APEmille: A PARALLEL PROCESSOR IN THE TERAFLOPS RANGE

A. Bartoloni, C.Battista, S.Cabasino, M.Cosimi, U.D'Agostini, F.Marzano, E. Panizzi, P.S.Paolucci, F.Rapuano, W.Rinaldi, R.Sarno, G.M.Todesco, M.Torelli, P.Vicini
INFN, Sezione di Roma and Dipartimento di Fisica, Universita' di Roma,
La Sapienza, 00185 Roma, Italy.

N.Cabibbo
INFN, Sezione di Roma and Dipartimento di Fisica, II Universita' di Roma,
Tor Vergata, 00173 Roma, Italy.

A. Fucci
CERN, Geneva, Switzerland

R.Tripiccione
INFN, Sezione di Pisa, 56100 Pisa, Italy

Abstract

This proposal describes APEmille, a 3-D SIMD scalable parallel processor in the Teraflop range. This machine is very efficient for LGT simulations as well as for a broader class of numeric applications requiring massive intensive floating point computations. The proposed architecture satisfies the requirements for a massively parallel scientific processor described in [1].

The APEmille architecture grows from APE100[2]. Several new features characterise this evolution. In particular local addressing capabilities are added in all computing nodes. APEmille also exhibits a higher degree of integration with a network of workstations acting as a global host system.

An APEmille system in the Teraflops range will be completed in three-four years, provided the necessary resources are available.

The architecture proposed in this paper is being currently simulated and evaluated.

Architectural Overview

An APEmille machine can be viewed as a 3-D processing grid with periodic boundaries, composed of Processing Nodes. Each Processing Node is directly connected to its own 6 near-neighbours through synchronous data communication channels.

The Processing Nodes are optimised for single precision floating point arithmetic, but integer and double precision floating point operations are supported, albeit with reduced performance. The SIMD paradigm (groups of nodes which execute the same instruction on different data) is complemented with a local addressing feature. Therefore all Processing Nodes may access their own local memories using different local addresses. This new feature is the most important extension to the present APE100 architecture, opening a path for coding of algorithms which could not be efficiently implemented on APE100 machines. The local addressing capability is a valuable addition to the power, already present in the APE100 architecture, of the local conditional statements (Where(local condition)... Endwhere) and of global conditions derived by the set of local conditions (e.g. If(All(local condition)) ... Endif). A further enhancement to the APE100 architecture is represented by more general data routing capabilities among non-first-neighbour nodes.

While the computational kernel of APE100 is built using replicas of three kinds of boards (Controller Boards, Processing Boards and Communication Boards) an APEmille computational engine is based on multiple instances of just one Processing Board (PB). This design solution implies scalability and engineering advantages in comparison with the APE100 arrangement.

Each PB integrates all system functionality: flow control, data processing, internode communication and host<->APEmille I/O. The host consists of one or more networked workstations, each controlling a group of PBs.

An APEmille machine can be partitioned into smaller SIMD machines, each comprising one or more Processing Boards, executing different instructions on each partition.

Each host in the network, using a high performance communication channel, maps the memories of a portion of APEmille (up to 128 nodes) on its own bus. The close integration with a network of host workstations allows a high input/output bandwidth with disks and peripherals in the range of 100 MByte per second per I/O device. Close integration of APEmille with standard workstation also adds the flexibility needed to customise the I/O system to the requirements of specific applications.

On each APEmille PB a CPU takes care of the program flow and drives the communication protocol with the host network. The CPU broadcasts a VLIW (Very Long Instruction Word) instruction to 8 Processing Nodes located on the PB. At the same time, as a result of its calculations, the CPU broadcasts a Global Address to the Processing Nodes. The Nodes can access their own local memory using this Global Address either directly or after adding a local offset, thus producing a local address.

The 8 Processing Nodes of a Board are arranged in a cubic lattice (a 2x2x2 topology). Each processing node has an arithmetic and logic unit including 2 floating point adders and 2 floating point multipliers, a large multiport register file plus a memory controller with address generation capability, interfacing to local memory. Integer and

bitwise operations and local addressing are features of the Processing Nodes which had not been included in APE100.

The first APEmille systems, based on presently available Dynamic Ram and VLSI ASIC technologies, will adopt a 50 MHz, 200 MFlops processor. We expect however that the clock frequency may eventually increase up to 100 MHz. A 4096 100 MHz nodes system would have a peak performance of 1.6 TeraFlops.

According to our application requirements, the size of the local memory of each Processing Node will range from 2 to 8 MWord. Through a Communication Device each Processing Node directly accesses the data on its own 6 neighbouring Nodes. Some routing capabilities allow (slower) access to far-away nodes.

At the hardware level, the Processing Boards have 6 data pathways connecting with other Processing Boards, and 1 data pathway to connect the PB with the Host Network through a Host Interface.

The Processing Board

The Processing Board hosts the Control System, including a CPU, 8 Processing Nodes, a Communication Device and all required memories.

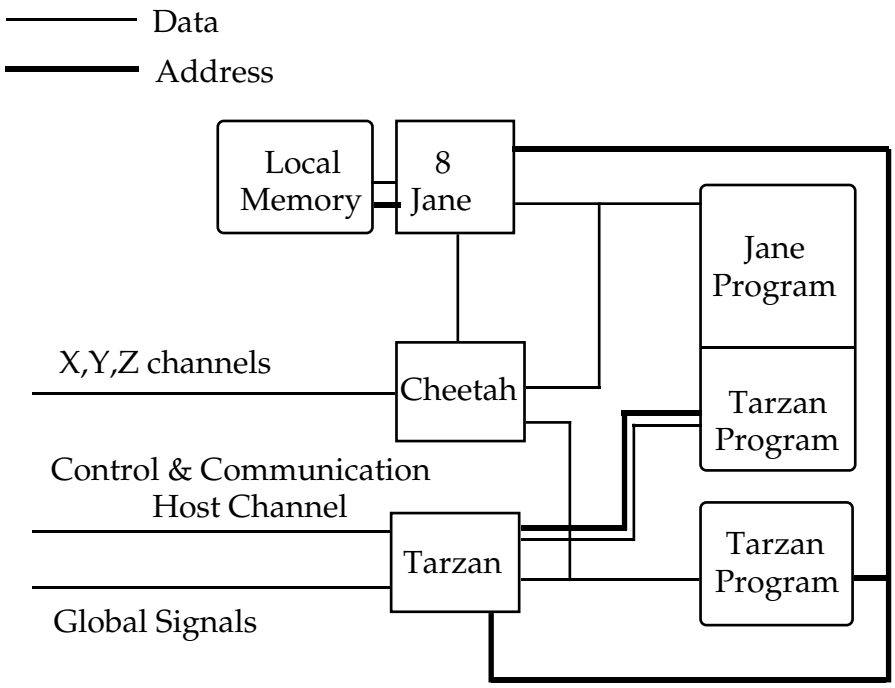

Figure 1 APEmille Processing Board

The CPU, named TARZAN is connected to the Host, to its own data memory, to an instruction memory and to the Communication Device. It also drives the Address Bus.

Each Processing Node is composed of a floating point processor (called JANE) which drives its own Local Memory, receives Instructions and Global Addresses from TARZAN and communicates with the CHEETAH Communication Device.

CHEETAH interfaces the Processing Nodes inside the Board and connects them to the Processing Nodes placed on other Boards. CHEETAH is implemented as a nibble oriented bit sliced architecture. The Communication Device also has a channel

connected to the CPU, and through this pathway, to the Host. Through this pathway the Host loads the program and data memories of the floating point processor.

All the CPUs and all the Nodes of a SIMD machine partition execute the same instruction. Each PB is connected with a few Global Signals to manage Synchronisation, Exceptions and Global Flow Control Conditions. Each Processing Board interfaces the Host through a dedicated synchronous channel going from the CPU to the Host Interface.

The Control System

APEmille lives in two main modalities: Program Execution and Host I/O. The TARZAN CPU manages both of these operating states.

During Program Execution TARZAN fetches and executes its own instruction word and gives the same Program Address to the JANE's Program Memories. In fact, each word of the program memory is composed of two parts. The first part is fetched to be executed by the CPU, while the second part of the instruction word is broadcast for execution to the 8 Processing Nodes. TARZAN handles the program flow, receives the collection of the status of local logical conditions generated by the Nodes and distributes a Global Data Memory Address to all the Processing Nodes.

When APEmille performs Host I/O operations TARZAN manages the APEmille side of the APEmille<->Host I/O protocol. TARZAN is connected to the Host Interface and, directly or through the CHEETAH Communication Device, accesses all memory locations of the Processing Board.

The Processing Node

Each Processing Node is composed of a JANE floating point processor, attached to a RAM local memory.

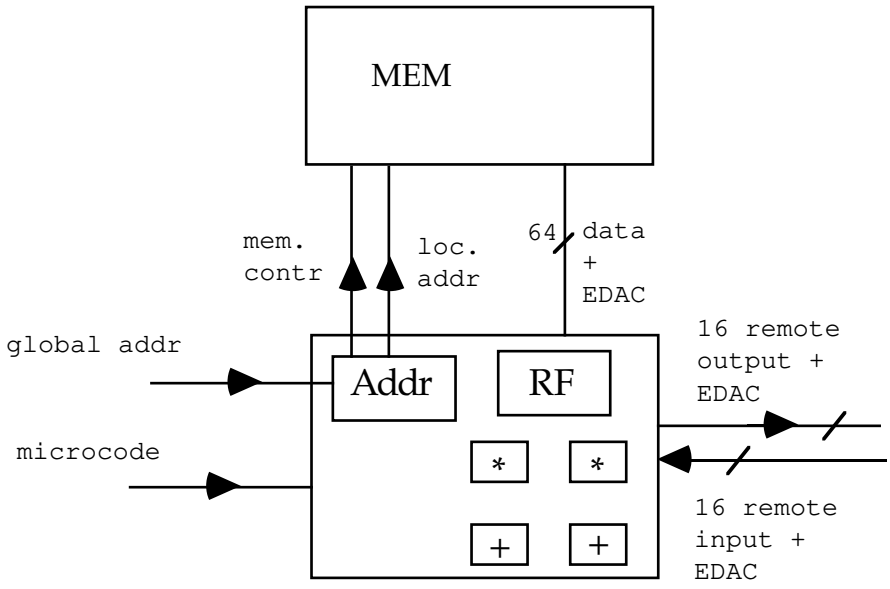

Figure 2 APEmille Processing Node

It receives a flow of instructions and addresses from the CPU of the Control System (not drawn in the figure below).

A CHEETAH Communication Device is connected to all 8 nodes of a PB. It establishes connections among the nodes of its own board (using channels internal to the PB), with adjacent PB's (through channels connected to the CDs of other PBs) and with the Host Interface through a dedicated channel directed to the CPU. In the following paragraphs we will discuss these interconnections.

Each Processing Node generates Status Signals (Global conditions, Exceptions etc.). These Status Signals are collected into a Global Status Return managed by the Control System. This connection allows the Control System to execute flow control instructions based on simultaneous logical conditions produced by the set of Processing Nodes.

The instruction word read by JANE specifies the set of addressed location inside the large multiport Register File internal to JANE, and controls the Arithmetic Devices inside the processing node. Moreover it specifies Local Conditional Operation, Local Addressing functions, and Special Arithmetic function calculations to be performed by JANE. In the following section we will discuss the Processing Node Local Memory, the Internode Communication and the architecture of the Floating Point kernel.

The Node Memory

While the internal clock of the floating point processor could range between 50 and 100 MHz it is hard today to adopt a similar speed for a large dynamic memory bank. At present (1994), we can imagine to use memories with an operating cycle (page mode) of 25 MHz, while it is possible to expect 50 MHz clock memories in the near future (synchronous DRAM).

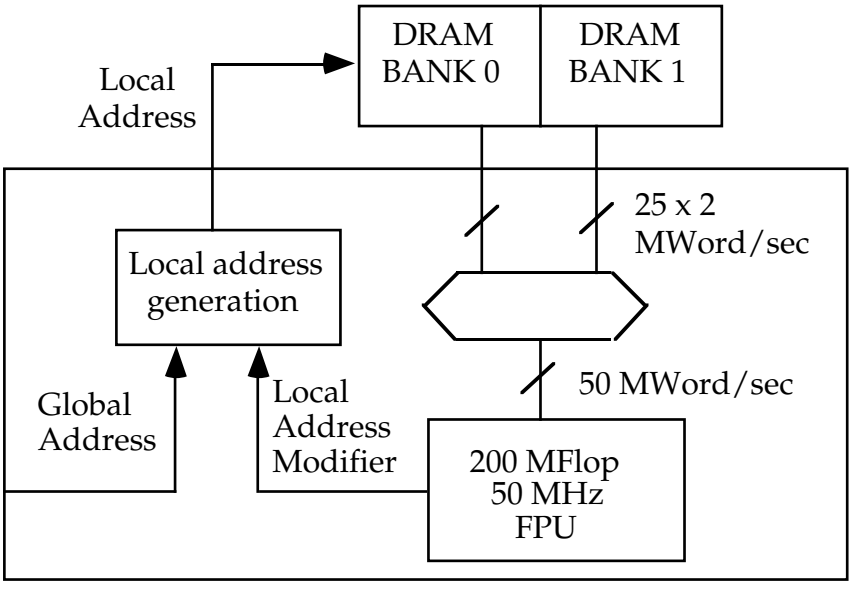

Figure 3 Memory Architecture for a 50 MHz Floating Point Unit coupled with 25 MHz memories

A key architectural parameter in a number cruncher is the balance between arithmetic performance and bandwidth to/from memory. We stick to a figure of 4/1

for the ratio between floating-point-ops per unit time and data-word accesses per unit time. A large value (> 1) of this parameter characterizes architectures optimized for computationally intensive applications, while standard high performance CPU's tend to have lower values, in order to support more general classes of programs. This is made possible by the use a cache system to support the processor bandwidth. In the suite of APE architecture (including APEmille) the role of the cache system has been played by a large multiport Register File, inside each floating point processor.

In APEmille, we propose to use a double-word bus (2 * 32 bits) between Register File and the Local Memory. In this way the memory banks operate at a frequency half that of the floating point devices (two single precision words are accessed in each memory cycle). This scheme is shown in the picture (40 ns Fast Page Mode devices are considered):

We will adopt memory packages and technologies driven by market standards, in order to reduce the technical and financial risks connected with non standard memory components. Moreover we will design the interface with memory modules maintaining a degree of flexibility adequate to follow the evolution of the memory market.

The total memory of a massively parallel computer with a few thousand nodes, each one with several Megawords, is in the GigaWords range. An Error Detection and Correction (EDAC) strategy should be adopted to reduce the error rates to manageable levels. This also enhances the reliability of internode communications. We plan to use independent EDAC systems for each 32 bit word. The total chip count per node therefore increases from 16 to 20.

As an example, we can use twenty 16 MBits chips, organised as 4 * 4 MBits. In this way we can arrange 8 MegaWords (1 word = 40 bits: 32 Data + 7 EDAC) per node. For application requiring a smaller memory 2 MWord local memories can be built using twenty 4 Mbit chips.

A new feature of the APEmille architecture in comparison with the APE100 architecture is the ability to perform local addressing, exploiting a SIMAMD architecture (Single Instruction, Multiple Address, Multiple Data) instead of a SIMD one.

We propose the following scheme:

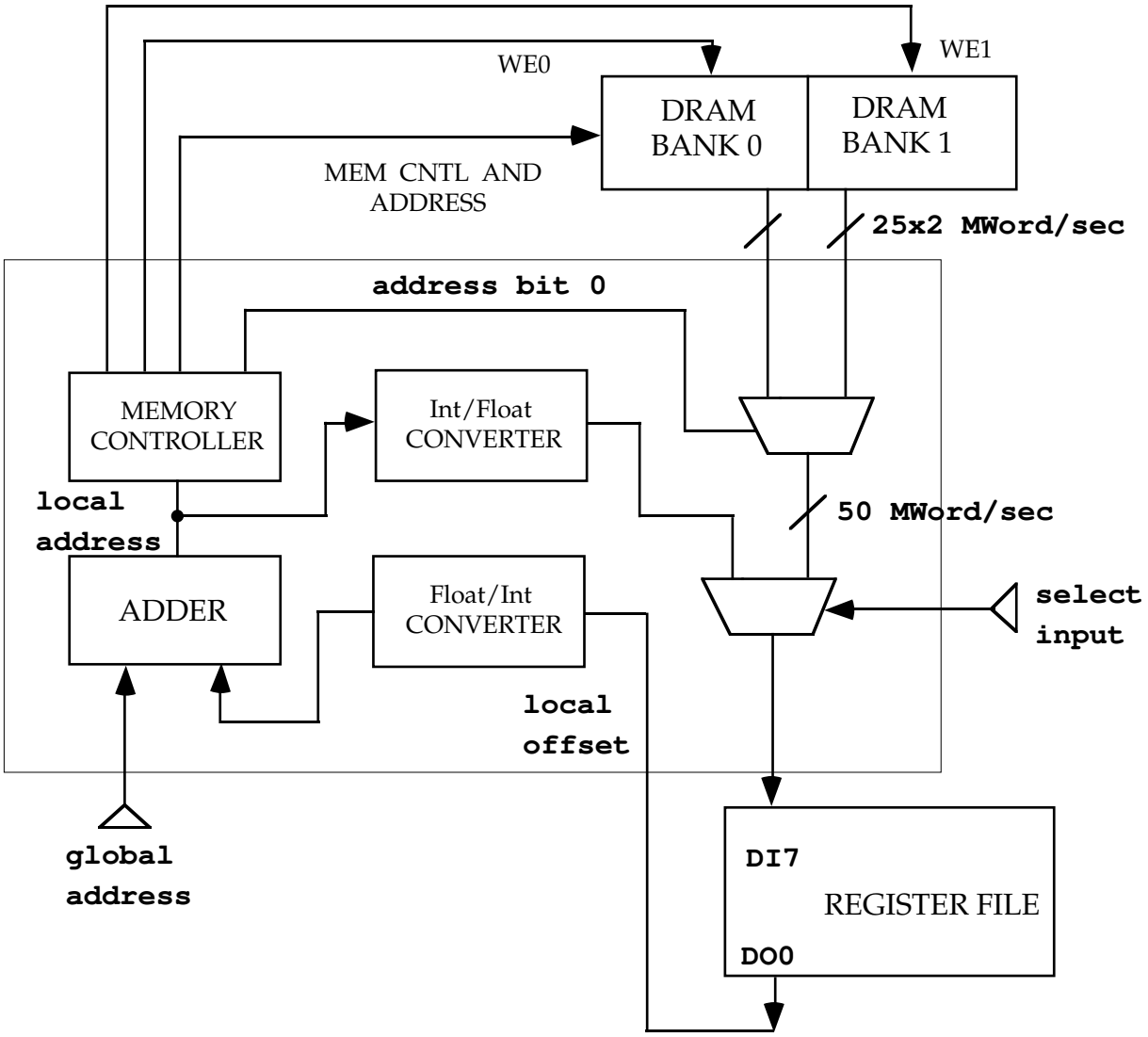

Figure 4 Local address generation block

The memory controller is integrated in each node. It generates the local address as the sum of a global address computed from the CPU of the Control System and a local offset read from the Register File and converted from floating point to integer format. A default offset can be taken equal to zero from the zero register. Moreover it will be possible to read the resulting address in a register for further computations.

Internode Communication

The ratio between the performances in local memory access and internode communication as well as the communication topology and flexibility are characterising properties of any massively parallel architecture. Most of our application requirements fix the maximum ratio between the local memory access bandwidth vs. internode communication bandwidth to 4 and are satisfied by a 3D first neighbour communication topology with all the nodes accessing the same communication direction. The schema in fig. 5 shows the connection between the 8 Processing Nodes and the Communication Device inside each Processing Board and depicts the Data Channels linking the Communication Device with the other PBs and with the Host Network.

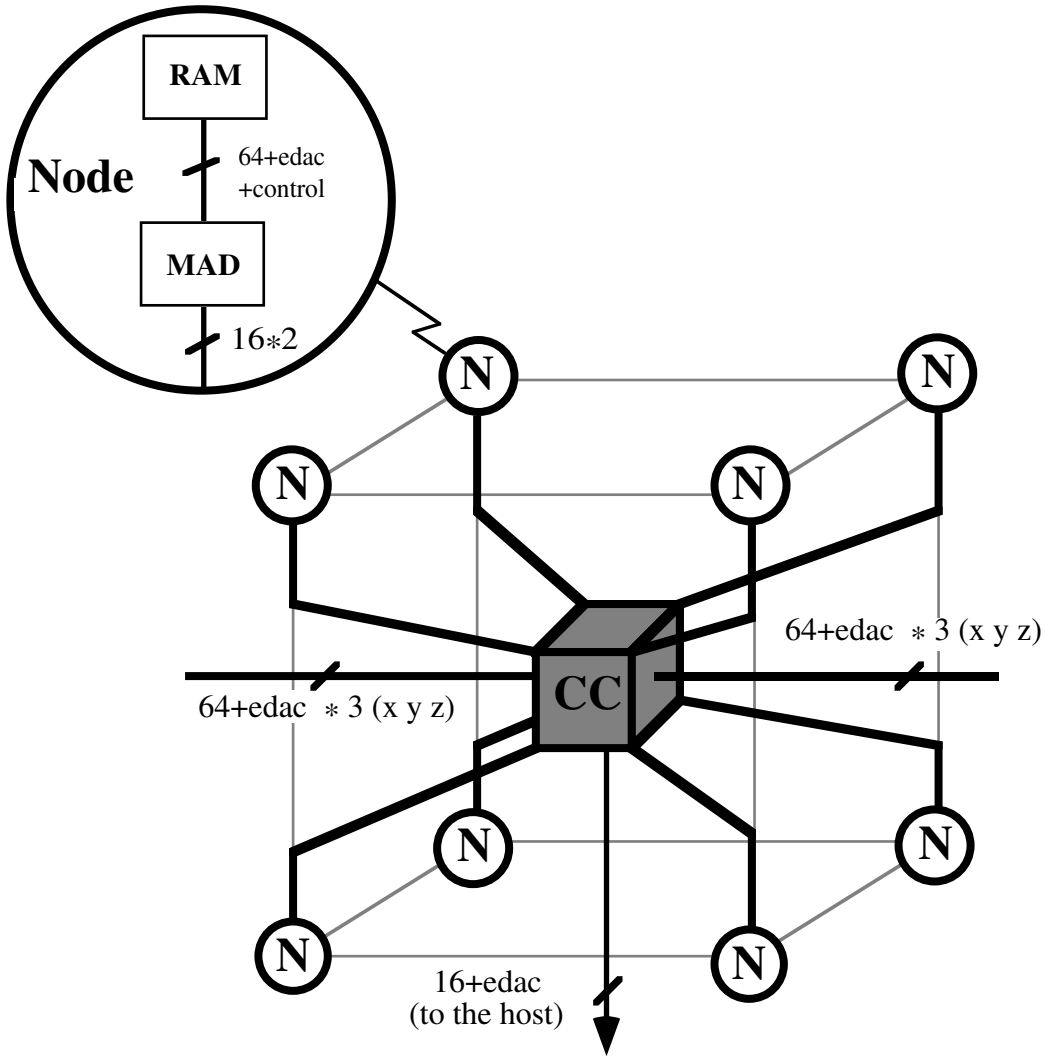

Figure 5 Internode Communication

Each Node prepares the data to be transferred to the Communication Device. If the memory channel is capable to transfer 2 words (2*32 bits + EDAC) per clock cycle, the Processing Node sends 1/2 of a word to the Communication Device which will

reroute it, let us say to the right. At the same time the Processing Node receives the same number of bits coming from the left direction.

The bandwidth needed for board-to-board communications is satisfied by one 64 bit channel running at the same speed as the memory busses. Indeed, if the Processing Nodes of each Processing Board are cubically arranged, only four of them need to use external connections while the other four Nodes receive data from another Processing Board. Therefore the Communication Device assembles 64 bits of data for the external connection assembling data of 4 nodes, and manages the other communications internal to the board. The communication device has to generate a new EDAC to guarantee reliable communications between PBs. A block diagram of the connections of the Communication Device is shown in the next picture:

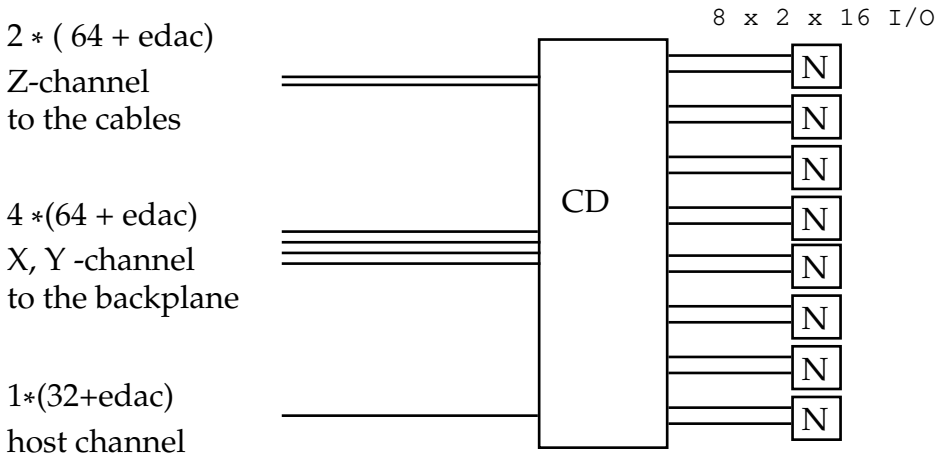

Figure 6 Processing Board interconnections

This interconnection scheme differs from that adopted in APE100.
In fact, on APE100, the connections follow the scheme depicted below:

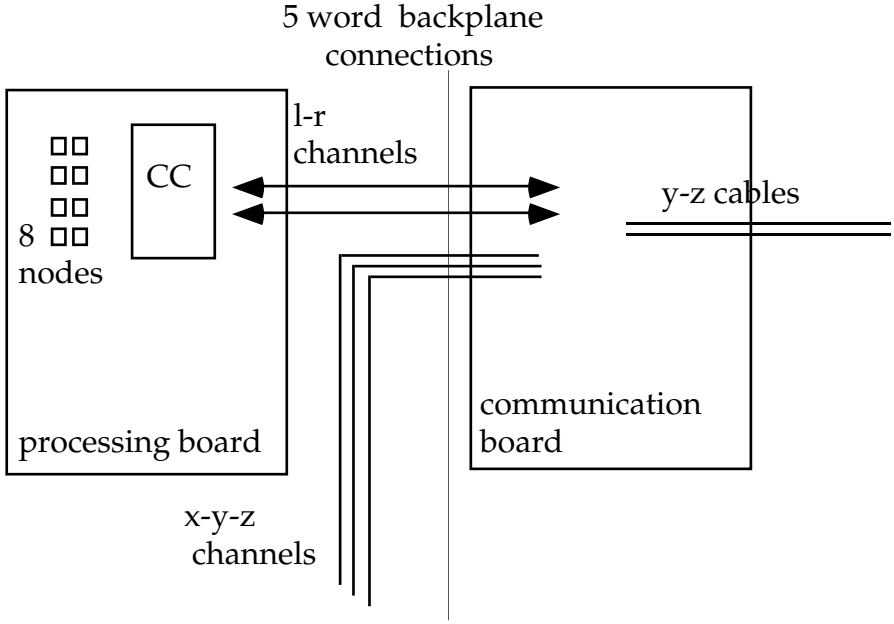

Figure 7 Processing Board interconnections in APE100

The 8 nodes of an APE100 machine are connected to a Communication Controller and the Communication Controller has two channels toward the backplane.

This interconnection scheme allowed a great interconnection flexibility, and the partition of the project into processing board and communication board was easy. However, with improved technology being available, we prefer to imagine a board provided with all the communications facilities, thus enclosing in a single board both processing units and communication logic.

Following this idea, the number of connections toward the backplane can be decreased as shown in figure 8:

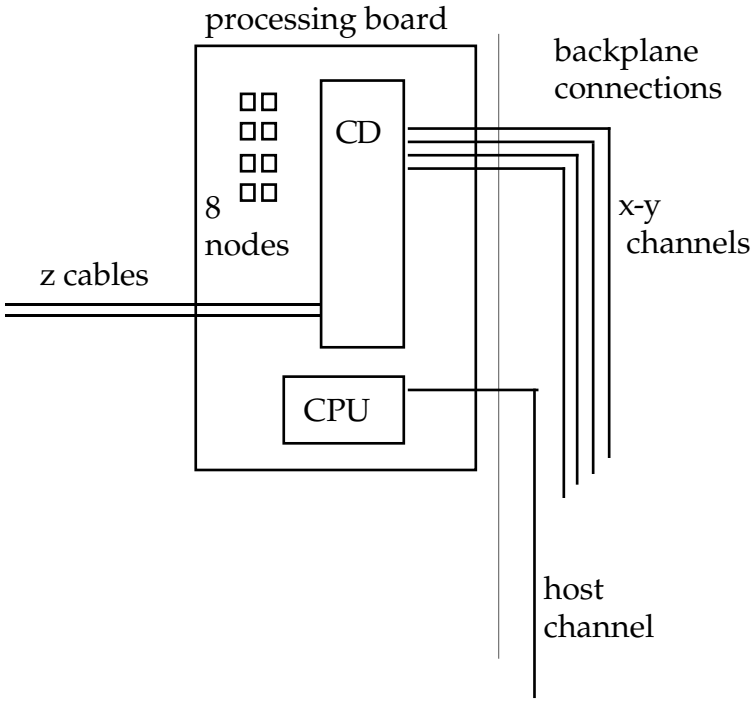

Figure 8 APEmille interconnections

Data Routing

The previous paragraph has described the proposed interconnections for APEmille under the hypothesis that the connectivity of the new machine is the same as that of APE100. We are considering however the possibility to add more general routing capabilities to the new machine. There are a number of constraints on these new developments, namely:

1) the standard APE100 nearest neighbour connections must be kept in the new machine.

2) any addition must not be paid in term of a loss of performance of the first neighbour connections.

3) the added complexity implied by a more general routing scheme must be a small fraction of that of the first neighbour system.

A radical change to the present architecture is that of allowing arbitrary routing between the nodes: each node specifies the destination node for the data word been moved to a remote destination. Some consistency constraints must be imposed. For

instance it may be forbidden that two nodes specify the same destination node (an exception is raised if this happens).

Arbitrary routing is not a pre-requisite for any of the application described in [1]. It may however be extremely useful in several applications involving inherently non local operations, such a Fast Fourier Transforms (in the case the data structure could not be distributed among the nodes adopting a "cut along the row" data distribution strategy) or cluster algorithms. It is also useful to map geometrically simpler topologies (for instance 2-D planes) on the machine.

Implementing global routing is in general difficult for two reasons. First, the best route must be selected for any set of communication sources and destinations. Second, potential clashes on the network has to be dealt with. We are considering to use a very simple scheme that respects the constraints discussed above but sacrifices the communication speed for far away (non nearest-neighbour) connections. A decision on to include a routing facility in the final design will be taken as soon as we have checked the performance of this system on a few realistic applications and evaluated its hardware costs.

The basic idea is as follows:

-Data movements occur in lock step mode between CHEETAH devices according to a predefined sequence that avoids clashes (for instance, transfers in the positive direction occur in even cycles and transfers in the negative directions take place in odd cycles).

-Each node loads its data word onto its CHEETAH device, along with its destination.

- The communication devices move data in any direction that reduces their distance to the destination point. A stack is available on each device to temporarily store the data words that cannot be routed immediately.

- The previous step is repeated till all data words have reached the final destination. At this point, all data are loaded onto the destination node.

- The internal operation of the nodes is halted till the slowest data word has reached destination.

This arrangement does not require any change in the architecture of the interconnection network. Indeed nearly all enhancements take place inside the CHEETAH device. The latency for a generic data move can be however very large, and depends heavily on the actual details of the algorithms. For instance, in an array of N^3 processor, the worst case transmission can take as long as $2*N + N^2$ moves, if moves are performed orderly in the x direction first, followed by y and z. More clever strategies however reduce this value to about 3*N except for a few pathological cases. Several promising strategies are currently being simulated.

The floating point kernel of the node

The custom devices of APEmille will be based on a state of the art CMOS technology. A sub micron CMOS, 3-metals will be probably adopted for the custom components. This silicon technology can be safely used with a 50 MHz clock. However, technological improvements during the APEmille project lifetime may stretch the clock cycle towards 100 MHz. Therefore while the first JANE prototypes will probably adopt a conservative 50 MHz clock cycle we plan to produce 100 MHz JANE chips as soon as possible. The floating point processor of APE100 is based on a 1.2 μm CMOS, 2-metals technology and extends on an a die of 0.91 cm^2 (with core area 0.84 cm^2). Using this 4 years-old technology the APE100 MAD hosts 2 floating point devices and one register file with 128 register locations. Using at least a 0.7 μm or better a 0.5μm technology we estimate at least a four-fold improvement in density and a factor 2 in speed with respect to the APE100 MAD. On each JANE processing node we plan to fit, in roughly 1 cm^2 four floating point devices and one 256 location register file. We plan to maintain the silicon core area under 1 cm^2 in order to get the best price/performance ratio from the silicon technology. As well known, the yield rapidly falls down beyond 1 cm^2 areas. As already stated, an architectural keystone in computer design is the balance between arithmetic performances and memory access speed. We accept a value of 4 for the ratio between floating point ops per second and local words accessed per second. Therefore we accept a register file with a single input-output port (directed toward the JANE's local RAM memory) and as much read and write ports as needed to feed (and to empty) a pipeline with four arithmetic devices. These 4 devices have to be arranged in such a way to provide high efficiencies on our typical applications targets. In the following we discuss several different schemes for the floating point part of JANE and the relative application efficiencies.

A first reasonable schema for the floating point kernel of a node with four arithmetic devices could be the following:

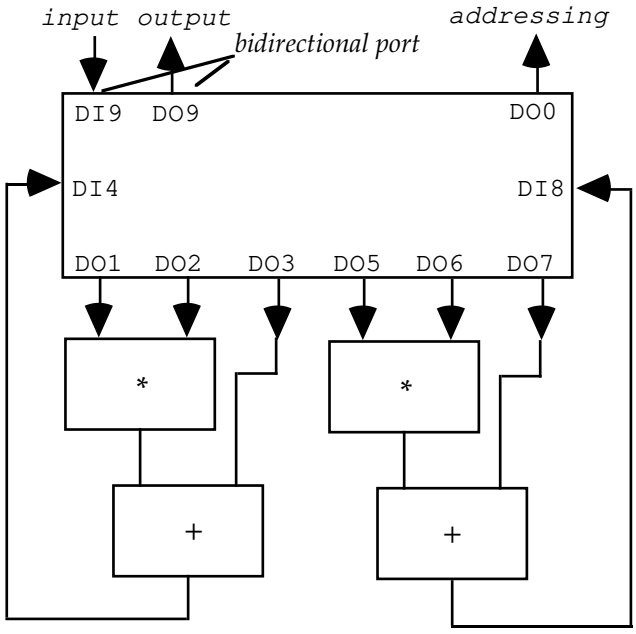

Figure 9. A first possible floating data path for JANE (type A)

It uses a register file with 7 output ports, 2 input ports and 1 bi-directional port. This scheme is well balanced for QCD applications if the input-output port connected with the external RAM memory can be used while the floating point units write their results. In a more general application context we have an excess of floating point power compared with the I/O bandwidth. This device can complete the two "NORMAL" operations *DI6=DO1*DO2+DO5, D7=DO3*DO4+DO0* while performing just one external memory access.

A different arrangement saves one read port and one write port, while maintaining high efficiencies on our application targets.

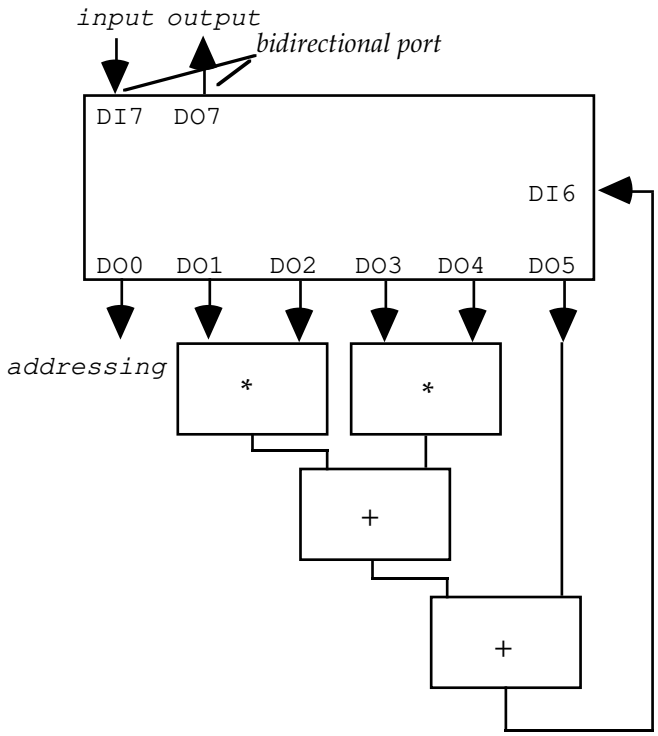

Figure 10. Another possible JANE floating architecture: type B

The register file has 6 output, 1 input and 1 bi-directional ports in this case. This device can execute an operation of the form *DI6 = (DO1*DO2 + DO3*DO4) + DO5* ("*ABNORMAL*" operation) in one clock cycle. Let us note that this solution is algebraically equivalent to *DI6 = (DO1*DO2+ DO5) + DO3*DO4).* Nevertheless for QCD applications (and in general for applications which require an extensive use of floating point resources versus memory access) the absence of a second input port may produce a performance degradation in particular circumstances. Also the length of the arithmetic pipe may negatively affect performances. To reduce this problem, we propose a third *flexible* FPU architecture:

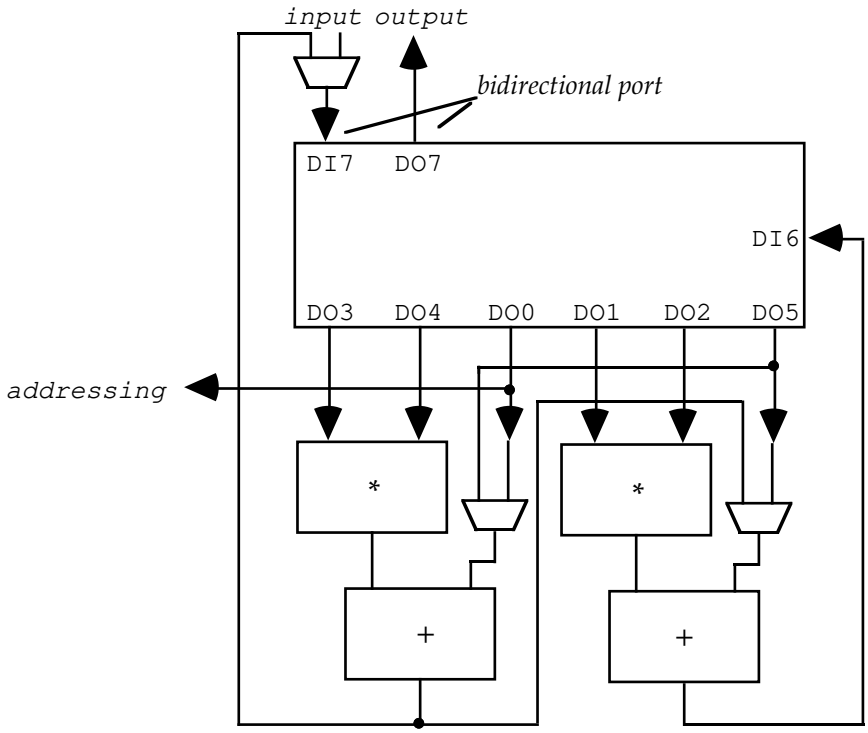

Figure 11. The third possibility: type C

At every cycle we are able to select between two alternatives. The first is executing an "ABNORMAL" operation *DI6 = DO1 * DO2 + (DO3 * DO4 + DO5)* . The second option may be adopted only when the*DI7* port is not used by the external memory. In this situation we may execute two NORMAL operations *DI6 = DO1 * DO2 + DO5* and *D7 = DO3 * DO4 + DO0*.

The Type C design does not require more ports than Type B. Moreover two shorter and independent arithmetic pipes improve performances.

We now report on the evaluation and comparison of the performances of the proposed architecture for several computational kernels. Given a real machine with fixed peak performance we may define as the corresponding "ideal" machine a system running, on each application kernel, with the same peak performance. In other words each application kernel defines a number of required floating point operations. On an ideal machine with a given number of floating point devices the number of execution cycles of an that kernel is obtained dividing the required total floating point operation by the available devices. In principle we should choose as efficiency index the ratio between the number of cycles of the application kernel on the ideal machine and the kernel execution cycles on the real machine. Actually we are mainly interested in a APEmille design preserving the satisfactory efficiencies of APE100 on a selected set of applications on which APE100 shows satisfactory performances. JANE contains 4 floating point devices while MAD hosts only 2 devices. Therefore we may define the efficiency index of the APEmille architecture Vs APE100 architecture as the ratio:

$$\eta(APEmille\ vs\ APE100) = \frac{execution\ cycles(APE100)}{2 * execution\ cycles(APEmille)}$$

In the framework of our application requirements the target APEmille architecture would have $\eta = \mathbf{1}$. This index has been evaluated through a simulation of the proposed architectures. The simulated results are shown in the next table for several key examples:

KERNELS	**A**	**B**	**C**
LGT QUENCHED	96,5%	90,2%	96,0%
SPIN GLASSES & NEURAL NETWORKS	91,1%	89,2%	92,5%
MATRIX CALCULATION	88,3%	91,9%	90,2%
FLUID DYNAMICS	75,5%	69,4%	80,8%

Let us note that the sources were written in a form optimised for the present APE100 architecture. A careful writing of the code for the new architecture could allow better performances.

As a global result, architecture (C) obtains the most satisfactoring performances.

Let us explain in some detail the optimisation strategy adopted for the flexible C design. The micro code is generated starting from an APE100 assembler code produced by the present APE100 compiler. A pre optimisation procedure merges, when possible, instruction belonging to the NORMAL family into ABNORMAL instruction (i.e. g=a*b+f, f=c*d+e -> g=a*b+c*d+e). A device allocation phase follows adopting a "shaker" strategy inherited from APE100: it attempts to start each instruction at the earliest moment compatible with hierarchical dependencies and device occupation. Moreover if the resulting instruction belongs to the ABNORMAL family the left and right NORMAL pipes are arranged into a unique ABNORMAL pipe. On the other hand, when the instruction is a NORMAL one, the right pipe is allocated first, if available. If this is not the case, the left pipe is chosen. This last choice is rejected if the bi-directional port is needed for input-output. To evaluate the impact of having a write port shared between the left pipe and the external I/O we have simulated three different options on the LGT codes:

C1. The left pipe has no writing pathways to the register file.
C2 ≡ C. The left pipe shares the external I/O port.
C3. There is an additional, independent write port for the left pipe.

Efficiencies over LGT codes of the C1,C2,C3 options.

C1	**C2 = C**	**C3**
90.2%	96.0%	96.9%

We observe the limited advantage given by the costly C3 solution. The simple upgrade of C1 to C2 on the other hand gives an appreciable increase of performance.

In all proposed scheme the register file of APEmille requires more ports than the APE100 one. Let us note that we have a backup solution ready, if unexpected difficulties arise. In fact the register file shown in the last scheme can be obtained applying a mirror technique to the APE100 register file, as in the following figure:

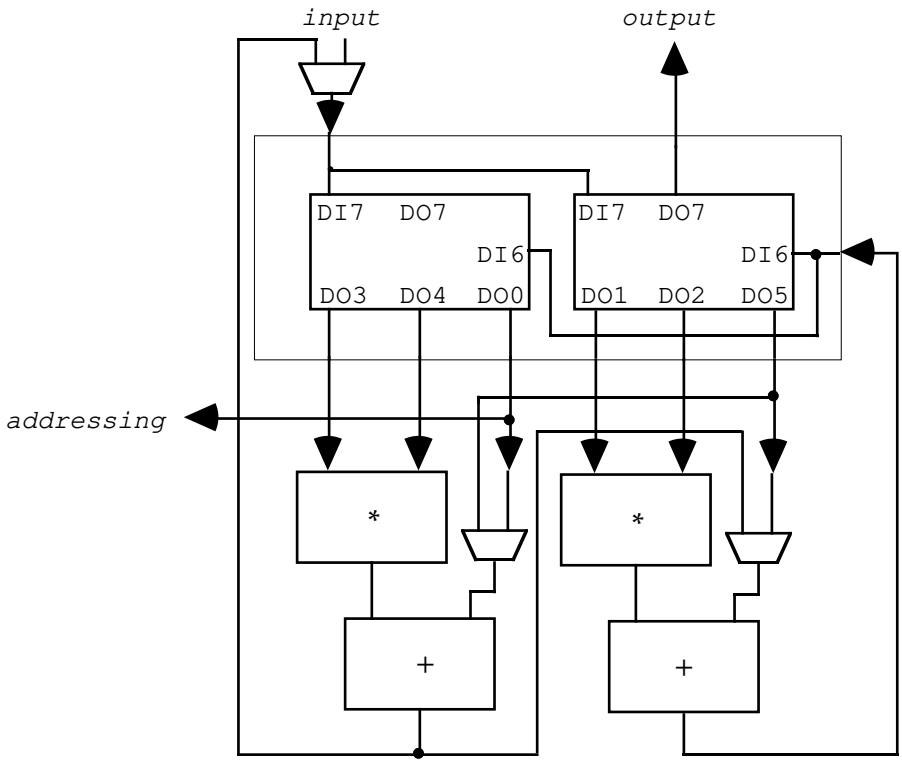

Figure 12. Mirror register file for fig. 9

This mirror register file can be obtained storing data in the same location of two identical register file. The write ports are connected together and the data are written at the same address in both the register files. The read ports are independent. The number of read ports is doubled while the number of write ports is the same.

Double precision and integer arithmetic

APEmille is optimised for single precision arithmetic. Extended precision arithmetic will be provided however, since it is needed in several sensitive computations occurring in the applications described in [1]. We are considering to provide an extended precision performance somehow different from standard double precision. We plan to use the floating point architecture developed for the IBM Power Risc processor, that we plan to restrict to hardware single precision. This architecture compute the floating point operation A = B * C + D without intermediate rounding between the multiplier and the adder. In fact a result with accuracy equal to three times the accuracy of the operands is computed before the final rounding. This arithmetic block fits naturally in the architecture of JANE (version C). The main advantage of this architecture in this context is that it allows a software emulation of an extended precision with a mantissa having a number of bits two times bigger than single precision. An extended precision number is a pair of single precision floating point numbers (say (a,b)), with relative exponents such that the sum (a+b) yields at least 48 bits. An extended precision data word has therefore 48 bit accuracy in our case,

slightly less than available in the standard IEEE compliant representation (53 bits). There is also a loss of dynamic range, since the smallest extended precision representable number is of order 10^{-32}, while the smallest standard single precision number if of order 10^{-39}. Performing an extended precision operation requires six to seven standard operations, so the extended precision peak speed of the machine will be reduced accordingly. These operations can however be intermixed with other operations in the arithmetic path and optimised with standard techniques.
Integer arithmetic is useful in several areas of physics simulations. Basic integer arithmetic operations can be added to the floating point data path with a minimum of added complexity. Integer adds, logic operations and shifts in fact may use sections of the hardware circuitry used for floating point arithmetic. Integer multiply is more delicate, since it requires an integer multiplier significantly larger than required for single precision purposes. At present we plan to use two passes of the floating point multiplier. Bitwise operations require a negligible silicon area.

The Host Interface

The Processing Boards of APEmille will be interfaced with a network of low cost workstations. A Synchronous Interface will be implemented to be connected with a standard workstation bus and a single workstation will be able to drive a set of Processing Boards. Depending on the application requirements the number of PB driven by each workstation will be chosen to satisfy the needed I/O data rates.

We will base the workstation network on a market standard solution and obviously each workstation will be able to access disks and peripherals.

The operating system distributed on the network will manage the data distribution and all the APEmille system services and exception handling requirements.

The host will be able to access the Control System memories as well as the data memory of the Processing Nodes.

Software

The software of APEmille can be divided into 5 main areas:

a) High Level Compiler (Language Compiler)
b) User and system service interface with the operating system
c) Low Level Compiler (Code Generation and Optimisation)
d) Low level interfaces and drivers of the operating system
e) APEmille Simulator

We intend to maintain the same Language and Operating system user interfaces of APE100, thus we can imagine that (a) and (b) will not generate major extra activities.

The development of (c) and (e) has to evolve at the same time and the corresponding activities have been started.

We have to integrate the new host interface. This implies the activation of the task (d). Also in this area a large part of the APE 100 Operating System architecture can be reused. This task has to be ready in time to test the first prototypes.

System implementation

The simplification of replicating the CPU on each board reduces the number of different kind of boards in the system as well as the total number of boards. This solution reduces development time, while improving the reliability of the machine.

Each node communicates with the first neighbours in a 3-D space. The connections over the X and Y directions run over the backplane. The connections over the Z axis are obtained using cables connecting PB to PB across crates.

A crate should host 16 Processing Boards, while a rack should host 4 crates.

The smallest APEmille configuration contains just one PB. One interface steers up to 16 PBs and an intermediate configuration with 4 PBs is also possible.

The single board solution permits to figure out a reliable two step development path from 50 MHz version toward a 100 MHz system, according with the following table.

		50 MHz version	100 MHz version
1 PB	=>	2x2x2 - 1.6 GFlops	3.2 GFlops
4 PB	=>	8x2x2 - 6.4 GFlops	12 GFlops
8 PB	=>	8x4x2 - 12 GFlops	25 GFlops
16 PB	=>	8x8x2 - 25 GFlops (1 full crate)	50 GFlops
4 crates == 1 rack	=>	8x8x8 - 100 GFlops	200 GFlops
4 rack	=>	8x8x32 - 400 GFlops	800 GFlops

Should the technological growth proceed too slowly to permit this clock transition, an 800 GFlops system can be assembled as an array of 8*8*64 nodes (8 racks) running at 50 MHz. There is no architectural limit in the growth on the Z direction, because the connection is obtained by means of a ring of external cables all of the same size. Our experience suggests that for many practical reasons it is better to limit the growth to 64, reaching maximum system configuration of 800 GFlops with a 50 MHz clock and 1.6 TeraFlops at 100 MHz.

Maintenance

APEmille must be an highly reliable machine. All the data paths will be controlled by means of EDAC (Error Detection and Correction) or by parities.

The host will be able to identify malfunctioning components or data paths while the machine will be able to run in a self test modality allowing a fast identification of failing processing nodes or data paths though a hardware monitoring mechanism which verifies at run time the identity of test code distributed among the nodes.

All the internal data paths will be readable from the host to verify in all the subsystems the coherence with the simulation results and the test vectors.

Design Methodology

The definition of the APEmille Assembler and of a prototypical Compilation Chain has been already started. The resulting code will be executed on a C++ functional model allowing to evaluate the efficiencies obtained by different architectural choice at different level of abstraction. A functional and then detailed definition of the Hardware in a VHDL like description language is going to be started. These two simulation chains will proceed in parallel and will generate cross feedbacks and checks. Our strategy is to consider the system as a whole. We defined a set of applicative kernels to have a reference workload and we will always verify the impact of each design solution on the APEmille efficiencies on this workload.

A key point in the APE100 project development, which will be maintained in the APEmille one, is the control over the whole software and hardware components. Being able to define our Assembler, Programming Language Compiler, Optimisers and Functional Simulator is a strength point which allows us to obtain a system which runs on our applicative with high efficiencies. Moreover the control over all the software components allows us to explore all the conceivable hardware solutions.

On the hardware site we plan to adopt the state of the art ESDA (Electronic System Design Automation) Tools. The VLSI components could be designed using the CADENCE tools, which the APE group successfully used to develop the APE100 custom chip set.

We will use a VHDL compiler and simulator and some synthesis tool if needed. The whole system will be designed with a Concurrent Engineering approach. We feel that a strict co-operation with the industries which will be in charge for the mass production should be highly appreciable since the beginning of the design phase. This way the quality, test and production constraints will be taken in account at the early design stages.

Costs and Timetable

We estimated the costs and timetable for 4096 APEmille Nodes equipped with memories of 2 or 8 MWord per node.

We reached the conclusion the machine can be built at the costs of 14.6 GLit for 2 Mword version and 24 GL for 8 Mword version in 4 years +- six months provided enough human resources.

References

[1] C.R. Allton et al. "Physics Projects on APEmille" Internal Note n. xxx (1994) INFN Sez. Roma.

[2] C. Battista et al. " The APE-100 Computer: (I) The Architecture" Int. Journ. High Speed Comp. **5** (1993) 637-656;

A. Bartoloni et al. "A Hardware Implementation of the APE100 Architecture" Int. Journ. of Mod. Physics C **4** (1993) 969-976;

A. Bartoloni et al. "The Software of the APE100 Processor" Int. Journ. Mod. Pys. C **4** (1993) 955-967;

A. Bartoloni et al. "MAD, a Floating-Point Unit for Massively -Parallel Processors" Particle World **2** (1991) 65-73;

G. Bastianello et al. "A High Performance Single Chip Processing Unit for Parallel Processing and Data Acquisition Systems" Nucl. Instrum. Meth. **A324** (1992) 543.