Cache miss analysis of WHT algorithms

Mihai Furis[1] and Paweł Hitczenko[2†] and Jeremy Johnson[1‡]

[1] *Dept. of Computer Science, Drexel University, Philadelphia, PA 19104, USA. {mfuris,jjohnson}@cs.drexel.edu*
[2] *Dept. of Mathematics, Drexel University, Philadelphia, PA 19104, USA. phitczenko@cs.drexel.edu*

On modern computers memory access patterns and cache utilization are as important, if not more important, than operation count in obtaining high-performance implementations of algorithms. In this work, the memory behavior of a large family of algorithms for computing the Walsh-Hadamard transform, an important signal processing transform related to the fast Fourier transform, is investigated. Empirical evidence shows that the family of algorithms exhibit a wide range of performance, despite the fact that all algorithms perform the same number of arithmetic operations. Different algorithms, while having the same number of memory operations, access memory in different patterns and consequently have different numbers of cache misses. A recurrence relation is derived for the number of cache misses and is used to determine the distribution of cache misses over the space of WHT algorithms.

Keywords: Cache, Divide and Conquer Recurrences, Geometric Distributions, Memory Access Patterns, Performance Analysis, Random Compositions, Walsh-Hadamard Transform

1 Introduction

The complexity of modern computer architectures has made it increasingly difficult to optimize the implementation of an algorithm to the underlying architecture on which it is implemented. Moreover, high-performance demands that an implementation be tuned to take advantage of features of the architecture such as pipelining, superscalar execution, and the memory hierarchy (see Hennessy and Patterson (2002) for a discussion of these and many other features of modern computer architectures). In particular, effective utilization of cache can be more important than operation count in obtaining high-performance implementations, and can lead to an order of magnitude improvement in performance.

The importance of cache utilization for high-performance implementation of algorithms has been known for some time. Optimizing compilers perform many transformations to obtain better locality of reference and hence better cache utilization (see for example Allen and Kennedy (2002)). Several theoretical models have been proposed that abstract memory performance and allow the algorithm designer to focus on the key issues involved in designing algorithms with efficient utilization of the memory hierarchy Aggarval and Vitter (1988); Ladner et al. (1999); Sen et al. (2002). Most of this work deals with lower bounds for the worst case behavior and the design of asymptotically optimal algorithms for key problems such as sorting and the fast Fourier transform (FFT). The paper by Ladner et al. includes some probabilistic analysis for certain memory access patterns.

Despite the theoretical results and the tools available through optimizing compilers obtaining practical algorithms that efficiently utilize cache remains a major challenge. The difficulty in achieving high-performance on modern architectures with deep memory hierarchies, combined with the short time between the introduction of new architectures, has led to the field of automated-performance tuning, where a large class of algorithm and implementation choices are automatically generated and searched to find an "optimal" implementation (see Moura et al. (2005)).

This paper continues the exploration in Hitczenko et al. (2004) on the analysis of the performance of a family of algorithms (see Johnson and Püschel (2000)) to compute the Walsh-Hadamard transform (WHT). The WHT is a transform used in signal and image processing and coding theory Beauchamp (1984); Elliott and Rao (1982); MacWilliams and Sloane (1992) and has an algorithmic structure similar to the FFT. The algorithms in Johnson and Püschel (2000) have a wide range of performance despite having exactly the same number of arithmetic operations. Different numbers of instruction due to different

[†] Supported in part by NSA Grants #MSPF-02G-043 and #MSPF-04G-054.
[‡] Supported in part by NSF Grant ITR/NGS #0325687: Intelligent HW/SW Compilers for DSP.

amounts of recursion and iteration can account for some of the differences in performance, and the work in Hitczenko et al. (2003) developed and analyzed a performance model based on instruction count. However, instruction count by itself does not accurately predict performance; especially for larger sizes where cache performance becomes a significant factor. In this paper a model for cache performance is developed and the space of WHT algorithms is analyzed with respect to this model. A recurrence relation is derived for the number of cache misses, for a direct-mapped cache, and it is used to determine the distribution of cache misses over the space of WHT algorithms. The restriction to a direct-mapped cache simplifies some of the analysis, though the framework is general, and a similar analysis should be possible for other cache configurations.

2 Walsh-Hadamard Transform Algorithms

The Walsh-Hadamard transform of a signal x, of size $N = 2^n$, is the matrix-vector product $\mathbf{WHT}_N \cdot x$, where

$$\mathbf{WHT}_N = \bigotimes_{i=1}^{n} \mathbf{DFT}_2 = \overbrace{\mathbf{DFT}_2 \otimes \cdots \otimes \mathbf{DFT}_2}^{n}.$$

The matrix

$$\mathbf{DFT}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

is the 2-point DFT matrix, and $\otimes$ denotes the tensor or Kronecker product. The tensor product of two matrices is obtained by replacing each entry of the first matrix by that element multiplied by the second matrix. Different algorithms for computing the WHT can be derived from different factorizations of the WHT matrix (see Johnson and Püschel (2000)). Let $n = n_1 + \cdots + n_t$, then

$$\mathbf{WHT}_{2^n} = \prod_{i=1}^{t} (\mathrm{I}_{2^{n_1+\cdots+n_{i-1}}} \otimes \mathbf{WHT}_{2^{n_i}} \otimes \mathrm{I}_{2^{n_{i+1}+\cdots+n_t}}) \tag{1}$$

This equation encompasses both the iterative ($t = n$, $n_i = 1$ for $i = 1, \ldots, t$) and recursive algorithms ($t = 2$, $n_1 = 1$ and $n_2 = n - 1$), and provides a mechanism for exploring different breakdown strategies and combinations of recursion and iteration. The breakdown strategy used by a particular algorithm can be encoded in a tree called a partition tree. The root of the tree is labeled by n, the exponent of the size of the transform, and the children of a node are labeled by the exponents, $n_1, \ldots, n_t$ of the recursive transforms used in the application of Equation 1. Figure 1 shows the trees corresponding iterative and recursive algorithms for $\mathbf{WHT}_{16}$.

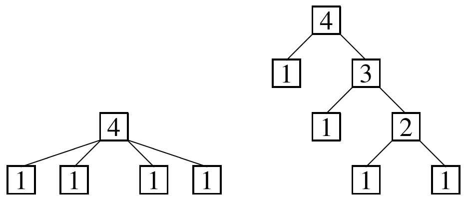

Fig. 1: Partition Trees for Iterative and Recursive WHT Algorithms

The WHT package, from Johnson and Püschel (2000), provides a mechanism to implement all of the algorithms that can be derived from Equation 1. Let $N = N_1 \cdots N_t$, where $N_i = 2^{n_i}$, and let $x_{b,s}^{M}$ denote the subvector $(x(b), x(b+s), \ldots, x(b+(M-1)s))$. Then evaluation of $x = \mathbf{WHT}_N \cdot x$ using Equation

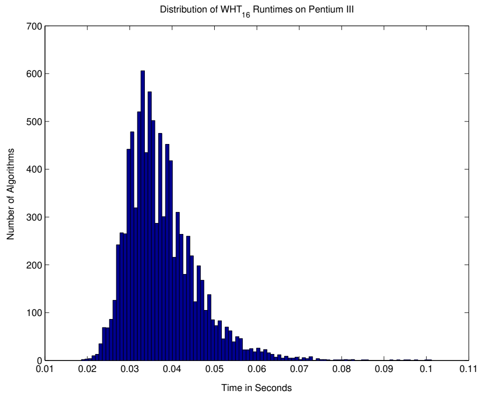

Fig. 2: Performance histogram on the Pentium III

1 is performed using

$$
\begin{array}{l}
R = N;\;\; S = 1; \\
\text{for } i = 1, \ldots, t \\
\;\; R = R/N_i; \\
\;\; \text{for } j = 0, \ldots, R-1 \\
\;\;\;\; \text{for } k = 0, \ldots, S-1 \\
\;\;\;\;\;\; x^{N_i}_{jN_iS+k,S} = \mathbf{WHT}_{N_i} \cdot x^{N_i}_{jN_iS+k,S}; \\
\;\; S = S * N_i;
\end{array}
$$

The computation of W_{N_i} is computed recursively in a similar fashion until a base case of the recursion is encountered. Observe that $\mathbf{WHT}_{N_i}$ is called N/N_i times. Small WHT transforms, used in the base case, are computed using the same approach; however, the code is unrolled in order to avoid the overhead of loops or recursion. This scheme assumes that the algorithm works in-place and is able to accept stride parameters.

Equation 1 leads to many different algorithms with a wide range in performance, as shown in the histogram of runtimes in Figure 2 for 10,000 randomly generated WHT algorithms.

It is easy to show that all algorithms derived from Equation 1 have exactly the same number of arithmetic operations ($N \lg(N)$). Since arithmetic operations can not distinguish algorithms, the distribution of runtimes must be due to other factors. In Huang (2002), other performance metrics, such as instruction count, memory accesses, and cache misses, were gathered and their influence on runtime was investigated. Different algorithms can have vastly different instruction counts due to the varying amounts of control overhead from recursion, iteration, and straight-line code. Different algorithms access data in different patterns, with varying amounts of locality. Consequently different algorithms can have different amounts of instruction and data cache misses.

3 Cache Model

Measurements from Huang (2002) and Furis (2003) show that different WHT algorithms have different numbers of cache misses and that the number of cache misses affects the runtime performance. In general the number of cache misses do not fully account for memory performance (other factors are the number of memory operations, miss penalty, and the overlap of memory operations with other instructions); however, since all of the WHT algorithms have similar number of memory operations, the number of cache misses provides an effective way of measuring memory performance.

In this section an idealized memory model is presented from which it is possible to determine the exact number of cache misses for each of the WHT algorithms using different cache configurations. A cache configuration is specified by the size of the cache, the block size and its associativity (see Hennessy and Patterson (2002)). The memory model includes only the elements of the vector x used to compute $x = \mathbf{WHT}x$ (memory to hold instructions, index variable and temporary values are not considered) and each element occupies a single memory location.

Tab. 1: Memory Access Pattern for Algorithm (a) in Figure 3

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$(\mathrm{I}_8 \otimes \mathbf{WHT}_2)x_{0,1}^{16}$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$\mathbf{WHT}_8 x_{0,2}^{8}$	2		2		2		2		2		2		2		2	
	3				3				3				3			
			3				3				3				3	
$\mathbf{WHT}_8 x_{1,2}^{8}$		2		2		2		2		2		2		2		2
		3				3				3				3		
				3				3				3				3

The memory access pattern of a WHT algorithm is obtained from a trace of the read and write accesses to the vector x. This pattern can be determined from the partition tree representing the algorithm (all accesses correspond to leaf nodes of the tree). Tables 1 and 2 show the access patterns for the algorithms represented by the trees in Figure 3. The memory accesses proceed from left to right and from top to bottom and are labeled by the leaf node in the tree where the memory access occur (leaf nodes are labeled right to left). The actual implementation from Johnson and Püschel (2000) accesses some of these elements multiple times. In particular, the computation of the base case $\mathbf{WHT}x_{b,S}^{m}$ performed by `small[m]` of size $M = 2^m$, accesses the elements $x[b], x[b+S], x[b], x[b+S], x[b+2S], x[b+3S], x[b+2S], x[b+3S], \ldots, x[b+(M-2)S], x[b+(M-1)S], x[b+(M-2)S], x[b+(M-1)S], x[b], x[b+S], \ldots, x[b+(M-1)S]$. The first $2M$ accesses are reads and the last M accesses are writes. The elements are read twice, first to compute their sum and second to compute their difference. The sum and difference are stored in temporary variables and the rest of the computation proceeds without accessing the vector x until the result of $\mathbf{WHT}x_{b,S}^{m}$ is stored back to x.

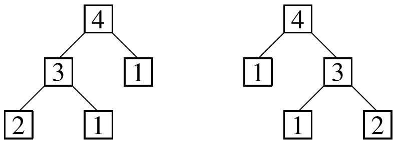

Fig. 3: Partition Trees for Algorithm (a) and (b) for $\mathbf{WHT}_{16}$

A program was written to generate the memory trace (reads and writes were not distinguished) of a WHT algorithm specified by a partition tree and the trace was fed to a cache simulator to count the number of cache misses. For example, the trace generated by Algorithms (a) and (b) in Figure 3 have 144 memory accesses and 80 and 112 misses respectively when using a direct-mapped cache of size 4 with blocks of size 1. When the associativity is increased to 4 (fully associative) the number of misses drops to 48 for both algorithms. If a direct-mapped cache of size 4 with blocks of size 2 is used the number of misses is 72 and 88 respectively. In contrast, the iterative and recursive algorithms from Figure 1 both have 192 accesses with 128 and 112 misses when using a direct-mapped cache of size 4.

Many experiments were performed and a particularly simple pattern was observed for direct-mapped caches (for data sizes larger than cache size $C = 2^c$, there are c different values for the number of misses). For example, the possible values for the number of misses for a WHT algorithm of size $N = 2^{10}$ when using a direct-mapped cache of size 2^c for $c = 1, \ldots, 6$ is shown in Table 3.

4 A Formula for the Number of Cache Misses

There is a simple formula for the number of cache misses for a given WHT algorithm provided the cache is direct-mapped (i.e. has associativity equal to one) and has block size one. The assumption of a direct-mapped cache implies that the number of misses encountered when computing the WHT using the factorization in Equation 1 can be computed independently for each of the factors provided the stride is taken into account. It should be possible to analyze more general cache configurations, however, the simplifying assumption allows a concise result which is the focus of this paper.

Tab. 2: Memory Access Pattern for Algorithm (b) in Figure 3

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathbf{WHT}_8 x^8_{0,1}$	1	1	1	1	1	1	1	1								
	2				2											
		2				2										
			2				2									
				2				2								
$\mathbf{WHT}_8 x^8_{8,1}$									1	1	1	1	1	1	1	1
									2				2			
										2				2		
											2				2	
												2				2
$(\mathbf{WHT}_2 \otimes \mathrm{I}_8) x^{16}_{0,1}$	3								3							
		3								3						
			3								3					
				3								3				
					3								3			
						3								3		
							3								3	
								3								3

Tab. 3: Possible Number of Cache Misses for $\mathbf{WHT}_{2^{10}}$ on Direct-Mapped Cache of Size 2^c

c	m_1	m_2	m_3	m_4	m_5	m_6
1	28672					
2	25600	26624				
3	22528	23552	24576			
4	19456	20480	21504	22528		
5	16384	17408	18432	19456	20480	
6	13312	14336	15360	16384	17408	18432

Let $\mathbf{Misses}(\mathcal{W}_n, r)$ be the number of misses to compute $\mathcal{W}_n x_{b,S}^N$ where $S = 2^r$ is the input stride. The following theorem provides a recurrence relation for computing $\mathbf{Misses}(\mathcal{W}_n, r)$ under the assumption of a direct-mapped cache.

The first test is to determine if the input data, accessed at stride 2^r fits in cache. The second and third conditions, assume that the data does not fit in cache and that the base case in the algorithm has been encountered. If the stride is larger than the cache size, the consecutive pair of reads (see the memory access pattern discussed in the previous section) both cause misses due to the assumption that the cache has associativity one. Even if the stride is not bigger than the cache, the assumption that the data does not fit in cache and the fact that the size of the data and cache are both powers of two implies that the writes following the reads will all incur misses.

The last case assumes the data does not fit in cache and that Equation 1 was applied in the algorithm. The computation of $(\mathrm{I}_{n_1+\cdots+n_{i-1}} \otimes \mathcal{W}_{n_i} \otimes \mathrm{I}_{n_{i+1}+\cdots+n_t})$ makes 2^{n-n_i} recursive calls to compute $\mathcal{W}_{n_i}$ with input data accessed at stride $2^{n_{i+1}+\cdots+n_t+r}$. Since the input does not fit in cache and the associativity is one, the data accessed in each of the recursive calls will not be in cache when the recursive call is made. This implies that the number of misses for each recursive call can be computed independently.

Theorem 1 *Let $\mathcal{W}_n$ be a WHT algorithm to compute $\mathbf{WHT}_N$ with $N = 2^n$. Assume $\mathcal{W}_n$ is split into t children $\mathcal{W}_{n_1}, \ldots, \mathcal{W}_{n_t}$, where $\mathcal{W}_{n_i}$ computes $\mathbf{WHT}_{2^{n_i}}$. Then the number of misses to compute $\mathcal{W}_n x_{b,S}^N$ where $S = 2^r$ using a direct-mapped cache of size $C = 2^c$ is*

$$\mathbf{Misses}(\mathcal{W}_n, r) = \begin{cases} 2^n, & \text{if } 2^n \le \lceil C/2^r \rceil \\ 3 \cdot 2^n, & \text{if } \mathcal{W}_n \text{ is small and } 2^r \ge C \\ 2 \cdot 2^n, & \text{if } \mathcal{W}_n \text{ is small and } 2^r < C \\ \sum_{i=1}^{t} 2^{n-n_i} \mathbf{Misses}(\mathcal{W}_{n_i}, n_{i+1} + \cdots + n_t + r), & \text{otherwise.} \end{cases}$$

This recurrence relation can be used to compute the number of cache misses without having to do a simulation. For example, let $\mathcal{W}_4$ be Algorithm (a) from Figure 3. The number of misses encountered by $\mathcal{W}_4$ with a direct-mapped cache of size 4 with blocksize 1, is equal to

$$\begin{aligned} \mathbf{Misses}(\mathcal{W}_4, 0) &= 2\mathbf{Misses}(\mathcal{W}_3, 1) + 8\mathbf{Misses}(\mathcal{W}_1, 0) \\ &= 2(2\mathbf{Misses}(\mathcal{W}_2, 2) + 4\mathbf{Misses}(\mathcal{W}_1, 1)) + 8\mathbf{Misses}(\mathcal{W}_1, 0) \\ &= 2^2\mathbf{Misses}(\mathcal{W}_2, 2) + 2^3\mathbf{Misses}(\mathcal{W}_1, 1) + 2^3\mathbf{Misses}(\mathcal{W}_1, 0) \\ &= 3 \cdot 2^4 + 2^4 + 2^4 = 5 \cdot 2^4 = 80. \end{aligned}$$

Similarly, let $\mathcal{W}_4'$ be Algorithm (b) from Figure 3.

$$\begin{aligned} \mathbf{Misses}(\mathcal{W}_4', 0) &= 8\mathbf{Misses}(\mathcal{W}_1, 3) + 2\mathbf{Misses}(\mathcal{W}_3, 0) \\ &= 8\mathbf{Misses}(\mathcal{W}_1, 3) + 2(2^2\mathbf{Misses}(\mathcal{W}_1, 2) + 2\mathbf{Misses}(\mathcal{W}_2, 0)) \\ &= 2^3\mathbf{Misses}(\mathcal{W}_1, 3) + 2^3\mathbf{Misses}(\mathcal{W}_1, 2) + 2^2\mathbf{Misses}(\mathcal{W}_2, 0) \\ &= 3 \cdot 2^4 + 3 \cdot 2^4 + 2^4 = 7 \cdot 2^4 = 112. \end{aligned}$$

It is possible to obtain a closed form solution to this recurrence, and from the closed form solution it is easy to determine the possible values for cache misses. Let $C = 2^c$ be the cache size and $N = 2^n$ the size of the transform. To simplify the analysis, the algorithms considered are restricted to fully expanded partition trees (i.e. those trees whose leaf nodes are all equal to one) – the general analysis is similar but more complicated to write down.

Under this assumption, when $n \ge c$, the partition tree induces a composition (ordered partition) of c (see Andrews (1976) for basic properties of compositions). Starting at the right, include the values of the children until the sum is greater than or equal to c. If the sum is equal to c, a composition of c is obtained. Otherwise, the process is recursively applied to the last child selected until a sum exactly equal to c is obtained (this will always happen since the tree is fully expanded). For example, if the trees in Figure 3 are fully expanded so that the leaf node equal to two is further split into two children of size one, then the induced compositions of $c = 2$ for Tree (a) is [1,1] and Tree (b) is [2].

The formula in the following theorem is easily proven by induction.

Theorem 2 *Let $\mathcal{W}_n$ be a fully expanded WHT algorithm of size $N = 2^n$ and assume a direct-mapped cache of size $C = 2^c$ with blocksize one and $n > c$. The number of cache misses is equal to*

$$3(n-c)2^n + k2^n,$$

where k is the number of parts in the composition of c induced from the partition tree for $\mathcal{W}_n$.

This theorem implies that there are c different values for the number of misses, and the formula can be used to check the misses found by the simulator for the applicable examples in the previous section. For example, the number of misses for a direct-mapped cache of size $C = 2^2$ using the iterative and recursive algorithms is equal to $3(4-2)2^4 + 2 \cdot 2^4 = 128$ and $3(4-2)2^4 + 2^4 = 112$ respectively, and these are the only two possible values. In the next section, the distribution of k for random WHT algorithms is determined.

5 Distribution of Cache Misses

Let $T(n, c)$ denote the value of k in Theorem 2, which is equal to the number of parts in the composition of c induced by a partition tree of size n. For notational convenience we will consider the induced composition from the left instead of the right. In order to analyze the distribution of $T(n, c)$ we will use a bijection between compositions and strings of black and white dots of length n whose last dot is black. A detailed description of this bijection is given e.g. in Hitczenko and Louchard (2001); Hitczenko and Savage (2004) and its consequence is that a random composition can be identified with a sequence

$$(\Gamma_1, \Gamma_2, \dots, \Gamma_{\tau-1}, n - S_{\tau-1})$$

where $\Gamma_1, \Gamma_2 \dots$ are i.i.d. geometric r.v.s with parameter $1/2$, GEOM($1/2$) (that is $\mathsf{Pr}(\Gamma_1 = j) = 1/2^j, j = 1, 2 \dots,$), $S_k = \Gamma_1 + \dots + \Gamma_k$, $S_0 = 0$, and $\tau = \inf\{k \geq 1 : \ S_k \geq n\}$. It follows, in particular, that $\tau \stackrel{d}{=} 1 + \text{BIN}(n-1, 1/2)$, where BIN$(m, p)$ denotes a binomial random variable with parameters m an p and $\stackrel{d}{=}$ denotes equality in distribution.

To analyze $T(n, c)$ define $\tau_c = \inf\{k \geq 1 : \ S_k > c\} = \inf\{k \geq 1 : \ S_k \geq c + 1\}$. As is evident from the following picture

$$\begin{array}{ccccccccccc} & S_1 & & S_2 & & & & S_3 & \dots & S_{\tau_c-1} & \qquad\qquad c \qquad\qquad\quad S_{\tau_c} \\ \circ & \bullet & \circ & \bullet & \circ & \circ & \circ & \bullet & \dots & \bullet & \underbrace{\circ \quad \circ \quad \bullet \quad \circ \quad \circ \quad \bullet}_{\Gamma_{\tau_c}} \quad \circ \end{array}$$

$T(n, c)$ is the sum of the number of parts obtained at the first partitioning step, plus the number of parts obtained in the subsequent partitioning of a block of the size Γ_{τ_c}. To be precise, the size of this block is $\Gamma_{\tau_c} \wedge (n - S_{\tau_c - 1})$, but the second term is smaller, iff $S_{\tau_c} \geq n$. Since this happens with probability $1/2^{n-c}$ and, realistically, n is much larger than c by adopting this simplification we commit only exponentially small error; more precise analysis could be carried out with minimal changes, but would result in more complicated expressions. For the purpose of this note we will contend ourselves with this slightly simplified model. We have

$$T(n, c) = (\tau_c - 1) + T(\Gamma_{\tau_c}, c - S_{\tau_c - 1}).$$

This formula is also correct when $S_{\tau_c - 1} = c$ provided we set $T(k, 0) = \delta_{k,0}$, where $\delta_{k,j}$ is Kronecker's symbol. It follows from the above discussion that $\tau_c - 1$ is BIN$(c, 1/2)$. To analyze the second summand, for $j = 1, \dots, c$ write

$$\mathsf{Pr}(T(n, c) = j) = \sum_{m=0}^{j} \mathsf{Pr}(\tau_c - 1 = m, T(\Gamma_{\tau_c}, c - S_{\tau_c - 1}) = j - m), \qquad (2)$$

and then, for the term underneath summation

$$\begin{aligned} &\mathsf{Pr}(\tau_c - 1 = m, T(\Gamma_{\tau_c}, c - S_{\tau_c - 1}) = j - m) \\ &\qquad = \sum_{k=0}^{c} \mathsf{Pr}(S_m = k, \Gamma_{m+1} > c - k, T(\Gamma_{m+1}, c - k) = j - m) \end{aligned}$$

Conditioning on $\{S_m = k, \Gamma_{m+1} > c - k\}$, using independence of S_m and Γ_{m+1}, and memoryless property of geometric random variables, after calculation similar in spirit to those in (Hitczenko et al., 2003, Section 5.5) we obtain the following expression for the right hand side of (2)

$$\frac{1}{2^c} \sum_{m=0}^{j} \sum_{k=0}^{c} 2^k \mathsf{Pr}(S_m = k) \mathsf{Pr}(T(\Gamma + c - k, c - k) = j - m),$$

where $T(\Gamma+r,r)$ is to be understood as a compound distribution. For $k,m\ge 1$,

$$2^k\mathsf{Pr}(S_m=k)=2^k\sum_{\substack{i_1,\dots,i_m\ge 1\\ i_1+\dots+i_m=k}}\mathsf{Pr}\left(\bigcap_{\ell=1}^{m}\{\Gamma_\ell=i_\ell\}\right)=2^k\sum_{\substack{i_1,\dots,i_m\ge 1\\ i_1+\dots+i_m=k}}\frac{1}{2^{i_1+\dots+i_m}},$$

is the number of compositions of k into m parts. That number is $\binom{k-1}{m-1}$. Further, $\mathsf{Pr}(S_m=k)$ and $\mathsf{Pr}(T(\Gamma+c-k,c-k)=j-m)$ vanish unless $k\ge m$ and $c-k\ge j-m$, and if $m=0$ then the only nonvanishing term in the inner sum corresponds to $k=0$. Combining all of these observations with (2) we obtain

$$\begin{aligned}\mathsf{Pr}(T(n,c)=j) &= \frac{1}{2^c}\mathsf{Pr}(T(\Gamma+c,c)=j) \\ &+ \frac{1}{2^c}\sum_{m=1}^{j}\sum_{k=m}^{m+c-j}\binom{k-1}{m-1}\mathsf{Pr}(T(\Gamma+c-k,c-k)=j-m).\end{aligned} \tag{3}$$

We need to find the numbers $\mathsf{Pr}(T(\Gamma+r,r)=q)$, for $0\le r\le c$ and $0\le q\le r$. In fact, these are the only numbers we need to find because, up to an exponentially small (in $n-c$) error, we have $\mathsf{Pr}(T(n,r)=q)=\mathsf{Pr}(T(\Gamma+r,r)=q)$. This equality follows from (that is the place where we approximate)

$$\mathsf{Pr}(T(n,r)=q)=\mathsf{Pr}(T(S_{\tau_r},r)=q)+O(2^{c-n}),$$

and then conditioning on the value of S_{τ_r} and calculating that $\mathsf{Pr}(S_{\tau_r}=k+r)=2^{-k}$. Let $p_{j,r}:=\mathsf{Pr}(T(\Gamma+r,r)=j)$. Then, (3) reads as

$$\begin{aligned}p_{j,c} &= \frac{1}{2^c-1}\sum_{m=1}^{j}\sum_{k=m}^{m+c-j}\binom{k-1}{m-1}p_{j-m,c-k} \\ &= \frac{1}{2^c-1}\sum_{m=1}^{j}\sum_{k=0}^{c-j}\binom{m-1+k}{m-1}p_{j-m,c-m-k}\end{aligned} \tag{4}$$

for $j=1,\dots,c$. For any given c this system, together with the initial condition $p_{0,c}=\delta_{c,0}$ can be solved recursively. The solutions for the first five nontrivial values of c, rounded to three significant digits are given below. These values were compared to empirical values obtained from generating 10,000 random WHT algorithms of size 2^{10} and counting the number of misses for each algorithm using a cache of size $C=2^c$ for $c=2,\dots,6$. The first value listed is the theoretical result and the second is the empirical value.

$c\backslash j$	1	2	3	4	5	6	7	8
2	.333,.336	.667, .664						
3	.143,.145	.476, .473	.381,.382					
4	.0667,.0691	.298,.303	.432,.424	.203,.206				
5	.0323,.0316	.179,.178	.363,.361	.321,.326	.105,.103			
6	.0159,.0149	.104,.103	.269,.269	.342,.348	.215,.212	.0533,.0535		

Remark. Although the recurrence (4) does not look very inviting it can be used to gain an additional information about the variable $T(n,c)$. For example, one can show that

$$\mathsf{E}T(n,c)=\sum_{j=0}^{c-1}\frac{2^j}{2^{j+1}-1}.$$

We will not include the details, but only mention that after changing the order of summation in

$$\mathsf{E}T(n,c)=\sum_{j=1}^{c}jp_{j,c}=\frac{1}{2^c-1}\sum_{j=1}^{c}\sum_{r=0}^{j-1}\sum_{\ell=0}^{c-j}j\binom{c-\ell-r-1}{j-r-1}p_{r,r+\ell},$$

and further manipulations involving binomial coefficients identities and geometric summation one is led to recurrence

$$(2^c-1)\mathsf{E}T(n,c)=\sum_{k=0}^{c-1}2^{c-1}\frac{\mathsf{E}T(n,k)}{2^k}+c2^{c-1},\quad T(n,0)\equiv 0.$$

This immediately gives,

$$\mathsf{E}T(n,c) = \mathsf{E}T(n,c-1) + \frac{2^{c-1}}{2^c - 1} = \cdots = \sum_{j=0}^{c-1} \frac{2^j}{2^{j+1}-1},$$

as claimed.

References

A. Aggarval and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31:1116–1127, 1988.

R. Allen and K. Kennedy. *Optimizing Compilers for Modern Architectures*. Morgan Kaufmann Publishers, San Francisco, 2002.

G. E. Andrews. *The theory of partitions*. Addison–Wesley, Reading, MA, 1976.

K. Beauchamp. *Applications of Walsh and related functions*. Academic Press, 1984.

D. F. Elliott and K. R. Rao. *Fast Transforms: Algorithms, Analyses, Applications*. Academic Press, 1982.

M. Furis. Cache miss analysis of Walsh-Hadamard Transform algorithms. Master's thesis, Drexel University, 2003.

J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, third edition, 2002.

P. Hitczenko, J. R. Johnson, and H.-J. Huang. Distribution of a class of divide and conquer recurrences arizing from the computation of the Walsh–Hadamard transform. preprint, 2003.

P. Hitczenko, J. R. Johnson, and H.-J. Huang. Distribution of WHT recurrences. In *Mathematics and computer science. III*, Trends Math., pages 161–162. Birkhäuser, Basel, 2004.

P. Hitczenko and G. Louchard. Distinctness of compositions of an integer: a probabilistic analysis. *Random Struct. Alg.*, 19:407–437, 2001.

P. Hitczenko and C. D. Savage. On the multiplicity of parts in a random composition of a large integer. *SIAM J. Discrete Math.*, 18:418–435, 2004.

H.-J. Huang. Performance analysis of an adaptive algorithm for the Walsh-Hadamard transform. Master's thesis, Drexel University, 2002.

J. Johnson and M. Püschel. In Search for the Optimal Walsh-Hadamard Transform. In *Proceedings ICASSP*, volume IV, pages 3347–3350, 2000.

R. E. Ladner, J. D. Fix, and A. LaMarca. Cache performance analysis of traversals and random accesses. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, pages 613–622, 1999.

F. MacWilliams and N. Sloane. *The theory of error-correcting codes*. North-Holland Publ.Comp., 1992.

J. Moura, M. Püschel, J. Dongarra, and D. Padua, editors. *Proceedings of IEEE*, volume 93, Feb. 2005. Program Generation, Optimization, and Adaptation.

S. Sen, S. Chatterjee, and N. Dumir. Towards a theory of cache–efficient algorithms. *Journal of the ACM*, 49:828–858, 2002.