Pontificia Universidad Javeriana

Facultad De Ingeniería
Ingeniería De Sistemas

Bogotá D.C

Documentación Algoritmo de Multiplicación de Matrices

Sistemas Operativos

Juan Pablo Hernández Ceballos

John Corredor Franco

4/4/2024

Introducción

Este documento presenta en detalle el proceso de implementación del algoritmo en lenguaje C para la multiplicación de matrices. El objetivo principal es hacerlo de forma eficiente y práctica, resolviendo problemas relacionados con la programación y la concurrencia.

El algoritmo desarrollado utiliza la biblioteca ‘Pthreads para’ lograr el paralelismo, lo que permite el uso completo de los recursos del sistema y optimiza el rendimiento de la multiplicación de matrices.

Además, el algoritmo clásico de multiplicación de matrices se utiliza para adaptarse específicamente a matrices cuadradas para garantizar un funcionamiento correcto en diversas condiciones. Se detallan los pasos tomados, desde el almacenamiento en memoria hasta la multiplicación eficiente de matrices, enfocándose en la funcionalidad implementada y los procedimientos de inicialización y visualización de matrices.

Implementación

En la sección inicial del código se incluyen las siguientes bibliotecas:

#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <sys/time.h>

Estas bibliotecas proporcionan funcionalidades clave e diferentes para el desarrollo del programa, las cuales son:

La inclusión de **‘pthread.h’** permite la implementación de hilos en el código, facilitando la ejecución paralela de tareas.

‘stdio.h’ se emplea para funciones esenciales como **printf**, utilizada para mostrar mensajes en la consola y recibir entrada del usuario.

‘stdlib.h’ proporciona operaciones para la gestión de memoria, conversiones de datos y manipulación de cadenas, entre otras. Se utiliza, por ejemplo, para la función **atoi**, que convierte cadenas de caracteres a números enteros.

‘sys/time.h’ se usa para trabajar con mediciones de tiempo y temporizadores, permitiendo calcular el tiempo de ejecución de partes del código, útil para análisis de rendimiento y optimización del algoritmo de multiplicación de matrices.

A continuación, se procede con la definición de la reserva de memoria de la siguiente manera:

#define RESERVA (1024*128*64*8)

Luego, se genera un bloque de memoria estática destinado a almacenar matrices adicionales:

static double MEM_CHUNK[RESERVA];

La estructura, datosMM organiza los datos para la multiplicación de matrices. Contiene:

1. N: Tamaño de las matrices a multiplicar.
2. mA, mB, mC: Punteros a las matrices de entrada y salida. Permite pasar estos datos de manera organizada a la función de multiplicación de matrices.

```
struct datosMM{
  int N;
  double* mA;
  double* mB;
  double* mC;
};
```

Funciones

- **inicializarMatrices**

Inicializa las matrices **'mA'**, **'mB'** y **'mC'** con valores predefinidos. Recibe como parámetros el tamaño de las matrices (**'n'**) y punteros a las matrices **'mA'**, **'mB'** y **'mC'**. Utiliza un bucle para recorrer las matrices y asignarles valores basados en el índice del bucle.

```
void inicializarMatrices(int n, double* mA, double* mB, double* mC){
  //Inicializacion de matrices
  for (int i = 0; i < n*n; i++) {
    mA[i] = i*1.1;
    mB[i] = i*2.2;
    mC[i] = i;
  }
}
```

- **imprimirMatriz**

Recibe como parámetros el tamaño de la matriz ('n') y un puntero a la matriz 'm'. Utiliza un bucle para recorrer la matriz y mostrar sus elementos en filas y columnas. Además, imprime un separador al final para mejorar la legibilidad.

```
void imprimirMatriz(int n, double* m){
  //Impresion de matriz
  if(n<9){
     for (int i = 0; i < n*n; i++) {
       if(i%n == 0){printf("\n");}
       printf("%f ", m[i]);
     }
```

```
printf("\n*****************************************************\n");
  }else{

printf("\n*****************************************************\n");
  }

}
```

- **multiplicacionMatriz**

Recibe una estructura datosMM que contiene las matrices de entrada (mA y mB) y la matriz de salida (mC). Utiliza tres punteros (pA, pB y sumaTemp) para recorrer las matrices y realizar la multiplicación elemento por elemento. Al finalizar, llama a la función imprimirMatriz para mostrar la matriz resultante en la consola.

Se verifica la cantidad de argumentos ingresados al programa mediante la variable **'argc'**, que indica el número de argumentos en la línea de comandos. Si la cantidad de argumentos es menor que 3, se muestra un mensaje indicando que no se ingresaron suficientes elementos y se termina la ejecución del programa con un código de error.

Además, se verifica que los argumentos ingresados sean mayores que 0. Si alguno de los argumentos es menor o igual a 0, se muestra un mensaje indicando que los elementos son inválidos y se termina la ejecución del programa con un código de error

```
if( argc < 3){
   printf("No se ingresaron suficientes elementos\n");
   return 1;
  }else if(atoi(argv[1]) <= 0 && atoi(argv[2]) <= 0){
   printf("Los elementos son menores o iguales a 0\n");
   return 1;
  }
```

Se utilizan las funciones **'atoi'** para convertir los argumentos de entrada a enteros. Estos enteros representan el tamaño de la matriz (**'N'**) y la cantidad de hilos (**'H'**).

Se declaran punteros para las matrices **'mA'**, **'mB'** y **'mC'**, que serán utilizadas en la multiplicación de matrices. Estos punteros se asignan para que apunten a los bloques de memoria reservados previamente en **'MEM_CHUNK'**, que se divide en tres partes: una para la matriz **'mA'**, otra para **'mB'** y otra para **'mC'**.

La creación de punteros y la asignación de memoria se realizan para preparar el espacio necesario para almacenar las matrices que se multiplicarán en el siguiente paso del programa.

```
// Creacion de punteros para las matrices
  int N = atoi(argv[1]); // Tam matriz
  int H = atoi(argv[2]); // Tam Hilos
  pthread_t hilos[H];
```

```
// Declaración de punteros para referenciar diferentes espacios de memoria
  double *mA, *mB, *mC;
  // Definicion de los espacios de memoria a los que van a hacer referencia los punteros.
  //los punteros.
  mA = MEM_CHUNK;
  mB= mA + ( N*N );
  mC= mB + ( N*N );
```

Se llama a la función **'inicializarMatrices'** para asignar valores iniciales a las matrices **'mA'**, **'mB'** y **'mC'**.

```
//Inicializacion de matrices.
  inicializarMatrices(N, mA, mB, mC);
```

Se imprimen las matrices **'mA'** y **'mB'** en la consola para visualizar sus contenidos.

```
//Impresion de cada una de las matrices.
  printf("Matriz A\n");
  imprimirMatriz(N, mA);
  printf("Matriz B\n");
  imprimirMatriz(N, mB);
  // printf("Matriz C\n");
  // imprimirMatriz(N, mC);
```

Se instancia una estructura **'datosMM'** (**'dataM'**) que contiene el tamaño de las matrices y los punteros a las matrices de entrada y salida. Luego, se utiliza un bucle para crear hilos y se invoca la función **'multiplicacionMatriz'** para realizar la multiplicación de matrices de forma paralela.

```
 struct datosMM dataM;
  dataM.N = N;
  dataM.mA= mA;
  dataM.mB = mB;
  dataM.mC = mC;

  printf("Matriz Producto mA * mB\n");

  for(int h=0; h<H; h++){
    pthread_create(&hilos[h], NULL, multiplicacionMatriz(dataM), &h);
  }

  printf("\nFin del programa\n");
  return 0;
}
```

Conclusiones

Este código proporciona una solución eficiente y escalable para la multiplicación de matrices en C utilizando el paralelismo en la biblioteca Pthreads. Al dividir el trabajo en varios subprocesos, puede aprovechar al máximo los recursos del sistema y acelerar significativamente el proceso informático. Además, se divide en tres versiones con acciones específicas para facilitar el mantenimiento, la depuración y la optimización del código.