**Full Name: Harry Q. Bovik**
**Andrew Id:** `bovik`

# 15-418/618
# Exercise 3 SOLUTION

## Problem 1: Understanding OpenMP

### 1A: Performance of OMP Critical

Using the `omp critical` pragma ensured that the sum is computed correctly, but the performance is very poor—ranging from 0.036 GFLOPS for one thread down to 0.006 for 16 threads. To what do you attribute this poor performance? Why does it get worse with more threads?

The overhead of using locks is very high—much higher than performing a single addition, even when there is no contention for the locks. This cost increases with more threads, negating any speedup due to multithreading.

### 1B: Thread-specific accumulation

Both versions the row- and column-wise accumulation functions achieve speedup with more threads, but their sequential performance (around 0.40 GFLOPS) is worse than that of the baseline. To what do you attribute this performance?

Array `accum` is stored in memory, and hence every update requires a memory write. By contrast, the sequential code sums the value in a register.

## 1C: Row- vs. column-wise accumulation

The column-wise accumulation code has slightly better speedup than the row-wise accumulation code. To what do you attribute this difference?

This demonstrates the problem of false sharing. A row of `accum` will have contiguous memory locations, and so all of the accumulators will be packed into one or two cache blocks. This will lead to contention among the threads for this block or blocks.

## 1D: Top performers

The implementation making explicit use of threads and the one making use of OpenMP reduction have nearly identical performance. To what do you attribute this similarity?

The explicit threaded implementation captures how OMP implements the reduction operation.

## 1E: Top performers vs. others

Why do the two top performers outperform the versions using row-wise and column-wise accumulators?

The top performers avoid any contention and do their accumulations in registers, avoiding the pitfalls of the other implementations

## 1F. Bonus Question: Why the upward bends?

The curves indicate that the performance scales *better* for 8–16 threads than it did for 1–8. To what do you attribute this phenomenon?

It's hard to say exactly. Hyperthreading should work well here—it will hide the cost of cache misses and the sequential dependency of the sum operations. That would explain a linear performance increase. Most likely, the superlinear speedup is due to clock frequency scaling. Perhaps the processor recognizes that it is being fully utilized and therefore increases the clock speed.

## Problem 2: Heterogeneous Systems

### 2A: Homogeneous Model Speedup

Explain how the equation for the homogeneous model speedup follows the form of the general Amdahl's Law equation. What resources are used and what is the duration of $T_{\text{seq}}$? What resources are used and what is the duration of $T_{\text{par}}$?

The sequential part of the computation involves computing fraction $1 - f$ of the total using a processor with performance $\mathit{perf}(r)$. The parallel part of the computation involves computing fraction $f$ of the total using $\lfloor n/r \rfloor$ processors, each with performance $\mathit{perf}(r)$.

### 2B: Heterogeneous Model Speedup

Explain how the equation for the heterogeneous model speedup follows the form of the general Amdahl's Law equation. What resources are used and what is the duration of $T_{\text{seq}}$? What resources are used and what is the duration of $T_{\text{par}}$?

The sequential part of the computation involves computing fraction $1 - f$ of the total using a processor with performance $\mathit{perf}(r)$. The parallel part of the computation involves computing fraction $f$, but using both the large processor and the smaller ones. To maximize performance, we split the load, mapping part onto the larger processor, and the remainder on the others, fully using the combined computing power $\mathit{perf}(r) + \lfloor n - r \rfloor$.

## 2C: Optimizing Parameter $r$ for the Homogeneous Model

Fill in the table below giving the values of $r^*$ and $n/r^*$ for the Homogeneous model, and the resulting speedups.

| $\alpha$ | $f$ | $r^*$ | $n/r^*$ | $S_{\text{ho}}$ |
|---|---|---|---|---|
| 0.4 | 0.80 | 42.667 | 6 | 13.46 |
| 0.4 | 0.95 | 8.828 | 29 | 28.88 |
| 0.4 | 0.99 | 1.718 | 149 | 74.60 |
| 0.8 | 0.80 | 256.000 | 1 | 84.45 |
| 0.8 | 0.95 | 51.200 | 5 | 97.10 |
| 0.8 | 0.99 | 10.240 | 25 | 129.65 |

## 2D: Understanding 2C

Explain the trends you see in your answer to 2C: Why do changes to $\alpha$ and/or $f$ cause the changes to $r^*$ and $S_{\text{ho}}$ that you have computed?

For $\alpha = 0.4$, the processing power does not scale very well. Even using $r = 256$ would only yield a processor with performance 9.2. For low values of $f$, we want to have a large processor for the sequential part, but there is limited value in increasing its size too much. For very high values of $f$, we can shrink the size of the processor for the sequential portion and make better use of the other processors.

For $\alpha = 0.8$, the processor power scales much better, giving performance of 84.4 for $r = 256$. This pushes us to use a smaller number of bigger processors. For low values of $f$, we do best with a single processor. Even with high values of $f$, we can benefit from the power of the larger processors. Even with high values of $\alpha$ and $f$, however, we only get around 1/2 of the benefit of increasing the resources from 1 to 256.

## 2E: Optimizing Parameter $r$ for the Heterogeneous Model

Fill in the table below giving the values of $r^*$ for the Heterogeneous model, and the resulting speedups.

| $\alpha$ | $f$ | $r^*$ | $S_{\text{he}}$ |
|---|---|---|---|
| 0.4 | 0.80 | 156 | 29.44 |
| 0.4 | 0.95 | 96 | 72.62 |
| 0.4 | 0.99 | 44 | 147.65 |
| 0.8 | 0.80 | 145 | 116.38 |
| 0.8 | 0.95 | 81 | 165.56 |
| 0.8 | 0.99 | 39 | 211.28 |

## 2F: Understanding 2E (and 2C)

Explain the trends you see in your answer to 2E: why do changes to $\alpha$ and/or $f$ cause the changes to $r^*$ and $S_{\text{he}}$ that you have computed? How do these trends compare to those computed for the Homogeneous model?

As before, for $\alpha = 0.4$, the processing power does not scale very well. Nonetheless, for low values of $f$, we want to dedicate 61% of the total resources to the large processor. Even for $f = 0.99$, the optimum dedicates 17% of the resources to the large processor.

For $\alpha = 0.8$, the improved processor power enables us to get the necessary sequential performance using a smaller fraction of the total resources. Furthermore, this processor can provide more power to the parallel portion of the code, leading to substantially better speedup.

Comparing the Homogeneouse to the Heterogeneous models, we can see a clear advantage for heterogeneity. The overall speedups are $1.6\times$–$2.2\times$ better. We can use a much bigger large processor, since only one is needed. For high values of $\alpha$ and $f$, we see the performance reaching over 80% of the ideal limit of 256.