CS 537 Discussion

5 April 2023

Agenda

- Project 6 overview
- Pthreads Programming
- Example: Parallelizing an algorithm
- Parallel sorting

Project 6 Overview

Pthreads Programming

PThreads: The POSIX threading interface

- Portable Operating System Interface for UNIX
- A standard Interface to OS utilities

Pthreads library contains functions for:

- Creating parallelism
- Synchronizing threads
  - Coordinating their access to shared state

To compile: gcc myprog.c -lpthread

pthread_create()

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);
```

- Creates a new thread.
- Thread id is stored in variable pointed-to by thread parameter.
- The attr parameter specifies attributes (NULL for default attributes.)
- The created thread executes the start routine function, which is passed arg as its parameter.
- Returns 0 if successful.

pthread_join()

```
#include <pthread.h>

int pthread_join(pthread_t thread, void **retval);
```

- Waits for specified thread to finish.
- Only attached threads can be waited for.
- Value returned by exited thread is stored in the variable pointed-to by retval.

pthread_detach

```
#include <pthread.h>

int pthread_detach(pthread_t thread);
```

- Changes the specified thread to be detached, so that its resources can be freed without another thread explicitly calling pthread_join.

Demo

- Basic thread create and arg parsing: https://github.com/shivaram/cs537-sp23-discussion/blob/main/discussions/Week10/pthread_example_1.c

Synchronization with Pthreads

- Pthreads provides several synchronization primitives, including mutexes, condition variables, and semaphores.
- Mutexes can be used to ensure that only one thread accesses a shared resource at a time.
- Condition variables can be used to allow threads to wait for a specific condition to be true before proceeding.

Locks

- int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr);
  - Initializes a new mutex.
- int pthread_mutex_lock(pthread_mutex_t *mutex);
  - Acquires a mutex (blocking if it is not available).
- int pthread_mutex_unlock(pthread_mutex_t *mutex);
  - Releases a mutex that you previously locked.

Demo

- Variable sharing and unexpected results: https://github.com/shivaram/cs537-sp23-discussion/blob/main/discussions/Week10/pthread_mutex_example.c

Condition variables

- int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
- int pthread_cond_signal(pthread_cond_t *cond);
- int pthread_cond_broadcast(pthread_cond_t *cond);

Demo

- Condition variables: https://github.com/shivaram/cs537-sp23-discussion/blob/main/discussions/Week10/cond_var.c

How can you parallelize an algorithm?

Demo algorithm: Matrix multiplication

Step 0: Sequential baseline - https://github.com/shivaram/cs537-sp23-discussion/blob/main/discussions/Week10/serial_1.c

Step 1: Accelerate the program - https://github.com/shivaram/cs537-sp23-discussion/blob/main/discussions/Week10/serial_2.c

Step 2: Add pthreads - https://github.com/shivaram/cs537-sp23-discussion/blob/main/discussions/Week10/parallel_1.c

How can you parallelize an algorithm?

Demo algorithm: Matrix multiplication

Step 3: Distribute workloads - https://github.com/shivaram/cs537-sp23-discussion/blob/main/discussions/Week10/parallel_2.c

Step 4: Synchronize - https://github.com/shivaram/cs537-sp23-discussion/blob/main/discussions/Week10/parallel_3.c

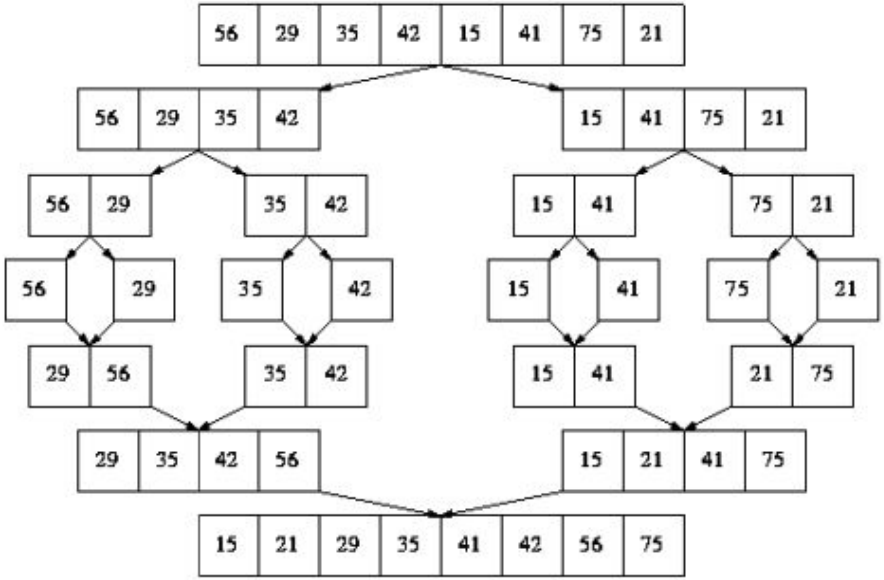
Step 5: Remove contention - https://github.com/shivaram/cs537-sp23-discussion/blob/main/discussions/Week10/parallel_4.c

Parallel Sorting

- Parallel sorting is the process of sorting a large dataset using multiple processors or threads simultaneously.
- Parallel sorting algorithms aim to improve the speed and efficiency of sorting by exploiting the parallel processing capabilities of modern hardware

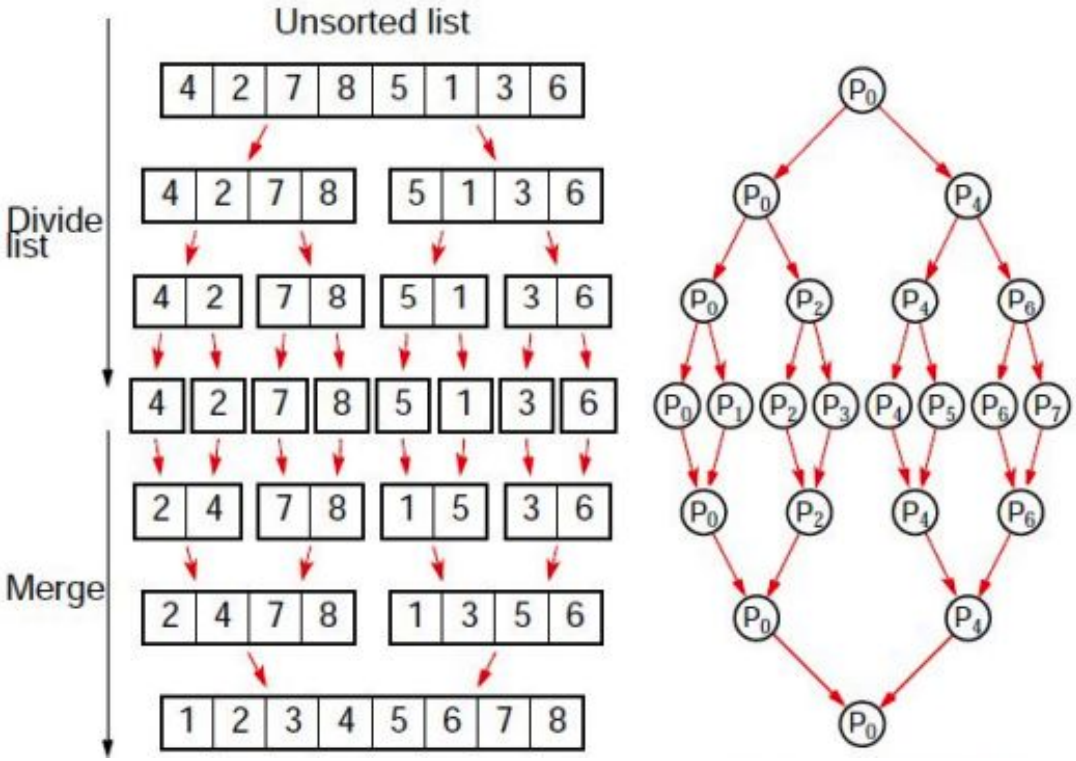
Mergesort

- Uses divide-and-conquer approach,
- The initial unsorted list is first divided in half, each half sublist is then applied the same division method until individual elements are obtained.
- Pairs of adjacent elements/sublists are then merged into sorted sublists until the one fully merged and sorted list is obtained



Parallel Mergesort

Idea: Take advantage of the tree structure of the algorithm to assign work to threads.



Resources

- https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html
- https://pages.cs.wisc.edu/~travitch/pthreads_primer.html

Demo

Parallel sum:

Example: Parallelizing an algorithm

- Write a method named sum that computes the total sum of all elements in an array of integers.
  - How can we parallelize this algorithm if we have 2 CPUs/cores?

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	22	18	12	-4	27	30	36	50	7	68	91	56	2	85	42	98

sum1 = 22+18+12+-4+27+30+36+50 = **191**

sum2 = 7+68+91+56+2+85+42+98 = **449**

sum = sum1 + sum2 = **640**

- Compute sum of each half of array in a thread.
- Add the two sums together.