A Novel Paradigm of Parallel Computation and its Use to Implement Simple High-Performance Hardware

R. W. Hartenstein, A. G. Hirschbiel, K. Schmidt, M. Weber

Universitaet Kaiserslautern, F.B. Informatik, Bau 12,
Postfach 3049, D - 6750 Kaiserslautern, F. R. G.
phone: (+49-631) 205-2606 or: (+49-7251) 3575

Abstract. *This paper introduces a novel (non-von Neumann) paradigm of parallel computation supporting a much more efficient implementation of parallel algorithms. Acceleration factors of up to more than 2000 have been obtained experimentally on the MoM architecture for a number of important applications - although using a hardware being more simple than that of a single RISC microprocessor. The machine organization and the most important hardware features of xputers are briefly introduced. The programming paradigm and its flexibility is illustrated by simple DSP and image processing examples.*

Key Words: data-driven, ultra micro parallelism, sparse control, fine granularity scheduling, control overhead, high level synthesis.

1. Introduction

For a number of real-time applications extremely high throughput (up to several kiloMIPS) is needed at very low hardware cost. For at least another decade this mostly will be possible only with dedicated hardware, but not with programmable von-Neumann-type universal hardware. Even technologically advanced processors very often will be still too slow and/or to expensive. Also parallel or concurrent computers do not meet the requirements, or, are by far too expensive. Their sustained average performance is by orders of magnitude lower, than the peak rate. The reason is, that communication mechanisms offered by this hardware are not sufficiently powerful and/or too inflexible: the hardware is compiler-hostile, since most of the dense data dependencies of parallel algorithms cannot be mapped onto it. Next section gives more details.

1.1 Implementing Parallel Algorithms on Contemporary Hardware

Communication mechanisms within concurrent computer systems are extremely hostile to optimizing compilers. Also vector machines have fundamental performance bottlenecks [33][35] and their sustained average performance is by several orders of magnitude lower, than their peak rate [15, 33], even when creative coding techniques help the compiler [34]. VLIW (Very Long Instruction Word) architectures [11, 7] are much more optimizer-friendly by lower level of parallelism (at instruction level) [4, 27, 14] and relatively good optimization results have been reported for systolizable algorithms [4], but only for algorithms with only locally regular data dependencies (systolic algorithms or systolizable algorithms). VLIW architectures still have substantial drawbacks.

Also data flow machines are optimizer-hostile, since indeterministic operation does not permit compile-time optimization. Data flow machines throughput is also affected by other drawbacks: several new kinds of bottlenecks have been introduced. Code causes an enormous addressing overhead and data accessing conflicts [13].

A higher degree of parallelism may be achieved by Application-specific Array Processors (ASAPs). Even ASAPs have substantial draw-backs: extensive I/O overhead is caused by scrambling and unscrambling of data streams, expensive design of special hardware is required. A more important drawback is, that only algorithms with locally regular data dependencies (systolic or systolizable algorithms, see [32] and others) are supported. This drawback also holds for parallel computer architectures for systolic

application area	algorithm example		acceleration factor	r-ALU size**	r-ALU size *
VLSI design automation	CMOS design rule check (pattern matching)		>2000	45	4
	electrical rules check		>300	5	0.5
	Lee routing		>160	5	0.5
digital signal processing and image processing	vector-matrix multiplication	• single scan cache	>9	7	0.5
		• double scan cache	150	7	0.5
	two-dimensional filtering (3 by 3 window)		>300	< 0.5	0.02

*) number (or fraction) of PLD chip(s) needed with Plessey ERA 60400, **) with Altera MAX EPM 5128

Figure 1: Acceleration factors by single processor MoM xputer - compared to a VAX-11/750.

emulation [37].

1.2 Data-driven Ultra Micro Parallelism

A more detailed comparative analysis has been published elsewhere [26]. We strongly believe in the following fundamental requirements to avoid most of these problems, to obtain sufficiently optimizer-friendly hardware, to avoid most of the massive overhead caused (within software and hardware) by von Neumann principles. To obtain sufficiently flexible communication mechanisms parallelism should be implemented at a level much lower than usual: (1) below instruction level (ultra micro parallelism). Optimization (parallelization) should be based on very fine granularity resource allocation and scheduling (2) - determined at compile time to a much larger extent than usual (3). The paradigm should be deterministically data-driven (4).

The non-von Neumann xputer paradigm being introduced in this paper is an approach into this direction. Its novel processor organization (and its hardware implementation) supports parallel algorithms in a drastically more efficient way by avoiding overhead via residual control (which we also call sparse control), very fine granularity intra-ALU parallelism (which we call ultra micro parallelism) and deterministic data scan cache use. It is based on data sequencing (in contrast to the control flow sequencing paradigm of von Neumann machines), so that also optimization methods based on data-dependencies are efficiently supported. We are not aware of any other development of programmable hardware machine principles, which yields such a good utilization of hardware. In many DSP and image processing applications xputer use can avoid the need for special DSP processors or expensive image processing computers.

1.3 Technology-independent Cost / Performance Evaluation

Encouraging performance results (fig. 1) have been obtained experimentally on the MoM xputer architecture at Kaiserslautern [25, 6] showing, that in several important applications a single xputer processor may even outperform larger parallel computer systems. A computer-to-xputer performance comparison seems to be the best possible way to evaluate the merits of these results. Since an xputer does not have a hardwired "instruction set", it does not make sense to use MIPS, normally used for computer-to-computer comparison - to indicate the progress of technology and physical design, rather than the efficiency of machine principles. But also other computational devices benefit from progress of technology. That's why we prefer the technology-independent measure of acceleration factor obtained experimentally (fig. 1) from two equivalent implementations of the same algorithm (compare fig. 15): one from a computer (VAX-11/

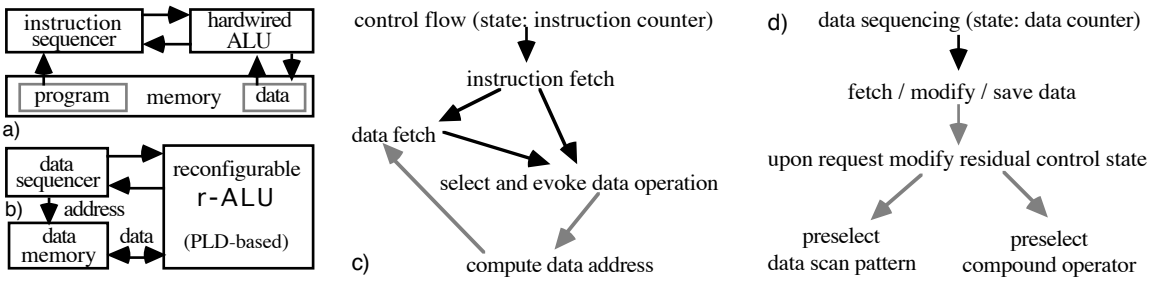


Figure 2: Computers vs.xputers. basic structure: a) computers, b) xputers; causality in c) computers, d) xputers.

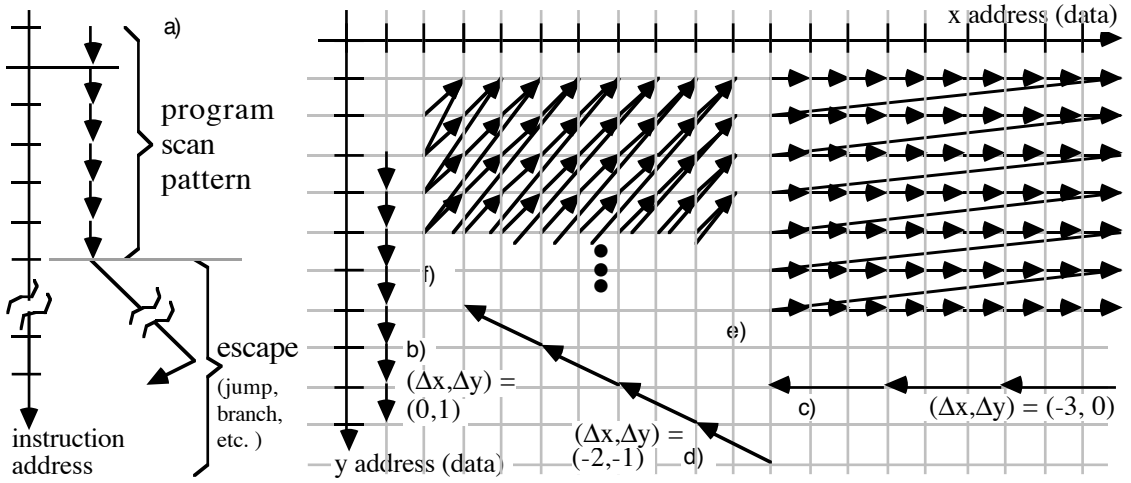


Figure 3: Scan pattern(s): a) von Neumann control "scan" pattern, b through f) some xputer scan pattern examples

750) and one from the technologically comparable MoM xputer [25]). We also have found out, that for computed acceleration factor estimates a good model is obtained from comparing the total number or duration of primary memory cycles.

Another important measure is the r-ALU size depending on the computation needed for a particular application and on the number of applications resident simultaneously. A rough measure of expense is the number of PLDs needed of a particular type. PLDs (programmable logic devices) are available commercially from a billion US-dollar world market. Fig. 1 shows some such expense figures obtained experimentally on the MoM [25] xputer with code from an optimizing compiler having been implemented and tested at Kaiserslautern [6].

2. Xputer Machine Organization

For clarification xputers are compared to computers. The ALU of computers is a very narrow bandwidth device: it can carry out only a single simple operation at a time. Xputers, however, use a PLD-based r-ALU ([25] fig. 2 b), reconfigurable such, that several highly parallel data paths form also powerful compound operators, which need only a few nanoseconds per execution, due to highly parallel dedicated intra-chip read / modify / write interconnect between register files (scan caches) and r-ALU (fig. 4 a). The r-ALU is configured only during loading, not at run time, so that PLD set-up slowness does not affect performance: dedicated wires are fast and avoid buses' multiplexing overhead [5]. Although 2 ns gate delay PLDs have available

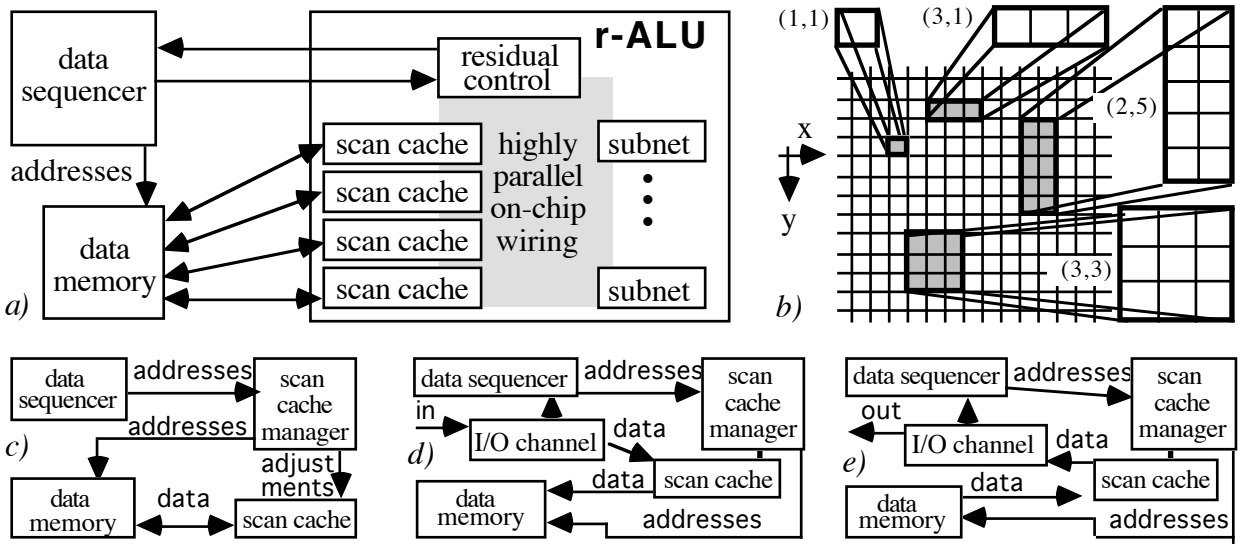


Figure 4: The MoM xputer architecture: a) basic hardware structure, b) scan cache size adjustment examples c) details of data memory interface, d) scan cache input sequencing, e) scan cache output sequencing.

commercially, PLDs might be slower than traditional ALU technologies. This is more than compensated by its micro parallelism and other xputer features.

In computers control flow is the primary activator (fig. 2 c): the instruction counter is the control state register. The rate of control flow is very high (control flow overhead): for each single data manipulation action at least one preceding control action is needed, which requires at least one memory cycle each. If no emit address nor emit data is used, additional control flow and even data operations are needed for address computation (addressing overhead).

Driven by the data sequencer, a hardwired data address generator (fig. 2 b, instead of computers' instruction sequencer: fig. 2a), xputers are data-driven (fig. 2d). Let's illustrate the role of this data sequencer by the MoM xputer architecture example featuring a 2-dimensional data address space (fig. 3 b - f). The MoM has 4 register files which we call scan windows or scan caches (fig. 4 a), because each such cache operates like a size-adjustable window on the data memory (fig. 4 b). The hardwired data sequencer provides a repertory of generic data address sequences without any addressing overhead. Such an address sequence makes a scan cache move through data memory space, step by step, scanning a predefined 1- or 2-dimensional segment of primary memory space along a path, which we call a scan pattern.

Figures 3 b - d show examples of linear scan patterns (step width / direction indicated by vector (Δx,Δy): a relative jump). Fig. 3 e shows a video scan pattern useful e. g. in 2-D filtering. Other scan pattern examples (also see fig. 8) are: reflect, shift, shuffle, butterfly etc. Also special scan patterns to emulate systolic arrays, as well as data-dependent scan patterns (e. g. for image preprocessing etc.), are available in hardwired form (i. e. free of any overhead) [6, 25]. For more about particular scan patterns and their applications see [25, 6]. For a textual scan pattern language see [28, 36]. For stack-based hardware support of nested scan patterns see [16].

Looking back at computers: their control flow has only a single "scan pattern" (fig. 3 a, compare 3 b) scanning instructions one by one (as long as no branch nor jump is encountered, which we consider to be an escape from the scan). In contrast to those of xputers this scan pattern is not free of overhead: each step requires its own instruction fetch. Each instruction fetch requires a memory access cycle. This especially makes iterative operations inefficient, since the same instruction is fetched again and again. Looping instructions cause additional control overhead and thus additional memory access cycles. From this point of view it is obvious, that the computer paradigm is extremely overhead-prone, whereas the xputer paradigm strongly tends to avoid most kinds of overhead.

3. The Data Sequencing Paradigm

For high level programming of xputers we use a simple model which we call data sequencing paradigm, and, which will be illustrated here by 2 simple algorithm examples. The first example (a systolic algorithm: fig. 5) is not a good one to demonstrate the merits of xputers over vector machines.

It has been selected for easy illustration of the data sequencing paradigm. Fig. 5 a shows it textually and fig 5 b its data dependence graph (DG). From this DG the compiler derives: a data map (fig. 5 c + 6 c), from this map (s. partial data map in fig 6 a) a cache format spec (middle of fig. 6 b) and r-ALU subnet spec including wiring (left side in fig. 6 b, derived from a single iteration in fig. 6 a), and finally a scan pattern (arrows in fig. 6 c). At each step of a scan the r-ALU subnet currently activated applies a read / modify / write cycle to the cache(s) currently active. In our example 8 steps (width = 2) are carried out (fig. 6 c shows initial and final cache locations).

Fine Granularity Scheduling. This first example has illustrated the task of the innovative kind of compilers needed for xputer [6, 36]: a kind of fine granularity scheduling (or: ultra micro scheduling) of data words, caches and rALU subnets. This is fundamentally different from sequentially piling up sequential code like conventional compilers do it for computers. Later in a section on xputer high performance features a more detailed impression on this scheduling task will be given.

3.1 Organization of Residual Control

At the end of the above data sequence example the cache finds a tagged control word (TCW: fig. 6 c) which then is decoded (right side in fig. 6 b) to change the state of the residual control logic (fig.4a) to select further actions of the xputer. This sparse TCW insertion into data maps we call sparse control. Note, that the control state changes

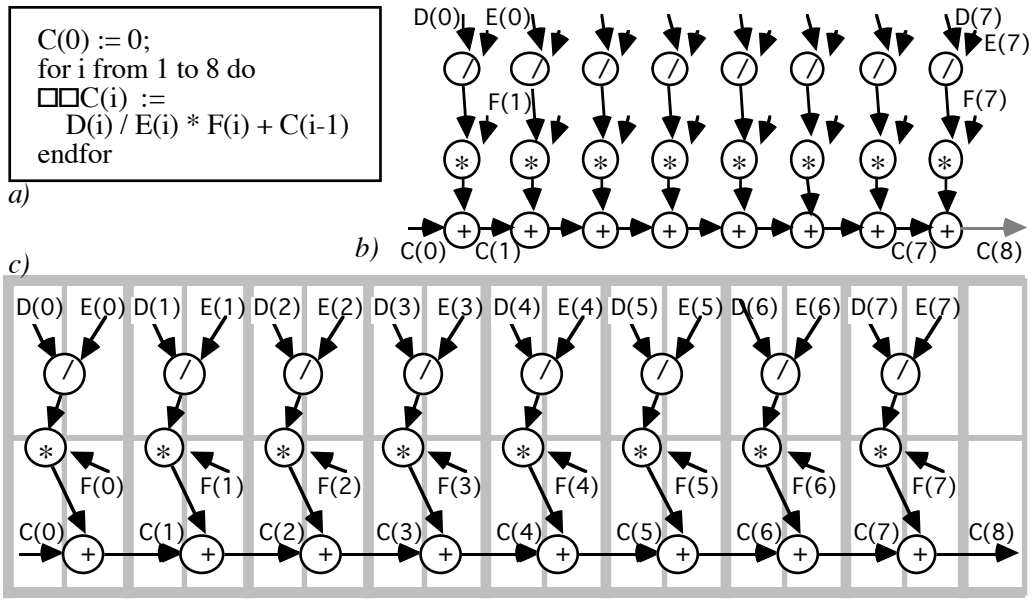


Figure 5: Systolic algorithm example: a) procedural, b) data dependency graph (DG), c) derive data map from DG

only after many data operations (driven by the data sequencer). That's why we use the term residual control or sparse control for this philosophy. Note, that xputer operation is data-driven so that TCWs may be encountered only from within a data sequence. A TCW decoder is defined at compile time and configured as a subnet within the r-ALU. Fig. 7 a illustrates distribution of the residual control state between a scan state register (holding scan pattern select code and parameters), an ALU state register (holding subnet select code) and residual control state register.

We define, that only conditional branching, operator select and scan pattern select, but not data addressing, are control actions. Thus during a scan there is no control action: the data counter is not a state register. But escape from a scan is a control action (like in computers, see fig. 3 a). Escapes are (fig. 7 a): normal escape (by end of scan flag from data sequencer), delimiter escape (on TCW encounter), off-limits escape (address exceeds memory segment limits), conditional branch escape (by decision data from r-ALU), and, event escape (by external event flag). Upon

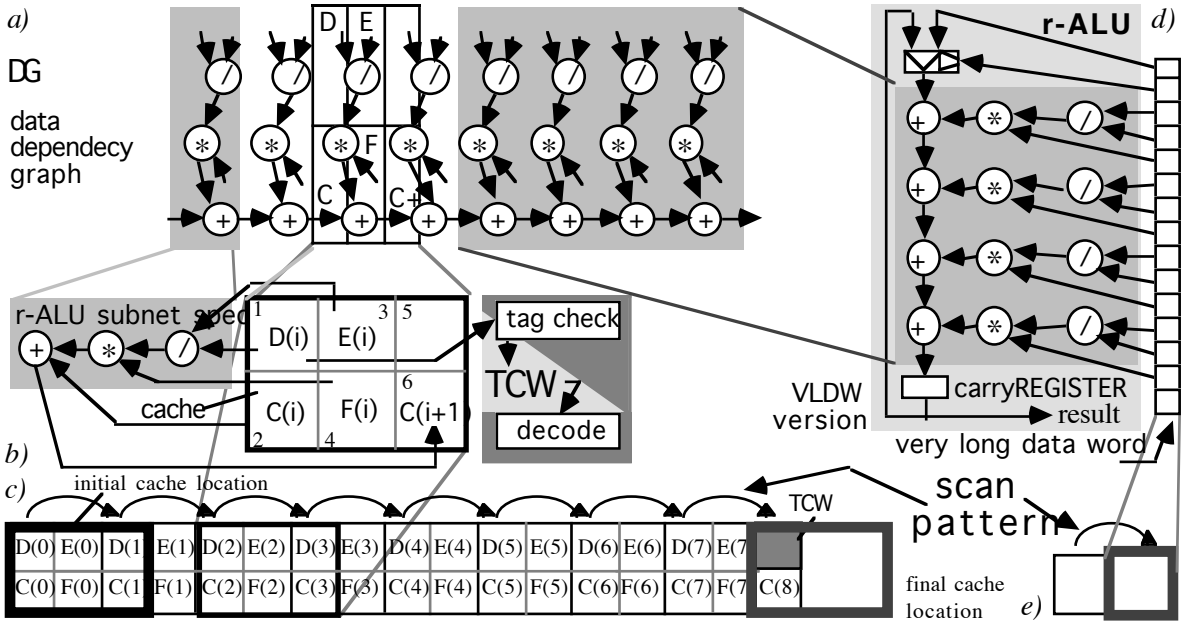
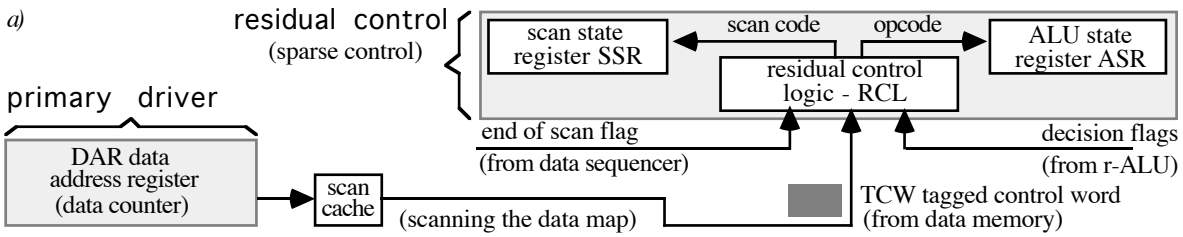


Figure 6: Mapping a parallel algorithm (fig. 5) onto xputer hardware: a) dependency graph, b) a r-ALU subnet spec + cache size spec (3 by 2 words), c) its data map and scan pattern: 8 steps of width=2; d) VLDW version r-ALU subnet spec (scan cache size: 1 by 1), e) its data map and scan pattern: single step of width=1.



b) *the role of:*	(von Neumann) computers	xputers	data flow machines
data flow	derived (by pointers) from control flow	primary activator scheduled at compile time	primary activator determined at run time
control flow	primary activator	derived - **only when needed:** sparse control - residual control	(not existing explicitely)

Figure 7: Xputer control principles; a) state distribution of residual control , b) comparison to other devices.

off-limits escape, branch escape, or event escape a remote control word (RCW) or remote address word (RAW) is fetched from a remote memory segment via an escape cache. A second decision mechanism (implicit branching, because residual control state is not affected) is activated only within data-dependent scans (i. e. without escape: curve following etc. [6, 25]). Such a data-dependent scan may be exited by conditional branch escape or off-limits escape.

To achieve xputer universality scan patterns also non-generic scans and individual data accessing are needed, implemented by list-directed scan: next data address is read from a TAW (tagged address word) within the data map or from a RAW (in case of an escape). This list mode can be entered directly during a scan upon TAW encounter. If no TCW is found, a TAW does not activate residual control. Reading addresses from primary memory means addressing overhead, so that list-driven sequencing is slower than hardwired scan patterns. But also in this mode of operation the xputer paradigm is still superior to the computer paradigm.

3.2 I/O Data Sequencing

Xputer I/O is simple: the scan-cache-based data sequencing hardware (more details in fig. 4 c) is linked to an I/O channel (fig. 4 d/e), which is more powerful than DMA known from computers. The data streaming in are not just downloaded into a memory segment. Via a suitable scan pattern selection, along with proper scan cache adjustment, the data sequencer sets up a structured data map already during input operation. Also during output (fig. 4 e) data may be picked (by the data sequencer) from memory in a structured way.

3.3 Highly Flexible Cost / Performance Ratio

Xputer word lengths are compiler-defined: data path, cache, and control words. Thus extensible xputer architectures are feasible, upgradable by inserting more PLDs into free r-ALU sockets and more boards into free memory slots. E. g. it is easy to design a VWL memory (Variable Word Length Memory), where data word length could be changed under software control to support VLDW (very large data word) strategies for more parallelism.

Figure 6 d/e shows the VLDW version of fig. 6 a-c. Instead of only a single iteration of the DG this time the r-ALU subnet spec is derived from 4 iterations: the new version of the compound operator (fig. 6 d) is 4 times as powerful. Its use requires a vertical cache format (VLDW cache at right side in fig. 6 d), where a single word holds 14 operands.The scan pattern is very short, so that the 1-by-1 cache visits only 2 locations (fig. 6 e). In total the number of primary memory semi-cycles has been reduced from 41 to 2, so that a speed-up by about a factor of 20 has been obtained. This illustrates the extremely high flexibility of the xputer paradigm with respect to cost/performance trade-off.

3.4 Data Address Generator Hardware

This section illustrates the address generator operation. Fig. 9 a shows the structure of a stepper unit. The MoM uses a twin stepper for (x, y) addresses, providing a separate twin for each cache. The address stepper generates linear address sequences (A, A+ΔA, A+2*ΔA, ...) between limits B and L (e.g.fig. 3 b,c,d). The B stepper generates linear sequences B = (B0, B0+ΔB, ...) up to limit L and within F...C, to group bursts of A sequences. L stepper operation is

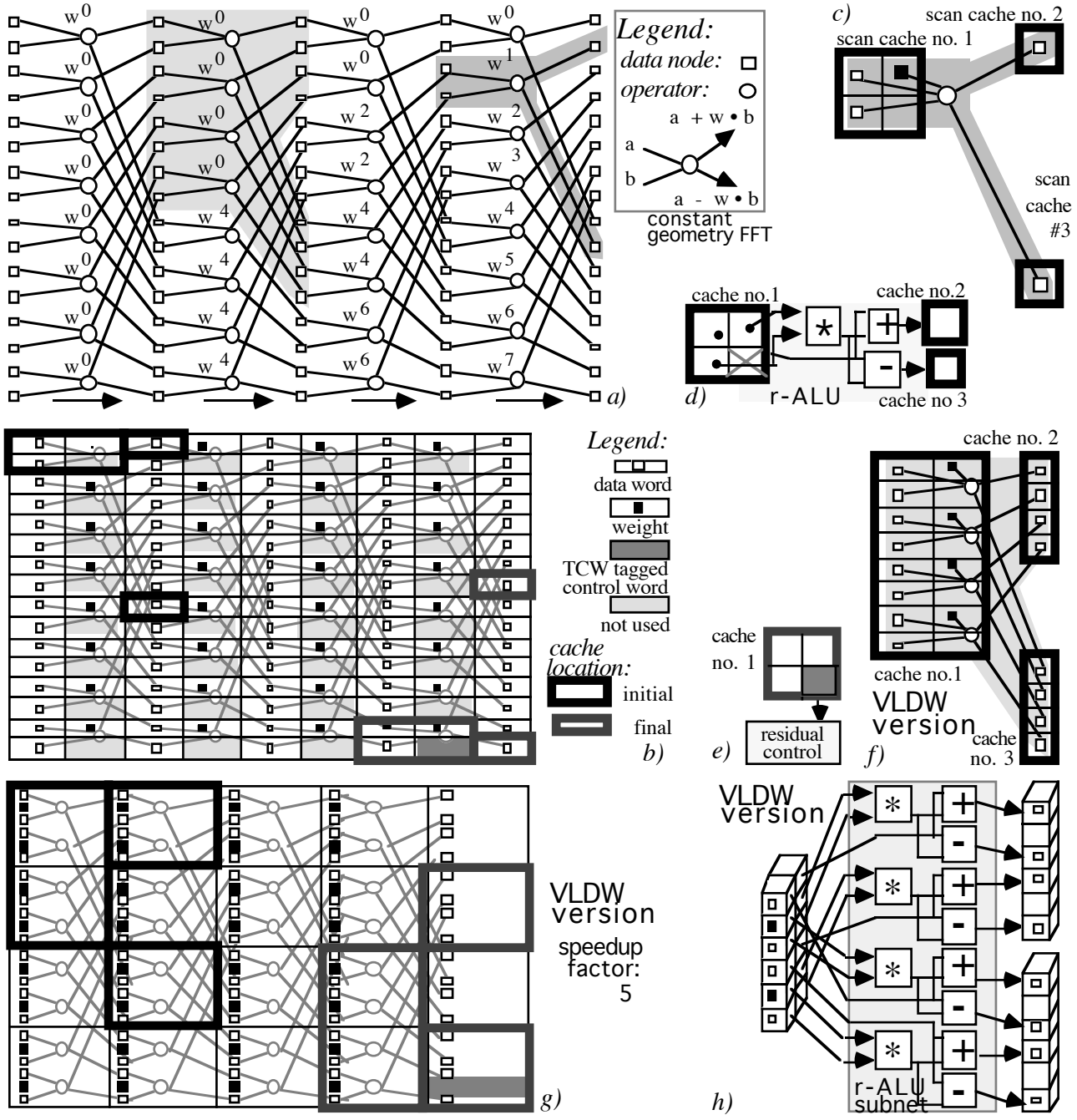


Figure 8: Non-systolizable algorithm example (FFT): a) dependency graph (DG), b) deriving a xputer data map, c) deriving a r-ALU configuration and cache assignment / size adjustment, d) r-ALU compound operator from c, e) detect end of scan pattern, f) deriving a VLDW implementation from fig. a, g) VLDW version data map, h) VLDW version r-ALU subnet specification.

independent of that of the B stepper. Fig. 9 b shows a 2 cache example for a shuffle exchange scan, where 2 steppers cooperate synchronously on the y address only. A simple linear scan is applied to cache no. 1, a 3-by-3 warped scan to cache no. 2, where shuffling is controlled by equidistant B and L slides within boundaries defined by F and C.

4. Non-systolizable Algorithms

The introductory application in fig. 5 / 6 has been a systolic algorithm, being easy to convert into a data sequencing scheme because of the locality of data communication. In digital signal processing, however, also non-systolizable algorithms are very important. In contrast to parallel computer systems and VLSI arrays, xputers smoothly accept also non-systolic data sequencing schemes. Fig 8 shows such an algorithm (a 16 point example constant geometry FFT). Fig. 8 a shows the DG, including also non-local data communication (also see fig. 8 c). Fig. 8 b illustrates the ease of deriving a data map from

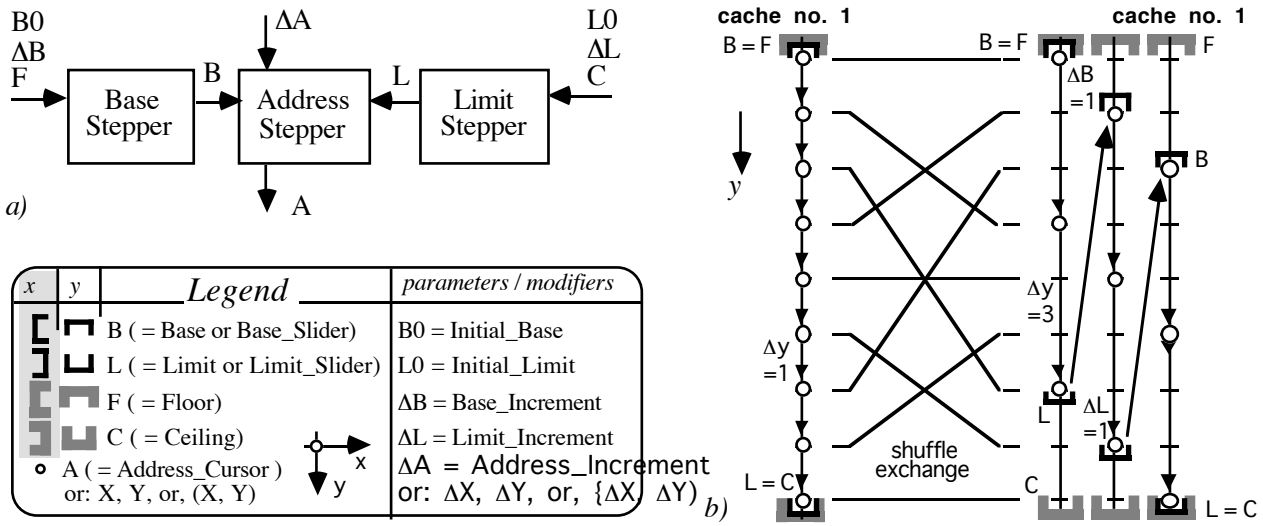


Figure 9: Address Generator: a) Block Structure, b) Snapshots of Shuffle Exchange Addressing Operation

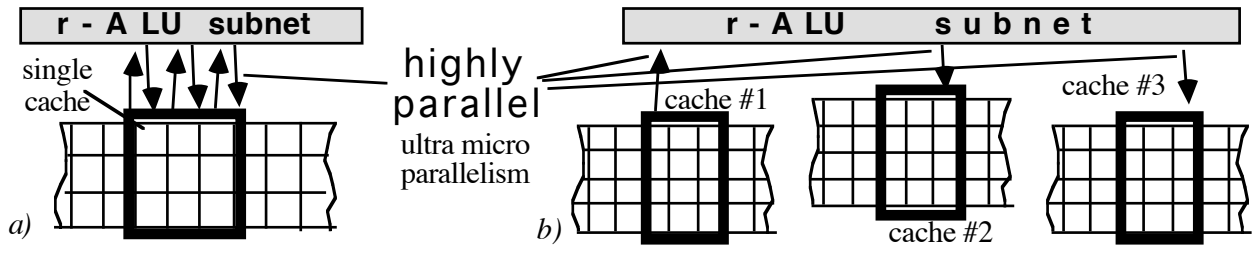
fig. 8 a. Fig. 8 c shows, how the r-ALU subnet spec (fig. 8 d) and a 3 cache configuration are easily derived from fig. 8 a: just take a single iteration (spider-shaped). Fig. 8 b also shows initial and final locations of the 3 caches being scanned in parallel.

Also with non-systolizable algorithms a VLDW strategy may be used. Figures 8 f-h illustrate the VLDW version of the example from fig. 8 a-e. Instead of a single spider (iteration) 4 adjacent "spiders" are picked from the DG (compare fig. 8 f). Fig. 8 g shows the VLDW data map holding 6 operands in a single very long data word. Fig. 8 h shows the more powerful VLDW version r-ALU subnet and its connections to the scan caches. Compared to the above example this VLDW version yields a speed-up factor of about 5. Due to their high performance xputers may replace specialized digital signal processors. Due to their universality xputers may accelerate also any other kind of parallel algorithms. For mass production applications xputers may also be used in stand-alone mode, so that no host is needed which substantially reduces the total chip count.

5. Xputer High Performance Features

Having explained introductory sequencing examples we may obtain deeper insight into xputer performance issues more easily. Xputer performance stems from a number of different phenomena and concepts. Fig. 11 surveys the most important mechanisms contributing to the efficiency of parallel algorithm implementations running on xputers, which will be discussed throughout this chapter. Important roots of xputer efficiency are: the r-ALU's ultra micro parallelism, the data sequencing paradigm, and, the high flexibility of xputer memory interface architecture.

Much wider varieties of optimization strategies than possible with computers can be efficiently mapped onto this innovative methodology. Compound operators' ultra micro parallel-ism reduces memory access by substantially minimizing the number of stored intermediate variables. Often the r-ALU's flexible data path width facilitates better utilization of r-ALU space (e. g. see 2-D filtering example in next



Figuire 10: Scan cache: the xputer's basic communication mechanism. a) systolic example, b) non-systolic example

feature, reducing memory bandwidth requirement	memory access cycles saved by:	comment
compound functions[1] (highly parallel r-ALU data paths)	≥ 1 complex expressions computed combinationally	no storage of intermediate variables
ultra micro parallelism[1] (very fine granularity parallelism)	each on-chip wire connected individually [1]	run time performance not affected [1]
no multiplexing overhead	no buses within CPU	(buses cause overhead [Bus])
data sequencing[2]	sparse control[2]	avoids control overhead
generic scan patterns[3]	avoids addressing overhead	with parallel algorithms
non-generic scan patterns[3]	reduces addressing overhead	with 'glue' software
minimized cache updating[1]	access mode tag-controlled[2]	
high hit rate memory interleaving[3]	by optimized data map[1]	
VLDW - very long data word[2]	by VWL (variable word length) memory[1,2]	xputers don't impose format constraints
xputer programming paradigm	by far less overhead-prone[4]	in all levels of software [3]

1) set-up or adjusted at compile time or loading time
2) hardwired feature
3) experimental results: reasons not yet well understood
4) - than von-Neumann-based organization, or, model, rspectively

Figure 11: Influence factors contributing to Xputer Efficiency - compared to those of Computers

chapter). Dedicated intra-r-ALU interconnect avoids using buses being slow and causing multiplexing overhead [5].

The data sequencing paradigm obviously is by far less over-head-prone, than the von Neumann control flow paradigm. Control flow overhead is almost completely avoided (also no instruction fetch cycles are needed). The above examples have demonstrated, that addressing overhead is substantially reduced not only by hardwired address generator (also see the pattern matching example in next chapter). Not yet all mechanisms of overhead reduction in xputer programs are well understood: we propose basic research also covering overhead mechanisms of the von Neumann paradigm.

Now let's look at memory bandwidth. We may distinguish two kinds of factors: reduced memory bandwidth requirements due to the xputer paradigm, the r-ALU concept, and optimizing compilers (see above), and, providing higher memory bandwidth. Interface flexibility offers an extremely wide variety of strategies (optimum data maps) to meet the bandwidth requirements having been left over, where the xputer scan cache model is an important concept in finding such strategies. Important means are wide memory data paths (VLDW approaches, see above) supported by VWL memories (Variable Word Length memories, see above).

More hardware features having been developed for the MoM xputer [1, 16, 25] support further reduction of memory access time. Special access mode tags per cache word reduce the number of memory semi cycles needed for cache updating. For demonstration let's see the cache con-figuration in fig. 6 b: using a read-only tag for words no.1-4, write-only tag for word no.6, and an ignore tag for word no. 5, reduces the number of semi cycles per scan step from 12 to 5. The MoM cache mechanism also makes possible very high hit rates in interleaved memory access utilization. See example in fig. 6 c: if in a 4-phase interleaving scheme the groups of all C[i], all D[i], all E[i], and of all F[i] would be stored in separate memory banks, the number of semi cycles for cache update (see cache in fig. 6 b) would be further reduced from 5 to 2 (total speed-up factor: 6).

In unit step sequencing of large caches memory bandwidth bottlenecks can be reduced (due to optimizing compiler strategies) by another cache feature reducing repetitive access to memory locations. The MoM 2-D cache hardware also provides a multidirectional shift path, separately for each dimension, such that, for instance for e. g. a 4-by-4 cache in a video scan (see example in fig. 13) the number of semi cycles is reduced from 32 to 8 [25]. By combination of this feature with interleaving the memory access rate may be further reduced to 2 (total speed-up factor: 16). Thus several relatively cheap hardware features supporting optimization may total up another order of magnitude of acceleration.

Let's revisit the implementation of parallel algorithms by a second look at xputer communication mechanisms - from a higher level point of view. The highly parallel dedicated combinational path between cache(s) and r-ALU (fig. 10) is the basic communication mechanism of xputers (also compare fig. 2 a). Single cache use only supports local communication

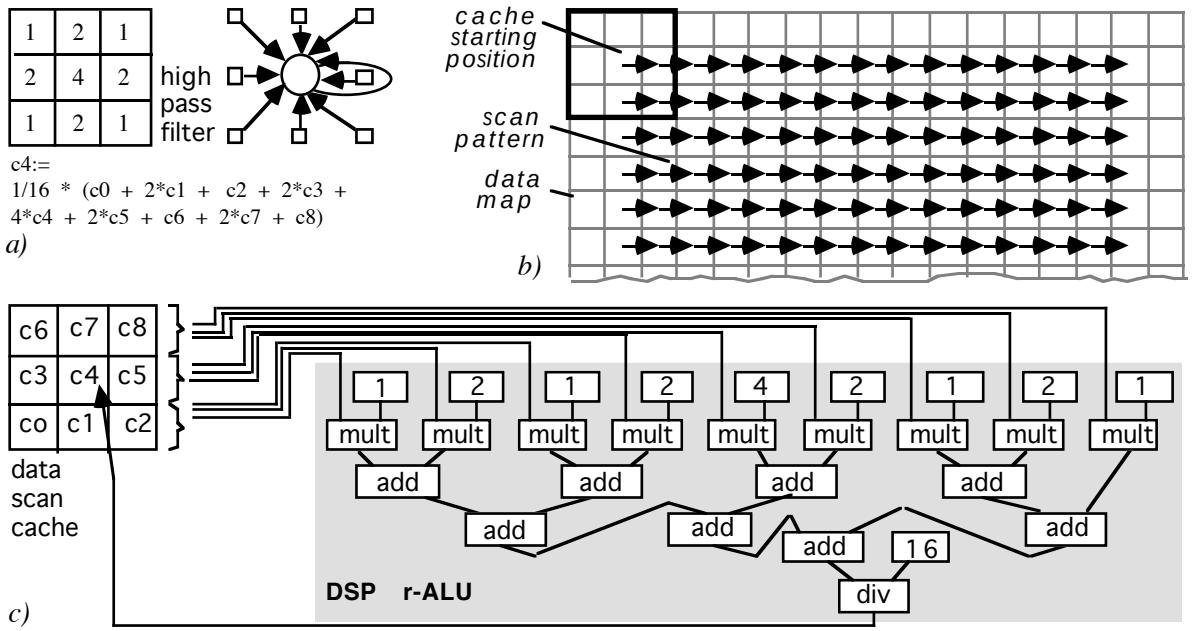


Figure 12: Xputers in image preprocessing: 2-dimensional filtering example. a) filtering expression and cache map of weights, b) scan pattern example, c) r-ALU subnet derived from DG in fig. a.

(systolizable algorithms only), such as e. g. between a data subarray [i] and subarray [i+1] (fig. 10 a) by overlapping cache positioning. Multiple cache use may also support global communication between different data arrays (fig. 10 b) or between distant subarrays. Also comparing acceleration factors in lines no. 4 and 5 within the table in fig. 1 shows, that here multiple-cache solutions tend to be much more efficient. Like cache memories of computers, scan caches in xputers help to reduce performance degradation due to the memory access bottleneck. It is obvious that xputer cache use is fully deterministic, due to a data scheduling strategy being completely compiler-driven.

That's why a much larger variety of optimization strategies may be applied, in contrast to computers permitting only probabilistic strategies which yield only low hit rates. By xputer cache use, however, extraordinarily high hit rates may be achieved, since cache traffic can be scheduled very precisely in detail to the optimum, tailored to any particular sequencing problem. This is because xputer hardware accepts almost any optimized schedule which always provides the right data at the right location at the right time. Thus compilation for xputers is a kind of very high level synthesis, where the number of visits to data locations in memory is minimized. This has similarities to the travelling salesman problem, where space-to-time mapping derived from systolic array synthesis methodology is an important method [18, 21].

6. Xputers in Image Processing

Xputers are especially well suitable for image preprocessing, so that no specialized and much more expensive image processing computers are needed. Due to its universality also other kinds of parallel algorithms may be accelerated by the same xputer, and, in mass product applications stand-alone xputer use substantially reduces the total chip count. In image preprocessing systolizable algorithms (mainly using simple scan patterns, see fig. 3 e, f) and methods using data-dependent scan patterns are dominating. This section illustrates xputer use here by electronics design automation examples having been implemented at Kaiserslautern, where integrated circuit layout uses data structures being quite similar to those, well known from image preprocessing.

6.1 Two-dimensional digital filtering

Fig. 12 shows a 2-D digital filtering example implemented at Kaiserslautern: a systolic algorithm example in image preprocessing. A video scan pattern (fig. 12 b) is used to move a 3-by-3-sized single scan cache, which at each location recomputes the center pixel c4, by an expression shown in fig. 12 a. The cache map in fig. 12 a shows integer weight distribution. The r-ALU subnet (fig. 12 c) is derived from the local DG in fig. 12 a. Although including 18 arithmetic functions this compound function is purely combinational and fits on a small fraction of a

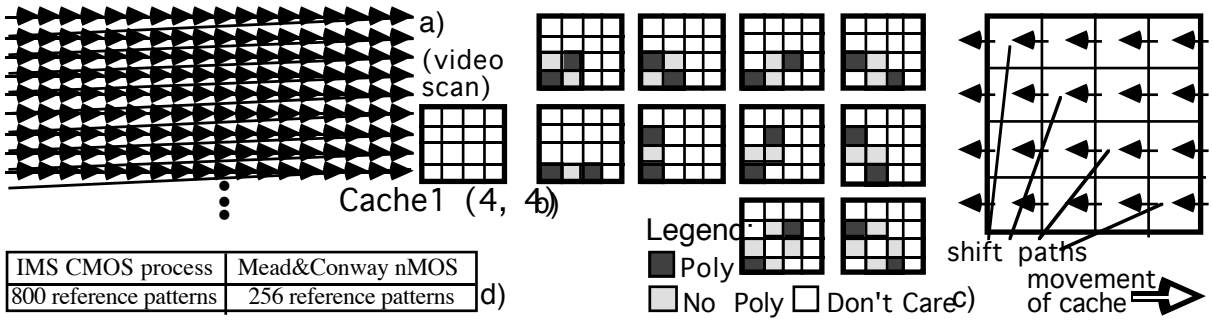


Figure 13: Image preprocessing method used for grid-based design rule check: a) scan pattern and cache size, b) reference pattern examples (poly-to-poly distance ≥ 2), c) on-cache shift paths to minimize memory access
d) number of reference patterns needed

single 5128 chip (last line in fig. 1) - due to the extraordinarily efficient minimization made possible by the high flexibility of xputer r-ALUs. Since xputer data path width is not hardwired a low path width (e. g. 8 bits for the adders in fig. 12) may save PLD space. Multiplication by 1 saves a multiplier entirely. In case of binary coded integers multiplication by 2 or 4 (see fig. 12) may be replaced by a shift left by 1 bit, or, by 2 bits, respectively. All this demonstrates xputers' high acceptance of a wide variety of optimization strategies. Further minimization yields from memory accessing strategies, possible with xputers only. On-cache shift paths (compare fig. 13 c for 4-by-4 example) minimize the number of memory access cycles needed to 1 per word and video scan per line. Combined with suitable memory interleaving this may total up to an order of magnitude (see section 5 for explanations).

6.2 Pattern Matching Applications on Xputers

We use pattern matching examples to illustrate image preprocessing capabilities of xputers, such as applicable also to integrated circuit layout

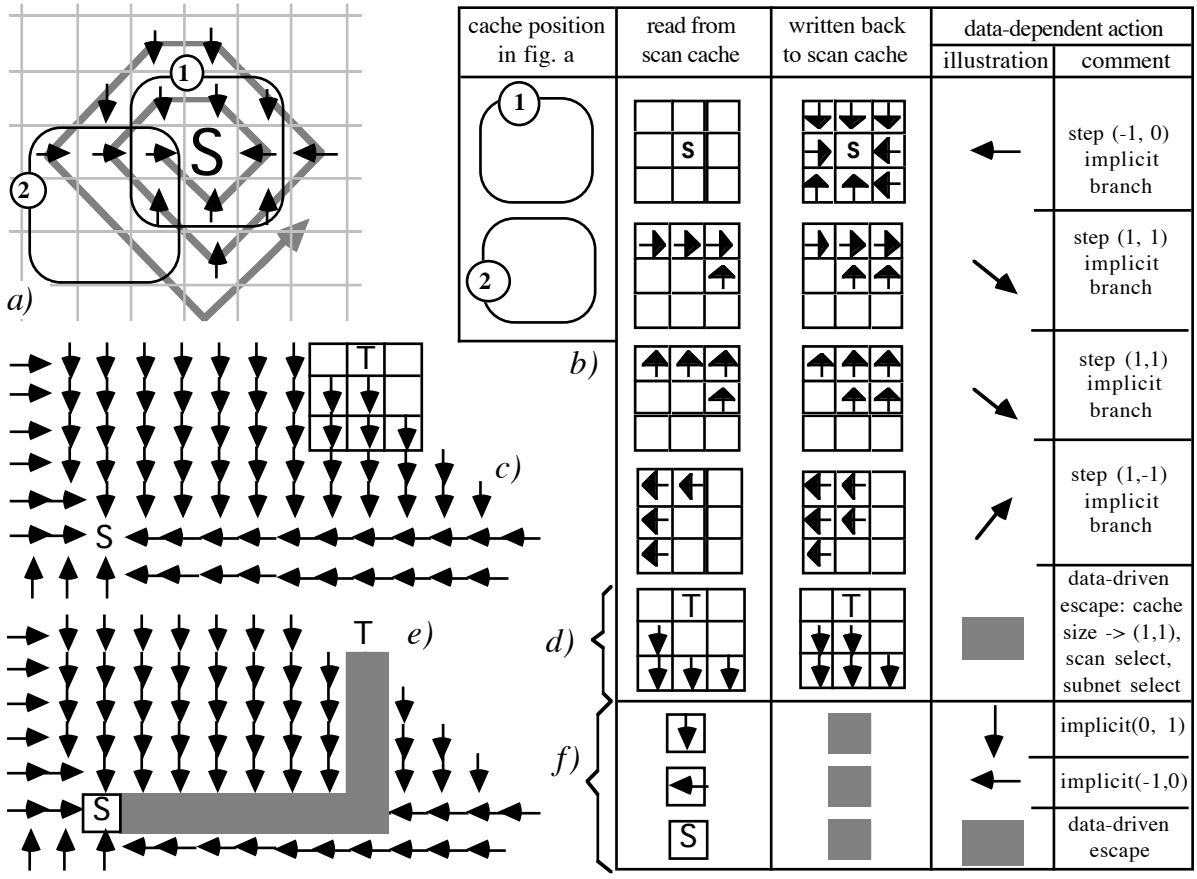


Figure 14: Data-dependent scan pattern examples a) spiral scan pattern driven by feed back from cache input data, b) a few r-ALU function examples from a; c) target T found during spiral scan d) new cache size upon c,
e/f) wire generated by backtracking from T

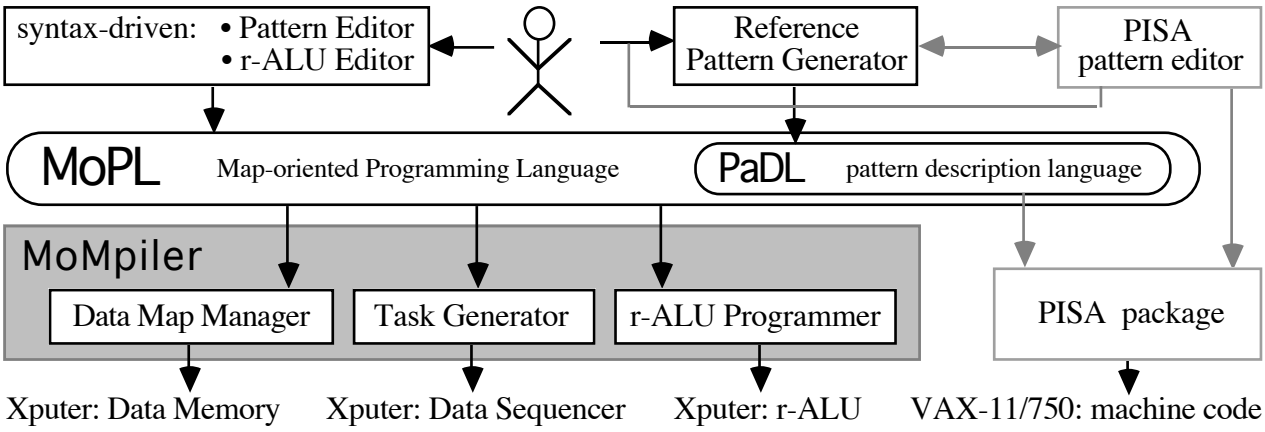


Figure 15: Block structure of the MoM-DE development environment

verification and routing using grid-based design rules [24, 23]. A DRC may be carried out by a finite state machine [12] or combinational logic [31]. Such algorithms run very fast on ASIC hardware which, however, have to be reimplemented for changed design rules and for portation. Due to very large primary memories modern work stations also conventional software implementation is feasible which, however, is very inefficient because of sequential processing of the very large number of reference patterns. But to measure acceleration factors such implementations are needed. The MoM-DE environment with tools like a reference pattern generator and the PISA [31] package facilitate comparative performance measurement by convenient gene-ration of such pattern matching algorithms (fig. 15). In contrast to computers, here the performance of xputers is competitive to ASIC solutions. E. g. for a grid-based design rule check (DRC) the MoM xputer has been programmed such, that a single video scan over the layout is sufficient (fig. 13 a). Substantial acceleration is obtained also for other kinds of grid-based layout processing, such as Lee routing [1, 6], ERC (electrical rules check [2]), compaction [8], fault extraction [9], etc. Reference patterns are configured combinationally into the r-ALU as a single very powerful compound function linked with a video scan sequence within a 2-dimensional bit map memory segment. A single read-modify-write data loop is performed per cache location without using decision data (figure 13 a). Experimental results in grid-based DRC with 4-by-4 cache are acceleration factors of up to 2000 (CMOS design rules [17]).

The extremely high acceleration factor is due to mainly two reasons: all (hundreds of) reference patterns (fig. 13 d) are bundled by a huge compound Boolean operator (massive ultra micro parallelism) and caching completely avoids addressing overhead (an analysis of the VAX version of this algorithm has shown about 90% CPU time for addressing). Also MoM on-cache shift and access mode flag features (fig. 13 c, also see section 5) contribute to the high performance by minimized storage access time.

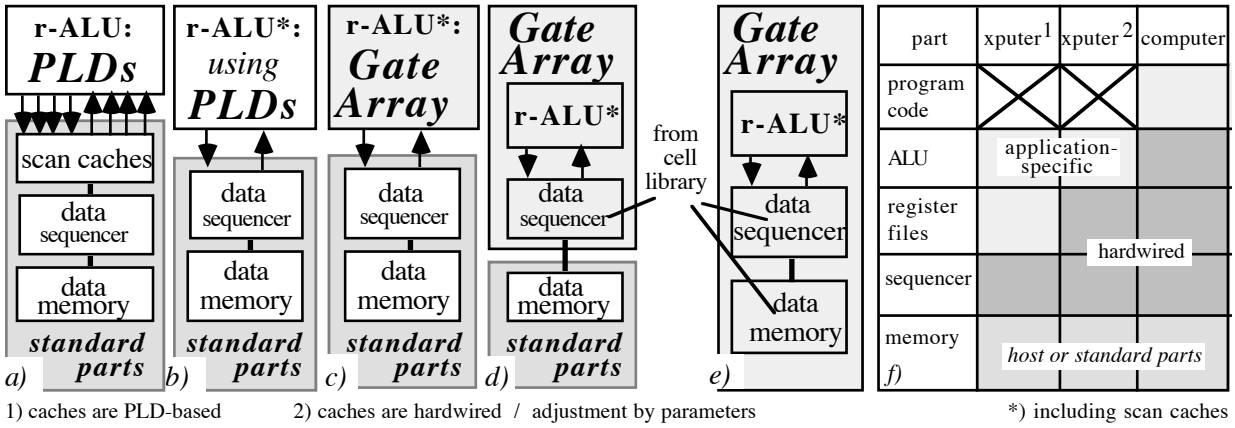


Figure 16: Xputer alternatives: a/b) programmable xputer, c) specialized xputer, d) embedded xputer ASIC (exASIC), e) ex-ASIC with embedded memory, f) compared to computer technology.

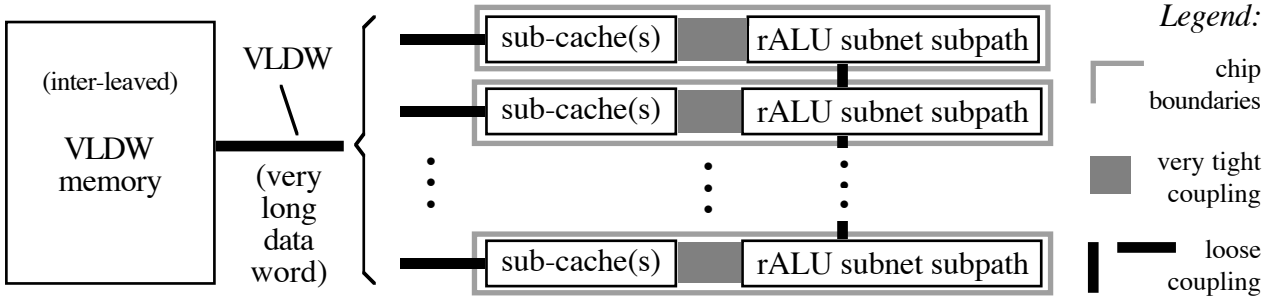


Figure 17: Illustrating partitioning schemes for applications needing very large (multi-chip) r-ALUs

Also the Lee routing algorithm [19, 20, 22] is an image preprocessing example. But this time data-dependent scan patterns are dominating, such as e. g. in curve following: decision data from the r-ALU influence data sequencer operation telling to which nearest neighbour location to go next (fig. 14 b,d,f). In Lee routing the cache (size 3-by-3) first performs a spiral scan around the start cell S, propagating a wavefront around S (fig. 14 a) until the target T is found (fig. 14 c). This is an example of a data-dependent scan pattern (compare fig. 14 b). When T has been found, a (hardwired) data-driven escape is started (fig. 14 d) after switching cache size to 1-by-1 and activating another rALU subnet. Back-tracking from T (fig. 14 f) generates the wire (fig. 14 e). Also this scan pattern is exited by a data-driven escape (conditional branch, see last line in fig. 14 f). Note, that also the data-dependent scan patterns are hardwired, which prevents address computation overhead by direct r-ALU / sequencer interaction. For the Lee algorithm (160 reference patterns) an acceleration factor >160 has been achieved.

6.3 Application Development Support

MoM-DE, the MoM application development environment is running on a host (a μVAX [26]) featuring a self-explanatory syntax-driven editor for a high level language MoPL (Map-oriented Programming Language), roughly a Pascal extension (fig. 15). MoPL sources are accepted by the MoMpiler the "code generator" of which includes a commercial PLD programming tool needed for r-ALU personalization.

MoPL includes a sublanguage PaDL, which efficiently supports pattern matching applications in general. An optimizing reference pattern generator has been implemented [31], which accepts VLSI layout design rules [1]. Fig. 13 b shows an example: 10 reference patterns needed to detect the violation of minimum poly-to-poly separation by 2 lambda. For inclusion of other kinds of pattern matching applications an interactive graphic pattern editor has been implemented [36] for easy editing, modification, inspection and surveying of sets of reference patterns.

7. Embedding and Technology Issues

The most common PLD application is hardware prototyping. But recently an innovative kind of PLD use has been commercialized: ASIC emulation from netlist sources [29, 30]: replacing simulation since being a more efficient way of ASIC verification. In contrast to xputers, however, ASIC emulation does not provide a new design paradigm: the netlist is imported: the result of a separate (conventional) hardware design process. Xputers have a programming paradigm: a very high level model of parallel algorithms. Running an implementation on an xputer is execution - but not emulation. Since for some PLDs also compatible gate arrays are available commercially (e. g. by Plessey): xputer machine code may be directly submitted for fabrication. That's why the xputer paradigm may be considered to be an alternative high level synthesis approach to ASIC design [6] - more precisely: very high level synthesis. ASIC emulation nor simulation is needed, since direct execution is available for design verification. Fig. 16 gives a survey, which illustrates different degrees of embedding customized xputers and compares it to computers.

Partitioning large r-ALUs. To avoid communication bandwidth problems, cache(s) and r-ALU should be on the same chip (fig. 16 b / c). If for "large" applications or for VLDW approaches more than a single PLD chip is needed for the r-ALU, also more expensive inter-chip wiring is needed - in addition to the very efficient intra-chip wiring. This is rather a packaging issue of than a performance issue, since still primary memory access remains the only critical bottleneck. In implementing several such "large

	(von Neumann) computers	xputers
instructions	hardwired, fixed repertory of simple operations	tailored at compile time: powerful compound functors
machine code	sequential: scanned from a program store	combinational: for PLD configuration (at loaeding or compile time)
sequencer	instruction sequencer	data sequencer
parallelism at	process level or (VLIW:) instruction level	data path level - gate level: ultra micro parallelism
compilation techniques:	traditional: from procedural sources	data-driven high level + logic synthesis from algorithm specification
data format	dependant of instruction format	highly flexible: variable word length memory is feasible

Figure 18: Computer features versus Xputer features: Summary and Conclusions

algorithms" we have experienced, that we could always find heuristically a clever partitioning scheme, e. g. by slicing caches into multi-bit slices and distributing the compound operator such, that only loose coupling is required between chips (fig. 17).

8. Conclusions

With xputers an innovative computational machine paradigm has been introduced and implemented which achieves for parallel algorithms (also non-systolizable ones) drastically much better performance and hardware utilization and drastically more (compiler-) optimizer-friendliness than the von Neumann paradigm (comparative summary: fig. 18). Acceleration factors up to more than 2000 have been obtained experimentally with a simple monoprocessor. For many applications xputers may outperform large parallel computer systems or ASIC solutions. Due to convenient conversion into a gate array the xputer also provides an alternative ASIC design methodology.

Xputers fit well to image preprocessing and digital signal processing, so that often special DSP processors or expensive special image processing computers are not needed. Due to xputer universality also other kinds of parallel algorithms and glue software may run on the same xputer, and, in mass product applications a stand-alone use is possible, which substantially reduces the total chip count (compare fig. 16). For xputer architectures an extremely low amount of specific hardware is needed, not being performance-critical, so that it's easy to keep up with technology.

An exciting new R&D scene has been opened up: immature, thus promising and challenging. Not really a new theory, but a new mix of backgrounds is needed, derived from languages, compilation, algorithms and applications. Not yet all phenomena are well understood which contribute to the high acceleration factors found experimentally. We need a new direction of (very) high level synthesis, a new direction in hardware / software performance evaluation redefining the notion of overhead, and, a data-sequencing-oriented new direction of research in programming languages.

9. Acknowledgements

Early versions of MoM concepts have been developed in the multi university E.I.S. project, having been jointly funded by the German Federal Ministry of Research and Technology and the Siemens-AG, Munich, F.R.G., coordinated by GMD Birlinghoven. We also acknowledge the various kinds of personal support we have received from Elfriede Abel, Herwig Heckl, Gustl Kaesser from GMD and Klaus Woelcken (now with the Commission of the European Communities). We also appreciate valuable ideas from Klaus Singer at eltec GmbH at Mainz, F.R.G. Last but not least we appreciate the contributions of more than 30 of our students.

10. Literature

[1] R. W. Hartenstein, A. G. Hirschbiel, M. Weber: MoM - Map Oriented Machine, in: Chiricozzi, D'Amico: Parallel Processing and Applications, North Holland, Amsterdam / New York 1988.

[2] C. M. Baker, C. J. Terman: A Tool for Verifying Integrated Circuit Designs; Lambda 1st Qu. 1980

[3] A. Ast, et al.: Using Xputers as Inexpensive Universal Accelerators in Digital Signal Processing; Int'l Conf. on New Trends in Signal Processing, Communication and Control, Ankara, Turkey, July 1990, North Holland 1990

[4] J.Fisher, J.Ellis, J. Ruttenberg, A. Nicolau: A Parallel Compiler and a Dumb Machine; Proc. ACM SIGPLAN '84 Symp on Compiler Correctness; SIGPLAN Notices 19,6 (June 1984)

[5] R.W.Hartenstein, G.Koch: The Universal Bus Considered Harm-ful; in: R.Hartenstein, R.Zaks: The Microarchitecture of Computing Systems; North Holland, Amsterdam/ New York 1975;

[6] R. Hartenstein, A. Hirschbiel, M. Weber: MoM - a partly custom-designed architecture compared to standard hardware; Proc. IEEE Comp Euro '89, Hamburg, FRG, IEEE Press, 1989

[7] D. D. Gajski, et al.: CEDAR; COMPCON Spring 1984

[8] D. Boyer, N.Weste: Virtual Grid Compaction using the most recent Layer Algorithms; ICCAD 1983

[9] I. Stamelos et al.: A Multi-Level Test Pattern Generation and Validation Environment; Int'l Test Conf. 1986

[10] T. Mayer: POLU (Problem Oriented Logic Unit), Diplomarbeit, Universität Kaiserslautern, 1989.

[11] J. A. Fisher: Very Long Instruction Word Architectures and the ELI-512; Proceedings 10th ISCA, 1983

[12] R. Eustace, A. Mukopadhyay: Deterministic Finite Automaton Approach to Design Rule Check for VLSI; Proc. DAC 1982

[13] D. D. Gajski, D. A. Padua, D. J. Kuck, R. H. Kuhn: A Second Opinion on Data Flow Machines; Computer, 15, 2 (Febr. 1982)

[14] D.D. Gajski, D. J. Kuck, D. A. Padua: Dependence Driven Computation; Proc. COMPCON Spring 1981, IEEE Press 1981

[15] J. J. Hack: Peak versus Sustained Performance in Highly Concurrent Vector Machines; Computer, Sept. 1986

[16] A. G. Hirschbiel: (Ph. D. thesis), Univ. Kaiserslautern, 1990

[17] G. Zimmer: Lambda Designregeln für das EIS-Projekt; report; IMS Duisburg, F.R.G., 1986.

[18] R. W. Hartenstein, A. G. Hirschbiel, M. Weber: Mapping Systolic Arrays onto the Map-Oriented Machine (MoM), Proc. 3rd Int'l Conf. on Systolic Arrays, Kilarney, Ireland, May 1989.

[19] C. Y. Lee: An Algorithm For Path Connections And Its Applications. IEEE TrC-10 (Sept. 1961)

[20] M. A. Breuer and K. Shamsa: A Hardware Router. In: Journal of Digital Systems, 4, 4, 1981.

[21] R. Hartenstein, K. Lemmert, SYS3 - A CHDL-Based CAD System for the Synthesis of Systolic Architectures, Proceedings IFIP CHDL '89, North Holland, Amsterdam / New York 1989.

[22] I. Velten: Implementierung des Lee-Algorithmus auf der MoM, Dipl. Thesis, Univ. Kaiserslautern, 1987

[23] R. F. Lyon: Simplified Design Rules for VLSI Layout; Lambda, 1st quarter 1981

[24] C. Mead, L. Conway: Introduction to VLSI Systems, Addison-Wesley, 1980.

[25] R. W. Hartenstein, A. G. Hirschbiel, M. Weber: MoM - Map Oriented Machine; in: Ambler et al.: (Prepr. Int'l Worksh. on) Hardware Accelerators, Oxford 1987, Adam Hilger, Bristol 1988.

[26] R. Hartenstein et al.: Xputers: an Open Family of non-von-Neu-mann Architectures; Proc. GI/ITG Conf. on the Architecture of Computing Systems, Munich, 1990; VDE-Verlag Berlin 1990

[27] D. Padua, D. Kuck, D. Lawrie: High Speed Multiprocessors and Compilation Techniques; IEEE TC-29, 9 (Sept 1980)

[28] R. W. Hartenstein, A. G. Hirschbiel, M. Weber: The Machine Paradigm of Xputers: and its Application to Digital Signal Processing Acceleration; ICPP-90 Int'l Conf. on Parallel Processing, 1990; IEEE Press, Wash., D.C., 1990

[29] M. D'Amour, et al.: ASIC Emulation cuts Design Risk; High Performance Systems, Oct. 1989

[30] P. A. Kaufman: Wanted: Tools for Validation, Iteration; Computer Design, Dec. 1989

[31] R. W. Hartenstein, R. Hauck, A. G. Hirschbiel, W. Nebel, M. Weber: PISA - A CAD package and special hardware for pixel oriented layout analysis, Proc. ICCAD 1984, Santa Clara 1984.

[32] S. Y. Kung: VLSI Array Processors; Prentice-Hall, 1988

[33] J. J. Hack: Peak vs. Sustained Performance in Highly Concurrent Vector Machines; Computer, Sept 1989

[34] J. J. Dongarra, E. Eisenstat: Squeezing the Most out of an Algorithm in Cray Fortran; report, ANL/MCS-TM-9, 1983

[35] J. J. Dongarra: A Survey of High Performance Computers; COMPCON Spring 1987

[36] M. Weber: (Ph. D. thesis), Univ. Kaiserslautern, October 1990

[37] M. Annarratone et al.: The Warp Computer: Archi-tecture, Implementation, and Performance; IEEE TC-36(12), (Dec. 1987)