HandBench: A Benchmarking Suite for Processors Embedded in Handheld Devices

Pepijn de Langen Ben Juurlink Stamatis Vassiliadis
Computer Engineering Laboratory
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4 2628 CD Delft
The Netherlands
phone: +31-15-278-3644
email:{pepijn|benj|stamatis}@ce.et.tudelft.nl

Abstract— **Most microprocessors are employed in embedded systems. Since many of these systems are powered by batteries, energy consumption has become an important design aspect. In order to quantify the performance and energy consumption of different architectures, a collection of benchmarks is required that closely represents the applications found on these devices. In this paper, we present a benchmarking suite called *HandBench* that consists of benchmarks which can be considered typical workloads of contemporary and emerging handheld devices. Existing benchmark suites such as *MediaBench* [4] and *MiBench* [3] are inappropriate for our goal for the following reasons. First, these suites contain a number of applications that are, in general, not found on battery-powered devices (e.g., applications from the automotive domain). Also, we included workloads that where not yet present in either of these collections. Furthermore, for a number of applications that are contained in these suites, we have found that the input datasets are not representative of those used in real systems (e.g., using an incorrect sampling-rate).**

keywords: benchmarking, handheld, embedded processors

I. Introduction

The majority of manufactured processors are used in embedded systems. As technology increases, these processors are becoming increasingly important. The number of cellular phones, digital camera's, and similar equipment sold each year is still rising, as is the demand that is put on these devices. Besides the need for computational power, there is another important design aspect for these devices. Being powered by a battery, the amount of available energy is limited. This battery lifetime is also very important to users, since the device becomes useless if the battery runs out. The choice between energy and performance is often not an easy one. In some cases, however, it is clear that increased performance is only useful if it can be achieved with an equal amount of energy usage or less. An example of this is that any device being able to decode a DVD, should also be able to perform this for at least the duration of a normal movie.

In order to measure the performance and energy consumption of different embedded processors, a number of representative applications is required, often called benchmarks or workloads.

II. Related Work

Currently, there exists a number of different benchmarking suites, all targeted at a certain domain. The benchmarking suites by SPEC [1] (Standard Performance and Evaluation Corporation) are probably most well-known. Several suites have been published by SPEC, like SPECint and SPECfp, and more recently also more specific ones, for example SPECmail2001 for benchmarking mail servers. None of these, however, was useful for research in handheld embedded systems. Lee et.al. introduced the MediaBench [4] suite, usefull for systems that run applications from the multimedia domain. Many of the handheld devices we consider run applications from this domain. However, besides the lack of typical handheld applications from other domains, also the formats and sizes of the inputs and outputs of the workloads were not always representative. In 1997, the Embedded Microprocessor Benchmark Consortium [6] (EEMBC) was formed to develop benchmarks for the hardware and software used in embedded systems. The goal of the EEMBC, however, is to deliver benchmarks and certified scores to processor manufacturers. In order to get access to the benchmarks, a high fee must be payed to join the consortium. Guthaus et.al. proposed MiBench [3], a free benchmarking suite for embedded systems. This suite makes an effort to provide

free benchmarks for the whole spectrum of embedded systems. Also, there exists number of suites, like Java SciMark 2.0 [5], that provide benchmarks to measure the performance of scientific and numerical computing on JVM's (Java Virtual Machines).

For our research in low-power handheld devices, there was a need for a collection of representative applications. The benchmarking suite we propose in this work is largely based on MediaBench and MiBench. We share several applications with the two, but there are some differences in how we use them. Besides the fact that we ofcourse chose to use more up-to-date versions of some programs, we also made an effort to use representative inputs. Also, we did not consider the encoding and decoding to always be equally important. With MP3 (MPEG1 layer 3 audio), for example, we consider decoding far more important than encoding, since it is used significantly more often.

III. HandBench

Typically, benchmarks consists of two components: an application, implementing one or more algorithms, and an input set. In the following subsections, we will discuss how both components were chosen for our benchmarks.

A. Applications

The applications included in this benchmarking suite are considered typical workloads of contemporary and emerging handheld devices. In the selection process, we have made the following limitations. The first limitation we put on the programs, is the way it is licenced. Our goal is to present a suite of benchmarks which are redistributable, useable, and modifiable by both academics and industry without any licencing fee. In order to allow for optimizations (and portability), these the source code of these applications should also be available under these conditions. Therefore, we have chosen to only include benchmarks with a licence that is compatible with these needs. Examples of these licences are the GNU Public Licence (GPL), the BSD licence, and the MIT licence. Secondly, the benchmarks should be able to run to a large number of architectures. This implies that the the program under consideration is either available for a number of architectures, or is coded in such way that porting it to a new architecture would require only few code changes. Therefore, use of non-standard libraries should be avoided. Third, the benchmarks should be relatively easy to compile. This is somewhat similar to the second limitation, only a bit more general. Often, researchers use a subset of the benchmarks in a suite, because they were not able to compile the others. To avoid this, we have not included programs that showed to be difficult to compile.

As mentioned before, the benchmark suite proposed in this work shares a number of applications with suites like the multimedia suite MediaBench and the embedded benchmarking suite MiBench. However, these existing suites are inappropriate for our goal, which is energy reduction of handheld devices. Although our target is embedded systems and many handheld devices implement applications from the multimedia domain, we require only a subset of the programs of both domains. Both earlier mentioned suites contain a number of benchmarks which are not found on battery-powered devices. We also included workloads that were not yet present in either of these suites. Furthermore, we have found that existing suites do not always supply a representative input. Besides this, our suite defines the input-sets in a slightly different manner.

The HandBench benchmarking suite consists of programs from the following software packages:

`GSM 1.0.10` (*Jutta Degener and Carsten Bormann*) GSM 06.10 implements the ETSI specifications of the widely used Global System for Mobile telecommunication (GSM) protocol, using a RPE-LTP (Regular-Pulse Excitation Long-Term Predictor).

`GNU Ghostscript 7.07` (*artofcode LLC*) An interpreter for both the PostScript™and the Portable Document Format™(PDF) languages.

`JPEG 6b` (*Thomas G. Lane / The Independent JPEG Group*) Libjpeg is a widely used library to manipulate JPEG images.

`madplay 0.15.2b` (*Robert Leslie*) MAD (MPEG Audio Decoder) is a high-quality MPEG audio decoder that supports MPEG-1 and the MPEG-2 extension with layer I, II, and III (i.e., MP3).

`Sablotron 1.0.1` (*Ginger Alliance et.al.*) Sablotron is a fast, compact and portable XML toolkit. This version implements XSLT 1.0, DOM Level2 and XPath 1.0.

`ffmpeg 0.4.9-pre1` (*Fabrice Bellard et.al.*) FFmpeg is a fast video and audio converter that supports a plethora of codecs.

Table I lists for each benchmark the originating software package along with a short description of the main algorithm and the corresponding handheld device or application. All listed programs compiled with our ARM GCC 2.95.2 cross-compiler and executed fine on the ARM port of SimpleScalar.

Package	Benchmark	Algorithm	Device/Application
GSM 06.10	toast untoast	GSM encoding GSM decoding	Cellular phones
GNU Ghostscript 7.07	gs	PostScript™rendering	PDA's
JPEG 6b	djpeg cjpeg	JPEG decompression JPEG compression	Cellular phones / PDA's
madplay 0.15.2b	madplay	MP3 decoding	Portable music players
Sablotron 1.0.1	sabcmd 'nothing' sabcmd 'complex' sabcmd 'extract' sabcmd 'replace'	XML transforms	Office applications
ffmpeg 0.4.9-pre1	ffmpeg 'decode DVD' ffmpeg 'encode DVD' ffmpeg 'encode DV'	MPEG-2 decoding MPEG-2 encoding DV encoding	Movie players Digital video camera's Digital video camera's

TABLE I
BENCHMARKS INCLUDED IN HANDBENCH.

The benchmarks in our suite can be divided up in two groups: the benchmarks that use a streaming input and the ones that do not. Tables II and III list the benchmarks and inputs of both groups.

B. Input Sets

Most, if not all, benchmark suites are collections of programs, accompanied by a *standard* input set. Since programs can behave differently on different inputs, a fixed input assures the researcher that the produced results are always comparable. In our opinion, the results of a benchmark should definitely not be biased by using a different input set. However, this does not imply that the input should be exactly the same. In fact, different inputs of the same kind should result in about the same numbers. If two different inputs do not result in comparable numbers, these inputs are clearly not of the same kind. In this case, these combinations of program and inputs define different benchmarks. An example of this can be found with decoding an MPEG stream. Here, the use of different prediction schemes (i.e., the number of predicted frames following each intra-coded frame) will largely determine the behavior of the decoding kernel. Another example is found when encoding to a variable bit-rate MPEG stream. In this case, the contents of each frame determines the required bit-rate for corresponding encoded frame. From this, we conclude that any input can be used, as long as it is comparable (i.e., leads to the same results) as a reference input. This does not imply that just any input can be used. In fact, it means one should use different inputs, and that these inputs should produce nearly the same results. This comparison is also the recommended way to ascertain that the the results are independent of the input.

The same reasoning holds for the length of an input set. An input set should be sufficiently large, to make sure that all time-independent measurements (i.e., cache miss-rates, average power consumption) have time to reach a stabilized value. Often, researchers 'fast-forward' a number of instructions from the beginning of a benchmark. One of the reasons for this, is to to 'warm up' components like branch predictors and caches. Another reason might be that the at the beginning of an application, significant time is spend on code that is not considered part of the main algorithm. However, this code might still be important since it might be used, for example, to load the input data or setup decoding tables. If the input is large enough, the influence of this start-up code is either insignificant or, if it is still significant, it should not be neglected. Therefore, blindly skipping a number of instructions is, in our opinion, not advisable. Only under certain circumstances, where using a larger input is not feasible, should instruction skipping be employed.

If a benchmark is executed twice on the same sufficiently large input, it is of course expected to produce relatively the same results. Therefore, the size of the input is not important, as long as it sufficiently large. This too can be used to check for errors: If the concatenation of (the raw data of) twice the same input results in different rates or averages than the single in-

put, the chosen input is either too small or something else is wrong.

IV. Characterization

In this section, we will characterize the benchmarks of HandBench.

A. Methodology

We have used the ARM port of the SimpleScalar [2] toolkit in order to characterize the different benchmarks. We have used a configuration that resembles a SA-1 StrongArm pipeline, almost similar to the one distributed by the authors of the simulator. All benchmarks have been cross-compiled using GCC 2.95.2 on an Intel Pentium 4 running Debian GNU/Linux 3.0. The main advantage of this setup is that, in contrast to the SimpleScalar PISA target, the ARM target is an existing processor. Because of this, there is a large number of libraries available for this target. Furthermore, simulating an existing processor allows for verification of the simulator using a real system.

We have simulated the applications using increasing sizes of input data. By comparing the results of different input sizes, in case of the streaming benchmarks, we can verify that the used inputs are sufficiently large. This comparison also allows us to estimate the number of instructions issued in the start-up part of the programs. For the non-streaming benchmarks, this comparison is interesting, but cannot be used to determine the correctness of the input.

During our research, we came across a number of abnormalities, which will be discussed below. First, it is known that SimpleScalar only simulates the user-mode instructions. Operating system calls are handled by a 'proxy', which simply translates the ARM calls to operating system calls of the hosting machine. Here, a problem arises when a program uses system calls to load its input data, something that most programs do. When returning from a repeatedly executed system call, this causes the simulator to have the illusion that the newly loaded data already resides in the cache. In a system that employs physical addresses in its cache, this would indeed be possible. However, even in this case the operating system must have experienced a cache-miss. By default, SimpleScalar counts these accesses as hits, even though the data has just been fetched from disk. A simple program that only reads data was used to verify this. For this program, the cache-miss-rate could arbitrarily be decreased by simply increasing the size of the input. We have resolved this by flushing the caches on system calls. SimpleScalar was altered to support this, since it was only partly implemented. Secondly, a number of system calls were not implemented yet. In most cases, this was not a problem. Sometimes, however, it limited the possible usage of the benchmarks. For example, with `ffmpeg` it was not possible to seek to a certain time in the input video stream. Therefore, we could not easily select the part of the video streams that we wanted to use for our benchmarks. The third problem we noted was that in `sim-outorder`, for the number of memory references (i.e., `sim_num_refs` and `sim_total_refs`) the number of decoded micro-operations is counted instead of the number of load/store instructions. Since load-multiple and store-multiple instructions are decoded to separate load/store micro-operations, these numbers cannot be easily compared with, for example, the total number of committed instructions (i.e., `sim_num_ins`). Fourth, for some benchmarks, we found that the simulator would exit with an error when ran certain benchmarks on our chosen input data. Until now, we were not able to determine the cause of this problem. Furthermore, the results we got from `sim-outorder`, did not make sense in many cases. This has let us to distrust most of the generated result. Because of this, the amount of results we present here are limited.

Benchmark	**inst./sec**	**Startup Inst.**
toast	10 M	125 k
untoast	56 M	11 k
madplay	141 M	202 k
ffmpeg 'decode DVD'	350 M	46 M
ffmpeg 'encode DVD'	?	?
ffmpeg 'encode DV'	?	?

TABLE IV
Instructions per second of data for the streaming benchmarks.

B. Results

Figure 1 depicts the sizes of the data and text segments of each benchmark. Most interesting here is the huge data size allocated for the `ffmpeg` benchmark. For the other benchmarks, the data segments are not significantly large. Both `gs` and `ffmpeg` also have a significantly larger text size than the other benchmarks.

Table IV list the number of instructions required for encoding or decoding 1 second of data for each stream-

Benchmark	Input	Minimum Size
toast	king.sw	3 seconds
untoast	king.gsm	3 second
madplay	jonobacon-freesoftwaresong.mp3	1 second
ffmpeg 'decode DVD'	'Taxi' (DR90061)	10 seconds
ffmpeg 'encode DVD'	taxi.yuv & taxi.wav	?
ffmpeg 'encode DV'	taxi.yuv and taxi.wav	?

TABLE II
INPUT TO THE STREAMING/REAL-TIME BENCHMARKS INCLUDED IN HANDBENCH.

Benchmark	Input	Input Type
gs	debian-faq.en.page1.ps	PostScript™Level 2, 1 page
djpeg	DSC_5025.jpg	3008x2000 JPEG at quality 96
cjpeg	DSC_5025.ppm	3008x2000 TrueColor Portable anymap
sabcmd 'nothing'	input1.xml	XML 1.0
sabcmd 'complex'	input1.xml	XML 1.0
sabcmd 'extract'	input1.xml	XML 1.0
sabcmd 'replace'	input1.xml	XML 1.0

TABLE III
INPUTS TO THE NON-STREAMING BENCHMARKS INCLUDED IN HANDBENCH.

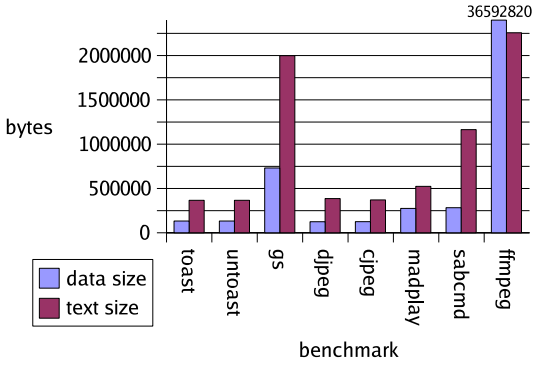

Fig. 1. Data and text size of the benchmarks.

ing benchmark. In case of the `toast`, the `untoast`, and the `madplay` benchmarks, it is clear that the amount of start-up code is insignificant, even when only 1 second of input data is used. For the DVD decoding benchmark, at least 10 seconds of data would be best, so that the relative number of instructions spend on start-up code would be in the order of 1%.

V. Conclusions & Future Work

We have presented HandBench, a benchmarking suite for handheld devices. For a number of benchmarks, we have shown that the small inputs are sufficiently large. For some others, this was not yet possible due to problems with the simulator.

We found that the SimpleScalar toolset did not perform as well as expected. Several errors were found, some of which were not easily fixed. Improvements to this toolset, such as simulation of system calls, are definitely needed.

As the performance of handheld devices improves rapidly, so will the demand for more computational intensive programs. Therefore, this benchmarking suite is not fixed in time. As newer standards come to market, we will have to update the corresponding benchmarks. The current version as well as updates to the suite will be available from the authors website.

We could not produce all of the results we would have liked yet. Therefore, part of the characterization of the benchmarks will be done in future work.

References

[1] Standard Performance Evaluation Corporation. `http://www.spec.org/`.

[2] T. Austin et al. Simplescalar 3.0. `http://www.simplescalar.com/`.

[3] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. In *WWC*, 2001.

[4] Ch. Lee, M. Potkonjak, and W.H. Mangione-Smith. Media-

Bench: A Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems. In *MICRO*, 1997.

[5] R. Pozo and B. Miller. Java scimark 2.0. `http://math.nist.gov/scimark2/`.

[6] the Embedded Microprocessor Benchmark Consortium. `http://www.eembc.org/`.