

Specific Issues Related to Embedded Processor Architectures

Sorin Manolache

sorma@ida.liu.se



Outline

- General approach
- SIMD example
- VLIW example



Programmable Architectures of Interest (1)

- Microcontrollers (MCU)
 - CISC
 - High code density
 - Reduced computational resources
- RISC processors
 - Large file of GP registers
 - Orthogonal instruction set
- DSP
 - DSP-specific data path architectures (irregular)



Programmable Architectures of Interest (2)

- Multimedia processors
 - VLIW
 - High performance
 - Low code density
 - High power consumption
- Application-Specific Processors (ASIP)
 - Application-specific data paths
 - Sometimes customizable (reg. file size, reg. width, word width)



Optimization Levels

- Source level optimization
- Optimized instruction set mapping
- Assembly level optimizations



Source Level Optimization

- Architecture independent
 - Constant folding
 - Common subexpression elimination
 - Jump optimization
- Address code transformations
 - Attempt to optimize address generation for array indexes (up to 50% code for addr. gen.)
- Loop transformations
 - Loop unrolling
 - Loop folding (software pipelining)
- Function inlining

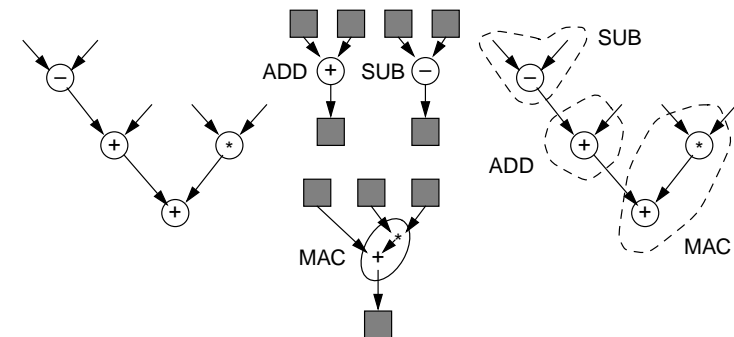


Optimized Instruction Set Mapping

- Mapping of machine-independent intermediate representation (IR) to matching instructions
- Consider:
 - Special-purpose registers
 - Complex instruction patterns (MAC, SIMD)
 - Inter-instruction constraints (resource)
- Phases:
 - Tree pattern matching (mapping of “functionality”)
 - Register allocation (mapping of data)
 - Instruction scheduling (multiprocessor scheduling, with dependency and communication constraints)



Tree Pattern Matching (1)



Tree Pattern Matching (2)

- Optimal cover in linear time with dynamic programming
- There are compiler generators for such problems
- Limitations:
 - Only for DFT, not DFG
 - Difficult to apply for irregular data paths
 - Doesn't apply to SIMD processors



Register Allocation and Instruction Scheduling

- Graph colouring problems (heuristics)
- Simulated annealing, ILP, CLP
- Cost function, number of register spillings
- Better results when applied concurrently with instruction scheduling (*phase coupling*)
- Mutation scheduling, list scheduling
- Algebraic transformations on operations



Assembly Level Optimization

- Goals:
 - Memory access optimization
 - Instruction scheduling optimization
- Consider:
 - Memory architecture (many DSP have two memory banks)
 - Address generation hardware
 - Instruction level parallelism
- Fewer pipeline stalls
- Speed-ups of 24% reported



Even More Problems

- Compiling for low-power (Siemens reported 60% power saving for a mobile phone in stand-by mode achieved only from software!)
- Retargetability
- Industrial trends: VLIW



Compilation for SIMD (1)

- Authors go for an ILP approach
- Write a grammar for tree covers
- A derivation in this grammar is a possible cover
- A cost is associated to each rule

DFT: **store**(REG, **plus**(REG, OFFS))

REG: **plus**(REG, REG)

REG_LO: **plus**(REG_LO, REG_LO)

REG_HI: **plus**(REG_HI, REG_HI)

- same cost associated to the three rules for covering an ADD.
- The problem is to find a derivation with an optimal cost



Compilation for SIMD (2)

- Constraints:
 - Each node in the DFT has to be covered by exactly one machine operation
 - Result destination of an operation have to be the same as operand sources for result consumers
 - Common subexpressions
 - SIMD selection: same operation, correct alignment in memory, correct placing relative to the SIMD register, no dependency
 - Data dependency constraints



Code Generation for Irregular Data Paths

- Authors go for a CLP approach
- Representation by means of a *factored machine operation*

(Op, R, [O₁, O₂, ..., O_n], ERI, Cons)
- Works even if **Op** is not available on the target processor (algebraic transformations)
- Able to model chained operations (MACs), large class of restrictions on instruction-level parallelism
- For the most complex models, performance improvement to up to 50%, at the expense of sometimes 24 hours of compilation

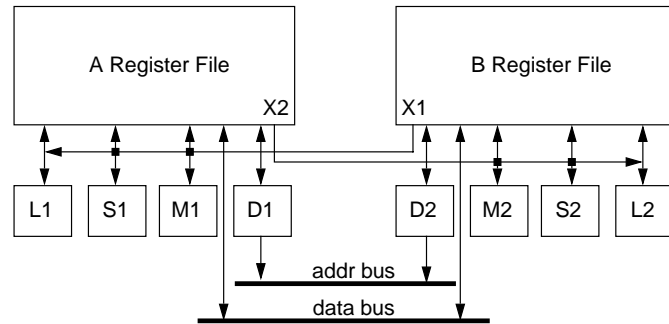


Compilation for VLIW

- Consider only the partitioning (mapping) and scheduling phases (code selection is considered to be done)
- Done concurrently by feeding back the results of the scheduling phase to a new iteration of the mapping phase
- Mapping by means of a simulated annealing approach
- Scheduling based on a list scheduling approach



The Problems Come from the Architecture



Scheduling Algorithm (1)

```

mark all nodes as not scheduled
S = empty_set
while not scheduled nodes
    m = NEXT_READY_NODE();
    S = SCHEDULE_NODE(S, m, P);
    mark m as scheduled
  
```

- NEXT_READY_NODE selects the node that can be scheduled the earliest
- ties are broken by selecting the node that has the smallest ALAP time (first_fail in CLP)



Scheduling Algorithm (2)

```

cs = EARLIEST(m) - 1;
repeat
    cs = cs + 1;
    fm = GET_NODE_UNIT(m, cs, P);
    if (fm == 0) continue;
    if (m has an argument in a diff. cluster)
        CHECK_ARG_TRANSFER();
        if (at least one transfer impossible)
            continue;
        SCHEDULE_TRANSFERS();
    until m is scheduled
    if (m is a LOAD)
        DETERMINE_LOAD_PATH(m)
    if (m is a CSE) INSERT_FORWARD(S, m);
  
```



Scheduling Algorithm (3)

- Use copies
- If no copies, use cross path rather than inserting a move instruction in an earlier step
- Exploit commutativity
- Approach best when the ratio of the length of the critical path and the number of instructions is low



Conclusions

- Generally difficult problem
- Additionally, difficulty very much dependent on the particular problem instance => difficult to create a retargetable compiler
- Heuristics, but which? General (simulated annealing)? Problem specific (CP)?

