ABSTRACT

Performance is always a delicate matter for algorithms to function properly in real-time or any other critical scenario. Depending on the application complexity, the processor power from a single core may not be enough to accomplish all the desired tasks; in order to surpass this boundary without multi-core solutions, it is only natural to consider the use of more than one machine, which is widely known as clustering. Following previous work, this paper intends to discuss a cluster built at the “Universidade de Sao Paulo – USP”, now using Beagle Boards DSP cores in addition to ARM cores. The Beagle Boards main processor is a Texas Instruments OMAP3530. The additional steps to accomplish were made in the software.

1. INTRODUCTION

When system engineers have to develop computer applications that demand a large amount of computer performance due to complex algorithms or size of the data, a regular approach is to use many computers combined in a cluster. This refers to a largely applied technique [1] which high demanding tasks are somehow split into lighter jobs and distributed among all the nodes from a network. The final result is the increasing the overall system performance. A remarkable example of resource consuming applications is real time processing codes for embedded devices [2].

However, finding a faster processor is not always possible due to different reasons, from availability and price [1] to its very non-existence. In terms of software optimization, besides the effort to implement each of the improvements, the code designer must be aware that a practical limit will eventually be reached [2]. Therefore, system engineers often choose clustering as a method for increasing performance in a relatively inexpensive way.

Also, the combination of many nodes can increase dramatically the power consumption. As an important figure of merit of this approach is the cluster power consumption. As building blocks of the cluster nodes, were used Beagle Boards(BBs), an open hardware platform with its main processor being OMAP3530[3](Fig. 1 shows a block diagram), a dual-core processor, composed by a cortex-A8 ARM core and a TMS320C64x+ fixed-point DSP core. Despite last work, the DSP were also used to do all the calculations.

With previous built cluster already working in the laboratory[4], the work were done in the direction of allowing DSP cluster processing. The available system were composed by two Beagle Boards(BBs) interconnected by an Ethernet link, and each of the them had mouse, keyboard and display.

The current work was involved in studying few techniques to use DSP within ARM core and which is more appropriate;

Choosing a Linux kernel compatible with the techniques ;

Modifying the desired application make possible to run in the DSP core ;

Generating a binary capable of running MPI implementation in the ARM core while using DSP core for calculating;

Running the tests.

In Materials & Methods section we will introduce the environment and tools used. The Results section will show the performance of our proposal and in Conclusion sections we will present final remarks and next steps.

Figure 1 - The OMAP 3530 internal architecture. Its possible to observe both DSP and ARM core. [5]
2. MATERIALS & METHODS

For the experiments, two Beagle Boards[6], a fan-less single board computer measuring only 3” x 3”, were used (Fig. 2 shows a simplified representation of the board). The BB is based on the Texas Instruments OMAP3530[3] and are able to work under clock rates up to 700MHz. One of the boards is in revision B5 and presents less memory, 128MB of LPDDR RAM, then the other in revision C4, with 256MB of LPDDR RAM.

Besides the ARM cortex-A8, the OMAP3530 also displays a DSP TMS320C64x+ core and supports the main existing operational systems [3].

Each BB had it USB-OTG ID pin grounded to make it act as a simple USB-HOST port[7]. As the boards hasn't enough power to supply external peripherals, an external powered USB hub was needed. To the hub, was attached a mouse, a keyboard and an USB Ethernet adapter. Both boards were connected to an Ethernet hub, becoming the cluster as shown in Figure 3[4].

The hardware was finished accessing each of the Beagle Boards and changing some environment variables in the second stage bootloader (U-boot[8]) to get them booting from SD card[9].

2.1 – Technique to use DSP core

Reading about the OMAP3 platform, it was found that there were many available forms to use DSP, however the attention were spent into three well documented ways.

The first studied method were using the Texas Instruments C64x+ DSP Library[10], which was rapidly dropped, since as a library with predefined functions, it wouldn’t allow compile the code chosen to DSP code.

Next step was to look the TI C6Accel[11], and them, the same problem occurred, it didn’t allow to compile standard c source code.

With the left tool available being TI C6Run[12], it fitted well. C6Run is a set of tools which will take in C files and generate either an ARM executable(C6RunApp), or an ARM library(C6RunLib) which will leverage the DSP to execute the C code[12].

Since the communications had to be done in the ARM core running Linux and its Ethernet connection while the DSP would stay with calculations, it was chosen to generate a ARM library with DSP calculation code inside, instead of a ARM executable containing all the application as DSP code that would generate problems since it hadn't direct access to the Ethernet inter-connection.

2.2 – Choosing a Linux Kernel

To be able to execute the code in the ARM library generated by C6RunLib, it was necessary to have two modules running withing the kernel in order to access the DSP core. One of the kernels capable of running both modules was the 2.6.32 and them it was chosen. It was opted to generate a custom Angstrom image[13], with kernel 2.6.32 and containing Dropbear[14] SSH[15] client to be used by MPI[16]
as a button layer of the communication.

The image was unpacked in the previous generated SD card. After it, a cross-compiled version of MPICH was installed in the cards as a MPI implementation.

**2.3 – The application to test the ARM+DSP cluster**

As tested against the previous pure ARM cluster, the application used to test the cluster was a parallel Pi calculating algorithm that uses numerical integration.

With the source code shipped with MPICH, two modifications were made. One was increase the number of iterations from 10,000 to 10,000,000, so its execution time could be measured much more accurately. The other was separate the main loop of the program in a different source file and its header file (Fig. 4 shows the process), so both source files (main.c and critical.c) could be compiled separately.

![Figure 4 - Separating the pi application](image)

**2.4 – Creating the binaries for the test**

To the test, two binaries were created, while both has its communication running in the ARM core, the difference between them is that one relies the calculation on the DSP core(pi dsp) when the other relies the calculation on the ARM core(pi arm).

The pi dsp were created firstly compiling the critical.c with the C6RunLib tool into a library file(critical.lib) and the main.c into an c object file with MPICH GCC wrapper mpicc(as back-end it uses CodeSourcery [18] cross-compiler, which is based in GCC ARM compiler). After it, both object code and library code were linked generating the binary pi dsp(Fig 5 shows detailed how it was done).

The pi arm was created using only the MPICH GCC wrapper, mpicc(Fig 6). Its interesting to note that no modifications in the source code were necessary between both binaries.

![Figure 5 - Compilating pi application with calculations left to DSP core. Note that all commands were done in the development computer (x86 architecture)](image)

![Figure 6 - Compilating pi application with calculations left to ARM core](image)

**2.5 – Test itself**

The first test consist in running the pi dsp executable in each board alone and them running pi dsp in the cluster environment. This test all the calculations are done in the DSP core, while the communication stuff is done in the ARM core.

The second test is the same as the first, but this time, running pi arm, which leverage both calculation and communication to the ARM core.

The execution of the test sets was done by accessing the boards through SSH. Note that the cluster execution started in the board with revision C4.

To increase the tests reliability, a set of 5 executions per chosen situation was run and then, a simple average of the results was taken.
3. RESULTS & CONCLUSION

The results of the tests are expressed in terms of run time regarding the Pi calculating program (the less time the better), and are shown in Figures 7 and 8.

Since the boards presented different execution times separately for the same program, the worst case (WCT – worst case time) was picked as a reference to analyse the cluster results. Because the Pi algorithm divides the jobs equally among the nodes, it is reasonable to assume that, ideally, the cluster (made of 2 boards) should be able to execute the same work of one board in WCT/2.

Looking into the data obtained from Pi program running on the DSP core (TEST 1, see 2.5), the results were above all expectations since the theoretical minimum time (TMT) to calculate Pi using the cluster is 24.7245s (WCT/2), and the running time of the cluster was only 0.12% slower than TMT, which is good, considering all the overhead caused by MPI distributed-memory architecture and its communications.

Now, observing the data acquired with calculations in the ARM core (TEST 2, see 2.5), the results were even better. The execution time was only 0.04% bigger than TMT (8.193s). This better performance can be explained because although the ARM core were being used for both calculations and communications, it didn't have the overhead caused by communications between the ARM core and the DSP core as in the TEST 1.

Comparing TEST 1 and TEST 2 execution times, it can be seen that the ARM core is faster (about 3 times) for the Pi application, and much of this is due the fact that the DSP core is fixed-point while the application is floating-point which cause performance drop due emulation of floating-point.

As was proven that a DSP cluster is possible and has a small performance drop due clustering (at least for 2 nodes), next steps are to test different algorithms to compare its speed between DSP and ARM and add more nodes to the cluster to measure performance drop regarding increasing number of nodes.

REFERENCES


