Performance considerations for real time FFT intensive DSP systems and benchmarks vs. the PowerFFT

Peter Beukelman, Richard Middendorf, Eric Verhulst
Eonic BV http://www.eonic.com/

Abstract

When solving real life DSP problems in high data rate environments like radar, sonar and 2D image processing, developers are often left disappointed with the final performance of their systems. Initial system parameters are derived from a benchmark performance of a 1024 point FFT on the latest DSP, but end with a performance that is only a fraction of the benchmark. This is often due to a narrow focus on peak GFlops and little attention on I/O, memory bandwidth and memory addressing inefficiencies. The impressive benchmark results provided by the vendors can only be achieved with in place operations, i.e. with all data remaining in the processor cache and never leaving the processor. For applications with large (outside cache) data sets, access to main memory is needed on top of I/O operations. This causes the peak I/O performance to drop far below the theoretical peak value. Another common operation that further reduces effective memory bandwidth is reading/writing from/to a square data set. The stride addressing results in many cache misses and extra cycles for reading from external SDRAM. The PowerFFT chip is designed to overcome these problems. This programmable DSP has separate busses for input and output and has four additional memory busses with specific addressing schemes targeted on achieving high throughput on square datasets.

Sustained data rate processing vs. FFT benchmarks

In many real world applications data is coming in from a sensor and has to be continuously processed at a high rate. For processors with a single bus (worst case), this means following steps :

- incoming data has to be placed in RAM (best with DMA),
- transferred from RAM to the processor,
- processed in the processor,
- transferred back to RAM
- exported from RAM (best with DMA).

When the input data rate matches the memory bandwidth, the input DMA channel will monopolize access to the memory practically preventing the processor from doing much processing. Benchmarks explaining how fast a processor can process the data often don't take into account that new data has to enter the main memory and results have to leave it. The most optimistic benchmarks test a 1024 point FFT just from L1 cache.

To increase the processing performance in systems where data is coming in and going out at a steady rate, the PowerFFT has separate busses on the processor for input and output. To keep the I/O running concurrently with the processor, the PowerFFT has 4 memory banks attached to it (see Fig 1). Note however that this external memory is only needed when the FFT has more than 1024 points. These banks allow concurrent memory access while processing the data, e.g. :

- data from the input can flow into the first memory bank
- the output port is being supplied with results of a second memory bank
- the processor reads from the third memory bank
- the processor writes to the fourth memory bank.

The processor core also has separate input and output busses (2 inputs and 1 output bus), allowing uninterrupted stream processing on large datasets that can only fit in external memory (e.g. SDRAM). An internal switched fabric connects the I/O busses and the processing core input and output busses with each other or with the memory banks in an on-the-fly programmable way. It makes the memory banks interchangeable, so when one input bank gets full, an empty one can be put in its place.

The architecture mentioned above is a clear example how fully distributed memory banks, combined with programmable point to point connections easily can overcome the bottleneck of shared memory. This problem has gotten increasingly worse with the increasing bandwidth gap between the CPU clock and memory, notwithstanding clever architectural tricks like pipelining, access protocols, caches (but these are a serious problem for predictable real-time behavior), larger datapaths and multiple ALU units to somehow hide the effects

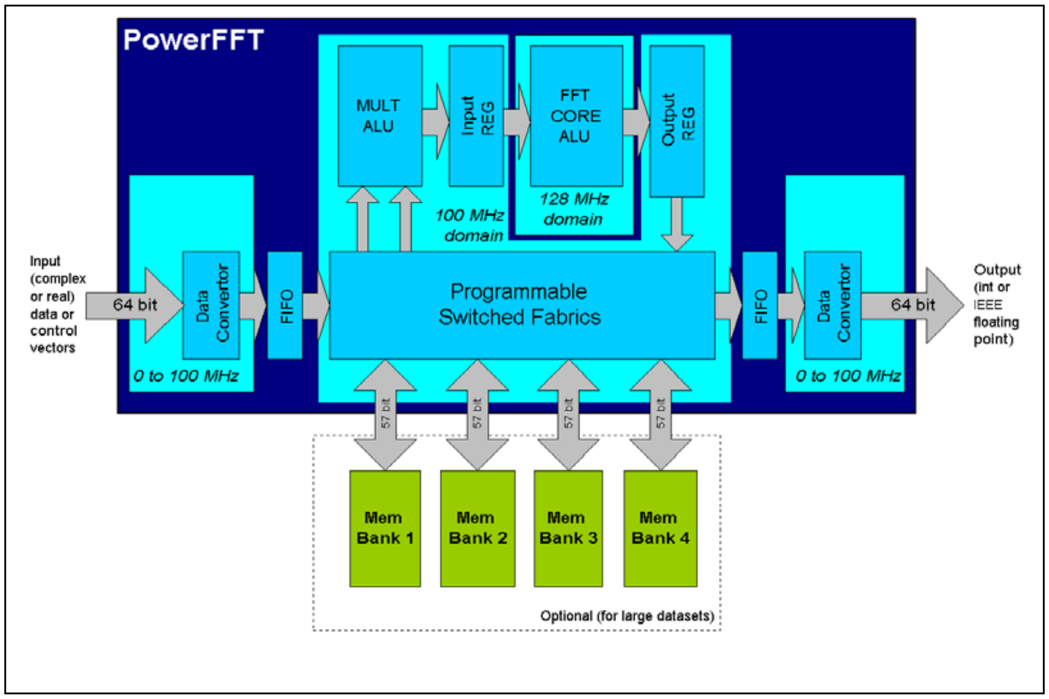


Figure 1 . PowerFFT block diagram.

Memory is not randomly accessible in a uniform way.

To make processing possible at a sustained rate, also the memory has to support it. Unlike the name suggests Random Access Memory does not really give random access at its clock frequency, unless one can use static RAM and accept the penalty of high cost and lower density. Internally all dynamic RAM types are built up in rows, columns and banks. Only after opening a row and waiting for a few cycles, are the contents of this row available for random access. Most processors therefore have a cache between the core CPU and main memory, which gives fast and random access to the most frequently used data, and fetches cache misses from a DRAM. A cache however does not give very predictable timings, especially when the program exhibits dynamic behavior and only works well when the data and program can fit in it. When large square datasets need to be handled (that don't fit in cache), they will be stored in main memory. Accessing the data row-wise might give high efficiency, since the addressing coincides with rows in DRAM. A cache miss results in a new

row being fetched with all new samples being useable samples and available in cache. When the data is approached column wise, stride addressing occurs. This has the effect of either a cache miss every sample, followed by a row fetch of unusable data (worst case) or (best case) by a single sample fetch at the expense of closing and opening a new row in DRAM for every sample. In the PowerFFT a special addressing scheme is used to make use of almost the full bandwidth of the memory, in both horizontal and vertical access to the data. This method (patent pending) uses a combination of bank switching, burst addressing, interleaving, address translation and command hiding to achieve a high throughput in both horizontal and vertical addressing modes of SDRAM memory.
A benefit of this addressing scheme is that for 2D operations, the so-called corner turn memory operation is essentially for free. On traditional DSPs, and this remains even valid for FPGA implementations, corner turn operations on large datasets are rather costly as it requires a copy of the dataset or inefficient indexed addressing scheme. With the PowerFFT, the data can be left in place.
Now all 3 components (separated I/O and memory busses, multiple memory banks, efficient 2 way memory addressing) are ready to provide sustained data FFT processing. For an example such an implementation, see e.g. the PowerFFT-PCI64 board (datasheet available on http://www.eonic.com/).

Benchmarks

To realistically benchmark the processor, 1D CFFTs of different lengths were used for one-dimensional data, and 2D CFFTs for two-dimensional data. All tests are in floating point format. To get the benchmark independent of the bandwidth limitations of a specific board I/O interface like PCI, the processing time measured includes I/O times to the processor, but excludes the time to move data over a PCI or other bus. PCI bandwidth is often the limiting factor and the result would be a benchmark of the PCI bus rather than of the processor.

The PowerFFT (clocked at 100 MHz) is compared with :

- A G4 processor clocked at 450 MHz on an Eonic Atlas3-3U-G4 board with 2MB L2 cache and 100 MHz 128 MB SDRAM, CAS3, 64 bit. As memory controller the Tundra chipset is used.
- An ADSP-TS101 TigerSHARC clocked at 250 MHz on an Analog Devices Evaluation board with 83.3 MHz 32 MB SDRAM, CAS 2, 64 bit
- An ADSP-21160 Hammerhead SHARC clocked at 72 MHz on an Atlas2-HS V1.1 board with 36 MHz, 0 WS, 1 MBytes SRAM, 64bit

In order to be realistic, the benchmarks were done under similar conditions as experienced by most engineers : all code used was publicly available, most often selected as the best one can obtain from the semiconductor manufacturer and tested on real hardware. If needed some C routines were written and compiled with all optimization switches on. The code will is available for review from Eonic's website (www.eonic.com). Readers are invited to submit any better code or remarks ('Lies, lies and benchmarks' according to a famous paper in the transputer days, we did our best).

For the G4 board the Motorola library functions were used and the GNU C cross-compiler version 2.95.2. The best code we could find was tested and the compiler used the Altivec switch. For the TigerSHARC both VisualDSP++ routines as well ADI library functions were used (optimized assembly routines). For the 2D FFT only VDSP++ library functions were used. For the G4 both in cache operations and out of cache operations were measured. The full SDRAM version calculates the FFT from SDRAM and puts the results back in SDRAM, hereby ignoring the penalties of I/O

operations that would result in further performance degradation. The TigerSHARC can use its links to input and output data, but no hardware was available to connect I/O to the links in our test, hence input and output data is stored in external RAM. This gives a realistic performance, since I/O from/to memory can be compared to I/O through the links. In practice this will only be possible for small datasets, since large datasets cannot reside in the processor's internal memory and have to come/go from/to main memory anyway.
Note that these test clearly indicate the importance of the system level architecture. E.g. in the case of the TS101, its performance is better than the G4, even at approximately half the clock frequency for as long as the benchmark can be run from internal memory. For larger sizes however, the G4 can benefit from a L2 cache but when the data really gets large, the external memory kicks in. This conclusion is confirmed by comparing the results with those of a slower 21160 board but with SRAM. The use of fast external SRAM results in less penalties so that the results remain proportionally better.

Normalizing the benchmarks

To show the efficiency of a processor, one does not only have to measure the absolute timings, but one has also to relate it to the clock frequency and the maximum possible performance.
Therefore, all benchmark results are normalized to a virtual 1.8 GFlops processor that works at 100% efficiency. If the G4-Altivec would be used at full efficiency of 4*450 MFlops, it would have 1.8 GFlops available and its benchmark would read 1 in the graph. To calculate how long a virtual 1.8 GFlops processor would take for a 1D CFFT, the number of required floating point operations is divided by 1.8 GFlops to get the number of seconds. In case of a radix-2 FFT the complexity is n/2 * ^{2}log (n) butterfly operations with 10 floating point operations per butterfly The resulting number is used to normalize all benchmark times. If the software routines are highly optimized, an efficiency higher than one can eventually be achieved, because using mixed radix and higher radix transformations reduces the Flop count moderately for an FFT.
The times for the PowerFFT assume sustained processing and don't take initializations and pipeline delays into account. The PowerFFT shows a drop in efficiency when it goes from 1024 points to 2048 points, because the CFFT has to be done in 2 passes on shorter lengths that have a lower efficiency. The PowerFFT was designed for an optimal processing of 1K CFFT.

Impact of windowing

In most applications, the FFT operation (or one of its derived algorithms like correlation, convolution) is just one of the processing steps. Often it will be combined with some sort of filtering or windowing operation in sustained mode. The PowerFFT has a built-in Multiplier Unit in the data pipeline that can be programmed to provide this function with no overhead. The impact of adding the windowing function however for general purpose DSPs is significant. On the PowerFFT boards, most often one will also find large FPGAs, allowing to add post and pre-processing algorithms in the datastream.

Conclusion

From the timing tables and graphs it can be concluded that the dataflow based design of the PowerFFT outperforms more traditional DSPs like TigerSHARC, G4-Altivec and 21160 processors on every FFT length. This is primarily due to the inherent dataflow nature of DSP algorithms whereby data moving and data processing steps do overlap. On the other hand traditional DSPs are basically still improved sequential von Neumann machines. More importantly it shows that the efficiency of most general purpose processors or DSPs drops significantly outside the benchmark domain of in place, in cache/memory short FFTs. When main memory must be accessed by these processors, direct SDRAM addressing reduces the performance of the processor, although a cache can hide partly the performance penalty.

I/O operations are not even taken into account in this number, as input data is assumed to be in SDRAM and results are supposed to remain in SDRAM. In a system with an I/O stream and a G4 processor, I/O has to share bandwidth on the processor bus, reducing the effective memory bandwidth of the processor. As we noticed that the Tundra chipset is not always providing the best bandwidth, this can further complicate the benchmarks.

For the TigerSHARC this is less of an issue, since it can receive new data and send results through its links. Still the 1024x1024 2D FFT is effectively transformed by the TigerSHARC at only 225 MFlop (2.4 GFlop on chip), most likely caused by SDRAM addressing. Degradations caused by caching, inefficient addressing and sharing of the processor bus by data and I/O, have to be taken into consideration when building a real time system.

Last but not least, the overall result is that DSP systems using the programmable PowerFFT can be significantly smaller and will consume substantially less power than those based on traditional general purpose DSPs. The PowerFFT typically uses 1 to 2 W. Hence for FFT intensive applications, the use of the PowerFFT is beneficial in all aspects. As most DSP systems also need other types of algorithms, and taking into account the relatively better results obtained with the small size, fairly low power TS101, a combination of the two will provide a very powerful system.

The graphs

Legend description

PowerFFT@100 :	PowerFFT running at I/O speed of 100 M complex samples/sec
TS101@250_ASM_IRAM	TigerSHARC 250MHz, ADI C internal memory
TS101@250_IRAM_asm_no_opt	TigerSHARC 250MHz, ADI C non optimal internal memory
TS101@250_ASM_inp_IRAM	TigerSHARC 250MHz, ADI C input buffer in SDRAM
TS101@250_all_SDRAM	TigerSHARC 250MHz, ADI C input, output and twiddles in SDRAM
TS101@250_ADI_IRAM	TigerSHARC 250MHz, ASM assembler routines in internal memory (data sheet value)
TS101@250_ADI_IRAM_no_opt	TigerSHARC 250MHz, ASM different internal memory configuration
TS101@250_ADI_IRAM_SDRAM	TigerSHARC 250MHz, ASM input buffer in SDRAM
TS101@250_ADI_all SDRAM	TigerSHARC 250MHz, ASM input, output and twiddles in SDRAM
21160@72_opt	Hammerhead 72 MHz, optimized assembly code
21160@72_Virtuoso	Hammerhead 72 MHz, Benchmark run under Virtuoso RTOS, kernel interferes FFT-computation. Data in internal memory.
G4@450	G4 450 MHz, first iteration of a loop
G4@450_cache	G4 450 MHz, subsequent iterations of a loop

Figure 2 Relative processor speed for 1D CFFT (relative to virtual 1.8 Gflop 100% efficient processor)

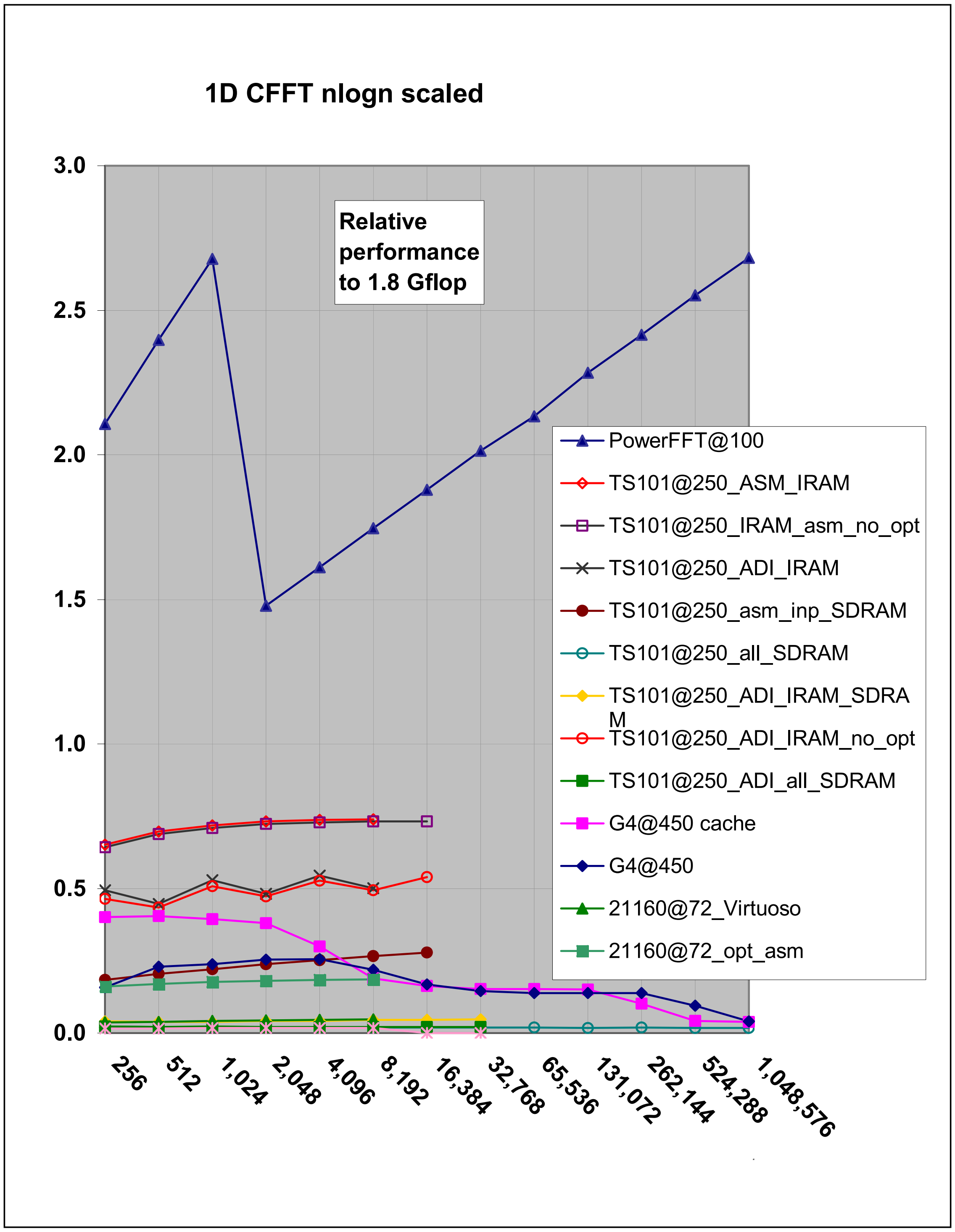


Figure 3 Relative speed of processor for 2D CFFT (relative to virtual 1.8 Gflop 100% efficient processor)

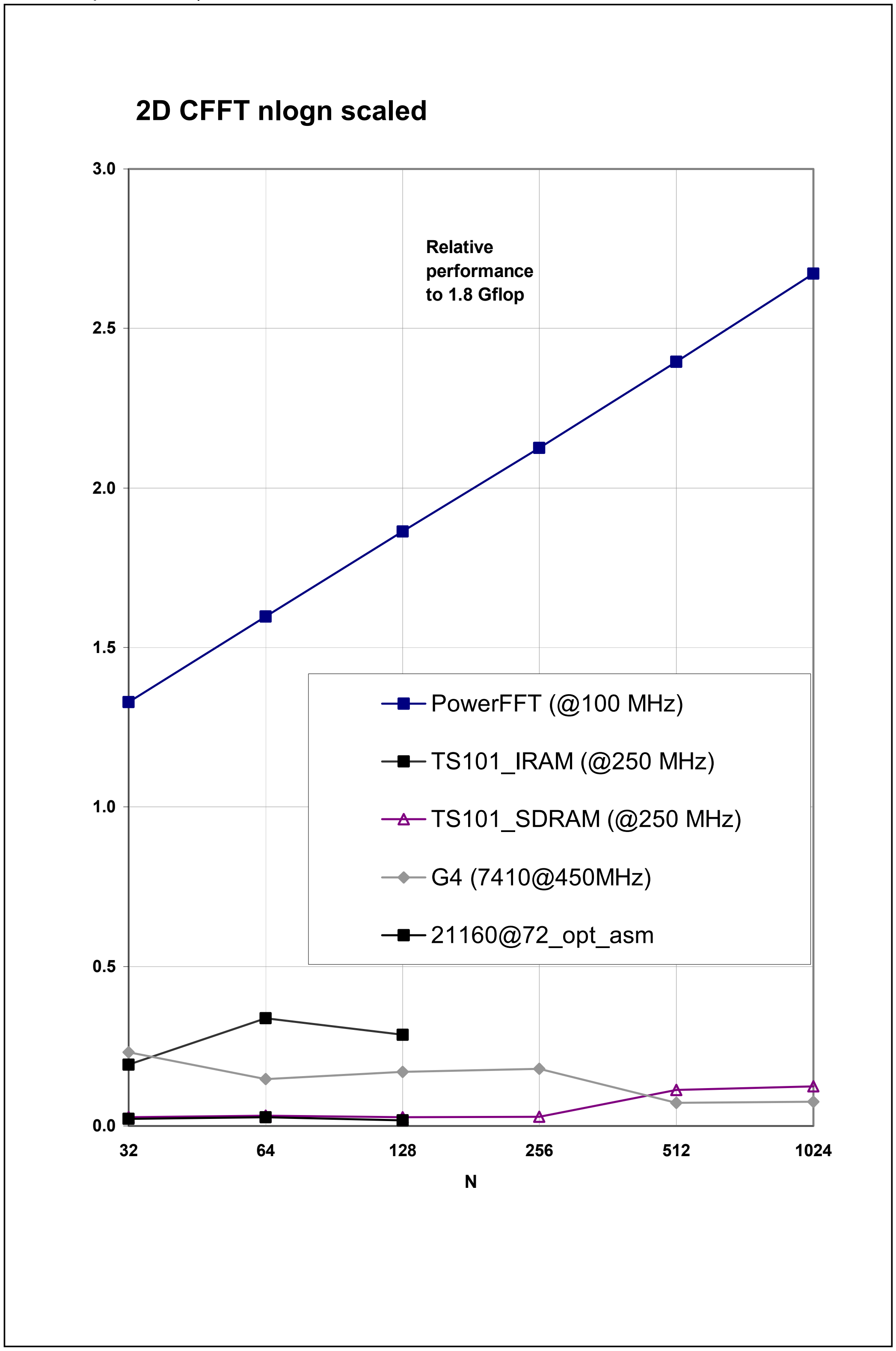


Figure 4 Absolute speed for 1D FFT up to 2K – 1M points (microseconds)

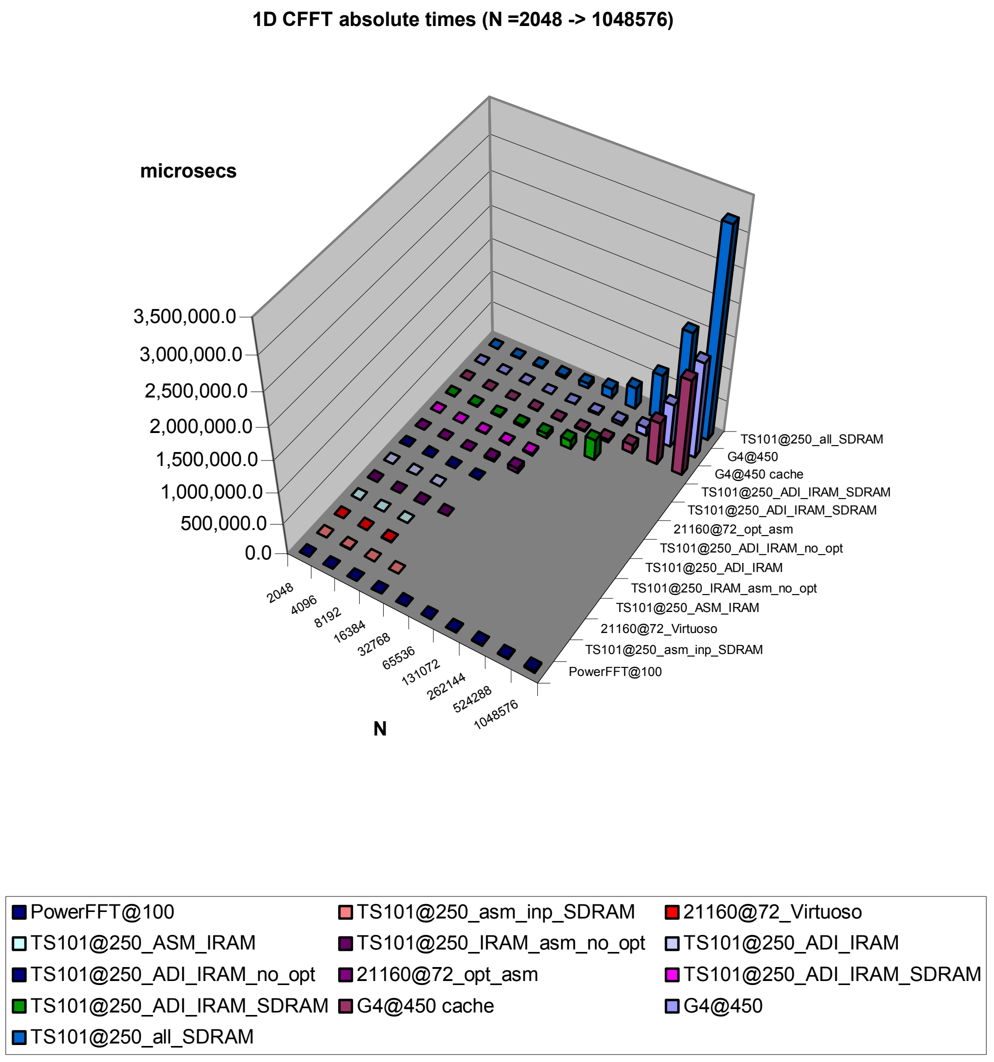


Figure 5 Absolute speed for 1D FFT up to 256 –8K points (microseconds)

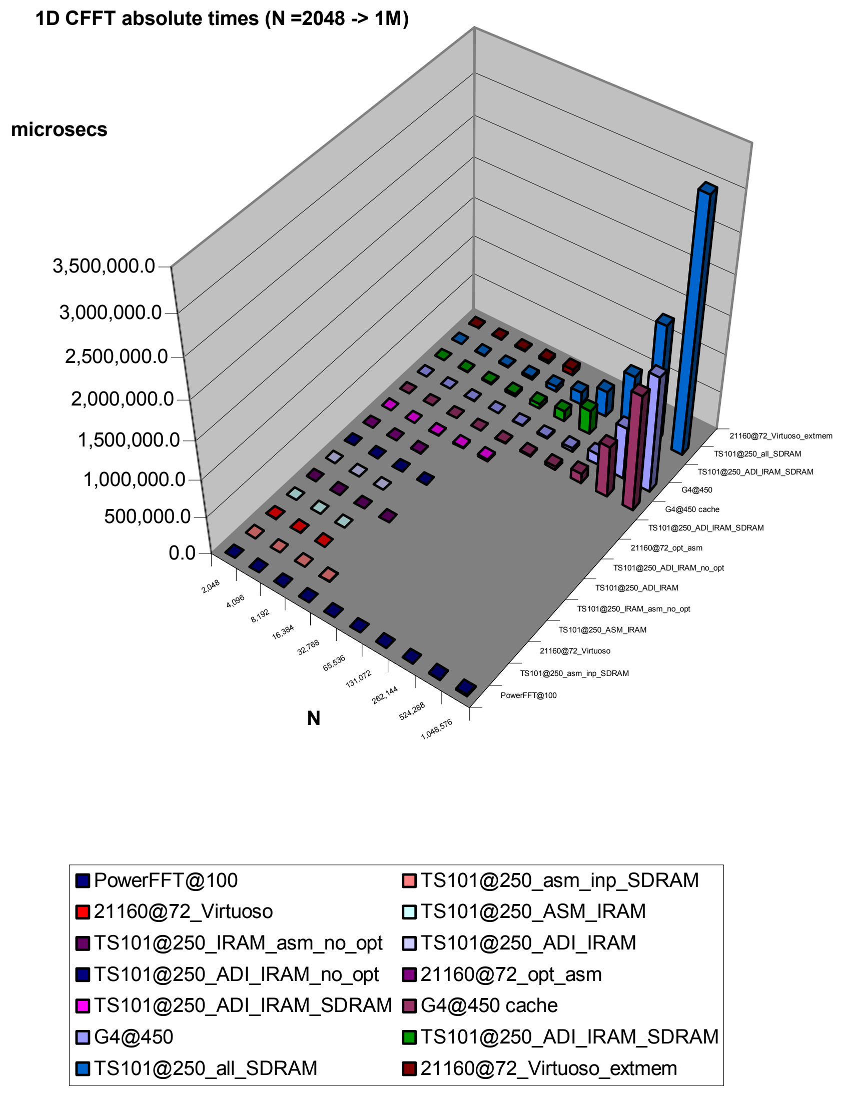


Figure 6 Absolute speed for 2D FFT (microseconds)

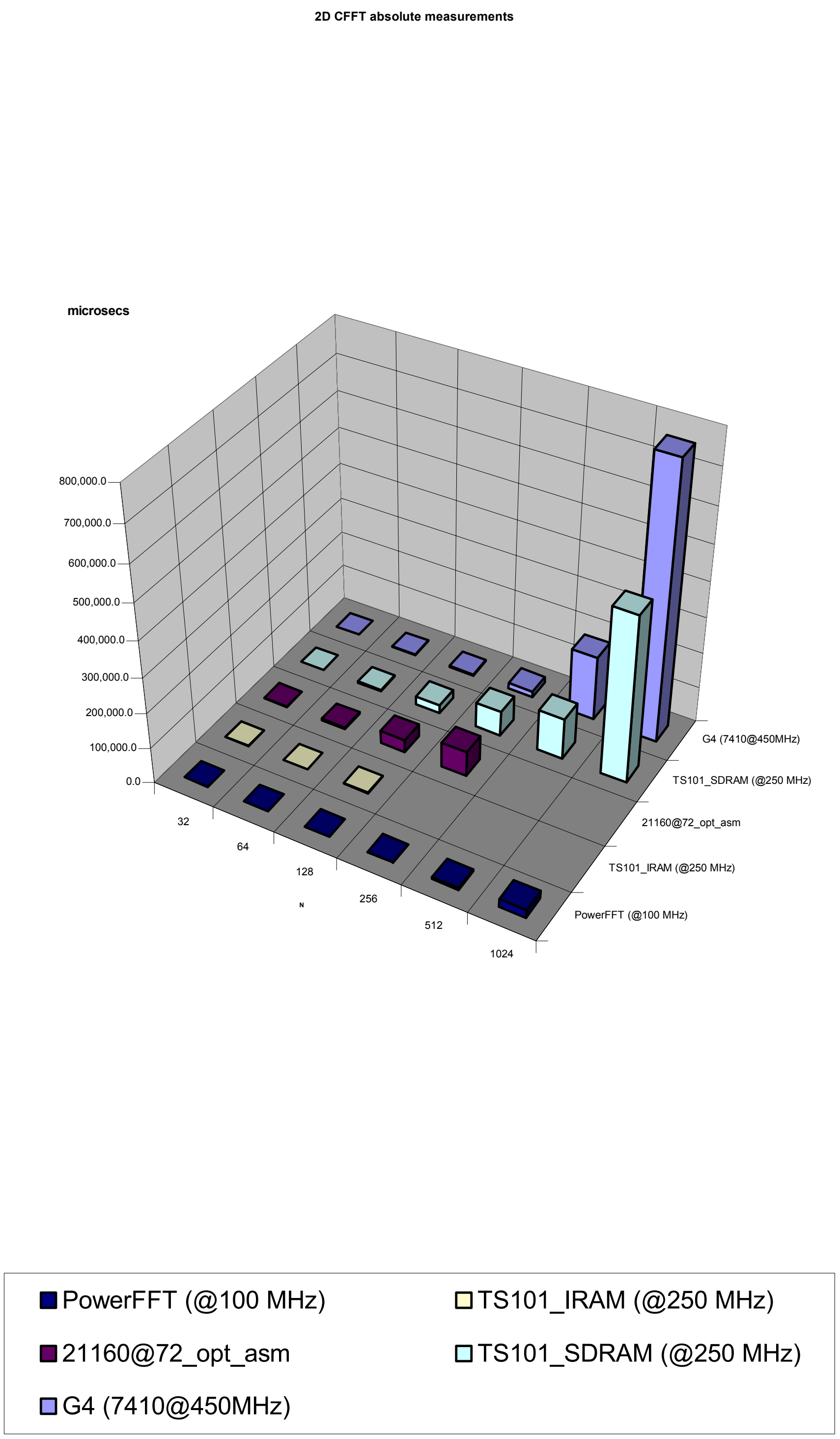


Figure 7 Absolute speed for 1D FFT with window up to 256-8192 points (microseconds)

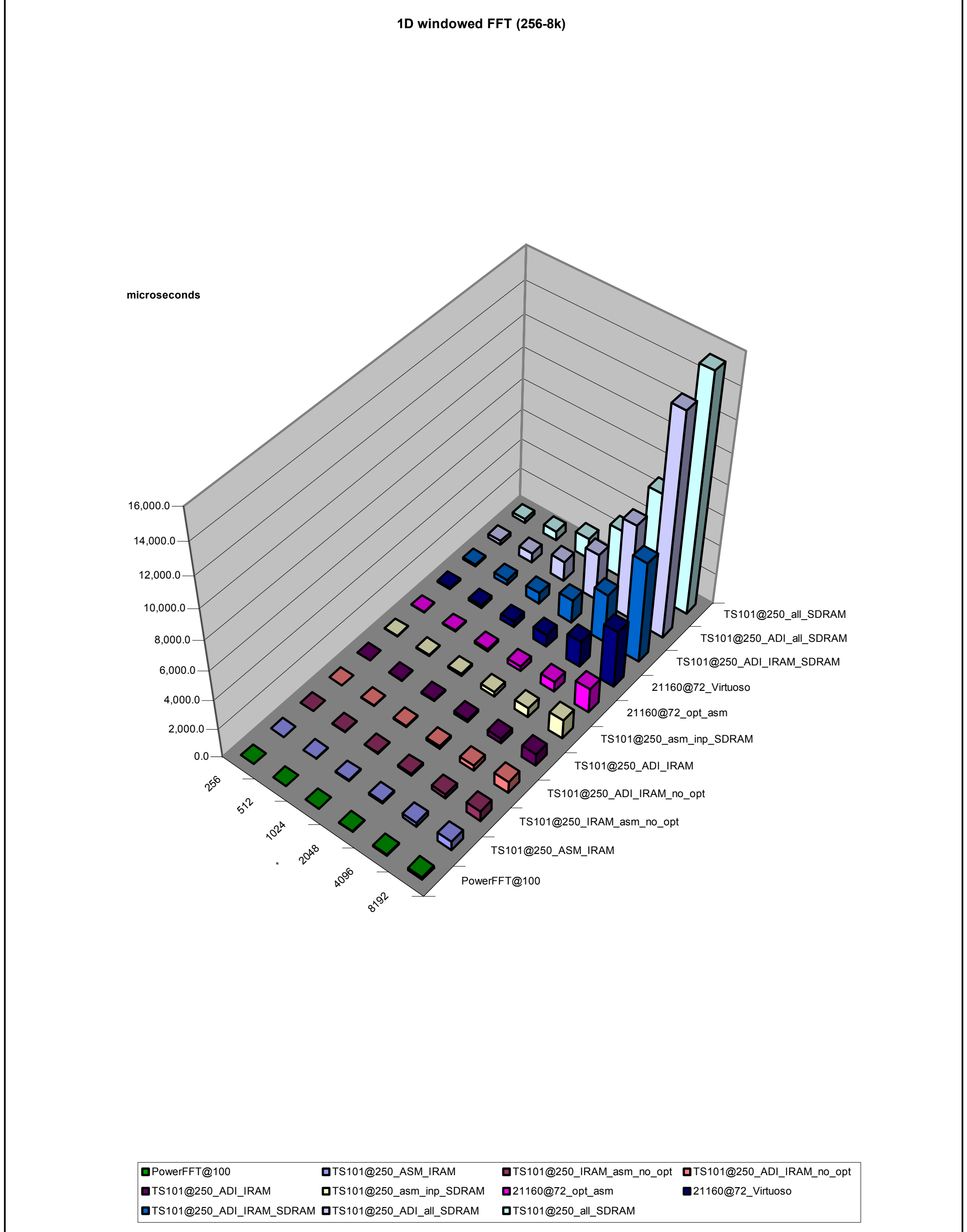
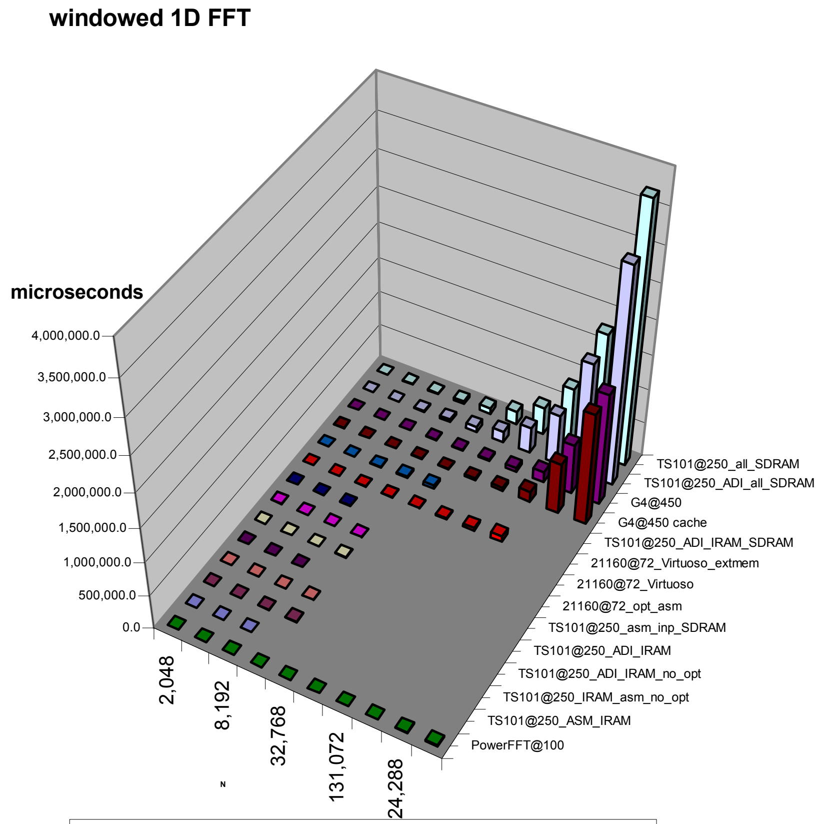
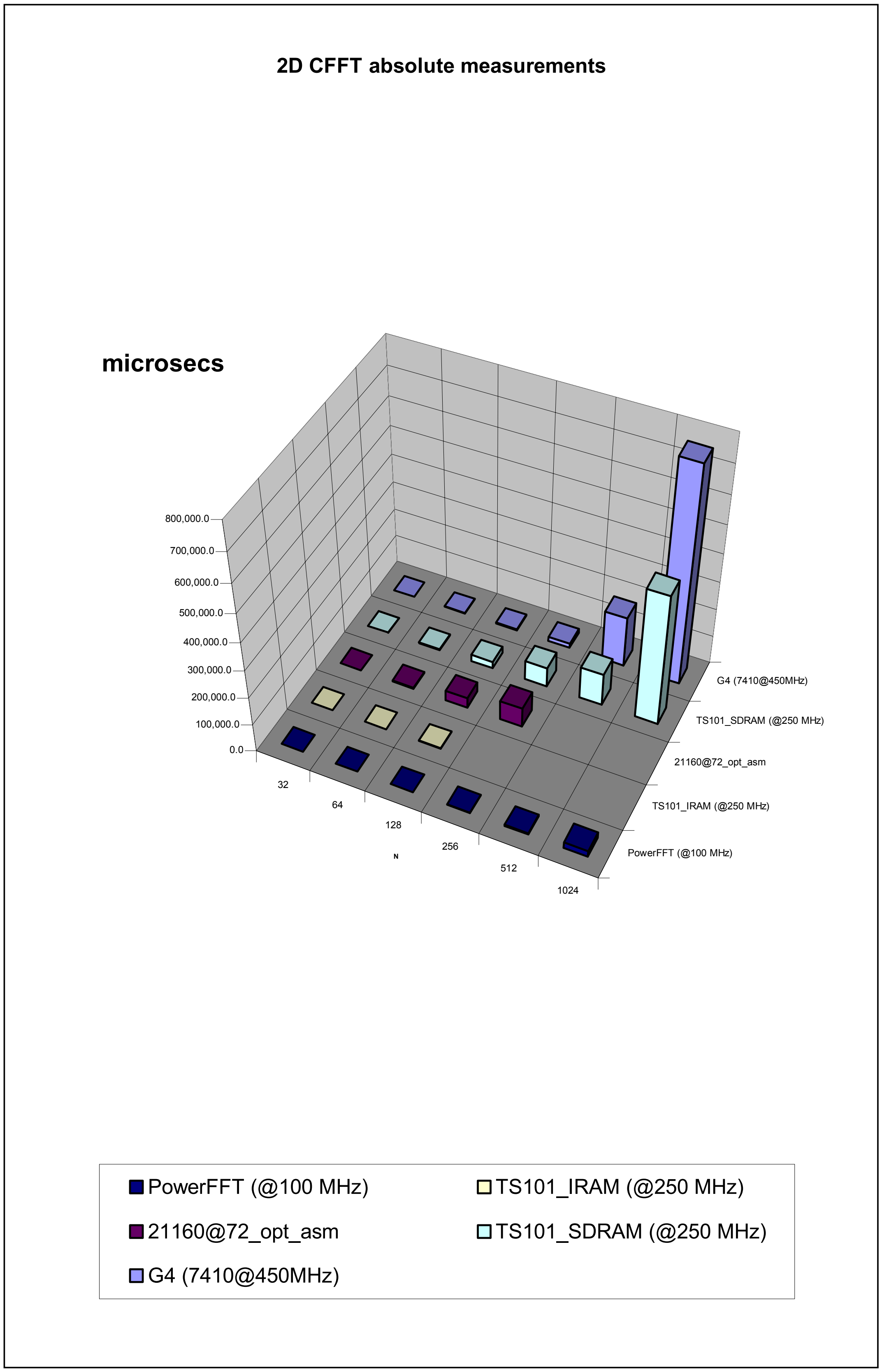


Figure 8 Absolute speed for 1D FFT with window up to 2048-1M points (microseconds)



🟩 PowerFFT@100
🟦 TS101@250_ASM_IRAM
🟪 TS101@250_IRAM_asm_no_opt
🟥 TS101@250_ADI_IRAM_no_opt
🟪 TS101@250_ADI_IRAM
⬜ TS101@250_asm_inp_SDRAM
🟪 21160@72_opt_asm
🟦 21160@72_Virtuoso
🟥 21160@72_Virtuoso_extmem
🟦 TS101@250_ADI_IRAM_SDRAM
🟥 G4@450 cache
🟪 G4@450
⬜ TS101@250_ADI_all_SDRAM
⬜ TS101@250_all_SDRAM

Figure 9 Absolute speed for 2D FFT with window (microseconds)



TABLES

Legend processor tested	
G4@450	A
G4@450 cache	B
PowerFFT@100	C
TS101@250_ASM_IRAM	D
TS101@250_IRAM_asm_no_opt	E
TS101@250_asm_inp_SDRAM	F
TS101@250_all_SDRAM	G
TS101@250_ADI_IRAM	H
TS101@250_ADI_IRAM_no_opt	I
TS101@250_ADI_IRAM_SDRAM	J
TS101@250_ADI_all_SDRAM	K
21160@72_Virtuoso	L
21160@72_opt	M

N	Gflops relative to virtual 1.8 Gflops processor for 1D CFFT												
	A	B	C	D	E	F	G	H	I	J	K	L	M
256	0.16	0.40	2.11	0.65	0.64	0.18	0.02	0.49	0.46	0.04	0.02	0.04	0.16
512	0.23	0.40	2.40	0.70	0.69	0.20	0.02	0.45	0.43	0.04	0.02	0.04	0.17
1,024	0.24	0.39	2.68	0.72	0.71	0.22	0.02	0.53	0.51	0.04	0.02	0.04	0.18
2,048	0.25	0.38	1.48	0.73	0.72	0.24	0.02	0.48	0.47	0.04	0.02	0.04	0.18
4,096	0.26	0.30	1.61	0.74	0.73	0.25	0.02	0.55	0.53	0.04	0.02	0.05	0.18
8,192	0.22	0.19	1.75	0.74	0.73	0.27	0.02	0.50	0.49	0.04	0.02	0.05	0.18
16,384	0.17	0.16	1.88		0.73	0.28	0.02		0.54	0.05	0.02		0.19
32,768	0.15	0.15	2.01				0.02			0.05	0.02		
65,536	0.14	0.15	2.13				0.02				0.02		
131,072	0.14	0.15	2.28				0.02				0.02		
262,144	0.14	0.10	2.42				0.02				0.02		
524,288	0.10	0.04	2.55				0.02				0.02		
1,048,576	0.04	0.04	2.68				0.02				0.02		

N	Gflops relative to virtual 1.8 Gflops processor for 2D CFFT				
	A	C	H	H	M
32	0.23	1.33	0.19	0.03	0.02
64	0.15	1.60	0.34	0.03	0.03
128	0.17	1.86	0.29	0.03	0.02
256	0.18	2.13		0.03	
512	0.07	2.40		0.11	
1024	0.08	2.67		0.12	

N	1D CFFT Absolute timings in microseconds												
	A	B	C	D	E	F	G	H	I	J	K	L	M
256	24.7	14.2	2.7	8.7	8.9	30.9	247.9	11.5	12.3	143.6	242.5	155.8	35.5
512	53.7	31.7	5.3	18.4	18.6	62.7	594.1	28.7	29.4	336.0	621.3	326.6	75.0
1,024	112.2	72.1	10.6	39.6	40.1	128.4	1,328.3	53.8	56.1	725.3	1,285.3	685.9	160.7
2,048	244.3	164.3	42.4	85.5	86.5	263.1	3,072.3	130.2	132.4	1,501.0	3,041.3	1,439.9	346.0
4,096	625.0	456.3	84.8	185.2	187.3	540.5	6,798.9	250.5	258.9	3,184.9	6,255.3	3,018.9	744.6
8,192	1,764.0	1,558.0	169.4	399.8	403.9	1,110.4	15,290.3	591.0	599.5	6,579.9	14,391.7	6,318.8	1598.4
16,384	4,363.0	3,920.0	339.0		869.6	2,282.7	32,914.5		1,180.5	13,862.9	29,469.9		
32,768	9,816.0	8,921.0	677.7				72,709.0			28,580.1	65,249.7		
65,536	21,000.0	19,218.0	1,365.4				154,838.0				133,480.0		
131,072	44,798.0	41,191.0	2,711.0				337,362.0				315,323.0		
262,144	137,370.0	128,530.0	5,426.0				712,561.0				595,132.0		
524,288	675,700.0	667,200.0	10,844.0				1,536,076.0				1,383,380.0		
1,048,576	1,507,330.0	1,507,970.0	21,725.0				3,223,886.0				2,625,006.0		

N	1D CFFT with windowing absolute timings in microseconds												
	A	B	C	D	E	F	G	H	I	J	K	L	M
256	36.0	18.8	2.7	14.9	15.1	37.1	254.1	17.7	18.5	149.8	248.7	177.6	57.3
512	70.3	40.8	5.3	30.7	31.0	75.1	606.5	41.0	41.8	348.4	633.6	369.8	118.2
1,024	143.6	90.3	10.6	64.2	64.8	153.1	1,353.0	78.5	80.7	749.9	1,309.9	771.8	246.5
2,048	304.3	222.0	42.4	134.7	135.7	312.3	3,121.5	179.4	181.6	1,550.2	3,090.5	1,611.1	517.2
4,096	748.5	589.7	84.8	283.6	285.6	638.8	6,897.3	348.8	357.3	3,283.3	6,353.7	3,360.8	1,086.5
8,192	2,117.0	1,812.0	169.4	596.5	700.6	1,307.1	15,487.0	787.7	796.1	6,776.6	14,588.3	7,229.6	2,509.2
16,384	5,089.0	4,420.0	339.0		8,638.6	10,051.7	40,683.5		8,949.5	21,631.9	37,238.9		5,462.9
32,768	11,357.0	10,071.0	677.7				88,247.0			44,118.1	80,787.7		
65,536	24,454.0	22,022.0	1,365.4				185,914.7				164,556.7		
131,072	52,672.0	47,993.0	2,711.0				399,515.7				377,476.7		
262,144	154,162.0	151,886.0	5,426.0				836,868.5				719,439.5		
524,288	724,656.0	735,929.0	10,844.0				1,784,691.4				1,631,995.4		
1,048,576	1,622,529.	1,622,440.	21,725.0				3,721,117.0				3,122,237.0		

	0	0											

N	2D CFFT Absolute timings				
	G4 (7410@450MHz)	**PowerFFT (@100 MHz)**	**TS101_IRAM (@250 MHz)**	**TS101_SDRAM (@250 MHz)**	**21160@72_opt**
32	123.0	21.4	148.5	1,042.0	1,257.9
64	929.0	85.5	403.7	4,258.0	5,070.1
128	3,750.0	342.0	2,230.8	23,708.0	36,191.0
256	16,230.0	1,370.0		69,706.3	69,707.3
512	180,482.0	5,470.0		115,704.7	
1024	768,044.0	21,800.0		466,845.7	

N	2D CFFT with windowing absolute timings				
	G4 (7410@450MHz)	**PowerFFT (@100 MHz)**	**TS101_IRAM (@250 MHz)**	**TS101_SDRAM (@250 MHz)**	**21160@72_opt**
32	229.0	21.4	550.5	1,444.1	1,352.6
64	1,040.0	85.5	1,998.7	5,853.1	5,429.7
128	4,405.0	342.0	8,593.4	30,060.7	42,133.3
256	18,876.0	1,370.0		95,190.3	
512	175,722.0	5,470.0		217,761.7	
1024	804,936.0	21,800.0		875,192.7	