International Journal of Electronics, Electrical and Computational System
IJEECS
ISSN 2348-117X
Volume 6, Issue 8
August 2017

A Hybrid Method for Selection of Tile Sizes

Arka Ghosh S. Balasubramanian
Sri Sathya Sai Institute of Higher Learning

ABSTRACT

Restructuring a piece of code to make optimal use of the processor power can enhance performance of an application. Loop tiling is one such restructuring technique to foster reuse of data and avoid expensive memory accesses. It is important to choose an appropriate tile size as it determines whether the cache is underutilized, optimally utilized or overwhelmed. In this work we focus on the aspect of tile size selection (TSS).We propose a Hybrid kernel specific model, which is a combination of a kernel independent and a kernel specific machine learning (ML) based model, to predict optimal rectangular tile dimensions. Obtaining the optimal rectangular tile dimensions by exhaustive search is nearly impossible because of the prohibitively large size of the search space. Our model is a 2-phase approach which prunes the search space for rectangular tile sizes and performs an empirical search in the pruned space to obtain the optimal tile sizes. The inputs to our model are read and write non-prefetched features and the output is the optimal rectangular tile dimension thus making it portable across platforms. We show that our model can predict tile dimensions with which the execution times of the kernels deviate at the most 14.44% from those predicted by the KS model which is based on exhaustive search. Further, we observe that whenever the difference in execution time is greater than 1%, it is due to the Hybrid model performing better. For instance, when the Hybrid model performs worse than the KS model the deviation is at most 0.86%. In the worst case our model requires a traversal of at most 54.77% of the search space for a kernel. In most of the cases the size of the search space that requires to be traversed is less than 4%.

Keywords

Tile Size Selection, Artificial Neural Network, Machine Learning, Performance Modeling.

I INTRODUCTION

One of the most important factors to be considered in the design of an application is its performance. This demand restructuring the code using certain transformation techniques to make best use of the processor power. It is important to automate the optimization procedure for different architecture/compiler combinations. This is much faster than hand tuning a piece of code for the same. In the current work we focus on loop tiling which is a loop transformation technique, which reorders loops to enable data reuse. This requires selection of the appropriate tile size for a given problem. We focus on automating the process of TSS.

A. LOOP TILING

Loop tiling has two steps:

(i) Strip mining

(ii) Loop permutation

During the strip mining phase, the loops are split into intra-tile and inter-tile loops with the former iterating within tiles and the latter iterating across tiles. During the loop permutation phase the inter-tile loops are moved outwards. An illustration of the steps is as follows.

Original Code:

```
for(i =  0; i <  N;   i++)
        for(j =  0; j <  N;   j++)
```

```
                    mat(i,j);
```

Stripmined Code:

```
for(it =  0; it <  N;   it+=T)
        for(i =  it; i <  min(it+T, N); i++)
                for(jt =  0; jt <  N;   jt+=T)
                        for(j =  jt; j <  min(jt+T, N); j++)
                                mat(i,j);
```

Here, we have strip mined the original 'i' loop into the intra-tile loop 'i' and inter-tile loop 'it' and the 'j' loop into intra and inter-tile loops 'j' and 'jt' respectively. In the next step, the loops are permuted so that the inter tile loops are moved above and the intra tile loops are moved below. After loop permutation, we obtain the following:

Code After Loop Permutation:

```
for(it =  0; it <  N;   it+=T)
        for(jt =  0; jt <  N;   jt+=T)
                for(i =  it; i <  min(it+T, N); i++)
                        for(j =  jt; j <  min(jt+T, N); j++)
                                mat(i,j);
```

In the code snippets above, T is the tile size. The choice of T determines whether cache is underutilized, used optimally or it is overwhelmed. Tiling can be depicted as in Fig 1.

B. DIFFERENT APPROACHES TO FINDING OPTIMAL TILE SIZE

The problem of tile size selection (TSS) is difficult due to

- Memory hierarchy.
- Cache complexity.
- Hardware prefetcher.
- Complex optimizations of the compiler.
- Evolution of architecture.

The different approaches to finding the optimal tile size can be categorized as one of the following.

- Static model based approach
- Model driven empirical search based approach
- MachineLearning (ML) Based Models

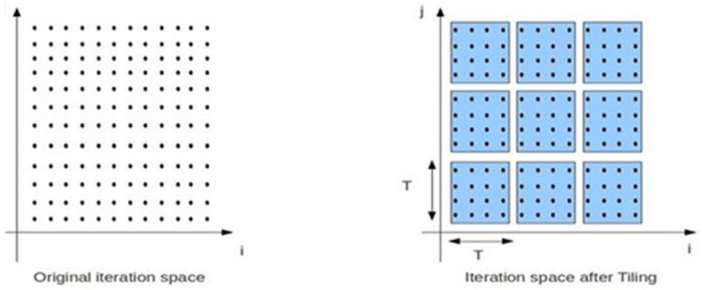

Fig 1: Illustration of Tiling

C. STATIC MODEL BASED APPROACH

These are analytical models developed based on observing the hardware, some software details, and behavior of a set of programs. This pre-designed static model directly gives as output the optimal tile size for a given problem. The compiler then uses this tile size to generate a tiled version of the program. These models can predict the optimal tile size for a given program on a specific architecture/compiler combination. This is a major drawback. They are not portable across different architecture/compiler combinations. Due to rapid change in the hardware technology, this constraint renders them difficult to use.

D. MODEL DRIVEN EMPIRICAL SEARCH BASED APPROACH

In this approach a TSS model is used to prune the search space of good tile sizes. For each tile size in this pruned space a version of the program is generated and executed on the target architecture/compiler combination and the tile size with the best execution time is selected. The Automatically Tuned Linear Algebra Software (ATLAS) performs an empirical search to obtain the optimal tile size for the kernels. These models are portable across architecture/compiler combinations but the bottleneck for this approach is the size of the search space. For example, it is impossible to use this approach to find the optimal rectangular tile sizes as the search space for rectangular tile sizes is exponentially larger than that for square tiles.

E. MACHINE LEARNING BASED MODELS

ML based models have become popular in recent years because of their portability across architecture/compiler combinations. Further, this enables selection of rectangular tile sizes, overcoming the size constraint of the search space in case of the empirical search based approaches. In this approach a ML model is trained with a small subset of the whole search space. This model can then be used to predict rectangular tile sizes.

In this work we propose a kernel specific hybrid model for TSS. Our model is a combination of a kernel independent and a kernel specific ML based method for TSS.

II. RELATED WORK

Substantial work has been undertaken for optimizations of CPU kernels. [2], [1], [3] and [4] present optimization techniques for CPU kernels. Markus Kowarschik et al. [2] provide a walk through the architecture and performance evaluation of caches. They further discuss data layout and data access optimizations for improving cache efficiency. [4] present a methodology to dynamically select tile dimensions at run time based on machine characteristics and system load at that instant.

Rahman et al. [1] proposed a neural network based tile size selection model to obtain optimal rectangular tile sizes for arbitrary programs. The work proposes kernel specific models. It is a near impossible task to obtain the optimal rectangular tile sizes by exhaustive search. In this work, the authors find empirically, the execution time of the concerned kernel for a small sample of tile sizes keeping the problem size fixed. This result is used for training an artificial neural network (ANN) which is then used to predict the execution times of the kernel with other tile sizes. The tiles with the best predicted execution times are then shortlisted. Finally, the kernel is run empirically with these tile sizes and the best performing tile size is chosen.

Tomofumi Yuki et al. [3] show that fairly accurate TSS models can be automatically created for the class of programs having three dimensional loops and two dimensional data by using a set of six program features namely read prefetched, read non-prefetched, read invariant, write prefetched, write non-prefetched, and write invariant features. The model is validated on six architecture compiler combinations using nine linear algebra kernels. The approach used in this paper has the following four steps:

1) generation of synthetic programs which fit into the class of interest
2) collection of training data
3) learning the TSS model using ANN
4) use of the ANN based TSS model to predict the tile size

The performance of the model is shown when it is used as a part of model-driven empirical search based approaches. The true optimal is found in most of the cases by searching tile sizes within a distance of 10 or 20. A comparison with hand-crafted models is also demonstrated.

III. MOTIVATION

The method proposed by Rahman et. al in [1] requires the complete space of rectangular tile sizes to be searched to predict the optimal tile size. This process can be time consuming if the search space is large. We intend to use the method proposed by Tomofumi Yuki et al. [3] to prune the search space for the kernel specific model [1]. The method proposed in [3] is a general method for determining optimal tile sizes for kernels. It endeavors to create a machine learning model which requires as input the number of prefetched, non-prefetched and invariant features (both read and write) to predict an optimal tile size for a kernel, thus making it a model independent of the kernel. However, we consider only the non-prefetched features in the current work and build a ML model based on only read and write non- prefetched features. Therefore, in our case the input to the model would be only the number of non-prefetched features in the kernel. The output remains the optimal square tile size as for the model in [3]. Let us call this model the Minimal Feature Set TSS (MFSTSS) Model. To obtain the optimal rectangular tile size we search in the neighborhood of the tile size predicted by the MFSTSS model using the method in [1] along all the 3 dimensions. A question may arise here as to why 2 models have to be used instead of using only the kernel independent model. The kernel independent model predicts only square tiles. This causes a prediction from a highly restricted space of tile

sizes. Therefore, the predictions of this model can often be suboptimal. This model has been observed to predict tile sizes which perform up to more than 200% slower than the space optimal.
The model we propose is a hybrid of the MFSTSS model and the one proposed in [1]. Henceforth, we refer to the model proposed in [1] as the Kernel Specific (KS) model.

IV. OUR APPROACH

The approach is diagrammatically represented in Fig 2. Our approach has two phases:

1) Use the MFSTSS model to obtain a square tile size.
2) Search in the neighborhood of the tile size predicted in step 1 for all the three dimensions using the KS model.

A. USING THE MFSTSS MODEL TO OBTAIN A SQUARE TILE SIZE

In this phase of our model we use the MFSTSS model to obtain a square tile size for the concerned kernel. The MFSTSS model is built in the same way as the model proposed in [3]. The same steps of synthetic program generation, data collection, learning TSS models using ANN and use of the ANN to obtain a square tile dimension for the kernel as in [3] are followed. The ANN configuration is also the same as that in [3]. It is a backpropagation ANN with 2 hidden layers of 30 nodes each. There are 2 input parameters in the input layer and 1 output parameter in the output layer. The scaled conjugate gradient algorithm is used for training the ANN. The activation function used is the tangent sigmoid function for the hidden layers and a linear function for the output layer. The learning rate is 0.03.

B. SEARCH IN THE NEIGHBORHOOD OF THE SQUARE TILE SIZE ALONG THREE DIMENSIONS USING THE KS MODEL

This phase of our model consists of finding the execution times of the kernel with tile sizes varying from the

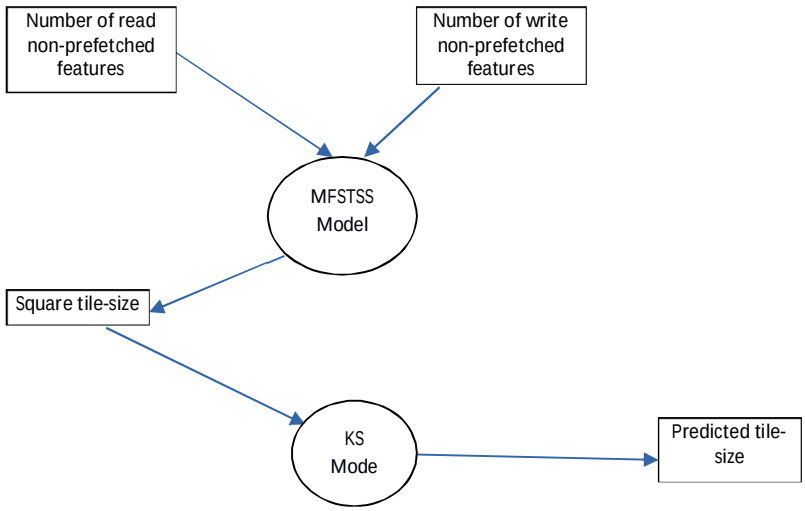

Fig 2. Hybrid Model

largest tile size less than or equal to the one predicted by the MFSTSS model to the largest in the search space along all the three dimensions using the KS model. The ANN used in this phase is the same as the one used in [1].

It is a feedforward ANN with 1 hidden layer of 30 neurons. There are 3 input parameters in the input layer and 1 output parameter in the output layer. The training algorithm used is resilient backpropagation. The activation function used is sigmoid for the hidden layer and a linear function for the output layer. The learning rate is 0.001. We use the ANN to obtain the execution times for the kernel with the tile dimensions from the pruned search space. Following this, 30 tile dimensions which have the best predicted execution times are chosen and the kernel is executed with these tile sizes and the best performing one is chosen.

V. EXPERIMENTAL SETUP

A. PLATFORM

Our experimental platform runs Linux powered by Intel Xeon X5680 running at 3.33 GHz. It has 2 sockets with 6 cores each. There are data caches of 32 KB/core for L1, 256 KB/core for L2, and 12 MB/socket for L3. The compiler used is GCC version 4.1.2 with option -O3. The model we propose is not dependent on the platform. It is portable across platforms due to the features used as input.

B. KERNELS USED

The kernels that we use for our experiments are fdtd-2d, gemm, jacobi-2d-imper, lu, syr2k, trmm, syrk, symm and ludcmp from the polybench-2.0 [5] benchmark suite. We use the Orio-0.2.1 for parametrically tiling the kernels.

C. OTHER PARAMETERS

The search space of tile sizes is constrained to 22 sizes along each dimension from the set {1, 2, 4, 6, 8, 10, 12, 16, 30, 32, 40, 48, 64, 100, 128, 150, 200, 256, 300, 400, 500, 600}. Since tiling is along 3 dimensions we get 10648 possible tile dimensions. The problem size for all the kernels is fixed at 600.

VI. RESULTS AND ANALYSIS

The results of our experiments are presented in the Tables 1 to 4. Table 1 shows the space optimal execution times of the kernels used for our experiments. Table 2 compares the execution times of the kernels run with tile sizes predicted by the KS model against those of kernels run with tile sizes predicted by the Hybrid model. As we see in the fourth column of Table 2 the deviation of the execution times is at most 14.44%. It must be noted that when the deviation in execution times is greater than 1%, it is due to the Hybrid model performing better. For instance, when the Hybrid model performs worse than the KS model the deviation is at most 0.86%. Table 3 compares the execution times of the kernels run with tile sizes predicted by the MFSTSS model against those of kernels run with tile sizes predicted by the Hybrid model. Table 4 shows the fraction of the search space that has to be traversed to obtain the optimal tile sizes for each of the kernels. We observe that in a few cases, at most little more than half the search space has to be traversed to obtain the optimal tile sizes using our model. But in most of the cases the fraction of the search space that has to be traversed is less than 4%.

Table 1. KERNELS AND THEIR SPACE OPTIMAL EXECUTION TIMES

KERNEL	SPACE OPTIMAL EXECUTION TIME (IN SECS)
fdtd-2d	0.073848
gemm	0.330515
Jacobi-2d-imper	0.026621

lu	0.086851
syr2k	0.416135
trmm	0.204085
ludcmp	0.098651
syrk	0.291083
symm	0.217621

Table 2. PERFORMANCE COMPARISON OF HYBRID ALGORITHM AND KS MODEL

KERNEL	EXECUTION TIME WITH TILE SIZE FROM KS MODEL (in SECS)	EXECUTION TIME WITH TILE SIZE FROM HYBRID MODEL (in SECS)	DEVIATION OF HYBRID MODEL TIME FROM KS MODEL TIME (in %)
fdtd-2d	0.074908	0.074908	0
gemm	0.377257	0.375856	0.371365
Jacobi-2d-imper	0.037897	0.037897	0
lu	0.089167	0.089167	0
syr2k	0.416176	0.416218	0.010092
trmm	0.310674	0.313334	0.856203
ludcmp	0.123618	0.123618	0
syrk	0.309863	0.291083	6.06074
symm	0.283041	0.242176	14.43784

VII. CONCLUSIONS AND FUTURE WORK

We have proposed a Hybrid model for TSS for CPU kernels that are independent of architecture. The model has two phases. Both the stages make use of an ANN. The first phase is independent of the kernel and therefore, a generic ANN can be constructed for this phase. The second phase is specific to kernels and the ANN constructed for this phase can be constructed for any arbitrary kernel. The input to our model is the number of non-prefetched features in the kernel. The first phase predicts a square tile size which is the input to the second phase. The final output of the model is a rectangular tile dimension. We have shown that our model can predict tile dimensions with which the execution times of the kernels deviate at most 14.44% from those predicted by the KS model which is based on exhaustive search. However, it must be noted that whenever the difference in execution times is greater than 1%, it is due to the Hybrid model performing better. For instance, when the Hybrid model performs worse than the KS model the deviation is at most 0.86%. Further, our model requires a traversal of at most 54.77% of the search space. In most of the cases the size of the search space that requires to be traversed is less than 4%. One important extension to this work is a generalization of the proposed model to a kernel independent case.

Table 3. PERFORMANCE COMPARISON OF HYBRID ALGORITHM AND MFSTSS MODEL

KERNEL	EXECUTION TIME WITH TILE SIZE FROM MFSTSS MODEL (in SECS)	EXECUTION TIME WITH TILE SIZE FROM HYBRID MODEL (in SECS)	DEVIATION OF HYBRID MODEL TIME FROM MFSTSS MODEL TIME (in %)
fdtd-2d	0.151556	0.074908	50.574045
gemm	0.933614	0.375856	59.741821
Jacobi-2d-imper	0.059707	0.037897	36.528380
lu	0.230973	0.089167	61.395055
syr2k	0.753884	0.416218	44.790180
trmm	0.614578	0.313334	49.016398
ludcmp	0.223976	0.123618	44.807479
syrk	0.492364	0.291083	40.880527
symm	0.614913	0.242176	60.616217

Table 4. FRACTION OF THE SEARCH SPACE TO BE SEARCHED FOR EVERY KERNEL

KERNEL	FRACTION OF THE SEARCH SPACE REQUIRED TO BE EVALUATED (IN %)
fdtd-2d	3.22
gemm	3.22
Jacobi-2d-imper	46.14
lu	0.25
syr2k	0.25
trmm	40.25
ludcmp	46.14
syrk	0.25
symm	54.77

REFERENCES

[1] Mohammed Rahman, Louis-Noël Pouchet, and P. Sadayappan. Neural network assisted tile size selection. In International Workshop on Automatic Performance Tuning (IWAPT2010), 2010.
[2] Markus Kowarschik and Christian Weiß. An overview of cache optimization techniques and cache-aware numerical algorithms. In Algorithms for Memory Hierarchies, 2002.
[3] Tomofumi Yuki Lakshminarayanan Renganarayanan Sanjay Rajopadhye Charles Anderson Alexandre E. Eichenberger Kevin O Brien. Automatic creation of tile size selection models. In CGO10 April 24-28, 2010, Toronto, Ontario, Canada, 2010.

[4] Louis-Noël Pouchet J. Ramanujam Atanas Rountev P. Sadayappan Sanket Tavarageri. Dynamic selection of tile sizes. In Proceedings of HiPC. 2011.
[5] www.cse.ohio-state.edu/ pouchet/software/polybench/.