Optimization of Embedded Linux systems without FPU

Sergey Panasyuk, SUNY Institute of Technology, panasys@cs.sunyit.edu
Scott Spetka, SUNY Institute of Technology and ITT Corp., scott@cs.sunyit.edu

Abstract

Linux is a popular operating system for embedded systems. Some embedded systems use processors without floating point accelerators (FPA) or floating point units (FPU), in order to satisfy cost and power consumption requirements. This paper describes and compares optimization options for software implementation of floating point operations on systems without FPA or FPU. Our results show that software approximations perform very well, even when compared to systems with hardware FPA or FPU. Software approximations to IEEE standard floating point can be used in many important applications, such as cell phones, games, etc.

Introduction

Linux is often selected for embedded systems, because it is free, open source and easily modified. Linux is available under the GNU General Public License. As an embedded operating system Linux is fully ported to a wide variety of different architectures: Alpha, ARM, MIPS, PowerPC, SH, SPARC and others.

Low cost, optimum performance and long battery life are common requirements for many embedded systems, including cell phones, media players and industrial hand-held testing devices. Low cost combined with low power consumption puts restrictions on the type of CPU that an embedded system can have, normally it is low frequency (50 – 500 MHz) CPU without hardware FPA or FPU, which we refer to generically as FPU.

The ARM type of processors is very popular for this type of embedded system. Integer operations are performed well by such processors, but when it comes to floating point operations, those systems perform poorly. Applications that require fast floating point computations range from games to commercial solutions like protocol and media testers.

The absence of FPU requires careful design and proper system and code optimizations in order to deliver high system performance. We compare software optimizations for floating point operations using a PDA-like device and a PC. The test applications generates a high demand for floating point operations, both single and double precision.

We compare software floating point emulation with kernel trapped hardware FPU instructions in the first section. In the second section, we compare approximation techniques with software emulation of IEEE standard floating point. We then describe our test environment and present our results.

Floating Point Emulation vs. Hardware FPU

To optimize an embedded Linux system, avoid using Linux kernel supported floating point emulation. Because FPU instructions from binary image will be fetched to CPU for execution, it will produce unknown instruction exception. A Linux kernel running on systems without a hardware FPU must include support to handle such exceptions from CPU - unknown instruction trap. ARM Linux kernel includes floating point emulation (Software FPE) routines that are capable of computing FPU instructions by substituting them with a set of integer instructions and producing the same result (or close approximation) as FPU instruction [1,2]. The Linux kernel (2.6.17.9) has two different configuration options for handling the absence of FPU: NetWinder Floating Point Emulator (NWFPE), Fast Floating Point Emulator (FastFPE). Table 1 shows the result of disassembly for a program that adds two floating point numbers, for the FPU and FPE approaches.

Table 1: Disassembly of Add two Floats

FPU	FPE
ldfs f0, [r3, #0]	ldr r0, [sl, r4, asl #2]
ldfs f1, [r1, #0]	ldr r1, [r8, r4, asl #2]
adfs f0, f0, f1	bl __addsf3
stfs f0, [r2, #0]	str r0, [r5, r4, asl #2]

Software FPE vs. Approximation

While software FPE provides big performance boost on systems without FPU it may still be considered as slow compared to approximation alternatives. In order to satisfy IEEE standard and perform accurate computation FPU instructions must be emulated to the maximum precision. Complex trigonometric functions will use a lot of software FPE function calls to produce accurate result. It is possible to increase performance of embedded system without FPU even further by implementing custom function calls. In this project, we evaluate Intel's Integrated Performance Primitives (IPP) [3]: a set of libraries for XScale based architecture to perform floating point operations by using integer arithmetic. IPP converts float type to integer with scale factor.

Test Systems

Currently there are a lot of different architectures available for PDA's, Smartphones, handhelds and other embedded systems. ARM architecture is well supported by Linux community and popular in embedded devices. For the project we selected a PXA270 based device that falls into the ARM version 5 with thumb instructions (ARM5TE) architecture. PXA270 designed by Intel and is part of XScale family or processors (PXA2xx). PXA270 is being used in a large set of commercially available handheld and embedded systems. This processor does not have a hardware FPU and there is no option of adding an FPA to it. Because of its popularity and lack of FPU it is a good candidate for floating point optimization; to get most from the embedded system. A 300MHz Intel Celeron System was used to compare our software approaches with a comparable system with a FPU. Table 2 describes our test hardware.

Table 2: Test Systems

	PXA270 312MHz	Celeron 300Mhz
System Bus	208 MHz	33 MHz
Memory	64 MB	512 MB
Storage	64 MB (Flash)	60GB (Hard Disk)
Network	Build-in NIC	Build-in NIC
OS	Linux 2.6.17.9	Win XP

Results

We tested a set of floating point operations, ranging from simple assignment to a function involving multiplication and log. The precision of each of the experiments performed on the PXA270 was compared to IPP on the x86, since IPP/x86 produced 24/53 bits mantissa accuracy for floats and double respectively. As an example of our precision results, table 3 shows that the precision on the PXA270 for IPP on was poor, compared to IPP on the x86.

Table 3: Float Errors on PXA270 with IPP

Function	Min	Mean	Mean
F=X	-2.98E-08	-2.47E-12	2.98E-08
F=X+Y	-1.19E-07	4.83E-12	1.19E-07
F=X*Y	-3.81E-06	6.51E-10	3.81E-06
F=X/Y	-9.54E-07	3.86E-11	9.54E-07
F=LN(X)	-4.77E-07	1.24E-09	4.77E-07

Although the precision was poor for PXA270 with IPP software floating point emulation, figure 2 shows that it performed best, after hardware FPU, for some operations. This is not a surprise because IPP uses approximation and conversion of float to integer with scale factor. Loss of precision because of conversion can be observed by examining F=X, the first row in the table. The cost of assignment, as determined experimentally, was subtracted from the performance measurements that resulted from assigning the results of the floating point expressions to variables for our other tests. This allowed us to accurately measure the cost of each floating point operation.

While IPP provides great performance, its accuracy is very poor and must be used with this in mind. Also, IPP does not fully support sqrt, exp and sin, thus these results are not presented in figure 2. With IPP, care must be taken to avoid overflow. For example, if the scale is 24, then the integer part of a float cannot exceed a range from -127 to +127. A major disadvantage for IPP is that it is not free.

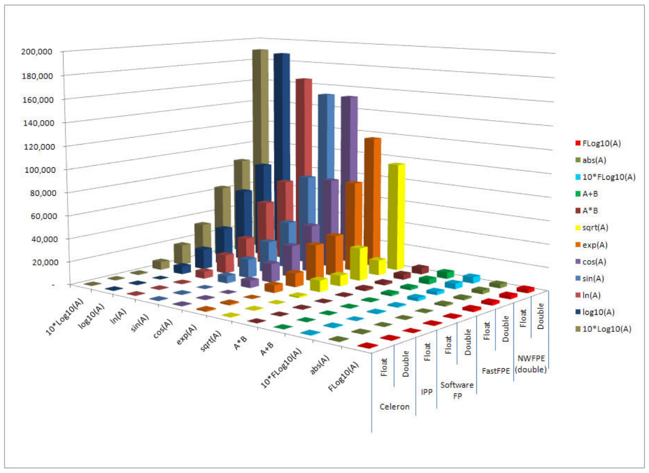

Figure 1: Performance Results.

Conclusion

In conclusion, we recommend the following:

1) Perform computation on floats instead of doubles (If it is not critical to use doubles). All computations were performed in the same or shorter period of time by using floats. On average Software FPE performed twice as fast on floats as on doubles. FastFPE had 1/3 increase, while NWFPE 1/6 increase in performance.

2) If Software FPE is not an option then select FastFPE in the Linux kernel configuration. It is experimental and has a higher difference in results then Software FPE or NWFPE, but performs much better then NWFPE: on average at least twice as fast.

3) Approximation libraries like IPP can be used to speed up computation of functions like natural logarithm. However, it is important to perform benchmark to see what the actual difference in output is and what speed boost can be achieved. In this work, only the natural logarithm function would make sense to replace with a call to IPP, since the difference in results are not bad and performance increased 8, 39 and 107 times for Software FPE, FastFPE and NWFPE respectively. Since it is common for many embedded test systems to convert data into a logarithmic domain, it would be proper place for IPP library.

References

[1] ARM Application Note 23

[2] ARM Application Note 55

[3] Intel Integrated Performance Primitives www(dot)intel(dot)com/cd/software/products/asmo-na/eng/302910.htm