

# Memory Organization for Video Algorithms on Programmable Signal Processors<sup>†</sup>

Eddy De Greef, Francky Catthoor<sup>‡</sup>, Hugo De Man<sup>†</sup>

VLSI systems design group

IMEC

Kapeldreef 75

B-3001 Leuven, Belgium

## Abstract

*In this paper, several DSP system design principles are presented which are valid for a large class of memory-intensive algorithms. Our main focus lies on the optimization of the memory and I/O, since these are dominant cost factors in the domain of video and imaging applications. This has resulted in several formalizable mapping principles, which allow to prevent the memory from becoming a bottleneck.*

*First, it is shown that for this class of applications, compile-time data caching decisions not only have a large effect on the performance, but also can have an even larger effect on the overall system cost and power consumption. This is illustrated by means of experiments in which the whole range of no cache up to large cache sizes is scanned.*

*Next, it is shown that when enforcing constant I/O rates to reduce buffer sizes, the area gain may be far more important than the small performance decrease associated with it. A technique to achieve this in an efficient way is proposed.*

*The main test-vehicle which is used throughout the paper to demonstrate our approach is the class of motion estimation type algorithms.*

## 1 Introduction

The goal of this paper is to investigate the memory and I/O related problems encountered when mapping memory intensive signal processing algorithms on programmable DSP's. Also the interaction of these problems with the available parallelism in the DSP is investigated. The end result is a set of mapping principles which are valid for a broad class of video applications.

The main objective is to guide the algorithm/system and architecture designer in better exploring the large space of alternative memory and I/O related decisions. The cost functions which are rele-

vant here are cycle count, memory count or size, and power budget. Each of these is crucial in current system design. The first one has to be within the real-time operation constraint. Efficient storage and communication of signals is the most important issue for system cost in terms of both area and power. To illustrate the power issue, which is only seldom stressed in literature related to storage, it should be taken into account that data sheets of conventional 1Mbit SRAM's show power budgets between 0.5 and 1.5 W when fully utilized. The main influence is due to the transfer count but also size plays an important role because of increased capacitive loading [4]. Even future low-power oriented SRAM's [19] still require about 0.25W at the required clock rates. This is considerably more than what the data-path and controller logic consume for these submicron technologies (on the order of 10-100 mW). Another important source of consumption are the global bus transfers. Power issues are considered as crucial though for realization of many video applications [3].

Our methodology will be illustrated in the mapping of representative memory-intensive video algorithms onto several commercially available programmable fixed-point DSP's. In particular, we will look at the Texas Instruments TMS320C50, the Motorola DSP56116 and the Analog Devices ADSP-2101 processors. The parameterized algorithm used as a demonstrator, namely a template derived from video motion estimation, is described in section 2. Next, we will review the state-of-the-art methods related to our objectives (section 3). Several options for the memory organization are being incorporated. In particular, we will devote much attention to both the caching mechanism (section 4) and to the fine-grain synchronization issues (section 5). The results of the experiments and the formalized design principles are presented and evaluated throughout these two sections. Finally, some conclusions will be drawn.

<sup>†</sup>This research was sponsored by Siemens A.G., München, Germany

<sup>‡</sup>Professor at the Katholieke Universiteit, Leuven, Belgium

## 2 Envisioned class of algorithms

For our experiments, we used a version of the motion estimation (ME) algorithm [7, 11, 12] which is commonly referred to as the "full-search full-pixel" implementation [12]. However, our results remain valid for a large class of similar algorithms. The ME algorithm is a.o. used in moving image compression algorithms (e. g. MPEG [14]). It divides image frames into small blocks, called current blocks (CB's) and tries to estimate the motion vector for each of these blocks by matching them with blocks in the previous frame in a region around the same location. This region is called the reference window (RW). This is illustrated in figure 1, together with some of the parameters of the algorithm. In our experiments, we assumed that the images are gray-scaled (in practice, for color images only the luminance is considered). The algorithm is typically executed in 6 nested loops, as shown next:

```
for (g=0;g<H/n;g++) { [vert. CB counter]
  for (h=0;h<W/n;h++) { [hor. CB counter]
    for (i=-m;i<m;i++) { [hor. search in RW]
      for (j=-m;j<m;j++) { [vert. search in RW]
        for (k=0;k<n;k++) { [hor. CB traversal]
          for (l=0;l<n;l++) { [vert. CB traversal]
            Operation 1 (O1) on CB/RW pixel
          }}
          Operation 2 (O2) on result of O1
        }}}}
```

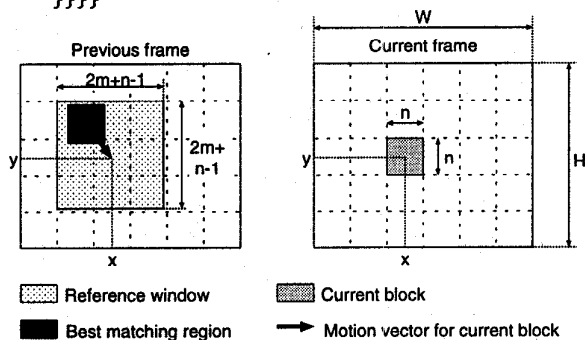


Figure 1: The motion estimation algorithm and its parameters

The parameters we used in our experiments are the following (standard television):  $W=720$ ,  $H=576$ ,  $m=8$ ,  $n=8$  pixels and a frame rate of 25 Hz. For real-time execution, these parameters would lead to a required processing rate of 5.3 G pixels/second. The operations to be performed are relatively simple (e.g. O1 consist of a subtraction, an absolute value and an accumulation). It is clear that the main problem will be the transfer of the huge amount of data.

## 3 State of the art

Up till now, most of the research on mapping algorithms onto programmable parallel architectures has

been concentrated on the optimization of the throughput, by means of task scheduling and/or allocation techniques. In [10, 16, 8], some examples of algorithms mapped according to these principles onto (multiple) programmable processors are presented, but these design exercises have not been formalized towards broader domains. A few papers discuss more formal methods [5, 13, 18], but in video applications the memory and interconnection cost are usually dominant and these are not sufficiently taken into account. Only little research has been spent on memory optimization related issues [9], and this is targeted for numerical applications where the crucial real-time aspect is not considered.

The fact that memory bandwidth is becoming a problem due to the permanent increase in processing power of today's processors is well known. Some research has been done on generic techniques for reducing the required memory bandwidth (e.g. [24]), but this is usually based on some local optimizations, which have only a limited overall effect. In this paper, the interaction between memory management and performance is investigated thoroughly for memory intensive applications. One of our main aims is to reduce the required off-chip bandwidth by means of an efficient on-chip/off-chip buffering strategy.

Some papers present design considerations for shared-memory multiprocessors ([2] and [15]), but these are not targeted towards real-time video algorithms with intensive data flow.

Several designs for motion estimation type video algorithms have been presented, which make use of systolic or linear arrays or tree-like architectures [7, 11, 12, 17, 1]. In general, these architectures are customized to specific applications and lack flexibility and programmability. The design methods to derive the memory related issues are not formalized.

Partly based on the results in this paper, also a parallel architectural template using existing programmable video processors in a custom but parameterizable configuration has been defined by us. This architecture is able to efficiently execute the motion estimation type of algorithms in real-time, as shown in a previous paper [6].

## 4 Data caching strategies

It is a well known fact that caching techniques can greatly improve the performance of processors in general. In the past a lot of effort has been spent on heuristic techniques [20]. These techniques usually concentrate on determining optimal cache sizes, or improving hit-rates given a certain cache structure [21] and have been proven to be very useful for general-

purpose processing.

In real-time signal processing applications however, where fixed execution times are crucial, these generic techniques are much less suited since one cannot predict the exact execution time of an algorithm being implemented on a system with a heuristic cache protocol. Even the worst case execution time can be very hard to predict.

Moreover, the kind of applications which are mapped on digital signal processors (DSP's) usually exhibit a relatively simple control flow. This is especially true in the case of video signal processing applications. In those cases, the designer can usually derive an optimal caching strategy in advance. Instead of using a classical least-recently-used cache replacement protocol, one can use a more optimal protocol, such as a least-soon-to-be-used protocol, without run-time overhead.

A fact that is less well studied (certainly in research literature) is that in signal processing applications, an efficient use of this cache not only improves the performance (cycle count), but can also have a large effect on the overall system cost and power consumption. By using the cache in an optimal way, one can usually reduce the bandwidth requirements of the external memories considerably. As a consequence, not only the size of the memories will decrease, but also the power consumption due to memory accesses and external transfers will decrease.

It is clear that for video applications, caching is a must. For instance, for the typical parameters presented in section 2, a frame memory bandwidth of 5.3 GByte/second would be required when no cache would be used. This bandwidth could only be achieved at the expense of an extremely high cost and power consumption.

In order to illustrate the validity of these principles, we performed some experiments with the cache size as the main parameter. The ME algorithm was mapped onto the 3 more or less equivalent fixed-point processors mentioned in section 1. Each of them have a certain amount of on-chip data memory, which we used as a data cache. The decisions about which data should be stored in the on-chip memory at which time, were taken at compile time. It was not our intention to obtain real-time operation on any of these processors, but merely to study the trade-offs between (sometimes hypothetical) cache size, cycle count, and external transfer count (power). It is our belief that these trade-offs are relatively independent of the chosen DSP architecture. The summarized results of our experiments can be found in figure 2.

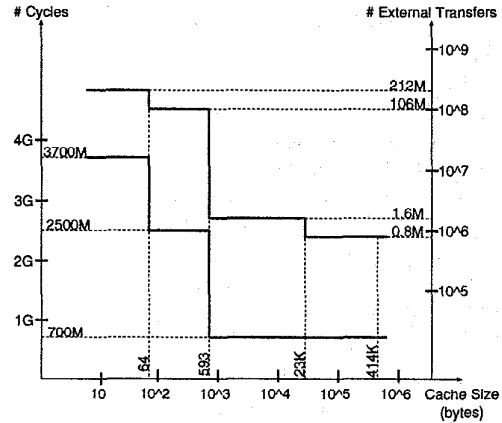


Figure 2: Summarized results of data caching strategies on cycle count and external transfers

From these experiments, it is clear that the size of the available cache can have an enormous effect on the cycle count, but especially on the number of external transfers. Even for a reasonably small cache size of 23 KBytes, the external transfer count was reduced by more than 2 orders of magnitude, compared to the case without a cache. Since for this class of applications the power consumption is mainly determined by the external transfers, the power consumption will also be reduced considerably.

## 5 Buffer size reduction through synchronization

In many signal processing applications, and especially in video applications, the area cost due to memories is dominant. In the future, this may become even more important due to the integration of complete systems on a single chip. Those systems usually consist out of several processing cores, each performing a specific task. The data-flow between those processing cores can give rise to large buffers if the cores are not completely synchronized. For instance, in video applications, it is common practice to use several buffers containing several frames.

In the sequel we will show how the sizes of those buffers can be reduced. First, we will look at the classical way in which systems are built, and next we will present an alternative approach, where we concentrate on the memory related issues. We will also show that this approach is compatible with the caching techniques presented in the previous sections. This will be illustrated by means of some experiments.

### 5.1 Frame-synchronous versus fine-grain-synchronous operation

The classical way to implement video-algorithms in real time requires the use of two kinds of buffers: I/O

buffers and active buffers (containing the data being processed). For an algorithm that requires the pixels of  $F$  successive image frames, at least  $F + 1$  frame buffers are required: 1 I/O buffer for storing incoming pixels and  $F$  active buffers for executing the algorithm. Synchronization then only occurs at the beginning of each new frame. The advantage of this approach is that it is relatively simple to implement.

However, even for algorithms such as ME, which operate on only a relatively small number of consecutive frames ( $F=2$ ), it is clear that the system cost will be dominated by the size of the frame buffers. For the parameters presented in section 2, a frame buffer size of about 1.2 MByte would be required. This is certainly much more expensive than the datapath and controller.

It is a well known principle that synchronization can reduce buffer sizes. Less well known is the fact that it can not only reduce the sizes of the I/O buffers, but also the size of the active buffers. In general, for a block-oriented algorithm operating on  $F$  successive frames, such as ME, it is possible to reduce the total buffer size to little more than  $F-1$  frames by using fine-grain synchronization. In figure 3, it is shown how this can be done for the ME algorithm.

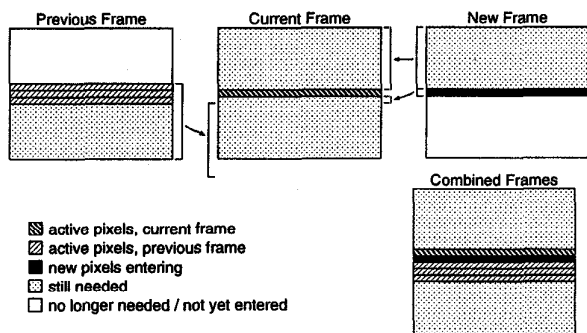


Figure 3: Merging of frame buffers

The I/O buffer (new frame) has been merged with the active buffers (current and previous frame) such that the total buffer size has been reduced to little more than one frame. Instead of gathering the pixels of the new frame and delaying them by 1 frame cycle, they are now being processed almost immediately after they enter the system.

This result has been obtained by carefully synchronizing the I/O data streams and the data processing by means of the technique described in section 5.2. However, an important factor is the execution order of the algorithm. In general, an algorithm can be executed in different orders, but these can lead to dif-

ferent storage requirements. For instance, for a block-oriented algorithm, it is obvious that the execution order of the blocks should be row-wise in case the input data enter row-wise. This way, the life-time of the data can be kept as small as possible.

In general, the selection of the optimal execution order can be very difficult. Several techniques described in literature, such as loop transformations [21, 22, 23], can significantly reduce the "life-time" of data items, leading to lower memory sizes.

## 5.2 Synchronizing I/O and processing

In this section, we'll present a technique which can be used for synchronizing I/O data streams with the data processing in order to reduce the buffer sizes. The main goal is to make the time interval between successive I/O operations constant. In general, two common programming constructs can be responsible for disturbing the regularity in the data processing speed: conditional branches and nested loops. It is well known that in the case of conditional branches the lengths of all branches can be made equal by padding them with dummy code (e.g. NOP's).

For nested loops however, the problem is less well known. In general, every loop has some kind of entry and exit code, i.e. the code between the loop borders. Even in case of a so-called zero-overhead loop, there is a non-negligible overhead associated with the initialization of the loop. These pieces of code are responsible for disturbing the regularity of the processing speed. If the loop nest contains an I/O operation, consequently the I/O rate will be irregular too, giving rise to buffers. In this case also, the problem can be solved by inserting dummy code, as illustrated in figure 4 for the case of two nested loops.

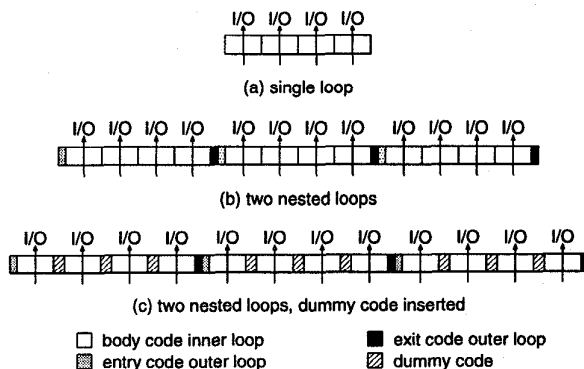


Figure 4: Fine-grain synchronization problems with nested loops

The resulting pseudo-assembly code would then typically look like this:

```

i=0
lp_i      j=0          [entry code i]
          goto lp_j_entry [entry code i,
                          skip dummy code]

lp_j      [ dummy code ]
lp_j_entry [I/O] operation
          j++
lp_j_end  if (j<n) goto lp_j
          i++          [exit code i]
lp_i_end  if (i<m) goto lp_i [exit code i]

```

In general, in case of  $N$  nested loops, the length of the dummy code to be inserted in loop  $l$ , for  $l$  ranging from 1 (inner loop) to  $N-1$ , can be expressed by the following formula:

$$[dummy\ code]_l = \sum_{k=l+1}^N [exit\ code]_k + [entry\ code]_k$$

This length does not necessarily relate to the physical size of the dummy code (the number of code words), but rather to the number of cycles spent on this dummy code. A long number of cycles can usually be obtained with only a few code words, typically by means of a repeat instruction.

### 5.3 Experiments

Again, we mapped the ME algorithm on the processors described in section 4, but this time using the synchronization technique presented above. Our first experiments yielded a striking result: more than 60 % of the cycle budget was consumed by dummy code. The reason is that the synchronization technique tries to make the execution time needed per pixel constant, i.e. equal to the worst-case execution time. The problem was that the worst-case execution time was much larger than the average execution time. Under normal circumstances, large irregularities in the outer loops have only little effect on the total execution time, but in case the processing speed has to be fixed, these irregularities can play an important role.

Therefore, we tried to avoid those irregularities as much as possible. In our case, the irregularities were caused by border effects. Each time the right border of the image was reached, there was a relatively large period during which no new pixels were needed. But when going to the next row of blocks to be processed, suddenly a relatively large amount of new pixels was needed. This irregularity could be avoided by already starting to read the pixels required for the first block of the next row of blocks during the processing of the last block of the previous row. This required a slightly larger on chip buffer size (data cache), though. As a result, the overhead due to dummy code had almost vanished (less than 6%).

The total cycle count was then still 40 to 60 % larger than for the frame synchronous case, but this

was mostly caused by the addressing overhead both for internal and external transfers. On modern video processors, such as the TMS320C80, which have very powerful address generation capabilities, the overhead would be much smaller, such that the time penalty would be reasonably small. In contrast, the gain in memory size can be very large, as indicated above.

The summarized results of our experiments can be found in figure 5.

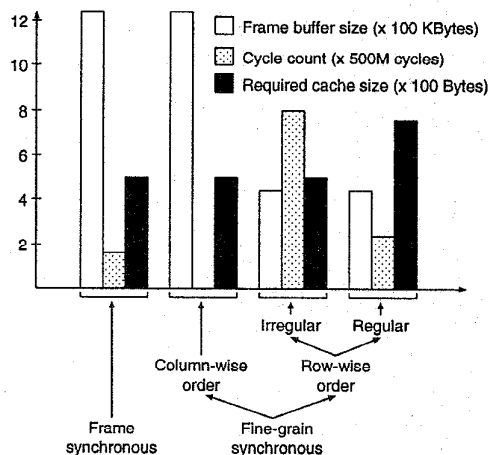


Figure 5: Summarized results of buffer size reduction through synchronization

## 6 Conclusions

- In video signal processing applications, caching of data is a must. By making use of knowledge about the execution order of the algorithm, an optimal cache replacement protocol can be deduced at compile time, without run-time overhead. This form of data-caching not only reduces the cycle count, but it is also very beneficial for the cost of the memories and the overall power consumption.
- Fine-grain synchronization in order to reduce or even eliminate both I/O and active buffers is feasible, both for conditional branches and nested loops. The overhead introduced by this technique can be kept relatively small, while the savings in buffer size can be very large. To keep the overhead low, irregularities in the algorithm should be kept as small as possible. This may require a slightly larger on-chip data cache.

## References

- [1] T.Akiyama, H.Aono, K.Aoki, K.Ler, B.Wilson, T.Araki, T.Morishige, H.Takeno, A.Sato, S.Nakatani, T.Senoh, "MPEG2 video codec using image compression DSP", *IEEE Trans. on Consumer Electronics*, Vol.CE-40, No.3, pp.466-472, August 1994.

- [2] B. K. Bose, P. M. Hansen, C. Lee, D. A. Patterson, "Fast Scientific Computation in CMOS VLSI Shared-Memory MultiProcessors", Proceedings of ISCAS '88, pp. 811-814, May 1988.
- [3] R. Brodersen, A. Chandrakasan, S. Sheng, "Low-power signal processing systems", *Proc. IEEE workshop on VLSI signal processing*, Napa Valley CA, Oct. 1992. Also in *VLSI Signal Processing V*, K. Yao, R. Jain, W. Przytula (eds.), IEEE Press, New York, pp.3-13, 1992.
- [4] F. Catthoor, F. Franssen, S. Wuytack, L. Nachtergaele, H. De Man, "Global communication and memory optimizing transformations for low power signal processing systems", *IEEE workshop on VLSI signal processing*, La Jolla CA, Oct. 1994. Also in *VLSI Signal Processing VII*, J. Rabaey, P. Chau, J. Eldon (eds.), IEEE Press, New York, pp.178-187, 1994.
- [5] Y.-Y. Chen, Y.-C. Hsu, C.-T. King, "MULTIPAR: behavioral partition for synthesizing multiprocessor architectures", *IEEE Trans. on VLSI Systems*, Vol.2, No.1, pp.21-32, March 1994.
- [6] E. De Greef, F. Catthoor, H. De Man, "A memory-efficient, programmable multi-processor architecture for real-time motion estimation type algorithms", *Intl. Workshop on Algorithms and Parallel VLSI Architectures*, Leuven, Belgium, August 1994.
- [7] L. De Vos, M. Stegherr, "Parameterizable VLSI Architectures for the full-search block-matching algorithm", *IEEE Transactions on Circuits and Systems*, vol. 36, pp. 1309-1316, Oct. 1989.
- [8] M. Engels, R. Lauwereins, J. Peperstraete, A. Van Roermond, "Design of a processor board for a programmable multi-VSP system", *Journal of VLSI signal processing*, special issue on "Video/image signal processing", T. Nishitani, P. Ang, F. Catthoor (eds.), Vol.5, No.2-3, Kluwer, Boston, pp.171-184, April 1993.
- [9] J. Z. Fang, M. Lu, "An iteration partition approach for cache or local memory thrashing on parallel processing", *IEEE Trans. on Computers*, Vol.C-42, No.5, pp.529-546, May 1993.
- [10] T. Fujii, N. Ohta, "A Load Balancing Technique for Video Signal Processing on a Multicomputer Type DSP", Proceedings of ICASSP '88, pp. 1981-1984.
- [11] Y. Jehng, L. Chen, T. Chiueh, "An efficient and simple VLSI tree architecture for motion estimation algorithms", *IEEE Transactions on Signal Processing*, vol. 41, pp. 889-900, Feb. 1993.
- [12] T. Komarek, P. Pirsch, "Array Architectures for Block Matching Algorithms", *IEEE Transactions on Circuits and Systems*, vol 36, Oct. 1989.
- [13] K. Konstantinides, R. Kaneshiro, J. Tani, "Task allocation and scheduling models for multiprocessor digital signal processing", *IEEE Trans. on Acoustics, Speech and Signal Processing*, Vol. ASSP-38, No.12, pp.2151-2161, Dec. 1990.
- [14] D. Le Gall, "MPEG: A video Compression Standard for Multimedia Applications", *Communications of the ACM*, Vol. 34, No. 4, pp. 46-58, April 1991.
- [15] D. Lilja, "The impact of parallel loop scheduling strategies on prefetching in a shared memory multi-processor", *IEEE Trans. on Parallel and Distributed Systems*, Vol.5, No.6, pp.573-584, June 1994.
- [16] B. I. Pawate, M. L. McMahan, R. H. Wiggings, G. R. Doddington, P. K. Rajasekaran, "Connected Word Recognizer on a Multiprocessor System", Proceedings of ICASSP '87, pp. 1151-1154, 1987.
- [17] J. Rosseel, F. Catthoor, H. De Man, "The systematic design of a motion estimation array architecture", Proceedings of the IEEE International Conference on Application Specific Array Processors, pp. 40-54, IEEE Computer Society Press, 1991.
- [18] M. Schwiengershausen, M. Schönfeld and P. Pirsch, "Mapping complex image processing algorithms onto heterogeneous multi-processors regarding architecture dependent performance parameters", *Intl. Workshop on Algorithms and Parallel VLSI Architectures*, Leuven, Belgium, August 1994.
- [19] T. Seki, E. Itoh, C. Furukawa, I. Maeno, T. Ozawa, H. Sano, N. Suzuki, "A 6-ns 1-Mb CMOS SRAM with Latched Sense Amplifier", *IEEE Journal of Solid-State Circuits*, Vol.28, No.4, pp.478-483, Apr. 1993.
- [20] R. Hundal, V. G. Oklobdzija, "Determination of Optimal Sizes for a First and Second Level SRAM-DRAM On-Chip Cache Combination", Proceedings of ICCD '94, pp. 60-64.
- [21] D. Lilja, "The impact of parallel loop scheduling strategies on prefetching in a shared memory multi-processor", *IEEE Trans. on Parallel and Distributed Systems*, Vol.5, No.6, pp.573-584, June 1994.
- [22] R. Allen, K. Kennedy, "Vector register allocation," *IEEE Trans. on Computers*, Vol.41, No.10, pp.1290-1316, Oct. 1992.
- [23] M. van Swaaij, F. Franssen, F. Catthoor, H. De Man, "Modeling Data Flow and Control Flow for High Level Memory Management", Proceedings of EDAC '92, pp. 8-13.
- [24] D. Kolson, A. Nicolau, N. Dutt, "Minimization of memory traffic in high-level synthesis", Proc. 31st ACM/IEEE Design Automation Conf., San Diego, CA, pp.149-154, June 1994.