Benchmarking FPGA High-Level Synthesis

Zhihua Li[†,‡], Colin Yu Lin[†], Juan Huang[†], Yuanqiang Li[†], Cheng Liu[¶],
Liqun Yang[†,‡], Hayden Kwok-Hay So[¶], and Haigang Yang[†,§]
† System on Programmable Chip Research Department,
Institute of Electronics, Chinese Academy of Sciences, Beijing, China
‡ The University of Chinese Academy of Sciences, Beijing, China
¶ Department of Electrical and Electronic Engineering, University of Hong Kong, Hong Kong
§ Corresponding Author: yanghg@mail.ie.ac.cn

***Abstract*—Due to the lack of benchmarks, currently, the FPGA High Level Synthesis(HLS) tool cannot be evaluated effectively. To solve this problem, this paper develops a suit of open-source benchmarks in C/C++ format for HLS. 20 benchmarks, which cover numerous fields including network, communication, multimedia and image processing, are carefully selected from the well-known SPEC CPU 2006, MiBench, MediaBench and LINPACK benchmarks. Since the current High Level Synthesis is not supportive of specific characteristics in C/C++, to apply CPU benchmarks to HLS, modifications and refactoring of code are carried on the 20 C/C++ benchmarks without changing the original functions. During the process, this paper proposes template metaprogramming to accommodate the recursion algorithm in HLS and an efficient memory allocation algorithm to support functions for dynamically allocating memory, such as malloc. After conversion, we implement the 20 benchmarks on Vivado, Xinlinx, and evaluate the resources cost by each benchmark. The experimental results show that the benchmark suit cost BRAM from 124KB to 321451KB, DSP from 1232 to 23455, FF from 4566 to 12256, and LUT from 2134 to 124324.**

I. Introduction

The increasing design complexity pushes the design community to seek high level design abstractions with higher productivity over the conventional register transfer level (RTL). High-level synthesis (HLS), which enables the automatic synthesis of high-level specifications to low-level cycle-accurate RTL specifications for efficient implementation on ASICs and particularly FPGAs, is one of the most promising solutions. There are already a great number of HLS tools so far from both the industry and academia as summarized in [1].

However, the HLS tools usually could only support a subset features of an high level language and the synthesizable subset also varies a lot. Therefore, a benchmark that could fit most of the HLS tools is still greatly needed. As HLS tools target random logic synthesis, the benchmark should cover a large number of representative domains of applications. For these purposes, we build a benchmark HLS-Bench which involves 20 applications selected from LINPACK, MediaBench, MiBench, and SPEC CPU 2006. It consists of various application domains including networking, communication, security, multimedia, image processing and so on. In addition, instead of using a static data set, we have three diverse data sets for each of the application, which helps to gain insight in the scalability of the HLS tools. To make sure the benchmark be easily supported by the general HLS tools, we further performed a series of code transformation like replacing dynamic link list data structure with static arrays, changing the recursive implementation with non-recursive one, developing dynamic memory manager to support some of the *malloc* and *free* and so on. This benchmark suite is available *HLS-Bench* under LGPL license.

Finally, we have the benchmark implemented using Xilinx Vivado HLS and evaluate the timing as well as the hardware overhead. The experiments show that the benchmark covers a distinct scale of implementations where BRAM, DSP48, FF and LUT consumption range from 124kb to 321451kb, 1232 to 23455, 4566 to 12256 and 2134 to 124324 respectively.

The contributions of this paper can be summarized as below:

- An open source benchmark suite in c/c++ for FPGA HLS is presented, which covers a wide range of application domains.
- Code transformations are studied and implemented to fit various HLS tools with limited syntax support.
- A state-of-art commercial FPGA HLS tool is evaluated using HLS-Bench.

In Section II, we review the evolution of both ASIC and FPGA HLS benchmarks. In Section III, we briefly describe features of each application in HLS-Bench. In Section IV, code transformation strategies that the ease the availability of HLS tools are discussed. In Section V, we present the implementation results of HLS-Bench using Vivado HLS. Finally, in Section VI, we conclude this paper.

II. Related Works

Some of the commencing efforts in HLS benchmarking were made in the late 1980s. High Level Synthesis Workshop 1992 Benchmarks [2] and 1995 High Level Synthesis Design Repository [3] were two standard benchmarks released for HLS. However, HLS tools in [2] [3] actually referred to compiling the HDL code to circuits and they were written in VHDL. These benchmarks were transplanted to C later, but they were tiny including less than one hundred lines of C code. Therefore, they are no longer suitable for benchmarking the latest HLS tools.

CHStone [4] is then proposed to provide a more practical benchmark suite for the HLS tools. It consists of 12 easy-to-use programs written in C and includes a number of application domains such as arithmetic, security, microprocessor, media processing and so on. Moreover, the programs in CHStone are

much larger compared with the HLS92 and HLS95. However, the application domain included is quite limited. Also construct data structure and dynamic memory allocation, which are removed for the HLS tools at that time, now turn to important supporting features of the latest HLS tools. In addition, just as the authors declared, CHStone doesn't intend to evaluate the commercial HLS tools.

S2CBench written in SystemC was proposed to enable the direct comparison of commercial HLS tools [5]. It consists of 12+1 programs which target a variety of applications. The 12 benchmarks comply with the latest SystemC synthesizable subset draft, while the rest one is FFT and it is non-synthesizable due to the trigonometric and floating point operations. The non-synthesizable FFT is added in order to help the users to understand how the different HLS tools work around these special occasions. Another unique feature of S2CBench is that each application is accompanied by its corresponding testbench. Nevertheless, the application domain covered in S2CBench is relatively limited. Typical applications such as Multimedia and network are not involved.

BDTI High-Level Synthesis Tool Certification Program **citation is needed** adopts a video motion analysis application and a wireless receiver to evaluate the high-level synthesis tools. The application is big enough for evaluation but the application domain is again quite limited.

HLS-Bench developed in this paper includes a large range of application domains. Also it have data structures and syntax supported in the latest HLS tools, which helps evaluate the HLS tools. Moreover, each application is provided with three diverse data sets and it is beneficial to gaining insight into the HLS tools.

III. HLS-Bench Suite

In this section, we present an overview of HLS-Bench, and then briefly introduce features of each application in the benchmark suite.

A. Overview

HLS-Bench is a collection of 20 C/C++ programs selected from well-known benchmarks such as SPEC CPU2006, MiBench, MediaBench and LINPACK as shown in I. It covers a wide range of application domains including network, communication, safety, multimedia, image processing and so on, which are suitable for implementing on FPGA.

On top of the broad application domains, programs involved in HLS-Bench are big enough to cover all the basic code structures like loop and condition. In addition, some of the syntax or features such as construct data structure and constant dynamic memory allocation are not well supported in earlier HLS tools, but they are gradually supported in the latest HLS tools. Therefore, they are also included in this benchmark. Finally, recursive algorithms in the benchmark that is still not supported in HLS tools are reconstructed as normal loops. II presented a detailed statistic of all these features mentioned above, which will hopefully give the users an insight into the benchmark.

TABLE I. FPGA HLS Benchmarks

Benchmark	Source	Domain
linpack	LINPACK Benchmark	Arithmetic
astar	SPEC CPU 2006	Network and Games
qsort	MiBench	Automotive and Industrial Control
stringsearch	MiBench	Office
fft	MiBench	Telecomm
crc32	MiBench	Telecomm
adpcm	MiBench	Telecomm
susan	MiBench	Multimedia and Consumer
ispell	MiBench	Office
Dijkstra	MiBench	Network
bitcount	MiBench	Automotive
basicmath	MiBench	Automotive and Arithmetic
rijndael	MiBench	Security
sha	MiBench	Security
blowfish	MiBench	Security
More...		

B. Benchmark programs

In this section, each application included in the benchmark is briefly introduced.

(FIXME change font)linpack: It is a linear equation solver of Ax = b, where A is a dense matrix and b is a vector. It is widely used in engineering.

astar: It is a computer algorithm that is widely used in pathfinding and graph traversal. It uses a best-first search and finds a least-cost path from a given initial node to target nodes.

qsort: qsort is a classical divide and conquer algorithm and is widely used for sorting [6].

stringsearch: **what is the name of the stringsearch algorithm, please specify it**.

fft: It is used to perform the discrete Fourier Transform and inverse. It can convert time to frequency and vice versa and is widely used in engineering.

crc32: It is used to perform a 32-bit Cyclic Redundancy Check (CRC) on a file(FIXME).

adpcm: Adaptive Differential Pulse Code Modulation (ADPCM) is a variation of the well-known standard Pulse Code Modulation (PCM). A typical implementation could convert 16-bit linear PCM samples to 4-bit samples, yielding a compression rate of 4:1.

susan: Susan is an image recognition package. It can be used to smooth an image and perform adjustments for threshold, brightness, and spatial control and is widely used in recognizing corners and edges in Magnetic Resonance Images of the brain. **I don't know this algorithm, but I think it is better to describe its main functionality first and typical application implemented in the benchmark later.**

ispell: Ispell is a fast spelling checker. It supports English word spell correction suggestions.

Dijkstra: Dijkstra's algorithm is a well known solution to the shortest path problem and can be completed in O(n^2) time.

basicmath: The basic math program includes a group of typical mathematical calculations such as cubic function solving, integer square root as well as degree and radian conversion.

TABLE II. FPGA HLS BENCHMARKS

Benchmark	Lines of code	Clauses of condition	Clauses of loop	Strcut types (C struct and C++ class)	Dynamically memory allocation	Change recursion to iteration
linpack	741	57	38	0	0	No
astar	550	41	11	0	0	No
qsort	75	1	0	2	1	Yes
stringsearch	407	25	27	0	0	No
fft	385	10	7	0	0	No
crc32	254	0	2	0	0	No
adpcm	351	22	3	1	0	No
susan	2281	162	60	3	5	No
ispell	925	36	34	4	1	No
Dijkstra	150	5	3	3	0	No
bitcount	885	21	18	0	0	Yes
basicmath	350	1	4	1	0	No
rijndael	1767	36	33	1	0	No
sha	350	6	14	1	0	No
blowfish	534	4	9	1	0	No
More	TOADD					

bitcount: The bitcount program is used to count the number of bits of an integer. Five different algorithms including an optimized 1-bit per loop counter, recursive bit count by nibbles, non-recursive bit count by nibbles using a look-up table, non-recursive bit count by bytes using a look-up table and shifting and accumulation **not sure about the last algorithms described here, please double check**.

rijndael encrypt/decrypt: Rijndael is a block cipher with the option of 128-bit, 192-bit and 256-bit keys and blocks. It is in the National Institute of Standards and Technologies Advanced Encryption Standard (AES).

sha: SHA is a secure hash algorithm and it will produce a 160-bit message digest for a given input. It is widely used for cryptographic key exchange, digital signature generation and well-known MD4 and MD5 hashing.

blowfish encrypt/decrypt: Blowfish is a symmetric block cipher with a variable length key. Its key length ranges from 32 to 448 bits and it is suitable for domestic and exportable encryption.

patricia: a patricia trie is an efficient data structure to represent full trees with sparse leaf nodes. It helps reduce the traversal time, and are typically used to implement routing tables in network applications.

h263enc/h263dec: H.263 is a video compression standard originally designed as a low-bitrate compressed format for video conferencing. h263enc and h263dec are used for encoding and decoding respectively.

mp3enc/mp3dec: They come from LAME benchmark in Mibench. LAME is a free software codec used to encode/compress audio into the lossy MP3 file format. Mp3enc encodes a raw (headerless) PCM stream or Waveform Audio File Format (commonly known as WAV due to its filename extension) to MP3. It supports constant, average and variable bit-rate encoding. Mp3dec decodes the input MP3 to WAV or raw PCM format.

IV. CODE TRANSFORMATIONS

As the benchmarks are selected from conventional CPU benchmarks, some of them can't be used for FPGA HLS directly. Therefore, refactoring and modification are crucial for implementing these algorithms using FPGA HLS tools. While these processing is actually what a software developer may confront using HLS tools, we detail the processing in this section. Hopefully, it may help the software developer to access the FPGA HLS tools.

A. Code Refactoring

As FPGA HLS tools couldn't support file reading and writing, they are removed from the original source code and replaced with **How did you replace the file reading and writing? What do you mean by data processing and controlling? On top of the file reading and writing, are there any other "data controlling"?**. Execution time extraction code which is useless for FPGA HLS is also removed accordingly.

B. Pointer

A pointer is an address to a location in memory and it can used as function parameters, array handling, pointer to pointer, and type casting. Despite of the convenience brought by the pointer, FPGA HLS tools could mainly support the pointers that are known at compilation time. While the pointers that are known at runtime are just partially supported at the moment. Figure 1 and Figure 2 present two typical code styles that are un-synthesizable and synthesizable respectively.

```
int *pointer_runtime = malloc(sizeof(int) * 100);
```

Fig. 1. Unsupported Dynamic Memory Allocation in HLS

```
int data[100];
int *pointer_to_data = data;
```

Fig. 2. Supported Pointer in HLS: Managing Array Access with a Pointery

Although dynamic pointers are not well supported inner the function to be synthesized, they can actually be perfectly synthesized at the interface. The pointers at the interface can either be synthesized as a memory-mapped IO or stream IO. For memory mapped IO, it is required to specify the memory size at compilation time. Fortunately, you can specify it using the compilation pragma or directives instead of source code directly. If it is a streamed IO, hand shaking protocol will be synthesized and you don't have to specify the size of the

```
int accumulate1(int *data,int length) {
    int sum = 0;
    for (int i = 0; i < length; ++i) {
        sum += data[i];
    }
    return sum;
}
```

Fig. 3. Unsynthesizable code: accumulate an array

```
#define LENGTH 100
int accumulate2(int data[LENGTH]) {
    int sum = 0;
    for (int i = 0; i < LENGTH; ++i) {
        sum += data[i];
    }
    return sum;
}
```

Fig. 4. Synthesizable code: accumulate an array.

input. Figure 3 and Figure 2 are examples of dynamic pointer synthesis at the interface.

As explained above, the dynamic pointers inner the function to be synthesized in the benchmark source code are replaced with static allocation. The ones at the interface will be left unchanged.

C. Library Functions

There are many C string and memory library functions such as memset, memcpy, strlen and strcmp in the original benchmark. Although they can actually be implemented on FPGA, they are not supported in FPGA HLS tools at the moment. To bridge this gap, we develop a synthesizable library for the benchmarks so that the original can be updated with minimal efforts. Figure 5 shows a simple example on how the synthesizable library be used.

```
#define MAX_CHAR_SIZE 1024

unsigned hls_strlen(char string[MAX_CHAR_SIZE]) {
    unsigned length = 0;
    for (length = 0; string[length] != 0; ++length);
    return length;
}
```

Fig. 5. Rewrited HLS strlen Function.

D. Dynamic Memory Allocation

Dynamic memory allocation is one of the typical memory management techniques in C and C++ programming languages. It can be used to allocate or release resource as needed at runtime. Essentially, this feature helps to share the limited physical memory. However, when the dynamic memory behaviour is compiled from high level language program to FPGA using HLS tools, it is actually different, though the synthesized hardware may still maintain the same functionality of the original program. The main reason is that the declared memory is implemented in a static way on FPGA and can't be released until the FPGA is reconfigured.

```
int array[100];
```

Fig. 6. HLS-Compliant Automatic Memory Allocation

```
static int array[100];
```

Fig. 7. HLS-Compliant Static Memory Allocation

To keep the memory behaviour the same with that described in high level language program, we develop a memory allocation algorithm that could support dynamic memory allocation and sharing as software. The dynamic memory allocation algorithm is detailed in Algorithm 10. First of all, we declare a static array which can be synthesized as a shared static memory using FPGA HLS tools. Then the shared memory is divided into multiple chunks logically and the availability of each chunk of the memory is recorded in a bitmap as shown in Figure 9. Whenever there is a memory allocation request, corresponding amount of memory is allocated in chunks and the bitmap is updated as well. When the allocated memory is released, the bits in bitmap are reset accordingly. **As the mmeory allocation and release are usually sequential, the memory manger is actually simple and will not cost much hardware. In addition, bitmap is small even when the chunk size is tiny, thus the overhead can be negligible.** While the main challenge is the size of the pre-allocated shared memory, because it must be big enough to meet all the memory allocations request involved. And the exact number depends on the target benchmark.

E. Data Structure and Algorithm Modifications

Dynamic data structures such as list constructed with link list are not supported by the HLS tools, and they are re-implemented using static data structures like array instead. Note that the array can be allocated dynamically taking advantage of the proposed dynamic memory allocation algorithm as discussed in section IV-D.

F. Recursive Algorithm

Recusion provides an effective way to implement many recursive algorithms. However, its dynamic characteristic makes

```
int *array = malloc(sizeof(int) * allocated_size);
```

Fig. 8. Dynamic Memory Allocation

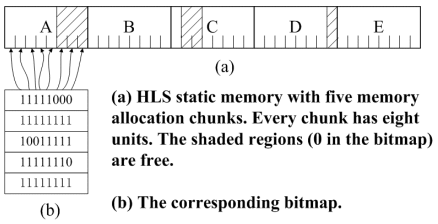

Fig. 9. TODO FIXME (a) HLS static memory with five memory allocation chunks. Every chunk has eight units. The shaded regions (0 in the bitmap) are free. (b) The corresponding bitmap.

```
Prerequisite:
M, pre-allocated static memory
B, bitmap for allocation chunks, all bits initialized with 0
K, user requested K units memory at runtime
S, memorizing allocated memory, used for deallocate

Seek for K consecutive 0 bits in the bitmap B.
Let the found location be F.
Set K consecutive 0 bits to 1 at the beginning F in bitmap B.
Return the corresponding location in M to the location F of bitmap B.
Record F, K and the above returned result in S.
```

Fig. 10. (TODO, FIXME) Dynamic Memory Allocation Algorithm in HLS

it tricky for hardware design let alone HLS tools. Therefore, all the recursive implementation is replaced with a plain loop instead. Figure 11 and 12 give an example how a recursive quicksort algorthm is implemented using plain loop.

```
quicksort(array, low, high):
  if low < high:
    p := partition(array, low, high)
    quicksort(array, low, p - 1)
    quicksort(array, p + 1, high)
```

Fig. 11. Pseudocode of Recursive quick sort Function.

```
iterative-quicksort(array, low, high):
  create an auxiliary stack SA to store low, high
  push initial values of low and high to SA
  while SA is not empty:
    pop high and low from stack SA
    p := partition(array, low, high)
    if (p - 1) > low:
      push low and (p - 1) to stack
    if (p + 1) < high:
      push (p + 1) an high to stack
```

Fig. 12. Pseudocode of iterative quck sort Function.

V. Experiments

In this section, we have HLS-Bench synthesized using the latest commercial HLS tools and then the hardware overhead of the benchmark is analyzed.

In the experiments, Xilinx Vivado HLS [7] (2013.4 Version) is used. xc7v2000tflg1925-1, which is large enough as shown in III to accommodate any of the application in the benchmark, is chosen as the target device. Timing constrain is set to be 100MHz and default synthesis options are used directly. **I guess it will be better to have some basic optimization strategy such as interface setup? loop unrolling strategy? pipelined or not?. But it seems that we don't have time to do that...**

TABLE III. Available resources of xc7v2000tflg1925-1

BRAM_18K	DSP48E	Flip-Flop	LUT
2584	2160	2443200	1221600

Typically each application in the benchmark is provided with three different data sets for more extensive exploration of the HLS tools. While bitcount, basicmath and blowfish do not depend on any specific parameters, and they have only a single implementation instance in the experiments.

The experiment results are displayed in Table IV. It is clear that the DSP48 consumption doesn't have much difference among diverse data sets. The main reason is that there is no loop unrolling and the loop bodies implemented are about the same. Applications with large arrays tend to consume more BRAM blocks as the arrays are typically implemented as memory.

TABLE V. HLS sythesis result for cosine, sine and exponent, logarithm function

	Utilization Estimates			
C math function	BRAM_18K	DSP48E	FF	LUT
cos	19	17	2574	8896
sin	19	17	2573	8821
exp	0	26	1124	2666
log	0	61	1755	1464

basicmath costs the largest amount of FF and LUT due to the exponent and logarithm operations that need a lot of FF and LUT as shown in Table V. It also consumes moderate number of BRAM blocks because of the cosine and sine function implemented with lookup table.

VI. Conclusions

This paper present an open-source C/C++ benchmark suite selected form well-known CPU benchmarks for FPGA high-level synthesis tools. It covers a number of typical application domains. We further provide each benchmark with diverse data sets to facilitate the exploration of HLS tools. The benchmark is implemented using the latest Xilinx Vivado HLS. The experiments show that the benchmark consumes a large range of hardware overhead and computation, which makes it representative for HLS benchmarking.

Acknowledgment

This research is funded by National Natural Science Foundation of China (61271149, 61106033).

References

[1] G. Martin and G. Smith, "High-level synthesis: Past, present, and future," *IEEE Design & Test of Computers*, vol. 26, no. 4, pp. 18–25, 2009.

[2] N. D. Dutt and C. Ramachandran, *Benchmarks for the 1992 high level synthesis workshop*. Information and Computer Science, University of California, Irvine, 1992.

[3] P. R. Panda and N. D. Dutt, "1995 high level synthesis design repository," in *Proceedings of the 8th international symposium on System synthesis*. ACM, 1995, pp. 170–174.

[4] Y. Hara, H. Tomiyama, S. Honda, H. Takada, and K. Ishii, "CHstone: A benchmark program suite for practical C-based high-level synthesis," in *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*. IEEE, 2008, pp. 1192–1195.

[5] B. Carrion Schafer and A. Mahapatra, "S2CBench: Synthesizable SystemC benchmark suite for high-level synthesis."

[6] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*. IEEE, 2001, pp. 3–14.

[7] T. Feist, "Vivado design suite," *Xilinx, White Paper Version*, vol. 1, 2012.

TABLE IV. SYTHESIS RESULT FOR FPGA HLS BENCHMARK PROGRAMS

Benchmark Feature	Input Parameters	Timing Estimates Clock Period (ns)	Utilization Estimates BRAM_18K	 DSP48E	 FF	 LUT
linpack Solve ax=b with matrix dimension size N of b	N=50	8.7	0	114	11307	13738
	N=100	8.7	0	114	11522	14018
	N=1000	8.7	0	114	12184	14824
Dijkstra Calculates the shortest path between every pair of nodes in a graph represented by adjacency matrix with size N×N	N=10	7.98	451	0	402	975
	N=100	7.98	482	1	478	1080
	N=1000	7.98	2436	1	551	1190
qsort Sorts an array with size N	N=1000	7.92	35	0	1861	2367
	N=5000	7.92	35	0	1910	2404
	N=10000	7.92	35	0	1929	2417
stringsearch Searches for given words with total length N	N=1k	7.12	2	0	644	1135
	N=4k	7.12	3	0	652	1147
	N=16k	7.12	9	0	660	1159
fft Performs a Fast Fourier Transform on an array of data with size N	N=1M	8.64	19	73	10057	17936
	N=4M	8.64	19	73	10103	17983
	N=16M	8.64	19	73	10149	18028
crc32 Performs a 32-bit Cyclic Redundancy Check on an array of data with size N	N=1M	5.62	1	0	104	191
	N=4M	5.62	1	0	108	197
	N=16M	5.62	1	0	112	204
adpcm Takes 16-bit linear PCM samples and converts them to 4-bit samples. The input samples are an array of data with size N	N=1M	8.47	4	0	809	2056
	N=4M	8.47	4	0	811	2062
	N=16M	8.47	4	0	813	2068
astar Finds a least-cost path from a given initial node to one goal node in the map with width and height N	N=20	8.53	15	28	3549	9432
	N=60	8.53	78	28	3620	9513
	N=100	8.53	295	30	3754	9516
susan Smooth an image and has adjustments for threshold, brightness, and spatial control. The image data is in an array with size N.	N=1k	8.44	300	98	14764	28029
	N=4k	8.44	300	98	14934	28427
	N=16k	8.44	300	98	15104	28825
ispell Spelling checker for English words in a word array with size N.	N=100	7.92	110	10	4598	8038
	N=1000	7.92	110	10	4610	8054
	N=10000	7.92	110	10	4635	8108
rijndael Uses AES to encrypt input stream in a word array with size N	N=1k	8.34	40	2	4434	13808
	N=16k	8.34	40	2	4462	13849
	N=256k	8.34	40	2	4490	13891
sha A hash algorithm that produces a 160-bit message digest for a given array with size N	N=16k	7.19	6	0	2815	5272
	N=128k	7.19	6	0	2824	5281
	N=1M	7.19	6	0	2833	5290
blowfish (a symmetric block cipher with a variable length key)	-	5.67	3	0	879	1526
basicmath (performs simple mathematical calculations)	-	8.64	38	199	24440	43978
bitcount (counts the number of bits in an array of integers)	-	8	7	0	468	1191
patricia (searchs an IP in routing tables represented by Patricia trie)	-	7.25	68	0	220	430
h263enc (encodes a frame of H.263 video)	-	8.7	1231	135	16483	35098
h263dec (decodes a frame of H.263 video)	-	8.7	259	314	71890	162082
mp3enc (encodes a frame of MP3 audio)	-	8.71	197	718	105880	160235
mp3dec (decodes a frame of MP3 audio)	-	8.70	194	681	85291	140049