Un altro modello

Variabili:

$$x_j \in \{0 \ldots n_p\}$$

Vincoli:

- La deadline deve essere rispettata:

$$\forall i \quad \sum_{j=0}^{n-1} dur(t_j) \cdot (x_j = i) \leq deadline$$

METAVINCOLI:
Un vincolo può essere utilizzato all'interno di una espressione: denota 1 se è vero, 0 se è falso

- Obiettivo: minimizzare il numero di processori utilizzati

$$\min z = \max_j (x_j)$$

Il più alto indice di processore utilizzato

Un altro modello

Variabili: $x_j \in \{0 ... n_p - 1\}$

```
IloIntVarArray Proc(env, ntask, 0, nproc-1);
```

Vincoli di deadline: $\forall i \quad \sum_{j=0}^{n-1} dur(t_j) \cdot (x_j = i) \leq deadline$

```
for(int i = 0; i < ntask; i++){
    IloIntExpr sum(env);
    for(int j = 0; j < ntask; j++){
      sum += durations[j]*(Proc[j] == i);
    }
    model.add(sum <= deadline);
}
```

Un altro modello

Cambia anche la funzione obiettivo:

$$\boxed{\min z = \max_j (x_j)}$$

```
model.add(IloMinimize(env, IloMax(Proc)));
```

Il nuovo modello ha la stessa semantica (=> le stesse soluzioni), ma un numero minore di variabili e una diversa funzione obiettivo

- Il codice del nuovo modello è in “base2.cpp”
- Scaricate, provate a compilare e ad eseguire

Output

```
Number of fails                    : 425
Number of choice points            : 432
...
Running time since creation        : 0.03

Proc[0]:0
Proc[1]:0
Proc[2]:0
Proc[3]:0
Proc[4]:1
Proc[5]:2
Proc[6]:3
Proc[7]:1
Proc[8]:2
Proc[9]:3
```

Esercizio 1

- Cosa succede con i due modelli aumentando il numero di processori a 16?
- **Perché?**

Esercizio 2

- Supponiamo che la durata dei task sia dur(t_j) sui processori da 0 a 2 (escluso), e dur(t_j)/2 su quelli di indice da 2 a p-1
- Modificare il modello di conseguenza

Vincoli globali

Utilizzo dei vincoli globali

Perché i vincoli globali

I vincoli globali modellano alcune sottostrutture particolarmente frequenti. Hanno diversi vantaggi:

- modellazione più compatta
- propagazione più **efficace**
- (a volte) propagazione più **efficiente**

Esempio:
Gli ultimi p task devono essere assegnati a processori diversi

Modello con vincoli binari

- Modellando con vincoli "!=":

```
for (int i = ntask-nproc; i < ntask; i++){
    for (int j = i+1; j < ntask; j++){
      model.add(Proc[i] != Proc[j]);
    }
}
```

- Output:

```
Number of fails                : 44496
Number of choice points        : 44505
...
Running time since creation    : 1.5
```

Modello con alldifferent

- Alldifferent in ILOG

```
IloAllDiff(IloEnv, IloIntVarArray)
```

- Nel nostro caso:

```
IloIntVarArray diff(env);
  for(int j = ntask-nproc; j < ntask; j++)
diff.add(Proc[j]);

model.add(IloAllDiff(env, diff));
```

- **ATTENZIONE!** Dopo aver estratto il modello:

```
solver.setDefaultFilterLevel(IloAllDiffCt, IloExtendedLevel);
```

Modello con alldifferent

- Alldifferent in ILOG

```
IloAllDiff(IloEnv, IloIntVarArray)
```

- Nel nostro c

```
IloIntVarAr
  for(int j                k; j+
diff.add(Pr

model.add(I
```

```
IloAllDiffCt
IloDistributeCt
IloSequenceCt
IloAllMinDistanceCt
IloPartitionCt
IloAllNullIntersectCt
IloEqUnionCt
IloNotOverlapCt
IloBoxCt
IlcFilterLevelConstraint
```

```
IloExtendedLevel
IloMediumLevel
IloBasicLevel
IloLowLevel
IlcFilterLevel
```

- **ATTENZIONE!** Dopo aver estratto il modello:

```
solver.setDefaultFilterLevel(IloAllDiffCt, IloExtendedLevel);
```

Output

```
Number of fails                : 8
Number of choice points        : 16
...
Running time since creation    : 0

problem solved
Proc[0]:0
Proc[1]:0
Proc[2]:0
Proc[3]:1
Proc[4]:0
Proc[5]:1
Proc[6]:2
Proc[7]:3
Proc[8]:4
Proc[9]:5
```

IMPORTANTE:
ILOG Solver permette anche di definire nuovi vincoli, ma per il momento non ci interessa...

Esercizio 3

- Nuovo vincolo: non più di tre task per processore
- **SUGGERIMENTO:** il vincolo gcc in ILOG è:

```
IloDistribute(IloEnv, IloIntVarArray cards,
              IloIntArray vals, IloIntVarArray vars)
```

Strategie di ricerca

Modificare la strategia di ricerca

Search strategy

CP permette l'impiego di diverse strategie di ricerca

- Per esempio si può scegliere il criterio con cui scegliere la variabile su cui fare branching

```
IloGenerate(IloEnv, IloIntVarArray, IloChooseIntIndex)
```

E' un `IloGoal`, va passato come argomento di `solver.solve(...)`

```
IloChooseFirstUnboundInt
IloChooseMaxMaxInt
IloChooseMaxMinInt
IloChooseMaxRegretMax
IloChooseMaxRegretMin
IloChooseMaxSizeInt
IloChooseMinMaxInt
IloChooseMinMinInt
IloChooseMinRegretMax
IloChooseMinRegretMin
IloChooseMinSizeInt
```

First Fail Principle

Output

```
solver.solve(IloGenerate(env, Proc, IloChooseMinSizeInt))
```

```
Number of fails                 : 71
Number of choice points         : 77
...
Running time since creation     : 0.01
```

IMPORTANTE:
ILOG Solver permette anche di definire nuove strategie o di intervenire sulla ricerca in modo ancora più complesso, ma per il momento non ci interessa...

Esercizio 4

- `Cosa succede con altre strategie di ricerca?`
- `Cosa succede utilizzando il first fail principle (IloChooseMinSizeInt) nel primo modello? Perché?`

Scheduling con ILOG

Introduzione ad ILOG Scheduler

Problema

Siano:

- $T = \{t_0, t_1, ..., t_{n-1}\}$ = insieme degli **n** tasks
- $P = \{p_0, p_1, ..., p_{p-1}\}$ = insieme dei **p** processori
- $dur(t_i)$ = durata del task i-mo

> Ogni task esegue su un solo processore
> Su ogni processore i task eseguono in sequenza
> Il tempo totale di esecuzione non deve superare una **deadline**

Un nuovo elemento:

> **Tra alcune coppie di task sono definite delle relazioni di precedenza:**
> $\mathbf{t_i < t_j \iff end(t_i) \leq start(t_j)}$

Esercizio

- Come modellare il problema?
- **SUGGERIMENTO:** usare le seguenti variabili
  - Assegnamento: $x_j \in \{0 ... n_p - 1\}$
  - Start times: $s_j \in \{0 ... dl\}$