02635 Fall 2018 — Module 9 (solutions)

Exercises

1. Using the OpenMP wallclock timer:

```
#include <omp.h>

double t1, t2;
double tomp1;
...
t1 = omp_get_wtime();
for (i = 0; i < N; i++)
    omp_dgemv_v1(m, n, 1.0, A, x, 0.0, y);
t2 = omp_get_wtime();
tomp1 = (t2 - t1) / N;
...
```

`omp_get_wtime()` returns the time in seconds! If you want to have the time in ms, like last week, you have to multiply the result by 1000.

Another solution is, to use a similar macro, as shown in last week's solution:

```
#ifdef _OPENMP
#include <omp.h>
#define mytimer omp_get_wtime
#define delta_t(a,b) (1e3 * ((b)-(a)))
#else
#include <time.h>
#define mytimer clock
#define delta_t(a,b) (1e3 * ((b) - (a)) / CLOCKS_PER_SEC)
#endif
```

and somewhere inside `main()`:

```
#ifdef _OPENMP
  double t1, t2;
  fprintf(stderr, "OpenMP version: timing wallclock time (in ms)! ");
#else
  clock_t t1, t1;
  fprintf(stderr, "Serial version: timing CPU time (in ms)!\n");
#endif
```

Note: `omp_get_wtime()` returns the result in `double`, while `clock()` returns the result as `clock_t`!

2. Using `init_data()` and `check_results()`:

```
#include "datatools.h"
...
/* Allocate memory */
A = malloc_2d(m, n);
x = malloc(n * sizeof(*x));
y = malloc(m * sizeof(*y));
r = malloc(m * sizeof(*y));
if (A == NULL || x == NULL | y == NULL | r == NULL) {
    fprintf(stderr, "Memory allocation error...\n");
    exit(EXIT_FAILURE);
}

/* initialize with useful data - last argument is reference */
init_data(m,n,y,A,x,r);

...

/* check the results - bail out if an error is encountered */
if (check_results("row", m, n, y, r) > 0) exit(EXIT_FAILURE);
```

3. Implement the function `omp_dgemv_v1`:

```
void omp_dgemv_v1(
  int m,          /* number of rows                   */
  int n,          /* number of columns                */
  double alpha,   /* scalar                           */
  double ** A,    /* two-dim. array A of size m-by-n  */
  double * x,     /* one-dim. array x of length n     */
  double beta,    /* scalar                           */
  double * y      /* one-dim. array x of length m     */
) {
  int i,j;

  #pragma omp parallel for private(i,j)
  for (i=0;i<m;i++) {
    y[i] *= beta;
    for (j=0;j<n;j++) {
      y[i] += alpha*A[i][j]*x[j];
    }
  }
  return;
}
```

4. Implement the function `omp_dgemv_v2`:

```
void omp_dgemv_v2(
  int m,          /* number of rows                   */
  int n,          /* number of columns                */
  double alpha,   /* scalar                           */
  double ** A,    /* two-dim. array A of size m-by-n  */
  double * x,     /* one-dim. array x of length n     */
  double beta,    /* scalar                           */
  double * y      /* one-dim. array x of length m     */
) {
  int i,j;
  #pragma omp parallel for
  for (i=0;i<m;i++) y[i] *= beta;

  for (j=0;j<n;j++) {
    #pragma omp parallel for
    for (i=0;i<m;i++) {
      y[i] += alpha*A[i][j]*x[j];
    }
  }
  return;
}
```

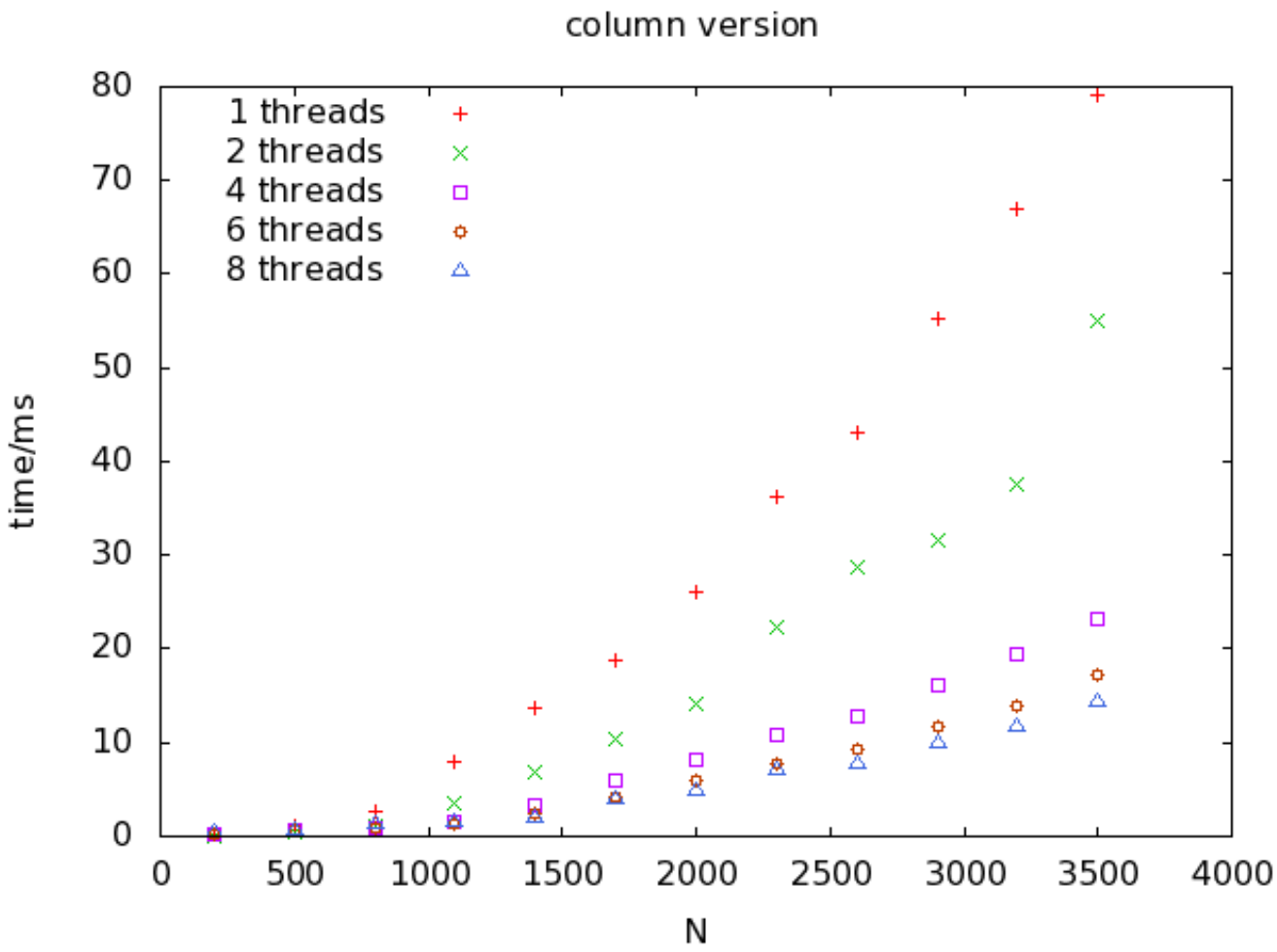
However, this can be slow, since the OpenMP runtime system will enter the parallel region for every iteration of `j`!

5. Improved version of the function `omp_dgemv_v2` :

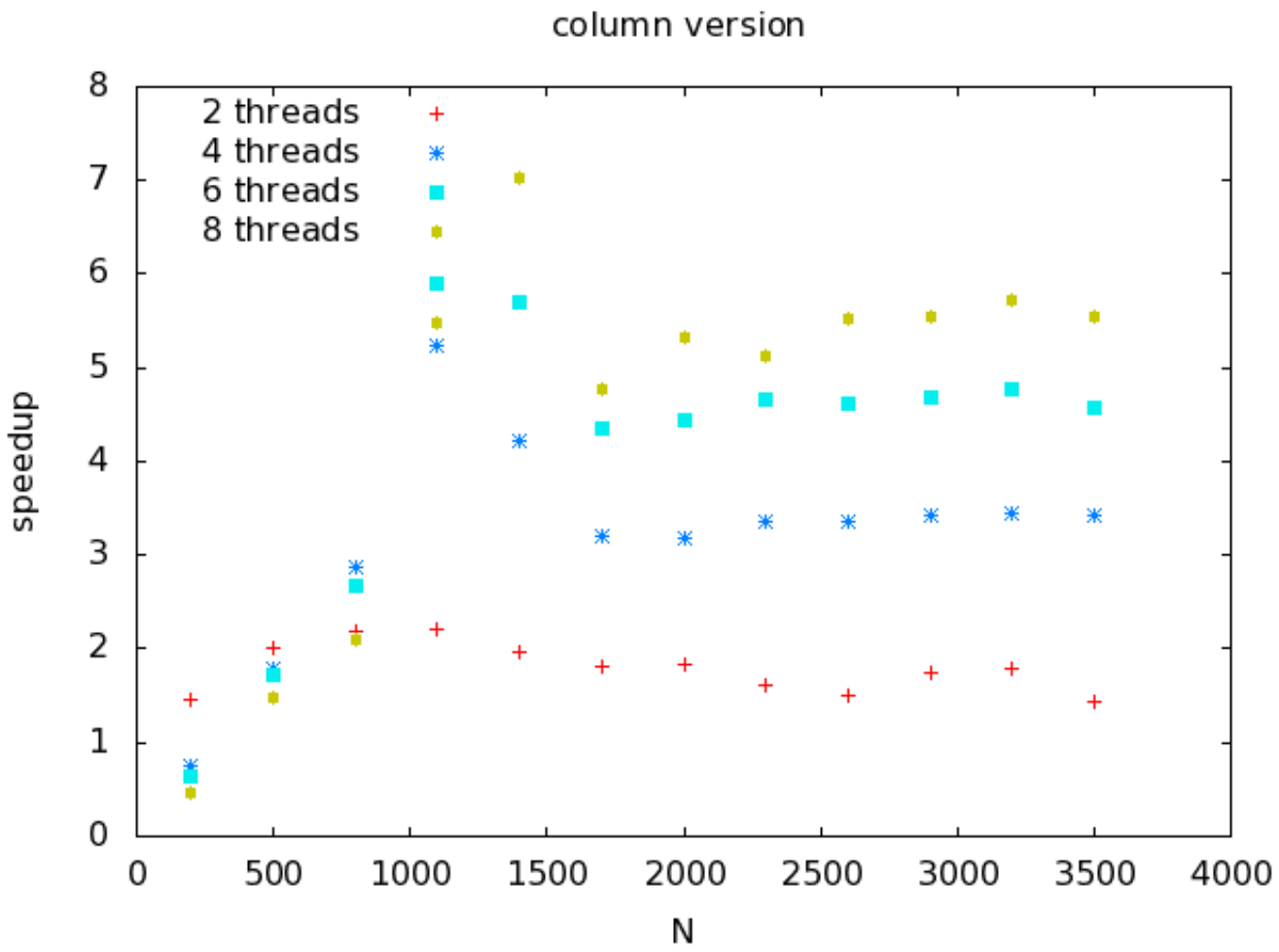
```
void omp_dgemv_v2(
  int m,          /* number of rows                   */
  int n,          /* number of columns                */
  double alpha,   /* scalar                           */
  double ** A,    /* two-dim. array A of size m-by-n  */
  double * x,     /* one-dim. array x of length n     */
  double beta,    /* scalar                           */
  double * y      /* one-dim. array x of length m     */
) {
  int i,j;
  #pragma omp parallel private(i,j)
  {
  #pragma omp for
  for (i=0;i<m;i++) y[i] *= beta;

  for (j=0;j<n;j++) {
    #pragma omp for
    for (i=0;i<m;i++) {
      y[i] += alpha*A[i][j]*x[j];
    }
  }
  } // end parallel
  return;
}
```

This version will perform better, since the parallel region is entered once, only!

Timings for the column-wise version (same system as above):



Speed-up values for the column-wise version:



The speed-up values here are not as good as for the row-wise version above. This is a consequence of the already bad memory access in the serial version (see last week's exercises).