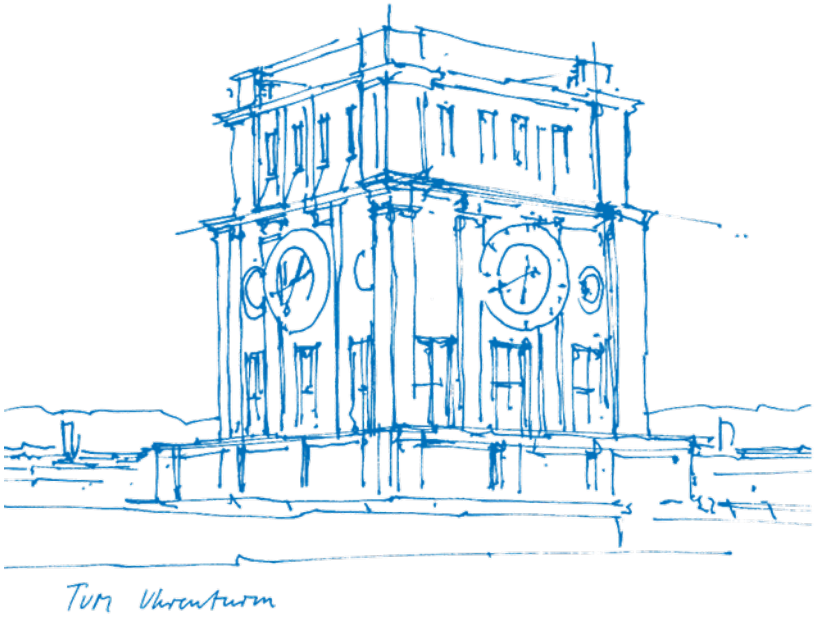
Parallel Programming Tutorial - Profiling tools

M.Sc. Bengisu Elis
Philipp Czerner
Hasan Ashraf
Technical University of Munich
17. July 2019



MPI - Non-blocking collectives

Example; blocking collective

```
int main(int argc, char *argv[])
{
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    double a[SIZE],b[SIZE],c[SIZE];
    srand(time(NULL));
    double sum_a=0, sum_b=0, sum_c=0;
    double avg_a=0, avg_b=0, avg_c=0;
    double min_a=RANGE, min_b=RANGE, min_c=RANGE;
    double max_a=-1, max_b=-1, max_c=-1;

    for (int i = 0; i < SIZE; ++i) { // init
      a[i]=rand()%RANGE; b[i]=rand()%RANGE;
      c[i]=rand()%RANGE;
    }
    for (int i = 0; i < SIZE; ++i) {
      sum_a+=a[i]; // partial sums over array "a"
    }
    avg_a = sum_a / SIZE;
    MPI_Allreduce(&avg_a, &avg_a, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
    avg_a/=size; // aggregate the average over all processes
```

Example; blocking collective (Cont.)

```
    for (int i = 0; i < SIZE; ++i) {
      b[i]*=avg_a;
    }
    for (int i = 0; i < SIZE; ++i) {
      min_b=MIN(min_b, b[i]);
      max_b=MAX(max_b, b[i]);
    }
    MPI_Allreduce(&min_b, &min_b, 1, MPI_DOUBLE, MPI_MIN, MPI_COMM_WORLD);
    MPI_Allreduce(&max_b, &max_b, 1, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);

    for (int i = 0; i < SIZE; ++i) {
        c[i]+=avg_a;
        c[i]+=max_b/2.0;
        c[i]+=min_b/2.0;
        sum_c+=c[i];
    }
    avg_c = sum_c / SIZE;
    MPI_Allreduce(&avg_c, &avg_c, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
    avg_c/=size;

    MPI_Finalize();
    return 0;
}
```

Example; non-blocking collective

```
...

  for (int i = 0; i < SIZE; ++i) {
      min_b=MIN(min_b, b[i]);
      max_b=MAX(max_b, b[i]);
  }

  MPI_Request req_min, req_max;
  double temp_min = min_b, temp_max = max_b;
  MPI_Iallreduce(&temp_min, &min_b, 1, MPI_DOUBLE, MPI_MIN, MPI_COMM_WORLD, &req_min);
  MPI_Iallreduce(&temp_max, &max_b, 1, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD, &req_max);

  for (int i = 0; i < SIZE; ++i) c[i]+=avg_a;
  MPI_Wait(&req_max, MPI_STATUS_IGNORE);

  for (int i = 0; i < SIZE; ++i) c[i]+=max_b/2.0;
  MPI_Wait(&req_min, MPI_STATUS_IGNORE);

  for (int i = 0; i < SIZE; ++i) c[i]+=min_b/2.0;
  for (int i = 0; i < SIZE; ++i) sum_c+=c[i];

  ...
```

mpiP; a lightweight MPI Profiling tool

mpiP; a lightweight MPI Profiling tool

- Open source; https://github.com/LLNL/mpiP
- Portable
- easy-to-use; single output file

Usage:

- Option1: add libmpip.a/.so to the link line
- Option2: set LD_PRELOAD to mpiP
- compile with -g for better accuracy

```
mpiP:
mpiP: mpiP: mpiP V3.4.2 (Build Jul 16 2019/16:35:20)
mpiP: Direct questions and errors to mpip-help@googlegroups.com
mpiP:
Time: 0.008275 seconds
Found 874 occurence(s) of string 'you'
mpiP:
mpiP: Storing mpiP output in [./search_par.2.87447.1.mpiP].
mpiP:
```

Profiling first MPI homework

step 1: install mpiP:

```
>> git clone https://github.com/LLNL/mpiP.git
>> cd mpiP
>> ./configure
>> make all
```

step 2: open up the Makefile and apply the following changes:

```
change line#  4 to -> LDFLAGS = --lm -lrt -I ($CURDIR)$ -L<path to libmpiP.so> -lmpiP -ldl -lm -lunwind
change line# 14 to -> CFLAGS += -g
```

step 3: compile your code

```
>> make
```

step 4: run

```
>> mpirun -np 3 ./student/search_par treasure_island.txt <string to search in file>
```

Output - Metadata

```
@ mpiP
@ Command : ./student/search_par treasure_island.txt you
@ Version                  : 3.4.2
@ MPIP Build date          : Jul 16 2019, 19:44:26
@ Start time               : 2019 07 16 19:46:41
@ Stop time                : 2019 07 16 19:46:41
@ Timer Used               : PMPI_Wtime
@ MPIP env var             : [null]
@ Collector Rank           : 0
@ Collector PID            : 21242
@ Final Output Dir         : .
@ Report generation        : Single collector task
@ MPI Task Assignment      : 0 i10se1
@ MPI Task Assignment      : 1 i10se1
@ MPI Task Assignment      : 2 i10se1
```

Output - Overview

```
---------------------------------------------------------------------------
@--- MPI Time (seconds) ---------------------------------------------------
---------------------------------------------------------------------------
Task    AppTime    MPITime     MPI%
   0    0.00899   0.000998    11.11
   1    0.00869    0.00573    65.88
   2    0.00879    0.00597    67.85
   *     0.0265     0.0127    47.94
```

Output - Callsites

```
@--- Callsites: 10 ---------------------------------------------------------
------------------------------------------------------------------------------
 ID Lev File/Address        Line Parent_Funct             MPI_Call
  1   0 search_par.c          73 search_text              Recv
  2   0 main.c               118 main                     Bcast
  3   0 search_par.c          47 search_text              Send
  4   0 main.c               117 main                     Bcast
  5   0 main.c               119 main                     Bcast
  6   0 main.c               126 main                     Bcast
  7   0 search_par.c          66 search_text              Send
  8   0 main.c               127 main                     Bcast
  9   0 main.c               125 main                     Bcast
 10   0 search_par.c          52 search_text              Recv
```

Output - per Function Timing and Message Size

```
---------------------------------------------------------------------------
@--- Aggregate Time (top twenty, descending, milliseconds) ----------------
---------------------------------------------------------------------------
Call                 Site       Time     App%    MPI%      Count    COV
Bcast                   9         11    41.49   86.55          2   0.00
Recv                   10      0.648     2.45    5.10          2   0.48
Bcast                   2      0.495     1.87    3.90          1   0.00
Send                    3      0.459     1.73    3.62          2   0.00
......

---------------------------------------------------------------------------
@--- Aggregate Sent Message Size (top twenty, descending, bytes) ----------
---------------------------------------------------------------------------
Call                 Site      Count      Total       Avrg  Sent%
Send                    3          2   4.05e+05   2.02e+05  99.96
Bcast                   9          2         80         40   0.02
Bcast                   2          1         40         40   0.01
.........
```

Output - Callsite Time statistics

```
---------------------------------------------------------------------------
@--- Callsite Time statistics (all, milliseconds): 15 ----------------------
---------------------------------------------------------------------------
Name                 Site  Rank   Count       Max      Mean       Min   App%   MPI%
Bcast                   2     0       1     0.495     0.495     0.495   5.51  49.62
Bcast                   2     *       1     0.495     0.495     0.495   1.87   3.90

Bcast                   4     0       1   0.00436   0.00436   0.00436   0.05   0.44
Bcast                   4     *       1   0.00436   0.00436   0.00436   0.02   0.03
........

Send                    7     1       1    0.0122    0.0122    0.0122   0.14   0.21
Send                    7     2       1    0.0179    0.0179    0.0179   0.20   0.30
Send                    7     *       2    0.0179    0.0151    0.0122   0.11   0.24
```

Output - Callsite Message statistics

```
---------------------------------------------------------------------------
@--- Callsite Message Sent statistics (all, sent bytes) -------------------
---------------------------------------------------------------------------
Name                 Site Rank    Count        Max       Mean        Min        Sum
Bcast                   2    0        1         40         40         40         40
Bcast                   2    *        1         40         40         40         40

Bcast                   4    0        1          4          4          4          4
Bcast                   4    *        1          4          4          4          4

.......

Send                    3    0        2 2.023e+05 2.023e+05 2.023e+05 4.046e+05
Send                    3    *        2 2.023e+05 2.023e+05 2.023e+05 4.046e+05

Send                    7    1        1          4          4          4          4
Send                    7    2        1          4          4          4          4
Send                    7    *        2          4          4          4          8
---------------------------------------------------------------------------
@--- End of Report --------------------------------------------------------
---------------------------------------------------------------------------
```

mpiP options

- You can change the parameters get better results.
  - More details
  - Reduce the size of output and also overheads
  - Change the stack trace length
  - Output paths

- You can use environment variables for changing the parameters
  - MPIP = "-c -o -k 4" (stack trace 4, include callsites)

- You can also limit the scope of profiling in the code,
  - MPI_Pcontrol(x)

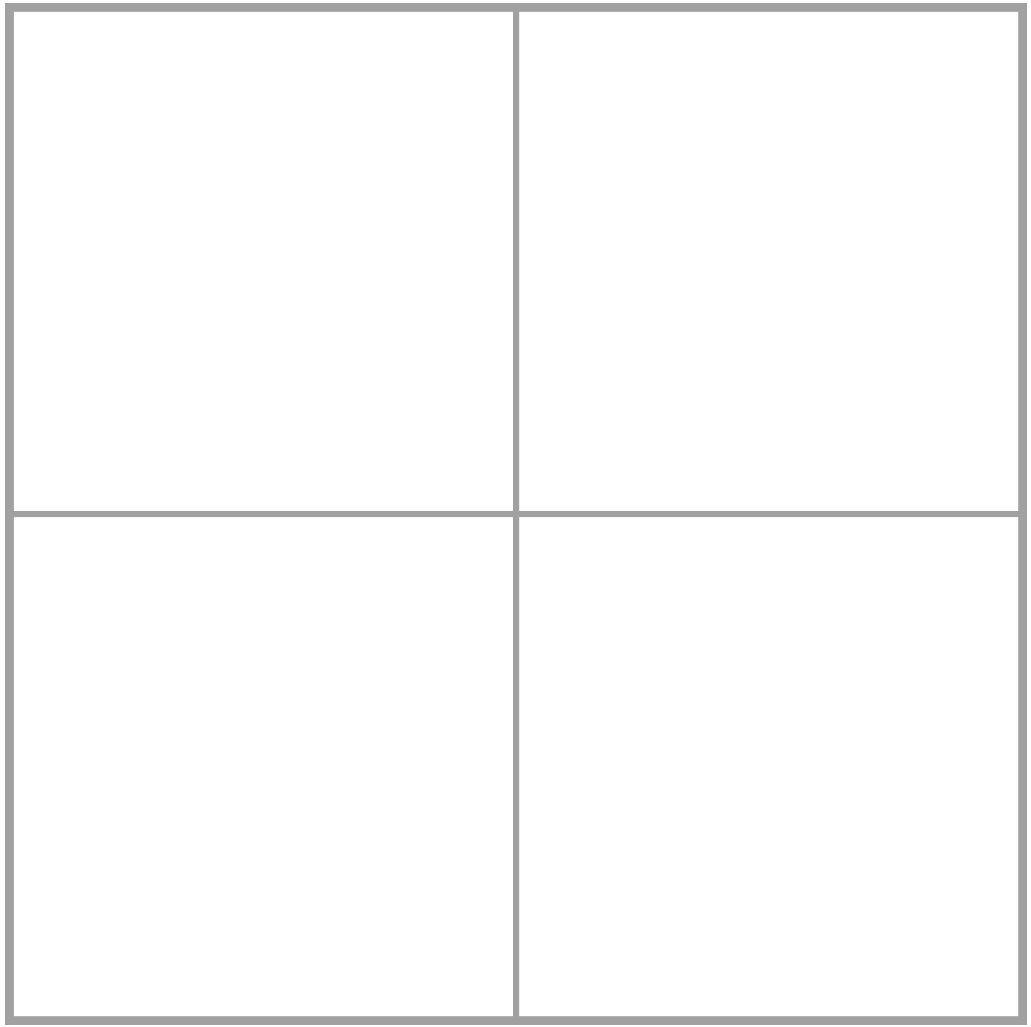
Assignment 10

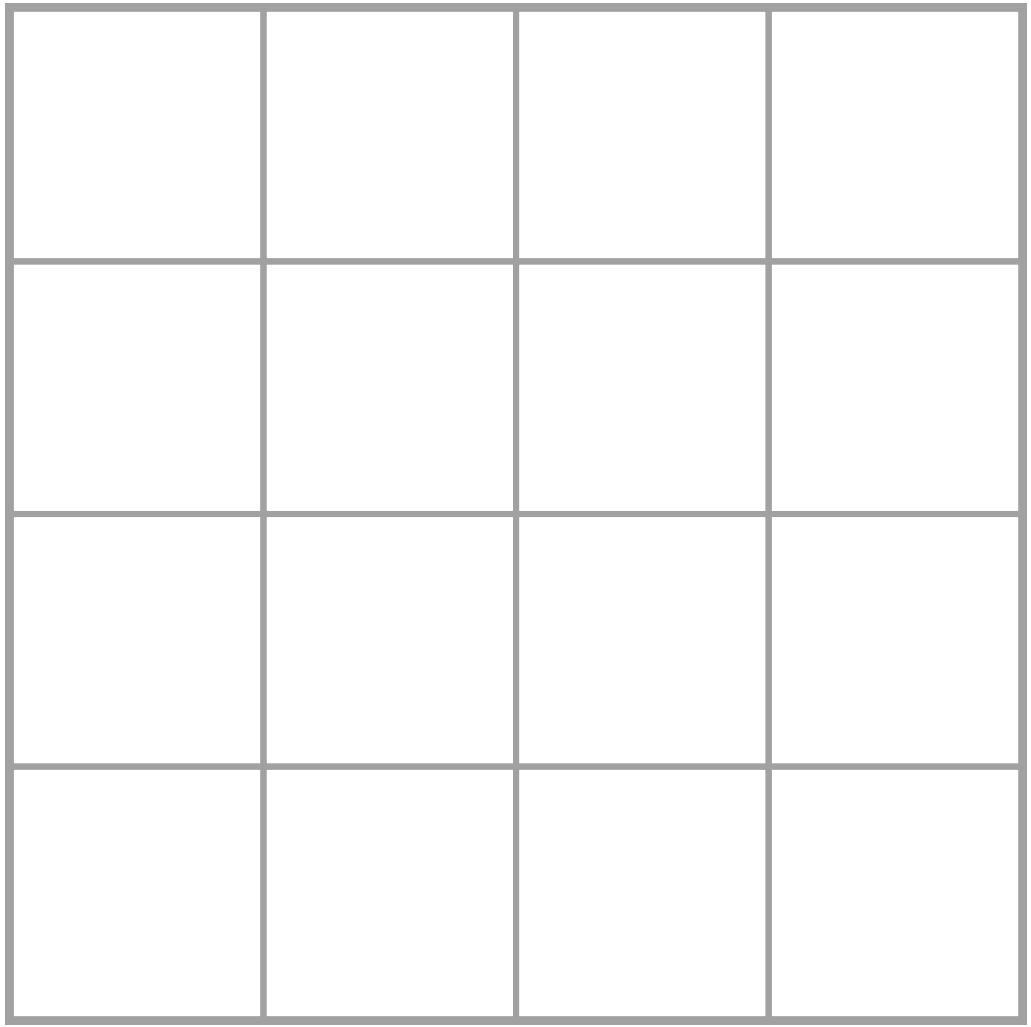
Assignment 10

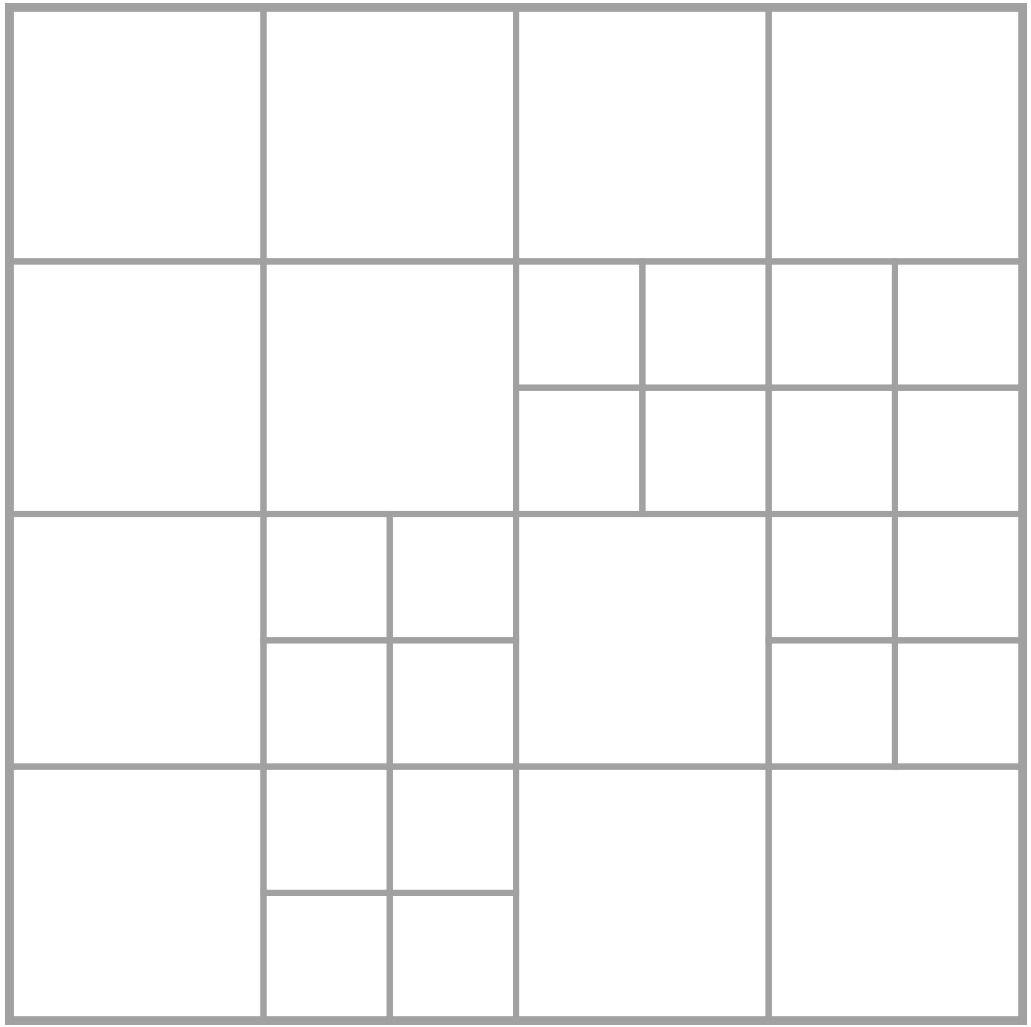
Demo...

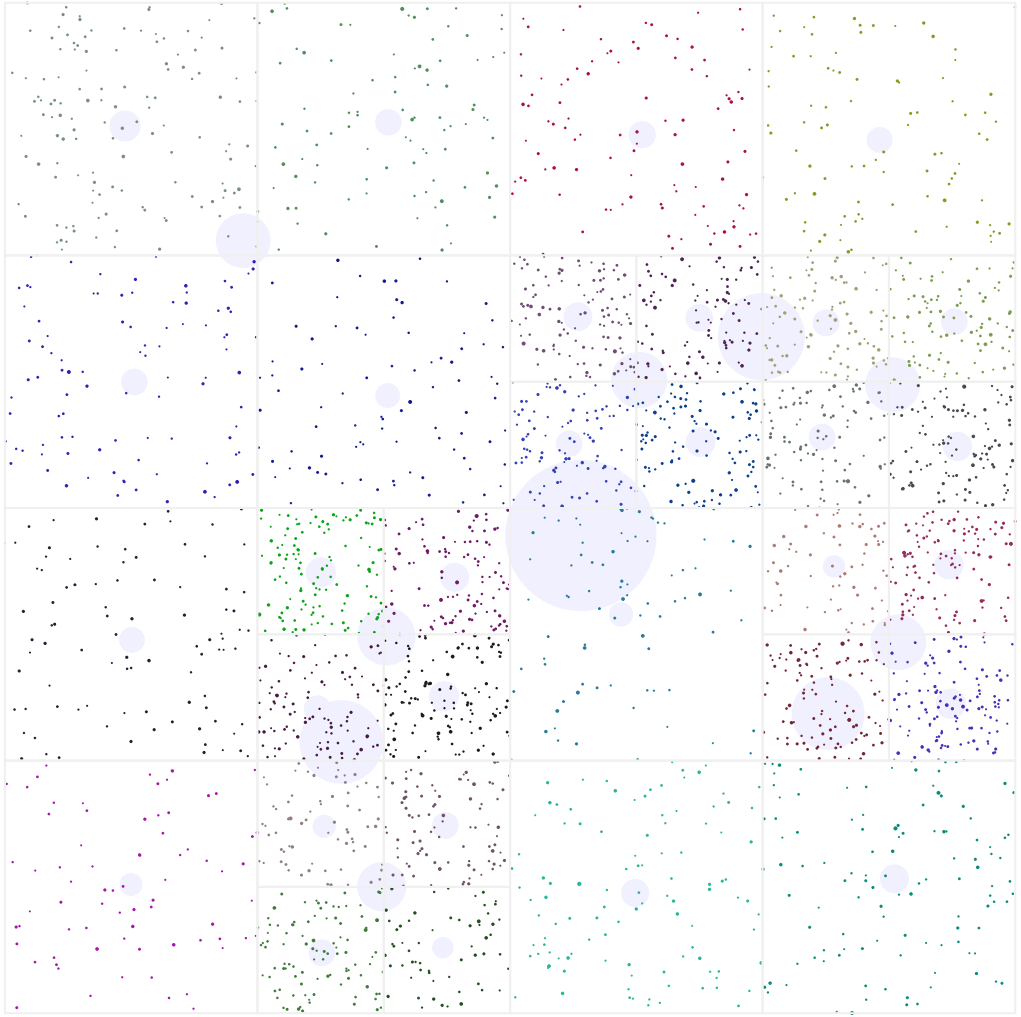
Assignment 10

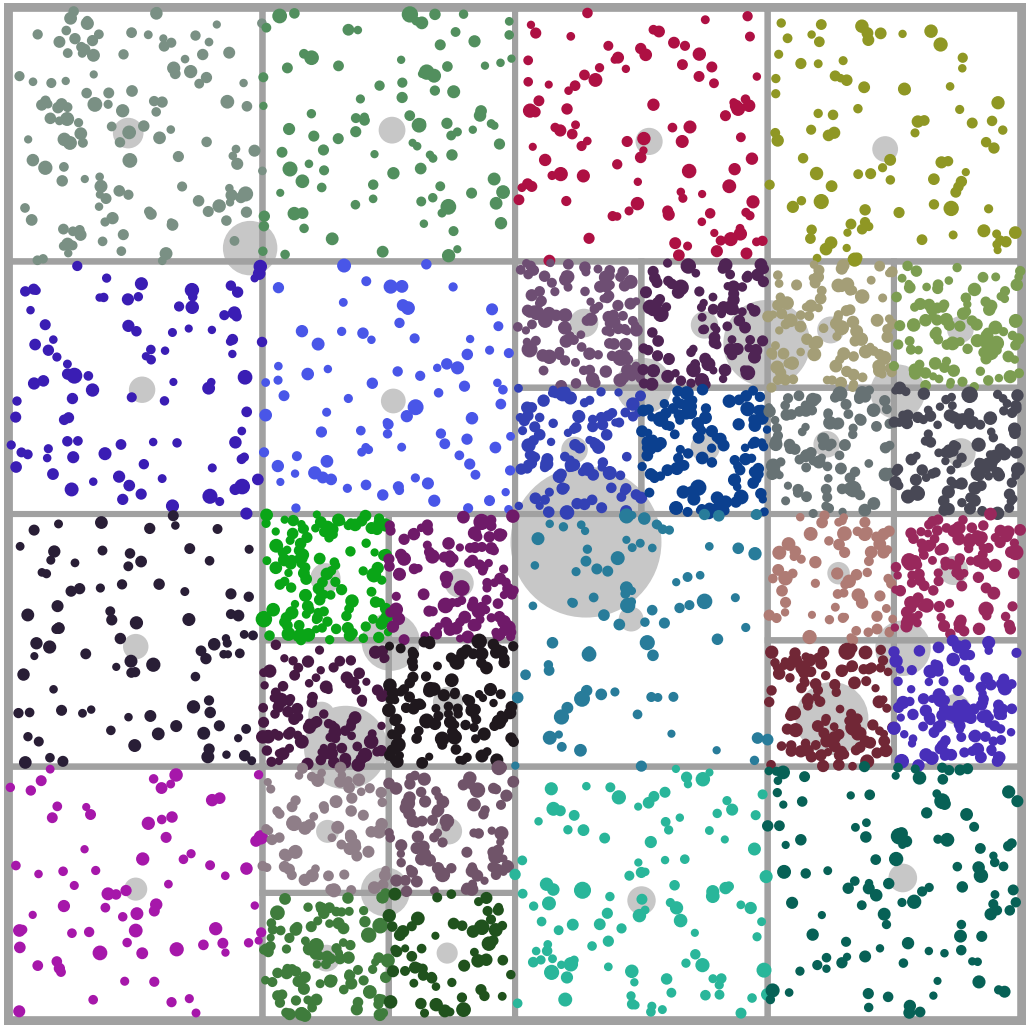
- You have to parallelise an n-body simulation using MPI
- Hierarchical (non-adaptive) domain decomposition (quadtree)
- Speedup ≥ 12 using 31 processes
- Deadline is 24th July 08:00 !

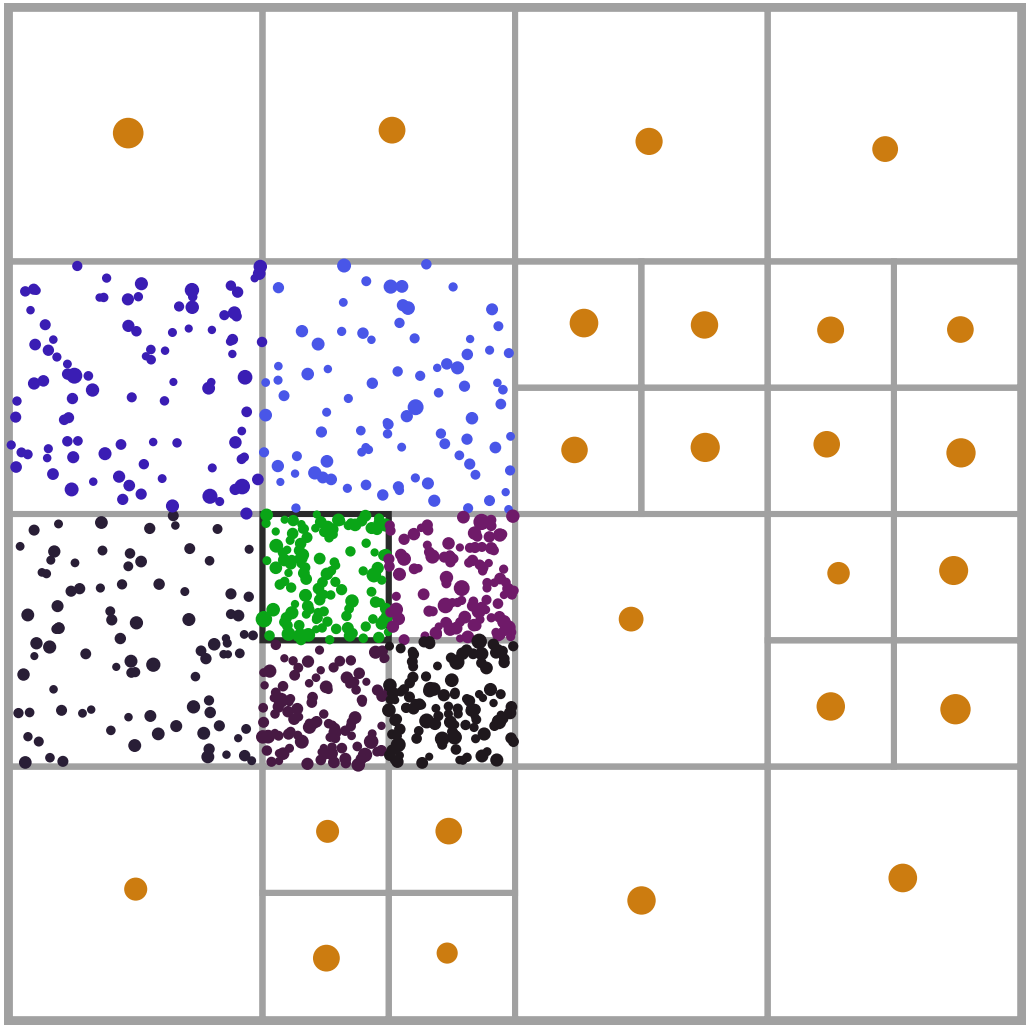








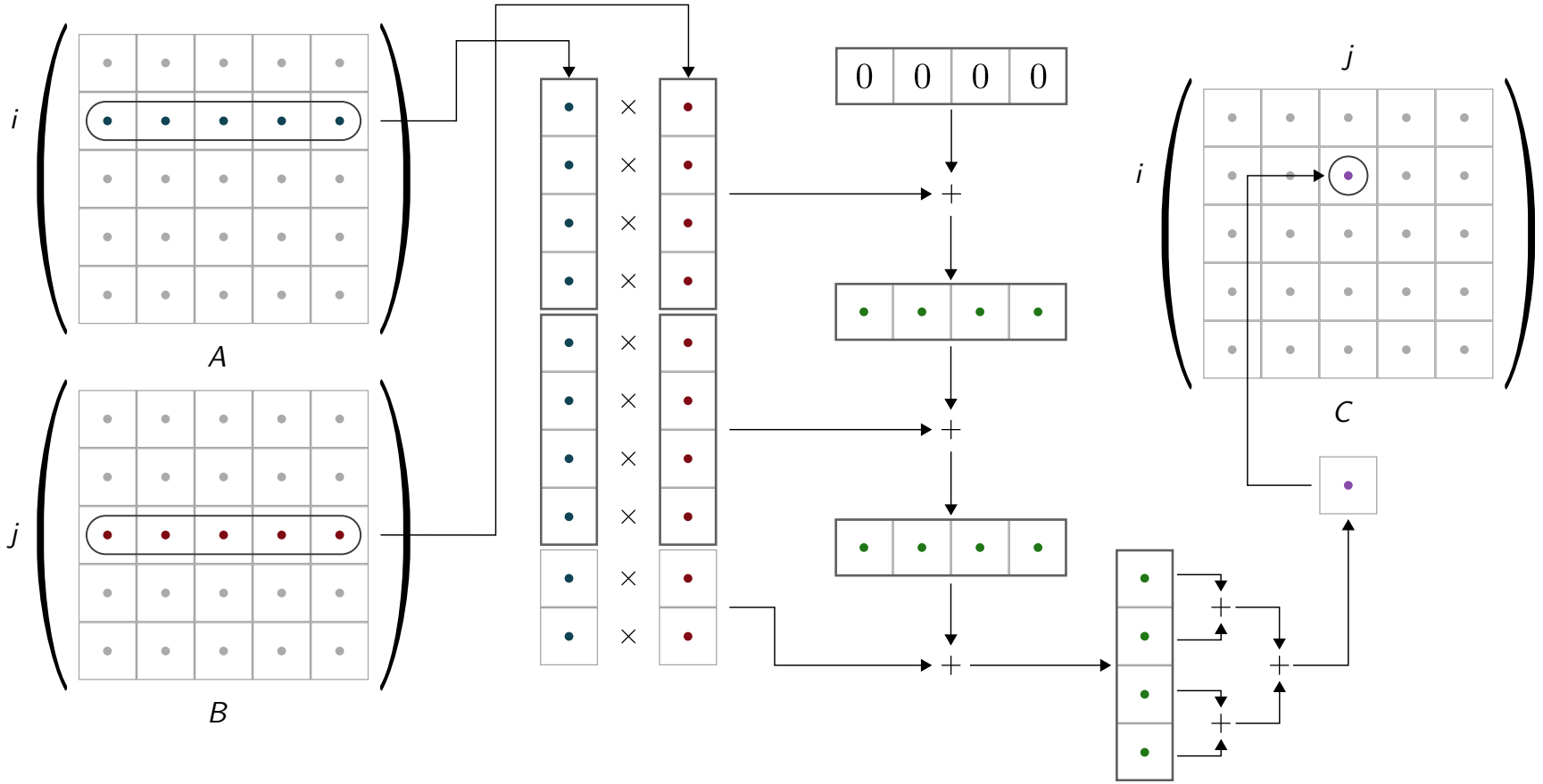


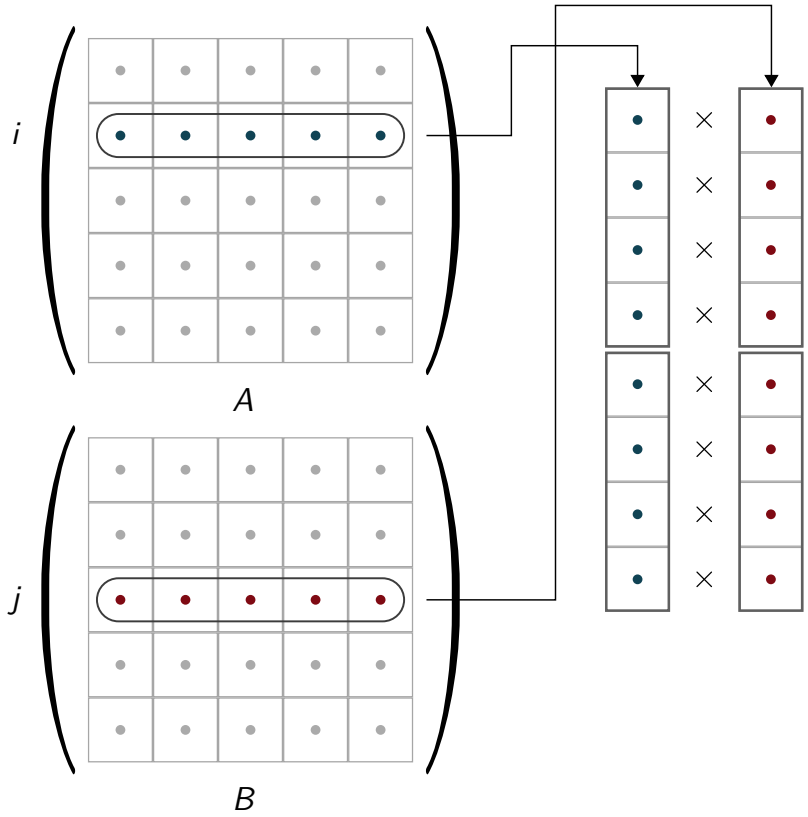


Hints

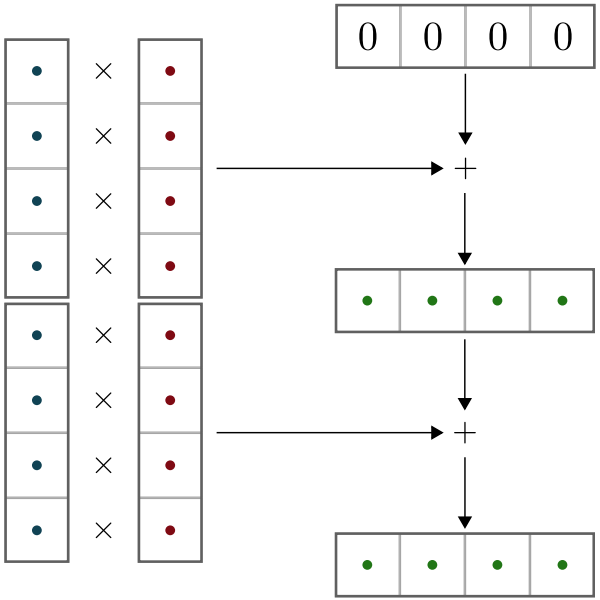
- Read the `README`
- You can inspect the generated SVG files to debug
  - Instead of using `printf` you can also call the drawing routines yourself on an intermediate state
- There is a gui as well, but it only shows rank 0
- The code uses timestamps to verify that the processed data is current
  - You may need to updated them manually for data transmitted between processes
- No need to distribute initial data, but you do need to collect it at the end
- There are precisely 31 leaves, with ids 0 to 30
- Also remember the Q&A sessions, Fr 08-10, 01.06.020

Solution for Assignment 7

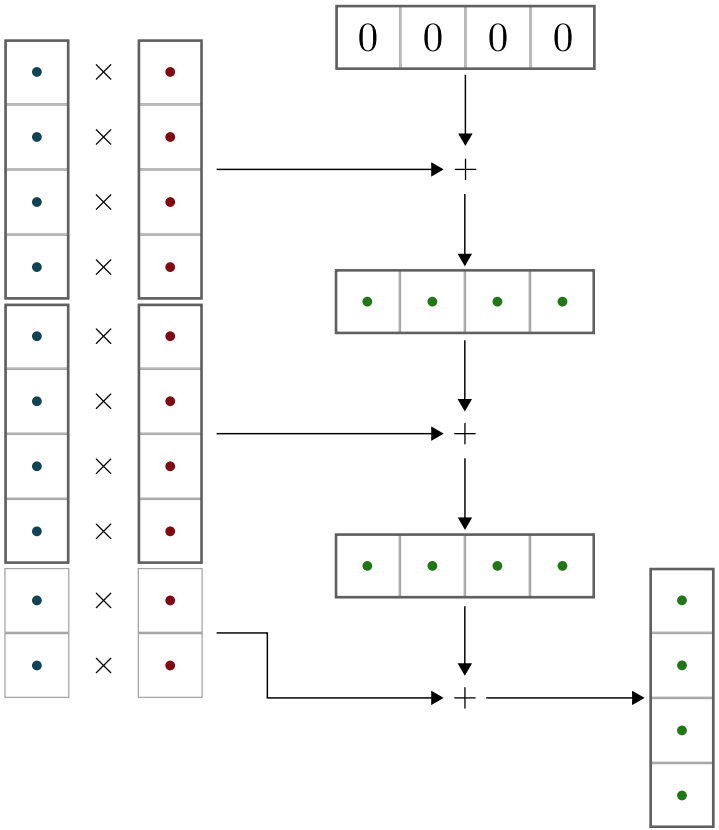

i
A
j
B
0 0 0 0
j
i
C




```
void dgemm(float* a, float* b, float* c, int n) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++){
      for (int k = 0; k + 8 <= n; k += 8) {
        __m256 a_i = _mm256_loadu_ps(a+i*n+k);
        __m256 b_j = _mm256_loadu_ps(b+j*n+k);
        __m256 t0 = _mm256_mul_ps(a_i, b_j);
      }
    }
  }
}
```

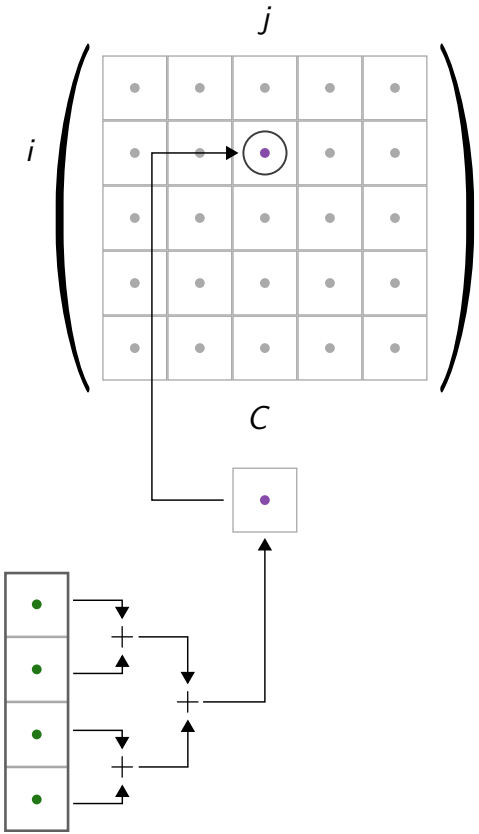



```
void dgemm(float* a, float* b, float* c, int n) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++){
      __m256 temp = _mm256_set1_ps(0.f);
      for (int k = 0; k + 8 <= n; k += 8) {
        __m256 a_i = _mm256_loadu_ps(a+i*n+k);
        __m256 b_j = _mm256_loadu_ps(b+j*n+k);
        __m256 t0 = _mm256_mul_ps(a_i, b_j);
        temp = _mm256_add_ps(temp, t0);
      }
    }
  }
}
```



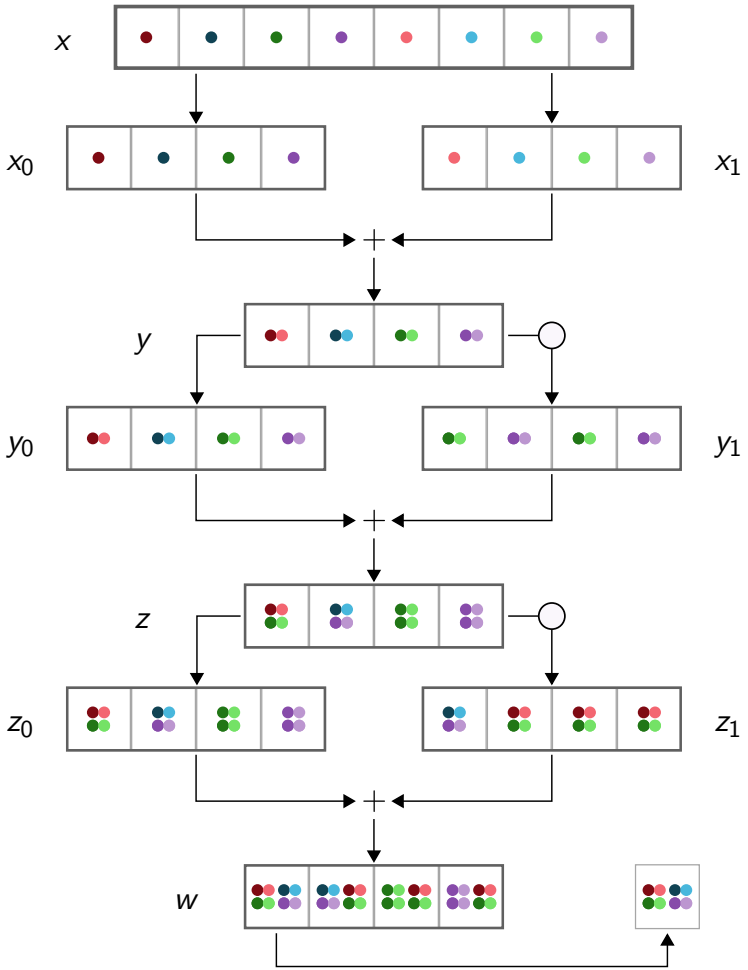

```
void dgemm(float* a, float* b, float* c, int n) {
  for (int i = 0; i < n; i++) {
    // 11...1  n%8 times, then 00...0  8-n%8 times
    __m256i mask = _mm256_setr_epi32(
      -(0 < n%8), -(1 < n%8), -(2 < n%8), -(3 < n%8),
      -(4 < n%8), -(5 < n%8), -(6 < n%8), -(7 < n%8)
    );
    for (int j = 0; j < n; j++){
      __m256 temp = _mm256_set1_ps(0.f);
      int k = 0;
      for (; k + 8 <= n; k += 8) { ... }

      __m256 a_i = _mm256_maskload_ps(a+i*n+k, mask);
      __m256 b_j = _mm256_maskload_ps(b+j*n+k, mask);
      __m256 t0 = _mm256_mul_ps(a_i, b_j);
      temp = _mm256_add_ps(temp, t0);
    }
  }
}
```




```
void dgemm(float* a, float* b, float* c, int n) {
  for (int i = 0; i < n; i++) {
    ...
    for (int j = 0; j < n; j++){
      __m256 temp = ... ;

      c[i*n+j] = sum8(temp);
    }
  }
}
```




```
float sum8(__m256 x) {
  __m128 x0 = _mm256_castps256_ps128(x);
  __m128 x1 = _mm256_extractf128_ps(x, 1);
  __m128 y  = _mm_add_ps(x0, x1);
  __m128 y0 = y;
  __m128 y1 = _mm_movehl_ps(y, y);
  __m128 z  = _mm_add_ps(y0, y1);
  __m128 z0 = z;
  __m128 z1 = _mm_shuffle_ps(z, z, 0x1);
  __m128 w  = _mm_add_ss(z0, z1);
  return _mm_cvtss_f32(w);
}
```

```
void dgemm(float* a, float* b, float* c, int n) {
  for (int i = 0; i < n; i++) {
    // 11...1  n%8 times, then 00...0  8-n%8 times
    __m256i mask = _mm256_setr_epi32(
      -(0 < n%8), -(1 < n%8), -(2 < n%8), -(3 < n%8),
      -(4 < n%8), -(5 < n%8), -(6 < n%8), -(7 < n%8)
    );

    for (int j = 0; j < n; j++){
      __m256 temp = _mm256_set1_ps(0.f);
      int k = 0;
      for (; k + 8 <= n; k += 8) {
        __m256 a_i = _mm256_loadu_ps(a+i*n+k);
        __m256 b_j = _mm256_loadu_ps(b+j*n+k);
        __m256 t0 = _mm256_mul_ps(a_i, b_j);
        temp = _mm256_add_ps(temp, t0);
      }

      // Process remainder
      __m256 a_i = _mm256_maskload_ps(a+i*n+k, mask);
      __m256 b_j = _mm256_maskload_ps(b+j*n+k, mask);
      __m256 t0 = _mm256_mul_ps(a_i, b_j);
      temp = _mm256_add_ps(temp, t0);

      c[i * n + j] = sum8(temp);
} } }
```

```
// Compute the sum of the 8 components of the vector
float sum8(__m256 x) {
  __m128 x0 = _mm256_castps256_ps128(x);
  __m128 x1 = _mm256_extractf128_ps(x, 1);
  __m128 y  = _mm_add_ps(x0, x1);
  __m128 y0 = y;
  __m128 y1 = _mm_movehl_ps(y, y);
  __m128 z  = _mm_add_ps(y0, y1);
  __m128 z0 = z;
  __m128 z1 = _mm_shuffle_ps(z, z, 0x1);
  __m128 w  = _mm_add_ss(z0, z1);
  return _mm_cvtss_f32(w);
}
```

Notes

- A more straightforward remainder loop and horizontal sum (`sum8`) are both possible
  - Only executed once per loop, not a bottleneck
- Summing up all 8 elements in each iteration is slower, but accepted with an efficient horizontal sum implementation

Solution for Assignment 8

```c
/* Distribute Data */
void search_text (char* text, int num_lines, int line_length, char* search_string, int *occurences){
  int rank, num_procs;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &num_procs);

  int num_local_lines = num_lines / num_procs + ( rank < ( num_lines % num_procs));
  char *lines = NULL;
  if (rank != 0)
    {
      lines = (char* )malloc ( sizeof(char ) * num_local_lines * line_length);
    }
  // total_lines contains the number of lines already sent
  int total_lines = num_lines / num_procs + ( 0 < (num_lines % num_procs));
  if (rank == 0)
    {
      lines = text;
      // distribute data to other processes
      for (int i = 1; i < num_procs; i++)
        {
          //  distribute remainder evenly
        int proc_lines = num_lines / num_procs + ( i < (num_lines % num_procs));
          MPI_Send(text + line_length * total_lines , line_length * proc_lines, MPI_CHAR, i, 0, MPI_COMM_WORLD);
          total_lines += proc_lines;
    }
    }
  else
    {
      MPI_Recv(lines, num_local_lines * line_length, MPI_CHAR, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
```

```
/* Compute and Reduce */
  int running_count = 0;
  for (int i = 0; i < num_local_lines; i++)
    {
      running_count += count_occurences( lines + i * line_length, search_string);
    }
  int sum = 0;
  // Reduction
  if (rank != 0)
    {
      MPI_Send( &running_count, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }
  else
    {
      for ( int i = 1; i < num_procs; i++)
        {
          int temp;
          MPI_Recv(&temp, 1, MPI_INT, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE  );
          sum += temp;
        }
    }
  if (rank == 0)
    {
      // add root's own count
      *occurences = sum + running_count;
    }
    else
        free(lines);
}
```

Solution for Assignment 9

```c
/* Define arrays for Scatterv */
void simulate(int height, int width, int grid[height][width], int num_iterations)
{
  int rank, num_procs;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &num_procs);

  int real_height = height - 2;
  // equally distribute rows; + 2 for halo rows
  int local_rows = real_height / num_procs + ( rank < (real_height % num_procs ) ) + 2;

  // define local grid buffers - includes halo rows
  int (*local_grid)[width] = malloc(sizeof(int[local_rows][width]));

  // prepare arrays for Scatterv
  int *sendcounts = malloc(sizeof(int) * num_procs);
  int *displs = malloc(sizeof(int) * num_procs);

  int sum = 0;
  for (int i = 0; i < num_procs; i++)
    {
      int num_rows = (height - 2  ) / num_procs + (i < ( (height - 2) % num_procs ) );
      sendcounts[i] = num_rows * width;
      displs[i] = sum;
      sum += sendcounts[i];
    }
```

```
/* Distribute Data */
  //int MPI_Scatterv(const void *sendbuf, const int *sendcounts, const int *displs,
  //                 MPI_Datatype sendtype, void *recvbuf, int recvcount,
  //                 MPI_Datatype recvtype, int root, MPI_Comm comm)
  MPI_Scatterv( &(grid[1][0]), sendcounts, displs,
                MPI_INT, &(local_grid[1][0]), sendcounts[rank],
                MPI_INT, 0, MPI_COMM_WORLD);

  // define neighbouring ranks
  int top, bottom;
  top = rank - 1;
  bottom = rank + 1;
  if (rank == 0)
    {
      top = num_procs - 1;
    }
  if (rank == num_procs - 1)
    {
      bottom = 0;
    }
  // allocate space for top and bottom halo rows
  int *top_halo = malloc(sizeof(int[width]));
  int *bottom_halo = malloc(sizeof(int[width]));
  // temporary data storage
  int (*temp)[width] = malloc(sizeof(int[local_rows][width]));
  // In each iteration
  // 1. make local rows periodic
  // 2. fix halo rows
  // 3. send recv data
  // 4. compute
```

```
/* Compute */
  for (int i = 0; i < num_iterations; i++)
    {
      // 1.
      make_local_rows_periodic(local_rows, width, local_grid);
      // 2.
      prepare_halo_rows(local_rows, width, local_grid, top_halo, bottom_halo);
      //int MPI_Sendrecv(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
      //                 int dest, int sendtag,
      //                 void *recvbuf, int recvcount, MPI_Datatype recvtype,
      //                 int source, int recvtag,
      //                 MPI_Comm comm, MPI_Status *status)
      // 3. send to top process
      MPI_Sendrecv( top_halo, width, MPI_INT,
                    top, 0,
                    &(local_grid[local_rows-1][0]), width, MPI_INT,
                    bottom, 0,
                    MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      // send to bottom process
      MPI_Sendrecv( bottom_halo, width, MPI_INT,
                    bottom, 0,
                    &(local_grid[0][0]), width, MPI_INT,
                    top, 0,
                    MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      // 4.
      evolve_inner(local_rows, width, local_grid, temp);

      if (global_show_gui) gui_draw(local_rows, width, local_grid[0]);
    }
```

```
  /* gather results back at root */
  //MPI_Gatherv(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
  //            void *recvbuf, const int *recvcounts, const int *displs,
  //            MPI_Datatype recvtype, int root, MPI_Comm comm)
  MPI_Gatherv( &(local_grid[1][0]), sendcounts[rank], MPI_INT,
               &(grid[1][0]), sendcounts, displs,
               MPI_INT, 0, MPI_COMM_WORLD );

  free(local_grid);
  free(temp);
  free(sendcounts);
  free(displs);
  free(top_halo);
  free(bottom_halo);
}
```

```c
/*Convenience functions*/

void prepare_halo_rows(int height, int width,
int grid[height][width], int top[width],
int bottom[width])
{
  // copy inner part
  for( int j = 1; j < width - 1; j++)
    {
      // first row
      top[j] = grid[1][j];
      // last row
      bottom[j] = grid[height-2][j];
    }
  // handle edge values:
  // top
  top[0] = grid[1][width-2];
  top[width-1] = grid[1][1];

  //bottom
  bottom[0] = grid[height-2][width-2];
  bottom[width-1] = grid[height-2][1];
}
```

```c
void make_local_rows_periodic(int height,
 ^^Iint width, int grid[height][width])
{
  /*
    Make rows at each process periodic.
  */
  for (int i = 1; i < height - 1; i++)
    {
      // first column same as second last
      grid[i][0] = grid[i][width-2];
      // last column same as second
      grid[i][width-1] = grid[i][1];
    }
}
```