Enhancing Non-Linear Kernels by an Optimized Memory Hierarchy in a High Level Synthesis Flow

Stéphane Mancini, Frédéric Rousseau TIMA laboratory - CNRS, Grenoble INP, UJF
46, Av Félix Viallet, 38031 Grenoble Cedex, France

***Abstract*—Modern High Level Synthesis (HLS) tools are now efficient at generating RTL models from algorithmic descriptions of the target hardware accelerators but they still do not manage memory hierarchies. Memory hierarchies are efficiently optimized by performing code transformations prior to HLS in frameworks which exploit the linearity of the mapping functions between loop indexes and memory references (called linear kernels). Unfortunately, non-linear kernels are algorithms which do not benefit of such classical frameworks, because of the disparity of the non-linear functions to compute their memory references.**

In this paper we propose a method to design non-linear kernels in a HLS flow, which can be seen as a code pre-processing. The method starts from an algorithmic description and generates an enhanced algorithmic description containing both the non-linear kernel and an optimized memory hierarchy. The transformation and the associated optimization process provides a significant gain when compared to a standard optimization. Experiments on benchmarks show an average reduction of 28% of the external memory traffic and about 32 times of the embedded memory size.

I. Introduction

Memory management in embedded systems is an ever increasing challenge as the "memory wall" is becoming a bottleneck. Indeed the latter corresponds to the gap between the low performances of external memories and the increasing computing power available thanks to the integration density of chip technology. Generally speaking a memory hierarchy enables to overcome the "memory wall" and allows trade-offs along performance metrics such as chip area, power consumption, latency, real time constraints and so on.

A design flow should provide an efficient solution to cope with the "memory wall" at the highest abstraction level. Unfortunately, industrial High Level Synthesis (HLS) design frameworks do not allow to optimize memory hierarchies [1]. These frameworks efficiently manage embedded SRAMs but fail to deal with external low cost memories such as xDDR-SDRAMs. However there are several research frameworks dealing with the optimization of the implementation of linear kernels which are algorithms made of loop nests where data references are linear functions of the loop indexes [2], [3], [4], [5], [6].

Non-linear kernels are algorithms made of references which are non-linear functions of the loop indexes. This specificity prevents the use of research and industrial above-mentioned frameworks. However these non-linear kernels are important stages in many signal and image processing algorithms such as optical distortion [7], large angle acquisition (fisheye camera) [8]. Some works [9] have tried to approximate such non-linear kernels with linear kernels but the used methods have led to overestimations.

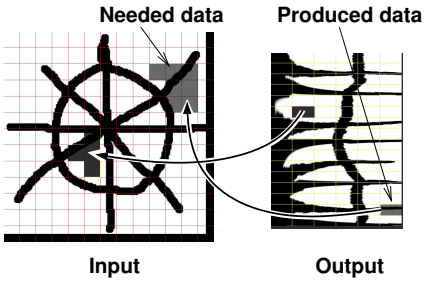

Fig. 1. The disparity of non-linear kernels, namely a polar transform in this picture, makes standard optimizations inefficient.

In this paper we propose a method and a tool to optimize the memory management at system level of non-linear kernels in the context of a HLS flow. The main idea is to start from a behavioral description of the non-linear kernel and to produce an optimized behavioral description of both a kernel implementation and an optimized memory hierarchy suitable for a C-to-RTL HLS tool. The proposed method can be seen as a source-to-source transformation for C-to-RTL pre-processing.

This optimization process at the system level helps the designer to find the best trade-off between the embedded memory area, the computing time, the traffic to the main external memory and the power consumption. Low level optimizations such as pipelining are left to the HLS tool.

The second section provides an insight into the problematic of optimizing the memory hierarchy of non-linear kernels. A general description of the proposed method is made in the third section. The fourth section presents the obtained results on a set of benchmarks followed by conclusions and perspectives.

II. Problematic

A non-linear kernel is an algorithm for which data references are computed thanks to non-linear functions of the loop indexes. Figure 1 provides an example of a non-linear kernel, namely the polar transform. To compute a pixel $O(x, y)$ of the output image, the kernel needs the pixel $I(x', y')$ of the input image, with the non-linear mapping function $(x, y) \mapsto (x', y') = (x\cos(\alpha y\pi), x\sin(\alpha y\pi))$. The pixels of the output image are computed by iterating over

978-3-9810801-8-6/DATE12/©2012 EDAA

the coordinates (x, y). For each loop iteration the kernel makes a reference to the input data $I(x', y')$.

The main difficulty to optimize the memory management of non-linear kernels relies in the disparity of the function to compute the indexes of the input data. This is very different from the uniformity of kernels with linear indexing. A common strategy to increase the data reuse and to optimize a memory hierarchy is to tile the iteration space [3], [4] (which is also called loop blocking). Then the output data is produced tile by tile, one after the other. When the input data is tiled, it is possible to determine which input tiles are needed to compute an output tile. In Figure 1 we can see that the set of input tiles (left image) required to compute an output tile (right image) varies both in shape and in area.

The disparity of memory references prevents the use of standard memory optimization frameworks which assume that the memory references are linear with the loop indexes [3], [4]. As an example, in such a classical framework, the polyhedral model represents the loop nests and memory references by algebraic expressions. The memory resources are then optimized by performing algebraic transformations on these expressions. Performing loop blocking with linear laws leads to tiles that are equal by translation and the set of input tiles to compute an output tile is constant in shape and area.

In order to deal with non-linear kernels, one could set up an approximate model to fit with the linear constraints of such frameworks [9]. Doing so results in a dramatic overestimation of the memory requirements because these tools conform with the very worst case.

Unfortunately, the standard C-to-RTL HLS flows do not deal with the optimization of the memory management at a system level [1]. However, when dealing with large multi-dimensional arrays, data are stored in a large low cost external memory such as a xDDR-SDRAM. A memory hierarchy is then introduced to copy parts of the data in more expensive but faster embedded memories. Many if not all C-to-RTL tools do not allow to manage such a memory hierarchy and they make the assumptions that the needed input data are present in embedded SRAMs at the beginning of a computation .

The main idea to deal with the disparity of non-linear references is to set up an optimization flow to parametrize a template architecture. The architecture template provides a generic memory hierarchy connected to processing units. The optimization flow produces a schedule of the computations in order to optimize the performance criteria. The generated hardware accelerator is sufficiently regular to fit with HLS tools constraints and and copes with the irregularity of non-linear kernels.

III. General view

The general view of the proposed method is illustrated in Figure 2. The method starts from a high level code of the kernel, such as a C/C++ code. A first transformation applied on the entry code produces an instrumented code in order to profile the memory references. This instrumented

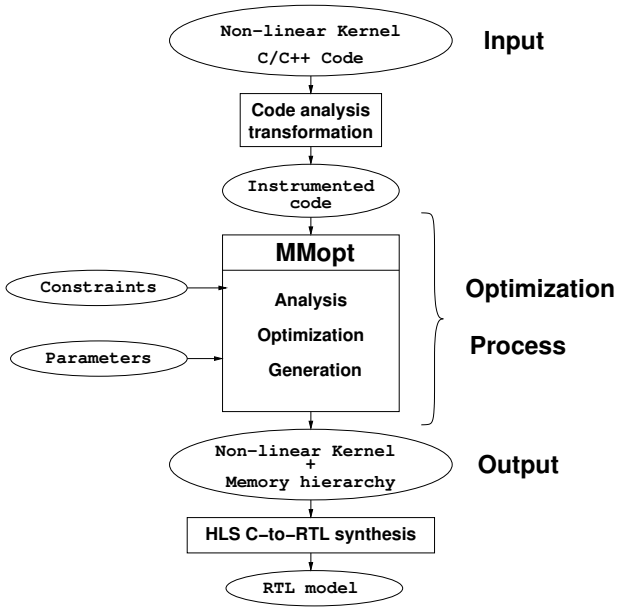

Fig. 2. General view of the joint HLS and memory optimization method.

code is the entry point of the optimization process. The optimization starts by profiling the kernel references and catch them in an internal representation. On the one hand, the system level optimizations take into account some system parameters such as the bandwidth and latency of the external memory and on the other hand they depend on some user constraints such as the maximum computing time when real time constraints are mandatory. At last a parametrized code is generated as the entry point of the C-to-RTL HLS tool.

The basis of the optimization is to tile both the iteration space of the kernel and the input and output data structures[1]. This tiling allows to store the whole input data in a low cost huge external memory whereas the generated kernel engine architecture stores only the small set of input data tiles needed to compute an output tile. Pre-fetching the input tiles fits with the constraints of HLS tools requiring that the input data is present at the beginning of a computation. Due to the disparity of non-linear kernels, the amount of embedded memory needed to store the input tiles as well as the traffic to the external memory highly depends on the schedule of the computations.

The architecture template is made of a Processing Engine$^{\delta}$ (PE) connected to a Memory Hierarchy Controller$^{\beta}$ (MHC) to form a Tile Processing Unit (TPU) as shown in Figure 3. The PE implements the kernel and requests the data to the MHC which manages a set of Input Tile Buffers$^{\phi}$ (ITB). The buffers are addressable through a level of indirection, called the Indirection Table$^{\epsilon}$ (IT), in order to cope with the varying sharing of the buffers between used (by the PE) and pre-fetched (by the MHC). Along with the computations, the buffer indirection is stored in the IT and updated by the MHC from a set of Compressed Configurations$^{\alpha}$ (CC) memories. These

[1] In many kernels the iteration space fits the indexes of the output data.

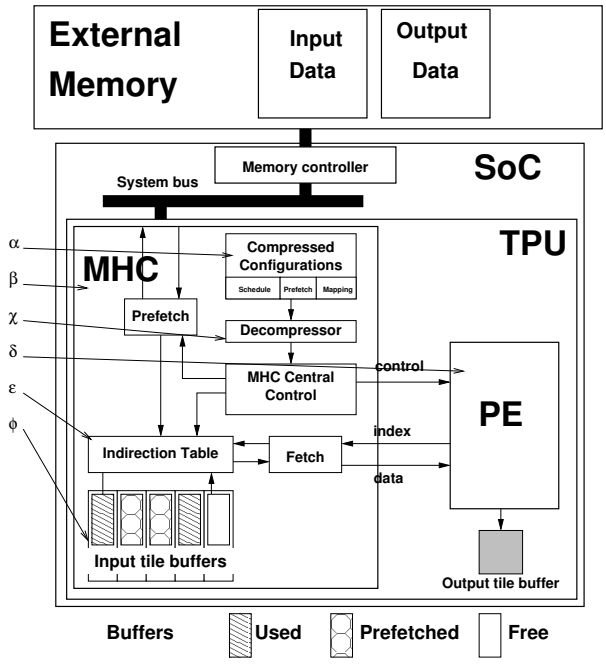

Fig. 3. Architecture template. The fixed number of input tile buffers is dynamically split over Used, Pre-fetched and Free tiles.

later are compressed off-line and are decompressed on the fly thanks to a low cost hardware Run Length Encoding (RLE) Decompressor$^{\chi}$.

The computations of the PE overlap the pre-fetches performed by the MHC. The output tiles are computed one after the other by the PE. Within a computation step, after a short setup phase, the PE requests data in the ITBs. Meanwhile the MHC pre-fetches the input data tiles for the next computations. To reduce the memory resources, the optimization process breaks this scheme by inserting fake computations sometimes. Fake computations may be replaced by wait states or pipeline freeze when the target HLS tool provides a fine control of the generated hardware. With fake computations, new trade-offs between the embedded memory area and the computing time are enabled. The result section will show that a sparse use of fake computations saves consequent area.

The optimization process generates a static parametrization of the MHC and a set of compressed configurations stored in the MHC. The CC memories store (i) the output tiles schedule, (ii) the pre-fetched tiles at each step and (iii) the mapping of tiles to buffers. To save embedded memory, these tables are compressed thanks to a RLE encoder.

Actually the resulting TPU model is a transformation of the input kernel. The original iteration loop is replaced by a two fold loop nest: the PE implements an inner loop nest inside a tile and an outer loop corresponding to the generated schedule is managed by the MHC. The HLS tool synthesizes the whole TPU and optimizes the resulting pipeline from the computation of the requested data indexes, the IT access, up to the processing of the data coming from the ITB. The small static range of the most inner loop makes the optimization process at the C-to-RTL level easier. The final RTL architecture is much more efficient than a basic connection of modules designed aside.

IV. Experiments and results

The MMopt tool implements the optimization process and have been stressed on a set of 10 benchmarks from real-life non-linear image processing kernels [8], [10]. As summarized in Table I, the benchmarks are variations of three kernels for which the input data structure is modified in order to improve the resulting image quality. Indeed, non-linear kernels need to be preceded by a space variant filter of the input data to cope with the varying sampling rate. This filter can be advantageously replaced by a multi-resolution (an)isotropic mipmap input data and a suitable modification of the kernel to select the appropriate resolution depending on the sampling rate [11].

The gains and penalty of the MMopt optimization given in Table I are computed relatively to a standard optimization made of a linear scheduling and a direct modulo mapping of buffers. The trade-offs are tuned to minimize the area, which maximizes the inserted fake computations. The "Processed tile increase" is the relative amount of inserted fake computations to reduce the memory resources. The "Traffic saving" corresponds to reduction of the loaded input tiles achieved by the scheduling. The "Memory reduction" is relative (in times) to the memory necessary for a standard linear schedule. For each kernel the measures are averaged on a set of tilings. The last line of the table provides the average gains for all the kernels.

As illustrated in Table I, comparing the results of MMopt with a standard HLS tool such as CatapultC is not relevant. Indeed, when considering HLS tools the kernel is then synthesized without a memory hierarchy and the best timings are obtained by embedding all the data memory in the hardware accelerator, which is not conceivable.

In Table I the embedded memory is reduced by 32 times in average. This average gain does not take into account the kernel for which the gain reaches 1264 times. This kernel presents a huge amount of disparity due to references in very different levels of resolution in the input mipmap data. The traffic to the external memory is reduced up to 38%, with an average of 28%. Minimizing the embedded memories leads to a 14% increase of the computing time. These measures validate the optimization process and the post-optimization stages.

As a proof of concept, the polar transform TPU has been generated and synthesized with the CatapultC HLS C-to-RTL tool from Mentor Graphics. In a Virtex 4 technology, the MHC occupies around 300 FG and the PE occupies around 1800 FG. The PE unit grabs one input data each clock cycle, which corresponds to the maximum available throughput when the buffers are made of single port memories. The HLS tool allows to optimize the TPU at the RTL level and pipelines the local memory references.

There is little to no similar works on the optimization of non-linear kernels. To our knowledge, the closest work [9] implements a fisheye correction unit within the

Kernel	Input data type	Input data dimension	MMopt versus CatapultC Memory reduction	MMopt versus Linear scheduling Memory reduction (times)	Processed tile increase	Traffic saving
fisheye	image	2D	94	16	19 %	29%
fisheye	mipmap anisotropic	4D	>> 100	28	19 %	34%
polar	image	2D		34	26 %	27%
polar	mipmap anisotropic	4D		41	15 %	18%
polar	mipmap anisotropic	2D (flat)		568	08%	21%
polar	mipmap isotropic	3D		21	02 %	38%
polar	mipmap isotropic	2D (flat)		1264	02 %	37%
pseudolog	image	2D		24	20 %	19%
pseudolog	mipmap anisotropic	4D		63	20 %	20%
pseudolog	mipmap anisotropic	2D (flat)		344	09 %	37%
Average				**32**	**14 %**	**28%**

TABLE I
COMPARISON BETWEEN THE MMOPT OPTIMIZATION AND A STANDARD LINEAR SCHEDULING.

Proteus framework. Their method is to tile the output and approximate this non-linear fisheye kernel by a set of linear constraints. Pre-fetching is enabled by doubling the number of input buffers and there is no sharing of input data between successive computations, which increases the bus traffic. [9] reports 4*2 buffers[2] of 6864 pixels, which totals 429 Kbit to compute output tiles of 1536 pixels.

Applying the MMopt tool to the same fisheye kernel results in 14 Kbit of total memory (ITB+IT+CC) to process $32 * 32$ output tiles with $32 * 32$ input tiles. When the input tiles size $8 * 8$, the total memory is 18 Kbit. When scaling everything, the optimization allows to save memory between 20 to 15 times, which corresponds to the results of the Table I.

The large design space due to the target application constraints prevents a much more detailed comparison. The point is that our method saves memory by dynamically sharing buffers between processing and pre-fetching and saves the traffic to the background memory.

V. CONCLUSION

This paper presents an original method to design embedded non-linear kernels and an associated optimization tool. This method focuses on the optimization of the memory hierarchy in a HLS C-to-RTL flow. We have demonstrated the feasibility of enhancing an input high level code by a memory hierarchy to allow a high level synthesis of the generated code. The presented method comes within the field of code transformation as a pre-processing for HLS.

The optimization tool enables several trade-offs at system level between memory area, computing time and external memory traffic by exploiting the disparity of non-linear kernel references and take into account system parameters. When compared to a standard optimization, it allows to save an average 28% of external memory traffic, which comes to save as much power consumption due to external memory transfers. The embedded memory area is also significantly reduced.

[2]scaled to consider monochrome images instead of YUV images.

These results make us confident to continue on this way. Minor improvements are allowable such as increasing the bandwidth to buffers and much is expected by parallelizing the units and design communicating multi-kernel engines. More generally speaking, providing methods aiming at enhancing high level IP models by optimized memory hierarchies is a field of high added value.

REFERENCES

[1] P. Coussy and A. Morawiec, Eds., *From Algorithm to Digital Circuit*. Springer, 2008.

[2] P. R. Panda, F. Catthoor, N. D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle, and P. G. Kjeldsberg, "Data and memory optimization techniques for embedded systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 6, no. 2, pp. 149–206, 2001.

[3] F. Catthoor, K. Danckart, C. Kulkarn, and al, *Data Access and Storage Management for Embedded Programmable Processors*. Kluwer Academic Publishers, 2002.

[4] J. Keinert and J. Teich, *Design of Image Processing Embedded Systems Using Multidimensional Data Flow*. Springer, 2011.

[5] H. Zhu, H. Luican, and F. Balasa, "Mapping multi-dimensional signals into hierarchical memory organizations," in *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, 2007.

[6] F. Balasa, P. Kjeldsberg, A. Vandecappelle, M. Palkovic, Q. Hu, H. Zhu, and F. Catthoor, "Storage estimation and design space exploration methodologies for the memory management of signal processing applications," *Journal of Signal Processing Systems*, vol. 53, 2008.

[7] W. Yu, "An embedded camera lens distortion correction method for mobile computing applications," *Consumer Electronics, IEEE Transactions on*, vol. 49, no. 4, 2003.

[8] J. Jiang, G. Zhang, F. Zhou, D. Yu, H. Xie, and H. Liu, "Distortion correction for a wide-angle lens based on real-time digital image processing," *Optical Engineering*, vol. 42, no. 7, 2003.

[9] N. Bellas, S. Chai, M. Dwyer, and D. Linzmeier, "Real-time fisheye lens distortion correction using automatically generated streaming accelerators," in *Field Programmable Custom Computing Machines,. 17th IEEE Symposium on*, 2009.

[10] S. Zokai and G. Wolberg, "Image registration using log-polar mappings for recovery of large-scale similarity and projective transformations," *Image Processing, IEEE Transactions on*, vol. 14, no. 10, 2005.

[11] J. P. Ewins, M. D. Waller, M. White, and P. F. Lister, "Mipmap level selection for texture mapping," *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, 1998.