R. W. Hartenstein, K. Schmidt, H. Reinig, M. Weber: A Novel Compilation Technique for a Machine Paradigm Based on Field-Programmable Logic; in Will Moore, Wayne Luk (ed.): FPGAs; Oxford 1991 International Workshop on Field Programmable Logic and Applications, Abingdon EE&CS Books, Abingdon, 1991

# A Novel Compilation Technique for a Machine Paradigm based on Field-Programmable Logic

**Reiner W. Hartenstein, Karin Schmidt, Helmut Reinig and Michael Weber**

Fachbereich Informatik, Universität Kaiserslautern
Postfach 3049, D-6750 Kaiserslautern, Germany

***Abstract***

*This paper introduces an innovative compilation technique which is essential to a novel class of computational devices called Xputers, being by up to several orders of magnitude more efficient than von Neumann paradigm of computers. Xputers areas flexible and as universal as computers. But the central technology platform of flexibility is field-programmable logic (we would prefer the term interconnect-reprogrammable media), rather than the RAM which gives the flexibility of computers. The paper first briefly summarizes the Xputer paradigm as a prerequisite needed to understand the fundamental issues of this new compilation technology.*

## INTRODUCTION

For quite a number of commercially important applications extremely high throughput is needed at very low hardware cost. Very often these goals cannot be met by using the von Neumann paradigm nor by ASIC design. Very often neither parallel computer systems (Hirschbiel 1991,Weber 1990) nor dataflow machines (Gajski et al 1982) meet these goals because of massive parallelization overhead (in addition to von Neumann overhead) and other problems. The Xputer machine paradigm is such an approach. To clearly distinguish this *data-procedural* execution model from the *control-procedural* model of von Neumann computers the term *Xputers* has been coined.

This paper deals with the novel compilation techniques needed for this approach which is mainly based on a novel (*data-procedural*) machine paradigm. For a very large class of commercially important algorithms (with regular data dependencies) the Xputer paradigm is by several orders of magnitude more efficient than the von Neumann paradigm. For unstructured spaghetti-type sources the Xputer paradigm is at least half an order of magnitude more efficient. This paper especially introduces a compilation techniques which is needed to derive executable code for Xputers from high level source programs. Traditional compilers can not be

used, since their code does not run on Xputers. To distinguish such new tools from traditional compilers, we sometimes use the term *xpilers* and *xpilation*. The compilation of programs written for this novel machine has relations to high level synthesis and systolic array synthesis.

The superiority of Xputers has several reasons: (1) Their data-procedural operational principles avoid most kinds of overhead, which are typical to be the von Neumann machine paradigm, (2) Xputers support compiled ultra micro granularity parallelism by a reconfigurable ALU (*rALU*), (3) *a smart memory interface* contributes to further reduction of memory bandwidth requirements, (4) Xputers are highly compiler-friendly by supporting more efficient optimizing compilation techniques, than known from compilers for computers.

## A new technology platform

Commercial exploitation of Xputers has become feasible by the progress and commercial availability of modern field-programmable technology (Freeman 1988). Until recently field-programmable logic and related technologies have been used mainly for hardware prototyping in a way which could be called a kind of microminiaturized breadboarding emulation. But now some researchers have recognized, that on the basis of such a technology platform completely new computational paradigms can be developed, which are not possible on classical technology platforms which we used as the basis of von Neumann world computation. This new platform offers an implementation basis for a new class of programming languages and programming methods. Of course, also new compilation techniques are needed in such a fundamentally different target technology. For technology transfer reasons new cross compilation techniques are also needed to bridge the gap between both classes of computational paradigms.

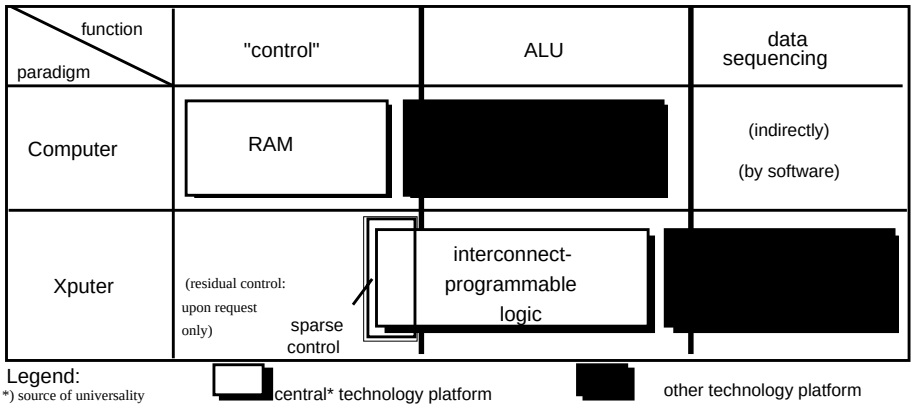



**Figure 1** The difference of the central technology platforms: computers and Xputers

For Xputers a novel compilation technique is needed anyway, since its data-procedural operational principles are totally different from the control-procedural operations for computers. This compiler for an Xputer (xpiler) contains two parts: (1) the mapper and optimizer part to transform a high level language source into the Xputer language, and (2) the synthesizer and code generator part to transform the Xputer language source into Xputer machine code, addressing explicitly the Xputer-specific hardware and the chosen field-programmable media.

Von Neumann principles (RAM-centred) are based on *sequential code*, which is laid down in a RAM (memory) and scanned by an instruction sequencer. That is why the RAM (random access memory) is the central technology platform for computers already for several decades. The same RAM is used for both, data and program, so that compiler and object code may run on the same hardware. This has been important because of high hardware cost in the past. Such RAM use is also a source of the elegance and the universality of the von Neumann machine paradigm: most of the control structure (instructions) is pushed into the RAM sequentially- (at run time it is popped off sequentially). Only a simple instruction sequencer - a small fraction of the “controller” - remains hardwired (see Figure 1). Due to the RAM-based mechanism the machine code needed is highly overhead prone (Hirschbiel 1991). This is the main reason, why the von Neumann machine code usually is massively complex (this is the main source of the well-known “von Neumann bottlenecks”).

*A new technology platform.* Field programmable platforms permit much more efficient machine paradigms. In this context it has been tried out to obtain implementations of the Xputer machine paradigm (Hartenstein et al 1990a, Hartenstein et al 1990b, Hartenstein et al 1990c). Field-programmable logic, or more precisely *interconnect-reprogrammable media* (irM), are programmable by combinational code (Schmidt 1990), quickly electrically alterable, incrementally programmable, with sufficiently high integration density (1991: 60,000 gates/chip) and switching speed.This is just the beginning: much more is expected in the near future from this niche of the semiconductor market. Most of the von Neumann bottlenecks can be avoided by machine principles based on this alternative technology platform.

*Retargeting.* Until recently such media have mainly been used for “microminiaturized breadboarding” for prototyping of relatively simple hardware. But recently irM have become available, which are also suitable for non-trivial, really innovative irM-based methodologies, also by capabilities like e. g. retargeting (Plessey 1990). Due to code compatibility irM personalization code can easily be translated into that of real gate arrays (being faster and of higher integration density). Retargeting also provides an efficient bridge between computational paradigms and ASIC design, where simulation is replaced by execution being orders of magnitude more efficient. Recently massive attention has turned over to the topic of retargeting: more than a dozen of papers have been presented at Design Automation Conference 1991 which more or less deal with retargeting (DAC 1991). This indicates that the high significance of retargeting for the future trends has been widely recognized.

*Xputers’ central technology platform.* For Xputers irM (instead of the RAM) is the central technology platform (see lower row in Figure 1), source of simplicity, universality and elegance of hardware principles. This platform provides a reconfigurable ALU (called *rALU*), which is the base of problem-oriented ultra micro parallelism within compound operators (very low level parallelism at functional level or gate level). That is why Xputer machine code is primarily non-sequential, so that because of the lack of control code a new method of sequencing has to be found: data sequencing. But still a little bit of control is needed, which is called *sparse control*, or, *residual control* (Hirschbiel 1991).The only hardwired part (except external interfaces) by Xputer principles is the data sequencer, a part which is not found in computers (Figure 1). (The hardware of the MoM (Map-oriented Machine), a forerunner of Xputers, has been the first implementation of a irM-based machine paradigm.

*Xputers: much more compiler-friendly.* For this new machine paradigm a new kind of compiler (*xpiler*) is needed. Code to be generated by xpilers is not sequential because of the use of field-programmable media. That's why Xputer operation principles are data-procedural (in contrast to the control-procedural von Neumann hardware) in using a data sequencer, which mainly is hardwired (Figure 1). The Xputer paradigm supports several measures to improve performance, which cannot be provided on a von Neumann basis: (intra-ALU) low level parallelism, avoiding several kinds of overhead (addressing overhead, control overhead, synchronization overhead), thus substantially reducing code size. This and a smart register file, also called *scan cache* and smart memory organization support rich optimization strategies to substantially reduce memory traffic. All this contributes that Xputers offer drastically more performance with less hardware and compilation technique is essential to obtain these benefits.

*Xpiler.* For Xputers a novel compilation technique is needed anyway, since its data-procedural operational principles are totally different from the control-procedural operations for computers. This compiler for an Xputer (*xpiler*) contains two parts: (1) the mapper and optimizer part to transform a high level language source into the Xputer machine code, addressing explicitly the Xputer-specific hardware and the chosen field-programmable media.

For the first part of the xpiler, the established backgrounds of the systolization methods (Fortes et al 1988, Moldovan 1987) known from systolic array generation can be adapted to rearrange the sequence of computations of an algorithm. This compilation is completely based on and driven by data dependence analysis. By time- and space-transformations an optimal data map for the algorithm is automatically generated, that takes care that the right data is at the right memory location at the right time throughout the whole execution of the algorithm. Furthermore optimal scan cache sizes are determined and the Xpiler chooses a scan pattern from the repertory of the data sequencer.

The second part of the compiler generates code for the flexible, reprogrammable platform in the Xputer's data sequencer and especially in its reconfigurable ALU (rALU). This rALU provides several parallel data paths, wherein powerful compound operators are configured depending on the application. To improve comprehensibility the new methodology will be compared to von Neumann computers and parallel computer systems.

In the following section the hardware operation principles of Xputers are introduced and also the Mom-4 experimental Xputer architecture is shortly described. Subsequent sections illustrate the execution mechanism and the novel compilation technique. Some conclusions will finish the paper.

## THE UNDERLYING MACHINE PARADIGM

The main stream of high level control-procedural programming principles and its compilation techniques are heavily influenced by the underlying von Neumann machine paradigm. Most programmers with more or less awareness use as a von-Neumann-like abstract machine model as a guideline for conversion of algorithms into executable form. Without such a model compilation issues would be incomprehensible. Also compilation techniques for Xputers are strongly influenced by the underlying *data-procedural* Xputer machine paradigm.

## Xputer machine principles

This section illustrates the Xputer basic machine principles (Hirschbiel 1991), which are used to implement the control-procedural Xputer programming paradigm, also called *data sequencing paradigm*. First the principle is explained, before the architecture example of the MoM-4 is described, which later will be used as a medium to illustrate Xputer execution mechanisms. The fundamental operational principles of Xputers are based on fully *auto data sequencing* mechanisms with only *sparse control*, so that Xputers are deterministically *data-driven* and most known sources of overhead are avoided. Xputer hardware supports *fine granularity parallelism* (also below instruction set level: at data path or gate level) in such a way that internal communication mechanisms are cheaper and more simple than known from other parallel computer systems.

The ALU of computers is a narrow bandwidth device which can carry out only a single simple operation at a time and uses for this operation only one, two or in some cases three operands at a time. This ALU has a fixed hardwired instruction set which cannot be changed. It is used not only to manipulate the data itself, but also to compute addresses in the data memory space. An instruction sequencer scans over the program memory to get the sequence of instructions to be executed. Often the decoded instructions are control instructions, causing control flow overhead, and do not force any data manipulation. The program memory and the data memory are accessed sequentially. So the following throughput bottlenecks *(von Neumann bottlenecks*) exist: only one simple operation at a time, control flow overhead, addressing overhead, sequential data access, sequential program access. Figure 2b illustrates Xputer architecture principles. The key differences to computers is, that data sequencer and a reconfigurable ALU replace von Neumann's program store with instruction sequencer and the hardwired ALU.

*Field-programmable (reconfigurable) ALU (r-ALU).* Xputers (Figure 2a) have a reconfigurable ALU (rALU), partly using the technology of field-programmable logic. Figure 2a shows an example: the rALU of the MoM-4 Xputer architecture. The four smart register files called scan caches are explained later (lower left side in Figure 2a). The MoM-4 rALU has a repertory of hardwired operator subnets (see lower right side in Figure 2a). Within the field-programmable part of the rALU additional operators needed for a particular application may be compiled by logic synthesis techniques (upper right in Figure 2a) A global interconnect-programmable structure (centre in Figure 2a) is the basis of connecting these operators to form one or more problem-specific compound operators, what will be illustrated later by a simple algorithm implementation example.

*Compound Operators.* The rALU may be configured such a way, that several sets of highly parallel data paths form powerful compound operators which need only a single basic clock cycle to be executed. This rALU uses no fixed instruction set. All compound operators are user-defined. Since their combinational machine code is loaded directly into the rALU, Xputers do not have a program store nor an instruction sequencer. Due to the capability to store "large" operations and functions, the rALU operations can be much more powerful than instructions of even complex instruction set von Neumann processors. The acceleration effect behind the rALU implementation is based on the idea, that the user-defined data operations should be that powerful, that no scanning of instruction sequences in some control or program store is needed between two accesses to data memory. Instead a data sequencer is used which steps through the data

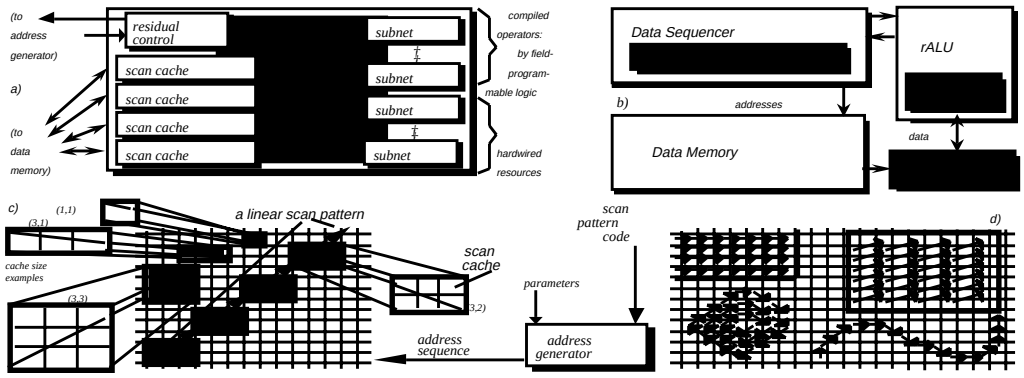



**Figure 2** Basic structures of Xputers and the MoM architecture: a) reconfigurable ALU (rALU) of the MoM, b) basic structure of Xputers, c) MoM cache size examples (left side) and a scan pattern example, d) a few other scan pattern examples.

memory to access the operands via register files called *data scan caches*. Xputers operate data-driven but unlike data flow machines, they feature deterministic principles of operation which are called *data sequencing*.

*RALU design: a challenge to FPL vendors.* The mix of hardwired resources and field-programmable parts has been arranged for area efficient reasons. The objective is to achieve a best compromise between flexibility and performance. Designing a single chip universal rALU of this kind of high performance arithmetic applications is a severe challenge. It is more difficult than designing a microprogrammable ALU (which do not allow compound operators). The problem is to achieve maximum flexibility with a minimum of interconnect area consumption. The microarchitectures of field-programmable IC's available currently from commercial vendors are far away from being suitable for compound operators including arithmetic operators. Also multi-chip solutions are difficult to achieve because of partitioning problems caused by pin-limitations. Only full custom solutions are admissible candidates, where handling of interconnect-programmable circuit elements by the design team is a must.

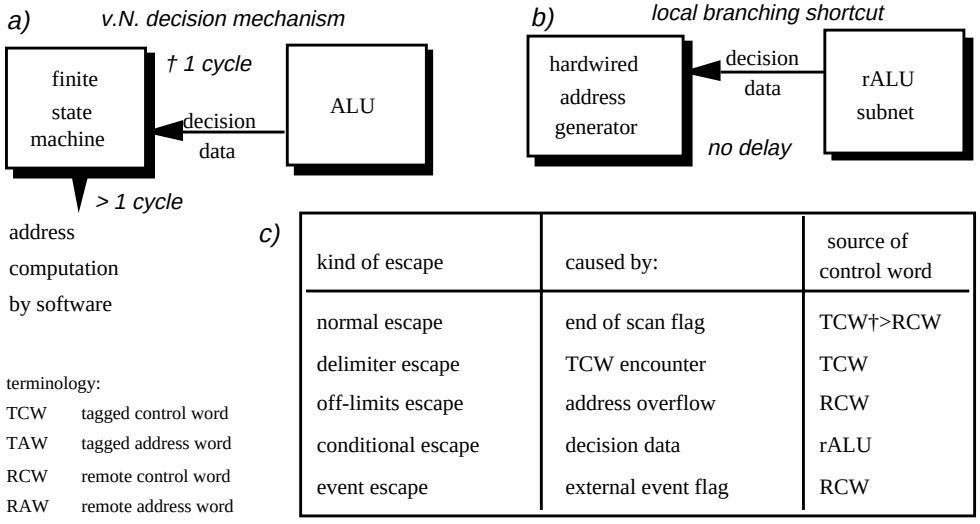
*Data Sequencer.* The data sequencer hardware provides accessing sequences for a controlled scan cache movement over the memory space (Figure 2c). This hardwired data sequencer features a rich and flexible repertory of *scan patterns* making the location of a scan cache travel step by step or jump by jump to follow a particular path through the data memory space. The data sequencer and its role in xputing will be explained next section.

## The MoM Xputer architecture

To use a practical and comprehensible example for illustration of the novel task of the compiler a simple algorithm example implementation on the MoM Xputer architecture will be used. This MoM (Map-oriented Machine) uses some extra features which further support optimization efforts of the compiler: the concept of the scan cache (a smart register file: featuring some hardwired smartness).

*Scan Cache.* Due to the higher flexibility of their basic paradigm Xputers - in contrast to computers - may have a completely different processor-to-memory interface which efficiently supports the exploitation of parallelism within algorithms. An example of such an interface is the smart register file of the MoM architecture, also called *scan cache*. Such a scan cache (Figure 2b and 2c) implements a hardwired window to some adjacent words in the memory space (which is 2-dimensional in case of the MoM). Its format is adjustable at run time (some size examples in Figure 2c). Such a scan window may be placed onto a particular location in memory by a scan cache address generated by the address generator within the data sequencer (lower right side in Figure 2c, also compare Figure 2b).

*Minimizing Memory Access.* Furthermore Xputers avoid most of the von Neumann bottlenecks mentioned above except one: the memory access (data memory only) is still sequential. The number of such sequential accesses, however, can be reduced by optimizing compilation by Xpilers and by special features of the data scan caches or smart interface. To achieve higher throughput rates Xputers may have *multiple scan caches*, i.e. several such data scan caches may connect an Xputer processor to the same primary memory. All caches are connected to the same r-ALU subnet. Each cache can be used to read, write, or read and write data from and to the data memory. Such multiple caches are not allowed to overlap at any stage of their movements in order to avoid memory accessing conflicts and the administration of critical sections.




| kind of escape | caused by: | source of control word |
|---|---|---|
| normal escape | end of scan flag | TCW†>RCW |
| delimiter escape | TCW encounter | TCW |
| off-limits escape | address overflow | RCW |
| conditional escape | decision data | rALU |
| event escape | external event flag | RCW |

**Figure 3** MoM Decision mechanisms: b) branching hardware more efficient than a) von Neumann branching, c) type of escapes from scan patterns

*Data sequencer.* The MoM *data sequencer* hardware provides accessing sequences for the scan cache movement over a 2-dimensional memory space (Figure 2c). This hardwired data sequencer features a rich and flexible repertory of *scan patterns* making the location of an scan cache travel step by step or jump by jump to follow a particular path through the data memory space. Examples of such data scan patterns are single steps as well as longer generic scan sequences, such as *video scan sequences, shuffle sequences, butterfly sequences, trellis sequences* and others (for a very few examples see Figure 2d). The data scan patterns can be adjusted in parameter registers of the data sequencer or they can be evoked by the decision data feedback loop from the r-ALU. With this feedback loop data dependent cache movements can be performed. Also *tagged control words* having been inserted into the data memory map (Figure 3c) can be recognized and decoded within the rALU to derive suitable decision data to select the next scan pattern, which allows to move control code in between the data. By this means a sequence of several data scan patterns can be executed.

*Efficient branching.* The MoM architecture provides branching mechanisms which are more efficient than those known from von Neumann architectures. The von Neumann branching requires one or more control accesses to primary memory, because only after the decision the next control state is known (Figure 3). Depending on the kind of loop exit control code the number of memory accesses may be higher, even if no address computation is involved (which would cost further memory address cycles). For a number of cases such as e.g. nearest neighbour transitions in data memory space (in curve following, for example). The MoM architecture provides more efficient branching, no control action at all is needed, so that no extra memory cycle is activated (Figure 3b). This is achieved by direct manipulation of the least significant data address bits by decision data bits. Because this decision data bypasses the sequencer we call this mechanism a *local branching short-cut*. We will se later, that during Xputer execution of a scan pattern (which may sometimes have the length of thousands of memory cycles) no control accesses are needed. The address generator provides special hardware features for escapes from a scan upon encounter of decision data. Figure 3c lists all types of escapes available, which is also avoid the need for control accesses to primary memory.

## COMPILATION OF XPUTERS (XPILATION)

For Xputers a novel compilation method is needed. The best way to clarify its major differences to conventional compilation techniques targeting von Neumann hardware is to use a simple execution example. By such an example is illustrated what items have to be generated and why they are needed for running a typical algorithm on an Xputer.

### Operation illustration by a simple example

The following example demonstrates the essentials of the Xputer execution mechanism. This is useful to illustrate later the task of the innovative kind of compilers needed for Xputer (Weber 1990): a kind of fine granularity scheduling (or: ultra micro scheduling) of caches and rALU subnets and of data words, ready to be auto-sequenced. Figure 4a shows a textual notation of the algorithm, given in a high level language. Figure 4b shows its graphical representation: a signal flow graph (SFG). Figure 4d illustrates, how this algorithm is executed on the MoM Xputer architecture. The upper side of Figure 4d shows the scan cache (format: 1 by 4 words),

the rALU subnet for the compound operator (also compare Figure 4a and Figure 4b) and the interconnect between cache and subnet. The register inside the rALU subnet saves memory accesses, because the intermediate operands c(0) through c(7) do not need to moved forth and back between the scan cache and the memory. The bottom of Figure 4d shows the data map: how the operands are stored in the memory.

Based on this set-up a very simple execution mechanism may run this algorithm: the scan cache is first placed at the left end of the data area (location shown in Figure 4d). Then it is scanning this memory segment by stepping to the right until it reaches the end of the memory area (shaded rectangle at the right of the data map in Figure 4d). The sequence of arrows below the data map shows the scan pattern having been used: the sequence of column shaped memory locations the scan cache has visited. No further action has to be taken. Note that no control action is needed because the *auto-xfer* mode has been used. This means, that whenever the scan cache is placed somewhere in the memory space, two things are carried out automatically (i.e. without having been called explicitly): the movement of data between scan cache and memory (*auto-xfer* mode), as well as the application of the active rALU subnet to the variables held by the scan cache (what we call *auto-apply* mode). At each such step of a scan via auto-apply and auto-xfer actions, the rALU subnet currently selected, operates and applies a selective read / modify / write cycle to the cache(s) currently active. Note that by access mode tags only a minimum of memory semi cycles is carried out.: read only tags for all 4 words by this example. In our example 8 steps (x width=1, y with = 0) are carried out (Figure 4d shows initial and final cache locations).

At the end of the above data sequence example the cache finds a *tagged control word* (TCW) which then is decoded (right side of the map in Figure 4d) to change the state of the *residual control logic* to select further actions of the Xputer. This sparse TCW insertion into data maps we call *sparse control*. Note that the control state changes occur only after many data operations (driven by the data sequencer). That is why we use the term *residual control* or *sparse control* for this philosophy. This is fundamentally different from the piling up sequential code like conventional compilers do it for computers.

## The compilation technique

To combine the advantages of a language approach and a compiler approach we allow the user on one side to write programs in a Xputer specific language which fully exploit the hardware resources and on the other hand he is allowed to run the old programs he has developed for a conventional von Neumann machine in a common high level language. So two major tasks have to be done: (1) to develop an Xputer language which is easy enough to learn, but which also is rich enough to explicitly exploit the hardware resources the Xputer offers, (2) to develop compilation techniques which are able to transform programs written in a conventional high level programming language into code that runs on an Xputer (Figure 5).

## A systolizing compilation technique for the MoM

With the concepts of an Xputer language at hand it is now possible to examine compilation techniques, which transform a program written in a conventional high level programming language into an equivalent Xputer “program”. Since Xputer applications mainly focus on applications with regular data-dependencies, we first restrict our point of view to the automatic derivation of efficient code for systolizable algorithms. Since an Xputer cache provides neigh-

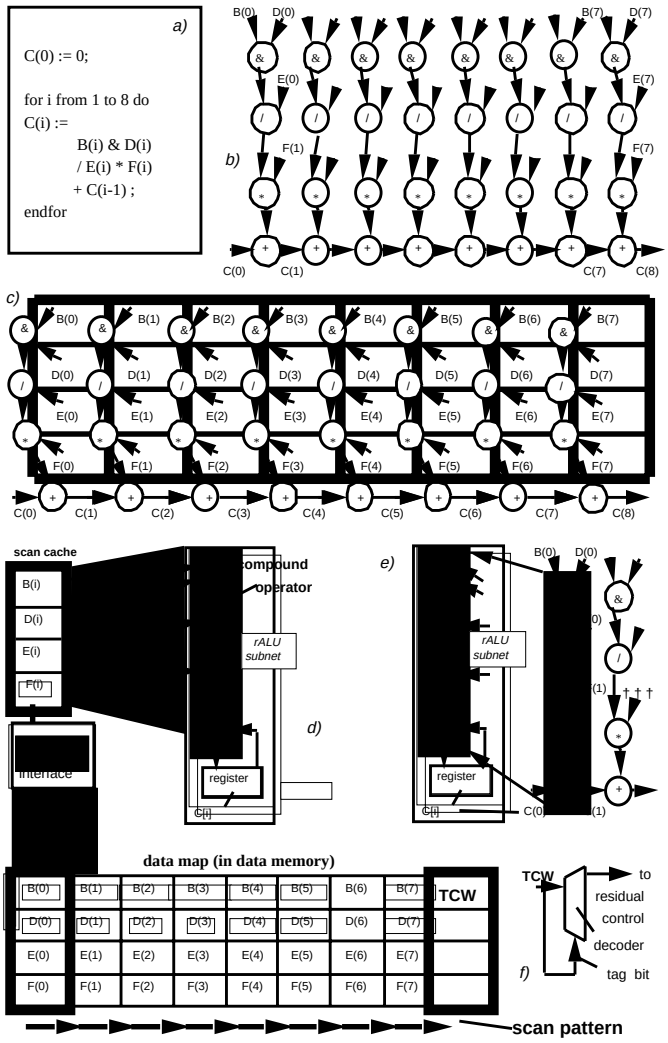



**Figure 4** Simple systolizable algorithm MoM execution example illustrating the compilation task: a) textual algorithm specification, b) graphic version of specification: signal flow graph (SFG), c) deriving a data map from SFG, e) deriving a compound operator for rALU from SFG, d) deriving scan cache size, rALU interconnect and scan pattern (also illustrating auto-apply and auto-xfer operation: needed for data-procedural machine principles)

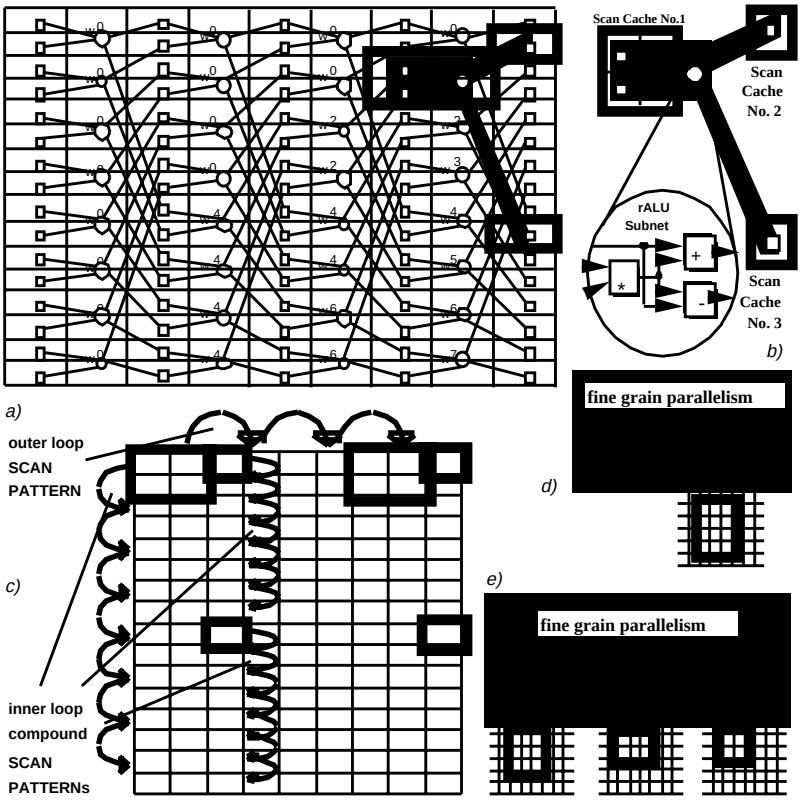



**Figure 5** Field-programmable rALU as a communication mechanism: a) through c) FFT algorithm implementation example using 3 scan caches synchronously in parallel: a) SFG, data map, and a scan cache location snapshot, b) deriving rALU subnet, scan cache sizes and interconnect, c) illustration of nested and compound scan patterns; d, e) illustration of fine grain parallelism: d) single cache application, e) multiple scan caches.

bourhood communication very efficiently, it is most promising to look at systolizing compilation. Since the term of systolizing compilation is used in three different research and development areas where it causes different associations we would like to clarify the terminology. We distinguish three different uses of *systolizing methods*:

- compilation of systolic arrays (also ASAP compilation): automatic derivation of systolic arras or similar ASAPs from a given algorithm specification
- systolizing (parallelizing) compilation: methods used by parallelizing compilers for von Neumann parallel computer systems (to distribute work load to several von Neumann computers running concurrently)
- systolizing xpilation: automatic derivation of Xputer object code which simulates systolic array operation on a monoprocessor Xputer.

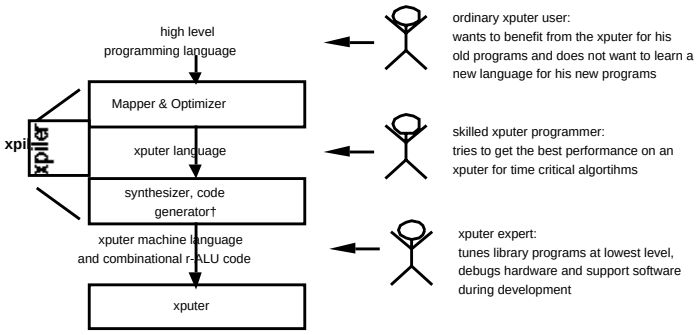



**Figure 6** Basic building blocks of an Xputer environment

If not indicated otherwise, in this paper we use also the term systolizing compilation for systolizing xpilation, systolizing compilation enhances the neighbourhood communication by projecting the data dependency graph of an algorithm into time and space. The physical space in the ASAP world is usually covered by processing elements (PEs), each being located at a particular point in that space. For a systolizing compilation onto von Neumann computer software, processes are used to model the processing elements. So always on a multi computer system is derived. An Xputer, however, is a monoprocessing system, i.e. a single stream of auto apply actions steers a single, but inside highly parallel operation unit. The problem therefore is to map the spatially distributed parallelism of a systolic system onto Xputer hardware resources. The MoM-DE (MoM Development Environment) system developed at Kaiserslautern practices systolizing xpilation by using ASAP compilation as a front end for the xpiler (Figure 7).

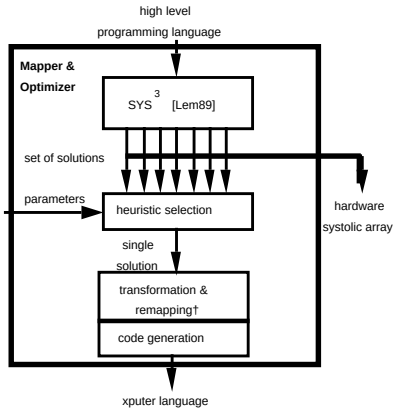



**Figure 7** inside the xpiler's mapper and optimizer

By a systolic ASAP synthesis system (Lemmert 1989) an algorithm is transformed to an systolic array. The xpiler (Weber 1990) uses only the information which is contained in a description of the systolic array. The program source itself is not necessary any more, since its sequential form is completely rearranged by the systolization process. All required information about the program has been extracted and is provided by the xpiler. The systolic array thus obtained can be mapped onto the Xputer using only a single scan cache. The Xputer scan cache at a time accesses those data stream variables from an execution field which are accessed by a single processing element in the systolic array. The rALU performs the same operation as a single processing element does. The data sequencing of the scan cache has to emulate the systolic array performance by moving the scan cache in the execution field in a way that the current sequence of visits to data is performed. This sequence moves the cache countercurrent to the data streams.

The MoM-DE does xpilation in 4 steps: (1) For the first step of the systolization the $SYS^3$ method (Lemmert 1989) is used. All possible space-time transformations which allow to implement the algorithm at the shortest execution time are computed. This results in quite a number of alternative systolic designs, all with different spatial shape and different directions and delays in the data streams. (2) By a heuristic search with different special conditions and criteria (Weber 1990) the best solution is supported (Figure 8). The heuristic selection of one systolic solution is followed by the actual transformation of the gained information about the algorithm into the Xputer language.

**Formal derivation of scan patterns**

The most interesting part is the derivation of scan patterns. Using only a single cache means (due to systolization), that the parallel work of the many processing elements form the ASAP has to be done sequentially by the Xputer. Although it sounds confusing: from this point of view a systolic array with only a single PE is a model of the XPuter paradigm. But because of its high efficiency its performance often is competitive. The directions of the cache movements to be selected are opposite to the directions of the data streams in the ASAP. Otherwise data needed again later would be overridden (Weber 1990). A straightforward approach is to move the cache along the execution wavefronts which are given by the locations of the ASAP's active PEs during the snapshot sequence of systolic time steps.

To achieve this data sequencing the partial order over the basic statements of the algorithm by the time transformation $M_t(i)$ has to be expanded until a total order (an ordered sequence of statements is gained). This order is developed inductively over the dimensions of the space domain. First the set of processor locations in a 2-D area is split into sets of processor locations on a 1-D line. On these lines a total order of the locations can be derived. The ordering transformations are controlled by the shape of the data flow graph of the space-time transformed algorithm. The Cut-Set temporal localization procedure, which can be used to extract the waveforms in the systolic data flow graph, can be adapted to control the ordering, but here another methods is used which exploits the features of the data sequencer architecture more efficiently. Finally, according to the "nested" orders the cache movements can be coded as nested data sequences in the Xputer language.

Finally a code generator for the MoM does the generation of all the code needed for execution: for loading the data to the memory, for rALU configuration and for starting the appropriate scan patterns. After the steps described above, a program in Xputer language is obtained. Code generation for the Xputer Hardware has to be done now: the data sequencing is transformed

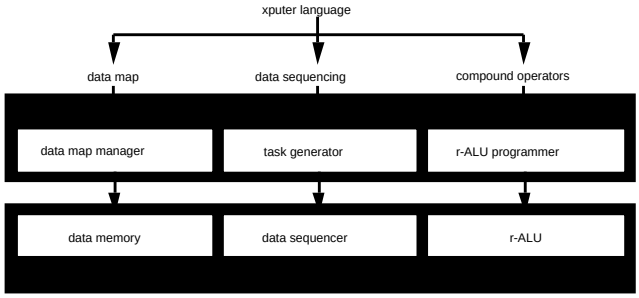



**Figure 8** Inside the MoM-DE synthesizer and code generator

into task parameter for the data sequencer, the data map description is used to load into the data memory and the compound operators are transformed into combinational code to be laid down in the rALU subnets (see the application environment Figure 7 and Figure 8).

### Generalisation of systolizing xpilation

So far only a single scan cache has been used by systolizing xpilation methods dealt with above. Figure 5 shows the underlying Xputer hardware model. That is why from the ASAP point of view an Xputer looks like an ASAP with only a single PE. The only difference is, that data are not streaming (like ASAP use), but directly are accessed from memory. Instead of data, the scan cache location is moving due to a scan pattern, which also may be a complex one. This method suffers from the same locality restriction, which stems from ASAPs, where PEs can communicate only with their nearest neighbours. Since Xputer efficiency requires regular data dependencies the consequence would be, that the superiority of Xputers over computers only holds for systolic or systolizable algorithms.

However, a single PE does not have neighbours: so, why locality restrictions? These restrictions are not imposed by the target hardware, but only by the use of systolizing Xpiler front end, which only processes algorithms with local regular data dependencies (what we call systolizable algorithms). To overcome this restriction another mapping methods has to be used. If the hardware architecture provides more than a single scan cache (hardware model by Figure 5e) also algorithms with non-locally regular data dependencies can be implemented with the same efficiency. Figure 5a illustrates an example application: the FFT (fast fourier transform) by showing its signal flow graph (SFG), where also long distance referencing occurs (which would violate typical ASAP's locality restrictions). The grid in the background demonstrates that a regular data map can be derived from this SFG. Figure 5 shows that the rALU using three caches sized individually, can be easily derived. Figure 5c indicates the nested scan patterns needed to run these three scan caches synchronously in parallel.

## CONCLUSIONS

The paper has introduced a novel compilation technique needed for the new machine paradigm of Xputers. It has been published elsewhere, that for algorithms with regular data dependencies such Xputers are by several orders of magnitude more efficient than von Neumann computers. The essential task of such new compilers and their differences to conventional compilation techniques have been illustrated by describing the execution of a source algorithm example. Details of the compilation technique itself and its implementation have been described. An essential new aspect of this new compilation technique is the fact, that for universality the essential technology platform of Xputers is field-programmable logic, or interconnect-reprogrammable hardware technology) in contrast to computers using the RAM as a central technology platform).

For systolizable algorithms the xpiler uses a program generator as a front end, which generates optimum MoPL programs by using projection techniques having been derived from those known from the scene of ASAP synthesis. For this its target hardware model is defined by (highly efficient) systolic array simulation. where only a single Xputer scan cache is used. This program generator has been implemented on a VAXStation under ULTRIX. It has been shown, that by using multiple scan caches this method will be generalized to cope with all kinds of algorithms with regular data dependencies: also with non-locally regular data dependencies, i.e. also for non systolizable algorithms with regular data dependencies.

## REFERENCES

DAC 1991 DAC 91,Proceedings of the ACM/IEEE Design Automation Conference, San Francisco, CA, USA, June 1991.

Fortes et al 1988Fortes, J. A. B., Fu, K. S. and Wah, B.J.,“Systematic Approaches for Algorithmically Specified Systolic Arrays”, in Computer Architecture: Concepts and Systems, Milutinovic e.d, North Holland,1988.

Freeman 1988Freeman, R.,“User Programmable Gate Arrays”,IEEE Spectrum,Dec.1988.

Gajski et al 1982Gajski, D., Padua, D., Kuck, D. and Kuhn, R. H., “A Second Opinion on Data Flow Machines and Languages”, Computer, pp. 58-69,Febr. 1982.

Hartenstein and Lemmert 1989Hartenstein, R. W. and Lemmert, K.,“A CHDL-based CAD-System for the Synthesis of Systolic Architectures”, Proc. Int´l Symposium on Hardware Description Languages and their Applications, Washington, DC, North Holland Publ. Company, Amsterdam,1989.

Hartenstein et al 1990aHartenstein, R. W., Hirschbiel, A. G. and Weber, M.,“A Novel Paradigm of Parallel Computation and its Use to Implement Simple High Performance Hardware”, Proc. of the Joint Conference on Vector and Parallel Processing, CONPAR ‘90 - VAPP IV, Zürich, Sept. 1990.

Hartenstein et al 1990bHartenstein, R. W., Hirschbiel, A. G., Lemmert, K., Schmidt, K. and Weber, M.,“A Novel Paradigm of Parallel Computation and its Use to Implement Simple High Performance Hardware”, Int´l Conf. on Information Technology, Tokyo, Japan, Oct. 1990.

Hartenstein et al 1990cHartenstein, R. W., Hirschbiel, A. G., Lemmert, K., Schmidt, K. and Weber, M.,“The Machine Paradigm of Xputers and its Application to Digital Signal Processing”, Proc. of 1990 International Conference on Parallel Processing, St. Charles, Oct. 1990.

Hirschbiel 1991 Hirschbiel, A. G., Xputers - Novel High Performance Architectures based on a New Machine Paradigm, PhD. Thesis, Kaiserslautern University,1991.

Kung 1982Kung, H.T.,“Why Systolic Architectures?”, IEEE Computer, pp. 37-46,Jan. 1982.

Lemmert 1989Lemmert, K.,SYS3 - a Systolic Synthesis System around KARL, PhD Thesis, Kaiserslautern University,1989.

Moldovan 1987Moldovan, D. I.,“ADVIS: A Software Package for the Design of Systolic Arrays”, IEEE Transactions on Computer Aided Design, pp. 33-40,Jan. 1987.

Plessey 1990Plessey,Quickgate (Product Overview), Plessey Semiconductors, Swindon, U.K.,May 1990

Schmidt 1990Schmidt, K.,Programmierbare Logikbausteine, Architekturen und ihre Programmierung,Handout,Proseminar SS90, Universität Kaiserslautern, Germany,1990

Weber 1990Weber, M.,An Application Development Method for Xputers, PhD Thesis, Kaiserslautern University,1990.