ELEN 503
Hardware-Software Co-Design

Homework #3
HW-SW Mapping/Scheduling Optimization using Gurobi

Prof. Hoeseok Yang
Santa Clara University

Spring Quarter 2023

Homework #3 - Overview

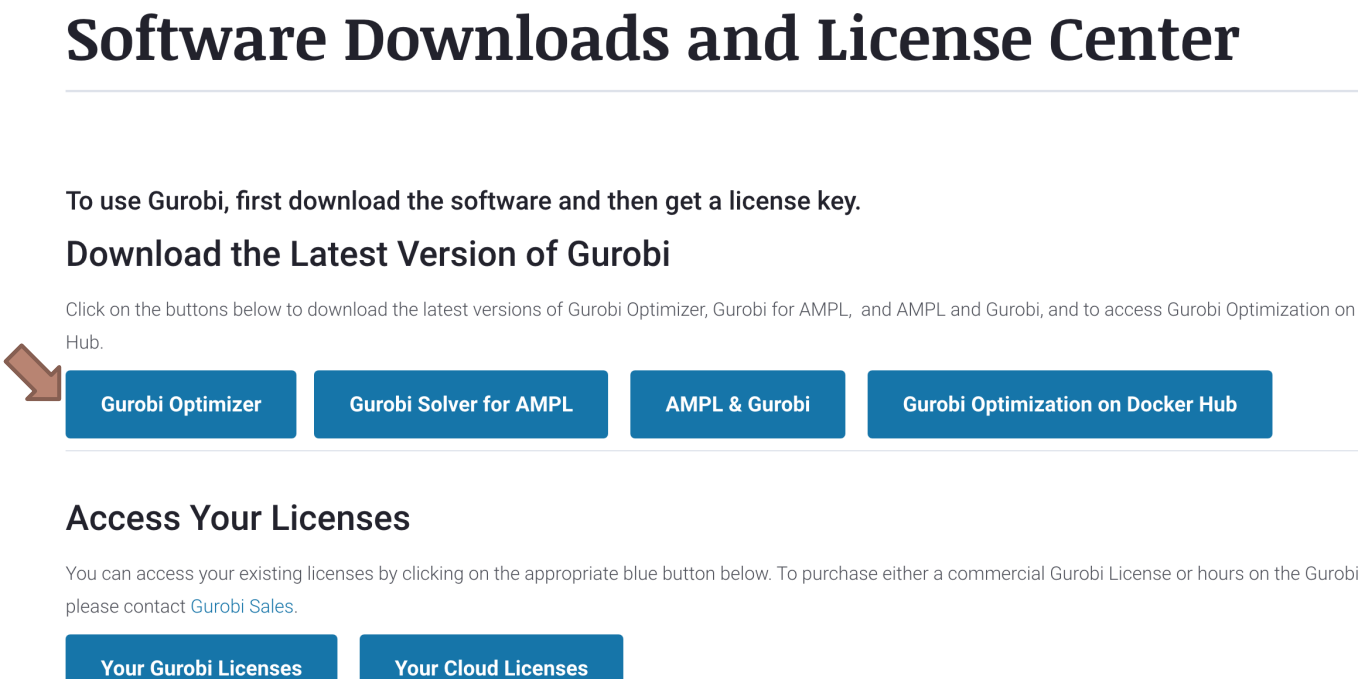
- In this homework, you will practice a combinatorial optimization using Gurobi

- Tasks – 10 points
  - Part 0 (set up and warming up): 1 point
    - Set up Gurobi in your local computer
    - Test the product planning example
  - Part 1 (Completing mapping optimization): 3 point
    - Complete the *sh* and *ah* constraints
  - Part 2 (Extending the basic formulation): 2 points
    - Different objective
    - More optimization with additional task graphs
  - Part 3 (Adding communication delay): 2 points
    - Extend the basic formulation to consider communication delay into account
  - Report: 2 points

- Submission
  - Upload your report (in pdf) and source codes (in zip) to Camino
  - Due: June 17th, 11:59pm

Part 0: Install Gurobi

- Download and Install Gruboi
  - https://www.gurobi.com/downloads/



Part 0: Obtain free academic license

- First, create your account (register) here
  - https://www.gurobi.com/downloads/end-user-license-agreement-academic/
- Log in with your account and "Request a free academic license" → Individual Academic License
- You will be given your own license key
- After installing the gurobi optimizer, you must setup the license file on your computer (the one you installed the tool)
  - grbgetkey [YOUR_ASSIGNED_KEY]

Part 0: Language Interfaces

- In this homework, I used Python for skeleton codes

- But, Gurobi is available in many programming languages other than Python,
  - such as C, C++, Java, .NET, MATLAB, and R
  - They have comprehensive documentations and code examples online
    - https://www.gurobi.com/documentation/

- If you prefer, you may choose your own programming interface

Part 0: A simple product planning example

	Desk	Table
Selling Price	$15	$20

	Desk	Table	Availability
Lumber	1	2	20
Carpentry	2	1	20

- Variables (integer)
  - x1: number of desks (≥ 0)
  - x2: number of tables (≥ 0)
- Constraints
  - Amount of Lumber = 1*x1 + 2*x2 ≤ 20
  - Amount of Carpentry = 2*x1 + 1*x2 ≤ 20
- Objective: maximize revenue = 15*x1 + 20*x2

Part 0: A simple product planning example

- Type in
  - $ python3 product.py

```python
import gurobipy as gp
from gurobipy import GRB

try:

    # Create a new model
    m = gp.Model("prod")

    # Create variables
    x1 = m.addVar(vtype=GRB.INTEGER, name="x1")
    x2 = m.addVar(vtype=GRB.INTEGER, name="x2")

    # Set objective
    m.setObjective(15 * x1 + 20 * x2, GRB.MAXIMIZE)

    # Add constraint c2: x1 + 2 * x2 <= 20
    m.addConstr(x1  + 2 * x2 <= 20, "c2")

    # Add constraint c2: 2* x1 + x2 <= 20
    m.addConstr(2 * x1  +  x2 <= 20, "c3")

    # Add constraint x1>=0, x2>=0
    m.addConstr(x1 >= 0, "c4_1")
    m.addConstr(x2 >= 0, "c4_2")

    # Optimize model
    m.optimize()

    for v in m.getVars():
        print('%s %g' % (v.varName, v.x))
        print('%s %g' % (v.varName, v.x))

    print('Obj: %g' % m.objVal)
```

```
hyang8@ENGR-L-01613 product % python3 product.py
Set parameter Username
Academic license - for non-commercial use only - expires 2023-05-18
Gurobi Optimizer version 9.5.1 build v9.5.1rc2 (mac64[arm])
Thread count: 8 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 4 rows, 2 columns and 6 nonzeros
Model fingerprint: 0xbb9dddab
Variable types: 0 continuous, 2 integer (0 binary)
Coefficient statistics:
  Matrix range     [1e+00, 2e+00]
  Objective range  [2e+01, 2e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+01, 2e+01]
Found heuristic solution: objective 150.0000000
Presolve removed 2 rows and 0 columns
Presolve time: 0.00s
Presolved: 2 rows, 2 columns, 4 nonzeros
Variable types: 0 continuous, 2 integer (0 binary)
Found heuristic solution: objective 155.0000000

Root relaxation: objective 2.333333e+02, 2 iterations, 0.00 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0  233.33333    0    2  155.00000  233.33333  50.5%     -    0s
H    0     0                     230.0000000  233.33333  1.45%     -    0s
     0     0  233.33333    0    2  230.00000  233.33333  1.45%     -    0s

Explored 1 nodes (2 simplex iterations) in 0.00 seconds (0.00 work units)
Thread count was 8 (of 8 available processors)

Solution count 3: 230 155 150

Optimal solution found (tolerance 1.00e-04)
Best objective 2.300000000000e+02, best bound 2.300000000000e+02, gap 0.0000%
x1 6
x1 6
x2 7
x2 7
Obj: 230
hyang8@ENGR-L-01613 product %
```

Part 1: ILP-based mapping optimization

- Input
  - A Homogeneous SDF (task graph) <*V*, *E*>
    - an SDF in which all producing/consuming rates are 1
    - Set of nodes (tasks): *V*
    - Set of directed edges: *E*
      - $e_{i,j} \in E$ denotes data dependency from v_i to v_j
  - Set of processors *P*
    - Each $p_i \in P$ costs $cost_i$
  - Execution time table
    - $exec_{i,j}$: execution time of v_i running on p_j
  - Latency constraint: *L*

- Output
  - Find out a mapping that minimizes the total cost

Part 1: Given skeleton code – map_skel.py

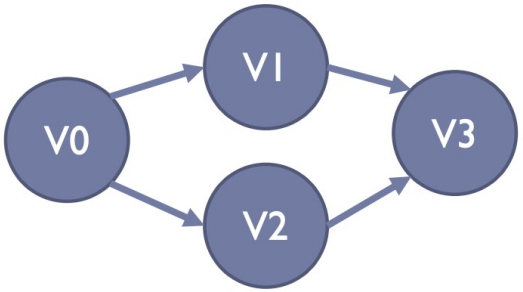
- Input task graph (Homogeneous SDF) and processors

```
# num tasks: 4
num_tasks = 4
# num procs: 4
num_procs = 4
# latency constraint
latency_const = 5

# exec. info
exec_time = [
[1,2,3,4],
[2,1,3,4],
[2,3,1,4],
[2,3,3,1],
]

# edge info
edge = [
[0,1,1,0],
[0,0,0,1],
[0,0,0,1],
[0,0,0,0],
]

# cost info
cost = [1, 2, 3, 4]
```



exec	p0	p1	p2	p3
v0	1	2	3	4
v1	2	1	3	4
v2	2	3	1	4
v3	2	3	3	1

	p0	p1	p2	p3
cost	1	2	3	4

Part 1: Given skeleton code – map_skel.py (cont'd)

- Binary variables for mapping, m
  - $m_{i,j} = 1 \rightarrow v_i$ is mapped on p_j
- Mapping constraints
  - For all $v_i \in V$, $\quad \Sigma_j m_{i,j} = 1$

```
# mapping variables m
m_var = [] # m variables
for i in range(num_tasks):
        temp_m_var = []
        mylhs = gp.LinExpr(0.0)
        for j in range(num_procs):
                temp_m_var.append(m.addVar(vtype=GRB.BINARY, name="m_" +str(i) + "_" + str(j))) # binary variable m_i_j
                mylhs.add(temp_m_var[j])
        m.addConstr(lhs=mylhs, sense=GRB.EQUAL, rhs=1, name="mapping_const_"+str(i))
        m_var.append(temp_m_var)
```

Part 1: Given skeleton code – map_skel.py (cont'd)

- Integer variables for scheduling (start time), *s*
  - s_i → task v_i starts its execution at time s_i

- Execution time
  - For all $v_i \in V$, $t_i = \Sigma_j m_{i,j} * exec_{i,j}$

```
# starting time variables s
s_var = [] # s variables
for i in range(num_tasks):
        s_var.append(m.addVar(vtype=GRB.INTEGER, name="s_" + str(i))) # integer variable s_i

# execution time variables t
t_var = [] # t variables
for i in range(num_tasks):
        t_var.append(m.addVar(vtype=GRB.INTEGER, name="t_" + str(i))) # integer variable t_i
        myrhs = gp.LinExpr(0.0)
        for j in range(num_procs):
                myrhs.add(exec_time[i][j]* m_var[i][j])
        m.addConstr(lhs=t_var[i], sense=GRB.EQUAL, rhs=myrhs, name="exec_time_const_"+str(i))
```

Part 1: Given skeleton code – map_skel.py (cont'd)

- Latency constraint

```
# latency constraints
for i in range(num_tasks):
        mylhs = gp.LinExpr(0.0)
        mylhs.add(s_var[i])
        mylhs.add(t_var[i])
        m.addConstr(lhs=mylhs, sense=GRB.LESS_EQUAL, rhs=latency_const, name="latency_const_"+str(i))
```

- Edge (Dependency) constraints
  - For all $e_{i,j} \in E$, $s_i + t_i \leq s_j$

```
# edge constraints
for i in range(num_tasks):
        for j in range(num_tasks):
                if edge[i][j] == 1:
                        m.addConstr(s_var[i] + t_var[i] <= s_var[j], name="edge_const_"+str(i)+"_"+str(j))
```

Part 1: Given skeleton code – map_skel.py (cont'd)

- Binary variables for processor selection
  - $used_i$ = 1 if there exists any v_j s.t. $m_{j,i}$=1
  - $used_i$ = 0 otherwise (for all $v_j \in V, m_{j,i}$=0)
    - For all $p_i \in P, v_j \in V$ $used_i \geq m_{j,i}$
- Objective: minimize the total cost
  - Minimize $\Sigma_i used_i * cost_i$

```
# binary variables used
used_vars = []
for i in range(num_procs):
        used_vars.append(m.addVar(vtype=GRB.BINARY, name="used_"+str(i)))
        myrhs = gp.LinExpr(0.0)
        for j in range(num_tasks):
                m.addConstr(lhs=used_vars[i], sense=GRB.GREATER_EQUAL, rhs=m_var[j][i], name="used_const_"+str(i)+"_"+str(j))


# objective
myobj = gp.LinExpr(0.0)
for i in range(num_procs):
        myobj.add(used_vars[i] * cost[i])
m.setObjective(myobj, GRB.MINIMIZE)
```

Part 1: Given skeleton code – map_skel.py (cont'd)

- Optimize the model and display the results

```
m.write("formulation.lp")
m.optimize()

print("============================================")
print("Optimization Results:")
m.printAttr('X')
print("============================================")
```

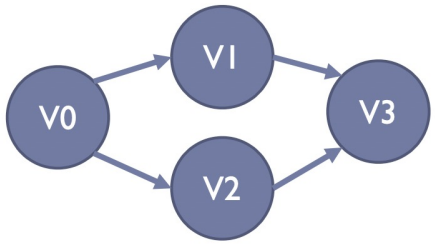
Part 1: Your task

- Complete the scheduling constraint
  - Make sure that two tasks mapped on the same processor never overlap in time
    - Using two binary variables
      - $sh_{i,j}$: 1 iff v_i and v_j are mapped on the same processor
      - $ah_{i,j}$: 1 iff v_i is scheduled earlier than v_j

- Linearized forms are given as follows
  1. $s_i + t_i \leq s_j + (1 - sh_{i,j})*\infty + (1 - ah_{i,j})*\infty$
  2. $s_j + t_j \leq s_i + (1 - sh_{i,j})*\infty + ah_{i,j}*\infty$

- Use *my_infty* for ∞

Part 1: Expected Outcome



exec	p0	p1	p2	p3
v0	1	2	3	4
v1	2	1	3	4
v2	2	3	1	4
v3	2	3	3	1

	p0	p1	p2	p3
cost	1	2	3	4

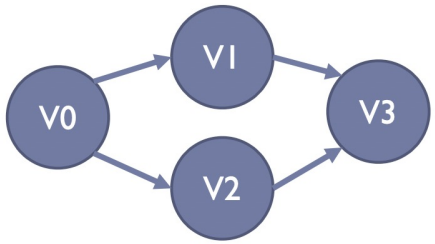
Latency constraint: 2 → no solution

```
Solution count 0

Model is infeasible
Best objective -, best bound -, gap -
=============================================
Optimization Results:

    Variable            X
-------------------------
```

Part 1: Expected Outcome (cont'd)



exec	p0	p1	p2	p3
v0	1	2	3	4
v1	2	1	3	4
v2	2	3	1	4
v3	2	3	3	1

	p0	p1	p2	p3
cost	1	2	3	4

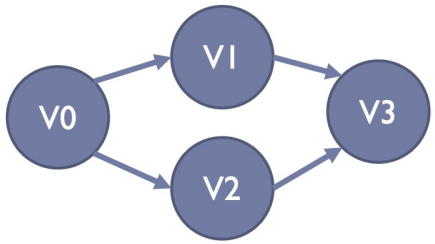
Latency constraint: 3

```
Solution count 1: 10

Optimal solution found (tolerance 1.00e-04)
Best objective 1.000000000000e+01, best bound 1.0000
============================================
Optimization Results:

    Variable            X
-------------------------
       m_0_0            1
       m_1_1            1
       m_2_2            1
       m_3_3            1
         s_1            1
         s_2            1
         s_3            2
         t_0            1
         t_1            1
         t_2            1
         t_3            1
      ah_0_1            1
      ah_0_3            1
      ah_1_3            1
      ah_2_3            1
      used_0            1
      used_1            1
      used_2            1
      used_3            1
============================================
```

Part 1: Expected Outcome (cont'd)



exec	p0	p1	p2	p3
v0	1	2	3	4
v1	2	1	3	4
v2	2	3	1	4
v3	2	3	3	1

	p0	p1	p2	p3
cost	1	2	3	4

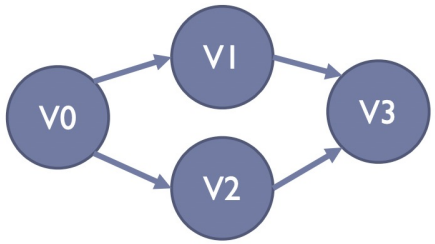
Latency constraint: 4

```
Solution count 2: 6 10

Optimal solution found (tolerance 1.00e-04)
Best objective 6.000000000000e+00, best bound 6.0000000000
============================================
Optimization Results:

    Variable            X
------------------------
       m_0_0            1
       m_1_1            1
       m_2_2            1
       m_3_0            1
         s_1            1
         s_2            1
         s_3            2
         t_0            1
         t_1            1
         t_2            1
         t_3            2
      ah_0_1            1
      ah_0_2            1
      sh_0_3            1
      ah_0_3            1
      ah_1_3            1
      ah_2_3            1
      sh_3_0            1
      used_0            1
      used_1            1
      used_2            1
============================================
```

Part 1: Expected Outcome (cont'd)



exec	p0	p1	p2	p3
v0	1	2	3	4
v1	2	1	3	4
v2	2	3	1	4
v3	2	3	3	1

	p0	p1	p2	p3
cost	1	2	3	4

Latency constraint: 5

```
Solution count 4: 3 7 8 10

Optimal solution found (tolerance 1.00e-04)
Best objective 3.000000000000e+00, best bound 3.000000000000e+00
========================================
Optimization Results:

    Variable            X
-------------------------
       m_0_0            1
       m_1_1            1
       m_2_0            1
       m_3_0            1
         s_1            1
         s_2            1
         s_3            3
         t_0            1
         t_1            1
         t_2            2
         t_3            2
      ah_0_1            1
      sh_0_2            1
      ah_0_2            1
      sh_0_3            1
      ah_0_3            1
      ah_1_2            1
      ah_1_3            1
      sh_2_0            1
      ah_2_1            1
      sh_2_3            1
      ah_2_3            1
      sh_3_0            1
      sh_3_2            1
      used_0            1
      used_1            1
========================================
```

Part 1: What to turn in?

- Your source code
- Generated lp file
- Optimization result (with screenshot)

- Explanation of your implementation (report)

Part 2-1: Extending the basic formulation

- Previously, we try to minimize the "cost" with respect to the given "latency constraint". Let's try the other way around.

- Extend your formulation to minimize the "latency" with respect to the given "cost constraint"
  - You may want to add an integer Gurobi variable for latency
    - m.addVar(vtype=GRB.INTEGER, name="lat")

Part 2-2: Extend the basic formulation

- Now, test the changed formulation with your own task graph

- Build a task graph that has at least 10 tasks and 8 edges
  - And properly set the *exec_time* and *cost* tables
  - You need to choose a right cost constraint to make your optimization feasible, i.e., at least one feasible solution exists

- Perform the latency optimization and analyze the mapping optimization result in your report

Part 2: What to turn in?

- Your source code
- Generated lp file
- Your task graph information
  - With exec_time and cost tables
- Optimization result (with screenshot)

- Explanation of your implementation (report)

Part 3: Communication delay

- Now you have to extend the basic formulation to consider communication delay

- Let us assume that each dependency results in communication delay of 1 time unit if the source and destination tasks are mapped on different processors

- So, this communication delay is conditional!
  - if there is $e_{i,j} \in E$ and there exists k s.t. $m_{i,k} = m_{j,k} = 1$
    - Communication delay applies
  - Otherwise, no communication delay

Part 3: Communication delay (cont'd)

- You need to properly extend the edge constraint using a meta-variable *sh*

- Edge (Dependency) constraints
  - For all $e_{i,j} \in E$,
    - if $sh_{i,j} = 1$, $s_i + t_i \leq s_j$
    - otherwise, $s_i + t_i + 1 \leq s_j$

- Linearize the above conditional equations and implement them in the gurobi file

Part 3: What to turn in?

- Your source code
- Generated lp file
- Optimization result (with screenshot)

- Explanation of your implementation (report)
  - Including your linearized formulations