Minimizing Embedded Software Power Consumption Through Reduction of Data Memory Access

Shan Li, Edmund M-K. Lai
School of Computer Engineering
Nanyang Technological University
Nanyang Avenue, SINGAPORE 639798

Mohammed Javed Absar
STMicroelectronics Asia Pacific Pte. Ltd.
20 Science Park Road, SINGAPORE 117674

Abstract

Software applications that involve multimedia signal processing typically have to process large amounts of data. They often involve the handling of data arrays in the form of nested loops. Experiments show that for this kind of applications data transfer (memory access) operations consume much more power than data-path operations. Our objective is to reduce memory access related power consumption by reducing the number of data transfers between processor and memory, or between a higher (closer to processor) level of memory and a memory at a lower level using source program transformation. The procedure involves profiling, inlining and global transformation. The effectiveness of this procedure is illustrated by applying it to the software for a wideband adaptive multi-rate (WB-AMR) speech decoder which can be obtained from the official website of the 3rd Generation Partnership Project (3GPP).

1. Introduction

Power has been an increasingly important cost factor in embedded system designs. One of the reasons is that portable hand-held devices require long battery life. Moreover, high power also means costly packaging and cooling requirements, and lower reliability. Consequently, when designing an embedded system, power-efficient design has become a critical concern.

Power consumption of an embedded application is a complex function of several components. We could analyze the problem at several levels and devise strategies for power reduction that takes into account one or more of these levels [7]. Low-level techniques like voltage scaling, clock frequency scaling, clock-gating [8] and pipeline gating [9] have been known to be very useful. At the application level, source code transformation using optimizing or restructuring compilers have been gaining prominence lately. The advantage of optimizing compilers lies in the fact that they are not application specific. However, different classes of applications and different kinds of target platforms require slightly different types of optimizations. A scientific application, which contains large amounts of computations, needs an optimizing compiler that could parallelize or vectorize the application (e.g. compilers for Cray-T4). Applications running on networked computers require optimization for minimum message passing. For embedded systems, especially those running multimedia applications, the requirement is for methods that optimize data transfers. This will be the focus of this paper.

Compiler optimizations can broadly be divided into two levels. At the lower level, the translation is more hardware-dependent, which limits the usage of the compiler to a certain class of processors. Higher level compiler optimization techniques perform source-to-source transformations. Typically, loop transformations and data transformations are performed on the source codes, with the output in the same language as the input source. Considerable research has been done in the area of loop and data transformation for data locality and increased parallelism. A good example is the decade-long SUIF project [17] at Stanford University. The main objective of our work, however, is in applying these high-level compiler techniques for the purpose of reducing embedded software power.

Our specific target of power reduction is multimedia applications, which are data-dominated. This kind of applications access large data streams represented by arrays inside (nested) loops. Power consumption of this kind of applications is largely contributed by data transfer and memory access operations [1,2,3]. In contrast, data-path operations consume much less power. Hence, a method should be found to reduce the power consumption of the data memory.

Our solution is to minimize data transfer and memory access operations in multimedia applications, which is equivalent to minimizing array accesses inside (nested) loops. The way to achieve this goal is to perform the high-level (source-to-source) transformation on arrays inside loops, such that multiple accessed data can be stored into registers and redundant memory accesses are minimized. The procedure involves profiling, inlining and global transformation.

The three procedures are described in more detail in Section 2. They are applied to the WB-AMR speech decoder application and their effectiveness in power
reduction is discussed in Section 3, followed by the conclusions.

2. Approach

2.1 Profiling

Starting from the source code of an application, which is currently restricted to be written in the “C” programming language here, a profile of the real-time execution of the program is required. In the profile, the number of the data array reads and writes for various parts of the program is first obtained. This counting process is done by means of a software profiling tool. From the profile, the areas of heavy data array access (referred to as bottlenecks later) and the call-path of these arrays are identified. These bottlenecks are candidates for later transformations.

2.2 Inlining

Function inlining replaces a call to a function or a subroutine with the body of the function or subroutine. In writing procedural programs, the concept of functions provides clarity of programming and ease of debugging. However, as far as program optimization is concerned, the drawback of functions is that they may act as brick walls between sections of code. The aim of inlining is to remove these brick walls and to enlarge the exploration space for optimization. At the end of this step, an inlined version of the source code based on the profile is obtained.

In the experiment, a call-based inlining scheme is adopted. Here, the decision to inline may be made independently at each call-site (where a function call is raised). With the bottlenecks identified in the previous step, the call-sites at these bottlenecks are candidates for inlining. The criteria of inlining is that it should facilitate the global transformation step within acceptable code size. For example, if there is redundant access to a common array in both the calling function and the called function, the called function could be inlined into this calling function to enable memory access reduction transformations. The main drawback of inlining is that it will increase the code size. It is demonstrated in [5] that maximizing the reduction in function calls under code size constraint is NP-hard. In the experiment, only the bottleneck functions are evaluated for inlining.

2.3 Global Transformation

In order to obtain improvements in memory access reduction, loop transformations are applied to the inlined version of the source code. This transformation process is currently performed manually.

Memory access is reduced if the number of the load instruction and the store instruction is reduced. This can be achieved by increasing the exploitation of array data reuse in space (spatial reuse) and data reuse in time (temporal reuse). Using multiple data points of a cache line before the line is replaced with some other line is an example of spatial reuse. Temporal reuse means multiple usage of the same data very close in time.

Reuse is something inherent in the computation and does not depend on the particular way the loops are written [13]. Loop transformation aims to obtain a better data locality and increase the exploitation of reuse. For example, an array data is written first and followed by a read some time later after a few iterations. There is reuse of this array data. If the write and the read are too far away in time, such that the data has to be put into memory after the write and before the read then there is poor locality. However, if a loop transformation can be performed to bring the read and the write closer (better locality), then the data can be put into register. In this example, the memory access is reduced by exploiting temporal reuse.

In the experiment, different types of loop transformations are applied to increase reuse and register usage. The feasibility and the extent of applying transformation are constrained by data dependence, number of registers available and the increment in the code size. Experiment shows that loop merging and loop unrolling along with scalar replacement are the most effective transformations in reducing memory access. The following of this section will use the filtering operation as an example and discuss how to apply unrolling and scalar replacement on it to reduce memory access.

Filtering is one of the most common signal processing operations. It is helpful to have a method to minimize the number of memory accesses for this frequently used operation. As an example, we consider the forward prediction:

\[ y[n] = \sum_{i=0}^{D} h[i]x[n-i] = h[0]x[n] + h[1]x[n-1] + \ldots + h[D]x[n-D+1] \]

\[ n = 1, \ldots, F \]

\( D \) is the filter order. \( F \) is the frame length. \( y[n] \) is the signal, which is computed from the \( D \) samples of the input signal \( x[n] \). \( h[i] \) is the prediction parameter. The “C” code for this filter operation is shown in figure 1(a), which consists of a double-loop. During the unrolling process, unrolling factors are chosen for both the outer loop and the inner loop, denoted by \( u1 \) and \( u2 \) respectively. The statement \( S \) in the inner loop is unrolled to \( u1 \ast u2 \) statements. The unrolled code, figure 1(b), reveals the redundant access to the array \( x \) illustrated by the dotted diagonals in figure 1(c). On each dotted line, the same array element is accessed. The number of accesses is the number of black nodes on a diagonal, black node represents one iteration. These redundant accesses are reduced by scalar replacement, which requires \( 2 \ast u1 + u2 \).
additional registers (assuming scalars are stored in
registers). The choices of $u1$ and $u2$ depend on the number
of registers available and the constraints on the size of the
unrolled code. For this example, they are set to 3. The
memory access for the arrays is then reduced to a fraction
of the original number of accesses. It is
$\frac{(u1 + u2 - 1)}{(u1 * u2)} = \frac{5}{9}$ for $x$ and $1/u1 = 1/3$ for $h$.

(c): Illustration of $u1$ and $u2$

Although loop unrolling increases the code size, it is
effective in power reduction. This is because loop
unrolling reduces the loop overhead and increases the
amount of computation per iteration. From the power
point of view, fewer computations mean less power
dissipation [6]. Together with scalar replacement, the
power consumption is further reduced by less memory
accesses.

Scalar replacement increases the register usage, which is
the most effective way of reducing memory operands [10].
Register operand also has shorter running times due to
elimination of potential stalls and cache misses.

The example above shows how to apply loop
transformations to increase register usage. A very
important constraint of loop transformation is that it should
not violate the data dependences in the original source
code [12]. When applying loop transformations such as
interchange, skewing and reversal, the data dependences
should always be preserved.

Figure 2 Loop unrolling followed by scalar replacement

In the code above (Figure 3) the distance vectors for array
$x$ are $\{(1, 0), (0, 1)\}$. All the distance vectors are positive.
After any transformation, the data dependence should still
be positive. That is to say, the polarity of the distance
vectors remains constant, only the absolute values of the
distance vectors can change. For illustration, loop skewing
is applied on this piece of code to improve data locality.
The transformed code is shown in Figure 4.

Figure 3 Simple code for transformation

(b): After unrolling

Figure 4 Code after loop skewing
The codes in Figure 3 and Figure 4 are equivalent in terms of the final result. This is shown in Figure 5.

The graph on the left hand side shows the data dependence of the original code, and the right hand side is the new data dependence of the transformed code. The new distance vectors after loop skewing are \{(1, 1), (0, 1)\}, which are positive vectors. Hence, the data dependence is said to be preserved.

### 3. Example

We applied the procedures to the entire WB-AMR speech decoder program [4], which has about 15,000 lines of code. This codec has nine transmission modes. Table 1 shows the number memory accesses (reads and writes) for all nine modes of operation before and after the optimization.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Org. Total Accesses</th>
<th>Opt. Total Accesses</th>
<th>Reduction (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>25021926</td>
<td>18140790</td>
<td>27.5</td>
</tr>
<tr>
<td>1</td>
<td>23330696</td>
<td>16763844</td>
<td>28.1</td>
</tr>
<tr>
<td>2</td>
<td>22436274</td>
<td>16066076</td>
<td>28.4</td>
</tr>
<tr>
<td>3</td>
<td>22465554</td>
<td>16089769</td>
<td>28.4</td>
</tr>
<tr>
<td>4</td>
<td>22493862</td>
<td>16113250</td>
<td>28.4</td>
</tr>
<tr>
<td>5</td>
<td>22536148</td>
<td>16155028</td>
<td>28.3</td>
</tr>
<tr>
<td>6</td>
<td>22561164</td>
<td>16177250</td>
<td>28.3</td>
</tr>
<tr>
<td>7</td>
<td>22617641</td>
<td>16229155</td>
<td>28.2</td>
</tr>
<tr>
<td>8</td>
<td>26815942</td>
<td>18339204</td>
<td>31.6</td>
</tr>
</tbody>
</table>

The results show that the percentage reduction in memory access for the nine modes is between 27.5% and 31.6%.

### 4. Conclusions

In this paper, we proposed a novel solution to the problem of power reduction for embedded software. The specific target of the research is multimedia applications, in which case data memory access is a big contributor to the system power. The approach presented here performs power reduction by minimizing the amount of memory accesses through high-level compiler transformations on the source program. It involves three steps: profiling, inlining and global transformation. The effectiveness of this approach is demonstrated by applying it to a practical application: WB-AMR speech decoder. On-going work includes the incorporation of these procedures into a compiler.

### Acknowledgements

The authors would like to thank the Centre for Multimedia and Network Technology in the School of Computer Engineering, Nanyang Technological University and the Audio DSP Systems Group of STMicroelectronics Asia Pacific Pte Ltd for the support provided for this research work.

### References


