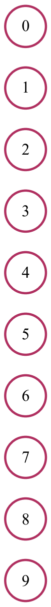
Specification of the applications

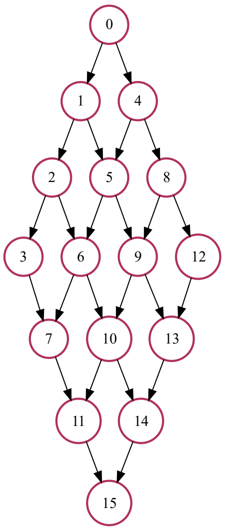
```
double *x = (double*)malloc(sizeof(double) * N);
double *y = (double*)malloc(sizeof(double) * N);
for(int i = 0; i < N; ++i)
  x[i] = y[i] = 1;
#pragma omp parallel
  #pragma omp single
  {
    for (int i=0; i<num_iter; i++)
      #ifdef TDG
      #pragma omp taskgraph tdg_type(static)
      #endif
      {
        saxpy(x, y);
      }
  }

void saxpy(double *x, double *y) {
  for (int i = 0; i < N; i+=BS) {
    #ifdef PREALLOC
    #pragma omp task dep_check(dynamic)
    #else
    #pragma omp task
    #endif
    {
      for(int j = 0; j < BS; j++)
        y[i+j] += a * x[i+j];
    }
  }
  #pragma omp taskwait
}
```



(a) (b)

Figure 1. The *Axpy* application: (a) OpenMP-based program; (b) graphical representation of the DAG, where the number of tasks (i.e., blocks) is 10 and the block size (BS) is N/9. Note that there is no data dependency in the TDG.

```
int bx,by; bx = by = NP/NB;
#pragma omp parallel
  #pragma omp single
    #ifdef TDG
    #pragma omp taskgraph tdg_type(static)
    #endif
    for (int i=0; i<num_iter; i++) {
      for (int ii=0; ii<NB; ii++) {
        for (int jj=0; jj<NB; jj++) {
          int inf_i = 1 + ii * bx;
          int sup_i = ((inf_i + bx) < sizex - 1) ? inf_i + bx : sizex - 1;
          int inf_j = 1 + jj * by;
          int sup_j = ((inf_j + by) < sizey - 1) ? inf_j + by : sizey - 1;
          #pragma omp task depend(in: u[inf_i-bx][inf_j], u[sup_i][inf_j], \
                                      u[inf_i][inf_j-by], u[inf_i][sup_j]) \
                                      depend(inout: u[inf_i][inf_j]) \
                                      firstprivate(sizex, sizey, u)
          {
            for (int i = inf_i; i < sup_i; ++i)
              for (int j = inf_j; j < sup_j; ++j)
                  u[i][j] = 0.25 * (u[i][j-1] + u[i][j+1] + u[i-1][j] +
                  u[i+1][j]);
          }
        }
      }
    }
```

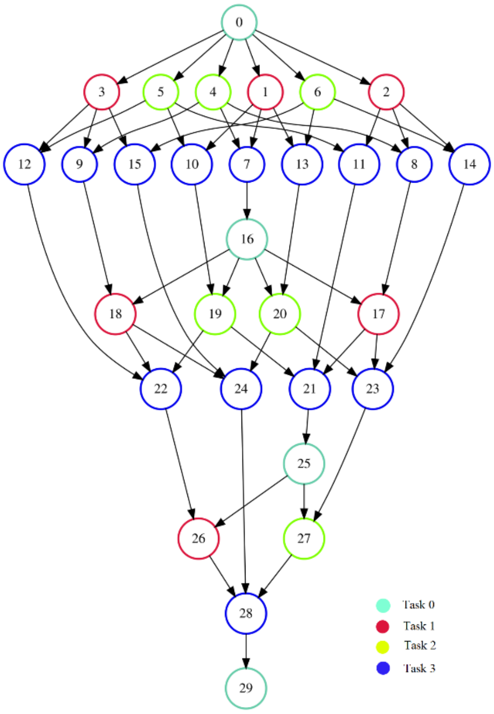


(a) (b)

Figure 2. The *Heat* application: (a) OpenMP-based code block to create the TDG structure; (b) instance of the DAG, where NB is 4. The number of blocks is NB*NB and the resolution (i.e., the number of points) is NP*NP. The *omp taskgraph* clause generates the TDG and the code is divided into NB*NB explicit tasks, where each task includes a block of bx*by iterations. The relationship between the tasks indicates their data dependency, so each task can be executed if all its input dependencies have been already met.

```
1  #pragma omp parallel
2    #pragma omp single
3    {
4      for (int iter = 0; iter < num_iter; iter ++)
5        #ifdef TDG
6        #pragma omp taskgraph tdg_type(static)
7        #endif
8        {
9          s = S;
10         for (kk=0; kk<S; kk++) {
11           // Task 0
12           #pragma omp task firstprivate(kk) shared(M) \
13                            depend(inout: M[kk*s+kk])
14           ...
15
16           for (jj=kk+1; jj<S; jj++) {
17             // Task 1
18             #pragma omp task firstprivate(kk, jj) shared(M) \
19                              depend(in: M[kk*s+kk]) \
20                              depend(inout: M[kk*s+jj])
21             ...
22           }
23           for (ii=kk+1; ii<S; ii++) {
24             // Task 2
25             #pragma omp task firstprivate(kk, ii) shared(M) \
26                              depend(in: M[kk*s+kk]) \
27                              depend(inout: M[ii*s+kk])
28             ...
29           }
30           for (ii=kk+1; ii<S; ii++)
31             for (jj=kk+1; jj<S; jj++) {
32               // Task 3
33               #pragma omp task firstprivate(kk, jj, ii) shared(M) \
34                                depend(in: M[ii*s+kk], M[kk*s+jj]) \
35                                depend(inout: M[ii*s+jj])
36               ...
37             }
38         }
39       }
40   }
```

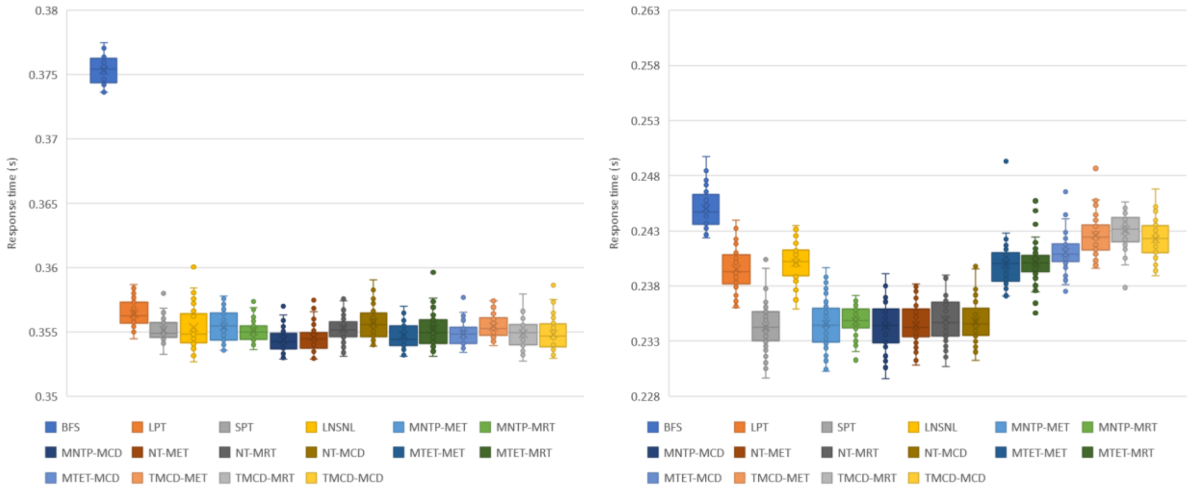
(a)



(b)

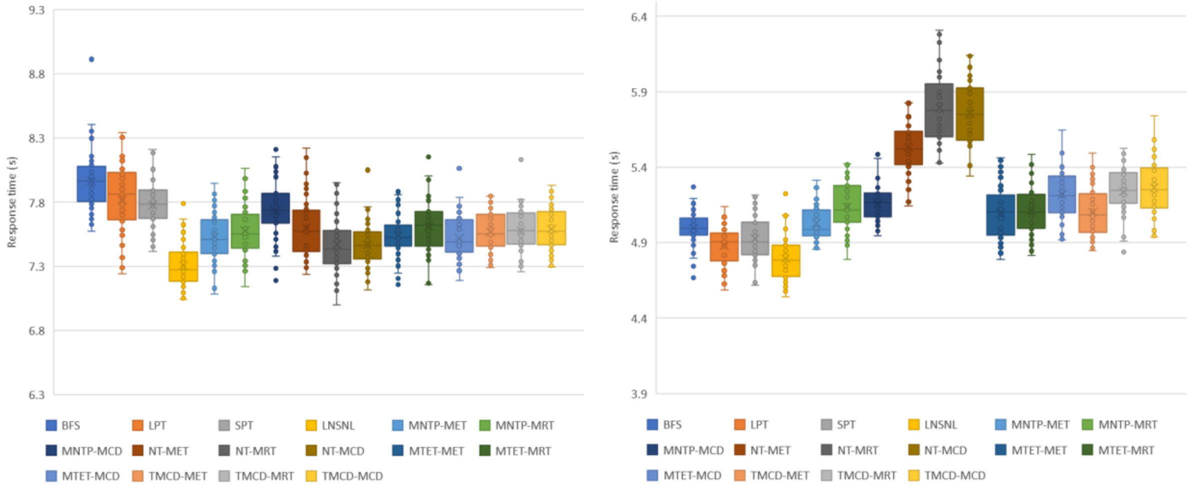
Figure 3. The *SparseLU* application: (a) main code block of the OpenMP-based program to generate the TDG; (b) example of the DAG, where the matrix size (S) is 4 and the block size (BS) is 16. The TDG is generated using four explicit tasks through a nested structure, creating an irregular form of the parallel tree. The number of Task 0 placed in lines 12 and 13 is S, the number of Task 1 placed in lines 18-20 and the number of Task 2 placed in lines 25-27 is $\frac{S(S-1)}{2}$, as well as the number of Task 3 placed in lines 33-35, which are created through two nested loops, is greater than the previous tasks.

Experiments



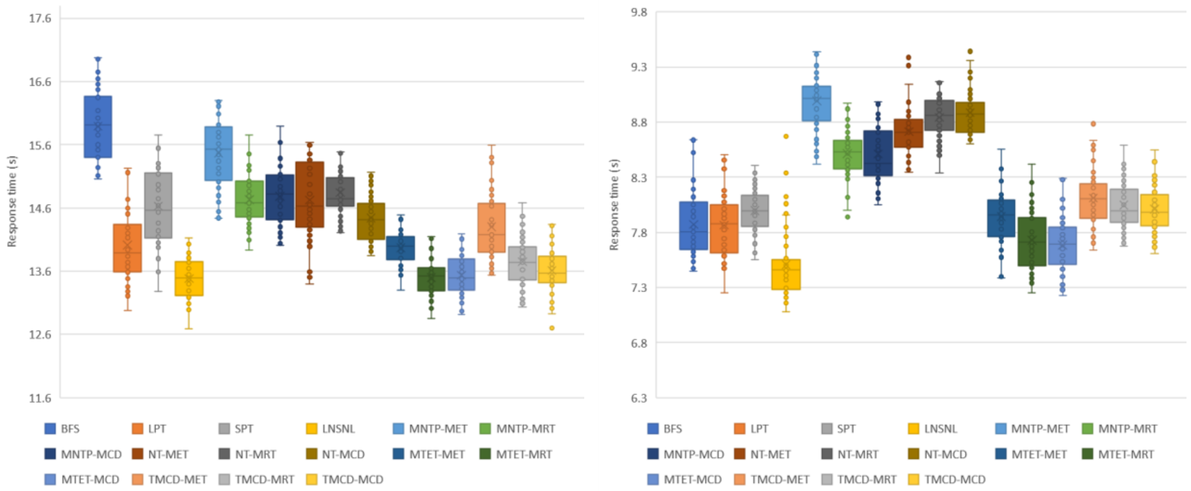
(a) (b)

Figure 4. Experimental results of the *Axpy* application: (a) 4 threads; (b) 8 threads.



(a) (b)

Figure 5. Experimental results of the *Heat* application: (a) 4 threads; (b) 8 threads.



(a) (b)

Figure 6. Experimental results of the *SparseLU* application: (a) 4 threads; (b) 8 threads.