

# New DSP Benchmark based on Selectable Mode Vocoder (SMV)

Erh-Wen Hu, Cyril S. Ku, Andrew T. Russo, Bogong Su, and Jian Wang

## Abstract

*Digital signal processing (DSP) industry has been growing rapidly over the past few years; it remains the technology driver for the recovering semiconductor industry. Performance evaluation is essential for the users and manufacturers of DSP processors. Since DSP application programs become larger and more complicated, people need new benchmarks for performance evaluation of different DSP processors. We build a new DSP benchmark based on Selectable Mode Vocoder (SMV), a speech-coding program from the 3G wireless applications. Our new DSP benchmark, called SMV benchmark, consists of eight kernel functions. In this paper, we introduce the criteria of selecting kernels and our methodology to build SMV benchmark. We also discuss the characteristics and static analysis of the kernels.*

**Keywords:** Digital signal processor, DSP, benchmark, vocoder, optimizing compiler, instruction level parallelism.

## I. Introduction

Digital signal processing (DSP) industry has enjoyed rapid growth over the past few years and the growth is expected to continue at an annual rate of 15 % for next few years [16]. To address the need for many different applications, various types of DSP processors based

on different architectures such as enhanced conventional DSP, SIMD, DSP/Microcontroller hybrids, VLIW, general-purpose microprocessor with multimedia extensions, special-purpose multimedia processors, and configurable processors have been introduced to the market [6, 19]. As a result, performance measurement of DSP processors becomes an important issue for both the manufacturers and the users. DSP manufactures rely on the performance data to improve their design and customers need them to select the processors that meet their requirements in a cost effective way.

Recently, several new benchmarks have been proposed for DSP processors [2, 7]. However, these benchmarks are not without problems. For example, the BDTI 2000 benchmark of Berkeley Design Technology, Inc. uses 12 DSP kernels hand-coded in assembly language to measure the performance of DSP processors. Most of these 12 program segments are single-block code only, which are not representative of the increasingly complicated new application programs. Moreover, handwriting the assembly code for a new DSP processor can be time consuming and are not objective. Another recent benchmark of interest comes from EEMBC or EDN Embedded Microprocessor Consortium [7]. The EEMBC benchmark incorporates more than 30 core algorithms used in five areas of applications: the telecommunications, networking, automotive, industrial automation, consumer electronic, and office automation. However the EEMBC benchmark is not readily accessible to academic researchers because of the high cost of joining the consortium. There are two other recent benchmarks: the MediaBench [12] and the MiBench [10] benchmarks. However, these two benchmarks are based on some old applications. In this paper, we select benchmark from a new application.

Our new DSP benchmark, referred to as the SMV benchmark hereafter, consists of eight kernels chosen from the Selectable Mode Vocoder (SMV) application program [3]. SMV is a new speech coding technology that provides significant capacity and quality improvements on the CDMA 2000 system.

E. W. Hu is with the Department of Computer Science, William Paterson University, Wayne, NJ 07470, USA (e-mail: hue@wpunj.edu)

C. S. Ku is with the Department of Computer Science, William Paterson University, Wayne, NJ 07470, USA (e-mail: kuc@wpunj.edu)

A. T. Russo was a CS major student at William Paterson University. He is now a Ph. D student in the Department of Computer Engineering, University of Delaware, Newark, DL 19716, USA (e-mail: xavier776@yahoo.com)

B. Su is with the Department of Computer Science, William Paterson University, Wayne, NJ 07470, USA (e-mail: sub@wpunj.edu)

J. Wang is with Wireless Speech and Data Processing, Nortel Network, 2351 Blvd. Alfred-Nobel, St. Laurent, QC, Canada, H4S 2A9 (e-mail: jiwang@nortel.com)

A speech coder compresses the speech signals into digital code and the decoder does the opposite [1]. The international standards body 3GPP2 has recently completed the development of the SMV algorithm and released it to the public for implementation. The advanced technology utilized in SMV benefits the subscribers with superior voice quality while allowing service providers to increase capacity dynamically as needed.

The major reasons for selecting the SMV application as the basis of our new benchmark are as follows:

1. Wireless communication is a major DSP application which accounts for more than two thirds of the DSP processor market share today [15]. To measure the performance of their DSP processors, some manufacturers still use the 2G wireless applications as their benchmarks. Since CDMA 2000 is a very efficient 3G technology available today [4, 5], it is expected to replace the 2G technology in the near future. Furthermore, since the speech coding programs are a key part of the CDMA 2000 system, choosing them as the DSP benchmark is of great practical importance. To our knowledge, there has been no report on using the 3G wireless applications as DSP benchmarks.

2. [19] uses a few DSP kernels and multimedia applications including speech coding and compression programs as the benchmark to compare the performance between the general-purpose Pentium II processor and TIC62, a high end of DSP processor. They found that although TIC62 performs well on DSP kernels, its performance on more complicated applications is relatively poor largely due to the inability of its compiler to exploit the instruction-level parallelism for frequent control-dependent data dependencies in the application programs. Therefore, how to improve the performance of speech coding of the 3G wireless applications on DSP processors continues to pose a big challenge to DSP architectural design and to improving compiler optimization techniques. Furthermore, because of Moore's Law, embedded systems are moving to in the direction of embedded computing [8], i.e. the application software will be larger and more complicated in the years to come. Newer and larger application programs must be used to evaluate and choose hardware.

This paper is organized as follows: Section II discusses the criteria for selecting the kernels, Section III presents our methodology, Section IV introduces kernel functions, Section V presents the static analysis

of kernels, and Section VI provides concluding remarks and discusses the future work.

## II. Criteria of Kernels

Our goal is to design a new DSP benchmark and use it to measure the performance of the DSP platform [9], which includes the DSP processor and its compiler. In addition, we also seek to obtain useful information from the performance result and relate it to DSP architecture and compiler design. In order to facilitate the selection of kernels from hundreds of functions in the SMV application as benchmarks, we define our selection criteria below.

1. Most frequently executed leaf functions

Kernels must have long execution time, so we first identify functions that are most frequently called upon. Furthermore, these functions must not call other functions so that instruction level parallelism in the code can be effectively exploited.

2. Representative of other functions of similar code structure

There are certain groups of functions that exhibit similar code structure. Each of these groups of functions also accounts for a significant portion of the total execution time of the entire SMV program. We select from each of these groups a function that is most representative of the entire group.

3. Complexity

In order to be able to measure the effectiveness of the optimizing compiler in DSP compiler and DSP architecture, kernels must have certain level of complexity. Complexity may be measured by using the number of DSP operations in the innermost loops of kernels and in terms of the modified cyclomatic complexity defined in Section III below.

4. Diversity

Since the major task of our SMV benchmark is to evaluate DSP compilers and architectures, we select functions with diverse types of kernels so that the optimization approaches used in various DSP compilers can be effectively assessed.

### III. Methodology

#### 1. Build profiling tool

Our work is based on SMV v3.5fx program downloaded from [20]. SMV v3.5fx is written in C and can run on PC platform where all DSP operations are simulated by small C functions stored in a library. The *gprof* is a popular profiling tool in UNIX; however, it is not suitable for our study as described below. When the SMV program runs on a general-purpose computer such as a PC, all DSP operations are simulated by the built-in library functions. Consequently, each of these DSP operations often takes many machine cycles to complete. But when the SMV program runs on a real DSP machine or its simulator, the execution of most DSP operations takes only a single cycle. Therefore the result of *gprof* does not truly reflect the profiles of real DSP processors.

To solve the profiling problem, we extend the functionality of the WMOPs tool, where WMOPs stand for Weighted Million of Operations that can be used to represent the execution time when the SMV program runs on a DSP processor. In the SMV program, there are some small functions capable of calculating the weighted DSP operations at function level. We modified these small functions and extended their capabilities such that they can be used to obtain profiling information on nested function calls and on nested loops. They can also be used to measure the worst case, the maximum and minimum data, and the net WMOPs of any function. We apply these modified tools to nearly 100 functions and more than 250 loops [18].

#### 2. Define modified cyclomatic complexity CC\*

Cyclomatic complexity (CC) measurement was introduced in 1976 [13, 14] and has since been applied extensively in software engineering to calibrate and measure the complexity of programs. To adapt it to the analysis of more complex code structure in this study that involves instruction level parallelism, we modified its definition and used CC\* values of the modified version to compare functions, to evaluate the diversity and complexity of these functions, and to determine which of these functions are to be selected as kernels. A program can be graphically depicted by a control flow graph. In the control flow graph,  $CC = e - n + p + 1$ , where  $e$ ,  $n$  and  $p$  denote the number of edges, nodes, and connected components, respectively. In fact, CC is actually the number of independent paths through the control flow graph. It can be proved that CC is equal to the number of conditions (loops and

branches in a program) plus one. If there are two programs, one with two sequential loops and one with two nested loops, the CCs for these two programs are the same since both programs contain two conditions only. Considering the fact that it is more difficult to optimize nested loops and branches at instruction level, we take nested levels into account and define a modified cyclomatic complexity CC\*, which equals the number of loops and branches in the program, plus the number of nested levels of loops and branches.

CC or CC\* is the static analysis based on the structure of a program. It relates to the anticipated difficulties when testing, maintaining, and understanding of the software. In our future research, we will investigate the use of dynamic (execution) complexity of algorithm such as asymptotic complexity or Big-O [11] to categorize the functions.

#### 3. Kernel selection

SMV v3.5fx consists of two parts: the decoder and the encoder. We first used a sample audio file to test the correctness of both parts. We then focused our study on the encoder only because both parts have very similar functions and the encoder is more complicated accounts for 86% of WMOPs of the whole SMV program.

Considering the fact that the encoder contains 300 functions, we could not instrument our measuring code on all functions, either we could not instrument on all 82 top level functions, it is too many and some low level functions have more WMOPs because which are called by many top level functions. We adopted the divide and conquer technique. First of all we ran the encoder with the original WMOPs measuring code which produced WMOPs data of 27 functions, then we instrumented 86 functions including 10 functions, with highest WMOPs number, and their children and grandchildren. Second, by using a threshold 50 WMOPs we selected 44 functions from those 86 functions as shown in Figure 1, the total WMOPs of those 44 functions is 87 % of the entire Encoder. Part of the SMV program.

We then use combined WMOPs (total WMOPs of function and its similar functions), number of DSP operations in innermost loops and CC\* as the criteria to select kernels.

#### 4. Design testing programs for kernels

We run the sample audio file on SMV program and gather inputs to and outputs from the eight kernel

functions in the SMV program. These input and output data are then used in the testing programs for kernels.

## IV. Kernel Functions of the SMV Benchmark

The SMV benchmark consists of the following eight kernel functions:

*LSF\_Q\_New\_ML\_search\_fx*:

*LSF\_Q\_New\_ML\_search\_fx* performs the ML search. It is the most time consuming function in the SMV program, accounting for 12% of the WMOPs of the entire SMV encoder. *LSF\_Q\_New\_ML\_search\_fx* is a long and complicated function with a high CC\* value and contains a 4-level nested loop, although the innermost loop is not too complicated.

*FLT\_filterAP\_fx*: *FLT\_filterAP\_fx* performs frame data all poles fixed point filtering in the SMV program. Combining with two other similar functions its WMOPs account for more than 15% of the entire SMV programs. *FLT\_filterAP\_fx* is a very short function having a simple 2-level nested loop; its innermost loop contains only two lines.

*PIT\_LT\_Corr\_Rmax\_fx*: *PIT\_LT\_Corr\_Rmax\_fx* performs a first pitch determination by finding the maxima in the autocorrelation. It has a 3-level nested loop. The innermost loop is very simple with one MAC operation only; however, the second level loop is quite complicated.

*FCS\_Excit\_Enhance\_fx*: *FCS\_Excit\_Enhance\_fx* just simply performs arithmetic operations on an array. It has three very similar 2-level nested loops with simple innermost loop. However, it is good to evaluate the optimization approach to nested loops.

*C\_fft\_fx*: *c\_fft\_fx* is a decimation-in-time complex FFT/IFFT function. It is a little bit more complicated than the usual *fft* program with a moderate CC\* value. Combining with a similar function *r\_fft\_fx*, the total WMOPs is about 5% of the entire of the SMV encoder.

*LPC\_Chebbs\_fx*: *LPC\_Chebbs\_fx* evaluates the Chebichev polynomial series in the SMV program, which is frequently called in SMV program. *LPC\_Chebbs\_fx* is a simple function having a single level “for” loop with many DSP operations; it is good to evaluate loop optimization approaches of compiler such as software pipelining.

*LPC\_autocorrelation\_fx*: *LPC\_autocorrelation\_fx* calculates the auto-correlation for a given data vector.

It is a very simple function. However combining with some other similar functions, the total WMOPs is almost 11% of the entire SMV encoder.

*Fcb\_add\_sub\_contrib\_phi*:

*fcb\_add\_sub\_contrib\_phi* removes one pulse’s contribution. It is just a single 2-level nested loop in the function, which is good to evaluate the instruction-level parallelism cross over branches.

Table 1 (at the end of the paper) presents the characteristics of the eight kernels. Figure 2 shows their WMOPs. The total combined WMOPs of the eight kernels account for 58% of the entire SMV encoder.

## V. Benchmark Analysis

### 1. Cyclomatic complexity of kernels

[14] categories the complexity of a program in terms of a CC value. Programs with CC values between 1 and 10 are defined as *simple*. A CC value between 11 and 20 is termed *moderate complex*, and between 21 and 50 *complex*. Based on their CC\* values, Table 1 shows that 4 out of the 8 kernels in the SMV benchmark are simple. 3 are moderate, and 1 is complex.

### 2. Loop behaviors

Since loop optimization is an important issue in DSP compiler design, we focus on loop behaviors of the kernels and compare them with loops in the SMV program. For simplicity, we selected 46 functions with the highest number of WMOPs to represent the entire SMV program. These 46 functions, referred to as major functions, account for about 87% of the total number of WMOPs of the entire Encoder program.

#### (1) Type of innermost loops

We focus on the types of the innermost loops which are the target of most optimization approaches due to their high rate of execution. Table 2 shows that most loops in both the major functions of the SMV Encoder and in the kernels are “for” loops; “while” loops only account for less than 5%, suggesting that the number of iterations is static.

Table 2: Loop types of the innermost loops

Loop type	Major functions in SMV encoder	Kernels
for	96%	95%
while	4%	5%

## (2) Extra exits

Table 3 shows all the innermost loops in kernels that have no extra exits.

Table 3: Extra exits in innermost loops

No. of extra exits	Major functions in SMV encoder	Kernels
0	97%	100%
1	3%	0%

## (3) Maximum (max) and minimum (min) iterations

We measure both max and min numbers of iterations of the innermost loops, which helps determine whether these loops can benefit from the software pipelining technique. Loops with small max and/or min number of iterations are not suitable to apply software pipelining technique. Table 4 shows that the majority of the innermost loops can benefit from the software pipelining. However, 45% of the innermost loops are with small min iterations. Comparing with only 6% of the innermost loops with small min iterations in regular DSP kernels [17], the software pipelined code in SMV will be more complicated.

Table 4: Max and min iterations in loops

max	min	Major functions in SMV encoder	Kernels
$\leq 4$	/	25%	5%
$> 4$	$\leq 4$	20%	45%
$> 4$	$> 4$	55%	50%

## (4) Conditional branches

Table 5 shows that most innermost loops contain no conditional branch. For the rest of the innermost loops, both the number of conditional branches and the depth of nested conditional branches are not very large.

Table 5: Conditional branches in loops

Nested levels of Branch	No. of conditional branches in loop	Major functions in SMV encoder	Kernels
0	0	76%	79%
1	1	14%	11%
	2	2%	11%
	3	2%	0%
2	1	3%	0%
	2	2%	0%
4	1	1%	0%

## (5) Nested loops

Table 6 categorizes nested loops into four types. In type 1, there is no instruction between the outer loop and inner loop, thus loop interchange technique can be applied to select better optimization. In type 2, some instructions exist in between the nested loops, which make optimization complicated. For type 3 nested loops, the inner loop is short; software pipelining can be applied to the inner loop first then to the outer loop. As to type 4, it is very difficult to optimize the outer loop, people can do the scheduling the outer loop including the software pipelined inner loop only. The type distribution of kernels is slightly different from that of major functions in the SMV program

Table 6: Categories of nested loops

Type	Statements between inner and outer loops	Length of inner loop	Major functions in SMV encoder	Kernels
1	0	/	19%	24%
2	$< 3$	/	24%	41%
3	$> 3$	$\ll$ outer loop	22%	24%
4	$> 3$	not $\ll$ outer loop	35%	12%

## 3. DSP operations in the innermost loops

The average DSP operations in the innermost loops can be used to estimate the resource requirements of a balanced DSP architecture [17]. Table 7 shows that the kernels have very closed numbers of DSP operations in relation to those of the major functions of the SMV Encoder.

Table 7: Average DSP Operations in Innermost Loops

	Major functions in SMV encoder	Kernels
ALU	2.1	2.8
Mult	0.8	1.2
Memory	3.1	3.7

## VI. Conclusion

The SMV benchmark consists of eight kernels which accounts for 58% of the execution time of the entire SMV Encoder. The loop behavior and average number of DSP operations of kernels represent nearly those of the entire SMV program. We believe that the SMV benchmark is suitable for both evaluating DSP

performance and improving DSP compilers and architecture.

Currently, we are using the SMV benchmark to test various kinds of DSP processors and their compilers, comparing their performance and analyzing their architecture and compilation techniques.

## VII. Acknowledgments

Su and Russo would like to thank the Center for Research, College of Science and Health, William Paterson University for research support in the summer of 2004. Su and Hu would like to thank the ART awards of William Paterson University.

## VIII. References

- [1] S. Ahmadi, Tutorial on the Variable-Rate Multimode Wideband Speech Codec, CommsDesign, Sept. 2, 2003.
- [2] The BDTImark2000™: A Summary Measure of DSP Speed, [www.bdti.com](http://www.bdti.com), Sept. 2004.
- [3] M. Chalamalasetti, Selectable Mode Vocoder (SMV), [www.bsnl.in](http://www.bsnl.in), Feb. 2003.
- [4] CDMA Enhancements Build on a Strong Foundation, [www.cdg.org](http://www.cdg.org), 2003.
- [5] CDMA Technology: Selectable Mode Vocoder. [http://www.cdg.org/technology/6\\_technology/vocoder/index.asp](http://www.cdg.org/technology/6_technology/vocoder/index.asp)
- [6] N. Dutt and K. Choi, Configurable Processors for Embedded Computing, *IEEE Computer*, Jan. 2003.
- [7] EEMBC Brings Embedded Benchmarking out of the Pits, 2000, [www.eembc.org](http://www.eembc.org)
- [8] J. Fisher, Moving from Embedded Systems to Embedded Computing, Keynote addressing, *CASES 03*, 2003.
- [9] M. Genutis, E. Kazanavièius, O. Olsen, Benchmarking in DSP, ISSN 1392-2114 ULTRAGARSAS, Nr.2(39). 2001.
- [10] M. Guthaus, etc., MiBench: A free, commercially representative embedded benchmark suite, *IEEE 4th Annual Workshop on Workload Characterization*, Austin, TX, December 2001.
- [11] D. E. Knuth, *The Art of Computer Programming: Fundamental Algorithms*, 2<sup>nd</sup> Edition, Addison-Wesley, 1973.
- [12] MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems, Proc. Of *MICRO-30*, 1997.
- [13] T. McCabe, A Complexity Measure, *IEEE Tran. On Software Engineering*, 2(4):308-320, 1976.
- [14] Software Engineering Institute, Cyclomatic Complexity, Software Technology Roadmap, Carnegie Mellon University, [http://www.sei.cmu.edu/str/descriptions/cyclomatic\\_body.html](http://www.sei.cmu.edu/str/descriptions/cyclomatic_body.html), 2005.
- [15] W. Strauss, Forward Concepts' *DSP Market Bulletin*, [www.fwdconcepts.com](http://www.fwdconcepts.com), Nov. 2003.
- [16] W. Strauss, Forward Concepts' *DSP Market Bulletin*, [www.fwdconcepts.com](http://www.fwdconcepts.com), Nov. 2005.
- [17] B. Su et al, Impact of Source-Level Loop Optimization for DSP Code Generation, Proc. of *the International Conference on Acoustics, Speech and Signal Processing (ICASSP'99)*, March 1999.
- [18] B. Su et al, Analysis of Loop Behavior of Selectable Mode Vocoder (SMV) and Its Impact of Instruction Level Parallelism, Proc. of *GSPx 2005*.
- [19] D. Talla et al, Evaluating Signal Processing and Multimedia Applications on SIMD, VLIW, and Superscalar Architectures., Proc. Of *ICCD'00*, 2000.
- [20] [www.3gpp2.org](http://www.3gpp2.org)

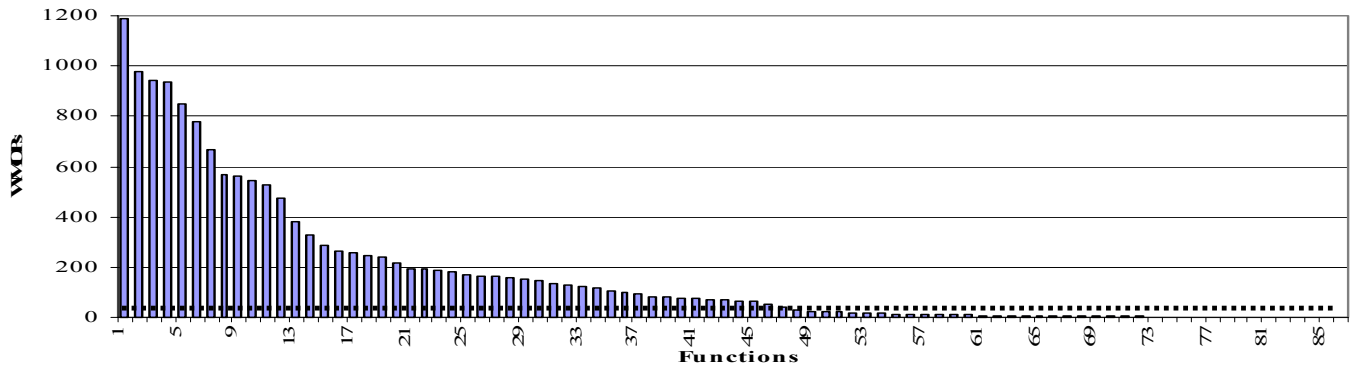


Figure 1: WMOPS Distribution of Functions Studied

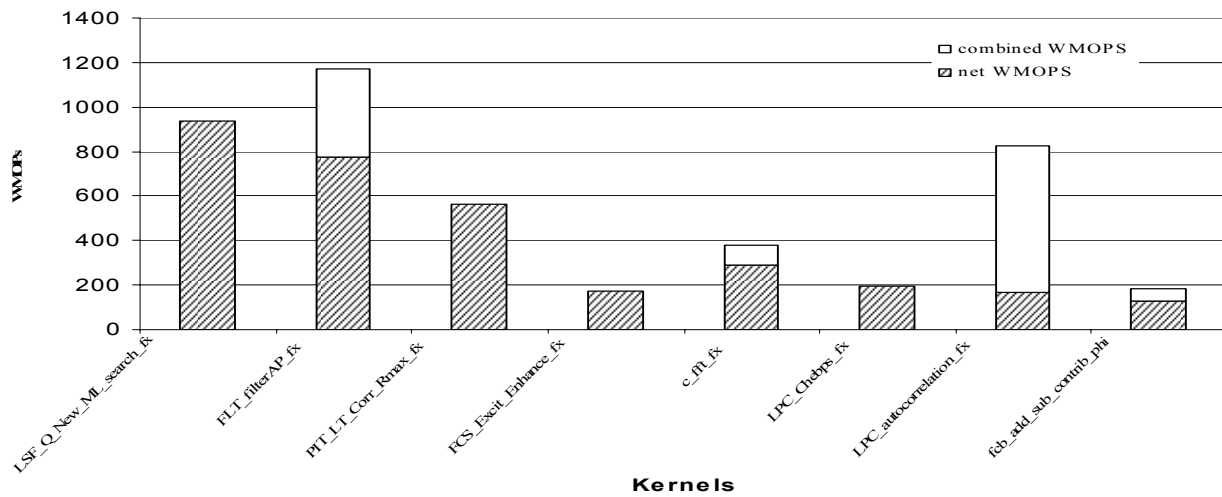


Figure 2: WMOPS of the Kernels

Table 1: Eight Kernels of SMV Benchmark

No.	Function Name	WMOPS	combined WMOPS	No. of lines	No. of loops	levels of nested loops	No. of DSP ops in inner most loops	CC*
1	LSF_Q_New_ML_search_fx	939	939	68	13	3	11	32
2	FLT_filterAP_fx	776	1173	11	2	2	5	3
3	PIT_LT_Corr_Rmax_fx	562	562	44	4	2	2	10
4	FCS_Excit_Enhance_fx	174	174	30	8	2	9	12
5	c_fft_fx	288	381	52	6	2	28	15
6	LPC_Chebbs_fx	196	196	16	1	1	6	1
7	LPC_autocorrelation_fx	166	826	10	3	2	5	4
8	fcb_add_sub_contrib_phi	129	183	21	1	1	9	6