Special Topics:
CSci 8980 Edge Computing
Outsourcing III

Jon B. Weissman (jon@cs.umn.edu)

Department of Computer Science
University of Minnesota
Parametric Analysis for Adaptive Computation Offloading
Motivation

• Optimal code generation is sensitive to execution instances such as execution options, input parameters and data files.

• Optimal program partitioning for computation offloading depends on tradeoff between the computation cost and communication cost.
Issues

• Create a static task graph, TCFG
• Prevent redundant data transfer
• Enforce data dependencies
• Program partitioning constraints
Example

```c
audio encoding application

f()
{
    for(j = 0; j < x; j++) {
        f1: for(i = 0; i < y; i++) {
            get audio sample p;
            inbuf[i] = p;
        }
    }
    g();
    f2: for(i = 0; i < y; i++) {
        q = outbuf[i];
        write q to output;
    }
}
}
```

What is different compared to our other outsourcing papers? Native code!
Partitioning

Given values of $x$, $y$, $z$, knowledge of compute and data transfer
=> Partition (based on parameters)
Nodes are tasks but not functions, why?

Too limiting: a function does a little I/O and a lot of computing …
TCFG

• Algorithm to build TCFG
  – tasks ~ basic blocks: separated by conditionals, function calls, etc.
  – arcs are dependencies

• GOAL: Given TCFG, compute task assignment, for each task $v$
  – $M(v) = 1$ (server)
    = 0 (client)
Data Validity

\[ V_{si}(v, d) : \text{Is data item } d \text{ valid on the server at the entry of task } v? \]

\[ V_{so}(v, d) : \text{Is data item } d \text{ valid on the server at the exit of task } v? \]

\[ V_{ci}(v, d) : \text{Is data item } d \text{ valid on the client at the entry of task } v? \]

\[ V_{co}(v, d) : \text{Is data item } d \text{ valid on the client at the exit of task } v? \]

How are these functions used?

To enforce correctness and eliminate redundant transfers

For each \( d \), compute \( V_{\{s/c\}{i/o}} \)
Tracking Data Accesses

\[ N_s(d) = \begin{cases} 
1 & \text{data item } d \text{ is accessed on the server} \\
0 & \text{data item } d \text{ is not accessed on the server} 
\end{cases} \]

\[ N_c(d) = \begin{cases} 
1 & \text{data item } d \text{ is accessed on the client} \\
0 & \text{data item } d \text{ is not accessed on the client} 
\end{cases} \]

Is data really needed on a machine?
Constraints

Read Constraint: If task $v$ has a (definite, possible or partial) upward exposed read of data item $d$, then $M(v) \Rightarrow V_{si}(v, d)$ and $\neg M(v) \Rightarrow V_{ci}(v, d)$.

Write Constraint: If data item $d$ is (definitely, possibly or partially) written in task $v$, then $M(v) = V_{so}(v, d)$ and $\neg M(v) = V_{co}(v, d)$.

Transitive Constraint: If data item $d$ is definitely not written in task $v$, then $V_{so}(v, d) \Rightarrow V_{si}(v, d)$ and $V_{co}(v, d) \Rightarrow V_{ci}(v, d)$.

Conservative Constraint: If data item $d$ is possibly or partially written in task $v$, then $M(v) \Rightarrow V_{si}(v, d)$ and $\neg M(v) \Rightarrow V_{ci}(v, d)$.

The data access states depend on the task assignments. Data are accessed on a host only if certain tasks that access the data are assigned to this host. Hence we have the following data access state constraint:

Data Access State Constraint: if data item $d$ is (definitely, possibly or partially) accessed within task $v$, then $M(v) \Rightarrow N_{s}(d)$ and $\neg M(v) \Rightarrow N_{c}(d)$. 
## Cost Factors

### Computation Cost
\[
\sum_{v \in V} M(v)c_s(v) + \neg M(v)c_c(v)
\]

### Data Communication Cost
\[
\sum_{(v_i,v_j) \in E,d} \neg V_{so}(v_i,d)V_{si}(v_j,d)csd(v_i,v_j,d) + \\
\neg V_{co}(v_j,d)V_{ci}(v_i,d)scd(v_i,v_j,d)
\]

### Task Scheduling Cost
\[
\sum_{(v_i,v_j) \in E} \neg M(v_i)M(v_j)c_{st}(v_i,v_j) + \neg M(v_j)M(v_i)c_{ct}(v_i,v_j)
\]

### Data Registration Cost
\[
\sum_d N_c(d)N_s(d)c_a(d)
\]

---

small
Symbolic Analysis

• Determine data sizes \((s)\) and execution counts \((r)\)
  – Needed for \(c_s, c_c, csd, scd\)

• \(r, s\) are functions of input parameter vector \(\lambda\) (e.g. input parameters)
Partitioning Problem

Problem 1: Assign binary values to variables $M$, $V_{si}$, $V_{so}$, $V_{ci}$, $V_{co}$, $N_s$ and $N_c$ subject to program partitioning constraints in section 2.4 and minimize the sum of the cost in (1) - (4).

The authors prove this can be cast into a min-cut optimization problem.
Min-Cut Model

S source node
exiting arcs = client computation cost

T sink node
entering arcs = server computation cost

I input
O output
Min-Cut

• What is the min-cut of a graph?
  – the smallest total weight of the edges which if removed would disconnect the source from the sink

• Why is this useful?
Min-Cut

Do not offload f or g

Statically solve based on ranges of $\lambda$
## Performance

<table>
<thead>
<tr>
<th>Program Name</th>
<th>Description</th>
<th>No. of Parameters</th>
<th>No. of Source Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>rawcaudio</td>
<td>ADPCM in Mediabench, Speech Compression</td>
<td>1</td>
<td>205</td>
</tr>
<tr>
<td>rawdaudio</td>
<td>ADPCM in Mediabench, Speech Decompression</td>
<td>1</td>
<td>178</td>
</tr>
<tr>
<td>encode</td>
<td>G.721 in Mediabench, CCITT Voice Compression</td>
<td>4</td>
<td>1118</td>
</tr>
<tr>
<td>decode</td>
<td>G.721 in Mediabench, CCITT Voice Decompression</td>
<td>4</td>
<td>1248</td>
</tr>
<tr>
<td>fft</td>
<td>FFT in Mibench, Discrete Fast Fourier Transforms</td>
<td>3</td>
<td>332</td>
</tr>
<tr>
<td>susan</td>
<td>susan in Mibench, Photo Processing</td>
<td>12</td>
<td>2122</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program</th>
<th>No. of Tasks</th>
<th>No. of Annotations</th>
<th>No. of Partitioning Choices</th>
<th>Analysis Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rawcaudio</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td>164</td>
</tr>
<tr>
<td>rawdaudio</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td>185</td>
</tr>
<tr>
<td>encode</td>
<td>107</td>
<td>4</td>
<td>4</td>
<td>2247</td>
</tr>
<tr>
<td>decode</td>
<td>87</td>
<td>4</td>
<td>4</td>
<td>2159</td>
</tr>
<tr>
<td>fft</td>
<td>26</td>
<td>3</td>
<td>2</td>
<td>748</td>
</tr>
<tr>
<td>susan</td>
<td>95</td>
<td>13</td>
<td>3</td>
<td>3482</td>
</tr>
</tbody>
</table>

37% improvement
Performance

<10% error
Discussion

• **Pros**
  – Native code
  – Fully compiler-driven approach

• **Cons**
  – Cannot respond to run-time dynamics
  – Server tasks and client tasks do not run simultaneously
  – Complexity
ThinkAir: Dynamic Resource Allocation and Parallel Execution in Cloud for Mobile Code Offloading
Motivation

• MAUI
  – Does not address the scaling of execution in cloud (opportunity!)

• CloneCloud
  – Offline benchmarks have to be built for every application
ThinkAir

• Smart phone virtualization in the cloud + method-level computation offloading
  – Parallelizing method execution using multiple VM images
    • On-demand resource allocation
  – Online method-level offloading
Design Goals

• Dynamic adaptation to changing environment
• Ease of use for developers
  – Annotate Java methods with @Remote
• Performance improvement through cloud computing
• Dynamic scaling of computation power
Architecture
Cloud Infrastructure

- OS: customized version of Android x86
- 6 types of VMs
Execution Controller

• Make offloading decisions
• Four policies (nice):
  – Execution time
  – Energy
  – Execution time and energy
  – Execution time, energy, and cost
• Uses profiling
Client Handler ~ cloud server

- **Code execution**
  - Manage connection
  - Execute code
  - Return results

- **VM management**
  - Add VM with more computing power/resources
  - Distributes task among VMs, and collects results
Experiments

• BIV (boundary input value)
  – The minimum value of the input parameter for which offloading would be beneficial

• Offloading policy: execution time

• Different Scenarios:
  – Phone
  – WiFi-Local (RTT 5ms)
    • router attached to cloud server
  – WiFi-Internet (RTT 50ms)
  – 3G (RTT 100ms)
## Micro-Benchmark Results

### Table II
Boundary input values of benchmark applications, with WiFi and 3G connectivity, and the complexity of algorithms.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>BIV WiFi</th>
<th>BIV 3G</th>
<th>Complexity</th>
<th>Data Tx (bytes)</th>
<th>Data Rx (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fibonacci</td>
<td>18</td>
<td>19</td>
<td>$O(2^n)$</td>
<td>392</td>
<td>307</td>
</tr>
<tr>
<td>Hash</td>
<td>550</td>
<td>600</td>
<td>$O(n^2\log(n))$</td>
<td>383</td>
<td>293</td>
</tr>
<tr>
<td>Methcall</td>
<td>2500</td>
<td>3100</td>
<td>$O(n)$</td>
<td>338</td>
<td>297</td>
</tr>
<tr>
<td>Nestedloop</td>
<td>7</td>
<td>8</td>
<td>$O(n^6)$</td>
<td>349</td>
<td>305</td>
</tr>
</tbody>
</table>

- Network latency clearly affects the BIV
Face Detection Results

Fig. 4. Execution time and energy consumed for the face detection experiments.

- Compares photo against DB or 10 or 100 photos
Face Detection Results (Cont.)

- 100 pictures

Fig. 5. Energy consumed by each component for face detection with 100 pictures in different scenarios.
Parallelization with Multiple VM Clones

- Workloads are evenly distributed among VMs

Fig. 7. Time taken and energy consumed on the phone executing 8-queens puzzle using $N = \{1, 2, 4, 8\}$ servers.

Fig. 8. Time taken and energy consumed for face detection on 100 pictures using $N = \{1, 2, 4, 8\}$ servers.

Fig. 9. Time taken and energy consumed for virus scanning using $N = \{1, 2, 4, 8\}$ servers.
Discussion

• Pros
  – VM selection + scaling/parallelization
  – BIV notion is nice

• Cons
  – Paper is somewhat weak on both ‘what’ and ‘how’ details
  – How is profile information used?
    • Seems to rely solely on past execution of a method + BIV
  – How is the parallelization done, who picks VM type?
On Thursday

Mobile Cloud Outsourcing Finished!

Cloudlets