Avaliação de Desempenho de um Sistema de Programação Paralela Baseado em Tarefas Assíncronas

Denilson S. Bélo, Liria M. Sato, Edson T. Midorikawa

Departamento de Engenharia de Computação e Sistemas Digitais (PCS)
Escola Politécnica da Universidade de São Paulo (EPUSP)
Av. Professor Luciano Gualberto, tv 3, nº 158 – São Paulo – SP – Brasil

{denilson.belo, liria.sato, emidorik}@usp.br

Abstract. *Currently, there is a dissemination in the efficient exploitation in parallel computing processing resources, including e.g., coprocessors and remote resources. Parallel programming is a complex task because it requires a parallelization strategies knowledge. The use of heterogeneous resources further increases this difficulty. This paper presents the performance evaluation of the RTE system, which provides a common parallel programming interface based on running asynchronous tasks in heterogeneous resources. This evaluation used a Bioinformatics application to analyze various genomes. The results show that the RTE facilitates the development and the RTE implemented application has a similar performance to the application using the MPI interface.*

Resumo. *Atualmente há uma disseminação pela exploração eficiente de recursos de processamento em computação paralela, incluindo coprocessadores e recursos remotos. A programação paralela é uma tarefa complexa, pois exige o entendimento das estratégias de paralelização. O uso de recursos heterogêneos intensifica essa dificuldade. Este artigo apresenta a avaliação de desempenho do sistema RTE, que oferece uma interface comum de programação paralela baseada em tarefas assíncronas executadas em recursos heterogêneos. Esta avaliação foi feita em uma aplicação em Bioinformática para análise de diversos genomas. Os resultados mostram que RTE facilita a tarefa de desenvolvimento e a aplicação apresentou um desempenho similar à aplicação com MPI.*

1. Introdução

O uso da computação paralela está mais difundido atualmente [Dash 2019]. Os equipamentos de uso comum como computadores pessoais, telefones celulares e *tablets* possuem processadores com mais de um núcleo (*core*), exigindo que seus aplicativos utilizem esses recursos com mais eficiência. Os computadores podem estar conectados em um *cluster* aumentando assim sua capacidade de processamento e utilizar coprocessadores tais como *General Purpose Graphics Processing Unit (GPGPU)*[Kirk and Hwu 2017] e/ou Intel® Xeon Phi™ [Jeffers et al. 2016] para melhorar o desempenho. Computadores que estão na lista dos mais rápidos do mundo [Top500 2019], como o supercomputador brasileiro Santos Dumont (SDumont) [LNCC 2019], utilizam esses recursos para melhorar o desempenho. Ambientes comerciais e corporativos exploram cada vez mais a utilização da computação distribuída como por exemplo em *Internet of Thinks (IoTs)* [Yasuda-Masuoka et al. 2019] e na infraestrutura de *cloud computing*. Há,

também, a utilização de recursos de alto desempenho computacional em Ciência dos Dados [Schmidt and Hildebrandt 2017], como *Atmospheric Radiation Measurement (ARM) Climate Research Facility*[1] [Devarakonda et al. 2019] que realiza pesquisa na área de estudos climáticos e atmosféricos globais; e na área de biotecnologia [Broner 2017], em genética mais especificamente, vem crescendo o uso de supercomputadores em suas pesquisas.

Contudo a programação paralela continua sendo uma tarefa complexa, pois o desenvolvedor necessita ter o entendimento dos mecanismos e estratégias de paralelização e a exploração de paralelismo dos recursos disponíveis. Padrões de programação paralela como o OpenMP [OpenMP 2018], voltado a sistemas de memória compartilhada, e o MPI [MPI 2015], utilizado em computação distribuída, juntamente com os ambientes de programação como NVIDIA® CUDA®[NVIDIA 2019], OpenACC™[OpenACC 2019], OpenCL™[Khronos 2019], SYCL™[Keryell et al. 2019] e mais recentemente a Intel® oneAPI™[Intel 2020], exigem uma formação específica dos desenvolvedores de aplicações paralelas. Diferentes coprocessadores presentes no computador ou até mesmo recursos remotos poderiam ser utilizados na execução de aplicações paralelas. Neste cenário ferramentas de programação poderiam fornecer interfaces únicas entre o computador principal e tais recursos.

Este artigo apresenta a avaliação de desempenho de um sistema, o *Remote Tasks Execute (RTE)*, voltado para a programação e processamento de aplicações paralelas baseadas em tarefas remotas assíncronas, executadas em coprocessadores ou recursos remotos. O RTE busca facilitar o uso de paralelismo de tarefas em aplicações paralelas para ambientes com recursos de processamento heterogêneos. Esta facilidade é dada através da chamada de funções de uma interface única, independente do recurso a ser utilizado. Uma restrição do RTE é que as tarefas são do tipo *Bag-of-Tasks (BoT)*, ou seja, não se comunicam entre si. A avaliação de desempenho foi realizada utilizando-se computadores remotos na execucão das tarefas de uma aplicação na área de Bioinformática, buscando analisar o comportamento do sistema quanto ao custo de comunicação e ao desempenho comparativo ao uso da interface MPI que é de uso corrente nas aplicações paralelas para sistemas distribuídos. Mostra-se também a facilidade de programação com o uso do RTE.

O restante deste trabalho está organizado da seguinte forma: a seção 2 apresenta os Trabalhos Relacionados; a seção 3 apresenta a interface de programação paralela do sistema RTE através do seu modelo de programação e da sua arquitetura; na seção 4 há o detalhamento dos testes realizados para a análise de desempenho e os resultados alcançados com o sistema RTE; a seção 5 apresenta as conclusões obtidas neste trabalho juntamente com as sugestões de trabalhos futuros.

2. Trabalhos Relacionados

[Dolbeau et al. 2007] apresenta *Hybrid Multicore Parallel Programming (HMPP)* como um padrão para programação de computação heterogênea composta por CPUs e aceleradores de *hardware*, GPGPUs, com um conjunto de diretivas de compilação, semelhante ao OpenMP. HMPP provê o uso das linguagens C e Fortran para a implementação das aplicações. As chamadas de funções a serem executadas remotamente nas GPUs podem

[1] `http://www.arm.gov`

ser síncronas ou assíncronas. Chamadas assíncronas permitem que múltiplas funções sejam executadas simultaneamente em múltiplas GPUs heterogêneas.

A API OpenMP [OpenMP 2018] oferece, como um de seus recursos, chamadas de funções *Offloading* em dispositivos conectados em uma única placa mãe através da diretiva de compilação `#pragma omp target`. Nesta diretiva são especificadas clausulas como: `device` que cria um ambiente em um dispositivo, `map` que especifica uma área de memória mapeada no dispositivo, e `data` que especifica um ambiente de dados no dispositivo. Essa funcionalidade está presente desde a versão 4.0, inicialmente disponível somente para o acelerador de hardware Intel® XEON Phi™. As versões mais recentes, 4.5 e 5.0, disponibilizam chamadas *Offloading* também para GPGPUs.

O modelo de programação heterogênea oneAPI™ [Intel 2020] fornecido pela Intel® busca simplificar a programação de CPUs e aceleradores, tais como GPUs e FPGAs, através de uma nova linguagem de programação denominada *Data Parallel C++ (DPC++)*. Esta linguagem foi projetada para dar suporte ao paralelismo de dados e à programação heterogênea paralela. É baseada em C++ e oferece extensões para SYCL™ [Keryell et al. 2019], a qual oferece uma forma de codificação mais fácil em comparação a OpenCL. Em um programa escrito em DPC++, a execução de operações em um dispositivo (GPU ou FPGA) é paralela à execução no *host* (CPU). O *host* após solicitar a execução de operações a um dispositivo continua a sua execução, não permanecendo em espera. Assim, operações em múltiplos coprocessadores (GPU e FPGA) podem ser executadas simultaneamente.

Os trabalhos apresentados têm suas contribuições para o desenvolvimento de aplicações paralelas em um computador com CPU e coprocessadores. Alguns apresentam soluções voltadas somente para CPU e GPGPUs e provêm paralelismo de tarefas. Todas as soluções apresentadas visam oferecer uma interface única e fácil de ser aplicada na programação de aplicações com múltiplas chamadas de tarefas a serem executadas remotamente nos dispositivos heterogêneos. A Intel® oneAPI™ provê uma solução para CPU e múltiplas GPGPUs e FPGAs, voltada ao paralelismo de dados em cada dispositivo. Já o sistema RTE, cuja análise de desempenho é aqui apresentada, busca proporcionar ao desenvolvedor da aplicação o aumento do desempenho de sua execução através do paralelismo de tarefas em um ambiente de processamento que além do computador principal contendo a CPU e múltiplos coprocessadores, pode incluir múltiplos computadores remotos.

3. Sistema RTE e Suporte a Programação Paralela

Esta seção apresenta uma interface de programação paralela (RTE) que provê a execução remota de tarefas em coprocessadores e computadores remotos. Tais tarefas são executadas assincronamente, não permanecendo bloqueada a execução da sequência em que foi chamada. O sistema transfere os dados de retorno para os respectivos endereços das variáveis de retorno, assim que recebidos. Desta forma, múltiplas tarefas podem ser executadas simultaneamente. O programa do usuário, em conjunto com a interface RTE, pode utilizar as diretivas de programação paralela oferecidas pelo OpenMP, inclusive internamente nas tarefas.

São apresentadas as descrições do programa do usuário, da interface RTE e do sistema RTE.

3.1. Programa do Usuário e interface RTE

Uma aplicação é composta por um programa na linguagem C a ser executada no computador principal, aqui denominado de Programa do Usuário, e por uma biblioteca dinâmica, contendo as funções correspondentes às tarefas do programa que serão executadas remotamente em um computador ou em um coprocessador. Tal biblioteca deverá estar presente no computador remoto ou coprocessador. O programa `rted` do sistema RTE, que providenciará a execução das tarefas nos computadores remotos, deverá ser colocado em execução em cada um destes.

Um exemplo de Programa do Usuário é apresentado na Figura 1 e a seguir é descrita a interface RTE.

```
1  int main(int argc, char *argv[])
2  {
3      char *file = "busca.cfg";
4      rte_init(file);
5      strncpy(word_in_file.file,"genomic1.fna.tx",sizeof(word_in_file.file));
6      strncpy(word_in_file.word, "TCGATTCC\0", sizeof(word_in_file.word));
7      memcpy((size_t*)argin, (size_t *)&word_in_file, sizeof(word_search_t));
8      tarefa1 = rte_tsk_call(0, 0, "find_word", "libbusca-par-mmap.so", argin,sizeof(
           word_search_t), sizeof(int), (void*)&n_ocorrencias);
9      strncpy(word_in_file.file,"genomic2.fna.tx",sizeof(word_in_file.file));
10     strncpy(word_in_file.word, "TCGATTCC", sizeof(word_in_file.word));
11     memcpy((size_t*)argin, (size_t *)&word_in_file, sizeof(word_search_t));
12     tarefa2 = rte_tsk_call(0, 0, "find_word 0", "libbusca-par-mmap.so", argin, sizeof(
           word_search_t), sizeof(int), (void*)&n_ocorrencias);
13     rte_sync(tarefa1); // Synk Task 0
14     printf("genoma1 OCORRENCIAS = %d\n", n_ocorrencias);
15     rte_sync(tarefa2); // Synk Task 1
16     printf("genoma2 OCORRENCIAS = %d\n", n_ocorrencias);
17     rte_finalize();
18     return 0;
19 }
```

Figura 1. Programa a ser executado no computador principal

No exemplo de programa, apresentado na Figura 1, são chamadas 2 tarefas a serem executadas remotamente. Na linha 4 é chamada a função `rte_init(file)`, a qual inicia a execução, provendo as conexões com os computadores remotos, cujos os endereços IPs encontram-se em `file`, especificado na linha 3. Nas linha 5 a 7 é preparado o argumento de entrada para a tarefa 1, a qual é chamada na linha 8, e colocada em execução no computador remoto. O mesmo se repete nas linhas 9 a 12 para a tarefa 2. Tem-se então as execuções simultâneas nos computadores remotos das duas tarefas. Na linha 13, aguarda-se o término da tarefa 1 e o recebimento do resultado retornado, podendo-se a seguir na linha 14 utilizar o conteúdo da variável de retorno. O mesmo ocorre para a tarefa 2, nas linhas 15 e 16. Finalmente na linha 17, o sistema RTE é finalizado com a chamada da função `rte_finalize()`.

A interface RTE provê as seguintes funções:

`int rpt_init(const char *rpt_cfg_file)`: provê as conexões com os computadores remotos, cujos endereços IPs estão no arquivo de configuração e cria as *threads* que gerenciam as chamadas de tarefas remotas e a recepção dos dados de retorno. Onde `rpt_cfg_file` é nome do arquivo de configuração.

`int tsk_call(const unsigned int gst_id, const unsigned int dvc, const const unsigned int dvc, const char *tsk_name, const char *tskz_set_name, const char *argin, const size_t sz_argin, const sizez_t sz_resp, void *resp)`: solicita ao computador remoto a execução da tarefa e retorna o identificador da tarefa. Onde: `gst_id`: o identificador do dispositivo; `dvc`: tipo do dispositivo chamador; `tsk_name`: nome da tarefa; `tsk_set_name`: nome da biblioteca de tarefas; `argin`: cadeia de *bytes* contendo os valores dos argumentos da tarefa; `sz_argin`: tamanho em *bytes* de `argin`; `sz_resp`: tamanho em *bytes* do retorno da tarefa; `resp`: ponteiro contendo o endereço da variável de retorno.

`int tsk_sync(int tsk id)`: Espera pelo término da tarefa. Onde: `tsk_id`: identificador da tarefa.

`void rpt_finalize()`: Finaliza o sistema RTE, enviando aos PROCESSADORes REMOTOs a mensagem de finalização e liberando todos recursos alocados.

3.2. Arquitetura do Sistema RTE

A Figura 2 apresenta a arquitetura do Sistema RTE.

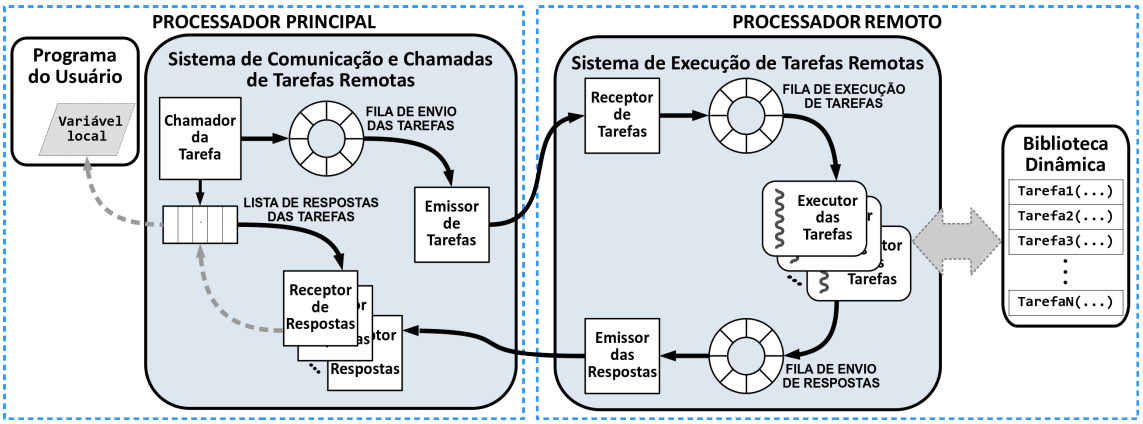

Figura 2. Arquitetura do Sistema RTE

O Computador Principal, aqui denominado PROCESSADOR PRINCIPAL, processa a aplicação que efetua chamadas de tarefas a serem executadas em PROCESSADORes REMOTOs. Cada PROCESSADOR REMOTO pode ser um computador com processadores multicore ou um computador na núvem ou um coprocessador, como por exemplo, um Intel® XEON Phi™ ou uma GPU. No sistema RTE implementado e utilizado nas análises aqui apresentadas o PROCESSADOR REMOTO pode ser um computador remoto ou um coprocessador Intel® XEON Phi™.

A aplicação do usuário é composta pelo programa, denominado como Programa do Usuário que será executado no PROCESSADOR PRINCIPAL e por Bibliotecas Dinâmicas contendo as funções correspondentes às tarefas a serem executadas nos PROCESSADORes REMOTOs.

O Sistema de Comunicação e Chamadas de Tarefas Remotas é uma biblioteca que contém o conjunto de funções da interface RTE. Através destas funções o sistema

RTE provê a comunicação, envio das requisições de execução de tarefas remotas, gerenciamento do término de uma tarefa e recepção das respostas retornadas. Essa biblioteca deverá ser ligada ao Programa do Usuário no PROCESSADOR PRINCIPAL.

No PROCESSADOR PRINCIPAL, através da função `rte_init` do Sistema de Comunicação e Chamadas de Tarefas Remotas, inicialmente são criadas duas *threads* que processam o Emissor de Tarefas e o Receptor de Respostas. O Programa do Usuário ao efetuar a chamada de uma tarefa remota chama a função `tsk_call`, que armazena as informações da requisição da execução da tarefa em Fila de Envio de Tarefas. O Emissor de Tarefas, quando houver uma requisição nesta fila, faz a leitura das informações referente à requisição e envia os dados da tarefa ao Receptor de Tarefas no PROCESSADOR REMOTO, como também, armazena o endereço da variável de retorno em Lista de Respostas de Tarefas. O Receptor De Respostas, ao receber uma mensagem com os dados de retorno enviados pelo Emissor de Respostas no PROCESSADOR REMOTO, atualiza o valor no endereço da variável de retorno armazenada na Lista de Respostas de Tarefas e atualiza o controle de término da tarefa para finalizado.

O Sistema de Execução de Tarefas Remotas é um programa (`rted`), que deve ser colocado em execução nos PROCESSADORes REMOTOs. Recebe as requisições de processamento de tarefas do PROCESSADOR PRINCIPAL, gerencia o processamento de cada tarefa e o envio dos dados de retorno para o PROCESSADOR PRINCIPAL. Inicialmente são criadas duas *threads* que processam o Receptor de Tarefa e o Emissor de Respostas. O Receptor de Tarefa aguarda a mensagem enviada pelo PROCESSADOR PRINCIPAL com os dados da requisição de uma tarefa. Uma vez recebida os escreve na Fila de Execução de Tarefas e cria uma *thread* que processa Executor de Tarefa, esta *thread* obtém os dados da requisição de uma tarefa, providencia a sua execução e o retorno de dados, armazenando-os na Fila de Envio de Respostas. A execução da tarefa é efetuada através da chamada da respectiva função contida na Biblioteca Dinâmica da aplicação, utilizando a função `dl_open`[2]. O Emissor de Respostas lê os dados de retorno de Fila de Envio de Respostas e os envia para o PROCESSADOR PRINCIPAL.

4. Avaliação de Desempenho

Para avaliar o desempenho do sistema RTE foi desenvolvida uma aplicação em Bioinformática que realiza uma contagem da quantidade de ocorrências de genes, cadeias de bases hidrogenadas, em genomas de 7 animais distintos (Baleia Orca, Bicho Preguiça, Cavalo, Leão, Pavão Azul, Peixe Betta, Peixe Boi e Tigre), cujos arquivos apresentavam tamanhos variando de 440MB a 3,1GB. Esses arquivos foram extraídos do banco de dados do *National Center for Biotechnology Information (NCBI)*[3].

4.1. Plataforma de Teste

Os testes foram realizados em um sistema distribuído composto por 5 computadores homogêneos com processador Intel Core i7 de 3,4GHz, quad-core, com 16GB de RAM, interconectados numa rede Gigabit Ethernet. Cada nó possui o seguinte configuração de *software*: sistema operacional Suse Linux 13.2 com *kernel* 3.16 e compilador gcc versão 4.8.3.

[2] `http://man7.org/linux/man-pages/man3/dlopen.3.html`
[3] `https://www.ncbi.nlm.nih.gov/genome/`

Para os testes de paralelização foram utilizadas as diretivas do OpenMP na versão do gcc 4.8.3 e na execução dos testes com MPI os programas foram compilados com o OpenMPI. Todos os executáveis utilizaram a *flag* de otimização -O3 em suas compilações.

4.2. Descrição dos Testes

Desenvolveu-se um algoritmo para realizar a busca do número de ocorrências de um gene em um genoma específico. O programa com os dados faz uma leitura do arquivo e compara a existência de nenhuma, uma ou mais ocorrências do gene neste genoma. Ao encontrar uma evidência incrementa o contador e continua até terminar a leitura do arquivo. Ao finalizar retorna o número das ocorrências encontradas ou zero caso não localize nenhuma. Para analisar o desempenho do RTE e comparação com o MPI foram desenvolvidas 5 versões desse algoritmo, apresentadas a seguir: **Programa Sequencial**: Um programa sequencial escrito em linguagem C, foi implementado para validar o comportamento da execução em uma busca genica e comparar seu comportamento com a utilização do sistema RTE. Sua execução é somente em um único *core* no Processador Principal; **Código Sequencial com RTE**: Esta versão implementa uma biblioteca dinâmica, *shared object*, para a utilização de uma chamada da tarefa de busca do gene pelo Programa do Usuário utilizando os recursos do sistema RTE. É feita, ainda, uma adaptação dos parâmetros de entrada da função que irá compor a biblioteca dinâmica para os padrões de execução do sistema RTE; **Código Paralelo com RTE**: Também foi desenvolvida uma versão para utilizar com eficiência o paralelismo de *threads* em todos os *cores* disponíveis em cada computador no sistema distribuído. Essa implementação foi reescrita com as diretivas de programação paralela do OpenMP; **Código Sequencial com MPI**: Esta versão foi escrita em linguagem C junto com MPI para realizar chamadas de tarefas em até 4 nós do sistema distribuído; **Código Paralelo com MPI**: Assim como a versão Código Paralelo com RTE, incluiu as diretivas do OpenMP ao código acima para poder avaliar os ganhos com o paralelismo no nível das *threads*.

4.2.1. BLOCO A – Análise do Custo de Comunicação do Uso do Sistema RTE

Esse bloco de testes tem como finalidade verificar o custo de comunicação no processamento de uma aplicação utilizando o sistema RTE. Os testes TesteA1 e TesteA2 fazem a mesma execução do Código Sequencial com RTE com uma única conexão, com a diferença da especificação do nó para executar a tarefa. A Tabela 1 apresenta a lista dos testes desse bloco. Esse teste foi criado para analisar o custo de uma execução local da tarefa em comparação a uma execução remota.

Tabela 1. Testes de Custo de Comunicação do Uso do Sistema RTE.

Nome do Teste	Tipo da Execução	Nó utilizado para executar tarefa
TesteA1	Código Sequencial com RTE	*localhost* (node01)
TesteA2	Código Sequencial com RTE	node02

4.2.2. BLOCO B – Análise de Desempenho no Uso do Sistema RTE

O objetivo desse bloco de testes é avaliar o comportamento do sistema RTE com a variação do número de tarefas (TesteB1 e TesteB2), como também, da quantidade de processamento em cada tarefa (TesteB3 e TesteB4). Em cada tarefa foi processado exclusivamente o Código Paralelo com RTE. As execuções são acionadas no node01, o Processador Principal, e executadas em um nó ou mais nós remotos (node02 até node05). Foram realizados 4 tipos de avaliações distintas, que são: **TesteB1**: Esse teste realiza uma busca no número de ocorrências da cadeia de gene `TCGATTCC` em genomas distintos com tamanho aproximado de 2,4 GB cada. O teste é dividido em 4 subtestes: o primeiro faz uma busca do gene em 1 genoma executando em um nó remoto, TesteB1a. O próximo, TesteB1b, faz a busca do gene em 2 genomas executando em 2 nós. O número de genomas analisados aumenta, até atingir 4 genomas no TesteB1d; **TesteB2**: Aqui é feita a contagem de genes distintos com o mesmo número de bases hidrogenadas em um genoma específico, do Peixe Boi com 3,1GB. O teste é composto por 4 subtestes: o primeiro, TesteB2a, faz a busca de 1 gene em um nó remoto, os outros subtestes mudam a quantidade de genes até alcançar a busca de 4 genes; **TesteB3**: Realiza buscas simultâneas nos genomas da Baleia Orca, do Tigre, do Cavalo e do Leão. Cada arquivo possui um tamanho aproximado de 2,4GB. Todos os nós verificam um mesmo gene em um genoma distinto, pois cada busca é realizada em um nó específico, do node02 ao node05. Cada um dos 5 subtestes varia o número de bases hidrogenadas do gene utilizado em cada execução, que são: 10 bases para o TesteB3a, 50 para o TesteB3b, 100 para o TesteB3c, 150 para o TesteB3d e 200 para o TesteB3e; **TesteB4**: Realiza a contagem das ocorrências de 4 genes específicos, com tamanhos aproximados, ao mesmo tempo em um mesmo genoma. Cada nó possui uma cópia do arquivo com o genoma e realiza a busca de um único gene. Cada subteste varia o tamanho dos arquivos com os genomas, que possuem 440 MB (TesteB4a), 1GB (TesteB4b), 1,5GB (TesteB4c), 2,4GB (TesteB4d) até 3,1 GB (TesteB4e).

4.2.3. Bloco T – Comparação do Uso do Sistema RTE com MPI

Esta seção apresenta os 4 blocos de testes usados para comparar o desempenho do MPI, um dos ambientes mais usados em HPC, em relação ao sistema RTE. Todos os testes executam uma ou mais tarefas sequenciais para localizar o gene `TCGATTCC` no genoma do Peixe Boi. Este bloco é dividido em 4 testes especificados conforme o seguinte critério: **Teste T1**: Realiza 1 execução remota do nó node01 com o nó node02; **Teste T2**: Realiza 2 execuções paralelas, uma conexão do node01 com o nó node02 e outra com o node03; **Teste T3**: Realiza 3 execuções paralelas, onde o node01 executa ao mesmo tempo buscas nos nós node02, node03 e node04; **Teste T4**: Realiza 4 execuções paralelas, onde o node01 executa ao mesmo tempo buscas nos nós node02, node03, node04 e node05.

4.3. Resultados Experimentais

Esta seção apresenta a análise dos resultados experimentais realizados e visa verificar o comportamento do sistema RTE para o conjuntos de testes aplicados. Cada teste foi executado ao menos 10 vezes e, após a eliminação de *outliers*, foram calculadas a média aritmética, o desvio padrão e o intervalo de confiança de 95% dos tempos de execução, usando a distribuição t de *Student*. Os tempos de execução são medidos em segundos. A

Tabela 2 abaixo apresenta os tempos obtidos para os testes do Bloco A. São mostrados o tempo médio das execuções, o desvio padrão e o intervalo de confiança de 95%.

Tabela 2. Resultados obtidos para os Testes do Bloco A.

Nome do Teste	Tempo de execução	Desvio padrão	Intervalo de confiança
TesteA1	7,250	0,001	[7,2493 ; 7,2508]
TesteA2	7,293	0,004	[7,2908 ; 7,2957]

Uma análise dos valores obtidos mostra que o sistema RTE apresentou um desempenho similar na execução dos testes deste bloco. A diferença entre a execução da tarefa em outro nó (node02) e da tarefa no nó local (*localhost*) foi um aumento de apenas 0,6%, causado pelo *overhead* de comunicação e criação da tarefa remota em outro nó do sistema distribuído. Desta forma, este resultado mostra que o sistema RTE apresenta um custo baixo para a execução remota de tarefas.

A Figura 3 apresenta os tempos de execução dos testes TesteB1 e TesteB2. Ambos os testes realizam a busca da ocorrência de genes em genomas armazenados em nós remotos do sistema distribuído. De uma maneira geral, podemos verificar que os tempos de execução obtidos mostram valores próximos a 2,5s. No gráfico da Figura 3a temos valores entre 2,1 a 2,56s que podem ser explicados pela heterogeneidade dos genomas analisados. Embora o gene buscado fosse o mesmo, o processo de busca apresenta valores heterogêneos devido aos genomas diferentes. Os tempos semelhantes da Figura 3b podem ser explicados pelo fato de todos os nós usarem o mesmo genoma para as buscas. Nota-se o ganho de desempenho com a execução paralela das 4 tarefas nos 4 nós remotos. O tempo de execução é praticamente o tempo da tarefa mais custosa. Se as 4 tarefas fossem executadas sequencialmente o tempo de execução seria a soma dos tempos de cada tarefa. De uma maneira geral, estes testes mostram que o sistema RTE apresenta estabilidade de execução e boa escalabilidade com o aumento do número de nós da aplicação paralelizada.

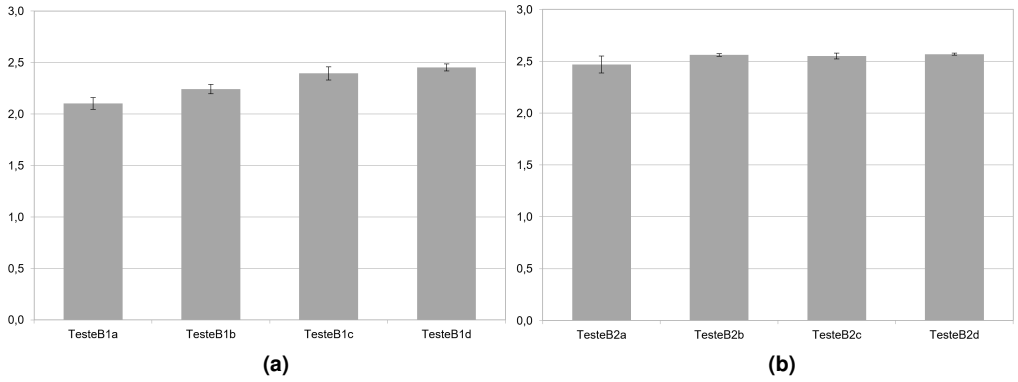

Figura 3. Desempenho do RTE para os Testes do Bloco B1 e B2.

A Figura 4 apresenta os tempos de execução dos testes TesteB3 e TesteB4. Ambos os testes visavam verificar a escalabilidade da aplicação paralela com o aumento da sequência de bases do gene (TesteB3) e com o aumento do tamanho do arquivo do genoma

(TesteB4). O gráfico da Figura 4a mostra que há pouca variação no tempo de execução em função do tamanho do gene. Como cada nó do sistema distribuído analisava um genoma diferente, a variação observada pode ser explicada pelo fato de cada genoma apresentar um número diferente de ocorrências do gene buscado em cada teste. O padrão de tempos de execução da Figura 4b mostra a escalabilidade do sistema RTE em função do tamanho do genoma. Embora o padrão de acessos de cada genoma seja diferente, uma análise dos tempos em função do tamanho do arquivo de entrada mostra um comportamento próximo do desempenho linear. Isto mostra que o sistema RTE apresenta características interessantes para uma plataforma de suporte a programação paralela.

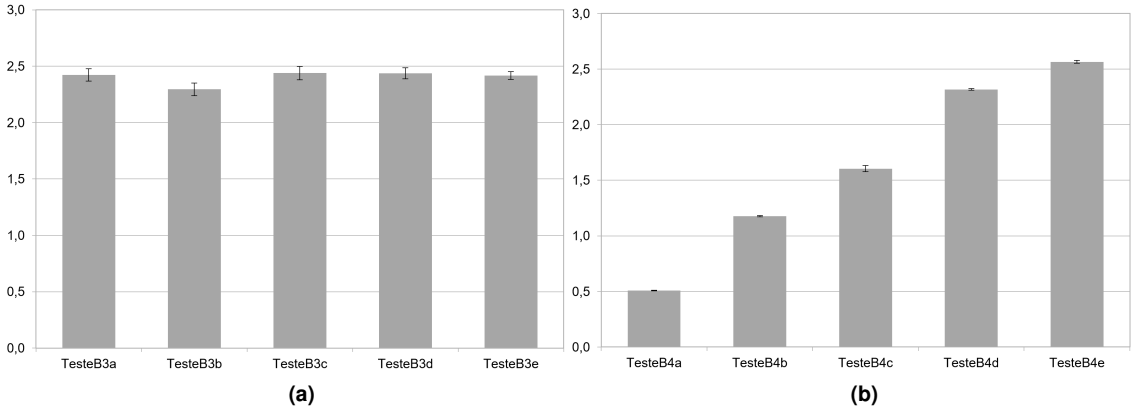

Figura 4. Desempenho do RTE para os Testes do Bloco B3 e B4.

O conjunto de testes final cujos resultados são apresentados na Figura 5 apresenta a comparação de desempenho do sistema RTE com a interface MPI para a implementação da aplicação paralela de busca em genomas. O gráfico apresentado mostra os resultados obtidos para a execução sequencial das tarefas em cada nó do sistema distribuído. Este teste mostra que os valores obtidos pelo sistema RTE são comparáveis aos apresentados pela implementação OpenMPI da interface MPI.

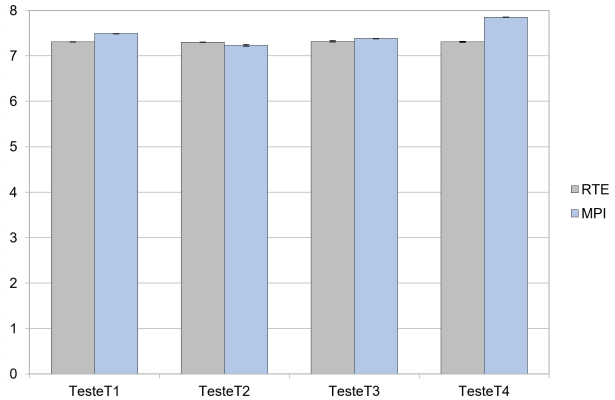

Figura 5. Comparação de desempenho entre RTE e MPI.

5. Conclusões e Trabalhos Futuros

O objetivo deste artigo foi avaliar o desempenho do sistema RTE com uma aplicação que necessitasse de alto poder computacional e compará-lo ao padrão MPI, como é apresen-

tado na seção 4. Os testes iniciais mostraram que o RTE apresenta características adequadas de estabilidade e escalabilidade com a quantidade de dados de entrada, o número de tarefas e o número de nós do sistema distribuído. Após as execuções dos testes descritos na seção 4.2 os resultados experimentais também demonstraram que o RTE possui desempenho compatível com o MPI, em execuções distribuídas em modo sequencial para cada nó, conforme é observado na seção 4.3. Com base nesses resultados, as considerações finais desse trabalho apontam na viabilidade em implementar uma solução baseada no sistema RTE que possui uma sintaxe que facilita o desenvolvimento de aplicações paralelas e possibilita ganhos de processamento equivalentes ao ambiente padrão para aplicações distribuídas.

A pesquisa, em que esse trabalho faz parte, envolvendo um modelo de programação baseado em *Bag-of-Tasks* para execuções remotas e assíncronas encontra-se em andamento e constante evolução. Inicialmente são propostos os seguintes trabalhos futuros: Implantação do modelo de programação para uma adaptação GPGPUs e outra para *Cloud Computing*; Comparação com outros modelos de programação paralela, p. ex. Intel®oneAPI™ e Execução de novos testes de desempenho.

Agradecimentos

O presente trabalho foi realizado com o apoio e a infraestrutura do *Laboratory of Architecture and High Performance Computing (LAHPC)* do Departamento de Engenharia de Computação e Sistemas Digitais (PCS) da Escola Politécnica da Universidade de São Paulo (EPUSP).

Referências

Broner, G. (2017). Supercomputing is the future of genomics research. *Genetic Engineering & Biotechnology News*, 37(3):18–19. https://www.genengnews.com/magazine/286/supercomputing-is-the-future-of-genomics-research/. Acessado em: 06/02/2020.

Dash, Y. (2019). An insight into parallel computing paradigm. In *2019 2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*, volume 1, pages 804–808. IEEE. https://ieeexplore.ieee.org/abstract/document/8993136. Acessado em: 10/02/2020.

Devarakonda, R., Prakash, G., Guntupally, K., and Kumar, J. (2019). Big federal data centers implementing fair data principles: Arm data center example. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 6033–6036. IEEE. https://ieeexplore.ieee.org/abstract/document/9006051. Acessado em: 19/02/1972.

Dolbeau, R., Bihan, S., and Bodin, F. (2007). Hmpp™: A hybrid multi-core parallel programming environment. 28. https://www.researchgate.net/publication/240064180_HMPP_A_hybrid_multi-core_parallel_programming_environment. Acessado em: 18/01/2020.

Intel (2020). Intel® oneAPI™ Programming Guide. Version Beta. https://software.intel.com/sites/default/files/oneAPIProgrammingGuide_9.pdf. Acessado em: 27/03/2020.

Jeffers, J., Reinders, J., and Sodani, A. (2016). *Intel® Xeon Phi™ Processor High Performance Programming: Knights Landing Edition*. Morgan Kaufmann.

Keryell, R., Rovatsou, M., and Howes, L. (2019). Sycl™ Specification: Sycl™ integrates OpenCL™ devices with modern C++. *Khronos Working Group*, Version 1.2.1. https://www.khronos.org/registry/SYCL/specs/sycl-1.2.1.pdf. Acessado em: 27/03/2020.

Khronos (2019). The OpenCL™ Specification. *Khronos Working Group*, Version V2.2-11. https://www.khronos.org/registry/OpenCL/specs/2.2/pdf/OpenCL_API.pdf. Acessado em: 20/01/2020.

Kirk, D. B. and Hwu, W.-m. W. (2017). *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan kaufmann, 3rd edition.

LNCC (2019). Configuração do sdumont. https://sdumont.lncc.br/machine.php?pg=machine#. Acessado em: 15/12/2019.

MPI (2015). MPI: A Message-Passing Interface Standard. *Message Passing Interface Forum*, Version 3.1. https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf. Acessado em: 11/01/2020.

NVIDIA (2019). CUDA C++ Programming Guide. *NVIDIA Coporation*, Version v10.2. https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf. Acessado em: 17/01/2020.

OpenACC (2019). The OpenACC™ Application Programming Interface. *Retrieved March*, Version 3.0. https://www.openacc.org/sites/default/files/inline-images/Specification/OpenACC.3.0.pdf. Acessado em: 23/01/2020.

OpenMP (2018). Openmp application program interface. Version 5.0. https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf. Acessado em: 06/01/20201.

Schmidt, B. and Hildebrandt, A. (2017). Next-generation sequencing: big data meets high performance computing. *Drug discovery today*, 22(4):712–717. https://www.sciencedirect.com/science/article/pii/S1359644617300582. Acessado em: 15/02/2020.

Top500 (2019). TOP500 List - November 2019. https://www.top500.org/list/2019/11/. Acessado em: 15/12/2019.

Yasuda-Masuoka, Y., Kwon, S., and Yoon, J. (2019). Foundry platform technology from high-performance to low-power for new high-performance computing (hpc) and iot era. pages 1–3. https://ieeexplore.ieee.org/abstract/document/8731116. Acessado em: 11/02/2020.