Sustainable Computing: Informatics and Systems 12 (2016) 34–42

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

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

Software power modeling method at architecture level based on complex networks

Deguang Li [a], Bing Guo [a,*], Yan Shen [b], Junke Li [a], Yanhui Huang [a]

[a] College of Computer Science, Sichuan University, Chengdu 610065, China
[b] School of Control Engineering, Chengdu University of Information Technology, Chengdu 610225,China

ARTICLE INFO

Article history:
Received 19 January 2016
Received in revised form 30 June 2016
Accepted 30 August 2016
Available online 4 November 2016

Keywords:
Software power modeling
Architecture
Complex networks
BP neural network
Network characteristics of software

ABSTRACT

The architecture of software systems can be naturally represented in the form of complex networks, especially for object-oriented software. Software as a kind of artificial complex networks, where entities of the software are nodes and interactions between entities are edges. These interactions are data-flows, instruction-flows and control-flows of the software, and these flows driving hardware circuit is the internal cause of power consumption of software. In this paper, we model software systems as complex networks at architecture level, assuming that the relation between the network characteristics of software and its power consumption is nonlinear. Based on this assumption, we propose a software power modeling method at architecture level. The model first measures network characteristics of software and then fit the nonlinear relation between the network characteristics of software and its power consumption by BP neural network. Experimental results show that our model could accurately estimate power consumption of the software, and the error is less than 11.2% compared to the measured value, which indicates our assumption is reasonable and our model is effective.

1. Introduction

In recent years, power dissipation and energy saving have acquired comprehensive concerns. According to a report of Gartner entitled "Green IT: a new wave of industry impact" in 2007, carbon emissions of ICT (Information and Communications Technology) equipment one year accounts for about 2% of global carbon emissions, they pointed out that when we Google once, the energy consumed is equal to that consumed by boiling half pot of water [1]. Meanwhile, the power consumed by ICT equipment accounts for 10% of all power consumption of the United Kingdom in 2008, and 8% for the United States. With internet of things, cloud computing and new information technology applications developing, global ICT industry is still growing rapidly. According to a forecast by METI in 2010 [2], the power consumed by ICT equipment is forecasted to reach about 20% of all power consumed worldwide by 2025.

Computer system is a typical system controlled by software, which directly drives the underlying hardware. For example, different instruction execution, data access operation and other operations of the software drive the underlying hardware circuit, which indirectly result in power generation. Thus we can conclude that software is the consumer and manager of hardware, software itself does not cause energy consumption, energy consumption is "by-products" of software execution. Also we can say that software consumes energy actively and determines energy consumption of computer system. In order to have a better understanding and improve energy efficiency of computer system, much research has been done from different aspects. Such as upgrade manufacturing process, change circuit structure at circuit level [3,4]; compiler optimization, instruction transformation, instruction rearrangement, loop structure optimization and power analysis at instruction layer [8–11]; expression changes, optimizing data representation, program structure rearrangement, elimination of redundant computation, compressed data storage space, algorithm selection and estimation of energy consumption and execution time at source code level [10–15]; estimating the energy consumption of pervasive Java-based software and distributed Java-based at component level [20,21] and fine-grained power management using process-level profiling at process level [22–24]; high-level software energy macro-modeling, system structure selection, transformation and simplification at architecture level [24,21]; and power management technique at system level, such as DPM, DVS, RTOS task scheduling, cooperative game theoretical technique which presents dynamic method for voltage-scaling based task scheduling for simultaneous optimization of performance, energy, and tem-

* Corresponding author.
E-mail addresses: lideguang.00@163.com (D. Li), guobing@scu.edu.cn (B. Guo).

http://dx.doi.org/10.1016/j.suscom.2016.08.002
2210-5379/

perature under dynamically varying task and system conditions [22–29].

While software power consumption measuring and estimation is the basis of software power optimization, current software power consumption measuring and estimating method has different levels and granularity, including hardware-based level [5–7]; instruction level [8–13]; source code level [16–21]; software component level [25,26] process level [27–29] and architecture level [30–32]. Hardware-based power measuring method always leverages multimeter to sample the current and voltage of the software and then calculates energy consumption of the software; Instruction-level power model first measures power consumption of each assembly instruction, and then calculates whole power consumption by summating all instructions of the program; while source code level power model first recodes execute paths and power consumption of the software, then calculates power consumption of each line of the code by linear regression; component level power model which estimates energy consumption of software component by combining construction-time estimation and runtime estimation; architecture level modeling focuses on high level characteristics of software, finds the relation between the characteristics and its power consumption, which can quickly analyze and forecast energy consumption of the software. However, research on building power model at architecture level is few. Software as a kind of artificial complex system, especially for object-oriented software, its architecture can be naturally represented in the form of complex networks and many studies investigate different characteristics of the software from this perspective [34–39]. Inspired by this, we try to explore software power model at architecture level based on complex networks in this paper.

First we model software as complex networks, analyze different network characteristics of software and their influence on software power consumption. These network characteristics include number of nodes, number of directed edges, average path length, clustering coefficient and average degree of the software. Then we assume that the relation between these network characteristics of software and its power consumption is nonlinear, and we use BP neural network to fit the nonlinear relation. Finally we validate our model by experiments and the results show our model can achieve 11.2% error compared to measured value, which indicates our assumption is reasonable and our model is effective. Thus our contributions in this paper are:

(1) A software power model at architecture level is proposed in this paper, which can estimate power consumption of the software with small error and meet requirements of high level software power modeling.
(2) We test our power model on real computer platforms and the results validate the rationality of our assumption and our method of measuring network characteristics of software is effective.
(3) Our experiment proves that software power consumption could be analyzed from high level, which has important meaning for low-power software design at architecture level.

2. Related work

Many software power models have been put forward from different levels, including hardware-based level [5–7], instruction level [8–13], source code level [16–21], software component level [25,26], process level [27–29] and architecture level [30–32]. Now we have a brief introduction to each model.

2.1. Hardware-based level

Hardware-based power measurements [5–7] mainly are divided into three types, which are power measurement with meters [5], measurement with special designed devices [6] and measurement by integrating sensors into hardware [7]. Direct power measurement with meters is a straight forward method to understand the power dissipation of devices and the full system, the differences of various measurement methods are which type of meters is used to do the measurement and at which place it is done. Though direct measurement with meters is simple, it does not supply methods to control the process of the measurement process, thus some special designed power measurement devices are presented to measure the power in these circumstances. The third type of approach is mainly used in high-performance servers, which are integrated with power sensors to monitor the power consumed, and then this information is supplied to the administrator for power management. Although we can get power information accurately by hardware, this method requires professional knowledge and it does not easy to operate for ordinary users.

2.2. Instruction level

Tiwari [8–10] first put forward the concept of software power consumption and proposed instruction-level power model, which was shown in Formula (1), total power consumption Ep of the program p is composed of three parts:

$$E_p = \sum_i B_i \times N_i + \sum_{i,j} (O_{i,j} \times N_{i,j}) + \sum_k E_k \quad (1)$$

B_i is the base power cost of instruction i, $O_{i,\ j}$ is the power consumption caused by circuit state switch between instruction i and instruction j. E_k is the power consumption caused by other effects between instructions (such as pipeline stalls, Cache miss, etc...), all of them are determined by corresponding hardware circuit. N_i is execution times of the instruction i, $N_{i,j}$ is the number of occurrences in the program, all these two parameters are determined by program execution path, path information can be obtained by dynamic analysis of the program. The model helps in formulating instruction level power models which provides the fundamental information needed to evaluate power cost of the entire programs, and has been applied to two commercial microprocessors: Intel 486DX2 and Fujitsu SPARClite 934. Based on the instruction power model, Nikolaidis [6] and Leite [7] proposed fine-grained approach for power consumption analysis and prediction, also put forward new methods for low power applications.

2.3. Source code level

Brandolese [16] presented a fully automatic method which combines instruction-level simulation and static-time source characterization for estimating the execution time and power consumption of a C program, also Julien [18] proposed functional approach for estimating the power of an algorithm directly from the C code without compilation. Šimunić [17] employed a profiler, which exploits a cycle-accurate energy consumption simulator to relate the embedded system energy consumption and performance to the source code. Li [19] put forward a power model at source code level, which is implemented by combining hardware-based power measurements with program analysis [22] and statistical modeling. First they recorded code execution paths and their power consumption information by hardware profiling tool as software running, and then used linear regression to calculate the power consumption of each line by the statistical path information and energy consumption. What's more, they put forward software power model at method level and byte-code level [23] based on source code power