Rome Preprint 1255/99
IFUP-TH 34/99
Bicocca-FT-99-17
DESY-99-105

PROPOSAL FOR A MULTI-TFLOPS LGT COMPUTING PROJECT

F. Aglietti, A. Bartoloni, C. Battista, N. Cabibbo, M. Cosimi, A. Lonardo, A. Michelotti, F. Rapuano, D. Rossetti, M. Torelli, P. Vicini
Dipartimento di Fisica, Università di Roma 'La Sapienza' and
INFN, Sezione di Roma, P.le A. Moro 2, I-00185 Roma, Italy.

R. Alfieri, R. Di Renzo, E. Onofri
Dipartimento di Fisica, Università di Parma, and
INFN, Sezione collegata di Parma, Parco Area delle Scienze, I-43100 Parma, Italy.

I. D' Auria, W. Errico, G. Magazzù, F. Schifano, R. Tripiccione
INFN, Sezione di Pisa, Via Livornese 1291, I-56010 San Piero a Grado (Italy).

P. Deriso, R. Petronzio
Dipartimento di Fisica, Università di Roma II 'Tor Vergata' and
INFN, Sezione di Roma II, Via della Ricerca Scientifica, 1 - 00133 Roma

C. Destri, G. Marchesini
Dipartimento di fisica, Università di Milano-Bicocca and
INFN, Sezione di Milano, Via Celoria 16, I-20100 Milano, Italy

W. Friebel, U. Gensch, A. Kretzschmann, H. Leich, S. Menschikov, N. Paschedag, G. Sacco, U. Schwendicke, H. Simma, R. Sommer, K. Sulanke, P. Wegner
DESY, Platanenallee 6, D-15738 Zeuthen, Germany.

A. Fucci, B. Martin
CERN, CH-1211 Geneva 23.

E. Panizzi

Dipartimento di Ingegneria Elettrica, Università de l'Aquila and
INFN, Sezione di Roma, P.le A. Moro 2, I-00185 Roma, Italy.

ABSTRACT

In the next five years Lattice Gauge Theory (LGT) simulations will be mainly focused on full QCD studies on lattices of intermediate size and on quenched QCD simulations on very large lattices at large β, with emphasis on b-physics. This program requires computing power in the order of tens of TFlops, sustained for periods of the order of one year, along with the flexibility needed to handle effectively also smaller lattices. In this letter of intent, we propose to develop a computer architecture suitable for these problems, to be implemented in terms of small scale PC-based systems and of large scale systems based on custom processors. The common feature of both implementations is a LGT-optimized high-bandwidth communication network.

1 Introduction

The APE group has been involved for more than one decade in the development of high performance parallel processors for Lattice Gauge Theory (LGT) [1],[2]. The scientific achievements of all generations of APE machines have been recognized world wide. APEmille, the last in the series[3], is now in the final stage of development, and physics programs are running on prototype systems. Although APEmille will sizeably increase available computing power at INFN and DESY, LGT simulations will eventually require multi Tflops performance, which is beyond the range of APEmille. We believe that this is the right time to launch a new project, aiming to provide computing power of the order of several Tflops within three or four years from now.

This document is a letter of intent/proposal for such a project, provisionally called **apeNEXT**.

This paper is structured as follows. In section two we discuss the general physics requirements motivating our proposal. In section three we outline the technological scenario shaping the new project. Section four provides an architectural outline, whereas section five describes the organization of the project. Sections six and seven discuss timescale and costs.

2 Physics Requirements

In the definition of the new LQCD engine project we keep a clear focus on a very limited number of (broad) physics simulation areas, that set the physics requirements for the new project.

The translation of physics requirements into machine parameters requires certain assumptions about the algorithms to be used. We base our considerations on tested algorithms such as SSOR-preconditioned BiCGstab and Hybrid Monte Carlo, for Wilson fermions with improved action [4]. New theoretical developments (domain wall fermions, Wilson-Dirac operators satisfying the Ginsparg-Wilson relation, ...) are likely to be implemented in a way which is very similar to the standard Dirac operator.

We expect that in the years 2003-2006, large *production* LQCD simulations will be mainly focused on the following lines:

- full QCD simulations (including dynamical fermions) on lattices with sizes of the order of $48^3 \times 96$ (a physical system of $L = 2 \ldots 4\, fm$ and $a = 0.1 \ldots 0.05\, fm$). Dynamic quark masses should also decrease as much as possible with a reasonable target corresponding to m_π/m_ρ of about 0.35 (although it is not realistic to expect that both goals are obtained in the same simulation).

- simulations in the quenched approximation on very large lattices ($100^3 \times 100 \div 200$) and large β ($L = 1.5 \ldots 2.0\, fm$ and $a = 0.1 \ldots 0.02\, fm$) for the study of b physics with as little extrapolation as possible in the mass of the heavy quark.

The first item is heavily CPU limited, since one has to solve the Dirac equation repeatedly during the updating process. The second item is basically memory limited, due to very large lattice size. In both cases, our target is a resolution about two-times better than currently possible (implying, as discussed later on, an increase in computing power of two orders of magnitude).

As a guideline to define a new LQCD engine for these classes of problems, we require that:

1. The node-topology and communication network is optimized for the lattice sizes required in full QCD simulations. Since for many problems of LQCD it is important to perform a finite-size scaling analysys, the new machine must perform efficiently not only on *large* but also on comparatively *small* lattices, eg., in full QCD one may think of $N_L^3 \times N_T$ lattices with $N_L = 8, 16, 20, ..., 32$ and 48, and $N_L \leq N_T \leq 2N_L$.

2. The communication network has enough bandwidth to handle the large degree of data exchange between neighbouring sites (and hence compute nodes) needed in LGT computations. The interconnect architecture should support the natural (APE-like) programming model with direct remote data access [6]. This approach minimizes software and memory overhead (and coding effort) for pre-loading of remote data.

3. The processing nodes sustain high performance on the execution of the arithmetic and control operations which are relevant for the codes (or at least their basic kernels) of full-QCD algorithms, in particular double precision floating point arithmetics, memory access to field variables of composed data structures, local and global program-flow control, etc.

4. Memory size, disk space and disk-bandwidth match each other and are well suited to the problems we want to study. This means that all compute intensive kernels must not be slowed significantly because required data is not available in main memory. We must keep all data in physical memory as long as possible. In all cases in which this is not possible (e.g., for light-fermion propagators on very large lattices) we must be able to temporarily store on (and retrieve from) disk with large enough bandwidth.

These requirements shape the global architecture of the machine:

1. We consider architectures based on three dimensional grids of processors, with nearest neighbour data-links. Reasonable sizes of the mesh of *processors* that will be used for the simulation of large lattices are somewhere in the range $8^3 \cdots 12^3 \cdots 16^3$ nodes, where a physical lattice of $48^3 \times 96$ points can be readily mapped. For finite size analsys on small lattices, a mesh of $4^3 \cdots 6^3$ processors is envisaged.

   The size of the processor mesh dictates a lower bound on the communication bandwitdh between neighbouring processors. We define by ρ the ratio of local memory accesses (transfers between processor and its memory) over remote memory accesses (transfers between neighbour processors), which depends on the lattice size and the algorithm. Under the assumption of balanced local bandwidth (i.e., processors are able to access enough data in local memory to sustain their potential performance, see later for details), bandwidth for remote communications must not be lower than $1/\rho$ local bandwidth. Estimates of the required ratio for a naive implementation of the Dirac operator using Wilson fermions are given in table 1 for a sublattice of $N_S^3 \times N_T$ physical points per processor (note that, to first approximation, $\rho \simeq 2N_S$).

Linear lattice size	ρ
3^3	5.8
4^3	7.8
6^3	11.6
8^3	15.5

Table 1: Local vs remote memory access patterns: ρ is the ratio of memory accesses to local memory over memory accesses to neighbour nodes in a simple implementation of the solver for the Dirac operator. ρ is estimated as a function of the linear size of the sub-lattice mapped onto *each* processor.

2. To discuss memory-size requirements in more details, one has to distinguish between the case of full QCD simulations and calculations in the quenched approximation.

   In full QCD simulations, by far the largest amount of time is spent in the *updating* process. In this case, our on-line memory has to be large enough to allow for the implementation of efficient algorithms. State-of-the-art update algorithms need a large number of auxiliary fields on each lattice site. We use as unity the amount of memory taken by one fermion field (24 data words, corresponding to 192 bytes in double precision. We will call

$U_{ab}(x,\mu)$	gauge fields	72 W	3 feq
$S_{ab}^{\alpha\beta}(x,0)$	fermion propagator	288 W	12 feq
$\psi_a^{\alpha}(x)$	(pseudo-) fermion field	24 W	1 feq
$(\sigma\cdot F)_{ab}^{\alpha\beta}(x)$	Pauli term for improvement	72 W	3 feq

Table 2: Data structures used in Lattice QCD and corresponding memory requirements (in words and fermion equivalent storage) per lattice point. Greek indices run from 1 to 4 and latin indeces from 1 to 3. The first three entries are general complex matrices, while the Pauli term is hermitian: $(\sigma\cdot F)_{ab}^{\alpha\beta} = [(\sigma\cdot F)_{ba}^{\beta\alpha}]^*$.

this quantity a fermion equivalent - feq - in the following). A generous estimate, leaving space for more sophisticated, presumably more memory intensive algorithms, is about $\simeq 200 feq$ per site.

On the other hand, in the case of the quenched approximation, the updating process may be neglected for both computing power and memory requirements (less than $10 feq$ per lattice site are needed). Instead, we have to consider the memory requirement originating from the *measurement* of a heavy-light form-factor. The database needed for such a calculation consists of one gauge field configuration, one Pauli term, $N_l + N_h$ fermion propagators (N_h and N_l are the numbers of heavy and light fermions) each replicated for the number of momenta and operator insertions used and for each lattice site (typical cases, being $N_h = N_l = 4$, 3 momenta and one operator insertion). Quenched QCD will be used essentially for heavy quark phenomenology. Here the real problem is the extrapolation to the b quark mass. To be safe one should have a physical cutoff much larger than the masses that enter the simulation. Then *large* lattices, of the order of 100^4, are necessary.

We summarize our memory requirements in table 2 (where the size of the relevant data structures are presented) and in table 3, where actual memory sizes are collected, under the assumptions of using double precision [1] throughout. From the first two lines of table 3, we see that we cannot expect to keep the whole data-base in physical memory when *large* lattices are considered. However, if only two propagators at the time are kept in memory, for ease of programming, while the others are either recalculated (the heavy ones) or stored and reloaded from disk (the light ones), memory requirements reduce sharply (third line in the table).

We conclude that, by judiciously swapping data to disks, a memory size of the order of $\geq 1 Tbyte$ is a nice compromise for both our case studies.

[1] The necessity of double precision arithmetic in full or large scale quenched QCD has been investigated in the literature [5] and will not be discussed here.

Alternatively, one might consider two memory options: a *small memory* machine ($\simeq 500 Gbyte$) for full QCD and a *large memory* version ($1 \div 2Tbyte$) for quenched studies.

Case	updating	measurement
small lattice, full QCD	$400G$	$1.4T$
large lattice, quenched QCD	$200G$	$13.T$
large lattice + disk	$200G$	$1.8T$

Table 3: Total memory requirements for the case studies discussed in the text. The line labelled *+ disk* refers to the case in which two propagators only are kept in memory (all others being swapped onto disk or re-computed).

3. Fast input-output is mandatory, as obvious from the previous point, for studies on large lattices. As a rule of thumb, we may want to load or store one (*large* lattice) propagator ($\simeq$ 250 Gbytes) in little more than one minute. This requires a global bandwidth of the order of 2-3 Gbytes/sec.

   Permanent storage of full QCD configurations is also required due to the computing effort needed. This is a storage-density (as opposed to bandwidth) problem that, given the sizes involved, does not appear to be critical. In the case of large lattices in quenched QCD the strategy of computing *on the fly* without saving configurations is the best. Only the final correlation functions are saved and this means at most a few tens of MB per configuration.

Processing performance is strictly speaking not a clear-cut requirement: the more is available, the better. We can estimate how much is enough, however, by extrapolating the present state of the art. A sustained performance of 300 GFlops (with perhaps 40% efficiency) is now heavily used for full QCD simulations on lattices of size $24^3 \times 48$ [7]. If we assume a critical slowing down where computer time grows like a^{-7}, we would like to have a sustained performance two orders of magnitude higher if we want to halve a.

An ambitious target for our project is therefore a *total installed performance* in the order of $10 \ldots 30 TFlops$. From the point of view of physics requirements, it is not important that this computing power be sustained on a *single* system. Several smaller machines can perform equally well (or perhaps better), as long as *each of them* is able to handle large enough lattices.

Also, we must envisage the operation of some lower performance (and correspondinlgy smaller memory) machines, where small lattices are handled and algorithms, programs and physical parameters are tuned before a large calculation is moved onto a large production machine.

3 Technological Scenarios

Standard off-the-shelf processors have increased in performance by one order of magnitude in the last 8-10 years, with a remarkable improvement in the efficiency of floating point computations. Benchmarks with LGT kernels carefully coded in standard high-level languages indicate that up to about 30% of the peak performance can be reached for *local* lattices with realistic sizes (i.e. exceeding the cache size)[8].

This progress is roughly matched by improvements made by the dedicated processors of the APE family. The latter have today about one half of the peak performance of a commercially available high-end processor, with similar figures holding in the early 90's. On the other hand dedicated processors, such as those of the APE family, have a sustained efficiency of the order of 60% for LGT kernels [9], a factor two better than off-the-shelf components.

If we extrapolate from the APEmille experience, we may expect that a new generation custom processor, using 0.18μ silicon technology, has a clock frequency a factor 3 higher (about 200 Mhz). The most used floating point operation in almost all LGT kernels is the *normal* operation $a \times b + c$, performed on complex double precision values, equivalent to 8 Flop. This gives a peak performance of at least 1.6 Gflops per processor, again comparable (within a factor two) with trends of reasonably priced off-the-shelf processing boards (e.g., a dual processor PC running at 800 Mhz) .

A bandwidth with local memory of the order of $3Gbyte/sec$ is needed to keep such a processor busy.

This value results from the observation that typical LGT kernes have a value of about $R \simeq 4$ for the ratio R of the number of floating point operations performed over the number of data words accessed during the computation(assuming that double precision is used throughout). The required bandwidth is in line with forecasts made for high-end PC memory systems using either new generation SDRAM at 100 Mhz, active on both clock edges (with a data bus of 128 bits, exactly one complex operand), or more sophisticated RAMBUS interfaces.

These basic considerations lead us to the conclusion that if we consider *single node performance only* there is no clear-cut choice between a custom processor or a commercial sytem of the PC class. A choice must be therefore based on problems related to the assembly of complete systems.

We believe that a custom architecture is superior for *very large* ($\geq 500 nodes$) *systems* for the following reasons:

- lower power consumption by one order of magnitude.
- significantly more compact mechanical design.
- better scalability once the basic units are operating (reliability and software issues of large systems).
- easier interfacing with the necessary custom remote communication network and the host system.
- better control of technological aspects and less dependence on changing commercial trends during the realization of the project.

On the other hand, we see several advantages stemming from the use of PC-derived systems for *smaller machines*:

- limited hardware development effort.
- standard software is readily available for major parts of the compiler and the operating system.
- short lead time to commission a prototype system.

An important architectural feature with technological implications is synchronization between processors. Regardless of the technological choice made for the processor, we think that no real advantage is gained by departing from the Single Instruction Multiple Data (SIMD) or Single Program Multiple Data (SPMD) programming style used in previous generation APE machines. At the hardware level, APE processors of all previous generations have been hardware-synchronized with an accuracy of a fraction of clock cycle. Although logically very neat, this is rapidly becoming impossible, for clock frequencies higher than 100 Mhz and across physical scales of several meters. We intend to follow an approach in which independent processors, while running at the same frequency, are only loosely synchronized. Logical synchronization will have to be enforced by some form of software-controlled barrier.

We now consider *remote communications* which, in our opinion, is the most serious technological problem of the project.

Let us assume a nominal performance figure of 1.6 Gflops per processor (a good guess, as discussed above, for custom or PC-derived nodes), along with $R = 4$, and $\rho = 8$. Under these assumptions, we need an inter-processor communication bandwidth of about 400 Mbytes/sec. We estimate that careful (and painful) programming efforts, maximizing the overlap between communication and calculation and minimizing the amount of data to be transferred, is not able to reduce this figure by more than a factor of $O(2-4)$. Communications are needed between nearest-neighbours (and possibly, along L-shaped paths, between next-to-nearest neighbours) in a 3-d array of processors, where each processor has 6 direct links to its nearest neighbours.

These communication requirements are a challenge for *both* kinds of processing nodes. Indeed, although several commercial technologies with the right order-of-magnitude bandwidth have been developed recently, they are typically too expensive and poorly optimized for regular 3-d arrays. They also have large startup latencies, badly adversing performance, and often put a formidable control burden on the host processor.

We think that a solution optimized for our specific needs can be worked out at the *signalling* level by appropiately using established technologies (remember that we only need short-distance point-to-point links). At the *interface* level, a simple light-weight protocol suitable for our communication patterns can be developed and, adapted with little effort *both* to a PC-based or a custom node.

4 Architecture Outline

In this section, we outline a possible architecture, scalable from about 100 Gflops to about 5 Tflops peak performance for a standalone machine. Just one such high-end machine would offer a ten-fold increase in peak performance with respect to currently available systems. Several (5 to 10) high-end machines, working together with a comparatively larger number of low-end systems, would allow to complete the physics program outlined in previous paragraphs.

Our architecture has two different implementations, which share a common programming environment and a common interconnection network. One implementation with PC-based processing nodes is suitable for small systems (say up to $6 \times 6 \times 6$ processors, or $\leq 500 Gflops$) with a possibly wider and more exploratory appication profile. The other implementation is based on an evolution of the traditional APE architecture and is optimized for large systems (say $\geq 1 TFlops$), for which PC's become difficult to manage and maintain, as already mentioned.

In *both* cases we propose the following structure:

- a three dimensional array of processing nodes, linked together by nearest-neighbour links. Each node is a complete and independent processor. All nodes execute the same program and are *loosely synchronized*, i.e., they are started at approximately the same time and proceed at approximately the same pace. They synchronize only when requested by the logical consistency of the program (e.g., before exchanging data). This mechanism is similar to the concept of `ifall` instruction ofAPE100 and APEmille.

- Remote communications use FIFO-based asynchronous connections between neighbouring nodes. The SIMD/SPMD programming paradigm a là APE does not require complex handshaking protocols, since transmitting nodes may assume that the receiving partner is always ready to receive the incoming message.

  This simple mechanism (still to be studied in details) brings several architectural advantages:

  1. It allows to use for the remote communications a programming style which is very similar to APE100/APEmille. The later has in fact the very convenient feature that no explicit distinction between local and remote memory accesses is required.
  2. It can be easily modified to allow hidden data transfers (data are moved on the links while the processing node is performing calculations).
  3. It drastically simplifies the global hardware synchronization logic of the system.

- The communication interface is an independent component. As discussed, the communication interface is based on FIFOs, allowing "elastic" connections between nodes. This novel feature has to be carefully simulated, but no serious problem is anticipated here. The real challenge in this area is a fast, yet cheap and reliable[2] data-link. Using $\rho \simeq 8$, we need links of $400 Mbyte/sec$ (or multiples thereof). Similar links are now used, in configurations not optimized for LGT, in several kinds of LAN stretching on small physical distances (often called SAN, for System Area Networks). The Myrinet link (now an IEEE standard) is one such example. The evaluation of the appropriate technology is the most important R & D activity of the project.

We now discuss the *specific* features of the two options, starting with the PC-based system.

[2]Note that due to the asynchronous operation of the machine, requirements on the bit-error rate of the communication system are less demanding than in previous APE generations, since it allows for repetition of transfers with minor performance loss.

- Based on preliminary tests, at the moment, the best choice for the nodes is a dual-processor Pentium-II or Pentium-III based system. This may change in the future, according to the rapid evolution of this market.
- Each (dual-processor) node should talk to its neighbours through a PCI board that handles all six links stemming out of the node. By using master PCI capabilities, a transfer rate from network to processor in the order of 100 Mbyte/sec can be achieved.
- Double size (64 bit) PCI interface will increase the available speed by a factor two. If double speed and double size PCI busses become available (and if they do so at a reasonable price-performance ratio), we will be able to better exploit our target link speed of 400 Mbytes/sec.
- Note however that we do not depend on the long term availability of the PCI interface, since we can hook our system to any other (equally or better performing) I/O bus or directly connect our communication interface to the memory bus.

The implementation for a large system, on the other hand, is based on the assembly of from about 1000 to about 4000 fully independent custom nodes.

- Each processor runs at about 200 Mhz clock. The processor merges the functions of the control (T1000) and floating-point (J1000) processors of APEmille on a single chip. It has its own data memory and program memory. We still need to evaluate whether data and program memory are independent banks (as in APEmille) or if they use the same physical chips. In the latter case some form of instruction-cache is needed on the processor. The basic floating point instruction is the *complex normal* operation, so peak performance is 1.6 Gflops (double precision). As already remarked, this requires a memory bandwidth of 3.2 Gbyte/sec ($R = 4$). Memory size per node depends on the number of processors (the total memory has to be larger than a minimum value, as discussed in section 2), but is likely to be in the range 256 Mbyte - 1 Gbyte per node. The actual choice may be heavily affected by cost factors. We are studying the possibility to increase performance by factors $2 \ldots 4$, by using some form of vector processing, in which several normal operations are performed concurrently. This is probably not too serious a challenge for processor development. The real problem is the need to increase local (and also remote) bandwidth by a corresponding factor.
- The processor array will probably have between $8 \times 8 \times 16 = 1024$ and $12 \times 12 \times 24 = 3456$ nodes, depending on peak performance of each processor. We assemble nodes on processing boards, similar to APEmille. Each

processor is more compact than in previous generations, and glue logic is almost completely absent.

One key technological advantage of this implementation is compactness. We expect to place from 10 to 30 processors per board, interconnected according to an appropriate topology. Selecting the best topology is an optimization issue, to be discussed at a later stage of the project.

- The node (and the network) should support not only data transfers between memory and register (as available on APE100 and APEmille), but also register to register. This can be used to reduce bandwidth requests by splitting a complex computation on more nodes, each node using local data as much as possible.
- A host system analogous to the one used in APEmille is a possible choice for the new machine. Based on networked Linux PC's and the CPCI bus, it is mechanically compact and reliable. Each PC will be in charge of several boards. The actual number of boards connected to each PC is dictated by the bandwidth available on the PCI bus to move data from APE to disk and vice-versa. For the sake of definiteness, assume a system distributed on 100 boards, with a total bandwidth of 2 Gbytes/sec (that is 20 Mbytes/sec per board). In this case, up to 4 boards can be handled by present generation CPCI CPU's. Higher performance PCI busses (double size and/or double speed) may allow to increase the number of boards connected to each PC. The host PC's will be networked with the most appropriate technology available in due time.
- We note that it is a relatively easy task to design the (fully self-contained) processing node(s) in such a way that they can be connected to a standard PCI desk-top PC. This possibility is very appealing for program debugging and small scale application. We plan to pursue this design characteristic.

An important point in the global project is the availability of a common programming environment for the two types of machines. What we want is basically one (or two) programming languages available on both implementations. It is also highly desirable that programming tricks to increase performance have a large degree of commonality between the two environments. We think that this target is not unrealistic as long as the well-tested SIMD/SPMD programming style of APE100/APEmille is adhered to. In practice, we can either extend a TAOmille compiler to the PC system or develop a (limited) C/C++ or Fortran compiler for the custom machine (or, even better, perform both tasks).

5 Project Organization

This section describes a very sketchy partition of the project in individual tasks. A detailed task definition, a share of responsibilities and an identification of the critical paths will be performed later on (Note that at this point in time, we are discussing with several institutions the possibility that they join the project).

In APEmille, an important role in the coordination of the project has been played by the existence of a complete and detailed model of the system, based on a VHDL description. We want to use this approach also in the new project. All collaborating institutions should maintain the VHDL-based simulation environment for both machine implementations. This *virtual* machine is the key tool needed to allow all collaborators to:

- understand and improve the global architecture.
- develop application programs and estimate performances.
- debug (and cross debug) both the global architecture and specific elements of the machines.

Once the global architecture is roughly outlined and understood, specific development tasks must be considered. At this point in time, we identify at least the following broad areas of development:

1. Development of the signalling level of the communication network, and of a basic transmission protocol.
2. Development of the interface hardware between communication network and PC-system.
3. Development of the custom processor.
4. Development of the interface hardware between the custom processor and the communication network.
5. Development of the board housing the custom processor.
6. Development of the mechanics/power-supply/backplane/cooling-system assembly for the custom machine.
7. Development of the mechanics for the PC-based machine.
8. Development of the programming environment common to both machines.

9. Development of the interface software between the custom processors. and the network of host PC's.
10. Development of software drivers and system level software for the PC's.

6 Project Timescale

We want to follow a fast project schedule, made possible by the logical simplicity of the project and by the fact that a number of important components are either similar to APEmille or readily available from commercial sources.

We define a development and prototyping roadmap, divided in three phases, namely,

- **R & D phase.** In this phase, the following tasks must be performed:
  - detailed analysis of the logical structure. of the machine (with emphasis on the synchronization issues),
  - logical design of the network.
  - evaluation and test of the remote link technology.

  At the end of this phase, the following items should be ready:
  - a VHDL simulation model of (at least a portion of) the global machine.
  - a simple programming environment to exercise the simulator with real life program kernels.
  - a preliminary hardware design.
  - one (or more) experimental test-benches to evaluate the best communication link. Some of these test benches are obviously going to be PC based.
- **Prototype phase.** In this phase, we develop and build prototypes of all components of the system. At the end of this phase, we should have:
  - The prototype mechanical/power supply system for the custom machine.
  - A prototype host setup.
  - A prototype processing board.
  - A first iteration of the network processor, with interfaces both for the PC's and for the custom system.

- A custom node version pluggable inside a PC.
- A small PC-based machine, using the network processors.
- A reasonably complete software environment.

- **Early production phase.** During this phase, final versions of the main hardware components should be built. At the end of this phase, we should have both a large scale prototype system, coming close to the threshold of 1 TFlops peak performance and a reasonably large PC-based system (say, 128 PC's)

We estimate that each of the three phases can be about one year long, starting from late 1999. At the end of the year 2002, we could move to a production stage in which one or more large systems are assembled.

7 Cost Estimates

Cost estimates are divided in development costs and production costs for the early production phases. We also quote very preliminary forecasts for the costs of a full scale production. All our figures do not cover basic costs, such as travel expenses and the like.

- **Development Costs** The bulk of costs here is associated with the development costs for VLSI components, printed circuits and mechanical systems. Estimates are given in the following table (in Euro).

VLSI development	600K
Mechanics dev.	150K
Printed Circuits	200K
Grand Total	950K

Table 4: Preliminary estimate of development costs for the key elements of the project.

- **Early production Costs** In the following we collect estimates, based on our experience with APEmille of the costs (in Euro) for the early production phase of a custom based machine, defined in the previous section, with a limited amount of disk space.

  For the PC based system, we estimate a cost of 300 $KEuro$, assuming that the cost of one PC board ($\simeq 2000 Euro$) remains constant in time.

600 procs @ 200 Euro	120K
300 Gbyte memory @ 1.5 Euro/MB	450K
4 Crates @ 10000 Euro	40K
50 Boards @ 5000 Euro	250K
10 PC's @ 2500 Euro	25K
2 Tbyte Disk @ 20KEuro	40K
Grand Total	925K

Table 5: Preliminary estimates of costs for the key items of a large prototype system using custom processors

- **Full Production Cost Forecasts** For definiteness, we consider costs for a system with 2000 nodes and 512 Mbytes memory per node. We include about 50 PC's and 30 Tbytes of disk space. Costs are shown in the following table (in Euro). The actual price break-up is of course very

2000 procs @ 100 Euro	200K
1Tbyte memory @ 1 Euro/MB	1000K
16 Crates @ 10000 Euro	160K
160 Boards @ 4000	640K
50 PC's @ 2000 Euro	100K
30 Tbyte Disk @ 15KEuro	450K
Grand Total	2.550 M

Table 6: Preliminary forecast of the production costs of a large system. Such a system would deliver $\geq$ $3.2Tflops$, leading to an estimate of $\leq$ $0.8Euro\ /\ Mflops$.

approximate. Using the figures in the table the impact of memory on global costs is very high. We remark however that prices for items like memory and disks are very volatile. Accurate estimates will have to be revised in due course. Considering that each processor has 1.6 Gflops top performance, we arrive at a final cost estimate of $\leq 0.8Euro/Mflops$.

The comparable figure for PC based systems (we assume that a 2 Gflops PC node costs 2000 Euros) is of the order of $1Euro/Mflops$. As already remarked, the two technologies (in their respective optimal range of performance) have very similar price/performance figures.

8 Conclusions

This document describes the physics motivations and basic architecture of a next generation LGT computer project. We think that the well-tested SIMD/SPMD architecture of the previous APE generation is still the best choice for a LGT focused high performance engine. At the engineering level, this architecture can be best implemented in two different ways, according to the size of the target machine, using either high-density simplified processors (for large scale machines) or traditional off-the-shelf PC boards (for small scale machines). In both cases, a dedicated (hence high performance and cheap) communication network is needed. We plan to work heavily on the development of such a network (largely common for both implementations) and to build both types of machines.

References

[1] N. Christ, "Proceedings of Lattice99", Nucl. Phys. B (Proc. Suppl.), in press.

[2] A. Bartoloni et al., Intl. Journal of High Speed Computing, 5 (1993) 637.

[3] R. Tripiccione "APEmille", Parallel Computing, in press.

[4] See for example: P. de Forcrand, Nucl. Phys. B (Proc.Suppl.) 47 (1996) 228; A. Frommer, Nucl. Phys. B (Proc.Suppl.) 53 (1997) 120; K. Jansen, Nucl. Phys. B (Proc.Suppl.) 53 (1997) 127, and references therein.

[5] R. G. Edwards, I. Horvath and A. D. Kennedy, Nucl. Phys. B484 (1997) 375; B. Bunk, S. Elser, R. Frezzotti and K. Jansen, CERN preprint, CERN-TH/98-127, hep-lat/9805026; K. Jansen and C. Liu, Nucl. Phys. B (Proc.Suppl.) 53 (1997) 974;

[6] TAO Reference Manual, (E. Panizzi and V. Ghisalberti Eds.), 1994.

[7] S. Aoki et al., HEP-LAT/9903001.

[8] H. Simma, "PC's for Multi_TFlops LGT Compute Engines?", Proceedings of the Workshop PC-NETS, INFN/TC-99/09.

[9] A. Bartoloni et al., Nucl. Phys. B (Proc. Suppl.) 63 (1998) 991.