FAST TEMPLATE MATCHING FOR UMTS CODE KEYS WITH GRAPHICS HARDWARE

Mario Vigliar, Giancarlo Raiconi
DMI, Università degli Studi di Salerno, via Ponte don Melillo, 84083, Fisciano SA, Italy

Alan David Sanders
Wider Networks, 881 Ponce de Leon Ave Suite 10, 30306 Atlanta GA, U.S.A.

Keywords: Measurements on 3G networks, GPU computation, FFT.

Abstract: Template matching is a milestone application in Digital Signal Processing, and sets its roots in fundamental numeric filtering theory, as well as in time and frequency domain analysis. Radio signals, with mass spreading of high bandwidth cellular networks, have become in recent years much more critical to handle in terms of *QoS (Quality of Service)*, *QoE (Quality of Experience)* and *SLA (Service Level Agreement)*, putting mobile carriers in the need to monitor their network status in a more detailed and efficient way than in past. Here an efficient use case of GPU computing applied to fast signal processing will be illustrated, with particular interest in study and development of a SIMD Linear and FFT-based cross correlation of multiple code keys in air-captured streams for UMTS networks. Developed techniques have been used with success in a commercial available 3G geotagged scanning equipment.

1 INTRODUCTION

The aim of this paper is to present a parallel computing approach to solve the problem of precisely measure the maximal exposure to UMTS (*Universal Mobile Telecommunications System*) signals in an area. Because the particular nature of this signal characterized by a high crest factor such task cannot be simply carried out by a field intensity meter but the only means consists in really receive the signal by an appropriate receiver and evaluate the signal quality.

Traditionally to this purpose, which is a computationally intensive task, the receiver was mounted on a vehicle, the received signal was recorded and used to signal exposition evaluation carried out offline. Such approach result on a set of costly operations in terms of time spent and specialized components used. Our approach consist of direct computation of necessary data directly onboard using a standard laptop, costly signal processing operation are performed efficiently exploiting the power of a standard GPU of the laptop using NVidia's CUDA framework.

The paper is organized as follows. In the next section the problem is stated by as template matching problem, solvable using classical Digital Signal Processing (DSP) technique in order to compute the cross correlation between sequences. the third section is devoted to describe the computational technique implemented in the Wider Network's scanner based on a specialized FFT algorithm running on GPU. In the following section the computational aspects of the algorithm are studied in detail and results of benchmark are reported.

Finally there is a concluding section.

2 PROBLEM STATEMENT

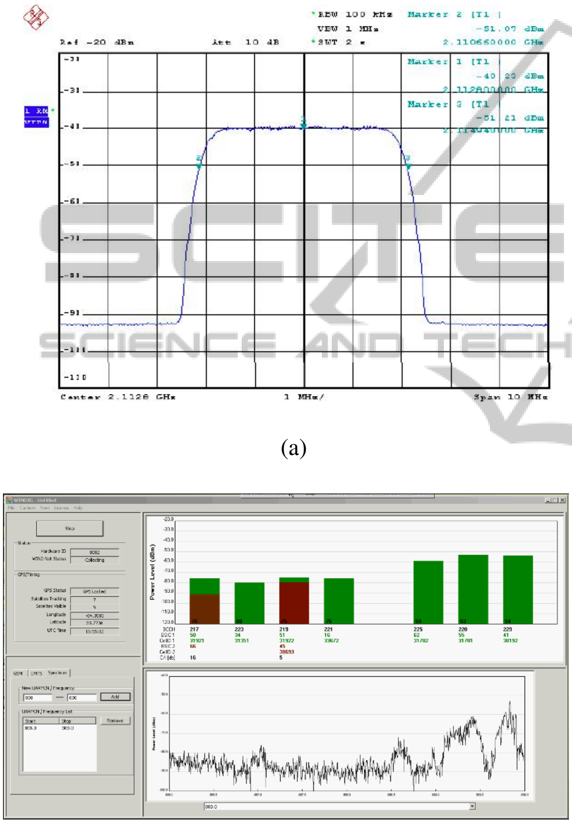
In this section we recall briefly relevant features of the UMTS signal and describe what computational tasks need to perform in order to assess the quality of the signal itself.

2.1 A Brief Sketch of the UMTS Signal

To understand the algorithm which is used to evaluate UMTS signal quality the knowledge of some details of its characterization are necessary. UMTS uses a multiple access technique, where several users as well as the signalization is separated by different spreading

Vigliar M., Raiconi G. and David Sanders A..
FAST TEMPLATE MATCHING FOR UMTS CODE KEYS WITH GRAPHICS HARDWARE.
DOI: 10.5220/0003362902650272
In *Proceedings of the 1st International Conference on Pervasive and Embedded Computing and Communication Systems* (PECCS-2011), pages 265-272
ISBN: 978-989-8425-48-5

codes (WCDMA - Wideband Code Division Multiple Access). By multiplication with the spreading code, the data signal is spread in the spectrum. Fig. 1a taken from (Bornkessel and Wuschek, 2006) shows a real measured UMTS signal in frequency domain. The 10 dB bandwidth is about 4.3 MHz. UMTS signals are *noise like* showing a large difference between short time maxima U_{Peak} and the RMS value U_{RMS} (*crest factor* $C = 20 \cdot \log(U_{Peak}/U_{RMS})$ in the 8 to 10 dB range, variable with the traffic of the station).



(a)

(b)

Figure 1: UMTS signal spectrum and its analysis with WIND3G.

In order to determine the maximal exposure in a defined volume, the *sweeping method* is recommended, due to signal nature. *Common Pilot Channel (CPICH)* is a fixed rate (30 kbps, *Spreading Factor* - *SF*=256) downlink physical channel that carries a pre-defined bit sequence. Figure 2 shows the frame structure of the *CPICH*. The Primary *CPICH* is used by the *UEs (user equipments)* to first complete identification of the Primary Scrambling Code used for scrambling Primary Common Control Physical Channel (P-CCPCH) transmissions from the *Node B* (in UMTS is the *Base Transceiver Station*). Later CPICH channels provide allow phase and power estimations to be made, as well as aiding discovery of other radio paths. There is one primary CPICH (P-CPICH), which is transmitted using spreading code 0 with a spreading factor of 256, notationally written as $C_{ch,256,0}$. Optionally a Node B may broadcast one or more secondary common pilot channels (S-CPICH), which use 256 codes arbitrarily chosen, written as $C_{ch,256,n}$ where $0 < n < 256$. The CPICH contains 20 bits of data, which are either all zeros, or in the case that Space-Time Transmit Diversity (STTD) is employed, is a pattern of alternating 1's and 0's for transmissions on the Node B's second antenna. The first antenna of a base station always transmits all zeros for CPICH. See below for P-CPICH and S-CPICH characterization.

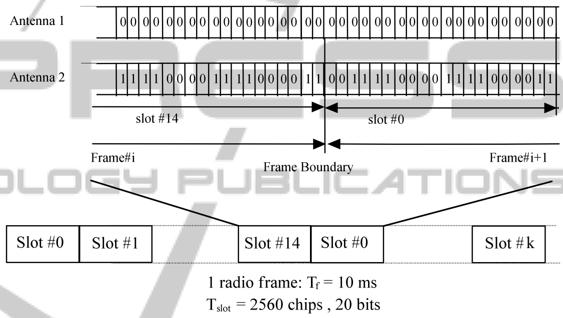


Figure 2: Frame structure and modulation pattern for Common Pilot Channel.

In case transmit diversity is used on P-CCPCH and SCH, the CPICH shall be transmitted from both antennas using the same channelization and scrambling code. In this case, the pre-defined bit sequence of the CPICH is different for Antenna 1 and Antenna 2, see figure 2 (top section). In case of no transmit diversity, only the bit sequence of Antenna 1 is used.

The channelization codes are the same codes as used in the up-link, namely Orthogonal Variable Spreading Factor (OVSF) codes that preserve the orthogonality between downlink channels of different rates and spreading factors. The channelization code for the Primary CPICH is fixed to $C_{ch,256,0}$ as previously stated, and the channelization code for the Primary CCPCH is fixed to $C_{ch,256,1}$. In each cell the UE may be configured simultaneously with at most two scrambling codes. The scrambling code sequences are constructed by combining two real sequences into a complex sequence. Each of the two real sequences are constructed as the position wise modulo 2 sum of 38400 chip segments of two binary m-sequences generated by means of two generator polynomials of degree 18. The resulting sequences thus constitute segments of a set of Gold sequences.

The scrambling codes are repeated for every 10 ms radio frame. Let x and y be the two sequences respectively. The x sequence is constructed using the primitive (over $GF(2)$) polynomial $1+X^7+X^{18}$. The y sequence is constructed using the polynomial $1+X^5+X^7+X^{10}+X^{18}$.

2.2 Monitoring and Mapping UMTS Signal Quality

Our main interest is in "measurement for performances" and so our main aim will be to decode air-captured data to evaluate if it is possible to decode some subsequences in useful cell broadcasting packets. As signal matching measure was used *cross correlation:*

$$(f \star g)[n] \stackrel{\mathrm{def}}{=} \sum_{m=-\infty}^{\infty} f^*[m]\, g[n+m]. \qquad (1)$$

for a fixed n, in the interval $[0...n]$ it is possible to compute the single terms of sum sequence just iterating the product between f^* and g terms. For each iteration, a single correlation value will be obtained, and finding a local maxima will indicate the best match position in the search reference of the candidate sequence. This computing method is often referred as *linear cross-correlation* index, and used in online systems when seeking for short length templates in streaming references, due to overall small memory footprint and good independency from floating point precision on the target architecture. Linear cross-corr. in fact, relies only on sum and multiply operations, thus ensuring very good computing performances on DSP-like systems, where *MADD (Multiply and ADDition)* primitives are present in hardware, and often within vectorial structures.

Let $\mathbf{x},\mathbf{y}$ two sequences and $\mathbf{X},\mathbf{Y}$ their Fourier transforms it can be shown that:

$$\mathcal{F}^{-1}\{\mathbf{X}^* \cdot \mathbf{Y}\}_n = \sum_{l=0}^{N-1} x_l^* \cdot (y_N)_{n+l} \stackrel{\mathrm{def}}{=} (\mathbf{x} \star \mathbf{y_N})_n\,, \quad (2)$$

which is the cross correlation of the $\mathbf{x}$ sequence with a periodically extended $\mathbf{y}$ sequence defined by:

$$(\mathbf{y_N})_n \stackrel{\mathrm{def}}{=} \sum_{p=-\infty}^{\infty} y_{(n-pN)}. \qquad (3)$$

This formulation is often referred as *circular correlation*. Efficient implementations of Fourier transform on *PoT-length (Power of Two)* sequences, like FFT in its many flavors, offer us the chance to lower the $O(n^2)$ complexity bound of linear correlation, to improve performances in terms of no. of possible templates checkable for time unit (samples/secs).

3 WIDER NETWORKS CASE STUDY

Wider Networks' UMTS scanner (WIND3G - Figure 1b) has an CPICH Ec/Io[1] detection of -25 dB. Lower Ec/Io detection comes from increased signal processing power.

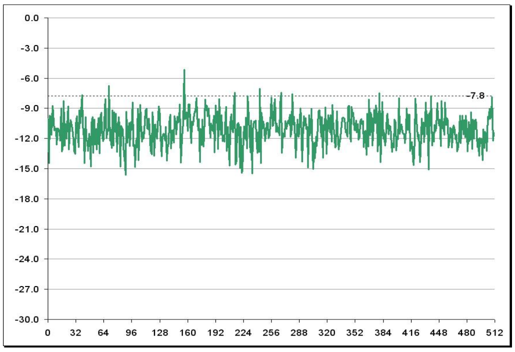
Given a fixed number of CPICH chips as templates, the lower is the requested Ec/Io ratio (to avoid misdetections) the more processing power is needed. Samples' length is another key point when evaluating the real computational complexity of the process, as linear correlation needs to evaluate the signal fitness on the whole target sequence. See Figure 3 to better understand the detection thresholds in the seek process.

We are aimed to improve timing for full cross correlation between air-captured streams and a given number of filters (512 scrambling codes for first stage), starting from a vectorized version of linear cross correlation (provided in Intel Performance Primitives libraries). Actual routines were running in SSE2 cores, floating point single precision IEEE-754 32 bits, and were multicore aware. There are 512 CPICH scrambling code sequences in UMTS. WIND3G takes about 3580 chips worth of each and correlates against one complete UMTS frame, which is 38400 chips to find where these CPICHs have peaks. This step is called "Searching code", and it is actually target of these performance benchmarks. After searching the peak, a DSP algorithm called Tracker takes it in charge to monitor variations over time. Theoretical compound performances of the Searcher+Tracker modules have a bottom limit of 25.5 dB. A peak with this magnitude (and lower) has a considerable probability of being a false detection.

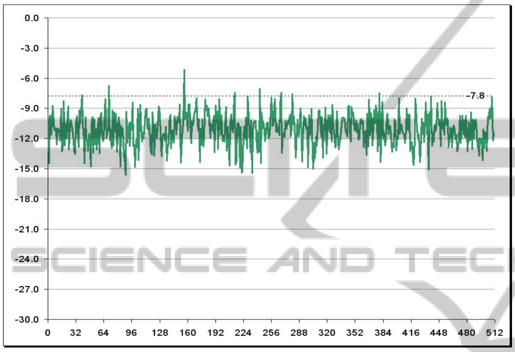
Wider Networks' system are designed to work in mobile environments (cars, service vans) as the UMTS scanner is bundled with a GPS receiver and a geotagging application, to write down on updated maps the dB values, signal efficiency and other measurements done when roaming across cell network covered area. For this reason, their main computing unit is often a notebook, and so with no particular and/or optimized processor for signal processing (up to 2 cores, no SMP, no expansion slots for accelerators, frequencies up to 2.5 GHz due to thermal and power issues).

To overcome to these performance limits, a fast circular cross correlator has been designed to run in GPU, and most notably on NVidia GeForce 8xxx

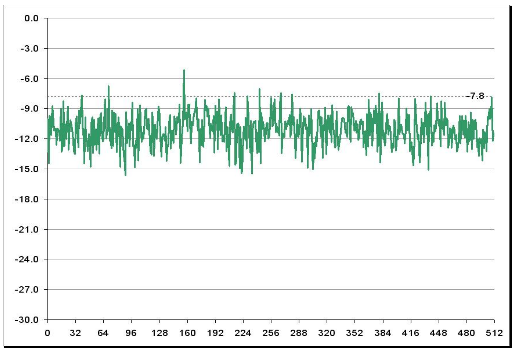
[1] In UMTS/CDMA Ec/Io refers to the portion of the RF signal which is usable. It's the difference between the signal strength and the noise floor.



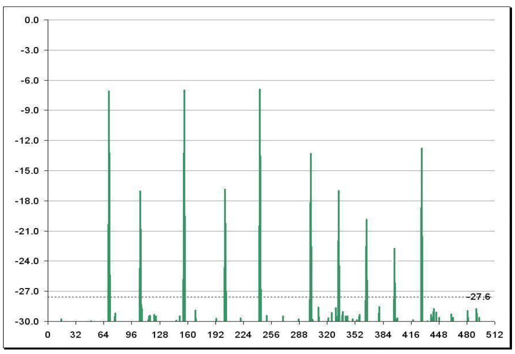
(a) -7.9 dB.



(b) -16.6 dB.



(c) -23.0 dB.



(d) -27.6 dB.

Figure 3: CPICH Ec/Io detection thresholds.

chipsets and newer, by using CUDA development framework under Windows XP (both 32 and 64 bits). The OS choice was mandatory to minimize porting efforts of the whole WIND3G application, already available for that system.

UMTS Cell Search

- Cell Search by correlating CPICH associated with all 512 scrambling codes (α-search)
- In fast-mode (β-search) specified top few candidates (sample positions) are correlated

1. Find the maximum correlation value for each scrambling code $\rightarrow$ 512 max values (run code in 1)
2. Sort these 512 max values, pick the top k scr. codes (because the maximums of the rest scr. codes will be anyway discarded later)
3. For each of these k scr-codes, sort their correlation values for all positions disregarding the values within +/-2 chips of a local max, and pick the top j or so positions $\rightarrow$ In total, this will give us $k * j$ initial candidates
4. Sort the $k * j$ candidates to find the final m candidates to β-search.

Running system use this setup: $k = 200$, $j = 10$, $m = 200$. Values have been tested on field and provide satisfactory results for detection.

3.1 Circular Cross Correlation with CUDA

Once tested the IPP[2] cross correlation code, strictly optimized for x86 platforms, an offloaded routine moving part of the computing costs in GPU is proposed here. See Listing 1 for a significant snippet from `cudaCrossCorr_32fc`.

Listing above requires some explanations for some critical and interesting points. Firstly, this CUDA code is totally "masked" (with singular exception at line 21), keeping reading and maintenance as easy as for plain C/C++ source code. Thanks to modular CUFFT structure derived from FFTW library it has been possible in fact to remove any direct kernel invocation and substitute 1:1 the IPP calls in this algorithm.

Remarkable point is moreover the *stateful* nature of cross corr. code. A major drawback of circular correlation vs. linear one, especially if built up to have exactly the same external interface, would have been

[2]Intel® Integrated Performance Primitives - see http://software.intel.com/en-us/articles/intel-ipp/ for further information.

Listing 1: cudaCrossCorr_32fc source.

```
extern "C" void
cudaCrossCorr_32fc( Complex* h_filter_kernel,
  int filterSize, \
  Complex* h_signal, int signalSize, \
  Complex* h_corr_signal, int corrSize, \
  int lowLag, bool signFFTed)
{
  // Pad signal and filter kernel
  Complex* h_padded_signal, h_padded_filter_kernel;
  [...]
  // Transform signal and kernel
  if (!signFFTed) {
    cufftExecC2C(plan, (cufftComplex *)d_signal, \
    (cufftComplex *)d_signal, CUFFT_FORWARD);
  }
  cufftExecC2C(plan, (cufftComplex *)d_filter_kernel, \
    (cufftComplex *)d_filter_kernel, CUFFT_FORWARD);

  // Multiply coefficients together as in corr a.*conj(b)
  ComplexConjPointwiseMulAndScale<<<32, 256>>> \
    (d_signal, d_filter_kernel, new_size, 1.0f / new_size
         );

  // Check if kernel execution generated and error
  CUT_CHECK_ERROR \
    ("Kernel␣failed␣[ComplexPointwiseMulAndScale]");

  // Transform signal back
  cufftExecC2C(plan, (cufftComplex *)d_signal, \
    (cufftComplex *)d_signal, CUFFT_INVERSE);

  [...]
}
```

the re-computation of reference signal FFT. At every cycle, in fact, a new chunk of ref. signal and a new scrambling code would have been pushed to the cudaCrossCorr function, that would have done the FFT twice per iteration and then apply convolution theorem, multiplying the resulting vectors and anti-transform the result. Designed to work on multiple cross correlations per session sharing the reference signal, instead, using static variables to keep track of how many FFT have been computed, on which part of incoming signal and most important where results are stored in memory, cudaCrossCorr can save up to 30% computing power (1 out of 3 FFT-based operations) per cycle, with a great advantage in terms of execution time. See details in Listing 1 to see how this simple mechanism works.

The only non-masked operation in cudaCrossCorr body is the recombination of spectra after FFT execution, $(f^* * g)$ in eq. 1.

3.2 About Precision in Cross-correlation

Several numeric tests have been executed, using MATLAB as golden standard with its `xcorr` function that implements circular cross correlation, computed in double precision IEEE-754, complex arguments.

CUDA Version showed overall good performances when compared to reference standard. See Figure 4. CUDA FFT `crossCorr`, matched against MATLAB FFT `xcorr` shows a residual term in the order of $\pm 5.0 \cdot 10^{-6}$, so perfectly in line with expectations for single precision 32 bit arithmetic of G92 cores.

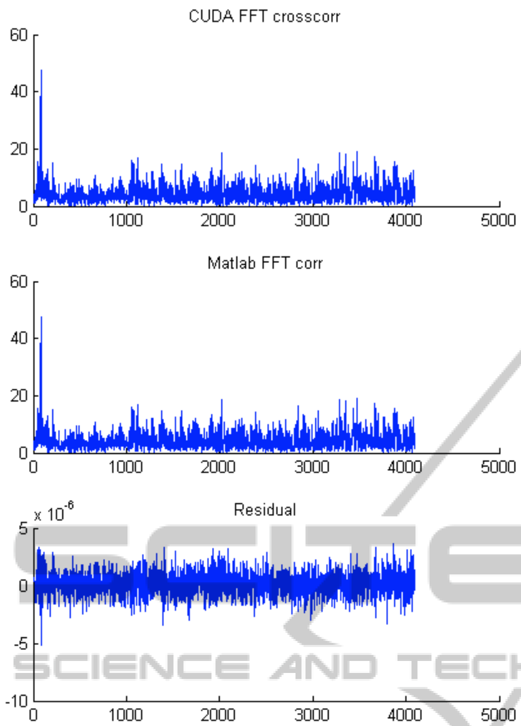
`ippsCrossCorr32fc` on the other hand shows almost the same performance (single precision) on the whole signal length, not visible for bigger Y scale, with a notable exception on master signal's tail. When overlap between reference and template signal exceeded former's length, cross correlation index starts dropping towards zero, for effect of zero padding inside the matching function. Even if this behavior has no practice relevance when looking for "best" match (cross correlation index is guaranteed to become lower than in other parts of data), it could have dangerous effects if resulting vector will be normalized against its L1 or L2 norm. These manipulations are quite frequent in digital signal processing, and in this case such these results could imply a loss of precision of successive steps of computation.

4 COMPUTATIONAL CONSIDERATIONS AND BENCHMARKS

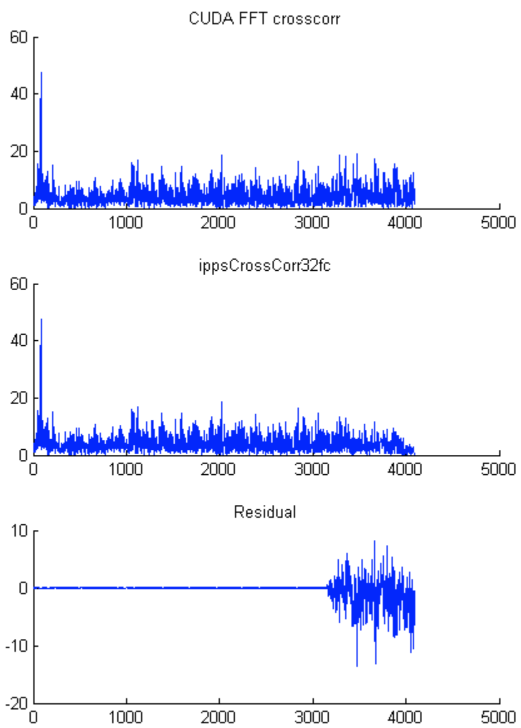
As previously shown in CrossCorr code analysis, all FFT computations have been offloaded from CPU to GPU, and in first instance this process was flawless and simple thanks to the CUFFT library usage. CUFFT provides a simple interface for computing parallel FFTs on an NVIDIA GPU, which allows users to leverage the floating point power and parallelism of the GPU without having to develop a custom, GPU based FFT implementation. FFT libraries typically vary in terms of supported transform sizes and data types. CUFFT API is modeled after FFTW (see http://www.fftw.org), which is one of the most popular and efficient CPU based FFT libraries.

Sadly, CUFFT implementation, or at least until v2.1 of CUDA Toolkit, even showing a general speedup against the underlying CPU structure ($>$ 40 GFlop/s for a grand total of $>$ 20 GB/s on GeForce G92 8800GTS 512), doesn't exploited the full potential of NV's hardware computing power. That library is in fact a collection of 5 different implementations of part of Cooley-Turkey structure, often referred as Radix2 FFT Scheme as in (Tian et al., 2004), extending it to cardinality of 3,4 and 5 elements per block, thus adding to their library the capability to use also Radix-3, Radix-4 and Radix-5 schemes.

Well accepted theory of *"Mixed radix FFT schemes"* is described in detail in (Stasinski and Potrymajo, 2004). Radix-3 and Radix-5 schemes use



(a) CUDA vs. MATLAB.



(b) CUDA vs. IPP.

Figure 4: xCorr precision comparisons.

respectively 35% and 17% more add-mul operations than Radix-2, but for certain sequence lengths Radix-2 have high probability to hit a cache miss when looking up for input operands. Once again, source code for CUFFT for scheduling part, the one spawning the computing threads, is not available in source format, and so we can't figure how authors handled such these optimization problems.

These speculations, anyway, can suggest us an interesting direction to focus attention to develop better code for GPU. Some points:

- FFT stages show a pre-computable locality of input arguments. it is possible to know in advance which data chunks are needed in a processor in a warp, for each step.
- Increasing local FFT size (Radix-3,4,5...N-k) will increase computing time for node, but also will represent a leap forward in stage's map.
- Increasing local FFT size will increase probability of cache miss in input argument, but this event is related to local cache size.

4.1 Increasing Radix Size Approach

The few considerations above are valid for any computing environment, as they're resulting from algorithmic analysis, but have great effect when applied to NVidia GPUs context. All the points highlight the need for local cache of arguments (see (Frigo et al., 1999)), as picking them up from central memory would represent a great penalty in terms of latencies and so for global execution times. Using global memory in GPU is even more problematic than in CPU, due to lack of pre-fetching units for each processor in the array (there is a central memory handler that pushes data to the common bus). Memory latencies for GPU can sum up to 400 cycles for a single transfer, and for this reason keeping low the risk of cache miss is mandatory. Register file for each processor in the array counts up to 32 entries, with lowest latency access, but it would not be large enough to improve data chunk size for thread.

In our help comes the shared memory structure, 16KBytes in pre-GT200 series of CUDA-enabled GPUs, that has low time access too and can be used just declaring variables with `__shared__` attribute. NVCC will understand the memory map to be put in shared separated space, and will push data closer to processing units. This memory bank is shared on Multiprocessor basis, and so many threads can access in parallel to these information.

4.2 Numerical Benchmarks

In (Volkov and Kazian, 2008) is shown an implementation of the FFT that achieves 160 GFlop/s on the GeForce 8800 GTX, which is 3x faster than 50 GFlop/s offered by the CUFFT. It is designed for N = 512, which is hard-coded. Later in the text this version will be referred as *"VV"* code. The implementation also includes cases n = 8 and n = 64 working in a special data layout. This implementation lacked of support for inverse FFT, and moreover used a not CUFFT-like calling scheme. It was so updated and tested always against MATLAB to test precision and CUFFT to benchmark performances. Special kernels for Complex Conjugate operations have been supplied externally, as reported in Listing 1. This latter version, which is the most performant one, will be referred as *"VV+MV mod"* later in the tables.

Using shared memory banks, and keeping N sizes as power of two even in internal stages produces a huge speedup, especially on longer sequences where CUFFT likely spawns a very high number of computing threads. Using mixed radix sizes, moreover, it is very hard to keep summation error as low as possible, as input terms for recombination are divided by odd numbers, and more in general not for 2. For this reason, on longer sequences, CUFFT shows a not optimal precision measure and always higher than VV+MV implementation. All the tests were executed for FFT/iFFT sequences.

Some FFT benchmarks between standard CUFFT design and VV+MV one. CUDA Toolkit v.2.1 on XP 32 bit used. Errors reported are in order of 1 ULP for IEEE-754 32 bit data.

Device: GeForce 8600M GS
900 MHz clock, 256 MB memory
(CUDA Toolkit v. 2.1 - Windows XP 32 bit)

	CUFFT			VV+MV mod		
N / Batch	GFlop/s	GB/s	Err	GF/s	GB/s	Error
8 / 524288	0.4	0.4	1.8	7.2	7.7	1.6
64 / 65536	2.5	1.4	2.3	10.5	5.6	2.2
512 / 8192	2.9	1.0	2.9	12.3	4.4	2.5

Device: GeForce 8800 GTS 512
1674 MHz clock, 512 MB memory
(CUDA Toolkit v. 2.1 - Windows XP 32 bit)

	CUFFT			VV+MV mod		
N / Batch	GFlop/s	GB/s	Err	GF/s	GB/s	Error
8 / 524288	6.0	6.3	1.8	46.5	49.6	1.6
64 / 65536	37.6	20.1	2.3	94.4	50.3	2.2
512 / 8192	43.4	15.4	2.9	125.4	44.6	2.5

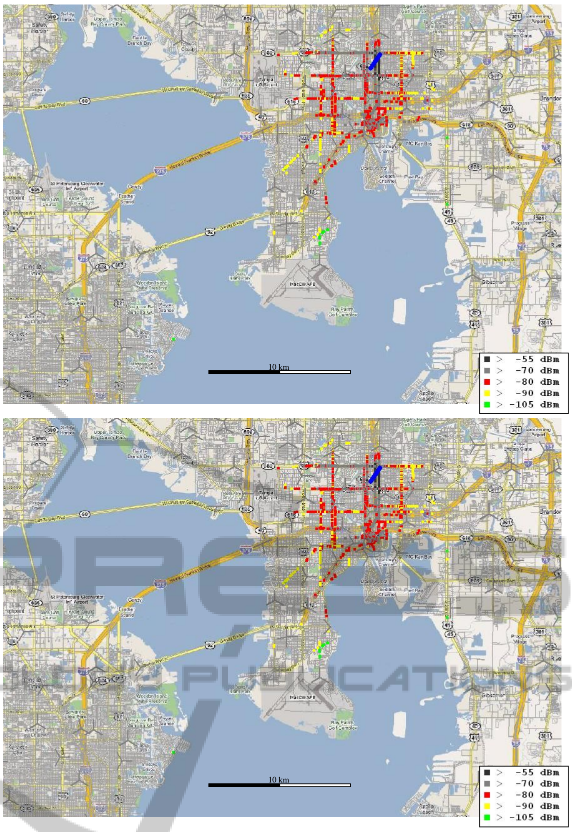


Figure 5: WIND3G Sensitivity improvements in a GPS-tagged context.

In late 2009, NVidia released in 2.3 beta version of their CUDA toolkit an updated CUFFT library, with matches best performances above on longest sequences. Anyway, it still doesn't handle properly shorter length vectors (used often in multimedia processing, expecially for noise samples in audio digital restores).

Here are the same tests with updated values.

Device: GeForce 8800 GTS 512
1674 MHz clock, 512 MB memory
(CUDA Toolkit v. 2.3beta - Windows XP 32 bit)

	CUFFT			VV+MV mod		
N / Batch	GFlop/s	GB/s	Err	GF/s	GB/s	Error
8 / 524288	5.7	6.1	1.7	45.9	49.0	1.6
64 / 65536	36.2	19.3	2.4	92.8	49.5	2.2
512 / 8192	129.1	45.9	2.1	121.9	43.4	2.5

5 CONCLUSIONS

UMTS code scrambling service has proven to be a very demanding task in a production environment.

Before the CUDA solution was developed, the mobile system to be used on the field should have been powered by a HPC-like system, with an obvious expensive price tag - and really power hungry ($> 2 \times 400$ Watts). On the road, such a requirement is to be considered stressful for final users. The CUDA improvement enabled Wider Networks to ship a really mobile system, based upon a standard off-the-shelf GPU equipped laptop, to be used in companion to the WIND3G Scanner. Total cost of ownership of such a system is considerably lowered w.r.t to the original setup. On the results side, the most notable improvement is the higher sensitivity of the scanning procedure, as shown in Figure 5 with a substantially lower energy used in a shorter computation time slice. As side effect, CPU+GPU usage, better scheduled in self-containing computing threads, lowered the total load to the underlying OS from $\geq 90\%$ to $< 35\%$.

REFERENCES

Bornkessel, C. and Wuschek, M. (2006). Exposure measurements of modern digital broadband radio services. Technical report, IMST GmbH, Test Centre EMC, Kamp-Lintfort, Germany University of Applied Sciences Deggendorf, Edlmairst, Deggendorf, Germany.

Frigo, M., Leiserson, C. E., Prokop, H., and Ramachandran, S. (1999). Cache-oblivious algorithms. In *In Proc. 40th Annual Symposium on Foundations of Computer Science*, pages 285–397. IEEE Computer Society Press.

Stasinski, R. and Potrymajo, J. (2004). Mixed-radix fft for improving cache performance. *EUSIPCO 2004, Proceedings of.*

Tian, J., Xu, Y., Jiang, H., Luo, H., and Song, W. (2004). Efficient algorithms of fft butterfly for ofdm systems. In *Emerging Technologies: Frontiers of Mobile and Wireless Communication, 2004. Proceedings of the IEEE 6th Circuits and Systems Symposium on*, volume 2, pages 461–464 Vol.2.

Volkov, V. and Kazian, B. (2008). Fitting fft onto g80 architecture. Technical Report CS 252, University of California, Berkeley.