Hot topic session: How to solve the current memory access and data transfer bottlenecks: at the processor architecture or at the compiler level?

Francky Catthoor,
IMEC , Leuven, Belgium (catthoor@imec.be)

Nikil D. Dutt,
U.C.Irvine, CA (dutt@ics.uci.edu)

Christoforos E. Kozyrakis,
U.C.Berkeley, CA (kozyraki@ikaros.CS.Berkeley.EDU)

Abstract

Current processor architectures, both in the programmable and custom case, become more and more dominated by the data access bottlenecks in the cache, system bus and main memory subsystems. In order to provide sufficiently high data throughput in the emerging era of highly parallel processors where many arithmetic resources can work concurrently, novel solutions for the memory access and data transfer will have to be introduced.

The crucial question we want to address in this hot topic session is where one can expect these novel solutions to rely on: will they be mainly innovative processor architecture ideas, or novel approaches in the application compiler/synthesis technology, or a mix.

1. Motivation and context

At the processor architecture side, previous work has focused on microarchitecture enhancements like intelligent management of cache hierarchies, streaming buffers and value or address prediction, techniques that exploit spatial/temporal locality in memory references. But can this approach provide the memory performance necessary to feed highly parallel processors? We will discuss alternative instruction set architectures that attempt to provide the hardware with explicit information about parallelism in memory, and examine the opportunities and challenges from the emerging combination of multiprocessing and multithreading in a single chip.

At the side of the system design technology and compilation for embedded data-dominated multi-media applications, also much evolution is present. We will show that decisions made at this stage heavily influence the final outcome when the appropriate architectural issues of the embedded memories are correctly incorporated. What is more controversial still however is how to deal with dynamic application behaviour, with multithreading and with highly concurrent architecture platforms.

2. Explicitly parallel architectures for memory performance enhancement (Christoforos E. Kozyrakis)

While microprocessor architectures have significantly evolved over the last fifteen years, the memory system is still a major performance bottleneck. The increasingly complex structures used for speculation, reordering, and caching have improved performance, but are quickly running out of steam. These techniques are becoming ineffective in terms of resource utilization and scaling potential. In addition, memory system performance will become even more critical in the future, as the processor-memory performance gap is increasing at the rate of 50% per year [28]. Multimedia and embedded applications, that are expected to dominate the processing cycles on future microprocessors, have significantly different memory access characteristics compared to typical engineering workloads [26]. Temporal locality is not always available, leading to poor performance from traditional caching structures.

We believe the enhancement of memory system performance requires the synergy of software and hardware techniques. Each system component should be focused on the task it is most efficient with:

- Compilers and/or run-time tools can view and analyze several hundred lines of source code. They can detect instruction/data parallelism and regular memory access patterns, or transform the code so that parallelism and desired patterns for the given hardware are created (see the two following sections).

- The processor can utilize the parallelism information to execute concurrently a large number of memory and computation operations using the massive hardware resources available in current and future chips. The run-time (dynamic) information available to the hardware can also be used to apply further optimizations within a small window of instructions.

The instruction sets used by CISC and RISC processors today are inherently sequential and hide from the hardware the parallelism and memory access information that is available at various software levels. For example, a group of independent operations are described to the hardware using sequentially ordered instructions. Complex dependency analysis has to be performed in hardware for the parallelism to be "re-discovered" and utilized. Similarly, a set of sequential or strided accesses to an array will be expressed as a sequence of simple loads or stores to a single memory location. To apply any prefetching or other memory access optimization, the hardware must observe the sequence of addresses and guess the access pattern first.

To enable high-performance, yet efficient, processor designs, future instruction set architectures (ISAs) should allow software to pass explicit parallelism and memory access information to the hardware. Instructions should explicitly identify operations that can be issued and executed in parallel. Memory instructions should allow control of memory system features such as cache space allocation, and provide information that describes the access characteristics of the application. This could include access pattern (e.g. sequential or strided), temporal and spatial locality hints, and expected caching behavior at the various levels of memory hierarchy. Using this information, the processor can issue in parallel or overlap a large number of memory accesses, employ aggressive prefetching, and tune the use of the memory hierarchy for maximum performance and efficiency. From the point of view of compilers and run-time tools, explicitly parallel architectures allow fine control of hardware features and create new opportunities for higher-level optimizations.

There have been several recent examples of explicitly parallel architectures both from academia and industry. Some examples of different approaches are:

- The EPIC (IA-64) architecture [27] exposes parallelism to the hardware using VLIW instructions. Memory operations provide explicit hints about the expected caching behavior at each level of the memory hierarchy, which is used for guiding cache space allocation. Software speculation is used to issue memory accesses as early as possible without compromising program correctness.

- The VIRAM architecture [30] expresses parallelism to hardware in the form of vector operations. Vector memory instructions explicitly specify a large number memory accesses to be issued in parallel, along with their spatial relation (sequential, strided, or indexed accesses). The architecture includes support for software speculation as well. The implementation allows control of address mapping at a per process or a per memory page granularity.

- The Impulse architecture [25] allows software to describe regular memory access patterns directly to the memory controller. The controller accesses memory in an optimized manner for each pattern, performs prefetching and groups the requested data for maximum caching efficiency. In addition, it provides support for data remapping by the application or the compiler.

While instruction set architectures should focus on exposing parallelism to the hardware, processor design should focus on implementations of these ISAs that can tolerate high memory latency, even in the case of poor caching behavior. Increased memory latency is a technology problem unlikely to be solved in the near future. On the other hand, high memory bandwidth is already available through technologies like multi-bank embedded DRAMs, high-performance memory interfaces (Rambus, DDR), and cost-efficient MCM packaging. Processor designs can utilize high memory bandwidth in order to hide the performance penalty of high memory latency. This is in harmony with the characteristics of many multimedia and e-commerce workloads, where throughput is by far more important than latency. Frequently, latency can also be addresseed with software techniques like those presented in the next section. But hardware techniques for Ha hiding latency are still necessary as they can be applied to all applications, regardless of the availability of source code or their suitability to compiler analysis.

There are several architectural approaches to utilizing high memory bandwidth. Some of the techniques proposed or used recently include the following:

- Multithreaded processors, such as Sun MAJC and Compaq Alpha EV-8, attempt to execute concurrently several fine-grain threads. When some thread is blocked due to memory latency (e.g. a cache miss) or lack of parallelism, the hardware switches to issuing instructions from another thread within a couple of clock cycles. Given a large number of fine-grain threads and high memory bandwidth, multithreaded designs can efficiently hide high memory latency.

- Multiprocessing designs combine two to four processors on the same chip forming a symmetric multiprocessor (IBM Power4, Sun MAJC, Stanford Hydra). Each processor can run a separate execution thread,

process or application. The high memory bandwidth is used to feed data to the multiple processors and keep the overall system busy despite the high latency of each individual access.

- Vector processors (Berkeley VIRAM, Cray SV-2) utilize the vector instructions in order to have a large number of memory accesses pending at any time from a single execution thread. Vector designs can tolerate high latency by deeply pipelining memory accesses. They also amortize access delay over a large number of operations (vector loads). For multimedia applications, such processors can achieve high performance even without any SRAM caches [29].

In conclusion, memory performance can be enhanced by properly employing synergetic hardware and software techniques. Instruction set architectures must expose explicit parallelism information to the hardware, such as instructions independence and memory access pattern characteristics. The processor can use this information to issue a large number of memory accesses in parallel, and make optimal caching and prefetching decisions for the specific application. The second major focus for enhancing memory system performance in hardware is hiding memory latency by utilizing the high memory bandwidth available. Multitheading, multiprocessing and vector architectures can be used, or even combined, to address this issue. Given the memory latency and bandwidth trends, and the fact that multimedia and server applications require mainly high throughput, we expect such architectures to be the main focus of processor design in the near future.

3. Compile-time techniques to remove the data transfer and storage bottlenecks (Francky Catthoor)

Current high-level compilation and system design technology research for ”general-purpose” systems realized on programmable targets is mostly focussing on the pure CPU performance issues (see e.g. [2, 10]). For embedded systems the emphasis should especially incorporate issues like system bus loading, embedded memory size and energy consumption (LSE costs). These require (at least partly) different techniques from pure CPU performance oriented approaches [4]. For data-dominated applications, especially in multi-media and telecom, this means that the emphasis should shift to the reduction of the data accesses in the higher levels of the hierarchical memory organisation (see fig. 1). At the same time, also the data storage requirements should be limited as much as possible, especially in the lower levels of this same memory hierarchy. The current L1 sizes lead to a too high power and area budget for the embedded memory component [11, 20].

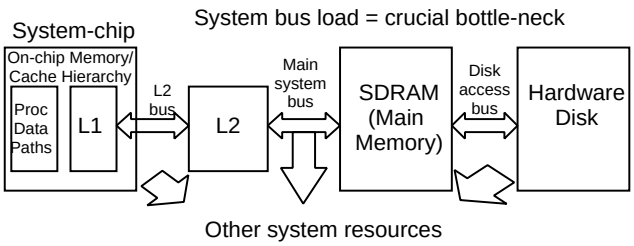

Figure 1. System bus load issue at the board level

Apart from the hardware solutions which are emerging to help to address these bottlenecks (see previous section), a clear need exists to also address them at the compiler level, as also advocated in [1]. In this stage, high-level source-to-source code transformations can indeed heavily influence all these relevant cost issues for embedded systems. Previous work has demonstrated that important gains can be achieved for all 3 LSE factors combined, without having to sacrifice pure CPU performance (see e.g. [6, 17, 7]). These savings are achieved by techniques which are actually complementary to the approaches which are typically taken to remove the data supply bottleneck for the CPU [14]. For instance, hiding the memory bandwidth and latency (e.g. by prefetching) is not an issue for reducing the LSE cost. In most cases prefetching will even increase the LSE.
Instead the focus should lie on (see also fig. 2):

- carefully removing the unnecessary transfers between the memory hierarchy levels by judiciously selecting which data are copied when from the main memory storage (e.g. the disk) to the lower levels. A compile-time data reuse and memory hierarchy decision step [9] can select how many partitions are useful in a given application, in which partition each of the signals (most of which are only intermediate data in data-dominated applications) and its temporary copies will be stored, and the connections between the different memory partitions (if e.g. bypassing is feasible). An important goal in this step is to perform code transformations which introduce extra transfers between the different memory partitions and which are mainly reducing the power cost. In particular, these involve adding loop nests and introducing temporary values – to be assigned to a “lower level” – wherever a signal in a “higher level” is read more than once. This clearly leads to a trade-off between the power (and partly also memory size) lost by adding these transfers, and the power gained by having less frequent access to the larger more remote memories at the higher level(s). This systematic exploration step is complementary to

conventional compiler approaches.

- distributing the timing/throughput constraints over the different concurrent tasks, conditional paths and loop nests so that no unnecessary bus load bottlenecks occur. Even for a fully predefined memory organisation, this step is still required to achieve the maximal exploitation of the available memory ports and this with a minimized amount of memory bank activation (especially within the external memories, which are typically based on the SDRAM concept) [5].

- assigning data to different memory partitions or banks (internal in the SDRAM e.g.) such that the number of activations of the memory planes is reduced to a minimal [5]. Purely for performance improvement, memory bank assignment has been proposed in [24] but this is not sufficient. In [21] also the size of the embedded memories is addressed but it has a limited application scope.

- distributing the data over the different memories in each hierarchical layer. Especially in the lower (on-chip) layers, choices are becoming available for multimedia oriented instruction-set processors, and these should be exploited to reduce the memory power and size [20, 23]. So further effort is required here.

- exploiting the limited life-times of the data in the program to overlap them in an as tight as possible data layout to reduce both the main memory size and the capacity misses in the L2 or L1 cache. An advanced in-place mapping approach [8, 15, 22] enables a better exploitation of the caches and it allows to remove nearly all capacity misses. The results are very promising both for software controlled caches [11] as in the Philips TriMedia processor, and for more traditional hardware controlled ones like the Pentium [12]. At the same time, this technique reduces the overall size of the required (external) memories (and hence C_{load}).

- reorganizing the data layout in the main memory to remove all conflict misses in all the cache levels, even when this comes at a cost in (a controlled) code and data memory size and when it does not improve the CPU performance. Extended padding [19] and even more aggressive main memory data layout reorganisations allow to remove nearly all the conflict misses for non fully-associative caches.

We have shown that decisions made in such a "precompilation" stage heavily influence the final outcome of conventional compilers, when the internal organisation of the SDRAM and embedded memories is correctly incorporated. This has been demonstrated on several real-life exam-

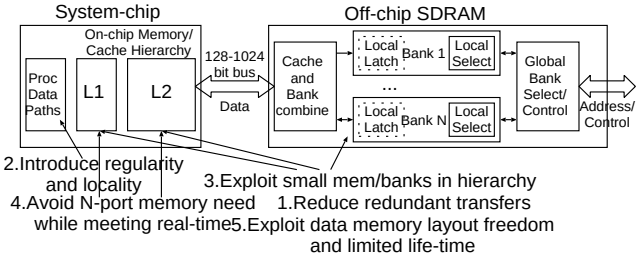

Figure 2. Steps used in ACROPOLIS methodology for reducing the data transfer and storage cost in the background memory hierarchy.

ples like medical imaging [7], voice coding [11], and video coding [6, 17].

All these steps are based on compile-time analysis of the source code and modifications to the way this code is written. For instance, for the data reuse step, new code with extra temporary variables and the corresponding loop nests, is explicitly entered. As a result of all these steps, the code of the critical kernels indeed increases in size, but the overall size expansion is very acceptable (less than 10%) for complex (real) applications. The main bottleneck to apply these techniques is however the complexity of the resulting code itself, especially in terms of index expressions, conditions and loop bounds. To produce this code in a purely manual way is too tedious and error-prone so novel high-level compilation techniques are essential to support the application designer in this stage. In addition, when the code produced by these steps is entered as such in a "low-level" compiler stage, the CPU penalty due to this complex code will be large. Therefore, additional precompiler steps are required to remove this again. We have shown that this removal is not only feasible, but due to the full elimination of the data transfer bottle-neck in the initial DTSE stage, it is typically even possible to achieve speed-ups compared to the non-optimized code of a factor 2 and more [7]. But again novel compiler techniques have to be developed to achieve this fully automatically [16].

What is more controversial still however is how to deal with dynamic application behaviour, with multithreading and with highly concurrent architecture platforms. Static compile-time analysis is then not sufficient any longer to provide enough information to steer either the traditional or the novel code transformation steps. Still, also here we believe that the boundary of what can be reached is much further than what is currently believed. Much of the worst-case analysis for data-dependent constructs can indeed by augmented by profiling analysis to identify the likely cases. Based on this information, code can be added which works also in the worst-case, but which gives even better results

for the most likely cases. The potential of this has been demonstrated in experiments on highly data-dependent applications like graphics libraries (e.g. OpenGL) [13] and the emerging multi-media compression standard MPEG4 [3].

In conclusion, we believe that the first reaction on the "new" wave of emphasis on the data transfer and storage bottlenecks should be to look for novel compiler techniques, which augment (and are complementary to) the existing ones. These should not concentrate merely on CPU performance but especially on the LSE costs which are crucial especially in an "embedded" system context (as in the explosive multi-media or telecom markets). This applies even for (highly) data-dependent and concurrent algorithms. Only when nothing more can be achieved at that stage, or when the cost of achieving the gains surpasses a reasonable level, the remainder of the problem should be solved by adding appropriate hardware support which can exploit the dynamic run-time analysis complementing the compile-time techniques. Currently, many of the proposed hardware solutions are too ambitious and are even hampering the application of compile-time techniques so that compiler developers have to attempt circumventing the "wrong" steering by this hardware. The latter is usually only partially feasible and the outcome is an overall suboptimal result. Experiments for a voice coder application on a fully hardware controlled cache versus a partially software lockable one, have clearly illustrated this [11, 12].

4. Processor-Memory Architecture Exploration using Memory-Aware Software Toolkit Generation (Nikil D. Dutt)

Advances in System-on-Chip (SOC) technology make it possible to utilize *customizable* embedded processor cores, together with a variety of novel on-chip/off-chip memory hierarchies, allowing customization of SOC architectures for specific embedded applications and tasks. Traditionally, computer architects have used simulation studies to perform architectural exploration and evaluate the effects of changes to the architecture (resources, connectivity, storage) of processor systems. Such studies require, at a minimum, a compiler and simulator that accurately model the architecture. The need for exploration tools and environments becomes even more critical since while we are faced with rapidly shrinking time-to-market windows, we are also faced with a plethora of choices:

- diverse processor architectural choices, ranging from more "traditional" RISC, VLIW, and DSP, to hybrid VLIW/DSP, EPIC, multi-threaded, and vector-machines
- reusable hard and soft (parameterizable) IP cores
- new families of memory components (e.g., on-chip DRAM, SDRAM, DDR, RAMBUS, etc.)
- heterogeneous memory organizations, including disjoint memory address spaces, multiple cache hierarchies, direct-mapped memory spaces, stream/vector memories, and adaptive cache/memory control
- new coarse- and fine-grain compilation techniques to alleviate the memory bottleneck

These choices present us with a sort of "embarrassment of riches": how do we select the right (or acceptable) processor-memory configurations to meet the multiple design goals (performance, power, code/memory size, etc.), and yet do this rapidly?

This is a major challenge in the context of memory-dominated applications, since current tools do not allow rapid exploration and evaluation of this multi-objective design space for an embedded systems/embedded SOC target. Thus not only are individual techniques important, but a method to bring them all together so that the system designer can try (explore) and refine (optimize) promising processor-memory configurations, also with the goal of integrating both compile- and run-time mechanisms.

The only hope for achieving increasingly shorter time-to-market windows for such memory-intensive systems, is a software toolkit generation approach, driven by an Architecture Description Language (ADL) [31]. The ADL specifies the detailed processor architecture (including static and dynamic execution units) *and* the detailed memory-subsystem architecture (as well as any partitionings/mappings). The ADL is then used to (semi-)automatically generate a customized software toolchain for the specified processor-memory configuration, comprising estimators, a memory-aware compiler, (functional, cycle-, phase-, bit- accurate) simulators, assemblers, and assorted software support tools such as debuggers and validation/verification subsystems [32, 33].

Since compiler- and simulator-generators have traditionally produced inferior quality tools/results, many will question the feasibility of this toolkit generation approach. Firstly, time-to-market concerns will leave no alternatives *but* to use this toolkit generation approach. Secondly, by limiting the templates of the processor-memory architectural configurations, we can also achieve high-quality results. (However, we may also miss many other superior processor-memory architecture candidates.) The challenge is to achieve generality without loss of fidelity (i.e., consistency in estimation) during design space exploration.

While earlier sections have focused mainly on architectural styles and (coarse-grain) compiler techniques related to memory-dominated designs, many additional opportunities for memory optimization exist at the specification level,

at the operating system level, at the fine-grain (ILP) level, as well as using combinations of these; due to limited space, we highlight only some of them below.

- *Dealing with Dynamic Data Structures.*
  Dynamic data structures in the application behavior have traditionally posed a significant challenge in management of memory traffic. Recently several lines of work have sought to address this issue. First, through profiling data, actual behavior can be modeled when good stimuli streams are selected – this partially alleviates the "unknown" nature of dynamic data types. Second, the abstract (unoptimized) data types in the input specification can be transformed into more optimized (customized) data types for the specific application [40]. Third, the virtual memory management system provided by default operating system can be replaced by customized (and therefore optimized) memory management system [43, 44]. Both of the above approaches have been applied in a global manner at IMEC for system design involving dynamic data types [42, 41]; it is important to note that while these approaches significantly reduce the system memory traffic (improving system performance), they also indirectly reduce power and the system bus load. Finally, as the need for aggressive speculative compiler technology increases, more sophisticated disambiguation/anti-aliasing techniques, relying heavily on symbolic analysis will also become necessary. In particular, traditional optimizing compiler approaches are at best conservative for data-dominated applications that heavily employ pointer-based data structures. Indeed, pointer-based data structures are traditionally notorious for resulting in poor performance – mainly due to high data cache miss ratios resulting from a fragmented allocation of these data structures in memory. Approaches developed at UC Irvine, including semantic retention [34] and sophisticated pointer analysis [35], allow for intelligent management of dynamic and pointer-based data structures to radically improve data cache hit ratios (e.g., through data layout).

- *Operating System Support for Memory Management.*
  With increasing amounts of on-chip multiprocessing and multithreading, new techniques will have to be developed to take advantage of fast communication/synchronization features for multiprocessors on a chip. Integration of operating systems with compiler technologies will become increasingly important for such multiprocessors. Previous sections have already discussed compiler support for memory management (of the cache hierarchy). Additionally, techniques need to be developed to compile and embed mini-operating systems and custom (application-specific) runtime libraries in user code. For instance, in the presence of dynamic data structures, static compiler techniques can go only so far in optimizing the memory characteristics. As described earlier, at IMEC techniques have been developed to customize the default OS-based memory management system with a more customized one for the specific application, both for access-dominated [43] and for size-dominated [44] data types. At UC Irvine, we have also developed techniques that analyze usage patterns for such data structures, which in turn can guide the memory manager to intelligently allocate such data structures for improved data cache performance.

- *Integration/Unification of Coarse- and Fine-grain Parallelism.*
  Traditionally the steps of coarse-grain (e.g., task and loop) parallelization and fine-grain (Instruction Level – ILP) parallelism was done separately. However, interactions between the two phases can result in many opportunities for modifying the range of performance, memory and power profiles of the generated software. The PROMIS project being developed at UIUC/UCIrvine [37] provides such a platform for considering the interactions between these two traditionally distinct compiler phases. Furthermore, there is an urgent need for estimation tools that can predict the impact of parallelism on the amount of memory size at the system level to help guide these coarse- and fine-grain compiler transformations (e.g., the MEMOREX project [38]).

- *Memory-aware Instruction-Level Parallelization (ILP) Techniques.*
  Last, but not the least, more aggressive memory-aware ILP techniques are further required to optimize both the memory requirements, as well as to compile efficiently (e.g., for low power) given a specific memory architecture and organization. For instance work at UC Irvine has examined extraction of timing information from both the application behavior, *and* detailed timing protocols of the processor-memory subsystem to create a customized, timing-driven memory-aware compiler for the instantiated processor-memory architecture [39]. Our preliminary experiments show that a memory-aware ILP compiler can combine the detailed timing and access mode information for memories with the processor architecture pipeline timings to get performance improvements of 30-50 % over and above existing ILP optimizing compilers. Moreover, this interaction opens up a larger space of other feasible design points that can be further optimized for memory size and power.

Looking forward, the diversity of both processor architectural styles *and* memory modules/configurations will necessitate a software-toolkit generation approach that permits many different configurations to be evaluated rapidly. Such an approach will allow the system designer to effectively achieve cooperation between hardware-driven (e.g., run-time mechanisms, special architectural/IS features) and software-oriented (e.g. sophisticated compile-time analyses, and optimizations, assisted by profiling information) techniques for rapid exploration/evaluation of different architectures, as well as detailed tuning/refinement of promising processor-memory configurations to meet the desired system constraints. Finally, the traditional notion of architectural regularity (which has been advocated as a means to overcome complexity in both hardware and software design), needs to be be replaced with the notion of *heterogeneous architectural regularity*: finding heterogeneous islands of processor-memory configurations customized for the application/domain behavior, and then imposing a measure of regularity within each processor-memory configuration.

References

[1] S.Adve, D.Burger, R.Eigenmann, A.Rawsthorne, M.Smith, C.Gebotys, M.Kandemir, D.Lilja, A.Choudhary, J.Fang, P-C.Yew, "The interaction of architecture and compilation technology for high-performance processor design", *IEEE Computer Magazine*, Vol.30, No.12, pp.51-58, Dec. 1997.

[2] U.Banerjee, R.Eigenmann, A.Nicolau, D.Padua, "Automatic program parallelisation", *Proc. of the IEEE*, invited paper, Vol.81, No.2, pp.211-243, Feb. 1993.

[3] E.Brockmeyer, L.Nachtergaele, F.Catthoor, J.Bormans, H.De Man, "Low power memory storage and transfer organization for the MPEG-4 full pel motion estimation on a multi media processor", *IEEE Trans. on Multi-Media*, Vol.1, No.2, pp.202-216, June 1999.

[4] F.Catthoor, "Energy-delay efficient data storage and transfer architectures and methodologies: current solutions and remaining problems", special issue on "IEEE CS Annual Workshop on VLSI" (eds. A.Smailagic, R.Brodersen, H.De Man) in *Journal of VLSI Signal Processing*, Vol.21, No.3, Kluwer, Boston, pp.219-232, July 1999.

[5] 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.

[6] K.Danckaert, F.Catthoor, H.De Man, "System-level memory management for weakly parallel image processing", *Proc. EuroPar Conference*, Lyon, France, August 1996. "Lecture notes in computer science" series, Vol.1124, Springer Verlag, pp.217-225, 1996.

[7] K.Danckaert, F.Catthoor, H.De Man, "Platform independent data transfer and storage exploration illustrated on a parallel cavity detection algorithm", *Proc. ACM Conf. on Par. and Dist. Proc. Techniques and Applications*, PDPTA'99, Vol.III, pp.1669-1675, Las Vegas NV, June 1999.

[8] E.De Greef, F.Catthoor, H.De Man, "Reducing storage size for static control programs mapped onto parallel architectures", presented at *Dagstuhl Seminar on Loop Parallelisation*, Schloss Dagstuhl, Germany, April 1996.

[9] 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 CA, pp.30-35, Aug. 1997.

[10] M.Hall, J.Anderson, S.Amarasinghe, B.Murphy, S.Liao, E.Bugnion, M.Lam, "Maximizing multiprocessor performance with the SUIF compiler", *IEEE Computer Magazine*, Vol.30, No.12, pp.84-89, Dec. 1996.

[11] C.Kulkarni, F.Catthoor, H.De Man, "Cache transformations for low power caching in embedded multimedia processors", *Proc. Intnl. Parallel Proc. Symp.(IPPS)*, Orlando FL, pp.292-297, April 1998.

[12] C.Kulkarni, F.Catthoor, H.De Man, "Hardware cache optimization for parallel multimedia applications", *Proc. EuroPar Conference*, Southampton, UK, Sep. 1998.

[13] C.Kulkarni, F.Catthoor, H.De Man, "Optimizing Graphics Applications: A Data Transfer and Storage Exploration Perspective", accepted for *Proc. 1st Wsh. on Media Proc. and DSPs*, in IEEE/ACM Intnl. Symp. on Microarchitecture, MICRO-32, Haifa, Israel, Nov. 1999.

[14] G.Lee, P.Yew, special issue on "Interaction between Compilers and Computer Architectures", *IEEE TC on Computer Architecture Newsletter*, June 1997.

[15] V.Lefebvre, P.Feautrier, "Optimizing storage size for static control programs in automatic parallelizers", *Proc. EuroPar Conference*, Passau, Germany, August 1997. "Lecture notes in computer science" series, Springer Verlag, Vol.1300, 1997.

[16] M.Miranda, F.Catthoor, M.Janssen, H.De Man, "High-level Address Optimisation and Synthesis Techniques for Data-Transfer Intensive Applications", *IEEE Trans. on VLSI Systems*, Vol.6, No.4, pp.677-686, Dec. 1998.

[17] L.Nachtergaele, T.Gijbels, J.Bormans, F.Catthoor, M.Engels, "Power and speed-efficient code transformation of multi-media algorithms for RISC processors", *IEEE Intnl. Wsh. on Multi-media Signal Proc.*, Los Angeles CA, pp. 317-322, Dec. 1998.

[18] P.R.Panda, N.D.Dutt, A.Nicolau, "Efficient utilization of scratch-pad memory in embedded processor applications", *Proc. 5th ACM/IEEE Europ. Design and Test Conf.*, Paris, France, pp., March 1997.

[19] P.R.Panda, H.Nakamura, N.D.Dutt and A.Nicolau, "A data alignment technique for improving cache performance", *Proc. IEEE Int. Conf. on Computer Design*, Santa Clara CA, pp.587-592, Oct. 1997.

[20] P.R.Panda, N.D.Dutt, A.Nicolau, "Data cache sizing for embedded processor applications", *Proc. 1st ACM/IEEE Design and Test in Europe Conf.*, Paris, France, pp.925-926, Feb. 1998.

[21] P.R.Panda, "Memory bank customization and assignment in behavioral synthesis", *Proc. IEEE Int. Conf. Comp. Aided Design*, Santa Clara CA, pp.477-481, Nov. 1999.

[22] F.Quillere, S.Rajopadhye, "Optimizing memory usage in the polyhedral model", *Proc. Massively Parallel Computer Systems Conf.*, April 1998.

[23] P.Slock, S.Wuytack, F.Catthoor, G.de Jong, "Fast and extensive system-level memory exploration for ATM applications", *Proc. 10th ACM/IEEE Intnl. Symp. on System-Level Synthesis*, Antwerp, Belgium, pp.74-81, Sep. 1997.

[24] A.Sudarsanam, S.Malik, "Memory bank and register allocation in software synthesis for ASIPs',, *Proc. IEEE Int. Conf. Comp. Aided Design*, San Jose CA, pp.388–392, Nov. 1995.

[25] J. Carter et al. Impulse: Building a Smarter Memory Controller. In *5th Int. Conference on High Performance Computer Architecture*, pages 70–9, Jan. 1999.

[26] K. Diefendorff and P. Dubey. How Multimedia Workloads Will Change Processor Design. *IEEE Computer*, 30(9):43–45, Sept. 1997.

[27] C. Dulong. The IA-64 architecture at work. *IEEE Computer*, 31(7):24–32, July 1998.

[28] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, 1996.

[29] C. Kozyrakis. A Media Enhanced Architecture for Embedded Memory Systems. Technical Report UCB//CSD-99-1059, University of California at Berkeley, 1999.

[30] C. Kozyrakis and D. Patterson. A New Direction in Computer Architecture Research. *IEEE Computer*, 31(11):24–32, Nov. 1998.

[31] A. Halambi, P. Grun, H. Tomiyama, N. Dutt, and A. Nicolau. Automatic Software Toolkit Generation for Embedded Systems-On-Chip. In *Proc. ICVC-99*, Seoul, Korea, October 1999.

[32] A. Halambi, P. Grun, V. Ganesh, A. Khare, N. Dutt, and A. Nicolau. EXPRESSION: A language for architecture exploration through compiler/simulator retargetability. In *Proc. DATE-99*, March 1999.

[33] P.R. Panda, N.D. Dutt and A. Nicolau. *Memory Issues in Embedded Systems-on-Chip: Optimizations and Exploration.* Kluwer Academic Publishers, Boston, Massachusetts, 1998.

[34] S. Novack, J. Hummel, and A. Nicolau. A Simple Mechanism for Improving the Accuracy and Efficiency of Instruction-Level Disambiguation. In *Languages and Compilers for Parallel Computing (August 1995)*, Springer-Verlag LCNS volume 1033, 1995.

[35] J. Hummel, L. Hendren, and A. Nicolau. A Framework for Data Dependence Testing in the Presence of Pointers. In *23rd Annual International Conference on Parallel Processing*, 1994.

[36] *Architectures and Compilers for Embedded Systems (ACES) Laboratory.* http://www.cecs.uci.edu/~aces/.

[37] *PROMIS Project Home Page.* http://www.cecs.uci.edu/~aces/promis.html.

[38] P. Grun, F. Balasa, N. Dutt. Memory Size Estimation for Multimedia Applications. In *Proc. CODES/CASHE*, 1998.

[39] P. Grun, A. Halambi, N. Dutt, and A. Nicolau. RTGEN: An algorithm for automatic generation of reservation tables from architectural descriptions. In *Proc. ISSS*, 1999.

[40] S.Wuytack, F.Catthoor, H.De Man, "Transforming Set Data Types to Power Optimal Data Structures", *Proc. IEEE Intnl. Workshop on Low Power Design*, Laguna Beach CA, pp.51-56, April 1995.

[41] S.Wuytack, J.L.da Silva, F.Catthoor, G.De Jong, C.Ykman, "Memory management for embedded network applications", *IEEE Trans. on Comp.-aided Design*, Vol.CAD-18, No.5, pp.533-544, May 1999.

[42] J.L.da Silva Jr, C.Ykman-Couvreur, M.Miranda, K.Croes, S.Wuytack, G.de Jong, F.Catthoor, D.Verkest, P.Six, H.De Man, "Efficient System Exploration and Synthesis of Applications with Dynamic Data Storage and Intensive Data Transfer", *Proc. 35th ACM/IEEE Design Automation Conf.*, San Francisco CA, pp.76-81, June 1998.

[43] J.L.da Silva Jr, F.Catthoor, D.Verkest, H.De Man, "Power Exploration for Dynamic Data Types through Virtual Memory Management Refinement", *Proc. IEEE Intnl. Symp. on Low Power Design*, Monterey CA, pp.311-316, Aug. 1998.

[44] J.L.da Silva Jr, F.Catthoor, D.Verkest, H.De Man, "Trading-off Power versus Area through a Parameterizable Model for Virtual Memory Management", *IEEE Alessandro Volta Memorial Intnl. Wsh. on Low Power Design (VOLTA)*, Como, Italy, pp.34-42, March 1999.