Offline Compression for On-Chip RAM

Nathan Cooprider and John Regehr
University of Utah
Microcontrollers (MCUs)

• 10 billion units / year
• $12.5 billion market in 2006

• Products with embedded computing are usually…
  – Low cost, high volume
  – Very sensitive to unit cost

• Bottom line: Pressure to use cheap MCUs
On-Chip RAM is Small

• Kilobytes, not megabytes or gigabytes
• Atmel AVR (8-bit RISC) examples
  - mega48 – 0.5 KB RAM – $1.50
  - mega128 – 4 KB RAM – $8.75
  - mega256 – 8 KB RAM – $10.66
• SRAM can dominate power consumption of a sleeping chip
Out of RAM – What Next?

- Remove application features?
- Buy MCUs with more RAM?
- Manually reduce RAM usage?
Out of RAM – What Next?

- Remove application features? (crossed out)
- Buy MCUs with more RAM?
- Manually reduce RAM usage?
Out of RAM – What Next?

- Remove application features?
- Buy MCUs with more RAM?
- Manually reduce RAM usage?
Out of RAM – What Next?

- Remove application features?
- Buy MCUs with more RAM?
- Manually reduce RAM usage?
A Closer Look

- Is RAM used efficiently?
  - Performed value profiling for embedded apps
    - Apps already heavily tuned for RAM usage
    - Result: Average byte stores four values!

- MCUs are Harvard architecture
  - Data in SRAM – 6 transistors / bit
  - Code in Flash “ROM” – 1 transistor / bit
  - 4–32x more ROM than RAM
RAM Compression

- Automated sub-word packing for statically allocated scalars, pointers, structs, arrays
  - No heap on most MCUs

- Driven by whole-program dataflow analysis
  - Sound for interrupt-driven concurrency

- Compression level can be tuned
Method

\[ x \text{ variable that occupies } n \text{ bits} \]
Method

\( x \) variable that occupies \( n \) bits

\( V_x \) conservative estimate of value set
Method

\[ x \text{ variable that occupies } n \text{ bits} \]

\[ V_x \text{ conservative estimate of value set} \]

\[ \left\lfloor \log_2 |V_x| \right\rfloor < n \Rightarrow \text{RAM compression possible} \]
Method

\( x \) variable that occupies \( n \) bits

\( V_x \) conservative estimate of value set

\( \lfloor \log_2 |V_x| \rfloor < n \Rightarrow \text{RAM compression possible} \)

\( C_x \) another set such that \( |C_x| = |V_x| \)
Method

\[ x \text{ variable that occupies } n \text{ bits} \]

\[ V_x \text{ conservative estimate of value set} \]

\[ \left\lceil \log_2 |V_x| \right\rceil < n \Rightarrow \text{RAM compression possible} \]

\[ C_x \text{ another set such that } |C_x| = |V_x| \]

\[ f_x \text{ bijection from } V_x \text{ to } C_x \]
Method

$x$ variable that occupies $n$ bits

$V_x$ conservative estimate of value set

$\lceil \log_2 |V_x| \rceil < n \Rightarrow$ RAM compression possible

$C_x$ another set such that $|C_x| = |V_x|

$f_x$ bijection from $V_x$ to $C_x$

$n - \lceil \log_2 |C_x| \rceil \Rightarrow$ bits saved through compression of $x$
Example Compression

```c
void (*function_queue[8])(void);
```
Example Compression

\[
\text{void (*function_queue[8])(void);} \\
\]

\[ n = \text{size of a function pointer} = 16 \text{ bits} \]
Example Compression

\[ x \quad V_x \]

\&function_A
\&function_B
\&function_C
NULL
Example Compression

\[ \|V_x\| = 4 \]

\[ \left\lfloor \log_2 |V_x| \right\rfloor < n \]

\[ 2 < 16 \]
Example Compression

\[ f_x \quad V_x \text{ to } C_x \quad \text{compression} \]

\[ f_x^{-1} \quad C_x \text{ to } V_x \quad \text{decompression} \]
Example Compression

\[ V_x = \{ \text{cloud}, \text{cloud}, \text{cloud}, \text{cloud}, \text{no symbol} \} \]
Example Compression

\[ V_x = \{ \text{clouds}, \text{cloud}, \text{cloud}, \text{no cloud} \} \]

128 bits reduced to 16 bits
112 bits of RAM saved
Implementation

• Source-to-source transformation for C
  – Rewrite declaration, initializer, reads, writes

• What about value sets of size 1?
  – Constant propagation (then DCE, DDE)

• Optimizations
  – Avoid table-driven compression funcs when possible
  – Align hot compressed values on word boundaries
  – Merge redundant compression tables
  – Compile-time compression when storing constants
RAM Compression Results

Change from compression:

- **Robot2**
- **GenericBase**
- **CntToLedsAndRfm**
- **Ident**
- **TinyDB**

Legend:
- Blue: duty cycle
- Green: code size
- Red: data size

*⇒ simulator unavailable
RAM Compression Results

Constant Prop / DCE
- 10% RAM reduction
- 20% ROM reduction
- 5.9% duty cycle reduction

Change from

Robot2
GenericBase
CntToLedsAndRfm
Ident
TinyDB

⇒ simulator unavailable
RAM Compression Results

**Compression**
- 22% RAM reduction
- 3.6% ROM reduction
- 29% duty cycle increase

**Constant Prop / DCE**
- 10% RAM reduction
- 20% ROM reduction
- 5.9% duty cycle reduction

⇒ simulator unavailable
Tuning RAM Compression

- Can elect to not compress some compressible variables
  - But which ones?
- For each compressible variable compute a cost / benefit ratio
  - Cost – estimated penalty in ROM or CPU cycles
  - Benefit – RAM savings
- Sort compressible variables by ratio and compress until some threshold is met
Cost/Benefit Ratio

\[ \sum C_i (A_i + B_i V) \]

- **C**: access profile
- **A,B**: platform-specific costs
- **V**: cardinality of value set
Cost/Benefit Ratio

\[
\sum C_i (A_i + B_i V)
\]

- **C**: access profile
- **A, B**: platform-specific costs
- **V**: cardinality of value set

\[
S_u - S_c
\]

- **S_u**: original size
- **S_c**: compressed size
Turning the RAM Knob

Percent change vs. Percent of compressible RAM compressed

duty cycle
code size
data size

0%
Turning the RAM Knob

![Graph showing the impact of turning the RAM knob on duty cycle, code size, and data size. The graph indicates a 10% decrease in the percent of compressible RAM compressed.]
Turning the RAM Knob

Percent change vs. Percent of compressible RAM compressed

- duty cycle
- code size
- data size

20%
Turning the RAM Knob

Percent change

Percent of compressible RAM compressed

-20 -15 -10 -5 0 5 10 15 20 25 30 35

-20 -15 -10 -5 0 5 10 15 20 25 30 35

-20 -15 -10 -5 0 5 10 15 20 25 30 35

duty cycle  code size  data size

30%
Turning the RAM Knob

Percent change

Percent of compressible RAM compressed

- duty cycle (dotted blue)
- code size (dashed green)
- data size (solid red)

40%
Turning the RAM Knob

Percent change

Percent of compressible RAM compressed

- duty cycle
- code size
- data size

50%
Turning the RAM Knob

- Duty cycle
- Code size
- Data size

60%
Turning the RAM Knob

70%
Turning the RAM Knob

Percent change

Percent of compressible RAM compressed

- duty cycle
- code size
- data size

80%
Turning the RAM Knob

Percent change vs. Percent of compressible RAM compressed

- Duty cycle
- Code size
- Data size

90%
Turning the RAM Knob

Percent change

-20  -15  -10  -5  0  5  10  15  20  25  30  35
Percent of compressible RAM compressed

-5  0  5  10  15  20  25  30  35

duty cycle  code size  data size

100%
Turning the RAM Knob

Percent change

Percent of compressible RAM compressed

duty cycle  code size  data size

95%
Compression Spectrum

Trading RAM for duty cycle

Percent change in code size

Percent change in duty cycle
Compression Spectrum

Trading RAM for duty cycle
Trading RAM for ROM

Percent change in code size

Percent change in duty cycle
Conclusion

- RAM likely to remain scarce in low-cost, low-power systems
- RAM is used inefficiently
- Manually tweaking data sizes (even among char, short, long, etc.) is unpleasant
- Useful to trade haves for have-nots

- CComp implements RAM compression

http://www.cs.utah.edu/~coop/research/ccomp/
## Analysis Times

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>HH:MM:SS</th>
<th>Benchmark</th>
<th>HH:MM:SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>drive1</td>
<td>00:00:13</td>
<td>sensetorfm</td>
<td>00:01:42</td>
</tr>
<tr>
<td>drive2</td>
<td>00:00:22</td>
<td>testtimestamping</td>
<td>00:02:09</td>
</tr>
<tr>
<td>osc</td>
<td>00:00:44</td>
<td>ident</td>
<td>00:03:13</td>
</tr>
<tr>
<td>genericbase</td>
<td>00:01:17</td>
<td>surge</td>
<td>00:03:42</td>
</tr>
<tr>
<td>ap</td>
<td>00:01:18</td>
<td>hfs</td>
<td>00:06:06</td>
</tr>
<tr>
<td>rfmtoleds</td>
<td>00:01:23</td>
<td>testtinysec</td>
<td>00:12:34</td>
</tr>
<tr>
<td>cnttoledsandrfm</td>
<td>00:01:32</td>
<td>tinydb</td>
<td>01:58:45</td>
</tr>
<tr>
<td>testdrip</td>
<td>00:01:42</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
No RAM Compression

- **duty cycle**
- **code size**
- **data size**

Change from optimization

<table>
<thead>
<tr>
<th>Component</th>
<th>Duty Cycle</th>
<th>Code Size</th>
<th>Data Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robot2</td>
<td></td>
<td>-10%</td>
<td></td>
</tr>
<tr>
<td>GenericBase</td>
<td></td>
<td>-20%</td>
<td>-10%</td>
</tr>
<tr>
<td>CntToLedsAndRfm</td>
<td>-20%</td>
<td>-30%</td>
<td>-20%</td>
</tr>
<tr>
<td>Ident</td>
<td>-30%</td>
<td>-40%</td>
<td>-30%</td>
</tr>
<tr>
<td>TinyDB</td>
<td></td>
<td>-10%</td>
<td></td>
</tr>
</tbody>
</table>

*⇒ simulator unavailable