CS 61C Fall 2021 — Parallelism, WSC, ECC, RAID

Discussion 13: November 29, 2021

1 Thread-Level Parallelism

As powerful as data level parallelization is, it can be quite inflexible, as not all applications have data that can be vectorized. Multithreading, or running a single piece of software on multiple hardware threads, is much more powerful and versatile.

OpenMP provides an easy interface for using multithreading within C programs. Some examples of OpenMP directives:

- The `parallel` directive indicates that each thread should run a copy of the code within the block. If a for loop is put within the block, **every** thread will run every iteration of the for loop.

```
#pragma omp parallel
{
    ...
}
NOTE: The opening curly brace needs to be on a newline or else there
    will be a compile-time error!
```

- The `parallel for` directive will split up iterations of a for loop over various threads. Every thread will run **different** iterations of the for loop. The following two code snippets are equivalent.

```
#pragma omp parallel for
for (int i = 0; i < n; i++) {
    ...
}
```

```
#pragma omp parallel
{
#pragma omp for
    for (int i =0; i < n; i++) { ... }
}
```

There are two functions you can call that may be useful to you:

- `int omp_get_thread_num()` will return the number of the thread executing the code
- `int omp_get_num_threads()` will return the number of total hardware threads executing the code

1.1 For each question below, state and justify whether the program is **sometimes incorrect**, **always incorrect**, **slower than serial**, **faster than serial**, or **none of the above**. Assume the default number of threads is greater than 1. Assume no thread will complete before another thread starts executing. Assume `arr` is an `int[]` of length `n`.

(a)

```
// Set element i of arr to i
#pragma omp parallel
{
    for (int i = 0; i < n; i++)
        arr[i] = i;
}
```

Slower than serial: There is no **for** directive, so every thread executes this loop in its entirety. n threads running n loops at the same time will actually execute in the same time as 1 thread running 1 loop. Despite the possibility of false sharing, the values should all be correct at the end of the loop. Furthermore, the existence of parallel overhead due to the extra number of threads could slow down the execution time.

(b)

```
// Set arr to be an array of Fibonacci numbers.
arr[0] = 0;
arr[1] = 1;
#pragma omp parallel for
for (int i = 2; i < n; i++)
    arr[i] = arr[i-1] + arr[i - 2];
```

Always incorrect (when $n > 4$): Loop has data dependencies, so the calculation of all threads but the first one will depend on data from the previous thread. Because we said "assume no thread will complete before another thread starts executing," this code will always read incorrect values.

(c)

```
// Set all elements in arr to 0;
int i;
#pragma omp parallel for
for (i = 0; i < n; i++)
    arr[i] = 0;
```

Faster than serial: The **for** directive actually automatically makes loop variables (such as the index) private, so this will work properly. The **for** directive splits up the iterations of the loop into continuous chunks for each thread, so there will be no data dependencies or false sharing.

1.2 What potential issue can arise from this code?

```c
// Decrements element i of arr. n is a multiple of omp_get_num_threads()
#pragma omp parallel
{
    int threadCount = omp_get_num_threads();
    int myThread = omp_get_thread_num();
    for (int i = 0; i < n; i++) {
        if (i % threadCount == myThread) arr[i] -= 1;
    }
}
```

False sharing arises because different threads can modify elements located in the same memory block simultaneously. This is a problem because some threads may have incorrect values in their cache block when they modify the value `arr[i]`, invalidating the cache block.

2 Amdahl's Law

In the programs we write, there are sections of code that are naturally able to be sped up. However, there are likely sections that just can't be optimized any further to maintain correctness. In the end, the overall program speedup is the number that matters, and we can determine this using Amdahl's Law:

$$\text{True Speedup} = \frac{1}{S + \frac{1-S}{P}}$$

where S is the non-sped-up part and P is the speedup factor (determined by the number of cores, threads, etc.).

2.1 You are going to run a convolutional network to classify a set of 100,000 images using a computer with 32 threads. You notice that 99% of the execution of your project code can be parallelized on these threads. What is the speedup?

$1/(0.01 + 0.99/32) \approx 1/0.04 = 25$

2.2 You run a profiling program on a different program to find out what percent of this program each function takes. You get the following results:

Function	% Time
f	30%
g	10%
h	60%

(a) We don't know if these functions can actually be parallelized. However, assuming all of them can be, which one would benefit the most from parallelism?

h

(b) Let's assume that we verified that your chosen function can actually be parallelized. What speedup would you get if you parallelized just this function with 8 threads?

$1/(0.4 + 0.6/8) \approx 2.1$

3 Warehouse-Scale Computing

Sources speculate Google has over 1 million servers. Assume each of the 1 million servers draw an average of 200W, the PUE is 1.5, and that Google pays an average of 6 cents per kilowatt-hour for datacenter electricity.

3.1 Estimate Google's annual power bill for its datacenters.

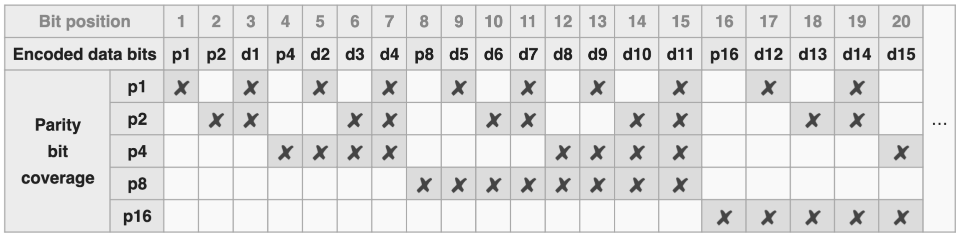
$1.5 \cdot 10^6$ servers $\cdot$ 0.2kW/server $\cdot$ \$0.06/kW-hr $\cdot$ 8760 hrs/yr $\approx$ \$157.68 M/year

3.2 Google reduced the PUE of a 50,000-machine datacenter from 1.5 to 1.25 without decreasing the power supplied to the servers. What's the cost savings per year?

$\text{PUE} = \frac{\text{Total building power}}{\text{IT equipment power}} \implies Savings \propto (PUE_{old} - PUE_{new}) * \text{IT equipment power}$
$(1.5 - 1.25) \cdot 50000$ servers $\cdot$ 0.2kW/server $\cdot$ \$0.06/kW-hr $\cdot$ 8760hrs/yr $\approx$ \$1.314 M/year

4 Hamming ECC

Recall the basic structure of a Hamming code. We start out with some bitstring, and then add parity bits at the indices that are powers of two (1, 2, 8, etc.). We don't assign values to these parity bits yet. **Note that the indexing convention used for Hamming ECC is different from what you are familiar with.** In particular, the 1 index represents the MSB, and we index from left-to-right. The ith parity bit $P\{i\}$ covers the bits in the new bitstring where the *index* of the bit under the aforementioned convention, j, has a 1 at the same position as i when represented as binary. For instance, 4 is `0b100` in binary. The integers j that have a 1 in the same position when represented in binary are 4, 5, 6, 7, 12, 13, etc. Therefore, $P4$ covers the bits at indices 4, 5, 6, 7, 12, 13, etc. A visual representation of this is:



Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits		**p1**	**p2**	**d1**	**p4**	**d2**	**d3**	**d4**	**p8**	**d5**	**d6**	**d7**	**d8**	**d9**	**d10**	**d11**	**p16**	**d12**	**d13**	**d14**	**d15**	
Parity bit coverage	**p1**	✗		✗		✗		✗		✗		✗		✗		✗		✗		✗		
	p2		✗	✗			✗	✗			✗	✗			✗	✗			✗	✗		...
	p4				✗	✗	✗	✗					✗	✗	✗	✗					✗	
	p8								✗	✗	✗	✗	✗	✗	✗	✗						
	p16																✗	✗	✗	✗	✗	

Source: **`https://en.wikipedia.org/wiki/Hamming_code`**

4.1 How many bits do we need to add to 0011_2 to allow single error correction?

m parity bits can cover bits 1 through $2^m - 1$, of which $2^m - m - 1$ are data bits. Thus, to cover 4 data bits, we need 3 parity bits.

4.2 Which locations in 0011_2 would parity bits be included?

Using P to represent parity bits: PP0P011_2

4.3 Which bits does each parity bit cover in 0011_2?

Parity bit 1: 1, 3, 5, 7
Parity bit 2: 2, 3, 6, 7
Parity bit 3: 4, 5, 6, 7

4.4 Write the completed coded representation for 0011_2 to enable single error correction. Assume that we set the parity bits so that the bits they cover have even parity.

$\underline{10}0\underline{0}011_2$

4.5 How can we enable an additional double error detection on top of this?

Add an additional parity bit over the entire sequence.

4.6 Find the original bits given the following SEC Hamming Code: 0110111_2. Again, assume that the parity bits are set so that the bits they cover have even parity.

Parity group 1: error
Parity group 2: okay
Parity group 4: error
To find the incorrect bit's index, we simply sum up the indices of all the erroneous bits.
Incorrect bit: $1 + 4 = 5$, change bit 5 from 1 to 0: 0110011_2
$0110011_2 \to 1011_2$

4.7 Find the original bits given the following SEC Hamming Code: 1001000_2

Parity group 1: error
Parity group 2: okay
Parity group 4: error
Incorrect bit: $1 + 4 = 5$, change bit 5 from 1 to 0: 1001100_2
$1001100_2 \to 0100_2$

5 RAID

5.1 Fill out the following table:

	Configuration	**Pro/Good for**	**Con/Bad for**
RAID 0	Split data across multiple disks	No overhead, fast read / write	Reliability
RAID 1	Mirrored Disks: Extra copy of data	Fast read / write, Fast recovery	High overhead $\to$ expensive
RAID 2	Hamming ECC: Bit-level striping, one disk per parity group	Smaller overhead	Redundant check disks
RAID 3	Byte-level striping with single parity disk.	Smallest overhead to check parity	Need to read all disks, even for small reads, to detect errors

RAID 4	Block-level striping with single parity disk.	Higher throughput for small reads	Still slow small writes (A single check disk is a bottleneck)
RAID 5	Block-level striping, parity distributed across disks.	Higher throughput of small writes	The time to repair a disk is so long that another disk might fail in the meantime.