Day6

Table of Contents

1. Agenda

- Manual Reduction
- Reduction
- Array Addition
- Measuring time
- Error checking
- Dealing with larger data in a parallel program

2. Perform manual reduction on your data

```
#include<stdio.h>
#include<omp.h>
#define N 1000000000
#define T 13
int main(){
    int chunksize = N / T;
    long long sum[T];
    #pragma omp parallel num_threads(T)
    {
        int tid = omp_get_thread_num();
        long long localsum = 0;
        int start = tid * chunksize;
        int end = start + chunksize;
        if(tid == T - 1) end = N;
        for(int i = start; i < end; i++){
            localsum += i + 1;
        }
        sum[tid] = localsum;
    }

    long long totalSum = 0;
    for(int i = 0; i < T; i++) totalSum += sum[i];

    printf("Calculated sum = %lld\n", totalSum);
    long long expectedSum = (N * ((N + 1) * 1L) / 2);
    printf("Expected sum = %lld\n", expectedSum);
    if(totalSum == expectedSum){
        printf("_____Passed_____\n");
    }
    else printf("_____Failed_____\n");

    return 0;
}
```

```
gcc manualReduction.c -fopenmp -o manualReduction.out
```

```
./manualReduction.out
```

```
Calculated sum = 500000000500000000
Expected sum = 500000000500000000
_____Passed_____
```

3. conversion1

```
#include<stdio.h>
#include<omp.h>
#define N 1000000000
#define T 13
```

```
int main(){
    long long sum[T];
    #pragma omp parallel num_threads(T)
    {
        int tid = omp_get_thread_num();
        long long localsum = 0;
        #pragma omp for
        for(int i = 0; i < N; i++){
            localsum += i + 1;
        }
        sum[tid] = localsum;
    }

    long long totalSum = 0;
    for(int i = 0; i < T; i++) totalSum += sum[i];

    printf("Calculated sum = %lld\n", totalSum);
    long long expectedSum = (N * ((N + 1) * 1L) / 2);
    printf("Expected sum = %lld\n", expectedSum);
    if(totalSum == expectedSum){
        printf("_____Passed_____\n");
    }
    else printf("_____Failed_____\n");

    return 0;
}
```

```
gcc manualReduction1.c -fopenmp -o manualReduction1.out
```

```
./manualReduction1.out
```

```
Calculated sum = 500000000500000000
Expected sum = 500000000500000000
_____Passed_____
```

4. conversion2

```
#include<stdio.h>
#include<omp.h>
```

```
#define N 1000000000
#define T 13
int main(){
    long long sum = 0;
    #pragma omp parallel for reduction(+ : sum) num_threads(T)
    for(int i = 0; i < N; i++){
        sum += i + 1;
    }

    printf("Calculated sum = %lld\n", sum);
    long long expectedSum = (N * ((N + 1) * 1L) / 2);
    printf("Expected sum = %lld\n", expectedSum);
    if(sum == expectedSum){
        printf("_____Passed_____\n");
    }
    else printf("_____Failed_____\n");

    return 0;
}
```

```
gcc reduction.c -fopenmp -o reduction.out
```

```
./reduction.out
```

```
Calculated sum = 500000000500000000
Expected sum = 500000000500000000
_____Passed_____
```

5. Measuring Time

```
#include<stdio.h>
#include<omp.h>
#define N 1000000000
#define T 13

int main(){
    long long sum = 0;
    double parallelTime, serialTime;
    double startTime = omp_get_wtime();
    #pragma omp parallel for reduction(+ : sum) num_threads(T)
    for(int i = 0; i < N; i++){
        sum += i + 1;
    }
    double endTime = omp_get_wtime();
    parallelTime = endTime - startTime;

    startTime = omp_get_wtime();
    long long serialsum = 0;
    for(int i = 0; i < N; i++){
        serialsum += i + 1;
    }
    endTime = omp_get_wtime();
    serialTime = endTime - startTime;
    printf("Calculated sum = %lld\n", sum);
    printf("Time taken by parallel = %lf\n", parallelTime);
    printf("Time taken by serial = %lf\n", serialTime);
    long long expectedSum = (N * ((N + 1) * 1L) / 2);
    printf("Expected sum = %lld\n", expectedSum);
    if(sum == expectedSum){
        printf("_____Passed_____\n");
    }
    else printf("_____Failed_____\n");

    return 0;
}
```

```
gcc measuringTime.c -fopenmp -o measuringTime.out
```

```
./measuringTime.out
```

```
Calculated sum = 500000000500000000
Time taken by parallel = 0.323162
Time taken by serial = 2.913337
Expected sum = 500000000500000000
_____Passed_____
```

6. Array Addition

```c
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>
#define N 100000000
#define T 13

int main(){
    int *a, *b, *c, *cs;
    a = (int *) malloc(sizeof(int) * N);
    b = (int *) malloc(sizeof(int) * N);
    c = (int *) malloc(sizeof(int) * N);
    cs = (int *) malloc(sizeof(int) * N);
    for(int i = 0; i < N; i++){
        a[i] = i + 1;
        b[i] = i + 1;
        c[i] = 0;
        cs[i] = 0;
    }

    double startTime = omp_get_wtime();
    #pragma omp parallel for num_threads(T)
    for(int i = 0; i < N; i++){
        c[i] = a[i] + b[i];
    }
    double endTime = omp_get_wtime();
    double parallelTime = endTime - startTime;

    startTime = omp_get_wtime();
    for(int i = 0; i < N; i++){
        cs[i] = a[i] + b[i];
    }
    endTime = omp_get_wtime();
    double serialTime = endTime - startTime;
    for(int i = N - 5; i < N; i++){
        printf("%d ", c[i]);
    }
    printf("\nSerial time = %lf\n", serialTime);
    printf("\nParallel time = %lf\n", parallelTime);

    free(a);
    free(b);
    free(c);
    free(cs);

    return 0;
}
```

```
gcc arrayAddition.c -fopenmp -o arrayAddition.out
```

```
./arrayAddition.out
```

```
199999992 199999994 199999996 199999998 200000000
Serial time = 0.213609

Parallel time = 0.061018
```

7. Serial and parallel code in same file _OPENMP

```
#include<stdio.h>
#ifdef _OPENMP
#include<omp.h>
#endif
#include<stdlib.h>
#define N 100000000
#define T 13

int main(){
    int *a, *b, *c, *cs;
    a = (int *) malloc(sizeof(int) * N);
    b = (int *) malloc(sizeof(int) * N);
    c = (int *) malloc(sizeof(int) * N);
    cs = (int *) malloc(sizeof(int) * N);
    for(int i = 0; i < N; i++){
        a[i] = i + 1;
        b[i] = i + 1;
        c[i] = 0;
        cs[i] = 0;
    }

    #ifdef _OPENMP
    omp_set_num_threads(T);
    #endif

    #pragma omp parallel for
    for(int i = 0; i < N; i++){
        c[i] = a[i] + b[i];
    }

    for(int i = N - 5; i < N; i++){
        printf("%d ", c[i]);
    }

    free(a);
    free(b);
    free(c);
    free(cs);

    return 0;
}
```

```
gcc errorChecking.c -fopenmp -o errorChecking.out
```

```
./errorChecking.out
```

```
199999992 199999994 199999996 199999998 200000000
```

8. Perform matrix addition

Author: Abhishek Raj
Created: 2024-12-18 Wed 17:16