1. OpenMP를 이용하여 palindrome 단어를 병렬적으로 찾는 프로그램을 작성하였다. 먼저 입력 파일을 읽어 사전의 내용을 배열에 저장한다. 그리고 OpenMP를 이용하여 여러 개의 쓰레드가 각각 palindrome 단어를 찾는다. 각 쓰레드는work sharing을 통하여 배열에 병렬적으로 접근하며 palindrome 단어를 찾았으면 result.txt에 써서 저장한다. 이 프로그램을 쓰레드 개수를 바꿔가며 성능을 비교하였다.

위의 그래프는 쓰레드의 개수에 대하여 수행시간을 나타낸 그래프로 각 쓰레드 개수당 5번씩 실행한 결과의 평균값을 보여준다. 전체 단어의 수는 255143개이다. 쓰레드가 늘어날수록 수행 시간에 줄어드는 것을 볼 수 있다. 그러나 10개의 쓰레드에 다가갈수록 감소폭이 줄어드는 것도 확인할 수 있다. 이러한 원인으로는 찾은 결과를 파일에 쓰는 시간으로 인하여 생기는 오버헤드로 생각된다.

2. 다양한 방법을 통하여 행렬을 곱셈하는 프로그램을 작성하였다. 먼저 가장 일반적인 방법인 Host의 CPU에서 순차적으로 곱하는 프로그램을 작성하였으며 다음으로 이를 OpenMp를 이용하여 병렬적으로 처리하는 프로그램을 작성하였다. 마지막으로 CUDA를 이용하여 GPU에서 병렬적으로 연산을 수행하는 프로그램을 작성하였다. CUDA를 이용할 때는 2차원 행렬을 사용할 수 없어서 1차원 행렬로 배열을 저장하여 사용하였다. 이 프로그램들의 성능을 비교하면 아래와 같다.

위의 그래프는 각각의 프로그램별 행렬 곱셈에 걸린 시간을 나타낸 그래프로 각 프로그램당 5번씩 실행한 결과의 평균을 나타낸다. 행렬의 크기는 1024x1024이며 각 원소는 임의로 생성된 숫자이다. 일반적인 방법으로 host cpu에서 행렬을 곱셈을 했을 때는 약 7.5초 정도의 시간이 소요되었다. 이를 OpenMP를 이용하여 병렬적으로 처리했을 경우 약 1.5초 정도로 소요 시간이 줄어들었다. CUDA를 이용하여 행렬을 곱했을 때는 약 0.0025초로 매우 적은 시간에 수행이 완료되었다.

3. Reduction 알고리즘을 이용하여 배열의 최대값을 찾는 프로그램을 작성하였다. 2번과 마찬가지로 가장 간단한 방법인 host cpu에서 순차적으로 배열을 탐색하여 최대값을 찾는 프로그램을 먼저 작성하고 이를 CUDA를 이용하여 최대값을 찾는 프로그램을 작성하였다. 또한 이 프로그램을 수정하여 path divergency가 발생하게 만들었다. 이 프로그램들의 성능을 비교하면 아래와 같다.

위의 그래프는 각 프로그램에 대한 수행시간을 나타낸 그래프로 각각 5번씩 실행한 결과의 평균값이다. 각 프로그램은 10000개의 임의로 생성된 숫자의 배열에서 최대값을 찾는다. 그래프를 보면 CPU에서 순차적으로 배열을 탐색하여 최대값을 찾는데 약 0.06msec가 걸렸지만 CUDA를 이용했을 경우 0.035msec로 줄어든 것을 볼 수 있다. 그렇지만 path divergency가 발생할 경우 0.05msec로 다시 늘어나는 것을 확인할 수 있다.