The context

General Purpose processor architectures have the following memory hierarchies:
- L1 cache
- L2 cache
- Main memory

- Difficult to estimate cache behavior as we saw during WCET analysis
- Can we tackle the problem at a system level as well?
  - Develop an on-chip memory whose contents will be fixed during program execution
  - Conceptually a locked cache, called scratch-pad memory
  - Compiler controlled memories!

Cache

- Processor-memory performance gap → memory optimizations

- Cache:
  - hardware-managed
  - access determined during runtime
  - unpredictable timing
  → problematic for hard real-time systems

- Scratchpad
  - On chip memory, more predictable than cache

Scratch-pad

- Scratchpad Memory
  - software-managed
  - access: pre-defined address range
  - lower die area, energy consumption (Banakar et al., 2002)
  - job shifted to compiler

- An alternative to or on top of cache
Address Space

- SRAM (on chip) with 1 cycle access
- DRAM (off chip) with N cycles access

Allocation Strategy

- Allocate scalars to scratch-pad
- For arrays:
  - Find out which arrays overlap in life-time, and
  - Result in cache conflicts (if mapped to cache)
  - Map one of them to scratch-pad
- The above strategy has a caveat:
  - Which array results in cache conflict requires a program trace
  - Not based on static program analysis
  - This problem is hard even when we rule out the data cache (quite common in Real-time systems)

Simplified Address Space

- Scratchpad
- On-chip
- Main Mem (Off-chip)

ACET based Allocation

- Profile the program for selected inputs
  - Build up memory access profile
- Allocate heavily accessed variables
  - Optimal solution via 0-1 Knapsack
- If inputs are representative
  - Aims to reduce Average-Case Execution Time
- In real-time systems WCET is a key metric

Knapsack formulation

- Possible for ACET based allocation
  - Items: \( V_1 \cdots V_{|\text{allvars}|} \)
  - Capacity: scratchpad space
  - Weight of each variable \( u \in \{V_1 \cdots V_{|\text{allvars}|}\} = \text{area}_u \)
  - Gain of each variable \( u \in \{V_1 \cdots V_{|\text{allvars}|}\} \)
    - Constant for a given path
    - Not constant for WCET based allocation

ACET vs WCET based
Difficulty in WCET-based allocation

Allocate V

Path1  | Path2
---|---
Before | 90 | 100
After  | 90 | 80

Contribution of V to WCET path = 20
Reduction in WCET by allocating V = 10

Path2 is now the WCET path which has different var access frequencies from Path1

Sub-optimal (local) allocation

V, U both appear in WCET path. Suppose #V > # U

Path1  | Path2
---|---
Before | 90 | 100
After  | 90 | 80

WCET path based allocation = \{V\}
Optimal allocation = \{U\}

Path1  | Path2
---|---
Before | 90 | 100
After  | 75 | 85

Path1 is now the WCET path which has different var access frequencies from Path2

Summary of bad news

- Optimal WCET based allocation
  - Must not be profile-guided (WCET path).
  - Cannot take WCET contribution of variables as constant.
    - Rules out Knapsack like solutions.
  - Should ideally be a global optimization procedure aware of infeasible paths.
    - Paths in CFG not executed on any program input.

Why not take ACET-based?

ACET based allocation should also reduce WCET, after all.
Yes, but ...

- 46% additional WCET reduction
- Varies in other benchmarks, but ~20%

Allocation methods

- Integer Linear Programming (ILP)
  - Cannot cater for infeasible paths.
- Branch and Bound (BnB)
  - High Complexity.
- Greedy Heuristic
  - Diff. from greedy allocation using WCET path.
- Experiments
  - WCET reduction, Running time.
**ILP Formulation**

\[
\sum_{v \in \text{allvars}} S_v \cdot \text{area} \leq \text{scratchpad size}
\]

\[
\forall v \in \text{allvars}, \ S_v \geq 0, \ S_v \leq 1
\]

**Allocation methods**

- **Integer Linear Programming (ILP)**
  - Cannot cater for infeasible paths.
- **Branch and Bound (BnB)**
  - Rule out easy solutions - ACET based alloc.
  - High Complexity of BnB.
  - Greedy Heuristic
    - Diff. from greedy allocation using WCET path.
  - Infeasible path detection
- **Experiments**
  - WCET reduction, Running time.

**WCET est. with/without infeasibility checking**

**Knapsack?**

- Possible for ACET based allocation
  - Items: \(v_1 \ldots v_{\text{allvars}}\)
  - Capacity: scratchpad space.
  - Weight of each variable \(u \in \{v_1 \ldots v_{\text{allvars}}\} = \text{area}_u\)
  - Gain of each variable \(u \in \{v_1 \ldots v_{\text{allvars}}\}\)
    - Constant for a given path.
    - Not constant for WCET based allocation.
Re-cap on Knapsack problem

- Given n objects and a knapsack
  - Capacity of knapsack W
  - Object i has weight w_i and value v_i
  - Fill up the knapsack so as to maximize value
- Perfect fit for our allocation problem if the gain by allocating a variable to scratchpad memory is a constant
  - Holds for ACET based allocation
  - Not true for WCET based allocation
- Knapsack problem can be easily solved by dynamic programming.

Dynamic Programming

- Array V[0..n,0..W]
- V[i,j] = maximum value if we are restricted to objects 1..i, and the weight limit is j
- Define
  - V[i,j] = max(V[i-1,j],V[i-1,j-w_i] + v_i) for i > 0
  - V[0,0] = 0
- Final Answer V[n,W]
- This solution is useful only for ACET based allocation.

For WCET-based

Knapsack?
- gain in WCET reduction due to a variable v
  - not a constant, depends on
    - current WCET path
    - Diff in exec. Times of WCET path and other paths
    - # Occurrence of v in WCET path and other paths

Search tree

But, cannot search it exhaustively.

Branch-and-bound
How to estimate the upper bound?

- For partial allocation $V$ at level $k$,
  - $V \subseteq \{ v_k, \ldots, v_n \}$
  - $UB_v = (WCET reduction due to V) + \max_x$,
  - $\max_x$ computed as knapsack problem:
    - items: $v_{k+1}, \ldots, v_n$.
    - weight of each variable $u = (v_{k+1}, \ldots, v_n)$, area.
    - gain of each variable $u = (v_{k+1}, \ldots, v_n)$ = maximum contribution of $u$ towards WCET of any path.
      - Can be obtained by bottom-up pass of syntax tree.

Efficiency Issues

- Search exponential in #vars. even after pruning.
- Efficient heuristics with near-optimal allocation.
- Do not only consider the WCET path.
- WCET est. for a given scratchpad allocation:
  - Invoked several times by Branch-and-bound.
  - Needs to consider Infeasible path information.
    - Infeasible pairs of assgn/branch in a loop iteration.
    - Must run very fast (we do not discuss it here).
      - Find heaviest iteration in a loop considering this info.
      - Avoid backtracking without path enumeration.

Greedy Heuristic

- Initially:
  - No variable allocated.
- Iteratively:
  - Identify current WCET path $P$ (considering current allocation).
  - Allocate variable with max. gain in $P$ that can fit into remaining space in scratchpad.
- Stop when:
  - scratchpad is filled, or
  - no more variable can be allocated from current WCET path.

Notes on sub-optimality

- Becomes close to WCET path based allocation if
  - Scratchpad is very small.
  - One of $U, V$ can be allocated.
  - Less overlap among vars. appearing in diff. paths.
  - $V$ does not appear in Path1.

Allocation methods

- Integer Linear Programming (ILP)
  - Can start for infeasible paths.
- Optimal with infeasible path info.
  - Branch-and-Bound
    - High Complexity.
    - Greedy Heuristic: Diff. from greedy allocation using WCET path.
    - Infeasible path detection.

Other Works

- Scratchpad allocation for data memory
  - WCET-based.
  - Easily extensible to:
    - Variables with disjoint lifetimes.
    - Code memory.
- Extensions to Multi-processor SoCs
  - Allocate to reduce bus contention among processors.
- ACET-based allocation can be modified to get scratch-pad allocation to reduce energy for example --- simple change.
- Any such allocation must be combined with WCET analysis & infeasible path detection.