Low power storage for hierarchical graphs*

Erik Brockmeyer, Sven Wuytack, Arnout Vandecappelle, Francky Catthoor†

*Partly sponsored by the Esprit ESD-LPD project 25518 : DAB-LP

†Also professor at the Katholieke Univ. Leuven, Belgium

Abstract

In data dominated applications, like multi-media and telecom, data storage and transfers are the most important factors in terms of power, area and system performance. Several steps which optimize these costs are present in our systematic Data Transfer and Storage Exploration (DTSE) methodology. Here, we focus on the Storage Cycle Budget Distribution (SCBD), a step which is crucial for meeting the real-time constraints in typical multi-media and telecom applications.

This paper proves the effectiveness of the developed prototype tool on driver applications of several application domains. The derived cost information can be effectively used to trade off the cycles assigned to the different concurrent operating system tasks or between memory subsystem and to the processor data-path.

1 Introduction

In data transfer and storage intensive applications, the memory accesses are often the limiting factor to the execution speed, both in custom "hardware" and instruction-set processors ("software"). Data processing can easily be sped up through pipelining and other forms of parallelism. Memory bandwidth, on the other hand, is a lot more expensive. Multi-port memories cause a large cost in area and power. Because memory accesses are so important, it is even possible to make an initial system level performance evaluation based solely on the memory accesses to complex data types [3]. Data processing is then temporarily ignored except for the fact that it introduces dependencies between memory accesses.

Our Data Transfer & Storage Exploration (DTSE) [3] methodology allows to systematically optimize data-dominated applications for power and system bus load reduction. Here the "back-end" of DTSE is considered, which has the highest priority for tool support. It includes the Physical Memory Management (PMM) step. In contrast, the "front-end" steps in the script are mainly transformations which can still be done manually (but in a systematic way [6]). Physical memory management involves the design of a (partly) custom memory organization for an application. The memory organization must be as cost-efficient as possible, but on the other hand the performance must (just) meet the real-time constraints.

We believe that the first step in physical memory management should involve making the trade off between the cost of providing a certain amount of memory bandwidth, and the real-time constraints. The approach taken in our Storage Cycle Budget Distribution (SCBD) method is to partly order the memory accesses such that the maximally required memory bandwidth is minimized [7]. This can be done by a (more) efficient use of the memory ports. The result of this SCBD step is a set of constraints for the memory hierarchy. In our application domain, an overall target storage cycle budget is typically imposed, corresponding to the overall throughput. In addition other real-time constraints can be present which restrict the ordering freedom.

To carefully evaluate the effect of the SCBD step, a detailed internal organization in each memory has to be decided. This objective is tackled in the memory allocation and assignment step, for which a systematic technique has been published in [6, 2].

2 Related work

In the register allocation domain, the allocation techniques start from a fully scheduled flow graph and are scalar oriented [12]. In the less explored domain of memory allocation and assignment for hardware systems, the current techniques start from a given schedule [9], or perform first a bandwidth estimation step [8] which is a kind of crude ordering that does not really optimize the conflict graph either. These techniques have to operate on groups of signals instead of on scalars to keep the complexity acceptable, e.g., the *stream model* of Phideo [9] or the *basic sets* in the ATOMIUM environment [8].

In the scheduling domain, the techniques for optimizing the number of resources given the cycle budget are of interest to us. Also here most techniques operate on the scalar-level, e.g. [10, 13]. The minor exceptions currently are the Phideo stream scheduler [14] and the Notre-Dame rotation scheduler [11]. Many of these scalar techniques try to reduce the memory related cost by estimating the required number of registers for a given schedule. Only few of them try to reduce the required memory bandwidth, which they do by minimizing the *number* of simultaneous data accesses [13, 14]. They do not take into account *which* data is being accessed simultaneously. Also no real effort is spent to optimize the data access conflict graphs such that subsequent register/memory allocation tasks can do a better job.

The main difference between our SCBD and the related work discussed here is that we try to minimize the required memory bandwidth in advance by optimizing the access conflict graph for groups of scalars within a given cycle budget. We do this by putting ordering constraints on the flow graph, taking into account *which* data accesses are being put in parallel (i.e., will show up as a conflict in the access conflict graph).

3 Brief summary storage cycle budget distribution technique

The main goal of the approach is to find which arrays must be stored in different memories in order to meet the cycle budget. Ordering the memory accesses within a certain number of memory cycles determines the required memory bandwidth. Moreover, the data dependencies must be obeyed.

If two memory accesses ordered in the same memory cycle are in conflict, parallelism is required to perform both accesses. This can take the form of a multi-port memory, or, if two different arrays are being accessed, multiple memories. Thus, the conflict incurs a certain cost. A tradeoff between the performance gain and the conflict related cost has to be made. On the one hand, every array in a separate memory is optimal for speed and seemingly also for power. On the other hand, having many memories is very costly for area, interconnect and complexity and due to the routing overhead also for power in the end. Due to the presence of the real time constraints and the complex control and data dependencies a difficult tradeoff has to be decided. Therefore automatic tool support is crucial.

The previous published technique [7, 2] can only be directly used for flat Control Data Flow Graphs (CDFG), i.e. when no loops and no function calls are present. Loops could be unrolled and function calls could be inlined, but this destroys the hardware or code reuse possibilities present in the original specification. Moreover, the ordering problem would explode. Therefore, ordering the memory accesses has to be done in a hierarchical way. This is now solved in a new tool.

Without going into more detail we will now present different practical tradeoff considerations and how to interpret the result. Moreover, in the following section two industrial driver applications will be presented.

4 Tradeoff considerations

The tool combination explores different solutions in the performance, power and area space. Indeed, every step of the SCBD generates a valid solution for a different cycle budget. Therefore it becomes possible to make the right tradeoff within this solution space. Note that without the tool support described above, the type of tradeoffs discussed here were never achievable on real life applications. We believe that this is also the first time such graphs have been published.

In case of a single thread and the memory subsystem being power dominant, the tradeoff can be based solely on the tool output. The given cycle budget defines a conflict graph which can be used for the memory allocation and assignment tool [2, 6]. Obviously, the power and area cost will increase when the cycle budget is lowered. More bandwidth will be needed which will cause multi-port memories (increase power) or more memories are needed (increase area).

The performance-power function can be used to tradeoff cycles assigned to a task on the system level. Such a function can indeed be generated per task. Assigning too few cycles to a single task causes the entire application to perform poorly. The cycle and power estimates clearly helps the designer to assign tasks to processors and to distribute the cycles within the processors over the different tasks (see Figure 1). Minimizing the overal power within a processor is possible by applying function minimization on all the power-cycle functions together.

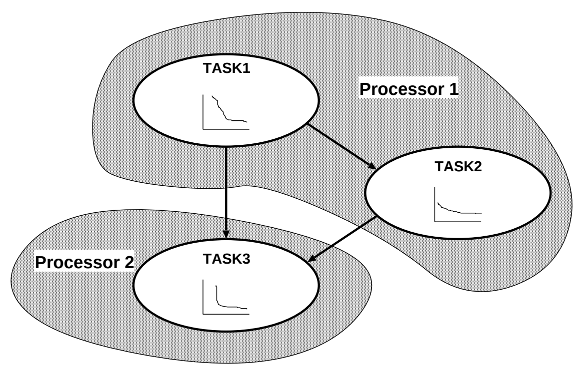

Figure 1. Tradeoff cycles assigned tasks.

Another tradeoff can be made when the data-path contribution is incorporated. The assignment of cycles to memory accesses and to the data-path is important for the overall power consumption. A certain percentage of the overall time can be spent to memory accesses and the remaining part to computational issues taking into account the system pipelining (see Figure 2). The data-path cost versus performance is well known. When less cycles are available, more hardware is needed and more power is consumed (see dotted curve). Our tools provide the other half of the graph, the memory related cost (solid curve). Combining the two leads to a real optimized implementation (dashed curve).

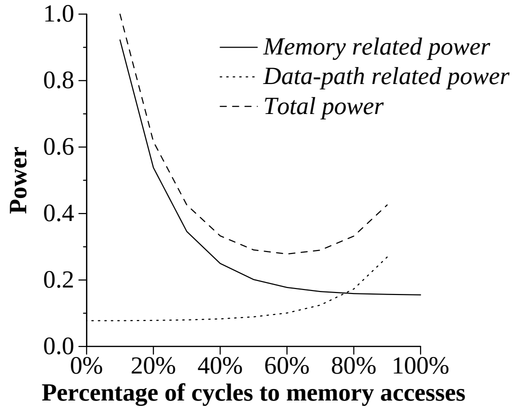

Figure 2. Tradeoff cycles assigned to memory accesses and data-path

5 Experimental SCBD results

The new prototype tool has been applied to drivers from multiple application domains to prove the effectivness. The power numbers in this section are based on real memories. A Motorola library memory model is used to estimate the on-chip memories (see [3]). For the off-chip components, we have used the EDO DRAM series of Siemens.

5.1 Synchro core of DAB

In the near future mobile radios with Digital Audio Broadcast (DAB) reception will be produced. A DAB broadcaster is able to provide either 6 high quality radio programs with associated data, or for instance a MPEG 1 video signal. DAB provides any combination of services up to a total of about 1.8Mb/s. DAB uses Orthogonal Frequency Division Multiplex (OFDM) modulation which is resistant against multi path interference and frequency-selective fading.

Here, we will focus on the synchronization processor only [1]. The synchronization is performed in two steps. In the first step a correlation to a know reference signal determines the frequency error. Next, an IFFT is performed obtaining the channel impulse response.

Figure 3 shows the power curve representing the usefull tradeoff betweem cycle budget and power cost, from the fully sequential case down to the case close to the critical path. An equal graph can be made for the area cost. Note the discontinuities in the cost function when a dual ported memory is added.

The rightmost point of the graph is the fully sequential case. All arrays can be assigned to a single memory because no conflicts are available. However, for power and area considerations four single-port memories are allocated. In case of multiple memories, the signals are stored in smaller memories and less bitwaste due to bitwidth differences in the signals. The cycle budget can be decreased downto 85 kCycles without increasing the power nor the area. Note that the SCBD step generates more points in the graph but the MAA step could sustain an equally (good) memory hierarchy so the extra points are not usefull in the pareto curve. In order to reduce the budget below 85 kCycles, dual-ported memories are needed. Allocation of 4 dual-ported memories allows to decrease the cycle budget close to the critical path (55047 cycles).

The "front-end" DTSE optimization stages have been manually applied on the DAB application. As a result, the optimized implementation needs less memories for the same cycle budget. The optimized implementation also consumes much less power. Moreover, it can sustain this low power by a much lower cycle budget. It can nearly reach the critical path (45378 cycles) without sacrificing memory power. To obtain the SCBD results, the tool execution time is in the order of several minutes. So it can easily be used to give feedback to the designer.

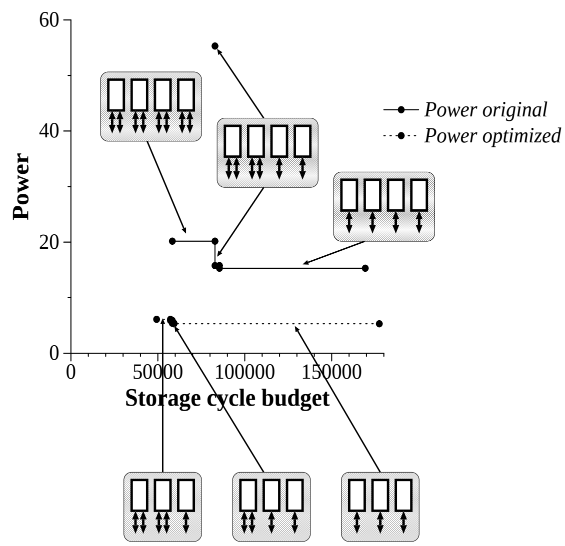

Figure 3. Power cost and memory architecture given the cycle budget for original and optimized description.

A tradeoff is involved in how much time is assigned to the synchronization process and to the rest of the application. The real time constraint in the DAB synchro core is given by the number of times the synchronization process is started. The synchronization is started in every time frame (=96 ms). The algorithm is mapped on a separate processor which is running on 40 MHz. Therefore timing constraints are not very narrow. However, when the synchronisation task and the FFT task are mapped to the same processor, timing becomes crucial. Then the available time for the synchronisation process is limited to one symbol (=1.2 ms $\approx$ 52 kCycle). Using the feedback of the tool makes the tradeoff possible. The hardware cost saving (a second processor) should be bigger than the additional memory cost needed for a higher bandwidth. Here, the memory cost is not affected much in the optimized description when the constraint is put at 52Kcycle.

This exploration clearly shows the different solutions and design tradeoffs and is very useful for the system designer to allow a globally motivated distribution of the timing constraints over the system modules. This indirectly also has a significant impact on overall power.

5.2 Binary tree predictive coder

Another illustration of the SCBD and MAA tools focusses on an industrial image processing application. Binary Tree Predictive Coding (BTPC) [5] is a lossless or lossy image compression algorithm based on multi-resolution. The image is successively split into a high-resolution image and a low-resolution quarter image, where the low-resolution image is split up further. The pixels in the high-resolution image are predicted based on patterns in the neighbouring pixels. The remaining error is then expected to achieve high compression ratios with an adaptive Huffman coder. Six different Huffman coders are used, depending on the neighbourhood pattern. For lossy compression, the predictors are quantized before the Huffman coding. Figure 4 shows the structure of the considered algorithm. Only the most important parts, and the loop structure around them, are shown. The loop structure is manifest, but inside the loop bodies, many data-dependent conditionals are present. This shows that our approach can handle such heavily data-dependent code.

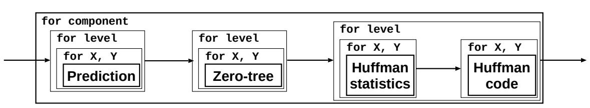

Figure 4. Most important blocks and loop structure of the BTPC application

For a hardware implementation, some of the parameters of the original (software) specification have to be fixed. For example, the maximum input image size is set at 1024×1024 pixels. Fixing these parameters is necessary to be able to determine the sizes or the bitwidths of some of the arrays. For example, the size of the input image determines the size of three arrays containing image data (in total 18 Mbit), and it determines the bitwidth of the arrays gathering the Huffman statistics (in total 80 kbit, distributed over 16 signals).

The front-end steps of our overall DTSE methodology [3] are applied manually on this example and the back-end steps (using tools) give accurate feedback about performance and cost. Figure 5 shows the relation between speed and power for the original, optimized and intermediate transformed specifications. The off-chip signals are stored in separate memory components. Four memories are allocated for the on-chip signals. Every step leads to a significant performance improvement without increasing the system cost. For every description, the cycle budget can be traded for system cost.

The original implementation needs 28 million cycles when executed fully sequentially. In principle, all the signals could be stored in one single memory having one port. However, more memories will decrease the power consumption. The original description can be sped up to 18 million cycles by customizing the memory architecture. The power overhead of this change is negligible due to an optimized ordering.

Further cycle reduction forces a dual-ported off chip memory, introducing large costs (power, area, pin count). We cannot accurately estimate this part of the impact since no appropriate model is available from vendors. A reasonable assumption is a power overhead of 50% for a dual-ported memory. Using this assumption we can estimate the power increase for lower budgets. Adding this dual-ported off-chip memory dramatically increases the global power (see dotted part in Figure 5). We didn't redo this (useless) exercise for the other applied transformations.

By global loop merging, the access count is reduced in the zero-tree generation. This has a positive effect on both cycle budget and power consumption. Moreover, this is an enabling transformation for the SCBD step. The more memory accesses in a basic block, the more ordering freedom available.

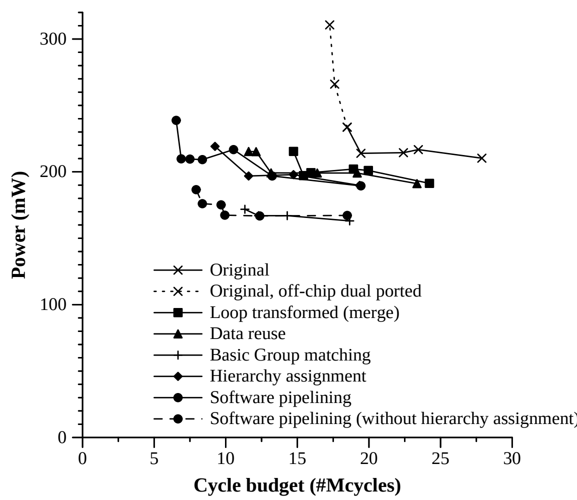

Figure 5. Power versus performance for BTPC example.

The data reuse step [4] can be applied on two levels: firstly register level data reuse, secondly line buffer based. Here, the register level reuse decreases the number of accesses little. Not much data is reused on the average actually. The main benefit is the reduction of the critical path in the worst case conditional branch (which is not executed very often) but has a large memory impact. The large predicting condition trees cause a large cycle budget, and the worst case path had to be taken into account. By introducing the register reuse, all data dependent background memory accesses can be removed, leading to a more manifest behaviour. The line buffer data reuse is not introduced in this step because it has a too negative impact on power.

Basic group matching [6] has a big positive impact on the off-chip power. The signals pyr and $ridge$ are nearly always accessed together (same time, same address). These two main signals are merged together, decreasing the off-chip memory count from three to two. Also some on-chip signals are merged. Though this has a large possible power impact, the minimal cycle budget is nearly unchanged.

In the hierarchy assignment the data reuse copies are grouped into a final memory hierarchy. The on-chip memories in this hierarchy can have multiple ports at a relatively low cost. For these performance reasons, we have added the line-based data-reuse level. Here, we were able to introduce multiple line based signals for which we could prevent dual-ported memories. This transformation is not applied when the main emphasis is power. The power and performance exploration path do separated here.

The power oriented path (without the memory hierarchy assignment) is given with dashed line in the graph. It does not reach the same cycle budget but the power remains low (the last 18% performance gain costs 28% in power). Of course it is a matter of constraints and tradeoffs which implementation is chosen finally.

In a last step, the code has been software pipelined. Data dependencies initially exclude optimal off-chip memory bandwidth usage. A certain timeslot can be blocked to be used efficiently. For instance when a loop body needs data from an on-chip memory before writing to an off-chip memory then the first cycle of the loop body cannot be used for the off-chip access. By moving the memory accesses to a next or previous loop iteration, the dependency can be broken.

In total, a minimal cycle budget of 6.5 million cycles is achieved. The manual transformations have lead to a factor 4 performance improvement using the same power budget. This is the fastest achievable when having a single-ported off-chip memory (every memory cycle contains an off-chip access to the in signal).

6 Conclusions

Memory bandwidth is the limiting factor for performance of current multi-media systems. More bandwidth becomes available by adding multiport or more memories. These extra memories involve extra costs. The Storage Cycle Budget Distribution tool optimizes the memory architecture within the real-time constraint. The power, area and timing information obtained from the tool is very usefull especially at the system level. Clear tradeoffs can be made on the operating task level using these estimations. Moreover, on the lower level, the global time budget can be divided over the memory subsystem and the processor data-path.

Visit our web-site and download a demo version of the tool:

`http://www.imec.be/acropolis/`

References

[1] E.Brockmeyer, J.D'Eer, N.Busa', F.Catthoor, P.Lippens, J.Huiskens, "Code transformations for reduced data transfer and storage in low power realization of DAB synchro core", Patmos'99, Kos, Greece, Oct 6-8, 1999.

[2] F.Catthoor, S.Wuytack, E.De Greef, F.Franssen, L.Nachtergaele. H.De Man, "System-level transformations for low power data transfer and storage", in paper collection on "Low power CMOS design" (eds. A.Chandrakasan, R.Brodersen), IEEE Press, pp.609-618, 1998.

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

[4] J.P.Diguet, S.Wuytack, F.Catthoor, H.De Man, "Formalized methodology for data reuse exploration in hierarchical memory mappings", *Proc. IEEE Intnl. Symp. on Low Power Design*, Monterey, pp.30-35, Aug. 1997.

[5] J. Robinson. Efficient general-purpose image compression with binary tree predictive coding. *IEEE Transactions on Image Processing*, 6(4):601–608, Apr. 1997.

[6] A.Vandecappelle, M.Miranda, E.Brockmeyer F.Catthoor, D.Verkest, "Global Multimedia System Design Exploration using Accurate Memory Organization Feedback" accepted for *Proc. 36th ACM/IEEE Design Automation Conf.*, New Orleans LA, June 1999.

[7] S.Wuytack, F.Catthoor, G.De Jong, H.De Man, "Minimizing the Required Memory Bandwidth in VLSI System Realizations", accepted for *IEEE Trans. on VLSI Systems*, Vol.7, No., pp., 1999.

[8] F.Balasa, F.Catthoor, H.De Man, "Dataflow-driven memory allocation for multi-dimensional processing systems", *Proceedings IEEE International Conference on Computer Aided Design*, San Jose CA, Nov. 1994.

[9] P.Lippens, J.van Meerbergen, W.Verhaegh, A.van der Werf, "Allocation of Multiport Memories for Hierarchical Data Streams", *Proceedings IEEE International Conference on Computer-Aided Design*, pp.728-735, Santa Clara, Nov. 1993.

[10] P.Paulin, J.Knight, "Force-directed scheduling for the behavioral synthesis of ASIC's", *IEEE Transactions on Computer-Aided Design*, Vol.8, No.6, pp.661-679, June 1989.

[11] N.Passos, E.Sha, "Push-up scheduling: optimal polynomial-time resource constrained scheduling for multi-dimensional applications", *Proceedings IEEE International Conference on Computer-Aided Design*, San Jose CA, pp.588-591, Nov. 1995.

[12] L.Stok, "Data path synthesis", *INTEGRATION, the VLSI journal*, Vol.18, pp.1-71, June 1994.

[13] W.Verhaegh, P.Lippens, E.Aarts, J.Korst, J.van Meerbergen, A.van der Werf, "Improved Force-Directed Scheduling in High-Throughput Digital Signal Processing", *IEEE Transactions on CAD and Systems*, Vol.14, No.8, Aug. 1995.

[14] W.Verhaegh, "Multidimensional Periodic Scheduling", *Ph.D. dissertation*, Eindhoven University of Technology, Oct. 1995.