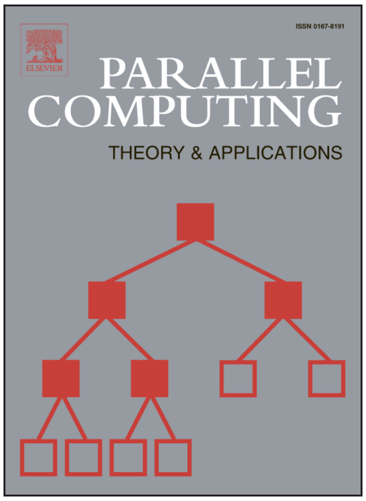

Short communication

Optimized Fast Walsh-Hadamard Transform on GPUs for Non-Binary LDPC Decoding

Joao Andrade, Gabriel Falcao, Vitor Silva

PII: S0167-8191(14)00081-7
DOI: http://dx.doi.org/10.1016/j.parco.2014.07.001
Reference: PARCO 2196

To appear in: *Parallel Computing*

Optimized Fast Walsh-Hadamard Transform on GPUs for Non-Binary LDPC Decoding

Joao Andrade[1,*], Gabriel Falcao, Vitor Silva

Instituto de Telecomunicações, Department of Electrical and Computer Engineering University of Coimbra, 3030-290 Coimbra, Portugal

Abstract

The Fourier Transform Sum-Product Algorithm (FT-SPA) used in non-binary Low-Density Parity-Check (LDPC) decoding makes extensive use of the Walsh-Hadamard Transform (WHT). We have developed a massively parallel Fast Walsh-Hadamard Transform (FWHT) which exploits the Graphics Processing Unit (GPU) pipeline and memory hierarchy, thereby minimizing the level of memory bank conflicts and maximizing the number of returned instructions per clock cycle for different generations of graphics processors, with considerable speedup gains in FT-SPA based non-binary LDPC decoding.

Keywords: Non-Binary LDPC Codes, Parallel Processing, Walsh-Hadamard Transform

1. Introduction

Non-binary Low-Density Parity-Check (LDPC) decoding is a relatively new area of research in digital communications. Non-binary LDPC codes are known for having superior Bit Error Rate (BER) performance regarding to the binary case [1]. In order to accommodate the high computational power demanded, ASIC decoders have already been developed [2]. We propose a new GPU-based approach that exploits the processing power of modern multicore systems. Among all the sub-kernels of the Fourier domain decoder [3],

[*]Corresponding author: joao.andrade@co.it.pt.

[1]This work was supported by Fundação para a Ciência e Tecnologia scholarship grant SFRH/BD/78238/2011 and project grant PEst-OE/EEI/LA0008/2013.

the Walsh-Hadamard Transform (WHT) presents the highest computational workload, occupying 45∼95% of the global processing time [4, 5], and its efficient parallelization is absolutely fundamental and represents a challenge if the memory hierarchy of the GPU system is fully exploited, as it should be. In particular, we analyse different radix-n factorizations [6] and address the performance impact of memory bank conflicts occurring on the fast shared memory of the GPU system.

2. The FWHT role in the SPA LDPC Decoding Context

The Sum-Product Algorithm (SPA), used for decoding binary LDPC codes can be extended to deal with LDPC codes over GF(2^m) [1]. Its numerical complexity, however, grows non-linearly with the field's order m which detracts its usage as a suitable decoding algorithm over GF(2^m), namely due to the Check Node (CN) processing step [1, 3]:

$$m_{cv}^{(i)}(x)=\sum_{\mathbf{v}:v_z=x} p(z_c=0|\mathbf{v}) \prod_{v'\in V(c)\setminus c} m_{v'c}(x), \forall x \in \mathrm{GF}(2^m)\ , \tag{1}$$

where d_c denotes the CN degree; m_{cv} and m_{vc} are the messages exchanged from CN to Variable Node (VN) and from VN to CN respectively; $V(c)\setminus v$ is the set of all VNs participating in parity-check equation z_c evaluated for a given $v \in \mathrm{GF}(2^m)$; and $p(z_c = 0|\mathbf{v})$ evaluates to one whenever the combination of the remaining VNs verify the parity-check equation. The complexity of equation (1), follows $\mathrm{O}(M \cdot d_c \cdot 2^{d_c \cdot m})$ and can be ameliorated by switching from the *probability mass function* (*pmf*) domain to the Fourier domain [7], which transforms the convolution (1) into a product:

$$m_{cv}^{(i)}(x)=\mathrm{WHT}\left\{\prod_{v'\in V(c)\setminus c} \mathrm{WHT}\left\{m_{v'c}(x)\right\}\right\}. \tag{2}$$

Equation (2) uses the WHT instead of the DFT since the Fourier Transform of *probability mass functions* (*pmfs*) defined over GF(2^m) consists of the WHT, which possesses the involution property [7]. Hence, instead of utilizing the Fast-Fourier Transform (FFT) algorithm we can interchange the domain through the Fast Walsh-Hadamard Transform (FWHT) (2). The Walsh-Hadamard matrix is obtained by following the Kroenecker product $\bigotimes$ [6]:

$$\mathbf{H}_{2^k}=\mathbf{H}_2 \bigotimes \mathbf{H}_{2^{k-1}}, \text{where } \mathbf{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{3}$$

Since the numerical complexity of (2) is $O(M \cdot d_c \cdot m \cdot 2^m)$ and the FWHT follows $O(2^m \log_2 2^m)$, the FWHT represents the highest computational burden in (2). One of the primary challenges of the Fourier Transform Sum-Product Algorithm (FT-SPA) decoder is to develop an efficient FWHT, as we have identified in [4]. The LDPC decoding context favours the execution of several WHTs in parallel, one transform per each *pmf* vector [7]. Thus, there are two levels of parallelism exposed: *i)* intra-FWHT by assigning multiple threads to compute the WHT; and *ii)* inter-FWHT by computing several transforms per execution grid, assigning one block per transform. Other authors have considered the probability and the log-likelihood domains to develop non-binary LDPC decoding algorithms on GPU [8, 9, 10] that do not consider the use of the FWHT.

3. Parallel FWHT on GPU

The Compute Unified Device Architecture (CUDA) GPU engine is a heavily-threaded processor, capable of launching several thousands of concurrent threads which execute a kernel. It is composed of several Streaming Processors (SPs) grouped into Stream Multiprocessors (SMs) where spawned threads run in an execution grid. The grid defines a number of blocks with a given pre-selected thread width. Threads belonging to the same block have access to synchronization and memory fencing functions. Its memory hierarchy is shown in Figure 1, and it comprises the following memory spaces: the register file; the on-chip shared-memory space – organized in B memory banks – where threads belonging to the same thread block can exchange data; the L1-cached read-only constant memory; and the off-chip global memory providing high latency and high bandwidth and an addressing space for CPU-GPU communication [11].

For the GPU platform, we are interested in exploring the parallel capabilities of the hardware and also in exploiting the memory hierarchy in order to find the best performing solution. We have analyzed transform sizes $N = \{128, 256\}$, which are high-order finite field dimensions under the non-binary LDPC decoding context [4, 7]. Thus, we are able to define FWHT kernels that fully utilize the low latency, high bandwidth register file and shared memory spaces, by developing a GPU-aware parallel FWHT with two-levels of parallelism, as shown in Figure 1. Complementary to the work presented in [4], we focus on the particularities of the shared memory architecture across the CUDA GPUs that enable high throughput computation of the FWHT in the GPU engine.

A naive view of the FWHT computation is shown in Figure 1. Shuffling data through the global memory would prove too slow due to high latency accesses, but use of the shared memory space requires proper alignment of the data elements. In the illustrated example, whenever thread t_i accesses the same bank no conflict occurs, although when t_i needs to access different banks, it creates a conflict with other threads. For instance, bank 0 is accessed conflict free on the first stage but is accessed with conflict by threads t_0 and t_2 in both the second and third stage, respectively. Also, arbitrary strided shared memory accesses on the GPU can unravel bank conflicts. Assuming threads access data with a stride s, threads tid and $tid + n$ will create a bank conflict when $s{\times}n$ is a multiple of B. Considering that $B{=}\{16, 32\}$ [11], this conflict occurs whenever n is a multiple of B/d, d the greatest common divisor of B and s, hence no conflict occurs when s is odd.

Hence, conflict free bank access can be addressed by using a proper access ordering of the data elements for each transform stage [12], which exploit the thread scheduler ability to mask computation with memory accesses. Defining the correct stride s requires an allocated shared memory space larger than the dimension of the FWHT, in order to find $d{=}1$. In this case, the classical in-place computation must be dropped in favour of out-place computation. By ensuring that d remains 1, equivalently making the stride s odd, threads tid and $tid + n$ will access the same bank only when $n{=}B$, which unfolds two distinct memory transactions, and no conflicts occur on the shared memory. This lead to the development of a mixed radix [6] FWHT factorization based on radix-4 and -8 for $N = 128$, and radix-4 for $N = 256$, using 64 threads per block. This allowed us to find $d = 1$ for all stages in a $B{=}32$ GPU, but not for all stages when $N{=}256$ and $B{=}16$.

4. Experimental results

The tests were run on a 12GB of RAM Intel i7 950 @ 3.07GHz 64-bit GNU/Linux 3.4.4 platform with three GPUs: *a)* a Tesla C1060 ($B{=}16$); *b)* a Tesla C2050 ($B{=}32$); *c)* and a GeForce GTX680 ($B{=}32$), using CUDA 5.0 [11]. The proposed kernels were benchmarked against the 16-bank optimized radix-2 FWHT [13], which computes one WHT per block for 128-threaded blocks, and used in the LDPC decoder available at [5], computing 768 WHTs in parallel.

Table 1: Throughput and profiling metrics shown for the developed 16- and 32-bank optimized, and for the 16-bank optimized [13] FWHT kernels. The throughput is expressed as the number of Walsh-Hadamard Transforms computed per millisecond (WHTs/ms). The profile metrics included are: Instructions per Clock (IPC) and bank conflict probability handled by the memory engine.

Optimization	GPU	N	WHTs/ms	IPC	Bank Conf. Prob. (%)
Ours, $B = 16$	*a)*	128	37434	N/A	0
		256	18665		0.25
	b)	128	67766	0.678	0.51
		256	18783	0.578	0.69
	c)	128	69521	0.731	0.44
		256	17178	0.509	1.34
Ours, $B = 16$	*a)*	128	38959	N/A	0
		256	27303		0.07
	b)	128	75007	0.725	0
		256	34035	0.938	0
	c)	128	72754	0.788	0
		256	51699	1.424	0
$B = 16$ [13]	*a)*	256	3531	N/A	2.01
	b)		4039	0.364	5.92
	c)		3138	0.278	4.85

The obtained results are shown in Table 1. As expected, the GPU memory engine handled the fewer bank conflicts for the versions that matched the number of banks they were optimized for. The FWHT used to benchmark has been developed for $N = 256$ using a greedy algorithm for reducing the number of bank conflicts on B=16 bank architectures [13]. This, however, has a bank conflict probability an order of magnitude higher than that of the proposed B=16 bank optimized FWHT on B=32 bank architectures. In fact, running the algorithm on the architecture for which it has been optimized, produces speedups of 1.08 and 1.81, for N=128 and N=256, respectively, on the compute capability 2.0 GPU device. For the same settings, on the 3.0 compute capability processor, speedups obtained were 1.04 and 3.01, also for N=128 and N=256, respectively. In relative terms, the bank conflict probability is very low, at most 1.34% for the proposed implementations and at most 5.92% for the FWHT proposed in [13]. However, the overall effect in terms of IPC variation, shown in Table 1, is significant. For N=256 on the 2.0 GPU, eliminating the 0.69% bank conflict probability leads to an IPC increase of $\sim$ 50%, while on the 3.0 GPU the 1.34% of conflicting accesses leads to a three-fold increase in the achieved IPC. For N=128 on the 3.0 GPU, eliminating the 0.44% probability elevated the IPC by 1.09. The potential penalization incurred with even a low probability of bank conflicts justifies the development of proficient bank conflict-free accesses on the GPU engine. The most efficient FWHT achieves a $\sim$10 times speedup when compared with the benchmark FWHT in [13], being able to lower the FWHT execution time from 45% to 95% of the total FT-SPA decoding time [4].

The impact of the variation in bank conflict probability on the throughput can be assessed by comparing the FWHT optimized for $B = 32$ with the one for B=16, since the former always yields lower conflict probabilities than the latter. The speedup range for this comparison is represented in Figure 2, and spans from 1.04 to 3. As expected the highest speedup occurs for point F at 3 which experienced a 1.34% conflict probability reduction, whereas point A sees a negligible 1.04 speedup. As observed, speedups are higher for the larger 256-point FWHT, regardless of experiencing a lower variation in bank conflict probability as seen with setups B and D (B=32) and C and E (B=16), experiencing speedups of 1.46, 1.81, 1.11 and 1.05, for variations of 0.18%, 0.69%, 0.51% and 0.44%, respectively.

5. Conclusions

The developed parallel FWHT kernels exploit the full memory hierarchy of the GPU, minimizing the number of bank conflicts for shared memory accesses, and achieved very high throughput performances, one order of magnitude higher than the benchmarked FWHT [13]. Moreover, we have shown that even very small bank conflict probabilities on the shared memory accesses can be dauntingly penalizing over the achieved performance. Therefore, bank conflict elimination should be tightly connected to the development of GPU algorithms which maximize the obtained arithmetic intensity. The impact of the optimizations carried out can increase the throughput of the FWHT from 45% to 95% of the total computation time for the FT-SPA algorithm [4].

References

[1] M. Davey, D. J. C. MacKay, Low Density Parity Check Codes over GF(q), in: Information Theory Workshop, 1998, 1998, pp. 70–71.

[2] A. Voicila, F. Verdier, D. Declercq, M. Fossorier, P. Urard, Architecture of a Low-complexity Non-binary LDPC Decoder for High Order Fields, in: IEEE ISCIT2007, 2007, pp. 1201–1206.

[3] L. Barnault, D. Declercq, Fast Decoding Algorithm for LDPC over GF(2q), in: Information Theory Workshop, 2003, 2003, pp. 70–73.

[4] J. Andrade, G. Falcao, V. Silva, K. Kasai, FFT-SPA Non-binary LDPC Decoding on GPU, in: IEEE ICASSP 2013, 2013, pp. 5099–5103.

[5] K. Kasai, Y. Fujisaka, M. Onsjo, FFT-Based Parallel Decoder of Non-Binary LDPC Codes on GPU: KFO_NBLDPC_GPU, [Online; accessed February/2014]. URL : http://www.comm.ss.titech.ac.jp/~kenta/KFO_NBLDPC_GPU.tar.gz

[6] C. V. Loan, Computational Frameworks for the Fast Fourier Transform, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

[7] R. A. Carrasco, M. Johnston, Non-Binary Error Control Coding for Wireless Communication and Data Storage, Wiley, Chichester, 2008.

[8] G. Wang, H. Shen, B. Yin, M. Wu, Y. Sun, J. Cavallaro, Parallel Non-binary LDPC Decoding on GPU, in: IEEE ASILOMAR 2012, 2012, pp. 1277–1281.

[9] D. Romero, N. Chang, Sequential Decoding of Non-binary LDPC Codes on Graphics Processing Units, in: IEEE ASILOMAR 2012, 2012, pp. 1267–1271.

[10] M. Beermann, E. Monro, L. Schmalen, P. Vary, High Speed Decoding of Non-binary Irregular LDPC Codes using GPUs, in: IEEE SiPS 2013, 2013, pp. 36–41.

[11] NVIDIA, CUDA C Programming Guide 5.5, NVIDIA, 2013.

[12] N. Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, J. Manferdelli, High Performance Discrete Fourier Transforms on Graphics Processors, in: ACM/IEEE SC 2008, 2008, pp. 1–12.

[13] M. Onsjo, K. Kasai, O. Watanabe, CUDA Implementation of Iterative Updating: the Radix-2 Algorithm and Discrete Fourier Transforms, Tech. Rep. C-268, Tokyo Institute of Technology (Feb. 2011).

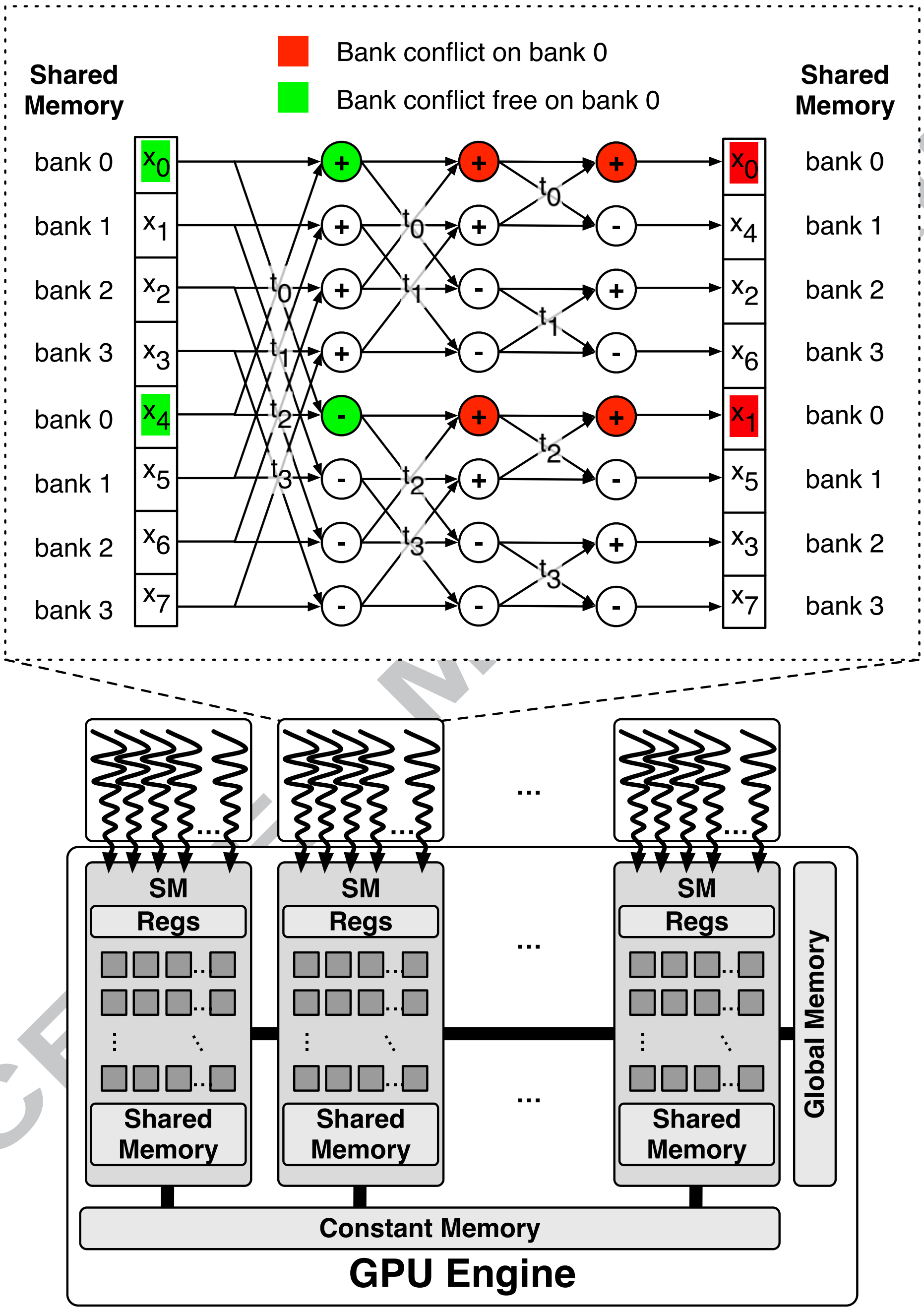

Figure 1: GPU architecture and corresponding memory hierarchy showing the execution of an 8-point radix-2 factorization FWHT example with in-place computation, hypothetically assuming B=4 shared memory banks, and assigning a GPU thread t_i per butterfly. In this case, the in-place computation of the FWHT will be conflict free for bank 0 in the first stage – only thread t_0 accesses it – but will yield conflicts in the second and third ones – since it is accessed by threads t_0 and t_2 in both cases.

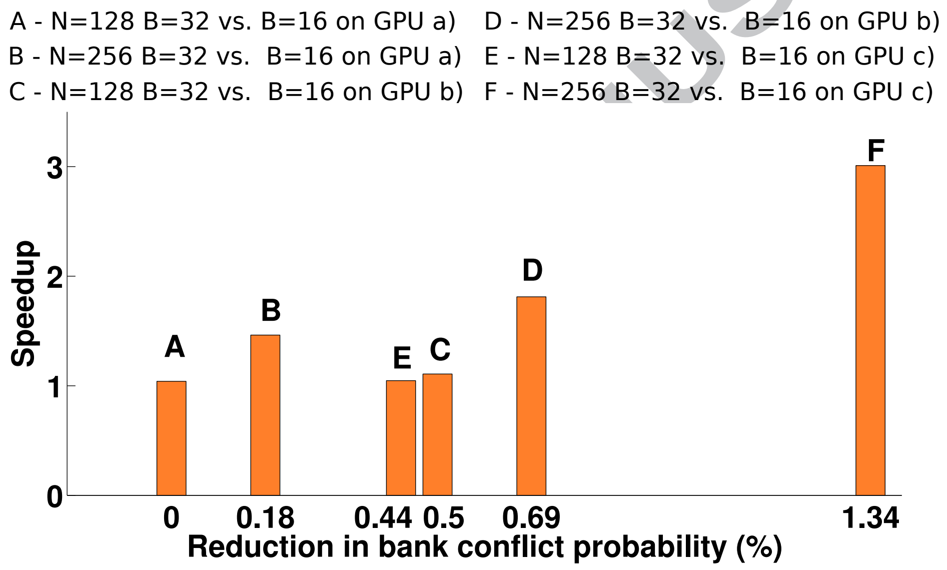

Figure 2: Speedup achieved for the proposed B=32 FWHT when compared to the B=16 FWHT versus the corresponding reduction in bank conflict probability in percentage.

We analyse the role of the FWHT under the non-binary LDPC decoding problem.

We quantify the trade-off between memory bank conflicts and the throughput on GPUs.

The FWHT employs radix-n approaches tuned to the number of shared memory banks.

The FWHT was tuned for both 16 and 32 shared memory bank GPU architectures.