Example: Multi-threaded way of solving a computational problem

Typical application is heat wave propagation computation:

-- Averaging and testing for convergence

(a) Single Threaded Solution &
(b) Multi-threaded approach using Shared Memory Multiprocessor System
(c) Message Passing approach (suitable for HPC)

```
Procedure Solve(A)        /* Averaging operation 5 elements */
begin
  diff = done = 0;
  while (!done) do
     diff = 0;
     for i ← 1 to n do
       for j ← 1 to n do
          temp = A[i,j]; /* saving the original value */
          A[i,j] ← 0.2 * (A[i,j] + neighbors);
          diff += abs(A[i,j] – temp); /* difference computation */
       end for
     end for
     if (diff < TOL) then done = 1;   /* convergence testing */
  end while
end procedure
```

SINGLE THREDED EXECUTION

```
int  n, nprocs;
float  **A, diff;
LOCKDEC(diff_lock);
BARDEC(bar1);


main()
begin
  read(n); read(nprocs);
  A ← G_MALLOC();
  initialize (A);
  CREATE (nprocs,Solve,A);
  WAIT_FOR_END (nprocs);
end main
```

MULTI-THREDED EXECUTION - SM

```
procedure Solve(A)
  int i, j, pid, done=0;
  float temp, mydiff=0;
  int mymin = 1 + (pid * n/procs);
  int mymax = mymin + n/nprocs -1;
  while (!done) do
    mydiff = diff = 0;
    BARRIER(bar1,nprocs);
    for i ← mymin to mymax
      for j ← 1 to n do
        ...
      endfor
    endfor
    LOCK(diff_lock);
    diff += mydiff;
    UNLOCK(diff_lock);
    BARRIER (bar1, nprocs);
    if (diff < TOL) then done = 1;
    BARRIER (bar1, nprocs);
  endwhile
```

```
main()
  read(n); read(nprocs);
  CREATE (nprocs-1, Solve);
  Solve();
  WAIT_FOR_END (nprocs-1);

procedure Solve()
  int i, j, pid, nn = n/nprocs, done=0;
  float temp, tempdiff, mydiff = 0;
  myA ← malloc(...)
  initialize(myA);
  while (!done) do
    mydiff = 0;
    if (pid != 0)
      SEND(&myA[1,0], n, pid-1, ROW);
    if (pid != nprocs-1)
      SEND(&myA[nn,0], n, pid+1, ROW);
    if (pid != 0)
      RECEIVE(&myA[0,0], n, pid-1, ROW);
    if (pid != nprocs-1)
      RECEIVE(&myA[nn+1,0], n, pid+1, ROW);
    for i ← 1 to nn do
      for j ← 1 to n do
        ...
      endfor
    endfor
    if (pid != 0)
      SEND(mydiff, 1, 0, DIFF);
      RECEIVE(done, 1, 0, DONE);
    else
      for i ← 1 to nprocs-1 do
        RECEIVE(tempdiff, 1, *, DIFF);
        mydiff += tempdiff;
      endfor
      if (mydiff < TOL) done = 1;
      for i ← 1 to nprocs-1 do
        SEND(done, 1, I, DONE);
      endfor
    endif
  endwhile
```

MESSAGE PASSING APPROACH