Scalable Computing: Practice and Experience
Volume 11, Number 3, pp. 277–288. http://www.scpe.org
ISSN 1895-1767

HIGH PERFORMANCE COMPUTING THROUGH SOC COPROCESSORS

GIANNI DANESE, FRANCESCO LEPORATI, MARCO BERA, MAURO GIACHERO, NELSON NAZZICARI, AND ALVARO SPELGATTI*

Abstract. In this paper we describe DPFPA (Double Precision Floating Point Accelerator), a FPGA-based coprocessor interfaced to the CPU through standard bus connections; it is conceived to accelerate double precision floating point operations, featuring two double precision floating point units, a pipelined adder and a pipelined multiplier with a suitable number of stages. We tested its performance by implementing a Montecarlo-Metropolis simulation of a dipolar system, using a proper software development environment designed and realized in our laboratory. DPFPA can provide a speed-up equal to 4, with respect last generation PC, showing also a good scalability in terms of clock frequency, memory capability and number of computing units.

Key words: FPGA; hardware accelerator; high performance embedded system; parallel processing.

1. Introduction. Scientific research owes a lot to computer systems which allowed the achievement of results otherwise unthinkable [Marsh, 2005][Boghosian et al., 2005]. A powerful computing system permits the study of several phenomena through the employment of simulations like statistical ones into which the system under analysis is made to evolve from a certain initial condition, by modifying a few of its characteristic parameters and by evaluating the feasibility on the basis of a proper merit function. These operations are iterated thousands of times to bring the system in a new stable state.

Several of these simulations perform double precision floating point operations since they provide the accuracy required to appreciate even the smallest fluctuations in the typical variables of the simulated phenomena. On the other hand, this could represent a hard task even for the most powerful processors which take a lot of clock cycles to execute a single floating point operation.

The lack of computing power is generally overcome by resorting to supercomputers or clusters [Dongarra et al., 2005] but in the last years the use of accelerators, i. e. dedicated hardware systems, is gradually establishing as a valid alternative, due to the feature of these devices which allow to perform those operations in less time than traditional processors [Buell et al., 2007][Herbordt et al., 2007]. Several researchers worked in these years not only in this sense but also to improve "methodology, tools and practices supporting the integration of hardware and software components during system design and development" [Hankel et al., 2003][Wolf, 2003].

At present a similar project concerning a Double Precision Floating Point Accelerator (DPFPA) to process complex functions has been carried out in the Microcomputer laboratory at the University of Pavia (Italy). This activity suites well with the mission of the laboratory which aims to design and develop special purpose architecture for computationally intensive applications. The designed accelerator is implemented onto a FPGA device lodged on a board interconnected with a Personal Computer and is able to execute floating point operations faster than a traditional processor [Danese et al, 2007]. Moreover, a proper specific programming language and a suitable software development environment were realised allowing the user to write, translate and load proper instructions sequences written in a specific language.

This paper describes the implementation, onto the accelerator, of a Montecarlo-Metropolis simulation of a dipolar system, a typical computational challenge for supercomputers.

The Montecarlo Metropolis algorithm is an excellent benchmark to test performance of a special purpose calculation system, since its computational core consists of few floating point operations (double precision) repeated over and over: this represents the ideal condition to exploit an application specific architecture devoted to the acceleration of only particular instructions.

Moreover, the algorithm features a SIMD fashion so it is suitable for a distributed implementation which can exploit more calculation units so increasing the overall achieved speed up.

Finally, typical Montecarlo simulations involve hundreds thousands particle systems and can run for weeks or months on the most performing computers with a single CPU: the availability of powerful accelerating units, in case connected into a cluster configuration, makes possible simulations currently unfeasible or simulations with more particles than now, achieving a better comprehension of the physical phenomena under analysis.

* Department of Informatica and Sistemistica, Pavia University, Italy, E-mail: francesco.leporati@unipv.it, Phone: +39 0382 985678

In the past other research groups proposed accelerators based on FPGA for Montecarlo simulations:

- one of the first proposal is presented in [Postula et al., 1996] where is described a metallurgical sintering simulation implemented on a FPGA device with a two orders of magnitude speed-up with respect to a mid 90's workstation;
- in the same years, other authors conceived a FPGA implementation of a particular Montecarlo technique (Swendsen-Wang clustering) with a considerable acceleration with respect to a 15 MHz DSP or making use of cellular automata [Cowen et al, 1994][Monaghan et al, 1992];
- more recently, a reconfigurable computer was designed devoted to heat transfer simulations, working on single precision floating point data and achieving an order of magnitude speed-up relative to a 3 GHz P4 processor [Gokhale et al, 2003]; the peculiarity of this contribute is the idea of using widely available floating point libraries for implementing a calculation function onto FPGA, thus shortening design time;
- finally, in [Zhang et al, 2005] it is presented a simulation of a financial model implemented on a FPGA device to accelerate double precision floating point calculations. The achieved speed-up is 26 relative to a 1.5 GHz P4 processor;
- with regard to FPGA based architectures specifically devoted to physics simulations, the recent literature proposed the works of Cruz and Belletti [Cruz et al, 2001][Belletti et al, 2006]; the first one provides interesting architectural issues although using Altera Flex 10K30 components limits the working frequency to 48 MHz; the second is a project subsequent to our one, employing Altera Stratix family components and aims to build a cluster of accelerators based on the most recent FPGA devices.

For what concerns a more general use of SoC for computing intensive applications there is a wide literature to which the reader could refer. The most part of the October 2007 issue of IEEE Computer was devoted to that topic [Wolf, 2007].

In the next section the architectural features of the accelerator, of the specific language designed and of its software development environment will be described. Then, the basic physical principles of the simulation and its needed modifications for optimizing the use of the accelerator will be highlighted. Finally, we will see the implementation of the algorithm on the accelerator, taking advantage from the use of a 'dedicated stage' pipeline and the comparison with a few commercial and popular processors showing a clear speed-up. Some remarks explaining the evolution of the project will conclude the paper.

2. The Accelerator. We realized a FPGA-based accelerator connected to a host PC to accelerate the hardest part of a calculus. Our idea refers to a board with a FPGA device (Altera Stratix family) and a Flash memory storing the configuration code; a JTAG port is used to send the program to the Flash memory from the PC. Recently, Altera has made available some boards with these features. These boards can communicate with PC through the network requiring a proper network manager. In this case, both the accelerators and the network processor can be loaded on the same FPGA. The board we bought is equipped with a Stratix 1S40 FPGA component on which a 32 bit RISC CPU, called Nios, is implemented; this processor can be programmed using C language and is supplied with basic libraries to easily handle the on board devices: 2 MB Ram, 8 MB Flash Memory, 16 MB Compact Flash Memory, 100 Mb/s Ethernet Interface, 2 Serial ports.

We designed an accelerating unit that is able to implement different functions (also complex like sin, cos, log, ..., through Taylor series). Thus, it can be used for several applications, also very diversified. Moreover, the instruction set is fully re-programmable according to the particular calculation to be performed.

The designed unit (DPFPA) can exploit the parallelism present in the operations since double precision Floating Point MAC operations can be executed at the same time in the sum and multiply pipelines present onto it. The main part of DPFPA is DPFPP (double precision floating point processor), whose architecture consists of (fig. 2.1):

- 2 accelerating units, independently working;
- a Cache Memory (4 banks), which can store input data and results for the two accelerating units;

A suitable bus devoted to communication between Accelerator and Nios processor ("sub bus") has been also implemented. The Math Unit functional core is a double-precision floating-point ALU, which integrates both an adder and a multiplier operating in a parallel fashion. Both devices are pipelined (9 stages for the adder and 15 for the multiplier) so that high clock rates are achievable. Note that, in the expected applications, accurate coding can minimize the negative effects of such latency.

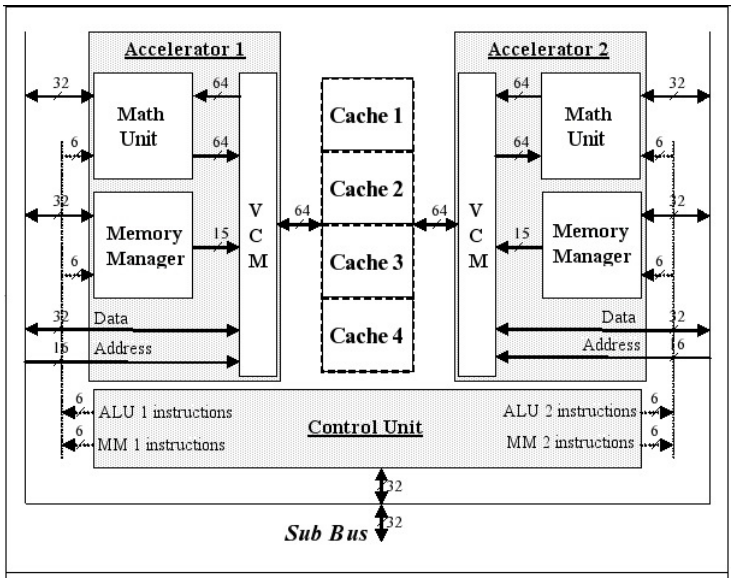

FIG. 2.1. *Architecture of the computational unit implemented onto the FPGA device.*

Together with the adder and the multiplier, the ALU also contains 3 register banks, each able to store 4 double-precision floating point numbers. The banks are each tied to a particular purpose (one is for input data, one for adder results and one for multiplier results).

Like in many similar applications, to make computing elements and storage space independent, a FIFO memory for both inputs and outputs is implemented (there are two FIFO queues on the output since arithmetic results are separated from logical ones).

The ALU operations are encoded in 37-bit words, able to simultaneously trigger either a sum or a comparison, a multiplication, a data fetch, 3 write operations to the internal register banks and the output of a result.

To achieve better performance with our specific task, the operands of the adder can optionally be multiplied by $[-2, -1, 2]$ for the first operand, and $[-1, -0.5, 0.5]$ for the second one. In a similar way, the multiplier result can be doubled, halved or negated without extra clock cycles.

Since feeding the op-codes would require a large and mostly wasted bandwidth (the code is essentially cyclic, so that the same op-codes are executed over and over again) the code sequences are stored in a Microcode Sequencer. This device stores the program sequences in an internal RAM and associates to them a 6-bits op-code (this is much like having a CPU with a micro-programmed control unit whose code can be changed by the application to define a custom instruction set).

The Math Unit itself has no addressing capabilities toward either input or output channels, so every memory I/O operation must be managed by an external device. A Memory Manager was deemed to that task and conceived for a specific application class: those where most computations are performed on data logically organized in three-dimensional matrices. Decoupling the allocation issues from the computing algorithm, the Memory Manager computes the memory addresses from semantic-level inputs, such as addresses in the matrix domain ($X - Y - Z$ coordinates) or offsets between elements (the matrix is supposed to be cyclic, so that e.g. the leftmost element in a row is adjacent to the rightmost element in the same row). This is of extreme importance, since otherwise the same code would require at least a recompilation to be executed on matrices with different sizes.

The internal Control Unit (CU) decodes instructions coming from the host computer and drives the control signals implementing the requested function. It mainly consists of 3 units:

- **Instruction Decode**: selects between data and instructions from host to the DPFPA. Only in the last case it generates proper control signals;
- **Jump Unit**: sets the RAM address to the starting point of the next instruction sequence to be executed;
- **RAM**: stores sequences corresponding to the instruction set for the particular function to implement.

Instructions are 64 bit wide exploiting part of the redundancy present in the IEEE 754 standard of floating point representation, to distinguish them from double precision numbers. Two kinds of instructions have been implemented:

- *Programming instructions* to store in the CU RAM executive sequences.
- *Executive instructions* to perform specific calculations, recalling sequences already loaded.

Programming instructions to store in the CU RAM executive sequences. Executive instructions to perform specific calculations, recalling sequences already loaded.

A great advantage of our approach is that the sequences of an executive instruction are performed in an iterative manner until a new executive instruction will be received by the CU. So, during the execution of the calculus, CU has to decode only few instructions and can save a great amount of time.

3. Programming DPFPP. As previously stated, DPFPP can handle two types of instructions: programming instructions and executive instructions. The former are used to store microcode sequences into the CU RAM, making microcode words to be loaded at the correct address into the RAM of CU. The word of microcode, allows the assertion of needed control signals for each clock cycle.

Each executive instruction allows, on the other hand, the recalling of sequences already stored.

We realised soon, that the sequence development using binary microcode was a very hard and inefficient work. Thus, we chose to design and develop a pseudo-assembly dedicated language that simplifies the sequence writing. The instructions of the language are mapped directly on the hardware and reflect the operation that DPFPP can execute. Table 3.1 shows the list of the instructions and their syntax.

TABLE 3.1
List and syntax of the language instructions.

Instruction Syntax
MOV reg;
SUM c1 op c2 op ; SUM c1 op ; SUM c1 op op SUM op c2 op; SUM op op
MUL c op op; MUL op op; MUL c op;
OUT xx;
INT;

A proper translator was also developed, using standard Unix tools such as Lex and Yacc.

Furthermore, we developed an allocator for an easy generation of the file with the programming instructions that must be sent to the DPFPP. Finally, we designed a simulator, reproducing exactly the DPFPP working and enabling pipeline and register inspection. The simulator also allows the visualisation of the clock cycles needed by a specific sequence or by a set of sequences. Thanks to this tool, we can execute microcode sequences without loading them into the DPFPP; thus, we can simplify the sequence debug, verify the results' correctness and check the performance.

All these tools are integrated in a unique development environment, realised in the Microcomputer laboratory to ease the sequence development. There are four main steps: first, we write and compile source code using an internal editor, then we test the code using the simulator. Finally, we produce the programming file that has to be sent to the DPFPP by using the allocator. More details on the hardware and software for DPFPA are in [Danese et al., 2003].

4. The Considered Problem. Liquid crystals and colloidal suspensions are two examples of systems for which the orientation order has been widely studied through simulations. In both cases interactions among particles play a dominant role. In previous works, we realized a cubic lattice model describing the interactions effects in a dipolar system in presence of an external lattice field [Bellini er al., 2001]: simulations made with this model identified the presence of two phase transitions and the obtained results could in part explain the phenomenon known as "anomalous bi-refringence" as analyzed in [O' Konski et al., 1950][Radeva et al., 1996].

On the other hand, simulations take unacceptably long times even on the most recent and powerful computing systems ranging from a few days up to some weeks depending on the size of the simulated system. The core of the computation is, in fact, the evaluation of the energy since, according to the implemented algorithm (Montecarlo-Metropolis), equilibrium in a system with N particles is reached through a sequence of moves, carried out by randomly selecting a spin, changing its orientation through a random angular displacement and

evaluating the corresponding change in energy. Each move can be accepted or rejected depending on the variation of the energy associated with it [Metropolis et al., 1953]. We simulated lattice systems with particles ranging from a few hundreds up to 100.000 considering only first neighbor interactions, i. e. the interaction between each spin and the six closest ones in the $X+$, $X-$, $Y+$, $Y-$, $Z+$, $Z-$ directions. Periodic boundary conditions were applied [Frenkel et al., 1996]. The associated energy of each dipole due to the presence of an external field oriented toward z axis is:

$$E_{dip} = mom_z(dip) \tag{1}$$

The terms due to the interactions between the considered dipole and each of its first neighbours are:

$$\begin{aligned}
E_{X+} &= 2 * mom_x(dip) * mom_x(X+) + \\
&\quad -mom_y(dip) * mom_y(X+) - mom_z(dip) * mom_z(X+) && (2)\\
E_{X-} &= 2 * mom_x(dip) * mom_x(X-) + \\
&\quad -mom_y(dip) * mom_y(X-) - mom_z(dip) * mom_z(X-) && (3)\\
E_{Y+} &= -mom_x(dip) * mom_x(Y+) + \\
&\quad +2 * mom_y(dip) * mom_y(Y+) - mom_z(dip) * mom_z(Y+) && (4)\\
E_{Y-} &= -mom_x(dip) * mom_x(Y-) + \\
&\quad +2 * mom_y(dip) * mom_y(Y-) - mom_z(dip) * mom_z(Y-) && (5)\\
E_{Z+} &= -mom_x(dip) * mom_x(Z+) + \\
&\quad -mom_y(dip) * mom_y(Z+) + 2 * mom_z(dip) * mom_z(Z+) && (6)\\
E_{Z-} &= -mom_x(dip) * mom_x(Z-) + \\
&\quad -mom_y(dip) * mom_y(Z-) + 2 * mom_z(dip) * mom_z(Z-) && (7)
\end{aligned}$$

where the components of the moments for each dipole are:

$$mom_x(dip) = cos(\theta) * sin(\theta) * cos(\varphi) \tag{8}$$

$$mom_y(dip) = cos(\theta) * sin(\theta) * sin(\varphi) \tag{9}$$

$$mom_z(dip) = cos'(\theta) \tag{10}$$

and θ, φ are the angular co-ordinates of a generic dipole. The overall energy of the dipole is the sum of all these contributes:

$$E_{TOT}[dip] = -0,5 * [E_{dip} - k * (E_{X+} + E_{X-} + E_{Y+} + E_{Y-} + E_{Z+} + E_{Z-})] \tag{11}$$

The global energy in the system is the sum extended on the whole dipolar set.

The simulated system is characterised by an initial random particle distribution not corresponding to that achievable at the equilibrium. This means that the change in the orientation of a dipole will modify the moments and the energy in the others, mainly in the neighbours. These ones, in turn, will influence their respective neighbours and so on, propagating those variations in the moments throughout the lattice. This reflects in energy fluctuations that disappear only after a sufficient number of cycles into which ETOT for each dipole is calculated (equilibration). Only at this point, the Metropolis test on energy variation can be applied. This loop series corresponds to nearly the 85% of the calculation but it consists of only few instructions, so justifying the idea of an accelerator specialized in processing only those operations. To do this, we employed the FPGA technology, which is cheaper and simpler than ASIC in terms of design and test.

However, during the design phase, we considered convenient to realise a more general chip able to accelerate those double precision floating point instructions which can be often found in scientific simulations. This extends the applicability of the DPFPA both to models different to that used (i. e. hexagonal lattices instead of cubic ones) or to completely different fields where high performance computing is mandatory.

5. Energy Evaluation and Implementation. To simplify the readability of the energy calculation on the DPFPA, as it will be described in the following, let's rewrite the expressions reported in section 3. The interaction energy of each dipole can be written as $-\frac{(CT*CT)}{2}$ and the sum on all the dipoles will return the

global energy in the system. CT is the local field generated by the neighbors of the considered dipole and can be expressed as:

$$CT = CTX * SC * k + CTY * SS * k + CTZ * C * k + C \tag{12}$$

where k is a constant depending on the system density and $SC = sin(\theta)cos(\varphi)$, $SS = sin(\theta)sin(\varphi)$, $C = cos(\theta)$, with θ, φ angular co-ordinates of the dipole. CTX, CTY e CTZ are the local components of the field generated by the neighbour dipoles. They are respectively equal to:

$$CTX = (MXX + MX * X + MYX + MY * X + MZX + MZ * X) \tag{13}$$

$$CTY = (MXY + MX * Y + MYY + MY * Y + MZY + MZ * Y) \tag{14}$$

$$CTZ = (MXZ + MX * Z + MYZ + MY * Z + MZZ + MZ * Z) \tag{15}$$

We identify with MXX, MXY, MXZ the local field components generated by the first neighbor dipole in the direction $X-$, and with $MX * X$, $MX * Y$, $MX * Z$ the local field components generated by the first neighbor dipole in the direction $X+$. The other terms due to the effect of dipoles in directions $Y+/Y-$ and $Z+/Z-$ are defined accordingly to the same notation. Moreover the local field, due to the neighbors, changes the components of the dipolar moment. These should be evaluated each time according to the following expressions:

$$mom_x(dip) = CT * SC, \quad mom_y(dip) = CT * SS, \quad mom_z(dip) = CT * C \tag{16}$$

While the SC, SS and C terms are evaluated at each movement, the other terms should be re-calculated for the number of cycles necessary to equilibrate the energy in the system. All these operations, finally, are repeated $M * N$ times with M =cycle number (i. e. 10.000) and N = number of dipoles in the system. This accounts for the high computational weight of the elaboration.

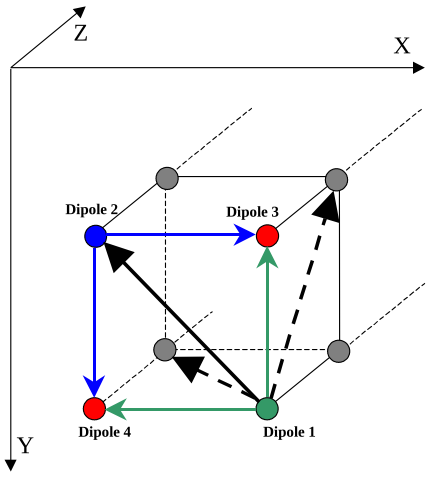

FIG. 5.1. *Diagonal scanning.*

With '*scanning*' we mean the order through which the dipoles are processed during the simulation. The identification of a suitable order can significantly affect the algorithm efficiency in terms of memory access and reuse of data. If we would not use any particular scanning order but if we only would consider dipoles in the same order of memorization (1^{st}, 2^{nd}, 3^{rd}, ...), their elaboration would need 21 input data (SC, SS and C of the moved dipole plus the moments of its six first neighbors), returning the 3 new components of the moment of the considered dipole.

However, if the selection order considers dipoles close to each other toward a diagonal direction, these last ones share two first neighbors whose parameters are no more needed for the elaboration of the new dipole. Fig. 5.1 shows an example of this, since passing from dipole 1 to 2, dipoles 3 and 4 are preserved as first neighbors. This reduces to 15 the number of input data needed, and a correspondent saving in transfer time per each dipole is obtained. Another advantage yielded by the diagonal scanning consists in avoiding calculations. Considering

again fig. 5.1 we note that dipoles 3 and 4 give the following contributes to each component of the local field in dipole 1:

$$MX * X + MY * Y = 2 * mom_x(4) - mom_x(3) \tag{17}$$
$$MX * Y + MY * Y = -mom_y(4) + 2 * mom_y(3) \tag{18}$$
$$MX * Z + MY * Z = -mom_z(4) - mom_z(3) \tag{19}$$

If we now consider the contribute of the same dipoles to dipole 2, the next reached by the diagonal scanning, we find:

$$MXX + MYX = 2 * mom_x(3) - mom_x(4) \tag{20}$$
$$MXY + MYX = -mom_y(3) + 2 * mom_y(4) \tag{21}$$
$$MXZ + MYZ = -mom_z(3) - mom_z(4) \tag{22}$$

The values on the right are obtained by substituting at the terms on left, those values reported in equations in section 3.

Equations 19 and 22 are equal and can be calculated only once. The same considerations are applicable in case of movements toward YZ or XZ direction with a consistent sparing of operations.

Finally, the moment components involved in equations 17–19 for the dipole 1 are also present (with different coefficients) in equations 20–22 and, again, they can be calculated only once (i. e. for dipole 1, storing them in registers from which they can be retrieved later for the next dipole) with a further saving of time.

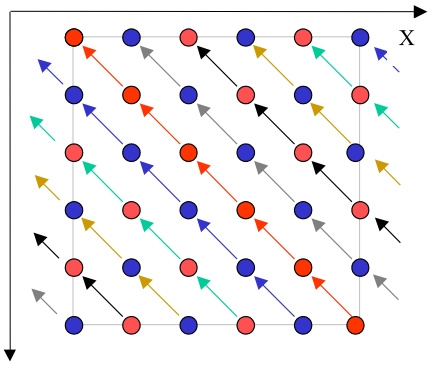

Fig. 5.2. *Diagonals for scanning in XY face.*

The diagonal scanning basically consists of XY movements as shown in fig. 5.2.

The cubic lattice is considered as made by 'slices' and when the last dipole is reached on an XY face, a little movement toward the YZ or XZ direction allows to skip to the next XY slice. In each slice, different starting points can be chosen depending on the odd/even number of dipoles present on the edge of the lattice, but for sake of simplicity we don't want to excessively detail these simulation aspects.

6. Implementation on DPFPA. As previously said, a sequence consists of a microinstruction set and could be identified as a Setup or a Loop sequence. The first problem to deal with is the definition of those operations more frequently executed which should be inserted into the Loop sequence. In the diagonal scanning, the most frequent operation regards the interaction between dipoles located on diagonals belonging to the XY side: thus, the Loop sequence should implement the energy calculation of these dipoles, while the Setup should execute the movements in the XZ or YZ faces of the lattice, through which the algorithm considers the first dipole of the next XY 'slice' and another Loop sequence begins.

According to what said in the previous section, the number of the needed sums is 14 for evaluating CTX, CTY e CTZ (one add is shared with the previous dipole), 3 for CT and 1 more to add these values to the partial total energy obtained from the previous dipoles considered. Thus the adder pipeline is used as its best, if 18 clock cycles are taken. For what concerns multiplications, instead, 6 are needed to calculate CT, 3 for the new moment components of the considered dipole and 1 more for its global energy. Thus, 10 multiplications are required. Let's see how these operations could be efficiently implemented.

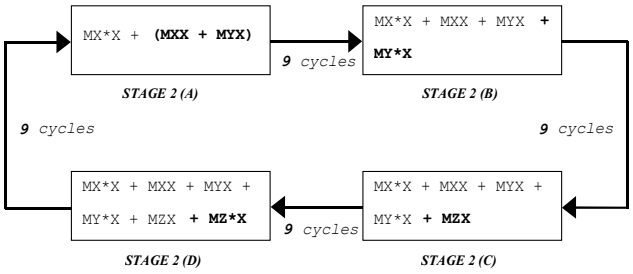

FIG. 6.1. *Stage 2 in the adder pipeline during the Loop phase*

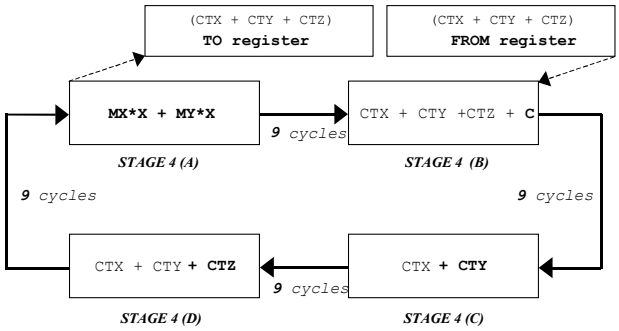

FIG. 6.2. *Stage 4 in the adder pipeline during the Loop phase*

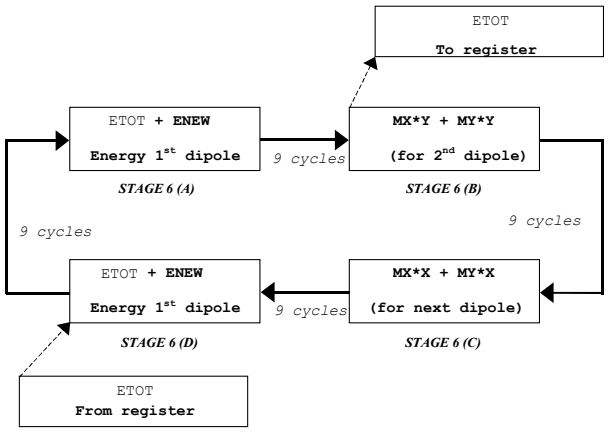

FIG. 6.3. *Stage 6 in the adder pipeline during the Loop phase*

6.1. Adder Unit. Each stage is considered as an independent register containing the partial result which can be stored every L clock cycles (L is the pipeline length). The Loop sequence evaluates the energy of dipoles considered in the XY direction: 4 stages of adder pipeline were devoted to calculate CTX, CTY, CTZ and CT. In fig. 6.1, the second pipeline stage devoted to the calculation of CTX is shown, with the particular calculation highlighted in bold in each of the four sums needed. In the first step, the term in parentheses is 'shared' with the previous dipole considered and does not need to be re-calculated (see previous section). Each partial result is available only when it has run across the whole pipeline i. e. after 9 clock cycles and the complete value of CTX is available after 36 clock cycles. Then the stage proceeds to evaluate the CTX for the next dipole. The same considerations can be made for CTY and CTZ. The calculation of CT is implemented in the stage 4, which works again for 36 clock cycles. The CTX, CTY, CTZ values used in this case are those coming from the multiplier where they have been multiplied by SC, SS and C. Since the calculation of CT takes less than 36 cycles, the first stage is used to calculate that value shared with the next dipole:

Stage	*Term*	Cycles 1 - 9	Cycles 10 - 18	Cycles 19 - 27	Cycles 28 - 36
9	CT2	**CTX2+CTY2 (n-3)**	CTX2+CTY2+ **CTZ2 (n-3)**	**mx*y + my*y**	CTX2+CTY2+CTZ2 + **C2 (n-3)**
8	CTZ2	mx*z+my*z + **(mxz + myz) (n-1)**	Mxz+myz+mx*z+ my*z + **mzz (n-1)**	mxz+myz+mx*z+ my*z+mzz+ **mz*z (n-1)**	**mx*z + my*z (n+1)**
7	CTY2	mxy+myy+mx*y+ my*y + **mzy (n-1)**	mxy+myy+mx*+ my*y+mzy+ **mz*y (n-1)**	**(mxy+myy)+ mx*y (n-1)**	mx*y+myy+mxy + **my*y (n+1)**
6	ETOT	Etot + **Enew1 (n-6)**	**mx*y + my*y**	**mx*x + my*x**	Etot+ **EneW2 (n- 5)**
5	CTX2	mxx+myx+mx*x+ my*x + **mzx (n-1)**	mxx+myx+mx*x+ my*x+mzx + **mz*x (n- 1)**	**(mx*x+mxx)+ myx (n+1)**	mx*x+myx+mxx+ **my*x (n+1)**
4	CT1	**mx*x + my*x**	CTX1+CTY1+CT Z1 + **C1 (n- 4)**	**CTX1 + CTY1 (n-2)**	CTX+CTY+**CTZ (n-2)**
3	CTZ1	mx*z+my*z+mxz+ myz+mz*z + **mzz (n-2)**	**mx*z + my*z (n)**	mx*z+my*z+ **(mxz + myz) (n)**	mx*z+my*z+mxz+ myz + **mz*z (n)**
2	CTX1	**mx*x + (mxx+myx) (n)**	mx*x+mxx+myx + **my*x (n)**	mx*x+my*x+mxx+ myx + **mzx (n)**	mx*x+my*x+mxx+ myx+mzx+ **mz*x (n)**
1	CTY1	**(mxy+myy) + mzy (n)**	mxy+myy+mzy + **mx*y (n)**	mxy+myy+mx*y+ my*y + **mzy (n)**	mxy+myy+mx*y+ my*y+mzy+ **mz*y (n)**

FIG. 6.4. *Stage 6 in the adder pipeline during the Loop phase.*

Therefore the partial value of CT is saved in a register from which it will be retrieved during stage 4B (fig. 6.2). To optimise the use of the pipeline the remaining stages are devoted to implement the same calculations for a second dipole, so as to process 2 dipoles in 36 clock cycles. This corresponds, as previously seen, to an optimal use of the adder. Finally, stage 6 is devoted to add to the global energy value E_{TOT}, those two energy contributes (E_{NEW}) calculated in the other stages of the pipeline up to this moment (fig. 6.3). Basically it works in the same way as stage 4, including two sums shared with the successive elaborated dipoles (again to optimise the pipeline use). Even though, during the 36 clock cycles all the sums needed for the energy of two dipoles have been performed, the dipoles involved in the elaboration are more than 2. In fact, while the adder is evaluating CTX, CTY and CTZ for the two dipoles, it is not possible to determine at the same time the correspondent CT terms, since the previous calculations (CTX, CTY and CYZ) should be completed and they should also be multiplied by $SC1 * k$, $SS1 * k$ and $C1 * k$ (k is a suitable constant depending on the system density). Therefore the CT term really computed refers to the previous Loop sequence. This means that while CTX, CTY and CTZ for dipoles (n) and $(n+1)$ are evaluated, the CT terms refer to $(n-3)$ and $(n-4)$ dipoles and the E_{NEW} corresponds to the couple $(n-5)$ and $(n-6)$ previously started. Moreover, also the couple $(n-1, n-2)$ is subjected to a partial elaboration making the pipeline always working.

This configuration brings a consistent level of parallelisation in the execution of the algorithm. Fig. 6.4 shows the complete set of operations calculated during the 36 clock cycles of each Loop sequence. Per each stage and clock cycle, the effective sum performed is reported in bold.

6.2. Multiplier Unit. This unit executes the multiplications needed in the terms that must be added, i. e. 10 per each of the two dipoles of the adder unit (globally 20) and in a sequential way. To synchronise the operations in the multiplier pipeline with those of the adder, the length of the pipeline (15 stages) is extended to 18 by adding three NOP (no operation) cycles: this means that in 36 clock cycles the multiplier works effectively for 30 cycles, a time sufficient to execute the required 20 products, without loosing the synchronisation with the correspondent terms in the adder unit. Fig. 6.5 describes the operations performed together with the output from the pipeline at that instant, per each clock cycle. In parenthesis the order number is reported of the dipole to which the calculation refers: n is the dipole for which the calculation of the energy is initiated in the current sequence. At the end of each Loop sequence the pipeline outputs new moments and energy of the dipole couple which started the evaluation 3 sequences before. Fictitious products have been inserted when needed to force the pipeline going one step beyond.

7. Results. The whole system has been tested by executing Montecarlo simulations of different size lattices ($4 < ND < 100$, where ND is the number of dipoles on each side of the cubic lattice).

Output	Cc	OPERATION		Output	Cc	OPERATION
SC1*K (n-2)	1	Fictitious product		---	19	SC1*K (n)
C1*K (n-2)	2	Fictitious product		---	20	C1*K (n)
SS1*K (n-2)	3	CTX1*(SC1*K) (n-2)		CTX1*SC1*K (n-2)	21	SS1*K (n)
-1/2*CT1^2 (n-5)	4	CTY1* (SS1*K) (n-2)		CTY1*SS1*K (n-2)	22	-1/2*CT1^2 (n-3)
CTX2*SC2*K (n-3)	5	SC2*K (n-1)		SC2*K (n-1)	23	CTX2*(SC2*K) (n-1)
C1*1 (n-2)	6	NOP		C1*1 (n-2)	24	NOP
C1*1 (n-2)	7	C1*1 (n-2)		C1*1 (n-2)	25	C1*1 (n)
C1*1 (n-4)	8	Fictitious product		---	26	C1*1 (n-2)
CTY2 * SS2 * K (n-3)	9	SS2*K (n-1)		SS2*K (n-1)	27	CTY2*(SS2*K) (n-1)
CT1*C1 (n-6)	10	CT2*SC2 (n-5)		CT2*SC2 (n-5)	28	CT1*C1 (n-4)
CT1*SC1 (n-6)	11	CT2*SS2 (n-5)		CT2*SS2 (n-5)	29	CT1*SC1 (n-4)
CT1*SS1 (n-6)	12	NOP		CTZ1*C1*K (n-2)	30	NOP
CT1*SS1 (n-6)	13	CTZ1*(C1*K) (n-2)		CTZ1*C1*K (n-2)	31	CT1*SS1 (n-4)
C2*1 (n-5)	14	CT2*C2 (n-5)		CT2*C2 (n-5)	32	C2*1 (n-3)
---	15	-1/2*CT2^2 (n-5)		-1/2*CT2^2 (n-5)	33	Fictitious product
C2*1 (n-3)	16	C2*1 (n-1)		C2*1 (n-1)	34	C2*1 (n-1)
CTZ2*C2*K (n-3)	17	C2*K (n-1)		C2*K (n-1)	35	CTZ2*(C2*K) (n-1)
---	18	NOP		SC1*K (n)	36	NOP

FIG. 6.5. *Operations performed in the multiplication pipeline during 36 clock cycles.*

Performance has been evaluated as speed-up respect to the execution of the same simulation on an Intel P4 processor with 1GB Ram memory; also FPGA occupation was used as a performance parameter. Simulation code was written in C language and optimized using Microsoft Visual C++ environment. The Accelerator elaboration times were measured by means of the clock counters implemented in the interface between Nios and the coprocessor previously described.

In fig. 7.1 we show the performance as speed-up factor respect to two Intel P4 processors with 3 GHz and 1.7 GHz frequency respectively, calculating the dipolar energy of the simulated system. That computational core is repeatedly executed $k*N*10000$ times where k is the coefficient responsible for the interaction settlement (equilibration) and N is the dipole number: this gives reason of the high computational load which can lead (for big particle systems, e. g. 100000 dipoles) to wait a lot for results, if the simulation should be performed on a PC. The speed-up factor is increasing for the 1.7 GHz processor due to cache effect, while for the most performing Intel processor (3 GHz) sets around 2.

Considering the size of the FPGA we used, other 2 accelerating units could be implemented, we can reasonably state that a speed-up factor equal to 4 can be achieved in case of a "full" implementation on the FPGA component we chose (Stratix EP1S40). Further speed-up could be obtained if other components of the Altera's family (Stratix2 or Stratix3 now available) should be employed.

The cost of each board we bought was nearly $1200: this represents an important indication when predicting trade-off between a cluster of workstations versus a cluster of FPGA based accelerators. In practice, our work indicates that each FPGA unit gives a computational power 4 times greater, only doubling costs with respect

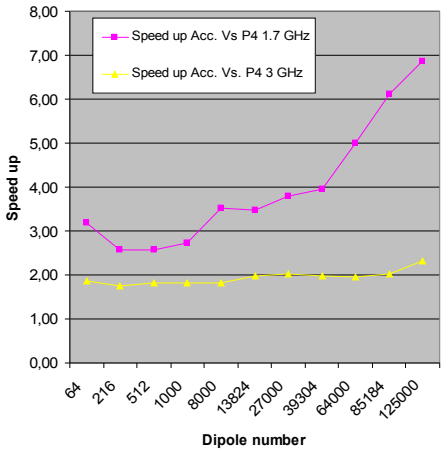

FIG. 7.1. *Speed-up of the FPGA based accelerator with respect the P4 Intel processors.*

to a computational unit in a PC cluster, providing the scientist with a COTS desktop computing system on which he/she can run simulations.

8. Conclusions. Simulations allow the analysis of a physical system, even complex, without experimental measures or, sometimes, to confirm what was experimentally observed. In certain situations such as microscopic systems, simulations represent the simplest if not the only way to quickly foresee the behaviour of a particle system in different environmental conditions. The high number of variables involved together with complex interaction laws often make simulation times unacceptably long. Finally, several of the requested calculations ask for double precision floating point arithmetic, further increasing the computational power needed.

In this paper, we have shown how an application specific architecture (DPFPA) specifically designed for this kind of problems and based on FPGA technology could represent a good compromise between processing capabilities and low costs. DPFPA can be programmed with a dedicated language to execute complex floating point functions and it is equipped with a suitable software development environment. We executed the dipole energy calculation through the simulator, achieving, thanks also to the new scanning algorithm purposely designed and here described, a performance twice as that of a last generation Personal Computer but can be easily "extended" to 4.

A further improvement could be achieved by a full custom ASIC implementation of the Accelerator which is not justified at a prototyping level while it allows a large scale manufacturing with reduced costs. This would make available several computing units connected in cluster fashion by means of a point to point network, providing the user with a great computing power.

REFERENCES

[BELLETTI F., et al.,] "An adaptive FPGA computer", IEEE Computing in Science & Engineering, vol. 8(1), January-February 2006, pp. 41-49.

[BOGHOSIAN B., et al.] "Scientific applications of grid computing", IEEE Comp. in Science & Engin., vol. 7(5), Sept.-Oct. 2005, pp. 10-13.

[BUELL D., et al] "High Performance Reconfigurable Computing", IEEE Computer. March 2007, pp. 23–26.

[COWEN C. P. et al.] "A reconfigurable Montecarlo clustering processor (MCCP)", FPGAs for Custom Computing Machines, 1994. Proceedings. IEEE Workshop on 10–13 April 1994, pp. 59 – 65.

[CRUZ A., et al.] "A Special Purpose Computer for spin glass models", Computer Physics Communications, vol. 133, n° 2-3, 2001, pp. 165–176

[DANESE G., et al.] "A development and simulation environment for a floating point operations FPGA based accelerator", Proc. of DSD '03 – 3rd Euromicro Symposium on Digital System Design, Belek (Turkey), September 2003, pp. 173-179.

[DANESE G., et al.] "An application specific processor for Montecarlo simulations", IEEE conference on Parallel and Distributed Processing (PDP07), Naples, February 2007, pp. 262-269.

[DANESE G., et al.] "Field induced anti-nematic ordering in assemblies of anisotropically polarizable spins", Europhysics Letters 55(3), pp. 362-368, 2001.

[DONGARRA J., et al.] "High-Performance Computing: Clusters, Constellations, MPPs, and Future Directions", IEEE Comp. in Science & Engin., vol. 7(2), Mar–Apr 05, pp. 51–59.

[FRENKEL D., et al.] "Understanding computer simulations", Acad. Press New York, pp. 28–30, 1996.

[GOKHALE M., et al.] "Monte Carlo radiative Heat Transfer Simulation on a reconfigurable Computer", Proc. of FPL 2004, LNCS 3203, pp. 95–104, Springer-Verlag ed.
[HANKEL J., et al.] "Taking on the embedded system design challenge", IEEE Computer, April 2003, 35–37.
[HERBORDT M. C.] "Achieving High Performance with FPGA-Based Computing", IEEE Computer. March 2007, pp. 50-57.
[MARSH P.] "High performance horizons", Computing & Control Engineering Journal, vol. 15(6), December–January 2004/2005, pp. 42–48.
[METROPOLIS N., et al.] "Equation of State Calculations by Fast Computing Machines", Journal of Chem. Physics, 21, (1953), pp. 1087–1092.
[MONAGHAN S., et al.] "Reconfigurable special purpose hardware for scientific computation and simulation", Computing & Control Engineering Journal, Vol. 3, Issue 5, Sept. 1992, pp. 225–234.
[O' KONSKI C. T., et al.] "New method for studying electrical orientation and relaxation effects in aqueous colloids: preliminary results with tobacco mosaic virus", Science, 111, pp. 113–116 (1950).
[POSTULA A., et al.] "The design of a specialized processor for the simulation of sintering", EUROMICRO 96. 'Beyond 2000: Hardware and Software Design Strategies', Proc. of the 22nd EUROMICRO Conference, 2–5 September 1996, pp. 501 – 508.
[RADEVA T., et al.] "Electric Light Scattering from Polytetrafluorethylene Suspensions", Coll. And Surf. 119, 1 (1996).
[WOLF W.] "A decade of hardware/software codesign", IEEE Computer, April 2003, 38–42
[WOLF W.] "The embedded systems landscape", IEEE Computer, October 2007, 29–33.
[ZHANG G. L., et al.] "Reconfigurable Acceleration for Montecarlo based financial simulation", Proc. of FPT05, IEEE Conference on field programmable technology, Singapore Dec. 11–14 2005.

Edited by: Pasqua D'Ambra, Daniela di Serafino, Mario Rosario Guarracino, Francesca Perla
Received: June 2007
Accepted: November 2008