for (i = 1; i <= NUM_MAQ; i++)
391          {
392               tabela[i] = (memoria_livre[i] * bench_memoria[i]);
393        }
394              break;
395     case num_escrita_leitura: //num_escrita e leitura
396          for (i = 1; i <= NUM_MAQ;  i++)
397          {
398               tabela[i] = ((disco[i]) * bench_disco[i]);
399          }
400          break;
401     case pacotes_in_out: //pacotes_in_out
402          for (i = 1; i <= NUM_MAQ;  i++)
403          {
404               tabela[i] = ((rede[i]) * bench_rede[i]);
405          }
406          break;
407     case VIP: //índice de desempenho:
408          Normaliza();
409          for (i = 1; i <= NUM_MAQ;  i++)
410          {
411              tabela[i] = sqrt(pow((cpu_norm[i]),2) + pow((memoria_norm[i]),2) +
412   pow((disco_norm[i]),2) + pow((rede_norm[i]),2));
413          }
414          break;
415     case PVIP: //índice de desempenho:
416          Normaliza();
417          for (i = 1; i <= NUM_MAQ;  i++)
418          {
419              tabela[i] = sqrt(((3*Tipo_aplicacao.proc/4)/100.0 *
420   pow((cpu_norm[i]),2)) + ((Tipo_aplicacao.proc/4)/100.0 * pow((memoria_norm[i]),2)) +
421   (Tipo_aplicacao.disco/100.0 * pow((disco_norm[i]),2)) + (Tipo_aplicacao.rede/100.0 *
422   pow((rede_norm[i]),2)));
423          }
424          break;
425    }
426   }
427   main(int argc, char *argv[])
428   {
429   // Abre o arquivo de saída da simulação
430     sprintf(nomearqsaida,"T%s_I%s.out", argv[1],argv[7]);
431     saida = fopen(nomearqsaida,"a");
432   if ((p = sendto(saida)) == NULL)
433       printf("Erro na saida\n");
434          if(argc > 1 && argc > 6)  {
435                NumSemente = atoi(argv[2]);
436                TAppProc = (float)atoi(argv[3]);
437                TAppDisco = (float)atoi(argv[4]);
438                TAppRede = (float)atoi(argv[5]);
439          TAppMemoria = (float)atoi(argv[6]);
440          indice_id = atoi(argv[7]);
441          }
442          else {
443                printf("usage: NumSemente(15) TAppProc(40) TAppDisco(30) TAppRede(30)
444   TAppMemoria(10) Indice(0) \n");
445                return(-1);
446          }
447    /* prepara o sistema de simulacao e da nome ao modelo */
448    smpl(0,"Programa Escalonador");
449    //trace(2);              // depurador do programa...
450    stream(NumSemente);                   //escolhe a semente com que se vai trabalhar
451   (até 15)
452    /* cria e da nome as facilidades */
453    facility("Escalonador",1);
454   char *fim_str = "\0";
455    for (maquina = 1; maquina <= NUM_MAQ; maquina++)
456          {
457           asprintf(&recursos, "Proc%d", maquina);
458           facility(recursos, 1);
459           asprintf(&recursos, "Disco%d", maquina);
460            facility(recursos, 1);  //n máquinas
461            asprintf(&recursos, "Rede%d", maquina);
462            facility(recursos, 1);
```

```
463         }
464  Inicializa_variaveis();  // definir os quantuns, o tipo da aplicacao
465                          // qual indice vai usar ... etc
466  Benchmark();  // define hierarquia de máquinas
467  //Inicializacao_Tabela();  //define valores iniciais da tabela de índices.
468  /* escalona a atualizacao da tabela  */
469  schedule(0, 0.0, Atualiza_tabela);
470  /* escalona a chegada das aplicacoes*/
471  for (i = 1 ; i <= NUM_TASK; i++)                    // o token nunca deve ser igual
472 a zero por isso deve ser
473  {
474      schedule(1,poisson(Ta1),i);                                  // iniciado no 1
475  }
476 while ( num_tours )             // igual ao número de aplicações que passarão pela
477 simulação
478 {
479    cause(&Event,&i);
480    Cliente = &Aplicacao[i];
481    switch(Event)
482  {
483     case 0:
484      // Atualiza a tabela
485      Atualizar_tabela(tabela);
486      flag_atualizou_tabela = 1;
487      // Escalona a próxima atualização para daqui a tempo_atualizacao unidades de
488 tempo
489      schedule(0,tempo_atualizacao, Atualiza_tabela);
490     break;
491
492     case 1:
493       if (request("Escalonador",1,i,0) == 0) //solicita escalonador
494       {
495          if (flag_atualizou_tabela == 1)
496          {
497           flag_atualizou_tabela = 0;
498           tmp_aux = (expntl(Ts1) + expntl(Tat) + expntl(Tat2));
499              schedule(2,tmp_aux,i);
500          }
501          else
502              {
503               schedule(2,expntl(Ts1),i);
504               }
505        }
506     break;
507     case 2:
508     release("Escalonador", i);                              // libera o escalonador
509     melhor = Verifica_melhor_indice(tabela);// na variavel melhor vai estar o
510                                             // numero da maquina que possui o
511                                             // melhor indice para a execucao
512      Aplic_Maq[i].Maquina = melhor;
513      schedule(3, 0.0, i);        // escalona na melhor maquina
514       if (Tipo_aplicacao.rede != 0.0)
515       {
516         rede[melhor] = rede[melhor] + ((Tipo_aplicacao.rede/100.0) * Tsr[melhor]);
517       }
518       if (Tipo_aplicacao.disco!= 0.0)
519       {
520         disco[melhor] = disco[melhor] + ((Tipo_aplicacao.disco/100.0) *
521 Tsd[melhor]);
522       }
523       if (Tipo_aplicacao.proc!= 0.0)
524       {
525         cpu[melhor] = cpu[melhor] + (((3*Tipo_aplicacao.proc/4)/100.0) *
526 Tsp[melhor]);
527       }
528      break;
529     /*************/
530     /* MAQUINA   */
531     /*************/
532     case 3:
533       maquina = Aplic_Maq[i].Maquina;
534       asprintf(&recursos,"Proc%d", maquina);
535       if (request(recursos,3,i,0) == 0)     // requisita o Processador
536       {
537         if (Cliente->PrimeiraVezProc == 1)
538         {
539           Cliente->TempoServicoProc =  (Tipo_aplicacao.proc/100.0) *
540 expntl(Tsp[maquina]);
```

```
541           cpu[maquina] = cpu[maquina] + (Cliente->TempoServicoProc);
542           Cliente->PrimeiraVezProc = 0;
543           if ((memoria_livre[maquina] < 0) || ((memoria_livre[maquina] -
544   Tipo_aplicacao.Qmemoria) < 0))
545           {
546              schedule(4, 0.0, i);
547              schedule(7, 0.0, i);            // Escalona disco 1 por causa do swap
548              Cliente->TirouMemoria = 0;
549              break;
550           }
551           else
552              memoria_livre[maquina] = memoria_livre[maquina] -
553   Tipo_aplicacao.Qmemoria;
554         }
555          if ((Cliente->TempoProcessamento + quantum[maquina]) <= Cliente-
556   >TempoServicoProc)
557           {
558               schedule(4,quantum[maquina],i);
559               Cliente->TempoProcessamento = Cliente->TempoProcessamento +
560   quantum[maquina];
561           }
562           else
563           {
564              schedule (4,Cliente->TempoServicoProc - Cliente->TempoProcessamento, i);
565              Cliente->TempoProcessamento = Cliente->TempoServicoProc;
566           }
567         }
568         break;
569       case 4:
570          maquina = Aplic_Maq[i].Maquina;
571          asprintf(&recursos, "Proc%d", maquina);
572          release(recursos, i);      // libera o Processador 1
573          if (Cliente->PrimeiraVezProc_com_disco_rede == 1)
574          {
575            if ((Tipo_aplicacao.disco != 0.0))
576            {
577               schedule(5, 0.0, i);               // Escalona disco 1
578            }
579            if ((Tipo_aplicacao.rede != 0.0))
580            {
581              schedule(9, 0.0, i);
582            }
583          Cliente->PrimeiraVezProc_com_disco_rede = 0;
584          }
585         if ((Tipo_aplicacao.rede != 0.0) || (Tipo_aplicacao.disco != 0.0))
586             {
587               if ((Tipo_aplicacao.rede != 0.0) && (Tipo_aplicacao.disco != 0.0)) //
588   disco e rede
589           {
590               if ((Cliente->TempoProcessamento == Cliente->TempoServicoProc) &&
591   (Cliente->acabou == 0))
592           {
593                  if (Cliente->TirouMemoria == 0)
594                  {
595                    if (Cliente->AcabouSwap == 1 && (Cliente->AcessouDisco == 1) &&
596   (Cliente->AcessouRede ==1))
597                    {
598                       Cliente->acabou = 1;
599                       num_tours--;
600                     }
601                  }
602                  else
603                  {
604                      if ((Cliente->AcessouDisco != 0) && (Cliente->AcessouRede !=0))
605                      {
606                   Cliente->acabou = 1;
607                   memoria_livre[maquina] = memoria_livre[maquina] +
608   Tipo_aplicacao.Qmemoria;
609                num_tours--;
610               }
611           }
612           }
613         }
614           if (Tipo_aplicacao.rede != 0.0 && Tipo_aplicacao.disco == 0.0) //só rede
615           {
616               if ((Cliente->TempoProcessamento == Cliente->TempoServicoProc) &&
617   (Cliente->acabou == 0))
618              {
```

```
                  if (Cliente->TirouMemoria == 0)
                  {
                    if (Cliente->AcabouSwap == 1 && (Cliente->AcessouRede !=0))
                    {
                       Cliente->acabou = 1;
                       num_tours--;
                    }
                  }
                  else
                  {
                     if (Cliente->AcessouRede !=0)
                     {
                   Cliente->acabou = 1;
                   memoria_livre[maquina] = memoria_livre[maquina] +
Tipo_aplicacao.Qmemoria;
                 num_tours--;
              }
            }
            }
         }
         if (Tipo_aplicacao.rede == 0.0 && Tipo_aplicacao.disco != 0.0) //só disco
         {
             if ((Cliente->TempoProcessamento == Cliente->TempoServicoProc) &&
(Cliente->acabou == 0))
            {
                  if (Cliente->TirouMemoria == 0)
                  {
                    if (Cliente->AcabouSwap == 1 && Cliente->AcessouDisco ==1)
                    {
                       Cliente->acabou = 1;
                       num_tours--;
                    }
                  }
                  else
                  {
                    if (Cliente->AcessouDisco !=0)
                    {
                   Cliente->acabou = 1;
                   memoria_livre[maquina] = memoria_livre[maquina] +
Tipo_aplicacao.Qmemoria;
                num_tours--;
               }
            }
            }
         }
      }
      else
      {
       if ((Tipo_aplicacao.rede == 0.0) && (Tipo_aplicacao.disco == 0.0) &&
(Cliente->TempoProcessamento == Cliente->TempoServicoProc) && (Cliente->acabou ==
0)) //só processador
           {
               if (Cliente->TirouMemoria == 0)
                  {
                    if (Cliente->AcabouSwap == 1)
                    {
                       Cliente->acabou = 1;
                       num_tours--;
                    }
                  }
                  else
                  {
                Cliente->acabou = 1;
                memoria_livre[maquina] = memoria_livre[maquina] +
Tipo_aplicacao.Qmemoria;
              num_tours--;
            }
           }
              }

      if ((Tipo_aplicacao.rede != 0.0) || (Tipo_aplicacao.disco != 0.0))
           {
             if ((Tipo_aplicacao.rede != 0.0) && (Tipo_aplicacao.disco != 0.0)) //
disco e rede
         {
                  if ((Cliente->acabou == 0) && (Cliente->TempoProcessamento !=
Cliente->TempoServicoProc))
             {
```

```
                    if (Cliente->TirouMemoria == 0)
                    {
                      if (Cliente->AcabouSwap == 1 && Cliente->AcessouDisco != 0 &&
(Cliente->AcessouRede !=0))
                      {
                        schedule(3, 0.0, i);       // Escalona o processador
novamente porque ainda
                        // não acabou o quantun
                      }
                    }
                    else
                    {
                          if ((Cliente->AcessouDisco != 0) && (Cliente->AcessouRede
!=0))
                          {
                  schedule(3, 0.0, i);       // Escalona o processador novamente
porque ainda
                        // não acabou o quantun
                      }
              }
                  }
         }
         if (Tipo_aplicacao.rede != 0.0 && Tipo_aplicacao.disco == 0.0) //só rede
         {
           if ((Cliente->acabou == 0) && (Cliente->TempoProcessamento != Cliente-
>TempoServicoProc))
               {
                   if (Cliente->TirouMemoria == 0)
                   {
                     if (Cliente->AcabouSwap == 1 && Cliente->AcessouRede ==1)
                       {
                         schedule(3, 0.0, i);       // Escalona o processador
novamente porque ainda
                       // não acabou o quantun
                        }
                    }
                    else
                    {
                          if (Cliente->AcessouRede ==1)
                          {
                    schedule(3, 0.0, i);       // Escalona o processador novamente
porque ainda
                         // não acabou o quantun
                }
              }
           }
         }
         if (Tipo_aplicacao.rede == 0.0 && Tipo_aplicacao.disco != 0.0) //só disco
         {
           if ((Cliente->acabou == 0) && (Cliente->TempoProcessamento != Cliente-
>TempoServicoProc))
           {
              if (Cliente->TirouMemoria == 0)
                          {
                 if ((Cliente->AcessouDisco !=0) && (Cliente->AcabouSwap == 1))
                {
                         schedule(3, 0.0, i);       // Escalona o processador
novamente porque ainda
                       // não acabou o quantun
                       }
             }
             else
             {
              if ((Cliente->AcessouDisco !=0))
               {
                         schedule(3, 0.0, i);       // Escalona o processador
novamente porque ainda
                       // não acabou o quantun
                 }
             }
           }
         }
       }
       else
       {
        if ((Tipo_aplicacao.rede == 0.0) && (Tipo_aplicacao.disco == 0.0)) //só
processador
          {
```

```
                if ((Cliente->acabou == 0) && (Cliente->TempoProcessamento !=
Cliente->TempoServicoProc))
                {
                   if (Cliente->TirouMemoria == 0)
                   {
                     if (Cliente->AcabouSwap == 1)
                     {
                       schedule(3, 0.0, i);       // Escalona o processador
novamente porque ainda
                      // não acabou o quantun
                      }
                   }
                   else
                   {
                schedule(3, 0.0, i);       // Escalona o processador novamente
porque ainda
                     // não acabou o quantun
              }
                  }
           }
             }

       break;
    case 5:
      maquina = Aplic_Maq[i].Maquina;
      asprintf(&recursos, "Disco%d", maquina);
      if (request(recursos,5,i,0) == 0)    // Disco 1 requisitado
      {
        disco[maquina] = disco[maquina] - ((Tipo_aplicacao.disco/100.0) *
Tsd[maquina]);
        Cliente->tempo_disco = (Tipo_aplicacao.disco/100.0) * expntl(Tsd[maquina]);
        schedule(6,Cliente->tempo_disco,i);
      }
      break;
    case 6:
      maquina = Aplic_Maq[i].Maquina;
      asprintf(&recursos, "Disco%d", maquina);
      release(recursos, i);              // Disco 1 liberado
      Cliente->AcessouDisco = 1;
      schedule(3, 0.0, i);
      break;
    case 7:
      maquina = Aplic_Maq[i].Maquina;
      asprintf(&recursos, "Disco%d", maquina);
      if (request(recursos,7,i,0) == 0)    // Disco 1 requisitado
      {
        Cliente->tempo_disco = (2.0/100.0) * expntl(Tsd[maquina]);
        schedule(8,Cliente->tempo_disco,i);
        disco[maquina] = disco[maquina] + (Cliente->tempo_disco);
      }
      break;
    case 8:
      maquina = Aplic_Maq[i].Maquina;
      asprintf(&recursos, "Disco%d", maquina);
      release(recursos, i);              // Disco 1 liberado por swap
      Cliente->AcabouSwap = 1;
      schedule(3, 0.0, i);
      break;
    case 9:
      maquina = Aplic_Maq[i].Maquina;
      asprintf(&recursos, "Rede%d", maquina);
      if (request(recursos,9,i,0) == 0)    // Rede 1 requisitada
      {
        rede[maquina] = rede[maquina] - ((Tipo_aplicacao.rede/100.0) *
Tsr[maquina]);
        Cliente->tempo_rede = (Tipo_aplicacao.rede/100.0) * expntl(Tsr[maquina]);
        schedule(10,Cliente->tempo_rede,i);
      }
      break;
    case 10:
      maquina = Aplic_Maq[i].Maquina;
      asprintf(&recursos, "Rede%d", maquina);
      release(recursos, i);              // Rede 1 liberada
      Cliente->AcessouRede = 1;
      schedule(3, 0.0, i);
      break;
 } //switch
} // while
```

```
853   // gera o relatorio da simulacao
854    fprintf(saida,"%f\n",time_sim());
855   /*for (maquina = 1; maquina <= NUM_MAQ; maquina++)
856         {
857           asprintf(&recursos, "Proc%d", Aplic_Maq[maquina].Maquina);
858            fprintf(saida, "Utilizacao Processador %d:
859   %f\n",Aplic_Maq[maquina].Maquina,utilizacao_recurso(recursos));
860           asprintf(&recursos, "Disco%d", Aplic_Maq[maquina]).Maquina;
861             fprintf(saida, "Utilizacao Disco %d:
862   %f\n",Aplic_Maq[maquina].Maquina,utilizacao_recurso(recursos));
863           asprintf(&recursos, "Rede%d", Aplic_Maq[maquina]).Maquina;
864             fprintf(saida, "Utilizacao Rede %d:
865   %f\n",Aplic_Maq[maquina].Maquina,utilizacao_recurso(recursos));
866          }*/
867   // Fecha arquivo de saida.
868    fclose(saida);
869   }
870   /* ------------------------------------------------------------------- */
871
```