Future Generation Computer Systems 34 (2014) 66–75

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Power-aware code scheduling assisted with power gating and DVS

Cheng-Yu Lee, Tzong-Yen Lin, Rong-Guey Chang *

Department of Computer Science and Information Engineering and the Advanced Institute for Manufacturing with High-tech Innovations, National Chung Cheng University, Chia-Yi, Taiwan, ROC

HIGHLIGHTS

- Our work applies power gating and dynamic voltage scaling together to save power.
- We build a model to analyze a code and set parameters in code scheduling.
- Our approach can outperform hardware power gating in terms of EDP and ED^{2}P.
- Our work can be applied to any compiler with architectural support.

ARTICLE INFO

Article history:
Received 25 April 2013
Received in revised form
25 November 2013
Accepted 3 December 2013
Available online 27 December 2013

Keywords:
DVS
Power gating
Code scheduling
Compiler

ABSTRACT

Traditionally, code scheduling is used to optimize the performance of an application, because it can rearrange the code to allow the execution of independent instructions in parallel based on instruction level parallelism (ILP). According to our observations, it can also be applied to reduce power dissipation by taking advantage of the properties of existing low-power techniques. In this paper, we present a power-aware code scheduling (PACS), which is a code scheduling integrated with power gating (PG) and dynamic voltage scaling (DVS) to reduce power consumption while executing an application. In other words, from the viewpoint of compilation optimization, PG and DVS can be applied simultaneously to a code and their impact can be enhanced by code scheduling to further save power. The result shows that when compared with hardware power gating, the proposed PACS can outperform by more than 33% and 41% in terms of energy delay product and energy delay^2 product for DSPStone and Mediabench.

1. Introduction

Global code scheduling is always used to improve performance across basic blocks by means of interprocedural analysis. Although it can be carried out by incorporating with other optimization techniques to enhance its impact, its ability would still be limited by instruction level parallelism (ILP). In our experience we found that it can also be applied to exploit opportunities to perform low-power optimizations. In this paper, we propose a global code scheduling named power-aware code scheduling (PACS), which is a code scheduling integrated with power gating (PG) and dynamic voltage scaling (DVS) to reduce power consumption while executing an application. The reason for choosing PG and DVS as the part of PACS is because they are two common effective low-power techniques.

In traditional code scheduling these opportunities are uncovered depending only on ILP. With PACS, however, in addition to the ILP it also depends on the features of existing low-power techniques to save considerable power. To perform the proposed PACS, we partition an application into many regions based on their use of functional units. Then the code scheduling is applied to maximize the idle period of a functional unit and apply PG to turn off idle functional units of each region. This phase aims at minimizing the switching overheads when applying PG to save power. In addition, modern processors are always designed with multiple power modes to support DVS. Consequently, the transitions among power modes also result in power dissipation. At this stage, the code scheduling is applied again to minimize the number of transitions during DVS. On the one hand, in order to let PACS take advantage of the PG, we add new instructions into the architecture to turn the functional units on and off. Normally, the functional units such as adder and multiplier might be turned on during execution. However, if we study the behavior of an application it is evident that this can lead to unnecessary power consumption. For example, if an application contains *n* distributed code segments that only perform additions, then this implies that only adders need to be activated in them at run time. To save power, we can apply the PACS to merge them together if no dependencies exist among them and then turn on the adders only once in these code segments. On the other hand, since the power dissipation is closely related to voltage and clock frequency, the architectures of modern processors provide several power modes to reduce power dissipation by dynamically adjusting the voltage and the clock frequency [1]. Previous

* Corresponding author.
E-mail addresses: lcy97p@cs.ccu.edu.tw (C.-Y. Lee), lty93@cs.ccu.edu.tw (T.-Y. Lin), rgchang@cs.ccu.edu.tw (R.-G. Chang).

0167-739X/$ – see front matter Crown Copyright © 2014 Published by Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.future.2013.12.011

a

```
int main(void)
{
    ...
    for (i = 0; i < 100; i++)
        a = a + 1;

    temp = a * c;

    for (j = 0; j < 200; j++)
        b = b + 2;

    temp2 = b * d;
    ...
}
```

b

```
int main(void)
{
    ...
    TURN_ON   adder
    TURN_OFF  multiplier
    for (i = 0; i < 100; i++)
        a = a + 1;
    for (j = 0; j < 200; j++)
        b = b + 2;

    TURN_OFF  adder
    TURN_ON   multiplier

    temp = a * c;
    temp2 = b * d;
    ...
}
```

Fig. 1. Motivating example.

work has shown that the switching overhead of the power modes has a significant impact on power dissipation [2,3]. After reducing the switching overhead of the functional units, we assign power modes to the regions, as described later. Then we apply PACS to this code again. The regions that have no dependencies among them and with the same power mode are then merged into a new region. Thus the proposed PACS can continue minimizing the number of transitions arising from DVS to save power.

Obviously, the scheduling process described in the above two stages is very complicated since it must check the dependency relationships among regions. The scheduling process is described in detail in the following sections. Our work implements the compilation system based on the SUIF [4] compiler infrastructure and the Wattch simulator [5]. The experiments shows that the proposed PACS can save power and in addition slightly improve performance for DSPstone, and Mediabench benchmark suites. The remainder of this paper is organized as follows. Section 2 presents our main idea and provides a motivating example. Section 3 describes the proposed PACS algorithm in detail and the experimental results are given in Section 4. Section 5 contains related work and Section 6 concludes.

2. Basic idea

Fig. 1 shows an example to explain the basic idea of this paper, where Fig. 1(a) is the original code segment and Fig. 1(b) is the optimized code segment of Fig. 1(a). To demonstrate their differences in power dissipation, we make the following assumptions. (1) There are four regions in the original code, two loops and two expressions that do the multiplications. (2) Initially, we assume that the adder is turned on since some instructions such as load and store will use it to calculate the target address. The multiplier is turned off. (3) We provide two power modes, normal mode and power down mode. The normal mode is set to the default power mode. (4) Two loops are assigned to the power down mode since they perform additions many times and two multiplications are assigned to the normal node. The assignments of power modes will be explained in detail in Section 3. (5) To reduce power dissipation, the functional units are only turned off when they are not used. In Fig. 1(a), following the above assumptions, the adder will be turned on and the multiplier will be turned off in two loops, and the adder will be turned off and the multiplier will be turned on when doing multiplications. In Fig. 1(b), to begin with, two loops and two expressions can be merged into two regions respectively since they contain the same functional unit and they have no dependencies. Next, we turn off the multiplier in the region containing two loops and then turn off the adder in the region containing two expressions. Finally, we assign the region containing two loops the power down mode and assign the region containing two expressions the normal mode. In contrast with Fig. 1(a), the code segment in Fig. 1(b) can reduce more power consumption since it maximizes the idle periods of the adder and the multiplier and also saves two transitions of power modes during DVS.

3. Proposed power-aware code scheduling (PACS)

In this section, we first describe how an application is partitioned into regions, then we introduce the architectural support for PG and DVS, and finally we present our PACS algorithm.

3.1. Region partitioning

To easily perform our PACS, we must partition a given application into many small parts based on the use of functional units. In this paper, these parts are called regions and they are optimization units for PACS. To achieve this objective, we follow the compilation process to translate an application into the intermediate representation (IR) at the front end of the compiler and then IR is used to build the control flow graph (CFG) and the data flow graph (DFG). Since CFG and DFG are comprised of many basic blocks, which are straight code segments without branches, we analyze each basic block and partition it into regions based on the use of the functional unit. For example, there are four regions in Fig. 1(a) based on additions and multiplications. To perform PACS, we must record the information of each region including its scope, the functional units used in it, the power information, and the dependency relationship. Initially we record the functional units used in each region and insert instructions in the head of each region to turn off idle functional units. Then we apply the first PACS to reschedule the code based on the dependency relationship among regions. At this stage, the independent regions that have the same idle functional units will be merged together by PACS. Next, PACS is used to reduce the power dissipation with respect to DVS. Each region will be assigned a power mode based on its power information. Once again, the independent regions that have the same power mode will be merged together by PACS. These details of PACS are described in Section 3.3.

3.2. Architectural support for PG and DVS

After the region partitioning is finished, we can apply PACS to each region via interprocedural analysis. To perform PG on each