Threaded task assignment in CPLEX

Roberto Meroni* Hugo Sanz González†

November 2023

Consider the problem of distributing the workload required to perform a set of tasks T among a set of computers C. The workload of each task $t \in T$ is divided into a set of threads H_t, and each $h \in H_t$ requires $r_h \geq 0$ compute units to complete. Each computer $c \in C$ provides a set of cores K_c, all of which have $r_c \geq 0$ compute units available. Consider a mixed integer linear program P_3 where a thread is assigned to exactly one core, and all threads of a task are assigned to cores of a single computer. If the load of a computer $c \in C$ refers to the sum of the loads of the cores in K_c divided by its total capacity $|K_c| \cdot r_c$, the objective in P_3 is to minimize the load of the highest-loaded computer.

1. *Implement the P_3 model in OPL and solve it using CPLEX.*

Let H be the set of all threads among all tasks in T, and let K be the set of all cores among all computers in C. For each thread $h \in H$ and each core $k \in K$, define a binary variable x_{hk} denoting whether h is assigned to k. Also let x_{tc} be a binary variable denoting whether the threads of a task $t \in T$ are assigned to the cores of a computer $c \in C$. The corresponding variable definitions in the OPL language look as follows:

```
range H = 1..nThreads;
range C = 1..nCPUs;
float rh[h in H] = ...;
float rc[c in C] = ...;
```

All threads must be assigned to exactly one core, hence the first set of constraints is $\sum_{k \in K} x_{hk} = 1$ for all $h \in H$. In OPL this can be expressed as

```
forall(h in H)
  sum(k in K) x_hk[h, k] == 1;
```

where the sets `H` and `K` are declared as

```
range H = 1..nThreads;
range K = 1..nCores;
```

In addition, all the threads belonging to a task must be assigned to cores belonging to the same computer. If the threads of a task $t \in T$ are assigned to some subset of cores of a computer $c \in C$, then $|H_t|x_{tc}$ is equal to the number of threads of t; otherwise x_{tc} is zero and so this last quantity is also

*roberto.meroni@estudiantat.upc.edu
†hugo.sanz@estudiantat.upc.edu

nil. Since $\sum_{h \in H_t, k \in K_c} x_{hk}$ counts the number of threads belonging to task t that are assigned to cores in c, the constraints

$$\sum_{h \in H_t} \sum_{k \in K_c} x_{hk} = |H_t| x_{tc}, \qquad \forall t \in T, c \in C$$

ensure that the threads of a task are all assigned to cores of a single computer. In OPL,

```
forall(c in C, t in T)
  sum(h in H, k in K) x_hk[h, k] * CK[c, k] * TH[t, h]
    == x_tc[t, c] * sum(h in H) TH[t, h];
```

The entry i, j of matrix `CK` is a boolean value describing whether a core $k_j \in K$ is part of a computer $c_i \in C$. Similarly entry i, j of matrix `TH` defines whether a thread $h_j \in H$ forms part of task $t_i \in T$. We declare them in OPL as follows:

```
float CK[c in C, k in K] = ...;
float TH[t in T, h in H] = ...;
```

Another set of constraints ensures that the capacity of all cores K_c in a computer $c \in C$ is not exceeded. Note that $r_h x_{hk}$ is the load assigned to a core $k \in K_c$ corresponding to a thread $h \in H$, so

$$\sum_{h \in H} r_h x_{hk} \leq r_c, \quad \forall c \in C, k \in K_c.$$

In OPL this is

```
forall (c in C, k in K)
  sum(h in H) rh[h] * x_hk[h,k] * CK[c,k] <= rc[c];
```

The total number of compute units handled by all cores of a computer $c \in C$ is $\sum_{h \in H} \sum_{k \in K_c} r_h x_{hk}$, and the capacity of c is its number of cores $|K_c|$ multiplied by the capacity r_c of each one of them. Introduce a real decision variable z representing the load of the highest-loaded computer. This is precisely the nonnegative quantity to be minimized, and it is bounded below by the load of each computer. That is,

$$z \geq \frac{1}{|K_c| r_c} \sum_{h \in H} \sum_{k \in K_c} r_h x_{hk}, \quad \forall c \in C. \tag{1}$$

In OPL, this set of constraints may be written as

```
forall(c in C)
  z >= (sum(h in H, k in K) rh[h] * x_hk[h,k] * CK[c, k])
       / (rc[c] * sum(k in K) CK[c, k]);
```

At this point problem P_3 has been fully specified, and we may solve it for the particular instance specified in the provided data file. We have computers c_1, c_2, c_3 with cores k_{i1}, k_{i2}, k_{i3} for each $c_i \in C$, and tasks $t_1, \ldots, t_4$ with threads h_{i1}, h_{i2} for each $t_i \in T$. The CPLEX solver finds the assignment

$$\begin{array}{llll} h_{11} \to k_{12} & h_{21} \to k_{11} & h_{31} \to k_{31} & h_{41} \to k_{32} \\ h_{12} \to k_{22} & h_{22} \to k_{21} & h_{32} \to k_{11} & h_{42} \to k_{11} \end{array}$$

where $h_{ij} \to k_{pq}$ means thread h_{ij} of task $t_i \in T$ is assigned to core k_{pq} of computer $c_p \in C$.

2. *Generate instances of increasing size and use the P_3 model to solve them.*

   We modify the provided `InstanceGeneratorP3/config.dat` file in order to produce five instances having the following characteristics:

		Threads			Cores	
#	Tasks	Per task	Total	Computers	Per computer	Total
1	10	$[10, 15]$	128	5	$[100, 150]$	585
2	10	$[10, 20]$	152	10	$[400, 700]$	5383
3	10	$[5, 25]$	126	5	$[500, 1000]$	3789
4	10	$[5, 25]$	115	5	$[750, 1500]$	4801
5	12	$[5, 10]$	87	5	$[50, 100]$	362

   The number of threads per task and the number of cores per computer are random integers drawn from the corresponding intervals given in the table. The capacity per core is 5000 and the compute units required by each thread follow a uniform distribution in $[500, 800]$. Details regarding the problem size and the solutions found by CPLEX appear in Exercise 4.

3. *Modify the P_3 model to maximize the number of computers with all their cores empty.*

   Introduce a binary decision variable y_c for each computer $c \in C$ that denotes whether no thread has been assigned to any of the cores in K_c. If $y_c = 0$, then there exists a task $t \in T$ such that $x_{tc} = 1$. In other words, $y_c = 0$ implies that $\sum_{t \in T} x_{tc} \geq 1$, or equivalently $1 - \sum_{t \in T} x_{tc} \leq 0$. Since the number 1 is an upper bound on the left-hand side of this equation, the set of constraints that express these implications is

$$1 - \sum_{t \in T} x_{tc} \leq y_c, \quad \forall c \in C.$$

   In OPL this is expressed as follows:

```
forall(c in C)
  1 - sum(t in T) x_tc[t, c] <= y_c[c];
```

   where the y_c's are declared as

```
dvar boolean y_c[c in C];
```

   On the other hand, if $y_c = 1$ then $x_{tc} = 0$ for all $t \in T$. Since $|T|$ is an upper bound on $\sum_{t \in T} x_{tc}$, we want

$$\sum_{t \in T} x_{tc} \leq |T|(1 - y_c), \quad \forall c \in C.$$

   Analogously we may write this in OPL as follows:

```
forall(c in C)
  sum(t in T) x_tc[t, c] <= nTasks * (1 - y_c[c]);
```

#	Model	Variables	Constraints	Iterations	Runtime (s)
1	P_3	74 931	3108	429 390	28
	P_3'	74 935	3113	462	3
2	P_3	818 317	54 092	66 339	152
	P_3'	818 326	54 102	663	47
3	P_3	477 465	19 126	881 293	828
	P_3'	477 469	19 131	424	23
4	P_3	552 166	24 175	816 669	711
	P_3'	552 170	24 180	419	26
5	P_3	31 555	1962	1 203 320	44
	P_3'	31 559	1967	335	1

Table 1: The statistics of the instances generated in exercise 2 under the P_3 and P_3' models, as well as the solution time achieved by CPLEX.

Our goal is to maximize the number of computers with no work assigned, so we remove the constraint in Eq. (1) and replace the objective in P_3 by $\max \sum_{c \in C} y_c$ to obtain the new model P_3'.

4. *Compare models P_3 and P_3' in terms of number of variables, number of constraints and execution time for the generated instances.*

The problem size and results found by CPLEX after solving the instances generated in Exercise 2 under the two models appear in Table 1; all problems are feasible. In P_3, the appearance of a continuous decision variable makes the problem a mixed-integer linear program and this increases the time needed by CPLEX to find an optimal solution. In contrast, P_3' is a binary linear program which is more amenable to branch & cut techniques. For example, for instances 1, 3 and 5 the time decreased respectively by a factor of 9.33, 36 and 44.

It is interesting to compare the solver's behavior for instances 1 and 5; even if the former has roughly twice as many variables and constraints, the latter takes ≈ 1.57 times longer to solve under the P_3 model. This suggests that the total number of variables and constraints are not the only factor affecting the solving time; the shape of the problem also has a profound effect. (Here instance 1 has 2 less tasks to assign than instance 5.)

We could not produce instances that lead to solving times greater than 30 minutes due to memory limitations. Still, the results indicate that minimizing the load of the highest-loaded computer is a much harder problem to solve than maximizing the number of computers with no assigned tasks.