분산프로그래밍 과제 3

By 20141500 권태국

가. OpenMP Programming

1. Balanced Binary Search Tree (STL set)을 사용하여 구현했을 때,

단위 : sec					
Serial	**0.19035**				
thread	1	2	4	8	16
Parallel	0.2077	0.172063	0.186676	0.183619	0.188658

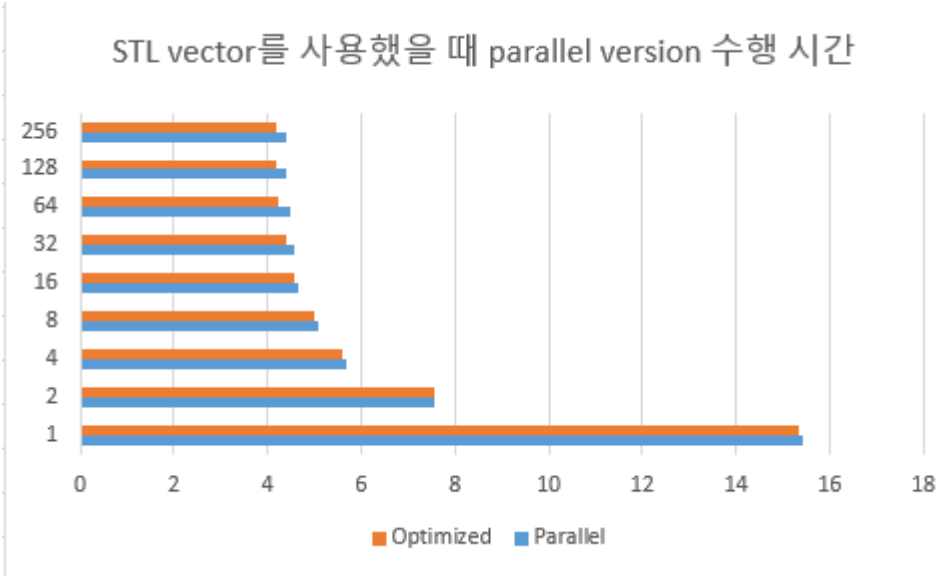
C++ STL에서 제공하는 set을 사용해서 구현했을 때, Serial Version과 Parallel 버전의 수행 시간 비교이다.

보다시피 STL set을 사용한 구현에서는 parallel로 인한 이점이 거의 없는 것을 알 수 있다. 이는 2가지 이유로 볼 수 있는데, 첫째는 STL set을 사용할 경우 탐색의 시간복잡도가 O(logN)이고, 현재 N이 2만 5000여개 정도에 불과하므로 parallel의 효과가 눈에 띄지 않는 것일 수 있다. 그리고 두번째로 애초에 STL set을 이용한 구현체에서 parallel version의 구현이 탐탁치 않기 때문으로 보인다. parallel하게 짜기 위해 추가되는 overhead가 많아져서 이로 인한 성능하락이 위와 같은 실험결과를 초래했을 것으로 생각된다.

그래도 parallel 버전에서 thread 개수가 2개 일 때 최적의 성능을 발휘함을 볼 수 있다.

2. Dynamic Array (STL vector)를 사용하여 구현했을 때,

단위 : sec									
Serial	**15.391**								
thread	1	2	4	8	16	32	64	128	256
Parallel	15.41685	7.56248	5.689485	5.100085	4.66621	4.550995	4.475125	4.40788	4.383275
Optimized	15.3298	7.579505	5.601155	4.988945	4.57627	4.419005	4.243195	4.1735	4.165025



C++ STL에서 제공하는 vector를 사용하여 구현했을 때, Serial Version과 Parallel 버전의 수행 시간 비교이다. Optimized는 Parallel version에서 약간의 최적화를 한 version이다. 일단 serial버전에 비교했을 때 thread 2개를 사용한 parallel 버전이 약 2배 더 빠른 것을 볼 수 있다. 이 것은 이론과 정확히 일치하는 결과임을 알 수 있다. 그리고 thread를 2개에서 4배로 늘렸을 때에도, 성능이 많이 향상하는 것을 볼 수 있다. 이 것은 실험을 수행한 환경과 관련이 크다. 내가 실험을 수행한 환경은 물리적 cpu core는 2개이고, 각 core에 hyperthreading이 적용돼서 logical core는 4개이다. 따라서 thread를 1개에서 2개로 늘렸을 때, physical core 레벨에서 각각 thread가 1개씩 작동하게 되어 속도가 2배 빨라진 것이다. 그리고 thread를 4개로 늘렸을 때, 속도가 많이 빨라졌지만 4배만큼 빨라지지는 않은 이유가 바로 hyperthreading에 의한 logical core 각각에서 thread가 1개씩 작동한 것이기 때문이다. 그리고 이러한 실험 환경의 이유로 thread개수를 4개 이상으로 증가시켰을 때 성능 차이가 매우 크지는 않음을 알 수 있다. 그럼에도 불구하고, thread 개수를 늘리면 계속해서 성능이 향상된다.

그리고 Optimized Version이 그냥 Parallel Version 보다 조금 더 빠른 것을 알 수 있다. 그 차이는 결과를 result file에 쓰는 작업을 하는 방법이 다르기 때문이다. Parallel 버전에서는 단순히 결과가 생성되면 바로 result file에 쓰는 데, 이 때 여러 thread에 의해 접근될 수 있으므로 openmp의 critical을 이용하여 동기화를 한다. 이로 인해서 성능하락이 발생하게 된다. 따라서 Optimized Version에서는 결과가 생성되면 각자 thread에게 할당한 메모리에 임시로 저장을 한다. 그리고 추후, 모든 결과가 생성되고 난 뒤, serial하게 메모리에 있는 결과들을 파일에 저장한다. 이 방법은 결과를 바로 파일에 쓰지 않고 임시 메모리에 저장하기 때문에 여기에 있어서 메모리가 더 쓰이고, 임시 메모리에 저장하는 시간이 추가되는 단점이 있다. 그러나 파일에 저장하는 코드가 동기화가 필요 없다는 점이 그 단점을 충분히 덮을 만큼의 성능 향상

을 가져다 주므로 이렇게 하는 게 훨씬 효율적이다.

나. OpenMP Thread Binding

1. 나의 Scheduling Algorithm 소개.

나는 OpenMP에서 제공하는 static과 dynamic을 합치는 식의 scheduling algorithm 방식을 사용하였다. 발상은 다음과 같다. static은 task를 정적으로 thread들에게 나눠주므로 scheduling에 대한 overhead가 적은 대신에, task들의 processing time이 제각각일 경우, 노는 thread들이 생길 수 있는 문제가 있다. 반면에 dynamic은 scheduling overhead가 큰 대신에, thread들에게 끊임없이 잘게 task를 쪼개서 나누어주므로 thread들이 잘 활용될 수 있다. 그래서 보통 프로그램의 성격에 따라 static 이나 dynamic을 사용하게 된다. 그런데 실제 top명령어 등으로 multicore usage를 모니터링해보면, static을 사용했을 때 노는 thread들이 생기는 문제는 보통 후반부에 발생하게 된다. 즉, 초중반에는 static처럼 스케쥴링 해도 노는 thread가 생기지 않는 것이다. 나는 여기에 착안하여 task를 절반으로 나누어, 절반의 task를 먼저 static 방식으로 스케쥴링하여 처리하고 그 다음 절반의 task를 dynamic 방식으로 스케쥴링하여 처리한다. 처음 절반의 task를 static방식으로 스케쥴링하는 openmp for는 물론 nowait와 함께 실행되어야 한다.

2. 성능 분석

단위 : sec				
Mandelbrot Example				
thread	1	2	4	8
my schd	25.2535	14.5819	9.8221	9.9735
static	25.3757	14.6476	10.3893	10.2325
dynamic	25.7046	13.5186	9.7984	10.1341
guided	28.7522	15.3648	11.002	10.4133

위가 실험결과이다. 나는 테스트용 예제 프로그램으로서 Mandelbrot 을 계산하는 프로그램을 사용하였다.

결과에서 static과 dynamic을 보면 dynamic이 미세하게 빠른 것을 알 수 있다. 그 이유는 Mandelbrot 프로그램의 경우 각 Task들이 처리시간이 다양하기 때문이다. 그러나 아주 많이 빠르지는 않은데, 그 것은 dynamic하게 스케쥴링을 하기위해 overhead가 많이 들기 때문이다. Thread가 8개일 때는, my scheduling 방법이 static과 dynamic보다 약간 빠름을 알 수 있는데, 발상대로 결과가 나온 것이라고 할 수 있다. 그러나 전체적으로 보면 실험결과가 다 비슷해서 사실 유의미한 분석

을 하기 힘들다.

나는 그래서 task들의 processing time이 일정한 예제 프로그램도 만들어 테스트해보았다.

단위 : sec				
Example2				
thread	1	2	4	8
my schd	8.0624	9.143	5.6928	5.6827
static	8.3825	8.9272	8.0904	11.6763
dynamic	12.766	13.2876	13.2379	13.239
guided	8.4611	6.2309	5.8353	8.024

그 결과는 위와 같다. 확연하게 dynamic이 느림을 알 수 있다. 그 이유는 scheduling overhead가 매우 큰데 반해서, task들이 processing time이 일정함으로 그것에 대한 이득은 보지 못하기 때문이다.

그런데 위에는 이상한 점이 2가지 있다. 첫 번째는 thread개수가 늘어나도 실행 속도가 거의 빨라지지 않고, 오히려 느려지기도 한다는 것이다. 두 번째는 guided가 static보다 빠르다는 것이다.

일단 thread개수가 늘어나도 실행 속도가 거의 빨라지지 않는 이유는 그 만큼 scheduling overhead가 크기때문으로 생각된다. 이 예제 프로그램의 경우 task 한 개가 하는 일은 단순히 곱셈연산 1개와 모듈러 연산 1개이다. 따라서 task를 scheduling하는 것에 대한 overhead가 부각이 되어 이런 결과가 나온 것으로 사료된다.

그리고 guided가 static보다 빠른 것은 OpenMP의 자세한 implementation을 봐야지 명확히 판단내릴 수 있을 것 같다. 그러나 실험적으로 봤을 때, static에서 chunk size를 키웠을 때, 속도가 빨라진 것으로 봐서 내부적으로 OpenMP의 static 스케쥴링 구현이 각 thread들에게 계산해야할 인덱스를 실제 코드로서 할당해주는 것으로 보인다. 이러면 당연히 이런 결과가 있을 것이다.

그리고 이제 나의 스케쥴링 방식의 성능을 분석해보면, static과 dynamic을 섞어씀으로서 static과 dynamic보다 더 좋은 성능을 발휘하고 있음을 알 수 있다.