**Problem 1:** Using data parallel programming mode to write a parallel program that adds two vectors:

```c
float v1[100], v2[100];
int nprocs; /* number of processes */
for (i = 0; i < 100; i++)
    v1[i] = v1[i] + v2[i];
```

**Solution:**

```c
float v1[100], v2[100];
int nprocs; /* number of processes */

int main()
begin
    read(nprocs);
    initialize(v1);
    initialize(v2);
    Add(v1, v2);
end

procedure Add(v1, v2)
begin
    DECOMP v1[BLOCK, nprocs];
    DECOMP v2[BLOCK, nprocs];
    for_all (i = 0; i < 100; i++)
        v1[i] = v1[i] + v2[i];
end_for_all
end
```

**Problem 2:** Using shared memory model and message passing model to write parallel programs that add up the elements of a float-point array (A), whose size (n) and data are input by users. ‘nprocs’ processes are used. You can assume that \( n \mod nprocs = 0 \).

```c
float *A;
int n; /* size of A */
int nprocs; /* number of processes */
float total = 0;

for (i = 0; i < n; i++)
    total += A[i];
```

**Solutions:**
(1) **Based on shared memory model:**

```c
int n, nprocs; /* matrix dimension and number of processors to be used */
float *A; /* A is a global, shared array */
float total = 0;
LOCKDEC(total_lock); /* declaration of lock to enforce mutual exclusion */

int main()
begin
    read(n);
    read(nprocs);
    A ← G_MALLOC (a float array);
    initialize(A); /* initialize A */
    CREATE (nprocs–1, Add, A);
    Add(A); /* main process becomes a worker too*/
    WAIT FOR END (nprocs–1); /* wait for all child processes created to terminate*/
end main

procedure Add(A)
    float *A; /* A is the shared array */
    int i;
    int mymin, mymax;
    float myTotal = 0;
    begin
        mymin = (pid * n/nprocs); /* assume that n is exactly divisible by nprocs */
        mymax = mymin + n/nprocs – 1;
        BARRIER(bar, nprocs);
        for i ← mymin to mymax do /* for each of my rows */
        mytotal += A[i];
    endfor
    LOCK(total_lock); /* update global total */
    total += myTotal; /* critical section */
    UNLOCK(total_lock);
end procedure
```

(2) **Based on message passing model:**

```c
int n, nprocs; /* array size and number of processors to be used*/
float *myA;
float mytotal = 0
int main()
begin
    read(n);
```
read(nprocs);
CREATE (nprocs-1, Add);
Add();    /*main process becomes a worker too*/
WAIT_FOR_END (nprocs–1);  /*wait for all child processes created to
terminate*/
end main

procedure Add()
begin
  int i, pid, n’ = n/nprocs;
  int temptotal = 0;
  myA ← malloc(n’ * sizeof(float)); /* my assigned elements of A*/
  initialize(myA);

  for i ← 0 to n’ do /* for each of my elements */
    mytotal += A[i];
  if (pid != 0) then
    SEND(mytotal, sizeof(float), 0, TOTAL);
  else    /*pid 0 does this*/
    for i ← 1 to nprocs-1 do /* process 0 holds global total */
      RECEIVE(&temptotal, sizeof(float), *, TOTAL);
      mytotal += temptotal; /*accumulate into mytotal of
pid0 */
  endif
  endfor
end procedure

Problem 3: Given two character strings s1 and s2. Using shared memory model and
message passing model to write parallel programs to find out the number of substrings, in
string s1, that is exactly the same as s2. The strings are ended with ‘\0’. For example,
suppose $\text{number\_substring}(s1, s2)$ implements the function, then
\begin{align*}
\text{number\_substring}(&\text{abcdab}, \text{ab}) &= 2, \\
\text{number\_substring}(&\text{aaa}, \text{a}) &= 3, \\
\text{number\_substring}(&\text{abac}, \text{bc}) &= 0.
\end{align*}
The size of s1 and s2 (n1 and n2) as well as their
data are inputs from users. ‘nprocs’ processes are used. You can assume that $n1 \mod nprocs = 0$ (n1 is the size of s1).

char *s1, *s2;
int n1, n2; /* sizes of s1 and s2 */
int nprocs; /* number of processes */

Solutions:

(1) Based on shared memory model:
char *s1, *s2;
int n1, n2; /* sizes of s1 and s2 */
int nprocs; /* number of processes */
int total_matches = 0;
LOCKDEC(total_lock); /* declaration of lock to enforce mutual exclusion */

int main()
begin
    read(s1);
    read(s2);
    read(nprocs);
    n1 = string_size(s1);
    n2 = string_size(s2);

    CREATE(nprocs-1, SubString);
    SubString();
    WAIT_FOR_END (nprocs–1); /*wait for all child processes created to terminate*/
end main

procedure SubString()
int i;
int mymin, mymax;
float my_matches = 0;
begin
    int mymin = pid * n/nprocs; /* assume that n is exactly divisible by nprocs */
    int mymax = mymin + n/nprocs – 1;
    for i ← mymin to mymax do /* for each of my chars */
        my_matches += string_match(s1, i, s2);
    endfor
    LOCK(total_lock); /* update global total */
    total_matches += my_matches; /* critical section */
    UNLOCK(total_lock);
end procedure

/* check if s2 matches a substring of s1 that starts at position ‘pos’ */
int string_match(s1, pos, s2)
char *s1, s2; int pos;
begin
    int i = 0;
    while (s1[pos+i] != ‘\0’ && s2[i] != ‘\0’ && s1[pos+i] == s2[i])
        i++;

if (s2[i] == ‘\0’)
    return 1;
else
    return 0;
end

(2) Based on message passing model:

int n, nprocs; /* array size and number of processors to be used*/
int n1, n2;
float my_matches = 0, temp_matches = 0;

int main()
begin
    read(n1); /* n1 is the size of s1 */
    read(nprocs);
    CREATE (nprocs-1, SubString);
    SubString(); /*main process becomes a worker too*/
    WAIT_FOR_END (nprocs–1); /*wait for all child processes created to terminates*/
end main

procedure SubString()
begin
    char *my_s1, s2;
    int temp_matches;
    int i, pid, n’ = n1/nprocs;
    /* the extra (n2-1) spaces are for reception of data from neighboring process */
    my_s1 ← malloc( (n’+n2) * sizeof(char));
    /* assume that s1 is evenly partitioned in the nprocs processes */
    initialize(my_s1);
    initialize(s2);

    /* passing boundary data, assuming that 0< n2 <= n’+1 */
    if (pid != 0 )
        SEND(my_s1, (n2-1)*sizeof(char), pid-1, BOUNDARY);
    if (pid != nprocs-1 )
        RECEIVE(my_s1+n’, (n2-1)*sizeof(char), pid+1, BOUNDARY);
        my_s1[n’n2] = ‘\0’;
    }

    for i ← 0 to n’-1 do /* for each of my substrings */
        my_matches += string_match(my_s1, i, s2);
    endfor
if (pid != 0) then
    SEND(my_matches, sizeof(int), 0, TOTAL);
else
    for i ← 1 to nprocs-1 do
        RECEIVE(&temp_matches, sizeof(int), *, TOTAL);
        my_matches += temp_matches;
    endfor
endif
end procedure

/* check if s2 matches a substring of s1 that starts at position ‘pos’ */
int string_match(s1, pos, s2)
char *s1, s2; int pos;
begin
    int i = 0;

    while (s1[pos+i] != ‘\0’ && s2[i] != ‘\0’ && s1[pos+i] == s2[i])
        i++;

    if (s2[i] == ‘\0’)
        return 1;
    else
        return 0;
end