Energy-Oriented Compiler
Optimizations for Partitioned
Memory Architectures

Power & Energy Management
Light Seminar
March 29, 2001
Motivation

- Processor market is estimated >90% in embedded systems
- Embedded systems’ applications spend up to 90% energy in memory system
Framework

• Partitioned memory architecture
  – Multiple memory banks (ie, 64x1MB, 32x2MB, 16x4MB, etc)
  – Multiple power modes

<table>
<thead>
<tr>
<th></th>
<th>Active</th>
<th>Standby</th>
<th>Napping</th>
<th>Power-Down</th>
<th>Disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy Consmp. (nJ)</td>
<td>3.570</td>
<td>0.830</td>
<td>0.320</td>
<td>0.005</td>
<td>0.000</td>
</tr>
<tr>
<td>Re-sync. Time (cyc.)</td>
<td>0</td>
<td>2</td>
<td>30</td>
<td>9,000</td>
<td>NA</td>
</tr>
</tbody>
</table>

Figure 1: Energy consumptions and re-synchronization times.
Optimizations

- Array Allocation
- Transformations (via SUIF infrastructure)
  - Loop fission
  - Loop splitting
  - Array renaming
Optimizations

- **Array Allocation:** Examine arrays with similar access patterns
  - Build Array Relation Graph (ARG)
    - Node == array
    - Edge weight == # of times incident arrays are accessed in loop
  - Find max weight cover
  - Place neighboring arrays into adjacent memory location
  - Bias towards large edge weights

```c
for (i=0; i<N; i++)
    {a[i], b[i]}
for (i=0; i<N; i++)
    {c[i], d[i], e[i]}
for (i=0; i<N; i++)
    {e[i], f[i]}
```
Optimizations

- Transformations

  - Loop Fission

```c
for (i=0; i<N; i++)
{
    {a[i], b[i]}
    {c[i], d[i]}
}

for (i=0; i<N; i++)
{
    {a[i], b[i]}
    {c[i], d[i]}
}
```

```plaintext
for(...) for(...) { S1 } for(...) for(...) { S1 S2 }
for(...) { S2 S3 }, for(...) { S2 S3 }, ...
for(...) { S3 S4 }, for(...) { S3 S4 }, ...
for(...) for(...) { S4 S5 }, ...
for(...) for(...) { S5 S6 }, ...
```

```c
for (i=0; i<N; i++)
{
    {a[i], b[i]}
    {c[i], d[i]}
}

for (i=0; i<N; i++)
{
    {a[i], b[i]}
    {c[i], d[i]}
}
```
Optimizations

- Transformations
  - Loop Splitting (Index Set Splitting)

\[
\begin{align*}
\text{for } (i=0; i<N; i++) & \quad \Rightarrow \quad \text{for } (i=0; i<N/2; i++) \\
\{ a[i], b[i] \} & \quad \Rightarrow \quad \text{for } (i=N/2 + 1; i<N; i++) \\
& \quad \quad \{ a[i], b[i] \}
\end{align*}
\]
Optimizations

- Transformations
  - Array Renaming (Live Variable Analysis)
    - Feasible if $a > b$ in size

```plaintext
for (i=0; i<N; i++)
  \{a[i], c[i]\}
......
for (i=0; i<N; i++)
  \{b[i], c[i]\}

\Rightarrow

for (i=0; i<N; i++)
  \{a[i], c[i]\}
......
for (i=0; i<N; i++)
  \{a[i], c[i]\}
```
Evaluation System

- SimplePower?
Evaluation System

- SimplePower?
- 64MB physical memory
Evaluation System

- SimplePower?
- 64MB physical memory
- No virtual memory
Evaluation System

- SimplePower?
- 64MB physical memory
- No virtual memory
- No cache
Evaluation System

- SimplePower?
- 64MB physical memory
- No virtual memory
- No cache
- Single program environment
Evaluation System

- SimplePower?
- 64MB physical memory
- No virtual memory
- No cache
- Single program environment
- Working set size < 64MB
Evaluation System

- SimplePower?
- 64MB physical memory (in most cases)
- No virtual memory
- No cache
- Single program environment
- Working set size < 64MB
- Mode Control
Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Source</th>
<th>Data Size (MB)</th>
<th>Bank Configuration</th>
<th>Energy Consumption (mJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>adi</td>
<td>Livermore</td>
<td>48.0</td>
<td>8×8MB</td>
<td>3.38</td>
</tr>
<tr>
<td>aps</td>
<td>Perfect Club</td>
<td>41.5</td>
<td>8×8MB</td>
<td>2.56</td>
</tr>
<tr>
<td>bmcm</td>
<td>Perfect Club</td>
<td>3.0</td>
<td>8×0.5MB</td>
<td>1,040.34</td>
</tr>
<tr>
<td>btrix</td>
<td>Spec’92</td>
<td>47.3</td>
<td>8×8MB</td>
<td>2.49</td>
</tr>
<tr>
<td>eflux</td>
<td>Perfect Club</td>
<td>33.6</td>
<td>16×4MB</td>
<td>826.46</td>
</tr>
<tr>
<td>matvec</td>
<td>[1]</td>
<td>16.0</td>
<td>8×8MB</td>
<td>675.86</td>
</tr>
<tr>
<td>mxm</td>
<td>Spec’92</td>
<td>48.0</td>
<td>8×0.5MB</td>
<td>10,572.57</td>
</tr>
<tr>
<td>phods</td>
<td>[3]</td>
<td>33.0</td>
<td>8×8MB</td>
<td>1,137.38</td>
</tr>
<tr>
<td>tomcatv</td>
<td>Spec’95</td>
<td>56.0</td>
<td>8×8MB</td>
<td>119.78</td>
</tr>
<tr>
<td>vpenta</td>
<td>Spec’92</td>
<td>60.0</td>
<td>32×2MB</td>
<td>2,026.66</td>
</tr>
<tr>
<td>wss</td>
<td>Perfect Club</td>
<td>3.0</td>
<td>8×0.5MB</td>
<td>7,032.03</td>
</tr>
</tbody>
</table>

- Usually 1 or 2 loop nests dominate energy consumption
- Focus on most costly nest
Results

• Array Allocation: Optimal solution is NP-Hard
  – Layout based on most costly nest
Results

- **Loop Fission**
  - Loops containing single instruction?
  - Second most costly nest?
  - Average improvement: 55.5%

Figure 11: Percentage energy improvements due to different loop fission alternatives.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
<th>#7</th>
<th>#8</th>
<th>#9</th>
</tr>
</thead>
<tbody>
<tr>
<td>adi</td>
<td>47.0%</td>
<td>47.0%</td>
<td>61.2%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aps</td>
<td>48.5%</td>
<td>48.5%</td>
<td>48.5%</td>
<td>48.5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eflux</td>
<td>47.0%</td>
<td>45.2%</td>
<td>43.3%</td>
<td>66.9%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>matvec</td>
<td>49.8%</td>
<td>33.2%</td>
<td>16.6%</td>
<td>58.2%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tomcatv</td>
<td>49.5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vpenta</td>
<td>8.2%</td>
<td>23.5%</td>
<td>16.6%</td>
<td>24.9%</td>
<td>18.0%</td>
<td>16.6%</td>
<td>20.7%</td>
<td>8.2%</td>
<td>48.4%</td>
</tr>
</tbody>
</table>

Table: Alternative Fission Strategies for the Most Costly Nest
Results

- Loop Splitting
  - 61.5% reduction vs. All ON
  - 42.8% reduction vs. Mode Control
Results

- **Array Renaming**
  - Only 2 benchmarks; expect more opportunities in larger codes
Memory Bank Configuration

- **Used eflux**: (showed best energy improvement via fission)
Cache

- Tested 4K, 2-way/4-way set-associative
- Results are comparable; why?
  - Reduces overall # of memory references
  - Longer interaccess reference times
Concluding Remarks