==========

Estratégias de Programação Multicore : Pipelining

Pipelining é similar a uma linha de montagem. Considere esta abordagem em *aplicações de streaming* ou qualquer outra aplicação na qual você terá que modificar um algoritmo sequencial de processamento intenso na CPU, onde cada etapa leva um tempo considerável.

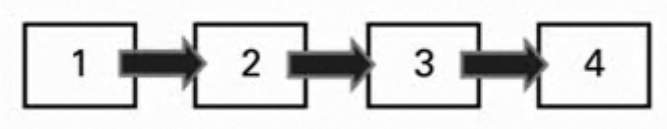


Figura 1 – Etapas sequenciais de um algoritmo

Aplicações de Streaming - A **transmissão contínua**, também conhecida por **fluxo de mídia** (bem como pelo anglicismo ***streaming***) é uma forma de distribuição digital, em oposição ao download de dados. A difusão de dados, geralmente, em uma rede é através de pacotes, e é frequentemente utilizada para distribuir conteúdo multimídia através da *Internet*. Nesta forma, as informações não são armazenadas pelo usuário em seu próprio computador. Assim não é ocupado espaço no disco rígido (*HD*), para a posterior reprodução — a não ser o arquivamento temporário no cache do sistema, ou que o usuário ativamente faça a gravação dos dados. O fluxo dos dados é recebido e a mídia é reproduzida à medida que chega ao usuário, dependendo da largura de banda seja suficiente para reproduzir os conteúdos, se não for o suficiente ocorrerá interrupções na reprodução do arquivo, por problema no *buffer*.

Isso permite que um usuário reproduza conteúdos protegidos por direitos de autor, na *Internet*, sem a violação desses direitos, similar ao rádio ou televisão aberta diferentemente do que ocorreria no caso do *download* do conteúdo, onde há o armazenamento da mídia no HD configurando-se uma cópia ilegal. A informação pode ser transmitida em diversas plataformas, como na forma *Multicast IP* ou *Broadcast*. Exemplos de serviços como esse são a YouTube, Netflix e o Spotify.

---

https://en.wikipedia.org/wiki/Pipeline_(software)

Em software, um *pipeline* consiste em uma cadeia de elementos de processamento - processos, threads, funções - dispostos de modo que a saída de cada elemento seja a entrada do próximo; Geralmente, uma certa quantidade de buffer é fornecida entre elementos consecutivos. A informação que flui nessas "tubulações" geralmente é um fluxo de bytes ou bits, e os elementos de um *pipeline* podem ser chamados de filtros; A conexão de elementos em um *pipeline* é análoga à composição de funções.

Como uma linha de montagem, cada estágio concentra-se em uma unidade de trabalho. Cada resultado passa para o próximo estágio até chegar ao último.

Para aplicar uma estratégia *pipelining* em uma CPU multicore, o algoritmo é separado em etapas que têm aproximadamente a mesma unidade de trabalho e executa cada passo com um núcleo distinto. O algoritmo pode repetir a execução em vários conjuntos de dados ou em fluxo de dados contínuo.

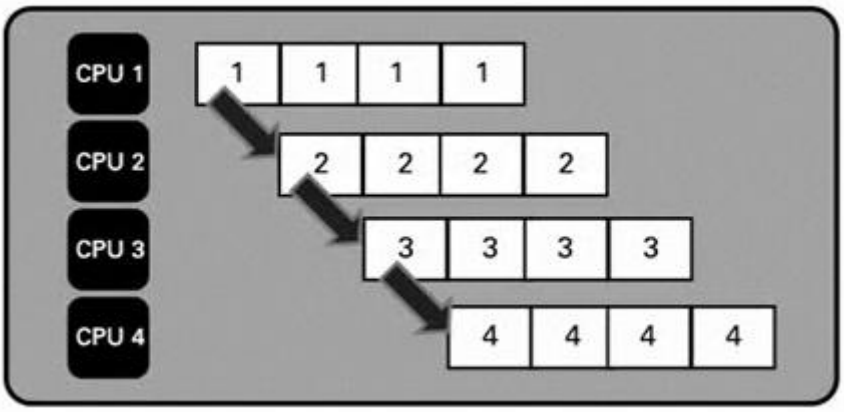


Figura 2 - Abordagem Pipeline

A chave do desmembramento do algoritmo em etapas que levem a mesma quantidade de tempo para ser executadas, uma vez que a etapa que leva mais

tempo determinará o tempo global de cada iteração. Por exemplo, se o passo 2 leva um minuto para ser executado, mas os passos 1, 3 e 4 levam 10 segundos cada, cada iteração terá uma duração de um minuto.

0. Visão geral

Quando desenvolvemos aplicações multicore, considerações especiais devem

ser feitas para aproveitar o poder dos processadores de hoje. Este documento discute o *pipelining*, uma técnica que pode ser usada para aumentar o ganho de desempenho (em processadores multicore) quando executam tarefas sequenciais inerentes.

1. Sumário

Atualmente, com processadores multicore e aplicações multithread, os programadores necessitam pensar constantemente sobre qual melhor maneira de aproveitar ao máximo as CPUs quando estão desenvolvendo suas aplicações.

Embora estruturar um código em paralelo em uma linguagem tradicional baseada em texto pode ser difícil de se programar e visualizar, ambientes de desenvolvimento gráfico, que permitem cada vez mais aos engenheiros e cientistas diminuir o tempo de desenvolvimento e executar rapidamente suas ideias.

O fato de se ter paralelismo inerente (baseada em fluxo de dados), programar aplicações multithread é tipicamente uma tarefa muito simples. Tarefas independentes do diagrama de blocos executam paralelamente sem nenhum trabalho extra necessário por parte do programador.

Mas, e as partes do código que não são independentes? Quando implementamos uma programação sequencial inerente, o que fazer para aproveitar o poder dos processadores multicore?

1. Introdução ao Pipelining

Uma técnica muito utilizada para aumentar o desempenho de tarefas sequenciais de software é o *Pipelining*. Pode se dizer que *Pipelining* é o processo que divide tarefas sequenciais em estágios distintos que podem ser executados no modelo de linha estruturada.

Considere o seguinte exemplo: suponha que você está produzindo carros em uma linha de produção automatizada.

Sua tarefa final é construir um carro completo, mas você pode separar isso em três estágios distintos: construção da armação, instalação de peças internas (como o motor), e pintura do carro quando terminado.

Levando em conta que a construção da armação, instalação das peças, e pintura levem uma hora cada parte serem concluídas, então, se você construir apenas um carro de cada vez, cada carro levará três horas para ficar pronto (veja a Figura 1 abaixo).

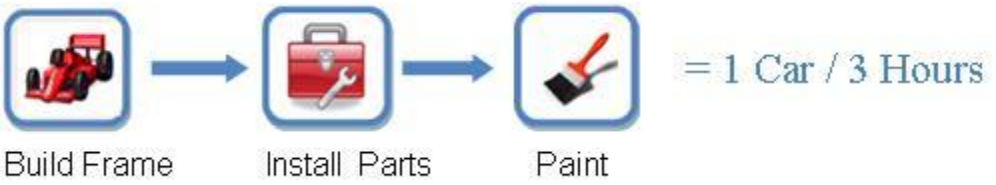


Figura 3. Nesse exemplo (sem *Pipeline*), a contrução do carro leva 3 horas para ser concluída.

Como esse processo pode ser melhorado?

Se for criada uma estação para a construção da armação, outra para a instalação das peças, e uma terceira pra pintura, agora, quando um carro estiver sendo pintado, o segundo carro pode estar instalando as peças, e o terceiro carro pode estar na construção da armação.

3. Como o *pipelining* melhora o desempenho

Embora cada carro ainda leve três horas para ser finalizado usando o novo processo, nós podemos, agora, produzir um carro a cada hora, melhor que um a cada três horas - uma melhoria de 3x no processo de produção do carro. Note que este exemplo foi bem simplificado para propósitos de demonstração.

Veja a seção de considerações importantes abaixo para mais detalhes sobre *pipelining*:

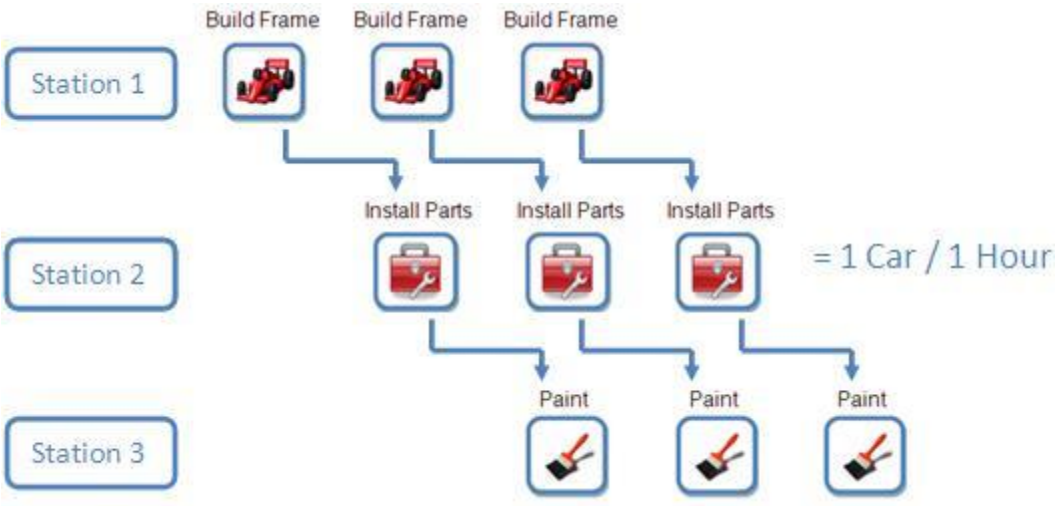


Figura 4 . Pipelining pode aumentar muito o rendimento da sua aplicação

4. *Pipelining* Básico

O mesmo conceito de *pipelining* como visto no exemplo do carro, pode ser usado em várias aplicações, onde você estará executando uma tarefa sequencial.

A seguinte ilustração mostra como uma aplicação *pipelining* pode funcionar em diferentes núcleos (cores):

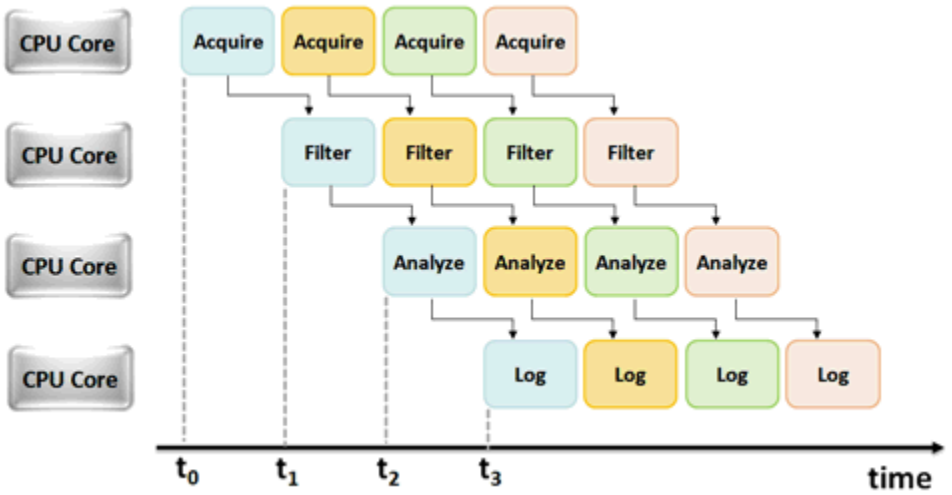


Figura 3 - Carta dos tempos para aplicação pipelining funcionando em cores diferentes.

Considerações Importantes

Quando são criadas aplicações multicore usando *pipelining*, o programador deve levar em conta diversos parâmetros importantes. Em especial, equillibrar os estágios em *pipelining* e minimizar a transferência de memória entre os cores são fatores importantes para aumentar o desempenho com o *pipelining*.

Balanceando Estágios

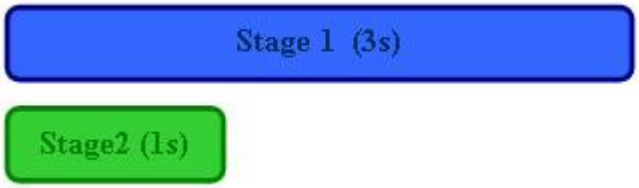
Em ambos os processos citados acima, na produção do carro e em outros exemplos, cada estágio *pipelining* assumiu o mesmo montante de tempo de execução; e podemos dizer que este exemplo de estágios de *pipelining* estavam equilibrados.

Entretanto, em aplicações reais isso raramente acontece.

Considere o diagrama a seguir: se o Estágio 1 toma três vezes o tempo do Estágio 2, então os dois estágios produzem um aumento pequeno de desempenho.

Sem Pipeline (tempo total = 4s):

Com Pipeline (tempo total = 3s):



Observação - Aumento de desempenho = 1.33x (caso não ideal para pipelining)

Para melhorar essa situação, o programador deve mover as tarefas para o Estágio 1 e o Estágio 2 até que ambos os estágios demorem aproximadamente o mesmo tempo de execução. Com um grande número de estágios no *pipeline*, essa pode ser uma tarefa difícil.