A description...

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL REI

UFSJ

**Paralelização**

Bruno Bernardes da Costa

Edson Coelho do Santos

SÃO JOÃO DEL REI, MG  
Setembro/2013

**Sumário:**

1.Introdução........................................................................................................

2. Proposta do trabalho.........................................................................................................................

3. Descrição da solução........................

4. Análise e discussões............................

5. Conclusão..........................................

6. Bibliografia....................................

**1. Introdução:**

Thread é uma forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas concorrentemente. O suporte à *thread* é fornecido pelo próprio [sistema operacional](http://pt.wikipedia.org/wiki/Sistema_operacional), no caso da linha de execução ao nível do núcleo (em [inglês](http://pt.wikipedia.org/wiki/Língua_inglesa):*Kernel-Level Thread (KLT)*), ou implementada através de uma biblioteca de uma determinada [linguagem](http://pt.wikipedia.org/wiki/Linguagem_de_programação), no caso de uma User-Level Thread (ULT).

Uma thread permite, por exemplo, que o usuário de um programa utilize uma funcionalidade do ambiente enquanto outras linhas de execução realizam outros cálculos e operações.

Em *hardwares* equipados com uma única CPU, cada thread é processada de forma aparentemente simultânea, pois a mudança entre uma thread e outra é feita de forma tão rápida que para o utilizador isso está acontecendo paralelamente. Em *hardwares* com múltiplos CPUs ou multi-cores, as threads são realizadas realmente de forma simultânea.

A linguagem C não possui nenhum recurso nativo para permitir a programação

concorrente. Contudo, este problema pode ser contornado utilizando-se uma biblioteca que provê a abstração de pthreads. A biblioteca utilizada para este trabalho foi a pthread.h.

**2. Proposta do trabalho:**

Neste trabalho deverá ser implementado e paralelizado um algoritmo de identificação de componentes e avaliar a eficiência da paralelização implementada.

O problema dos componentes é a identificação em uma imagem dos grupos de elementos contíguos na imagem. Dada uma imagem binária de dimensões n × n e n < 65536, armazenada como uma matriz de zeros e uns. Os uns representam objetos, enquanto os zeros representam espaços vagos entre os objetos. O problema dos componentes é identificar, através do assinalamento de um inteiro a cada um dos componentes. Quando o programa termina, cada pixel 1 receberá um identificador representado por um inteiro positivo. Dois pixels 1 receberão o mesmo identificador

se e somente se estão no mesmo componente. Os pixels 1 estão no mesmo componente se são ligados por uma cadeia de pixels 1. Dois pixels 1 são contíguos se são adjacentes entre si horizontal ou verticalmente.

**3. Descrição da solução:**

* 1. **Associação dos nomes dos algoritmos:**
* **Função main: tp4.c**
* **Funções auxiliares funcoes.c**

**bibliotecas**

* **stdlib.h**
* **stdio.h**
* **string.h**
* **pthread.h**

Estrutura de dados

para a implementação do algoritmo foi utilizada uma estrutura de dados chamada pmatriz,a qual consiste de informações da atual posição da matriz.

|  |  |  |
| --- | --- | --- |
| Nome da variável | Tipo de dado | Descrição |
| \*\*matriz | Inteiro | Endereço da matriz na memoria |
| m | Inteiro | Numero máximo de linhas |
| n | Inteiro | Numero máximo de colunas |
| i | Inteiro | Índice de linha da atual posição da pesquisa |
| j | Inteiro | Índice de coluna da atual posição da pesquisa |
| k | Inteiro | Numero do atual componente em pesquisa |

Implementação

O desenvolvimento de execução se da seguindo em primeiro passo criando uma matriz contendo 0 e -1 obtidos através da atual posição do arquivo multiplicada por -1,apos isso a matriz é percorrida de forma linear ate o aparecimento de uma flag -1 o que da inicio a divisão de trabalho para percorrer todo o componente,neste processo as posições adjacentes são analisadas e é disparada uma thread em cada direção (superior,inferior e laterais direita e esquerda) de forma a acelerar o a localização dos limites do componente encontrado,lembrando que estas threads não disparadas apenas quando as posições adjacentes são iguais a -1.

Para evitar o acesso simultâneo para escrita do dado na posição e manter a integridade dos dados das threads para escrita utilizamos de um semáforo(mutex) durante a faze de escrita,tornando possível apenas uma thead acessar uma devida posição por vez.

Após o termino da recurção no devido ponto de partida ocorre um join que reagrupa as threads e continua de forma sequencial a procura,lembrando que a busca se da por números -1 ao qual não inferi a procura por posições previamente modificadas pela recursão evitando assim o pior caso onde a matriz esta completa com posições -1,e caso a flag fosse 1 ocorreria uma reexecução do algoritmo n x m vezes.

**3.2 Implementações:**

O algoritmo tem formato de execução da forma:

./tp4

O algoritmo é iniciado alocando uma matriz dinamicamente com as dimensões da matriz no arquivo de entrada. A matriz é alocada com os valores da matriz do arquivo de entrada.

Foi utilizada uma estrutura de das chamada pmatriz armazenar os seguintes dados: a matriz dinâmica, o número máximo de linhas, o número máximo de colunas, a linha atual na recursão, a coluna atual na recursão e o número do grupo ao qual o elemento pode pertencer.

A criação de uma thread necessita de um identificador que é dado por: pthread\_t id. Todas as operações serão realizadas sobre esse identificador.

Para pesquisar na matriz é feito uma pesquisa sequencial na matriz, quando na pesquisa é encontrada a flag -1, quer dizer que foi encontrado um elemento de um grupo que não foi contado ainda. Assim é chamada a função recursão para ver se existem outros elementos desse grupo.

A função recursao() recebe a estrutura pmatriz e deve retornar um ponteiro do tipo void, isso é exigência quando a tread é criada. Dado a linha, a coluna atual que é encontrada a flag -1, é pesquisado em uma casa acima, abaixo, para esquerda e para a direita se existe outro elemento com flag -1, se existir este elemento é adicionado a contagem do grupo. Essa pesquisa é dada chamando novamente a função recursao(), que para este novo elementos são verificados todas as outras direções. Caso não tenha mais nenhum elemento a recursão é encerrada retornando para a posição imediatamente anterior e testando para as outras direções que não foram testadas.

Quando não existem mais elementos no grupo a ser descobertos, a função recursao() é encerrada, voltando para a pesquisa sequencial até descobrir o próximo elemento com uma flag -1.

**4. Análise e discussões:**

**4.1 Complexidade das funções:**

A função recursao() é uma função recursiva que foi implementada para percorrer a matriz em uma das quatro direções, esquerda, baixo, direita, cima. Se todo a matriz estivar preenchida será o pior caso do algoritmo de recursão já que será percorrida toda a matriz para contabilizar todos os elementos. Assim a complexidade da função recursao() tem complexidade que depende das dimensões da matriz, ou seja, se a matriz for m x n a complexidade de recursao() é O(m x n).

A pesquisa sequencial para encontrar um elemento que ainda não foi contabilizado foi implementada utilizando um loop interno que percorre nas colunas da matriz e um loop externo que percorre as linhas. O loop interno executa m vezes 3 operações de atribuições que apresentam complexidades O(1) e chama a função recursao(). Assim a complexidade do loop interno para um execução é o máximo entre essas atribuições e a recursao(), assim O(max(1;1;1;m x n)) = O(m x n). Como o loop interno é executado m vezes temos que a complexidade do loop interno é

O(m x (m x n)).

O loop externo executa n vezes o loop interno assim a complexidade total da pesquisa sequencial é O((m x n)x(m x n)) = O((m x n)²).

* 1. **Resultados:**

A description...

A description...

A description...

A description...

A description...

A description...

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Numero de linhas e colunas | Tempo em milissegundos usando 1 thread | Tempo em milissegundos usando 1 thread | Tempo em milissegundos usando 4 threads | Speed up |  |
| 10 | 0.315 | 0.315 | 0.276 | 1,1413 |  |
| 100 | 6.518 | 6.518 | 4.685 | 1,3913 |  |
| 1000 | 244.519 | 244.519 | 234.108 | 1,0445 |  |
| 10000 |  |  |  |  |  |

**5. Conclusão:**

**6. Bibliografia:**

* ZIVIANI, N. . Projeto de Algoritmos com Implementações em Pascal e C. 2a..

ed. São Paulo: Thompson Learning, 2004.