A Quantitative Code Analysis of Scientific Systolic Programs: DSP vs. Matrix Algorithms

R. Sernec
BIA d.o.o.
Ljubljana
Slovenia
radovan.sernec@guest.arnes.si

M. Zajc J. F. Tasič
Faculty of Electrical Engineering
University of Ljubljana
Slovenia
matej.zajc@fe.uni-lj.si

Abstract

In this paper we consider systolic programs of the most common DSP (convolution, FIR, IIR, FFT) and Matrix (multiplication, triangularisation, linear equation solving, modified Faddeev algorithm) algorithms, executed on systolic arrays of various topologies (linear, 2D mesh, hexagonal). We examine the algorithm-specific parameters (number of I/O paths, unit delays) and program-dependent parameters (program length, data location requirements, basic block lengths, branch behaviour, instruction usage, computation to communication ratio) of our program set, executed on a single processing-cell of systolic arrays. The analysis is based on the static object code

We found that basic block lengths are 17.1 (DSP) and 8.4 (Matrix) instructions long. The Divide/Square Root operations play a major role in Matrix algorithms (more than 15% of the weighted instruction set). Inter-cell communication must be efficient, since the computation to communication ratio is only 1.2 – 1.4 and is orders of magnitude smaller than in typical MIMD applications.

1. Introduction

The approach to the design of a new parallel processor architecture requires a thorough analysis of the application domain in which such a new architecture should perform. Statistics of relevant parameters are gathered. In the commercial microprocessor market there exists a variety of different benchmarks, which best describe certain application environments [1]. Unfortunately, these benchmarks are totally inadequate for the analysis and behaviour of systolic or SIMD algorithms and programs, which run on a single processing cell within a systolic array [2].

To the authors' knowledge there is no available information on the behaviour, statistics and requirements of DSP and Matrix algorithms executed on systolic processing elements.

We approached the problem by selecting the algorithms relevant to the planned architecture's application environment, analysing them and determining the features that are most important for the design.

The remaining chapters deal with the algorithm selection used in the analysis; the assumptions made in the program compilation; and the simulation methodology applied. The results show algorithm-dependent parameters, followed by program-dependent parameters.

2. Problem identification and algorithm selection

The analysis covered a set of systolic algorithms most frequently used in various DSP applications. Analysis results can be used as guidelines in designing systolic DSP architecture. It soon became clear that the inclusion of systolic algorithms for manipulation of matrices would greatly expand new DSP systolic architecture capabilities. Algorithms analysed were divided into two classes: classic DSP [3] (1D and 2D convolution, FIR, IIR, FFT radix-2 DIT, FFT radix-2 DIF) and those for matrix manipulation [4], [5] (transpose, triangularisation with pivoting, triangularisation with Givens rotations, triangular matrix inversion, triangular band matrix inversion, vector–matrix multiply/add, matrix–matrix multiply/add, linear equation solving, modified Faddeev algorithm).

3. Simulation methodology

We should point out that algorithms in systolic form, executing on each processing cell of the systolic array, are analysed in this paper. All results presented are based on a static compiled object code. No program tracing was done. The algorithms were coded in C programming language. No compiler optimisations were allowed, so the results presented here are the worst cases. Each program set had at least one solution that was compiled and analysed. In some Matrix systolic algorithms the array is composed of two distinct processing cells, each executing a different program. We treated these cases as being part of the same algorithm group, but two different programs were written. Altogether close to 50 such programs were analysed. Data input was limited to vectors of size 1024 (square array) or 1024^2 (linear array) and code analysis was done for the same number of loop iterations. All data was in floating-point, single precision format. Therefore it was quite easy to distinguish between integer and floating-point data in each program and thus more straight-forward to determine the size requirements of respective register files. The target processor architecture chosen was the Texas Instrument's TMS320C30 [6], since DSP processors have the ability to address data in a variety of ways.

4. Analysis of algorithm dependent parameters

Algorithm dependent parameters, presented in this section are the number of communication channels and the number of unit delays. These parameters are characteristics of algorithms and are not dependent on language constructs used in the simulation.

4.1 Communication channels

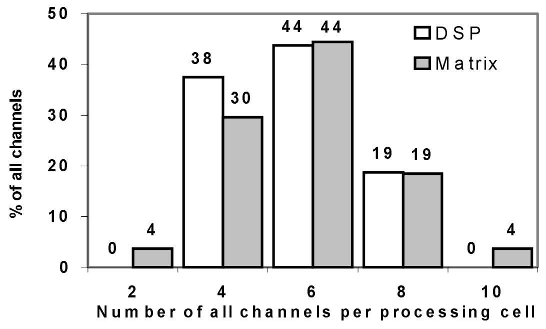

Figure 1: Communication channels per processing cell distribution; the larger number of channels in Matrix programs is due to the conditional execution within a processing cell, the conditions being transferred along with the data. Almost 80% of programs have six or fewer channels.

Each systolic processing cell communicates with its neighbours through uni-directional data paths called communication channel. The simulation of data transfers over communication channels was done by load/store instructions to memory locations. Figure 1 shows the distribution of the number of communication channels per processing cell.

In our analysis only one piece of data was allowed to convey over a single communication channel. The higher numbers of communication channels (more than four) in DSP programs are due to 2D filtering and complex FFT algorithms.

4.2 Unit delays

To achieve the proper synchronisation of data flow through a processing array, delay elements are used. Delay implementation in C language was done via assignments to temporary variables. Almost 90 % of DSP programs require up to three unit delays and two thirds of Matrix programs only one.

5. Analysis of code dependent parameters

Program, or code dependent parameters, were used in the static code analysis of compiled C programs of algorithms, running on each processing cell. The selected parameters are: program length, data storage requirements, register requirements – DSP architecture, addressing displacements – DSP architecture, integer register requirements – RISC architecture, floating-point register requirements – RISC architecture, basic block lengths, weighted static instruction count by instruction class, branch types and computation to communication ratio. The instruction set was divided into seven classes. Two processor architecture paradigms (DSP and RISC) were used when comparing register and external data requirements. It is important to know which addressing modes are used in the processing cells of a systolic array for fetching local data and for sending data to neighbour processing cells. The number of registers used for the implementation can be an important factor.

5.1 Program length

Program sizes of all compiled, unoptimised object C code for each systolic processing cell are less than 2 KBytes. The number of instructions in those programs peaked at around 60 and 90, respectively, for both categories.

5.2 Data storage requirements: general

This parameter directly influences the size of the register file, or local data memory, in each processing cell.

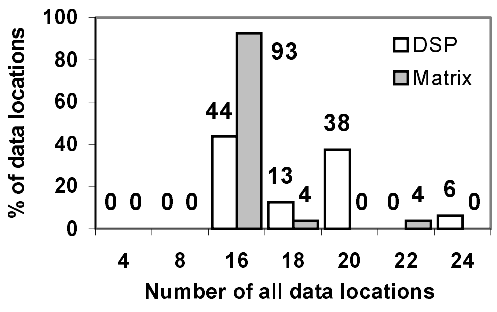

Figure 2: Data locations distribution; more than 50% of DSP and almost 90% of Matrix programs require up to 16 data locations, including ones for incoming data, temporary storage variables and program constants. The increased number of data locations in DSP programs is due to the software implementation of unit delays.

Figure 2 summarises the data storage requirements, which include all integer variables for loop management, floating-point constants, variables or data stored internally in the processing cell, and floating-point data transferred over communication channels. Data format of all data types was 32-bit word.

5.3 Data storage requirements: DSP architectures

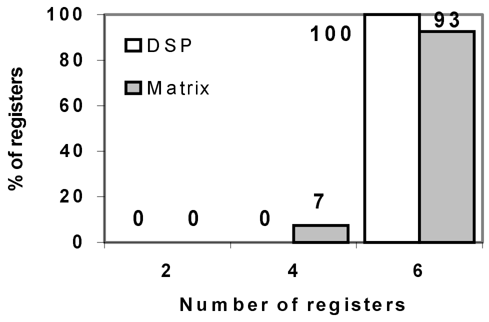

Figure 3: Register requirements in DSP-like processors; usually only a stack pointer, address register, loop counter and data accumulator suffice.

DSP processors usually store data in temporary accumulator and external data memories and contain only a small register file for managing program flow (stack pointer, loop registers, addressing registers). Figure 3 shows the register requirements for such architecture. There were only two distinct addressing modes used: indirect with pre and post-displacement add.

5. 4 Data storage requirements: RISC architectures

By interpreting the above results in a different way we can determine the required number of integer and floating-point registers, which are used in RISC processors.

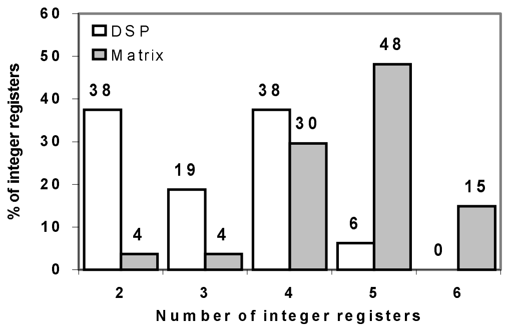

Figure 4: Integer register requirements in RISC like processors

Floating-point register file size can best be determined from Figure 5.

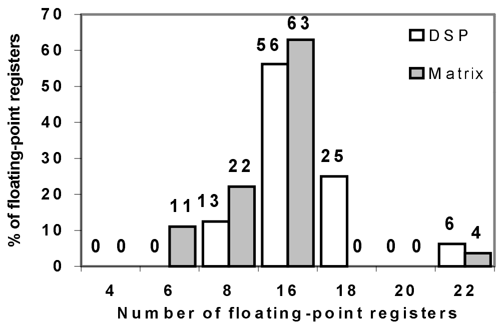

Figure 5: Floating-point register requirements in RISC like processors; almost 70% of DSP and 90% of Matrix programs use 16, or less, FP registers. Note that floating-point data types were used throughout the analysis.

5.5 Basic block sizes

By definition, basic block is a fragment of code without any control flow redirections, excluding only the exit branch. In our analysis we adhered to this definition. Its size influences the percentage of branches (including jumps and procedure calls) in programs, as well as potentially available instruction-level parallelism within one basic block [7], [8]. Results are presented on Figure 6.

We noted from the object code that the exit code fragment is the same for all programs and its size was 4 instructions. DSP algorithms have, in general, much longer basic block lengths and thus offer potentially much greater instruction level parallelism than their Matrix counter parts. It is true that all DSP algorithms can be represented with data flow graphs. From this structure it can be observed that functional units can be arranged in the same way as to allow pipelined data flow through the cell and thus exploit as much parallel operations as possible.

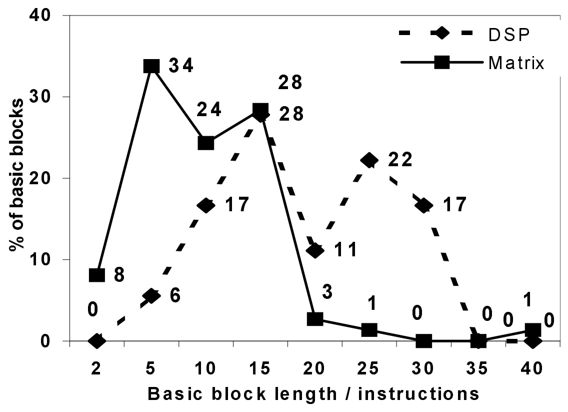

Figure 6: Basic block lengths without exit code fragment; the exit code fragment is usually 4 instructions long. More than 45% of DSP programs have run lengths longer than 20 instructions in this case. Observe that DSP programs have more than 100% longer average run-lengths.

5.6 Instruction set usage characteristics

Instruction classes	Latency
Load/Store	1
Branch	2
Integer Add/Sub	1
Integer other	1
Floating-point Add/Sub	3
Floating-point Multiply	3
Floating-point (Divide/Square Root) $^{-1}$	15

Table 1: Instruction classes, with their corresponding assigned latencies/weights used in static instruction count analysis; latencies are based on current high-end RISC processors.

We divided the whole instruction set of the object codes analysed into 7 instruction classes, as shown in Table 1.

We included Load/Stores only for data I/O over communication channels and movement of intermediate results, not taking into account any register and variable initialisation, which happened at the beginning of each program. “Branch” class includes conditional and unconditional branches, as well as jumps and procedure calls. In “Integer other” category stack-pointer manipulation instructions (Push, Pop) can be found, integer comparisons and instructions for conversion between integer and floating-point values and vice versa. Note that the division and square root operations were done through inverse functions, which were post-multiplied by explicit multiply instruction to obtain a final result. Square root operation was always present as a reciprocal operation in Matrix programs.

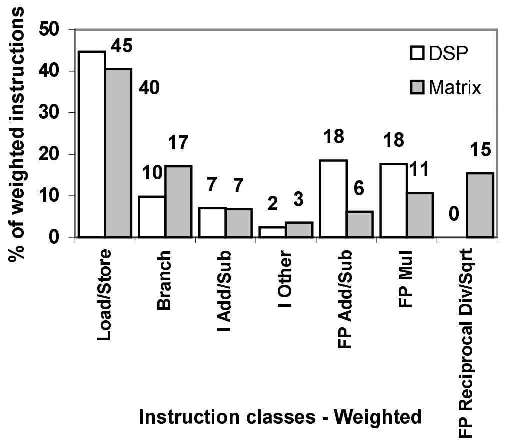

Figure 7: Weighted instruction set classes distribution; note the significant rise in the proportion of FP operations, especially for FP Div/Sqrt class, which experienced almost a 900% rise compared to the unweighted distribution. This makes them equally important as FP Add/Mul classes of instructions, in order to maintain high execution speeds in Matrix programs.

Latencies assigned to each instruction set group are not those of the TMS320C30 DSP processor, for which code compilation was done, but are rather based on contemporary superscalar RISC [9]. The most notable exception here is the latency of Load/Store instructions. We assumed a 1-cycle delay only for data coming from neighbouring cells within a single chip. Usage frequencies of each instruction group are weighted with execution latency times of instruction set classes (Figure 7). The Load/Store instructions account to 40% - 45%, but FP Div/Sqrt instruction class becomes a very important factor and should not be neglected when designing processing elements for the execution of Matrix algorithms. FP Mul/Add instructions present larger burden in DSP programs, mainly due to the lack of FP Div/Sqrt and a smaller number of branches.

5.7 Branch statistics and behaviour

Conditional branches in DSP are due to the coding of the algorithm as a main loop which is repeated as often as there are input data, and this loop contains a conditional branch. Unconditional branching is done at the end of all programs in exit-code fragment, as mentioned before, even for parts of both types of branches for all programs, although in Matrix programs this is not attributed to the same reason, but rather to the many more branches used in each program. On average, there are only two branches present in DSP and four in Matrix programs, with a maximum of 13 branches present in linear equation solving algorithm. Even portions of both kinds of branches in DSP programs are due to one unconditional exit-code branch and one loop conditional branch present in most DSP programs.

5.8 Computation to communication ratio

This measure is used many times in characterisation of MIMD programs to find the best architecture [Hwan 94]. If this ratio is small, than executing such a program on machines with long communication start-up times and long interprocessor latencies would give bad results in terms of execution time. A large ratio, on the other hand, means that a program is partitioned in a rather coarse-grain fashion and that a processor will spend lots of its time processing and, whilst performing well, the communication mechanisms are not the fastest.

These ratios are much smaller in systolic arrays than in MIMD machines and are 1.24 to 1.4 respectively for the two algorithm categories obtained on weighted instruction set. These figures are also due to software implementation of unit delays and use of explicit Load/Store instructions for interprocessor cell communication. These results suggest that it is imperative to devise efficient, yet simple communication mechanisms for systolic arrays.

6. Conclusion

We performed algorithm and compiled object static code analysis of programs executed on single systolic processing cell. Algorithms were divided into DSP and Matrix classes. All data had floating-point format. Algorithm analysis examined the number of communication channels in each cell and the number of unit delays required for proper synchronisation of the whole systolic array. Communication channel usage is balanced for both algorithm classes and approx. 80% of them require 6 or less I/O data paths. Unit delays are much more extensively used in DSP programs, which is attributed to their natural data flow. Program sizes of the systolic cell algorithms are below 2 KBytes and usually require less than 20 data storage locations altogether. Data storage requirements are presented from the point of view of DSP and RISC processor architectures and, as such, show register requirements (integer and floating-point) as well as addressing modes and their displacements (90% of programs used up to 16 words) used in data access. DSP programs contain branches only for loop management and subroutine returns, whereas Matrix counter parts use branches more extensively (4 branches per program on average). The average basic block lengths are much longer in DSP programs (17.1 vs. 8.4 instructions). The most important finding of the weighted instruction set analysis is that Load/Store instructions for interprocessor cell communication play a major role. Also Divide/Square Root instructions are a very important element in Matrix programs and should not be neglected. The computation to communication ratio balanced for both algorithm categories, and is around 1.5. This shows that in order to achieve high throughput in systolic processing cells one has to design an efficient communication mechanism inside.

Overview of the architecture which resulted from this study is published elsewhere [10].

7. References

[1] J. Hennessy, *Computer architecture: A quantitative approach*, Second edition, Morgan Kaufmanm, 1996.

[2] H. T. Kung, “Why systolic architectures”, *Computer*, Vol. 15, No. 1, pp. 37-46, 1982.

[3] N. Petkov, *Systolic parallel processing*, North-Holland, 1992.

[4] W. M. Gentleman, “Matrix triangularisation by systolic arrays”, *SPIE, Real Time Signal Processing IV*, Vol. 298, pp. 19-26, 1981.

[5] J. G. Nash, “Modified Faddeeva algorithm for concurrent execution of linear algebra operations”, *IEEE Trans. on Comp.*, Vol. 37, No. 2, pp. 129-136, Feb. 1988.

[6] Texas Instruments Inc., *TMS320C30 User’s Manual*, 1993.

[7] D. W. Wall, “Limits of instruction level parallelism”, *APLOS-IV*, pp. 176-188, 1991.

[8] M. S. Lam, “Limits of control flow on parallelism”, *19th Annual Symp. on Comp. Arch.*, pp. 46-57, 1992.

[9] Digital Equipment Corp., *Alpha 21164 User’s Manual*, 1995.

[10] R. Sernec, M. Zajc, J. Tasic, "Multithreaded Systolic/SIMD DSP Array Processor – MUS2DAP", *Workshop SiPS97*, Leicester, UK, Nov. 1997.