A Statistical Method for the Evaluation of Compiler Switches

Thijs van Ommen

Bachelor thesis
Mathematical Institute, Leiden University
Supervisors: Prof. S. van de Geer, Dr. E.W. van Zwet

December 15, 2005

1 Introduction

When a computer program is compiled, the settings of many compiler switches affect the efficiency of the resulting program. The optimal set of switches will depend not only on the architecture of the computer on which the program is to be run, but will also differ from one program to another. It is very difficult to determine analytically how a given set of switches will interact, even if the precise implementation of the compiler itself is known. A statistical method to find a good set of switches is therefore desirable. In their article 'Feedback-Directed Selection and Characterization of Compiler Optimizations'[2], Kingsum Chow and Youfeng Wu propose such a method.

We will first look at the method used by Chow and Wu. From sections 6 onwards, we will discuss a number of variations on this method.

2 Model Description

If the compiler switches functioned independently, finding the set of switches for which the expected performance is best among all those sets would be straightforward, as the effects of the switches could all be measured individually. However, the performance can't be solely determined from the sum of these main effects, because in practice many dependencies occur between the switches. In addition to their main effects, two switches may contribute a positive or negative combined effect to the compiled program's performance. Such a combined effect is called a second order interaction effect. Similar higher-order interactions may exist for larger subsets of switches.

A complete analysis of all main effects and interaction effects would require testing all possible combinations. An experimental design which includes all these combinations is called a full factorial design. Given n binary switches, this design would require 2^n test runs, each consisting of compiling and running the program under study. This number soon becomes impractically large. Fortunately, much information can be gained from testing only a fraction of all existing combinations.

3 Fractional Factorial Design

There are various ways to select a subset of the full factorial design. The method used in [2] starts by completing a full factorial design for a reduced number of switches. The setting of a switch is denoted by $+1$ or -1, which signifies that the switch is on or off, respectively. For every combination, the value of each remaining switch is then determined by multiplying the values of several other switches. For example (see table), a full factorial design is generated for the switches X_1, X_2 and X_3, and the value of X_4 in each run is chosen equal to $X_1X_2X_3$.

run	X_1	X_2	X_3	X_4
1	-1	-1	-1	-1
2	-1	-1	$+1$	$+1$
3	-1	$+1$	-1	$+1$
4	-1	$+1$	$+1$	-1
5	$+1$	-1	-1	$+1$
6	$+1$	-1	$+1$	-1
7	$+1$	$+1$	-1	-1
8	$+1$	$+1$	$+1$	$+1$

In this instance, the one formula $X_4 = X_1X_2X_3$ was used, which caused the number of runs required to be cut in half. More such identities would further reduce the number of experiment runs.

4 Analysis Using a Linear Model

By performing the above experiment, a value Z_i is found for the running time of run i. To analyze this data, a linear model is used. In such a model, the results are expressed as linear combinations of the vector $Y_{i\bullet}$. Each Y_{ik} corresponds to either the general mean, or the effect of one of the compiler switches or an interaction between those switches:

$$Y_{i0} = 1, Y_{i1} = X_{i1}, Y_{i2} = X_{i2}, Y_{i3} = X_{i1}X_{i2}, \ldots, Y_{i15} = X_{i1}X_{i2}X_{i3}X_{i4}. \qquad (1)$$

The order in which these are numbered can be freely chosen. We will only use that $Y_{i0} = 1$ for all i. Because each Y_{ik} is a product of the factors 1 and -1 from $X_{i1} \ldots X_{in}$, it will again be 1 or -1, depending on which switches were set in test run i. If N runs are performed, then each $i \in \{1, 2, \ldots, N\}$ gives rise to a linear equation

$$Z_i = C_0 + C_1Y_{i1} + C_2Y_{i2} + \ldots + C_{15}Y_{i15}. \qquad (2)$$

The values of main and interaction effects can be determined by solving the system of linear equations in the unknowns $C_0, C_1, \ldots, C_{15}$ using the method of least squares. Here, C_0 is the general mean, and each C_j with $j > 0$ is a main or interaction effect. More generally, for an experiment involving n compiler switches, the unknowns to be estimated are $C_0, C_1, \ldots, C_p$ where $p = 2^n - 1$.

However, we cannot do this for all higher-order interactions. Due to what is called aliasing in the experiment, the value of $X_1X_2X_3$ is always equal to that of X_4. This means that for each run where the effect of $X_1X_2X_3$ occurs in the model with a positive contribution, the effect of X_4 has a positive contribution, and the same is the case for their negative contributions. It is thus impossible to distinguish between the effects of $X_1X_2X_3$ and X_4; the model will only allow us to estimate their combined effect. This can be seen from the system of linear equations by looking at the columns corresponding to $X_1X_2X_3$ and X_4. These columns are identical, meaning that the coefficient matrix in singular.

Because the values of the X_i are either 1 or -1, the identity $X_4 = X_1X_2X_3$ can be rewritten in many ways, for example $X_1 = X_2X_3X_4$ and $X_1X_2 = X_3X_4$.

These equivalent identities imply more aliases between switch combinations: each main effect is confounded with a third order interaction effect, and each second order interaction effect is confounded with another second order interaction effect. This represents a sacrifice in information gained from the experiment, which is the necessary consequence of performing fewer test runs.

It is assumed that the main effects of the switches will generally be larger than the effects of the interactions. Under this assumption, the contribution of $X_1X_2X_3 + X_4$ which was found by the model will be largely ascribed to X_4, whereas $X_1X_2X_3$ is assumed to be near zero. The identity $X_1X_2 = X_3X_4$ cannot be handled in such a way, as without further assumptions, there is no way of knowing which of X_1X_2 and X_3X_4 is the main contributor. If the estimate of $X_1X_2 + X_3X_4$ is small, it is assumed that neither X_1X_2 nor X_3X_4 is significant. However, if the estimate appears to be significant, then it is desirable to know the effects of X_1X_2 and X_3X_4 independently. More experimental information would be needed to make any statements about this.

5 Subsequent Test Runs

In the article by Chow and Wu, further test runs are carried out in order to resolve the aliases described above. Such runs are designed by turning off any switches that showed clear negative effects and turning on those that contributed positively. This leaves open some switches that did not show a clear effect either way; new runs are generated for each combination of such switches. However, the details of this process are unclear from the article, which demonstrates the procedure only by means of a simple example.

6 Modification of the Model

Each of the data points which are used as input for the model is found by compiling the target program using the given setting of compiler switches. The compiled program is executed and its execution time is measured. The measurements will vary from one execution to another for a single program, due to interference from other processes running on the computer. To represent this measurement error, an error term e_i must be added to each equation in the model.

The model used by [2] assumes that the effects of different switch combinations can be expressed additively. It seems more natural to assume that the main effects and interaction effects are of a multiplicative nature: applying a certain optimization or combination of optimizations will not result in the program's execution time being increased or decreased by a certain constant dependent only on the compiler switch settings, but will rather result in a proportional adjustment in execution time. The same is reasonable to assume for the measurement errors. This does not affect the expressive power of the model, but it does change which situations are represented by smaller values for the parameters, which is

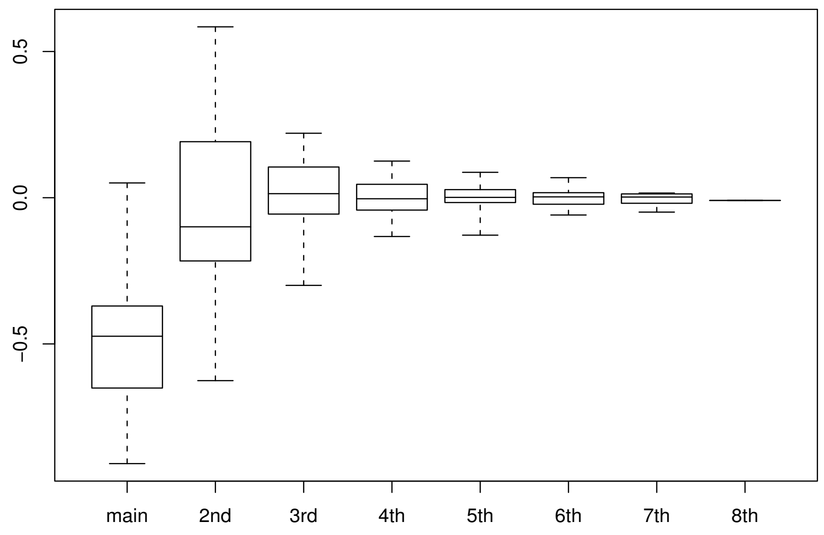

Figure 1: Simulated values for main and interaction effects

relevant as we will see now.

It was mentioned that we expect the values of higher-order interaction effects to be near zero. To incorporate this belief, before any measurements are made, we assume that each jth-order interaction's effect follows a normal distribution with mean 0 and variance τ_j^2. In some cases, we already have some information about the effects before the first test is run. As an example, assume that several switches are already known to improve the compiled program's performance. The values of the associated main effects are then assumed to follow a normal distribution with mean $\mu_1 < 0$ (because the program's running time is expected to decrease) and variance τ_1^2. Figure 1 shows an example of a set of values that were randomly drawn using a distribution as described above. To make further analysis of the model easier, we further transform the measurements Z to account for such adjustments in the mean, so that in the transformed model, all effects including the main effects are again distributed about 0. We do not make any assumptions about the general mean C_0.

The model now has the form

$$Z_i = \log(\text{measurement}_i) = C_0 Y_{i0} + C_1 Y_{i1} + C_2 Y_{i2} + \ldots + C_p Y_{ip} + e_i. \qquad (3)$$

To allow this model to be analyzed, the following assumptions are made regarding the error term: the error terms for all measurements are assumed to obey the same normal distribution with mean 0. Further, the error terms for different measurements are assumed to be independent.

7 Regularization

Due to the presence of measurement errors, solving the linear system described earlier will not give desirable results. The solution of the system will overfit, as a small error in a single measurement causes large changes in the estimated values C_i. Many statistical approaches exist which may help to counteract this phenomenon.

In [2], results with high uncertainties are discarded in order to avoid focusing on measurement errors. The disadvantage of this method is that throwing away some variables is a discrete process. The result will often be subject to high variance. Other methods exist that do not have this disadvantage, known as regularization methods. An example is ridge regression. The method of least squares is modified to find a vector C which gives a good fit while keeping the values of C small. This is done by including penalty terms for each C_i other than C_0 in addition to the sum of squared differences, and minimizing this result. These terms are of the form $\lambda_i C_i^2$. The larger we choose λ_i, the harder the method will try to keep the associated value C_i small.

Another advantage of ridge regression is that it allows us to work with uncertainties regarding the impact of higher-order interactions on performance. Due to confounding in a fractional factorial design, assumptions must be made about the actual source of the summed contribution of a set of aliased switch combinations. Ridge regression solves this problem in an elegant way, as we may choose a different λ_i for each C_i. Because we expect to explain most of our data by the main effects of the compiler switches and assume that higher-order interactions will have progressively smaller effects on performance, a value is chosen for λ_i depending on the number of factors X in Y_i. This leads to the expression

$$\sum_{i=1}^{p}(Z_i - \sum_{j=0}^{p} C_j Y_{ij})^2 + \sum_{j=1}^{p} \lambda_j C_j^2 \tag{4}$$

where Y_{ij} equals $X_{ik_1} X_{ik_2} \ldots X_{ik_l}$, the product of values of the switches in run i.

Using matrix algebra, the ridge regression criterion can be written as

$$(Z - YC)^T (Z - YC) + C^T P C \tag{5}$$

where Z is the N-vector of observations, Y is an $N \times (p+1)$ matrix consisting of ± 1, C is the $(p+1)$-vector of the mean and values for the switches and interactions, and P is the diagonal matrix with $P_{ii} = \lambda_i$ for $i > 0$ and $P_{00} = 0$, where the rows and columns are numbered from 0. The outcome of minimizing this criterion can be computed as

$$\hat{C}^{\text{ridge}} = (Y^T Y + P)^{-1} Y^T Z \tag{6}$$

([3]).

To find values for the penalty coefficients λ_i, we adopt a Bayesian point of view. Let

$$f(Z|C) = (2\pi\sigma^2)^{-n/2} \exp\left[\frac{-1}{2\sigma^2}(Z - YC)^T(Z - YC)\right] \tag{7}$$

be the probability density function of the sampling distribution denoting that the error terms are normally distributed with mean 0 and variance $\sigma^2 > 0$, and let $g(C)$ be the prior distribution, representing our assumptions about the values of the main and interaction effects before measurements are collected. If the prior follows a multivariate normal distribution with mean 0, variances given by the vector τ, and covariances 0, then its density is given by

$$g(C) = (2\pi)^{-n/2}(\tau^T\tau)^{-1/2} \exp\left[-\frac{1}{2}C^T \mathrm{diag}(\tau)^{-2} C\right] \tag{8}$$

and the posterior distribution of C adjusted by observations Z is given by

$$h(C|Z) = \frac{f(Z|C)g(C)}{\int f(Z|U)g(U)dU} = cf(Z|C)g(C). \tag{9}$$

Taking the logarithm of this expression, we find

$$\begin{aligned} -\log(h(C|Z)) &= c + \frac{1}{2\sigma^2}\sum_{i=1}^{N}(Z_i - \sum_{j=0}^{p} Y_{ij}C_j)^2 + \frac{1}{2}\sum_{j=0}^{p}\frac{C_j^2}{\tau_j^2} && (10) \\ &\propto c' + \sum_{i=1}^{N}(Z_i - \sum_{j=0}^{p} Y_{ij}C_j)^2 + \sum_{j=0}^{p}\frac{\sigma^2}{\tau_j^2}C_j^2 && (11) \end{aligned}$$

which is equal up to the constant c' to (4), the expression minimized by ridge regression with $\lambda_j = \sigma^2/\tau_j^2$. This means that the ridge regression estimate for C maximizes the likelihood according to the posterior distribution. The posterior is normally distributed, so its mode is also its expectation. ([3], page 60; [4], page 589)

Because we choose large values of τ_j for those j corresponding to the mean, main effects and the lower-order interactions, the prior distributions for the main effects and lower-order interaction effects follow a flat curve. This is called a noninformative prior, and its result is that the posterior will depend to a greater extent on the observations. The opposite is true for the higher-order interaction effects, which were given prior distributions with low variance. In these cases, the observed data has much less effect on the posterior distribution.

8 Computing the Estimates

Expression (6) for the mean of the posterior distribution involves taking the inverse of a $(p+1) \times (p+1)$ matrix. Because $p+1$ will typically be much larger than N, the expression

$$TY^T(YTY^T + \sigma^2 I_N)^{-1}Z \tag{12}$$

where $T = \mathrm{diag}(\tau)^2$ ([1], page 36) will be more useful. We see that the inverse of $YTY^T + \sigma^2 I_N$ exists because this matrix can be written as

$$\sum_{k=0}^{p} \tau_k^2 Y_{\bullet k} Y_{\bullet k}^T + \sigma^2 I_N, \tag{13}$$

where $\sigma^2 I_N$ is a positive definite matrix, and $\tau_k^2 Y_{\bullet k} Y_{\bullet k}^T$ is positive semidefinite for each k. Thus we see that the matrix to be inverted is positive definite, which implies that its determinant is positive.

If the τ_i are finite, positive values, then T is invertible, and we see from

$$\begin{aligned}
(Y^TY + \sigma^2 T^{-1})TY^T &= (Y^TYTY^T + \sigma^2 Y^T) && (14)\\
&= Y^T(YTY^T + \sigma^2 I_N) && (15)\\
TY^T &= (Y^TY + \sigma^2 T^{-1})^{-1} Y^T (YTY^T + \sigma^2 I_N) && (16)\\
TY^T(YTY^T + \sigma^2 I_N)^{-1}Z &= (Y^TY + \sigma^2 T^{-1})^{-1} Y^T Z && (17)
\end{aligned}$$

that the two expressions for $\hat{C}$ in (6) and (12) are equal. A problem occurs because $P_{00} = \sigma^2/\tau_0^2 = 0$ (we do not want to penalize the general mean under ridge regression), which we solve by taking

$$\hat{C} = \lim_{\tau_0 \to \infty} TY^T(YTY^T + \sigma^2 I_N)^{-1}Z. \tag{18}$$

and this will converge to $\hat{C}^{\text{ridge}}$ with $P_{00} = 0$.

Because $P = \sigma^2 T^{-1}$ does not exist if $\tau_i = 0$ for some i, the ridge regression formula could not deal with such a choice of τ. T^{-1} does not appear in the new expression for $\hat{C}$, so that option is now open. Choosing $\tau_i = 0$ expresses a prior distribution where effect i is 0 with probability 1. It is easy to see that the rows and columns of T and the columns of Y which correspond to i can be omitted from the matrices to reduce the computational load. For example, if we decide to set $\tau_i = 0$ for all interactions i of order k or greater for some constant k, the size of the matrices in the calculation will be polynomial in n, rather than exponential. This has the disadvantage that the benefit of regularization does not apply to the higher-order interactions anymore, but that will be an acceptable tradeoff in many cases.

It is now possible to compute the expected performance value of each given switch setting under the posterior distribution. We are interested in knowing which switch setting maximizes this value. For a small number of switches, this can be done by computing

$$\hat{Z}_i = \hat{C}_0 \pm \hat{C}_1 \pm \hat{C}_2 \pm \ldots \pm \hat{C}_p \tag{19}$$

for each $i \in \{1, 2, \ldots, 2^n\}$. For large numbers of switches, more advanced methods must be used. Finding the optimal values for $X_1, X_2, \ldots, X_n$ can be expressed as a linear optimization problem. To see this, we note that the relation

$Y = X_1X_2$ where all variables are ±1 can be expressed by the set of linear inequalities

$$\begin{aligned} Y &\geq X_1 + X_2 - 1 & (20)\\ Y &\geq -X_1 - X_2 - 1 & (21)\\ Y &\leq X_1 - X_2 + 1 & (22)\\ Y &\leq -X_1 + X_2 + 1. & (23) \end{aligned}$$

Computationally efficient methods are available from operations research to solve problems of this type. If the number of switches is very large but we are only interested in estimating effects up to a certain order, solving the problem this way is significantly more efficient than using the brute force method.

9 A Heuristic for Choosing Test Runs

Also from [1] comes the expression

$$T - TY^T(YTY^T + \sigma^2 I_N)^{-1}YT \tag{24}$$

for the covariance matrix of the posterior distribution. This tells us which of the estimates of the effects are reliable, and which we should gather more test data on. We would like to use this information to determine which switch setting we should use next if we wish to perform another test run.

The estimate of the general mean C_0 gives no information about the optimality of switch settings, so we ignore it here. For each switch setting, there is a region in the space of the other elements of C where that switch setting is optimal. These regions are separated by hyperplanes. When $C_1 = C_2 = ... = C_p = 0$, all these regions meet. We aim to arrive at a posterior distribution which is concentrated in a single region. The posterior distribution theoretically allows us to calculate the probability mass in each region, but that would be a cumbersome calculation.

By projecting the distribution onto the line through 0 and $\hat{C}$ and looking at the variance of the resulting univariate normal distribution, we find a heuristic indication of how well the probability mass falls within a single region. An expression for this variance is

$$\hat{C}^T \Sigma_Y \hat{C} \tag{25}$$

([1], page 33), where $\hat{C}$ is the existing estimate of C and Σ_Y is the covariance matrix that results from a given design matrix Y (it does not depend on the measurements Z). Instead of minimizing (25), we can substitute the expression for the covariance matrix (24) and maximize

$$\hat{C}^T TY^T(YTY^T + \sigma^2 I_N)^{-1}YT\hat{C}. \tag{26}$$

For each switch setting we could use in the next test run, we modify Y to have that switch setting in row $N + 1$ and compute (26).

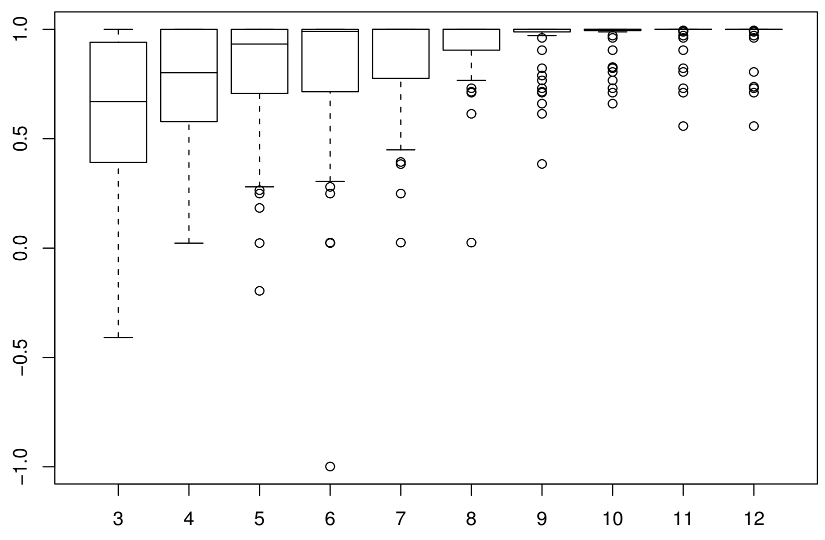

Figure 2: Results of performing 50 experiments on a very small design consisting of 4 switches, where the test runs were chosen randomly, but without repeating a single switch setting for more than one test run. The x axis gives the number of test runs, and the y axis gives an indication of the true quality of the switch setting which was expected to be optimal at that point. The true optimal setting is at 1, and the general mean is at 0.

Using this criterion to choose test runs for us can replace using another form of experimental design. Dynamically designing an experiment based on the data gathered so far has obvious advantages over using a traditional, static experimental design. Subsequent test runs will be focused on finding more information regarding the effects which are most relevant to the choice of an optimal setting, or those effect where the estimate is least precise. An alias will be resolved if the posterior distribution reflects that resolving it will contribute to the quality of the estimate, and remain unresolved in favor of more important measurements otherwise.

Unfortunately, this computation must be performed for each possible switch setting, which becomes infeasible for even modest values of n. For larger n, more efficient ways to calculate or approximate the criterion must be found, or a different heuristic must be designed.

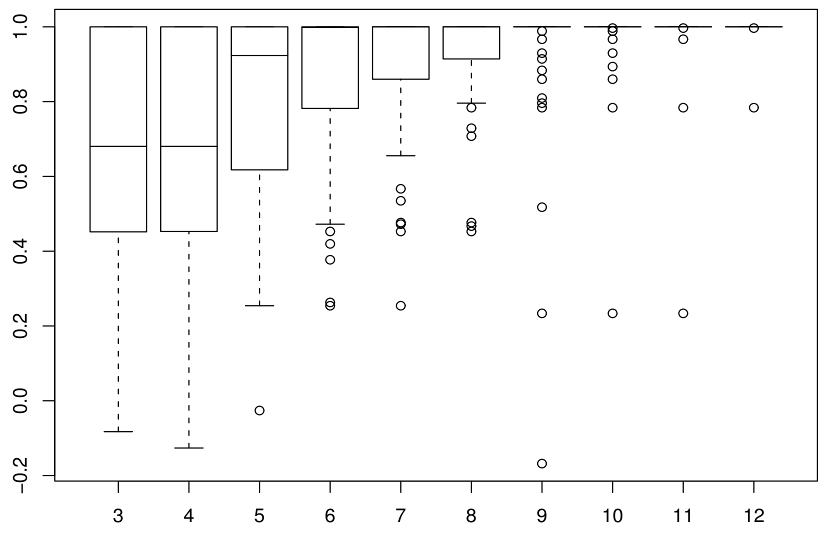

Figure 3: Again 50 experiments were performed, but here the heuristic described in this section was used to select the test runs.

10 Comparison to Other Methods

10.1 Factory Settings

Compiler writers usually supply standard settings for the optimization switches. These settings were chosen with knowledge of the inner workings of the compiler, but without information regarding the specific platform and program to be compiled. Because the effectiveness of the switches depends on the latter for a large part, the factory-supplied settings usually are not optimal; sometimes even far from it. They can easily be outperformed by the settings found by a method such as described here.

10.2 Evolutionary Algorithms

Evolutionary algorithms have been used to solve the problem of finding good settings of compiler switches. The main difference between those algorithms and the method described here is that evolutionary algorithms are designed to focus on local information, trying to find a good solution by taking small steps in the setting space. The method described here emphasizes global information, because the model assumes that higher-order interactions do not contribute significantly to the performance of the program. Information regarding the main effects and lower-order interaction effects is applicable globally.

11 Conclusion

So far, only binary switches which can be either on or off have been discussed. Many compilers also offer switches that can assume more than two values. Adaptation of the model for such switches is a topic for further research.

Execution of the compiled programs was used to measure performance of a set of compiler switches. Alternatively, simulation could be used. This would yield exact results for the number of clock cycles required to execute a program. If the same program were to be simulated twice, identical measurements would be found. The disadvantage of the approach is that simulation is usually many times slower than execution. The changes to the model implied by these differences form another interesting point for further research.

Despite the regularization, ridge regression is still somewhat sensitive to outliers in the data. Methods for detecting data points that are likely to be outliers and methods for dealing with such potential outliers might give a significant improvement in some cases.

The application of ridge regression described here assumes that the variance of the error terms and suitable prior distributions for the main and interaction effects are known. Some tests could be allocated to verify the validity of this information for a given combination of compiler, architecture, and target program.

Different computational methods from linear algebra and operations research may be applied to perform the calculations required in a more efficient manner.

References

[1] P. J. Brockwell, R. A. Paris, Time Series: Theory and Methods, Springer, 1991.

[2] K. Chow and Y. Wu, Feedback-directed selection and characterization of compiler optimizations. In *Proc. 2nd Workshop on Feedback Directed Optimization*, 1999.

[3] T. Hastie, R. Tibshirani and J. Friedman, The Elements of Statistical Learning, Springer, 2001.

[4] J. A. Rice, Mathematical Statistics and Data Analysis, Second Edition, Duxbury Press, 1995.

A Appendix

Reproduced below are the programs that were used to simulate the model.

A.1 init.R

```
# Creates data structures to be used by simulation.R when creating
#   a simulation, and sets up some functions.
# This file only needs to be sourced when the number of switches
#   (dims) in the model changes.
dims <- 4

# parameters for creating a simulation
# - standard deviation of the error term
sigma <- 5

# - standard deviation of the general mean, main effects, and
#   higher-order interactions
sdbyorder <- function(order) {
        if (order == 0)
                ret <- 100
        else
                ret <- 100 / order^2
        ret
}

# - expected values of general mean, main effects and interactions
meanbyorder <- function(order) {
        if (order == 0)
                ret <- 1000
        else
                ret <- 0
        ret
}

# Construct vectors using the above functions
xlist <- list()
sds <- vector("integer", 2^dims)
for (i in 1:dims) {
        # 2^(dims-i) times -1, then that many +1, for 2^dims total
        xlist[[i]] <- gl(2, 2^(dims-i), 2^dims, labels = c(-1, 1))
        # sum all levels for each factor to use in sd for rnorm
        #   (as.vector returns 1 and 2, ignoring the labels)
        sds <- sds + as.vector(xlist[[i]], "integer")
}
sds <- sds - dims
# sds is now a vector with an element for each of the 2^n (mean,
#   main/interaction effects). It is 0 for the mean, 1 for main
#   effects, and n for nth order interaction effects.
# Next, let means and sds be vectors of the means and standard
#   deviations of each (mean, main/interaction effect). Using
#   these vectors, some random "actual" values can be drawn for
#   use in a simulation.
means <- sapply(sds, meanbyorder)
sds <- sapply(sds, sdbyorder)
# construct taumat, the matrix with entries tau_i^2 on the main
#   diagonal
taumat <- array(0, c(2^dims, 2^dims))
diag(taumat) <- sds^2
taumat[1,1] <- 1e10 # We want the limit of this going to infinity.
#   R can't (easily) do this for us, so we settle for picking a
#   large number. This will lead to computational inaccuracies,
#   especially in the posterior variance of the general mean.

# find the vector of coefficients
getY <- function(test) {
        Y <- vector("integer", 2^dims)
        for (i in 1:(2^dims)) {
                # For each X that appears as a factor in Y[i],
                #   multiply Y[i] by that X
                Y[i] <- 1
                for (k in 1:dims) {
                        if (xlist[[k]][i] == 1
                            && xlist[[k]][test + 1] == -1)
                                Y[i] <- -Y[i]
                }
        }
        Y
}

# Cause data from previous experiments to be cleared on next call
#   of iteration.
beginexperiment <- function() {
        count <<- 0
}

# Perform a given test run (getting the result from dotestrun) and
#   update all data structures accordingly.
iteration <- function(test) {
        count <<- count + 1

        Yvector <- getY(test)
        if (count == 1)
                Z <<- dotestrun(Yvector)
        else
                Z <<- c(Z, dotestrun(Yvector))
        if (count == 1)
                Y <<- array(Yvector, c(1, 2^dims))
        else
                Y <<- rbind(Y, Yvector)
        # first compute diag(tau) t(Y) Sigma22^-1
        intermediate <- taumat %*% t(Y) %*% solve(Y %*% taumat
                        %*% t(Y) + sigma^2 * diag(count))
        # compute posterior mean and covariance
        estfx <<- intermediate %*% Z
        estcov <<- taumat - intermediate %*% Y %*% taumat
        estfx
}
```

A.2 simulation.R

```
# Sets truefx, which is used when simulating measurements, and
#   creates functions required for simulation.

# Draw random values for general mean, main effects and
#   interactions.
truefx <- rnorm(2 ^ dims, means, sds)

# Simulate a test run. The argument specifies which switches are
#   on for this run, using binary encoding.
# For practical application, the logarithm of a measurement should
#   be returned instead.
dotestrun <- function(Yvector) {
        rnorm(1, t(Yvector) %*% truefx, sigma)
}
```

A.3 method.R

```
# A collection of functions which use the posterior mean and
#   covariance to design an experiment.

# Gives an indication of the value of choosing a particular
#   switch setting for the next experiment run. The returned
#   value equals the constant t(estfx) %*% taumat %*% estfx
#   minus the variance of the posterior distribution projected
#   on the line through 0 and estfx.
estimprovement <- function(test) {
        if (count == 0)
                return(0)

        # add a row, then remove the first column
        Yvector <- getY(test)
        newY <- rbind(Y, Yvector)[,-1]
        (t(estfx[-1]) %*% taumat[-1,-1] %*% t(newY) %*% solve(newY
                        %*% taumat[-1,-1] %*% t(newY)
                        + sigma^2 * diag(count + 1))
                        %*% newY %*% taumat[-1,-1] %*% estfx[-1])
}
```

```
# Use a heuristic to determine which switch setting might be
#   a good choice for a next test run.
choosesubsequent <- function() {
        estimpr <- sapply(0:(2^dims-1), estimprovement)
        which.max(estimpr) - 1
}

# Returns the switch setting with the best (here: highest)
#   expected performance.
bestsetting <- function() {
        best <- 0
        bestval <- t(getY(0)) %*% estfx
        for (i in 1:(2^dims - 1)) {
                val <- t(getY(i)) %*% estfx
                if (val > bestval) {
                        bestval <- val
                        best <- i
                }
        }
        best
}

# Do a complete experiment consisting of totalruns test runs.
#   The return value is the switch setting with the highest
#   expected performance.
performmethod <- function(totalruns) {
        beginexperiment()
        for (i in 1:totalruns) {
                setting <- choosesubsequent()
                print("Test run using switch setting:")
                print(setting)
                iteration(setting)
        }
        bestsetting()
}
```