Co-scheduling of Software and Hardware Real-Time Tasks for FPGA-based Embedded Systems

Marco Caccamo
Department of Computer Science
University of Illinois at Urbana-Champaign
Performance vs. Flexibility

- Can we increase flexibility of FPGA-based real-time embedded systems?
Example Application: Handheld devices

- Handheld device manufacturers have stringent time-to-market, cost, battery life and printed circuit boards area constraints.

- Handheld devices are subject to soft real-time constraints (mobile phone, mobile TV, interactive games, and other emerging applications).

- A hybrid platform composed by a (soft) CPU and a FPGA can meet the aggressive requirements of handheld electronics:
  1. integrating performance and predictability of hardware tasks with the flexibility of software tasks,
  2. providing longer battery life,
  3. supporting run-time reconfiguration.
Example Application: UAVs (1/2)

- Computing platform on UAVs
- Software task performance might not be sufficient:
  - Ex: optic flow
    - Velocity field is used for navigation and collision avoidance
  - Massively parallelizable computation:
    - 3 fps on 2Ghz general purpose CPU with fair quality, no filtering.
    - 30 fps on 50Mhz, half area prototype FPGA, with good quality and filtering.
Example Application: UAVs (2/2)

- High flexibility is needed to:
  - Adapt to different mission scenarios
  - Provide fault tolerance

- Strict real-time requirements (fail-safe navigation system).

- Platform with a Piccolo Plus avionics box (for navigation) and a reconfigurable SoC (Xilinx Virtex-4) for video processing etc.

- Virtex-4 should be reconfigurable in real-time to provide backup navigation functionalities in case of Piccolo’s failure.
Proposed System Model (1/2)

- Hybrid platform composed of a CPU and Reconfigurable Device (RD) for hardware tasks

- Tasks can be provided in both a software and a hardware configuration.

- We developed a HW-SW co-scheduling framework that:
  1. supports real-time partial reconfiguration,
  2. enables the seamless execution of real-time tasks either in hw or sw configuration,
  3. takes into account all the physical constraints of a modern FPGA (such as the Xilinx Virtex family of FPGAs).
Proposed System Model (2/2)

- Real-time relocation is exploited to cope with dynamic workloads.
- Hardware tasks are allocated on the RD (1D model) and provide improved QoS (improved functionality or higher rates).

- **Objective function**: maximize the number of accepted real-time tasks.
Multiple hardware tasks can run simultaneously on the same device.

An Operating System for Reconfigurable Devices (OSRD) is used to manage the underlying resources and provide run-time support (reconfiguration, communication).
Hardware Tasks and Area Model

- Slotted
  - Subject to internal fragmentation.
  - Complex task decomposition

- 1D
  - Subject to internal fragmentation.
  - Subject to external fragmentation.
Model Limitations: Reconfiguration Time

- Total reconfiguration time for entire SRAM device: 10-100ms
- Ex: assuming $T_{REC} = 50$ ms, $U_{task} = (e + t_{rec}) / \text{period} = \frac{1}{4}$.
- Tasks \{1,2,3,4\} run in slot1, tasks \{5,6,7,8\} run in slot2, etc.
- If HW tasks execute preemptively: minimum possible period is equal to 200 ms (max rate: 5Hz).

![Diagram showing reconfiguration times and slot distribution of tasks with notation $t_{rec} = T_{rec} / \#slots$]
Model Limitations: Relocation

- Relocating a hardware task at arbitrary points is hard.
- State migration is allowed only at job completion.
- Since reconfiguration takes time, special care must be taken to relocate CPU -> RD.
Admission Control

- The goal is to maximize the number of tasks admitted in the system.
- Admission control problem:
  - Tasks are subject to time (deadline) and space (area) constraints.
  - We can admit a new task only if all constraints are guaranteed for already running tasks.

- General strategy:
  1. New tasks are first admitted on the CPU (to minimize admission delay).
  2. Tasks are then relocated to minimize the CPU utilization.

- The problem is decomposed into:
  1. Allocation
  2. Relocation


Allocation Problem

- Given a task set, determine the allocation that minimizes CPU utilization.
- For each task $\tau_i$:
  - $p_i$: period
  - $D_i = p_i$: relative deadline
  - $e_i$: wcet (software configuration)
  - $a_i$: area (hardware configuration)
  - $U_i$: utilization (software configuration)

Software constraint: \[ U = \sum \frac{e_i}{p_i} \leq 1 \]

Hardware constraint: \[ \sum a_i \leq A \]
Allocation Problem

- It is possible to express the problem as an Integer Linear Programming Problem.
- It is equivalent to the well-known 0-1 Knapsack problem.
- Problem is NP-Hard in the weak sense.
- We consider a sub-optimal polynomial algorithm (greedy, $O(N \log N)$ complexity).
  - Order tasks in non increasing order of $\frac{U_i}{a_i}$.
  - Allocate tasks on the RD following the above (non increasing) ordering.
Relocation

- For each task that arrives/terminates in the system:
  1. Compute a new allocation
  2. Relocate tasks to achieve the computed allocation
- Relocation must respect all area and utilization constraints.

\[
\begin{array}{ccccccc}
\tau_1 & \tau_2 & \tau_3 & \tau_4 \\
2/9 & 2/9 & 4/9 & 1/9 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
\tau_5 & \tau_6 & \tau_7 & \tau_3 & \tau_1 \\
3/5 & 2/2 & 2/2 & 3/3 & 4/3 & 2/1 \\
\end{array}
\]

**RD**

**CPU**
Relocation

- For each task that arrives/terminates in the system:
  1. Compute a new allocation
  2. Relocate tasks to achieve the computed allocation
- Relocation must respect all area and utilization constraints.
Relocation

- For each task that arrives/terminates in the system:
  1. Compute a new allocation
  2. Relocate tasks to achieve the computed allocation
- Relocation must respect all area and utilization constraints.
Relocation: How To (Slotted)

1. Partition tasks RD -> CPU and CPU -> RD in swapping groups with the same area.
2. To efficiently create swapping groups, we add a constraint on task area
   - valid task areas should follow a harmonic sequence (e.g., 1,3,6,12,...)
3. Order swapping groups based on total software utilization.
   - H→S: non decreasing software utilization
   - S→H: non increasing software utilization
4. Create pairs of swapping groups (RD -> CPU with CPU -> RD).
5. Swap tasks based on group pairs.
Relocation & Admission Control

- Relocation has low utilization overhead.
- After relocation CPU utilization is minimized.
- Relocation should not introduce additional external fragmentation (1D model).

Theorem 5 ([23]) Under the slotted area model, consider allocations $\mathcal{A}_T^l = \{T_{S \rightarrow S}, T_{S \rightarrow H}, T_{H \rightarrow H}, T_{H \rightarrow S}\}$, $\mathcal{A}_T^u = \{T_{S \rightarrow S}, T_{H \rightarrow S}, T_{H \rightarrow H}, T_{S \rightarrow H}\}$ and associated swapping pairs $\{P^1, \ldots, P^M\}$ and $P^{M+1} = \{S^{M+1}_{H \rightarrow S}, S^{M+1}_{S \rightarrow H}\}$; furthermore, suppose that $P^{M+1}$ is swapped before $P^1$. Then the following are sufficient feasibility conditions to relocate $\mathcal{A}_T^l$ to $\mathcal{A}_T^u$:

1. $U_{\mathcal{A}_T^l} + U_{S^{M+1}_{H \rightarrow S}} \leq 1$;
2. and $U_{\mathcal{A}_T^l} + U_{S^{M+1}_{H \rightarrow S}} - U_{S^{M+1}_{S \rightarrow H}} + U_{S^H_{H \rightarrow S}} \leq 1$;
3. and $U_{\mathcal{A}_T^u} + U_{S^M_{S \rightarrow H}} \leq 1$.
Relocation: How To (1D)

- Same allocation model as slotted.
- Additional placement constraint
  - each HW task is placed at x-position that is multiple of task size.
- Use the same relocation strategy and perform defragmentation.
Relocation: How To (1D)

- Same allocation model as slotted.
- Additional placement constraint
  - each HW task is placed at x-position that is multiple of task size.
- Use the same relocation strategy and perform defragmentation.
Relocation: How To (1D)

- Same allocation model as slotted.
- Additional placement constraint
  - each HW task is placed at x-position that is multiple of task size.
- Use the same relocation strategy and perform defragmentation.
Relocation: How To (1D)

- Same allocation model as slotted.
- Additional placement constraint
  - each HW task is placed at x-position that is multiple of task size.
- Use the same relocation strategy and perform defragmentation.
- Main result: no additional overhead!
Implementation

- Active partial reconfiguration based on Xilinx Virtex-4 devices.
- Device is vertically homogeneous (4 rows).
- Multiple reconfigurable regions on the same column.
Implementation: block diagram

- Bus macro for communication with HW tasks
- Reconfiguration performed by external SysAce chip.
- Future work: use Internal Configuration Access Port (ICAP).
Implementation: floorplan

- HW tasks can be grouped in columns.
- We can route I/O interconnections over reconfigurable regions!
- HW tasks have harmonic sizes (ex: 1, 3, 6, 12).
- Each region uses 8 rows.
- Max size Virtex-4 (LX200, 192rx116c): up to 24 tasks/column.
- Bus macro area overhead: 6 columns.
HW/SW tasks with shared memory architecture

- Shared memory architecture.
- Task state between consecutive jobs is stored in **data (state)** section.
- Task relocation ➔ relocate state section.

### Table: SW access vs. HW access

<table>
<thead>
<tr>
<th>Operation</th>
<th>SW access</th>
<th>HW access</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock/unlock</td>
<td>Semaphores</td>
<td>Semaphores</td>
</tr>
<tr>
<td>Shared data r/w</td>
<td>Shared task data</td>
<td>Shared task data</td>
</tr>
<tr>
<td>Data (state) r/w</td>
<td>Data (DRAM)</td>
<td>Data (HW Task RAM)</td>
</tr>
<tr>
<td>Stack, Heap r/w</td>
<td>Stack, Heap</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Conclusions

- We have introduced a new system model for FPGA-based embedded systems by exploiting real-time relocatable tasks.

- We have proposed an admission control scheme to maximize the number of real-time tasks admitted in the system.

- We have decomposed the problem in two phases: allocation and relocation, and proved hard real-time schedulability bounds.

Future work

- Different objective functions for allocation will be analyzed to incorporate new metrics (rate adaptation, QoS maximization).

- Develop a complete prototype of real-time OSRD that supports relocatable tasks.

- Extend our analysis to 2D models
Simulations

- We compared our admission scheme (with relocation) to a reference baseline scheme (if there is space, admit, otherwise reject).

- Task arrivals and terminations are randomly generated (100,000 tasks each run).

- Offered Load: percentage of area (utilization) of tasks offered to the system (system capacity = 2).

- Two kinds of tasks:
  - Small tasks with area up to 32 and average 3.05.
  - Large tasks with area up to 64 and average 18.14.
Simulations: Small Tasks

\[ \text{Load}(\Delta) = \frac{\sum U_i T}{\Delta} = \frac{\sum \frac{a_i}{A} T}{\Delta} \]
Simulations: Large Tasks

\[ \text{Load}(\Delta) = \frac{\sum U_i T}{\Delta} = \frac{\sum \frac{a_i}{A} T}{\Delta} \]
Simulation Results

- The relocation scheme improves the number of accepted tasks substantially.
- Performance is similar between the optimal and greedy allocation, however:
  - Mean time for greedy algorithm: 243 us.
  - Mean time for optimal algorithm: 151 ms.
Relocation: Theoretical Results

- Overhead is equal to the smallest utilization of any swapping group.
- We propose algorithm **MIN_SWAPPING** to minimize the size of the smallest group.
- Complexity is $O(N^2)$.
Design flow

- Task Source code
  - Custom preprocessor
  - Behavioral synthesis
    - RTL code (user logic)
      - Map, Place & route
        - Placed & routed HW task
          - Relocatable HW task
          - Partial bitstream
        - VHDL
          - Bus Interface Module
            - VHDL
  - VHDL / Verilog
    - Behavioral synthesis
      - Place & route
        - Relocatable HW task
          - Relocatable HW task
          - Partial bitstream
        - System Bitstream (static+tasks)
          - System Bitstream (static+tasks)
          - Memory merger
            - Memory annotated bitstream
  - SW compiler
    - Object code
      - SW platform Configuration & libraries
        - Software Image
          - linker
            - OS image (uClinux)
              - OS image (uClinux)