차세대 에너지 관리시스템: 상정사고 해석 프로그램 성능 개선 방안 분석

배애경*, 강호영*, 김영인*, 김홍주*, 신용학*
*LS산전 안양연구소 시스템S/W연구단
e-mail : akbae@lsis.com

The Next Generation Energy Management System: Analysis and Performance Improvement of the Contingency Analysis Program

Ae-Kyoung Bae*, Ho-Young Kang*, Young-In Kim*, Hong-Joo Kim*, Yong-Hark Shin*
*System Software R&D Center, LSIS Co., Ltd.

요 약

차세대 에너지 관리시스템에서 계통 해석 프로그램은 집중원격감시 시스템으로부터 취득된 데이터를 이용해 토폴로지, 상태추정, 조류계산, 상정사고, 고장해석 등의 해석 정보를 계통 운영자에게 전달한다. 지속적으로 증가하는 전력수요에 신속하게 대처하기 위해서 계통 해석 프로그램은 대규모 계통의 계획과 운영을 빠르고 정확하게 분석하여 대책을 수립할 수 있어야 한다. 본 논문에서는 계통 해석 프로그램들 중 특히 많은 시간이 소요되는 상정사고 해석 프로그램의 특성을 분석하고 성능 개선 방안을 제안한다.

1. 서론

차세대 에너지 관리시스템(Energy Managemenet System, EMS)에서 계통 해석 프로그램은 집중원격감시 시스템(Supervisory Control and Data Acquisition, SCADA)로부터 취득된 데이터(계통을 구성하는 설비들의 상태, 값 등)를 이용해 토폴로지(Topology Processing, TP), 상태추정(State Estimation, SE), 조류계산(Dispatch power flow, DPF), 상정사고(Contingency Analysis, CA), 고장해석(Short circuit analysis, SCA) 등의 해석 정보를 계통 운영자에게 전달한다. 이러한 계통 해석 프로그램은 지속적으로 증가하는 전력수요에 신속하게 대처하기 위해 대규모 계통의 계획과 운영을 빠르고 정확하게 분석하여 대책을 수립할 수 있어야 한다.

본 논문에서는 계통 해석 프로그램들 중 특히 많은 시간이 소요되는 상정사고 해석 프로그램의 특성을 분석하고 프로그램 구현 레벨 및 컴파일 레벨에서의 최적화 작업을 수행한 결과를 기술한다. 또한 수행 결과를 토대로 상정사고 해석 프로그램의 성능 개선 방안을 제안한다.

2. 상정사고 해석 프로그램

상정사고 해석 프로그램[1]은 전력 시스템을 구성하는 설비(송전선로, 변압기, 차단기, 부하, 발전기 등)에 대해 가상으로 고장을 일으켰을 때, 전력 시스템에 미치는 영향을 계산하는 프로그램이다.

상정사고 해석 프로그램은 계통 운영자가 안정적으로 계통을 운영할 수 있도록 전력 설비에서 실제 일어날 수 있는 고장에 대해 영향을 분석하여 정보를 제공해야 한다. 하지만 전력 계통 내에는 수많은 전력 설비가 있고, 각 설비에서 일어날 수 있는 고장 상황 및 그 조합이 무수히 많기 때문에 이들 모두의 상정사고에 대해 분석하는 것은 많은 시간과 노력이 소요된다.

이를 개선하기 위해 상정사고 해석 프로그램을 분석하고 향후 프로그램의 성능 개선 방안을 모색하고자 한다.

3. 성능 개선 방안 적용

3.1 실험 환경

- OS: HP-UX aykemsSB B.11.23 U IA6 (RX4640)
- Compiler: HP C/aC++ B3910B A.06.15

3.2 코드 작성

표 1은 상정사고 해석 프로그램을 프로파일링 하여 시간이 많이 소요되는 원인을 찾아 작성한 샘플 코드의 일부다. 상정사고 해석 프로그램은 주로 수 천 개의 상정사고에 대해 연산을 수행하기 위한 loop() 연산과 연산에 필요한 많은 데이터를 memcpy()를 이용하여 복사하게 되는데 이 과정에서 많은 시간이 소요됨을 확인하였다.

<표 1> 상정사고 해석 프로그램 분석을 위한 샘플 코드

```
sample code :
                    ...
start: Elapsed time<for>
for(int i=0; i<MAX_A; i++)
{
        for(int j=0; j<MAX_B; j++)
        {
                array_for_copy[j] = j;
        }

        start: Elapsed time<memcpy>
        function_for_memcpy();
        end: Elapsed time<memcpy>
}
end: Elapsed time<for>
                    ...
```

표 1의 샘플 코드를 기본으로 3.3과 3.4의 최적화를 수행하였다. 최적화 전/후 성능 개선의 지표는 Elapsed time<for>와 Elapsed time<memcpy>를 사용하였다.

3.3 코드 최적화

C/C++ 프로그램의 메모리, 속도 최적화를 위한 가이드를 참조하여 다음의 코드 최적화를 적용해 보았다.

(1) 전역 변수 사용 억제
(2) 포인터 사용
(3) loop() 변형

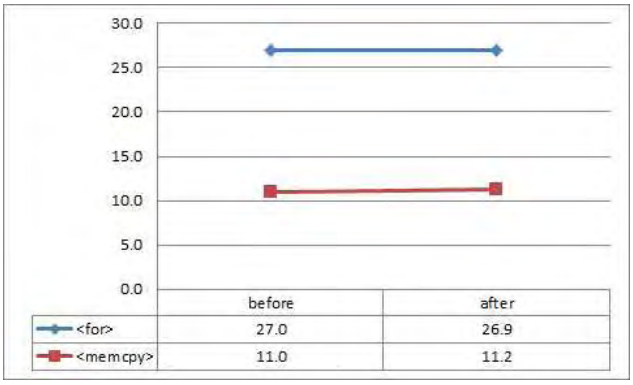
표 2는 코드 최적화를 수행한 결과다.

<표 2> 코드 최적화에 따른 성능 측정

코드 최적화 전					
1회(sec)		2회(sec)		3회(sec)	
(1)	(2)	(1)	(2)	(1)	(2)
26.5	11.0	27.2	10.8	27.9	11.4
4회(sec)		5회(sec)		평균(sec)	
(1)	(2)	(1)	(2)	(1)	(2)
26.4	10.8	26.9	10.8	27.0	11.0

코드 최적화 후					
1회(sec)		2회(sec)		3회(sec)	
(1)	(2)	(1)	(2)	(1)	(2)
27.1	10.9	27.0	11.2	27.1	11.9
4회(sec)		5회(sec)		평균(sec)	
(1)	(2)	(1)	(2)	(1)	(2)
26.5	10.7	26.7	11.1	26.9	11.2

그림 1은 코드 최적화에 따른 성능을 그래프로 나타낸 것이다. 간단하게 적용해 볼 수 있는 코드 최적화 수행으로는 성능 향상이 되지 않는다.



<그림 1> 코드 최적화에 따른 성능 비교

3.4 컴파일 최적화

표 3는 HP-UX[2]에서 제공하는 컴파일 옵션을 적용하여 최적화를 수행한 결과다. OS Schedule Time의 영향을 받아 발생하는 오차를 고려하여 동일한 컴파일 옵션에 대해 총 5번의 시행을 통해 평균을 구하였다.

(1): <for>, (2): <memcpy>

<표 3> 컴파일 최적화 옵션 적용에 따른 성능 측정

옵션		설명			
+O1		일반 디버그			
1회(sec)		2회(sec)		3회(sec)	
(1)	(2)	(1)	(2)	(1)	(2)
26.5	11.0	27.2	10.8	27.9	11.4
4회(sec)		5회(sec)		평균(sec)	
(1)	(2)	(1)	(2)	(1)	(2)
26.4	10.8	26.9	10.8	27.0	11.0

옵션		설명			
+O3		일반 릴리즈			
1회(sec)		2회(sec)		3회(sec)	
(1)	(2)	(1)	(2)	(1)	(2)
15.7	12.2	15.5	12.0	15.5	12.6
4회(sec)		5회(sec)		평균(sec)	
(1)	(2)	(1)	(2)	(1)	(2)
15.2	11.4	15.2	11.1	15.4	11.9

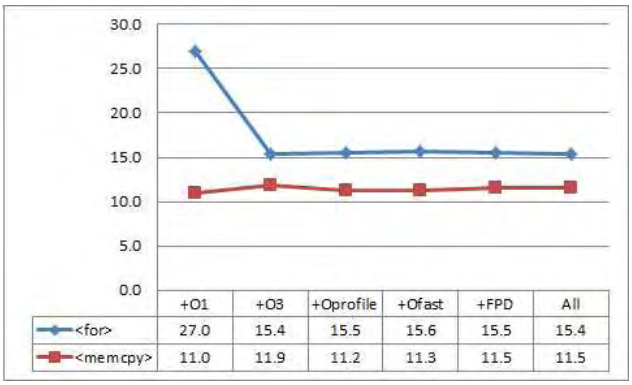
옵션		설명			
+Oprofile		프로파일 최적화			
1회(sec)		2회(sec)		3회(sec)	
(1)	(2)	(1)	(2)	(1)	(2)
15.2	9.6	16.1	12.0	15.2	11.2
4회(sec)		5회(sec)		평균(sec)	
(1)	(2)	(1)	(2)	(1)	(2)
15.0	12.1	15.9	11.1	15.5	11.2

옵션		설명			
+Ofast		O3 및 기타 최적화 옵션			
1회(sec)		2회(sec)		3회(sec)	
(1)	(2)	(1)	(2)	(1)	(2)
14.7	10.2	15.7	11.2	16.2	11.9
4회(sec)		5회(sec)		평균(sec)	
(1)	(2)	(1)	(2)	(1)	(2)
15.6	12.3	15.7	11.0	15.6	11.3

옵션		설명			
+FPD		부동 소수점 연산 최적화			
1회(sec)		2회(sec)		3회(sec)	
(1)	(2)	(1)	(2)	(1)	(2)
16.2	11.8	15.6	12.4	15.3	11.0
4회(sec)		5회(sec)		평균(sec)	
(1)	(2)	(1)	(2)	(1)	(2)
15.0	10.2	15.4	11.9	15.5	11.5

옵션		설명			
All		All			
1회(sec)		2회(sec)		3회(sec)	
(1)	(2)	(1)	(2)	(1)	(2)
15.4	11.2	15.0	10.7	15.7	12.4
4회(sec)		5회(sec)		평균(sec)	
(1)	(2)	(1)	(2)	(1)	(2)
15.6	11.4	15.3	12.0	15.4	11.5

그림 2는 컴파일 최적화 옵션 적용에 따른 성능을 그래프로 나타낸 것이다. +O3, 프로파일링, 부동소수점, 기타 최적화 옵션을 적용했을 때 일반 디버그 모드인 +O1에 비해 성능 향상이 되었지만, 최적화 옵션들 간 추가적인 성능 향상의 징후는 보이지 않는다.



<그림 2> 컴파일 최적화 옵션 적용에 따른 성능 비교

4. 결론

차세대 에너지 관리시스템의 상정사고 해석 프로그램 성능을 개선하기 위해 프로그램 분석을 수행하였다. 프로그램 분석 후 시간이 많이 소요되는 연산의 최적화를 시도하였다. 간단하게 적용해 볼 수 있는 전역 변수 사용 억제, 포인터 사용, loop() 변형 등의 코드 최적화와 최적화 컴파일 옵션을 적용해 본 결과 성능 향상이 크게 이루어지지 않음을 확인하였다.

계통 내 모든 상정고장은 비종속적이며 특정 고장 하나에 대해 상정고장 해석은 독립적인 문제로 간주 할 수 있기 때문에 향후 병렬 프로그래밍을 적용하여 성능 최적화를 수행해 볼 계획이다.

참고문헌

[1] 조윤성, 한국형 에너지 관리시스템용 상정고장 해석프로그램 개발, 전기학회논문지 제59권 제2호, 2010.2, 232-241.

[2] HP Compilers for HP Integrity Servers, 2008.

[3] R. Stallman, R.Pesch, S.Shebs, et al., Debugging with GDB, HP 18th Edition, Sep 2008.