Parallel Computing

Matrix Multiplication

By
Daniel Weitman

Background

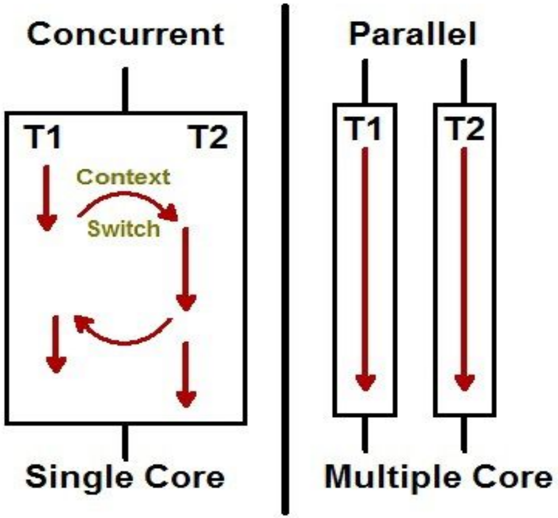
Parallel computing is a method for <u>optimizing the processing power</u> of a system

- Parallel systems utilize multiple processors/cores within the same hardware or share processors/cores over a network

- Parallel algorithms require overhead to initiate resulting in a <u>worse performance for small inputs</u>

Background

- Concurrency vs Parallelism
  - Concurrency is multiple processes are loaded into memory and compute based on scheduler
  - Parallelism is multiple processes/threads computing simultaneously
- Threads
  - Threads split a large program into several discrete subtasks
  - Each processor can independently work on a subtask
  - Run separately but share address space



Thread Example
Consider GPA.cpp
Thread1 (T1) - Finds the average GPA
Thread2 (T2) - Sorts the GPAs
Both threads share set<double> gpa;

Flynn’s Taxonomy

- SISD - Single instruction, Single data
  - Serial computations, oldest type of computer system
  - One piece of data follows a single instruction stream
- SIMD - Single instruction, Multiple data
  - Parallel computations, majority of modern PCs
  - Several chunks of data follow a single instruction stream
- MISD - Multiple instruction, Single data
  - Parallel computations, rare system as applications are highly limited
  - One piece of data is feed into several different instruction streams
- MIMD - Multiple instruction, Multiple data
  - Parallel computations, modern day supercomputers
  - Several data components are fed into multiple disjoint instruction streams

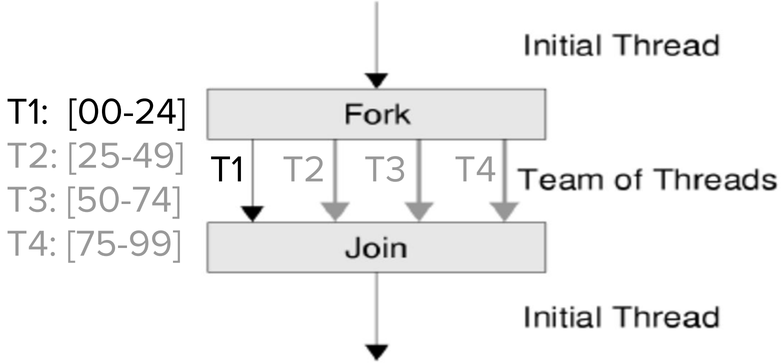
OpenMP - Overview

- **OpenMP** stands for Open Multi-Processing
- An API that simplifies the process of parallelizing algorithms
- Compatible with C/C++/Fortran
- Industry standard
- OpenMP is comprised of
    - Compiler directives
    - Library functions
    - Environment variables

OpenMP - Compiler Directives

- **#include <omp.h>**
- **#pragma omp parallel{...}** - Compiler directive marking the start of a parallel section
- **#pragma omp for** - Tells the compiler the following for loop is going to be parallelized

```
int array[100];
#pragma omp parallel for
for(i = 0; i < 100; i++){
        array[i] = calculation(i);
}
```

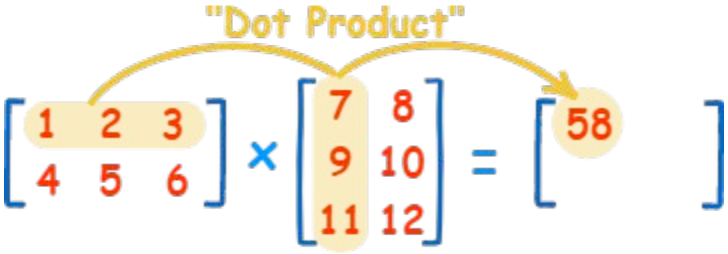


OpenMP - Library Functions & Environment Variables

- omp_set_num_threads(n);
  - Allocate up to n available threads to work on the current process
- omp_get_wtime();
  - Returns the time elapsed from an arbitrary point earlier in time
- omp_get_num_threads();
  - Returns an integer of the threads currently allocated
- omp_get_thread_ num();
  - Returns a number for identifying the thread

Matrix Multiplication

A: 2x3   B: 3x2   C: 2x2   3=3



C: $\begin{vmatrix} 58 & 64 \\ 139 & 154 \end{vmatrix}$

Matrix Multiplication

- Assuming all matrices are **nxn** in dimensions
- Uses nested for loops
- Time Complexity: $\mathbf{O(n^3)}$
    - Certain more complex algorithms can improve the time complexity
    - Optimal algorithm is still unknown
- Real-world applications
    - Computer Graphics
    - Linear Algebra
    - Physics
    - Eigenvectors (Google page ranks)

Matrix Multiplication

```
#pragma omp parallel shared(product)
       {
           #pragma omp for schedule (static, chunk)
               for(int i = 0; i < product.rows; i++){
                   for(int j = 0; j < product.columns; j++){
                       product.setElement(0, i, j);
                       for(int k = 0; k < columns; k++){
                           product.setElement
                           ((product.mat[i][j]+=(mat[i][k]*obj.mat[k][j])), i, j);
                       }
                   }
               }//End of matrix multiplication
       }//End of parallel section
```

Time analysis

- **n:** Number of rows/columns
- **S:** Time serially computed
- **#:** Time computed using # threads

n	S	2	4	8	16
10	0.000032	0.000164	0.000110	0.000847	0.000961
100	0.025608	0.015122	0.008320	0.005651	0.007415
1000	19.140239	10.315732	5.137203	3.840751	2.391150
2000	182.503947	95.196258	48.192473	32.737455	22.871749
3000	723.096576	297.284493	135.146529	71.187891	52.004610