What is the WHT anyway, and why are there so many ways to compute it?

Jeremy Johnson

1, 2, 6, 24, 112, 568, 3032, 16768,…
Walsh-Hadamard Transform

- \( y = \text{WHT}_N x, \quad N = 2^n \)

\[
\text{WHT}_N = \text{WHT}_2 \otimes \cdots \otimes \text{WHT}_2 \\
\text{WHT}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\
\text{WHT}_4 = \text{WHT}_2 \otimes \text{WHT}_2
\]

\[
= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}
\]
WHT Algorithms

• Factor $\text{WHT}_N$ into a product of sparse structured matrices

• Compute: $y = (M_1 M_2 \ldots M_t)x$
  
  $y_t = M_t x$

  $\ldots$

  $y_2 = M_2 y_3$

  $y = M_1 y_2$
Factoring the WHT Matrix

- \( AC \otimes BD = (A \otimes B)(C \otimes D) \)
- \( A \otimes B = (A \otimes I)(I \otimes B) \)
- \( A \otimes (B \otimes C) = (A \otimes B) \otimes C \)
- \( I_m \otimes I_n = I_{mn} \)

\[
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & -1 & 1 & -1 \\
1 & 1 & -1 & -1 \\
1 & -1 & -1 & 1 \\
\end{bmatrix}
\]

\[
WHT_4 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & -1 & 0 \\
0 & 1 & 0 & -1
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 1 & 0 & 0 \\
1 & -1 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 0 & 1 & -1
\end{bmatrix}
\]

\[
WHT_2 \otimes WHT_2 = (WHT_2 \otimes I_2)(I_2 \otimes WHT_2)
\]
Recursive and Iterative Factorization

\[ WHT_8 = (WHT_2 \otimes I_4)(I_2 \otimes WHT_4) \]

\[ = (WHT_2 \otimes I_4)(I_2 \otimes ((WHT_2 \otimes I_2)(I_2 \otimes WHT_2))) \]

\[ = (WHT_2 \otimes I_4)(I_2 \otimes (WHT_2 \otimes I_2))(I_2 \otimes (I_2 \otimes WHT_2)) \]

\[ = (WHT_2 \otimes I_4)(I_2 \otimes (WHT_2 \otimes I_2))((I_2 \otimes I_2) \otimes WHT_2) \]

\[ = (WHT_2 \otimes I_4)(I_2 \otimes WHT_2 \otimes I_2)((I_2 \otimes I_2) \otimes WHT_2) \]

\[ = (WHT_2 \otimes I_4)(I_2 \otimes WHT_2 \otimes I_2)(I_4 \otimes WHT_2) \]
Recursive Algorithm

\[
WHT_8 = (WHT_2 \otimes I_4)(I_2 \otimes (WHT_2 \otimes I_2)(I_2 \otimes WHT_2))
\]
Iterative Algorithm

\[
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\
1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\
1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 \\
1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\
1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\
1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\
\end{bmatrix} = \\
\begin{bmatrix}
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
\end{bmatrix} \\
\begin{bmatrix}
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
\end{bmatrix} \\
\begin{bmatrix}
1 & 1 \\
1 & -1 \\
1 & 1 \\
1 & -1 \\
1 & 1 \\
1 & -1 \\
1 & 1 \\
1 & -1 \\
\end{bmatrix} \\
\begin{bmatrix}
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
\end{bmatrix}
\]

\[WHT_8 = (WHT_2 \otimes I_4)(I_2 \otimes WHT_2 \otimes I_2)(I_4 \otimes WHT_2)\]
WHT Algorithms

• Recursive

\[ WHT_N = (WHT_2 \otimes I_{N/2})(I_2 \otimes WHT_{N/2}) \]

• Iterative

\[ WHT_N = \prod_{i=1}^{n} (I_{2^{-1}} \otimes WHT_2 \otimes I_2^{-n-i}) \]

• General

\[ WHT_2^n = \prod_{i=1}^{t} (I_{2^{n_1+n_t-n-1}} \otimes WHT_2^{n_i} \otimes I_2^{n_{i+1}+\cdots+n_t}) \]

where \( n = n_1 + \cdots + n_t \)
WHT Implementation

• Definition/formula
  - \( N = N_1 \times N_2 \times \ldots \times N_t \) \( N_i = 2^{n_i} \)
  - \( x = WHT_N^t x \quad x_{b,s}^M = (x(b), x(b+s), \ldots, x(b+(M-1)s)) \)

• Implementation (nested loop)
  \( R = \frac{N}{S}; \quad S = 1; \)
  for \( i = t, \ldots, 1 \)
  \( R = \frac{R}{N_i} \)
  \( WHT_{2^n} = \prod_{i=1}^{t} \left( I_2^{n_1+\ldots+n_{i-1}} \bigotimes WHT_{2^{n_i}} \bigotimes I_2^{n_{i+1}+\ldots+n_t} \right) \)
  for \( j = 0, \ldots, R-1 \)
  for \( k = 0, \ldots, S-1 \)

  \[ x_{jN_iS+k,S}^{N_i} = WHT_{N_i} \cdot x_{jN_iS+k,S}^{N_i} \]

  \( S = S \times N_i; \)
Partition Trees

Left Recursive

Right Recursive

Balanced

Iterative
Ordered Partitions

- There is a 1-1 mapping from ordered partitions of n onto (n-1)-bit binary numbers.

$\Rightarrow$ There are $2^{n-1}$ ordered partitions of n.

$162 = 1\, 0\, 1\, 0\, 0\, 0\, 1\, 0$

$1|1\, 1|1\, 1\, 1\, 1|1\, 1 \rightarrow 1+2+4+2 = 9$
Enumerating Partition Trees

00
3

01
3
2
1

01
3
2
1
1
1

10
3
1
2

10
3
1
2
1
1

11
3
1
1
1
Search Space

- Optimization of the WHT becomes a search, over the space of partition trees, for the fastest algorithm.
- The number of trees:

\[ T_n = 1 + \sum_{n_1 + \ldots + n_t = n} T_{n_1} \cdots T_{n_t} \]

<table>
<thead>
<tr>
<th>n</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_n</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>24</td>
<td>112</td>
<td>568</td>
<td>3672</td>
<td>16768</td>
</tr>
</tbody>
</table>
Size of Search Space

• Let $T(z)$ be the generating function for $T_n$

$$T(z) = z/(1-z) + T(z)^2/(1-T(z))$$

$$T_n = \Theta(\alpha^n/n^{3/2}), \text{ where } \alpha = 4 + \sqrt{8} \approx 6.8284$$

• Restricting to binary trees

$$B(z) = z/(1-z) + B(z)^2$$

$$T_n = \Theta(5^n/n^{3/2})$$
WHT Package
Püschel & Johnson (ICASSP ’00)

- Allows easy implementation of any of the possible WHT algorithms
- Partition tree representation
  \[ W(n) = \text{small}[n] \mid \text{split}[W(n_1), \ldots, W(n_t)] \]
- Tools
  - Measure runtime of any algorithm
  - Measure hardware events
  - Search for good implementation
    - Dynamic programming
    - Evolutionary algorithm
Histogram (n = 16, 10,000 samples)

- Wide range in performance despite equal number of arithmetic operations (n^{2n} flops)
- Pentium III consumes more run time (more pipeline stages)
- Ultra SPARC II spans a larger range
Theorem. Let $W_N$ be a WHT algorithm of size $N$. Then the number of floating point operations (flops) used by $W_N$ is $N \log(N)$.

Proof. By induction.

\[
\text{flops}(W_N) = \sum_{i=1}^{t} 2^{n-n_i} \text{flops}(W_{N_i})
\]

\[
= \sum_{i=1}^{t} 2^{n-n_i} n_i 2^{n_i} = 2^n \sum_{i=1}^{t} n_i = n 2^n
\]
Instruction Count Model

\[ IC(n) = \alpha A(n) + \sum_{i=1}^{3} \beta_i L_i(n) + \sum_{l=1}^{3} \alpha_l A_l(n) \]

\( A(n) = \) number of calls to WHT procedure
\( \alpha = \) number of instructions outside loops
\( A_l(n) = \) Number of calls to base case of size \( l \)
\( \alpha_l = \) number of instructions in base case of size \( l \)

\( L_i = \) number of iterations of outer (i=1), middle (i=2), and outer (i=3) loop
\( \beta_i = \) number of instructions in outer (i=1), middle (i=2), and outer (i=3) loop body
apply_small1:
    movl 8(%esp),%edx // load stride S to EDX
    movl 12(%esp),%eax // load x array's base address to EAX
   fldl (%eax) // st(0)=R7=x[0]
    fldl (%eax,%edx,8) // st(0)=R6=x[S]
    fld %st(1) // st(0)=R5=x[0]
    fadd %st(1),%st // R5=x[0]+x[S]
    fxch %st(2) // st(0)=R5=x[0],s(2)=R7=x[0]+x[S]
    fsubp %st,%st(1) // st(0)=R6=x[S]-x[0]
    fxch %st(1) // st(0)=R6=x[0]+x[S], s(1)=R7=x[S]-x[0]
    fstpl (%eax) // store x[0]=x[0]+x[S]
    fstpl (%eax,%edx,8) // store x[0]=x[0]-x[S]
    ret
Recurrences

\[ A(n) = 1 + \sum_{i=1}^{t} 2^{n-n_i} A(n_i), \quad n = n_1 + \cdots + n_t \]

\[ A(n) = 0, \quad n \text{ a leaf} \]

\[ A_l(n) = \nu_l 2^{n-l}, \quad \text{where } \nu_l = \text{number of leaves } = l \]
Recurrences

\[ L_1(n) = t + \sum_{i=1}^{t} 2^{n-n_i} L_1(n_i), \quad n = n_1 + \cdots + n_t \]

\[ L_2(n) = \sum_{i=1}^{t} 2^{n-n_i} L_2(n_i) + 2^{n_1+\cdots+n_{i-1}}, \quad n = n_1 + \cdots + n_t \]

\[ L_3(n) = \sum_{i=1}^{t} 2^{n-n_i} L_2(n_i) + 2^{n-n_i}, \quad n = n_1 + \cdots + n_t \]

\[ L_i(n) = 0, \quad n \text{ a leaf} \]
Histogram using Instruction Model (P3)

\[ \alpha_1 = 12, \alpha_1 = 34, \text{ and } \alpha_1 = 106 \]
\[ \alpha = 27 \]
\[ \beta_1 = 18, \beta_2 = 18, \text{ and } \beta_1 = 20 \]
Algorithm Comparison

Recursive/Iterative Runtime

Rec & Bal/It Instruction Count

Rec& It/Best Runtime

Small/It Runtime
Dynamic Programming

\[
\min_{n_1 + \ldots + n_t = n} \text{Cost}\left( \frac{n}{T_{n_1}, \ldots, T_{n_t}} \right),
\]

where \( T_n \) is the optimal tree of size \( n \).

This depends on the assumption that \( \text{Cost} \) only depends on the size of a tree and not where it is located. (true for IC, but false for runtime).

For IC, the optimal tree is iterative with appropriate leaves. For runtime, DP is a good heuristic (used with binary trees).
Optimal Formulas

Pentium

[1], [2], [3], [4], [5], [6]

[[4], [4]]

[[5], [4]]

[[5], [5]]

[[5], [6]]

[[2], [[5], [5]]]

[[2], [[5], [6]]]

[[2], [[2], [[5], [5]]]]

[[2], [[2], [[5], [6]]]]

[[2], [[2], [[5], [5]]]]

[[2], [[2], [[2], [[5], [5]]]]]

UltraSPARC

[1], [2], [3], [4], [5], [6]

[[3], [4]]

[[4], [4]]

[[4], [5]]

[[5], [5]]

[[5], [6]]

[[4], [[4], [4]]]

[[4], [[5]], [4]]

[[4], [[5]], [5]]

[[5], [5]], [5]]

[[5], [5]], [6]]

[[4], [[4], [5]], [4]]]

[[4], [[4], [5]], [5]]]

[[4], [[5], [5]], [5]]]

[[5], [[5], [5]], [5]]]

[[5], [[5], [5]], [5]]]

[[5], [[5], [5]], [5]]]

[[5], [[5], [5]], [5]]]

[[5], [[5], [5]], [5]]]
Different Strides

- Dynamic programming assumption is not true. Execution time depends on stride.