Experimental Data and Explanations

Question 1

Compilation Command

```
clang -O3 -mavx512f -Rpass-missed=loop-vectorize -Rpass=loop-vectorize vec1_main.c vec1a.c vec1b.c vec1c.c -o vec1_main
```

Clang Vectorization Output

```
[u1591673@granite2 PA3]$ clang -O3 -mavx512f -Rpass-missed=loop-vectorize -Rpass=loop-vectorize vec1_main.c vec1a.c vec1b.c vec1c.c
vec1_main.c:33:3: remark: vectorized loop (vectorization width: 8, interleaved count: 4) [-Rpass=loop-vectorize]
   33 |   for(i=0;i<N;i++)
      |   ^
vec1a.c:7:4: remark: vectorized loop (vectorization width: 16, interleaved count: 4) [-Rpass=loop-vectorize]
    7 |    for(i=1;i<size-1;i++) w[i] = w[i]+1;
      |    ^
vec1b.c:7:4: remark: vectorized loop (vectorization width: 16, interleaved count: 4) [-Rpass=loop-vectorize]
    7 |    for(i=1;i<size-1;i++) w[i] = w[i+1]+1;
      |    ^
vec1c.c:8:4: remark: loop not vectorized [-Rpass-missed=loop-vectorize]
    8 |    for(i=1;i<size-1;i++) w[i] = w[i-1]+1;
      |    ^
```

Loop Statements Under Test

- vec1a: `w[i] = w[i] + 1;`
- vec1b: `w[i] = w[i+1] + 1;`
- vec1c: `w[i] = w[i-1] + 1;`

Expected Vectorization Behavior

vec1a: `w[i] = w[i] + 1;`

Expected: Vectorizable

Reason:

- No loop-carried data dependencies
- All array accesses are at stride 0 (w.r.t. innermost loop)

vec1b: `w[i] = w[i+1] + 1;`

Expected: Vectorizable

Reason:

- Reads ahead from w[i+1], which is not modified in the loop
- No overlap between read and write
- All array accesses are at stride 1

vec1c: `w[i] = w[i-1] + 1;`

Expected: Not vectorizable

Reason:

- Loop-carried RAW dependency
- w[i] depends on the result of w[i-1], which was written in the previous iteration

Comparison with Expectations

Function	Statement	Expected	Actual	Matches?
`vec1a`	`w[i] = w[i] + 1;`	Yes	Yes	Yes
`vec1b`	`w[i] = w[i+1] + 1;`	Yes	Yes	Yes
`vec1c`	`w[i] = w[i-1] + 1;`	No	No	Yes

Function	Statement	Expected	Actual	Matches?

Performance Results (Granite Node: grn054)

After compiling and executing `vec1_main`, the following performance (in GFLOPS) was observed:

```
Matrix Size = 16384; NTrials=5
w[i] = w[i]+1:     Min: 3.21; Max: 3.26
w[i] = w[i+1]+1:   Min: 3.21; Max: 3.23
w[i] = w[i-1]+1:   Min: 0.39; Max: 0.40
```

Interpretation

- **`vec1a`** and **`vec1b`** show high GFLOPS values, confirming that vectorization significantly boosts performance.
- **`vec1c`** shows much lower GFLOPS, indicating that the lack of vectorization due to loop-carried RAW dependency results in slower execution.

Summary

Statement	Vectorized	GFLOPS (Min–Max)	Performance Impact
`w[i] = w[i] + 1;`	Yes	3.21-3.26	Fast
`w[i] = w[i+1] + 1;`	Yes	3.21-3.23	Fast
`w[i] = w[i-1] + 1;`	No	0.39-0.40	Slow

Question 2

Optimization Strategy: Loop Permutation

- Original loop had poor memory access (stride-N), blocking vectorization.
- Loop permutation enabled stride-1 access and SIMD execution.

Performance Results (Granite Node: grn054)

```
Matrix Size = 16384; NTrials=5
Reference: Min: 0.11; Max: 0.14
Optimized: Min: 4.94; Max: 4.97
```

Summary

Version	Vectorized	GFLOPS (Min–Max)	Speedup
Reference	No	0.11–0.14	-
Optimized	Yes	4.94–4.97	~35-45x

Question 3

Optimization Strategy: Loop Fission

- Original loop mixed updates to w and y, blocking vectorization.
- Loop fission separated operations, enabling SIMD.

Performance Results (Granite Node: grn054)

```
Matrix Size = 16384; NTrials=5
Reference: Min: 0.57; Max: 0.59
Optimized: Min: 2.40; Max: 2.43
```

Summary

Version	Vectorized	GFLOPS (Min–Max)	Speedup
Reference	No	0.57–0.59	-
Optimized	Yes	2.40–2.43	~4x

Question 4

Optimization Challenge

- Original loop has a cyclic loop-carried dependency:

  ```
  w[i+1] = y[i] + 1;
  y[i+1] = x[i] + w[i];
  ```

- This cycle prevents safe vectorization.
- Different optimization versions were tested but they could not preserve correctness because of the cyclic dependency.

Performance Results (Granite Node: grn054)

```
Matrix Size = 16384; NTrials=5
Reference: Min: 0.58; Max: 0.59
Optimized: Min: 0.58; Max: 0.60
```

Summary

Version	Vectorized	GFLOPS (Min–Max)	Speedup
Reference	No	0.58–0.59	-
Optimized	No	0.58–0.60	- (Same code)

Question 5

Optimization Strategy: Loop Permutation

- Reference version used naive triple loop.
- Optimized version used loop blocking and reordering for cache and SIMD efficiency.

Performance Results (Granite Node: grn054)

```
Matrix Size = 256; NTrials=5
Reference: Min: 0.50; Max: 0.64
Optimized: Min: 19.59; Max: 20.02
```

Summary

Version	Vectorized	GFLOPS (Min–Max)	Speedup
Reference	No	0.50–0.64	-
Optimized	Yes	19.59–20.02	~31-39x