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

15/4/2024

Introducción

Este documento aborda los avances y modificaciones esenciales en la implementación del algoritmo en lenguaje C para la multiplicación de matrices. La principal innovación de esta revisión se centra en la incorporación de estrategias de paralelismo mediante la biblioteca 'Pthreads'. Esta integración persigue potenciar la eficiencia computacional del algoritmo, logrando una distribución más equitativa de la carga de trabajo y, en consecuencia, un rendimiento optimizado.

En la versión actualizada del código, se ha introducido un enfoque avanzado para la gestión de hilos, mejorando la asignación de tareas y evitando cuellos de botella en ejecuciones secuenciales. Esta mejora es particularmente beneficiosa en sistemas con múltiples núcleos de procesamiento, donde la ejecución paralela puede traducirse en notables mejoras en la utilización de recursos y en la velocidad de procesamiento.

Adicionalmente, se ha adaptado el algoritmo clásico de multiplicación de matrices para operar específicamente con matrices cuadradas. Esta adaptación busca asegurar un rendimiento óptimo y coherente en diversos contextos y escenarios de aplicación. El ajuste realizado se enfoca en la optimización del uso de recursos y en la claridad del código, facilitando su mantenimiento y posibles actualizaciones.

Implementación

En la sección inicial del código, se incluyen las bibliotecas necesarias para el desarrollo del programa, las cuales facilitan la implementación de hilos, la gestión de memoria, la manipulación de cadenas y las mediciones de tiempo.

```
#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.

Estas bibliotecas son esenciales para las funcionalidades clave del programa, incluida la ejecución paralela de tareas y la medición del tiempo de ejecución para análisis de rendimiento.

A continuación, se presenta la reserva de memoria y la inicialización de los punteros para las matrices, tal como se realizaba en la versión anterior del código.

```
#define RESERVA (1024*128*64*8)
static double MEM_CHUNK[RESERVA];
```

Se ha añadido una nueva estructura **'datosMM'** que incluye un nuevo campo **'idH'**, permitiendo asignar una identidad única a cada hilo y distribuir la carga de trabajo de manera más eficiente.

```
struct datosMM{
  int N;                  //Dimensión matriz (NxN)
  int H;                  //Número de hilos
  int idH;        //Identidad de hilos
};
```

Funciones

- **inicializarMatrices & imprimirMatriz**

Se mantienen las funciones inicializarMatrices e imprimirMatriz, que se encargan de la inicialización de las matrices y la visualización de sus contenidos, respectivamente.

```
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;
  }
}

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**

La función **'multiplicacionMatriz'** recibe una estructura **'datosMM'** que contiene las matrices de entrada ('mA' y 'mB') y la matriz de salida (**'mC'**). Utiliza punteros para recorrer las matrices y realizar la multiplicación elemento por elemento.

Se calculan segmentos de trabajo para cada hilo, lo que permite dividir la tarea entre diferentes hilos y realizar la multiplicación de manera paralela. Cada hilo trabaja en una porción diferente de las matrices, optimizando así el proceso de multiplicación. Al finalizar, los resultados se almacenan en la matriz **'mC'**.

Antes de ejecutar la multiplicación, se verifican los argumentos ingresados al programa para asegurarse de que se proporcionen suficientes elementos y que sean válidos. Si no se cumplen estas condiciones, se muestra un mensaje de error y se termina la ejecución del programa con un código de error correspondiente.

```
void *multiplicacionMatriz(void *argMM){
  struct datosMM *intValor = (struct datosMM *)argMM;
  int n = intValor->N;
  int h = intValor->H;
  int idH = intValor->idH;

  int ini = idH * (n / h);
  int fin = (idH == h-1) ? n : (idH + 1) * (n / h);

  printf("Hilos= %d ; IdHilo= %d ; ini=%d; fin=%d \n", h, idH, ini, fin);

  for(int i=ini; i<fin; i++){
    for(int j=0; j<n; j++){
      double sumaTemp, *pA, *pB;
      sumaTemp = 0.0;
      pA = mA + i*n;
      pB = mB + j;
      for(int k=0; k<n; k++, pA++, pB+=n){
        sumaTemp += *pA * *pB;
      }
      mC[j+i*n] = sumaTemp;
    }
  }

  pthread_exit(NULL);
};
```

La función **‘main’** ha sido adaptada para integrar estos cambios, gestionando la creación y finalización de hilos de manera estructurada y eficiente. Se ha introducido un identificador para cada hilo, facilitando la gestión del paralelismo.

```
pthread_t hilos[H];
  int ids[H];  // Array para guardar los identificadores de los hilos
```

Además, se ha añadido un ciclo para gestionar y ordenar la creación de hilos, asegurando una distribución adecuada de la tarea entre los hilos y facilitando el paralelismo.

```
for(int h=0; h<H; h++){
   struct datosMM *valoresMM = (struct datosMM *)malloc(sizeof(struct datosMM));
   valoresMM->N = N;
   valoresMM->H = H;
   valoresMM->idH = h;
   pthread_create(&hilos[h],NULL,multiplicacionMatriz,valoresMM);
   ids[h] = h;
  }

  // Ordenar los identificadores de los hilos
  for (int i = 0; i < H - 1; i++) {
   for (int j = i + 1; j < H; j++) {
    if (ids[i] > ids[j]) {
     int temp = ids[i];
     ids[i] = ids[j];
     ids[j] = temp;
    }
   }
  }
```

Conclusiones

La implementación actual del algoritmo de multiplicación de matrices ofrece una solución más eficiente y escalable gracias a la incorporación de estrategias de paralelismo. Estos cambios permiten aprovechar al máximo los recursos del sistema y acelerar el proceso computacional, lo que se traduce en un mejor rendimiento y eficiencia. Con la división del trabajo en varios subprocesos y la optimización del algoritmo, se establece una base sólida para futuras mejoras y adaptaciones del código.

Para obtener más detalles y acceder al código completo, puedes visitar:
https://github.com/Nauj93x/S.O.Concurrencia