SIGPL 여름학교
2019. 8. 28.

동시성 프로그램에서 메모리를 수집하는 세가지 방법

강지훈, 정재황 (KAIST 전산학부)
https://cp.kaist.ac.kr


Google DeepMind
Challenge Match
8 - 15 March 2016
AlphaGo
ALPHAGO
02:00:00
LEE SEDOL
01:59:42
AlphaGo
Lee Sedol

병렬성의 시대 (parallelism)

- 모든 컴퓨터는 병렬적: CPU, 메모리, I/O 등 모든 자원이 병렬
  - cf. “latency hiding”: CPU와 I/O간 병렬성 이용

- 문제 (2005): 데나르 스케일링 종료 (같은 면적-코어당 트랜지스터 수 못늘림)
- 돌파구: **멀티코어 병렬 시스템**

- 문제 (2018?): 무어의 법칙 종료 (단위 면적당 트랜지스터 수 못늘림)
- 돌파구: **특수 병렬 시스템 (하드웨어 가속기, AI/IoT를 위한)**

동시성의 시대 (concurrency)

- 병렬성: 여러 자원
- 동시성: **공유된 수정가능한 자원**
  - 예: CPU, GPU, 메모리, 데이타베이스, 데이타센터, ...

- 병렬성 : 동시성 = 찐빵 : 앙꼬
  구슬이 서말이라도 꿰어야 보배

- 예: 웹서버 4개가 cache 서버 1개, DB서버 1개 공유
  - 질문 1: 어떤 자원?
  - 질문 2: 어떤 공유된 수정가능한 자원?

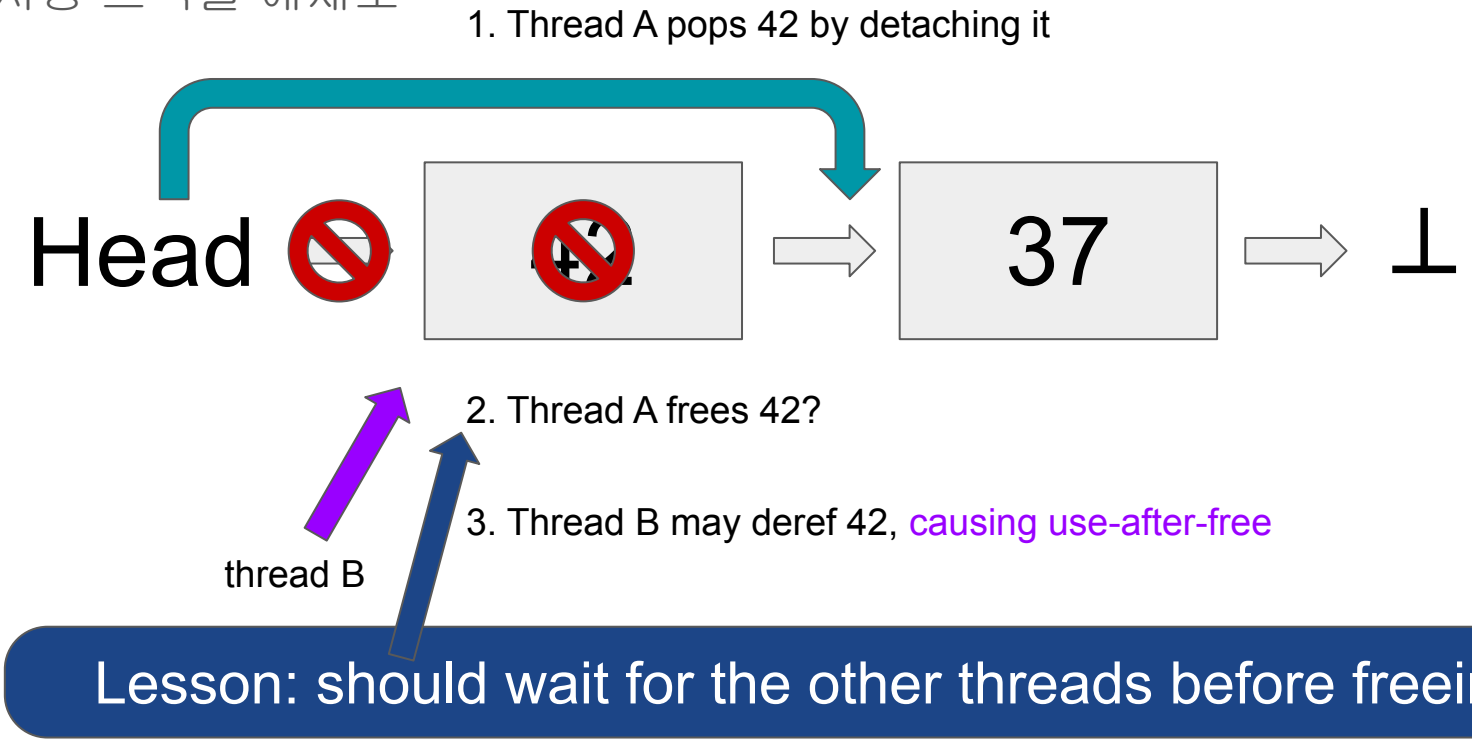
동시성 자료구조: 동시성 자원을 추상화

- 난점: **동시성 자원 사용하기 어려움**
- 이유: 여러 접근 사이에 조합폭발적인 비결정성
- 예: 동시성 메모리 (다른 쓰레드가 동시에 메모리에 읽고 쓰고…)

- **동시성 자료구조**: 비본질적인 비결정성을 숨김
- 장점: 동시성에 대해 잘 몰라도 쉽게 사용 가능
- 예: 동시성 queue, B-tree, hash table, LSM tree, …

동시성 자료구조의 메모리 수집 문제

동시성 스택을 예제로



메모리 수집 문제의 기존 해결책

- 포인터별로 free 미루기 (pointer-based reclamaation, PBR)
  - Thread B가 42를 free하지 말아달라고 부탁
- 시대별로 free 미루기 (epoch-based reclamation, EBR)
  - Thread B의 연산에서 접근한 블락을 모두 free하지 말아달라고 부탁
- QSBR, QSense, HPMB, Free Access, Snowflake, …
  - 모두 free하지 말아달라고 부탁
  - 부탁받지 않은 블락만 free
- **상충하는 성질때문에 압도적인 해결책 없음**

우리가 한일: 메모리 수집 문제의 기존 한계 돌파

- 메모리 수집 문제 해결책 평가기준 정립
  - 안멈춤 (nonblocking)
  - 견고함 (robust)
  - 빠름 (fast)
  - 가벼움 (compact)
  - 고이식성 (portable)
  - 고적용성 (versatile)

- 모든 기준을 충족하는 PEBR 개발 (PBR과 EBR 섞어서)
- 제출됨, 오픈 소스 (https://cp.kaist.ac.kr/gc)

실험 결과: 빠르고 견고함

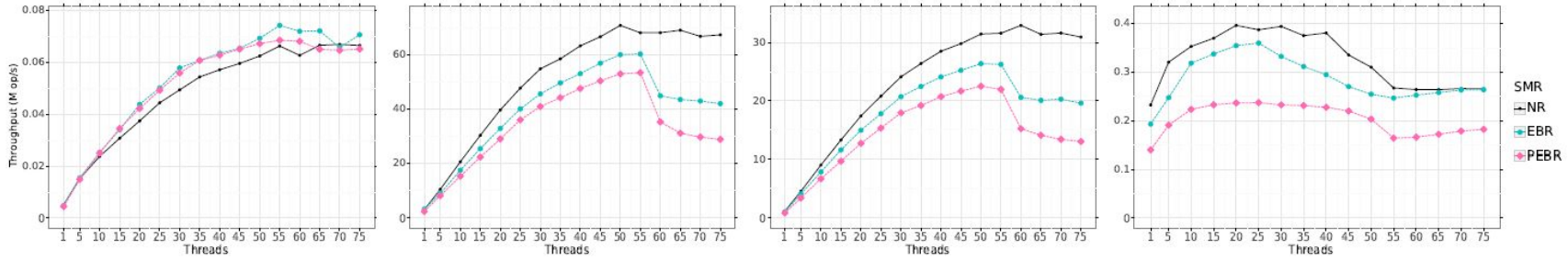


Figure 8. Throughput (million operations per second) for varying number of threads

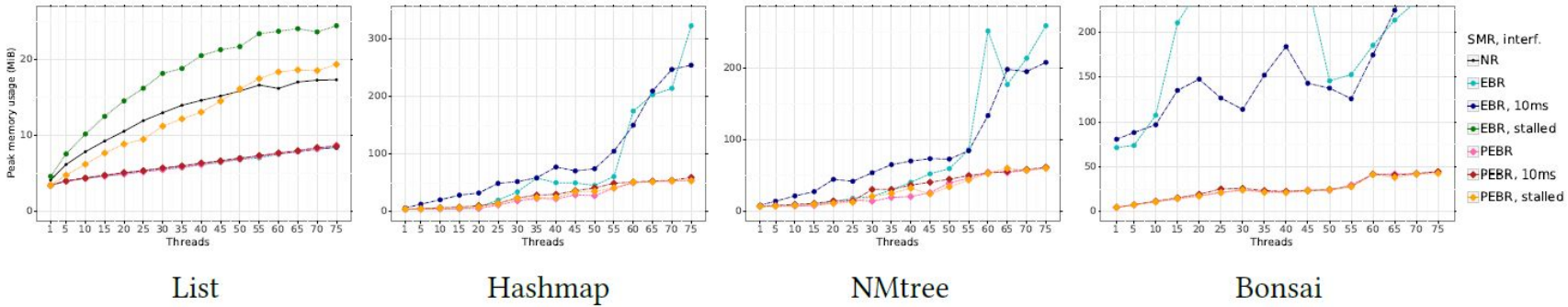


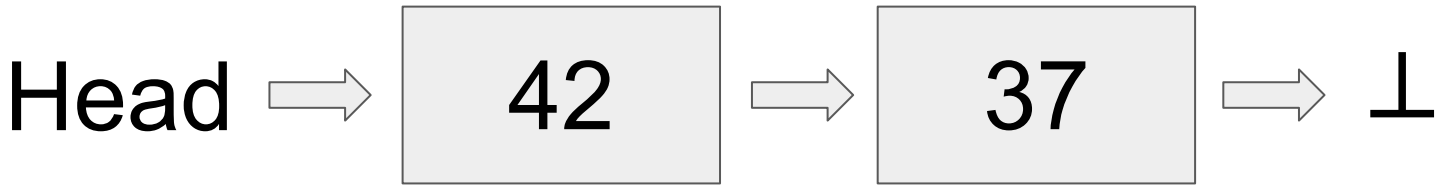
Figure 9. Peak memory usage for varying number of threads (Note: we will buy extra pages and enlarge figures, if allowed)

끝!

질문?

덤: 동시성 메모리를 수집하는 세가지 방법;;

- PBR: 포인터별로 free 미루기
- EBR: 시대별로 free 미루기
- PEBR: 포인터/시대별로 free 미루기



포인터별로 free 미루기 (Pointer-based reclamation)

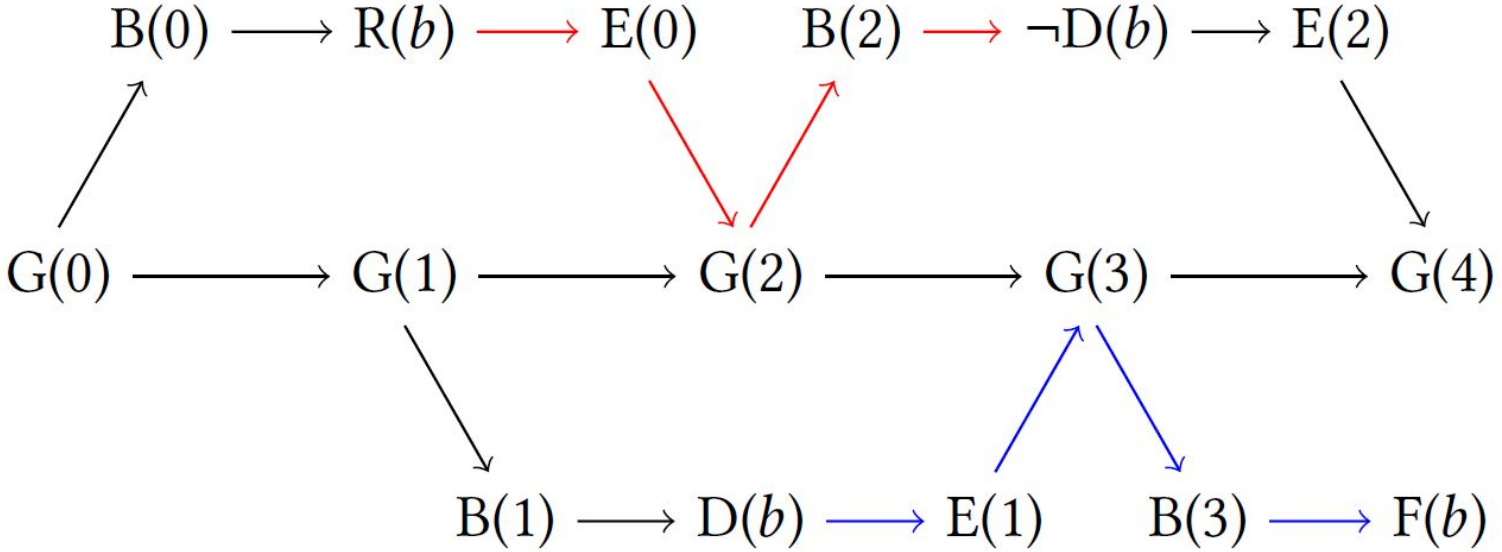
- 아이디어 1: 블락에 접근하기 전에 블락을 free하지 말라고 널리 알림
- // `ptr` points to the block to access.
  protect(ptr); // (simplified)
  next = ptr.next; // dereferencing `ptr`

- 아이디어 2: free하는 대신 freelist에 넣고 나중에 free
- // `node` is detached and to be freed.
  reclaim(node); // freeing `node` when not protected

- **문제: protect할 때마다 비싼 읽기-쓰기 동기화**

시대별로 free 미루기 (Epoch-based reclamation)

- 해결: 여러 접근을 하나의 단위(critical section)로 묶어서 동기화 비용 절감
- 단위마다 시대(epoch) 부여, 한 시대에서 접근한 모든 블락을 free하지 말라고 알림



시대별로 free 미루기 (Epoch-based reclamation)

- 문제: 견고하지 않음
  - 쓰레드가 단위(critical section)를 끝내지 않으면 메모리 수집 못함

- ~~연구 질문 1: EBR처럼 빠르면서 PBR처럼 견고한 방법은 없을까?~~
  - Snowflake, QSense, HPMB, ...

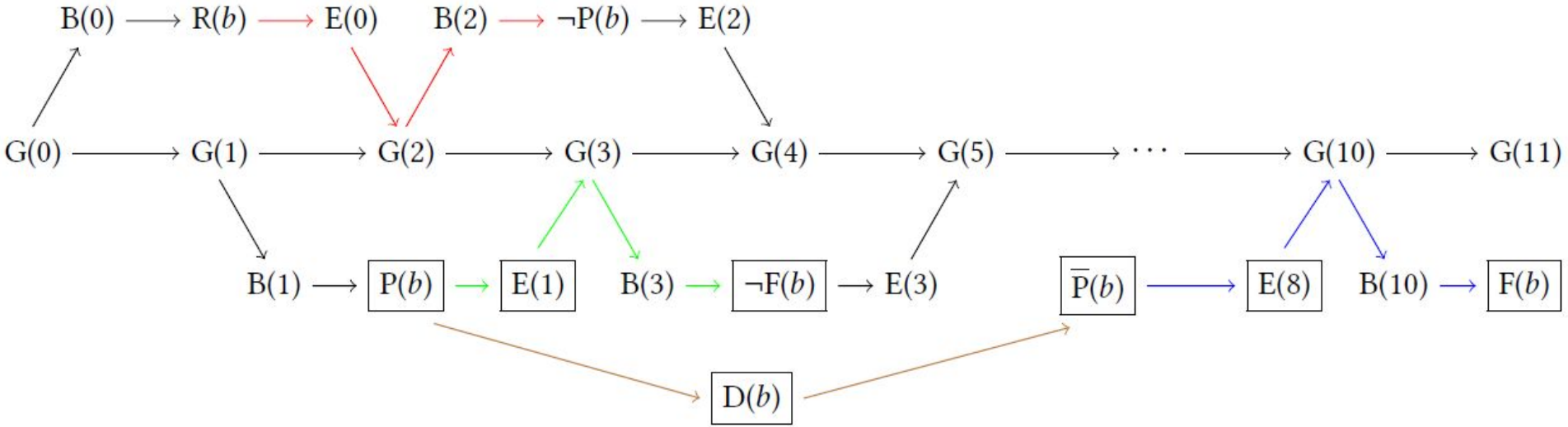
- 연구 질문 2: 안멈추고, 견고하고, 빠르고, 가볍고, 이식성 높고, 적용성 높은 메모리 수집 기법?

- 답: 포인터/시대별로 free 미루기 (PEBR)

포인터/시대별로 free 미루기 (PEBR)

- EBR처럼 단위별로 동기화 **(빠름)**
- PBR처럼 접근하기 전에 포인터를 찜콩, 단 단위 내에서만
  찜콩한 포인터는 단위 밖으로 나갈 때 개별적으로 지킴

질문: 견고한가?



견고한 수집을 위해 방해하는 쓰레드 쫓아내기

- A가 B를 쫓아낼 때...

- B에게 쫓아낼거라고 알려줌
- B가 찜콩한 포인터를 모두 개별적으로 지킴 (PBR처럼)
- B를 쫓아냄

- **이 과정이 안멈추고, 견고하고, 빠르고, 고이식성임**

진짜 끝!

질문?

덤2: 어쩌다 이런 일을 하게 되었나...

- (2014-현재) 컴파일러 검증에 관심

- (2015) **동시성 프로그래밍에 관심**
- (2015-2016) 동시성 실행의미가 엉망. 정리… (POPL 2017)

- (2016-현재) **메모리 수집 문제에 관심**. 라이브러리 관리… (Crossbeam)
- (2017-2018) 관련하여 뭐라도 논문을 써야하는데…
- (2018-현재) PEBR 알고리즘 찾고, 구현하고, 실험하고, 스토리 제작

- **Lesson: 연구는 자신을 믿고 멋모르고 하자**
- Research is what I'm doing when I don't know what I'm doing---Wernher von Braun

덤3: 맞게 짜긴 했나...?

- 동시성 프로그램은 너무 비결정적이라 맞게 짰는지 확신하기 힘듬
- 버그... 버그... 버그...
  - A, B, C 머신에선 잘 되는데 D 머신에서 버그
  - 100번 돌려서 1번 버그 나옴
  - 버그를 확인할 수 있는 지점과 버그를 일으킨 지점이 다름

- **Lesson: 컴퓨터 앞에서 자신을 믿지 말자...**

system. Practically, to get confidence in our implementation, we performed stress testing using harsh parameters (*e.g.*, ejecting the other threads for every 8 critical sections) and LLVM AddressSanitizer [45].

진짜 진짜 끝!

질문?