Locality Optimization in Wireless Applications

Javed Absar
IMEC (KULEUVEN)
75 Kapeldreef
Leuven, Belgium
javed.absar@gmail.com

Min Li
IMEC (KULEUVEN)
75 Kapeldreef
Leuven, Belgium
limin@imec.be

Praveen Raghavan
IMEC (KULEUVEN)
75 Kapeldreef
Leuven, Belgium
ragha@imec.be

Andy Lambrechts
IMEC (KULEUVEN)
75 Kapeldreef
Leuven, Belgium
lambreca@imec.be

Murali Jayapala
IMEC
75 Kapeldreef
Leuven, Belgium
jayapala@imec.be

Arnout Vandecappelle
IMEC
75 Kapeldreef
Leuven, Belgium
vdcappel@imec.be

ABSTRACT

There is a strong need now for compilers of embedded systems to find effective ways of optimizing series of loop-nests, wherein majority of the memory references occur in the form of multi-dimensional arrays, indexed primarily with linear functions of iterators and parameterized constants. The reason for this are the new wireless standards, e.g. 802.11n, WiMAX, Bluetooth, HIPERMAN, 3GPP-LTE and WiBro, where the codes are predominantly of the type described above. These standards provide high bitrate and mobility but are also extremely power and performance hungry. For even wider commercial applicability of these standards it is important to optimize their power consumption. We propose a novel solution to multiple loop-nest optimization problem using the concept of constraints. Experiments show that our technique leads to 47.5% reduction in external memory accesses over state-of-the-art.

Categories and Subject Descriptors

B.3.3 [**Memory Structures**]: Performance Analysis and Design Aids—*Formal models*; D.3.4 [**Programming Languages**]: Processors—*compilers, optimization*

General Terms

Algorithms, Performance

Keywords

loop-nest, reuse, temporal, spatial, access, layout

1. INTRODUCTION

New wireless standards such as 802.11n, WiMAX, Bluetooth, HIPERMAN, 3GPP-LTE and WiBro provide high bitrate and mobility but are extremely power and performance hungry. For even wider commercial exploitation of the new wireless technologies in homes and business environments, it is necessary to optimize their implementations for power and performance. While previously ASIC solutions were popular, it is now clear that for the same device to handle several standards, a software solution i.e. Software-Defined-Radio (SDR) is the right way forward. SDR solutions can migrate between different wireless standards, depending on coverage and cost considerations, simply by loading different software. However, for an efficient SDR solution, the effectiveness of the embedded compiler is paramount.

In our study of the wireless communications standards, e.g. 3GPP LTE (Long Term Evolution) [8] and 802.11n [7], we found the source-codes to be dominated by series of loop-nests. In these loop-nests, several medium-sized, multi-dimensional arrays are referenced through linear functions of loop-iterators and constant parameters. Another important aspect is that the computation and data accesses are not concentrated in just one kernel but are spread across several loop-nests. Multimedia applications, that are often coupled with devices with wireless capabilities (e.g. cellular phones with MP3), also exhibit similar characteristics.

Techniques for data locality optimization of loop-nest has been well studied [2][18][1]. But so far the main focus has been on the single loop-nest problem. But real applications are more than just one loop-nest. When we have several loop-nests and decisions made in one loop-nest have an impact on the rest, finding a good solution becomes a major challenge. We propose a systematic and scalable technique here that improves upon the best known previous approach [9]. Previous approach solves the problem by ranking the loop-nests based on profiling information. This, however, can lead to a poor solution. In our approach, we take into consideration the degree to which a loop-nest is constrained, by data-dependences and reuse, to rank it appropriately. Comparing two loop-nests to figure which one is more constrained is a complex problem. It needs a deep understanding of exactly how and when a loop-nest is constrained. We believe that this paper provides new insights into this problem. At the same time, it gives an automated scheme that is currently being implemented on industry-level ORC-URUK compiler framework [3].

CODES+ISSS'07, September 30–October 3, 2007, Salzburg, Austria.

2. MOTIVATING EXAMPLE

The code section below is from Graham-Schmidt orthogonalization which is used in wireless modems for signal factorization[15]. For this code segment we need to find the right combination of loop and data-layout transformations that together provide best combination of temporal and spatial locality.

```
for( i = 0 ; i < n ; i++ ){ ...
  for(j = i+1 ; j < n ; j++ ){  //Loop-Nest I
    for(k = 0 ; k < n ; k++ ){
      R[i][j] =  R[i][j] + A[k][j] * Q[k][i];
  } } ...
  for(j = 0 ; j < n ; j++ ){   //Loop-Nest II
    for(k = i+1 ; k < n ; k++ ){
      A[j][k] = A[j][k] - Q[j][i] * R[i][k];
  } }
}
```

Ideally, the locality problem should be solved by looking at the whole program, composed of several loop-nests. The reason is that data-layout decisions made in one loop-nest have impact on the spatial locality in other loop-nests. However, as we will show later, solving for all the loop-nests together leads to an unscalable, combinatorially explosive problem. The best existing solution [9] works around this problem by firstly ranking the loop-nests based on profiling. Let us first use the technique proposed in [9]. Since there is no difference in the number of memory accesses made by the two loop-nests, we could start with either loop-nest.

Suppose we start with Loop-Nest II. Loop-Nest II has good temporal locality for array `Q` since the innermost loop `k` accesses the same element of `Q` in its consecutive iterations. Interchanging the loops `j` and `k` (i.e. making `j` innermost) would improve temporal locality for `R` but then the temporal locality for `Q` would be destroyed. Next, let us focus on the data-layout. Array `A` is accessed as row-major in Loop-Nest II. Therefore for good spatial locality, data-layout of `A` is set as row-major. Similarly, data-layout of `R` is set to row-major. `Q` has exploited temporal reuse and therefore its data-layout is kept open. Having fixed the data layout for `A` and `R`, the technique proposed in [9] propagates the data-layouts as constraints to Loop-Nest I.

In Loop-Nest I, however, array `A` is accessed as column-major (see Fig. 1). One could do a loop interchange to make the access of `A` as row-major, but this is not done since Loop-Nest I has temporal reuse for array `R` (both for read and write) over `k` which would be destroyed by the interchange. Note that it is more profitable to exploit temporal rather than spatial locality and the approach by [9] therefore gives preference to temporal and so will avoid interchanging the loop. Therefore, one has to settle with poor spatial locality for `A` in Loop-Nest I.

The source of the problem in the above case was: inappropriate ranking of loop-nests. Loop-Nest II is more flexible compared to Loop-Nest I because an interchange of the loops in Loop-Nest-II does not reduce the temporal locality. On the other hand, similar loop-interchange in Loop-Nest I decreases its temporal locality. Therefore, in our technique we would start with the more constrained loop-nest, i.e. Loop-Nest I. As can be seen in Fig. 1, this puts data-layout of `A` and `Q` to column-major to have good spatial locality. Next, we propagate these layouts to Loop-Nest II where we can now perform a loop interchange to convert the access order of `A` from row-major to column-major. As a result, we arrive at a better temporal and spatial locality solution compared to state-of-the-art.

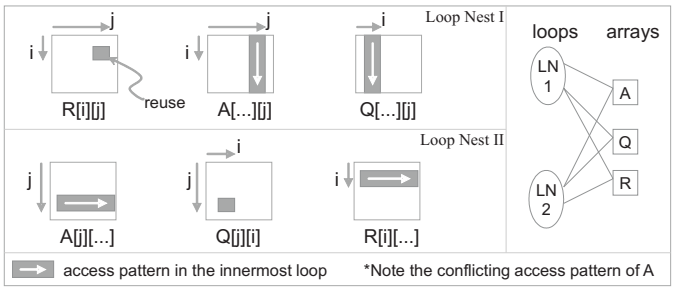

Figure 1: Array access pattern in Graham Schmidt Orthogonalization algorithm. Loop-Nest-I (LN1) is optimized first since it is more constrained. The data-layouts of the arrays are next propagated to Loop-Nest-II (LN2).

Here we used the simple concept of – the effect of interchange on temporal locality – to give an example of constraint. Later we will extend it to cover more general forms of loop-transformations. Also, the impact on more general data-layouts rather than just row and column major will be studied. Comparing sets of loop-nests to decide which ones are more constrained is a complex problem. We believe this paper provides new insights into this problem.

3. RELATED WORK

One of the fundamental breakthroughs in loop-nest optimization is due to Banerjee [2] which represents each iteration of the loop-nest as a vector in an iteration space. Loop transformations are then modeled as linear transformations of the iteration space using unimodular matrices. Li and Pingali extend the class to non-singular matrices [12], thereby allowing loop scaling. The legality of each transformation can be verified using dependence and direction vectors [1] [18], and in more complicated cases by the omega test [16].

In addition to reordering accesses [1, 2, 13], one can also modify the memory locations that are accessed by remapping the data elements. This is termed as data-layout transformation. One of the purpose of data-layout transformation is bringing close the data items that are also accessed close in time thereby improving spatial locality. For cache based system, good spatial locality means each cache line fetch contains useful data, while for scratchpad memories good spatial locality means fewer block transfers by the DMA. Extension to data-layout representation and transformation, beyond elementary row-major to column-major, was proposed in [11, 10] using hyper-plane equations.

As loop transformation changes the access order, it impacts both spatial and temporal locality. Cerniak [5] present an approach that integrates both data-layout and loop transformations. Their approach however restricts the search space by allowing only loop-interchange. Kandemir et al. [11] present an improved unified data-layout and loop transformation technique, for a single loop-nest, that allows all linear data-layout transformations. A scalable extension of the unified transformation technique to multiple loop nest is still in its infancy. Boyle et al. [14] illustrate with examples propagation of data-layouts. In [9], a concrete technique was proposed based on ranking of loop-nests using profiling. This approach has been discussed in detail in Sec. 2.

4. BACKGROUND AND NOTATION

A loop-nest of depth n is a finite convex polyhedron in the integer-space $\mathcal{Z}^n$, called the iteration-space [2]. The loop-bounds define the boundary of the polyhedron. Each iteration of the loop-nest then corresponds to a point in this polyhedron and is identified by its index vector $\vec{I} = [\ i_1 \quad i_2 \quad \cdots \quad i_n\ ]'$. The iterations are executed in lexicographical order. In the example below, arrays `A` and `B` are referenced inside a loop-nest of depth two, with the polyhedron being just a square.

```
for( i = 0 ; i < N ; i++)
   for( j = 0 ; j < N ; j++)
     A[i][j] = B[i+j] + 2 ;
```

Reference to an array made inside the loop-nest above can be represented with a reference matrix R and an offset $\vec{o}$. For instance, `B[i+j]` can be represented as: $R_B\vec{I} + \vec{o}_B = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix}$.

Let us now optimize the above code for temporal and spatial locality. Firstly, from a reference R, the reuse subspace $\vec{r}$ can be computed [18] as $\vec{r} = \ker R$, where ker is the kernel of the matrix. E.g. `B[i+j]` has reuse subspace $\vec{r}_B = \ker R_B = \alpha \begin{bmatrix} 1 & -1 \end{bmatrix}'$, where α is an arbitrary constant. The reuse subspace information can next be used to transform the loop-nest to improve temporal locality. A loop transformation is basically a linear mapping $\vec{I} \xrightarrow{T} \vec{I'}$ such that $\vec{I'} = T\vec{I}$, where T is an integer, invertible matrix and $\vec{I'}$ is the iteration vector after transformation. A reference $R\vec{I} + \vec{o}$ in the original loop nest will transform to $RT^{-1}\vec{I'} + \vec{o}$.

Now, good temporal locality will exist if consecutive iterations of the innermost loop access the same element of `B` after the transformation. In other words, we need to find a T such that $T\vec{r}_B = \begin{bmatrix} 0 & 1 \end{bmatrix}'$. Let $Q = T^{-1}$. Also, let $\vec{q}_2 = \begin{bmatrix} q_{12} & q_{22} \end{bmatrix}'$ represent the last column of the matrix Q. Now if we multiply both sides of $T\vec{r}_B = \begin{bmatrix} 0 & 1 \end{bmatrix}'$ by T^{-1} we obtain $\vec{r}_B = \vec{q}_2$. In other words, the reuse subspace vector forms the last column of Q. The rest of the columns of Q can be computed using matrix completion methods [4]. Therefore, $Q = \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix}$ and so $T = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$. By applying this transformation T to the loop-nest above we obtain:

```
for( i = 0 ; i < 2N-1 ; i++)
  for( j = max(0,i-N+1) ; j < min(N-1,i) ; j++)
    A[j][i-j] = B[i] + 2 ;
```

Now `B` has good temporal locality. As for array `A`, because loop transformation affects all references, its reference has changed from `A[i][j]` to `A[j][i-j]`. Assuming the default layout of row-major (C-Language), now `A` has poor spatial-locality since it is accessed in an anti-diagonal fashion.

However, we can improve the spatial locality of `A` by changing the order in which the array elements are stored. In this case the appropriate data-layout transformation matrix is $M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$, using the method explained in [10] using hyper-planes. Application of M to a reference $R\vec{I} + \vec{o}$ changes it to $M(R\vec{I} + \vec{o})$. So in this case `A[j][i-j]` reverts to `A[i][j]` after application of data-layout transformation M. From the above example we can conclude that loop and data-layout transformation applied together can improve spatial and temporal locality significantly.

5. COMPLEXITY ANALYSIS

Here we estimate the general complexity of the problem of locality optimization across several loop-nests. Suppose we have n loop-nests in an application and the iteration-space of the k^{th} loop-nest is $\vec{I}_k$. The arrays referenced in these loop-nests are $\{A_1, A_2, ..., A_m\}$.

Let transformation T_k be applied to loop-nest $\vec{I}_k$. If array A_p was accessed in $\vec{I}_k$ as $R_p\vec{I}_k + \vec{o}_p$, then after the transformation its index expression will become $R_pT_k^{-1}\vec{I'}_k + \vec{o}_p$. Let also data-layout transformation M_p be applied to array A_p. The new index expression is then $M_p(R_pT_k^{-1}\vec{I'}_k + \vec{o}_p)$. Now, to have spatial locality, the innermost loop in $\vec{I}'_k$ must access consecutive elements in same row of A_p. That is, to have spatial locality we need to satisfy the relation: $M_p\left(R_pT_k^{-1}(\vec{I}'_k + \vec{U}_{dim(\vec{I}_k)}) + \vec{o}_p\right) - M_p\left(R_pT_k^{-1}\vec{I}'_k + \vec{o}_p\right) = \vec{U}_{dim(A_p)}$, where $dim(\vec{I}_k)$ and $dim(A_p)$ are the dimensions of $\vec{I}_k$ and array A_p, respectively. $\vec{U}_d$ is a d-dim zero vector with last element as 1, e.g. $\vec{U}_3 = [\ 0 \quad 0 \quad 1\ ]'$. This spatial locality constraint further simplifies to:

$$M_pR_pT_k^{-1}\vec{U}_{dim(\vec{I}_k)} = \vec{U}_{dim(A_p)} \tag{1}$$

Eq. 1 provides an insight into the complexity of locality optimization problem. Suppose we fix the data-layout of array A_p to be M_p to have spatial locality for A_p in loop-nest $\vec{I}_k$. If A_p is also referenced in another loop-nest $\vec{I}_l$ and its spatial locality is poor in $\vec{I}_l$, then to improve it we can apply a loop transformation T_l to $\vec{I}_l$.

Next, suppose another array A_q is also accessed in both $\vec{I}_k$ and $\vec{I}_l$. T_l may destroy the spatial locality of A_q in $\vec{I}_l$. We can rectify that by applying M_q to A_q. Next, as A_q is also accessed in $\vec{I}_k$, if the spatial locality of A_q (with M_q) in $\vec{I}_k$ is also poor then a transformation T'_k must be applied (as layout of A_q is now already fixed). But now we have a problem of going in circles. The spatial locality of A_p in $\vec{I}_k$ may be destroyed by T'_k.

Conclusion: Decisions about M_p, M_q, T_k and T_l must be taken together as they affect each other. However, as M_p and T_k appear as product terms in Eq. 1, the problem at hand is at least as complex as integer quadratic constraint programming. Since T_k and M_p need to be invertible and integer-valued, the problem quickly becomes combinatorially explosive.

6. PROPOSED APPROACH

Having seen that locality optimization across multiple loop-nests is a complex problem, we will present in this section a set of assertions based on which a near-optimal and scalable solution is possible [1].

ASSERTION 1. *An improvement in temporal locality is preferred, in terms of energy and performance, over an improvement in spatial locality.*

[1]The proof is available and will be presented in the journal version

ASSERTION 2. *Temporal locality optimization decision in one loop-nest does not affect the temporal locality optimization decision in any other loop-nest.*

ASSERTION 3. *A loop-nest can always be transformed to provide spatial locality for a pre-defined data-layout if the loop-nest is not constrained. An array's data-layout can always be transformed to provide spatial locality for a pre-defined loop access order.*

Assertion 2 tells us that temporal locality optimization can be performed independently for a loop-nest without compromising temporal locality in any other loop-nest. Also, from Assertion 1 we see that temporal locality is more important than spatial. Therefore, if we tackle one loop-nest at a time and always give priority to temporal over spatial then we will still arrive at the best temporal locality solution in a very scalable way, while still being close to the overall optimal solution.

Next, let us look at spatial locality. Consider an array A_p in loop-nest $\vec{I}_k$. If the data-layout of A_p is not defined, then from Assertion 3 we know that we can always find a suitable data-layout so that A_p achieves good spatial locality in $\vec{I}_k$. If the data-layout of A_p is pre-defined but the spatial locality of A_p in $\vec{I}_k$ is not good, then Assertion 3 tells us that we can always find a loop transformation T_k to improve spatial locality, provided there are no constraints. There are basically two types of constraints that can prevent spatial locality optimization. We discuss them next.

6.1 Constraints of Data Dependences

In the code below, let us optimize LN1 (Loop-Nest I) first and then propagate the data-layouts to LN2. As LN1 has no reuse, we can only optimize for spatial locality. We set data-layout of B and C to row-major and column-major, respectively, to achieve good spatial locality. These layout are then propagated to LN2.

```
for( i = 0 ; i < 3*N ; i++) //Loop-Nest I
  for( j = 0 ; j < N ; j++)
    C[j][i] = B[i][j];
    ...
for( i = 0 ; i < N ; i++) //Loop-Nest II
  for( j = 0 ; j < N ; j++)
    A[i+1][j+1]=B[i+2*j][j]-A[i][j]-A[i][j+2];
```

In LN2, however, B is accessed in a semi-diagonal manner. By applying a transformation $T = \begin{bmatrix} 1 & 2 \\ -1 & -1 \end{bmatrix}$ to LN2, it is possible to change the access order such that B is accessed as row-major. The index expression of B after the transformation is $RT^{-1}\vec{I}' + \vec{o} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & -2 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} i' \\ j' \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} i' \\ i' + j' \end{bmatrix}$. That is, the new reference is $B[i'][i' + j']$. As now consecutive iterations of the innermost loop access consecutive elements in the same row, we have successfully obtained good spatial locality for B in LN2.

Let us now see if the transformation respects the data dependences. LN2 has two data dependences. There is a dependence between the write $A[i+1][j+1]$ and the read $A[i][j]$. Since the data written in iteration $[\ i \quad j\ ]'$ is read back in iteration $[\ i+1 \quad j+1\ ]'$, this dependence can be represented by the dependence vector $\vec{d}_1 = [\ 1 \quad 1\ ]'$. The second dependence $\vec{d}_2 = [\ 1 \quad -1\ ]'$ is between the write $A[i+1][j+1]$ and the read $A[i][j+2]$.

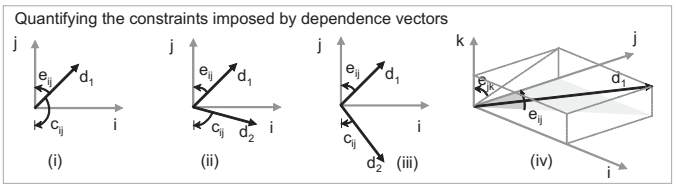

Figure 2: Data-dependences may limit the extend to which a loop-nest could be transformed to improve spatial locality. As all loop transformations to improve spatial locality rotate the dependence vectors, the constraints imposed can be measured using the angles subtended by the dependence vectors.

Given a dependence vector $\vec{d}$, a transformation T is valid only if [18]: $T\,\vec{d} \succ \vec{0}$. As $T\,\vec{d}_2 = \begin{bmatrix} 1 & 2 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \prec \vec{0}$, the transformation T turns out to be invalid. So we see that dependences can inhibit spatial locality optimization.

Now, transformations employed to improve spatial locality involve only rotation and reflection of the iteration space. For example, a loop-interchange involves a rotation by 90 degrees followed by a reflection on vertical axis. The rotation is to orient the access order to the way data is laid out for the array. Reflection as such does not create or destroy spatial locality. Whether an array is accessed as $A[0][1], A[0][2], A[0][3]$ or $A[0][3], A[0][2], A[0][1]$ is same for spatial locality.

Therefore, we only need a metric to measure how much data-dependences constrain spatial locality optimizations by limiting the rotation freedom of the iteration space. Fig. 2.i shows a dependence vector $\vec{d}_1$ in a loop-nest of depth two, that subtends an angle of e_{ij} with the j axis. A transformation that rotates $\vec{d}_1$ anti-clockwise by more than e_{ij} would be illegal as $\vec{d}_1$ then becomes lexicographically negative. Similarly, a rotation clockwise by more than c_{ij} would again be illegal.

Fig. 2.ii shows two dependence vectors $\vec{d}_1$ and $\vec{d}_2$. Note that the maximum rotation anti-clockwise is limited to e_{ij}. Similarly, clockwise rotation is limited to c_{ij}. We take the sum of these two and divide it by π.

The ratio $(e_{ij} + c_{ij})/\pi$, which we define as *dependence-ratio*, is always between 0 and 1 with higher values signifying more rotation freedom. Note that the dependences in Fig. 2.ii are less constraining than the dependences in Fig. 2.iii as the dependence-ratio in Fig. 2.ii has a higher value than the dependence-ratio in Fig. 2.iii. In a three dimension space, such as in Fig. 2.iv, the dependence-ratio can again be computed easily as $(e_{ij} + c_{ij} + e_{ik} + c_{ik} + e_{jk} + c_{jk})/3\pi$.

6.2 Constraints of Temporal Reuse

The freedom to transform a loop-nest to improve spatial locality can also be limited if spatial locality improvement comes at the cost of temporal locality. We classify constraints imposed by temporal locality into four *reuse-class* in increasing order of freedom.

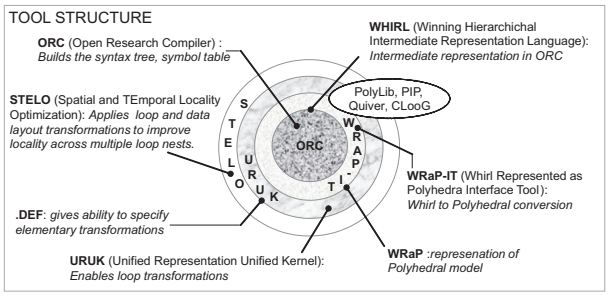

Figure 3: STELO (Spatial TEmporal Locality Optimizer) built with ORC (Open Research Compiler).

Class `SV` are loop-nests where there is just one *best* reuse option and it is spanned by a single reuse vector. Loop-Nest I in Sec. 2 has two reuse options: in references $Q[k][i]$ and $R[i][j]$. However, reference $R[i][j]$ appears twice, in read and in write, therefore there is only one best reuse option. The reuse vector that spans the reuse of $R[i][j]$ is $[\,0\ 0\ 1\,]'$. Temporal locality can be exploited in only one way for loop-nests in this reuse-class, which is when the best reuse vector is aligned parallel to the innermost loop.

Class `MV` contains loop-nests with multiple *equivalent* reuse opportunities but each reuse opportunity is still just a single reuse vector. Loop-Nest II in Sec. 2 has two reuse options: in reference $Q[j][i]$ and in $R[i][k]$. Both the reuse are equivalent in that they produce the same gains. Having multiple equivalent reuse vectors means more flexibility for spatial locality optimization.

Class `IV` contains loop-nests where the reuse is along two or more dimensions and is therefore spanned by more than one reuse vector. For example, a reference such as $D[i]$ in a loop-nest $[\,i\ \ j\ \ k\,]$ has reuse-space $\alpha[\,0\ \ 1\ \ 0\,] + \beta[\,0\ \ 0\ \ 1\,]$ since the reuse is both in j and in k direction. This class of loop-nests have more flexibility than the loop-nests in reuse-class `MV` since they provide in principle infinite choices for the exploitation of temporal locality.

Class `NV` contains loop-nests with no temporal reuse. Therefore the loop-nests can be transformed whichever way without affecting temporal locality.

6.3 Loop-Nest Optimization Algorithm

Fig. 4 presents our multiple loop-nests locality optimization algorithm. Let $L = \{l_1, l_2, ..., l_n\}$ be the set of loop-nests in the application. We partition L into four *reuse-classes* in the order of increasing flexibility as described in Sec. 6.2. Next, within each reuse-class we rank the loop-nests using dependence-ratios as described in Sec. 6.1. The different reuse-classes are then concatenated to form a complete ranked list of loop-nests with the most constrained loop-nest ranked first.

Next we start to optimize one loop-nest l_i at a time. The reuse vectors r_{ij} are used to design the loop transformation T_t which improves the temporal locality (when multiple reuse options are present, a set of T_ts are constructed).

Let Q be the set of arrays whose layout has been fixed. From the arrays in Q we select those which are referenced in l_i and for which temporal locality does not exist. We construct a transformation T_s which improves the spatial locality of such arrays. T_s should not undo the temporal locality achieved by T_t. The composite transformation $T = T_s * T_t$ is applied to l_i. Now, taking the new access order in

function: Locality optimization algorithm
let $L = \{l_1, l_2, ..., l_n\}$ be the set of loop-nests
let $A = \{a_1, a_2, ..., a_m\}$ be the arrays referenced in L
for $i = 1...n$ **do**
compute the reuse vectors $\vec{r}_{ij}$ for the loop-nest l_i
compute the dependence vectors $\vec{d}_{ij}$
compute the dependence-ratio using $\vec{d}_{ij}s$
endfor
partition L into four reuse-class:
`SV`: loop-nests with just one best reuse vector
`MV`: loop-nests with multiple equivalent reuse vectors
`IV`: loop-nests reuse along two or more dimensions
`NV`: loop-nests with no reuse opportunity
rank the loop-nests in each reuse-class using dependence-ratio
generate the new ranked list $L' =$ (`SV`,`MV`,`IV`,`NV`)
let Q be the set of arrays whose layout is fixed. Initialize $Q = \{\}$
for $i = 1...n$ **do**
using $\vec{r}_{ij}$ construct a T_t which optimizes temporal locality in l_i
construct a T_s that optimizes spatial locality for arrays ...
... in Q w/o destroying any temporal locality in l_i
apply composite transformation $T = T_s * T_t$ to l_i
fix layouts of arrays accessed in l_i to exploit spatial locality
add to Q the arrays whose data-layout were fixed in l_i
endfor
end `function`

Figure 4: Spatial and Temporal Locality Optimization Algorithm using Reuse-Class and Dependence-Ratio.

l_i as fixed, we fix the layout (to improve spatial locality) for those arrays accessed in l_i that are not in Q and for whom temporal locality does not exist. The arrays arrays whose layouts were fixed are added to Q.

7. EXPERIMENTAL RESULTS

The framework for our tool STELO (Spatial and TEmporal Locality Optimization) which implements the proposed technique is shown in Fig. 3. It is built on top of ORC [19], WRaP-IT and URUK [6][3] compiler. We applied our technique to real-life applications to evaluate the improvement. The following applications were used:

- **3GPP-LTE**: 3GPP enhancement of Universal Terrestrial Radio Access (UTRA) [8].
- **802.11n** : New Wi-Fi standard by IEEE LAN/MAN Standard Committee [7].
- **WB-AMR** : Wideband Adaptive Multi Rate speech coder, adopted by ITU-T as G.722.2.
- **Graham Schmidt** : Used in wireless to factorize signals into orthogonal components [15].

We compare our technique against the SOA (state-of-the-art) [9]. Improved locality leads to better cache performance. Our technique is at par with SOA in improving temporal locality. We, however, improve spatial locality significantly with our insights into dependence-ratio and reuse-class. The affects of better spatial locality become visible as we go to line (or block) size of two or more words (typically, line sizes are 8bytes-128bytes). Therefore, we demonstrate our improvement (Improved-SOA) by plotting the cache miss-rate against different cache line sizes as shown in Fig. 5. Note that miss-rate of Improved-SOA is significantly less than SOA. Even though SOA and Improved-SOA

improve temporal locality exactly to the same extent, we see Improved-SOA to be doing better because in most of these applications spatial locality plays an important part. Applications such as 3PP-LTE have small level of temporal locality and therefore our improved approach to spatial locality really makes a difference. The average reduction in miss-rate across all applications and line-sizes was found to be 47.5% over SOA.

Conclusion: We provide a scalable technique for locality optimizations of series of loop-nests such as found in wireless, that improves significantly upon the state-of-the-art. Our technique is based on new insights into role of dependence and temporal-reuse in constraining spatial locality optimizations. We propose new concepts of reuse-class and dependence-ratio that accurately measure how much a given loop-nest is constrained by data-dependence and reuse, so that it can or cannot be transformed further to improve layout locality for the arrays referenced inside this loop-nest. Our method is implemented on industry-level compiler framework.

8. ADDITIONAL AUTHORS

Francky Catthoor (email:catthoor@imec.be).

9. REFERENCES

[1] R. Allen and K. Kennedy. *Optimizing Compilers for Modern Architectures*. Morgan Kaufmann Publishers, 2001.

[2] U. Banerjee. *Data Dependencies*. Kluwer, 1988.

[3] C. Bastoul, A. Cohen, A. Girbal, S. Sharma, and O. Temam. Putting polyhedral loop transformations to work. In *Intl. Workshop on Languages and Compilers for Parallel Computers, LNCS 2958*, pages 209–225, 2003.

[4] A. J. C. Bik. *The Software Vectorization Book*. Intel Press, Reading, Mass., 2005.

[5] M. Cierniak and W. Li. Unifying data and control transformations for distributed shared memory machines. In *PLDI*, pages 205–217, 1995.

[6] A. Cohen, M. Sigler, S. Girbal and O. Temam. Facilitating the search for compositions of program transformations. In *Intl. Conference on Supercomputing*, pages 151–160, 2005.

[7] T. Cooklev. *Wireless Communication Standards: A Study of IEEE 802.11, 802.15 and 802.16*. IEEE Std. Assoc., 2004.

[8] IMEC. Imec software defined radio concept compliant with 3gpp-lte. *Design and Reuse, http://www.us.design-reuse.com/news/news13680.html*, (13680), 2006.

[9] M. T. Kandemir. A compiler technique for improving whole-program locality. *Proceedings of International Conference on Principles of Programming Language (POPL)*, 2001.

[10] M. T. Kandemir. Data relation vectors: A new abstraction for data optimizations. *IEEE Trans. on Computers*, 50(8):798–810, August 2001.

[11] M. T. Kandemir, J. Ramanujan, and A. Chowdhury. Improving cache locality by a combination of loop and data transformation. *IEEE Trans. on Computers*, 48(2), 1999.

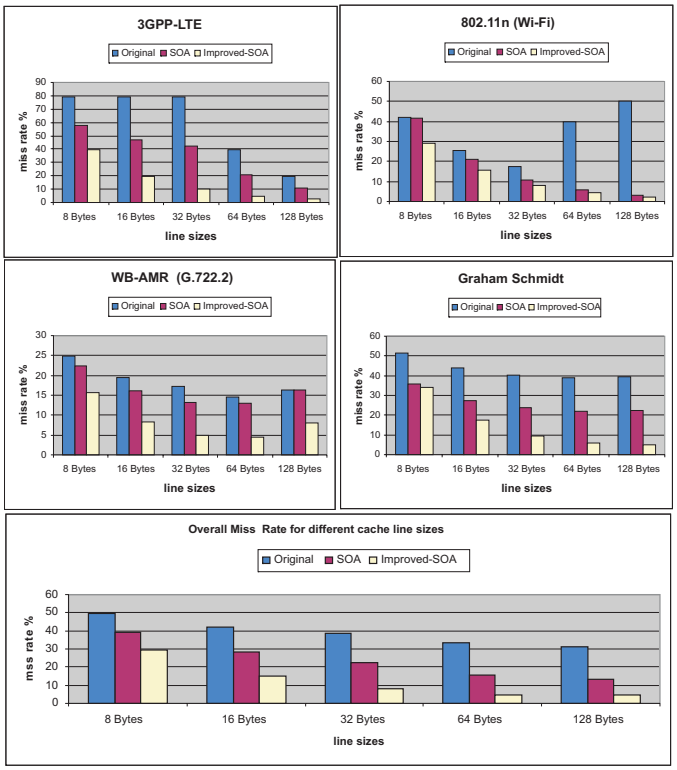

Figure 5: Miss-rate results for 3GPP-LTE (Long Term Evolution), IEEE 802.11n (Wi-Fi), WB-AMR (G.722.2) and Graham Schimdt. Note the reduction in miss-rate by our Improved-SOA over the SOA (state-of-the-art).

[12] W. Li and K. Pingali. Access normalization: loop restructuring for numa computers. *ACM Trans. Comput. Syst.*, 11(4):353–375, 1993.

[13] K. S. McKinley, S. Carr, and C.-W. Tseng. Improving data locality with loop transformations. *ACM Trans. Program. Lang. Syst.*, 18(4):424–453, 1996.

[14] M. F. P. O'Boyle and P. M. W. Knijnenburg. Non-singular data transformations: definition, validity and applications. In *Intl. Conference on Supercomputing*, pages 309–316, 1997.

[15] V. Poor and X. Wang. *Wireless Communications System: Advanced Techniques for Signal Reception*. Prentice Hall.

[16] W. Pugh and D. Wonnacott. Constraint-based array dependence analysis. *ACM Trans. Program. Lang. Syst.*, 20(3):635–678, 1998.

[17] C. Todd and G. Davidson. Ac-3: Flexible perceptual coding for audio transmission and storage. In *96th Convention of the Audio Engineering Society*, pages 89–102. AES, 1994.

[18] M. E. Wolf and M. S. Lam. A data locality optimizing algorithm. In *PLDI '91.*, pages 30–44, 1991.

[19] C. Wu, R. Lian, J. Zhang, R. Ju, S. Chan, and L. Liu. An overview of the open research compiler. *Lecture Notes in Computer Science*, 3602(2005):17–31, 2005.