Parallelism in Embedded Systems—
Runtime Support, Programming Languages, and Hardware Platforms

Phillip Stanley-Marbell
Technische Universiteit Eindhoven
Parallelism in Embedded Processing Systems

• Many real-world data sources have large amounts of data-level parallelism
  - Image processing, audio, sensor-driven systems

• Parallelism in hardware at two extremes of a spectrum
  - At the integrated circuit level—Multiprocessor Systems-on-Chip (MPSoCs)
  - At the printed circuit board and network levels—Networked Embedded Systems

• Many challenges, including
  - Runtime system support: how to harness multiple cores/processors/microcontrollers
  - Efficiency, flexibility: adapt to static hardware configuration, and dynamic program properties
  - Modeling and evaluation tools, benchmarks
Outline

• Motivation—Parallelism in Embedded Systems

• On-Chip Integration: A Thread Library for Heterogeneous Multi-Core Systems

• System-Level Integration: Programming Unreliable Networks of Computation

• Research Hardware and Tools

• Summary, Ongoing and Future Directions
Loosely Coupled Embedded Multiprocessors

- Multiprocessor systems-on-a-chip (MPSoC):
  - Multiple hardware cores integrated on a single die
  - General purpose processor cores, DSPs, other hardware accelerators
  - Harness hardware parallelism for both performance and energy efficiency
A Concrete Example

- **PortalPlayer PP5022**
  - The PP5022 is used in a very popular (>1E7 units sold) portable music player...
  - Two ARM7 cores at up to 100MHz, per-core caches are not kept coherent
MPSoCs: Challenges

- **MPSoCs pose many challenges**
  - Programming multi-processor systems, is not a new research problem! However:
    - Heterogeneity: cores of different types, from different vendors/ IP sources
    - Design reuse => loosely-coupled, not tightly integrated as in traditional (symmetric) multiprocessors
    - Communication costs can be large, vary widely across cores

- **Alternatives to programming MPSoCs**
  - No new interface: let each development project “roll its own”
  - New programming language: high barrier to adoption
  - OS-level interface: not all cores may be running an OS
  ✓ Use a prevalent multi-threading interface: programs link to a library
A Thread Library for Heterogeneous MPSoCs

- **Approach:** facilitate programs on MPSoCs using a thread library
  - Implements the IEEE Portable Operating Systems Interface (POSIX) 1003.1c threading standard ("Pthreads")
  - Existing applications using Pthreads easily compiled against new implementation

- **Optimizations**
  - Library implementation takes advantage of application-specific inter-thread communication patterns
  - Intelligent virtual-to-physical mapping of relevant structures to customize in-memory placement
  - Enables both programmer-directed and transparent control of such placement

---

Potential for Static Optimization: Example

Software-Defined Radio (SDR) Mapped to an MPSoC HW Platform:

- Memory access latencies can vary significantly
  - Access to local memory vs global memory vs local memory of another core
Potential for Static Optimization: Example

- Application-specific optimization through intelligent placement of buffers
  - Shared buffers used to pass data between threads
  - $2^4$ possible mappings of 4 shared buffers; mapping impacts performance—intuitive choice not optimum
Potential for Transparent Optimization (SDR)

- Many memory references are to library-controlled shared structures
- Implicit shared structures include mutual exclusion locks, sleep queues
Potential for Dynamic Optimization

- Optimum mapping may change over time
  - Example: 802.11a/b/g MAC layer encryption on an MPSoC
  - CRC (Cyclic Redundancy Check); WEP (Wired Equivalence Privacy)
  - Interaction b/n main, CRC, and WEP depends on class of MAC-layer frames: management/control/data

- Figure on right shows the “oracle” mapping of 36 structures to 3 CPU-local SRAMs, for 10s of 100ms windows
Hardware Support for Dynamic Optimization

- **Map Table Support Peripheral (MTSP)**
  - A peripheral on each core which implements performance counters to enable dynamic optimization
  - Library registers memory regions (i.e., data structures) with each MTSP
  - MTSP monitors memory bus, tracks access to registered regions
Static and Dynamic Optimization Results

Static Optimization: Software-Defined Radio (SDR)

- Best static mapping provides **up to 25% improvement** for SDR
- Overheads of dynamic optimization lead to **smaller benefit**; best dynamic mapping at 50ms update interval for MAC;

Dynamic Optimization: 802.11 MAC Layer Encryption

- Normalized maximum attainable encryption throughput

---

NLE, April 2008
Outline

• Motivation—Parallelism in Embedded Systems

• On-Chip Integration: A Thread Library for Heterogeneous Multi-Core Systems

• System-Level Integration: Programming Unreliable Networks of Computation

• Research Hardware and Tools

• Summary, Ongoing and Future Directions
Parallelism in Embedded Systems

**MPSoCs**

- **MPSoCs are one extreme** of the integration spectrum
  - Multiple processing cores integrated on a single silicon die

- **The other extreme are wired or wireless networks of discrete processors**
  - Communication is significantly more expensive, may be unreliable
Unreliable Networks of Computation

- Harnessing networks of failure-prone computational elements
  - To control such systems with software, we must contend with two issues

1. Constrained memory and compute resources
   - Can we create disjoint partitions of applications, across devices?
   - Utilize network as a single computational system

2. Failure rates compounded by large numbers of devices
   - Conjecture: must consider the role of failures from the ground up
Unreliable Networks of Computation

- **Observation**: programs often processing “inexact” values
  - Usually digital representation of noisy analog signals
  - e.g., if processing color values, small deviations might be acceptable
  - Amount of such tolerable deviations varies *between* and *within* applications

- **Observation**: some latencies and lost communications tolerable
  - Inter- and intra-application variation of tolerable latency and lost communications
M — A Programming Language for Failure-Prone Hardware

- **M** — a small concurrent programming language
  - A research vehicle for investigate programming systems such as in example of previous slide

- **Objectives**
  1. Facilitate creation of disjoint program partitions across devices in target platform
  2. Expose tolerable value deviations, communication latencies and erasures, at language level
  3. Enable the change of control flow when tolerances are violated

- **Language model in a nutshell**
  - Programs composed of name generators (sequential processes)
  - Interaction b/n name generators is through channels visible in a runtime name space
  - Constructs for making entries in runtime name space visible in programs & vice versa
  - Constructs for specifying error-, latency- and erasure-tolerance constraints in types

---

P. Stanley-Marbell and D. Marculescu — *A Programming Model and Language Implementation for Concurrent Failure-Prone Hardware, PMUP ’06*
Example

```plaintext
EdgeDetect : proctype
{
  READ    : const true;
  WRITE   : const false;
  img_row : namegen (bool, int,
                       byte epsilon(2.0, 0.01));(byte);
  init    : namegen ()::(args: list of string);
}

init =
{
    x, y    : int;
    image   : array [64] of chan of
               (bool, int, byte epsilon(2.0, 0.01));

    # Instantiate several name generators to hold dynamic
    # data structures across devices on network
    for (i := 0; i < 64; i++) {
        image[i] = name2chan img_row "img_row" 4E-6;
        out_image[i] = name2chan img_row "img_row" 4E-6;
    }
}

img_row =
{
    row := array [64] of byte;
    for (;;) match {
        <-img_row => {
            (op, idx, val) := <-img_row;
        }
    }
```

- Error-tolerance constraint
- Name generator instantiation
- Name (channel) communication
Outline

• Motivation—Parallelism in Embedded Systems

• On-Chip Integration: A Thread Library for Heterogeneous Multi-Core Systems

• System-Level Integration: Programming Unreliable Networks of Computation

• Research Hardware and Tools

• Summary, Ongoing and Future Directions
Architectural Modeling Across Computing Generations

Figure is not to scale!
**Simulator Overview**

- **Sunflower** is a “full-system” simulator for embedded systems
  - System/microarchitecture modeling of processors
  - Modeling networks of processing elements
  - Power estimation for computation and communication
  - Battery and voltage regulator modeling
  - Modeling analog signals in environment of computation
  - Modeling failures in network and computation

Open source, available online [http://sflr.org](http://sflr.org)

---

A Research Multi-Processor Platform

- System contains 24 low-power microcontrollers
  - Connected in a low-diameter Kautz topology
  - Low idle power — < 3uW
  - (Theoretical) peak perf. of 384 MIPS
A Research Multi-Processor Platform

- Plugs into rear of this handheld platform
- Can also be used as independent processor module

Hardware-measured “Eye Diagram” at 8 Mb/s

Hardware design is open source, available online [http://sflr.org/](http://sflr.org/)
Outline

• Motivation—Parallelism in Embedded Systems

• On-Chip Integration: A Thread Library for Heterogeneous Multi-Core Systems

• System-Level Integration: Programming Unreliable Networks of Computation

• Research Hardware and Tools

• Summary, Ongoing and Future Directions
Summary

- Opportunities for optimization in loosely-coupled embedded multiprocessors
  - Presented a Pthreads implementation targeting MPSoCs with heterogenous memory architectures
  - Facilitates static and dynamic in-memory placement of data structures

- Parallelism at system-level (e.g., circuit board, wired/wireless networks)
  - Language support for programming concurrent failure-prone hardware

- Modeling tools and prototype hardware
  - Sunflower tool suite http://sf1r.org – open suite of simulation and hardware tools for chip-level multiprocessors and networks of processors
Ongoing and Future Work

**Ongoing**
- Evaluation of thread library for larger collections of applications (ALP Bench suite)
- Public release of M language compiler, targets for hardware module and simulator
- Application of map table ideas to M language runtime system

**Future**
- Programmable logic implementation of hardware performance counters for dynamic optimization
Acknowledgments

• Energy-Aware Computing (EnyAC) Research Group at CMU
  - Prof. Diana Marculescu, Emil Talpeș, Koushik Niyogi, Nataša Miškov-Živanov, Sebastian Herbert, Siddharth Garg, Puru Choudhary

• NEC Labs Princeton
  - Kanishka Lahiri, Anand Raghunathan

• Technische Universiteit Eindhoven
  - Electronic Systems (ES) Group

Thanks
Dynamic Optimization with Hardware

**Software**

- Start
  - End of current window timer?
  - Read hardware performance statistics for previous window
  - Predict number of accesses to shared structures
  - Compute new mapping and potential speedup
  - Predicted performance gains outweigh overheads?
    - Yes
    - No
  - Registration of shared data structures, from thread library. Insert entry into library's Map Table

**Hardware**

- Create new MTSP performance counter for address range
- Update hardware map table counters
- DMA regions from old processor/memory location to new one; update MTSP

**Bus Access**
Optimizations: HW/SW Map Tables

• So far, we’ve talked about the potential for optimization
  - *What can one actually do?*

• Optimization Implementation: Map Tables
  - **Software Map Table:** contains variable names/tags and CPU mapping (or auto)
  - **Hardware Map Table:** contains address ranges for variable/structure instances, and CPU mapping
  - Entries are created in HW Map Table either as a result of explicitly calling `pthread_register()` in programs, or implicitly as a result of using standard Pthread interface functions (e.g., `pthread_cond_wait()`)

• Static Optimization
  - Software map table determines mapping of data structures to memories

• Dynamic Optimizations
  - Software map table used as initial “hint”, with addition of hardware map table (counters)…
Thread Library Implementation

- **Industry standard interface**
  - POSIX (Portable Operating Systems Interface) threading API (pthreads)
  - IEEE POSIX 1003.1c standard (1995)
  - Using a standard interface eases the barrier to adoption
  - Thread creation/destruction: e.g., `pthread_create()`
  - Synchronization: e.g., `pthread_cond_wait()`
  - Thread management: e.g., `pthread_setstacksize()`
  - Mutual exclusion: e.g., `pthread_mutex_lock()`

- **Threads on “bare metal”**
  - Library handles processor initialization, machine state save/restore for interrupts

- **Extension to standard Pthreads interface**
  - Programmer specifies placement of a data structure using `pthread_register()`
  - Programmer provides address, size and desired CPU core in which to place object
  - Example: `pthread_register(&buf, sizeof(buf) “shared buffer”, 3)`
    Request that the variable `buf` be placed in the local memory of processor core # 3
Specification of Tolerable Error

• Ideally, would specify distribution of tolerable deviation
  - e.g., “Let probability of deviation exceeding $x$ be $1/x$”
  - Simpler constraint: “Let probability of deviation exceeding $m$, be less than $A$”
    - Both $m$ and $A$ are constants and this constraint is denoted $\text{epsilon}(m, A)$

• Example

1. a : int $\text{epsilon}(1, 0.1)$;
2. b : int $\text{epsilon}(8, 0.01)$;
3. v : int;
4.
5. a = 2;
6. b = a + v;

- Similar constructs for latency and loss constraints