ARCHITECTURE-AWARE MAPPING AND SCHEDULING OF IMA PARTITIONS ON MULTI-CORE PLATFORMS

Aishwarya Vasu (1), Harini Ramaprasad (2)
(1) Southern Illinois University Carbondale
(2) University of North Carolina at Charlotte
INTEGRATED MODULAR AVIONICS

• Deploy multiple software functions with different criticality levels on single CPU
IMA PARTITIONS ON SINGLE CPU HARDWARE

- Results in bulky system with high power consumption
- To improve Size, Weight & Power considerations
  - Deploy multiple IMA partitions on one multi-core platform
ARCHITECTURAL ASSUMPTIONS

• Identical cores
• Private data cache with support for line level locking
• Cores connect to main memory via shared bus
  • Time Division Multiple Access arbitration policy on shared bus
• Data concentrator device on each core to support asynchronous communication
PARTITION AND TASK MODEL

Partition $P_i$

Local Scheduler

$\tau^1_i$, $\tau^2_i$, $\tau^n_i$

Task, $\tau^j_i$

$p_i \Rightarrow$ Activation Period
$s_i \Rightarrow$ Activation Window
$\chi_i \Rightarrow$ Criticality Level
$\Gamma_i \Rightarrow$ Task set
$U_i \Rightarrow$ Utilization

$T_{ij} = $ Period
$C_{ij} = $ Worst Case Exec time
$D_{ij} = $ Relative deadline
Partition and Task Scheduling

Partition P1

\( \tau_1^1 \) \( \tau_1^2 \) \( \tau_1^3 \)

Partition P2

\( \tau_2^1 \) \( \tau_2^2 \) \( \tau_2^3 \)

Activation period for both P1 and P2

P1 Activation Window

P2 Activation Window

\( t = 0 \) \( 5 \) \( 12 \)
OBJECTIVE

• Develop algorithm to map IMA partitions onto multi-core platform when:
  • High criticality partitions may communicate (asynchronous)
  • High criticality partitions may load and lock specific content in core’s private cache
  • Certain partition pairs cannot be allocated to the same core
    • Partition exclusion property
  • May Arise out of Security, Safety and Criticality Considerations or based on Risk Analysis

Cache requirements: \{ <SA, ne, freq > \} Provided by system integrators
Weight-based approach:

- **PE<sub>i</sub>** - Set of pairwise Partition Exclusion weights
  - Reflect safe or unsafe allocation of partition combinations
  - *Assumed to be provided by system integrators*
- **CO<sub>i</sub>** - Set of pairwise weights for partition P<sub>i</sub>
  - Reflect degree of communication with other partitions
- **CA<sub>i</sub>** - Set of pairwise weights for partition P<sub>i</sub>
  - Indicate degree of cache conflicts with other partitions
- Resultant Weight (ρ<sub>ij</sub>) calculated for every partition pair P<sub>i</sub>, P<sub>j</sub>
  - Indicates how suitable it is to allocate P<sub>i</sub> and P<sub>j</sub> on same core
• Two Phases:
  • Preprocessing Phase:
    • Extract & sort Strongly Connected Components (SCCs)
    • Derive pair-wise weights, core threshold weight

• Allocation and Scheduling Phase:
  • Allocate partitions based on resultant weight between partition pairs
PREPROCESSING PHASE – SCC EXTRACTION AND SORTING

- Extract Strongly Connected Components (SCCs)
  - \(< SCC_{id}, U_{scc}^{id}, L_{scc}^{id} >\)

- Sort SCCs
  - To help in keeping communicating partitions together
  - Improves Schedulability
## PREPROCESSING PHASE – SCC SORTING STRATEGY

<table>
<thead>
<tr>
<th>SCC Sorting Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration 1</td>
<td>SCCs are kept in increasing order of IDs; the partitions within each SCC are kept in the order in which they were added to the SCC.</td>
</tr>
<tr>
<td>Configuration 2</td>
<td>SCCs are sorted in non-increasing order of criticality; the partitions within each SCC are kept in the order in which they were added to the SCC.</td>
</tr>
<tr>
<td>Configuration 3</td>
<td>SCCs are sorted in non-increasing order of utilization; the partitions within each SCC are kept in the order in which they were added to the SCC.</td>
</tr>
<tr>
<td>Configuration 4</td>
<td>SCCs are sorted in non-increasing order of criticality; partitions within each SCC are sorted in non-increasing order of utilization</td>
</tr>
<tr>
<td>Configuration 5</td>
<td>SCCs are kept in increasing order of IDs; then a DAG traversal is performed on the SCC Acyclic graph</td>
</tr>
<tr>
<td>Configuration 6</td>
<td>SCCs are sorted in non-increasing order of criticality; then a DAG traversal is performed on the SCC Acyclic graph</td>
</tr>
<tr>
<td>Configuration 7</td>
<td>SCCs are sorted in non-increasing order of utilization; then a DAG traversal is performed on the SCC Acyclic graph</td>
</tr>
<tr>
<td>Configuration 8</td>
<td>SCCs are sorted in non-increasing order of criticality and utilization; then a DAG traversal is performed on the SCC Acyclic graph</td>
</tr>
<tr>
<td>Configuration 9</td>
<td>Isolated vertices on the SCC Acyclic graph is found and pushed to the end of the sorted list, to allocate communicating SCCs first</td>
</tr>
</tbody>
</table>
PREPROCESSING PHASE – DERIVATION OF CO$_i$

- Define Communication Weight between partition pairs:
  - $CO_{ij} = < co_{ij}, cost_{ij} >$
  - $co_{ij} = \begin{cases} 1, & \text{if } P_i, P_j \text{ communicate} \\ 0, & \text{otherwise} \end{cases}$

\[
\begin{align*}
\text{cost}_{i,j} &= \left[ \frac{n_{i,j}^{\text{trans}}}{n_{i,j}} \right] \times m_{\text{tx}}^{\text{latency}} \\
\text{where} & \\
& \quad n_{i,j} : \text{number of bytes transferred from partition } P_i \text{ to } P_j \\
& \quad n_{i,j}^{\text{trans}} : \text{number of bytes transferred per transaction} \\
& \quad m_{\text{tx}}^{\text{latency}} : \text{communication latency incurred per transaction}
\end{align*}
\]
PREPROCESSING PHASE – DERIVATION OF $C_{A_1}$

- Bipartite graph constructed
  - Partitions on top
  - Groupings of cache sets on bottom
- Edge weight
  - Represents number of cache lines that partition tries to lock in that group of cache sets
- A partition pair cannot have cache conflict if one of two conditions is satisfied:
  - No cache set that both partitions try to lock
  - Every cache set that both partitions try to lock has less incoming edges than capacity of set
- Cache Conflict Weight
  
  $C_{A_{i,j}} = \frac{(\text{Lines}_{\text{total}} - \text{Lines}_{\text{conflict}}_{i,j})}{\text{Lines}_{\text{total}}}$

  - $\text{Lines}_{\text{total}}$: Total number of lines in cache
  - $\text{Lines}_{\text{conflict}}_{i,j}$: Number of conflicting lines in cache for $P_i$ and $P_j$
**ALLOCATION PHASE - OVERVIEW**

- **Goal:** Find number of cores needed to allocate partition set

- **Two Schemes**
  - **NCU Scheme:**
    - Strict consideration of Communication, PE and Cache requirements
    - Partitions with potential cache conflicts allocated on different cores
  - **CU Scheme:**
    - Consideration of Communication and PE requirements
    - Cache requirements relaxed → allow conflicting partitions on same core if needed
    - Subset of conflicting lines are *unlocked* by one partition
    - Results in increase of utilization
ALLOCATION PHASE – HIGH CRITICALITY PARTITION ALLOCATION

- Allocate High Criticality Partitions based on weights
  - Define Core Threshold Weight, $\Omega$
    - Based on recommended weight for individual factors (provided by system integrators)
  - Partition pairs with resultant weight $\rho_{ij} \geq \Omega$ can be allocated on same core
  - For every partition:
    - Compute resultant weight on all cores (i.e., try allocating partition on each core)
    - Get information on actual cache conflicts
    - Remove cores with resultant weights less than Core Threshold Weight, $\Omega$
    - Sort remaining cores in non-increasing order of resultant weights
ALLOCATION PHASE – HIGH CRITICALITY PARTITION ALLOCATION

• Iterate over sorted cores
  • Compute communication costs if needed
  • Check schedulability of partitions that had change in utilization due to communication
    • Compute activation window, activation period
    • Based on an existing work in hierarchical scheduling
  • If successful, allocate partition to core and end iteration
• If core not found, next steps depend on CU / NCU scheme

ALLOCATION PHASE – HIGH CRITICALITY PARTITION ALLOCATION

• NCU Scheme:
  • “Add” new core to system
  • Allocate partition to new core if possible after accounting for communication costs
• CU Scheme:
  • Compute cache conflict latency for all partitions conflicting with P_i
    • Update Partition utilization
  • Sort cores in non-decreasing order of their change in utilization
  • Re-try cores and check schedulability
• If no core found
  • P_i deemed to be non-schedulable
  • Cache unlocking and utilization changes are reverted to previous values
• Allocated using Worst-Fit heuristic
• Sort partitions in non-increasing order of criticality and utilization
• For every partition $P_i$
  • Sort cores in non-increasing order of available utilization
  • Try core with maximum available utilization
  • “Add” new core if core with maximum available utilization cannot fit partition $P_i$
SIMULATION SETUP – PARTITIONS & TASKS

• Multiple partition utilization caps - 0.2, 0.3, 0.4, 0.5, 0.6 - considered
• For each cap, 100 sets of different partition and task characteristics generated
• Random directed weighted cyclic graph generated for communication between high criticality partitions
  • Degree of Communication (DoC): (0% - 25%), (25% - 50%)
• Random memory footprints generated for high criticality partitions
• Random Partition Exclusion weights generated between high criticality partitions
• Identical cores
• Private data cache on each core

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache line size</td>
<td>32 B</td>
</tr>
<tr>
<td>Element size</td>
<td>16 B</td>
</tr>
<tr>
<td>Associativity</td>
<td>1 (32 KB)</td>
</tr>
<tr>
<td></td>
<td>2 (64 KB)</td>
</tr>
<tr>
<td></td>
<td>4 (128 KB)</td>
</tr>
<tr>
<td></td>
<td>8 (512 KB)</td>
</tr>
<tr>
<td></td>
<td>16 (1 MB)</td>
</tr>
<tr>
<td>Memory Access latency</td>
<td>50 cycles</td>
</tr>
</tbody>
</table>
COMPARISON OF AVERAGE NUMBER OF CORES BETWEEN NCU AND CU SCHEMES: DOC = (0%-25%): UTIL CAP = 0.2

- **NCU**
  - More cores required to host partitions for 1 way set-associative cache configuration
  - Reason: increased number of cache conflicts

- **CU Scheme**
  - Tries to accommodate partitions by unlocking conflicting cache lines
  - Uses a less number of cores when compared to NCU scheme
  - When cache ways are increased, average number of cores decreases
  - Reason: reduced number of cache conflicts
COMPARISON OF PERCENTAGE ALLOCATION OF PARTITION SETS BETWEEN CU AND NCU SCHEMES

- For lower $\hat{U}$, (0.2, 0.3 and 0.4)
  - Configs 1 - 4 schedule lower percentage of partition sets than Configs 5 - 9
  - Configs 1 - 4 do not keep communicating partitions together unless they are within same SCC

- Beyond 1way cache configuration, no significant difference between performance of CU & NCU schemes
  - Although there are potential cache conflicts between partitions, not all of them manifest as actual conflicts even in NCU scheme
EFFECT OF DEGREE OF COMMUNICATION ON ALLOCATION – CU SCHEME: COMPARISON BETWEEN DOC = 0_25% AND DOC = 25_50%

Partition Utilization cap = 0.2

- As DoC is increased, % of successfully allocated partition sets decreases
- Change in % allocation with increased communication is higher for lower $\hat{U}$
  - More number of partitions for lower $\hat{U}$ => more communicating partitions => increased communication cost
EFFECT OF DEGREE OF COMMUNICATION ON ALLOCATION – CU SCHEME:
COMPARISON BETWEEN DOC = 0_25% AND DOC = 25_50%

Partition Utilization cap = 0.6

- As $\bar{U}$ increases
  - Lower number of partitions in a set => Lower communication => DoC less significant
similar trend observed for NCU scheme
**EFFECT OF DEGREE OF COMMUNICATION ON ALLOCATION – NCU SCHEME:**
**COMPARISON BETWEEN DOC = 0_25% AND DOC = 25_50%**

**Partition Utilization cap = 0.6**

<table>
<thead>
<tr>
<th>% ALLOCATION OF PARTITIONS</th>
<th>0_25</th>
<th>25_50</th>
<th>1 WAY</th>
<th>0_25</th>
<th>25_50</th>
<th>2 WAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config1</td>
<td>100</td>
<td>98</td>
<td>98</td>
<td>100</td>
<td>98</td>
<td>98</td>
</tr>
<tr>
<td>Config2</td>
<td>100</td>
<td>99</td>
<td>99</td>
<td>100</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>Config3</td>
<td>98</td>
<td>98</td>
<td>98</td>
<td>98</td>
<td>98</td>
<td>98</td>
</tr>
<tr>
<td>Config4</td>
<td>100</td>
<td>99</td>
<td>99</td>
<td>100</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>Config5</td>
<td>100</td>
<td>99</td>
<td>99</td>
<td>100</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>Config6</td>
<td>100</td>
<td>99</td>
<td>99</td>
<td>100</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>Config7</td>
<td>100</td>
<td>99</td>
<td>99</td>
<td>100</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>Config8</td>
<td>100</td>
<td>99</td>
<td>99</td>
<td>100</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>Config9</td>
<td>100</td>
<td>99</td>
<td>99</td>
<td>100</td>
<td>99</td>
<td>99</td>
</tr>
</tbody>
</table>

**COMPARISON OF PERCENTAGE ALLOCATION FOR DIFFERENT DEGREE OF COMMUNICATION FOR NCU SCHEME, PARTITION UTIL MAX = 0.6**

**DEGREE OF COMMUNICATION 25% TO 50% AND 50% TO 75%**
- SCC SORTING CONFIGURATIONS / CACHE CONFIGURATIONS

* Similar trend observed for NCU scheme for higher $\bar{U}$
CONCLUSIONS AND FUTURE WORK

• Outcome → design space exploration tool – useful during system integration phase

• **Allocation of partitions is impacted by:**
  • Order in which partitions are chosen for allocation
  • Degree of Communication (DoC) among partitions

• **Future Work:**
  • Enhance cache conflict generator to conduct sensitivity studies and observe how increasing conflicts affect our algorithm’s performance
  • Consider allocation and scheduling of partitions that share software resources