Homework 6: Parallel Programming

Tae-Hyun Oh
Associate Professor
Dept. Electrical Engineering
POSTECH, Korea

Slides by
Youngjoo Lee

EPIC LAB

Background

Background

OpenMP

- ✓ 공유 메모리 병렬 프로그래밍 API
- ✓ OpenMP의 pragma를 사용하면 컴파일러가 멀티 쓰레드를 사용해서 병렬적으로 실행 될 수 있도록 실행파일을 생성
- ✓ 컴파일 예제
  - `gcc -fopenmp -o hello.out hello.c`

Background

Example: Hello, World!

- ✓ example/hello.c
  - #pragma omp parallel
    - #pragma : 이후 부분들이 지시문(directive)임을 컴파일러에게 전달
    - omp : OpenMP
    - parallel : 이후 코드가 병렬적으로 실행될 것을 의미
  - 예제 프로그램을 반복적으로 실행하면 실행되는 thread 순서가 실행마다 변하는 것을 확인
    - 프로그래머가 실행될 thread의 순서를 지정해줄수 없음

Background

Example: Vector addition

✓ vector_addition.c

- 제공된 vec_simple() 함수는 OpenMP를 통해 병렬적으로 실행되지만, 모든 thread가 전체 vector addition, 0 ~ (ARRAY_SIZE-1)을 수행함
- hello.c에서 확인한 omp_get_num_threads() 함수와 omp_get_thread_num() 함수를 사용하여 이후 슬라이드에서 설명될 2가지 다른 방법으로 병렬화

Background

Vector addition with slicing

✓ Slicing 방법을 통해 vector addition을 병렬화

- 각 thread가 돌아가며 하나씩 sum을 처리
- 예를들어 전체 thread 수가 4라고 가정했을 때,
  - Thread 0은 `i % 4 == 0 (i = 0, 4, 8, … )`,
  - Thread 1은 `i % 4 == 1 (i = 1, 5, 9, … )`,
  - Thread 2은 `i % 4 == 2 (i = 2, 6, 10, … )`,
  - Thread 3은 `i % 4 == 3 (i = 3, 7, 11, … )`에 해당하는 데이터 처리

Background

Vector addition with chunking

✓ Chunking 방법을 통해 vector addition을 병렬화

- 총 thread 개수만큼의 덩어리로 나눠 각 thread에서 하나의 덩어리를 담당
- 예를 들어 전체 thread 수가 4이고 `for(i = 0; i < 8; i++)`을 수행하여야 한다고 가정
  - Thread 0은 0,1
  - Thread 1은 2,3
  - Thread 2는 4,5
  - Thread 3은 6,7에 해당하는 데이터를 처리
- 각 thread에 균일하게 나눠지지 않는 경우, 각 thread간 처리하는 갯수에 약간의 차이를 줘도 되고, 병렬 프로그래밍 바깥에서 남은 연산을 처리해도 무방

Background

Dot product

✓ dot_product.c

- Vector addition과 비슷하지만, 병렬적으로 계산된 product들을 하나의 변수(global_sum)에 더하는 과정에서 data race 문제가 발생 가능

✓ 해결 방법

- 복수의 thread가 동시에 실행할 수 없고 하나의 thread씩 순차적으로 실행하는 critical section을 설정하여 해결
- thread의 수가 많아질수록 성능 저하

Problem Definition

Problem Definition

Problem 1

✓ Vector addition

- Vector addition을 slicing과 chunking 방법으로 각각 병렬화 및 vec_simple()와의 연산처리 속도 비교
  - 각각 vec_slicing(), vec_chunking() 함수로 구현
  - 자동으로 for문을 병렬 수행해주는 #pragma omp for 문 사용 금지
  - ARRAY_SIZE = 1000000에 대해서 수행
- 에러 계산 코드 수정 X

- vec_slicing(), vec_chunking() 함수만 작성.
  - **이외의 코드 수정 금지.**
- "-fopenmp" 옵션 포함하여 컴파일

Problem Definition

Problem 2

✓ Dot product

- Vector 내적에 대해서 OpenMP를 이용하여 성능 최적화 및 비교
  - ARRAY_SIZE = 1000000에 대해서 수행
- dot_product.c의 dotp_omp 함수 작성
  - #pragma omp for 사용 가능
  - #pragma omp critical을 사용하여 data race 문제 해결
- 에러 계산 코드 수정 X

- dotp_omp() 함수만 작성.
  - **이외의 코드 수정 금지.**
- "–fopenmp" 옵션 포함하여 컴파일

Problem Definition

Problem 3

✓ 8x8 Matrix multiplication w/ NEON

- main.c
  - 8x8 matrix multiplication을 NEON을 활용하여 성능 최적화 및 비교
  - “Matrix multiplication with NEON” 주석이 시작되는 부분 부터 NEON을 이용하여 matrix multiplication 코드를 작성
    - NEON intrinsic 명령어를 이용하여 코드 작성
  - “put your code here”가 있는 부분에만 코드 작성
    - **이외의 코드 수정 금지.**
  - Makefile이 있는 폴더에서 “`make`” (또는 “`sudo make`”)를 입력 후 엔터, “`exec`” 파일이 생성되는 것을 확인
  - `./exec` 를 입력하여 코드의 동작을 확인
    - “PASS”가 정상적으로 터미널에 뜨는지 확인

Problem Definition

Problem 3

✓ 8x8 Matrix multiplication w/ NEON

- arr1, arr2, ans_neon, ans_for의 자료형이 모두 **int16_t**임을 주의 및 수정 금지
  - 단 “put your code here” 부분에서 자유롭게 변수 선언 및 사용 가능
- 에러 계산 코드 또한 수정 금지
  - 임의로 에러를 계산하는 경우 적절성을 판단 후 평가 예정

- 예제코드 참조하여 구현
  - example/Neon_example: vector 내적을 구현한 코드
  - 실행방법은 앞의 설명과 동일
- NEON intrinsic 명령어는 아래 주소를 참조
  - https://developer.arm.com/documentation/den0018/a/NEON-Intrinsics-Reference?lang=en

Submission & Evaluation

Submission & Evaluation

✓ 다음 내용을 포함하여 결과보고서 작성

- 1. 본인이 구현한 부분에 대해서 간단하게 설명
  - Source code에 대해서 line-by-line으로 설명할 필요 X
- 2. Discussion
  - Vector addition에 대한 결과 비교 및 분석.
  - Dot product에 대한 결과 비교 및 분석.
  - Matrix multiplication w/ NEON 에 대한 결과 비교 및 분석.

✓ 담당조교 - 권순현 (soonhyun.kwon@postech.ac.kr)

Submission & Evaluation

Submission

- ✓ Due date: 5.30 (목) 23:59
- ✓ Submission: **학번.zip** 파일의 형식으로 PLMS에 제출
  - 파일 이름 수정 X, 내부 코드만 수정하여 제출
  - 제출 예시

    20242228.zip

    ㄴ vector_addition.c

    ㄴ dot_product.c

    ㄴ main.c

    ㄴ arm_perf.h

    ㄴ Makefile

    ㄴ 20242228_report.pdf

Submission & Evaluation

Evaluation

- ✓ Implementation (90)
  - Vector addition using OpenMP (20)
  - Dot product using OpenMP (20)
  - Matrix multiplication using Neon (50)
- ✓ Report (10)
  - Sufficient and appropriate explanation & discussion (10)

0 credits for {cheating, late submission}