A Programming Environment with Runtime Energy Characterization for Energy-Aware Applications

Changjiu Xian
Department of Computer Science
Purdue University
West Lafayette, Indiana
cjx@cs.purdue.edu

Yung-Hsiang Lu
School of Electrical and Computer Engineering
Purdue University
West Lafayette, Indiana
yunglu@purdue.edu

Zhiyuan Li
Department of Computer Science
Purdue University
West Lafayette, Indiana
li@cs.purdue.edu

ABSTRACT

System-level power management has been studied extensively. For further energy reduction, the collaboration from user applications becomes critical. This paper presents a programming environment to ease the construction of energy-aware applications. We observe that energy-aware programs may identify different ways (called *options*) to achieve the desired functionalities and choose the most energy-efficient option at runtime. Our framework provides a programming interface to obtain the estimated energy consumption for choosing a particular option. The energy is estimated based on runtime energy characterization that records a set of runtime conditions correlated with the energy consumption of the options. We provide the procedure and general guidelines for using the environment to construct energy-aware programs. The prototype demonstrates that (a) energy-aware applications can be programmed easily with our interface, (b) accurate estimates are achieved by integrating multiple runtime conditions, and (c) the framework can make multiple devices collaborate for significant energy savings (15% to 41%) with negligible time and energy overhead (<0.35%).

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: design studies

General Terms

Design, Performance

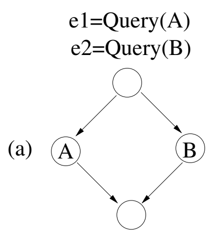
Keywords

energy-aware application, programming environment, energy characterization

1. INTRODUCTION

Reducing energy consumption is important for today's computer systems. Many energy conservation techniques have been proposed from hardware level to operating system (OS) level [2]. For over a decade, power management studies have been focusing on accurately predicting the idleness in workloads such that hardware components can be shut down or slowed down to save energy. These studies have been performed extensively and additional improvements become difficult. For further significant energy reduction, the collaboration from user applications becomes critical. Since the idleness in workloads is ultimately determined by applications, they can adjust their workloads to create more opportunities for power management. Such applications are called *energy-aware applications*. For example, a file-download program in a battery-operated computer can download compressed files and then decompress them. This consumes less energy than downloading the uncompressed files directly if the compression ratio is high and the wireless channel is congested. For another example, web browsers often cache data in a local disk. When the data are requested later, they can be retrieved from the local disk if it is spinning. If the disk is sleeping, retrieving the data from the server may be faster and consume less energy than spinning up the disk. These options may be specific to individual applications; according to the end-to-end argument in system design [7], such functionalities should not be placed into operating systems.

Existing studies [1, 5, 6, 10] are specific to individual programs and difficult to generalize. In this paper, we present a programming environment in which energy-aware programs can be constructed more easily. An energy-aware program may identify different ways (called *options*) to achieve desired functionalities and choose the most energy efficient option. The energy consumption of different options may vary depending on runtime conditions (e.g., the power states of the hardware). Our framework provides an application programming interface (API) for obtaining energy estimates of the options at runtime to decide which option to choose, as illustrated in Figure 1 (a). At the top, the program has two choices: A and B. For the first example described earlier, option A downloads the uncompressed files and option B downloads the compressed files



(b)
```
e1=Query(A)
e2=Query(B)
if (e1 < e2) {
    execute option A
} else {
    execute option B
}
```

Figure 1: (a) Energy consumption of different execution paths. (b) Pseudo-code of choosing options based on estimated energy consumption.