Research Repository UCD

Title	Effect of Compiler Optimizations on DSP Processor Power and Energy Consumption
Authors(s)	Casas-Sanchez, M., Rizo-Morente, J., Bleakley, Chris J., Gonzalez-Hernandez, Juan
Publication date	2007-11-23
Publication information	Casas-Sanchez, M., J. Rizo-Morente, Chris J. Bleakley, and Juan Gonzalez-Hernandez. "Effect of Compiler Optimizations on DSP Processor Power and Energy Consumption," 2007.
Conference details	XXII Conference on Design of Circuits and Integrated Systems (DCIS), Seville, Spain, 21 - 23 November, 2007
Item record/more information	http://hdl.handle.net/10197/7098

Downloaded 2024-04-16 05:22:14

The UCD community has made this article openly available. Please share how this access benefits you. Your story matters! (@ucd_oa)

© Some rights reserved. For more information

Effect of Compiler Optimizations on DSP Processor Power and Energy Consumption

Abstract— **This paper examines the effect of compiler optimizations on the energy usage and power consumption of DSP processor, specifically to the Texas Instruments TMS320VC5510. The effects of different levels of general and specific optimization on the energy and power consumption are measured for this processor. Given the special characteristics of DSP programs, the benchmark routines were selected from DSPStone, and some typical DSP applications. Finally, Texas Instruments library routines are compared with the compiled versions. The paper provides an analysis of the results together with recommendations for improving performance. The binaries used in this study were generated using the Texas Instrument C/C++ Compiler, which allows control over the whole set of optimizations.**

I. Introduction

The classic separation of hardware and software has led to the misconception that only hardware dissipates power, disregarding the power consumption effects of the code running on the machine. However, that viewpoint is analogous to stating that a car driver's behaviour does not affect the engine's fuel consumption. Different uses of processor resources can dramatically change the power and energy consumption of embedded processors. Given the shift of Digital Signal Processing (DSP) program development from assembler to high level languages, mainly C/C++, the importance of the compiler is increasing, since the mapping process between high level and assembly language is a primer factor in determining the power and energy consumption. Often DSP processors are embedded into battery-dependent systems, the power and energy consumption important design parameters.

In this paper, the compiler's impact on power and energy consumption is explored for the Texas Instruments (TI) C Compiler included in the Code Composer Studio for the TI TMS320VC5510 DSP Processor. It is important to assess the optimization process in power and energy terms since they are usually conflicting objectives, related by the number of cycles. The C5510 is a high performance processor targetted at low power, medium performance DSP applications. Succesful systems based on this DSP include modems (3Com), cell phone handsets (Nokia, Ericsson), portable MP3 players (Sanyo), digital still cameras (Sony) and digital video recorders (JVC).

Two DSP characteristics make the compilation for DSP architectures specialized: First, DSPs instruction sets are highly irregular, including several specific datapaths, application specific units, VLIW extensions, etc [1]. Second, DSP programs have several differences compared to General Purpose Procesors (GPP) software [2], such as no user interaction, previously arranged array data sizes, no string manipulation, extensive dual memory accesses, etc. Given this, test benches were extracted from the computational kernels included in the DSPStone benchmark [3]. These kernels are targeted to test specific Signal Processing routines, such as Fast Fourier Transforms (FFT), or matrix products. Fourteen DSP subroutines and some applications were compiled with various general optimization levels and the evolution of their consumption was measured. Also some larger applications within the scope of this processor were evaluated to confirm the results. The C5510 specific optimizations are evaluated for the same DSPStone benckmarks. Finally a comparison between the Texas Instruments library functions [4] giving C versions is provided. This library includes signal processing kernels such as FFT, filters and convolution, adaptative filtering, and general mathematical functions.

Literature in the field has been sparse and published only in recent years. In [5] there is a first attempt to understand the scope of compiler optimizations, simulating and measuring the power consumption of subsystems such as ALU or the register file. In [6], several loop nest optimizations were simulated for energy consumption, such as loop unrolling or loop fussion, for the *SimplePower* RTL core, and benchmarks extracted from Spec. In [7], the effect in terms of power/energy of the general optimizations and some individual optimizations, for the Alpha processor, is evaluated. The processor is simulated by means of Wattch [8], running different SpecInt95 and SpecFp95 benchmarks. Finally in [9] the effect of the Intel compiler general and specific optimizations, for energy and power consumption, were measured for the Pentium IV running some benchmarks extracted from Spec2000.

The structure of this article is as follows. Relevant architectural details of TMS320VC5510 are described in section II, along with the physical measurement setup, and details about the DSPStone benchmarks and routines used. Section III details the numerical results for compilations with general optimizations, exposes the consumption effect of the individual optimizations, and shows the results of the comparison between library routines and compiled counterparts where available. Finally some suggestions for improvement of performance, and the conclusions are gathered in sections IV and V respectively.

II. Methodology

The target embedded DSP used for the study is the fixed point Texas Instruments TMS320VC5510 [10], with variable core voltage and clock frequency up to 200MHz. It implements several special architectural features relevant to the work at hand. These are as follows:

- VLIW processor, capable of issuing two instructions at the same time, under some restrictions. Variable instruc-

tion length, from 8 to 48 bits. Some instructions have short and long versions.

- Instruction Buffer Queue (IBQ), that fetches four program bytes a cycle, up to 64, passes six bytes at a time to the instruction decoder, and is flushed each time the Program Counter jumps.
- Possibility to save a small basic block of instructions (<64 bytes) into the IBQ for execution, as a first type of hardware loop. Two levels of hardware based loops, apart from the previously mentioned, that retain the fetchof instructions but execute the sequencing and condition check by hardware.
- Two independent 40 bit MAC units, one 40 bit ALU, one 16 bit ALU, two swap units, and one barrel shifter. One program and one data address generation unit, featuring two extra ALUs for indirect addressing modes.
- Twelve independent buses to memory, divided into two program buses (data and address), six data read buses and four data write buses, along with internal buses for constant passing between units.
- Several low power capabilities such as configurable clock frequency and core voltage. Independent, hardware-configurable, idle domains off and on: CPU, DMA controller, Instruction Cache, Peripherals, Clock Generator and External Memory Interface (EMIF). It is worth noting that these domains can only be switched off by the programmer.

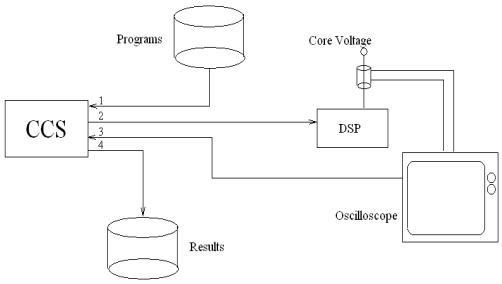

Figure 1: Instruction Energy Consumption Measuring Scheme

The physical measurement methodology was applied to the 5510 DSK Development Software Kit [11], that provides $1.1-1.6\mathrm{V}$ core voltage and $24\mathrm{MHz}$ frequency reference connected to a PC running the TI Code Composer Studio (CCS), with integrated compiler. The tool was used to download and run the test programs, as illustrated in Fig. 1. External software routines were used to trigger the measurements using the digital storage scope. The current drawn was measured with a non intrusive, $0.1\mathrm{mA}$ resolution current probe. The probe bandwith is around $50\mathrm{MHz}$ providing enough resolution for these purposes. The measurements were taken at $1.6\mathrm{V}, 24\mathrm{MHz}$ unless otherwise noted.

The benchmarks selected for the study at hand are those included in the fixed point DSPStone library [3]. This is a publicly available DSP oriented benchmark library. The DSPstone-Kernel Benchmarks are listed in Table I. The routines include dot product, several implementations of FIR and IIR filters, convolution, real and complex single and multiple vector updates, matrix operations, FFT's and the Least Mean Squares algorithm. Several routines from applications in the range of this processor were also evaluated. These are examples of larger programs for which the compiler might have more opportunities for optimization. All applications were modified for 16 bit fixed point. First, an Intel/Dvi ADPCM 4:1 mono coder and decoder implementation [12] (CCITT Recommendation G.721) was evaluated, for compression and decompression of a frame of one thousand 8-bit voice samples. Second, from the Blade MP3 encoder implementation [13], the polyphase, 32 channel filter bank and the multichannel modified DCT, together forming the filter bank, were measured for transformation of a frame of 512 samples into 576 spectral lines. Third, the TU Berlin implementation [14] of the RPE-LPC voice analysis for GSM was measured. This sub-system encodes a frame of $20\mathrm{ms}$ of PCM speech (1280 bits) as a 160 bit frame. A 9-tap filter, 4 level, 32x32 wavelet analysis code, was implemented and measured. Finally, a 8x8 Discrete Cosine Transform, forward and inverse, was evaluated. The first application core is mainly if-based while the others are composed of nested loops. In this way, the compiler's capability to handle different code structures can be investigated. DSPLib is a Texas Instruments supplied library of manually optimised C callable routines [4] for typical DSP and mathematical functions. TI provides a C counterpart for several. These were used for comparison of compiler performance with manually optimised library routines.

Benchmark Number	Description
1	2 by 2 Dot Product
2	Real Update, implements d=c+ab, all real
3	Convolution length 16, without state update
4	16 tap LMS step: filtering and coefficient update
5	Fir 2Dim, 3x3 coefficient block
6	One IIR Biquad section output sample
7	Four IIR Biquad sections
8	10x10 Matrix Product
9	3x3 Matrix times 3x1 vector
10	One FIR output sample calculation, length 16
11	Complex Update, implements d=c+ab, all complex
12	16 Complex Update
13	16 Real Updates
14	16-point FFT

Table I: DSPStone benchmarks

III. Numerical Results

The flags controlling the one pass compiler optimizer can be either general or specific. The former are those that enable a set of compiler optimizations. These flags are used to simplify compilation by enabling a group of optimizations with just one modifier and are described in subsection A. The explanation for most of them can be found in the classical text by Aho, Sethi, Ullman [15]. The specific optimizations are those that enable the user to control particular optimizations related to specific C5510 features, as explained in subsection B. A comparison for performance, between TI library functions and their original compiled versions, when provided, is presented in subsection C.

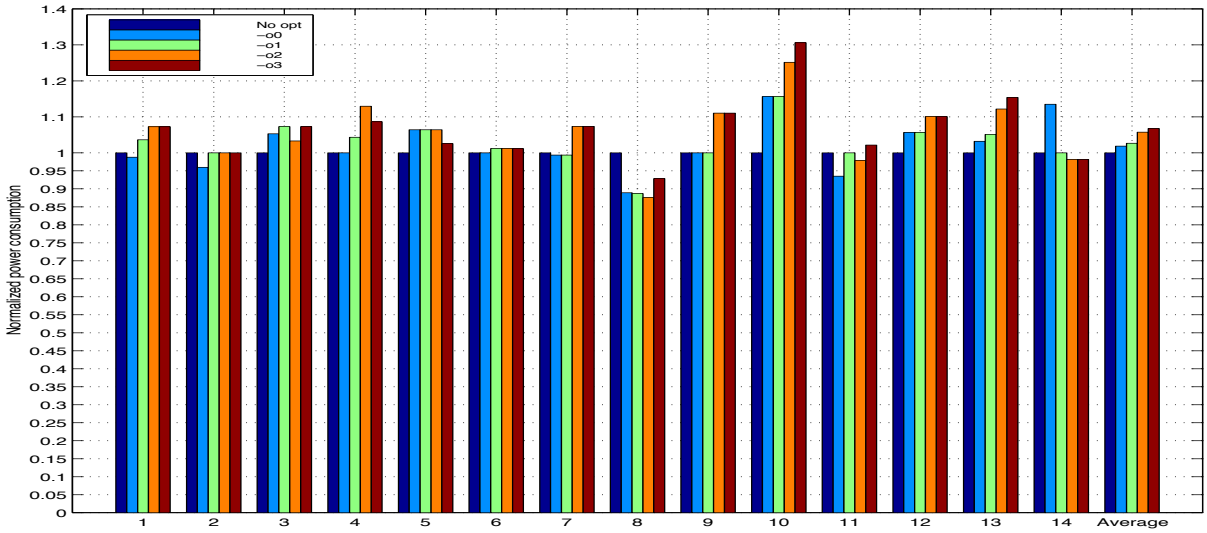

Figure 2: Normalised Power Consumption across Compiler Optimizations for the DSPStone Benchmarks

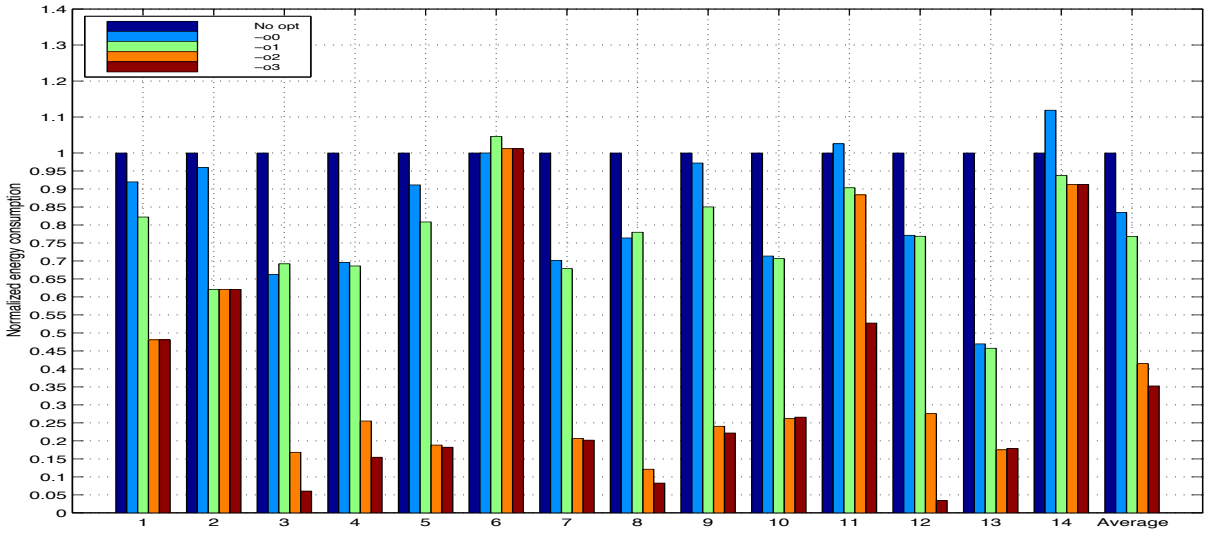

Figure 3: Normalised Energy Consumption across Compiler optimizations for the DSPStone Benchmarks

A. General optimizations

The general optimizations are activated with the four classical flags:

- No -o flag, this disables the optimization pass.
- -o0: This flag enables the following: control-flow-graph simplification, allocation of variables to registers, loop rotation, elimination of unused code, simplification of expressions and statements, expansion of calls to functions declared inline.
- -o1: Enables -o0 optimizations plus local copy/constant propagation, removal of unused assignments and elimination of local common expressions.
- -o2: Includes all -o1 activities and performs loop optimizations, elimination of global common subexpressions, elimination of global unused assignments and loop unrolling. Loop optimization in practise means the use of hardware loops whenever possible.
- -o3: Includes all -o2 optimizations plus removal of all functions that are never called, simplification of functions with return values that are never used, inlining of calls to small functions, reordering of function declarations so that the attributes of called functions are known when the caller is optimized, identification of file-level variable characteristics.

Results for power and energy consumption for the DSPStone benchmarks can be found in Fig. 2 and 3, respectively, along with their averages. Energy variations are closely related to cycle count changes. It is worth noting that benchmarks 1, 2, 6 and 11 are shorter than 35 cycles, giving large variations in consumption. Two groups of optimization results can be identified namely -o0 plus -o1, enhanced mainly by register mapping, and -o2 plus -o3 distinguished by the use of hardware (zero-overhead) loops.

Power generally increases with increasing optimization level since usually higher order optimization usually leads to a reduction in the number of stalls and a higher degree of parallelism (mainly through dual MAC operations) [16]. Increasing optimization means lower energy for the present benchmarks. This is due to a reduction in cycle count.

No optimization, -o0 and -o1 imply the use of pointer registers. However, in the first case, necessary data is fetched from memory in advance of the instruction actually making use of it. Meanwhile, in -o0 and -o1 levels, data is fetched in the actual instruction cycles. The latter solution saves cycles and energy as the fetch operations are execute simultaneously, while increasing the power consumption since more functional units are used simultaneously. If the program uses many variables the net effect is a power consumption increase and an energy consumption drop. Programs with few variables, nevertheless, present a similar energy consumption drop while power remains unchanged, since registers are loaded less frequently and reused more often. Using this classification, benchmarks 3, 5, 10, 12, 13 and 14 increase power consumption since they use many variables. On the other hand, short benchmarks such as 2, 6, 7, 9, present a smaller power consumption increase. In cases such as 3, 6, 8, the use of -o1 optimization leads to extra cycles when compared to -o0. The authors noticed that this arises from unnecessary stack pointer handling.

Optimization levels -o2 and -o3 are defined mainly by the use of hardware loops. Once set up, hardware loops parallelise counter update, comparison and branch operations, thus saving a fixed amount of cycles per loop iteration, mostly by avoiding pipeline stalls. Given that stalls have lower power consumption than normal instruction execution, shortening the programs in this way actually increases power consumption. However the magnitude of this increase depends on how long the loop kernel is and the instructions within it. To a lesser extent, elimination of global common subexpressions further reduces the cycle count and the power consumption.

DSP applications routines were also assessed in terms of cycle count as shown in Table II. Two optimization groups can be identified from the energy consumption reduction point of view: deeply loop based routines and shallow loop based. Deep loop based routines as in the MP3 or the two-dimensional DCT, in which up to four `for` loops are nested, achieve large 75%-80% savings at optimization levels -o2 and -o3. Shallow loop based routines, such as ADPCM, wavelet and RPE-LPC, with up to two nested `for` loops, present reductions of around 50%. In the case of the DCT routine, the optimization -o3 performs worse than -o2 due to the inclusion of an extra register-to-register move in an inner loop.

B. Specific optimizations

Five specific C5510 optimization flags were individually evaluated for the DSPStone benchmarks. Results can be found in Table III for cycle count and code size. The compile for code size flag (-ms) targets the compiler at achieving minimal code size. Shorter programs need less time to be brought from memory but this does not necessarily mean a shorter execution time. The large memory model modifier flag (−ml) forces the processor to use absolute addresses in jump instructions, thus avoiding the need for calculation of relative addresses but leading to longer instruction lengths. As can be seen from the results, -ms and -ml flags have mixed results. In half of the benchmarks the -ms flag provides slightly smaller code sises. The flag -ml seldom outperforms the original -o3 in terms of cycle count although it usually increases code size. Finally, the modifier for the specific processor and silicon revision (-v5510:1.1) and the modifier for on-chip only variables (-mb) were tested. The former avoids certain pipeline stalls, while the latter is intended to allow increase usage of dual MAC operations. No remarkable differences are measured with respect to the original -o3 optimization.

Program	No opt.	-o0	-o1	-o2	-o3
ADPCM	265K	129K (-51%)	129K (-51%)	122K (-54%)	122K (-54%)
MP3 Filterbank	1062K	469K (-56%)	460K (-57%)	264K (-75%)	264K (-75%)
RPE-LPC	355K	227K (-36%)	220K (-38%)	172K (-51%)	172K (-51%)
Wavelet	408K	286K (-29%)	283K (-30%)	193K (-52%)	191K (-53%)
DCT	21695	16971 (-22%)	14567 (-33%)	3957 (-82%)	4005 (-81%)

Table II: General optimization results (cycles)

Bench	-o3	-ms	-ml	-v5510:1.1	-mb
1	14 cy	14 cy	**9 cy**	14 cy	14 cy
	46h by	46h by	46h by	46h by	46h by
2	**6 cy**	6 cy	9 cy	6 cy	6 cy
	40h by	40h by	4Fh by	40h by	40h by
3	**35 cy**	35 cy	35 cy	35 cy	35 cy
	55h by	55h by	59h by	55h by	55h by
4	773 cy	799 cy	**529 cy**	775 cy	773 cy
	D2h by	**CDh by**	DAh by	D0h by	D2h by
5	683 cy	**550 cy**	692 cy	683 cy	683 cy
	146h by	143h by	160h by	146h by	**140h by**
6	**28 cy**	28 cy	30 cy	28 cy	28 cy
	98h by	98h by	A4h by	98h by	98h by
7	**129 cy**	129 cy	129 cy	129 cy	129 cy
	99h by	**97h by**	A2h by	99h by	99h by
8	**1434 cy**	1442 cy	1447 cy	1434 cy	1434 cy
	3Ah by	**33h by**	41h by	3Ah by	3Ah by
9	**30 cy**	38 cy	33 cy	30 cy	30 cy
	73h by	**70h by**	79h by	73h by	73h by
10	127 cy	**126 cy**	128 cy	127 cy	127 cy
	91h by	**90h by**	9Ah by	91h by	91h by
11	**17 cy**	17 cy	18 cy	17 cy	17 cy
	6Dh by	6Dh by	75h by	6Dh by	6Dh by
12	**234 cy**	283 cy	237 cy	234 cy	234 cy
	139h by	13Dh by	141h by	139h by	139h by
13	**250 cy**	250 cy	254 cy	250 cy	250 cy
	D0h by	**CFh by**	E0h by	D0h by	D0h by
14	**94156 cy**	94663 cy	94156 cy	94156 cy	94156 cy
	DBCh by	**D85h by**	DBCh by	DBCh by	DBCh by
15	2554 cy	2554 cy	2554 cy	**2553 cy**	2554 cy
	1821h by	**181Ah by**	1821h by	1821h by	1821h by

Table III: Specific optimizations results for DSPStone (cy-cycles, by-bytes)

C. Comparison compiler - DSPLib library functions

Seven TI DSP assembly language library routines have an equivalent in C, namely FFT, FIR filter (single and dual MAC implementations), autocorrelation, and three different variants of a IIR filter: cascade direct form I and II with 5 coefficients per biquad and direct form II with 4 coefficients per biquad.

The measured energy consumption for the compiled and library benchmarks can be found in Fig. 4. Again it is found that current variation is generally small (up to a 10%) compared to larger variations in cycle count (FFT and FIR especially). In comparison with the manually tuned library functions, the compiler achieves good results for the autocorrelation and the IIR functions, but does not produce competitive results for the FFT and FIR kernels. The authors noticed that circular (modulo) addressing and bit-reverse addressing are never used by the compiler, which is the main reason for the poor behaviour of the compiler for the FFT and FIR routines.

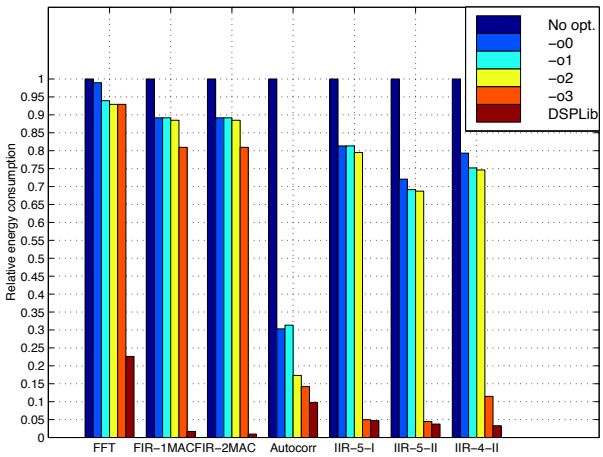

Figure 4: Results for compiled and DSPLib routines

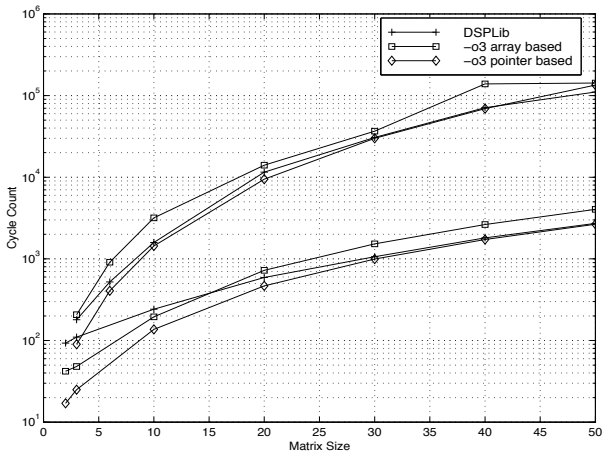

Figure 5: Results for matrix multiplication (above) and matrix-vector multiplication (below) routines

Two different algebraical functions often found in DSP software were also evaluated- matrix product and matrix-vector product. A comparison in terms of cycle count as a function of size, was made for the best compiled version of the C routines. Results show no appreciable difference in cycle count between the library functions and the compiled versions, as seen in Fig. 5. The compiler handles the pointer based routines better than the array based routines, the latter being 16%-25% better.

IV. Recommendations

A. Recommendations for the programmer

The programmer cannot be sure of how well the compiler will perform for a given application. The general rule is that increasing optimization level causes power consumption to increase and energy consumption to decrease, the greater the decrease the larger the number of `for` loops in the core of the algorithm. Given the comparatively small increase in power consumption, it is almost always best to compile with the -o3 flag for performance and energy consumption reasons. It is also worth noting that the compiler produces better results if the program complies with the compiler's C dialect. The authors noted differences in compilation results after the application of certain functional equivalent source code transformations, for instance, array based data references give worse results than pointer based.

The use of DSPLib routines is a good idea for most DSP related subroutines, but forces the data to be set up in a certain way in memory, which is not always possible. Use of assembly code program segments is the only option when programming bit manipulation routines and modulo addressing. Use of specific optimization flags is worth checking but significant improvements are not expected.

B. Recommendations for the compiler designer

It is difficult to suggest changes in the structure of the compiler since it is a black box. The one pass optimization strategy is simpler than the iterative improvement approach suggested in [17], [18]. The compiler in general works well but still misbehaves occasionally, for example -o0 sometimes provides better results than -o1, due to unusual stack pointer management. For the DCT routine, -o2 provides better results than -o3, due to the removal of a `move` instruction from a loop.

The authors noticed that circular (modulo) addressing and bit-reverse addressing are never used by the compiler, which is the main reason for the poor results for the compiler when compared with DSPLib for the FIR and FFT routines. The transformation of variable-limited loops into hardware loops depends heavily on how close the loop structure is to the compiler's dialect. Pragmas are provided by the compiler to counteract this problem. Examples given in this work suggest that the compiler is targeted mainly at recognizing loops. Mainly sequential code segments might be further improved by appropriate reordering of instructions to avoid stalls. This might be achieved by means of an increased compiler peephole length and utilization of a larger number of analysis steps.

C. Recommendations for the hardware engineer

The Instruction Buffer Queue is one of the most significant factors in this processor in terms of performance. In early optimization stages, stalls account for about half of the program's execution cycles. Around 75% of these wasted cycles come from inefficient utilization of this subsystem when no optimization is applied. Thus the large savings achieved by optimizations -o2 and -o3. The combination of one "repeat short loop" plus two hardware loops, is not sufficient for all applications. Two dimensional filtering applications like two dimensional DCT and Motion Prediction nest at least four `for` loops. Increasing the number of hardware loops would improve performance for these applications. Similarly, increasing the size of the "short loop", now limited to 64 bytes, would speed up applications with large inner `for` loop.

V. Conclusions

This paper describes an investigation into the impact of the compiler on power and energy consumption of the TMS320VC5510 DSP Processor. The methodology consisted of measuring the current consumption of the processor when using the different compiler compilation options across a range of DSP programs. Power and energy consumption were measured for different compilation optimization levels and specific optimization for benchmarks extracted from the DSPStone and some general DSP applications. Finally, the results of a comparison between the TI DSPLib kernel library optimized assembly functions and compiled equivalents was performed. Energy consumption savings for general optimization stages are large and come mainly from cycle count reduction. Power is slightly increased primarily due to increases in Instruction Level Parallelism. Stalls account for more than half of the execution cycles in early optimization stages. The number of stalls is reduced by the use of the IBQ in the higher optimization levels. As a general conclusion, power increases with optimization level by up to 30% (8% on average) while energy consumption is reduced by between 0% and 96% (35% on average). Some typical applications in the range of the processor were also examined showing savings of 82% to 54% in energy consumption, depending on the nature of the algorithm - mainly the number of nested `for` loops in the core of the algorithm. Specific compiler flags were examined and their effects tested for the DSPStone benchmarks. The effect of the specific optimization flags was in general not significant and not easily predictable. General DSP and math routines were compared to library routines showing mixed results. Sometimes the compiler achieved good performance (autocorrelation, IIR, matrix product), while in several other cases it proved to be worth calling the library routine (FIR, FFT). Finally, some recommendations for further improving power and energy consumption were provided for the programmer, the compiler designer and the hardware engineer.

Acknowledgements

This work is sponsored by Enterprise Ireland (EI) under agreement number PC/2004/311. The authors would like to thank the UCD Department of Electronic, Electrical and Mechanical Engineering, for their support.

References

[1] J. Staunstrup and W. Wolf (eds), *Hardware/Software Codesign: Principles and Practice*, chapter 5, Kluwer Academic Publishers, 1997.

[2] V. Zivojnovic, "Compilers for digital signal processors, the hard way from marketing to production tool," *DSP and Multimedia Technology*, vol. 4, no. 5, pp. 27–45, July/August 1995.

[3] V. Zivojnovic, J.M. Velarde, C. Schlager, and H. Meyr, "DSPstone: A DSP-oriented benchmarking methodology," Tech. Rep., Aachen University of Technology, 1994.

[4] Texas Instruments Inc., *TMS320C55x DSP Library Programmer's Reference*, 2004.

[5] L. N. Chakrapani, P. Korkmaz, III V. J. Mooney, K. V. Palem, K. Puttaswamy, and W. F. Wong, "The emerging power crisis in embedded processors: what can a poor compiler do?," in *CASES '01: Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems*, New York, NY, USA, 2001, pp. 176–180, ACM Press.

[6] M. Kandemir, N. Vijaykrishnan, M.J. Irwin, and W. Ye, "Influence of compiler optimizations on system power," in *Design Automation Conference*, 2000, pp. 304–307.

[7] M. Valluri and L. John, *Interaction between Compilers and Computer Architectures*, chapter 6. Is Compiling for Performance == Compiling for Power?, Kluwer Academic Publishers, 2001.

[8] V. Tiwari, D. Brooks, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *27th Annual International Symposium on Computer Architecture*, June 2000, pp. 83–94.

[9] J.S. Seng and D.M. Tullsen, "The effect of compiler optimizations on Pentium 4 power consumption," in *9th Annual Workshop on Interaction between Compilers and Computer Architectures (INTERACT-9), and 11th IEEE International Symposium on High-Performance Computer Architecture*, 2005.

[10] Texas Instruments Inc., *TMS320C55x DSP CPU Reference Guide*, 2004.

[11] Spectrum Digital Inc., *TMS320VC5510 DSK Technical Reference*, 2002.

[12] The Netherlands Stichting Mathematisch Centrum, Amsterdam, "Intel/DVI ADPCM coder/decoder," [web page] `http://www.cwi.nl/`, Sep. 1992.

[13] T. Jansson, "Bladeenc," [web page] `http://bladeenc.mp3.no/`, Ago. 2002.

[14] J. Degener and C. Bormann, "GSM 06.10 13 kbit/s RPE/LTP speech compression," [web page] `http://kbs.cs.tu-berlin.de/~jutta/toast.html`, Jul. 2002.

[15] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: principles, techniques, and tools*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[16] M. Casas-Sanchez, J.A. Rizo-Morente, and C.J. Bleakley, "Power consumption characterisation of the Texas Instruments TMS320VC5510 DSP," in *Proceedings of Power and Timing Modelling, Optimization and Simulation (PATMOS) 2005*, V. Paliouras, J. Vounckx, and D. Verkest, Eds., 2005, vol. 3728, pp. 561–571.

[17] G. Fursin, M.F.P. O'Boyle, and P.W.M. Knijnenburg, "Evaluating iterative compilation," in *Proceedings of LCPC 2002*, 2002.

[18] S.V. Gheorghita, H. Corporaal, and T. Basten, "Iterative compilation for energy reduction," *Journal of Embedded Computing*, 2005.