# Optimizing Embedded Software - A Look at the NEON SIMD unit in the ARM Cortex Family of Processors

Johan Dams
Vaasa University of Applied Sciences
Vaasa - Finland
Email: jd@puv.fi

***Abstract*—The cellphone is quickly becoming the ubiquitous device providing both phone conversations as well as Internet access. There are approximately 4 billion mobile phones in use, compared to about 1 billion PCs and Laptops. In the foreseeable future, the mobile phone will become the device of choice to access the internet. Some problems that appear when using cellphones for Internet access are the performance requirements, and the expected battery life. With customers demanding mobile phones capable of playing high quality video and audio, as well as secure Internet communication from the mobile platform, manufacturers are trying to get as much performance as possible into the small device, but this often goes at the cost of battery life.**

**In order to maximise the available computing power, yet still provide long battery life, one has to perform optimisations on the software level. Power management is one of the methods to maximise battery life by turning of services when the device is idle. The main issue however is how to be able to perform processor intensive tasks, such as video decoding and playback, as efficiently as possible.**

**The ARM Cortex family of processors contains a SIMD (Single Instruction, Multiple Data) unit known as NEON, primarily used for accelerating media applications such as MPEG decoding. It offers however a high degree of flexibility, and can be adapted quite easily to other applications besides multimedia acceleration.**

**In this paper we will present some performance comparisons between NEON optimised code and code using just ARM instructions. We will provide a look at how to optimize existing software, especially looking at the Linux operating system.**

## I. Introduction

Many new and upcoming mobile devices use the Linux operating system at their core. Examples of this include the phones using Google Android, and the new generation of Nokia phones and devices such as the N900. Linux has shown to offer a stable foundation for these kinds of devices, and its open source and free nature can bring costs down considerably for manufacturers. The added benefit of having a large and loyal developer community is also an important factor in this decision.

Linux has its popular roots in server applications. This means that development emphasis has always been on the ability to scale to huge amount of processors, network throughput, etc. While performance is important on server class hardware, it is even more important for mobile devices, where scalability don't really matter. Furthermore, power management and efficiency are much more important on mobile devices than on servers. All this indicates that there is quite some room for improvements in this area.

## II. Optimizing Linux

From the Linux kernel point of view, a lot of optimizations for specific hardware platforms have been made already. Due to the complexity of the software, it is not easy to determine where optimizations would be needed, or if they would be of benefit at all. Furthermore, it would be rather difficult to get proper optimizations officially accepted into the kernel within a reasonable amount of time. Added to this, the overhead of maintaining an own kernel with specific improvements can be a daunting task which requires resources that might not be available - especially for smaller companies with limited resources.

One part that gets often overlooked is the C library. While it is true that many different implementations of the the C library exist such as uClibc, Glibc, Eglibc, etc, these implementations are still written with a generic hardware platform in mind, offering for instance smaller code size, but still carrying the majority of the code from their bigger brother. These implementation contain only few, if any, platform specific optimizations.

Since the C library is considerably less complex than the Linux kernel, this is an ideal place to start optimizing. Individual functions from the C library can be replaced one by one, providing a focus point when analysing which functions are used most often in a certain application. It is also possible to provide an own library of functions, taking over from those in the standard C library.

### A. Optimising the C Library

A good point of entrance for starting this optimisation process is the memcpy() function. It is one of the most called functions throughout the entire system, also when running user applications. If a speed increase can be accomplished here, it will have direct impact on the rest of the system, and it can prove that further improvements to other parts of the C library make sense.

The NEON register file is viewed as 16 128-bit registers or 32 64-bit registers, each a vector of 8/16/32-bit integers or 32-bit floats. It is therefore ideal for operating on large memory blocks, often the case when using memcpy(). This also means, and is inherent to the way SIMD works, that the best performance improvements will be possible on larger memory areas compared to small areas of several bytes.

The memcpy() function has several optimisations for x86 and PowerPC. For ARM, there exists a Neon optimised implementation made by CodeSourcery[1]. This implementation however is C based and relies on compiler optimisations to generate Neon instructions. While this is a relatively easy way of working, compiler generated code is likely slower than hand coded Assembler, and compiler auto-vectorization for NEON code is still at an early stage. This is one of those instances where hand coded Assembler still makes sense in 2009.

Of course, it would be nice to know in advance what kind of improvements could be expected. A substantial amount of work on optimizing the C library functions for the Altivec SIMD unit, found inside certain PowerPC chips, has been done in the Freevec library project[2]. The results obtained show a possible factor 2 to 3 speed increase compared to the native glibc functions. It is reasonable to expect similar improvements for NEON optimised version.

The NEON memcpy() function presented here has been benchmarked to an already optimised version for ARM, written in assembler, and included in glibc-ports, part of the GNU C library[3]. The NEON version works in a similar way to the Glibc implementation, copying 32-bit blocks where possible. The results are shown in Fig. 1 below.

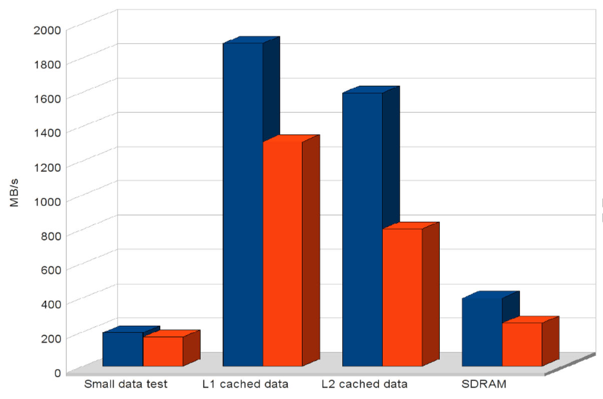

Fig. 1. NEON Vs. ARM memcpy()

The target hardware platform was the EfikaMX developer edition running a Freescale i.MX515 ARM Cortex-A8 System On Chip at 800MHz, produced by Genesi USA[4]. As one can see, the NEON version performs better on all tests, even those using small datasets of several bytes, normally not the target of SIMD units. The operations on L2 cached data show the most improvement with the NEON version twice as fast compared to the ARM version. The results have been averaged over both aligned and non-aligned data. The speed increase noted here suggests that similar improvements can be made in several functions dealing with memory such as memcmp(), and also string operations such as strlen() and strcpy(). Many of these functions do not have an ARM optimised version at all, and performance increase can be expected to be much higher.

### B. *Other Optimisations*

Besides the optimisations possible in the C library itself, many applications can benefit from NEON. Matrix and vector operations especially are prime candidates for this, and are relatively easy to implement. During this research, a 4x4 matrix multiplication was implemented which performed 8 times faster than the standard C version.

A vector multiply-and-accumulate (VMAC) function which took 2.3 seconds in standard C code on the ARM, took less than 100 milliseconds using NEON. This is especially useful in such applications such as MPEG audio decoding, especially during the sub-band decoding process. Some of the most impressive developments in this area have shown the possibility to play 720p definition video on NEON equipped ARM processors at 500Mhz. This was accomplished using a series of NEON optimizations in the open source ffmpeg library, posted here[5].

## III. Conclusion

This preliminary work has shown that there are a lot of potential optimizations possible on modern computer architectures using SIMD units. With the push towards ever more mobile computing, these optimizations will be able to not only speed up intensive applications such as audio and video decoding, they can also increase power efficiency leading to longer battery life.

The NEON optimised version of memcpy() presented here has shown that the underlying system of many mobile platforms is still highly unoptimised. Future work in this area, on optimisations, or even complete replacements, of the underlying C library can result in better performance and more efficient use of resources.

## Acknowledgement

We would like to thank Genesi USA for their generous hardware donations.

## References

[1] CodeSourcery Arm Toolchain and C Library. *http://www.codesourcery.com/sgpp/lite/arm*, Last accessed September 2009.

[2] SIMD Freevec library. *http://freevec.org/*, Last accessed September 2009.

[3] GNU C Library. *http://www.gnu.org/software/libc/*, Last accessed September 2009.

[4] Genesi USA. *http://www.genesi-usa.com*, Last accessed September 2009.

[5] ARM NEON patches for ffmpeg, *http://lists.mplayerhq.hu/pipermail/ffmpeg-devel/2008-December/056933.html*, Last Aaccessed September 2009.