멀티코어 프로그래밍 프로젝트

2013-11392 김지현

실험결과

먼저 CPU, GPU, MPI, SnuCL 각각에 대한 수행시간은 아래와 같다.

OpenCL (CPU)	OpenCL (GPU)	OpenCL (GPU) + MPI	SnuCL
10.6571초	**6.3127초**	**4.06775초**	**6.3829초**

최적화

프로젝트는 먼저 뼈대코드를 최적화하는것에서 시작하였다. 수행한 최적화는 아래와 같다.

- 대대적인 리팩터링
- 컴파일 옵션 조정 (C++0x로 컴파일하기, -O3 옵션 주기)
- 소스코드 하나로 합치기 (인라이닝)
- 글로벌 심볼일 필요 없는 함수/전역변수들에 static 붙이기
- 동적할당 제거
- 참조하는 메모리 영역이 겹치지 않는 포인터들에 대해 restrict 키워드 추가
- 불필요한 파라미터 전달 제거

리팩터링의 경우 직접적으로 성능에 연관이 되진 않지만, 이후의 원활한 과제수행을 위해 하였다. 이것으로 pthread를 사용하는 기본 뼈대코드의 수행시간을 **16.56**초 정도로 단축시킬 수 있었다. 병렬화는 위와같은 최적화가 이루어진 후에 수행되었다.

최적화를 병렬화보다 먼저 수행한것은, 한번 OpenCL로 병렬화를 수행하고 나면 디버깅에 어려움이 많아서였다.

병렬화

이 프로그램의 코드에는 여러가지 루프문들이 있는데, 대부분의 루프문은 반복횟수가 11회, 128회 이하이고, '-sm' 옵션으로 주어지는 시뮬레이션 갯수에 따라 실행하는 루프문이 반복횟수가 제일 크다 (1000000회). 이 루프문은 랜덤시드를 넘겨주는 코드와, 계산 결과를 합산하는 코드만 제거

하면 루프의 각 이터레이션 사이의 디펜던스가 완전히 사라진다. 그래서, 이 루프문을 병렬화의 대상으로 먼저 선정하였다. (앞으로 이를 Trial이라고 부르겠다)

또한, 소스코드에 Swaption이라고 이름붙여진 변수가 있다. 이는 같은 swaption 계산을 시작할때 인자만 달리하여 여러번 (128회) 수행하기 위해 있는것인데, 코드를 자세히 보면 swaption들 사이의 디펜던스는 전혀 없다는것을 알 수 있다. 그래서 이 또한 병렬화의 대상으로 선정하였다. (앞으로 이를 Task라고 부르겠다)

먼저 swaption함수에 넘겨주는 버퍼를 하나로 합친다. 이렇게 하면 OpenCL을 사용할때 버퍼를 한개만 초기화시켜줘도 되기때문에 효율적으로 변한다. Swaption 함수 외에 버퍼를 초기화시키는 코드, OpenCL 계산 이후 결과를 더하는 코드는 프로파일링을 해본 결과 수행시간이 굉장히 짧았기때문에 병렬화시키지 않고 두었다.

Task들 사이에는 버퍼를 초기화시킬때의 인자를 제외하면 Task 간의 디펜던시가 아예 없다. 그래서 이를 MPI를 이용해 Task들을 여러개로 분산시켰고, MPI의 각 노드 안에서의 Task들도 쪼개서 GPU 디바이스에게 각각 분배하였다.

Trial들끼리는 얼핏 보면 랜덤시드라는 디펜던시가 있는것처럼 보이지만, 주어진 랜덤함수의 형태를 살펴보면 랜덤함수에 시드로 넘겨주는 long 변수는 호출할때마다 그저 값이 1씩 늘어날뿐이라는것을 알 수 있다. 그래서 미리 커널코드 안에 배열을 할당하여 Trial들이 각자 사용할 랜덤시드의 값을 미리 계산하여 넘겨줌으로써 이를 최적화할 수 있다.

그 이후는 이전 과제들과 마찬가지로 OpenCL, MPI 함수들을 적절히 호출하여 고안한 병렬화 아이디어를 구현하였다.