https://eventos.utfpr.edu.br//sicite/sicite2018

# Uma implementação MPI tolerante a falhas do algoritmo de ordenação paralela Quickmerge

# A fault-tolerant implementation of the Quickmerge parallel sorting algorithm

**Felipe Natã de Camargo Xavier**
felipecamargoxav@hotmail.com
Universidade Tecnológica Federal do Paraná, Toledo, Paraná, Brasil

**Edson Tavares de Camargo**
edson@utfpr.edu.br
Universidade Tecnológica Federal do Paraná, Toledo, Paraná, Brasil

## RESUMO

Um dos problemas mais frequentes em computação é a ordenação de dados. Os algoritmos de ordenação resolvem o problema da ordenação de forma sequencial ou paralela. A versão paralela se destaca por ordenar uma sequência grande de elementos em tempo de execução menor. Para tanto, o problema é dividido em partes menores e distribuído para um conjunto de nodos ou processos computacionais. Cada processo é responsável por resolver parte do problema. Este trabalho apresenta os esforços iniciais na obtenção de uma implementação tolerante a falhas do algoritmo de ordenação paralela chamado Quickmerge no modelo de memória distribuída. O algoritmo Quickmerge é uma versão paralela do algoritmo Quicksort e destaca-se por utilizar a topologia de uma hipercubo virtual para organizar os processos. A implementação foi realizada usando o padrão MPI. O MPI é o principal padrão para o desenvolvimento de aplicações paralelas e distribuídas que fazem uso do paradigma de troca de mensagens. No entanto, o padrão não tolera falhas de processos. São apresentados resultados preliminares de desempenho do algoritmo implementado e um estudo das estratégias para transformá-lo em uma versão tolerante a falhas.

**PALAVRAS-CHAVE:** Ordenação de dados. Quickmerge. Message Passing Interface (MPI). Quicksort. Tolerância a falhas.

## ABSTRACT

One of the most common problems in computing is sorting. Sorting algorithms can solve the sorting problem sequentially or parallel. The parallel version stands out by ordering a large sequence of elements in a shorter runtime. The problem is divided into smaller parts and distributed to a set of nodes or computational processes. Each process is responsible for solving part of the problem. This paper presents the initial efforts for obtaining a fault-tolerant implementation of the parallel ordering algorithm called Quickmerge in the distributed memory model. The Quickmerge algorithm is a parallel version of the Quicksort algorithm. Quickmerge uses the topology of a virtual hypercube to organize the processes. The implementation was performed using the MPI. MPI is the facto standard for the development of parallel and distributed applications that make use of the messaging passing paradigm. However, MPI does not tolerate process failures. Preliminary results of the performance of the implemented algorithm are presented. We also present strategies to transform our implementation into a fault-tolerant version using ULFM.

**KEYWORDS:** Sorting of data. Quickmerge. Message Passing Interface (MPI). Quicksort. Fault Tolerance.

**Recebido:** 31 ago. 2018.
**Aprovado:** 04 out. 2018.

**Direito autoral:**
Este trabalho está licenciado sob os termos da Licença Creative Commons-Atribuição 4.0 Internacional.

## INTRODUÇÃO

O propósito de um algoritmo de ordenação é posicionar os elementos de um conjunto de acordo com um método de organização como, por exemplo, ordem crescente, ordem alfabética, ordem cronológica, entre outros. O objetivo da ordenação é auxiliar o processo de busca de elementos em um conjunto de dados (Aaron M. Tenenbaum et al, 1995).

O algoritmo sequencial Quicksort (Hoare, 1962) é um dos algoritmos de classificação mais utilizado devido ao seu bom desempenho. A estratégia utilizada pelo algoritmo é a de divisão e conquista. Inicialmente, um elemento pivô é escolhido. Os elementos menores que o pivô são posicionados à sua esquerda e os elementos maiores que o pivô são posicionados à sua direita. Esse processo é repetido até que todos os elementos estejam ordenados.

Uma maneira de aumentar significativamente a eficiência do algoritmo é paralelizá-lo. Uma das versões paralelas do algoritmo Quicksort é o algoritmo Quickmerge (Michael J. Quinn, 1989). O Quickmerge é um algoritmo paralelo que faz uso de uma arquitetura de memória distribuída. Um sistema distribuído é um conjunto de unidades de processamento, também chamados de processos, ou nodos, os quais cooperam entre si para resolver um problema (Peter S. Pacheco, 2011). Em um sistema distribuído de memória, cada processo tem sua própria memória. Todos os processos devem se comunicar, enviando ou recebendo mensagens através da uma rede.

Um dos principais modelos de programação paralela para memória distribuída é o padrão MPI (Message Passing Interface), atualmente na sua versão 3.1 (MPI Forum, 2015). Não se conhece uma implementação do algoritmo Quickmerge em MPI. Além disso, tradicionalmente, o padrão MPI é conhecido por não lidar com falhas de processos. A falha de processo enquanto uma ordenação está ocorrendo leva à interrupção completa do algoritmo, ou seja, todo o processamento realizado é perdido.

O objetivo deste trabalho é apresentar uma implementação MPI do algoritmo Quickmerge e descrever os passos para realizar uma implementação tolerante a falhas. Para realizar a programação tolerante a falhas do Quickmerge será utilizado o padrão ULFM (User-Level Failure Mitigation) (Wesley Bland et al., 2013) proposta pelos padronizadores do MPI. A ULFM oferece um conjunto de funções de programação para recuperar a capacidade do MPI de continuar transportando suas mensagens após a ocorrência de falhas. São apresentados resultados preliminares que exibem o tempo de execução do Quickmerge ao ordenar um conjunto grande de números.

## ALGORITMO QUICKMERGE

A ordenação no algoritmo Quickmerge ocorre da forma descrita a seguir (Quadro 1). Os processos, ou nodos, responsáveis pela ordenação são organizados de acordo com a topologia de um hipercubo virtual. O hipercubo é uma importante estrutura largamente utilizada, seja como topologia de interligação e comunicação de nodos ou para a execução de algoritmos paralelos

(Parhami, 1999, p.466). Um hipercubo de dim dimensões consiste de uma rede com $2^{dim}$ nodos numerados de 0 a $2^{dim}$- 1. Ou seja, um hipercubo de 3 dimensões possui os nodos 0, 1, 2, 3, 4, 5, 6 e 7. Cada nodo é identificado pelo código binário do seu identificador. Uma ligação entre dois nodos existe se os seus códigos diferem em um bit.

Inicialmente todos os processos recebem uma lista inicial com t/n elementos, onde t é igual ao tamanho da lista original e n é igual ao o número de processos. Todos os processos ordenam sua lista inicial utilizando o Quicksort sequencial (linha 3). Então, o processo 0 escolhe $2^{dim}$- 1 elementos de sua lista inicial que serão utilizados como pivôs em todo algoritmo por todos os processos (linha 7). Após a escolha de pivôs, o processo 0 os distribui para todos os processos (linha 10). A ordenação ocorre em rodadas de ordenação (linhas 11 a 23). A cada rodada os processos ordenam localmente suas listas e trocam parte da lista entre si de acordo com um elemento pivô, explicado abaixo.

A troca de lista entre os processos ocorre aos pares, que são formados a cada rodada. Os pares de processos que trocam listas utilizam o mesmo pivô. Um novo pivô é escolhido a cada rodada (linha 13). Após a escolha do pivô, cada processo divide sua lista entre elementos maiores e elementos menores que o pivô. Todos os processos, em paralelo, enviam sua lista com elementos menores que o pivô e recebem a lista do parceiro com elementos maiores que o pivô caso seu rank for maior que o rank do parceiro (linhas 15 e 17). O procedimento é invertido caso o seu rank for menor que o rank do parceiro (linhas 18 e 20). Ao final de cada troca, as duas listas são reunidas em uma lista local ordenada utilizando o Quicksort sequencial (linha 22). Para um hipercubo de dim dimensões são necessárias dim rodadas de ordenação para concluir a classificação de todos os elementos. Ao final, um processo i, onde i < j, possuirá elementos menores ou iguais aos elementos do processo j.

A comunicação para a troca de lista entre os processos fundamenta-se no padrão Message Passing Interface (MPI). O MPI fornece um conjunto de bibliotecas que possibilitam a comunicação ponto a ponto e a comunicação coletiva entre os processos. A estrutura de dados que estabelece a comunicação entre os processos do grupo é o comunicador. Cada processo dentro de um grupo recebe um identificador, um número inteiro positivo, chamado rank. A implementação do algoritmo MPI pode ser encontrada em https://bitbucket.org/feelipeexavieer/quickmerge.git.

Quadro 1 – Pseudocódigo do algoritmo Quickmerge (processos executam os comandos em paralelo)

```
0    {dim – Dimensão do hipercubo}
1    {rank – Número que identifica o processo (0 até 2^dim − 1)}
2    {n – Número de elementos por processo}
3    Ordena n elementos iniciais utilizando o Quicksort sequencial
4    Se dim > 0 então
5        Se rank > 0 então
6            Para i = 1 até 2^dim − 1
7                splitter [i] = a [ (i x n) /2^dim ]
8            Fimpara
9        Fimse
```

```
10          O processo 0 distribui o vetor de pivôs, splitter, entre os processos
11      Para i = dim – 1 até 0
12          parceiro = rank XOR 2^i
13          index = rank AND (2^dim – 2^(dim-1)) OR 2^(dim-1)
14          Particiona-se a lista entre lista menor e lista maior  que o splitter [index]
15          Se rank > parceiro então
16              Envia lista menor para o parceiro
17              Recebe lista maior do parceiro
18          Senão
19              Envia lista maior para o parceiro
20              Recebe lista menor do parceiro
21          Fimse
22          Reune as duas listas em uma única lista ordenada
23      Fimpara
24  Fimse
```

Fonte: Autoria própria (2018), adaptado de (Michael J. Quinn, 1989).

## ESTRATÉGIA TOLERANTE A FALHAS

A ULFM é a mais recente especificação para padronizar a tolerância a falhas em MPI. A ULFM oferece suporte para continuar as operações de trocas de mensagens do MPI após falha de algum processo. O objetivo não é uma estratégia de recuperação específica, mas permitir que o desenvolvedor escolha a técnica de tolerância a falhas que melhor se adeque ao programa. Para implementar a versão tolerante a falhas do algoritmo Quickmerge, será usado uma abordagem semelhante ao usado no trabalho (Camargo, 2016). Duas abordagens são possíveis. Na primeira, sempre que a falha de um processo é detectada, um novo processo é criado automaticamente. Na segunda abordagem, um processo do hipercubo assume a responsabilidade de executar as tarefas do processo que falhou. Em ambas as abordagens, cada processo salva sua lista ao final da rodada de ordenação, procedimento esse chamado de checkpointing. Dessa forma, os elementos ordenados pelo processo falho podem ser recuperados.

## RESULTADOS

Os resultados apresentados na Tabela 1 foram obtidos a partir da implementação em MPI em um *notebook* de uso pessoal. Importante informar que o *notebook* possui apenas 1 processador e, portanto, os testes serviram apenas para verificar a corretude da implementação. A Tabela 1 apresenta a quantidade de elementos ordenados por 2 e 4 processos MPI, respectivamente. É apresentado o tempo de ordenação em segundos. Como é possível perceber, os tempo de execução para 4 processos é superior à 2 processos, isso se deve a arquitetura do computador utilizado que acaba dividindo um único processador em 2 ou 4 processos. Os resultados devem ser diferentes quando executados em uma máquina multiprocessos/multicore.

Tabela 1 – Resultados preliminares

| Quantidade de números | 2 processos MPI | 4 processos MPI |
|---|---|---|
| 262144 | 0,694s | 0,985s |
| 524288 | 0,811s | 1,321s |

| Quantidade de números | 2 processos MPI | 4 processos MPI |
|---|---|---|
| 1048576 | 3,21s | 3,5s |

Fonte: Autoria própria (2018).

## CONCLUSÃO

Este trabalho apresentou os esforços iniciais para obtenção de uma implementação MPI tolerante a falhas do algoritmo Quickmerge. São apresentadas o algoritmo implementado em MPI e as possíveis estratégias a serem empregada na sua versão tolerante a falhas. Importante frisar que não se conhece na literatura uma implementação MPI do algoritmo. Os próximos passos deste trabalho são a execução do algoritmo em um *cluster*, ou seja, uma máquina que possua vários processadores. A partir de então, o algoritmo poderá ser executado para ordenar uma sequência muito grande de elementos como, por exemplo, 1 bilhão de números inteiros. Na sequência, a implementação será adaptada para tolerar falhas de acordo com a especificação ULFM.

## REFERÊNCIAS

CAMARGO, E. T. ; DUARTE, E. P. . Uma Implementação MPI Tolerante a Falhas do Algoritmo Hyperquicksort. In: Workshop de Testes e Tolerância a Falhas, 2016, Salvador. Anais do WTF 2016 - Workshop de Testes e Tolerância a Falhas, 2016.

Hoare, C. A. R. (1962). Quicksort. The Computer Journal, 5(4):10–15.

Michael J. Quinn:Analysis and Benchmarking of Two Parallel Sorting Algorithms: Hyperquicksort and Quickmerge. BIT 29(2): 239-250 (1989).

MPI Forum (2015). Document for a standard message-passing interface 3.1. Technical report, University of Tennessee, http://www.mpi-forum.org/docs/mpi-3.1.

Pacheco, Peter S. An introduction to parallel programming. Amsterdam Boston: Morgan Kaufmann, 2011. Print.

Parhami, B. (1999). Introduction to Parallel Processing: Algorithms and Architectures. Kluwer Academic Publishers, Norwell, MA, USA.

Tanenbaum, Andrew S., and Maarten Steen. Sistemas distribuídos. São Paulo: Pearson Educación, 2008. Print.

Tenenbaum, Aaron M., Yedidyah Langsam, and Moshe Augenstein. Estruturas de dados usando C. São Paulo (SP: Pearson Makron Books, 1995. Print.

Wesley Bland, Aurelien Bouteiller, Thomas Herault, George Bosilca, Jack J. Dongarra: Post-failure recovery of MPI communication capability: Design and rationale. IJHPCA 27(3): 244-254 (2013).