Research Methods in computer science
Spring 2017

Lecture 24

Omprakash Gnawali
April 18, 2018
Agenda

Conference Logistics
Remaining HWs
Posters
Adapted from Kristos Kozyrakis who borrowed from Dave Patterson
Posters
Where do we use posters?

Conference poster session
Maybe conference demo sessions
Research retreats
School hallways!
Some conferences:

“We are pleased to inform you that your paper is accepted for Poster Presentation.”
Example of Call for Posters

- POSTERS -

The poster session at SenSys provides a forum for researchers to present their work and receive feedback from experts attending the conference. We explicitly encourage submissions from students.

Posters must be submitted as a single PDF containing no more than 3 pages. The first two pages should contain an abstract describing the research content of the poster, along with title, authors, institutional affiliations and contact information. The third page should contain a thumbnail draft of the poster's contents.

For more information, please contact the poster chairs.
Evaluation of posters

Theory: same as papers
Practice: paper evaluation--
7 Poster Commandments for a Bad Poster

I. Thou shalt not illustrate.
II. Thou shalt not covet brevity.
III. Thou shalt not print large.
IV. Thou shalt not use color.
V. Thou shalt not attract attention to thyself.
VI. Thou shalt not prepare a short oral overview.
VII. Thou shalt not prepare in advance.
We describe the philosophy and design of the control flow machine, and present the results of detailed simulations of the performance of a single processing element. Each factor is compared with the measured performance of an advanced von Neumann computer running equivalent code. It is shown that the control flow processor compares favorably in the program.

We present a denotational semantics for a logic program to construct a control flow for the logic program. The control flow is defined as an algebraic manipulator of idempotent substitutions and it virtually reflects the resolution deductions. We also present a bottom-up compilation of medium grain clusters from a fine grain control flow graph. We compare the basic block and the dependence sets algorithms that partition control flow graphs into clusters.

Our compiling strategy is to exploit coarse-grain parallelism at function application level: and the function application level parallelism is implemented by fork-join mechanism. The compiler translates source programs into control flow graphs based on analyzing flow of control, and then serializes instructions within graphs according to flow arcs such that function applications, which have no control dependency, are executed in parallel.

A hierarchical macro-control-flow computation allows them to exploit the coarse grain parallelism inside a macrotask, such as a subroutine or a loop, hierarchically. We use a hierarchical definition of macrotasks, a parallelism extraction scheme among macrotasks defined inside an upper layer macrotask, and a scheduling scheme which assigns hierarchical macrotasks on hierarchical clusters.

We have demonstrated that to achieve the best execution time for a control flow program, the number of nodes within the system and the type of mapping scheme used are particularly important. In addition, we observe that a large number of subsystem nodes allows more actors to be fired concurrently, but the communication overhead in passing control tokens to their destination nodes causes the overall execution time to increase substantially.

The relationship between the mapping scheme employed and locality effect in a program are discussed. The mapping scheme employed has to exhibit a strong locality effect in order to allow efficient execution. We assess the average number of instructions in a cluster and the reduction in matching operations compared with fine grain control flow execution.

Medium grain execution can benefit from a higher output bandwidth of a processor and finally, a simple superscalar processor with an issue rate of ten is sufficient to exploit the internal parallelism of a cluster. Although the technique does not exhaustively detect all possible errors, it detects nontrivial errors with a worst-case complexity quadratic to the system size. It can be automated and applied to systems with arbitrary loops and nondeterminism.
Alternatives to Bad Posters (from Randy Katz)

- Answer Five Heilmeier Questions
  1. What is the problem you are tackling?
  2. What is the current state-of-the-art?
  3. What is your key make-a-difference concept or technology?
  4. What have you already accomplished?
  5. What is your plan for success?

- Do opposite of Bad Poster commandments
  - Poster tries to catch the eye of person walking by

- 9 page poster might look like

<table>
<thead>
<tr>
<th>Problem Statement</th>
<th>State-of-the-Art</th>
<th>Key Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accomplishment # 1</td>
<td>Title and Visual logo</td>
<td>Accomplishment # 2</td>
</tr>
<tr>
<td>Accomplishment # 3</td>
<td>Plan for Success</td>
<td>Summary &amp; Conclusion</td>
</tr>
</tbody>
</table>
**AME is the 21st Century Challenge**

- **Availability**
  - systems should continue to meet quality of service goals despite hardware and software failures
- **Maintainability**
  - systems should require only minimal ongoing human administration, regardless of scale or complexity
  - Today, cost of maintenance = 10X cost of purchase
- **Evolutionary Growth**
  - systems should evolve gracefully in terms of performance, maintainability, and availability as they are grown/updated/expanded
- **Performance was the 20th Century Challenge**
  - 1000X Speedup suggests problems are elsewhere

**People are the biggest challenge**

<table>
<thead>
<tr>
<th>Number of Outages</th>
<th>Minutes of Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- People > 50% outages/minutes of failure
  - “Sources of Failure in the Public Switched Telephone Network,” Kuhn: IEEE Computer, 30-4 (Apr 97)
  - FCC Records 1992-1994: Overload (not sufficient switching to lower costs) + 6% outages, 44 minutes

**ROC Principles:**

1. **Isolation and redundancy**
   - System is partitionable
   - to isolate faults
   - to enable online repair/recovery
   - to enable online HW growth/SW upgrade
   - to enable operator training/expand experience on portions of real system
   - Techniques: Geographically replicated sites, Shared-nothing cluster, Separate address spaces inside CPU
   - System is redundant
     - sufficient HW redundancy/data replication => part of system down but satisfactory service still available
     - enough to survive 2nd failure or more during recovery
     - Techniques: RAID-6; N-copies of data

2. **Online verification**
   - System enables input insertion, output check of all modules (including fault insertion)
   - to check module operation to find failures faster
   - to test correctness of recovery mechanisms
     - insert faults and known-incorrect inputs
     - also enables availability benchmarks
   - to test if proposed solution fixed the problem
   - to discover whether need to try another solution
   - to discover if warning systems are broken
   - to expose and remove latent errors from each system
   - to train/expand experience of operator
   - Techniques: Global invariants; Topology discovery; Program checking (SW ECC)

3. **Undo Support**
   - ROC system should offer Undo
     - to recover from operator errors
     - undo is ubiquitous in productivity apps
     - should have “undo for maintenance”
     - to recover from inevitable SW errors
     - restore entire system state to pre-error version
     - to recover from operator training via fault-insertion
     - to replace traditional backup and restore
     - Techniques: Checkpointing; Logging; and time travel (log structured) file systems

4. **Diagnosis Support**
   - System assists human in diagnosing problems
     - root-cause analysis to suggest possible failure points
       - track resource dependencies of all requests
       - correlate symptomatic requests with component dependency model to isolate culprit components
     - “health” reporting to detect failed/failing components
       - failure information, self-test results propagated upwards
     - unified status console to highlight improper behavior, predict failure, and suggest corrective action
     - Techniques: Stamp data blocks with modules used; Log faults, errors, failures and recovery methods

**Lessons learned from Other Fields**

- 1800s: 25% railroad bridges failed
  - Techniques invented since:
    - Learn from failures vs. successes
    - Redundancy to survive some failures
    - Margin of safety 3X
    - Redundancy to survive some failures
    - Learn from failures vs. successes
  - Safety now in Civil Engineering DNA
    - Structural engineering is the science and art of designing and making, with economy and elegance, structures that can safely resist the forces to which they may be subjected
    - Have we been building the computing equivalent of the 19th Century iron-truss bridges?
    - What is computer equivalent of safety margin?

**Recovery-Oriented Computing (ROC) Hypothesis**

“If a problem has no solution, it may not be a problem, but a fact, not to be solved, but to be cope with over time”
— Shimon Peres

- Failures are a fact, and recovery/repair is how we cope with them
- Improving recovery/repair improves availability
  - Availability = \(MTTF \over MTTR\)
  - Since MTTF >> MTTR, 1/10th MTTR just as valuable as 10X MTBF
- Since major Sys Admin job is recovery after failure, ROC also helps with maintenance

**Conclusion**

- New century needs new research agenda
  - (and its not performance)
- Embrace failure of HW, SW, people and still build systems that work
- ROC: Significantly reducing Time to Recover/Repair
  - much greater availability
  - much lower maintenance costs
Introduction

Twonet is a publicly available large-scale sensor network testbed with dual-radio sensor nodes based on the modern Cortex-M3 architecture. Twonet's creation is motivated by the increasing interest in research on multi-channel wireless networking in sensor networks.

Twonet consists of 100 Opal nodes [5] with 2.4 GHz and 900 MHz radios deployed across four floors of an academic building. Similar to the way early sensor network testbeds helped advance research on wireless sensor networks, Twonet will enable new research on multiband radio communications and will enable new class of sensor network applications that utilize the powerful yet energy-efficient Cortex-M3 CPU architecture.

System Architecture

Twonet is inspired by earlier sensor network testbeds and uses the time-tested three-tier architecture.

**Tier 1: Controller.** A single Linux server serves as controller. It provides a web front-end for users to interact with the testbed. The user may specify experiment parameters, upload binaries, and download results. The controller distributes the programming and control job to the Proxies at tier 2. The controller also collects the logs generated by the sensor nodes and saves it to a database.

**Tier 2: Proxy.** Twenty Raspberry Pi nodes constitute tier 2 of Twonet. Raspberry Pi is a low-cost embedded Linux platform with sufficient memory and CPU cycles to program, control, and collect the logs from the sensor nodes. Each Raspberry Pi is powered using a Power-over-Ethernet (PoE) switch and connects to sensor nodes through a USB hub. Raspberry Pi nodes can program the Opal nodes and collect the debug logs generated by the nodes. Each raspberry Pi node is connected to 5 Opal nodes.

**Tier 3: Sensor Node.** The Opal nodes constitute the leaves of the network. Each Opal node is connected to a custom-built debug board. The debug board and Opal node each connect to Raspberry Pi using separate USB cables. The debug board is required to reset Opal during programming and also provides additional mechanisms to debug programs running on Opal. Each Opal node has a custom plastic enclosure with two antenna mounted for 2.4 GHz and 900 MHz radios.

Debugging Modalities

**Serial output.** The Opal sensor node provides two separate serial ports for debugging: (1) a UART port running at 115200 baud, and (2) a high-speed USB 2.0 port operating at 480Mbits.

**Memory Tracing.** The Opal's Cortex-M3 microcontroller debug port implements the JTAG protocol to provide read/write access to the system memory and peripherals.

**Global Breakpoints.** The remote JTAG access to the Opal nodes allows for sending simultaneous start/stop requests to all nodes within the network.

Acknowledgements

This work is supported in part by the Sensors and Sensor Networks TCP of CSIRO and a generous gift from Cisco.
**Twonet: Large-Scale Wireless Sensor Network**

**Testbed with Dual-Radio Sensor Nodes**

Qiang Li, Dong Han, Omprakash Gnawali
Univ. of Houston USA

Philipp Sommer, Branislav Kusy
CSIRO, Australia

---

**Twonet Testbed**

- Consists of 100 Opal nodes with 2.4GHz and 900MHz radios
- Based on Cortex-M3 architecture
- Enable new research on multi-band radio communications
- Enable new class of sensor network applications that utilize the powerful yet energy-efficient Cortex-M3 CPU architecture

**Twonet Architecture**

- **Tier 1: Controller**
  - Web front-end provided for users to interact
- **Tier 2: Proxy**
  - Reprogram sensor nodes for users
  - Collect the debug logs generated by the nodes
- **Tier 3: Sensor Nodes**
  - Low-power 32-bit ARM CPU embedded in the node
  - Two antenna mounted for 2.4GHz and 900 MHz

---

**Debugging**

- **Serial Output**
  - A UART port running at 115200 baud rate
  - A high-speed USB 2.0 port
- **Memory Tracing**
  - Collect memory traces without affecting the timings of the application under test
- **Global Breakpoints**
  - Stop the application running on the node
  - Take a snapshot of the network state for further analysis

---

**Current Deployment at UH**

- 87 nodes currently deployed, moving up to 100

---

**User Interface**

- Schedule the tasks by specifying the start time and duration
- Inspect and change the real-time status of the nodes: program, stop, or restart the node
- Download log data through the serial interface
- Allow approval of users, setting user quota, updating node metadata

---

**http://twonet.cs.uh.edu**
Routing Principles in Wireless Mesh Networks

Omraksh Gnaeuli (University of Southern California), Rodrigo Fonseca (Yahoo and Brown University), Kyle Jamieson (University College London), Kannan Srinivasan (Stanford University), and Philip Lewis (Stanford University)

The Problems

<table>
<thead>
<tr>
<th>Example Challenge</th>
<th>Three Principles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wireless routing is hard</td>
<td>Abstracting the design principles from protocol implementation experiences</td>
</tr>
<tr>
<td>Link dynamics are common, resulting in:</td>
<td>1. Use cross-layer information to estimate link costs</td>
</tr>
<tr>
<td>1. Online/frequent link and route quality assessment</td>
<td>2. Use data path to actively validate routing topology</td>
</tr>
<tr>
<td>2. Stale states</td>
<td>3. Adapt beacon rates based on routing topology consistency</td>
</tr>
<tr>
<td>3. Uncontrolled churn</td>
<td></td>
</tr>
<tr>
<td>4. Loops</td>
<td></td>
</tr>
</tbody>
</table>

Burtness

Short term dynamics

Measures temporal correlation of packet reception

Hybrid Link Estimation

Data Path Validation

Adaptive Beaconing

Estimate link cost by actively measuring the data path.

Use network layer for hints on what links are most useful.

Beacons discover neighbors.

Merge data path and control path estimates using EWMA.

Data path estimates lead to very rapid changes in cost and route (e.g., 10 packet times). This dynamism causes routing loops.

Use the data path to quickly detect possible loops (cost does not monotonically decrease along route).

Beacons seed routing tables and tell neighbors of a node’s cost.

A node only needs to send beacons when stale information leads to routing errors or neighbors need candidates.

Use an exponential timer: reset on 1. Data path detection, 2. “pull” bit, or 3. large decrease.

Beacons seed routing tables and tell neighbors of a node’s cost.

A node only needs to send beacons when stale information leads to routing errors or neighbors need candidates.

Use an exponential timer: reset on 1. Data path detection, 2. “pull” bit, or 3. large decrease.

Beacons seed routing tables and tell neighbors of a node’s cost.

A node only needs to send beacons when stale information leads to routing errors or neighbors need candidates.

Use an exponential timer: reset on 1. Data path detection, 2. “pull” bit, or 3. large decrease.

Experimental results

Link metric that uses all information significantly more efficient than ETX due to lower cost and shorter paths.

Use multiple information sources

No disruption in packet delivery.

Quick repair of packet delivery.

Fewer beacons sent using adaptive beaconing than with periodic beacons.

Use adaptive “Trickle” timers to reduce overhead and save energy

Results consistent across 12 testbeds, 7 hardware platforms, and 6 link layers
A Benchmark for Low-power Wireless Networking

Motivation

Low-power wireless lacks a standard evaluation
- Almost two decades of WSN/IoT research
- …and yet no standard way to evaluate
- Challenges in evaluation
  - variety of settings
  - hard to compare against reference implementation
  - comparison: protocols vs protocols+platform

Our Goal

As a community, define common benchmarks
- Set of tools and practices for performance evaluation
- Enable fair comparison
- Enable repeatability (to a certain extent)
- As a complement to custom evaluations

Design Space

Approach #1: Specification only
- common conceptual framework
- common system infrastructure

Approach #2: Standardized testing architecture
- ideal goal: comparison in the same application and real-world environment conditions

Envisioned Approaches

Approach #1: Specification only
- Metrics
  - observed: e.g., wireless noise
  - input: e.g., traffic load and pattern
  - output: e.g., delivery, latency, energy
- Profiles
  - assignment of concrete value to input
  - interpretation of observed and output

Approach #2: Standardized Testing Architecture
- Separate networking code from experiment scenario
  - node runs networking code
  - testbed runs experiments
- Fully automated, no misinterpretation of profiles
- More complex and strict, more infrastructure maintenance

Discussion

- Many more plausible design points
- Which one to focus first?
- How to foster adoption?

- Please provide input/feedback!
- We are looking for contributions
- TBD: to insert QR code of website
**Benchmarking Low-power Wireless Networking**

**IoT Benchmarks Initiative**

---

**The Problem**

- 20 years of research and yet no standard evaluation of low-power wireless protocols!
- Huge variety of settings
- Hard to compare against references
- Heterogeneous comparison (protocols only) vs. (protocols+ platform)

**The Vision**

- A benchmark designed by and for the community
- Set of tools and practices for performance evaluation
- Enable fair comparisons
- Enable some repeatability

---

**The Challenge**

How to design a benchmark that realizes our vision?

**#1 Formalized Test Definition**

- Based on profiles
- Technology-agnostic
- Clear evaluation settings
  - # of runs, expected statistics

**#2 Standardized Testing Architecture**

- Isolate networking code from evaluation scenario
- Nodes run networking code
- Test environment (tested or simulated) controls the evaluation
  - Fully automated
  - No misinterpretation of profiles
  - More complex to setup
  - More infrastructure maintenance

---

**CPSBench**

1st Workshop on Benchmarking Cyber-Physical Networks and Systems

**Get in touch and contribute!**

Twitter
@iot_bench

Google Group
bit.do/iot-bench

CPSBench (satellite of CPSWeek 2018)
cpsbench2018.ethz.ch

More information
IoT Benchmarks Initiative
www.iotbench.ethz.ch
Some poster examples from PhD Showcase 2017