Performance Analysis of Divide and Conquer Algorithms for the WHT

CS 540 High Performance Computing
Jeremy Johnson
Mihai Furis, Pawel Hitczenko, Hung-Jen Huang
Dept. of Computer Science
Drexel University

www.spiral.net
Motivation

• On modern machines operation count is not always the most important performance metric.

• Effective utilization of the memory hierarchy, pipelining, and Instruction Level Parallelism is important.

• **Automatic Performance Tuning and Architecture Adaptation**
  - Generate and Test
  - FFT, Matrix Multiplication, …

• Explain performance distribution
Outline

- WHT Algorithms
- WHT Package and Performance Distribution
- Performance Model
  - Instruction Count
  - Cache
WHT Algorithms

• Recursive

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

• Iterative

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

• General

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

where \[ n = n_1 + \cdots + n_t \]
WHT Implementation

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

- Implementation (nested loop)
  \[ R = N; \ S = 1; \]
  \[ \text{for } i = t, \ldots, 1 \]
  \[ R = R/N_i \]
  \[ \text{for } j = 0, \ldots, R - 1 \]
  \[ \text{for } k = 0, \ldots, S - 1 \]
  \[ x_{jN_iS+k,S}^{N_i} = WHT_{N_i} \times x_{jN_iS+k,S}^{N_i} \]
  \[ S = S \times N_i; \]

\[
WHT_{2^n} = \prod_{i=1}^{t} \left( I_{2^{n_1}} \times \ldots \times I_{2^{n_{i-1}}} \times WHT_{2^{n_i}} \times I_{2^{n_{i+1}}} \times \ldots \times I_{2^{n_t}} \right)
\]
Partition Trees

Left Recursive

Right Recursive

Iterative

Balanced
Ordered Partitions

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

\[ \Rightarrow \text{There are } 2^{n-1} \text{ ordered partitions of } n. \]

\[ 162 = 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0 \]

\[ 1|1\ 1|1\ 1\ 1\ 1|1\ 1 \rightarrow 1+2+4+2 = 9 \]
Outline

• WHT Algorithms

• WHT Package and Performance Distribution

• Performance Model
  – Instruction Count
  – Cache
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 (coupled with PCL)
  - Search for good implementation
    - Dynamic programming
    - Evolutionary algorithm
Algorithm Comparison

Recursive/Iterative Runtime

Rec & Bal/It Instruction Count

Rec & It/Best Runtime

Small/It Runtime
Cache Miss Data

Recursive vs. Iterative Normalized to Best

Recursive vs. Iterative

Iterative vs. Best

Recursive vs. Best

- Recursive Time
- Iterative Time
- Instructions
- L1 Data Cache Misses
- L2 Data Cache Misses

- Instructions
- L1 Data Cache Misses
- L2 Data Cache Misses

size
Wide range in performance despite equal number of arithmetic operations ($n2^n$ flops)

- Pentium III consumes more run time (more pipeline stages)
- Ultra SPARC II spans a larger range
Dynamic Programming

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

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 – stride, state of the cache).
Outline

• WHT Algorithms

• WHT Package and Performance Distribution

• Performance Model
  – Instruction Count
  – Cache
WHT Implementation

\[ N = N_1 \times N_2 \ldots N_t \quad N_i = 2^{n_i} \]

\[ x = WHT_N \cdot x_{b,s}^M = (x(b), x(b+s), \ldots, x(b+(M-1)s)) \]

- **Implementation (nested loop)**

  \[
  R = N; \quad S = 1; \\
  \text{for } i = t, \ldots, 1 \\
  \quad R = R / N_i \\
  \quad \text{for } j = 0, \ldots, R-1 \\
  \quad \quad \text{for } k = 0, \ldots, S-1 \\
  \quad \quad \quad x_{jN_iS+k,S}^{N_i} = WHT_{N_i} \cdot x_{jN_iS+k,S}^{N_i} \\
  \quad S = S \times N_i; \\
  \]

\[
WHT_{2^n} = \prod_{i=1}^{t} \left( I_{2^{n_1+\ldots+n_{i-1}}} \otimes WHT_{2^n_i} \otimes I_{2^{n_{i+1}+\ldots+n_t}} \right)
\]
Operation Count

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.

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

\[
= \sum_{i=1}^{t} 2^{n-n_i} n_i 2^{n_i} = 2^n \sum_{i=1}^{t} n_i = n \cdot 2^n
\]
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\( I \)(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
Small[1]

    .file    "s_1.c"
    .version "01.01"

gcc2_compiled.:  
.text
    .align 4
.apply_small1
    .globl apply_small1
    .type    apply_small1,@function

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],st(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

\[ \mathbb{L}_1(n) = t + \sum_{i=1}^{t} 2^{n-n_i} \mathbb{L}_1(n_i), \quad n = n_1 + \cdots + n_t \]

\[ \mathbb{L}_2(n) = \sum_{i=1}^{t} 2^{n-n_i} \mathbb{L}_2(n_i) + 2^{n_1+\cdots+n_{i-1}}, \quad n = n_1 + \cdots + n_t \]

\[ \mathbb{L}_3(n) = \sum_{i=1}^{t} 2^{n-n_i} \mathbb{L}_2(n_i) + 2^{n-n_i}, \quad n = n_1 + \cdots + n_t \]

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

\[ \alpha_1 = 12, \quad \alpha_1 = 34, \quad \text{and} \quad \alpha_1 = 106 \]
\[ \alpha = 27 \]
\[ \beta_1 = 18, \quad \beta_2 = 18, \quad \text{and} \quad \beta_1 = 20 \]
Cache Model

• Different WHT algorithms access data in different patterns
• All algorithms with the same set of leaf nodes have the same number of memory accesses

• Count misses for accesses to data array
  – Parameterized by cache size, associativity, and block size
  – simulate using program traces (restrict to data vector accesses)
  – Analytic formula?
Blocked Access

\[ WHT_{16} = (WHT_2 \otimes I_8)(I_2 \otimes WHT_8) = \]
\[ = (WHT_2 \otimes I_8)(I_2 \otimes (WHT_2 \otimes I_4)(I_2 \otimes WHT_4)) \]
Interleaved Access

$$WHT_{16} = (WHT_8 \otimes I_2)(I_8 \otimes WHT_2) =
((WHT_4 \otimes I_2)(I_4 \otimes WHT_2) \otimes I_2)(I_8 \otimes WHT_2)$$
Cache Simulator

\[ PrintTree(W); \]
\[ T := TraceWHTRW(W,0,1); \]
\[ T := [0, 1, 0, 1, 2, 3, 2, 3, 0, 1, 2, 3, 4, 5, 4, 5, 6, 7, 6, 7, 4, 5, 6, 7, 0, 4, 0, 4, 0, 4, 1, 5, 1, 5, 1, 5, 2, 6, 2, 6, 2, 6, 3, 7, 3, 7, 8, 9, 8, 9, 10, 11, 10, 11, 8, 9, 10, 11, 12, 13, 12, 13, 14, 15, 14, 15, 12, 13, 14, 15, 8, 12, 8, 12, 8, 12, 9, 13, 9, 13, 9, 13, 10, 14, 10, 14, 10, 14, 11, 15, 11, 15, 11, 15, 15, 0, 8, 0, 8, 0, 8, 1, 9, 1, 9, 1, 9, 2, 10, 2, 10, 2, 10, 3, 11, 3, 11, 3, 11, 4, 12, 4, 12, 4, 12, 5, 13, 5, 13, 5, 13, 6, 14, 6, 14, 6, 14, 7, 15, 7, 15, 7, 15] \]
\[ nops(T); \]
144
Cache Simulator

> CacheSim(T,4,1,1,false);
  112
> CacheSim(T,4,4,1,false);
  48
> CacheSim(T,4,1,2,false);
  88
Cache Simulator

- 144 memory accesses
- C = 4, A = 1, B = 1  (80, 112)
- C = 4, A = 4, B = 1  (48, 48)
- C = 4, A = 1, B = 2  (72, 88)

- Iterative vs. Recursive (192 memory accesses)
- C = 4, A = 1, B = 1  (128, 112)
Cache Misses as a Function of Cache Size

$C = 2^2$

$C = 2^3$

$C = 2^4$

$C = 2^5$
Formula for Cache Misses

- \( M(L, W_N, R) = \) Number of misses for \((I_L \otimes \text{WHT}_N \otimes I_R)\)

\[
M(2^l, W_N, 2^r) = N \quad \text{if} \quad N \leq \left\lfloor \frac{C}{2^r} \right\rfloor
\]

\[= 3N \quad \text{if} \quad W_N \text{ is a leaf}\]

\[= \sum_{i=1}^{t} 2^{n-i} M(l+n_1+\ldots+n_{i-1}, W_{N_i}, r+n_i+\ldots+n_t)\]
Closed Form

- $M(0,W_n,0) = 3(n-c)2^n + k2^n$
- $C = 2^c$, $k =$ number of parts in the rightmost $c$ positions
- $c = 3$, $n = 4$

Iterative

Balanced

Right Recursive

$k = 1$

$k = 2$

$k = 3$
Performance and Speedup Models

• Linear combination of IC and Misses
  – Coefficients determined using linear regression

• Separate instruction classes for SIMD
  – SIMD Add, SIMD Ld, SIMD Shuffle, Other

• Estimate speedup using instruction counts
  – Estimate SIMD performance from speedup estimate

Predicted vs Actual Speedup

\[ \omega = \begin{bmatrix} \frac{1}{3 \cdot \nu}, \frac{3}{\nu}, \frac{9}{2}, 1 \end{bmatrix} \]
Performance Model Results

\[ \omega = \left[ \frac{1}{3 \cdot v}, \frac{3}{v}, \frac{9}{2}, 1 \right] \]

\[ \alpha = 0.6 \]

\[ \beta = 3.0 \]

- 100 Random WHTs (Sizes 9 to 19) Training Data
- 1,000 Random WHTs (Sizes 9 to 19) Evaluation Data
- Runtimes normalized by n log(n)
Error Distribution for Model

\[ \omega = \begin{bmatrix} \frac{1}{3 \cdot \nu}, \frac{3}{\nu}, \frac{9}{2}, 1 \end{bmatrix} \]

\( \alpha = 0.6 \)

\( \beta = 3.0 \)
## Model Classification

<table>
<thead>
<tr>
<th>Percentile of Best</th>
<th>True Positives</th>
<th>True Negatives</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model 1</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>73.00%</td>
<td>99.00%</td>
</tr>
<tr>
<td>5</td>
<td>83.00%</td>
<td>99.00%</td>
</tr>
<tr>
<td>10</td>
<td>85.00%</td>
<td>99.00%</td>
</tr>
<tr>
<td>15</td>
<td>88.00%</td>
<td>99.00%</td>
</tr>
<tr>
<td>20</td>
<td>91.00%</td>
<td>98.00%</td>
</tr>
<tr>
<td>25</td>
<td>92.00%</td>
<td>98.00%</td>
</tr>
<tr>
<td>50</td>
<td>95.00%</td>
<td>95.00%</td>
</tr>
<tr>
<td><strong>Model 2</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>68.00%</td>
<td>99.00%</td>
</tr>
<tr>
<td>5</td>
<td>78.00%</td>
<td>99.00%</td>
</tr>
<tr>
<td>10</td>
<td>81.00%</td>
<td>98.00%</td>
</tr>
<tr>
<td>15</td>
<td>86.00%</td>
<td>98.00%</td>
</tr>
<tr>
<td>20</td>
<td>90.00%</td>
<td>98.00%</td>
</tr>
<tr>
<td>25</td>
<td>92.00%</td>
<td>97.00%</td>
</tr>
<tr>
<td>50</td>
<td>94.00%</td>
<td>95.00%</td>
</tr>
</tbody>
</table>
Summary of Results and Future Work

- **Instruction Count Model**
  - min, max, expected value, variance, limiting distribution

- **Cache Model**
  - Direct mapped (closed form solution, distribution, expected value, and variance)

- Combine models
- Extend cache formula to include A and B
- Use as heuristic to limit search and predict performance