Performance Analysis of a Family of WHT Algorithms

Michael Andrews and Jeremy Johnson
Department of Computer Science
Drexel University
Philadelphia, PA USA

January 22, 2007

Abstract

This paper explores the correlation of instruction counts and cache misses to runtime performance for a large family of divide and conquer algorithms to compute the Walsh–Hadamard transform (WHT). Previous work showed how to compute instruction counts and cache misses from a high–level description of the algorithm and proved theoretical results about their minimum, maximum, mean, and distribution. While the models themselves do not accurately predict performance, it is shown that they are statistically correlated to performance and thus can be used to prune the search space for fast implementations. When the size of the transform fits in cache the instruction count itself is used; however, when the transform no longer fits in cache, a linear combination of instruction counts and cache misses is used. Thus for small transforms it is safe to ignore algorithms which have a high instruction count and for large transforms it is safe to ignore algorithms with a high value in the combined instruction count/cache miss model. Since the models can be computed from a high–level description of the algorithms, they can be obtained without runtime measurement and the previous theoretical results on the models can be applied to limit empirical search.

1 Introduction

A commonly used technique in the new field of automated performance tuning is called generate and test [1, 9]. This technique empirically finds a fast implementation, on a given computer, by generating a large number of alternative algorithms and implementation choices and searches amongst these alternatives for the algorithm/implementation with the smallest runtime. Intelligent search techniques are employed in order to avoid exhaustively generating all possibilities, which is usually infeasible. Since exhaustive search is not used, this approach does not guarantee that an optimal solution will be found. Nonetheless, the approach has been very successfully employed on a collection of important problems including matrix multiplication [3], the Fast Fourier transform (FFT) [4], and more general signal and image processing transforms [10].

Empirical runtimes are typically used in the search process due to the difficulty of accurately predicting performance on modern architectures. Despite these difficulties some initial success has been obtained using a combination of micro benchmarks and performance analysis in selecting high-performance matrix multiplication kernels [13]. Additional success has been obtained through the use of machine learning in selecting FFT algorithms and algorithms for the related Walsh-Hadamard Transform (WHT) [12, 11].

This paper explores the performance models presented in [5, 8] for the family of WHT algorithms presented in [7]. The package in [7] uses the generate and test technique to search for fast implementations of the WHT. The instruction count and cache miss models in [5, 8] were introduced to better understand the search space. The models do not accurately pre-

dict performance; however, they can be computed from the high–level description of the WHT algorithms used in the WHT package. Moreover, they are amenable to mathematical analysis. In particular, it is possible to theoretically compute the minimum and maximum number of instructions along with the mean and variance. It was also proven that the distribution in the number of instructions, over the space of algorithms, approaches a normal distribution. Similar results, subject to the constraint of a direct mapped cache, were obtained for cache misses, though the limiting distribution is different.

This paper presents an empirical study of the relationship of these models, and a linear combination of them, to actual performance. The results of our study show that even though the models do not accurately predict performance, there is a strong correlation. For small transform sizes, instruction count correlates fairly well with runtime, and best algorithms can be found, with high probability, by restricting the search to algorithms with small instruction count. For larger transform sizes, where cache comes into play, the correlation with instruction count is not as strong; however, if cache misses are taken into account, then a stronger correlation can be obtained. A performance model using a linear combination of instruction count and cache misses obtains nearly as strong a correlation for large sizes as instruction count alone for small sizes. The coefficients of the linear combination are chosen to maximize the correlation coefficient.

2 WHT Algorithm Space

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. Algorithms for computing the WHT can be derived by factoring the WHT matrix into a product of sparse structured matrices [7]. 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 factorization leads to a triply nested loop. Let $N = N_1 \cdots N_t$, where $N_i = 2^{n_i}$, and let $x^M_{b,s}$ denote the vector $(x(b), x(b+s), \ldots, x(b+(M-1)s))$. Then evaluation of $\mathbf{WHT}_N \cdot x$ using Equation 1 is performed using

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

The computation of $\mathbf{WHT}_{N_i}$ is computed recursively in a similar fashion until a base case of the recursion is encountered. Small WHT transforms 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. Alternative algorithms are obtained through different sequences of the application of Equation 1. In [5] it is shown that there are approximately $O(7^n)$ different algorithms (see the paper for more precise results). When $n_1 = \cdots = n_t = 1$ the algorithm is called *iterative* since there are no recursive calls. When $t = 2$ and $n_1 = 1$ and $n_2 = n-1$ the resulting algorithm is called *right recursive*, and when $n_1 = n-1$ and $n_2 = 1$ the algorithm is called *left recursive*. The right recursive and iterative algorithms are the two standard approaches to computing the WHT and they correspond the standard recursive and radix 2 iterative FFT algorithms.

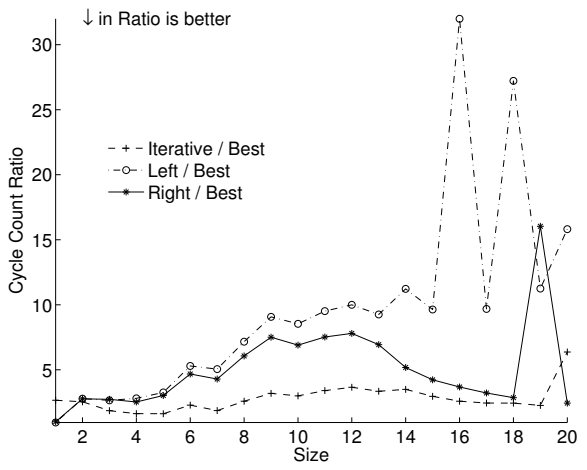

Figure 1: Ratio of Performance for canonical algorithms to Performance for best algorithms.

3 Performance Data and Models

Using performance counter measurements, it was shown in [6] that the two most important performance metrics were instruction count and data cache misses. Neither metric by itself can distinguish performance as, for example, the iterative algorithm executes fewer instructions than the right recursive algorithm for all sizes, yet the performance of the recursive algorithm, while initially worse, becomes better than the iterative algorithm as the size crosses cache boundaries. Such behavior is consistent with the observation that the recursive algorithm has fewer cache misses. This suggests a model which is a linear combination of instruction counts and cache misses. While such a model does correlate to actual performance, there is variance due to other factors such as the number of register spills, pipeline performance, functional unit utilization and other factors – it is simply not possible to accurately predict performance using this model. Nonetheless, the model does correlate with performance and can be used to prune the search space.

In this section, we measure cycle counts, instruction counts, and cache misses for a random sample of WHT algorithms of sizes 2^9 and 2^{18}. Experiments were performed on an Opteron (Model 224)– a single core 64 bit processor running at 1.8 GHz with a 64 Kb 2–way set associative L1 cache and a 1Mb 16–way set associative L2 cache. All performance measurements were taken with PAPI 1.3.2 [2]. All code was compiled with gcc version 3.4.4, using the optimization flags `-march=opteron -m64 -O2 -fomit-frame-pointer -fstrict-aliasing`. The sizes of WHT transforms were chosen so that one sample fit in L1 cache, while the other did not fit in L1 cache but did fit in L2 cache. Histograms showing the distribution of runtimes, instruction counts, and cache misses (for the larger size) are shown.

Additional measurements were made of the canonical algorithms (iterative, left recursive, and right recursive) discussed in the previous section and the best algorithm determined by the dynamic programming search performed by the WHT package in [7] (note that dynamic programming serves only as a heuristic since the optimal algorithm depends on the calling context). The best algorithm utilizes larger base cases (unrolled code) than used by the canonical algorithms. These measurements confirm the observations from [6] and show how much improvement can be obtained over the canonical algorithms.

Figure 1 shows the ratio of cycle counts of the canonical algorithms to the best algorithm for sizes 2^n with $n = 1, \ldots, 20$. As indicated, we expect that the iterative algorithm will outperform the recursive algorithms until a critical point, at which recursive algorithms will outperform the iterative algorithm. Furthermore, the analysis of the instruction count model in [5] suggests that the right recursive algorithm will outperform the left recursive algorithm as is the case. On the Opteron, the cross over occurs at the L2 cache boundary ($n = 18$).

Figures 2 and 3 show the ratio of instruction counts and cache misses of the canonical algorithms to the best algorithms. These figures show the relationship between instruction count, cache misses, and performance in transforms of large size. The iterative algorithm is closest to the best algorithm. Additionally it has the lowest instruction count for all sizes, and the fewest cache misses until the L1 cache boundary at $n = 2^{14}$. Despite more cache misses, the iterative

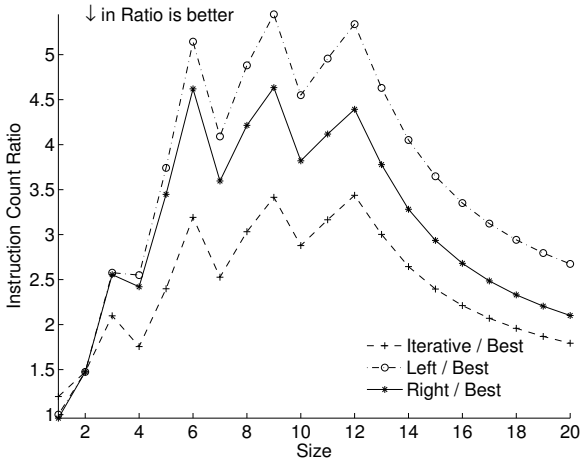

Figure 2: Ratio of Instruction Counts for canonical algorithms to Instruction Counts for best algorithms.

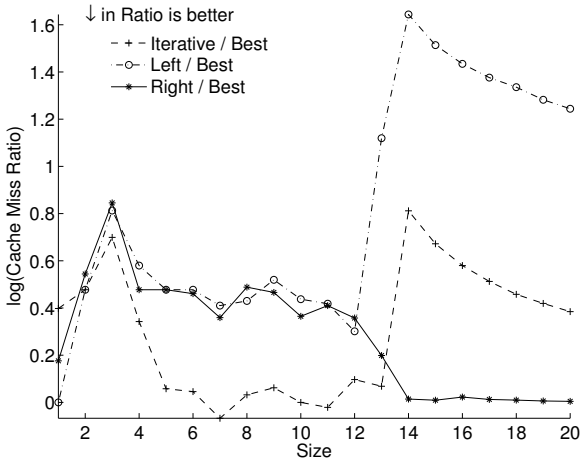

Figure 3: Ratio of Cache Miss Counts for canonical algorithms to Cache Miss Counts for best algorithms.

algorithm has performance closest to the best until $n = 2^{20}$.

Figure 4 shows histograms of cycle counts and instruction counts collected into 50 equally sized bins for 10,000 random samples of size 2^9. The random sample was obtained using a recursive split uniform distribution. That is, each time Equation 1 is applied we assume every composition $n = n_1 + \cdots + n_t$ is equally likely to occur (see [5]). The samples were filtered for extreme outliers beyond the "outer fences", i.e. we expect that valid data will lie within a range based on the interquartile range (IQR), specifically:

$$3.0 \times IQR(X) - Q_1 < X < Q_3 + 3.0 \times IQR(X).$$

Where Q_1 is the first quartile, and Q_3 is the third quartile.

Similar histograms, including cache miss counts, are provided in Figure 5 for 10,000 random samples of size 2^{18}.

Our expectation was that for the smaller transform sizes the correlation between performance and instruction count would be significantly higher then for the larger transform size. Also we expected that there would be significant correlation between performance and the number of cache misses in the larger size. This can be seen intuitively by observing the similarity in counts for cycles and instructions in the smaller transform sizes (Fig. 4). For the larger transforms, there appears to be a slight left skew in the performance histogram where there is none in the instruction histogram. Intuitively, this skew can be accounted for in the left skew of the L1 cache miss histogram (Fig. 5). These correlations are quantitatively determined in the next section.

4 Correlation of Performance Data and Models

To gain a more quantitative understanding of the relationship between instruction count, cache misses and performance, we computed the correlation between our observations using the Pearson correlation coefficient. In every situation we expect the quantities to be positively correlated, for instance a higher

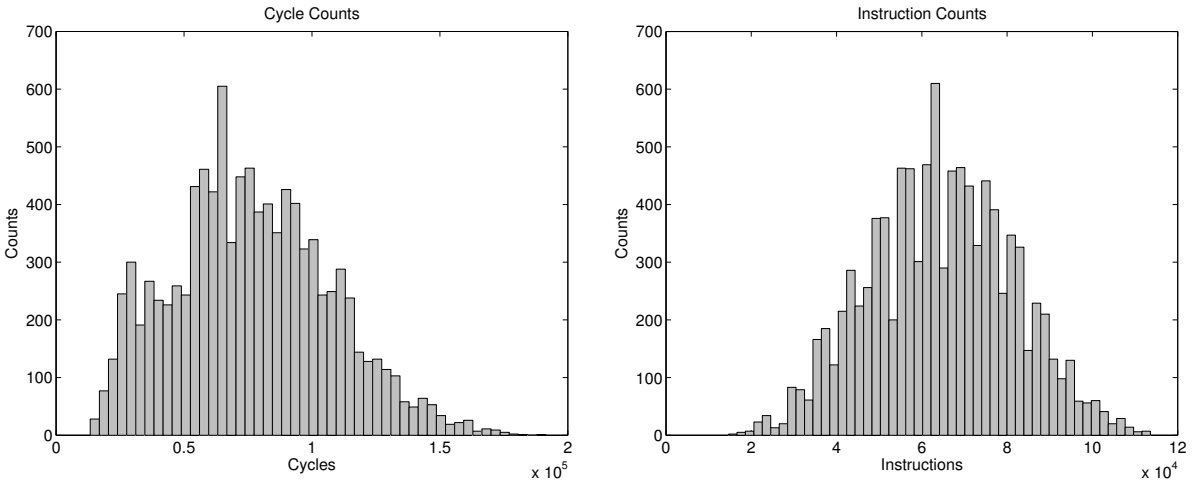

Figure 4: Cycle and Instruction counts collected into 50 bins for WHT_9.

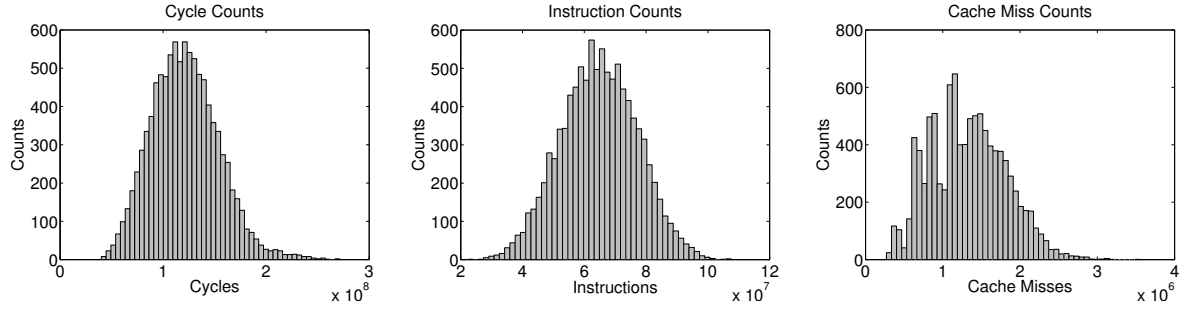

Figure 5: Cycle, Instruction, and Cache Miss counts collected into 50 bins for WHT_{18}.

instruction count should generally imply a higher cycle count. For small transform sizes whose input fits in the L1 cache, the correlation between instruction count and cycle count is very high, however, for the larger size, which does not fit in L1 cache, the correlation drops. Figure 6 shows scatter plots, with computed correlation coefficients, illustrating the dependency between performance and instruction count for the sample of WHT algorithms of size $n = 9$. Figures 7 and 8 show scatter plots and their respective correlation coefficients, illustrating the dependency between performance, instruction count, and cache misses for the sample of WHT algorithms with size $n = 18$. Included in the scatter plots are points indicating the canonical and best algorithms. The banding that occurs in Figure 6 results from similar banding in the number of load instructions.

For the larger transform size a model including both instruction count and cache misses is needed in order to obtain stronger correlation. The model is of the form $\alpha I + \beta M$, where I is the instruction count and M is the number of misses. The coefficients α and β were chosen in order to maximize the correlation. Figure 9 shows the correlation coefficient as a function of α and β where $0 \leq \alpha, \beta \leq 1$ are sampled uniformly in increments of 0.05. The optimal value, over this grid, occurs when $\alpha = 1.00$ and $\beta = 0.05$. Using this linear combination, the correlation coef-

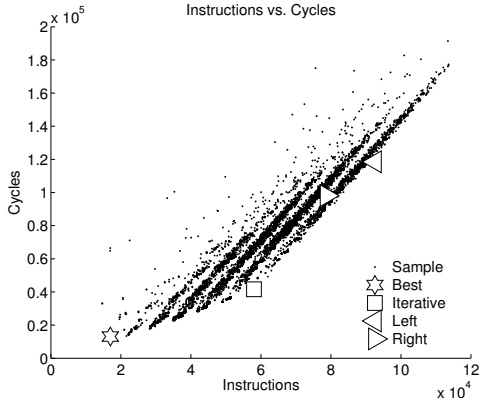

Figure 6: Instructions vs. Cycles for WHT_9 (correlation coefficient $\rho = 0.96$).

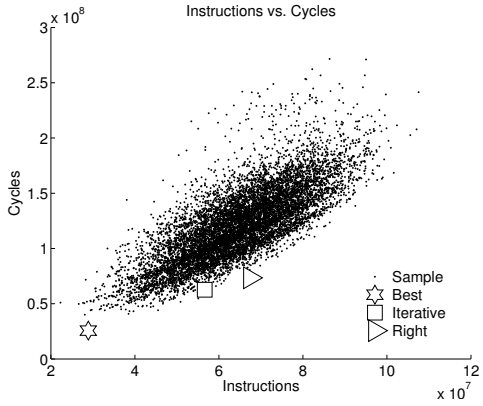

Figure 7: Instructions vs. Cycles for WHT_{18} (correlation coefficient $\rho = 0.77$, left recursive algorithm outside range).

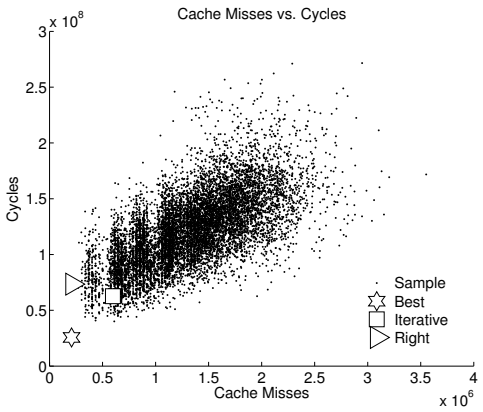

Figure 8: Cache Misses vs. Cycles for WHT_{18} (correlation coefficient $\rho = 0.66$, left recursive algorithm outside range).

ficient was increased from 0.77 for instruction count alone and 0.66 for cache misses alone to 0.92, a value close to the instruction count model for the sample that fits in L1 cache.

The correlation between instruction count and performance for small transforms and the correlation between the combined instruction count and cache misses can be used to prune the search for the algorithm with the best performance. While there is not a perfect correlation, the correlation that is shown allows us to prune algorithms which have a large number of instructions or a large combination of instructions and cache misses for large sizes. Figures 10 and 11 show the cumulative distribution functions for the samples of small and large transforms. Each curve in the figures show the cumulative probability of obtaining an algorithm outside of the p^{th} percentile as a function of instruction count or combined instruction count and cache misses. For a given instruction count, or combined count, the value of the curve gives the probability that an algorithm with fewer than or equal to the specified number has performance worse than the top p percent. In the limit as the instruction count or combined counts, approach the maximum value, the cumulative probability should approach $1 - p$. Whenever the curve gets close to $1 - p$ we do not need to consider algorithms with greater

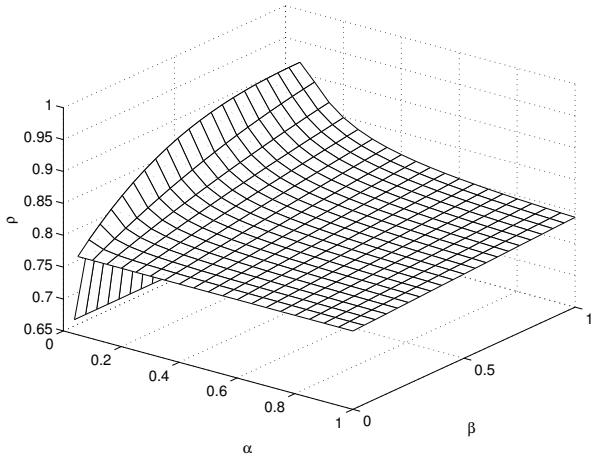

Figure 9: Correlation of Cycles and α Instructions + β Misses for WHT_{18}. Maximum $\rho = 0.92$ occurs when $\alpha = 1.00$ and $\beta = 0.05$.

counts. Thus for size $n = 9$, to find an algorithm whose performance is within 5% of the best we may discard all algorithms with more than 7×10^4 instructions (see Figure 10).

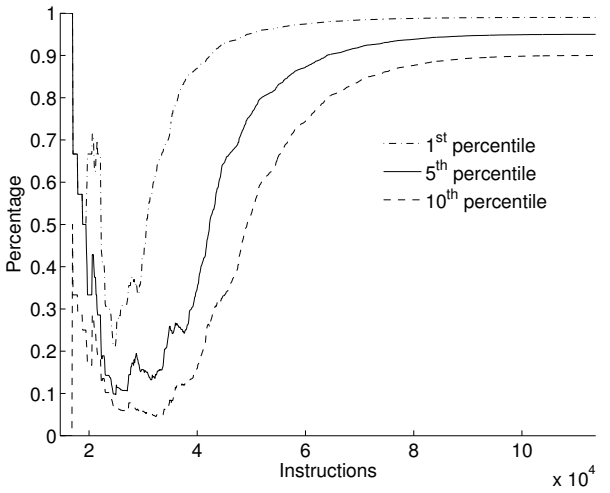

Figure 10: Cumulative percentage of WHT_9 algorithms with performance outside the p^{th} percentile as a function of Instructions.

5 Conclusion

This paper explored the empirical performance of a large family of algorithms for computing the WHT. The correlation between the actual performance and two performance models (instruction count and cache misses) was investigated on an Opteron processor. While these models do not accurately predict performance, there is a strong correlation (correlation coefficient = 0.96 for instruction counts for small transforms, and 0.92 for a linear combination of instruction counts and cache misses for large transforms) between the models and cycle counts. The correlation depends on the architecture on which the algorithms are executed and this dependence is currently being investigated for a variety of architectures. This correlation allows us to prune a large number of algorithms when searching for the optimal performance. Because the performance models can be computed from a high–level description of the algorithms it is possi-

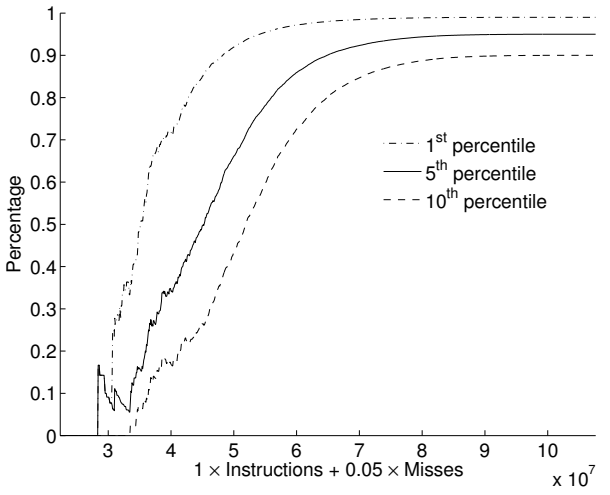

Figure 11: Cumulative percentage of WHT_{18} algorithms with Cycle counts outside the p^{th} percentile as a function of $\alpha \times$ Instructions $+ \beta \times$ Misses.

ble to calculate instruction counts and cache misses without running the programs. It is possible to systematically generate algorithms with small numbers of instructions and cache misses and consequently restrict a random or exhaustive search to this subspace of algorithms.

References

[1] V. Alexandrov, J. Dongarra, B. Juliano, R. Renner, and C. Tan, editors. *Computational Science – ICCS 2001*, volume 2073 of *Lecture Notes in Computer Science*. Springer, 2001. Session on Architecture–Specific Automatic Performance Tuning.

[2] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A portable programming interface for performance evaluation on modern processors. *The International Journal of High Performance Computing Applications*, 14(3):189–204, Fall 2000.

[3] Jim Demmel, Jack Dongarra, Victor Eijkhout an Erika Fuentes, Antoine Petitet, Rich Vuduc an Clint Whaley, and Katherine Yelick. Self adapting linear algebra algorithms and software. *Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation"*, 93(2), 2005.

[4] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation"*, 93(2), 2005.

[5] Pawel Hitczenko, Jeremy Johnson, and Hung-Jen Huang. Distribution of a class of divide and conquer recurrences arising from the computation of the Walsh–Hadamard transform. *Theoret. Comput. Sci.*, 352:8–30, March 2006.

[6] Hung-Jen Huang. Performance analysis of an adaptive algorithm for the Walsh–Hadamard transform. Master's thesis, Drexel University, 2002.

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

[8] Jeremy Johnson Mihai Furis, Pawel Hitczenko. Cache miss analysis of WHT algorithms. In *Proc. 2005 International Conference on Analysis of Algorithms*. DMTCS Proceedings, 2005.

[9] J. Moura, M. Püschel, J. Dongarra, and D. Padua, editors. *Proceedings of IEEE, special issue on "Program Generation, Optimization, and Adaptation"*, volume 93, February 2005.

[10] Markus Püschel, José M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan W. Singer, Jianxin Xiong, Franz Franchetti, Aca Gačić, Yevgen Voronenko, Kang Chen, Robert W. Johnson, and Nick Rizzolo. SPIRAL: Code generation for DSP transforms. *Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation"*, 93(2), 2005.

[11] B. Singer and M. M. Veloso. Automating the Modeling and Optimization of the Performance of Signal Transforms. *IEEE Trans. on Signal Processing*, 50(8):2003–2014, 2002.

[12] B. Singer and M. M. Veloso. Learning to Construct Fast Signal Processing Implementations. *Journal of Machine Learning Research*, 3:887–919, 2002.

[13] Kamen Yotov, Xiaoming Li, Gang Ren, Maria Garzaran, David Padua, Keshav Pingali, and Paul Stodghill. Is search really necessary to generate high–performance BLAS? *Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation"*, 93(2), 2005.