태스크 Speedup 평가 기반 스케줄링을 통한 이기종 멀티코어 시스템의 처리량 극대화

노순현 [O1] 홍성수 [1,2]

[1] 서울대학교 전기·정보공학부

[2] 차세대융합기술연구원 스마트시스템연구소

shnoh@redwood.snu.ac.kr, sshong@redwood.snu.ac.kr

Maximizing throughput of asymmetric multicore systems via scheduling based on task speedup evaluation

Soonhyun Noh[O1] Seongsoo Hong[1,2]

[1]Department of Electrical and Computer Engineering, Seoul National University

[2]Advanced Institutes of Convergence Technology Institute for Smart System

요 약

제한된 에너지로 수행되는 응용의 서비스 품질을 높게 유지시킬 수 있는 방법으로 이기종 멀티코어 프로세서가 제안되었다. 이기종 멀티코어 시스템의 처리량은 어떤 태스크를 고성능 코어에서 수행시키는가에 크게 의존한다. 저전력 코어 대비 고성능 코어에서의 speedup이 높은 태스크를 고성능 코어에서 수행시켜야 처리량을 극대화시킬 수 있다. 우리는 speedup을 계산해서 태스크를 배치한 후 예측 모델을 통해 speedup 변화를 관찰하는 스케줄링 알고리즘을 제안한다. PARSEC 벤치마크를 사용한 실험 결과 우리 솔루션은 기존 Linaro 스케줄러에 비해 수행 시간을 7.44% 줄이는 것을 확인하였다.

1. 서 론

제한된 에너지로 높은 서비스 품질을 유지할 수 있는 컴퓨팅 플랫폼에 대한 수요가 증가하고 있다. 이 수요를 충족시키기 위한 방법 중 하나로 이기종 멀티코어 프로세서가 제안되었다. 이기종 멀티코어 프로세서는 높은 성능을 요구하는 태스크들을 고성능 코어에서 수행하고 나머지 태스크들을 저전력 코어에서 수행하며 제한된 에너지로 높은 서비스 품질을 유지한다. 이런 강점 때문에 최근에도 ARM 사의 빅리틀(big.LITTLE) 프로세서나 NVidia 사의 칼엘(Kal-El) 프로세서 등 다양한 이기종 멀티코어 프로세서들이 출시되고 있다.

이기종 멀티코어 시스템의 처리량(throughput)은 고성능 코어에서 수행시킬 태스크를 선택하는 과정에 크게 의존한다. Speedup이 높은 태스크를 고성능 코어에 배치시킬수록 시스템의 처리량이 증가한다. 한 태스크의 speedup은 그 태스크가 고성능 코어에서 단독으로 수행되었을 때 걸린 시간 대비 저전력 코어에서 혼자 수행되었을 때 걸린 시간의 비율이다.

Speedup이 높은 태스크를 판별하여 고성능 코어로 보내는 연구들이 그 동안 많이 진행되었다. 이들은 speedup을 판단하는 시점에 따라 크게 두 분류로 나눌 수 있다. 첫 번째 분류의 연구들은 오프라인에 각 태스크의 speedup을 미리 계산한다 [1]. 하지만 이 연구들은 태스크의 speedup이 시간에 따라 가변적이라는 사실을 고려하지 못해 한계가 뚜렷하다.

두 번째 분류의 연구들은 스케줄링 시 동적으로 태스크의 speedup을 계산한다. 이들은 speedup의 계산방식에 따라 다시 두 분류로 나뉜다. 첫째는 태스크를 고성능 코어와 저전력 코어에 번갈아 수행시키며 직접 speedup을 측정하는 방식이다 [2][3]. 이 연구들은 정확하지만 주기적으로 태스크 이주가 강제되어 큰 오버헤드가 발생한다. 둘째는 태스크를 별도로 이주시키지 않고 여러 동작 특성으로부터 예측 모델을 만드는 방식이다 [4][5]. 하지만 제안된 예측 모델들은 정확성이 아직 충분하게 검증되지 않았다.

이와 같은 기존 연구들의 문제점을 해결하기 위해 본 논문에서는 동적으로 태스크 speedup을 계산하는 방식을 확장한다. 직접 태스크 speedup을 측정하되, 태스크 이주 오버헤드를 줄이기 위해 계산 빈도를 낮추고, 예측 모델을 통해 speedup 변화를 관찰한다.

우리는 제안한 기법을 Linaro Stable Kernel 14.09가 탑재된 Versatile Express TC2 보드에 구현했다. PARSEC 벤치마크로 실험한 결과 제안한 스케줄러가 동일한 벤치마크 프로그램의 수행 시간을 기존에 비해 7.44% 줄이는 것을 확인하였다.

2. 빅리틀 프로세서와 Linaro 스케줄러

ARM 사의 빅리틀 프로세서는 대표적인 이기종 멀티코어 프로세서이다. 이 프로세서는 고성능 코어인 빅 코어로 이루어진 빅 클러스터와 저전력 코어인 리틀

코어로 이루어진 리틀 클러스터로 구성된다.

Linaro 스케줄러는 빅리틀 프로세서를 지원하는 대표적인 스케줄러이다. Linaro 스케줄러의 동작은 클러스터 내의 스케줄링과 클러스터 간의 스케줄링으로 나눌 수 있다. 클러스터 내의 스케줄링은 바닐라 리눅스 커널의 CFS (Completely Fair Scheduler)를 그대로 사용한다 [6]. 클러스터 간의 스케줄링에서는 speedup을 예측하고 이에 따라 태스크를 이주시킨다.

클러스터 간 스케줄링에서는 speedup을 예측하기 위해 각 태스크의 `load_avg_ratio`를 관찰한다. `load_avg_ratio`는 태스크가 최근 수행 중이었거나 런큐(run-queue)에서 대기 중이었던 시간을 나타내는 지표이다. 이 값이 높은 태스크는 CPU 집약적이므로 speedup이 높다고 판단하여 빅 클러스터로 이주시키고, 반대일 경우 리틀 클러스터로 이주시킨다.

3. 시스템 모델 및 문제 정의

대상 시스템은 빅 클러스터와 리틀 클러스터로 구성된다. 빅 클러스터 P_b 는 r 개의 동일한 빅 코어들 $\{p_1^b, p_2^b, ..., p_r^b\}$ 로 이루어져 있고, 리틀 클러스터 P_L 는 s 개의 리틀 코어들 $\{p_1^L, p_2^L, ..., p_s^L\}$ 로 이루어져 있다. 빅 코어와 리틀 코어의 허용 동작 주파수의 집합을 각각 $F(P_b)$ 와 $F(P_L)$ 로 표시한다. 대상 시스템에서는 n 개의 태스크 $S = \{\tau_1, \tau_2, ..., \tau_n\}$ 가 수행된다. 그리고 태스크 τ_i 의 speedup을 s_i 로 표시한다.

우리의 목적은 어떤 시점에서도 s_i 가 가장 높은 r 개의 태스크를 빅 클러스터에서 수행시키는 것이다. 이를 통해 시스템의 처리량을 극대화시킬 수 있다. 특히, Linaro 스케줄러 상에서 동작하는 솔루션을 제안한다.

4. 해결 방안

본 장에서는 우리가 제안하는 솔루션을 설명한다.

4.1 솔루션 전체 구조

제안한 스케줄러는 주기적으로 학습 구간과 최적화 구간을 교대로 거친다. 학습 구간에는 각 태스크를 양 클러스터에서 수행시키며 speedup을 직접 계산하고, 이 값이 큰 태스크를 빅 클러스터에 배치한다. 최적화 구간에는 `load_avg_ratio`에 기반하여 각 태스크의 speedup 변화를 관찰하고, 문턱 값을 넘을 경우 해당 태스크를 다른 클러스터로 이주시킨다.

우리 솔루션은 speedup 계산기, 클러스터 간 스케줄러와 클러스터 별 스케줄러로 구성된다. Speedup 계산기는 학습 구간에 각 태스크의 speedup을 계산한다. 클러스터 간 스케줄러는 계산 값을 기반으로 각 태스크를 수행시킬 클러스터를 결정한다. 클러스터 별 스케줄러는 클러스터 내의 태스크 스케줄링을 담당하고, 기존 Linaro 스케줄러를 그대로 사용한다. 그림 1은 설명한 솔루션의 전체 구조를 나타낸다.

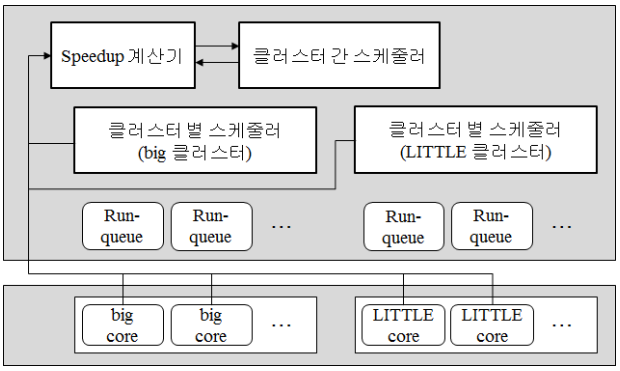

그림 1. 솔루션 전체 구조

4.2 Speedup 계산기

Speedup 계산기는 학습 구간 동안 태스크의 수행 정보를 관찰한 후 이를 기반으로 speedup을 계산한다. 다음 두 정보를 관찰한다.

- 한 태스크가 각 클러스터에서 수행한 명령어 개수 (Performance monitoring unit을 사용해 측정)
- 한 태스크가 각 클러스터에서 각 동작 주파수로 수행된 시간

학습 구간 동안 태스크 τ_i 가 클러스터 σ 에서 수행한 명령어 개수를 $I_i(\sigma)$ 로 표시하고, τ_i 가 σ 에서 동작 주파수 f 로 수행된 시간을 $C_i(\sigma, f)$ 로 표시한다.

우리는 각 태스크가 양 클러스터에서 초당 수행한 명령어 개수의 비를 speedup의 지표로 사용한다. 하지만 태스크는 동일한 클러스터에서도 다양한 동작 주파수에서 수행되었기 때문에 이를 고려해 speedup을 계산하여야 한다. 우리는 태스크 τ_i 의 클러스터 σ 에서의 주파수 보정 시간 $\hat{C}_i(\sigma)$ 를 정의한다.

$$\hat{C}_i(\sigma) = \sum_{f \in F(\sigma)} [C_i(\sigma, f) \times \frac{f}{f_{\text{ref}}(\sigma)}]$$

이 때 $f_{\text{ref}}(\sigma)$ 는 기준 주파수로 $F(\sigma)$ 의 값 중 하나를 사용한다. 각 클러스터의 최고 주파수를 선택했다.

태스크 τ_i 의 speedup s_i 는 다음 식으로 계산한다.

$$s_i = \frac{I_i(P_b) / \hat{C}_i(P_b)}{I_i(P_L) / \hat{C}_i(P_L)}$$

4.3 클러스터 별 스케줄러

클러스터 별 스케줄러의 동작은 학습 구간에서의 동작과 최적화 구간에서의 동작으로 나뉜다. 학습 구간에는 태스크들을 양 클러스터에 교대로 수행시키며

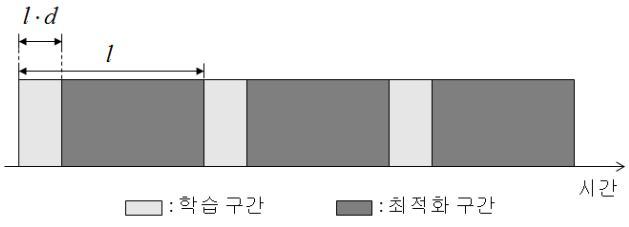

그림 2. 학습 구간과 최적화 구간

speedup 계산기의 동작을 돕는다. 최적화 구간에는 speedup 계산기로부터 받은 정보를 기반으로 태스크를 스케줄링 하고 변경 사항이 있는지 감시한다. 그림 2는 주기적으로 반복되는 학습 구간과 최적화 구간을 표시한다. l 은 반복 주기이고, d 는 한 주기 내에서 학습 구간이 차지하는 비율이다.

학습 구간에는 태스크들을 양 클러스터에 교대로 수행시킨다. 각 태스크를 직전까지 수행 중이던 클러스터에서 $(l \cdot d)/2$ 동안 수행시키고, 반대 클러스터로 이주시켜 $(l \cdot d)/2$ 동안 수행시킨다.

학습 구간이 끝나면 speedup 계산기의 결과를 기반으로 태스크들을 배치한다. 빅 코어의 개수가 r 일 때, 가장 speedup이 높은 r 개의 태스크를 빅 클러스터로 이주시키고 나머지는 리틀 클러스터에 둔다.

최적화 구간 동안 태스크가 이주되는 경우는 두 가지이다. 첫째, 빅 클러스터에서 수행 중이던 태스크가 수행을 종료했을 경우 리틀 클러스터에서 수행 중이던 태스크 중 가장 speedup이 높은 태스크를 빅 클러스터로 옮긴다. 둘째, 태스크의 `load_avg_ratio` 값이 빅 클러스터에서 문턱 값 δ 보다 더 많이 감소하거나 리틀 클러스터에서 δ 보다 더 많이 증가할 경우 반대 클러스터로 이주시킨다.

5. 실험 및 검증 결과

우리는 제안한 솔루션을 Linaro Stable Kernel 14.09를 탑재한 ARM 사의 Versatile Express TC2 보드에 구현하였다. 해당 보드는 3개의 ARM Cortex A7 코어와 2개의 ARM Cortex A15 코어로 동작한다. 메모리는 2 GB DDR2 램을 사용한다.

스케줄러의 검증을 위해 PARSEC 벤치마크를 사용하였다 [7]. PARSEC 벤치마크는 멀티 쓰레드 응용의 수행 성능을 평가하기 위해 사용된다. 우리는 이 중 태스크의 특성이 동적으로 변하는 streamcluster를 실험 워크로드로 사용하였다. 시스템의 처리량이 실제 어느 정도 증가하였는지를 확인하기 위해 벤치마크를 돌리며 전체 수행 시간을 측정하였다. d 값은 0.05, l 값은 10초, δ 값은 150을 사용했다.

실험 결과를 그림 3에 표시하였다. 세 번의 실험을 수행한 결과 기존 스케줄러의 수행 시간은 평균 294.4초이고, 제안한 스케줄러의 수행 시간은 평균 272.5초였다. 워크로드의 수행 시간을 7.44% 줄인다는 것을 확인하였다.

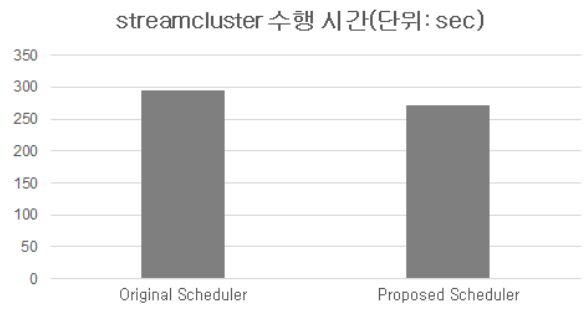

그림 3. 실험 결과

6. 결론

본 논문에서는 speedup이 높은 태스크를 고성능 코어에 배치시켜 이기종 멀티코어 시스템의 처리량을 극대화시킬 수 있는 스케줄러를 제안하였다. 이를 위해 speedup을 직접 계산하여 태스크를 배치한 후 변화를 관찰한다. 실험 결과 성능이 7.44% 향상했다.

향후에는 speedup 계산 알고리즘과 변화 관찰 모델을 고도화하여 speedup을 더욱 정밀하게 파악하여 처리량을 늘리는 알고리즘을 연구할 계획이다.

Acknowledgement

이 논문은 2015년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(No.R0101-15-237, 고집적 저전력 프로세서 기반 30% 이상 에너지 절감 범용 운영체제 및 가상화 핵심기술 개발)

참 고 문 헌

[1] D. Shelepov et al., "*HASS: a scheduler for heterogeneous multicore systems*", ACM SIGOPS Operating Systems Review, 2009.

[2] R. Kumar et al., "*Single-ISA heterogeneous multi-core architectures for multithreaded workload performance*", In Proc. of Int. Symp. on Computer Architecture, 2004.

[3] M. Becchi and P. Crowley, "*Dynamic thread assignment on heterogeneous multiprocessor architectures*", In Proc. of the 3rd Conf. on Computing Frontiers, 2006.

[4] D. Kaufaty et al., "*Bias scheduling in heterogeneous multi-core architectures*", In Proc. of the 5th European Conf. on Computer Systems, 2010.

[5] K. V. Craeynest et al., "*Scheduling heterogeneous multi-cores through performance impact estimation (PIE)*", In Proc. of Int. Symp. on Computer Architecture, 2012.

[6] S. Huh et al., "*Cross layer resource control and scheduling for improving interactivity in Android*", Software: Practice and Experience, 2014.

[7] C. Bienia, "*Benchmarking modern multiprocessors*", Ph.D. Thesis. Princeton University, 2011.