Hardware Support for a Wireless Sensor Network Virtual Machine

Hitoshi Oi

The University of Aizu

February 13, 2008

Mobilware 2008, Innsbruck, Austria

Outline

- Introduction to the Wireless Sensor Network and Virtual Machine
- Execution Overhead of Virtual Machine
  - Hardware Translation in JVM
  - WSN VM and JVM Comparison
- Hardware Support in WSN VM
  - Operand Stack Module
  - Synchronization Module
  - Results from source code analysis
- Conclusions and Future Work

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

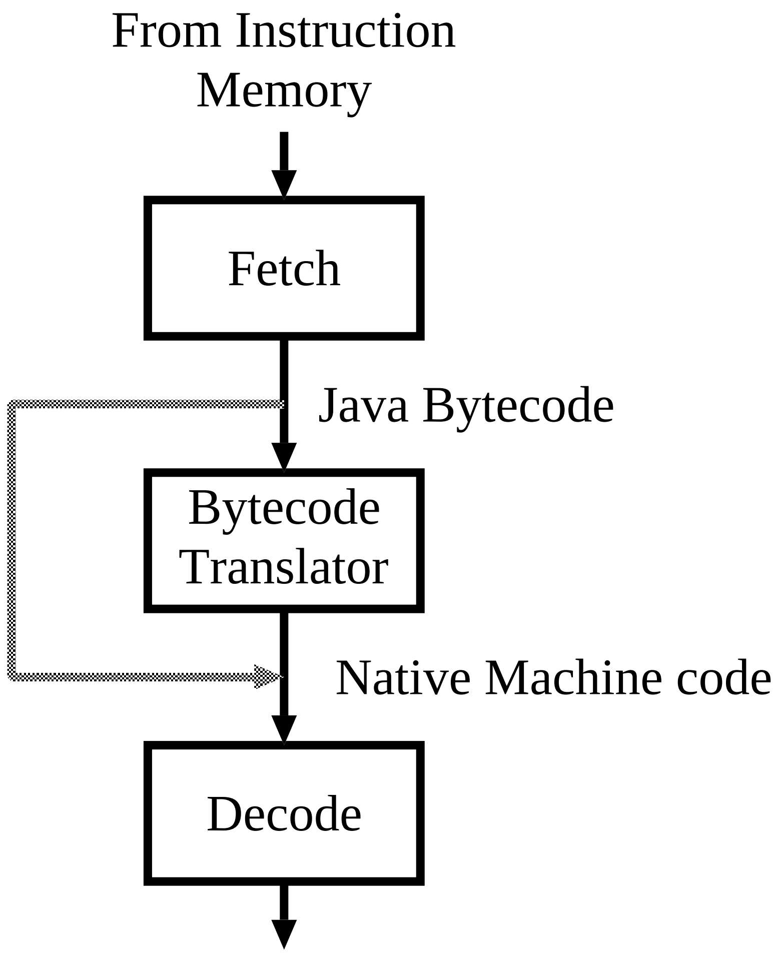
Execution Overheads of VMs

- 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)

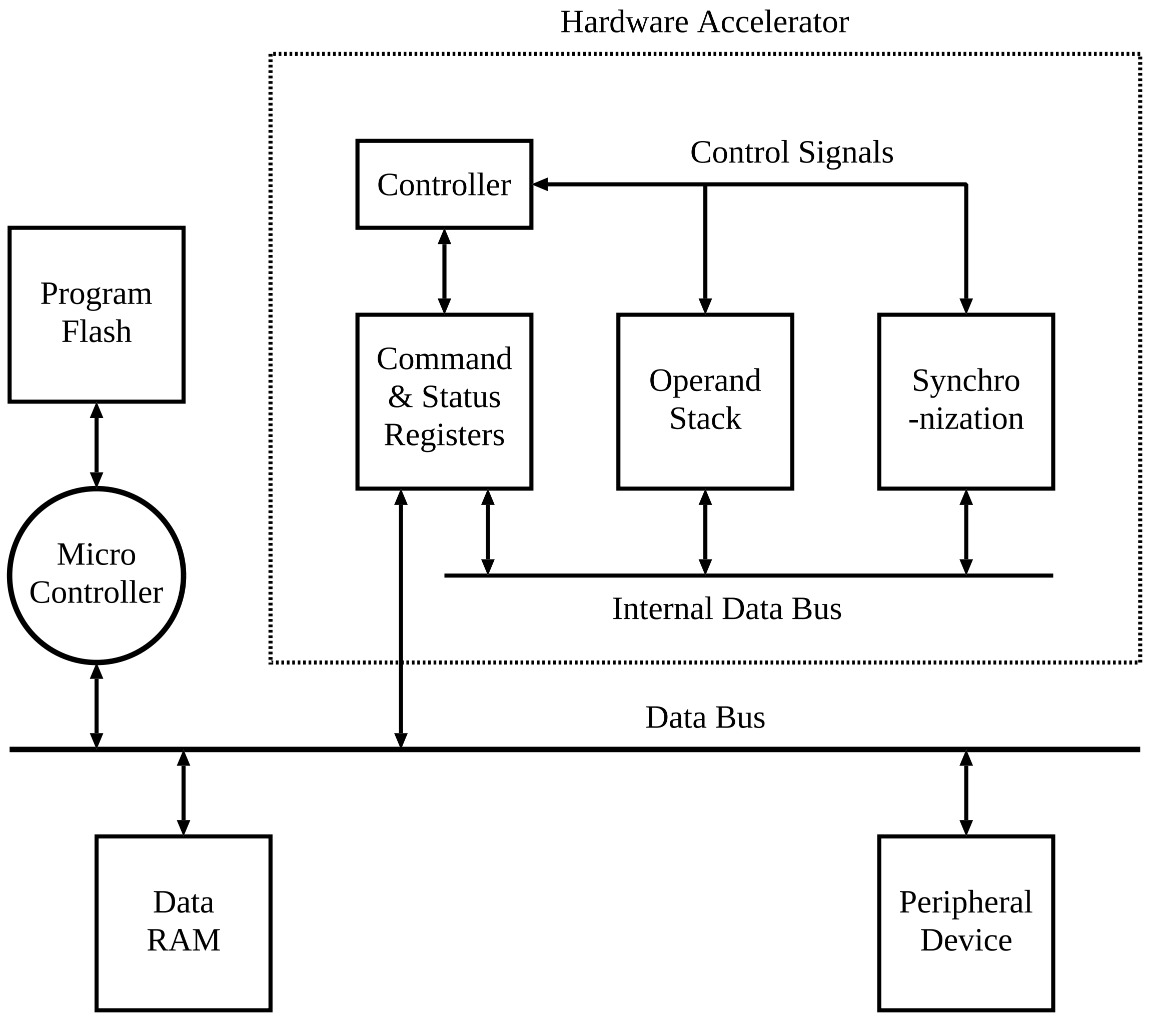
TinyOS and Maté

- TinyOS is an operating system for wireless sensor network, developed at UC Berkeley, and currently a de facto standard for WSN OS.
- Component based design: application programmers choose necessary components from the library and compile (wire) them together.
- Maté is a stack-based bytecode interpreter for TinyOS, providing synchronization control for concurrentl executed contexts.
- We use Maté on TinyOS as the base design.

Hardware Support for VM Execution

- From literature and source code analysis of Maté, synchronization and operand stack modules within Maté have been identified as the significant source of execution overhead.
- In these modules, large data structures (e.g. address pointers, bit vectors) are handled by a 8-bit processor. Software implementation of these modules results in longer execution time.
- Implementing these modules by the hardware results in shorter running time, which in turn possibility of putting the system in low power consumption mode for a longer period (i.e. lower duty cycle)

Proposed Architecture with HW Accelerator



Operand Stack Module

- An operand stack consists of 8 stack entries and a stack pointer.
- Each context has its own stack. In software implementation, TOS address is obtained by
  $BaseAddress + sizeof(OperandStack) * ContextID + SP$
  - All addresses are 16-bit
  - Multiplication is a costly operation for a 8-bit CPU
  - (VM) stack pointer (SP) is in memory
  - Over/under flow checking
- In hardware, stack entries can be implemented as a register file indexed by the contextID and SP, and over/under flow checking is done by a dedicated circuit (not by the ALU).

Synchronization Module

- In Maté, multiple context are executed in an interleaved manner and they share variables.
- Synchronization modules avoids race conditions between context by locking, unlocking and reporting status of variables.
- In the software implementation, bit vectors representing variables status are maintained for each context.
- Like operand stack, in software, synchronization operations take 10's of clock cycles while in hardware they can be implemented by the combinational circuits as contextID and lock variable numbers as input.

HW and SW Comparison (1) Methods

- Clock cycles for Stack and Synchronization modules in hardware and software implementations are compared.
- In hardware, we assume the following: the CPU write necessary parameters and then the command to the control registers. The return value and/or status are read from the status registers by the CPU.
- In software, BombillaMica's (an implementation of Maté included in TinyOS) source codes have been instrumented by Avrora (Atmel processor simulator).

HW and SW Comparison (2) Operand Stack

Command	Cycles		Description
	SW	HW	
pushValue	93	14	Push 16-bit Int
pushReading	93	16	Push sensor reading
pushBuffer	93	14	Push message buffer
pushOperand	91	14	Push untyped operand
popOperand	65	14	Pop TOS operand

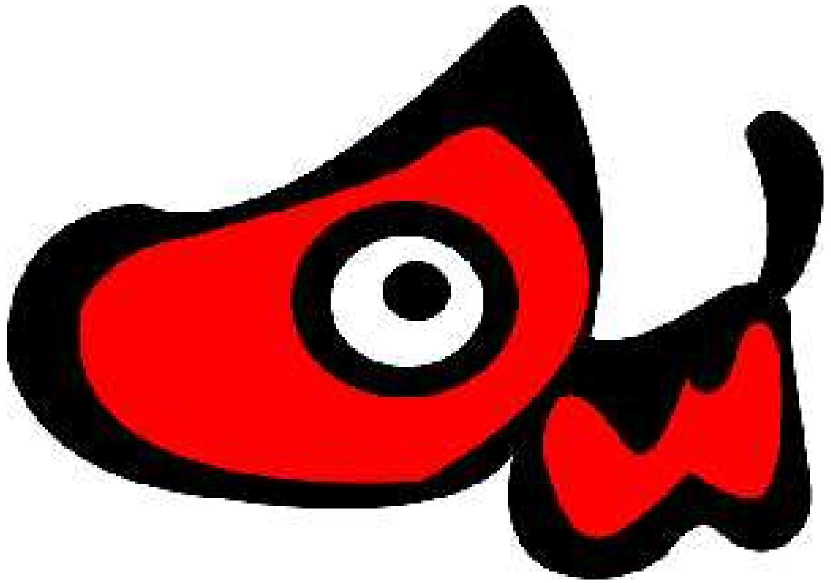
HW and SW Comparison (3) Synchronization

	Cycles		
Command	SW	HW	Description
lock	62	10	Lock a variable
unlock	64	10	Unlock a variable
isLocked	23	8	If a var is locked ?
isHeldBy	29	10	Am I locking a var ?
AnalyzeVars	1.3×10^4	270	Find vars in a capsule

Conclusions and Future Work

- Hardware support for WSN VM has been proposed and its quantitative benefit has been presented.
- The comparison results are limited in VM bytecodes: native codes and effect of peripherals must be taken into account.
- A possible simulation method is to combine Avrora (Atmel processor simulator) with the HDL model of the proposed modules.
- Standard benchmark for wireless sensor network virtual machine which includes not only the application but also the usage model (duty cycle, program update frequency) is needed.

Thanks for your attention



Any quesstion ?