Power saving in high-level programming languages on embedded devices

Péter Henrik Haldorsen Gombos

2015

PROJECT / MASTER THESIS

Department of Computer and Information Science

Norwegian University of Science and Technology

Supervisor 1: Associate Professor Gunnar Tufte

Preface

Trondheim, 2012-12-16

(Your signature)

Ola Nordmann

Acknowledgment

I would like to thank the following persons for their great help during ...

If the project has been carried out in cooperation with an external partner (e.g., a company), you should acknowledge the contribution and give thanks to the involved persons.

You should also acknowledge the contributions made by your supervisor(s).

O.N.

(Your initials)

Abstract

Contents

Chapter 1

Introduction

An embedded system is a computer system with a dedicated usage, that often are constrained on resources, for example memory, power or . When using the term embedded systems, many people are thinking of microcontrollers, which are small CPUs with easy access to hardware peripherals, for instance LEDs. Embedded systems include things like mobile phones, wearable computers, smart sensors and internet of things devices.

As a common constraint for embedded systems is battery life, energy efficiency is needed in every part of the system to conserve power. Some systems need to charge every day, as manufacturers have found a balance between performance and power usage. But in systems that can't be charged at regular intervals, there are completely different goals to the development. For instance, sensor systems in areas that are hard to reach, as on the bottom of the sea, need to be able to function for a long time on a single battery. As lower energy use also means lower heat dissipation, better power usage will lead to a cooler system, which is very important in handheld devices like mobile phones.

In the early days of computing, the way the computers were programmed was with binary code, which is tedious to write, and requires the programmer to remember various arbitrary codes. Quite early, assembly languages were developed, which provides a one to one mapping of easier to remember function names to the machine code. While far better for the programmer, assembly programming is still very tedious to write, and when creating a big complex language, the data models the assembler provides are too basic. Virtually all programs written today are written in a high-level programming language, which means they are not a direct mapping of machine functions, but code written in it need to be transformed into something the machine can run. This transformation can happen in two different ways, compilation or interpretation.

A compiler is a program that takes source code in a given language as input, and outputs code in some other form, either assembly or another high-level language (this is also known as transpiling). The compiled program when run takes the input and produces an output. An interpreter, on the other hand, is a program running the code, and takes the input at the same time and produces the output. See figure.

While compilation and interpretation give a lot of advantages, they do add some overhead while running programs. A perfect compilation will translate the source code into an equivalent with no more instructions than needed, which is impossible except in trivial cases, as high-level languages adds many abstractions that are not easily translated to the basic data models of assembly languages. In the case of interpretation, the interpreter running on the computer by definition adds extra instructions. However, the computer can do things to the source code that are not natural for a programmer, like statistical analysis, which can lead to other optimizations like dead code elimination, loop unrolling etc.

1.1 Earlier work

1.2 Problem

Chapter 2

Experiment

Bibliography

Lundteigen, M. A. and Rausand, M. (2008). Spurious activation of safety instrumented systems in the oil and gas industry: Basic concepts and formulas. *Reliability Engineering and System Safety*, 93:1208–1217.

Rausand, M. (2014). *Reiability of Safety-Critical Systems: Theory and Applications*. Wiley, Hoboken, NJ.

Rausand, M. and Høyland, A. (2004). *System Reliability Theory: Models, Statistical Methods, and Applications*. Wiley, Hoboken, NJ, 2nd edition.