[2025-5]

KEASE 기술서

한국형 엑사스케일 응용 SW 개발 환경 프레임워크

멀티 프로세스 응용 지원 경량 인과관계 프로파일러

2025. 3. 10

연 세 대 학 교

[개 정 이 력]

개정 일자	버전	설명	개정페이지	작성자	확인자
2025.03.10	1.0	INITIAL DRAFT	전체	정진규	

목 차

1. 개요

본 문서는 2차년도 연구 목표인 MPI 라이브러리 기반 멀티 프로세스 병렬 컴퓨팅을 수행하는 HPC 응용을 지원하는 인과관계 프로파일러의 설계 및 대표 응용인 HPCG의 성능 프로파일링 결과를 분석한 문서이다. 특히, 2차년도의 인과관계 프로파일링 대상 작업에는 프로세스간 통신 작업이 포함되며, 해당 작업은 응용이 분산된 다중 컴퓨팅 노드를 활용하는 경우 물리적으로 network I/O 작업으로 이어진다.

본 문서에서 서술하는 개발한 멀티 프로세스 응용 지원 인과관계 프로파일러는 프로세스간 통신 작업 실행 정보를 샘플링하는 task-clock-plus 이벤트를 Linux perf subsystem에 새로 추가하였다. 또한, task-clock-plus 이벤트 샘플링을 통해 획득한 응용 실행 정보를 기반으로 virtual speedup을 수행하여 성능 향상 예측치를 제공하는 인과관계 프로파일러를 개발하였다. 인과관계 프로파일러가 멀티 프로세스 응용을 지원하기 위해서는 응용 실행 도중에 개별 virtual speeedup 실행에 대한 세부 정보가 프로세스간 공유가 이뤄져야 한다. 이를 위해, Linux의 shared memory API(i.e., shmat, shmget)를 활용하여 프로세스간 공유되는 페이지를 생성하고, virtual speedup 실행 세부 정보(e.g., 성능 향상 예측 대상 코드 line, speedup 크기 등)를 공유하도록 구현하였다.

최종적으로, 본 문서에서는 개발한 멀티 프로세스 지원 인과관계 프로파일러를 활용한 MPI 기반 HPCG 응용 실행의 프로파일링 결과를 분석하고, 이를 해결하기 위한 최적화 전략을 제시한다.

2. 멀티 프로세스 지원 경량 인과관계 프로파일러 설계

1) 프로세스간 통신 작업 정보를 제공하는 Linux perf subsystem 확장

1차년도 기술서에 서술한 인과관계 프로파일러는 멀티 쓰레드 기반 병렬 응용 프로그램에 국한되어 멀티 프로세스 응용 성능 향상 예측에 활용될 수 없다. 본 기술서에는 BCOZ에 MPI 라이브러리를 지원하도록 기능을 확장하였고, 이를 위해 프로세스간 통신 작업 실행 정보를 수집하는 샘플링 기법이 필요하다.

이를 위해, blocked samples 기반 샘플링 기법을 개별 프로세스 및 쓰레드별 실행 정보를 수집하는 데 활용했다. 추가적으로, 기존의 Linux perf subsystem의 task-clock 이벤트를 샘플링하는 동작을 바탕으로 프로세스간 통신으로 인한 off-CPU 작업 정보를 포함한 실행 정보를 수집하는 task-clock-plus 이벤트를 새로 구현했다.

또한, blocked samples의 샘플링 결과에 대한 통계 정보를 제공하는 bperf 도구에 task-clock-plus 이벤트 샘플링을 지원하도록 확장하였으며, 이는 인과관계 프로파일링을 통해 개별 작업의 성능 향상 예측치를 확인하기 전에 대략적인 응용의 실행 특성과 병목을 파악하는 데 유용하게 활용될 수 있다.

bperf를 활용한 task-clock-plus 이벤트 샘플링을 수행하는 커맨드는 다음과 같다.

- $bperf record –g –e task-clock-plus –c 1000000 --weight ./[command]

여기서 '-g'는 callchain 샘플링을 의미하며, '-e task-clock-plus'는 샘플링 대상 이벤트, '-c 1000000'은 샘플링 주기(=1ms), '--weight'는 경량화된 샘플링을 위한 샘플 병합을 의미한다.

현재 task-clock-plus 이벤트는 최신 리눅스 커널 버전(v6.13)에 구현되었으며, 오픈소스(https://github.com/s3yonsei/linux-blocked_samples)화하여 제공하고 있다. 멀티 프로세스 응용에 대해 인과관계 프로파일러를 구동하기 위해서는 해당 오픈소스 커널을 설치해야한다.

2) MPI+OpenMP 형태의 응용 실행을 지원하는 인과관계 프로파일러 설계

멀티 프로세스 실행을 지원하기 위해 기존 멀티 쓰레드 응용을 지원하던 인과관계 프로파일러를 확장하였으며, 다음과 같은 두 가지 구현이 필요하다.

첫 번째는, 전술한 task-clock-plus 이벤트를 활용하여 프로세스간 통신 작업 정보를 포함한 응용의 실행 정보를 virtual speedup을 수행하는 데 활용하도록 구현해야한다. 이를 위해, BCOZ가 virtual speedup을 수행할 때 성능 향상 예측 대상 작업의 실행 여부를 파악하는데 task-clock-plus 이벤트 샘플링을 활용하여 개별 프로세스/쓰레드의 instruction pointer 및 callchain 정보를 수집하도록 구현했다.

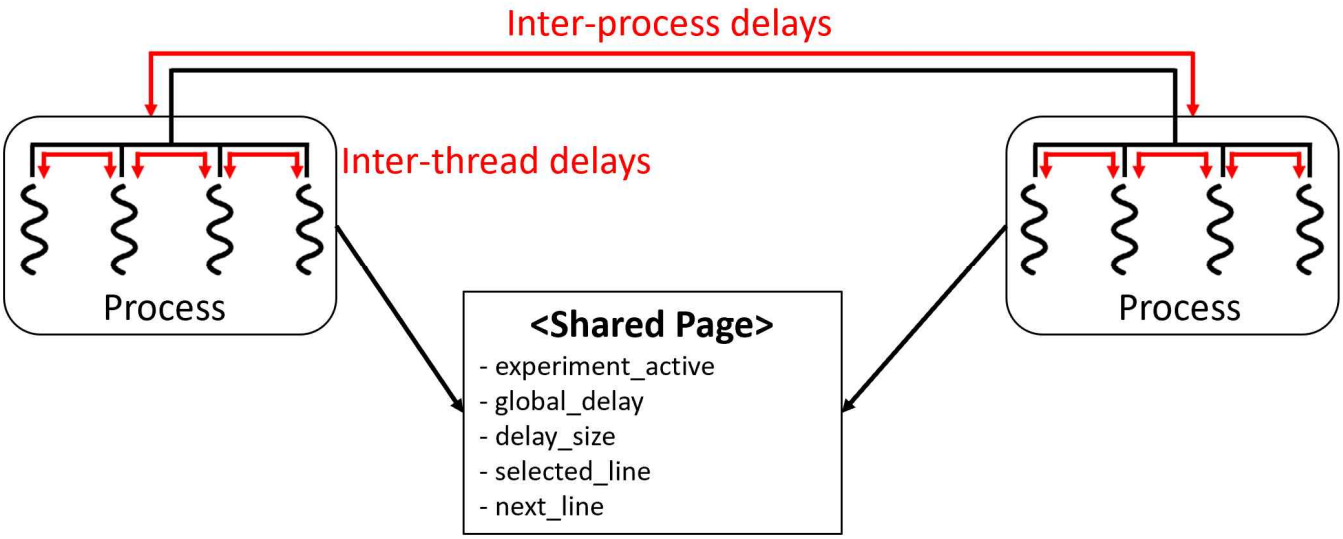
두 번째는, virtual speedup 수행 정보가 병렬적으로 실행되고 있는 프로세스간 공유가 되어야한다. 하지만, 프로세스간 공유되는 메모리 공간이 존재하지 않아 기존 BCOZ는 MPI 라이브러리를 활용한 HPC 응용에 대해 올바른 virtual speedup을 수행할 수 없다. Virtual speedup 수행 시 공유되어야 하는 세부 정보 및 이에 대한 설명은 아래의 표와 같다.

변수명	설명
experiment_active	응용 실행 중 experiment(i.e., virtual speedup 수행 단위)의 활성화 여부를 판단하는 boolean 변수임. 프로세스간 이 변수를 공유하지 못한다면 특정 프로세스는 virtual speedup 대상에서 제외될 수 있음.
global_delay	Virtual speedup 수행 시, 성능 향상 예측 대상 작업이 실행될 때마다 동시에 실행되고 있는 다른 프로세스/쓰레드들에게 부여되는 총 delay 크기를 의미함. 이를 기반으로 개별 프로세스/쓰레드는 자신이 delay 되어야하는 양을 계산하기 때문에 global_delay가 공유되지 않는다면 올바른 virtual speedup 결과를 획득할 수 없음.
delay_size	Virtual speedup 수행 시 성능 향상 예측 대상 작업이 관측될 때마다 다른 프로세스/쓰레드에게 부여하는 delay 크기를 의미함. Delay의 크기는 대상 작업이 가상적으로 빨라지는 정도에 의해 결정되며, 이를 공유하지 못하면 virtual speedup 결과를 부정확하게 함.
selected_line	Virtual speedup 수행 대상 작업을 의미하며, 개별 프로세스/쓰레드는 주기적으로 수집된 샘플들을 확인하여 대상 작업이 수행되고 있는 지 여부를 판단함. 따라서, 이를 공유하지 못한다면 delay 부여 횟수가 부정확해짐.
next_line	다음 virtual speedup 수행의 성능 향상 예측 대상 작업을 의미하며, 이는 이전 virtual speedup 수행에서 빈번하게 샘플링 된 작업 정보를 기반으로 결정됨. next_line 변수를 공유하지 못하면 다음 virtual speedup 수행 대상 정보(즉, selected_line)이 올바르지 않아 마찬가지로 올바른 virtual speedup을 수행할 수 없음.

<프로세스간 공유되는 virtual speedup 수행 정보>

프로세스간 공유되어야 하는 변수는 Linux의 shared memory API(i.e., shmat, shmget)를 활용하여 공유되는 페이지를 생성하고, 공유가 필요한 변수를 해당 페이지에 저장하도록 하여 원활한 공유가 이뤄지도록 구현했다.

최종적으로, 위의 두 가지 구현을 통해 개발한 멀티 프로세스 지원 인과관계 프로파일러의 개요도는 아래의 그림과 같다. 또한, 개별 프로세스/쓰레드는 다음과 같은 작업을 주기적으로 수행하도록 구현했다.



<멀티 프로세스를 지원하는 인과관계 프로파일러 개요도>

- (1) Virtual speedup을 수행하지 않지 않으면(i.e., experiment active가 false면) 다음 주기를 기다림.
- (2) 누적된 처리되지 않은 task-clock-plus 샘플링 결과를 읽음
- (3) 개별 샘플들을 탐색하며 예측 대상 작업(i.e., selected_line)이 수행되고 있는 지 여부를 판단함.
- (4) 예측 대상 작업이 관측된다면, 현재 global_delay 값에서 delay_size를 제외한만큼 sleep함. 만약, global_delay가 delay_size 보다 작다면, global_delay를 증가시켜 다른 프로세스/쓰레드들에게 delay를 부여함.
- (5) 샘플링 결과에 자주 관측되는 작업을 기록하여 next_line을 결정함.

3) 인과관계 프로파일러 사용법

우선, 본 인과관계 프로파일러를 활용하기 위해서는 새로 개발한 task-clock-plus 이벤트를 지원하는 리눅스 커널을 설치해야한다. 이는 현재 GIthub을 통해 오픈소스 형태로 제공하고 있으며, 링크는 다음과 같다(https://github.com/s3yonsei/linux-blocked_samples).

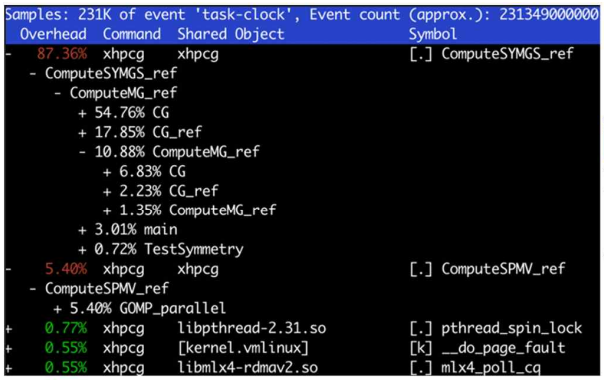
개발한 인과관계 프로파일러는 off-CPU 작업을 포함하기에 응용 실행 시 사용자 수준(user-level)의 instruction 뿐만 아니라, thread가 block 되기 전 커널 수준(kernel-level) instruction 정보까지 수집해야한다. 하지만, kernel 진입 지점(entry point) 전 마지막으로 호출되는 glibc 라이브러리는 이전 frame pointer를 저장하고 있지 않아 callchain unwind가 불가능하다. Off-CPU 작업에 대한 정확한 frame pointer 유지를 위해 '-fno-omit-frame-pointer'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

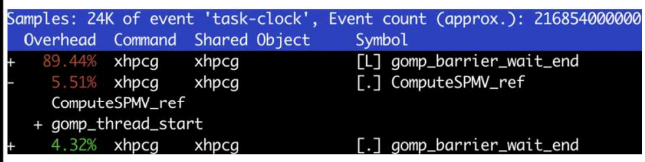
3. 멀티 프로세스 HPCG 응용 성능 프로파일링 결과

1) task-clock-plus 이벤트를 활용한 bperf 샘플링 결과

첫 번째로, task-clock-plus 이벤트 샘플링에 대한 통계 결과를 제공하는 bperf를 활용한 프로파일링 결과는 아래 그림과 같다. 실험은 Intel KNL(knights landing) 서버를 활용했으며, HPCG는 MPI 프로세스 4개, 각 프로세스 당 64개의 omp 쓰레드를 생성하여 실행하였다.



<MPI 프로세스 대상 샘플링 결과>



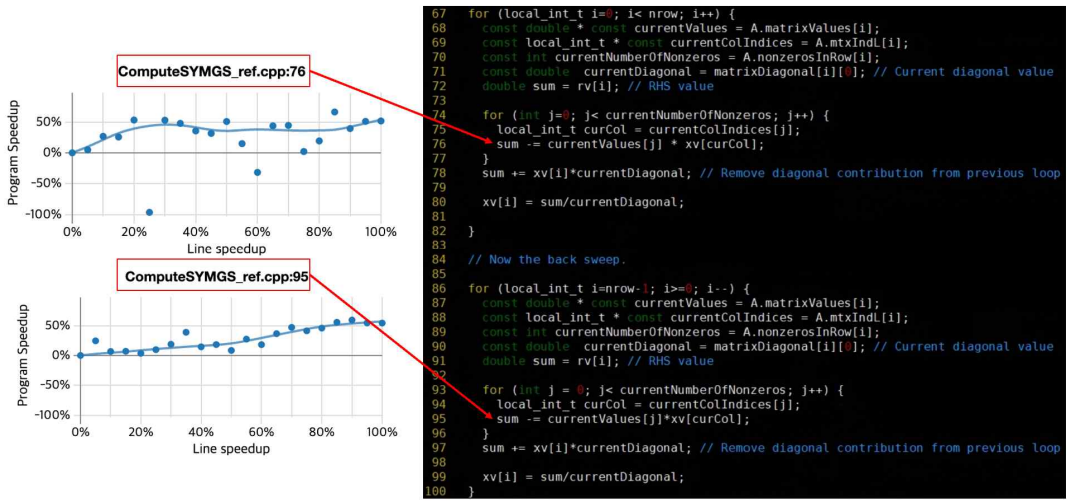
<OMP 쓰레드 대상 샘플링 결과>

<HPCG 샘플링 결과. (좌) MPI 프로세스 작업 샘플링, (우) OMP 쓰레드 작업 샘플링 결과.>

결과적으로, 샘플링 결과를 통해 특정한 HPCG의 성능 병목은 MPI 프로세스가 실행하는 SYMGS(symmetric gauss seidel) 컴퓨팅 커널이며, 모든 omp 쓰레드는 대부분 수행할 작업이 없어 대기하고 있는 lock 관련 off-CPU 작업(i.e., gomp_barrier_wait_end)을 수행하고 있음을 알 수 있다. 샘플링 결과를 통해 SYMGS 컴퓨팅 커널 실행은 직렬화(serialize)되어 있다는 것을 알 수 있으며, HPCG의 성능 병목은 멀티 프로세스화 됨에도 컴퓨팅 작업이 여전히 성능 병목임을 파악했다.

2) 인과관계 프로파일러를 활용한 성능 향상 예측 결과

동일한 HPCG 실행에 대한 인과관계 프로파일링 결과는 아래 그림과 같다. 인과관계 프로파일링 결과 그래프의 상단은 성능 향상 예측 대상 코드 line을 의미(i.e., [파일이름]:[line 번호])하며, x축은 대상 작업의 성능 향상 정도, y축은 그에 뜨른 응용의 성능 향상 정도를 의미한다.

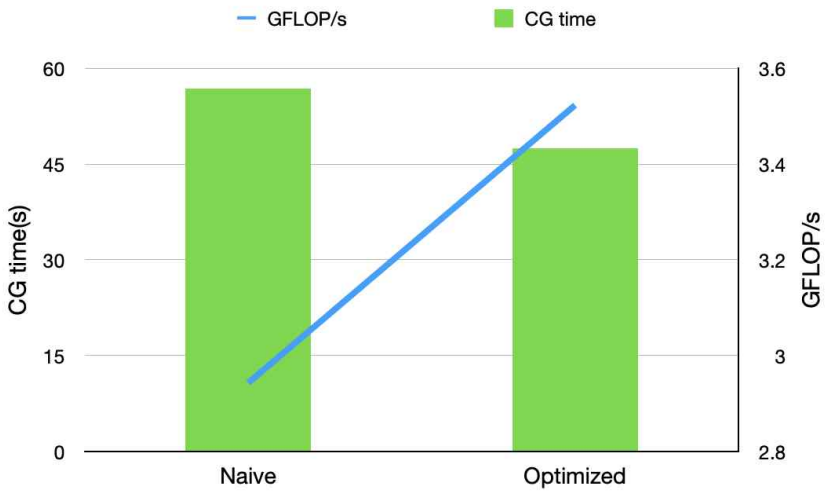


<멀티 프로세스 HPCG 실행에 대한 인과관계 프로파일링 결과>

멀티 프로세스를 지원하는 BCOZ를 활용한 인과관계 프로파일링을 통해 SYMGS 커널 내부 성능 병목을 유발하는 코드 line을 특정할 수 있다. 특정된 코드 line은 SYMGS 커널 내부 희소행렬(sparse matrix)인 xv를 접근하는 작업이며, 랜덤 접근으로 인해 캐시 미스(cache miss)가 발생하는 작업임을 알 수 있다. 또한, 행렬 연산에 있어 연속적인 행 사이에 의존성(dependency)가 존재하여 SYMGS 커널 실행은 직렬화 되어 있을 수 밖에 없음을 알 수 있다. 이와 같은 HPCG 응용의 병목을 최적화하기 위해 다음과 같은 전략을 수립하고 이를 구현했다.

첫 번째는 희소행렬 접근 시 발생하는 cache miss 문제를 해결하기 위해, xv 행렬의 한 행에서 non-zero 값들을 미리 prefetch하는 동작을 구현했다. 또한, 개별 행 컴퓨팅 사이의 의존성을 최소화하기 위해 계수 행렬(i.e., A 행렬)을 대각 값을 기준으로 파티셔닝(partitioning)하여 의존성이 존재하는 하삼각, 의존성이 존재하지 않는 상삼각으로 구분하여 일부 연산에 대해 병렬화를 수행했다.

이와 같은 최적화를 통해 달성한 HPCG 멀티 프로세스 실행의 성능 향상은 아래 그래프와 같다. 본 실험은 두 대의 KNL 노드를 활용하여 최적화 적용에 따른 성능 개선 정도를 측정한 결과이다.



<멀티 프로세스 HPCG 응용 최적화 결과>

성능 개선 기준은 개별 문제를 solving하면서 SYMGS 커널을 반복적으로 호출하여 CG 컴퓨팅 커널 실행 시간(i.e., CG time)과 초당 부동 소수점 연산 처리량(i.e., GFLOPS)를 활용했다. 최종적으로, 본 기

술서에서 서술한 인과관계 프로파일러를 통해 SYMGS 커널 내부 병목을 특정하였고, prefetch를 통한 cache miss 최소화, 행렬 파티셔닝을 통한 연산 의존성 최소화 최적화 기법을 통해 MPI 기반 HPCG 응용 실행의 연산 처리량을 16.47% 증가시켰고, CG 커널의 실행 시간을 16.50% 감소시켰다.