Automatic mapping to tightly coupled memories and cache locking

Peter Marwedel\textsuperscript{1,2}, Heiko Falk\textsuperscript{1}, Robert Pyka\textsuperscript{1}, Lars Wehmeyer\textsuperscript{2}

\textsuperscript{1}TU Dortmund
\textsuperscript{2}Informatik Centrum Dortmund (ICD)

peter.marwedel@udo.edu

Problems with memory speeds

Speed gap between processor and main DRAM increases

Similar problems also for embedded systems & MPSoCs

In the future:
Memory access times $>>$ processor cycle times

“Memory wall” problem

[P. Machanik: Approaches to Addressing the Memory Wall, TR Nov. 2002, U. Brisbane]
Problems with memory energy

Memories a major consumer of energy:
Example: mobile phone

Caches consume much of the available energy

[O. Vargas (Infineon): Minimum power consumption in mobile-phone memory subsystems; Pennwell Portable Design - September 2005.] Thanks to Thorsten Koch (Nokia/ TU Dortmund) for providing this source.

[Segars 01 according to Vahid@ISSS01]
Tightly coupled memories/Scratch pad memories (SPM): Fast, energy-efficient, timing-predictable

TCM/SPMs are small, physically separate memories mapped into the address space; Selection is by an appropriate address decoder (simple!)

Address space

<table>
<thead>
<tr>
<th>0</th>
<th>scratch pad memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFF..</td>
<td></td>
</tr>
</tbody>
</table>

Example

ARM7TDMI cores, well-known for low power consumption

Small; no tag memory
Migration of data and instructions to TCM/SPM

Example:

Which objects (array, loop, etc.) to be stored in SPM?

Non-overlying ("static") memory allocation:
Objects always in TCM while application is running

Overlaying ("dynamic") allocation:
Moving objects back and forth between hierarchy levels
A survey of algorithms for scratchpad allocation

- Non-overlaying (“static”) approaches
  Gain $g_k$ & size $s_k$ for each object $k$.
  Maximise gain $G = \sum g_k$, respecting $SSP \geq \sum s_k$.
  Knapsack
  - Code, static data, stack, heap,
  - Partitioning, handling large arrays

- Overlaying (“dynamic”) approaches
  - single/multiple hierarchy levels
  - for single process
  - for static number of multiple processes
  - for dynamic number of multiple processes
  - not using/using MMU

## Partitioning

<table>
<thead>
<tr>
<th># of partitions</th>
<th>number of partitions of size:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4k</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Example of all considered memory partitions for a total capacity of 4096 bytes
Results for a non-overlaying approach: parts of GSM coder/decoder

Energy model: based on ARM evaluation board
Using these ideas with an gcc-based tool flow

Source is split into 2 different files by specially developed memory optimizer tool *.

*Built with new tool design suite ICD-C available from ICD (see www.icd.de/es)
Hybrid Context Switch

- Disjoint + Shared SPM regions
- Good for all scratchpads
Multi-process Scratchpad Allocation: Results

- For small SPMs (64B-512B) Saving is better
- For large SPMs (1kB-4kB) Non-Saving is better
- Hybrid is the best for all SPM sizes.
- Energy reduction @ 4kB SPM is 27% for Hybrid approach
Approach overview

- 2 steps: compile-time analysis & runtime decisions
- No need to know all applications at compile-time
- Capable of managing runtime allocated memory objects
- Integrated into an embedded operating system

Using MPArm simulator from U. Bologna
Comparison of SPMM to Caches for SORT

- Baseline: Main memory only
- SPMM peak energy reduction by 83% at 4k Bytes scratchpad
- Cache peak: 75% at 2k 2-way cache

- SPMM capable of outperforming caches
- OS and libraries are not considered yet

Chunk allocation results:

<table>
<thead>
<tr>
<th>SPM Size</th>
<th>Δ 4-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>74.81%</td>
</tr>
<tr>
<td>2048</td>
<td>65.35%</td>
</tr>
<tr>
<td>4096</td>
<td>64.39%</td>
</tr>
<tr>
<td>8192</td>
<td>65.64%</td>
</tr>
<tr>
<td>16384</td>
<td>63.73%</td>
</tr>
</tbody>
</table>
Worst case timing analysis using aiT

C program

memory-aware compiler

SPM size

executable

ARMulator

aiT

Actual performance

Worst case execution time
Results for G.721

Using Scratchpad:

Using Unified Cache:

Locking of I-Caches

Many caches allow “locking” or “freezing” (no replacements). Can be used to improve timing predictability:
- Load promising Functions into Cache

Optimizations:
- Worst case paths can change during optimization
- Requires coupling of timing analysis tool and compiler

[Heiko Falk, Sascha Plazar, Henrik Theiling: Compile-Time Decided Instruction Cache Locking Using Worst-Case Execution Paths, CODES/ISSS, 2007]
Relative WCETs after I-Cache Locking

- ADPCM
- G723
- Statemate
- Compress
- MPEG2

Cache Size [bytes]

Rel. WCET [%]

(ARM920T) 64 128 256 512 1024 2048 4096 8192 16384
More information

Conclusion

- Major impact of the memory system on system speed, energy consumption and timing predictability.
- Memory hierarchies comprising TCMs/SPMs are fast, energy-efficient and timing predictable.
- Algorithms have been designed for
  - Code, static data, stack, heap
  - Single and multiple memory hierarchy levels
  - non-overlaying and overlaying allocation
  - saving, non-saving and hybrid context switches
  - mono- and multiprocessor systems.
- Very large improvements in terms of the considered figures of merit.
- Compatible with existing tool flows