스레드 배치 방식이 매니코어 프로세서의 성능에 미치는 영향

이정주[1], 이명호[2]
[1,2]명지대학교 컴퓨터공학과, 경기도 용인시 처인구 명지로 116
ryan20115@naver.com, myunghol@mju.ac.kr

Performance Effects of Thread Placement on Many-Core Processors

Jeongju Lee[1], Myungho Lee[2]
[1,2]Dept of Computer Engineering, Myongji Univ. 116 Myongji-ro, Cheoin-gu, Yongin-si, Gyeonggi-do
ryan20115@naver.com, myunghol@mju.ac.kr

요 약

Intel의 Knights Landing (KNL)은 최대 72개의 프로세서 코어들을 탑재한 고성능 매니코어 프로세서이다. KNL 상에서 메모리 연산이 집중된 병렬 응용 프로그램을 실행하면 코어들이 동시에 캐시, 캐시 인터커넥트, 메모리 버스, 등에 크게 부하를 가하는데, 병렬 프로그램의 성능을 높이기 위해서는 메모리 대역폭을 적절히 활용하는 것이 중요하다. 본 논문에서는 STREAM 벤치마크를 이용한 실험을 통해 병렬 스레드들의 배치 방식이 응용 프로그램의 성능에 영향을 미치는 영향을 분석한다.

1. 서 론

Intel의 Knights Landing (이하 KNL)은 최대 72개의 코어들을 탑재한 고성능 매니코어 프로세서이다 [1]. KNL 상에서 메모리 연산이 집중된 응용 프로그램을 병렬 실행할 경우 여러 스레드/프로세스들로부터의 메모리 접근 연산들로 인해 캐시, 캐시 인터커넥트, 메모리 버스, 등에 걸리는 부하가 크게 높아져 메모리 대역폭에 병목현상이 발생하는 등 성능이 저하될 수 있다. 응용 프로그램의 성능 최적화를 위해서는 메모리 대역폭을 효율적으로 사용하는 것이 중요한데, 본 논문에서는 STREAM 벤치마크를 사용한 실험을 통해 스레드의 배치 방식이 KNL의 성능에 미치는 영향에 대해 분석한다.

2. Intel KNL 아키텍쳐

본 논문에서는 Intel KNL 매니코어 아키텍쳐 기반의 프로세서들이 장착된 KISTI 슈퍼컴퓨터 5호기 누리온 상에서 실험을 진행하였다. 누리온 시스템은 Intel Xeon Phi 7250 프로세서들이 장착된 8,305개의 KNL 노드들과 Intel Xeon Gold 6148 프로세서들이 장착된 132개의 SKL 노드들로 구성되어 있다. 각 KNL 노드에 사용되는 프로세서는 68개의 코어들로 구성되며 1.4GHz로 동작한다. 68 코어들은 34개의 타일에 나누어 배치되는데, 각 타일은 1MB 크기의 L2 캐시를 공유한다. 따라서 L2 캐시의 총 용량은 34MB이다. 온패키지 메모리인 MCDRAM은 16GB(대역폭 490GB/s), 노드당 메모리는 96GB로 16GB DDR4-2400 메모리 6채널로 구성되어 있다 [2].

MCDRAM은 메모리 모드에 따라 사용 방식이 바뀐다. MCDRAM을 level-3 (L3) Cache로 사용하는 캐시 모드, DDR4의 추가적인 메인 메모리로 사용하는 Flat 모드, 캐시모드와 Flat 모드의 중간에 해당하는 Hybrid 모드(일부는 L3 캐시로 사용, 나머지는 메인 메모리로 사용), 등이 있다. 또한 클러스터 모드에 따라 코어와 메모리 사이의 거리를 설정할 수 있다. 자주 사용되는 클러스터 모드에는 Quadrant 모드와 SNC-4 모드가 있다. 두 모드는 칩을 4분면으로 분할한다. Quadrant 모드에서의 메모리 유형은 UMA이며 각 사분면에서 메모리에 접근하는 시간이 같다. SNC-4 모드에서의 메모리 접근 방식은 NUMA이며 각 사분면에서 가까운 메모리에 접근하는 시간이 더 짧다 [3].

3. 실험 환경

본 논문에서는 컴퓨터 시스템의 메모리 대역폭을 측정하는 STREAM 벤치마크를 사용하여 Intel KNL 상에서 실험을 수행하였다. STREAM은 4가지 연산인 Copy (c[i] = a[i]), Scale(b[i] = s * c[i]), Add(c[i] = a[i] + b[i]), Triad(a[i] = b[i] + s * c[i]) 로 구성된다 [4]. 실험에서 사용한 하나의 Intel KNL 노드는 68개의 프로세서 코어, 16GB 크기의 MCDRAM, 96GB 크기의 DDR4로 구성되어 있다. 메모리 모드는 Flat으로, 클러스터 모드는 Quadrant로 설정하였다. OpenMP를 사용하여 여러 개의 스레드가 STREAM을 병렬 실행하도록 하였다. OpenMP 환경 변수인 OMP_PLACES와 OMP_PROC_BIND를 사용하여 스레드가 배치되는 위치 및 방식을 지정하였다.

4. 실험 결과

스레드의 개수를 증가시켜 가며 STREAM 벤치마크를 실행하였다. 스레드가 실행될 위치를 지정하는 환경 변수 OMP_PLACES, 스레드가 배치되는 방식을 지정할 수 있는 OMP_PROC_BIND의 값들을 지정하여 실행하였다.

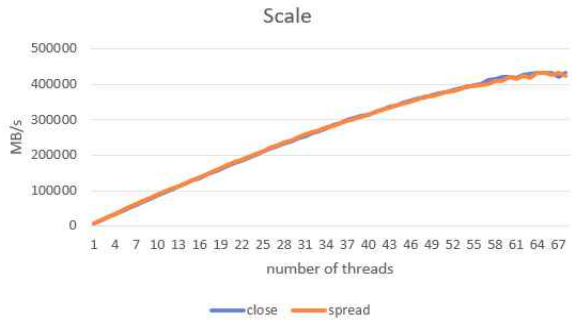
4.1 스레드의 배치 방식에 따른 성능 결과

우선 스레드의 배치 방식이 성능에 미치는 영향을 알아보기 위한 실험을 수행하였다. 환경변수 OMP_PLACES를 cores로 지정하면 스레드가 하나 이상의 하드웨어 스레드들이 탑재된 프로세서 코어에 배치되고, socket으로 지정하면 하나 이상의 코어로 구성되는 소켓에 배치된다. 실험에서는 cores로 지정하였다. OMP_PROC_BIND의 값을 close로 지정하면 i-번째 코어에 i-번째 스레드를 배치하여 스레드들이 가깝게 배치된다. Spread로 지정하면 스레드들을 분산시키는데, 전체 코어의 수를 스레드의 수로 나눈 값과 가까운 값을 간격으로 두어 배치한다. 실험에서는 close, spread를 모두 사용하였다. (그림 1)~(그림 4)는 STREAM의 각 연산에 대해 close와 spread를 비교한 실험 결과를 보여준다.

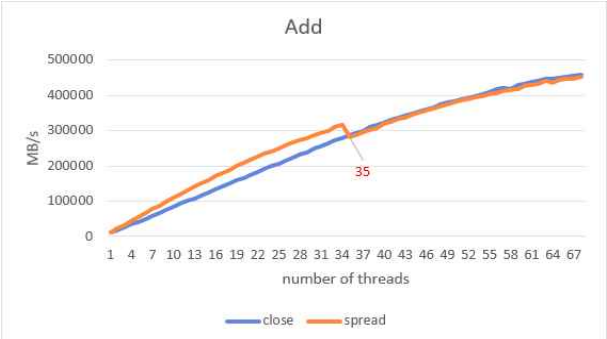
실험 결과 스레드 개수가 증가하면서 성능이 향상되는데, 1~34까지는 spread 방식이 close 방식에 비해 우수한 성능을 보인다. Spread를 사용하면 34-스레드까지는 각 타일에 1개의 스레드만 배치되어 L2 캐시와 메모리 대역폭을 독점하여 사용하기 때문이다. 스레드 개수가 34를 넘어서 35가 되면 spread의 성능이 하락했다가 점차 다시 향상되는데, 궁극적으로는 close의 성능과 거의 같아진다. 스레드 수가 증가하면서 각 스레드당 할당되는 데이터의 양이 줄어들어 실행시간이 단축되는 반면, 35-스레드부터는 close와 마찬가지로 각 타일에서 두 개의 코어들을 모두 사용하기 시작하면서 L2 캐시와 메모리 대역폭이 두 개의 코어들에 의해 공유되므로 하나의 코어가 독점하여 사용하던 경우에 비해 성능이 저하된다. 35-스레드의 성능 하락 현상은 후자의 성능저하가 전자의 성능향상 효과를 더 크게 상쇄하기 때문이다.

Spread 방식을 사용한 35-스레드에서의 성능 하락 현상은 연산 별로 다르게 나타나는데, Copy 연산의 경우에 성능 하락이 가장 크게 나타나며, 하락한 성능이 다시 상승하여 close의 성능과 같아지는 지점도 늦게(스레드 개수가 ~55개가 되어서야) 나타난다. 반면 곱셈을 사용하는 Scale 연산의 경우 close 방식과 spread 방식에서의 성능이 거의 같으며 스레드 개수가 35일 때 성능이 하락하는 현상이 나타나지 않는다. Add와 Triad 연산들에서도 35-스레드에서 성능저하가 나타나는데, 이때의 저하된 성능이 close 배치 방식의 성능과 같아지며, 스레드 개수가 늘어도 같은 성능이 유지됨을 볼 수 있다.

(그림 1) 스레드 배치 방식에 따른 Copy 성능



(그림 2) 스레드 배치 방식에 따른 Scale 성능



(그림 3) 스레드 배치 방식에 따른 Add 성능

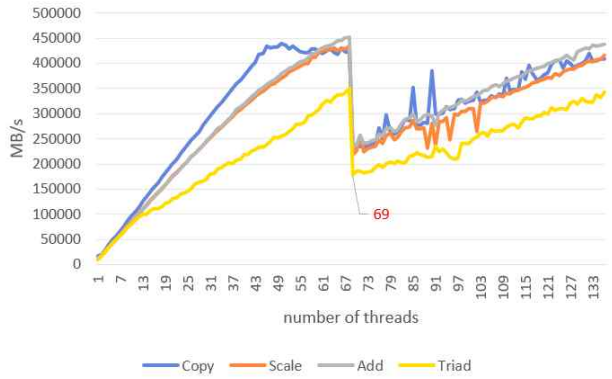
(그림 4) 스레드 배치 방식에 따른 Triad 성능

Spread를 사용할 경우 Copy 연산의 35-스레드에서의 성능 하락이 다른 연산들에 비해 크게 나타나는 이유는

메모리 접근 이외에 추가적인 연산이 없기 때문이다. 따라서 적은 수의 스레드로도 높은 성능을 보이지만 L2 캐시와 메모리 대역폭이 두 개의 스레드들에 의해 공유되기 시작하는 35-스레드부터 성능 하락이 크게 나타나는 것이다. 반면 Scale의 경우 Copy 연산 이외에 곱셈 연산을 수행하기 때문에 스레드 개수가 늘어나면 곱셈의 성능이 향상되면서 Copy 연산에서 하락한 성능을 상쇄하게 된다. Add, Triad에서는 세 개의 배열을 사용하여 Copy, 덧셈, 곱셈, 등의 연산을 수행하는데, Scale에 비해 더 많은 메모리 접근이 일어나고, 특히 Triad의 경우 Scale에 비해 추가로 덧셈 연산이 일어나므로 성능이 더 하락한다.

4.2 스레드 개수에 따른 성능 결과

두 번째 실험에서는 스레드 개수를 1개부터 136개까지 증가시키며 STREAM 벤치마크를 수행했다. (그림 5)는 STREAM의 각 연산에 대한 실험 결과를 보여준다. 환경변수 OMP_PLACES는 cores, OMP_PROC_BIND는 close를 각각 지정하였다.



(그림 5) 스레드 개수에 따른 STREAM 연산 별 성능

실험 결과, 모든 연산에서 68-스레드까지는 성능이 향상된다. 그러나 69-스레드에서 크게 하락하고 136-스레드까지 다시 향상된다. 69부터 스레드의 수가 코어의 수보다 많아지면서 코어당 두 개의 스레드들이 할당되는 타일에서 처리하는 데이터의 양이 그렇지 않은 타일의 데이터 양을 크게 초과하여 전체 실행시간을 증가(전체 성능 저하)시키기 때문이다. 또한 코어당 두 개 이상의 스레드들이 할당되면서 코어, L2 캐시, 메모리 대역폭, 등을 나누어 사용하기 때문이기도 하다. 그러나 스레드의 수가 늘어나 스레드당 할당되는 데이터의 양이 줄면 타일당 할당되는 데이터의 양이 전체적으로 비슷해지고 타일들의 부하가 균형이 맞추어지면서 다시 성능이 향상된다. 136-스레드의 성능은 68-스레드 성능보다 약간 낮은 수준까지 향상되는데, 코어 개수의 최대 2배에 달하는 스레드들이 각 타일의 메모리 대역폭 등을 공유함으로 인한 성능저하 정도가 크지 않음을 보여준다고 할 수 있다.

5. 결 론

KNL 상에서 스레드 배치 방식과 스레드 개수가 메모리 연산이 집중된 응용 프로그램의 성능에 미치는 영향을 STREAM 벤치마크를 이용한 실험을 통해 분석하였다. 각 타일당 하나의 코어에만 스레드가 할당될 경우 spread 방식이 close보다 우수한 성능을 보이나, 두 개의 코어들에 모두 스레드들이 할당되기 시작하면서 spread의 성능이 하락하여 close의 성능에 수렴한다. Spread의 성능 하락은 Copy 연산 이외의 추가적인 연산과 메모리 접근 비중에 따라 그 정도가 다르게 나타났다. Close를 이용하여 스레드 개수를 코어 개수의 2배인 136까지 증가시키면, 코어당 스레드가 하나만 할당될 때까지(68 스레드)는 성능이 일관되게 향상되나 그 후에 큰 폭으로 떨어졌다가 다시 향상되기 시작하여 136-스레드에서는 68-스레드보다 약간 낮은 수준까지 올라간다. 타일당 할당되는 스레드 수에 따라 타일들간의 데이터 처리량이 불균형할 때는 성능저하 현상이 발생하나 처리량의 균형이 맞아지면서 성능이 다시 향상된다. 이러한 결과는 코어 개수보다 최대 2배 많은 스레드들이 캐시와 메모리 대역폭 등의 자원을 공유함으로 인한 성능저하 정도가 크지 않음을 보여준다.

6. 사사

이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(RS-2023-00321688).

7. 참고 문헌

[1] A. Sodani, "Knights landing (KNL): 2nd Generation Intel® Xeon Phi processor," 2015 IEEE Hot Chips 27 Symposium (HCS), 2015, pp. 1-24

[2] 변은규. (2020), “인텔 KNL 프로세서 사례를 통한 고성능 온칩 메모리의 성능 병목 분석 및 해결 방안 연구,” 한국정보처리학회 학술대회논문집, 27(2), 92-95.

[3] 유진승, 박근철, 곽재혁, 류훈, 남덕윤. (2017), “매니코어 프로세서 기반 시스템 소개 및 성능 분석,” 정보과학회지, 35(10), 8-17.

[4] 노승우, 박근철, 최지은, 박찬열. (2020), “확장형 벤치마크 자동화 도구를 이용한 고성능컴퓨팅시스템 성능 평가,” 정보과학회 컴퓨팅의 실제 논문지, 26(2), 81-88.

[5] G. Bolet, G. Georgakoudis, K. Parasyris, K. W. Cameron, D. Beckingsale and T. Gamblin, "An Exploration of Global Optimization Strategies for Autotuning OpenMP-based Codes," 2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 741-750