Linux 상에서 NUMA 지원을 응용한 스크래치 패드 메모리 관리방법

박병훈*, 서대화*
*경북대학교 전자전기컴퓨터학부
e-mail : dna01@ee.knu.ac.kr

Scratchpad-Memory Management Using NUMA Infrastructure on Linux

Byung-Hun Park*, Dae-Wha Seo*
*School of Electriccal Eng. & Computer Science, Kyungpook National University

요 약

현재 많은 임베디드 SoC(System-On-Chip)에는 캐시 메모리의 단점을 보완하기 위해 온-칩(On-Chip) SRAM, 즉, SPM(Scratchpad Memory)를 내장하고 있으며 SPM 은 그 특성상 캐시 메모리와 달리 소프트웨어가 직접 관리해야 한다. 본 논문에서는 NUMA 를 지원하는 Linux 상에서 이식성이 높으면서 단순하게 구현할 수 있는 SPM 관리 방법을 제안한다.

1. 서론

범용 마이크로프로세서와는 달리 임베디드 SoC 는 칩(Chip)의 크기, 소모 전력, 지연시간 예측가능성을 이유로 캐시 메모리 사용에 제약이 있다. 이를 보완하기 위해 도입된 것이 SoC 에 온-칩(On-Chip)된 내부 메모리, 즉, SPM(Scratchpad Memory)이다.

SPM 은 캐시 메모리와는 달리 복잡한 하드웨어 컨트롤러를 가지고 있지 않기 때문에 소프트웨어에서 직접 관리해야 한다. 본 논문에서는 Linux 상에서 기존의 NUMA 지원을 응용한 SPM 관리방법을 소개한다.

2. 관련연구

소프트웨어 수준의 SPM 관리에 대한 기존 연구는 크게 컴파일러에 의한 방법과 운영체제에 의한 방법으로 나뉜다.

컴파일러에 의한 방법은 컴파일러가 컴파일 중에 응용프로그램의 소스 코드를 분석하여 SPM 관리 코드를 자동으로 생성한다[1]. 이 방법은 응용프로그램 개발자가 SPM 에 대해 신경을 쓰지 않아도 되고 소스 수준의 이식성을 보장하는 장점이 있다. 하지만 일반적으로 널리 사용되는 대부분의 컴파일러들은 아직까지 SPM 에 대한 지원이 없다.

운영체제에 의한 SPM 관리방법은 크게 간접 및 직접 방법으로 나뉜다. 간접 방법은 운영체제가 제공하는 전용의 API 및 하드웨어를 사용하여 응용프로그램에서 직접 SPM 상의 메모리 할당 및 해제를 관리하는 방법[2]이다. 이 경우 기존 컴파일러의 수정이 필요 없지만 응용프로그램 개발자가 직접 SPM 의 관리를 고려해야 한다. 또한 소스 수준의 이식성을 제공하지 못하며 기존 운영체제의 수정이 필요하다. 직접 방법은 가상 메모리의 요구-페이징(Demand-Paging) 및 Postpass-Optimizer Tool 을 사용하는 방법[3]이다. 이 경우는 기존 컴파일러의 수정 및 응용프로그램 개발자의 SPM 에 대한 고려가 필요치 않고 소스 수준의 이식성을 제공하지만 기존 운영체제의 많은 수정이 필요하고 해당 플랫폼(Platform)용 Postpass-Optimizer Tool 이 필요하다.

3. Linux 의 NUMA 지원 및 SPM 으로의 응용

Linux 는 커널버전 2.6 부터 NUMA 를 지원하고 있으며 Non-NUMA 시스템에서 하나의 물리적으로 연속된 메모리 공간을 위한 균일 메모리 모델(Flat Memory Model)외에 다수의 물리적으로 연속된 메모리 공간을 위한 비연속 메모리 모델(Discontiguous Memory Model)도 지원한다. 바로 이 비연속 메모리 모델의 경우 NUMA 시스템은 아니지만 NUMA 메커니즘을 사용하여 구현이 된다. 즉, 분리된 각각의 연속된 메모리 공간을 마치 NUMA 시스템에서의 각 노드(Node)처럼 간주하는 것이다.

여기서 SPM 을 오프-칩(Off-Chip) 메모리와 분리된 별도의 메모리 공간으로 생각한다면 비연속 메모리 모델을 SPM 에 적용할 수 있음을 알 수 있다.

4. 제안 방법의 구현

본 논문에서 제안하는 Linux 상에서 NUMA 지원을 응용한 SPM 관리는 ARM 아키텍처상에서의 커

널버전 2.6.31 을 기준한다.

4.1. 커널에서의 구현

기존의 비연속 메모리 모델(Discontiguous Memory Model)의 구현을 그대로 사용하기 때문에 커널의 변경은 필요하지 않으며 단지 해당 SoC 의 SPM 에 대한 정보(주소 및 크기)를 제공하기만 하면 된다.

4.2. 응용프로그램에서의 구현

응용프로그램에서는 SPM 을 관리하기 위해 libnuma API 를 사용하며 전체적인 구현 과정은 아래와 같다.

- **단계 1:** SPM 에 저장될 코드 및 데이터를 공유 라이브러리(.so)형태로 구현한다.
- **단계 2:** 응용프로그램에서 공유 라이브러리가 위치한 메모리의 주소 공간을 알아낸다.
- **단계 3:** 응용프로그램에서 2 단계에서 알아낸 주소 공간에 NUMA 메모리 정책을 설정한다.
- **단계 4:** 응용프로그램을 컴파일 할 때 구현된 공유 라이브러리를 사용하여 링크한다.

1 단계의 목적은 첫째, SPM 의 크기는 오프-칩(Off-Chip) 메모리보다 상대적으로 매우 작기 때문에 메모리를 절약하기 위해서이며 둘째, SPM 상에 저장될 코드와 데이터가 위치한 프로세스상의 주소 공간을 쉽게 알아내기 위해서이다.

다음 단계를 설명하기 앞서 그림 1 과 같은 코드를 응용프로그램에 적용한다. numa_available()는 시스템의 NUMA 지원여부를 알려주는 libnuma API 이다.

```
#include <numa.h>
...
int numa_usage; // Global Variable
...
   If(numa_available() < 0)
      numa_usage = 0; // not supported
   else
      numa_usage = 1; // supported
...
```

(그림 1) NUMA 지원여부 확인 코드

이와 같이 하는 이유는 해당 응용프로그램이 실행될 플랫폼(Platform)의 NUMA 지원 여부에 상관없이 응용프로그램의 소스 및 바이너리 수준의 이식성을 제공하기 위해서이다. 따라서 단계 2~3 은 응용프로그램이 실행 중에 시스템이 NUMA 를 지원하는 경우(numa_available == 1)에만 수행되게 한다.

2 단계에서 응용프로그램은 /proc/self/maps 파일을 사용하여 그림 2 와 같은 프로세스의 메모리 맵을 구하고 1 단계에서 구현한 공유 라이브러리가 저장된 주소를 알아 낸다.

```
00a5f000-00b82000 r-xp 00000000 08:06 1023304    /lib/tls/libc-2.3.4.so
00b82000-00b83000 r--p 00123000 08:06 1023304    /lib/tls/libc-2.3.4.so
00b83000-00b86000 rw-p 00124000 08:06 1023304    /lib/tls/libc-2.3.4.so
00b86000-00b88000 rw-p 00b86000 00:00 0
08048000-0804c000 r-xp 00000000 08:06 470527     /bin/cat
0804c000-0804d000 rw-p 00003000 08:06 470527     /bin/cat
091a6000-091c7000 rw-p 091a6000 00:00 0
```

(그림 2) /proc/self/maps 의 예

단계 3 에서 응용프로그램은 numa_tonode_memory() API(libnuma 라이브러리)를 사용하여 2 단계에서 알아낸 주소 공간의 메모리 할당에 사용할 NUMA 노드 ID, 즉, SPM 에 해당하는 NUMA 노드 ID 를 지정한다.

1~4 단계가 모두 구현된 응용프로그램이 실행되면 1 단계에서 구현된 공유 라이브러리의 코드 및 데이터를 액세스할 때 발생하는 요구 페이징(Demand Paging)에서 SPM 상의 페이지 프레임(Page Frame)이 할당된다

5. 결론

본 논문에서 SPM 을 관리하기 위한 새로운 방법을 제시했다. 본 논문에서 제안한 Linux 상에서의 NUMA 지원을 응용한 SPM 관리 방법을 사용하면 크게 네 가지의 장점이 있다. 첫째, 기존 컴파일러를 수정 없이 그대로 사용할 수 있다. 둘째, 운영체제의 수정이 필요 없다. 셋째, 다수의 프로세스에 의한 SPM 공유문제를 응용프로그램에서 고려하지 않아도 된다. 마지막으로 SoC 의 SPM 지원유무와는 상관없이 응용프로그램의 소스 및 바이너리 수준의 이식성을 제공해 준다.

참고문헌

[1] M. I. Aouad and O. Zendra, “A Survey of Scratch-Pad Memory Management Techniques for low-power and –energy,” ICOOOLPS, July 2007.

[2] B. Egger, J. Lee and H. Shin, “Scratchpad Memory Management for Portable Systems with a Memory Management Unit,” EMSOFT, pp.321-330, Oct. 2006.

[3] P. Francesco et al., “An Integrated Hardware/Software For Run-Time Scratchpad Management,” Design Automation Conference Proceedings 41th, pp.238-243, Oct. 2004.

[4] A. Kleen, “An NUMA API for Linux,” SUSE Labs, Aug. 2004.