7. Given the following parallel code in OpenMP, prepared to execute on a given number of threads (`nThreads`), that computes the histogram of vector `index` of `n` elements:

```
#define nThreads 8 // Number of threads
#define n 100000   // size of index vector, perfectly divides nThreads
#define m 5        // Number of bins in histogram

int index[n];                       // input vector
int hist[m];                        // histogram
int hist_containers[m][nThreads]; // per-thread histogram

int main() {
    int iThread, BS;
    InitIndex(index, n); // initialization of vector index
    initHistograms(hist, hist_containers, m, nThreads);

    #pragma omp parallel private(iThread, BS) num_threads(nThreads)
    {
        iThread = omp_get_thread_num();
        BS = n / nThreads;
        for (int i = iThread * BS; i < (iThread + 1) * BS; i++)
             hist_containers[index[i]%m][iThread]++;
    }
    for (iThread = 0; iThread < nThreads; iThread++)
        for (int i = 0; i < m; i++)
            hist[i] += hist_containers[i][iThread];
}
```

(a) Identify the main performance bottleneck that occurs during the execution of the parallel region

7. Given the following parallel code in OpenMP, prepared to execute on a given number of threads (`nThreads`), that computes the histogram of vector `index` of `n` elements:

```
#define nThreads 8 // Number of threads
#define n 100000   // size of index vector, perfectly divides nThreads
#define m 5        // Number of bins in histogram

int index[n];                       // input vector
int hist[m];                        // histogram
int hist_containers[m][nThreads]; // per-thread histogram

int main() {
    int iThread, BS;
    InitIndex(index, n); // initialization of vector index
    initHistograms(hist, hist_containers, m, nThreads);

    #pragma omp parallel private(iThread, BS) num_threads(nThreads)
    {
        iThread = omp_get_thread_num();
        BS = n / nThreads;
        for (int i = iThread * BS; i < (iThread + 1) * BS; i++)
            hist_containers[index[i]%m][iThread]++;
    }
    for (iThread = 0; iThread < nThreads; iThread++)
        for (int i = 0; i < m; i++)
            hist[i] += hist_containers[i][iThread];
}
```

(a) Identify the main performance bottleneck that occurs during the execution of the parallel region

Let's assume:

cache line size = 64 bytes
sizeof (int) = 4 bytes

7. Given the following parallel code in OpenMP, prepared to execute on a given number of threads (`nThreads`), that computes the histogram of vector `index` of `n` elements:

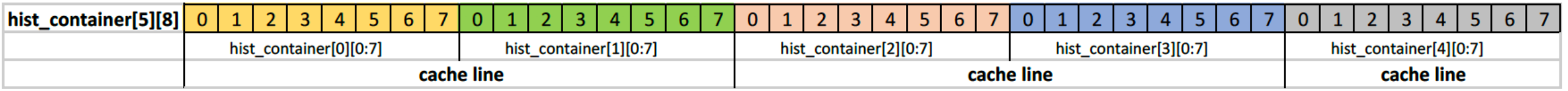
```
#define nThreads 8 // Number of threads
#define n 100000   // size of index vector, perfectly divides nThreads
#define m 5        // Number of bins in histogram

int index[n];                       // input vector
int hist[m];                        // histogram
int hist_containers[m][nThreads]; // per-thread histogram

int main() {
    int iThread, BS;
    InitIndex(index, n); // initialization of vector index
    initHistograms(hist, hist_containers, m, nThreads);

    #pragma omp parallel private(iThread, BS) num_threads(nThreads)
    {
        iThread = omp_get_thread_num();
        BS = n / nThreads;
        for (int i = iThread * BS; i < (iThread + 1) * BS; i++)
            hist_containers[index[i]%m][iThread]++;
    }
    for (iThread = 0; iThread < nThreads; iThread++)
        for (int i = 0; i < m; i++)
            hist[i] += hist_containers[i][iThread];
}
```

(a) Identify the main performance bottleneck that occurs during the execution of the parallel region

Let's assume:

cache line size = 64 bytes
sizeof (int) = 4 bytes

>> FALSE SHARING:
Different threads continuously writing to different elements that reside in the same cache line



7. Given the following parallel code in OpenMP, prepared to execute on a given number of threads (`nThreads`), that computes the histogram of vector `index` of `n` elements:

```
#define nThreads 8 // Number of threads
#define n 100000   // size of index vector, perfectly divides nThreads
#define m 5        // Number of bins in histogram

int index[n];                       // input vector
int hist[m];                        // histogram
int hist_containers[m][nThreads]; // per-thread histogram

int main() {
    int iThread, BS;
    InitIndex(index, n); // initialization of vector index
    initHistograms(hist, hist_containers, m, nThreads);

    #pragma omp parallel private(iThread, BS) num_threads(nThreads)
    {
        iThread = omp_get_thread_num();
        BS = n / nThreads;
        for (int i = iThread * BS; i < (iThread + 1) * BS; i++)
            hist_containers[index[i]%m][iThread]++;
    }
    for (iThread = 0; iThread < nThreads; iThread++)
        for (int i = 0; i < m; i++)
            hist[i] += hist_containers[i][iThread];
}
```

(a) Identify the main performance bottleneck that occurs during the execution of the parallel region

Let's assume:

cache line size = 64 bytes
sizeof (int) = 4 bytes

>> FALSE SHARING:
Different threads continuously writing to different elements that reside in the same cache line

How many cache lines occupies hist_containers ?

| hist_container[5][8] | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | hist_container[0][0:7] | | | | | | | | hist_container[1][0:7] | | | | | | | | hist_container[2][0:7] | | | | | | | | hist_container[3][0:7] | | | | | | | | hist_container[4][0:7] | | | | | | | |
| | cache line | | | | | | | | | | | | | | | | cache line | | | | | | | | | | | | | | | | cache line | | | | | | | |

7. Given the following parallel code in OpenMP, prepared to execute on a given number of threads (`nThreads`), that computes the histogram of vector `index` of `n` elements:

```
#define nThreads 8 // Number of threads
#define n 100000   // size of index vector, perfectly divides nThreads
#define m 5        // Number of bins in histogram

int index[n];                       // input vector
int hist[m];                        // histogram
int hist_containers[m][nThreads]; // per-thread histogram

int main() {
    int iThread, BS;
    InitIndex(index, n); // initialization of vector index
    initHistograms(hist, hist_containers, m, nThreads);

    #pragma omp parallel private(iThread, BS) num_threads(nThreads)
    {
        iThread = omp_get_thread_num();
        BS = n / nThreads;
        for (int i = iThread * BS; i < (iThread + 1) * BS; i++)
            hist_containers[index[i]%m][iThread]++;
    }
    for (iThread = 0; iThread < nThreads; iThread++)
        for (int i = 0; i < m; i++)
            hist[i] += hist_containers[i][iThread];
}
```

(a) Identify the main performance bottleneck that occurs during the execution of the parallel region

Let's assume:

cache line size = 64 bytes
sizeof (int) = 4 bytes

>> FALSE SHARING:
Different threads continuously writing to different elements that reside in the same cache line

How many cache lines occupies hist_containers ? 2 and a half

| hist_container[5][8] | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | hist_container[0][0:7] | | | | | | | | hist_container[1][0:7] | | | | | | | | hist_container[2][0:7] | | | | | | | | hist_container[3][0:7] | | | | | | | | hist_container[4][0:7] | | | | | | | |
| | cache line | | | | | | | | | | | | | | | | cache line | | | | | | | | | | | | | | | | cache line | | | | | | | |

7. Given the following parallel code in OpenMP, prepared to execute on a given number of threads (`nThreads`), that computes the histogram of vector `index` of `n` elements:

```
#define nThreads 8 // Number of threads
#define n 100000   // size of index vector, perfectly divides nThreads
#define m 5        // Number of bins in histogram

int index[n];                    // input vector
int hist[m];                     // histogram
int hist_containers[nThreads][m];

int main() {
    int iThread, BS;
    InitIndex(index, n); // initialization of vector index
    initHistograms(hist, hist_containers, m, nThreads);

    #pragma omp parallel private(iThread, BS) num_threads(nThreads)
    {
        iThread = omp_get_thread_num();
        BS = n / nThreads;
        for (int i = iThread * BS; i < (iThread + 1) * BS; i++)
            hist_containers[iThreads][index[i]%m]++;
    }
    for (iThread = 0; iThread < nThreads; iThread++)
        for (int i = 0; i < m; i++)
            hist[i] += hist_containers[iThreads][i];
}
```

(b) If you change the definition of the per-thread histogram to **`hist_containers[nThreads][m]`** (and all the accesses to it in the code accordingly), will this solve the performance bottleneck identified?

Let's assume:

cache line size = 64 bytes
sizeof (int) = 4 bytes

7. Given the following parallel code in OpenMP, prepared to execute on a given number of threads (`nThreads`), that computes the histogram of vector `index` of `n` elements:

```
#define nThreads 8 // Number of threads
#define n 100000   // size of index vector, perfectly divides nThreads
#define m 5        // Number of bins in histogram

int index[n];                       // input vector
int hist[m];                        // histogram
int hist_containers[nThreads][m];

int main() {
    int iThread, BS;
    InitIndex(index, n); // initialization of vector index
    initHistograms(hist, hist_containers, m, nThreads);

    #pragma omp parallel private(iThread, BS) num_threads(nThreads)
    {
        iThread = omp_get_thread_num();
        BS = n / nThreads;
        for (int i = iThread * BS; i < (iThread + 1) * BS; i++)
            hist_containers[iThreads][index[i]%m]++;
    }
    for (iThread = 0; iThread < nThreads; iThread++)
        for (int i = 0; i < m; i++)
            hist[i] += hist_containers[iThreads][i];
}
```

(b) If you change the definition of the per-thread histogram to **`hist_containers[nThreads][m]`** (and all the accesses to it in the code accordingly), will this solve the performance bottleneck identified?

Let's assume:

cache line size = 64 bytes
sizeof (int) = 4 bytes

7. Given the following parallel code in OpenMP, prepared to execute on a given number of threads (`nThreads`), that computes the histogram of vector `index` of `n` elements:

```
#define nThreads 8 // Number of threads
#define n 100000   // size of index vector, perfectly divides nThreads
#define m 5        // Number of bins in histogram

int index[n];                       // input vector
int hist[m];                        // histogram
int hist_containers[nThreads][m];

int main() {
    int iThread, BS;
    InitIndex(index, n); // initialization of vector index
    initHistograms(hist, hist_containers, m, nThreads);

    #pragma omp parallel private(iThread, BS) num_threads(nThreads)
    {
        iThread = omp_get_thread_num();
        BS = n / nThreads;
        for (int i = iThread * BS; i < (iThread + 1) * BS; i++)
            hist_containers[iThreads][index[i]%m]++;
    }
    for (iThread = 0; iThread < nThreads; iThread++)
        for (int i = 0; i < m; i++)
            hist[i] += hist_containers[iThreads][i];
}
```

(b) If you change the definition of the per-thread histogram to **`hist_containers[nThreads][m]`** (and all the accesses to it in the code accordingly), will this solve the performance bottleneck identified?

Let's assume:

cache line size = 64 bytes
sizeof (int) = 4 bytes

Problem not solved !
There is still more than thread writing the same cache line ...

7. Given the following parallel code in OpenMP, prepared to execute on a given number of threads (`nThreads`), that computes the histogram of vector `index` of `n` elements:

```
#define nThreads 8 // Number of threads
#define n 100000   // size of index vector, perfectly divides nThreads
#define m 5        // Number of bins in histogram

int index[n];                       // input vector
int hist[m];                        // histogram
int hist_containers[nThreads][m];

int main() {
    int iThread, BS;
    InitIndex(index, n); // initialization of vector index
    initHistograms(hist, hist_containers, m, nThreads);

    #pragma omp parallel private(iThread, BS) num_threads(nThreads)
    {
        iThread = omp_get_thread_num();
        BS = n / nThreads;
        for (int i = iThread * BS; i < (iThread + 1) * BS; i++)
            hist_containers[iThreads][index[i]%m]++;
    }
    for (iThread = 0; iThread < nThreads; iThread++)
        for (int i = 0; i < m; i++)
            hist[i] += hist_containers[iThreads][i];
}
```

(c) If the previous change did not solve the performance problem do the necessary changes in the definition of **`hist_containers`** and/or code, without introducing other performance problems.

Let's assume:

cache line size = 64 bytes
sizeof (int) = 4 bytes

7. Given the following parallel code in OpenMP, prepared to execute on a given number of threads (`nThreads`), that computes the histogram of vector `index` of `n` elements:

```
#define nThreads 8 // Number of threads
#define n 100000   // size of index vector, perfectly divides nThreads
#define m 5        // Number of bins in histogram

int index[n];                       // input vector
int hist[m];                        // histogram
int hist_containers[nThreads][m];

int main() {
    int iThread, BS;
    InitIndex(index, n); // initialization of vector index
    initHistograms(hist, hist_containers, m, nThreads);

    #pragma omp parallel private(iThread, BS) num_threads(nThreads)
    {
        iThread = omp_get_thread_num();
        BS = n / nThreads;
        for (int i = iThread * BS; i < (iThread + 1) * BS; i++)
            hist_containers[iThreads][index[i]%m]++;
    }
    for (iThread = 0; iThread < nThreads; iThread++)
        for (int i = 0; i < m; i++)
            hist[i] += hist_containers[iThreads][i];
}
```

(c) If the previous change did not solve the performance problem do the necessary changes in the definition of **`hist_containers`** and/or code, without introducing other performance problems.

Let's assume:

cache line size = 64 bytes
sizeof (int) = 4 bytes

To avoid FALSE SHARING we add PADDING to the current definition so that different threads write on different cache lines …

How many bytes do we have to add for PADDING ??

**`int hist_containers[nThreads][m+ ?? ]`**

7. Given the following parallel code in OpenMP, prepared to execute on a given number of threads (nThreads), that computes the histogram of vector index of n elements:

```
#define nThreads 8 // Number of threads
#define n 100000   // size of index vector, perfectly divides nThreads
#define m 5        // Number of bins in histogram

int index[n];                       // input vector
int hist[m];                        // histogram
int hist_containers[nThreads][m];

int main() {
    int iThread, BS;
    InitIndex(index, n); // initialization of vector index
    initHistograms(hist, hist_containers, m, nThreads);

    #pragma omp parallel private(iThread, BS) num_threads(nThreads)
    {
        iThread = omp_get_thread_num();
        BS = n / nThreads;
        for (int i = iThread * BS; i < (iThread + 1) * BS; i++)
            hist_containers[iThreads][index[i]%m]++;
    }
    for (iThread = 0; iThread < nThreads; iThread++)
        for (int i = 0; i < m; i++)
            hist[i] += hist_containers[iThreads][i];
}
```

(c) If the previous change did not solve the performance problem do the necessary changes in the definition of **hist_containers** and/or code, without introducing other performance problems.

Let's assume:

cache line size = 64 bytes
sizeof (int) = 4 bytes

To avoid FALSE SHARING we add PADDING to the current definition so that different threads write on different cache lines …

How many bytes do we have to add for PADDING ??

```
int hist_containers[nThreads][m+ ?? ]
```

# bytes for PADDING = 64 – m x 4 = 44 bytes
# additional elements in hist_containers per row = 44 bytes / 4 bytes = 11

7. Given the following parallel code in OpenMP, prepared to execute on a given number of threads (nThreads), that computes the histogram of vector index of n elements:

```
#define nThreads 8 // Number of threads
#define n 100000   // size of index vector, perfectly divides nThreads
#define m 5        // Number of bins in histogram

int index[n];                       // input vector
int hist[m];                        // histogram
int hist_containers[nThreads][m];

int main() {
    int iThread, BS;
    InitIndex(index, n); // initialization of vector index
    initHistograms(hist, hist_containers, m, nThreads);

    #pragma omp parallel private(iThread, BS) num_threads(nThreads)
    {
        iThread = omp_get_thread_num();
        BS = n / nThreads;
        for (int i = iThread * BS; i < (iThread + 1) * BS; i++)
            hist_containers[iThreads][index[i]%m]++;
    }
    for (iThread = 0; iThread < nThreads; iThread++)
        for (int i = 0; i < m; i++)
            hist[i] += hist_containers[iThreads][i];
}
```

(c) If the previous change did not solve the performance problem do the necessary changes in the definition of **hist_containers** and/or code, without introducing other performance problems.

Let's assume:

cache line size = 64 bytes
sizeof (int) = 4 bytes

To avoid FALSE SHARING we add PADDING to the current definition so that different threads write on different cache lines …

How many bytes do we have to add for PADDING ??

```
int hist_containers[nThreads][m+ 11 ]
```

# bytes for PADDING = 64 – m x 4 = 44 bytes
# additional elements in hist_containers per row = 44 bytes / 4 bytes = 11