Paralelização da busca local da metaheurística A-BRKGA com OpenACC

Willian da Silva Zocolau

Orientador: Prof. Dr. Álvaro Luiz Fazenda
Coorientador: Prof. Dr. Antônio Augusto Chaves

Universidade Federal de São Paulo
Instituto de Ciência de Tecnologia
Bacharelado em Ciência da Computação

Agradecimentos

Sumário

Introdução

Contextualização e Motivação

- Tempo proibitivo para encontrar soluções de problemas combinatoriais "grandes"
- Domínio que, em geral, é bastante amplo
- Verificar todas as instâncias do domínio é impraticável
- Soluções aproximadas também são desafiadoras

Contextualização e Motivação

- Abordagem metaheurística: A-BRKGA
  - Soluções interessantes
  - Tempo computacional adequado
- Técnica de programação paralela
  - Apresentar solução com tempo de execução menor

Objetivos

- Geral: propor estratégias para ganho de desempenho da busca local do A-BRKGA com ajuda do padrão OpenACC
- Específicos
  - Revisão bibliográfica sobre metaheurística, BRKGA e A-BRKGA
  - Descrever o desempenho do método A-BRKGA
  - Paralelização através do OpenACC
  - Resultados e discussões

Revisão Bibliográfica

Problemas de otimização combinatória

São caracterizados por:

- Estudo através de fundamentos matemáticos
- Encontrar a melhor solução em domínio grande (finito)
- Interesse por solução que:
  - Maximize o ganho
  - Minimize a perda
- Função objetivo: determina o valor de cada solução
- Conjunto de restrições: define o que é solução uma possível

MALAQUIAS (2006) [13]

BRKGA e A-BRKGA

- Baseado no princípio *darwinista* com mais detalhes
- Estrutura de chaves aleatórias definidas no intervalo de [0,1]
- Decodificador mapeia para solução e cálcula o custo
- BRKGA
  - Vários parâmetros precisam ser configurados
- A-BRKGA: principais contribuições
  - Parâmetros são ajustados para buscar balanceamento entre diversidade e intesidade
  - Nova estratégia para definir indivíduos pertencentes ao grupo elite

CHAVES (2013) [4]

Modelo abstrato proposto por OpenACC

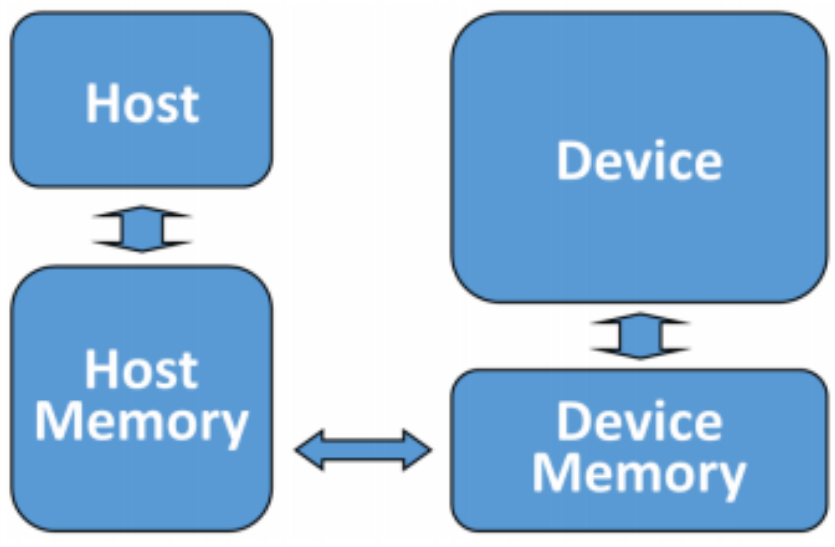

Figura 4: Modelo abstrato de aceleradores

Fonte: (OPENACC.ORG, 2015b)

- Suporte para arquiteturas iguais ou totalmente diferentes [20]

Sintaxe de OpenACC

```
1 #pragma acc <clausulas>
```

```
1 #pragma acc parallel loop
2 for (i = 0; i < N; i++) {
3   y[i] = 0.0 f;
4   x[i] = (float)(i + 1);
5 }
```

Métricas clássicas de desempenho

$$Speedup(P) = \frac{Texec(1 * proc)}{Texec(P * procs)} \quad (1)$$

[8]

- P = número de processadores
- $1 \leq Speedup \leq P$

$$E = \frac{Speedup(P)}{P} \quad (2)$$

[8]

- P = número de processadores
- 0 < Eficiência ≤ 1

Metodologia

Análises iniciais

```
1     %   cumulative   self              self     total
2    time   seconds   seconds    calls  ms/call  ms/call  name
3    79.48    523.25   523.25   199102     2.63     2.67  Decoder(TSol)
4    18.88    647.54   124.28     1588    78.26    78.26  LocalSearch(TSol)
5     1.13    655.00     7.46   198873     0.04     0.04  IC(int, int, double)
6     0.35    657.30     2.30   164217     0.01     0.02  ParametricUniformCrossover(int)
7     0.12    658.08     0.78   198831     0.00     0.00  CalculateFO(TSol)
8     0.03    658.28     0.20   164217     0.00     0.00  std::vector<TVecSol, std::allocator
9     0.02    658.40     0.12    30986     0.00     0.00  CreateInitialSolutions()
10    0.00    658.43     0.03                             A_BRKGA()
```

Figura 5: *Profiler* gerado pelo GProf

Análises iniciais

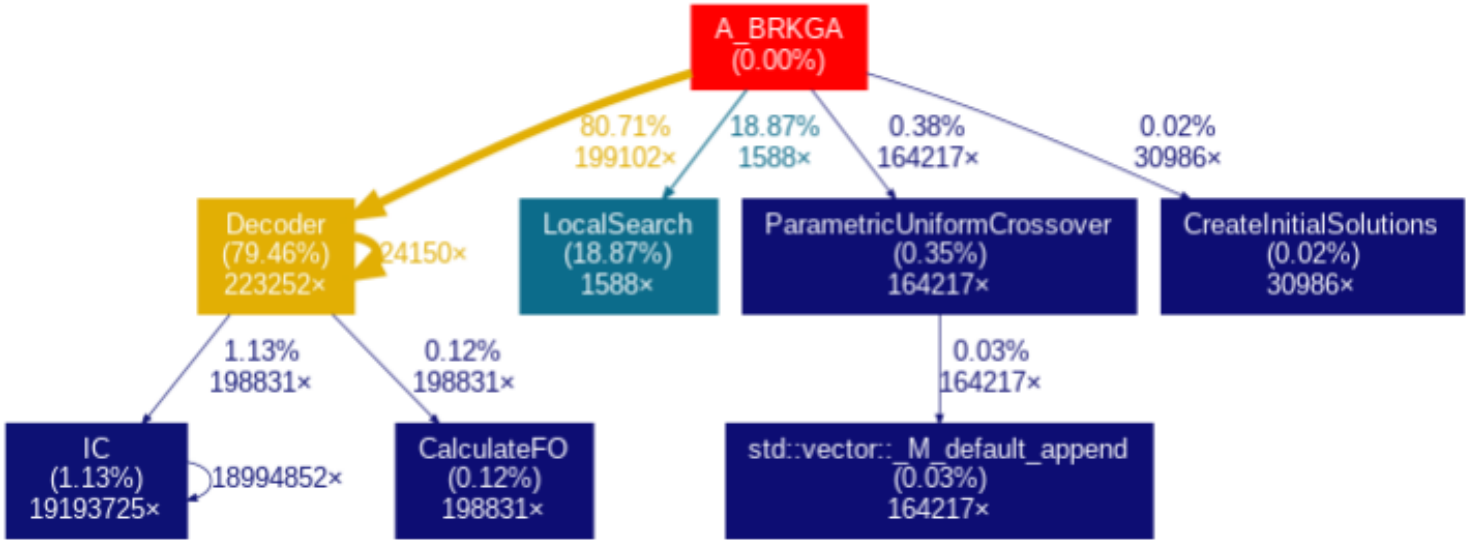

Figura 6: Análise de trechos demandantes com visualizador gprof2dot

Análises iniciais

```
1   ======== CPU profiling result (top down):
2   Time(%)      Time  Name
3    100.00%   736.31s  ???
4    100.00%   736.31s    main
5    100.00%   736.29s      A_BRKGA(void)
6    77.22%    568.61s      | Decoder(TSol)
7    21.75%    160.18s      | LocalSearch(TSol)
8
9   ======== Data collected at 100Hz frequency
10  ======== Percentage threshold: 5%
```

Figura 7: *Profiler* gerado pelo PGProf

- Paralelização da busca local em detrimento do Decoder
- A-BRKGA requer que Decoder seja implementado para cada tipo de problema [4]

Desenvolvimento

Análise de dependência

```
18 for (int i = 0; i < n - 1; i++) {
19     for (int j = i + 2; j < n; j++) {
           ...
           // Os valores de AfoOpt[i][j], AMi[i] e AMj[j] sao definidos
           ...

51         for (int k = 0; k < n; k++) {
52             temp_fo += dist[s.vec[k%n].sol][s.vec[(k+1)%n].sol];
53         }
54     }
55 }
```

Código 4.1.1: Laços de repetição para definir AfoOpt, AMi e AMj com dependência

```
60 for (int i = 0; i < n - 1; i++) {
61     for (int j = i + 2; j < n; j++) {
62         if (AfoOpt[i][j] < Bestfo) {
63             Bestfo = AfoOpt[i][j];
64             Besti = i;
65             Bestj = j;
66         }
67     }
68 }
```

Código 4.1.2: Laços de repetição para encontrar Bestfo, Besti e Bestj com dependência

Otimização dos laços de repetição

```
17 #pragma acc parallel loop collapse(2) \
18     private(foOpt, vi, viP, viM, vj, vjM, vjP, temp_fo)
19 {
20     for (int i = 1; i < n - 1; i++) {
21         for (int j = 3; j < n - 1; j++) {
22             if (j >= i + 2) {
                   ...
                   // Os valores de AfoOpt[i][j], AMi[i] e AMj[j] sao definidos
                   ...

53                 #pragma acc loop independent reduction(+: temp_fo)
54                 for (int k = 0; k < n; k++) {
55                     temp_fo +=
56                         temp_dist[stemp_vec[k\%n].sol][stemp_vec[(k+1)\%n].sol];
57                 }
58             }
59         }
60     }
61 }
```

Código 4.2.1: Laços de repetição para encontrar Bestfo, Besti e Bestj sem dependência e otimizado

Otimização dos laços de repetição

```
70 #pragma acc parallel loop collapse(2) reduction(min:Bestfo)
71 for (int i = 1; i < n - 1; i++) {
72     for (int j = 3; j < n - 1; j++) {
73         if (j >= i + 2) {
74             if (AfoOpt[i][j] < Bestfo) {
75                 Bestfo = AfoOpt[i][j];
76             }
77         }
78     }
79 }
80
81 #pragma acc parallel loop collapse(2)
82 for (int i = 1; i < n - 1; i++) {
83     for (int j = 3; j < n - 1; j++) {
84         if (j >= i + 2) {
85             if (AfoOpt[i][j] == Bestfo && (i >= Besti && j >= Bestj)) {
86                 #pragma acc atomic write
87                 Besti = i;
88                 #pragma acc atomic write
89                 Bestj = j;
90             }
91         }
92     }
93 }
```

Código 4.2.2: Laços de repetição para encontrar Bestfo, Besti e Bestj sem dependência e otimizado

Otimização das transferências de dados

```
1  void mallocAcc() {
2      AfoOpt = (double **)malloc(n * sizeof(double *));
3      temp_dist = (double **)malloc(n * sizeof(double *));
4      AMi = (int *)malloc(n * sizeof(int));
5      AMj = (int *)malloc(n * sizeof(int));
6
7      for (int i = 0; i < n; i++) {
8          AfoOpt[i] = (double *)malloc(n * sizeof(double));
9          temp_dist[i] = &dist[i][0];
10     }
11
12     #pragma acc enter data copyin(temp_dist[0:n][0:n]) \
13         create(AfoOpt[0:n][0:n], AMi[0:n], AMj[0:n])
14 }
```

Código 4.3.1: Alocação de memória e transferência de dados

```
1  void deallocAcc() {
2      for (int i = 0; i < n; i++) {
3          free(AfoOpt[i]);
4      }
5      free(AfoOpt);
6      free(temp_dist);
7      free(AMi);
8      free(AMj);
9      #pragma acc exit data \
10         delete(temp_dist[0:n][0:n], AfoOpt[0:n][0:n], AMi[0:n], AMj[0:n])
11 }
```

Otimização das transferências de dados

```
1   TSol LocalSearch(TSol s)
2   {
        ...
10      #pragma acc data \
11          pcopy(stemp_vec[0 : n], stemp_fo) \
12          pcreate(Bestfo, Bestj, Besti) \
13          present(temp_dist[0:n][0:n], AfoOpt[0:n][0:n], AMi[0:n], AMj[0:n])
14      {
            ...
        }
        ...
132     return s;
133 }
```

Código 4.3.3: Transferência de dados para cada busca local

Resultados

Resultados - GPU

Instância	Speedup	Porcentagem melhorada
zi929	1.16	14.03%
nrw1379	1.13	11.38%

Tabela 9: Medidas de desempenho do tempo total

Instância	Speedup	Porcentagem melhorada
zi929	2.77	63.91%
nrw1379	2.08	51.88%

Tabela 10: Medidas de desempenho da busca local

Resultados - Multicore

Cores	Speedup	Eficiência	Porcentagem melhorada
2	1.03	51.59%	3.09%
4	1.11	27.64%	9.57%
8	1.20	15.02%	16.79%
16	1.30	8.10%	22.82%
28	1.28	4.56%	21.62%

Tabela 15: Medidas de desempenho do tempo total (nrw1379)

Cores	Speedup	Eficiência	Porcentagem melhorada
2	1.15	57.49%	13.03%
4	2.00	50.03%	50.03%
8	3.93	49.18%	74.58%
16	7.58	47.40%	86.81%
28	12.13	43.31%	91.75%

Tabela 16: Medidas de desempenho da busca local (nrw1379)

Resultados

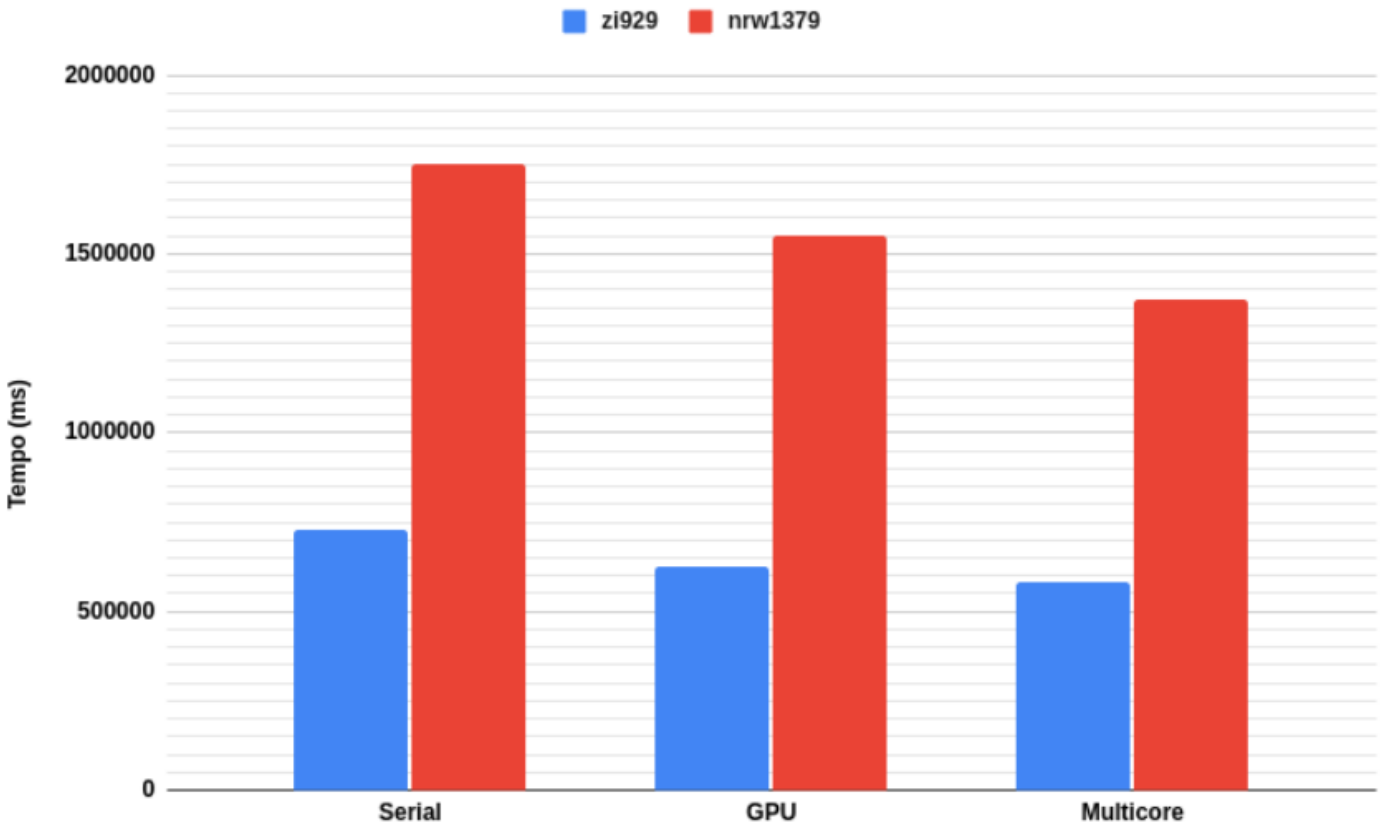

Figura 11: Gráfico do tempo total de execução para serial, GPU e multicore

Resultados

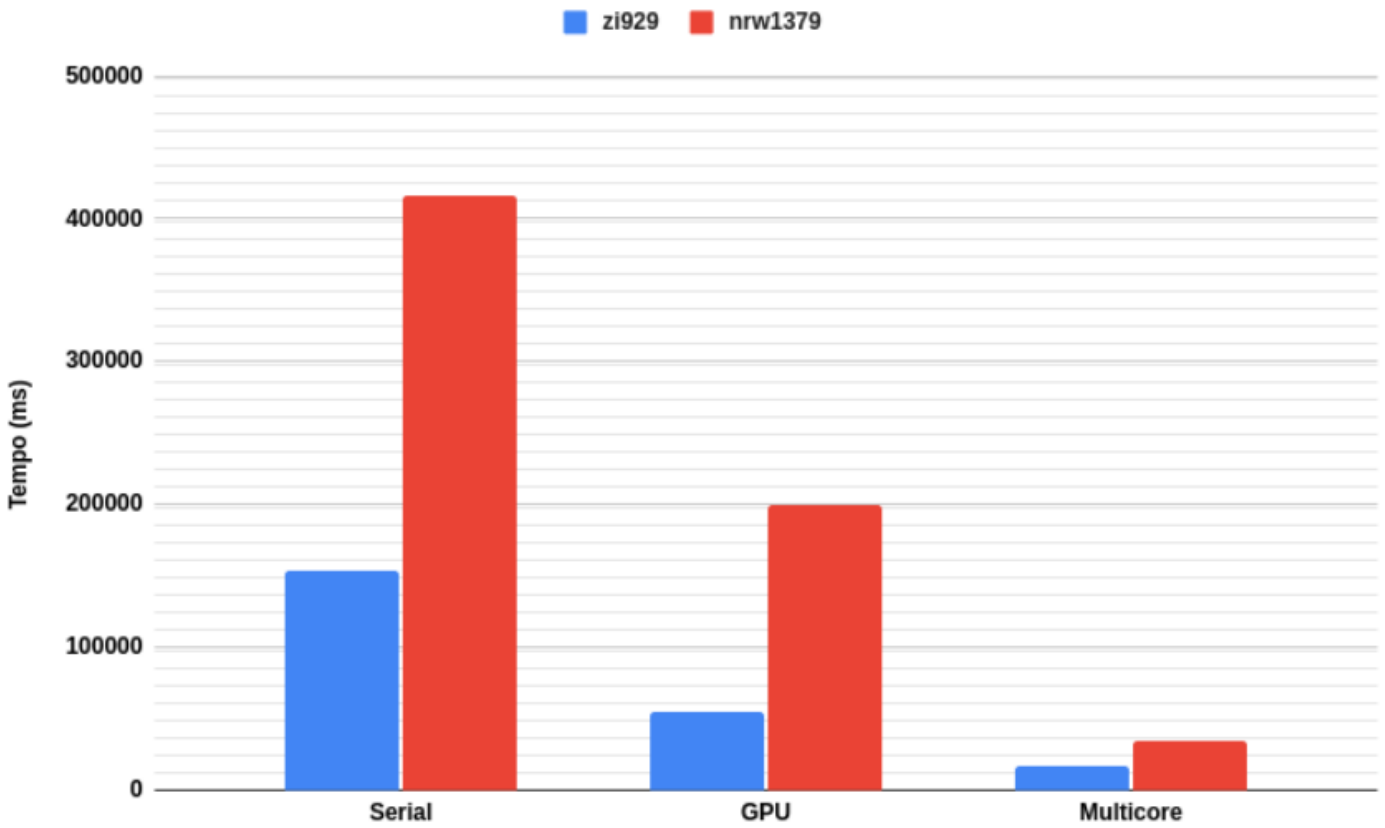

Figura 12: Gráfico do tempo da busca local para serial, GPU e multicore

Trabalhos futuros

Trabalhos futuros

- Notificações do PGProf
- Uso do padrão de programação C++17 com CUDA [16]
- Cuda C++ Standard Library (libcu++) [17]
- Kokkos C++ [7]

Referências

Referências

[1] BEAN, J. C. Genetic algorithms and random keys for sequencing and optimization. Journal on Computing, v. 6, p. 154–160, 1994.

[2] BEN-ARI, M. Principles of Concurrent and Distributed Programming (2Nd Edition) (Prentice-Hall International Series in Computer Science). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006. ISBN 032131283X.

[3] CHATAIN, P. L. Hybrid parallel programming - evaluation of openacc, 2012.

[4] CHAVES, A. A.; GONÇALVES, F. J. Desenvolvimento de um método híbrido flexível com calibração automática de parâmetros. FAPESP, 2016.

[5] CONTE, D. Introdução ao cuda, 2017.

[6] CORMEN, T. H. Introduction to algorithms. v. 3rd Edition, p. 1048–1128, 2009.

[7] EDWARDS, H. C.; TROTT, C. R.; SUNDERLAND, D. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. Journal of Parallel and Distributed Computing, v. 74, n. 12, p. 3202 – 3216, 2014. ISSN 0743-7315. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0743731514001257>

[8] FAZENDA, L.; STRINGHINI, D. Como programar aplicações de alto desempenho com produtividade. XXXIX Congresso da Sociedade Brasileira de Computação, 2019.

Referências

[9] GONÇALVES, J. F.; RESENDE, M. G. C.; MENDES, J. J. M. A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. Journal of Heuristics, v. 17, p. 467–486, 2011.

[10] HERLIHY, M.; SHAVIT, N. The Art of Multiprocessor Programming. [S.l.]: Morgan Kaufmann, 2006. ISBN B008CYT5TS.

[11] LEANDRO, Z.; FERREIRA, A.; MATSUMOTO, M. Arquitetura e programação de gpu nvidia. 2012.

[12] LISBOA, E. F. A. Pesquisa operacional. v. 1–4, 2002.

[13] MALAQUIAS, N. G. L. Uso de algoritmos genéticos para a otimização de rotas de distribuição. p. 28–29, 2006.

[14] MIYAZAWA, F. K.; SOUZA, C. C. Introdução à otimização combinatória. JAI-SBC, p. 2–3, 2015.

[15] MPI-FORUM.ORG. A message-passing interface standard. MPI Forum Org, v. 3, 2015.

[16] OLSEN DAVID; LOPEZ, G. L. B. The Cuda C++ Standard Library. 2019. Disponível em: <https://on-demand.gputechconf.com/supercomputing/2019/pdf/sc1942-the-cuda-c++-standard-library.pdf>.

Referências

[17] OLSEN DAVID; LOPEZ, G. L. B. Accelerating Standard C++ with GPUs Using stdpar. 2020. Disponível em: <https://developer.nvidia.com/blog/accelerating-standard-c-with-gpususing-stdpar/>

[18] OPENACC-STANDARD.ORG. Complex data management in openacc. www.openacc.org, 2014.

[19] OPENACC.ORG. The openacc: Application programming interface. OpenACC-Standard.org, 2015.

[20] OPENACC.ORG. Openacc programming and best practices guide. OpenACC-Standard.org, 2015.

[21] OPENMP.ORG. Openmp application programming interface. OpenMP Architecture Review Board, v. 5, 2018.

[22] PACHECO, P. S. An introduction to parallel programming. n. 1, 2011.

[23] PGPROF. Pgi profiler user's guide. www.pgroup.com, 2018

[24] RAYWARD-SMITH, V. J. et al. Modern heuristic search methods. John Wiley Sons, 1996.

[25] RESENDE, M. G. C. Introdução aos algoritmos genéticos de chaves aleatórias viciadas. Simpósio Brasileiro de Pesquisa Operacional, 2013.