Embedded Systems 19

REVIEW: multiprocessor scheduling, periodic tasks

	τ_1	τ_2	τ_3
T_i	4	8	6
C_i	2	8	3

$U = \frac{2}{4} + \frac{8}{8} + \frac{3}{6} = 2$

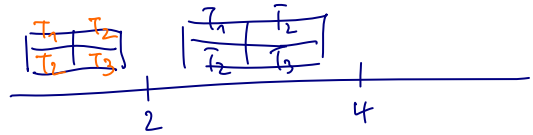
$T = \gcd(4, 8, 6) = 2$

In each slice,

T_1 has $2 \cdot \frac{2}{4} = 1$ unit

T_2 has $2 \cdot \frac{8}{8} = 2$ units

T_3 has $2 \cdot \frac{3}{6} = 1$ unit



REVIEW: Task migration time

Lemma: If U ≤ n, then **within each time slice** the tasks can meet the migration time requirement without missing deadlines, if the task migration time is one unit.

- Sort tasks according to non-increasing computation time
- If computation time = T
  ⇒ allocate processor exclusively.
- If computation time is < T:
  - allocate separately on 1 processor if possible ⇒ no migration
  - allocate part of computation at the end of proc i, rest at beginning of proc i+1
    ⇒ gap of at least 1.

REVIEW: Task migration time

Lemma: If U ≤ n, then **between time slices** the tasks can meet the migration time requirement without missing deadlines, if the task migration time is one unit.

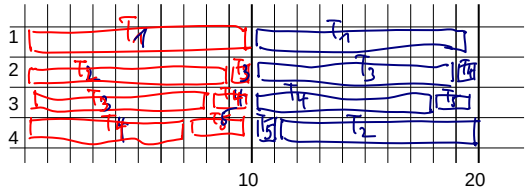
- For each slice, sort tasks according to non-increasing computation times
- If computation time = T → find processor that executed task at the end of the previous slice
  → no migration
  if no such processor exists ⇒ assign to some left-over processor at the end (migration time already accounted for in previous slice)

- If completion block < T
  => Find processor j that executed task at the end of the previous slice
  Assign as much as possible to current processor;
  if insufficient, use j from beginning.
  (no migration at beginning, >= 1 unit gap within slice)
  If no such processor exists => assign task later (migration time already accounted for in the previous slice).

Example (4 processors)

	Computation block
τ_1	10
τ_2	9
τ_3	9
τ_4	9
τ_5	3

T=10



Extension: Task migration time

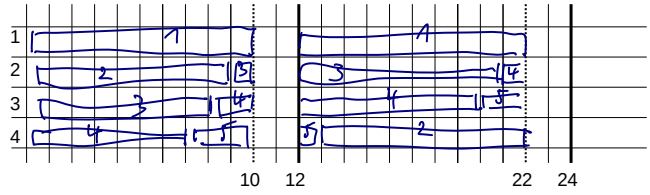
Theorem: Let $T=\gcd(T_1, \ldots, T_m)$ and let R be the task migration time. A **sufficient condition** for scheduling the m periodic tasks is that $U \leq n \cdot (T-R+1)/T$.

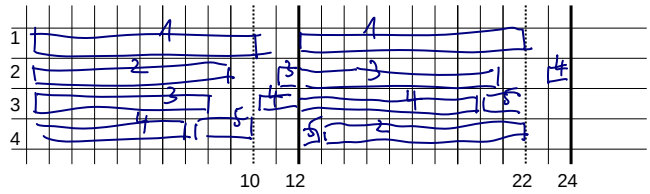
- Schedule as before, but only use $T-R+1$ units of time
- If migration time is too short
  ⇒ shift task(s) to the right end of schedule

Example (4 processors)

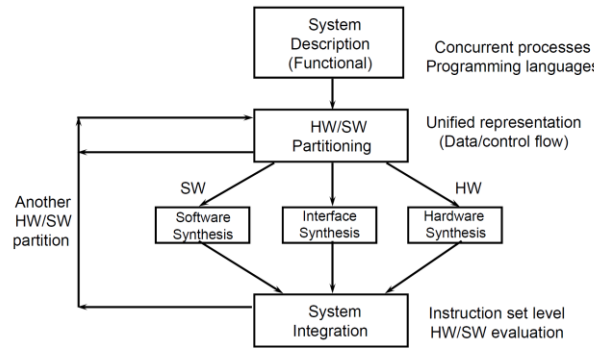
i	*Computation block*
τ_1	10
τ_2	9
τ_3	9
τ_4	9
τ_5	3

T=12,
R=3





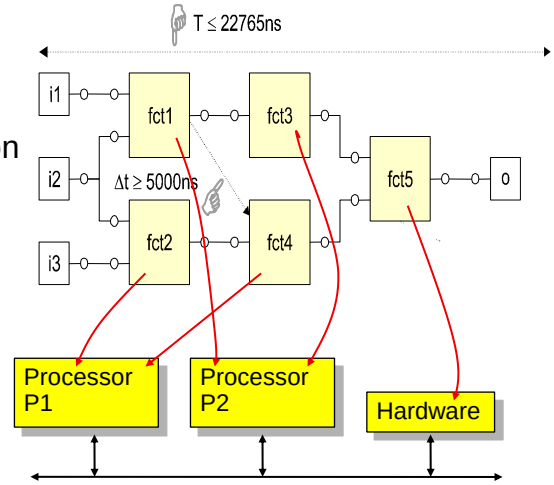
Hardware/Software Codesign



Hardware/software codesign

Specification

Mapping



Objective function

- Cost depends on components selected to implement the application
  - Software Processors: PowerPC, ARM, Pentium,...
  - Hardware: FPGAs, ASIC blocks, ...
  - Communication Infrastructure: buses, networks-on-chip, p2p links, ...
- Multiple metrics, such as cost, power, and performance are weighed against one another
- A function combining multiple metric values into a single value that defines the quality of a partition is called an **Objective Function**, the value returned is called **cost.**

The Partitioning Problem

The **partitioning problem** is to assign
n **objects** $O=\{o_1, \ldots, o_n\}$ to
m **blocks** (also called **partitions**) $P=\{p_1, \ldots, p_m\}$
such that

- $p_1 \cup p_2 \ldots \cup p_m = O$
- $p_i \cap p_j = \varnothing$ for all $i \neq j$, and
- cost c(*P*) is minimized.

Partitioning Methods

- **Heuristic methods**
  - Constructive methods
    - Random mapping
    - Hierarchical clustering
  - Iterative methods
    - Kernighan-Lin Algorithm
    - Simulated Annealing
- **Exact methods**
  - Enumeration
  - Integer Linear Programming (ILP)

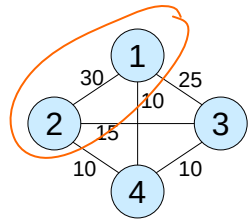
Constructive Methods

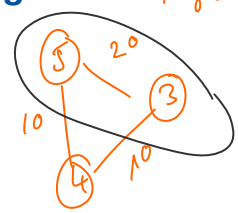
- **Random mapping**
  - Each object randomly assigned to some block
  - Used to find starting partition for iterative methods
- **Hierarchical clustering**
  - Assumes closeness function: determines how desirable it is to group two objects
  - Start with singleton blocks
  - Repeat until termination criterion (e.g., desired number of blocks reached)
    - Compute closeness of blocks (average closeness of object pairs)
    - Find pair of closest blocks
    - Merge blocks
  - Difficulty: find proper closeness function

Example: Hierarchical Clustering

Average closeness;

Termination: 2 blocks





Avg(15, 25) = 20

Hw/Sw Partitioning

- Special case: Bi-partitioning $P=\{p_{SW}, p_{HW}\}$
- Software-oriented approach: P={O,∅}
  - In software, all functions can be realized
  - Performance might be too low ⇒ migrate objects to HW
- Hardware-oriented approach: P={∅,O}
  - In hardware, performance is OK
  - Cost might be too high ⇒ migrate objects to SW

Greedy Hw/Sw Partitioning

Migration of objects to the other block (HW/SW) until no more improvement

```
repeat
  begin
  P'=P;
  for i=1 to n
      begin
        if (cost(move(P,oi)) < cost(P))
        then P':=move(P,oi);
      end;
  end;
until (P==P')
```


Iterative Methods: Kernighan-Lin (K-L)

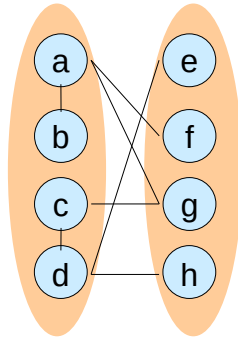
An iterative balanced partitioning (bi-sectioning) heuristic

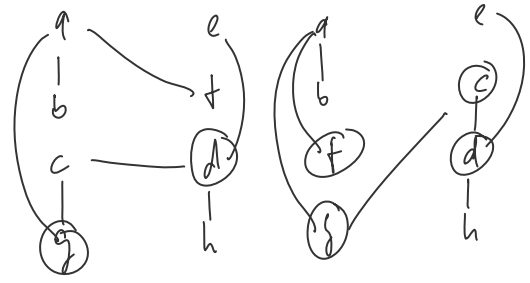
Given: Two sets A and B, such that $|A|=|B|=n$ and $A \cap B=\varnothing$
cost of edge (a,b) in cut: c_{ab}

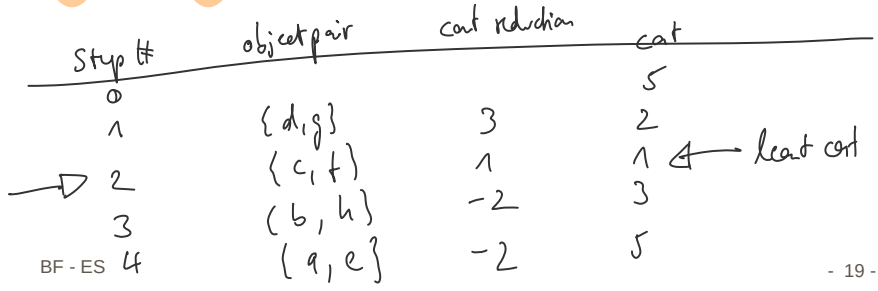
While the cost keeps decreasing

- Mark all objects as „unlocked“
- While there are unlocked pairs left
  - Select pair of unlocked objects (a,b) which give the largest decrease or the smallest increase in cut size
  - Mark a and b as „locked“
  - Exchange a and b
  - Record resulting partition and cost
- Continue with the partition with least cost

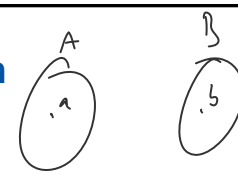
Example





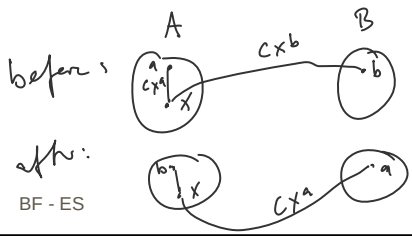


Computing the cost reduction



- External cost of a∈*A*: $E_a=\sum_{v\in B} c_{av}$
- Internal cost of *a*∈*A*: $I_a=\sum_{v\in A} c_{av}$
- Cost reduction for moving *a* : $D_a=E_a-I_a$
- Cost reduction for swapping *a* and *b*: $g_{ab}=D_a+D_b-2c_{ab}$
- Update to *D*-values when a and b are swapped:
  $D'_x = D_x + 2c_{xa} - 2c_{xb}$ for all $x\in A-\{a\}$
  $D'_y = D_y + 2c_{yb} - 2c_{ya}$ for all $y\in B-\{b\}$

gain $a \to B: D_a - c_{ab}$
$b \to A: D_b - c_{ab}$
$D_a + D_b - 2c_{ab}$



$c_{xb} - c_{xa}$
$c_{xa} - c_{xb}$
$\Delta\ 2c_{xa} - 2c_{xb}$

Weighted Example

A={a,b,c}

B={d,e,f}

C	a	b	c	d	e	f
a	0	1	2	3	2	4
b	1	0	1	4	2	1
c	2	1	0	3	2	1
d	3	4	3	0	4	3
e	2	2	2	4	0	2
f	4	1	1	3	2	0

$I_a = 1+2 = 3$ $E_a = 3+2+4 = 9$ $D_a = 9-3 = 6$

$I_b = 1+1 = 2$ $E_b = 4+2+1 = 7$ $D_b = 7-2 = 5$

$I_c = 2+1 = 3$ $E_c = 3+2+1 = 6$ $D_c = 6-3 = 3$

$I_d = 4+3 = 7$ $E_d = 3+4+3 = 10$ $D_d = 10-7 = 3$

$I_e = 4+2 = 6$ $E_e = 2+2+2 = 6$ $D_e = 6-6 = 0$

$I_f = 3+2 = 5$ $E_f = 4+1+1 = 6$ $D_f = 6-5 = 1$

Weighted Example

A={a,b,c}

B={d,e,f}

	a	b	c	d	e	f
a	0	1	2	3	2	4
b	1	0	1	4	2	1
c	2	1	0	3	2	1
d	3	4	3	0	4	3
e	2	2	2	4	0	2
f	4	1	1	3	2	0
D	6	5	3	3	0	1

$g_{ad} = D_a + D_d - 2c_{ad} = 6+3-2\cdot 3 = 3$

$g_{ae} = 6+0-2\cdot 2 = 2$

$g_{af} = 6+1-2\cdot 4 = -1$

$g_{bd} = D_b + D_d - 2c_{bd} = 5+3-2\cdot 4 = 0$

$g_{be} = 1$

$g_{bf} = 4$ ← max

swap b and f ($\hat{g}_1 = 4$)

$g_{cd} = 0$

$g_{ce} = -1$

$g_{cf} = 2$

Weighted Example

A={a,b,c}

B={d,e,f}

	a	b	c	d	e	f
a	0	1	2	3	2	4
b	1	0	1	4	2	1
c	2	1	0	3	2	1
d	3	4	3	0	4	3
e	2	2	2	4	0	2
f	4	1	1	3	2	0
D	6	5	3	3	0	1

A $\begin{cases} D'_a = D_a + 2c_{ab} - 2c_{af} \\ \quad = 6 + 2\cdot 1 - 2\cdot 4 = 0 \\ D'_c = 3 + 2\cdot 1 - 2\cdot 1 = 3 \end{cases}$

B $\begin{cases} D'_d = D_d + 2c_{df} - 2c_{db} = 3 + 2\cdot 3 - 2\cdot 4 = 1 \\ D'_e = D_e + 2c_{ef} - 2c_{eb} = 0 + 2\cdot 2 - 2\cdot 2 = 0 \end{cases}$

Weighted Example

A={a,b,c}

B={d,e,f}

	a	b	c	d	e	f
a	0	1	2	3	2	4
b	1	0	1	4	2	1
c	2	1	0	3	2	1
d	3	4	3	0	4	3
e	2	2	2	4	0	2
f	4	1	1	3	2	0
D	6	5	3	3	0	1

$g_{ab} = -5$

$g_{ac} = -4$

$g_{cd} = -2$

$g_{ce} = -1$ ⇐ max

swap c and e ($\hat{g}_2 = -1$)

$D''_a = 0$

$D''_A = 3$

$g_{ad} = -3$ swap a and d ($\hat{g}_3 = -3$)

partial sums: 4, 3, 0

⇑

largest partial sum

⇒ swap b and t.

Iteration 2:

$\hat{g}_1 = g_{cc} = -1$; $\hat{g}_2 = g_{ab} = -3$, $\hat{g}_3 = g_{dd} = 4$

partial sums: $-1, -4, 0$

largest partial sum: 0 ⇒ STOP

Kernighan-Lin

- **Repeat**
  - Compute D_v für all objects ← $O(n^2)$
  - Mark all vertices as unlocked
  - **For** i=1 to $n/2$ **do** ← $O(n^3)$
    - Compute g_{ab} for all pairs a,b ← $O(n^2)$
    - Pick unlocked a_i, b_i with largest $g_{ab,i}$
    - Mark a_i, b_i as locked
    - Store gain
    - Update D_v für all objects
  - Find k such that $G_k = \sum_{i=1}^{k} g_{ab,i}$ is maximal
  - **If** $G_k > 0$, **then** move $a_1,\ldots,a_k$ from A to B and $b_1,\ldots,b_k$ from B to A.
- **Until** $G_k \leq 0$

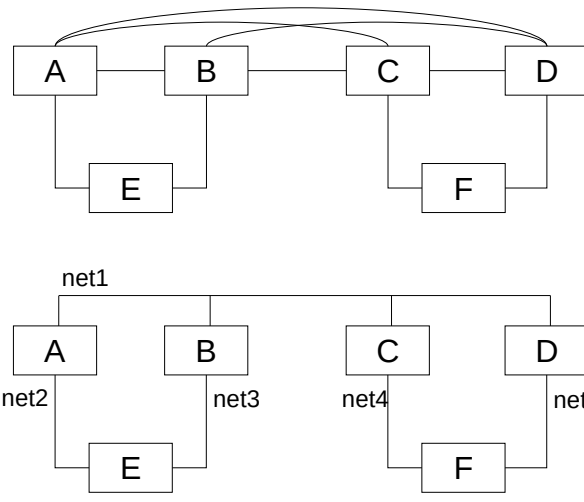
If repeat loop terminates after r iterations ⇒ $O(rn^3)$

"In practice r is small"

Extensions to K-L

- Different block sizes
  - If |A|<|B|, add |B|-|A| dummy objects to A.
    Dummy objects are connected to each other with infinite weight
  - Apply K-L
  - Remove dummies
- Objects with size > 1
  - Replace each object of size *s* with *s* objects of size 1
    new objects are connected with edges of infinite weight
  - Apply K-L
- More than 2 blocks
  - Apply K-L to each pair of blocks

Hypergraphs

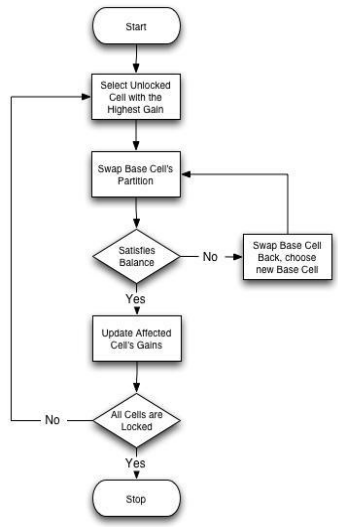


Fiduccia-Mattheyses Heuristic (F-M)

- Objects have size s(o)
- Size of block: sum of size of objects
- **Balanced two-way partition**:
  Given a fraction r, $0 < r < 1$,
  partition a graph into two blocks A and B such that
  $|A| / (|A|+|B|) \approx r$
  and cutset is minimized
- **Linear complexity**

Terminology: object=„cell“, hyperedges=„net“

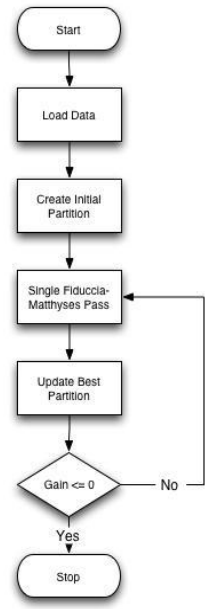
Single pass of the F-M heuristic



- Select the cell with the greatest gains that satisfies balance conditions
- Move the cell and lock it
- Update gains
- Repeat until all cells are locked or will dissatisfy balance conditions

Overall F-M heuristic

- Create an initial partition
- Execute a pass of the F-M heuristic
- Start again using the resulting partition as the initial partition
- Continue until the resulting gain is no longer greater than zero



Calculating Gain

- g(i) = FS(i) - TE(i)
- FS(i): The number of nets which contain cell i but no other object in the same partition as i
- TE(i): The number of nets that consist only of i and other cells currently in the same partition as i

object	FS	TE	gain
A	2	0	2
B	1	0	1
C	1	1	0
D	0	1	−1