Wishbone: Profile-based Partitioning for Sensornet Applications

- Ryan Newton,
  Sivan Toledo, Lewis Girod,
  Hari Balakrishnan,
  Samuel Madden
Example Application: Locating Marmots

- Gothic, CO deployment August 2007
- Voxnet Platform
  - 2x PXA255, 64MB RAM, 8GB Flash, 802.11B, Mica2 supervisor, Li+ battery, Charge controller
  - Sensors: 4x48KHz audio, 3-axis accel, GPS, Internal temp

with Lewis Girod & UCLA Blumstein Lab
We target sensing applications

- Animal localization
- Pothole detection
- Computer Vision
- Pipeline leak detection
- Speaker identification
- EEG Seizure detection

**Key application features**

- Independent stream processing tasks (e.g. dataflow graphs)
- Predictable data rates

**Challenges**

- Heterogeneous platforms
- CPU & radio bottlenecks
Heterogeneous Platforms

Router
weak cpu, strong radio

Smartphones
medium cpu, strong radio

Low power sensors
weak cpu/radio

Brew
JavaME
Symbian
Android
Linux microserver
TinyOS
Contiki
Java
Python

Mix and Match!

iPhone SDK
Contributions

Sensor source(s)

Network Boundary

Results
Contributions

Sensor source(s)

Results
Contributions

- First broadly portable sensenet programming
- Partitioning algorithm
- Optimize CPU/radio tradeoff even if app doesn’t “fit”
Architecture

Sample data (for profiling)

WS Compiler

Dataflow graph: operators containing code in portable intermediate language

Partitioner

Backend CodeGen

Wishbone

ANSI C

NesC/TinyOS

JavaME
Targeting TinyOS
Task granularity, messaging model

WaveScope:
Execute!

TinyOS:

• 16 bit microcontroller
• 10K RAM
• No mem. protection
• No threads

Profile-directed
Cooperative
Multitasking:

iterate x in S {
  f(); yield();
  for(i=...) {
    ...
    if (i==387) yield(); ...
    }
  g();
  yield(); g();
}
Profiling Streams and Operators

- Every sensor source is paired with sample data
- Includes timing info

- Measure rates, execution times

- Separately: profile network channel in deployment environment
  - → per-node send rate

```plaintext
audioStream = IFPROF(readFile("foo8kHz", readSensor()))
```

![Diagram showing network channels and send rates]
State, Replication, and Pinning

Pinning Constraints

• All stateless ops: unpinned
• Stateful replicated ops: unpinned
• Stateful global ops: pinned to server – don’t distribute!
**Problem Scenario**

**Embedded Node**

- **CPU:** \(\frac{7}{3}\)
- **Network:** 23

**Server / Base Station**

- **Network Boundary**

**Problem Inputs**

- profile data: net, cpu
- network channel capacity

**NP-Hard**

Network Diagram:

- Node 3 connected to Node 4 with weight 12
- Node 4 connected to Node 19 with weight 11
- The network boundary is indicated by the lines connecting the nodes.
Partitioning Algorithm: Integer linear program formulation

- Introduce variables $f_u \{0,1\}$ where $u=server$, $1=sensor$
- Introduce variables $g_{uv} \{0,1\}$ where $1=cut edge$
- Enforce resource bounds:
  - $cpu < C$
  - $net < N$
  - Where $cpu = f_u(compute_u)$
  - Where $net = g_{uv}(data_{uv})$
- Minimize objective function:
  - $\min(\Diamond cpu + net)$

Tricky bit (see paper): Relating $f$ and $g$ while staying linear

3 Parameters: $C, N, \alpha$

Proxy for Energy
Evaluation: Two Applications

Human speech detection/identification

- Source
- Preemph
- Hamming
- Prefilt
- FFT
- Filtbank
- Logs
- Cepstrals

EEG-based seizure onset detection

1400 operators
Observation:
Relative cost varies by platform
Visualizing Profile Data: Bandwidth vs. Compute

Processing reduces data quantity

Reasonable cutpoints

Operators:
- source
- preemph
- hamming
- prefit
- FFT
- filtBank
- logs
- cepstrals

Cumulative CPU Cost
Bandwidth (Right-hand scale)

Execution time of operator (microseconds)

Cumulative CPU cost (red)

Bandwidth of cut (KBytes/Sec)
Optimal partitions across platforms

EEG Application (1 of 22 channels)

Each line represents 2100 partitioner-runs
Speaker Detection: CPU performance across partitions/platforms

Putting the pieces together:

- Cpu & net bounds ➔ optimal partition (if exists)
- Partition ➔ est. throughput
- Binary search over rates (aka cpu bounds) ➔ max possible throughput

example: picks cutpoint after filtBank for speaker detection
Groundtruth: Testbed deployment, 20 motes

Speaker Detection
- On weakest platform, several partition points result in NO data
- But among “working” partition points, best is 20X better than worst

Pick the right one!
Related Work

- Graph partitioning for scientific codes
  - balanced, heuristic – e.g. Zoltan

- Task scheduling, commonly list scheduling

- Dynamic: Map-reduce, Condor, etc.

- Sensor network context: Tenet and Vango
  - Linear pipeline of operators
  - Manual partition
  - Run TinyOS code on both server and sensor
CONCLUSION

- **WS/Wishbone System**: *(available, open source)*
  - Convenient, efficient way to run one program on many devices
  - Dataflow graph profiling and visualization extremely useful even outside of auto partitioning

- **Contributions**
  - Algorithm for partitioning streaming programs
  - Backend(s) enable high level programming even on motes

- Techniques for node/server partitioning could be extended to handle heterogeneous cores intra-device.

Contact:  Ryan Newton - newton@mit.edu
**Partitioning: Algorithm Runtime**

- **Graph Preprocessing step**
  - Merge vertices until all edge weights are monotonically decreasing.
  - Eliminates the majority of edges.

- **Even without preprocessing**
  - 8000 runs,
  - partitioning the 1400-node EEG dataflow graph,
  - with different CPU budget,
  - took under 10 seconds 95% of the time.

- But there is a long tail... luckily ILP solvers produce approximate solutions as well!
Motivating Example

Unstable optimal partition. Flips between horizontal and vertical partition.