Embedded Systems
ASAP Schedules

\[
\text{ASAP}(G_S(V_S, E_S), w) \{ \\
\tau(v_0) = 1; \quad \text{REPEAT} \quad \{ \\
\quad \text{Determine } v_i \text{ whose predec. are planned;} \quad \tau(v_i) = \max\{\tau(v_j) + w(v_j) \forall (v_j, v_i) \in E_S\} \\
\} \quad \text{UNTIL } (v_n \text{ is planned}); \quad \text{RETURN } (\tau); \\
\}
\]
ALAP Schedules

\[
\text{ALAP}(G_S(V_S, E_S), w, L_{\text{max}}) \{ \\
\quad \tau(v_n) = L_{\text{max}} + 1; \\
\quad \text{REPEAT} \{ \\
\quad \quad \text{Determine } v_i \text{ whose succ. are planned;} \\
\quad \quad \tau(v_i) = \min\{\tau(v_j) \forall (v_i, v_j) \in E_S\} - w(v_i) \\
\quad \} \text{ UNTIL (} v_0 \text{ is planned);} \\
\quad \text{RETURN } (\tau); \\
\}\}
\]
List Scheduling

- List scheduling: extension of ALAP/ASAP method

- Preparation:
  - Greedy strategy (does NOT guarantee optimum solution)
  - Topological sort of task graph $G=(V,E)$
  - Computation of priority of each task:

Possible priorities $u$:

- Number of successors
- Longest path
- **Mobility** = $\tau$ (ALAP schedule) - $\tau$ (ASAP schedule)
  - Defined for each operation
  - Zero mobility implies that an operation can be started only at one given time step
  - Mobility greater than 0 measures span of time interval in which an operation may start $\rightarrow$ Slack on the start time
Integer linear programming models

- Ingredients:
  - Cost function
  - Constraints

> Involving linear expressions of integer variables from a set \( X \)

**Cost function**

\[
C = \sum_{x_i \in X} a_i x_i \quad \text{with} \quad a_i \in \mathbb{R}, x_i \in \mathbb{N} \quad (1)
\]

**Constraints:**

\[
\forall j \in J : \sum_{x_i \in X} b_{i,j} x_i \geq c_j \quad \text{with} \quad b_{i,j}, c_j \in \mathbb{R} \quad (2)
\]

**Def.** The problem of minimizing (1) subject to the constraints (2) is called an **integer linear programming (ILP) problem**.

If all \( x_i \) are constrained to be either 0 or 1, the IP problem said to be a **0/1 integer linear programming problem**.
(Time constrained) Force-directed scheduling

- Goal: balanced utilization of resources
- Based on spring model
- Originally proposed for high-level synthesis

Force
- Used as a priority function
- Related to concurrency – sort operations for least force
- Mechanical analogy: Force = constant x displacement
  - Constant = operation-type distribution
  - Displacement = change in probability

1. Compute time frames $R(j)$
2. Compute "probability" $P(j,i)$ of assignment $j \rightarrow i$

$$R(j) = \{ \text{ASAP-control step} \ldots \text{ALAP-control step} \}$$

$$P(j, i) = \begin{cases} \frac{1}{|R(j)|} & \text{if } i \in R(j) \\ 0 & \text{otherwise} \end{cases}$$
- Architecture Synthesis
- HW/SW Codesign
- Power Aware Computing

3.2.2011 Lecture by Bernd Finkbeiner, Head of Reactive Systems Group at Saarland University (http://react.cs.uni-sb.de/)
Codesign Definition and Key Concepts

- **Codesign**
  - The meeting of system-level objectives by exploiting the trade-offs between hardware and software in a system through their concurrent design

- **Key concepts**
  - **Concurrent:** hardware and software developed at the same time on parallel paths
  - **Integrated:** interaction between hardware and software development to produce design meeting performance criteria and functional specs
Low Power HW/SW Co-Design of Smart Cards: Approach

<table>
<thead>
<tr>
<th>Selection of Silicon Technology</th>
<th>Selection of Architecture</th>
<th>Apply Low Power Measures and Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology node (130nm, 90nm, ...)</td>
<td>CPU architecture (8/16/32-bit)</td>
<td>Low power design (supply voltage, ...)</td>
</tr>
<tr>
<td>Non-volatile memories (Flash, EEPROM, ...)</td>
<td>Security level</td>
<td>Automated tools (clock gating, ...)</td>
</tr>
</tbody>
</table>

Software Architecture
- SW power management
- High performance optimizations

Software Tools
- Power emulation
- Optimized compiler

Hardware

Software
Typical Codesign Process

- System Description (Functional)
  - FSM-directed graphs
  - Concurrent processes
    - Programming languages
  - HW/SW Partitioning
    - Unified representation
      - (Data/control flow)
    - SW
      - Software Synthesis
    - Interface Synthesis
    - HW
      - Hardware Synthesis
    - System Integration
      - Instruction set level
        - HW/SW evaluation
      - Another HW/SW partition
Co-design Flow in more detail

Informal Specification → System Model → System Simulation

Algorithmic Design

Hardware/Software Partitioning → Partitioned Model

Partitioned Model & Sch.

Schedule → HW/SW Co-simulation

Refine
Co-design Flow Cont…

Partitioned Model + Sch.

Communication Synthesis

Software Model

HW/SW Co-simulation

Compilation

Hardware Model

Refine

Synthesis

Binary Exec. Model

Emulate or Prototype

Refine

Gate-level Model

Fabrication
Categories of Codesign Problems

- Codesign of embedded systems
  - Usually consist of sensors, controller, and actuators
  - Are reactive systems
  - Usually have real-time constraints
  - Usually have dependability constraints
- Codesign of ISAs
  - Application-specific instruction set processors (ASIPs)
  - Compiler and hardware optimization and trade-offs
- Codesign of Reconfigurable Systems
  - Systems that can be personalized after manufacture for a specific application
Main Tasks of the Codesign Problem

- Specification of the system
- Hardware/Software Partitioning
  - Architectural assumptions - type of processor, interface style between hardware and software, etc.
  - Partitioning objectives - maximize speedup, latency requirements, minimize size, cost, etc.
  - Partitioning strategies - high level partitioning by hand, automated partitioning using various techniques, etc.
- Scheduling
  - Operation scheduling in hardware
  - Instruction scheduling in compilers
  - Process scheduling in operating systems
- Modeling/Simulation of the hardware/software system during the design process
Issues in Partitioning

- Specification abstraction level
- Granularity
- System-component allocation
- Metrics and estimations
- Partitioning algorithms
- Objective and closeness functions
- Partitioning algorithms
- Flow of control and designer interaction
Hardware Software Partitioning

- Decompose (i.e., partition) the function $F$ of the system into $N$ sub-functions $F_1, F_2, F_3 \ldots F_N$

- Decompose the constraints and design objectives of the system into sub-constraints and design sub-objectives

- Cluster $F_1, F_2, F_3 \ldots F_N$ into $M$ partitions to run on $M$ processors elements (mapping)

- Given:

$$F = \{ F_1, F_2, F_3 \ldots F_N \} ;$$
$$P = \{ P_1, P_2, P_3 \ldots P_M \}$$

- Find a lowest cost partition (cluster), as computed by an objective function

- Exhaustive approach $O(M^N)$

---

communication !!
Computation of Metrics

- Two approaches to computing metrics
  - Creating a detailed implementation
    - Produces accurate metric values
    - Impractical as it requires too much time
  - Creating a rough implementation
    - Includes the major register transfer components of a design
    - Skips details such as precise routing or optimized logic, which require much design time
    - Determining metric values from a rough implementation is called estimation
Estimation

- 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, …

- **Profiling tools** are used prior to partitioning to determine cost and also to determine critical parts of application
  - obtain **performance** (or **power**, **area**, …) metrics of the system
  - helps the designer optimize the design and decide whether to implement certain functions in hardware or software
Poweremulation

- POWERHOUSE vision

- Implementation of power model on emulation platform: *Power emulation (PE)*
- Generate power estimates as a by-product of functional emulation during system run-time
- Visualize and evaluate data within a software IDE
- Improve power-awareness based on power feedback
Objective and Closeness Functions

- Multiple metrics, such as cost, power, and performance are weighed against one another
  - An expression combining multiple metric values into a single value that defines the quality of a partition is called an Objective Function
  - The value returned by such a function is called cost
- Because many metrics may be of varying importance, a weighted sum objective function is used (and constr.)
  - e.g Cost = c1 * F(area, area_constr)
    + c2 * F(delay, delay_constr)
    + c3 * F(power, power_constr)
Partitioning Algorithm Classes

- Constructive algorithms
  - Group objects into a complete partition
  - Use closeness metrics to group objects, hoping for a good partition

- Iterative algorithms
  - Modify a complete partition in the hope that such modifications will improve the partition
  - Use an objective function to evaluate each partition
  - Yield more accurate evaluations than closeness functions used by constructive algorithms

- In practice, a combination of constructive and iterative algorithms is often employed
Partitioning Methods

- Exact methods
  - Integer Linear Programming (ILP)
  - ...

- Heuristic methods
  - Constructive methods
    - Random mapping
    - Hierarchical clustering
  - Iterative methods
    - Kernighan-Lin Algorithm
    - Simulated Annealing
    - ...

ILP HW/SW Partitioning
Example from Christian Plessl, Universität Paderborn
ILP HW/SW Partitioning

/* objective function */

/* variables in (0,1) */

/* unique mapping constraints */

```c
/* objective function */
*/

/**
 * @param x11
 * @param x12
 * @param x21
 * @param x22
 * @param x31
 * @param x32
 * @param x41
 * @param x42
 * @param x51
 * @param x52
 * @param x61
 * @param x62
 */

int x11;
int x12;
int x21;
int x22;
int x31;
int x32;
int x41;
int x42;
int x51;
int x52;
int x61;
int x62;
```
ILP HW/SW Partitioning

allocation & binding

<table>
<thead>
<tr>
<th>task</th>
<th>SW cost</th>
<th>HW cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>80</td>
<td>320</td>
</tr>
<tr>
<td>2</td>
<td>240</td>
<td>170</td>
</tr>
<tr>
<td>3</td>
<td>710</td>
<td>120</td>
</tr>
<tr>
<td>4</td>
<td>130</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>400</td>
</tr>
<tr>
<td>6</td>
<td>80</td>
<td>260</td>
</tr>
</tbody>
</table>

total cost = 570

n * m = 12 binary variables
ILP HW/SW Partitioning

- Constraint on the hardware cost
  - cost of all tasks mapped to hardware must not exceed 300

\[
\sum_{i=1}^{6} c_{i,2} \cdot x_{i,2} \leq 300
\]
Partitioning Methods

- **Exact methods**
  - Enumeration
  - Integer Linear Programming (ILP)
- **Heuristic methods**
  - Constructive methods
    - Random mapping
    - Hierarchical clustering
  - Iterative methods
    - Kernighan-Lin Algorithm
    - Simulated Annealing
    - ...
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
Case Study: YSC (IBM)

- **Yorktown Silicon Compiler:** functional partitioning of hardware
- **Input:** functional description on the level of arithmetic and logical expressions
- **Target:** partitioning to several chips
- **Abstraction level:** functional units of datapaths (ALUs, registers)
- **Method:** hierarchical clustering

\[
closeness(p_i, p_j) = \left( \frac{\text{sharedwires}(p_i, p_j)}{\text{maxwires}} \right)^{c_1} \cdot \left( \frac{\text{maxsize}}{\min\{\text{size}(p_i), \text{size}(p_j)\}} \right)^{c_2} \cdot \left( \frac{\text{maxsize}}{\text{size}(p_i) + \text{size}(p_j)} \right)
\]
Closeness function

\[
\text{Closeness} (+, =) = \frac{8 + 0}{8} \times \frac{300}{120} \times \frac{300}{120 + 140} = 2.9
\]

\[
\text{Closeness} (-, <) = \frac{0 + 4}{8} \times \frac{300}{160} \times \frac{300}{160 + 180} = 0.8
\]

\[
closeness(p_i, p_j) = \left(\frac{\text{sharedwires}(p_i, p_j)}{\text{maxwires}}\right)^{c_1} \cdot \left(\frac{\text{maxsize}}{\min\{\text{size}(p_i), \text{size}(p_j)\}}\right)^{c_2} \cdot \left(\frac{\text{maxsize}}{\text{size}(p_i) + \text{size}(p_j)}\right)
\]
Partitioning Methods

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

- Two broad categories:
  - Greedy algorithms
    - Only accept moves that decrease cost
    - Can get trapped in local minima
  - Hill-climbing algorithms
    - Allow moves in directions increasing cost (retracing)
      - Through use of stochastic functions
    - Can escape local minima
    - E.g., simulated annealing
Iterative Partitioning Algorithms

- The computation time in an iterative algorithm is spent **evaluating large numbers of partitions**.
- Iterative algorithms differ from one another primarily in the ways in which they **modify the partition** and in which they **accept or reject bad modifications**.
- The goal is to **find global minimum** while performing as little computation as possible.
HW/SW Partitioning

- Special case: Bi-partitioning $P=\{p_{SW}, p_{HW}\}$

- Software-oriented approach: $P=\{O,\emptyset\}$
  - In software, all functions can be realized
  - Performance might be too low $\Rightarrow$ migrate objects to HW

- Hardware-oriented approach: $P=\{\emptyset,O\}$
  - In hardware, performance is OK
  - Cost might be too high $\Rightarrow$ 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,o_i) < cost(P))
                    then P′:=move(P,o_i);
            end;
        end;
    until (P==P′)
Kernighan-Lin (Min-Cut)

Kernighan/Lin – Fidducia/Mattheyses algorithm
• Start with all task vertices free to swap/move (unlocked)
• Label each possible swap/move with immediate change in execution time that it causes (gain)
• Iteratively select and execute a swap/move with highest gain (whether positive or negative); lock the moving vertex (i.e., cannot move again during the pass),
• Best solution seen during the pass is adopted as starting solution for next pass
Questions: How to compute cost reduction? What pairs to be swapped?

Consider the change of internal & external connections.
Computing the cost reduction

- External cost of $a \in A$: $E_a = \sum_{v \in B} c_{av}$
- Internal cost of $a \in A$: $I_a = \sum_{v \in A} c_{av}$
- 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:
  \[ D'_x = D_x + 2c_{xa} - 2c_{xb} \text{ for all } x \in A - \{a\} \]
  \[ D'_y = D_y + 2c_{yb} - 2c_{ya} \text{ for all } y \in B - \{b\} \]
Kernighan-Lin

- **Repeat**
  - Compute $D_v$ für all objects
  - Mark all vertices as unlocked
  - **For** $i=1$ to $n/2$ **do**
    - Compute $g_{ab}$ for all pairs $a,b$
    - 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** $G_k \leq 0$

$O(n^2)$

$O(n^3)$.

Suppose the repeat loop terminates after $r$ passes.

The total running time: $O(rn^3)$

Polynomial-time algorithm?
Weighted Example

A={a,b,c}
B={d,e,f}

Repeat
- Compute $D_{a}$ for all objects
- Mark all vertices as unlocked
- For $i=1$ to $n/2$ do
  - Compute $g_{ab}$ for all pairs $a,b$
  - Pick unlocked $a,b$ with largest $g_{ab}$
  - Mark $a,b$ as locked
  - Store gain
  - Update $D_{a}$ for all objects
  - Find $k$ such that $G_{k}=\sum_{j=1}^{k} g_{ab}$ is maximal
  - If $G_{k}>0$, then move $a,\ldots,a_i$ from A to B and $b,\ldots,b_k$ from B to A.

Until $G_{k}=0$

Initial cut cost = $(3+2+4)+(4+2+1)+(3+2+1) = 22$

• Iteration 1:

$I_a = 1 + 2 = 3$; $E_a = 3 + 2 + 4 = 9$; $D_a = E_a - I_a = 9 - 3 = 6$
$I_b = 1 + 1 = 2$; $E_b = 4 + 2 + 1 = 7$; $D_b = E_b - I_b = 7 - 2 = 5$
$I_c = 2 + 1 = 3$; $E_c = 3 + 2 + 1 = 6$; $D_c = E_c - I_c = 6 - 3 = 3$
$I_d = 4 + 3 = 7$; $E_d = 3 + 4 + 3 = 10$; $D_d = E_d - I_d = 10 - 7 = 3$
$I_e = 4 + 2 = 6$; $E_e = 2 + 2 + 2 = 6$; $D_e = E_e - I_e = 6 - 6 = 0$
$I_f = 3 + 2 = 5$; $E_f = 4 + 1 + 1 = 6$; $D_f = E_f - I_f = 6 - 5 = 1$
g-Value Computation

- **Iteration 1:**

  \[ I_a = 1 + 2 = 3; \quad E_a = 3 + 2 + 4 = 9; \quad D_a = E_a - I_a = 9 - 3 = 6 \]
  \[ I_b = 1 + 1 = 2; \quad E_b = 4 + 2 + 1 = 7; \quad D_b = E_b - I_b = 7 - 2 = 5 \]
  \[ I_c = 2 + 1 = 3; \quad E_c = 3 + 2 + 1 = 6; \quad D_c = E_c - I_c = 6 - 3 = 3 \]
  \[ I_d = 4 + 3 = 7; \quad E_d = 3 + 4 + 3 = 10; \quad D_d = E_d - I_d = 10 - 7 = 3 \]
  \[ I_e = 4 + 2 = 6; \quad E_e = 2 + 2 + 2 = 6; \quad D_e = E_e - I_e = 6 - 6 = 0 \]
  \[ I_f = 3 + 2 = 5; \quad E_f = 4 + 1 + 1 = 6; \quad D_f = E_f - I_f = 6 - 5 = 1 \]

- **\( g_{xy} = D_x + D_y - 2c_{xy} \):**

  \[ g_{ad} = D_a + D_d - 2c_{ad} = 6 + 3 - 2 \times 3 = 3 \]
  \[ g_{ae} = 6 + 0 - 2 \times 2 = 2 \]
  \[ g_{af} = 6 + 1 - 2 \times 2 = -1 \]
  \[ g_{bd} = 5 + 3 - 2 \times 4 = 0 \]
  \[ g_{be} = 5 + 0 - 2 \times 2 = 1 \]
  \[ g_{bf} = 5 + 1 - 2 \times 1 = 4 \text{ (maximum)} \]
  \[ g_{cd} = 3 + 3 - 2 \times 3 = 0 \]
  \[ g_{ce} = 3 + 0 - 2 \times 2 = -1 \]
  \[ g_{cf} = 3 + 1 - 2 \times 1 = 2 \]

- **Swap \( b \) and \( f \) \((g_1 = 4)\)**
D-Value Computation

- $D'_x = D_x + 2\, c_{xp} - 2\, c_{xq}, \ \forall \ x \in A - \{p\}$ (swap $p$ and $q$, $p \in A$, $q \in B$)

\[
\begin{align*}
D'_a &= D_a + 2c_{ab} - 2c_{af} = 6 + 2 \times 1 - 2 \times 4 = 0 \\
D'_c &= D_c + 2c_{cb} - 2c_{cf} = 3 + 2 \times 1 - 2 \times 1 = 3 \\
D'_d &= D_d + 2c_{df} - 2c_{db} = 3 + 2 \times 3 - 2 \times 4 = 1 \\
D'_e &= D_e + 2c_{ef} - 2c_{eb} = 0 + 2 \times 2 - 2 \times 2 = 0 
\end{align*}
\]

- $g_{xy} = D'_x + D'_y - 2c_{xy}$.

\[
\begin{align*}
g_{ad} &= D'_a + D'_d - 2c_{ad} = 0 + 1 - 2 \times 3 = -5 \\
g_{ae} &= D'_a + D'_e - 2c_{ae} = 0 + 0 - 2 \times 2 = -4 \\
g_{cd} &= D'_c + D'_d - 2c_{cd} = 3 + 1 - 2 \times 3 = -2 \\
g_{ce} &= D'_c + D'_e - 2c_{ce} = 3 + 0 - 2 \times 2 = -1 \ (maximum) 
\end{align*}
\]

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

Repeat
- Compute $D_v$ for all objects
- Mark all vertices as unlocked
- For $i = 1 \text{ to } n/2$ do
  - Compute $g_{uv}$ for all pairs $u,v$
  - Pick unlocked $u,v$ with largest $g_{uv}$
  - Mark $u,v$ as locked
  - Store gain
  - Update $D_u$ for all objects
- Find $k$ such that $\hat{g}_k = \sum_{i=1}^{n} g_{uv}$ is maximal
- If $\hat{g}_k > 0$, then move $a_1, \ldots, a_k$ from A to B and $b_1, \ldots, b_k$ from B to A.
- Until $\hat{g}_k < 0$
Swapping Pair Determination

- Repeat
  - Compute $D_i$ for all objects
  - Mark all vertices as unlocked
  - For $i = 1$ to $n/2$
    - Compute $g_{ab}$ for all pairs $a, b$
    - Pick unlocked $a, b$ with largest $g_{ab}$
    - Mark $a, b$ as locked
    - Store gain
    - Update $D_i$ for all objects
- Find $i$ such that $G_i = \sum g_{ab}$ is maximal
- If $G_i > 0$, then move $a_1, \ldots, a_i$ from A to B and $b_1, \ldots, b_i$ from B to A.
- Until $G_i \leq 0$

\[ D''_x = D'_x + 2c_{xp} - 2c_{xq}, \quad \forall x \in A - \{p\} \]

\[
D''_a = D'_a + 2c_{ac} - 2c_{ae} = 0 + 2 \times 2 - 2 \times 2 = 0
\]

\[
D''_d = D'_d + 2c_{de} - 2c_{dc} = 1 + 2 \times 4 - 2 \times 3 = 3
\]

- $g_{xy} = D''_x + D''_y - 2c_{xy}$.

\[
g_{ad} = D''_a + D''_d - 2c_{ad} = 0 + 3 - 2 \times 3 = -3 (\hat{g}_3 = -3)
\]

- Note that this step is redundant ($\sum_{i=1}^{n} \hat{g}_i = 0$).
- Summary: $\hat{g}_1 = g_{bf} = 4$, $\hat{g}_2 = g_{ce} = -1$, $\hat{g}_3 = g_{ad} = -3$.
- Largest partial sum $\max \sum_{i=1}^{k} \hat{g}_i = 4$ ($k = 1$) $\Rightarrow$ Swap $b$ and $f$.\]
Next Iteration

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>e</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Initial cut cost = (1+3+2)+(1+3+2)+(1+3+2) = 18 (22-4)

• Iteration 2: Repeat what we did at Iteration 1 (Initial cost = 22-4 =18).

• Summary: \( \tilde{g}_1 = g_{ce} = -1, \tilde{g}_2 = g_{ab} = -3, \tilde{g}_3 = g_{fd} = 4. \)

• Largest partial sum = \( \max \sum_{i=1}^{k} \tilde{g}_i = 0 \) \( (k=3) \Rightarrow \text{Stop!} \)
Simulated Annealing

- General method for solving combinatorial optimization problems.
- Based the model of slowly cooling crystal liquids.
- Changes leading to a poorer configuration (with respect to some cost function) are accepted with a certain probability.
- This probability is controlled by a temperature parameter: the probability is smaller for smaller temperatures.
Simulated Annealing Algorithm

procedure SimulatedAnnealing;
var i, T: integer;
begin
  temp := temp_start;
  cost:=c(P);
  while (Frozen()==FALSE) do
  begin
    while (Equilibrium()==FALSE) do
    begin
      P' := RandomMove(P);
      cost'=c(P')
      deltax := cost' - cost;
      if (Accept(deltacost, temp)>random[0,1))
      then P=P'; cost=cost'
    end;
    temp:= decreaseTemp(temp)
  end;
end;

Simulated Annealing

- **Annealing schedule**: DecreaseTemp(), Frozen()
  - temp_start=1.0
  - temp = $\alpha \cdot \text{temp}$ (typical: $0.8 \leq \alpha \leq 0.99$)
  - stop at temp < temp_min or if no more improvement

- **Equilibrium**:
  - After certain number of iterations or when no more improvement

- **Complexity**:
  - From exponential to constant, depending on choice of Equilibrium(), DecreaseTemp(), Frozen()
  - The longer the runtime, the better the results
  - Usually functions constructed to obtain polynomial runtime
<table>
<thead>
<tr>
<th>Paper</th>
<th>Dynamic/ Static</th>
<th>Strategy</th>
<th>Criteria</th>
<th>Model/ Data Structure</th>
<th>Granularity of Partitioning</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>Static</td>
<td>Simulated Annealing Geedy</td>
<td>Minimal area, data-rate constraints</td>
<td>System Graph Model (like H-CDFG)</td>
<td>n/a</td>
<td>operations linear</td>
</tr>
<tr>
<td>41</td>
<td>Static</td>
<td>Geedy (see [42])</td>
<td>Minimal area, data-rate constraints</td>
<td>Hierarchical Sequence Graph Petri-nets, (annotated) CDFG</td>
<td>operations</td>
<td>n/a</td>
</tr>
<tr>
<td>77</td>
<td>Static</td>
<td>Simulated Annealing</td>
<td>Minimal communication cost</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>34</td>
<td>Static</td>
<td>Simulated Annealing</td>
<td>Hardware suitability (compare local phase [54])</td>
<td>(extended) C^t syntax graph</td>
<td>basic blocks</td>
<td>n/a</td>
</tr>
<tr>
<td>54</td>
<td>Static</td>
<td>GCLP</td>
<td>GC objective function (e.g. Area combined with speed)</td>
<td>n/a</td>
<td>Tasks (instruction level subgraphs)</td>
<td>O(ne), e=edges</td>
</tr>
<tr>
<td>88</td>
<td>Static</td>
<td>Binary Constraint Search</td>
<td>Constraints of encapsulated partitioning algorithm</td>
<td>n/a</td>
<td>n/a</td>
<td>O(par(S)) = encaps. part. alg. n/a</td>
</tr>
<tr>
<td>60</td>
<td>Static</td>
<td>Dynamic Programming</td>
<td>Temporal size of loops / leaf functions</td>
<td>n/a</td>
<td>loops, leaf functions</td>
<td>n/a</td>
</tr>
<tr>
<td>53</td>
<td>Static</td>
<td>GCLP (MIBS)</td>
<td>See [54]</td>
<td>CDFG</td>
<td>Tasks</td>
<td>O(n^3 + n^2 B), B=bins</td>
</tr>
<tr>
<td>82</td>
<td>Static</td>
<td>Evolutionary (Generic)</td>
<td>Minimal area, timing and concurrency constraints</td>
<td>CDFG</td>
<td>functional elements</td>
<td>O(gp), g=generations, p=population</td>
</tr>
<tr>
<td>30</td>
<td>Static</td>
<td>Clustering</td>
<td>Minimal cost, minimal power, timing and power constraints</td>
<td>Task Graph</td>
<td>task clusters</td>
<td>n/a</td>
</tr>
<tr>
<td>50</td>
<td>Dynamic</td>
<td>Geedy, Clustering</td>
<td>Minimize area, timing constraints</td>
<td>Task Graph</td>
<td>task clusters</td>
<td>n/a</td>
</tr>
<tr>
<td>65</td>
<td>Dynamic</td>
<td>Clustering</td>
<td>Area constraints maximize fitness (minimize area and interconnect)</td>
<td>CDFG</td>
<td>loop clusters</td>
<td>linear</td>
</tr>
<tr>
<td>69</td>
<td>Static</td>
<td>Evolutionary (Generic)</td>
<td>Minimal area, data-rate constraints</td>
<td>DFG</td>
<td>fine operations</td>
<td>n/a</td>
</tr>
<tr>
<td>84</td>
<td>Dynamic</td>
<td>Evolutionary</td>
<td>Maximum rank (Pareto ranking in power and price)</td>
<td>Task Graph</td>
<td>Tasks</td>
<td>n/a</td>
</tr>
<tr>
<td>91</td>
<td>Static</td>
<td>Geedy</td>
<td>Temporal size of loops / leaf functions</td>
<td>Task Graph</td>
<td>Tasks</td>
<td>loops</td>
</tr>
<tr>
<td>14</td>
<td>Static</td>
<td>Dynamic Programming</td>
<td>Minimum latency, resource constraints</td>
<td>DFG</td>
<td>Functions</td>
<td>polynomial</td>
</tr>
<tr>
<td>12</td>
<td>Static</td>
<td>Simulated Annealing, Kernighan-Lin</td>
<td>Minimize latency, area constraints</td>
<td>Call graph</td>
<td></td>
<td>n/a</td>
</tr>
</tbody>
</table>

Table 2.1: Inventorization of several papers on hardware software partitioning with corresponding partitioning schemes, criteria, and data structures
HW/SW Co-Simulation

System Architect Designer
Introduction: Co-Simulation

Co-Simulation:
- Simulation methodology
- Individual components simulated by different simulation tools
- Different modeling languages
- Different abstraction levels
- But: common co-simulation

Why use co-simulation?
- Handling increased complexity
- Flexibility
- Verification already in early design phases
- Simulation performance improvements
- Short development cycles
Multi-HDL design

- SystemC [SystemC]
- ModelSim [VHDL]
- ADVanceMS [VHDL/AMS]
- NCSIM-SimVision (AMS Designer) [VHDL/AMS]
- Saber [SaberMAST]
- Simulink [Matlab/Simulink]
Design Methodology

System Design Flow

- Top-Level System Design
- Partitioning / Block-Level Modelling
- Automatic Generation of Cosimulation Framework
- Cosimulation
- Verification
SyAD: Co-Simulation

Synchronisation method is implemented as decentralized, “synchronous”, conservative protocol
Motivation for Run-Time Co-Simulation Model Switching

System level:
- Validation and analysis of entire embedded systems
- Focus: short simulation time for longer simulated time (>> 1s)
- Abstracted behavior: hides low-level effects that might propagate

Physical level:
- High simulation time: simulation of complex analog components
- Relatively short simulated times (μs, ms)
- Detailed behavior

Co-simulation problems:
- Simulated times: physical level vs. system level
- Co-simulation performance: determined by slowest simulator
  → critical in physical level/system level co-simulation

Idea: Run-time switching of co-simulation models
Run-Time Co-Simulation Model Switching

Run-time co-simulation model switching:

- Modeling of a single component by using multiple HDL (discrete & continuous) and abstraction levels
- Synchronized run-time switching between the abstraction level models

Features:

- Long simulated time / high simulation speed (system level models) plus high accuracy (low physical level models)
  - Using fast high level models during normal circumstances
  - Switch to high-detailed models during time intervals of particular interest
- Enhances co-simulation speed
  - Using computational expensive simulation models only in a clearly defined area
TEODACS: Overview

TEODACS: Test, Evaluation and Optimization of Dependable Automotive Communication Systems

- Co-simulation platform
  - FlexRayXpert.Sim
  - Car simulator
    - CarMaker / AVL InMotion™

- Hardware prototype platform
  - FlexRayXpert.Lab
  - Car simulator
    - CarMaker / AVL InMotion™ (real-time)

- Interface testcase definition (car environment, stimuli)
- Stimulation and analysis
  - Signal level
  - Frame level
  - Sample level
  - Analog level

- FlexRay network:
  - Co-simulation framework CISC SyAD®
  - FlexRay Channel
    - FlexRay Topology (VHDL-AMS, SystemC)
  - Further FlexRay Nodes

- Further FlexRay Nodes
FlexRayXpert.Sim: Experimental Setup

1: VHDL-AMS Topology Model
- Communication Controller 2 (SystemC)
- Application
- Transceiver (VHDL-AMS)
- ... FlexRay Termination
- 1.0m ... Cable Segment Length

2: SystemC Topology Model
- Communication Controller 0 (SystemC)
- Application
- Transceiver (SystemC)
- ... FlexRay Node 0

- Communication Controller 1 (SystemC)
- Application
- Transceiver (SystemC)
- ... FlexRay Node 1

- Communication Controller 3 (SystemC)
- Application
- Transceiver (SystemC)
- ... FlexRay Node 3

- Communication Controller 4 (SystemC)

- Switches 0, 1, 2, 3, 4
PowerCard - Methodologies for Designing Power-Aware Smart Card Systems
Contactless Smart Cards as Future Mobile Devices

- Contactless smart card controllers are currently used in various demanding applications
  - payment, e.g. debit/credit cards
  - identification, e.g. electronic passport
  - pay TV

- …and there exist ideas for much more complex use cases by connecting displays, buttons, and finger print sensors to the controller
System Abstraction

- Requirements
  - In general independent of hardware and software
  - Basic smart card OS functionality should be provided
  - Focus on algorithm design and memory system (limited resource)

- Levels of Abstraction

<table>
<thead>
<tr>
<th>Level</th>
<th>Model representation</th>
<th>Model of computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional Level</td>
<td>Object oriented</td>
<td>Sequential execution</td>
</tr>
<tr>
<td></td>
<td>Interface-based commun.</td>
<td>Untimed/timed</td>
</tr>
<tr>
<td>Transaction Level</td>
<td>TL1 SoC Model</td>
<td>SoC: Parallel tasks, timed</td>
</tr>
<tr>
<td></td>
<td>Abstract processing units</td>
<td>SW: untimed, delays</td>
</tr>
<tr>
<td>Prototype Level</td>
<td>Cycle accurate HW</td>
<td>Parallel hw models, FSMs</td>
</tr>
<tr>
<td></td>
<td>Cross-compiled software</td>
<td>Sequential software</td>
</tr>
</tbody>
</table>
Abstract platforms are more stable
Different solutions can be derived from an abstract model
This results in more stable systems than old system redesign
Design space Exploration based on hierarchical platforms
# Design Space Examination

## Level of Abstraction

<table>
<thead>
<tr>
<th>Performance</th>
<th>Power</th>
<th>Chip size</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-defined delays</td>
<td>Memory</td>
<td>Memory</td>
<td>FP fault model</td>
</tr>
<tr>
<td>Memory delays</td>
<td>Programming</td>
<td>utilization</td>
<td>Fault injection</td>
</tr>
<tr>
<td>Communication delay</td>
<td>API objects</td>
<td>API usage</td>
<td></td>
</tr>
<tr>
<td>System busses</td>
<td>Bus SA</td>
<td>Processors</td>
<td>Architecture fault model</td>
</tr>
<tr>
<td>HW/SW interfaces</td>
<td>Memory blocks</td>
<td>Coprocessors</td>
<td></td>
</tr>
<tr>
<td>Task parallelism</td>
<td>API energy</td>
<td>Memory size</td>
<td></td>
</tr>
<tr>
<td>SW: IS simulator</td>
<td>SW: simulator</td>
<td>High-level synthesis</td>
<td>Final evaluation</td>
</tr>
<tr>
<td>HW: Estimation tools</td>
<td>HW: energy estimation tools</td>
<td>Code-size</td>
<td></td>
</tr>
</tbody>
</table>

**FP**

**TLP**

**PP**
Vertical Codesign

- Target Architecture
  - Existing processor platform
  - HW acceleration based on instruction-set extension and coprocessor
- Codesign Approach
  - Evaluation of different configurations
  - Optimization of the HW/SW interface
  - Cosimulation comprising hardware, all software layers and application
Vertical Codesign

**Functional Platform Model:**
- Interacting C++ objects
- SystemC simulation kernel
- Includes: Application, OS, HW

**Architecture Model:**
- HW/SW Mapping
- HW/SW Interface Optimization
- Memory access optimization
- Memory system design

**Transaction-level Model:**
- HW/SW Mapping
- HW/SW Interface Optimization
- Memory access optimization
- Memory system design

**Prototype Platform:**
- Software design
- Software power estimation
- Software power optimization
- Memory access optimization
Horizontal Codesign

- Target Architecture
  - New hardware components, application specific instruction-set processors
  - Optimized hardware for a dedicated application

- Codesign Approach
  - Design of hardware and software layers with regard to the target application
  - Stepwise refinement and cosimulation
Horizontal Codesign

Functional Model

Basic HW Architecture
- OS support
- Application support

Instruction-Set Definition
- OS Interface Optimization
- Application Interface Optimization

Refined Model
- HW&ISA fixed

HW Design Flow
- SW Design Flow

Final System
Design Flow with Security Extension based on Power Profile

- Smart cards store and deal with sensitive data
  - SIM cards in mobile phones
  - e-purse
  - contact-less ID systems
- Security attacks on smart cards
  - invasive or semi-invasive attacks
- Test robustness against attacks
- Attack simulation early in the design process using fault injection
  - ease design changes and
  - insertion of protection mechanisms
- SystemC for high simulation performance
  - can be applied on all SystemC designs
Attack Simulation Flow

Possible Attack

Smart Card Design (SystemC)

Fault Injection Unit

Faulty Smart Card Design (SystemC)

Fault Information

Regular System Behavior

Faulty System Behavior

Simulation & Analysis

Analysis Report

Presented at ATS’04
Fault Injection in Functional Design

- Functional Block
- Functional Block
- Memory 1
- Memory 2
- Fault Injection Control Unit

FIM ... Fault Injection Module
FIP ... Fault Injection Port
Methodology Evaluation

- Evaluation with a Java Card™ Virtual Machine Implementation
- Evaluation Steps:
  - Implementation JCVM functional platform model
  - Vertical Codesign
    - 32-bit Solution based on MIPS Architecture
    - 8-bit Solution based on 8051 Architecture
  - Horizontal Codesign
    - Application Specific Instruction-set Processor
Horizontal Codesign Solutions

- Vertical integration of functional units
- Model comprises virtual machine as well as JC runtime
JAVA Card ASIP Concept

3 Classes of instructions:
- simple byte codes
- instruction set extension
- complex instructions

Security concept:
- User and kernel mode
- different instructions for different memory areas
- large MMU

<table>
<thead>
<tr>
<th>JC API</th>
<th>Java Card System Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>JCVM</td>
<td>Complex Instructions</td>
</tr>
<tr>
<td></td>
<td>Simple Instructions</td>
</tr>
<tr>
<td></td>
<td>Additional Instructions</td>
</tr>
</tbody>
</table>
JAVA Card  ASIP Architecture

Performance comparison

MIPS
- asReturn: 74 bytes
- popFrame: 80 bytes
- pushFrame: 164 bytes

ASIP
- asReturn: 10 bytes
- popFrame: 20 bytes
- pushFrame: 129 bytes

Interpretation
- Code density is much higher
- Microcoded routines run therefor faster (less instructions)
- Specialized hardware gives additional performance boost