1 Distributing an image

1.1 (a)

$$r = M\%P$$
$$q = \lfloor M/P \rfloor$$
$$MP(p) = \begin{cases} q+1 & 0 \le p < r \\ q & r \le p < P \end{cases}$$

1.2 (b)

$$M = 13, P = 5$$
$$r = 3$$
$$q = 2$$
$$MP(p) = \begin{cases} 3 & 0 \le p < 3 \\ 2 & 3 \le p < 5 \end{cases}$$

p	0	1	2	3	4
M_p	3	3	3	2	2

1.3 (c)

Strategy in parts (a) and (b) can divide image blocks more evenly into each process than lab9.

1.4 (d)

```
function SUBIMAGE_ROWS(M, P, p)
    r ← M%P
    q ← int(M/P)
    if p < r then
        M_p ← q + 1
        j_p ← (q + 1) * p
    else
        M_p ← q
        j_p ← (q + 1) * r + q * (p − r)
    end if
    return M_p, j_p
end function
```

1.5 (e)

```
function DISTRIBUTE_SUBIMAGES(M, N, D, P, p, M_p, D_p)
    ALLOCATE(P, sendcnts)
    ALLOCATE(P, displs)
    for i = 0 to P − 1 do
        M_i, J_i ← SUBIMAGE_ROWS(M, P, i)
        sendcnts[i] ← M_i * N
        displs ← J_i * N
    end for
    MPI_SCATTERv(D, sendcnts, displs, IMAGE_DATATYPE, D_p, sendcnts[p], 0, MPI_COMM_WORLD)
    return D_p
end function
```

1.6 (f)

This may cause process 0 out of memory

1.7 (g)

```
function LOAD_IMAGE_EFFICIENTLY(image_filename)
    p ← GET_rank(COMMUNICATOR)
    P ← GET_SIZE(COMMUNICATOR)
    if P = 0 then
        M, N ← READ_IMAGE_SIZE(image_filename)
        SEND_BROADCAST(COMMUNICATOR, (M, N))
    end if
    M_p, j_p ← SUBIMAGE_ROWS(M, P, p)
    ALLOCATE_SUBIMAGE(M_p, N, D_p)
    D_p ← READ_SUBIMAGE(image_filename, M, N, D, P, p, M_p, D_p)
    return M, N, M_p, D_p
end function
```

2 Calculate basic image statistics

2.1 (a)

```
function CALC_MIN(M, N, P, p, M_p, D_p)
    submin ← D_p[0][0]
    for i = 0 to M_p − 1 do
        for j = 0 to N − 1 do
            if D_p[i][j] < submin then
                submin ← D_p[i][j]
            end if
        end for
    end for
    if P! = 0 then
        MPI_SEND(&submin, 1, MPI_INT, 0, 0, MPI_COMM_WORLD)
    else
        min ← submin
        for i = 1 to P − 1 do
            MPI_RECV(&min_item, 1, MPI_INT, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE)
            if min_item < min then
                min ← min_item
            end if
        end for
        return min
    end if
end function
```

2.2 (b)

```
function CALC_MAX(M, N, P, p, M_p, D_p)
    submax ← D_p[0][0]
    for i = 0 to M_p − 1 do
        for j = 0 to N − 1 do
            if D_p[i][j] > submax then
                submax ← D_p[i][j]
            end if
        end for
    end for
```

```
    if P != 0 then
        MPI_SEND(&submax, 1, MPI_INT, 0, 0, MPI_COMM_WORLD)
    else
        max ← submax
        for i = 1 to P − 1 do
            MPI_RECV(&max_item, 1, MPI_INT, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE)
            if max_item > max then
                max ← max_item
            end if
        end for
        return max
    end if
end function
```

2.3 (c)

```
function CALC_MEAN(M, N, P, p, M_p, D_p)
    mean ← 0
    sum ← 0
    for i = 0 to M_p − 1 do
        for j = 0 to N − 1 do
            sum ← sum + Dp[i][j]/M/N
        end for
    end for
    MPI_Reduce(&sum, &mean, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD)
    if p = 0 then
        return mean
    end if
end function
```

2.4 (d)

Due to the standard deviation of the calculation, the mean value is required.

2.5 (e)

```
function CALC_STDDEV(M, N, P, p, M_p, D_p)
    mean ← CALC_MEAN(M, N, P, p, M_p, D_p)
    stddev ← 0
    sum ← 0
    for i = 0 to M_p − 1 do
        for j = 0 to N − 1 do
            sum ← sum + (D_p[i][j] − mean)^2/M/N
        end for
    end for
    MPI_Reduce(&sum, &mean, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD)
    if p = 0 then
        stddev ← sqrt(mean)
        return stddev
    end if
end function
```

2.6 (f)

Because the global mean can be converted to: find the local mean of each block, and then average all the blocks. But it is difficult to translate the global median into sub-problems that are not interdependent.

3 Computing image histograms

3.1 (a)

```
function CALC_HISTOGRAM(image_filename, K)
    M, N, D ← LOAD_IMAGE(image_filename)
    H_array ← ALLOCATE_IMAGE(K, DATATYPE_FLOAT)
    for i = 0 to M − 1 do
        for j = 0 to N − 1 do
            for c = 0 to K − 1 do
                if D[i][j] = 1 then
                    H_array[K − 1] ← H_array[K − 1] + 1
                    break
                end if
                if D[i][j] < (1/K) ∗ (c + 1) then
                    H_array[c] ← H_array[c] + 1
                    break
                end if
            end for
        end for
    end for
    return H_array
end function
```

3.2 (b)

```
function CALC_HISTOGRAM_PARA(image_filename, K)
    p ← GET_RANK(COMMUNICATOR)
    P ← GET_SIZE(COMMUNICATOR)
    M, N, M_p, D_p ← LOAD_IMAGE_EFFICIENTLY(image_filename)
    ALLOCATE(K, subH_array)
    subH_array ← CALC_HISTOGRAM(image_filename, K)
    if p! = 0 then
        MPI_SEND(subH_array, K, MPI_FLOAT, 0, 0, MPI_COMM_WORLD)
    else
        ALLOCATE(K, H_array)
        allH_array ← COPY(subH_array)
        for i = 1 to P − 1 do
            MPI_RECV(subH_array, K, MPI_FLOAT, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE)
            for j = 0 K − 1 do
                allH_array[j] ← allH_array[j] + subH_array[j]
            end for
        end for
        return allH_array
    end if
end function
```

4 Weak scalability studies

4.1 (a)

$$N_p = \lfloor N_1 * \sqrt{P} + 0.5 \rfloor \tag{1}$$

4.2 (b)

$$N_p = \lfloor N_1 * P^{\frac{1}{3}} + 0.5 \rfloor \tag{2}$$

4.3 (c)

for i in 2 4 8 16 32 64 128 256 **do**
    $mpirun\ -np\ \$i\ \ ./time\_matmat$ 10000
end for