Memory Hierarchy Usage Estimation for Global Loop Transformations

Qubo Hu[1] Erik Brockmeyer[2] Martin Palkovic[2] Per Gunnar Kjeldsberg[1] Francky Catthoor[2],[3]
[1] Norwegian University of Science and Technology, Trondheim, Norway
[2] IMEC, Leuven, Belgium [3] also at Katholieke Universiteit Leuven, Belgium

Abstract

Major parts of the power dissipation for data dominated embedded systems is due to huge amounts of data transfers to and from large power consuming data memories. Global loop transformations play a crucial role in optimizing the memory accesses. By improving regularity and temporal locality of these memory accesses using loop transformations, data can be potentially stored closer to the datapath in smaller less power consuming memories. To steer the selection of which loop transformations to perform, high level memory estimation is used. In state of the art memory estimation techniques the mapping of data to different parts of the memory hierarchy are not considered since these decisions are made in later system design phases. However, estimates of these mapping decisions is crucial, since they greatly influence the consequences of different transformations. In this paper we propose a systematic methodology for hierarchical memory usage estimation which considers also future data mapping decisions for a memory hierarchy. The goal of the estimation is to evaluate global loop transformation decisions for different platforms and keep only the optimal decision (in terms of energy) for each platform. We demonstrate our methodology on a real- life multimedia video coder (QSD-PCM) which shows that for 1k layer one memory, a factor of 2 improvement in total energy can be achieved.

1 Introduction

In todays embedded systems, the memory subsystem is rapidly becoming the bottleneck in terms of power, performance and area. This is especially the case for embedded multimedia and communication applications, which typically use a large amount of data storage and transfers. Thus, significant effort has been devoted to memory related optimizations to reduce the energy consumption and to optimize the global memory accesses. For example, this has led to the Data Transfer and Storage Exploration (DTSE) methodology developed at IMEC [1]. A crucial factor in the methodology is to improve the regularity and locality of memory accesses by doing loop transformations [2, 3] of the application. Fig. 1 shows an example of doing loop fusion transformation. In which two statements *S1, S2* are fused together to improve the temporal locality of the array accesses.

Traditionally, the data regularity and locality is exploited by using hardware-controlled caches, ensuring that most memory accesses are made to the fast and small caches. An alternative is to use software-controlled cache – scratch pad memories (SPM), between the large off-chip memories and the processor. SPM has no tag circuitry and are more power-efficient then a cache. For SPM mapping techniques are necessary to map the application data onto the memory hierarchy at compile time. This is done through memory hierarchy layer assignment (MHLA) [4]. In order to have efficient memory hierarchy exploration during the mapping, data reuse analysis (DRA) e.g. [5] is explored before mapping. Data reuse analysis searches for the frequently accessed data which is usually a part of an array (called copy candidate (CC)), and identify how often it is accessed. MHLA minimizes the cost function of the data memory hierarchy for a given application by selecting a set of CCs and assigning them together with original arrays to memory layers. By exploiting data reuse analysis and MHLA, parts of arrays (CCs) are copied from one layer to a smaller layer, closer to the datapath, from where it is read multiple times. As a result, energy can be saved since most accesses take place at the smaller copy and not on the large more energy consuming original array.

```
for i = 0 to 9
  for j = 0 to 9
    for k = 0 to 9
      for l = 0 to 2
S1:    ... = f1(A[i*10+k])
for i = 0 to 9
  for j = 0 to 9
    for k = 0 to 9
      for l ) 0 to 2
S2:    ... = f2(A[i*10+k])
```
a. before transforamtion

```
for i = 0 to 9
  for j = 0 to 9
    for k = 0 to 9
      for l = 0 to 2
S1:    ... = f1(A[i*10+k])
S2:    ... = f2(A[i*10+k])
```
b. after transforamtion

Figure 1: An example of program transformation

Currently, techniques for improving regularity and locality of memory accesses at the early loop transformation stage does not reflect how efficient the memory hierarchy can be exploited. The estimated size of one single memory is typically used to check the quality of a given combination of loop transformations. This is not sufficient, and it is crucial to have an estimate on the important memory hierarchy mapping decisions at this stage in order to select the optimal loop transformations. At this early system level design stage, there are usually no memory hierarchy decided yet. This means one loop transformations alternative can be best for a set of memory hierarchies but not for all possibilities. It is strongly desired to have a platform independent memory hierarchy usage estimation. In this paper, we present how to do such a fast platform-independent memory hierarchical usage estimation at the loop transformations stage. This has not been addressed yet in any current estimation approaches, which only consider one layer memory size estimation. Our method includes two main steps: incremental data reuse analysis and fast MHLA estimation.

The rest of the paper is organized as follows. Section 2 reviews related work. Our estimation method is described in Section 3. In Section 4, a case study with real life QSD-PCM application demonstrates the importance of the work. Conclusion is drawn in Section 5.

2 Related Work

2.1 Loop transformations

Loop transformations have been studied quite extensively in program optimization for parallelism and performance since the 1960s. For low power design, it has also been studied for the efficient use of a memory hierarchy for data dominated embedded applications by increasing access locality (see e.g. [6, 7, 8]). Applying transformations to improve the cache usage e.g. [9, 10] and SPM usage utilization e.g. [11, 12] has been addressed. In order to reduce the complexity and enable better heuristics, [2, 3] propose a two-phase approach by improving regularity and locality of data accesses. They still only consider one single memory, however.

2.2 Memory requirement estimation

Recently, memory estimation for array dominated applications has attracted attention [13, 14, 15, 16, 17]. Compared to traditional scalar-based memory estimation, all these methods consider arrays or a parts of arrays as a groups and try to minimize the memory requirement by taking into account the lifetime analysis of the groups. Different groups which are not simultaneously alive can share the same memory location. Although with different approaches, all of this work only considers one layer memory requirement estimation. This is not realistic since existing memory systems always have more than one layer.

2.3 Data reuse analysis and MHLA

As introduced, data reuse analysis finds potential copy candidates and MHLA makes the mapping decision of these candidates and original arrays to the memory hierarchy. [5, 18, 19] provide different methods for data reuse analysis. These methods may lead to different mapping result. [18] proposes a method to evaluate the lifetime of stencil elements. This means an exact analysis of reuse factor but it is high complexity for analysis and for generated code. [5] attempts to explore tradeoffs between SPM size and power, assuming an optimal run-time placement of data in SPM. But no technique is presented yet for automatic tool implementation. [19] does the analysis by evaluating the reuse distance of the same set of data between the current and next iteration of a certain loop level. It ignores boundary effects and results in simpler generated code. However, all these methods work on geometrical model[1] which is quite computation expensive.

Currently a new method is under development at IMEC that works on a simplified geometrical model which support only hypercubes. Compared to generic geometrical model in which operations are more computation expensive, operations with hypercubes are much simpler and faster. In fact, all polyhedrons can be simplified to hypercubes with some loss of accuracy. This method is very fast and is used in our work. It gains speed while the accuracy is expected acceptable for the purpose of estimation. The analysis of this method is currently done based on the source C-code and does not fit our purpose of estimation. It has to extract the C-code at each time loop transformations are done. In our approach we couple loop transformations and fast data reuse analysis. Thus there is no need for dumping C-code from the geometrical model and parsing C-code into the geometrical model for each transformation iteration.

[4] presents techniques for MHLA. It performs incremental layer assignment for a given memory hierarchy based on trade-offs of power, performance and size. Compared to this, we do a platform-independent MHLA estimation and our heuristic is faster as shown below.

[1]all called polyhedral model. It has been quite extensively used to model and analyze execution of program statements, in the parallel compiler and regular array synthesis domain. Explanations of it can be found in [1].

3 Our Estimation Methodology

In order to do a fast platform-independent hierarchical memory exploitation estimation, we propose a three-steps approach:

1. incremental data reuse analysis
2. fast MHLA estimation with Pareto curve output
3. comparison of Pareto curves for different versions of an application to obliterate un-necessary transformed versions

3.1 Data reuse analysis

The data reuse analysis is done for each array separately. One array analysis can be done either for each reference individually or between different references. The analysis is proceeded at one loop level at a time, starting from outer most iterator (iterator i for the example code shown in Fig. 1.a) to the inner most iterator (l in Fig. 1. a). For a given loop level, we analyze reuse between different sets of data which are accessed at continuous iterations of this loop level. For iterator j of the array A referenced in statement $S1$ in Fig. 1.a, the procedure will be as follows. For a certain iteration of iterator j, we assume that the ancestor iterators (in this case i) do not change their values and descendant iterators (k and l) are iterating over their complete loop bounds. In case iterators i and j are assigned their lower bound iteration values (0), the range of values of the index expression of array A referenced in statement $S1$ is from 0 to 9 (expressed as $[0-9]$). It indicates the set of data accessed when i and j is 0. Since we in this analysis assume that i is unchanged, exactly the same data will be accessed for any other value of iterator j. The set of data is consequently reused at each iteration of j, indicating there exists an interesting CC at this loop level. The intersection for any two continuous sets has the same index range $[0-9]$. The intersection part is called the reuse part D_{reuse} as it can be reused at next iteration. Sometimes for the next iteration a set of new data is needed in addition to D_{reuse}. This set must be fetched before it is used and is called the update part D_{update}. In our example D_{update} is empty. The buffer size required for a CC equals the size of the set of data elements (10 in our example). The number of total transfers needed (called misses) for this CC A'_{S1} from original array is calculated based on the formula below and equals to 100 as shown to the left in Fig. 2.a:

$$\boxed{\#\mathbf{misses} = (\#\mathbf{D}_{reuse} + \#\mathbf{D}_{update} * \#\mathbf{iter}_{current}) * \#\mathbf{iter}_{ancestors}}$$

in which $\#\mathbf{iter}_{current}$ means the number of iterations at this loop level and $\#\mathbf{iter}_{ancestors}$ means the number of iterations of all ancestor loop levels.

In the same way, our analysis detects that there are another CC A''_{S1} at l-iterator for this reference in statement $S1$

since here also the same data is reused several times. Array A is referenced a second time in statement *S2*. This is analyzed separately as there are no reuse benefits between the two references in version a. of our code example. So now, the different CCs and the original array form a data reuse chain. This is annotated with information about the size requirement and the number of misses needed from the higher memory hierarchy array. For a given application, data reuse chains for each of its arrays together form a data reuse tree for the application. In our example, the data reuse tree in Fig. 2.a consists of just one data reuse chain. Fig. 2.b shows the reuse tree for the transformed version of the code (Fig. 1.b) in which the two references of array A are considered together. It results in 50% less misses in total for the two CCs A' and A'' compared to the total misses at the same loop level for the version before transformation.

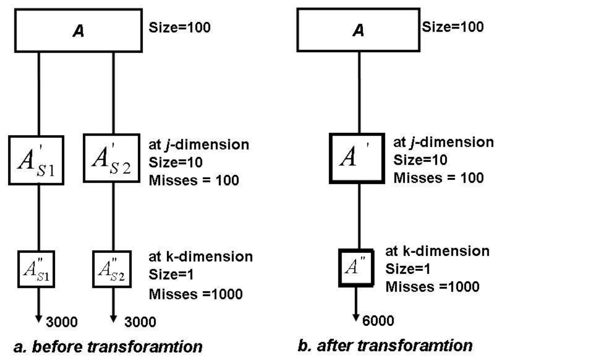

Figure 2: Data reuse trees for the two versions of the code

Here we emphasis how to do incremental data reuse analysis. When the loop transformations proceed, we know which statement and which loop levels that are changed. Then instead of computing a new reuse tree for the transformed code, we recompute only the data reuse chains of the transformed arrays. This approach can gain speed for real life applications when there is just a limited number of transformed arrays at a time while the huge number of other arrays is un- changed. In fact, the speed of the data reuse analysis can be further increased by only updating the reuse tree at the iterator levels for which the loop transformation changes the CCs.

3.2 MHLA estimation

Based on the data reuse tree achieved, we propose a fast platform independent MHLA estimation approach. The idea is to calculate a set of Pareto points assuming a two layer memory hierarchy: the main memory and a SPM layer. The MHLA estimation is done incrementally. When a CC/array is assigned to the SPM, we get a new Pareto point based on the previous one. The assigned CC/array contributes to the increased size and the decreased misses for the new point. When all CCs and arrays are considered, we get a set of Pareto points consisting of a pseudo Pareto curve. Different loop transformations alternatives result in different Pareto curves.

Since it is time consuming to estimate optimal layer assignment solution, a steering heuristic is used to speed it up. At each time, the CC/array having the highest access over size ratio is considered for assignment to the SPM layer. For the selected one, it will replace the pre-assigned CCs if the pre-assigned CCs are for the same reference and derived from inner loop levels (we call such a pre-assigned CC its child and it is the parent of the pre-assigned ones). The size and miss effect of its children will be taken out of the calculations. If the CC currently being considered has a parent CC/array already assigned to the SPM, the current CC will not be assigned. Fig. 3 shows the MHLA estimation for the two versions of example code. **#misses** is the number of transfers needed for the assigned CCs/array from the original array.

Our estimation method is fast. It takes only seconds for the QSDPCM applications illustrated in following section. For example the complexity of the heuristic used in [4] is $\#2^{\mathbf{n}}\mathbf{n}^2 log\mathbf{n}$ for a two layer memory hierarchy. $\mathbf{n}$ is here the total number of CCs and arrays in the data reuse tree. Compared to this, the complexity of our heuristic is $\mathbf{n}\ log\mathbf{n}$, obviously much faster. In [4], the complexity for general memory hierarchy would be $\#\mathbf{layer}^{\mathbf{n}}\mathbf{n}^2 log\mathbf{n}$ in which # **layer** means the number of layers in the memory hierarchy. This is too high to be accepted for our purpose. For realistic memory hierarchies, our estimated number of misses for a given SPM size is near independent of the hierarchy level at which the SPM is located. We can therefore simulate any realistic memory hierarchy with any number of layers and sizes without increased estimation complexity.

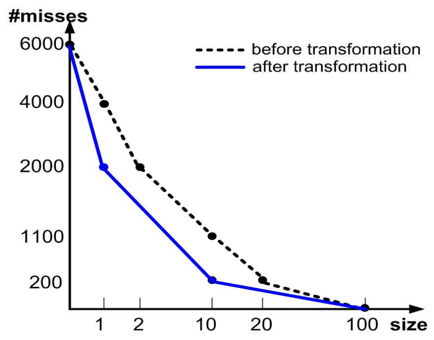

Figure 3: MHLA estimation output for two versions of the code

3.3 Comparison of the possible transformation solutions

Based on the different pseudo Pareto curves, a comparison is done to eliminate all uninteresting transformed versions of the application. If the points of one curve are always below the points of others for the same size, this curve should be kept as it indicates that this version of application will lead to the best memory exploitation for any kind of hierarchy. For any memory hierarchy with any number of layers of any size, this version will result in less access over size ratios and hence lower energy consumption. As shown in Fig. 3, the curve of the transformed version of our example is such a case. Typically there are no single curve which is below all others all the time. This means that one version of the application can be good for some memory hierarchies while other versions can be good for other cases. The easy way is to keep all these versions. At a later stage, the optimal solution can be found when the memory hierarchy is specified.

4 Case Study with QSDPCM

The Quadtree Structured Difference Pulse Code Modulation (QSDPCM) algorithm is an inter-frame compression technique for video images. It involves a hierarchical motion estimation step and a quadtree based encoding of the motion compensated frame-to-frame difference signal [20]. Fig. 4

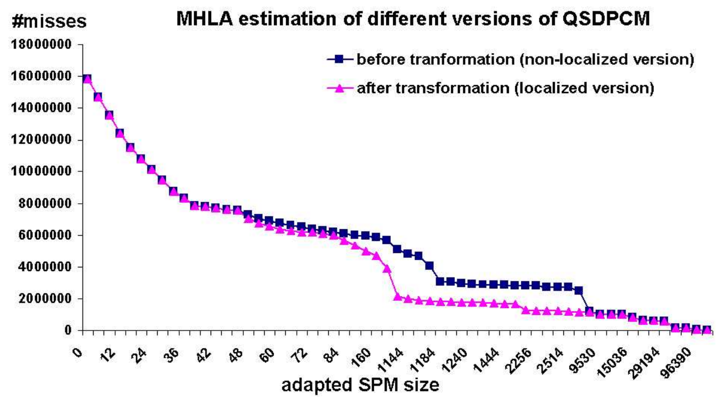

Figure 4: MHLA estimation output of two versions of QSDPCM application

shows the MHLA estimation result with two pseudo Pareto curves output for two versions of the QSDPCM application. The number of misses is decreasing as more CCs/arrays are assigned incrementally and the size of the SPM is increasing. In this case, the localized version is better then the non-localized version as the corresponding Pareto curve is always below the other for all SPM sizes. Fig. 5 shows a comparison of total dynamic energy consumed by these two versions of the application for a number of two layer memory hierarchies (the size in the figure is the size of SPM while the main memory size is assumed constant). As illustrated, the total power drops dramatically with data reuse analysis and MHLA exploration by adding the extra SPM layer between main memory and datapath. At a given point the curve rises because the energy cost of increased SPM size counteracts the benefit of the reduction in misses. The loop transformations for the transformed version enable further improvement and it can help the designer to find the lowest energy consuming memory hierarchy with SPM size set to 1024 bytes.

5 Conclusion and Future Work

This paper presents a systematic methodology for doing fast platform-independent memory hierarchy usage estimation at the loop transformations stage. The estimation evaluates the explicit effects of loop transformations on memory hierarchy usage. Using the QSDPCM application we demonstrated how the estimation can steer the selection of loop transformations towards an optimal solution. A prototype tool is currently being developed.

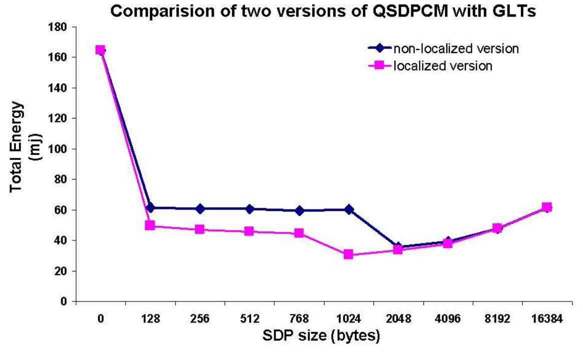

Figure 5: Energy comparison of two versions of QSDPCM application

References

[1] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle. *Custom Memory Management Methodology – Exploration of Memory Organisation for Embedded Multimedia System Design.* Kluwer Acad. Publ., Boston, USA, 1998. ISBN 0-7923-8288-9.

[2] K. Danckaert, F. Catthoor, and H. De Man. A loop transformation approach for combined parallelization and data transfer and storage optimization. In Proc. ACM Conf. on Par. and Dist. Proc. Techniques and Applications, PDPTA'00, pages 2591–2597, Las Vegas NV, USA, June 2000.

[3] S. Verdoolaege, K. Danckaert, F. Catthoor, M. Bruynooghe, and G. Janssens. An access regularity criterion and regularity improvement heuristics for data transfer optimization by global loop transformations. In *In 1st workship on Optimization for DSP and Embedded Systems (ODES)*, March 2003.

[4] E. Brockmayer, M. Miranda, H. Corporaal, and F. Catthoor. Layer assignment techniques for low energy in multi-layered memory organizeations. In *DATE'03*, Munich, Germany, March 2003.

[5] T. V. Achteren, F. Catthoor, R. Lauwereins, and G. Deconinck. Search space definition and exploration for nonuniform data reuse opportunities in data-dominant applications. *ACM Trans. on Design Automation of Electronic Systems*, 8(1), 2003.

[6] A. Darte and Y. Robert. Affine-by-statement scheduling of uniform and affine loop nests over parametric domains. *Journal of Parallel and Distributed Computing*, 29(1):33–59, 1995.

[7] M. E. Wolf and M. S. Lam. A data locality optimizing algorithm. In Proc. of the ACM SIGPLAN Conference on Programming Language Design and Implementation, pages 30–44, 1991.

[8] P. Feautrier. Some efficient solutions to the affine scheduling problem. *International Journal of Parallel Programming*, 22(5):313–348, 1992.

[9] M. Kandemir, J. Ramanujam, and A. Choudhary. Improving cache locality by a combination of loop and data transformations. *IEEE Trans. on Computers*, 48(1):159–167, Feb. 1999.

[10] P. R. Panda, N. D. Dutt, and A. Nicolau. Memory organization for improved data cache performance in embedded processors. In *Intnl. Symp. on System Synthesis*, pages 90–95, La Jola CA, 1996.

[11] M. Kandemir, I. Kadayif, A. Choudhary, J. Ramanujam, and I. Kolcu. Compiler-directed scratch pad memory optimization for embedded multiprocessors. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 12(3):281–287, Mar. 2004.

[12] P. R. Panda, N. D. Dutt, and A. Nicolau. Efficient utilization of scratch-pad memory in embedded processor applications. In *DATE'97*, Paris, France, 1997.

[13] F. Balasa, F. Catthoor, and H. De Man. Background memory area estimation for multi-dimensional signal processing systems. *IEEE Trans. on VLSI Systems*, 3(2):157–172, June 1995.

[14] Y. Zhao and S. Malik. Exact memory size estimation for array computation without loop unrolling. In Proc. 36th ACM/IEEE Design Automation Conf., pages 811–816, New Orleans LA, USA, June 1999.

[15] P. Grun, F. Balasa, and N. Dutt. Memory size estimation for multimedia applications. In Proc. ACM/IEEE Wsh. on Hardware/Software Co-Design (Codes), pages 145–149, Seattle WA, USA, March 1998.

[16] P. G. Kjeldsberg, F. Catthoor, and E. J. Aas. Data dependency size estimation for use in memory optimization. In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pages 908 –921, July 2003.

[17] Q. Hu, M. Palkovic, and P. G. Kjeldsberg. Memory size optimizaton and estimation with loop fusion and loop shifting. In Proc. EUROMICRO Conf., Rennes, France, September 2004.

[18] K. Beyls and E. D'Hollander. Reuse distance-based cache hint selection. In B. Monien and R. Feldmann, editors, *Proceedings of the 8th International Euro-Par Conference*, volume 2400, pages 265–274, Heidelberg, 8 2002. Springer-Verlag.

[19] I. Issenin, E. Brockmeyer, M. Miranda, and N. Dutt. Data reuse analysis technique for software-controled memory hierarchies. In *DATE'04*, France, April 2004.

[20] P. Strobach. Qsdpcm – a new technique in scene adaptive coding. In Proc. 4th Eur. Signal Processing Conf. (EU-SIPCO), pages 1141–1144, Grenoble, France, 1988.