Optimal Program Partitioning for Predictable Performance

Jack Whitham and Neil Audsley
Real-time Systems Group
University of York
Program Partitioning

- Programs are typically larger than local memory
  - L1 cache ~ 16kb
  - L1 scratchpad memory (SPM) ~ 16kb
  - Typical program size?
- How can a large program make good use of local memory?
Program Partitioning

• Program is divided into *regions*
  • Regions consist of methods, basic blocks, loops...

• One region is in local memory at a time
  • Regions are small enough to fit in local memory

• If execution leaves one region, another region is loaded
Program Partitioning

- Cache: implicit partitioning
  - Program elements loaded on demand
- SPM: explicit partitioning
  - Algorithm required to divide large programs into regions
- Extra complexity... why use SPM?
  - Predictability
  - Performance
fdct

- MRTC benchmark
- 223 words (Microblaze, `mb-gcc -Os`)
- 56 cache misses OR one SPM load
**fdct**

- 56 cache misses @ 29 clock cycles each (on my FPGA)
  = 1624 clock cycles waiting

OR

- One SPM load of 223 words
  = 286 clock cycles waiting
One SPM load

Program runs → Latency → Data Received

Request 1
One SPM load

- SPM load is pipelined
One SPM load

- SPM load is pipelined

- After initial latency, the bus is never idle
56 cache misses

- Cache load is *not* pipelined

- Cache misses depend on the program control flow
56 cache misses

- Cache load is *not* pipelined

- Cache misses depend on the program control flow
• Measured WCET on FPGA platform
  • 4213 clock cycles with 256 word cache
  • 2903 clock cycles with 256 word SPM
• 45% faster (real hardware)

• But this is a small program
• Larger programs would require...
fdct

- Measured WCET on FPGA platform
  - 4213 clock cycles with 256 word cache
  - 2903 clock cycles with 256 word SPM
- 45% faster (real hardware)

- But this is a small program
- Larger programs would require...
  - Partitioning!
Partitioning a Call Tree
Call Tree Notation

- Method X calls method Y

```c
void X(void)
{
    ...
    Y();
    ...
}
```
Call Tree Notation

• Method X calls method Y 94 times

```c
void X(void) {
    ...  
    for(i=0; i<94; i++)
        Y();
    ...  
}  
```
Call Tree Notation

- Method sizes

  - Method X has size 100 words
  - Method Y has size 20 words
Partitioning Example

- Program containing 10 methods
- Total method size 424 words
- SPM size 128 words

- Minimise the cost of region transitions
- Enforce upper bound on region size
- 10 methods
- Total size 424 words
- SPM size 128 words
Region 1
All these methods are loaded together

- 17 + 34 + 58 = 109 words
- 17 transitions
- Why not include m4? (123)
Region 1 (109)

Region 4 (101)

Region 5 (85)

Region 2 (33)

Region 3 (96)

- Solved
- Minimum cost (18)
- Size limit (128) respected
Partitioning Algorithms

- Exhaustive search
- Greedy (min-cut)
- Greedy (merging regions)
- Dynamic programming
Dynamic Programming

- Program represented as a tree
  - Typically a call tree
- Partitions created from leaves to root
- Optimal partition in polynomial time!
Dynamic Programming

- Program represented as a tree
  - Typically a call tree
- Partitions created from leaves to root
- Optimal partition in polynomial time!
  - Optimal wrt. program representation and a single “typical” execution path (not necessarily worst-case path)
Lukes' Algorithm

- J.A. Lukes (1974) invented an $O(nk^2)$ algorithm for partitioning call trees

- For each subtree root and each possible root region size, memoise the optimal partition
Contributions of the paper
Result 1

• Lukes' algorithm does not generate optimal solutions when the cost of loading regions is taken into account

\[
\text{Lukes' cost:} \\
10
\]

\[
\text{SPM loading cost:} \\
10 \times \text{loading 58 words} + \\
10 \times \text{loading 34 words}
\]
SPM loading cost: 429

Optimised cost according to Lukes: 108
SPM loading cost: 429
SPM loading cost: 360

SPM loading cost: 429

(SPM size 2)
Algorithm 1

- New partitioning algorithm ELA-1 which includes region sizes in cost calculations
- Principal difficulty – cost calculations depend on the caller as well as callee
- Caller region size is unknown
ELA-1

• Unknown caller region size represented by $\alpha$
• For each subtree root and each possible root region size and each possible $\alpha$, store optimal partition
• Lukes: $O(nk^2)$
  ELA-1: $O(nk^3)$
  (up to $k$ possible values of $\alpha$)
Result 2

- Comparison of ELA-1 and cache
- Recall: fdct program
  - Single region
  - 4213 clock cycles with 256 word cache
  - 2903 clock cycles with 256 word SPM
  - 45% faster \((\frac{4213}{2903} = 1.45)\)
- Repeated experiment with other MRTC programs
Improved evaluation

• Smaller SPM: increase pressure
  • Tried exlining loops

• Separate loading time and execution time
  • Clearer results for long-running benchmarks
## ELA-1 vs Cache

<table>
<thead>
<tr>
<th></th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1024</th>
<th>2048</th>
<th>4096</th>
</tr>
</thead>
<tbody>
<tr>
<td>binarysearch</td>
<td>-</td>
<td>-</td>
<td>2.12</td>
<td>2.12</td>
<td>2.12</td>
<td>2.12</td>
<td>2.12</td>
</tr>
<tr>
<td>bsort100</td>
<td>-</td>
<td>0.76</td>
<td>1.50</td>
<td>2.14</td>
<td>2.14</td>
<td>2.14</td>
<td>2.14</td>
</tr>
<tr>
<td>crc</td>
<td>-</td>
<td>-</td>
<td>0.94</td>
<td>1.13</td>
<td>1.74</td>
<td>1.74</td>
<td>1.74</td>
</tr>
<tr>
<td>edn</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.40</td>
<td>1.12</td>
<td>1.45</td>
</tr>
<tr>
<td>fir</td>
<td>-</td>
<td>-</td>
<td>0.62</td>
<td>1.90</td>
<td>1.90</td>
<td>1.90</td>
<td>1.90</td>
</tr>
<tr>
<td>insertsort</td>
<td>-</td>
<td>-</td>
<td>0.19</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
</tr>
<tr>
<td>jfdctint</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.44</td>
<td>1.50</td>
<td>1.62</td>
<td>1.62</td>
</tr>
<tr>
<td>matmult</td>
<td>-</td>
<td>1.16</td>
<td>2.40</td>
<td>3.93</td>
<td>1.78</td>
<td>1.78</td>
<td>1.78</td>
</tr>
</tbody>
</table>

0.19 3.93

Loading times only
Loops exlined
Algorithm 2

• Problem: call tree representation
  • Sometimes, methods don't fit
    – Small SPM size
  • Whole methods are loaded even if parts are rarely/never used
    – c.f. “compress”
• Solution: ELA-2: an attempt to extend ELA-1 for general control-flow graphs
## ELA-2 vs Cache

<table>
<thead>
<tr>
<th></th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1024</th>
<th>2048</th>
<th>4096</th>
</tr>
</thead>
<tbody>
<tr>
<td>binarysearch</td>
<td>2.73</td>
<td>2.25</td>
<td>2.21</td>
<td>2.21</td>
<td>2.21</td>
<td>2.21</td>
<td>2.21</td>
</tr>
<tr>
<td>bsort100</td>
<td>2.55</td>
<td>0.94</td>
<td>2.09</td>
<td>2.22</td>
<td>2.22</td>
<td>2.22</td>
<td>2.22</td>
</tr>
<tr>
<td>crc</td>
<td>1.33</td>
<td>1.08</td>
<td>2.07</td>
<td>1.46</td>
<td>2.05</td>
<td>2.05</td>
<td>2.05</td>
</tr>
<tr>
<td>edn</td>
<td>1.17</td>
<td>1.32</td>
<td>1.19</td>
<td>1.82</td>
<td>2.11</td>
<td>1.95</td>
<td>1.69</td>
</tr>
<tr>
<td>fir</td>
<td>1.63</td>
<td>1.69</td>
<td>0.81</td>
<td>2.02</td>
<td>2.02</td>
<td>2.02</td>
<td>2.02</td>
</tr>
<tr>
<td>insertsort</td>
<td>2.03</td>
<td>1.59</td>
<td>1.95</td>
<td>2.08</td>
<td>2.08</td>
<td>2.08</td>
<td>2.08</td>
</tr>
<tr>
<td>jfdctint</td>
<td>1.50</td>
<td>1.48</td>
<td>1.33</td>
<td>1.94</td>
<td>1.91</td>
<td>1.81</td>
<td>1.81</td>
</tr>
<tr>
<td>matmult</td>
<td>1.94</td>
<td>1.47</td>
<td>2.75</td>
<td>5.79</td>
<td>1.85</td>
<td>1.85</td>
<td>1.85</td>
</tr>
</tbody>
</table>

0.19 ... ... ... ... ... ... ... ... ... ... ... 3.93 Loading times only
ELA-2

• ELA-2 is not widely applicable
  • Loops are a problem, and poorly handled
  • $O(2^{L}nk^3)$ time for $L$ loops (!)

• A better solution is required
  • Greedy heuristics may be the best-known solution so far
Conclusions

• Partitioning brings the performance and predictability benefits of SPM to larger programs

• Optimal algorithm ELA-1 specified
  • ELA-1 is very useful if a call tree can be partitioned effectively
  • Difficulties in generalising ELA-1 for control flow graphs (ELA-2)
Thankyou