UNIVERSIDADE REGIONAL INTEGRADA DO ALTO URUGUAI E DAS MISSÕES
CAMPUS DE ERECHIM
DEPARTAMENTO DE ENGENHARIAS E CIÊNCIA DA COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

NOME COMPLETO DO AUTOR

TÍTULO DO TRABALHO: SUBTÍTULO DO TRABALHO

ERECHIM - RS
2018

NOME COMPLETO DO AUTOR

TÍTULO DO TRABALHO: SUBTÍTULO DO TRABALHO

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do grau de Bacharel, Departamento de Engenharias e Ciência da Computação da Universidade Regional Integrada do Alto Uruguai e das Missões Campus de Erechim.

Orientador: Nome do orientador

ERECHIM - RS
2018

Altere este texto inserindo a dedicatória do seu trabalho.

AGRADECIMENTOS

Edite e coloque aqui os agradecimentos às pessoas e/ou instituições que contribuíram para a realização do trabalho.

É obrigatório o agradecimento às instituições de fomento à pesquisa que financiaram total ou parcialmente o trabalho, inclusive no que diz respeito à concessão de bolsas.

Eu denomino meu campo de Gestão do Conhecimento, mas você não pode gerenciar conhecimento. Ninguém pode. O que pode fazer - o que a empresa pode fazer - é gerenciar o ambiente que otimize o conhecimento.

(PRUSAK, Laurence, 1997)

RESUMO

O Resumo é um elemento obrigatório em tese, dissertação, monografia e TCC, constituído de uma seqüência de frases concisas e objetivas, fornecendo uma visão rápida e clara do conteúdo do estudo. O texto deverá conter no máximo 500 palavras e ser antecedido pela referência do estudo. Também, não deve conter citações. O resumo deve ser redigido em parágrafo único, espaçamento simples e seguido das palavras representativas do conteúdo do estudo, isto é, palavras-chave, em número de três a cinco, separadas entre si por ponto e finalizadas também por ponto. Usar o verbo na terceira pessoa do singular, com linguagem impessoal, bem como fazer uso, preferencialmente, da voz ativa. Texto contendo um único parágrafo.

Palavras-chave: Palavra. Segunda Palavra. Outra palavra.

ABSTRACT

Elemento obrigatório em tese, dissertação, monografia e TCC. É a versão do resumo em português para o idioma de divulgação internacional. Deve ser antecedido pela referência do estudo. Deve aparecer em folha distinta do resumo em língua portuguesa e seguido das palavras representativas do conteúdo do estudo, isto é, das palavras-chave. Sugere-se a elaboração do resumo (Abstract) e das palavras-chave (Keywords) em inglês; para resumos em outras línguas, que não o inglês, consultar o departamento / curso de origem.

Keywords: Word. Second Word. Another word.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
DECOM	Departamento de Computação

LISTA DE SÍMBOLOS

Γ	Letra grega Gama
λ	Comprimento de onda
$\in$	Pertence

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

Tradicionalmente é ampla a aceitação de que existem ferramentas como *Spring* e *Ruby on Rails* que oferecem soluções robustas e eficazes para o desenvolvimento de aplicativos *web*. Estas ferramentas fazem jus à sua popularidade: é exageradamente descomplicado criar e oferecer manutenção à um aplicativo que as usa, além das garantias implícitas oferecidas por simplesmente estar usando um ambiente que é testado diariamente em condições reais. Mas, atualmente, é vivido um período de diáspora, onde ninguém consegue prever qual é o mais sensato próximo passo, devido ao fato de que as correntes circunstâncias exibem extrema volatibilidade, e a abordagem monolítica adotada por estas se mostra inflexível visto que quando surge a necessidade de adaptação, é o próprio mecanismo que precisa ser alterado, e não o código que faz uso deste. Tendo tais detalhes em mente, fica evidente a urgência de inovação.

Junto com a oportunidade de mudança, surge a chance não só de aprender com a experiência obtida durante todos estes anos em que estas ferramentas se mostraram a melhor opção, como também de analisar como os processos usados nos dias de hoje podem ser aperfeiçoados para a possibilidade da criação de mecanismos mais eficientes e que apresentam mais desempenho.

Para que seja atingido o propósito descrito neste trabalho, é necessária a análise do que são consideradas boas e más práticas realizadas não só por ferramentas mas também pelas linguagens hoje disponíveis. Uma das alternativas é investigar quais são os aspectos cujos contribuem com a criação de *softwares* de qualidade, da mesma forma que outra pode ser resumida na reflexão sobre quais são as características e funcionalidades que mais tendem a introduzir falhas, para que se consiga reduzir a quantidade de ou até mesmo prevenir por completo possíveis situações de risco.

Para que se inicie este trabalho, primeiro será mencionado quais foram as tecnologias que serviram como base para a pesquisa, como por exemplo, computação paralela, que contribui com o aumento do desempenho do *software* por meio do uso de recursos como processadores com múltiplos núcleos.

Assim que uma base for estabelecida, este será sucedido com o detalhamento de como é organizada uma estrutura que, ao invés de modelar dados, modela procedimentos. Este é o ponto central do trabalho, e é o tópico cujo receberá foco.

Os esforços descritos no decorrer deste trabalho nada mais são do que um passo em direção à novas formas de como podem ser representadas ideias em forma de código-fonte e quais são os meios mais eficazes pelos quais podem ser elaborados sistemas mais robustos e que apresentam melhor desempenho, sem fazer com que a função de desenvolvimento se torne uma tarefa hercúlea para o desenvolvedor.

2 PESQUISA

Como requerimento da introdução dos conceitos propostos neste trabalho, é necessário estabelecer uma base cuja será usada para solidificar a progressão da forma como são organizadas as abstrações apresentadas. Com a intenção da elaboração um modelo de estruturação de procedimentos, primeiro é indispensável definir o modelo de como é tradicionalmente estruturado um conjunto de informações para, então, explorar as possíveis formas de aumentar seu desempenho através de paralelização.

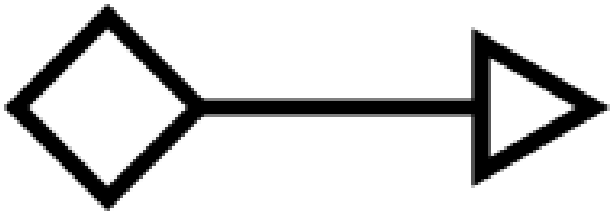
2.1 Estruturas de Dados

Em estruturas de dados convencionais, a tarefa proposta é modelar uma informação de forma que as relações entre os valores sejam claras e que o seu acesso seja eficiente. Cada estrutura de dado tem seus pontos fortes e fracos, e, por causa disso, é importante saber em quais situações cada uma delas será mais apropriada (CORMEN, 2009). Com esta concepção formalizada, apresenta-se a viabilidade do uso do mesmo conceito para modelar procedimentos ao invés de informações, e usar estes como blocos de construção para a criação de um processo maior, completamente personalizado para as necessidades que um desenvolvedor possa estar buscando atender.

2.2 *Pipelining*

Uma técnica de computação paralela muito utilizada em linguagens com suporte a CSP (*Communicating Sequential Processes*) é a de *pipelining*. Uma *pipeline* é uma série de estágios cuja entrada é a saída do anterior. Em cada um, o valor é recebido, algum tipo de processamento é realizado no mesmo e então ele é repassado para uma saída, onde será capturado pelo próximo ou pelo consumidor (AJMANI, 2014). Na figura 1 está sua representação: o losango representa a entrada, a flecha, a saída, e a linha entre os dois, a operação que é feita sobre os dados.

Figura 1 – Representação de um estágio do conceito de *pipelining*



Fonte: Ajmani (2014)

A entrada do primeiro estágio é chamada de produtor, enquanto a saída do último é conhecida como consumidor (AJMANI, 2014). O que garante o aspecto paralelo a esta técnica

é que cada estágio pode ser executado em uma *thread* própria, tendo um funcionamento similar à técnica de computação paralela conhecida por *worker pool*, sendo que cada estágio pode ser considerado um *worker*. A representação de uma pipeline é feita na figura 1, onde a flecha antecedendo o primeiro estágio simboliza o produtor, e o losango que sucede o último estágio, o consumidor.

2.3 Cuidados com a GPU

As características que mais influenciam a resistência de um processador são as que ressaltam a física dele. Por exemplo: O CPU tem IHS e o GPU não, esse já é um ponto a mais para o CPU na questão de resistência, pois o GPU não tendo IHS, terá o núcleo exposto e consequentemente será mais sensível. Só que o GPU aguenta mais calor do que o CPU,entretanto ele ainda consome mais *watts*.

Tabela 1 – Comparativo CPU X GPU

CPU	**CPU**
Tem dissipador de calor e cooler	tem dissipador de calor e cooler (a maioria)
possui IHS	Não possui IHS
é móvel	é fixo
pode consumir mais que 100 *watts*	geralmente consome mais que 300 *watts*
mede 118 mm^2 internos ou mais	mede 114 mm^2 internos ou mais

2.4 Equações

$$y = x \tag{1}$$

$$y = \frac{1}{x} \tag{2}$$

$$x_2 = \frac{5 - \sqrt{25 - 4 \times 6}}{2} = 2 \tag{3}$$

2.5 Código

```
#include
using namespace std;
int main()
{
  /* comentario */
  int n, i, a = 0, b = 1, F;
  cout << "Digite o numero de termos da sequencia de Fibonacci:
     ";
  cin >> n;
  cout << a << " " << b << " ";
  for (i = 0; i < n - 2; i++)   {
    F = a + b;
    cout << F << " ";
    a = b;
    b = F;
  } cout << endl; return 0;
}
```

2.6 Citação Longa

Segundo (CORMEN, 2009)

> O processamento modular de informações proposto pode ser usado para implementar um combinador de analisadores, técnica muito utilizada para criar programas e bibliotecas que fazem a análise de alguma informação e a transformam em uma estrutura correspondente.

3 ANÁLISE E DISCUSSÃO DOS RESULTADOS

ADICIONAR ALGORITMO

4 CONCLUSÃO

Este trabalho teve como objetivo reunir informações sobre formas de aperfeiçoar mecanismos de processamento de dados, tendo como resultado uma estrutura que modela computações ao invés de informações, que, neste, foi chamada de processo. Este processo possui um conceito simples onde etapas são encadeadas para formarem um processo maior que simplifica a forma como são desenvolvidos sistemas.

Para alcançar esta meta, foram pesquisadas práticas consideradas idiomáticas nas mais diversas linguagens, além de construções e sintaxes apontadas pela comunidade como auxílios genuínos, amplamente utilizados para elaborar *softwares* que expressam sua funcionalidade de forma concisa sem perder a segurança e expressividade. As principais influências que contribuiram para a formulação da ideia apresentada neste trabalho podem ser listadas da seguinte forma:

- A linguagem *Go* é a mais popular cuja apresenta suporte a CSP. Por causa desta característica, ela possui uma forma muito robusta de modelar processos sequenciais, devido ao fato de que não só o seu modelo de concorrência mas também a técnica usada para aproveitá-lo tem ao seu dispor várias pesquisas em torno de como fazer com que esta seja utilizada da melhor forma possível, algumas cujas foram citadas por este trabalho;
- Uma sintaxe que garante muita expressividade aos *softwares* que fazem o seu uso é o operador *pipe* presente em linguagens como *F#* e *Elixir*. Este operador é usado para fazer com que o valor ou retorno de função que antecede o operador seja usado como o primeiro argumento da função que o sucede. De forma simples, esta pode ser considerada uma maneira de como seria estruturado um processo composto por etapas;
- Outra contribuição feita por linguagens funcionais, como por exemplo *F#* e *Haskell*, é o conceito de que existe a separação entre valores e os procedimentos, mas até mesmo procedimentos podem ser modelados como valores. Isto se mostra pela (mas não está restrito a) noção de funções classificadas como *functors*, onde é possível realizar processos sobre conjuntos de informações onde um dos argumentos para este processo é outro processo, cujo retorno determina a transformação a ser realizada.

Na suposição de uma implementação concreta da estrutura apresentada por esse trabalho, a técnica de *worker pool* mostraria resultados mais estáveis, pelo fato de não haver trocas de contexto. No entanto, a técnica de *work stealing* mostraria mais desempenho em situações onde a carga a ser processada é maior, devido à realidade de que a sobrecarga gerada pela troca de contexto dificilmente seria maior ou pelo menos significativa em relação a carga, que é acompanhado pelo fato desta apresentar uma complexidade maior quando o tópico é a sua elaboração.

Assim que conceituada a estrutura proposta por este trabalho, foi apresentada uma teoria de como esta poderia ser usada de forma prática para desenvolver um servidor HTTP

versão 1.1. Foi observado que um processo modular teria a capacidade de auxiliar nas mais diversas tarefas de processamento de informações pertinentes ao contexo, e os dois exemplos mais prominentes se resumem à análise da requisição e geração de uma estrutura que representa a requisição e o tratamento desta, com o intuito de gerar uma resposta a ser enviada ao cliente.

Por mais que este trabalho consiga definir grande parte da estrutura proposta, ainda existem desafios a serem superados, e alguns deles podem ser listados como a seguir:

- Ainda não foi encontrada uma forma de lidar com erros durante o processamento de forma eficiente. Certos mecanismos para gerenciamento de erros, como por exemplo, o da linguagem *Java*, tendem a oferecer benefícios apenas para computações síncronas, e apresentam dificuldade em lidar com ambientes paralelos. Uma das alternativas seria tratar erros como se fossem valores como qualquer outro, como é feito, por exemplo, na linguagem *Rust*.
- Existe uma dificuldade em conciliar a necessidade de compartilhar informações entre etapas com o acomplamento fraco que se deseja que exista. A solução deste problema pode influenciar, inclusive, o mecanismo de gerenciamento de erros oferecido pela estrutura.

Um dos principais benefícios que é esperado ser obtido com uma implementação concreta desta estrutura é a descomplexificação de processos através de modularização. Contanto que seja possível extrair tarefas menores de um processo monolítico, é possível criar etapas e encadeá-las para, então, ter como resultado o processo original só que, desta vez, modular. Além disto, uma consequência da modularização de um processo é que se torna muito mais simples a execução de instruções paralelamente, aumentando o desempenho do procedimento. Para finalizar, é esperado, como existiria a composição de tarefas, que estas pudessem ser reutilizadas em outros sistemas, de forma que fizessem parte de diferentes processos.

REFERÃŁNCIAS

AJMANI, S. **Go Concurrency Patterns: Pipelines and cancellation**. 2014. <https://blog.golang.org/pipelines>. Acessado em 29/05/2017. Citado na página 2.

CORMEN, T. H. **Introduction to algorithms**. [S.l.]: MIT press, 2009. Citado 2 vezes nas páginas 2 e 4.