Towards a Low Power Virtual Machine for Wireless Sensor Network Motes

Hitoshi Oi

The University of Aizu

C. J. Bleakley

University College Dublin

November 17, 2006

FCST 2006, Aizu Wakamatsu, JAPAN

Wireless Sensor Network

- A network of tiny devices (motes) consisting of
  - a microcontroller (CPU)
  - wireless interface
  - battery
  - memory
  - sensors (e. g. temperature, sound)
- Possibly thousands of motes are deployed over a wide area and used for various applications such as habitat, environmental or traffic monitoring.
- Once deployed, reprogramming motes are difficult

Virtual Machine for WSNs

- Customized and uniform programming model
  - motes can be heterogeneous
  - choose instruction set and library for target applications
- Robust and Secure Platform that protect systems from malicious and buggy programs
- Concise program footprints
  - smaller memory size
  - shorter radio communication for program update over the network

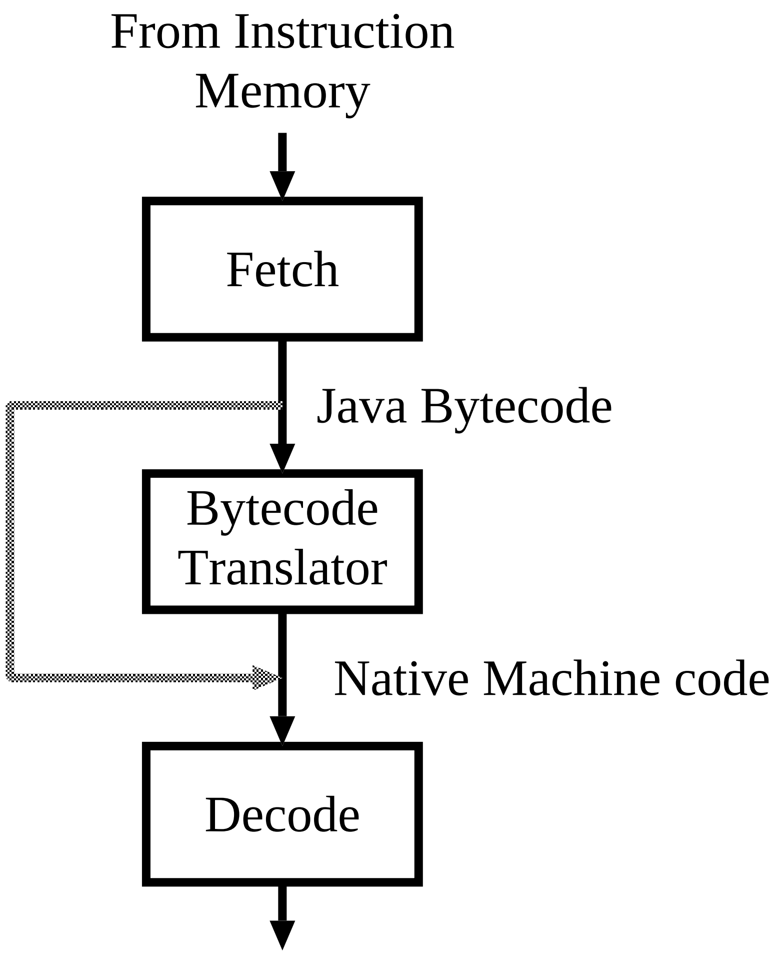
Execution Overheads of VMs

- Stack architecture on a register-based processor (redundant operations)
- Every few instructions executed as a TinyOS task (scheduling overhead)
- Polymorphic operands

These overheads make VM approach only suitable for applications frequently updated but infrequently executed.

Hardware Translation in Emb. JVM

- Mainly concerned with filling the gap between stack-based and register-based architectures.
- Small logic module translates Java Bytecodes into ARM instructions
- Translation is in on-the-fly single bytecode basis (no extra storage of translated code).



Virtual Machine WSNs

JVM

- Interactive system (response time)
- Relatively larger memory size
- Various and complex applications

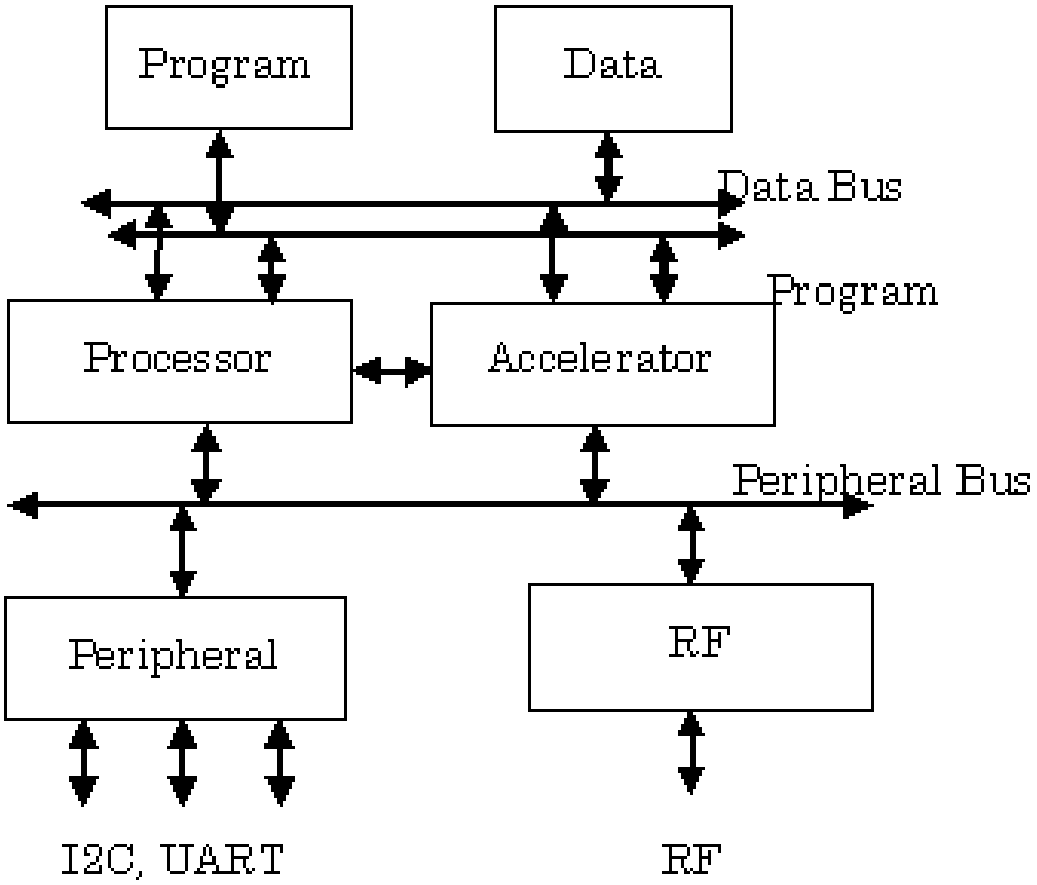
WSNVM

- Autonomous systems (power consumption)
- Smaller memory size
- Simple tasks (e.g. aggregation of sensor data)

Hardware Accelerator Approach

- Identify frequently executed functions in WSN (TinyOS) applications
- Investigate the feasibility of implementing these function by hardware modules (hardware accelerator)
- Execution of these functions by the simpler hardware modules should result in shorter running time and also lower power consumption.
- The hardware accelerator can also be used for reducing the execution overhead of VM execution.

WSN Node Architecture



Optimization Target (1) Synchronization

Operation	Cycles	Time (μs)
Lock	32	8
Unlock	39	10
Check Runnability	929	232
Run	1077	269
Resume	2038	510
Analysis	15158	3790

(From UCB//CSD-04-1343)

Check Runnability check if all resources available

Analysis extract resource usage in the capsule

Optimization Target (2) Stack Operation

Folding 'Simple' Instructions

- Example: `pushc6` + `add` → `addc6`
- Reduce the number of VM instructions to be executed effectively
- Reduce the task invocation overhead
  - Small number* of bytecodes are invoked as a task in Maté
  - * `MATE_CPU_QUANTUM` = 1 (ver. 1) or 5 (ver 2.19a)
- Should be done during the capsule analysis

Optimization Target (3) Polymorphic Operands

Some Instructions Take Multiple Operand Types

Example: `add`

`add Integer Integer` → `Integer` (Arithmetic Sum)

`add Message Integer` → Append `Integer` to Message

`add Message Message` → Message Concatenation

- Capsule analysis should disambiguate operand types
- Rewrite each `add` into type-specific `add` (such as `addII`, `addMI` or `addMM`)
- Eliminate instructions for type information retrieval and conditional branches

Polymorphic Operands Example: add

```
if ((arg1→type == MATE_TYPE_INTEGER)
  && (arg2→type == MATE_TYPE_INTEGER)) {
  call Stacks.pushValue(context, arg1→value.var
    + arg2→value.var);
}
else if (arg1→type == MATE_TYPE_BUFFER) {
  if (arg2→type ≠ MATE_TYPE_BUFFER) {
    call Buffer.append(context, arg1→buffer.var, arg2);
  }
  else {
    call Buffer.concatenate(context, arg2→buffer.var,
      arg1→buffer.var);
  }
}
```

Customizable Instruction Set (1)

- In Maté ver 2.19a, the instruction set is fully customizable.
- This is possible for Maté since it is implemented by software-only approach, but not an easy job for hardware assisted VMs.
- However, fully-customizable instruction set may not be necessary because:
  - Primitive arithmetic/logical instructions are needed for any applications (e.g. `add`)
  - Some instructions are mandatory for stack-based architecture (e.g. `push`)
  - Complex operations should be implemented as library functions rather than instructions

Customizable Instruction Set (2)

Our Approach

- Pre-refine commonly used instructions types
  (Foldable instructions are identified based on these types)
- Some parameters can be defined by users
  (such as # bits in constant or variable index field)
- Provide abundant library functions for the application target and let users select (for example) 256 out of them, so that users can invoke functions with a one-byte parameter.

Current and Future Work

- Identification of benchmark programs and usage model for WSN VMs
  - various application field of WSNs
  - VM approach for WSN is relatively new
  - Not only applications themselves but also program update frequency are important for the effectiveness of VM approach a
- Development of an analytical power consumption model for our WSN VM approach for initial evaluation of the approach.
- Development of a more detailed model and combining it with a TinyOS simulator (e.g. AVRORA).

Thank You !

Any Question ?