이종 멀티코어 프로세서 작업 스케줄링에 관한 연구 동향 분석

김성일*, 김종국*
*고려대학교 전기전자전파공학과
e-mail:sikim84@korea.ac.kr, jongkook@korea.ac.kr

Trends on Task Scheduling in Heterogeneous Multi-core Processors

Sung-il Kim*, Jong-kook Kim*
*School of Electrical Engineering, Korea University

요 약

이종 멀티코어 프로세서는 각기 상이한 마이크로아키텍처, 캐시 사이즈, 클록 주파수를 갖는 다수의 코어 또는 프로세싱 유닛으로 이루어진 마이크로프로세서이다. 저에너지 소비가 산업계의 키워드로 부상하고 있는 이 시기에 이종 멀티코어는 동종 멀티코어보다 더 낮은 전력을 소비하고 성능면에서도 더 나은 프로세서로 주목받고 있다. 하지만, 동종 멀티코어에서의 동작을 가정하는 현재의 운영체제의 작업 스케줄러로는 이종 멀티코어의 이종적인 특성을 잘 활용할 수 없다. 본 논문에서는 이종 멀티코어 프로세서 작업 스케줄링에 관한 연구를 다면적으로 분석하여 각 방법의 장점과 단점을 개략적으로 정리하고 관련된 이슈들을 살펴보고자 한다.

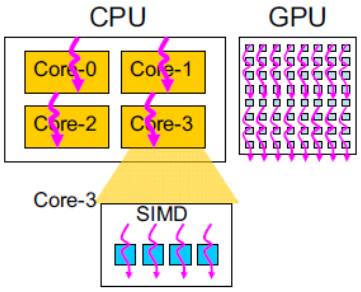
1. 서론

현대에 이르러 '친환경'이라는 키워드가 산업계 전반에 대두되면서 IT업계에서도 그린 IT (Green IT)라는 키워드 아래 IT로 인한 환경문제를 해결하려 노력하고 있다. 선진국들은 기업과 정부에서 정책을 추진하고 친환경적인 기술을 적용한 다양한 제품을 개발하여 시장에 출시하고 있다. 이 같은 세계적인 흐름은 CPU 기술에도 영향력을 발휘하여 저전력 CPU에 적용할 수 있는 다양한 전력관리 기술들이 연구되고 있다.

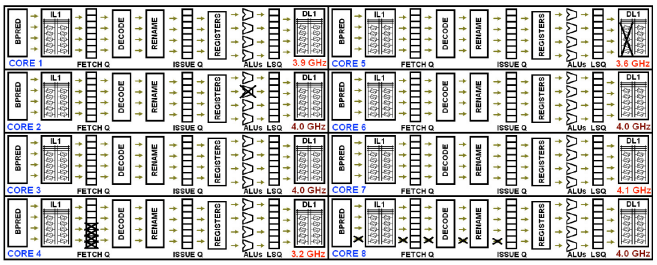
이종 멀티코어 프로세서(Heterogeneous Multi-Core Processor)는 다수의 코어 혹은 다수의 프로세싱 유닛마다 각기 다른 아키텍처, 캐시 사이즈, 클록 주파수를 갖는 마이크로프로세서로 모든 코어가 동일한 동종 멀티코어 프로세서에 비해 낮은 전력을 소비하고 더 나은 성능을 가지는 마이크로프로세서로 각광받고 있다[1]. 이종 멀티코어 프로세서는 다양한 어플리케이션의 요구에 맞춘 특화된 코어들의 다양한 조합을 바탕으로 상대적으로 자원을 덜 낭비하며 고성능 컴퓨팅, 효율적인 컴퓨팅을 가능하게 하기 때문에 미래 CPU 기술은 이종 멀티코어 프로세서를 기반으로 흐를 것임은 자명해 보인다. 하지만, 현재의 운영체제는 싱글코어 또는 동종 멀티코어 아키텍처를 가정하고 있기 때문에 이종 멀티코어 환경에 적합한 새로운 연구가 필요하다. 작업 스케줄링은 이와 관련된 중요한 문제중 하나로 이종적인 특성을 잘 활용하여 다양한 경우에서 종합적으로 높은 성능을 발휘하도록 효율적인 작업 스케줄링이 연구되어야 한다. 본 논문에서는 이종 멀티코어 프로세서에서의 종류를 먼저 살펴보고 작업 스케줄링 연구 동향을 대략적으로 분석 및 정리하여 이를 통해 연관된 다양한 이슈들을 짚어나가도록 하겠다.

2. 이종 멀티코어의 종류

이종 멀티코어 프로세서는 CPU, GPGPU, DSP 등 다양한 코어가 존재하는 만큼 다양한 아키텍처들이 제시되어 있다. [1, 2]에서는 이슈 폭(issue width), 캐시 사이즈, 클록 주파수가 다르나 같은 명령어 세트 아키텍처(ISA)를 가진 멀티코어를 다루었고 [3]에서는 빠른 코어와 느린 코어의 조합하고 명령어 세트를 공유하는 영역과 명령어 세트를 공유하지 않고 독자적인 기능을 가진 영역을 구분한 아키텍처를 제시하였다. 그림 1과 같이 코어의 개수가 적은 CPU와 다수의 특수목적 코어로 설계된 GPU, 즉 CPU-GPU 아키텍처를 기반으로 한 연구도 있다[4]. 한편, 동적 전압/클록 스케일링(DVFS)를 이용하여 동종 멀티코어의 코어 클록 주파수를 변경하여 이종 멀티코어 프로세서를 에뮬레이션하는 시스템[5, 6, 7, 8, 9, 10]은 전형적인 이종 멀티코어 환경에 속한다. 또한 동종 멀티코어 시스템에서 생산공정에서 발생한 하드웨어 결함(그림 2), 프로세스 바리에이션, 노후화로 인한 성능저하로 의도치 않게 이종 멀티코어 시스템이 만들어지기도 한다[10, 11]. 그리고 언급된 시스템 이외에도 수많은 이종 멀티코어 시스템이 있다.



(그림 1) CPU + GPU system[4]



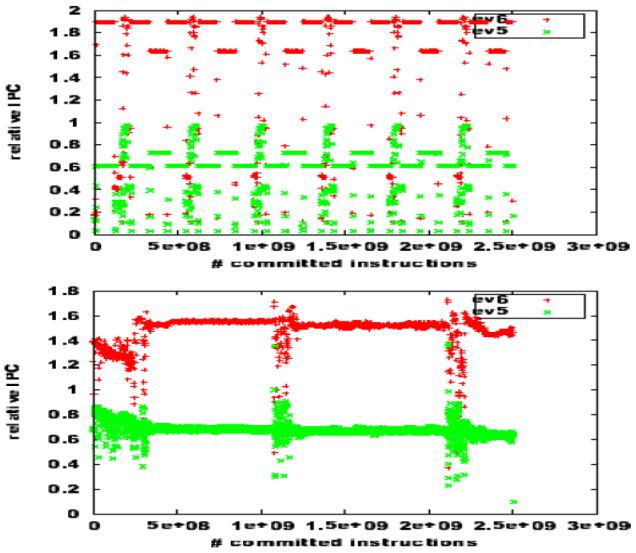
(그림 2) 부분 결함이 생긴 프로세서('X' 표시)[11]

3. 이종 멀티코어에서의 작업 스케줄링 연구 분석

작업을 프로세서에 할당하는 문제는 잘 알려진 NP-Complete 문제이다. 멀티코어에서는 작업을 코어에 할당하는 방법에 따라 성능 차이가 나게 되는데 동종 멀티코어에서는 모든 코어가 같은 특성을 가지기 때문에 동일한 작업을 어느 코어에 할당하든 같은 실행시간과 에너지를 소비하지만 이종 멀티코어에서는 코어에 따라 작업의 실행시간과 에너지 소비가 다르기 때문에 효율적인 작업 스케줄링이 필요하다. 이종성(heterogeneity) 활용을 극대화하기 위해서는 코어의 특성뿐만 아니라 작업의 특성, 작업이 들어오는 런타임 환경(캐시 메모리, 큐 등)을 고려한다면 문제의 복잡성은 더욱 증가한다.

[2, 12]는 이종 멀티코어 프로세서에서의 작업 스케줄링 초기연구 중 가장 잘 알려져 있다. 두 연구 모두 Alpha 프로세서(EV5, EV6)의 조합으로 이종 멀티코어 시스템을 모델링하였으며, 샘플링을 통해 얻은 IPC 정보를 바탕으로 시스템 전체의 IPC를 높이는 방향으로 작업이 동적으로 할당되는 동일한 구조를 가지고 있다. 즉, 설정된 샘플링 구간(sampling phase)에서 얻어진 정보를 가지고 작업을 알맞은 코어에 할당하여 유지 구간(steady phase)동안 적용을 하게 된다. 여기서 샘플링의 빈도가 높다면 샘플링하는 데 드는 오버헤드가 증가하여 성능저하가 발생하고 샘플링 구간길이가 길고 작업할당 알고리즘의 오버헤드가 크다면 그 시간동안 실행중인 작업들의 할당상태는 최적의 상태와 거리가 멀 가능성이 있다. 따라서, 샘플링 구간 길이는 최대한 작게, 샘플링 빈도는 높지 않으나 작업상태(혹은 작업부하)에 민감하게 반응할 수 있어야 한다. [2]에서는 주기적으로 정하거나, 런타임 환경을 모니터링하여 큰 변화가 있을 때 샘플링 구간을 갖고 다시 작업을 할당하도록 하는 방법을 제시하였다. 작업마다 프로그램 상태(program phase)가 변하는 시간이 다르기 때문에(그림 3) 런타임 환경을 모니터링하여 IPC에 큰 변화가 있을 때 샘플링 구간을 갖는 것이 더 효율적이다. 또한 [8]에서는 프로그램 상태 변화를 오프라인 분석을 통해 알아내는 방법을 제시하고 있다. [12]에서는 고속 코어에서 작업 이득이 크기 때문에 고속 코어를 가급적 유휴상태로 만들지 않고 최대한으로 이용하는 방법으로 성능을 향상시켰다. 유휴 코어가 있을 때 고속 코어부터 할당하고 저속 코어에서 작업이 실행중이고 고속 코어는 작업이 끝나서 유휴상태가 될 때 저속코어에서 고속코어로 강제적인 쓰레드 마이그레이션이 발생하게 되어 빠른 시간에 작업이 끝나게 된다. 하지만 위의 연구들[2, 8, 12] 처럼 단지 IPC 정보만을 바탕으로 작업을 할당하게 되면 IPC가 높은 코어에 작업이 많이 몰리게 되어 로드 밸런싱 문제를 일으킨다.

[2, 12]와 같은 동적 스케줄링 정책은 매순간 변화하는 런타임 환경과 어플리케이션 실행상태를 반영하여 스케줄링을 할 수 있다는 장점이 있지만 코어의 수가 늘어날 경우에는 샘플링 오버헤드가 커진다는 단점이 있다[5]. 이에 비해 정적 스케줄링 정책은 런타임 환경을 샘플링하지 않기 때문에 오버헤드가 적으며 동적 스케줄링에 비해서 구현이 용이하다. [5]에서는 프로그램의 오프라인 프로파일링을 통해 메모리 블록 재참조 길이(reuse-distance)정보를 기반으로 하는 정적인 스케줄링을 제시한다. 추출된 메모리 블록 재참조 길이 정보를 통해서 세트 연상(set-associativity), 캐시 사이즈별 캐시 미스율을 간접적으로 추측할 수 있게 되고 해당 쓰레드의 메모리 접근 지



(그림 3) 어플리케이션의 IPC변화
(위: mgrid, 아래: parser)[12]

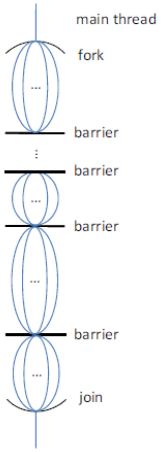
연 시간을 알 수 있게 되어 쓰레드의 코어별 성능을 예측하는데 도움을 준다. 그리고 동일한 코어로 이루어진 CPU로 그룹화하고 그룹내에서 몇 개의 코어로 다시 파티셔닝을 하여 이를 스케줄링의 기본 단위로 삼으면서 확장성있는 구조를 제시하였다. 이 경우에도 마찬가지로 성능이 높은 CPU 파티션에는 쓰레드가 많이 할당되고 성능이 낮은 CPU 파티션은 유휴상태가 되어 로드 분배가 적절하지 않은 상태를 일으킬 수 있으므로 모든 코어에서 쓰레드가 실행중인 CPU 파티션에는 쓰레드를 더 이상 할당하지 않는 방법을 통해 간접적으로 로드를 분배하였다.

퍼지 이론을 적용한 연구인 [13]에서는 성능보다 저에너지에 중점을 두었다. 프로그램에서 아키텍처에 독립적인 특성을 오프라인 프로파일링하여 퍼지 논리로 프로그램-코어 적합도를 계산하여 작업 스케줄링하는 방법을 제시하였다. 이 연구에서는 프로그램의 명령어 의존길이, 분기 전이율, 데이터 재사용 길이를 각각 하드웨어에서의 이슈 폭, 분기 예측기, L1 데이터 캐시와 대응하여 적합도를 판단한다. 그리고 세 가지 적합도는 표 1과 같이 퍼지논리에 의해 최종적인 적합도를 산출하는데 이용된다.

<표 1> 퍼지 시스템의 최종 적합도 판정[13]

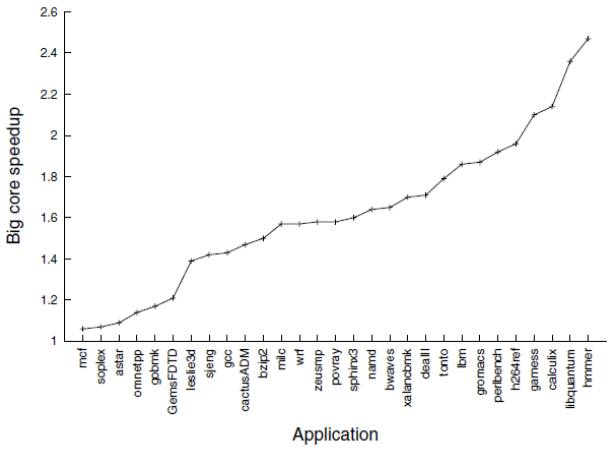
IF	THEN
(issue width suitability is low) **AND** (cache suitability is low) **AND** (branch predictor suitability is low)	(overall suitability is EL)
(issue width suitability is low) **AND** (cache suitability is low) **AND** (branch predictor suitability is high)	(overall suitability is VL)
(issue width suitability is low) **AND** (cache suitability is high) **AND** (branch predictor suitability is low)	(overall suitability is L)
(issue width suitability is high) **AND** (cache suitability is low) **AND** (branch predictor suitability is low)	(overall suitability is ML)
(issue width suitability is low) **AND** (cache suitability is high) **AND** (branch predictor suitability is high)	(overall suitability is MH)
(issue width suitability is high) **AND** (cache suitability is low) **AND** (branch predictor suitability is high)	(overall suitability is H)
(issue width suitability is high) **AND** (cache suitability is high) **AND** (branch predictor suitability is low)	(overall suitability is VH)
(issue width suitability is high) **AND** (cache suitability is high) **AND** (branch predictor suitability is high)	(overall suitability is EH)

한편 [9]는 멀티쓰레드 프로그램의 fork-join모델(그림 4)에서 비슷한 시기에 생성된 쓰레드는 실행시간도 비슷하다는 점에 착안하여 쓰레드 생성시간으로 쓰레드의 남은 실행시간을 예측하여 스케줄링에 이용하였다. fork해서 병렬처리되는 쓰레드들이 barrier에서 join하게 되는데 여기서 join하는 모든 쓰레드가 끝나야만 하나의 프로그램 흐름이 생성되므로 어느 하나의 쓰레드의 완료시간이 느리다면 프로그램 흐름은 그 쓰레드에 맞추어 느려지게 된다. 쓰레드의 실행시간을 예측하는 것은 상당히 어려운 문제이기 때문에 같은 시기에 시작된 쓰레드 중 가장 많은 실행시간이 남아있는 쓰레드를 예측하여 고속 코어를 그 쓰레드에 할당하는 방법을 제시하였다.



(그림 4) 멀티쓰레드 fork-join 프로그램 모델 [9]

지금까지 살펴보았던 연구에서는 온라인 샘플링 혹은 오프라인 프로파일링으로 프로그램의 특성 정보를 얻지만 CPI 스택과 같은 하드웨어로부터 얻은 정보로 해당 프로그램이 어느 단계에서 많은 시간을 소비했는지 알아내어 이용하는 연구[7]도 있다. 이 연구에서도 전형적인 이종 멀티코어 시스템인 고속코어-저속코어 조합을 모델링하였고 프로그램을 저속코어 대비 고속코어의 성능향상을 기준(그림 5)으로 두 가지 편향(big core bias/small core bias)으로 나누었다. 어떤 프로그램이 코어에서 연산이 되는 시간에 비해 메모리 접근과 같은 작업에 걸리는 시간의 비율이 크다면 그 프로그램은 잦은 메모리 요청 등으로 고속코어의 연산능력을 충분히 활용하지 못할 것이다. 따라서 이러한 프로그램은 저속 코어에 할당하여야 한다(small core bias). 반대의 경우(big core bias)라면 고속코어에 할당하여 고속 코어의 활용성을 높이는 것이 타당한 방법이다. 로드 밸런스 상태는 주기적으로 검사되고 로드가 균형상태에 있지 않다면 같은 종류의 코어에 한해서 쓰레드를 유휴코어로 할당하여 코어의 로드 균형을 유지한다.



(그림 5) SPEC CPU2006 벤치마크 프로그램에서의 저속코어 대비 고속코어의 성능향상[7]

4. 결론 및 제언

다양한 이종 멀티코어 프로세서 작업 스케줄링 방법에 대해서 개략적으로 살펴보았다. 더욱 많은 작업 스케줄링 방법들이 연구 및 제시되었고 스케줄링 정책의 공정성(fairness), 컴파일러의 지원 등의 이슈가 있지만 지면의 제약으로 더 이상 다룰 수 없었다.

이종 멀티코어 환경에서는 아키텍처 상에서 나타나는 이종성을 활용하기 위해 하드웨어에 대한 분석뿐만 아니라 프로그램에 대한 특성분석이 필수적이다. 이종 멀티코어는 그 설계에서부터 다양한 프로그램의 요구에 맞는 프로세싱 유닛들을 가지고 있기 때문에 효율적인 스케줄러가 필요하다. 지금까지 살펴본 논문들에서는 샘플링이나 프로파일링 등으로 프로그램 특성을 분석하였는데 앞으로는 더 많은 기법들이 제시되고 더욱 다양한 이종 멀티코어 환경에서 연구가 이루어질 것이다.

Acknowledgement

본 논문은 한국연구재단의 부분적 지원을 받아 연구되었음(2009-00763)

참고문헌

[1] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan and D. M. Tullsen. "Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction", In *Proceedings of the 36th International Symposium on Microarchitecture*, 2003.

[2] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi and K. I. Farkas. "Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance", In *Proceedings of the 31st Annual International Symposium on Computer Architecture*, 2004.

[3] T. Li, P. Brett, R. Knauerhase, D. A. Koufaty, D. Reddy and Scott Hahn. "Operating System Support for Overlapping-ISA Heterogeneous Multi-core Architectures", In *Proceedings of the 16th International Symposium on High Performance Computer Architecture*, 2010.

[4] C. K. Luk, S. Hong and H. Kim, "Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping", In *Proceedings of the 42nd International Symposium on Microarchitecture*, 2009.

[5] D. Shelepov, A. Fedorova, S. Blagodurov, J. C. Saez Alcaide, N. Perez, V. Kumar, S. Jeffery and Z. F. Huang. "HASS: A Scheduler for Heterogeneous Multicore Systems", *Operating Systems Review*, vol.43, no. 2, pages 66-75, 2009.

[6] T. Li, D. Baumberger, D. A. Koufaty and Scott Hahn. "Efficient Operating System Scheduling for Perfo-rmance-Asymmetric Multi-Core Architectures", In *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2007.

[7] D. Koufaty, D. Reddy and Scott Hahn. "Bias Scheduling in Heterogeneous Multi-core Architectures", In *Proceedings of the 5th ACM European Conference on Computer Systems*, 2010.

[8] T. Sondag and H. Rajan. "Phase-guided Thread-to-core Assignment for Improved Utilization of Performance-Asymmetric Multi-Core Processors", In *ICSE Workshop on Multicore Software Engineering*, 2009.

[9] N. B. Lakshminarayana, J. Lee and H. Kim. "Age Based Scheduling for Asymmetric Multiprocessors", In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009.

[10] R. Teodorescu and J. Torrellas. "Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors", In *Proceedings of the 35th annual International Symposium on Computer Architecture*, 2008.

[11] J. A. Winter, D. H. Albonesi and C. A. Shoemaker. "Scalable Thread Scheduling and Global Power Management for Heterogeneous Many-Core Architectures", In *Proceedings of the 9th International Conference on Parallel Architectures and Compilation Techniques*, 2010.

[12] M. Becchi and P. Crowley. "Dynamic Thread Assignment on Heterogeneous Multiprocessor Architectures", *In Proceedings of the Conference on Computing Frontiers*, 2006

[13] J. Chen and L. K. John. "Energy-Aware Application Scheduling on a Heterogeneous Multi-core System", In *Proceedings of the International Symposium on Workload Characterization*, 2008.