[2024-4]

KEASE 기술서

한국형 엑사스케일 응용 SW 개발 환경 프레임워크

단일 컴퓨팅 노드 대상 경량 인과관계 프로파일러

2024. 04. 20

연 세 대 학 교

[개 정 이 력]

개정 일자	버전	설명	개정페이지	작성자	확인자
2024.4.20	1.0	INITIAL DRAFT	전체	정진규	정진규

목 차

1. 연구의 개요

본 문서는 단일 컴퓨팅 노드에서 실행되는 exaflop 급 HPC 응용들에 대한 인과관계 프로파일러의 설계 및 대표 응용들을 프로파일링한 결과를 분석한 문서이다. 인과관계 프로파일러는 실제 응용 최적화를 한 것과 같은 효과를 상대적으로 재현하기 위해 성능 향상 예측치를 확인하고자 하는 작업이 실행될 때, 동시에 실행되는 다른 작업들을 느리게 하는 것이다. 기존의 state-of-the-art 인과관계 프로파일러인 COZ[1]는 on-CPU 작업(i.e., CPU에서 실행되는 instruction)에 대한 성능 향상 예측치를 제공하지만, off-CPU 작업(i.e., CPU를 사용하지 않는 대기 구간)을 포함한 성능 향상 예측치를 제공하지 못한다.

본 문서에서 서술하는 개발한 인과관계 프로파일러의 범위는 HPC 응용 수행 시 발생하는 연산 작업과 같은 on-CPU 작업뿐만 아니라, CPU 스케줄러, thread간 동기화와 같은 off-CPU 작업을 포함한 성능 향상 예측치를 제공한다. 최종적으로, 개발한 인과관계 프로파일러를 통해 NPB(NAS parallel benchmark)의 integer sort 워크로드의 CPU 스케줄링으로 인한 병목을 검출하여 성능 향상 예측치의 결과를 검증하고, HPCG benchmark test 프로파일링을 통해 SYMGS(Symmteric Gauss seidel) 연산 및 OpenMP thread들의 동기화로 인한 대기 작업 관련 프로파일링 결과를 서술한다.

2. Off-CPU 작업 별 Virtual Speedup 수행 인과관계 프로파일러 설계

1) Off-CPU 작업 별 virtual speedup 수행의 필요성

Exaflop 급 HPC 응용의 경우 높은 연산 요구량을 만족시키기 위해 분산된 컴퓨팅 노드를 활용하며, GPU와 같이 연산에 특화된 가속기를 활용한다. 또한, 연산에 활용되는 대용량의 데이터를 저장하고 관리하기 위해 컴퓨팅 노드와 분리된 스토리지 노드를 구성하여 I/O를 수행한다. 추가적으로, HPC 응용의 경우 병렬 연산을 수행하기 위해 MPI, OpenMP 등의 라이브러리를 활용하며, 개별 연산 결과를 동기화하여 최종 연산 결과를 산출하는 작업이 동반된다. 즉, 연산 중심적(compute-intensive)인 응용이라도, 응용 컴퓨팅 환경의 발전에 따라 off-CPU 작업의 양은 증가하고 있으며, 이 작업들이 응용 성능에 미치는 영향을 파악하여 최적화의 필요성을 판단하는 프로파일러의 중요성은 증가하고 있다.

작업별 성능 향상 예측치를 제공하는 state-of-the-art 프로파일러인 COZ는 on-CPU 작업에 대한 성능 향상 예측치를 제공하지만, off-CPU 작업은 실행 정보가 존재하지 않아 올바른 성능 향상 예측치를 제공하지 못한다. 또한, COZ는 응용의 code line별 virtual speedup을 수행하여 성능 향상 예측치를 제공하며, GPU, 저장장치, 네트워크 장치 등 off-CPU 작업 전체를 가속할 수 있는 최적화에 대한 성능 향상 예측치를 제공하지 못한다. 본 문서에서 서술하는 인과관계 프로파일러는 발생하는 off-CPU의 종류별 virtual speedup을 수행하여 개별적으로 성능 향상 예측치를 제공하기에 사용자는 응용 코드를 수정하는 것과 더불어 시스템 하드웨어 교체와 같은 최적화의 필요성을 확인할 수 있다. 개발한 인과관계 프로파일러의 off-CPU 프로파일링 범위는 I/O, CPU 스케줄링, 동기화로 인한 대기 구간들과 응용 성능 간의 인과관계성을 제공하는 것이다.

---

1) Curtsinger, Charline, and Emery D. Berger. “COZ: Finding code that counts with causal profiling.” Proceedings of the 25th Symposium on Operating Systems Principles. 2015.

2) Blocked sample을 통한 off-CPU 작업 종류 파악

기존 리눅스 perf 서브시스템의 확장을 통해 off-CPU 작업 정보를 샘플링하는 blocked sample을 활용한다. Blocked sample은 본 연구진이 사전 연구를 통해 획득한 연구산출물로, on-CPU와 off-CPU 작업을 통합한 응용 실행 전체에 대한 샘플링 기능을 제공한다. Blocked sample 내부에는 I/O, CPU 스케줄링, 동기화, sleep 등에 의한 대기와 같은 off-CPU 작업의 원인을 기록하고 있다.

개발한 인과관계 프로파일러는 런타임에 응용 실행 정보를 담은 sample들을 확인하면서, off-CPU 작업에 해당하는 blocked sample을 읽고 처리하는 경우 내부에 저장된 발생한 원인을 확인한다. 만약, 현재 처리하는 샘플이 blocked sample이고, 기록되어 있는 off-CPU 작업이 virtual speedup을 하고자 하는 off-CPU 작업 종류와 일치할 때 동시에 실행되고 있는 다른 thread들을 의도적으로 느리게 하여 상대적인 최적화를 수행한다. 이 과정은 기존 COZ가 code line별 virtual speedup을 수행하는 과정과 동일하다. 최종적으로, 개발한 인과관계 프로파일러는 각 off-CPU 작업 종류별 성능 향상 예측치를 제공하여 사용자는 이를 통해 새로운 최적화 전략을 수립할 수 있다.

3) Virtual speedup target off-CPU 작업 선정 방식

기존 COZ는 virtual speedup target을 명시적으로 지정할수도 있고 응용 실행 시 자주 검출되는 code line으로 선정되도록 자동화할 수도 있는 옵션을 제공한다. 본 문서에서 서술하는 개발한 인과관계 프로파일러는 virtual speedup을 수행하고자 하는 off-CPU 작업의 종류를 명시적으로 지정하는 옵션을 제공하며, 자동화된 target 선정 수행 시 선정될 수 있는 후보에 자주 검출되는 code line과 함께 off-CPU 작업 종류를 추가한다. 이를 통해, 사용자는 응용 실행에 있어서 가장 큰 영향을 미치는 off-CPU 작업의 종류를 파악할 수 있으며, 실제 성능 향상을 도출할 수 있는 최적화를 수행할 수 있다.

4) 인과관계 프로파일러 사용법

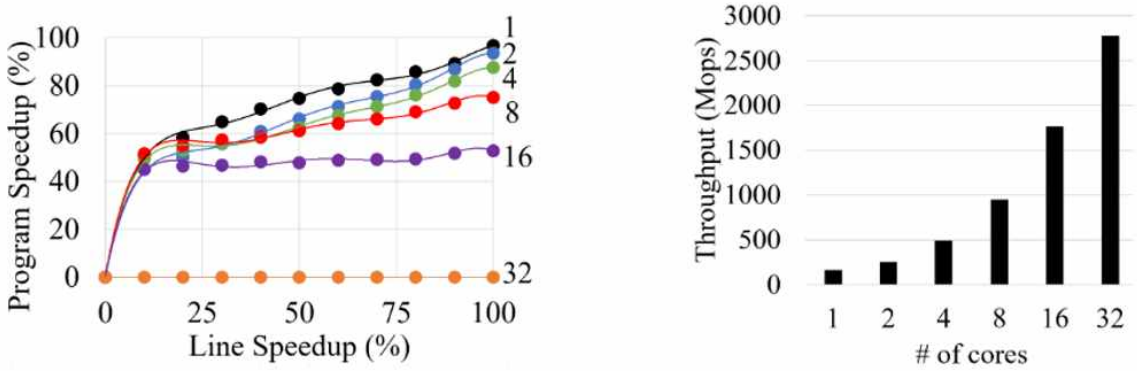
개발한 인과관계 프로파일러는 off-CPU 작업을 포함하기에 응용 실행 시 사용자 수준(user-level)의 instruction뿐만 아니라, thread가 block 되기 전 커널 수준 (kernel-level) instruction 정보까지 수집해야 한다. 하지만, kernel 진입 지점(entry point) 전 마지막으로 호출되는 glibc 라이브러리는 이전 frame pointer를 저장하고 있지 않아 callchain unwind가 불가능하다. Off-CPU 작업에 대한 정확한 frame pointer 유지를 위해 '-fno-omit-frame-pointer' 빌드 옵션을 통해 생성한 glibc 라이브러리를 활용하여 응용을 컴파일해야 한다.

응용 컴파일 시 사용해야 하는 옵션은 디버깅 정보를 포함하기 위한 '-g', dwarf 정보를 포함하기 위한 'gdwarf-3', frame pointer를 유지하기 위한 '-fno-omit-frame-pointer'를 활용해야 하며, linking 해야 하는 glibc 라이브러리 경로는 새로 빌드한 경로를 지정해야 한다.

3. 인과관계 프로파일러를 통한 Virtual Speedup 수행 결과

1) NPB(NAS parallel benchmark) Integer Sort 프로파일링

인과관계 프로파일링을 수행한 NPB는 OpenMP 버전을 실행했으며, 32개의 omp thread를 통해 병렬화된 연산을 수행하도록 했다. 또한, 의도적으로 CPU 스케줄링에 의한 대기를 조절하여 인과관계 프로파일링의 결과를 검증하기 위해 모든 omp thread를 사용하는 CPU core의 수를 1개부터 32개까지 변경하여 실행하도록 했다. 이론적으로, 개별 omp thread들은 약 3.1%($\frac{1}{32}$*100%)의 CPU를 점유하며, 나머지 96.9%의 시간 동안은 CPU 스케줄링에 의한 대기를 수행하고 있다.



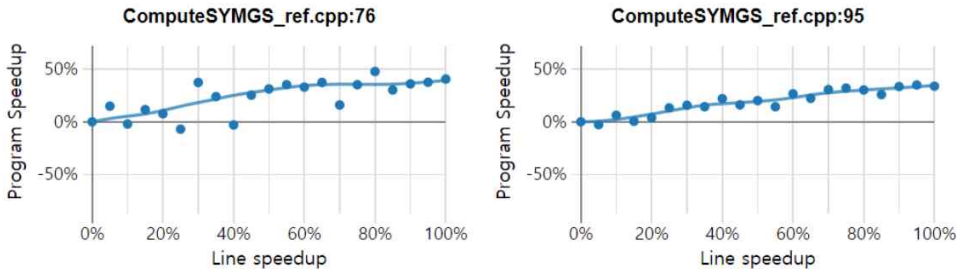
[그림 1] CPU 스케줄링에 인한 대기 작업의 성능 향상 예측치(좌) 및 실제 성능 향상(우)

[그림 1]은 사용하는 CPU core 개수에 따른 virtual speedup 결과(좌)와 실제 성능(throughput)의 변화(우)를 보여준다. 사용한 CPU core의 개수는 개별 virtual speedup 결과의 우측에 표기되어 있다. Virtual speedup 결과 그래프의 x축은 CPU 스케줄링이 해소된 정도이며, y축은 해소됐을 때의 응용 성능 향상 예측치이다. 즉, x축 값 50%가 의미하는 것은 CPU 스케줄링으로 인한 대기 구간이 절반으로 줄어든 상황(i.e., CPU core의 수가 2배가 된 상황)을 의미하며, 100%가 의미하는 바는 응용 실행이 CPU 스케줄링으로 인한 대기가 발생하지 않는(i.e., 각 omp thread가 개별적인 CPU core에서 실행되는) 상황을 의미한다.

NPB-is가 사용하는 CPU core의 수가 1개일 때 대비 32개일때의 성능 증가는 약 18.4배이다. 인과관계 프로파일러를 활용한 virtual speedup 결과는 CPU core가 1개일 때(그림 1좌 검정 그래프)의 최대 성능 향상 예측은 96.9%로 예측하고 있으며, 이를 대역폭 증가량으로 환산했을 때의 성능 향상 예측치는 약 31.2배이다. 전체적으로 virtual speedup 결과의 경향성은 CPU 스케줄링이 해소될수록 성능 증가가 늘어난다는 것과, CPU core가 많아질수록 성능 향상 예측치가 작아진다는 결과를 보여주며 이는 실제 성능 경향성과 유사하다. CPU 스케줄링에 의한 대기가 해소되었을 때 성능 향상 예측치와 실제 성능 향상 사이에 오차가 존재하는 것은 추가적인 원인 분석이 필요하다.

2) HPCG 벤치마크 프로파일링

인과관계 프로파일링을 활용한 HPCG 벤치마크의 프로파일링 결과는 그림 2와 같다.



[그림 2] HPCG benchmark의 인과관계 프로파일링 결과

```
Samples: 9K of event 'task-clock', Event count (approx.): 54095000000
  Overhead  Command      Shared Object      Symbol
+   83.94%  xhpcg_perf   libgomp.so.1       [L] gomp_barrier_wait_end
+    9.93%  xhpcg_perf   libgomp.so.1       [.] gomp_barrier_wait_end
+    3.63%  xhpcg_perf   xhpcg_perf         [.] ComputeSPMV_ref
+    2.28%  xhpcg_perf   libgomp.so.1       [.] gomp_team_barrier_wait_end
```

[그림 3] Blocked sample을 통한 HPCG benchmark 프로파일링 결과

[그림 2]의 그래프의 상단은 virtual speedup 대상으로 선정된 응용의 code line을 의미하며, 인과관계 프로파일링을 통해 ComputeSYMGS_ref.cpp 내부 SYMGS (Symmetric Gauss seidel) 연산 작업의 성능 향상 예측치가 가장 높은 것을 확인할 수 있었다. 추가적으로, blocked sample을 포함한 sampling을 수행한 결과는 그림 3과 같다. Blocked sample을 통해 OpenMP 라이브러리 내부 동기화 작업으로 인한 omp thread들의 대기 구간(gomp_barrier_wait_end)가 차지하는 비중이 약 83.9%인 것을 확인할 수 있었다. 결론적으로, blocked sample 및 이를 활용한 인과관계 프로파일링을 통해, HPCG benchmark의 응용 실행 시 수행되는 SYMGS 연산 kernel 작업은 순차화(serialize)되어 있으며, 해당 커널 최적화의 필요성을 파악할 수 있었다. 해당 병목은 잘 알려진 HPCG benchmark의 병목이며, 기존 최적화 연구들을 파악하여 실제 최적화를 수행하는 것을 계획 중에 있다.