CSci 8980
Mobile Cloud Computing

Outsourcing: Components
MAUI: Making Smartphones Last Longer With Code Offload

-Microsoft Research
Outline

• Motivation
• **MAUI system design**
  – **MAUI proxy**
  – **MAUI profiler**
  – **MAUI solver**
• Evaluation
• Conclusion
Battery is a scarce resource

- CPU performance during same period: \(246X\)
- A solution to the battery problem seems unlikely
Mobile apps can’t reach their full potential

- Slow, Limited or Inaccurate
- Too CPU intensive
- Power Intensive
- Not on par with desktop counterparts
- Limited
- Augmented Reality

Speech Recognition
One Solution: Remote Execution

• Remote execution can reduce energy consumption

• Challenges:
  – What should be offloaded?
  – How to dynamically decide when to offload?
  – How to minimize the required programmer effort?
Energy Results

What do these energy results tell us?
MAUI: Mobile Assistance Using Infrastructure

MAUI Contributions

• Combine extensive profiling with an ILP solver
  – Makes dynamic offload decisions
  – Optimize for energy reduction
  – Profile: device, network, application

• Leverage modern language runtime (.NET CLR)
  – To simplify program partitioning
  – Reflection, serialization, strong typing
Language Run-Time Support For Partitioning

• Portability
  – Mobile (ARM) vs Server (x86)
  – .NET Framework Common Intermediate Lang

• Type-Safety and Serialization:
  – Automate state extraction

• Reflection
  – Identifies methods with [Remoteable] tag
  – Automates generation of RPC stubs
Reflection: An aside

• Allows you to inspect and modify programs while running

• reflection
MAUI Architecture

Application

Maui Runtime

Client Proxy

Profiler

Solver

Smartphone

RPC

Maui Runtime

Server Proxy

Profiler

Solver

Maui Controller

Application

Maui server
How Does a Programmer Use MAUI?

• Goal: make it simple to MAUI-ify apps
  – Build app as a standalone phone app
  – C# compiler to CIL intermediate language
  – Add .NET attributes to indicate “remoteable”
  – Follow a simple set of rules

```csharp
[Remoteable]
ArrayList GetValidMoves(Square s)
{
    if (s.IsEmpty())
    {
        return new ArrayList();
    }
    if (s.Piece.IsEnemyOf(active))
    {
        // this piece does not belong to the active side, no moves possible
        return new ArrayList();
    }
    // forward the call to the Rule-class
    return rules.getMoves(s);
}
```
Design Choice

• Programmer selects ‘remoteable’
  – Granularity is the method
  – This method is eligible for remote execution
  – What is eligible or ineligible?

• Alternative is marking methods ‘local’
  – What is the tradeoff here?
Programmer Effort?

• Must be smart enough to tag methods

• Must be willing to optimize
  – Missile example
  – Call a method $X$ times with each $\text{pos}_i$
  – Call a method once with \{pos$_1$, pos$_2$, ...\}
Contrast

• Two versions of method for phone and server
  – either phone ships binary
  – or server obtains binary from the cloud

• Is it enough just to ship method code?
  – Must also transmit dependent state (e.g. accessed member vars, globals, ...)
  – Interesting delta optimization: Maui can remember what it has previously shipped
Example

//original interface
public interface IEnemy {
    [Remoteable] bool SelectEnemy(int x, int y);
    [Remoteable] void ShowHistory();
    void UpdateGUI();
}

//remote service interface
public interface IEnemyService {
    MAUIMessage<AppState, bool> SelectEnemy(AppState state, int x, int y);
    MAUIMessage<AppState, MauiVoid> ShowHistory(AppState state);
}
Failure?

- If remote method fails
  - E.g. network failure

- Maui re-executes it
  - Issues?
Decision Process

• Continuous monitoring when executing to build a program graph
  – Data size
  – Network cost
  – Execution time

• Server solves linear program to estimate where to run method(s)
Maui Graph

Energy model based on benchmarks
CPU cycles => Joules

User Interface

FindMatch
900 mJ
15/5 s

DetectAndExtract
Faces
15000 mJ
30/20 s

InitializeFace
Recognizer
5000 mJ
200/100 s

Past execution is a good predictor of future?
Solver

\[
\text{maximize } \sum_{v \in V} I_v \times E_v^l - \sum_{(u,v) \in E} |I_u - I_v| \times C_{u,v} \\
\text{such that: } \sum_{v \in V} \left( (1 - I_v) \times T_v^l + (I_v \times T_v^r) \right) \\
+ \sum_{(u,v) \in E} (|I_u - I_v| \times B_{u,v}) \leq L \\
\text{and } \quad I_v \leq r_v, \quad \forall v \in V
\]
A sample call graph

- **A**: Computation energy and delay for execution
- **B**: Energy and delay for state transfer
- **C**: 5000 mJ, 3000 ms
- **D**: 15000 mJ, 12000 ms

MAUI Solver
Is Global Program Analysis Needed?

Yes! - This simple example from Face Recognition app shows why local analysis fails.

- **User Interface**: 1000mJ
- **FindMatch**: 900 mJ
- **InitializeFace Recognizer**: 5000 mJ
- **DetectAndExtract Faces**: 15000 mJ

Cheaper to do local
Is Global Program Analysis Needed?

Yes! - This simple example from Face Recognition app shows why local analysis fails.

1. User Interface: 1000mJ
2. FindMatch: 900 mJ
3. InitializeFace Recognizer: 5000 mJ
4. DetectAndExtract Faces: 15000 mJ

Cheaper to do local.
Is Global Program Analysis Needed?

User Interface

FindMatch

InitializeFace Recognizer

DetectAndExtract Faces

1000mJ

Cheaper to offload

25900mJ
Can MAUI Adapt to Changing Conditions?

• Adapt to
  – Network Bandwidth/Latency Changes
  – Variability on method’s computational requirements

• Experiment
  – Modified off the shelf arcade game application
  – Physics Modeling (missiles)
  – Evaluated under different latency settings
Can MAUI Adapt to Changing Conditions?

- **DoFrame**
- **DoLevel**
  - 11KB + missiles
  - 11KB + missiles
  - 11KB + missiles
  - Required state is smaller
  - Complexity increases with # of missiles

*Missiles take around 60 bytes each*
Case 1

- Zero Missiles
- Low latency (RTT < 10ms)

*Missiles take around 60 bytes each

Offload starting at DoLevel

Computation cost is close to zero
Case 2

- 5 Missiles
- Some latency (RTT = 50ms)

*Missiles take around 60 bytes each*
Questions

• How much can MAUI reduce energy consumption?
• How much can MAUI improve performance?
  – Note: we haven’t focused on this so far
• Can MAUI Run Resource-Intensive Applications?
How much can MAUI reduce energy consumption?

Face Recognizer

- Smartphone only
- MAUI (Wi-Fi, 10ms RTT)
- MAUI (Wi-Fi, 25ms RTT)
- MAUI (Wi-Fi, 50ms RTT)
- MAUI (Wi-Fi, 100ms RTT)
- MAUI* (3G, 220ms RTT)

An order of magnitude improvement on Wi-Fi

Big savings even on 3G
How much can MAUI improve performance?

Face Recognizer

- Smartphone only
- MAUI (Wi-Fi, 10ms RTT)
- MAUI (Wi-Fi, 25ms RTT)
- MAUI (Wi-Fi, 50ms RTT)
- MAUI (Wi-Fi, 100ms RTT)
- MAUI* (3G, 220ms RTT)

Improvement of around an order of magnitude
Latency to server impacts the opportunities for fine-grained offload

Arcade Game

Solver would decide not to offload

Up to 40% energy savings on Wi-Fi

Per frame communication modest computation
Conclusions

- **MAUI enables developers to**
  - Bypass the resource limitations of handheld devices
  - Low barrier entry: simple program annotations

- **For a resource-intensive application**
  - MAUI reduced energy consumed by an order of magnitude
  - MAUI improved application performance similarly

- **MAUI adapts to**
  - Changing network conditions
  - Changing applications CPU demands
Discussion

• Take into account load on remote machine?
• Offload details not well specified
Cuckoo

- A Computation Offloading Framework for Smartphones
Computation Offloading Framework

- Runs on Android, integrates with Eclipse
- Multiple implementations of compute intensive parts of the code
  - method granularity
- Remote and local implementation bundled together
- Multiple shared remote servers available
  - nice addition over Maui
Programming Model

- Android activity/service model
  - Activity for I/O parts
  - Service for compute intensive parts
  - Programmer identifies through AIDL which methods are activities and which are services
  - For services cuckoo generates stub for remote service
  - Programmer must provide remote method implementation
Programming Model (cont’d)

• Services (methods) must be stateless
  – Why?
Computation Offloading
Decision Process

• Cuckoo runtime provides list of remote machine locations
• Remote execution always favored
  – find a remote machine that is available
• When to bind to server
  – speed vs. energy
eyeDentify: object recognition

• Compute intensive operation
  – Local implementation is inaccurate
  – Remote implementation runs in parallel and uses better accuracy parameters

• Results
  – Up to 40 times less energy
  – Up to 60 times faster
  – Better recognition quality
On Tuesday

Cloud Outsourcing cont’d

Towards an Elastic Application Model for Augmenting the Computing Capabilities of Mobile Devices with Cloud Computing

Tactics-Based Remote Execution for Mobile Computing