

# High-Performance Computing Using a Reconfigurable Accelerator

Jürgen Becker, Reiner W. Hartenstein, Rainer Kress, Helmut Reinig  
University of Kaiserslautern

Erwin-Schrödinger-Straße, D-67663 Kaiserslautern, Germany

Fax: ++49 631 205 2640, email: abakus@informatik.uni-kl.de

## Abstract

*This paper introduces the MoM-3 as a reconfigurable accelerator for high performance computing at a moderate price. By using a new machine paradigm to trigger the operations in the MoM-3, this accelerator is especially suited to scientific algorithms, where the hardware structure can be configured to match the structure of the algorithm. The MoM-3 efficiently uses reconfigurable logic devices to provide a fine-grain parallelism, and multiple address generators to have the complete memory bandwidth free for data transfers (instead of fetching address computing instructions).*

*Speed-up factors up to 82, compared to state-of-the-art workstations, are demonstrated by means of an Ising spin system simulation example. Adding the MoM-3 as an accelerator allows to achieve supercomputer performance from a low-cost workstation.*

## 1. Introduction

Scientific computing provides the greatest challenges to modern workstations and even supercomputers. A lot of different computer architectures have been presented, which take into account characteristics, that are common to many scientific algorithms. Vector processors [4] speed up operations on large arrays of data by the use of pipelining techniques. Parallel multiprocessor architectures [15] benefit from the fact, that many operations on large amounts of data are independent from each other. This allows to distribute these operations onto different processors (or processing elements) and execute them in parallel. But all of these architectures basically still follow the von Neumann machine paradigm with a fixed instruction set, where the sequence of instructions triggers the accesses to data in memory and the data manipulations.

The Map-oriented Machine 3 (MoM-3) is an architecture based on the Xputer machine paradigm [3]. Instead of a hardwired ALU with a fixed instruction set, an Xputer has a reconfigurable ALU based on field-programmable devices. All data manipulations, which are performed in the loop bodies of an algorithm, are combined to a set of compound operators. Each compound operator matches a single loop body and takes several data words as input to produce a number of resulting data words. The compound operators are configured into the field-programmable devices. After configuration, an Xputer's "instruction set" consists only of the compound operators as they are required by the algorithm actually running on the Xputer. The combination of several operations of a high level language description to one compound operator allows to introduce pipelining and fine grain parallelism to a larger extend, as can be done in fixed instruction set processors. E.g. intermediate results can be passed along in the pipeline, instead of writing them back to the register file after every instruction. Since many scientific algorithms compute array indices in several nested loops, the sequence of data addresses in a program trace shows a regular pattern. This leads to the idea to have complex address generators compute such address sequences from a small parameter set, which describes the address pattern. And instead of an instruction sequencer as a centralized control to trigger the operations in the reconfigurable ALU, the address generators themselves serve as a decentralized control. They automatically activate the appropriate compound operator, each time a new set of input data is fetched from memory and the previous results have been written back. This so-called data sequencing mechanism directly matches the loop structure of the algo-



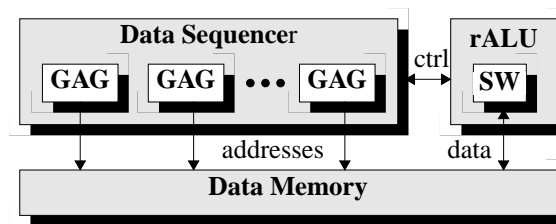


Figure 1. Block Diagram of an Xputer

rithm, where the index computations serve as a means to provide the right data to ever the same operations.

Field-programmable devices available commercially today are not well suited to implement arithmetic operations on wordlengths of 32 bits or more. Therefore, we developed our own architecture, the so-called rDPA (reconfigurable datapath architecture). Compared to other SRAM-based architectures like XILINX 4000 series [10], AT&T ORCA [9], or Altera's FLEX family [8], the rDPA is quite coarse grain. It provides a small array of so-called datapath units (DPU), where each can be configured to implement any operator from C programming language, as well as some others, that are stored in an extensible library. The wordlength of a single DPU is 32 bits. The configuration code for the rDPA and the address generators is derived from C programming language, without requiring further user interaction or compiler directives to obtain satisfactory results.

The following section introduces the hardware structure of the MoM-3, including the address generators and the rDPA. Afterwards, the features of the C compiler for the MoM-3 are outlined. The fourth section explains the way algorithms are executed on the MoM-3 by means of an example. The final sections provide a performance comparison and conclude the paper.

## 2. The MoM-3 Architecture

The MoM-3 architecture is based on the Xputer machine paradigm [3]. MoM is an acronym for Map oriented Machine, because the data memory is organized in two dimensions like a map. Instead of the von Neumann-like tight coupling of the instruction set to the data manipulations performed, an Xputer shows only a loose coupling between the sequencing mechanism and the ALU. That's why an Xputer efficiently supports a reconfigurable ALU (rALU). The rALU contains compound operators which produce a number of results from a number of input data (figure 1). All input and output data to the compound operators is stored in so-called scan windows (SW). A scan window is a programming model of a sliding window, which moves across data memory under control of a data address generator. All data in the scan windows can be accessed in parallel by the rALU operator. The rALU operators are activated every time a new set of input data is available in the scan window. This so-called data sequencing mechanism is deterministic, because the input data is addressed by Generic Address Generators (GAGs). They compute a deterministic sequence of data addresses from a set of algorithm-dependent parameters. An Xputer Data Sequencer contains several Generic Address Generators running in parallel, to be able to efficiently cope with multiple data sources and destinations for one set of compound operators.

In the MoM-3, the Data Sequencer is distributed across several computational modules (C-Modules), as can be seen in figure 2. The MoM-3 includes up to seven C-Modules. Each C-Module consists of a Generic Address Generator (GAG), an rALU subnet and at least two megabytes of local memory. All C-Modules can operate in parallel when each Generic Address Generator accesses data in its local memory only. Apart from the local memory access, two means of global communication are available. First, the rALU subnets can exchange internal results with their neighbours without disturbing parallel activity. Second, the Generic Address Generators can access data memory and rALU subnets on other C-Modules using the global MoMbus. This can be done only sequentially, of course. The global MoMbus is used by the MoM-3 controller (M3C) as well, to reconfigure the address generators and the rALU whenever necessary. The MoM-3 controller is the



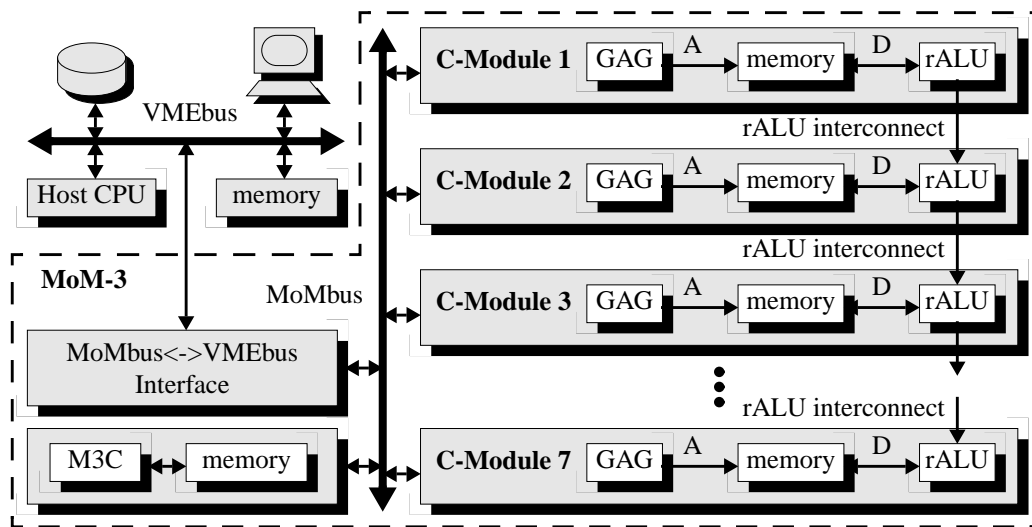


Figure 2. The MoM-3 Hardware

coordinating instance of the Data Sequencer, which ensures a well-defined parallel activity of the Generic Address Generators by selecting the appropriate parameter sets for configuration. Via the MoMbus-VMEbus interface, the host CPU has access to all memory areas of the MoM-3 and vice versa. The Generic Address Generators have DMA capability to the host's main memory, reducing the time to download a part of an application for execution on the MoM-3. The host CPU is responsible for all disk I/O, user interaction and memory allocation, so that the MoM-3 completely uses the functionality of the host's operating system.

The printed circuit boards with the MoM-3 controller and the C-Modules reside in their own 19 inch rack with the MoMbus backplane. The only connection to the host computer is through the bus interface. That way, the MoM-3 could be interfaced to a different host with minimal effort. The part of the bus interface that is injected into the host's backplane would have to be redesigned according to the host's bus specifications. All other parts could remain the same.

## 2.1 The Data Sequencer

The MoM-3 Data Sequencer consists of up to seven Generic Address Generators and the MoM-3 controller. Each Generic Address Generator controls one scan window. It operates in a two-stage pipeline. The first stage computes handle positions for the scan windows (Handle Position Generator). A handle position consists of two 16-bit values for the two dimensions of the data memory. The sequence of handle positions describes how the corresponding scan window is moved across the data memory (figure 3). Such a sequence of handle positions is called scan pattern. A Handle Position Generator can produce a scan pattern corresponding to four nested loops at the most. It is programmed by specifying a set of parameters, such like starting position, increment value, and end position of a loop, each both for the x and y dimension of the data memory.

The second pipeline stage computes a sequence of offsets to the handle positions, to obtain the effective memory addresses for the data transfers between the scan window / rALU and the data memory. Therefore this stage is called Memory Address Generator. The range of offsets may be -32 to +31 in both dimensions of the data memory. The sequence of offsets may be programmed arbitrarily, but at most 250 references to the data memory can be made from one handle position. The offset sequence may be varied at the beginning and at the end of the loops of the Handle Address Generator. This allows to perform extra memory accesses to fill a pipeline in the rALU at the beginning of a loop, as well as additional write operations to flush a pipeline at the end. The address parts of the two dimensions are combined to a linear memory address. After the concatena-



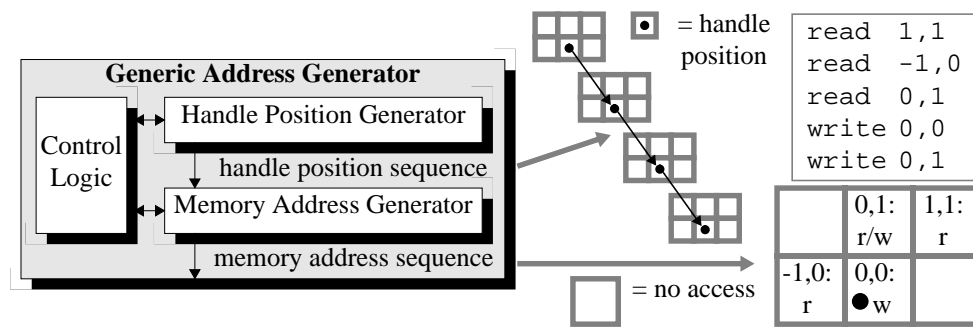


Figure 3. Generic Address Generator

tion of the two address parts, a 32-bit base address is added, to obtain the effective memory address. The base address typically is the starting address of the data array referenced by this Generic Address Generator, as determined by the compiler. The Memory Address Generator itself is a three stage pipeline. The first stage performs the offset calculations and the combination of address parts to a linear address. The second stage adds the 32-bit base address, and the third stage handles the bus protocol for data transfers.

The Generic Address Generators run in parallel. They synchronize their scan patterns through the activations of the rALU. All scan patterns proceed until either they detect an explicit synchronization point in the offset sequence of the Memory Address Generator, or they are blocked by a write operation to memory, waiting for the rALU to compute the desired result.

The hardware controlled generation of long address sequences allows to access data in memory every 120 ns, using 70 ns memory devices and a conventional, non-interleaved memory organization. A further speed-up of memory accesses could be obtained by interleaved memory banks and by the introduction of static RAM caches, like in conventional computers.

## 2.2 Reconfigurable ALU

One rALU-subnet of the MoM-3 is shown in figure 4. It contains an rDPA array (reconfigurable datapath architecture) made of eight rDPA chips, arranged in a two by four array. Each rDPA chip contains an array of four by four datapath units (DPU). A datapath unit can implement any unary or binary operator from C language on integer or fixed-point data types up to 32 bits length. Multiplication and division are performed by means of a microprogram, whereas all other operators can be executed directly. Additionally, operators like multiplication with accumulation are available in the

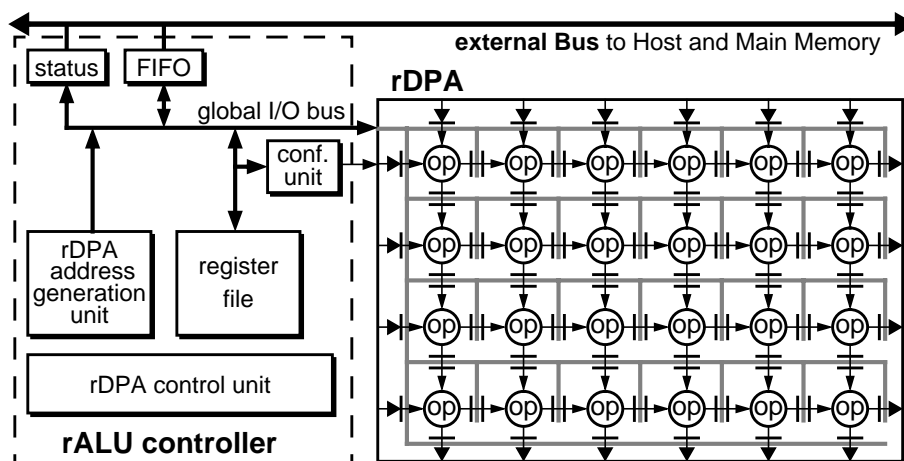


Figure 4. One subnet of the reconfigurable data-driven ALU



extensible library of configurable operators. Floating-point arithmetic cannot be done in a single datapath unit. But it is possible to build a pipelined floating-point operator using several adjacent datapath units, e.g. two DPUs for normalization and denormalization and at least one further DPU for the operation. This requires a sequential transfer of multiple intermediate values along the DPU interconnect, which are used as operands to a pipelined floating-point operation. This can be done, because each datapath unit consists of a conventional ALU, a microprogram memory and a sequencer for horizontal microcode. The microprogram for a floating-point operator simply contains multiple data transfers from the preceding datapath unit before the operation starts. Each datapath unit has two input registers, one at the north and one at the west, and two output registers, one at the south and one at the east. All registers can be used for intermediate results throughout the computation. The registers in the datapath units in fact are a distributed implementation of the scan window of the Xputer paradigm. The datapath units can read 32-bit intermediate results from their neighbours to the west and to the north and pass 32-bit results to their neighbours in south and/or east direction. A global I/O bus allows input and output data to be written directly to a datapath unit without passing them from neighbour to neighbour. The datapath units operate data-driven. The operator is applied each time, new input data is available either from the bus or from a neighbouring datapath unit. This decentralized control simplifies pipelining of operations, when each takes a different time to execute.

The array of datapath units expands across rDPA chip boundaries in a way that is completely transparent to the compiler software. To overcome pinout restrictions, the neighbour to neighbour connections are reduced to serial links with appropriate converters at the chip boundaries. A pair of converters behaves like a special-purpose datapath unit, restricted to routing operations in the appropriate direction. To the programming software, the serial links combined with the preceding DPU appear to be a DPU with an increased latency for data transfers. Although pipelining drastically reduces the penalty of the conversion from 32 bits to 2 bits, this still may turn out a bottleneck in the current rDPA prototype with some algorithms. With state of the art IC technology and packages larger than 144 pins, we could build an rDPA with 66 MHz serial links (instead of 33 MHz) and four bits wide communications channels. This fourfold speed-up would overcome the shortfalls of the current prototype.

In addition to the rDPA array, an rALU controller circuit interfaces to the MoMbus. It provides a register file as a kind of cache for frequently used data, and controls the data transfers on the global I/O bus. The rDPA, in combination with the rALU controller, supports pipelining on operator level (e.g. floating point operations are implemented as a pipeline across several datapath units), pipeline chaining [4], and pipelining of loop bodies, as shown in the example in section 4.1. That way, the compound operators of subsequent loop iterations are computed in parallel in a pipeline. Each of the loop iterations is finished to a different degree, depending on its progress in the pipeline.

### 3. The Software Development Environment

The C compiler for the MoM-3 takes an almost complete subset of ANSI C as input. Only constructs, which would require a dynamic memory management to be run on the MoM-3 are excluded. These are pointers, operating system calls and recursive functions. Since the host's operating system takes care of memory management and I/O, the software parts written for execution on the MoM-3 do not need such constructs. Especially for scientific computing, the restrictions of the C subset are not that important, since FORTRAN 77 lacks the same features and is most popular in scientific computing. There are no extensions to C language or compiler directives required to produce configuration code for the MoM-3. The compiler computes the parameter sets for the Generic Address Generators, the configuration code for the rDPA arrays, and the reconfiguration instructions for the MoM-3 controller, without further user interaction. First, the compiler performs a data and control flow analysis. The data structure obtained allows restructurations to perform parallelizations like those done by compilers for supercomputers. These include vectorization, loop



unrolling, and pipelining on expression level. The next step performs a re-ordering of data accesses to obtain access sequences, which can be mapped well to the parameters of the Generic Address Generators. Therefore, the compiler generates a so-called data map, which describes the way the input data has to be distributed in the data memory to obtain optimized hardware generated address sequences. After a final automatic partitioning, data manipulations are translated to a rALU description, and the control structures of the algorithm are transformed to Data Sequencer code. An assembler for the Data Sequencer translates the Data Sequencer code to parameter sets for the Generic Address Generators and a reconfiguration scheme for the MoM-3 controller.

The rALU description is parsed by the ALE-X assembler (arithmetic and logic expression language for Xputers). It generates a mapping of operators to datapath units, merges DPU operators where possible, and computes a conflict-free I/O schedule, which matches the operators' speed, to keep the datapath units as busy as possible. The separation of the rALU code generation from the rest of the compiler allows to use other devices than the rDPA to build a MoM-3 rALU, without having to rewrite the C compiler with all its optimizations. For a new rALU, only an assembler generating rALU configuration code from ALE-X specifications has to be written. This is not too difficult, because ALE-X describes only arithmetic and logic expressions on the scan windows' contents.

A more detailed description of the MoM-3 programming environment can be found in [14]. The most important benefit of the MoM-3 C compiler is the fully automatic code generation. The programmer neither has to be a hardware expert, to guide hardware synthesis with appropriate constraints or compiler directives, nor has he to interfere with different stages of the compilation to get reasonable results. All parallelizations are done from a C source without requiring special constructs like vector operations [13] or parallel flow control [5] to detect parallelism, in contrast to many parallelizing compilers for supercomputers. This allows to compile the same algorithm specification for execution on a conventional computer and on the MoM-3. Still better performance can be achieved, by manually optimizing the assembler output of the C compiler, e.g. to make use of special operators which can be combined into a single datapath unit.

#### 4. Example: Three-Dimensional Ising Model Simulations

The Ising model is used for the analysis of phase transitions. Originally it was employed to the understanding of ferromagnetism, but later on it was found to be of interest in the formation of binary alloys, neural networks, cellular automata, and other problems in chemistry and molecular biology. The Ising model can be used to explain, how short-range interactions between components of a larger structure (e.g. molecules in a crystal) give rise to a long-range, correlative behaviour, and to predict in some sense the potential for a phase transition.

An Ising spin system consists of a one-, two-, or three-dimensional lattice (figure 5). Each lattice point is associated with a so-called spin variable (or spin)  $S_i$ , which can take the values +1 (up) or -1 (down). The state of an Ising spin system consisting of N spins at a given moment is determined by the spin configuration  $\{s\}$ , i.e. the values of all N spins at the moment,  $\{s\} = (S_0, S_1, \dots, S_{N-1})$ .

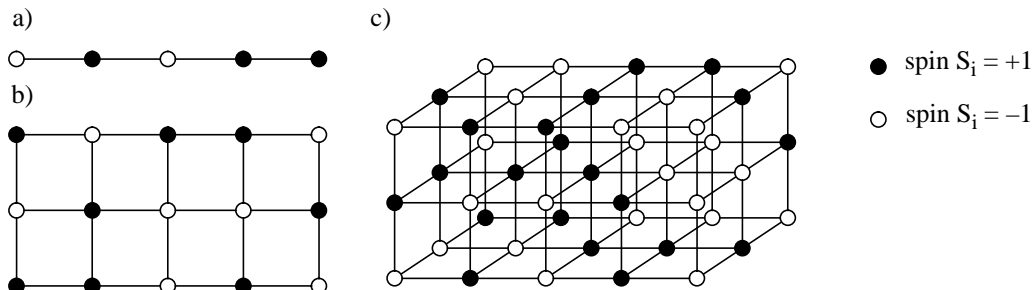


Figure 5. Ising model lattices: a) one-dimensional; b) two-dimensional; c) three-dimensional



Spins only interact with each other, if they are nearest neighbours in the lattice, connected by a line segment as shown in figure 5. A general introduction to the Ising model can be found in [2].

In this example we employ the heatbath algorithm [1] for Monte Carlo simulations of Ising spin systems. A spin  $S_i$  is set to the value +1 (up) with a probability  $P(S_i=1)$  depending on  $E_i$ , the interaction energy of spin  $S_i$  with the rest of the lattice, when its value is +1 (up):

$$\Delta E_i = \sum_j J_{ij} \cdot S_j \quad (1)$$

In the case of an Ising ferromagnet  $J_{ij}$  is equal to a negative constant  $J$ . For a three-dimensional lattice, six nearest neighbours have to be taken into account in a summation similar to (1) to compute  $P(S_i=1)$ . Since the probabilities  $P(S_i=1)$  are time-independent, they can be computed beforehand and stored in a look-up table of 64 entries. The six values of the interacting spins  $S_k$  are combined to an address by concatenating a set bit, if the spin  $S_k$  is +1 (up), and a cleared bit if the spin  $S_k$  is -1 (down). This six bit address is used to read the appropriate probability that  $S_i$  will become +1 from the look-up table. The probabilistic nature of a Monte Carlo simulation is introduced by comparing the probability  $P(S_i=1)$  from the look-up table with a random number  $N$ , and setting  $S_i$  to +1 if and only if  $N < P(S_i=1)$ .

One application of this procedure is called a spin update. About 1700 spin updates have to be performed, to generate a new valid spin configuration  $\{s\}$ . For each temperature this has to be repeated about 10000 times to compute an average on all the spin configurations obtained [11]. Near the critical temperature, where phase transitions are expected, several simulations at different nearby temperature values have to be run to obtain meaningful results.

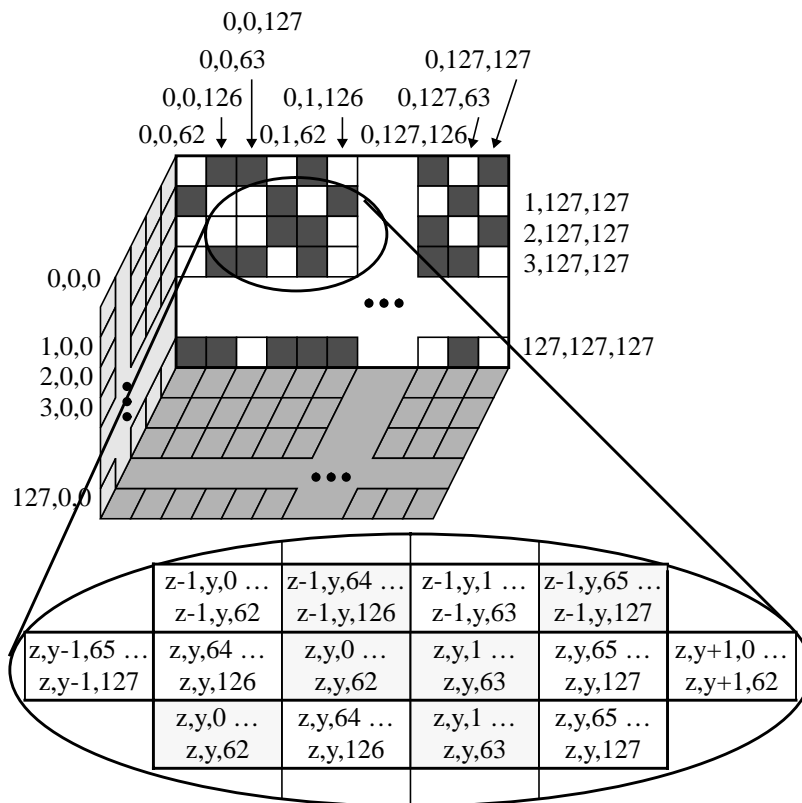
#### 4.1 MoM-3 Implementation

The simulation of up to three-dimensional Ising systems with nearest neighbour interactions can be handled efficiently on the MoM-3, independent from the size of the lattice. The spin configuration of the lattice is stored in conventional memory. Since typical Ising simulations require the averaging of a lot of runs, multiple C-Modules can be used most efficiently to perform separate Ising simulations in parallel. This eliminates the need to exchange intermediate results between C-Modules and simplifies the algorithm implementation.

The value of a single spin variable  $S_i$  can be coded in a single bit, where a set bit represents a +1 (up) and a cleared bit represents a -1 (down) spin. To reduce memory and I/O requirements, 32 spin variables are packed into a single 32-bit memory word and transferred in a single memory access. A 128 x 128 x 128 lattice requires  $128 * 128 * 4 = 65536$  memory words of 32-bit length (= 256 Kilobyte), for example. The rALU takes care for proper unpacking of the spin variables as well as the packing of 32 results to a single memory word. The rALU manipulations on such a 32-bit block of spin variables are done in a pipeline. The data map describing the memory organization can be found in figure 6. The distribution of the spin variables onto the memory cannot be done straightforward, because of the pipelining in the rALU. To produce correct results, the selection of spins has to make sure that only spin updates are performed, which can be computed independently from each other. Only after the first results have been written back from the pipeline, the spin updates can be performed, which require these spin variables as input. A proper pipelining can be achieved by computing the updates to the spin variables with even  $x$  indices first and then those with the odd  $x$  indices. Therefore, all spin variables with even  $x$  indices are packed into two 32-bit words, then those with odd indices (see lower part of figure 6). The results with the lower even  $x$  indices are available before the computation of the lower odd indices start, so that the pipelined operation works well.

As can be seen from the indices of the spin variables in figure 6, four adjacent data words are used to store a row of 128 spin variables of the same  $y$  and  $z$  dimension. 128 of such blocks of four data words are used to store a complete plane of the lattice, where all spins have the same  $y$  index. With





**Figure 6. Data map for a  $128^3$  Ising model simulation**

128 rows of data words, the complete lattice can be mapped onto the two-dimensional data memory organization of the MoM-3.

One spin update for  $S_{x,y,z}$  requires access to its six neighbours to form the address for the look-up table. At the border of the lattice, the common practice of wrapping around to the other end is applied. That way  $S_{127,y,z}$  is a neighbour to  $S_{0,y,z}$ , for example. An rALU operator for a single spin update is shown in figure 7.

The datapath units marked “up1...” (unpack 1 bit) accept one data word containing 32 spin values from the global I/O bus and feed a single bit per spin update to the following datapath unit (DPU). Only after all 32 bits have been passed on, the next data word has to be fetched from the bus. The DPUs marked “c...” (combine) are used to incrementally combine the bit sequences from the unpacking DPUs to a six bit address for the look-up table. The look-up table (LUT) is duplicated into two DPUs marked “lut”, because the look-up table access turns out to be the slowest operation in the pipeline. The output of the “lut” DPU is the probability value  $P(S_i=1)$  to the actual configuration of the six interacting spin variables. Every second value is retrieved from the own LUT, whereas in between the value from the other LUT is routed through.  $P(S_i=1)$  is compared to a random number produced by a DPU marked “rng”. Using a 64 bit linear feedback shift register, the random number can be computed efficiently and provides a cycle long enough for the large number of spin updates performed. The DPU marked “c&p” (compare and pack) not only compares the random number to the value from the look-up table to compute the resulting spin value, but packs 32 results to a single data word to be written back to memory every 32 spin updates. In order to make use of the unpacking of bits, and the random number generators in a single DPU, manual changes to the assembler output of the compiler have to be done. Since these DPUs perform a sequence of operations, the C compiler would interpret the corresponding code to consist both of



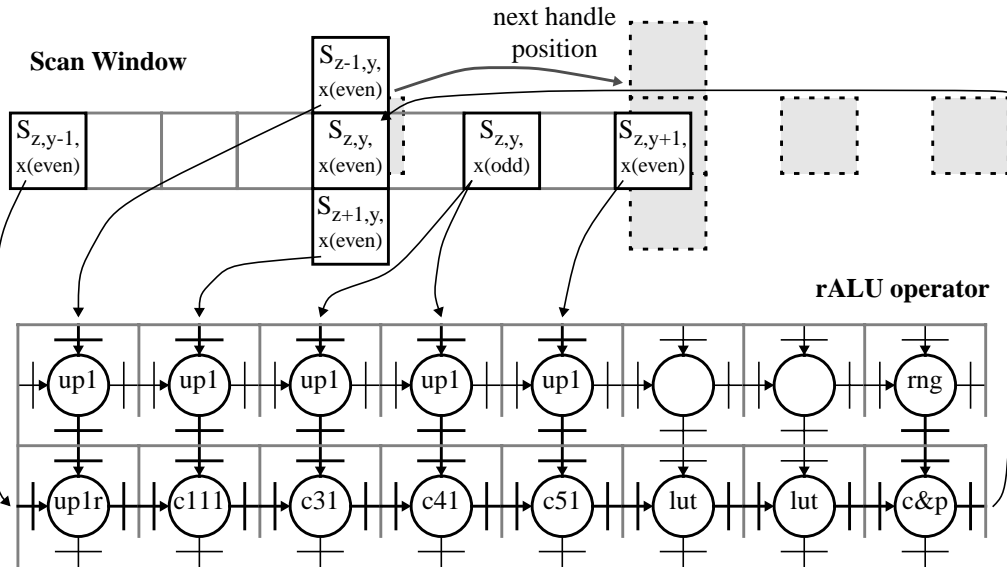


Figure 7. rALU operator and scan window to compute a single spin update

data sequencing and rALU operations. Therefore, the combination of these operations into a single DPU has to be done manually.

As can be seen in figure 7, the scan window to fetch the input operands from memory and write back the results resembles the form of a cross. The scan pattern to perform a complete sweep through the lattice is a simple video-scan like sequence, where the scan window moves in steps of four memory words within each row in a row by row pattern. The scan along the four words in  $x$  dimension is performed completely by the Memory Address Generator.

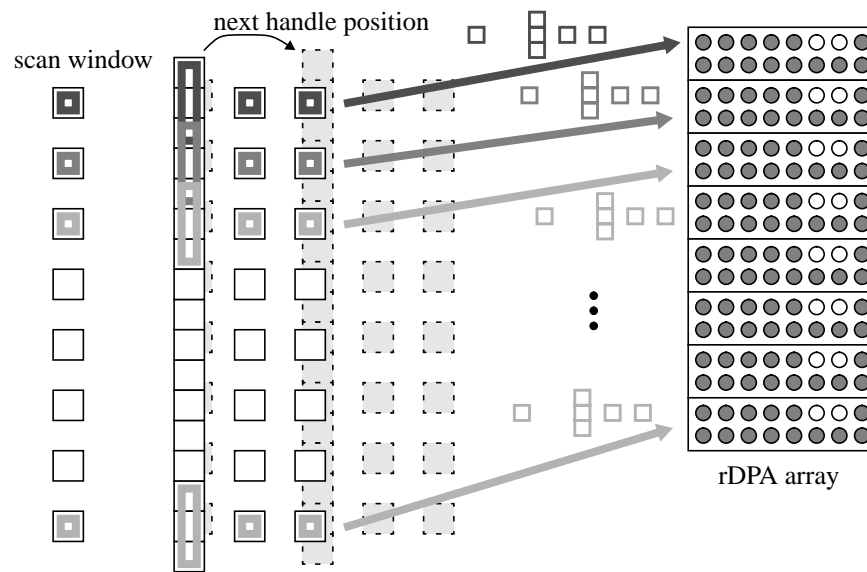
A whole compound operator for a single spin update fits into a bounding box of two by eight DPUs. Therefore eight spin updates can be computed in parallel on each C-Module. This leads to a modified form of the scan window, because multiple compound operators require access to their operands in parallel. The parallelization of spin updates has to obey the same restrictions as the pipelining of spin updates, in that only independent operations may be parallelized. This can be achieved most simply and efficiently, by parallelizing the update for every second spin variable in  $y$  dimension as shown in figure 8. All operands that are required by multiple spin update operators are fetched from memory only once and then read from the register file in the rALU controller for all other references to the same data. Therefore, only 33 words have to be read from memory, 15 words are read from the register file, and eight result words are written to memory every 256 ( $= 8 * 32$ ) spin updates.

The performance of the MoM-3 on this algorithm can be computed quite easily. First, the critical path has to be determined, which can either be the data transfers to/from memory, the passing on of intermediate results along the serial links, or the computations of the most complex operator in a single DPU. Since all DPUs operate in a pipeline, the overall computation time for a compound operator contributes only to the latency of the computation, but not to the throughput of the MoM-3 on a given algorithm. With a 33-MHz prototype of the MoM-3, using serial links of two data bits, the following performance can be achieved. The I/O time  $t_{IO}$  for a column of eight parallel spin updates results from the following equation

$$t_{IO} = \frac{\text{MemAcc} \cdot t_{\text{mem}} + \text{RFAcc} \cdot t_{\text{rf}}}{\text{Pack}} \quad (2)$$

where MemAcc is the number of memory accesses,  $t_{\text{mem}}$  is the memory access time, RFAcc is the number of register file accesses,  $t_{\text{rf}}$  is the register file access time, and Pack is the number of spin





**Figure 8. rALU configuration for eight Ising spin updates in parallel**

variables packed into one data word. With  $\text{MemAcc} = 41$ ,  $\text{RFAcc} = 15$ ,  $t_{\text{mem}} = 120$  ns,  $t_{\text{rf}} = 60$  ns, and  $\text{Pack} = 32$ , according to the description above, the I/O time  $t_{\text{IO}}$  per eight parallel spin updates sums up to 181.9 ns. The speed of the serial links is two clock cycles overhead plus 18 clock cycles to convert 36 bits to two-bit packages. With a 30 ns cycle time, this results in 600 ns for a serial link pipeline stage. The most complex operator in a single DPU is the look-up table, but because of the parallelization of two LUTs, the “c&p” comparison operator dominates the pipeline speed. For a column of eight spins it takes an average of 19.25 clock cycles to determine the new value for  $S_i$  by comparison and to pack the results. At 33 MHz, this accounts for 577.5 ns to get eight spins out of the pipeline. It is obvious that the current MoM-3 prototype is limited by the bandwidth of the serial links for this algorithm. The MoM-3 prototype can perform eight spin updates in parallel every 600 ns, resulting in 13.3 million spin updates per second with a single C-Module. Computing with seven C-Modules in parallel, this extends to 93.3 million spin updates per second.

A MoM-3 prototype based on state-of-the-art workstation technology could easily run at 50 MHz and utilize larger chip packages, allowing four data bits per serial link. The calculations for the Ising model simulation would change to the following then. Memory accesses can be assumed to take 60 ns with synchronous bus protocols and two-way interleaved memory banks. Register file accesses can be expected to take 30 ns only, which is still slower than the most second level caches in workstations. This reduces the I/O time to half its value, that is 90.9 ns. The speed of the serial links improves to 11 clocks because of the four bits wide data channels. The serial links can be expected to operate at a higher frequency of 66 MHz, because of their simplicity. This results in 165 ns to transfer one data word beyond rDPA chip boundaries. The computations in the pipeline take profit only from the reduced clock cycle time, leading to 390 ns per spin update operator. Now the computations are the bottleneck. Such a “new technology” MoM-3 NT would produce eight spin updates every 390 ns with a single C-Module, corresponding to 20.5 million spin updates per second. Seven C-Modules would perform 143.6 million spin updates per second. The following chapter compares these results to values published for other machines in literature.

## 5. Related Work and Performance Comparison

In literature, several performance figures of implementations of the Ising model on different machine architectures have been reported. A special-purpose processor has been built by Pearson,



Richardson, and Toussaint [11], which is capable of 25 million spin updates per second. Reddaway, Scott, and Smith [12] compare the performance of a 64 x 64 DAP to a CYBER 205 on Ising model simulations. With a 128 x 128 x 144 lattice, the DAP computes 218 million spin updates per seconds, and slightly less with a 128 x 128 x 128 lattice. They quote the CYBER 205 supercomputer to be capable of 22 million spin updates per second. Monaghan, O'Brien, and Noakes [7] have built a reconfigurable processor specialized on Ising model simulations. It is an extremely low-cost system based on two Xilinx 3000 series FPGAs and some static RAM, to be plugged into a PC-AT personal computer. It yields 4 million spin updates per second. We have implemented the same heatbath algorithm as on the MoM-3 (based on a look-up table, without the time-consuming bit-packing but direct accessible spin values instead) on a state-of-the-art Sun Sparcstation 10/51 running at 50 MHz and found that it is capable of 1.74 million spin updates per second. The performance of the MoM-3 prototype on the Ising model simulation has already been shown to be 93.3, respectively 143.6 million spin updates per second. The price per C-Module with a medium quantity production would be about \$3200. Adding \$3000 for a mounting rack, power supply and host interface, a full-sized MoM-3 would cost approximately \$25000, which is in the range of a high-end workstation.

The MoM-3 is not restricted to Ising spin systems, of course. A two-dimensional FIR filter algorithm has been implemented with different kernel sizes and weights. Input to the filter is a 1280 by 1024 pixel grayscale image using 8 bits per pixel. Only the time to filter the image in memory is counted, excluding disk I/O operations. The performance of this algorithm on the MoM-3 host computer and on a modern Sparcstation 10 is compared to the MoM-3 in table 1. The MoM-3 host computer is an ELTEC workstation running a Motorola 68020 processor at 16 MHz, with 9 Megabytes of RAM. The Sparcstation 10 is a Model 51 with 1 MByte Super-Cache, 50 MHz clock frequency and 128 Megabytes RAM. Performance measurements are done on a quiescent system, so that they are not invalidated by heavy multi-user activity.

Another application is a JPEG compression algorithm, based on the IJG program "cjpeg" from Independent JPEG Group [6]. Again, only in-memory compression is measured. A 704 by 576 RGB colour image using 24 bits per pixel is used as input for the time measurements.

Algorithms	CPU Time (in seconds)			Speed-up Factors		
	68020, 16 MHz	Sparc 10, 50 MHz	MoM-3, 33 MHz	MoM-3 vs. 68020	MoM-3 vs. Sparc	MoM-3 NT vs. Sparc
Ising	7328	120.8	2.247	3261	53.8	82.7
3x3 2-D FIR	32.1	0.71	0.018	1784	39.4	73.2
5x5 2-D FIR	145.9	1.73	0.038	3839	45.5	66.5
7x7 2-D FIR	300.4	3.20	0.058	5179	55.2	82.1
JPEG	74.5	1.51	0.207	360	7.3	13.2

**Table 1. Performance evaluation: MoM-3 vs. Sparc 10 and current host computer**

The first column of table 1 lists the algorithms used for performance comparison (Ising: 100 sweeps through the lattice). The second and third columns give the CPU times for the conventional computers in the comparison. They are based on compilations (GNU gcc) with all optimizations turned on, producing output specially adapted to exactly the kind of processor inside the computer. The fourth column indicates the time to run the same algorithm on the MoM-3 prototype.

The three rightmost columns of table 1 illustrate the speed-up factors obtained compared to the hosting ELTEC workstation and a modern SUN Sparc 10/51. The last column takes into account, that our prototype is built with an inferior technology compared to a modern Sparcstation. It states the speed-up factor for a new technology MoM-3 NT (as characterized at the end of section 4.1) compared to a Sparcstation 10/51.



## 6. Conclusions

The MoM-3, a configurable accelerator has been presented, which provides acceleration factors in the range of 7 to 82 when compared to a state-of-the-art workstation. The custom designed rDPA circuit provides a coarse grain field-programmable hardware, especially suited for 32-bit datapaths for pipelined arithmetic. A new sequencing paradigm supports multiple address generators and a loose coupling between sequencing mechanism and ALU. The address generation under hardware control leaves all memory accesses free for data transfers, providing de facto a higher memory bandwidth from the same memory hardware. The loose coupling of the data sequencing paradigm allows to fully exploit the benefits of reconfigurable computing devices.

The MoM-3 prototype is currently being built as a co-processor to a VMEbus-based ELTEC workstation. The address generators, the MoM-3 controller and the rALU controller have just returned from fabrication and are now under test. All three are integrated circuits based on 1.0  $\mu$ m CMOS standard cells, a technology made available by the EUROCHIP project of the CEC. The rDPA circuits will be submitted to fabrication in a 0.7  $\mu$ m CMOS standard cell process soon. All MoM-3 performance figures were obtained from simulations of the completed circuit designs, which were integrated into a simulation environment, describing the whole MoM-3 at functional level. The C compiler and its development environment are implemented in C programming language, running under SunOS on Sparcstations.

## References

- [1] K. Binder (ed.): Applications of the Monte Carlo Methods in Statistical Physics; Springer-Verlag, Berlin, 1984
- [2] Barry, A. Cipra: An Introduction to the Ising Model; American Mathematical Monthly, Vol. 94, pp. 937 - 959, 1987
- [3] R. W. Hartenstein, A. G. Hirschbiel, K. Schmidt, M. Weber: A Novel Paradigm of Parallel Computation and its Use to Implement Simple High-Performance Hardware; Future Generation Systems 7 (1991/92), p. 181-198, Elsevier Science Publishers, North-Holland, 1992
- [4] Jaap Hollenberg: The CRAY-2 Computer System; Supercomputer 8/9, pp. 17-22, July/September 1985
- [5] K. Hwang: Advanced Computer Architecture: Parallelism, Scalability, Programmability; McGraw-Hill, 1993
- [6] T. G. Lane: jpeg software, release 5; Independent JPEG Group (IJG), Sept. 1994
- [7] Sean Monaghan, Tom O'Brien, Peter Noakes: Use of FPGAs in Computational Physics; in Will Moore, Wayne Luk (eds.): FPGAs; Oxford 1991 Int'l Workshop on Field Programmable Logic and Applications, Abingdon EE&CS Books, 1991
- [8] N.N.: FLEX 8000 Data Book; Altera, Inc., 1993
- [9] N.N.: ORCA Preliminary Data Sheet, AT&T Microelectronics, 1994
- [10] N.N.: The XC4000 Data Book; Xilinx, Inc., 1994
- [11] Robert B. Pearson, John L. Richardson, Doug Toussaint: A Fast Processor for Monte-Carlo Simulation; Journal of Computational Physics, vol. 51, pp. 241 - 249, Academic Press, 1983
- [12] S.F. Reddaway, D.M. Scott, K.A. Smith: A Very High Speed Monte Carlo Simulation on DAP; Computer Physics Communications, vol. 37, pp. 351 - 356, North-Holland, 1985
- [13] John Reid: Fortran shapes up to the future; Scientific Computing World, January 1995
- [14] K. Schmidt: A Restructuring Compiler for Xputers; Ph. D. Thesis, University of Kaiserslautern, 1994
- [15] S.A. Zenios, R.A. Lasken: The Connection Machines CM-1 and CM-2: Solving Nonlinear Network Problems; Int'l. Conf. on Supercomputing, pp. 648-658, ACM Press, 1988

