Systematic Cycle budget versus System Power Trade-off: a New Perspective on System Exploration of Real-time Data-dominated Applications*

Erik Brockmeyer, Arnout Vandecappelle, Francky Catthoor†
Desics, IMEC, Leuven, Belgium

*partly sponsored by the Esprit ESD-LPD project 25518 : DAB-LP
†Also professor at Katholieke Univ. Leuven, Belgium

In contrast to current design practice for (programmable) processor mapping, which mainly targets performance, we focus on a systematic trade-off between cycle budget and energy consumed in the background memory organization. The latter is a crucial component in many of today's designs, including multimedia, network protocols and telecom signal processing. We have a systematic way and tool to explore both freedoms and to arrive at Pareto charts, in which for a given application the lowest cost implementation of the memory organization is plotted against the available cycle budget per submodule. This by making optimal usage of a parallelized memory architecture. We indicate, with results on a digital audio broadcasting receiver and an image compression demonstrator, how to effectively use the Pareto plot to gain significantly in overall system energy consumption within the global real-time constraints.

1. INTRODUCTION

In data transfer intensive applications, a costly and difficult to solve issue is to get the data to the processor quickly enough. A certain amount of parallelism in the data transfers is usually required to meet the application's real-time constraints. Parallel data transfers, however, can be very costly. Therefore, the tradeoff between data transfer bandwidth cost should be carefully explored. This paper describes the potential tradeoffs involved, and also introduces a new way to tradeoff the data transfer cost with other components (illustrated for total system energy consumption).

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. 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. Increasing the memory bandwidth, on the other hand, is much more expensive and requires the introduction of different hierarchical layers, typically involving also multi-port memories. These memories cause a large penalty in area and energy though. 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. This has lead to our Data Transfer and Storage Exploration (DTSE) approach [6] which allows to systematically optimize data-dominated applications for energy and system bus load reduction. Here we focus on a particular use of the "back-end" of DTSE. The Storage Cycle Budget Distribution (SCBD) step focuses on making the tradeoff between the cost of providing a certain amount of memory bandwidth in a (partly) customized memory organisation, and the real-time constraints for the tackled submodule. An accurate data transfer ordering is needed to come up with a suitable memory architecture within the given timing constraints. This as such is an impossible task to do by hand, especially when taking sophisticated cost models into account [17].

In this paper, we will not go into detail on how to obtain the information for the curve itself [1, 4, 5], but we will show how to use them effectively in a low-power design context. Different cycle budgets assigned for the data transfer and storage requirements allow to (significantly) modify the required energy in the memory subsystem (Section 3). In Section 4, our tools come up with Pareto curves visualizing the useful tradeoff space between the cycle budget assigned to a given system submodule and its corresponding energy and/or area consumption, i.e. involving 3 search space axes. These can then be used to optimize the tradeoffs between different system components at the global system level (Section 5). These tradeoffs are illustrated with real life demonstrators.

2. RELATED WORK

In the register allocation domain, the allocation techniques start from a fully scheduled flow graph and are scalar oriented [13]. 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 [2] 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 [2].

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. [11, 14]. 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

ISLPED '00, Rapallo, Italy.

accesses (Phideo [14, 15], Notre-Dame rotatiom scheduler [10]). 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 steps can do a better job.

The main difference between our SCBD step used to obtain the Pareto curves discussed above, 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).

As far as we know, no systematic exploration techniques have been published to obtain the cycle budget versus energy consumption tradeoff curves related to the storage of array data types in complex background memory organisations for real-time data-dominated applications.

3. ENERGY COST OF DATA BANDWIDTH

High performance applications do need a high bandwidth to their data. This high memory bandwidth requirement will typically involve a high system cost. A clear insight in these costs is necessary to optimize the memory subsystem. The cost will nearly always boil down to energy and area/size. These are the relevant costs for the end user and manufacturer. A high memory bandwidth can be obtained in several ways. Basically, two classes of solutions exist to obtain a high bandwidth: high access speed over a single port (see Section 3.1) and high parallelism (see Section 3.2). Both come with large costs in energy and area. An alternative approach is to lower the application demand in terms of peak bandwidth (see Section 3.3). Tool support is then needed to fully exploit all this.

3.1 Costs in high speed memory architectures

New memory architectures are developed to provide a higher basic access speed. These memories can keep up with the data-path speed when used efficiently, but in many cases these can be quite energy consuming. Other memory types have complex IO profiles and specific design issues, which make the over design more complicated.

To have a better understanding, we first briefly explain the virtual operation of a modern SDRAM[1] (see Figure 1). We abstract the physical implementation details which we cannot influence (for instance the bitwidth, the circuits used, the floor-plan and the memory plane organisation). The SDRAM consists out of several banks which are accessed in an interleaved fashion to sustain a high throughput (pipelining principle). Every bank typically has several planes which can be activated separately. The "virtual" plane is defined as the total storage matrix entity which is activated (and consumes energy) when a particular memory access is performed. Within the planes there is a page, which is copied to a local latch register.

Pipelined memory implementations improve the throughput of a memory, but they don't improve the latency. For instance in an Extended Data Output (EDO) RAM, the address decoding and the actual access to the storage matrix (of the previous memory access) are done in parallel. Synchronous DRAMs pipelines further and can provide a huge theoretical bandwidth when accessed in the bank interleaved fashion. The address needs to be known several

[1]RAMBUS and other alternative architectures have the same basic concept.

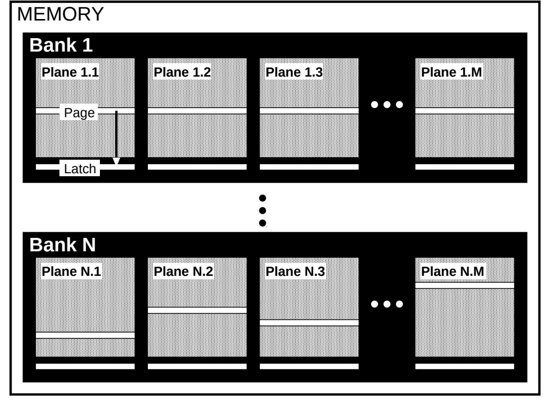

Figure 1: High-level synchronous DRAM organisation.

cycles prior to when the data is actually needed. Otherwise the data-path will stall, canceling the advantage of the pipelined memory. Therefore, the memory access ordering should match the bank interleaving constraint.

It is also beneficial to have successive accesses located in a single page. In an initial phase of a memory access, the row is selected. This row (=page) is sensed and partly or entirely stored in a local register. Next, the proper column is transfered to the output buffer. When the next access is within the same page, the row selection and sensing can be skipped. A high page hit rate is obtained by a proper memory access ordering and data layout. Typically a factor 2 to 4 in energy can be saved this way.

Large memories consist out of multiple planes which can be powered up separately. Multiple planes are used to avoid too long bit lines. By lowering switching between different planes the energy is saved further. This should again be enabled by optimizing the access ordering and the data placement in the memories.

These complex memory access optimizations cannot be dealt with manually. The larger the available cycle budget, the more freedom exists to optimize this memory access ordering. Hence, the cycle budget (or performance) can be traded off for energy reduction. We have tools which support these type of optimizations and can deal with the complex IO profiles [1].

3.2 Costs in highly parallel memory architectures

An orthogonal axis to enlarge the throughput, is increasing the amount of data elements transfered per memory cycle. This can be done by adding memories which can be accessed in parallel. This has a positive influence on both bandwidth and memory energy consumption. But the interconnect complexity increases, causing higher costs. The definition of the memory architecture and assignment of signals to these memories has a large impact on the available bandwidth. Multi-ported memories should be avoided, they increase the bandwidth roughly linearly with the number of ports but they also imply a large energy and area cost.

The memory architecture constraints must be minimal to keep the memory subsystem low cost. This is done by ordering the memory accesses in such a way that the needed parallelism and memory architecture is low cost[17, 5, 16].

3.3 Balance memory bandwidth

The previous two subsections have focused on how to use the memories efficiently and what can be done when we change the memory (architecture). In addition much can gained by balancing the bandwidth load over the application.

The required bandwidth and associated cost is as large as the maximum bandwidth needed by the application (see Figure 2). When a very high demand is present in a certain part of the code (e.g. a certain loop nest), the entire application suffers. By flattening this peak, an improved overall bandwidth is obtained (see lower part of Figure 2).

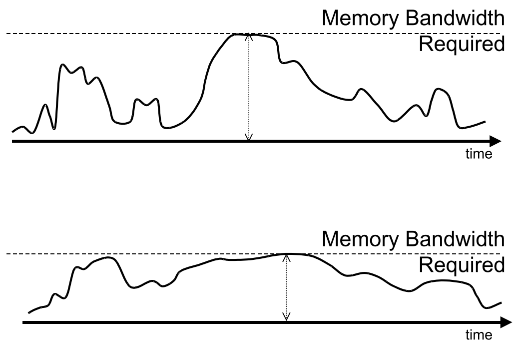

Figure 2: Balancing the bandwidth lowers the cost.

The rebalancing of the bandwidth load is again performed by reordering the memory accesses (also across loop scopes). Moreover, the overall cycle distribution over all the loops has a large impact on the peak bandwidth. Tool support is indispensable for this difficult task. An accurate memory access ordering is needed for defining the needed memory architecture.

4. ENERGY VERSUS SPEED TRADEOFF

The data transfer and storage ("memory") related cost clearly increases when a higher memory bandwidth is needed. The performance-cost tradeoff leads to the use of Pareto curves. This section explains the bounds of this tradeoff. The data transfer and storage related cycle budget ("memory cycle budget") is strongly coupled to the memory system cost (both energy and memory size/area are important here, as mentioned earlier). A Pareto curve is a powerful instrument to be able to make the right tradeoffs. The Pareto curve only represents the potentially interesting points in a search space with multiple axes and excludes all the solutions which have an equally good or worse solution for all the axes.

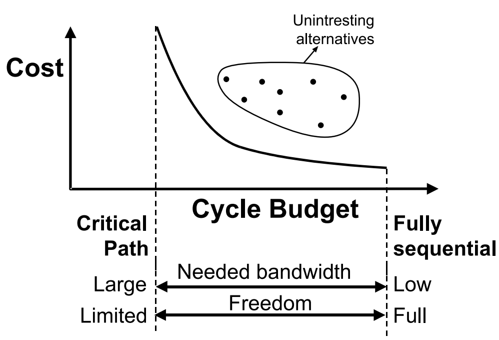

Figure 3: Pareto curve trades memory cycle budget vs. cost.

The memory cost increases when lowering the cycle budget (see Figure 3). When lowering the cycle budget, less ordering freedom is available causing a more energy consuming and more complex (larger) memory architecture is needed to deliver the required bandwidth. Note that the lower cycle budget is not obtained by reducing the number of memory accesses, as in the case of algorithmic changes and dataflow transformations. These higher-level code transformation related optimizations can be performed platform and cycle budget independent, prior to the tradeoff which is the focus here. Therefore, during the currently discussed step in the system design trajectory, the amount of data transfered to the datapath remains equal over the complete cycle budget range. Nevertheless, some control flow transformations and data reuse optimizations can be beneficial for energy consumption and not for the cycle budget (or vice versa). These type of optimizations still have to be considered in the tradeoff.

The interesting range of such a Pareto graph on the cycle budget axis, is defined from the critical path up to the fully sequential memory access path. In the fully sequential case all the memory accesses can be transferred over a single port (lowest bandwidth). However, the number of memories is fully free and the memory architecture can be freely defined. Therefore, the energy is the lowest. In the other extreme, the critical path, only a limited memory access ordering is valid. Many memory accesses are performed in parallel. Consequently, the constraints on the memory system are large and a limited freedom is available. As a result, the energy consumption goes up. Software pipelining techniques can be used to lower the critical path and increase performance. Note that this will come with even a higher bandwidth demand.

This energy-performance tradeoff can be made on real-life applications thanks to the use of powerful novel tools which we have developed in the last years[1, 5]. These tools optimize the memory access ordering using complex IO profiles. Next it constructs a optimized memory organisation which can deliver enough bandwidth. In our application domain the main power contributer is determined then and cost figures can be made up. The tools systematicly explore the optimization space steered by heuristics. The time complexity of the tools is moderate. For real applications (as given in the next section), the memory organisation constraints based on memory access ordering for the different cycle budgets are generated in several minutes. Next the memory organisation construction takes less then a minute for a near optimal solution per implementation. A script is added around the toolset, delivering a Pareto curve within reasonable time (¡¡hour). Synthesis tools cannot provide this fast feedback and make exploration of many possibilities infeasable. The tools do guarantee the excistance of a valid memory access schedule within the given timing constraints. Indeed, normal systhesis is needed for the final optimal solution only.

Up to now we have only published the single near optimal solution in the entire search space. To illustrate their use in this new Pareto curve context, two different demonstrators from different domains are explored. Firstly, the synchronisation core of a Digital Audio Broadcast (DAB) receiver. Secondly, a Binary Tree Predictive Coder (BTPC) for use in offset image compression. Both demonstrators, consist of several pages of *real* code containing multiple loop nests and arrays (in the range of 10 to 20) and a large number of memory access instructions (100 to 200).

4.1 Tradeoff in DAB application

The Digital Audio Broadcast (DAB) synchronization task is performed in three steps [3]. In the first step a correlation to a known reference signal determines the frequency error. Next, an IFFT is performed obtaining the channel impulse response. Finally, the channel impulse response is post-processed to find the optimal time error correction.

Figure 4 shows the tradeoff for the complete cycle budget range. This is obtained by letting the tool explore the cycle budget starting from the fully sequential mode, and then progress through the most interesting memory organisations (from a cost point of view) to reduce the cycle budget. The number of allocated memories is four for the entire graph shown in this figure, but an additional exploration can be added over that parameter, too. At a high cycle budget, only single-port memories are employed. When a dual-ported memory has to be added, a clear discontinuity is present in the energy function. In order to reduce the budget below 85 kCycles, dual-ported memories are indeed needed. The allocation of 4 dual-ported memories allows to decrease the cycle budget close to the critical path (55047 cycles). In principle, the tool could also explore even more costly memory organisations with multi-port memories but the cost becomes so huge then (if they would even exist in a realistic memory library) and the gains in cycle budget become so small that it does not make any practical sense to continue at that stage.

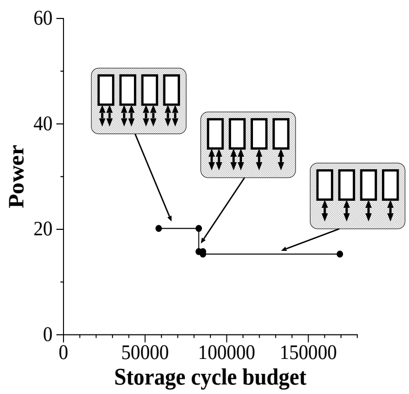

Figure 4: Pareto curve for Digital Audio Broadcast application.

4.2 Tradeoff in binary tree predictive coder

Binary Tree Predictive Coding (BTPC) [12] 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 neighboring pixels. The remaining error is then expected to achieve high compression ratios with an adaptive Huffman coder.

The overall Pareto curve is generated in the same way as for the DAB demonstrator. Now it is more complex because it uses a memory hierarchy and differently optimized code specifications, that are derived by applying several system-level DTSE transformations, prior to the individual Pareto curve generation (see Figure 5). More information about these different versions of the code can be found in [4, 16]. The largest (image-size) signals are stored in separate off-chip memory components. Four additional memories are allocated for the on-chip signals. The dotted curve part of the original implementation increases rapidly due to dual-ported off chip memories. Of course, such an implementation is not very realistic, demonstrating that code transformations are crucial for a cost-effective realisation.

The code optimizations are performed in several steps defined by the upper part of our DTSE script [6]. Platform-independent optimizations reduce the number of memory accesses and improve the data locality (box curve). Next, two different optimization paths are followed: an energy optimized one and a performance optimized one. The main difference is the modified use of the available memory hierarchy where using the technique of [8], another intermediate signal is created and assigned differently for the speed optimized implementation. As a consequence, an energy-intensive multi-port memory is added in the memory hierarchy, delivering more bandwidth and improving the worst-case execution path of the application. The real performance impact of adding such a intermediate memory can easily be evalutated using the tools.

The overall Pareto curve obviously combines both differently optimized "sub-Pareto" results. For the high cycle budget and low performance situations, the low energy implementation is better. Only for a very high performance, the speed optimized implementation should be used. We believe that a systematic exploration of this type of tradeoffs was not feasible prior to our work, and that this paper is the first to explore this on real-life demonstrators. We will now also show how these curves can be effectively used to significantly improve the overall system design of data-dominated applications.

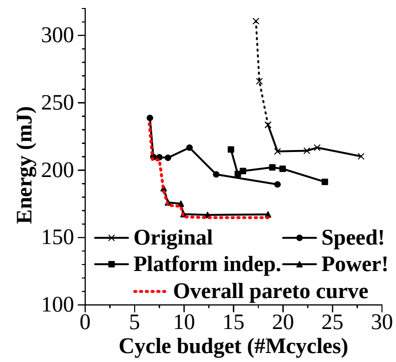

Figure 5: Pareto curve for Binary Tree Predictive Coder.

5. EXPLOITING THE USE OF THE PARETO CURVES

The previous section has shown the performance-cost tradeoff. In case of a single thread and the memory subsystem being energy dominant, the tradeoff can be based solely on the memory organisation Pareto curve. The given cycle budget defines the best memory architecture and its corresponding energy and area cost.

In most cases, designers of real-time systems will assume that the time for executing a given (concurrently executed) task is predefined by "some" initial system decision step. Usually, this is only seemingly so though. When several communicating tasks or threads are co-operating concurrently, then tradeoffs can be made between the cycle budget and timing constraints assigned to each of them, which makes the overall problem much more complex (see Section 5.2). Moreover, also in data-dominated applications the memory subsystem is not the only cost for the complete system. In particular after our DTSE optimisation steps, the data-path is typically not negligible any more afterwards, especially for instruction-set processor realisations (see Section 5.1).

5.1 Trading off cycles between memory organisation and data-path

Due to the use of latency hiding and a "system-level pipeline" between the computation and data communication, most of the cycles where memory access takes place are also useful for the data-path. Still, to reduce the cost of the data-path realisation, it is important to leave some of the total cycle budget purely available for computation cycles.

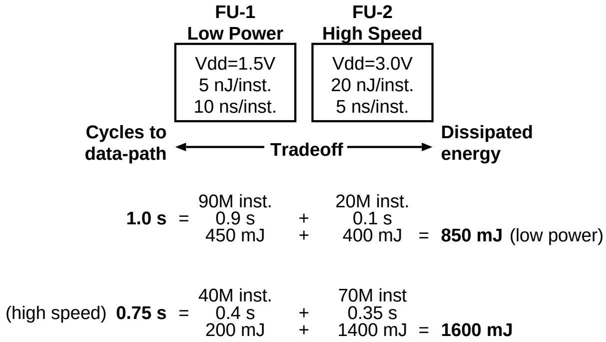

Figure 6: Datapath example of energy vs. speed tradeoff.

We will now illustrate how the assignment of spare cycles to the data-path can have a significant impact on its energy consumption. Assigning too little, indeed introduces an energy waste in the functional units when implemented in a low-power set-up. For instance, in several previous papers (see e.g. [7] and its refs) it has been motivated that functional units should operate at several voltages. Especially in recent papers, the number of different Vdd's has been restricted to a few specific values only because of practical reasons. Assume now that we either have a single unit that can operate part of the time at a higher Vdd and partly at a lower Vdd, or similarly two parallel units with different Vdd's over which all the operations can be distributed (see Figure 6). This duplication will require some extra area but on the entire chip this will involve a (very) small overhead and for energy reasons this can be motivated. The high Vdd unit will clearly allow a higher speed. In the simple example of Figure 6, 110 million instructions need to be executed for the entire algorithm. Most instructions can be executed in the low energy functional unit in a very "loose" data-path schedule (in total 1.0s $\rightsquigarrow$ consuming 850mJ). The same functionality executed 25% faster, consumes twice as much energy due to the requirement to execute more on the high speed functional unit (in total 0.75s $\rightsquigarrow$ consuming 1600mJ). The assignment of cycles to memory accesses and to the data-path is important for the overall energy consumption. Obviously, the contributions of both, the data-path and memory subsystem, are summed. This motivates a tradeoff between the memory system and the data-path itself.

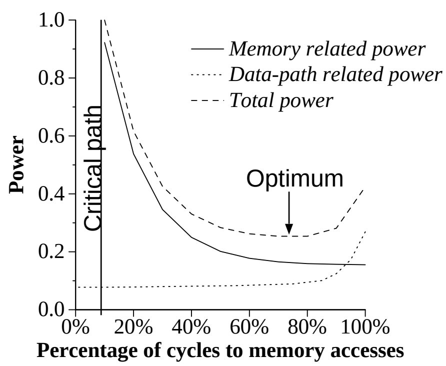

Figure 7: Trading off cycles for memory accesses versus cycles for data-path operations that due to dependency and ordering constraints cannot be executed in parallel with a data transfer.

The reality is still more complex though. 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 and the (large) overlap between memory accesses and computation (see Figure 7). However, the more time of the overall budget is committed to the memory accesses, the more restrictions are imposed on the data-path schedule which has less freedom for the ordering and the assignment to the functional units. As a result (see dotted curve), the Pareto curve for the data-path usually exhibits a cost overhead which is small and constant for small to medium memory cycle budgets, and which only starts to increase sharply when the memory cycle budget comes close to the globally available cycle budget. Due to the property that Pareto curve are monotonous in their behaviour, the sum of these 2 curves should exhibit a (unique) minimum. This also advocates for an accurate energy versus computation cycle budget modeling for the complex data-path functions.

5.2 Energy tradeoffs between concurrent tasks

The performance-energy function can be used even better to trade-off cycles assigned to different tasks in a complex system. Our Pareto curves should indeed be generated per concurrently executed task. It is then clearly visible that assigning too few cycles to a single task causes the cost of the entire system to go up. The Pareto curves also show the minimal (realistic) cycle budget per task. The cycle and energy estimates can clearly help the designer to assign concurrent tasks to processors and to distribute the cycles within the processors over the different tasks (see Figure 8). Minimizing the overall energy within a heterogeneous multi-processor is then possible by applying Pareto function minimization on all the energy-cycle functions together. Even when multiple tasks have to be assigned to a single processor under control of an operating system, interesting trade-offs are feasible.

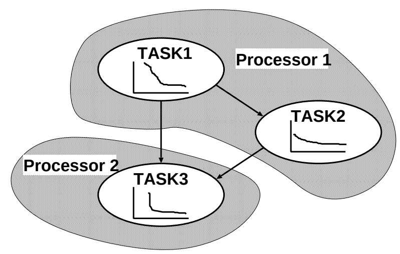

Figure 8: Tradeoff cycles assigned tasks.

Next, we will give a detailed illustration of this in the context of a DAB receiver. We focus here on three of its tasks. First there is an FFT task, to be executed for every symbol of a frame (in Mode I, a frame contains 76 symbols). Concurrent with this FFT is the synchronization; it works with the reference symbol (the 1^{st} symbol) of every frame, and consists of a correlation task followed by an Impulse Response Analysis (IR-Analysis) task. An overall real-time constraint of 96ms, the frame length, applies for all three tasks together. Assuming a speed of 40MHz, a low-speed and low-energy mode, 3840K cycles are available in total. In addition, constraints exist due to the delay before the data comes available from the receiver signal processing.

A naive design would allocate a separate processor for the FFT tasks and for the two correlation tasks. This would also require a separate memory organization and communication channels between the two processors. The top figure in Figure 9 shows how the tasks are organized in time, and what the memory architecture is for this processor. The arrows depicted with a "1" in Figure 10 are the corresponding points in the task Pareto curves.

With the correlation done in parallel to the FFT, the processing and memory resources are very much underutilized. A designer can then decide to map all three tasks onto one processor, and to make use of the "spare time" in the NULL symbol time to perform the correlation. However, the Pareto curve in Figure 10) shows that this leads to a very costly solution (the correlator task cost increases). A very high bandwidth to the memories is required to complete the correlation in time, and a four-port memory must be allocated to provide this bandwidth. The complexity of such a memory architecture makes this option simply infeasible.

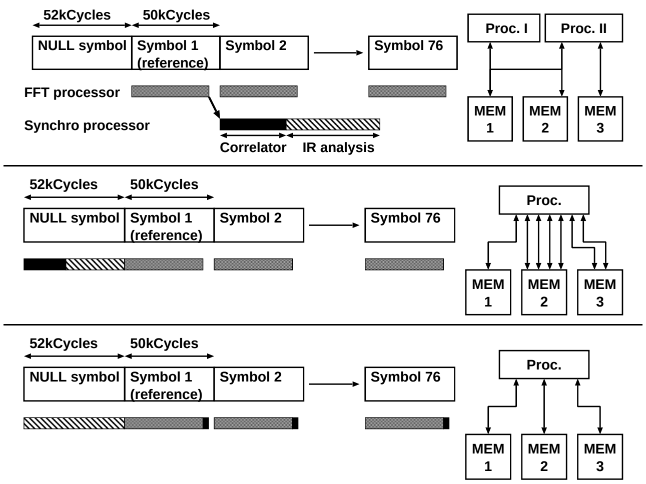

Figure 9: The original task to processor mapping needs 2 processors (top); A naive more energy consuming combination of the two tasks to a single processor (middle); An improved task rescheduling overcomes the cost increase (bottom).

From the FFT pareto curve of Figure 10 it is visible that there is some slack. This time can be used to do the correlation. Then only the IFFT and IR-analysis are performed during the NULL symbol time interval. A redefinition of the tasks corresponding to this decision, and the associated memory architecture, are shown at the bottom part of Figure 9. Finally the points anotated with "3" are chosen in the pareto curves of Figure 10 allowing a single processor with three memories.

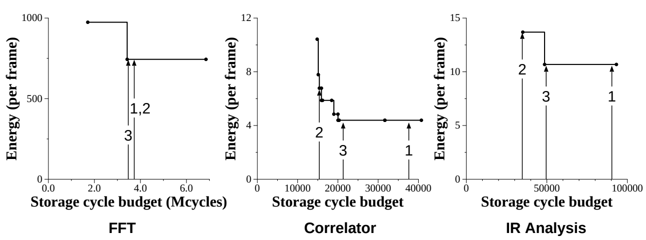

Figure 10: Pareto curves of three different subtasks of the DAB. The numbers at the arrows correspond to the implementation number of the previous figure.

Thus, by exploring the trade-off with the aid of the Pareto curves, the global system cost is effectively reduced (from two processors to one), while still meeting the real-time constraint and without an energy penalty.

6. CONCLUSION

Often the global system cost in terms of power/energy consumption, but also chip/board area or dollars, can be significantly reduced at the price of an acceptable increase in cycles spent in a particular submodule at the cost of less cycles for another less cost-sensitive submodule. Exploiting this tradeoff is only feasible for real complex systems if it can be systematically explored and when the useful tradeoff space is effectively visualized so that a designer can quickly evaluate the cost of different cycle budgets for a given submodule. In this paper, we have presented a methodology supported by tools to effectively achieve this. It has been demonstrated for several real-life multi-media applications.

7. REFERENCES

[1] Multi-media compilation project (Acropolis) at IMEC `http://www.imec.be/acropolis/`

[2] 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.

[3] 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.

[4] E.Brockmeyer, S.Wuytack, A.Vandecappelle, F.Catthoor, "Low power storage for hierarchical graphs", *3rd ACM/IEEE Design and Test in Europe Conf.*, France, Apr. 2000.

[5] E.Brockmeyer, S.Wuytack, A.Vandecappelle, F.Catthoor, "Low power storage cycle budget tool support for hierarchical graphs", to appear in *Proc. 10th ACM/IEEE Intnl. Symp. on System-Level Synthesis*, Madrid, Spain, pp., Sep. 2000.

[6] 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.

[7] A.Chandrakasan, V.Gutnik, T.Xanthopoulos, "Data driven signal processing: an approach for energy efficient computing", *Proc. IEEE Intnl. Symp. on Low Power Design*, Monterey CA, pp.347-352, Aug. 1996.

[8] 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.

[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] N.Passos, E.Sha, "Push-up scheduling: optimal polynomial-time resource constrained scheduling for multi-dimensional applications", *Proc. IEEE Int. Conf. Comp. Aided Design*, San Jose CA, pp.588-591, Nov. 1995.

[11] 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.

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

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

[14] 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 Trans. on CAD and Systems*, Vol.14, No.8, Aug. 1995.

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

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

[17] S.Wuytack, F.Catthoor, G.De Jong, H.De Man, "Minimizing the Required Memory Bandwidth in VLSI System Realizations", *IEEE Trans. on VLSI Systems*, Vol.7, No.4, pp.433-441, Dec. 1999.