HSFS : 우월성을 이용한 이종 멀티코어 환경에서의 DAG 태스크 스케줄링 알고리즘

한종우O 이승수 박성현 이창건*
서울대학교 컴퓨터공학부
spiraline@snu.ac.kr, sslee@rubis.snu.ac.kr, seonghyeonpark@rubis.snu.ac.kr, cglee@snu.ac.kr

HSFS : Heterogeneous Superiority-First Scheduling for DAG Task

Jongwoo HanO Seungsu Lee Seonghyeon Park Chang-Gun Lee*
Department of Computer Science and Engineering, Seoul National University

요 약

본 논문은 이종 멀티코어(Heterogeneous Multi-core) 환경에서 DAG(Directed Acyclic Graph) 태스크 모델을 효과적으로 스케줄링하는 알고리즘을 제안한다. 자율주행 시스템의 상용화를 위해서는 임베디드 보드와 같이 자원은 제한되어있으나 값이 싼 컴퓨터를 사용해야 한다. DAG로 모델링 가능한 자율주행 시스템을 Nvidia TX2와 같은 이종 멀티코어 시스템의 임베디드 보드에서 수행하기 위해서는 적절한 스케줄링 알고리즘의 고안이 필요하다. 기존의 이종 멀티코어 환경에서의 DAG 태스크 스케줄링 알고리즘은 데드라인을 아예 고려하지 않아 데드라인 실패율이 크거나, 데드라인만을 고려하여 통신 지연에 의한 유휴 시간에 의해 스케줄 길이가 길어진다는 문제점이 있다. 본 논문에서는 그래프의 특성을 반영한 우월성이라는 개념을 도입하여 우월성과 데드라인을 모두 고려한 알고리즘을 제시한다. 또한 실험을 통해 제안한 알고리즘의 시간 복잡도와 데드라인 실패율은 기존 알고리즘과 유사하되 스케줄 길이는 평균 7.2% 줄어듬을 확인할 수 있었다.

1. 서론

자율주행의 상용화를 위해서 임베디드 보드 등의 값싼 컴퓨터를 사용할 필요가 있다. 그러나 임베디드 보드는 자원이 극도로 제한된 환경이기 때문에 성능이 일반 컴퓨터보다 떨어진다. 이를 해결하고자 Nvidia TX2와 같이 서로 다른 프로세서를 통합하여 성능과 전력 소모에서 모두 강점을 가지는 이종 멀티코어(Heterogeneous Multi-core) 시스템을 가진 임베디드 보드가 등장하고 있다[1]. 프로세서마다 태스크(Task)의 실행시간이 다른 이종 멀티코어 시스템에서는 그에 적합한 스케줄링 알고리즘이 필요하다.

자율주행 시스템을 구현하기 위해 자주 쓰이는 플랫폼은 ROS(Robot Operating System)다[2][3]. ROS를 사용하는 시스템은 이벤트 기반의 실행 흐름으로 여길 수 있어 각 태스크를 DAG(Directed Acyclic Graph)의 노드(node)로 모델링할 수 있다. DAG는 노드들 사이에 의존성이 존재하여 부모 노드가 실행되어야 자식 노드가 실행 가능하다는 특징을 가진다.

그러나, 일반적으로 멀티코어에서 최적의 DAG 태스크 스케줄링 방법을 찾는 문제는 NP-Complete로 알려져 있다[4][5]. 이에 따라 HEFT(Heterogeneous Earliest Finish Time), HLBS(Heterogeneous Laxity-Based Scheduling) 등의 휴리스틱(heuristic)을 적용한 알고리즘이 등장했다. DAG 태스크 스케줄링 문제에서의 알고리즘 평가 기준은 크게 (1) 데드라인 실패(deadline miss)의 발생률, (2) 스케줄의 길이(workspan), (3) 알고리즘의 시간 복잡도로 나뉜다. HEFT는 기존 알고리즘에 비해 시간 복잡도를 줄였다[6]. 이에 더하여 HLBS는 데드라인을 고려하여 HEFT 등의 기존 알고리즘에 비해 시간 복잡도는 유지하되 데드라인 실패율을 줄였다[7]. 그러나 데드라인이 여유로운데도 데드라인까지 남은 시간만을 고려한다면 불필요한 유휴 시간(idle time)이 발생하는 경우가 생긴다.

본 논문에서는 DAG 태스크의 선행관계를 바탕으로 하는 우월성이라는 새로운 개념을 도입하고, 우월성과 데드라인을 모두 고려한 HSFS (Heterogeneous Superiority-First Scheduling) 알고리즘을 제안한다. 또한 실험을 통해 제안하는 알고리즘을 사용하면 데드라인 실패율과 시간 복잡도는 기존 알고리즘과 유사하되 스케줄의 길이를 줄일 수 있다는 것을 보인다.

2. 배경

본 장에서는 이종 멀티코어 환경에서의 DAG 스케줄링을 위한 시스템 모델을 정의하고, 기존 알고리즘인 HEFT와 HLBS에 대해 소개한다.

2.1 시스템 모델

본 논문에서는 다음과 같은 태스크 모델을 가정한다.

1. DAG $G=(V,E)$를 이루는 V개의 태스크가 주어지며 각 태스크는 한 번만 실행된다.
2. 각 태스크 v_i는 데드라인 D_i과 프로세서 H_p에서의 실행시간 w_{ip}를 가진다. 평균 실행시간을 $\overline{w_i}$라 한다.
3. 각 태스크는 부모 태스크들이 모두 실행 완료된 후에 실행할 수 있다.
4. 자식 태스크 v_j가 부모 태스크 v_i와 다른 프로세서에서 실행되는 경우 c_{ij}의 통신 지연이 발생한다.

* 교신저자
본 연구는 미래창조과학부 및 정보통신기술진흥센터의 SW컴퓨팅산업원천 기술개발사업(SW스타랩)의 연구결과로 수행되었음 (IITP-2015-0-00209)
이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사드립니다.

즉, 이 경우 v_j는 적어도 v_i의 실행 완료 시각을 기준으로 c_{ij} 이후에 실행할 수 있다. 하지만 부모 태스크와 같은 프로세서에서 실행되는 경우 통신 지연은 무시할 정도로 작다.

5. 모든 태스크의 오프셋은 0이라고 가정한다. 즉, 모든 시작 태스크가 시간 0부터 실행할 수 있다.

2.2 HEFT (Heterogeneous Earliest Finish Time)

HEFT를 포함한 일반적인 태스크 스케줄링 알고리즘은 태스크의 우선순위를 선정하고, 우선순위가 가장 높은 태스크 순으로 적절한 프로세서에 할당하는 두 단계를 거친다. HEFT는 다음의 $rank_{heft}$를 사용해 태스크의 우선순위를 결정한다.

$$rank_{heft}(v_i) = \overline{w_i} + \max_{v_j \in \mathrm{succ}(v_i)} (rank(v_j) + c_{ij})$$

$\mathrm{succ}(v_i)$는 태스크 v_i의 자식 태스크들을 의미한다. 자식 태스크가 없는 끝 태스크에 대한 $rank_{heft}$는 $\overline{w_i}$다. HEFT는 끝 태스크로부터 시작 태스크까지 순회하며 $rank_{heft}$를 계산한다. 이후 $rank_{heft}$가 가장 높은 태스크에 대해 각 프로세서에서 실행되었을 때의 최단 실행 종료 시각 (Earliest execution Finish Time)을 계산하여 최단 실행 종료 시각이 가장 짧은 프로세서를 선택한다. 이때, 이미 할당된 두 태스크 사이에 통신 지연 등으로 인해 충분한 유휴 시간이 있는 경우 해당 시간에 삽입하여 할당한다. 태스크 v_i가 프로세서 H_p에서 실행될 때의 최단 실행 종료 시각 $EFT(v_i, H_p)$는 아래와 같다[6].

$$EFT(v_i, H_p) = w_{ip} + \max\Big(available(H_p), \max_{v_j \in pred(v_i)} \big(EFT(v_j, host(v_j)) + c_{ji}\big)\Big)$$

$pred(v_i)$는 태스크 v_i의 부모 태스크들을 의미한다. $host(v_j)$는 v_j가 할당된 프로세서를 의미하며 $available(H_p)$는 H_p에서 v_i의 실행을 시작할 수 있는 가장 이른 시간을 의미한다. 부모 태스크들인 $pred(v_i)$중 다른 프로세서에서 실행된 것이 있다면 통신 지연 시간을 합하여 $available(H_p)$과 비교했을 때 큰 값이 최단 실행 종료 시각이 된다.

2.3 HLBS (Heterogeneous Laxity-Based Scheduling)

HEFT는 데드라인에 대한 고려가 없었으나 HLBS는 데드라인을 고려한 $laxity$를 이용하여 태스크의 우선순위를 선정한다. $laxity$는 데드라인 실패가 없기 위해 실행되어야 하는 최소 시간을 의미하며 $laxity$가 작을수록 높은 우선순위를 가진다.

- 끝 태스크의 경우

$$laxity(v_i) = D_i - \overline{w_i}$$

- 끝 태스크가 아닌 경우

$$laxity(v_i) = \max_{v_j \in \mathrm{succ}(v_i)} \big(laxity(v_j) - \overline{c_{ij}}\big) - \overline{w_i}$$

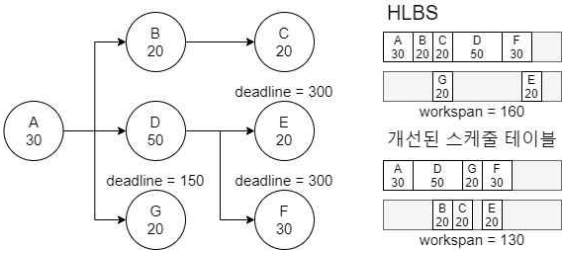
이후 $laxity$가 가장 작은 태스크부터 HEFT와 마찬가지로 최단 실행 종료 시각 $EFT(v_i, H_p)$가 가장 짧은 프로세서 H_p에 할당한다[7].

3. 제안하는 알고리즘

본 장에서는 제안하는 알고리즘인 HSFS를 소개하고, 기존 알고리즘인 HEFT, HLBS와의 성능 비교 실험 결과를 제시한다.

3.1 기존 알고리즘의 한계

멀티코어 환경에서는 모든 프로세서의 유휴 시간을 줄이는 것이 중요하다. 그러나, DAG 태스크 모델의 특성상 각 태스크는 부모 태스크가 모두 실행되기 전에는 실행할 수 없으므로 유휴 시간이 필연적으로 발생한다.



[그림 1] HLBS의 한계

[그림 1]의 DAG 태스크 모델에서 프로세서 수는 2개이며 모든 통신 지연 시간이 20이고 모든 프로세서에서의 실행시간이 노드에 쓰인 값과 같다고 가정하자. 데드라인만 고려하는 2.3의 HLBS 알고리즘을 이용한 스케줄 길이는 160이다. 그러나, 자식 수가 많은 D를 먼저 스케줄링하면 데드라인 실패 없이 스케줄 길이를 130으로 줄일 수 있다. 이를 통해 데드라인이 여유로울 때는 자식 태스크가 많은 태스크를 먼저 할당한다면 유휴 시간이 줄어들어 스케줄 길이가 줄어들 것이라고 기대할 수 있다.

3.2 HSFS (Heterogeneous Superiority First Scheduler)

HSFS는 데드라인을 기반으로 한 $laxity$ 외에 우월성(Superiority)이라는 요소를 도입하여 태스크의 우선순위를 정한다. 태스크 v_i의 우월성 $\mathrm{super}(v_i)$은 아래와 같이 계산한다.

$$\mathrm{super}(v_i) = 1 + \sum_{v_j \in \mathrm{succ}(v_i)} \mathrm{super}(v_j)$$

끝 태스크의 우월성은 1로 정의한다. 자식 태스크가 많을수록, level이 더 클수록 높은 우월성을 가지게 되며, 우월성이 높은 태스크부터 할당하면 3.1의 휴리스틱을 만족할 수 있다.

Algorithm 1: HSFS Algorithm

```
Input  : Target application representing DAG G(V,E)
Output: Scheduling of G(V,E)
while There are nodes with no values of laxity and superiority do
    Compute laxity, superiority by traversing graph from end node to
     entry node
end
Sort the tasks in ascending order of laxity and assign to L_lax;
Sort the tasks in descending order of superiority and assign to L_sup;
while There are unscheduling nodes do
    n_l ← node which has smallest laxity
    n_s ← node which has largest superiority
    if EFT(n_s) is larger than laxity(n_l) then
        Select n_l;
    else
        Select n_s;
    end
    for Exist available processor do
        Compute EFT using the insertion-based scheduling policy
    end
    Assign the task to the processor indicating the minimum EFT
end
```

[그림 2] HSFS의 의사 코드

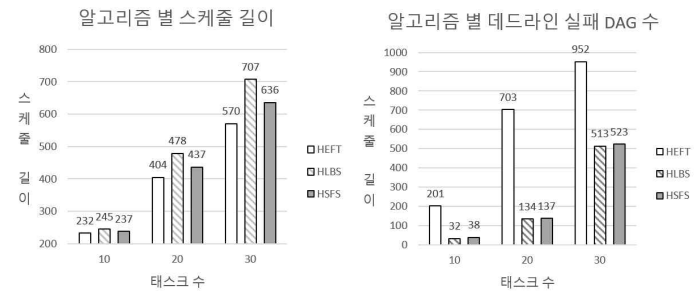
HSFS의 의사 코드는 **[그림 2]**와 같다. 먼저 그래프를 순회하며 우월성과 $laxity$를 모두 계산한 후, 우월성이 가장 높은 태스크 n_s의 EFT을 구한다. 이 때 $laxity$가 가장 작은 태스크 n_l의 $laxity$보다 EFT가 늦으면 n_l에 대해 데드라인 실패가 발생하게 될 수 있으므로, n_l부터 프로세서에 할당한다. 아닌 경우, n_s를 프로세서에 할당한다. 이 과정을 모든 태스크를 할당할 때까지 반복한다. HLBS에 비해 EFT와 $laxity$를 비교하는 부분만 추가되었기 때문에 시간 복잡도는 같다.

3.3 성능 비교 실험

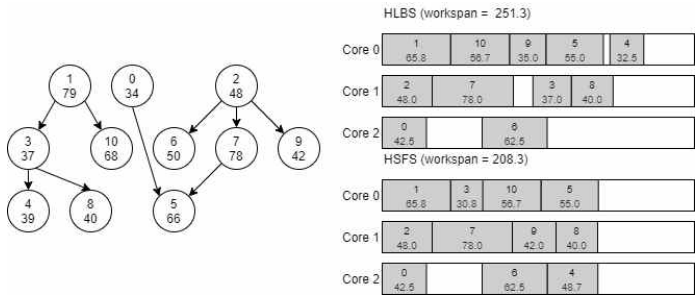
성능 비교를 위해 아래의 값들을 설정 가능한 무작위 DAG 그래프 생성 프로그램을 구현하였다.

- 태스크의 수, 프로세서의 수, 시작 태스크의 수
- 그래프의 깊이 : 태스크 수의 1/2 ~ 1/3
- 기본 프로세서 H_0에서의 태스크 실행시간 : 평균, 표준편차를 바탕으로 무작위로 설정한다.
- 프로세서별 성능 : H_0의 태스크 실행 효율을 1로 했을 때 각 프로세서별 성능을 의미한다. H_p에서의 태스크의 실행시간은 H_0에서의 실행시간을 H_p의 성능으로 나눈 값이다.
- CCR(Communication to Computation Ratio) : 태스크의 실행시간 대비 통신 지연 시간의 비율이다. CCR이 크면 통신 부하가 크고, 작으면 계산 부하가 큰 시스템이다.
- 끝 태스크의 데드라인은 해당 태스크의 레벨에 태스크의 이론적 실행시간을 곱한 값의 2배로 설정하였다.
- 최상위 레벨을 제외한 태스크들의 inbound edge는 1 이상이 되도록 하였다. 기본적으로 각 태스크에 대해 다음 레벨로 이어지는 아치(arc)만 생성하지만, 모든 DAG 그래프를 반영하고자 레벨 차이가 1 이상 나는 두 태스크에 대해서도 추가적인 아치를 태스크 수의 10%만큼 생성하였다.

비교 실험에서는 3개의 프로세서가 있다고 가정하였으며 실제 자율주행 시스템을 기반으로 하여 태스크의 수는 10, 20, 30개로 설정하였다[8]. 태스크의 이론적 실행시간의 평균은 50으로, CCR은 1.0으로 설정하였다. 프로세서별 성능은 1.2, 1.0, 0.8로 설정하여 첫 번째 프로세서의 성능이 가장 좋다고 가정하였다. 위 설정에 따라 무작위로 생성한 1000개의 DAG 태스크 모델에 대해 HEFT, HLBS와 제안하는 알고리즘의 스케줄 길이, 그리고 데드라인 실패 횟수를 비교하였다.



[그림 3] 태스크 수에 따른 알고리즘별 성능



[그림 4] HSFS의 스케줄 길이가 짧아지는 DAG 예시

[그림 3]에서 데드라인을 고려하지 않는 HEFT에 비해 데드라인을 고려하는 HLBS와 HSFS는 데드라인 실패율이 각각 평균 39.2%, 38.6% 낮은 것을 확인할 수 있다. 또한 스케줄 길이는 데드라인을 고려하는 HLBS가 HEFT에 비해 긴 것을 확인할 수 있는데, 제안하는 알고리즘인 HSFS는 데드라인을 고려함에도 불구하고 기존 알고리즘인 HLBS에 비해 스케줄 길이를 평균 7.2% 줄였음을 확인할 수 있다. **[그림 4]**는 HSFS가 HLBS에 비해 17.1% 짧은 스케줄 길이를 보여주는 DAG 형태이다. 자식 노드를 가지는 태스크 3을 먼저 실행하여 스케줄 길이에서 이득을 본 것을 확인할 수 있다.

4. 결론

본 논문에서는 우월성을 기반으로 한 DAG 태스크 스케줄링 알고리즘을 제안하였다. 제안한 알고리즘을 통해 시간 복잡도와 데드라인 실패율은 기존의 알고리즘의 값을 유지하되 스케줄 길이는 평균 7.2% 줄일 수 있다. 이는 데드라인 실패가 적다는 HLBS의 강점과 스케줄 길이가 짧다는 HEFT의 강점을 모두 가진 알고리즘이라고 할 수 있다. 제안하는 알고리즘을 주기적으로 반복되는 태스크에 적용한다면 전체 주기를 줄일 수 있을 것이다. 이를 통해 실시간 시스템에서 중요한 종단 간 반응 시간을 줄이는 데에 도움이 될 것으로 기대된다. 주기가 서로 같은 ROS 노드들로 이루어진 모듈을 사용할 경우, 본 이론을 실질적으로도 적용할 수 있을 것이다.

참고문헌

[1] N. Capodieci, et al., “Contending memory in heterogeneous SoCs: Evolution in NVIDIA Tegra embedded platforms,” in *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 1-10., 2020.

[2] Yukihiro Saito, et al., “Sensors fusion of a camera and a lrf on gpu for obstacle detection,” in *WiP session of the 2nd IEEE International Conference on Cyber-Physical Systems, Networks, and Applications*, 2014.

[3] S. Kato, et al., “An open approach to autonomous vehicles,” in *IEEE Micro*, vol. 35, no. 6, pp. 60-69., 2015.

[4] L. F. Bittencourt, et al., “Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm,” in *Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, pp. 27-34., 2010.

[5] M. Gary and D. Johnson, “Computers and Intractability: A Guide to the Theory of NP-Completeness,” 1979.

[6] H. Topcuouglu, et al., “Performance-effective and low-complexity task scheduling for heterogeneous computing,” in *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260-274., 2002.

[7] Y. Suzuki, et al., “HLBS: Heterogeneous laxity-based scheduling algorithm for DAG-Based real-time computing,” in *2016 IEEE 4th International Conference on Cyber-Physical Systems, Networks, and Applications*, pp. 83-88., 2016.

[8] S. Kato, et al., "Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems," in *ACM/IEEE 9th International Conference on Cyber-Physical Systems*, pp. 287-296., 2018.