Energy-Optimizing Source Code Transformations for Operating System-Driven Embedded Software

Yunsi Fei, S. Ravi, A. Raghunathan, and N.K. Jha


2008. 11. 5
AhRim Han
Contents

- Introduction
- Preliminaries: embedded system software model
- Overview of the approach
- Energy minimization through source code transformations
- Case studies
- Conclusion
- For my research
Limited battery life has made energy efficiency a critical issue.

Energy-efficient embedded software design is applicable to various levels of the design hierarchy:

- Instruction-level
- Algorithmic
- Program code
Traditional source code transformations
- Do not consider the effect of the OS
- Only applicable within a single process
- Do not target coarse-grained system-level concurrency and global data flow among multiple processes

However, in concurrent multiprocess programs, the following factors can significantly affect the overall energy consumption of an application
- Interprocess synchronization, data communication, context switches, OS intervention, etc.
In this paper, we propose a source code transformation methodology

- Employing source code transformation techniques to reduce the energy overheads associated with application/OS interactions
Studied system

- Processor: single (Intel StrongARM)
- OS: embedded Linux
- Software: written in C language, multiprocess programs
- Simulator: EMSIM\(^1\) - Instruction-level energy simulator
- IPC mechanisms for data communication
  - Pipe

---

Studied system (Cont’d)

- IPC mechanisms
  - Shared memory

```c
send process
mymsg sndmsg;
sndmsg.mtype = 100;
strcpy(sndmsg.mtext, "hello!");
msgsnd(queue, sndmsg, msgsz, flag)
```

```c
typedef struct msg{
    long mtype;
    char mtext[msgsz];
} mymsg;
```

```c
receive process
mym sg rcvmsg;
long type = 100;
msgrcv(queue, rcvmsg, msgsz, type, flag)
printf("%s\n", rcvmsg.mtext); //=> "hello!"
```

- Message queue
Control/data flow process network

Find-grained view - Functional call graph

Process level view - Processes, control/data flow and dynamic constructs (e.g., semaphores, IPC) are visible

Communication channel annotated with (volume/#IPC) tuple
- volume: average data volume communicated for a single communication instance
- #IPC: number of such communications

Input port connected to external h/w devices

Output port connected to external h/w devices
Overview

- Source code level energy minimization methodology

![Flowchart]

1. Extract control/data flow process network
2. Find possible transformations
3. Evaluate all transformations
4. Select best transformation
5. Apply transformation

Transformations & constraints

Energy consumption $E_0$ of the initial process network model can be obtained using EMSIM$^{[1]}$
1. Process network model extraction

- Generated automatically using advanced compilers
  - SUIF2 [1996, Stanford university]

- Steps
  - Defining the boundary of each process
  - Locating the IPC implementation as well as the communication between the process and its outer environment
    - Read (\texttt{read} and \texttt{msgrcv} system call) and write (\texttt{write} and \texttt{msgsnd} system calls) operations connected by one IPC channel form a communication pair
  - Determining the control flow between processes (synchronization)
2. Source code transformations

❖ Assumptions
  - Transformation only changes the view of the process network
    • Computation performed in each process remains the same
    • Overall functionality of the software program does not change

❖ Techniques
  - Process-level concurrency management
  - Message vectorization
  - Computation migration
  - IPC mechanism selection
Process-level concurrency management (1/2)

Objective

- Reduce the intervention of the underlying OS (e.g., Context switch, synchronization, data communication between processes) by minimizing the number of concurrent processes, while requiring each process to be efficient.
Process-level concurrency management (2/2)

Example: seat belt alert in cars

Control/data flow process network

State transition graph

<table>
<thead>
<tr>
<th>Source Code</th>
<th># Proc</th>
<th># Channels</th>
<th>Total Energy (mJ)</th>
<th>Energy Reduction (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>3</td>
<td>2</td>
<td>24.27</td>
<td>–</td>
</tr>
<tr>
<td>Transform 1 (merged with get_seatstate)</td>
<td>2</td>
<td>1</td>
<td>18.02</td>
<td>25.8</td>
</tr>
<tr>
<td>Transform 2 (merged with get_beltstate)</td>
<td>2</td>
<td>1</td>
<td>19.62</td>
<td>19.2</td>
</tr>
</tbody>
</table>

Comparison between original transformed source code

ⓒ KAIST SE LAB 2008
Objective

- Reduce the number of IPCs by vectorizing the messages and buffering the data communicated between two processes (IPC communication vs. process memory usage)
Message vectorization (2/2)

Example: DrawLine application

<table>
<thead>
<tr>
<th>Source Code</th>
<th>Total Energy ($mJ$)</th>
<th># Proc</th>
<th># Channels</th>
<th># Send/Receive IPC Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>8.24</td>
<td>2</td>
<td>1</td>
<td>1024</td>
</tr>
<tr>
<td>Optimized</td>
<td>7.17</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Reduction</td>
<td>13.0%</td>
<td>0</td>
<td>0</td>
<td>1023</td>
</tr>
</tbody>
</table>

Comparison between original and optimized source code

ⓒ KAIST SE LAB 2008
Objective

- Reduce the energy overheads resulting from synchronization and IPC by relocating computations from one process to another
Example: signal processing application

Inter- and intraprocess communication statistics for the 3 processes $P_1$, $P_2$, and $P_3$ with respect to function $\text{compute filter()}$
Computation migration (3/3)

Example: signal processing application (Cont’d)

Migrate `compute_filter()` to $P_1$

Migrate `compute_filter()` to $P_3$

<table>
<thead>
<tr>
<th>Source code configuration</th>
<th>Total energy ($mJ$)</th>
<th>#proc</th>
<th>#IPC</th>
<th>Volume of IPC data (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pipe</td>
<td>ShM</td>
<td>Msg</td>
<td></td>
</tr>
<tr>
<td>ORIGINAL</td>
<td>36.26</td>
<td>37.97</td>
<td>39.33</td>
<td>3</td>
</tr>
<tr>
<td>MIGRATION1</td>
<td>28.08</td>
<td>29.43</td>
<td>30.74</td>
<td>3</td>
</tr>
<tr>
<td>MIGRATION2</td>
<td>30.99</td>
<td>31.17</td>
<td>32.33</td>
<td>3</td>
</tr>
<tr>
<td>Optimized reduction</td>
<td>22.6%</td>
<td>22.5%</td>
<td>21.8%</td>
<td>0</td>
</tr>
</tbody>
</table>

Comparison between the original and transformed source code
Objective

- Choose the most energy-efficient mechanism for each IPC

Example: energy macromodels for 3 popular IPC mechanisms in embedded Linux

<table>
<thead>
<tr>
<th>IPC Mechanisms</th>
<th>Macromodels (nJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>shared memory (without protection) (x bytes)</td>
<td>$E = 67.3 + 71.33x$</td>
</tr>
<tr>
<td>shared memory (with semaphore protection) (x bytes)</td>
<td>$E = 2294.5 + 71.33x$</td>
</tr>
<tr>
<td>pipe write/read (x bytes)</td>
<td>$E = 1892.6 + 2.45x$</td>
</tr>
<tr>
<td>msgsnd/msgrcv (x bytes)</td>
<td>$E = 2518.6 + 4.41x$</td>
</tr>
</tbody>
</table>
3. Candidates evaluation

❖ One approach of design space exploration

✓ Method
  • Apply each candidate transformation to the source code obtained from last iteration
  • Generate the resultant program code
  • Utilize the low-level energy simulation framework to obtain energy consumption

✓ Problem
  • Need to perform a very slow low-level energy simulation for each transformation candidate
  • Need to apply all candidates of transformation to the source code obtained from the last iteration

Therefore, efficient high-level software energy macromodels that can capture the change in energy consumption for each move are desirable.
According to Tan et al.\cite{2} “the OS energy characteristic data provide a number of energy macromodels that express the relationship between the energy consumption of software functions and some characteristic parameters”.

**Two categories**

- **Explicit set**
  - System functions that are explicitly invoked by application software
    - e.g., IPC mechanisms and system calls (fork() and malloc())

- **Implicit set**
  - System functions that are not directly related to any OS primitives, instead the result the result of running the OS engine
    - e.g., Timer interrupt, scheduling, context switch, and signal handling

Evaluating candidate transformations

- Ex) Process merging
  - Two processes which are executed sequentially because one passes data or control to the other
    - Energy change $\Delta E$

$$\Delta E = -E_{ipc}(x)N_{ipc} - E_{ctx}N_{ctx}$$

- $E_{ipc}$: IPC energy
  - $E_{ipc}(x) = c_1 + c_2x$, $x$ is the number of bytes being transferred in each pair of calls
- $N_{ipc}$: Number of times that IPC is invoked
- $E_{ctx}$: Context switch energy
- $N_{ctx}$: Number of times that context switch is occurred
Case studies: Ethernet packet-processing system (1/4)

Transformation result

Used transformations: Process merging and vectorization

Original process network

Transformed process network

<table>
<thead>
<tr>
<th>Source Code</th>
<th>#Proc</th>
<th>#Channel</th>
<th>#IPC</th>
<th>IPC Volume (bytes)</th>
<th>Energy (mJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>4</td>
<td>2</td>
<td>200</td>
<td>52,000</td>
<td>25.8</td>
</tr>
<tr>
<td>Process merging</td>
<td>3</td>
<td>1</td>
<td>100</td>
<td>26,000</td>
<td>21.2</td>
</tr>
<tr>
<td>Message vectorization</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>26,000</td>
<td>18.6</td>
</tr>
</tbody>
</table>
Case studies: Ethernet packet-processing system (2/4)

**Buffer size analysis**

- Achieves highest energy reduction (27.9%) at buffer size of 20 packets
- Indicates trade-off between system memory usage (buffer size) and IPC overhead

Energy consumption variation with buffer size (after process merging)

© KAIST SE LAB 2008
## Case studies: Underwater navigation system (3/4)

### Transformation result

![Original process network](image1)

![Transformed process network](image2)

### Optimized statistics for original and transformed source codes

<table>
<thead>
<tr>
<th>Source Code</th>
<th>Trans</th>
<th># Proc</th>
<th># Channel</th>
<th># IPC</th>
<th>IPC Volume (bytes)</th>
<th>Energy (mJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>—</td>
<td>7</td>
<td>5</td>
<td>50</td>
<td>16640</td>
<td>53.3</td>
</tr>
<tr>
<td>Ver2</td>
<td>PM</td>
<td>6</td>
<td>4</td>
<td>40</td>
<td>11520</td>
<td>44.1</td>
</tr>
<tr>
<td>Ver3</td>
<td>PM</td>
<td>5</td>
<td>3</td>
<td>30</td>
<td>6400</td>
<td>37.5</td>
</tr>
<tr>
<td>Ver4</td>
<td>MV</td>
<td>5</td>
<td>3</td>
<td>21</td>
<td>6400</td>
<td>35.9</td>
</tr>
<tr>
<td>Ver5</td>
<td>REPL</td>
<td>5</td>
<td>3</td>
<td>21</td>
<td>6400</td>
<td>34.7</td>
</tr>
<tr>
<td>Ver6</td>
<td>MV</td>
<td>5</td>
<td>3</td>
<td>12</td>
<td>6400</td>
<td>33.8</td>
</tr>
<tr>
<td>Ver7</td>
<td>MV</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>6400</td>
<td>33.1</td>
</tr>
</tbody>
</table>

Optimized statistics for original and transformed source codes

- Reduction: 61.5%
- Reduction: 37.9%
- Achieves similar energy saving in both cases of final transformed code

- Needs smaller transformations in our framework than alternative scenario

- Suggests considering process merging transformations first (since process merging influences the IPC and tends to have the largest impact on energy)
Propose four types of source code transformations for OS-driven embedded software programs to reduce energy consumption

- Process-level concurrency management
- Message vectorization
- Computation migration
- IPC mechanism selection
For my research: Energy-optimizing model transformations

- Problems to solve
  - Model synthesis
  - Energy-aware model transformations
    - Define rules using ATL (Atlas Transformation language), QVT, etc.
  - Transformed models evaluation
    - Ex) Trade-off analysis between process communication and memory usage
  - Best model selection
  - Result of transformation application
  - Model matching with reversed model from optimized source codes for validation
Thank You.