ISTITUTO NAZIONALE DI FISICA NUCLEARE

Preventivo per l'anno **2001**

Nuovo Esperimento	Gruppo
APE (Pr.Sp.)	5

Struttura
ROMA I

Ricercatore responsabile locale: Federico Rapuano

Rappresentante Nazionale: Federico Rapuano

Struttura di appartenenza: Roma1

Posizione nell'I.N.F.N.: Primo Ricercatore

PROGRAMMA DI RICERCA

A) INFORMAZIONI GENERALI

Linea di ricerca	Calcolatori paralleli per la fisica teorica
Laboratorio ove si raccolgono i dati	
Acceleratore usato	
Fascio (sigla e caratteristiche)	
Processo fisico studiato	
Apparato strumentale utilizzato	
Sezioni partecipanti all'esperimento	Roma1, Roma2, Pisa, Parma, Milano
Istituzioni esterne all'Ente partecipanti	DESY, CERN, Univ. de Paris Sud (Orsay)
Durata esperimento	

B) SCALA DEI TEMPI: piano di svolgimento

PERIODO	ATTIVITA' PREVISTA
2001	Conclusione progetto
2002	Prototipaggio
2003	Produzione

Mod. EN. 1

(a cura del rappresentante nazionale)

ISTITUTO NAZIONALE DI FISICA NUCLEARE

Preventivo per l'anno **2001**

Nuovo Esperimento	Gruppo
APE (Pr.Sp.)	5

Struttura
ROMA I

PREVENTIVO LOCALE DI SPESA PER L'ANNO 2001

In ML

VOCI DI SPESA		DESCRIZIONE DELLA SPESA	IMPORTI		A cura della Comm.ne Scientifica Nazionale
			Parziali	Totale Compet.	
Viaggi e missioni	Interno	Viaggi Pisa, Milano, Parma GRID (1 fte)	20 6	**26**	
	Estero	Viaggi Berlino, Parigi, USA Data GRID (0.3 fte)	90 6	**96**	
Materiale Consumo		Produzione VLSI Licenze, fisiologia	2100 100	**2200**	
Trasp.e facch.					
Spese Calcolo		Consorzio / Ore CPU / Spazio Disco / Cassette / Altro			
Affitti e manutenz. apparecchiat.					
Materiale Inventariabile		WS Strumentazione PC's	100 100 50	**250**	
Costruzione Apparati		Prototipi PCB e meccanica	860	**860**	
		Totale		**3432**	

Note:

(a cura del responsabile locale)

ISTITUTO NAZIONALE DI FISICA NUCLEARE

Preventivo per l'anno **2001**

Nuovo Esperimento	Gruppo
APE (Pr.Sp.)	5

Struttura
ROMA I

PREVISIONE DI SPESA: PIANO FINANZIARIO LOCALE

PER GLI ANNI DELLA DURATA DEL PROGETTO

In ML

ANNI FINANZIARI	Miss. interno	Miss. estero	Mater. di cons.	Trasp.e Facch.	Spese Calcolo	Affitti e manut. appar.	Mat. inventar.	Costruz. apparati	TOTALE Competenza
2001	26	96	2200				250	860	**3432**
2002	20	80	100				100	1800	**2100**
2003	20	80	100				100	2000	**2300**
TOTALI	**66**	**256**	**2400**				**450**	**4660**	**7832**

Note:

Osservazioni del Direttore della Struttura in merito alla disponibilità di personale e di attrezzature:

L'esperimento non utilizza le risorse della sezione

Mod. EN. 3

(a cura del responsabile locale)

ISTITUTO NAZIONALE DI FISICA NUCLEARE

Preventivo per l'anno **2001**

Nuovo Esperimento	Gruppo
APE (Pr.Sp.)	5

Struttura
ROMA I

PREVISIONE DI SPESA

Piano finanziario globale di spesa

In ML

ANNI FINANZIARI	Miss. interno	Miss. estero	Materiale di cons.	Trasp.e Facch.	Spese Calcolo	Affitti e manut. appar.	Mat. inventar.	Costruz. apparati	TOTALE Competenza
2001	59	175	2555				410	860	**4059**
2002	50	150	300				200	2200	**2900**
2003	50	130	300				200	3700	**4380**
TOTALI	**159**	**455**	**3155**				**810**	**6760**	**11339**

Note:

Mod. EN. 4

(a cura del rappresentante nazionale)

ISTITUTO NAZIONALE DI FISICA NUCLEARE

Preventivo per l'anno **2001**

Nuovo Esperimento	Gruppo
APE (Pr.Sp.)	5

Struttura
ROMA I

PROPOSTA DI NUOVO ESPERIMENTO

Vedi allegato n.1: Proposal nuovo esperimento apeNEXT

Mod. EN. 5

(a cura del rappresentante nazionale)

ISTITUTO NAZIONALE DI FISICA NUCLEARE

Preventivo per l'anno **2001**

Codice	Esperimento	Gruppo
	APE (Pr.Sp.)	5

Struttura
ROMA I

COMPOSIZIONE DEL GRUPPO DI RICERCA

N	RICERCATORI Cognome e Nome	Qualifica				Affer. al Gruppo	Percentuale
		Dipendenti		Incarichi			
		Ruolo	Art. 23	Ricerca	Assoc.		
1	Cabibbo Nicola			P.O.		4	20
2	Panizzi Emanuele				R.U.	5	50
3	Rapuano Federico	I Ric				4	60
4	Simma Hubert				S.Str.	5	40

Numero totale dei Ricercatori **4,0**

Ricercatori Full Time Equivalent **1,7**

N	TECNOLOGI Cognome e Nome	Qualifica			Percentuale
		Dipendenti		Incarichi	
		Ruolo	Art. 23	Ass. Tecnol.	
1	Bartoloni Alessandro	Tecn			50
2	Battista Claudia	Tecn			20
3	Cosimi Maurizio		Tecn		100
4	Lonardo Alessandro		Tecn		100
5	Michelotti Andrea		Tecn		100
6	Rossetti Davide		Tecn		100
7	Torelli Mario	I Tecn			100
8	Vicini Piero	Tecn			100

Numero totale dei Tecnologi **8,0**

Tecnologi Full Time Equivalent **6,7**

N	TECNICI Cognome e Nome	Qualifica				Percentuale
		Dipendenti		Incarichi		
		Ruolo	Art. 15	Collab. tecnica	Assoc. tecnica	

Numero totale dei Tecnici

Tecnici Full Time Equivalent

(a cura del responsabile locale)

ISTITUTO NAZIONALE DI FISICA NUCLEARE

Preventivo per l'anno 2001

Codice	Esperimento	Gruppo
	APE (Pr.Sp.)	5

Struttura
ROMA I

REFEREES DEL PROGETTO

Cognome e Nome	Argomento

MILESTONES PROPOSTE PER IL 2001

Data completamento	Descrizione

COMPETITIVITA' INTERNAZIONALE

LEADERSHIPS NEL PROGETTO

Cognome e Nome	Funzioni svolte
Tripiccione Raffaele	Spokesman
Rapuano Federico	Responsabile nazionale
Marchesini Giuseppe	Responsabile locale Milano
Onofri Enrico	Responsabile locale Parma
Petronzio Roberto	Responsabile locale Roma2

(a cura del responsabile nazionale)

apeNEXT: A MULTI-TFLOPS LGT COMPUTING PROJECT

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.

A. Bartoloni, C. Battista, N. Cabibbo, M. Cosimi,
A. Lonardo, A. Michelotti, F. Rapuano, B. Proietti,
D. Rossetti, G. Sacco, S. Tassa,
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.

Ph. Boucaud, O. Pène
Laboratoire de Physique Théorique
Université de Paris-sud (Orsay)

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

P. De Riso, 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,
N. Paschedag, U. Schwendicke, H. Simma,
R. Sommer, K. Sulanke, P. Wegner
DESY, Platanenallee 6, D-15738 Zeuthen, Germany.

A. Fucci, B. Martin, J. Pech
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.

A. Petricola
Dipartimento di Ingegneria Elettrica, Università de l'Aquila and
INFN, Laboratori Nazionali del Gran Sasso, Assergi, Italy.

ABSTRACT

This paper presents the basic motivations and ideas of a next generation LGT computing project. The goal of the project, that we refer to as **apeNEXT**, is the construction and operation of several large scale Multi-TFlops LGT engines, providing an integrated peak performance higher that 10 TFlops, and a sustained (double-precision) performance on key LGT kernels of about 50 % of peak. The software environment supporting these machine is organized in such a way that it allows relatively easy migration between **apeNEXT** and more traditional computer systems. We describe the physics motivations behind the project and the hardware and software architecture of the new LGT engine. Several appendices provide details on preliminary work.

1 Introduction

Several research groups in the Lattice Gauge Theory (LGT) community have developed LGT optimized massively parallel processors [1]. These systems have provided in the last decade a significant fraction of all compute cycles available all over the world for lattice simulations. In this framework, INFN and DESY have developed the APEmille parallel processor. APEmille is an LGT oriented massively parallel number-cruncher [2], providing peak performance of several hundred Gflops. The first APEmille systems have been commissioned in late 1999 and more machines will become available in the next months (see later for details).

We expect APEmille machines to become the work-horse for LGT computing in several laboratories in Europe in the next three-four years. It is however clear (and explained in details in a following section) that APEmille is unable to support serious LGT simulations at the level expected after the year 2003.

The continuing physics motivation to pursue numerical studies of Lattice QCD and the level of needed computing resources have been analyzed in details by a review panel appointed by the European Comittee for Future Accelerator (ECFA) [3]. We fully endorse the conclusions of the ECFA report (which can be regarded as an ideal introduction to the present document). In this paper we present a proposal for a new lattice QCD project that builds on the experience of the previous generation APE machines and tries to implement several of the recommendations of the ECFA panel. This paper is an enlarged and improved version of a preliminary proposal [4], submitted to the INFN Board of Directors in summer 1999.

The new project (that we refer to as **apeNEXT**) is characterized by the following architectural goals:

- an expected peak performance for large machines in excess of 5 TFlops, using double precision floating point arithmetics.
- a sustained (double precision) efficiency of about 50% on key LGT kernels (such as the inversion of the Dirac operator).
- a large on-line data storage (512 GByte to 1 Tbyte for large machines).
- input/output channels able to sustain a data-rate of $0.5 Mbyte/sec/Gflops$.
- a programming environment that allows relatively straightforward and easy migration of physics codes between **apeNEXT** and more traditional computer systems.

From the point of view of the organization of the project, the following points are in order:

- the **apeNEXT** architecture will be very closely optimized to LGT simulations. In other words, **apeNEXT** will be more tuned towards LGT than APEmille.
- The general know-how of APEmille, as well as several important building blocks, will be heavily re-used in the new project (properly rescaled to keep technology advances into account). This is a key point that we plan to leverage on, in order to shorten development time.
- We plan from the beginning the installation of several large machines at approximately the same time at several collaboration sites. (Collaboration membership is also somewhat enlarged in comparison with APEmille). Stated otherwise, we plan to build up very high processing performance for LGT (of the order of several tens of TFlops) by operating in a loosely coordinated way several machines.
- Provisions to facilitate an industrial exploitation ofthe project are not one of the stated goals of the project. We do see however that several building blocks of the project (most notably in the area of inter-node communications) may have an important impact on other areas of computing for physics (and, more generally, for cluster computing or farming). We will do our best to make our results reusable.

This paper describes the hardware and software architecture that we plan to develop. It does not cover the organization of the project, the proposed schedule of our activities and any financial issues. These points are considered elsewhere. The paper is organized as follows:

- **Section 2** discusses the physics goals of the project and their corresponding computing requirements (in terms of processing performance, data storage, bandwidth).
- **Section 3** briefly summarizes the APEmille architecture and substantiates the need for a new project.
- **Section 4** reviews similar planned or started projects.
- **Section 5** surveys the status and prospects of some enabling technologies for our project.
- **Section 6** discusses advantages and disadvantages of custom versus off-the-shelf technologies for the processing element of the new computer.

- **Section 7** presents the global architecture of our new massively parallel LQCD machine.
- **Section 8** describes the details of the processing node.
- **Section 9** covers the architecture of the interconnection network.
- **Section 10** discusses several possible options for the topology and the mechanical set-up of the system.
- **Section 11** is the first section on software. Here we describe the programming environment that we plan to develop for **apeNEXT**.
- **Section 12** is a matching section discussing the operating system and other system-software issues.
- **Section 13** reviews the design methodology that we plan to follow in the development of the system.
- **Section 14** contains our conclusions.

Several appendixes present details on the R&D activities already under way.

2 Physics Requirements

In the definition of the new project we keep a clear focus on a very limited number of important 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 [5]. New theoretical developments (domain wall fermions, Wilson-Dirac operators satisfying the Ginsparg-Wilson relation, etc.) are likely to be implemented in a way which has very similar computational characteristics as 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, with a reasonable target corresponding to $m_\pi/m_\rho \simeq 0.35$ (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 analysis, it is desirable that the machine performs 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 $N_L = 16, 20, ..., 32$ and 48, and $N_L \leq N_T \leq 2N_L$. For smaller lattices, as the required computing performance decreases, more traditional machines (such as PC clusters) or previous generation dedicated systems can be used.

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.

   To obtain a good floating-point efficiency for the execution of a given computation, the compute power and memory bandwidth should be balanced accordingly. This balance is usually measured in term of the parameter R, defined as the ratio between the number of floating-point operations and the corresponding memory accesses (in the corresponding data format). A processor is balanced for a given algorithm if the R value required by the algorithm is roughly equal to the R value allowed by the processor itself. In the case of the Dirac operator, which dominates usually the cost in LQCD computations, a typical value is $R \simeq 4$.

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 analyses on small lattices, a mesh of $4^3 \cdots 6^3$ processors may be considered.

   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), effective bandwidth[1] for remote communications must not be lower than $1/\rho$ times the 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 $n_L^3 \times N_T$ physical points and local time direction per processor (note that, to first approximation, $\rho \simeq 2n_L$).

   A nice and simple trick can be used in the computation of the Dirac operator to reduce the number of remote accesses. For the negative directions the Dirac operator involves terms of the type $U_\mu(x-\mu)\psi(x-\mu)$ where the fermion term ψ and the corresponding gauge matrix (U) must be fetched from the same place. We can therefore evaluate the product $U_\mu(x-\mu)\psi(x-\mu)$ on the remote node and transfer the result only. In brief, all remote accesses involving gauge fields disappear. Table 2 contains the ρ values corresponding to the evaluation of the Dirac operator using the above mentioned technique. We consider the comfortably increased ρ values as an useful safety margin, that could be exploited to increase

[1] including the effect of the start-up latency for typical packet lengths.

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.

the floating point performance of each node, at fixed remote bandwidth. Clearly the actual values of ρ which can be accepted must be studied more carefully (possibly simulating architectural details of the mechanisms that hide remote communications)

Linear lattice size	ρ
3^3	7.5
4^3	10
6^3	15
8^3	20
$3 \times 6 \times 6$	11.25
$3 \times 4 \times 4$	9

Table 2: Local vs remote memory accesses: this table is the same as the previous one, except that ρ is estimated taking into account the trick, described in the text, that reduces remote accesses. The last two entries refer to non-square sub-lattices that might be used when simulating a lattice of spacial size 48^3 on large machines with $16 \times 8 \times 8$ or $16 \times 12 \times 12$ nodes.

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, 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 associated to one fermion field (24 data words, corresponding to 192 bytes in double precision. We call this quantity a fermion equivalent - feq - in the following). A generous

$U_{ab}(x,\mu)$	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 3: 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: $(\sigma\cdot F)_{ab}^{\alpha\beta} = [(\sigma\cdot F)_{ba}^{\beta\alpha}]^*$.

estimate, leaving space for more sophisticated, presumably more memory intensive algorithms, is about $\simeq 200feq$ 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 $10feq$ 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 respectively), 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 3 (where the size of the relevant data structures are presented) and in table 4, where actual memory sizes are collected, under the assumptions of using double precision [2] throughout. From the first two lines of table 4, 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 $\geq 1Tbyte$ is a good compromise for both our case studies. Alternatively, one might consider two memory options: a *small memory*

[2] The necessity of double precision arithmetic in full QCD has been investigated in the literature [7] and will not be discussed here.

machine ($\simeq 500 Gbyte$) for full QCD and a *large memory* version ($1 \div 2 Tbyte$) for quenched studies.

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

Table 4: 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.

   For full QCD permanent storage of the configurations is required due to the computing effort needed to generate them. This is a storage-density (as opposed to bandwidth) problem which is independent of the machine architecture and should be discussed in a different context, with potential links with the GRID project, likely to be supported by the European Commission. 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 $24^3 \times 48$ [8]. If we assume a critical slowing down where computer time grows like a^{-7} [9], 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 The APEmille project

In this section we briefly review APEmille.

APEmille is the present generation APE project. It is based on the standard structure of a large array of processing nodes arranged at the edges of a three dimensional mesh and operating in SIMD mode.

At present, several medium-size installations are up and running, while several larger units are under construction (see table 5). Considering all large and small machines, the integrated peak performance available in fall 2000 will be about 1 Tflops at INFN and about 400 Gflops at DESY. The largest single system will have a peak performance of 250 (possibly 500 Gflops). Other institutions in Europe are procuring (or considering to procure) APEmille machines.

Site	peak performance	status
Rome	130 Gflops	running
Zeuthen	64 Gflops	running
Rome	260 Gflops	planned June 2000
Zeuthen	260 Gflops	planned December 2000
Milano/Parma	130 GFlops	planned Sept. 2000
Pisa	130 Gflops	planned Sept. 2000
Rome II	130 Gflops	planned Sept. 2000
Rome	520 Gflops	under discussion Sept. 2000

Table 5: A short list of some large APEmille existing installations and of the largest APEmille installations planned for the near future.

In a typical critical LGT kernel (a solver for the Dirac operator) coded in the high level TAO programming language, measured sustained performance in single precision is about 44% of peak perfomance and in double precision it is about 19 % of peak single precision (i.e. 80 % of peak performance in double precision). Higher efficiency can be obtained with more careful programming: we have pushed single precision performances up to about 58 % of peak speed writing the key portions of the Dirac solver in assembly.

In a later section, we will claim that an architecture a la APEmille continues to be a very good choice for LGT computing. We see however a number of problems in APEmille, all pointing to the development of a new generation system:

- **Peak Performance** APEmille machines can be made larger than we plan to build, but not too large. The largest system that can be assembled with the present hardware building blocks is a configuration of $8 \times 8 \times 32$ nodes, corresponding to 1 TFlops peak performance (APEmille systems can be configured in principle as $8 \times 8 \times 2n$ arrays). Still larger systems would need some minor hardware development and would probably be not convenient in LGT, since they have an unusual large number of nodes along one dimension.
- **Memory Size** The very large APEmille machine described above has 64 Gbytes memory. This is still several times lower than discussed in the section on requirements.
- **Floating-point precision** APEmille is basically a single precision machine (performance decreases by factors from 2 to four in double precision). As discussed above, double precision will be necessary in future large LGT simulations.
- **Little space for improvements** APEmille is architecturally very simple, since it relies on accurate and rigid hardware synchronization. This style of synchronization is difficult to support if the system clock is increased significantly. For this reason, we see little space for incremental improvements in performance.

4 A review of similar projects

In this section, we gather some information on similar projects, carried out by other groups. To the best of our knowledge, the following activities are in progress:

- **CP-PACS**

  The CP-PACS collaboration have made a feasibility study of a future project which follows the CP-PACS project. Extrapolating the data of the performance obtained in recent full QCD simulations on the CP-PACS computer, they have estimated the computer time required for a large-scale full QCD calculation, with the quality of data comparable to that of the present quenched QCD study on the CP-PACS. They assume that lattice action and the simulation algorithm are identical to the present simulation on the CP-PACS. Their estimate, $\simeq$ 100 Tflops $\cdot$year [10], is somewhat larger than the one of the ECFA panel. In addition to their feasibility study, CP-PACS are carrying out basic research on the following two topics, that they consider as very important technologies for the next-generation of massively parallel computers:

  - Development of an architecture of high-performance memory-integrated processor for the next generation massively parallel computers.
  - Establishment of a model of parallel I/O, parallel visualization and man-machine interface, which can process efficiently and flexibly the enormous amount of data generated by massively parallel computers.

  CP-PACS hope they will have a chance to develop a next-generation computer using the results of their basic research in the near future, but they do not have a project at the present time.

- **Columbia** The Columbia group have officially embarked on the design and construction of their next machine [11]. The design effort is still on a fairly high level with choice of processor and communications technology being the first questions that have been resolved. Most significant is the choice of microprocessor, which is provided by an IBM PowerPC core. This follows from an arrangement with IBM that permits to exploit proprietary technology to construct a full processing node (memory included) on a single chip. This feature provides the name to the new project name, QCD on a Chip (QCDOC). The node will contain a PowerPC 440 core, one 64-bit, 1 Gflops FPU (an integrated part of the PowerPC architecture), 4 Mbytes of embedded DRAM and 8 bi-directional serial inter-processor links, each operating at 0.5 Gigabits/sec. If they are able to achieve this frequency, this would give a total off-node communications bandwidth of 1 Gbyte/sec.

  The group is now busy to determine the other details of the project and begin the detailed design of the node.

We also include an arbitrary selection of two (out of the many) interesting examples of PC-based cluster architectures for comparison.

- **The Wuppertal Cluster ALiCE**

  The "Institut für Angewandte Informatik" at Wuppertal University has installed the first half of the Alpha-Linux-Cluster-Engine (ALiCE) in 1999. When the system is fully installed, in May 2000, it will consist of 128 DS10 uni-processor workstations connected by a Myrinet multi-stage crossbar switch. All CPUs will be upgraded to 600 Mhz Alpha 21264 EV67 chips with 2 Mbyte second level off-chip cache [12].

The cluster is intended to perform efficiently in several HPC application profiles at the University of Wuppertal, including computational chemistry, electrical engineering, scientific computing and simulations of quantum field theories.

Of particular interest is the operability of this self-made system in a University's multi-user environment. In computer lab courses, the emphasis is on "Physics by High Performance Computers". Several student groups use the system simultaneously in interactive mode much alike a desk-top system.

A forward looking ALiCE-project, to be carried out together with the developers of the ParaStation communication software from Karlsruhe university, deals with optimization of efficiency and data organization for ALiCE under real life conditions, in particular with the goal to make parallel I/O and file system functionalities available.

- **PMS, The Poor Man's Supercomputer** A PC cluster has also been developed at Eötvös University in Budapest [13]. The current version of PMS has 32 PC's. Contrary to the previous example, the PMS project has developed QCD-optimized communication hardware. They use dedicated add-on boards to establish physical communications between nearest neighbour PC's in a three dimensional array. The actual configuration of 32 PC's can be imagined as a $2 \times 4 \times 4$ mesh of processors. The system uses a standard Linux operating system and the favoured programming style is the well tested SIMD paradigm.

  The present version of PMS is shaped by the requirement to reduce costs as much as possible. Indeed, PMS uses cheap AMD K6-2 processors (delivering only 225 Mflops each) while the special purpose communication interface has a bandwidth of just 2 MByte/sec. We consider the PMS as a very good trade-off between the advantages offered by the use of general purpose systems and the performance boost that dedicated hardware is able to provide.

5 Technological Scenarios

In this section we discuss forecasts about the state of the art for several enabling technologies in the years 2001-2002. We cover the following points:

1. basic digital VLSI technology.
2. memory technology.
3. data-links.

4. Off-the-shelf processors.

5. The Crusoe architecture.

We conclude the section with a discussion of the architectural implications of the technology-driven choice of overcoming the strictly synchronous operation of APE100 and APEmille.

5.1 VLSI technology

APEmille is based on a chip-set designed with a 0.5μ digital CMOS technology. A second source for the chip-set has been established, using a more advanced 0.35μ technology. In the next few years, 0.25μ and 0.18μ CMOS technologies will be readily available.

A comparison of some key features of the silicon technologies used in APEmille and of a representative of both 0.25μ and 0.18μ technologies is made in table 6.

Feature	ES2 0.5μ	Alcatel 0.35μ	UMC 0.25μ	UMC 0.18μ
V_{DD}	3.3 V	3.3 V	2.5 V	1.8 V
Gate delay	180 ps	100 ps	75 ps	36 ps
Gate density	$10K/mm^2$	$20K/mm^2$	$45K/mm^2$	$90K/mm^2$
Memory (1P)	$11Kb/mm^2$	$25Kb/mm^2$	$44Kb/mm^2$	$85Kb/mm^2$
Memory (2P)	$6Kb/mm^2$	$8Kb/mm^2$	$16Kb/mm^2$	$30Kb/mm^2$
Power/gate	$0.5\mu W/MHz$	$0.4\mu W/MHz$	$0.2\mu W/MHz$	$0.1\mu W/Mhz$

Table 6: A summary of some key parameters for digital silicon technologies used in APEmille and proposed for **apeNEXT**. All values are directly obtained from the relevant silicon foundries, except for the bit density of 1 Port or 2 Ports memory arrays in the UMC technology. The latter are based on conservatively applied scaling rules

The figures quoted in the table refer to processes that are (or will be) readily available through the same European silicon broker that helped us develop the second source of the APEmille chip set.

Let us consider a scaled version of the APEmille processor. If we use a 0.18μ process, it should be easy to reach a clock speed between three to five times higher than in APEmille, while we may expect to squeeze up to 9 times more transistors onto the same silicon area. We can stay on the safe side planning to use a clock frequency of $200MHz$. An LGT optimized processor running at this clock frequency with one floating-point pipeline would peak at 1.6 Gflops , using the well known *normal* operation $a \times b + c$, performed on complex operands. A chip three times more complex than J1000 (and three times faster) would dissipate less than two times more power.

5.2 Memory Technology

We limit ourselves to memory systems used in future high-end PC's or low-end workstations. This choice (the same as APE100 and APEmille) should be the most effective to provide the highest level of integration, reduce costs and guarantee part availability.

In the near future, planned memory systems are either RAMBUS DRAM's or DDR SDRAM's.[3]

The DDR SDRAM (Double Data Rate Syncronous DRAM), is the evolution of the mature SDRAM (Syncrounous DRAM) technology (widely used in the APEmille machine). The SDRAM is a low latency burst oriented device made of multiple (2 to 4) banks of asynchronous DRAM controlled by a synchronous controller which allows pipelining of the I/O interface (one word is accessed for every clock cycle). The Double Data Rate architecture realizes two data transfers per clock cycle using both edges of the clock and one special reference signal to fetch corresponding data.

The Rambus is a more advance memory architecture which works as a chip-to-chip system-level interface rather than a conventional memory device. The Rambus RDRAM (which stands for Rambus Direct Dram) shares the same architectural idea of the SDRAM one, a core asynchronous plus a synchronous controller. It makes use of a large degree of parallelism (32 interleaved memory banks) on a narrow internal bus. The Rambus RDRAM is based on the Direct Rambus Channel, a high speed 16-bit bus at a clock rate of 400 MHz, which thanks to the adoption of a dedicated signaling technology (Rambus Signaling Level) allows 600 MHz to 800 MHz data transfers.

In table 7 we summarize the main features of the two technologies, for currently available and next generation (less than 2 years from now) chips.

Some comments are in order:

- The simple architecture of the DDR SDRAM allows larger memory size per device. For a given fixed amount of memory, this reduces the number

[3] In the following we do not distinguish between traditional DDR SDRAM and new "flavour" DDR SDRAM like Sync-Link because both are evolutionary designs of the same basic structure.

	DDR	RDRAM	DDR	RDRAM
Data rate	200 MHz	800 MHz	400 MHz	800 MHz
Memory size	256 Mbit	128/144 Mbit	1 Gbit	256 Mbit
Organization	x4,x8,x16	x16,x18	x16,x32	x16,x18
Peak bandwidth	0.4 GB/s (x16)	1.6 GB/s	1.6 GB/s (x32)	1.6 GB/s
Package	TSOP(66)	BGA	TSOP(80)	BGA
Power (VCC)	2.5 V	2.5 V	1.8/2.5 V	1.8 V
I/O type	SSTL2	RSL	SSTL (?)	RSL (?)
Power cons.	80 mA	330 mA	?	?
Cost (norm.)	1.0	1.8	?	?
Sample/Prod.	Now/Now	Now/Now	3Q99/4Q00	?

Table 7: A summary of several important figures for two options of dynamic RAM's. The second and third columns refer to presently available DDR and Rambus devices. The fourth and fifth colums refer to the expected evolution of these devices in the next two years.

of used components.

- Since power consumption is proportional to the interface clock (a factor 4 between RAMBUS e DDR), aggregated memory systems using the DDR SDRAM reduce the global consumption.
- On the other hand the extremely high peak bandwidth of the RAMBUS allows to build a very fast memory system with minimum impact on board space occupancy (compact BGA packaging).
- The logic complexity of a RAMBUS interface is much larger than for a DDRAM controller (the latter could be easily designed on the basis of the experience done in the realization of the APEmille memory controller). On the other hand, several silicon foundries make a RAMBUS controller available as a core cell.

We conclude this section by presenting in table 8 two possible DDRAM-based memory systems for **apeNEXT**. The performance target is set by our basic performance figure, discussed in the previous subsection of $1.6 Gflops$ and $R = 4$, leading to a bandwidth requirements of at least $3.2 Gbytes/sec$ (assuming double precision data words throughout).

In conclusion, forthcoming memory technology is adequate to support the processor performance discussed above. There is in fact reasonable space to consider either fatter node processors, or multi-processor chips.

chip-size	1 Gbit	1 Gbit
chip organization	32 bits	32 bits
chip number	4	2
word size	128 bit	64 bit
bank size	512 Mbyte	256 Mbyte
frequency	300 Mhz	400 Mhz
total bandwidth	4.8 Gbytes/sec	3.2 Gbytes/sec
power consumption	640 mW	400 mW

Table 8: Basic features of two possible memory systems for **apeNEXT** based on DDRAM memory technology. Power consumption is estimated by re-scaling data available for present generation systems

5.3 Data-link Technology

We now consider *remote communications* which, in our opinion, is a key technological challenge for the project.

Assuming our reference figures - 1.6 Gflops per node, along with $R = 4$, and $\rho = 8$ (as defined in the previous sections) - we require an inter-processor communication bandwidth of about 400 Mbytes/sec. As discussed above, several code optimization steps are able to reduce the amount of data to be transferred. The overlap between computation and communication can also be increased. All this steps reduce bandwidth requirements. We will stick however to the previous figure, so a large safety margin is established.

The needed communication patterns are however very simple: communications are needed between nearest-neighbours (L-shaped paths, between next-to-nearest neighbours are also useful) in a 3-d array of processors, where each processor has 6 direct links to its nearest neighbours. The real challenge in this area is therefore more the implementation of a fast, reliable and cheap link than the development of any clever routing strategy.

In APE100 and APEmille, links use large, parallel and synchronous data paths. Data words are injected at the transmitting end of the line following a rising transition of the clock and are strobed into the receiving end of the line at the next rising edge of the clock. This works if

$$T_t < T_{clock} \tag{1}$$

$$\Delta T_{clock} << T_{clock} \tag{2}$$

where T_t is the travel time over the physical link, T_{clock} is the clock period and ΔT_{clock} is the phase spread between (nominally aligned) clock signals at various places in the machine. The conditions are met in APEmille, where $T_{clock} = 30ns$, $T_t \simeq 10ns$ and $\Delta T_{clock} \simeq 4ns$, while they become clearly unrealistic for frequencies of $\simeq 200Mhz$.

More advanced (high bandwidth) link technologies have recently become available, in which data and timing information are both encoded on the physical link, so asynchronous operation is possible. In the bandwidth range relevant for us, we have considered three different options:

- **Myrinet-like links.** The physical layer of the Myrinet interconnect uses low swing single-ended signalling. One byte is encoded onto ten signal lines, carrying also timing information. The full duplex link uses two such busses. The present generation Myrinet link has a bandwidth of 160 Mbytes/sec (using both edges of an 80 Mhz clock), while a new generation (Myrinet-2000, 320 Mbytes/sec) is under test. The main advantage of Myrinet links is that they pack a lot of bandwidth while keeping operating frequency low. Board layout details, connectors and cables are also very well tested. We are informally discussing with Myricom the possibility to use this link for **apeNEXT**. Myricom have agreed to allow us to use the link level (SAN-port) circuitry for their latest Myrinet chips (Myrinet 2000) as a basis for the **apeNEXT** links. Under a suitable non-redistribution agreement, Myricom will make available to the collaboration the layout of the basic cells, along with their Verilog models.
- **LVDS based links.** The Low Voltage Differential Signalling (LVDS) technology is now widely used in many telecom and network technologies, like the Scalable Coherent Interface (SCI). LVDS is designed to work up to $\simeq 622MHz$. Several redundant encoding schemes (e.g., 8 bits into 10 bits) have been proposed. LVDS cells are readily available from several silicon vendors. New generations FPGA chips have been announced including LVDS options. Work is in progress to test LVDS links, as described later in this document.
- **High speed proprietary links.** Several silicon houses (e.g., Texas Instruments (TI), National Semiconductor(NSC), LSI Logic) have developed very high speed proprietary links, aimed at the Gb Ethernet, Fiber-Channel, Infini-Band markets. The typical bandwidth is higher than 1 Gbit/s. Complete encoding-decoding black-boxes are usually available. This option has two main drawbacks: it makes the whole project dependent on a specific silicon house, and requires very careful layout of the printed circuits and proper choices of cables, connectors and the like.

Basic figures of the three options are compared in table 9, where we use for the LVDS case a more conservative frequency of 400 MHz.

Technology	Frequency	Pins	Bandwidth	Power Dissipation
Myrinet	160 Mhz	20	320 Mbyte/s	300 mWatt
LVDS	400 Mhz	40	400 Mbyte/s	200 mWatt
SerDes(TI)	1.24 Ghz	10	400 Mbyte/s	400 mWatt
NSC DS90CR483/484	784 MHz	18	672 Mbyte/s	1500 mWatt

Table 9: Basic figures for several link technologies. All figures refer to full duplex links. An 8 bit into 10 bit encoding has been assumed for the LVDS case.

An important issue is the reliability of the network, usually measured in BERR (average number of errors for transmitted bit). If we require fault-less operation of a large machine for one day (say, 2000 links active for 50 % of the time), we need a very low value of $BERR \simeq 10^{-17}$. For comparison's sake, **measured** stable operation of an APEmille machine with 250 nodes for periods of a few days implies $BERR \leq 10^{-15}$.

Machine reliability greatly improves if the network is able to recover from network errors by re-trying a failed communication (this impacts on link latency, but the impact can be made low with some care). For instance a comfortable $BERR \simeq 10^{-12}$ implies that one communication must be retried on the machine every second.

The above discussed feature however requires some degree of non asynchronous operations, with important technological implications. 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 consider 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.

5.4 Off-the-shelf processors

In this section, we briefly consider of-the-shelf processors as a potential building block for the computational core of **apeNEXT**. With one notable exception (see later), we choose to consider only the option of using commercially available boards (in other word, if we decide to use a commercially available option, we want to drop altogether any hardware development not involving the network). In the following section we will compare the relative merits of off-the-shelf versus custom processors.

Standard off-the-shelf processors have increased in performance by more than one order of magnitude in the last 8-10 years, with an even more remarkable improvement in the efficiency of floating point computations. Standard PC boards using off-the-shelf processors have been used for small scale LGT simulations. The relevant codes are written in familiar programming languages, like C (or C++) or Fortran. Efficencies are limited by bottlenecks in memory access as soon as the data base involved in the computation exceeds the cache size (which is the typical situation in realistic LGT simulations). These effects are discussed in more detail in Appendix D. Here we only quote the main conclusion that measured efficiencies on a Pentium II processor running at 450 Mhz are of the order of 30%, for real-life production programs (running on just one node, i.e., with no communication overheads) [14].

A detailed discussion of the expected technical road-maps for off-the-shelf processors in the next few years in general terms would exceed the scope of this document. Instead, we discuss the features of a typical high end microprocessor, that might be used today and apply usual scaling laws. For this purpose, we (rather arbitrarily) take the AMD Athlon. A number of features relevant for LGT simulations are shown in table 10.

Clock frequency	600 - 1000 Mhz
F. point ops (single precision)	4 per clock cycle
F. point ops (double precision)	1.6 per clock cycle
F. point latency	15 clock cycles
L1 Data Cache	64 Kbyte
Data bandwidth to L2 cache	1.6 Gbyte/sec
Sustained LGT performance	360 Mflops
Power consumption (750 Mhz)	35 W
Retail price (600 Mhz)	200 Euro
Retail price (750 Mhz)	375 Euro
Retail price (800 Mhz)	500 Euro

Table 10: Features of the AMD Athlon processor relevant for LGT simulations. Sustained performance is estimated under the assumptions discussed in the text.

In the rest of the discussion, we consider the version of the Athlon running at 750 Mhz. Indeed, Fig. 1 shows that for higher frequency, power dissipation increases faster than linearly.

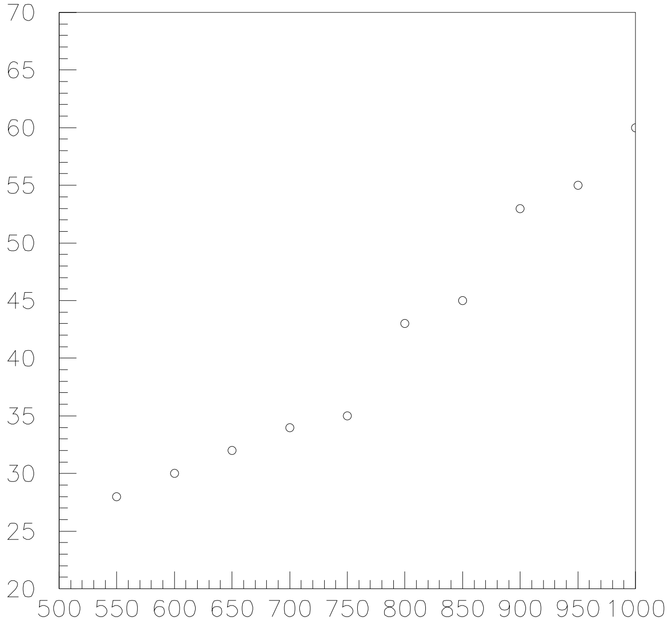

Figure 1: Power consumption (W) of the AMD Athlon processor as a function of the clock frequency (MHz) [15].

If we assume an efficiency comparable to the one measured on Pentium systems, we expect a sustained LGT performance of $\simeq 360 Mflops$ per processor. As discussed in appendix D, we can probably use dual-processor motherboards without jeopardizing efficiency (a quad-processor system would saturate the maximal theoretical bandwidth of 1.6 Gbyte/sec to access a memory bank working at 200 Mhz assuming our usual value of $R \simeq 4$).

In summary, a high end PC-like node should be able to sustain a performance of $\simeq 700 Mflops$ running LGT codes in double precision. We can take this as our basic building block, with just a few relevant figures summarized in table 11

This nodes needs a sustained interface to neighbour nodes in the three directions of the lattice grid with a bandwidth of $\simeq 200 Mbytes/sec$.

Sustained performance	700 Mflops
Power dissipation	90 Watt
Tag price	1500 Euro

Table 11: Basic figures for a PC-based node of an LGT engine, using currently available off-the-shelf hardware. Price estimates are made at current retail prices. They include 512 Mbyte main memory. No LGT networking or infrastruture is considered.

In conclusion, a system delivering 1 Tflops sustained LGT performance would cost more than 2.2 MEuro in processors only, and dissipate more than 130 KW power. We will discuss the implications of these numbers in the following section.

5.5 The Crusoe architecture

Very recently a new processor architecture (known as the Crusoe) has been proposed by Transmeta Corporation. The Crusoe is advertized as as a streamlined (hence very low consumption) processor, optimized for laptops or other mobile computers. The Crusoe has a very simple architecture, that, when used behind a core-level software environment, emulates the Intel X86 architecture. From our point of view, it is more relevant that thearchitecture of the Crusoe is extremely similar to the combination of the processing chips used in APEmille. Basically, the Crusoe core is a micro-coded system in which several functional units operate concurrently on data coming from a medium-size register file (see fig. 2). The chip has also a data instruction cache, as well as two different memory interfaces.

A high-end implementation of the Crusoe (advertised as available from Summer 2000) is called the TM5400. It runs at 500 (maybe 700) Mhz and dissipates about 2.5 W, when running at full speed.

At present, no Crusoe-based boards are available. It is likely that the first commercial products using Crusoe processors will be laptop machines, that obviously do not meet our requirements. We have therefore to consider the option of building a Crusoe-based **apeNEXT** processing board.

The main advantages of this choice are basically summarized by saying that we would be using an architecture very similar to APE, while being spared the burden of designing our own processor.

We have contacted Trans-Meta to explore this option. They stated that:

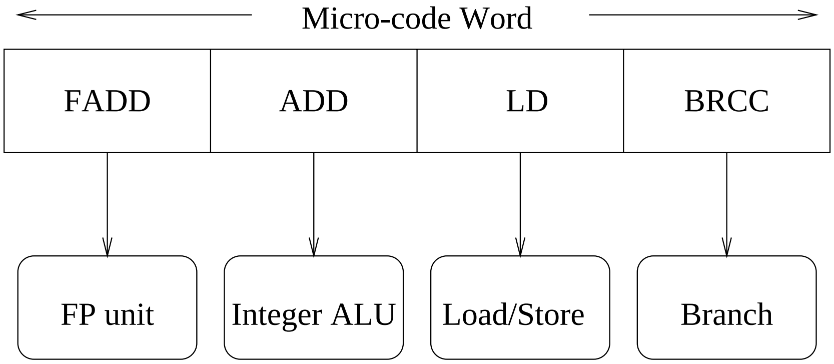

Figure 2: The Crusoe architecture (adapted from [16].

- They are not ready to provide critical details of the internal architecture (for instance, no information was provided on how many floating point operations can be executed at each clock cycle).
- Sufficient details of the VLIW core will not be given. Indeed Trans-Meta attitude is that all programming for the Crusoe must be done at the level of the Intel architecture, and must be translated with their proprietary software.

With these pieces of information available and considering also that:

- It is not clear whether chips can be procured at an early enough stage of the project.
- It is not obvious how fragile the whole Crusoe endevour is.

we think that the present situation does not suggest to base a new project on Crusoe. Of course, we will keep a close watch on any related development.

6 Custom or off-the-shelf processor

Previous generation LGT projects have used either custom processors, or substantial enhancements to standard processor architectures or processors developed for niche applications. No big project has been based on standard off-the-shelf processors sofar. Today, a decision to follow the same path is not as obvious as it has been in the past, since off-the-shelf processors have increased in performance by more than one order of magnitude in the last 8-10 years, with a remarkable and even more relevant improvement in the efficiency of floating point computations.

In table 12 we compare a few numbers relevant for APEmille, for the PC-based solution discussed in the previous section and for a custom-based **apeNEXT** architecture (in this case, we use several tentative numbers discussed in early sections).

—-	APEmille	**apeNEXT**: PC-based	**apeNEXT**: custom
Peak performance	500 Mflops	1200 Mflops	1600 Mflops
Sust. performance	250 Mflops	360 Mflops	800 Mflops
Power Dissipation	1.5 W	35 W	3.0 W

Table 12: Comparison of several key figures for APEmille processors and possible options for **apeNEXT**. We assume that a next generation custom processor has the same efficiency as APEmille.

No clear cut best choice emerges from these numbers. In general, we see advantages both in custom architectures and in PC-based architectures:

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.

We see at this point the need to make a clear decision between the two options: we decide to focus on the development of a LGT architecture based on an APE-like custom processing nodes, whose architecture is described in the next sections. We base our decision on the following points:

- we want to focus our project onto machines with very large performance. As explained earlier on, we will have to put together several machines to really arrive at a VERY LARGE scale.
- we think to be able to re-scale and re-use a large wealth of building blocks from APEmille, reducing the design time.
- We think that the commissioning of a very large PC-based system (involving several thousand PC's all over the collaboration) is a huge (and new for us) project in terms of hardware (thermal and power management) and software (control of a large network) issues for which we have no real background.

We obviously think that a PC-based system is still a viable alternative (discussed at some length in the preliminary proposal) for small or medium-scale systems. At this point in time, we do not consider however the development of such a PC-based cluster as a priority for the **apeNEXT** project. We are however willing to collaborate with any such project, making any **apeNEXT**-proper development that might be useful for a PC-based LGT cluster readily available for such purpose. To this end, two points are most important:

- We plan to design the network processor, supporting LGT-optimized point-to-point communication in such a way that it can be easily interfaced to a PC (say across a PCI interface). See the section on the network architecture for more details on this point.
- We start from the beginning the development of a programming environment that allows easy porting between PC-clusters and **apeNEXT** systems.

7 Architecture Outline

In this section, we outline an architecture, leading to standalone **apeNEXT** systems scalable from about 100 Gflops to about 6 Tflops peak performance.

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.

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 when requested by the logical consistency of the program (e.g., before exchanging data).
- Remote communications use FIFO-based weakly asynchronous connections between neighbouring nodes. The SIMD/SPMD programming style 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 brings several architectural advantages:

  1. It allows to use for the remote communications a programming style which is very similar to APE100/APEmille. The latter has the very convenient feature that no explicit distinction between local and remote memory accesses is required when coding a program.
  2. This programming style 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 in principle an independent component. As discussed, the communication interface is conceptually based on FIFOs, allowing "elastic" connections between nodes. This novel feature has to be carefully simulated, but no serious problem is anticipated here. We need a fast, yet cheap and reliable[4] data-link. Using $\rho \simeq 8$, we need links of $400 Mbyte/sec$. As discussed in the section on technology, two or three different solutions are available.

[4] 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.

As discussed in the previous section, we focus our project on an implementation of the above outlined architecture based on a closely packed array of custom processors. We have in mind an implementation allowing to build systems of between about 1000 to about 4000 processors, along the following lines:

- Each node is based on a VLSI processor running 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. Each node has a private memory bank, based on commodity chips. Memory size per node is likely to be in the range 256 Mbyte - 1 Gbyte per node. The actual choice may be heavily affected by cost factors. 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 ($R = 4$). We are studying the possibility to increase performance by factors $2 \ldots 4$, by using some form of super-scalar or vector processing, in which several normal operations are performed concurrently.

- A typical large system has between $8 \times 8 \times 16 = 1024$ and $16 \times 16 \times 16 = 4096$ nodes. 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. The envisaged hardware structure of the machine is described in a later section.

- 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, as remarked earlier.

- 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 approximately 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 plan to take advantage of all handles offered by the non fully syncronous structure of the machine to relax the requirements and to simplify the structure of the host to **apeNEXT** interface.

  Basically we will hook the interface to just one or two nodes belonging to each **apeNEXT** board. (This can be done conveniently by connecting to the corresponding network interface). All complex patterns of input/output data movements, for instance relevant to a write onto disk of a “slice” of **apeNEXT** processors are best performed by assembling the data words onto the input/output nodes under program control, and then issuing a single data transfer to disk.

  We can load executable programs in a similar way, by first moving the code to the input/output nodes and then having a “loader program” to move the data onto the whole array.

  We need a lower level system able to access all nodes independently even if the neighbouring nodes do not work corectly. This system is needed for debugging and test purposes and (for instance) to start the “loader”. Speed is not relevant in this case, so well tested standard systems (such as the JTAG interface) can be freely used.

- 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.

In the following sections, we describe in more details some key components of our new system.

8 Architecture of the Custom node

In this section we present the architecture of a simple custom node for **apeNEXT**. The main idea guiding our design has been that of re-using bits and pieces of APEmille as long as possible, while re-scaling in performance as much as made available by technology improvements. We use this guidelines to help shorten the design cycle.

The custom node on which a large scale **apeNEXT** system is based is called **J&T**, since it combines the functionalities provided in APEmille by the control processor (T1000) and the mathematical processor (J1000). The combined processor shares just one memory bank.

A basic block diagram of the architecture is shown in fig. 3. The picture does not cover in details the memory and network interface. These points will be discusse later on. J&T is centered around the register file, whose structure

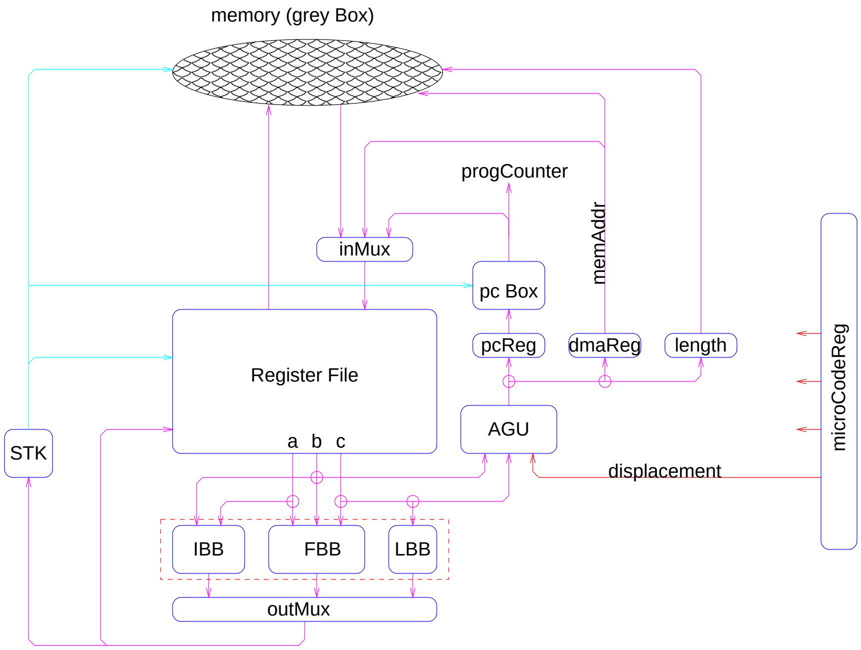

Figure 3: Architectural block diagram of J&T

is the same as the one used in APEmille. Data are transferred from memory to register file (and back) through a bi-directional port. Data available on the register file can be operated upon in just a few ways:

- Data words can be fed to the mathematical processor (the set of three units within the red dashed frame). The latter contains a floating-point datapath (Floating-point Building Block, FBB), an integer arithmetic unit (Integer Building Block, IBB) and a further unit providing first approximations of some useful mathematical functions, such as $\sqrt{x}, 1/x, \exp x$. This block is known as a Look-Up-Table Building Block (LBB) in APE jargon. Results of the mathematical block are written back to the register file (for later re-use or store onto the memory).

- Data can be moved to the Address Generation Unit (AGU), where a memory address or a branch-address can be computed out of two register-operands and one immediate-operand (the displacement). New (data or branch) addresses are stored in appropriate registers to be used at the next memory reference or branch.

- Logical tests can be evaluated on results computed from the mathematical processor. The outcome of such tests goes onto a stack where more complex logical conditions can be evaluated. The top of the stack is used to control program flow by acting on the program-counter circuitry (corresponding to `if (...)  then` in high level programs) or to block write operations onto memory or register file (`where (...)` clauses in APE-like high level programs).

The processor is controlled by a relatively large program word (called the Microcode Word) directly controlling the various devices in the node. (Almost) no instruction decoding is performed on chip. This scheme has been succesfully used in the node processors of both APE100 and APEmille. A word size of 128 bits is large enough to control the system.

In the following, we describe in more details several key units of the processor.

8.1 The memory interface and the network interface

In this section we describe the memory and network interface, sketched as memory grey box in fig. 3. A basic structure of this subsystem is shown in fig 4. The diagram shows several paths:

- there is a direct data path from the Register File to the physical memory interface (and vice versa), supporting normal memory access.
- Data from memory can be also fed to the Network Interface (and eventually routed to a remote node). Conversely, data arriving from the Network (from a remote node) can be routed to the Register File.
- Data words may be sent to the network from the register file. This is a novel feature, allowing registe-to-register remote communications. This feature reduces remote bandwidth requests in some cases (notably in the evaluation of the Dirac operator).
- The network interface receives data from the memory (or the registers) and route it to the appropriate destination through one of the six links (Details on the network itself will be provided later on).

In fig. 5 we further expand the Interface to physical memory. We see that memory is divided into cache memory and external memory:

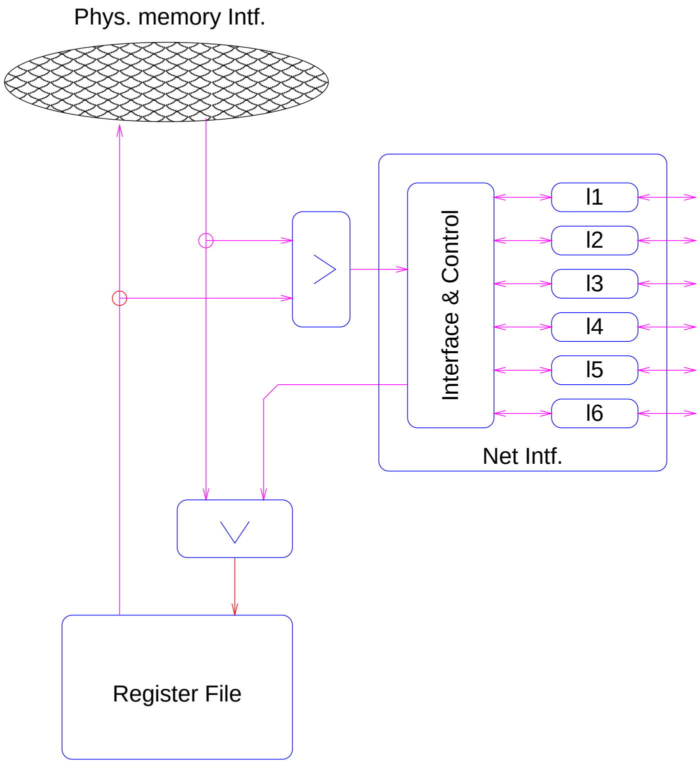

Figure 4: Top level block diagram of the memory and network interface.

- **external memory**. External memory implements the large memory bank of the node. As discussed in the section on technologies, we may use (for instance) DDR 1 Gbit memory chips. We have several options of memory bus width and bank size satisfying bandwidth constraints (see table 7). We want to leave these options open at this point in time. For this reason, figure 5 still has a grey box. This grey box contains the actual state machine controlling memory access, memory correction circuitry, refresh control circuitry and any other ancillary logic. The box will be designed in detail at a later stage of the project, after the actual memory technology has been selected. For the moment, we model the block by a simple interface in which data words coming from the memory are validated by an ad-hoc signal.
- **cache memory** A limited amount of on chip memory is needed in the node. Fast access on-chip memory will be used to store control variables (i.e. loop counters) and memory pointers. These variables were stored

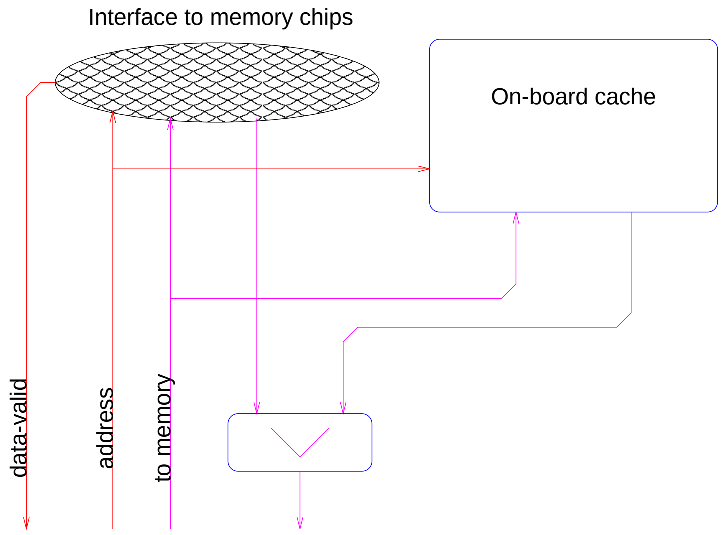

Figure 5: A lower level view of the memory system.

in the data-memory block of the control processor (T1000) in APEmille. Indeed, these variables have very irregular access patterns and very short access bursts. The use of relatively long-latency dynamic memory would adversely impact performance. On-chip memory does not need to be very large, of the order of 1K data words. Note that, in spite of the name, this is not a true hardware controlled cache system, since the decision to store variables on-board or otherwise is statically made at compile time (one very simple strategy would be to store on-board all non vector integer quantities defined by a program).

In any case, as seen by the processor, the memory interface has a word-width of 128 bits (one complex double precision number) and provides one new word at each clock cycle in burst mode. Addressing is done on 64-bit boundaries (so real and integer variables can be stored efficiently).

8.2 The instruction cache

Actual LGT simulations typically spend an extremely large fraction of the running time in just a few critical loops. For instance, a full-fermion hybrid Monte Carlo code spends nearly 95% of the time in the kernel used to compute the value of the Dirac operator on the fermion fields. Under these conditions, an instruction-cache system should have very large efficiency. We may exploit this feature by storing node programs in the same memory bank as data, with obvious advantages in terms of pin-count, real-estate reduction and cost savings.

We consider a control word (micro-code word) of 128 bits, equal to the word size that can be fetched from memory at each clock cycle. We may modify the memory interface as shown in fig 11.

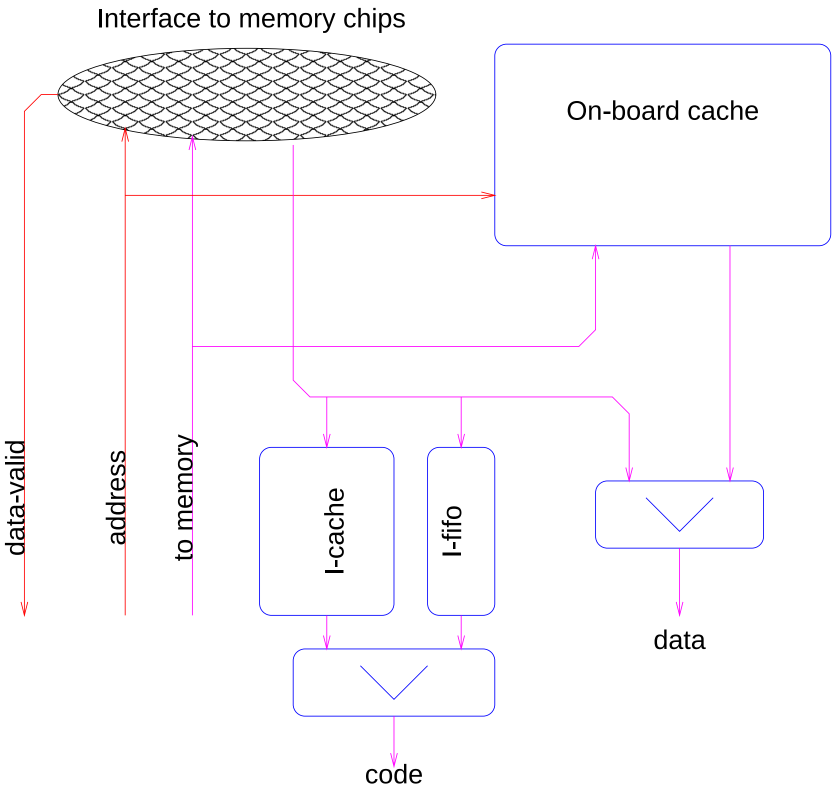

Figure 6: The instruction cache and the program look-ahead system.

Consider for the moment just the Instruction-Fifo. The memory controller (not shown in the picture) continously looks-ahead and prefetches instructions from the memory, at all machine cycles in which data-memory transactions are not in progress. Under the fully pessimistic assumption that all program cycles involve data-memory accesses, this mechanism reduces performance by a factor ≤ 2. Now consider the Instruction cache. The instruction cache is loaded (mostly simply, under program control; the program writer may advise the compiler through appropriate directives that some routine or do loop is a critical kernel to be loaded onto-cache) the first time the critical loop is executed.

The program then completes all following loops fetching instructions from the cache without incurring in any time penalty. The expected efficiency (ϵ) is (f is the fraction of cached program instructions):

$$\epsilon = \frac{1}{f + 2 \times (1 - f)} \tag{3}$$

If we expect to cache 90% of all used instruction (a rather pessimistic value) we may still expect 90% program efficiency.

The size of the cache needed to accomodate the computational kernels is an important parameter. We have estimated this value by analyzing several LGT kernels used in TAO and TAOmille physics programs. Results are shown in table 13. For each program, we list the size of the complete routine (labelled "complete"), the size of the critical kernel that could be "cut and pasted" immediately from the routine itself (labelled "medium"), and the size to which the critical routine could be shrinked with some re-programming effort ("basic"). For each routine we also record the minimum number of J1000 registers necessary to optimally schedule the program. This piece of information will be used later on. Note that no effort was made when writing these programs to keep code-size small. Inspection of the table shows that a cache size of the order of 16 Kwords is large enough.

kernel	min. registers	basic	medium	complete
Plaquette	166	2078	4139	6753
Dirac	194	736	1985	1985
Wilson	200	8989	11481	20000
LBE	130	-	2497	3467

Table 13: Code length (in machine cycles) and physical register usage of some LGT kernels running on APE100 and APEmille. The entries “complete, medium, basic” refer to the lenght of the complete routine, the lenght of the more time consuming loop and to the length to which the critical loop could be reduced with minor adjustments.

8.3 The register file

The register file has the same architecture as in APEmille. The register file has three read-only ports, one write-only port and one bi-directional port. The read-only ports are used to feed data to the mathematical processor, while the write-only port stores data from the mathematical processor onto the RF. The bi-directional port is used for memory access. All ports can be used at each clock cycle (five independent adresses are needed).

The word size of the processor is 64 bits, and complex numbers are stored as pair of (adjacent) registers.

The depth of the register file affects the performance of the node. If enough registers are not available, temporary results cannot be hold on-register. Memory bandwidth requirements increase and program efficiency decreases. In APEmille, 512 registers (or 256 register pairs) were used. Table 13 lists the number of registers used by critical LGT kernels programmed in TAO and compiled for APEmille. As we see, less than 256 are needed in all programs. We plan to design a register file of the same size as APEmille (512 lines). If we find out that such a large system does not work at the required speed, we know that the size can be halved without serious problems.

8.4 The mathematical processor

The computing engine contained in **J&T** performs three tasks:

1. it performs the floating-point (FP) (and, less frequently, integer) arithmetic operations heavily used in any scientific code. This is of course the most important functionality of the computing engine. All design trade-offs must provide the highest possible peak (and sustained) performance for this task. As already discussed, we will use the IEEE double precision format only. The system will be heavily optimized for the arithmetics of complex-valued numbers.

2. it computes first approximations of several important special functions (as already remarked, these functionalities are called LUT operations in APE jargon).

3. it performs all (mostly integer) arithmetic and logic operations needed to compute memory addresses. This task was carried out in a separate chip in APEmille, with dedicated hardware. We plan to share just one processor for this task and the previous one (and also for the fourth task, described below). As shown elsewhere, the price paid by this optimization in terms of performance is small.

4. it performs all arithmetic and logic operations supporting the evaluation of branch conditions. All considerations made before about addressing also apply here.

We plan, for obvious reasons, to re-use to a large extent the logical design and implementation of the arithmetic block used in APEmille (called FILU for Floating - Integer - Logic Unit). This goal is most easily reached by extracting from FILU the double-precision FP data path, the integer data path and the LUT circuitry, and building more complex operators as combinations of these building blocks. We recall that the FP data path performs the **normal** FP operation ($d = a \times b + c$) and conversions between FP and integer numbers, while the integer data path performs standard arithmetic and logic operations in integer format. We call these basic data paths the FBB (Floating Building Block), the IBB (Integer building Block) and the LBB (LUT Building Block).

Experience with the development of APEmille has taught us that a minor effort is needed to finalize the design of the IBB and LBB. Here we consider in details only the FBB. The architecture that we consider is shown in fig. 7. It uses data stored in the RF, that contains 256 register pairs. The two elements of the pair share the same address on each of the three ports. A complex operand has its real and imaginary parts stored on the same word of both registers, while a real operand sits on any location of either block. A vector operand finally is made up of two independent real values, stored in the same way as a complex operand. Vector operations can be effectively used in LGT codes for the generation of random numbers.

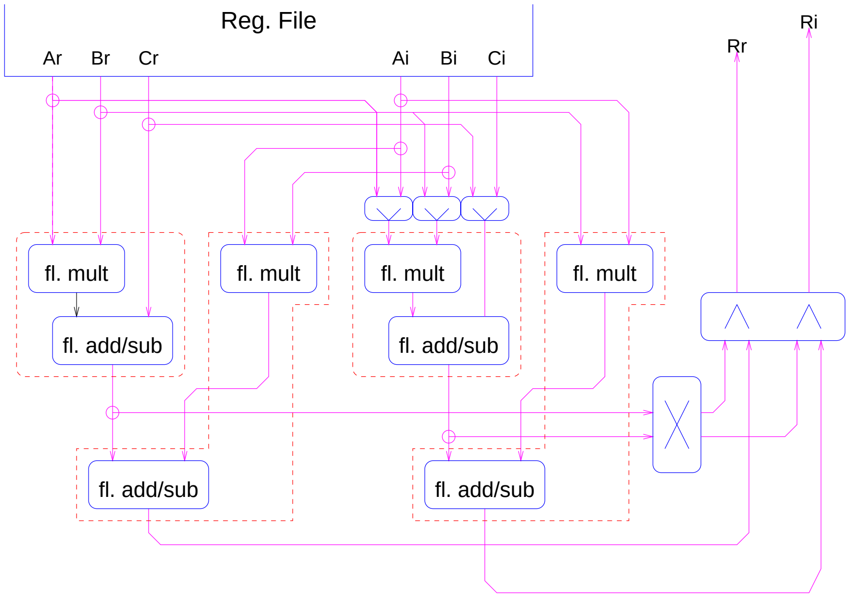

Figure 7: Block diagram of the floating point data-path (FBB) within the mathematical processor.

The FBB (see fig. 7) uses four basic floating point blocks, wired in such a way as to:

- compute the complex-valued version of the `normal` operation:

$$d.re = a.re \times b.re - a.im \times b.im + c.re \tag{4}$$

$$d.im = a.re \times b.im + a.im \times b.re + c.im \tag{5}$$

- compute one real-valued `normal` operation on operands coming from any element of any register.
- compute two real-valued `normal` operations on ordered operand pairs sitting on the right and left register banks respectively (vector mode).

Some basic figures of this architecture are collected in table 14. Control of the processor requires 5 bits in the microcode word and uses 4 RF ports. In total $8 \times 4 + 5 = 37$ control bits are needed.

type	performance	operands in RF
complex	1600 MFlops	256
real	400 MFlops	512
real vect.	800 Mflops	256
integer	200 Mips	512
int. vect.	400 Mips	256

Table 14: Basic parameters of the mathematical processor.

8.5 Performance Estimates

We have worked out some preliminary (but accurate) forecasts of the expected efficiency of the processor outlined in the previous sub-sections on a few computationally intensive kernels. The methodology used to reach these results is explained in appendix C. Our results are shown in table 15 for two versions of the kernel of the Dirac operator and for the main kernel of the Lattice Boltzmann Equation (LBE) solver.

The first two codes are appropriate for LGT programs, while the last kernel has been used for the simulation of turbulent fluid flows on APE systems. The two Dirac kernels refer to a simple program written in TAO (Dirac1) and to an accurately optimized code written in assembly (Dirac2). We see that in most cases we do safely better than in APEmille.

Kernel	normals	APEmille	**apeNEXT**
Dirac1	336	44 %	45 %
Dirac2	336	58 %	72 %
LBE	870	55 %	60 %

Table 15: Measured performance on APEmille and estimated performance of J&T on some critical kernels, described in the text.

We are still working to make our prediction more accurate and to test the efficiency of the processor on a larger set of computational kernels.

8.6 Implementation issues

In the following table we estimate the gate count of the largest logical blocks used in J&T. In the table we foresee a 0.18μ CMOS technology, as discussed in a previous section. Most values are evaluated by using appropriately scaled corresponding figures for APEmille and allowing large safety margins.

what	APEmille	scale Factor	**apeNEXT** (gates)	**apeNEXT**(mm^2)
Reg File	200 K	2	400K	5.0
Fl. Point.	100 K	2.5	250K	3.7
Intf.	30 K	2	60K	1.0
Data-cache	0	NA	$1K \times 128b$	4.4
Prog-cache	0	NA	$16K \times 128b$	34
Total	330 K	4.5 + cache	700K + caches	48

Table 16: Gate count and area estimate for the main components of the **apeNEXT** custom processor

Power dissipation for this system is less than 2.5 W at 200 Mhz (assuming that about 30% of the gates switch at each clock cycle). This processor fits into a reasonably small die and has a relatively small pin count.

Design of the processing node has already started. Details are given in appendix B.

9 The interconnection Network

The interconnection network is very sharply tailored to the needs of LGT simulations. The networks supports rigid data transfers between:

- nearest neighbour nodes in the positive and negative direction of the three axis (single hops)
- next to nearest nodes, whose node-coordinate differ by +-1 in two of the three dimensions. (double hops).

More formally, the network performs rigid shifts of the mesh of processors onto itself:

$$(x, y, z) \rightarrow (x + \Delta x, y + \Delta y, z + \Delta z) \tag{6}$$

where (x, y, z) labels the coordinates of each processor , $(\Delta x, \Delta y, \Delta z)$ are constant for all processors and $|\Delta i = 1|$ in no more than two terms.

Each link has a target bandwidth of at least 300 Mbytes/sec per link. Each node needs six links to support all the above described communication patterns.

From the point of view of system architecture the network is logically synchronous and support SIMD program flows, although at the layer of the physical link, no (wall-clock) time synchronization is needed.

This definition can be made more precise in the following way:

- Consider a SIMD program started on all nodes of the machine. Each and all nodes, while executing the program start a well defined sequence of remote communications. The sequence is the same for all nodes.
- we tag all remote communications by the following set of attributes:

  $$(\Delta x, \Delta y, \Delta z, S, N) \tag{7}$$

  where the Δ's were defined before, S is the size of the data packet associated to the communication and N is an identifier that labels all communications issued by each program (in the following, we call N the message-tag). N is initialized at 0, when starting the program and is incremented every time a new communication is started. In other words, N defines an ordering of all communications inside the program. Note that all attributes of each remote communication are equal on all nodes.
- The network interface of each node accepts data bound to a remote node and tries to send it to destination. Note that although all nodes necessarily

send the same sequence of packets, the (wall-clock) time at which a new data transfer starts may differ slightly among nodes. The following simple protocol controls the ensuing traffic:

1. Each network interface tries to start a remote communication originated from its node and identified by tag N only when it has received the data belonging to tag $N-1$.
2. Each network interface refuses to accept a data packet coming from a different node and tagged by N unless it has been already instructed by its own node to start transfering N.
3. Each network interface delivers incoming data in strict ascending N order.

This protocol is needed to make sure that all messsages reach destination in the appropriate ordering. As we see, very simple rules are needed to reach this goal under the assumpion that programs follow the SIMD paradigm.

Note that the network can perform several useful sanity checks:

- Once a node instructs the network to send a data-packet to a given destination, the network implicitely knows which packets it should expect on its links, with a given tag (for instance if a data-packet must be routed to `south - east`, then data with the same tag is expected from `west` for delivery at the local node and from `north` to be routed to `east`. The network interface can check that this is actually the case.
- The network can also check that the right sequence of tags is received within a (programmable) time-out delay.
- The network can further check that the data-sizes of all messages associated to a given tag are equal.

All these checks are important to help debug either ill-functioning hardware or wrong programs.

Error rates in the network are an important issue. In plain fact, we do not know the Bit Error Rate (BERR) that we may expected on fast links. We are only able to quote the upper limit on the BERR implied by the safe operation of the APEmille prototype (10^{-15}). The BERR value needed for an error free **apeNEXT** operation on runs lasting a few days is an embarassingly low 10^{-17}. For this reason, we have decided to stay on the safe side and to design a network partially able to recover from errors. If we are able to recover from errors a much

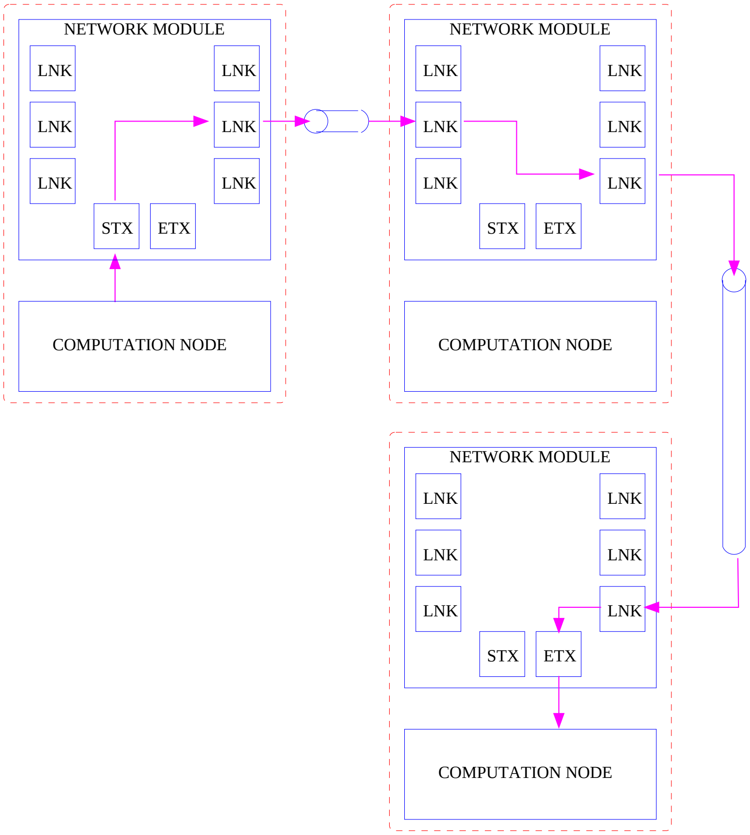

Figure 8: Architectural block diagram of the interconnection network.

more manageable picture emerges: for instance a more modest BERR $\simeq 10^{-12}$ implies the failure of one transmission burst every second on the whole machine.

We are considering a re-try mechanism, shown in figure 9 that has only a moderate impact on communication latency:

- We divide each data block travelling onto a remote link in relatively short bursts (say, 16 bytes) followed by a cyclic-redundancy-check (CRC).
- Data bursts are sent from the transmitting nodes, followed by their CRC. A small number of bursts is kept on the transmitting node, stored inside a FIFO queue, also after transmission.
- The receiving end of the link checks the CRC of each burst as it arrives. If the check is succesfull it delivers received data. The latency implied by

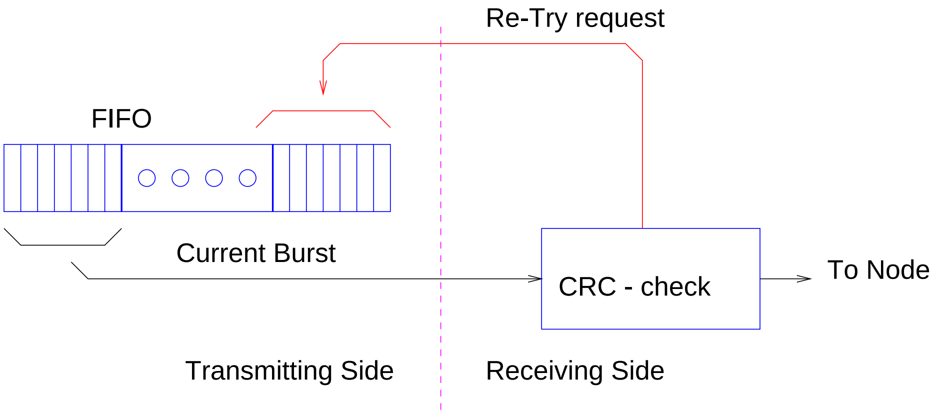

Figure 9: Low-latency re-try mechanism for the **apeNEXT** links.

this procedure is not longer that the size of each burst.

- If an error is detected, the receiving end requests the corrupted burst to be re-transmitted. This is possible, since relevant data is still available on the transmitting side of the link.

We plan to finalize most of the high level details of the network using a black box model of the physical link layer. In this way we can complete most of the design even before selecting the actual link technology. In parallel, we are already carring out tests on some of the link technologies.

10 Machine assembly and partitioning

We plan to assemble a certain number of APEmille processors on a printed circuit board (PCB). Preliminary evaluations suggest that 16 processors can be placed on one PCB, of roughly the same size as the one used for APEmille. For comparison, note that one APEmille PCB houses 8 processors. In this case however a large (almost 50%) fraction of the real estate is used by the control processor and ancillary circuitry. If we use PCB's of the same size as APEmille, we can re-use immediately the mechanical components of the older system.

At this point in time we have two options for the topology of the nodes belonging to one PCB. The first option is a three-dimensional structure with $2 \times 2 \times 4$ processors. The second option implies a two-dimensional set-up of 4×4 processors. PCB's are assembled inside a crate. All PCB's inside one crate are connected to a communication backplane. If we use the mechanical components developed for APEmille, we can reasonably house up to 16 PCB's inside one crate. Larger machines use more crates.

If we use the first option for the node topology inside a PCB, we can allocate onto the backplane all links in two of the three dimensions (say, directions y and z), building a system of size $4 \times 8 \times 8$. Communications in the x direction are implemented via cable links. If we assemble and connect together n crates, we obtain **apeNEXT** systems of size $(4 \times n) \times 8 \times 8)$. This option is very similar to the one used by APEmille, where systems of size $(2 \times n) \times 8 \times 8$ are allowed. We call this arrangement Option 1A.

It is possible to use the same structure of the PCB as above, wiring however the unit inside a crate according to a $4 \times 4 \times 16$ topology. In this case large machines contain $(4 \times n) \times (4 \times m) \times 16$ nodes. This is option 1B.

In the case that the second option for the PCB is selected, we allocate all links belonging to one of the spatial directions (say, direction z) onto the backplane. Links in the x and y directions stemming out of the PCB use cable links instead. Using this arrangement, systems of size $(4 \times n) \times (4 \times m) \times 16$ can be assembled. This is option 2. Some basic figures relevant to both options are collected in table 17.

-	Option 1A	Option 1B	Option 2
number of procs.	16	16	16
Peak PCB perf.	25.6 GFlops	25.6 GFlops	25.6 GFlops
topology	$2 \times 2 \times 4$	$2 \times 2 \times 4$	$1 \times 4 \times 4$
crate topology	$2 \times 8 \times 8$	$4 \times 4 \times 16$	$4 \times 4 \times 16$
Large-systems	$(2 \times n) \times 8 \times 8$	$(4 \times n) \times (4 \times m) \times 16$	$(4 \times n) \times (4 \times m) \times 16$
Remote links (PCB)	40	40	48
Remote links (BP)	32	24	32
Remote links (cables)	8	16	16

Table 17: Basic figures of three possible **apeNEXT** machine configurations.

In both cases, a large number of signal must be routed on the backplane. This is a serious but not formidable engineeering problem. Assuming that 20 data lines are needed per link, we have 640 pins carrying data from the PCB to the backplane (see again table 17). This requires as little as about 17 cm on the PCB edge, using high-density high-speed matched-impedance connectors developped by several vendors (see for instance [17]). Of course, special care must be taken in the design of accurately matched transmission lines, both on the backplane and on the main PCB.

As discussed earlier, the backplane must also house a CPCI backplane. This is made by a straightforward copy the well tested CPCI backplane developed for APEmille.

We will decide later on in the design phase on the selected topology, using information from test setups and taking also in consideration the relative merits of the two solutions from the point of view of physics simulations.

11 Software: The programming environment

The **apeNEXT** programming environment will be initially based on two main lines:

- The TAO programming language, extensively used in APE100 and in APEmille will be supported. This is necessary to allow easy and early migration of the large set of existing QCD programs on the new machine This large portfolio of programs is also going to be extremely useful for test and debugging purposes.

  We do not plan to make any substantial improvement to TAO. We will just modify the back-end section of the TAO compiler, so it produces **apeNEXT** assembly codes.

- We plan to develop a C/C++ language compiler for **apeNEXT** since the early phases of the project. The language will be a natural evolution of the GNU-gcc based compiler for APEmille [18] which is currently under evaluation.

  Very few extensions will be added to the standard C syntax, with the goal of minimising the effort for the programmer in learning a new language. SPMD parallelism will be realized by just a few special constructs, similar to the ones already present in TAO:

  1. the `where` instruction executes code based on local conditions.
  2. the `all, none, any` keywords in a standard C condition perform aggregate evaluation of local conditions.
  3. Remote communications will be specified by constant pointers.

  The compiler will be implemented by porting already available public domain compilers (like, for instance, the GNU C/C++ compiler or *lcc*) with the needed SPMD extensions in the front-end (the language definition) and all necessary changes in the back-end, to produce the target assembly.

> Note that, by using already available compilers, it will be relatively easy to implement all SPMD extensions of the language on more traditional machines (like PC clusters). Conversely, already developed parallel programs written in C (and following the SIMD/SPMD paradigm) will be easily ported onto **apeNEXT**. We regard this possibility as our main path to build a common programming environment between **apeNEXT** and more traditional systems.

We want to push still further the goal of a more general **apeNEXT** programming environment. We plan to merge to some extent the programming environments based on Tao and on C/C++, and at the same time enhance the portability of programs between APE systems and more traditional computer clusters. We plan to work according to the lines described in figure 10, that uses for definiteness the structure of the GNU compiler.

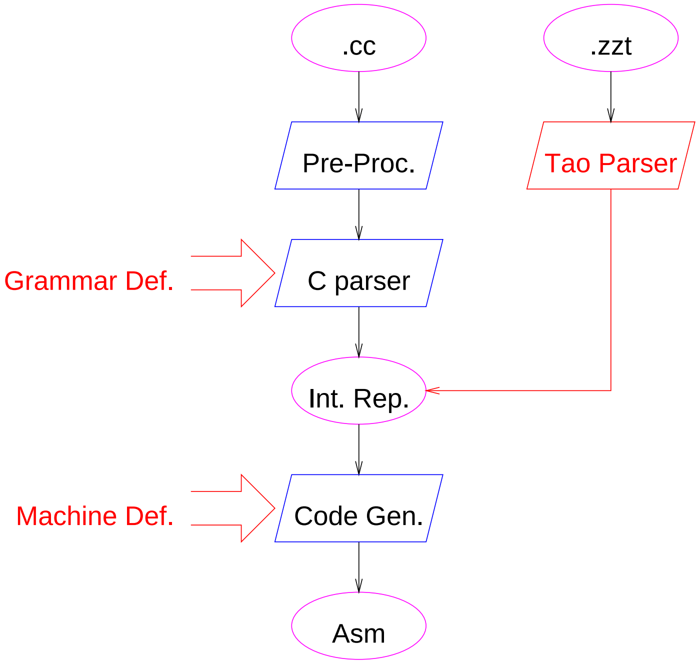

Figure 10: A sketchy view of the internal structure of the GNU compiler, including planned extensions for the **apeNEXT** software environment. Symbols in magenta are source or executable files. Symbols in blue are functional blocks. Synbols in red are **apeNEXT** specific extensions.

The blue boxes in figure 10 sketchily describe the overall organization of the standard GNU compilers. There is a front-end block with a configurable parser that transforms the user code into an internal representatation, based on a tree representation of the code and symbol tables. The back-end block maps the internal representation onto assembly code for a specific target architecture.

The APE C compiler can be implemented on the basis of existing and configurable front-ends with minor modifications to include the required syntax extensions for parallel processing. The back-end section must of course be custimized to produces **apeNEXT** assembly code.

It is also possible to add an additonal parser at front-end level (this is already done, in the GNU system, for the Fortran compiler). We think to follow this path to include the TAO parser, suitably modified to generate the GNU internal representation. Indeed, TAO cannot be easily handled by standard configurable parsers because of its dynamic grammar. In figure 10 the APE specific extensions are drawn in red.

When the program outlined above is accomplished, we will have a very neat portable environment in which:

- all powerful optimization techniques of the standard compiler core are available.
- TAO and C codes can be compiled for a standard computer system (e.g. a PC).
- C and TAO codes can be compiled for an APE system.

The design and implementation of this open programming environment is a long term and very high priority goal of our project. It is not going to be easy or fast. Physics exploitation of **apeNEXT** in the early phases does not depend on this environment, since the traditional APE software tools can be used.

At the machine level, we will port to the new architecture and improve the well-established VLIW code-scheduling and code-compressing tools already used in APEmille. Some preliminary results on this line are discussed in appendix C.

12 Software: The operating system

We plan to shape the **apeNEXT** operating system as a direct evolution of the APEmille system: the basic idea is that we use as much as possible the services provided by Linux on the network of host PC's.

- The **apeNEXT** operating system must load executable codes on the array of processing nodes and start execution of the whole system. We remind that all stages of program compilation and optimization are performed on the host PC's (or on any other Linux machine).
- The second task performed by the operating system is the support for input-output operations requested by the executing program. Note here that these operations use the standard file systems available on the host PC's (or, on any networked disk server). Of course, large data transfers, where high bandwidth is needed, are performed in parallel by all PC's on local disks. Later on, we will make some additional remarks on this point.
- The third task performed by the operating system is the monitoring and control of all nodes at a low level. Typical examples include the inspection and setting of status-registers, the analysis of error conditions, explicit writes or reads to memory locations.

All functions described above are handled by the APEmille operating system in a reasonably efficient and user-friendly way. Most operations can be easily moved onto **apeNEXT** by re-writing only the lowest layer levels of the operating system, like device drivers or the functions mapping a specific operation on a specific node onto the appropriate PC. We expect therefore to be able to put to work quickly an early version of the system.

An area on which we would like to bring new ideas, not needed however for the early commissioning of **apeNEXT**, is some version of a parallel file system, where large field and propagator configurations can be stored in a standard format. This is an obvious starting point to allow the sharing of QCD configurations among collaborating groups. We see this work as a partial contribution of **apeNEXT** to the GRID project. Work on this line will be therefore coordinated with GRID.

13 Design Methodology

In designing the needed VLSI components as well as the overall system, we want to follow the methodology used in APE100 and APE1000, with a number of improvements to make it more efficient and faster. The main advantage of this methodology has been shown in APE100 and APEmille: in both cases all components of the machine were designed "first-time-right".

The main ideas behind our methodology are some informal implementation of "hardware-software co-design" techniques:

- We base our design on a VHDL model of a large and significant fraction of the whole system. The model contains all in-house developed systems as well as all off-the-shelf components. Initially, the model will be a very crude approximation to the actual system, gradually incorporating all details. This reference model is available at all collaboration sites.

- All VLSI (or FPGA based) components of the system are derived with high quality synthesys tool from the VHDL design. In the (hopefully rare) cases where some component cannot be synthesized from a VHDL description, a VHDL model is built anyway, and test vectors for the actual implementation are derived by the VHDL model. Non-VLSI parts of the systems (i.e., processing boards) will be modeled in VHDL by their designers.

- The VHDL model supports a reasonable approximation of the interaction with the host system (operating system).

- All software developments are immediately tested on the VHDL model. At an early stage of the design, tests will involve performance estimates on crudely modelled architectural choices. As the level of details of the model increases, actual programs, in all their intricacies, will be executed on the model, giving quick feed-back on any design detail.

As an improvement with respect to APEmille, we will insist on:

- a continuous availability of the model at all collaboration sites, so all members of the collaboration can easily monitor the effects of a design change made elsewhere. This can be achieved with reasonable effort by keeping a master copy of the model on an AFS cell available from all sites.

- an effort to allow access to the model from a basic version of the operating system, so that even the more physics-oriented members of the collaboration can exercise it.

- a systematic use of blind-test procedures: at all stages of the design phase, test sequences for any portion of the machine will be prepared and executed by someone who has not been directly involved in the design.

Finally, we address the issue of the design of some VLSI blocks that depend critically on some portion which is either not under our direct control or not completely defined at this stage. Examples of this situation are the details of the memory system (heavily dependent on the type of memory available at the time when prototypes are built) for the custom processor, or the actual choice for the physical layer of the interconnection links. Changes made in memory technology during the design lifetime of APEmille have indeed adversely affected that project, as large subsystems within J1000 and T1000 processors had to be modified to adapt to changing memory specifications.

We want to solve these problem by confining all details of the memory and link interfaces in a "grey box", that interacts with the rest of the design with some simple and rather general data and control path. In this way, a very large fraction of the design can be finalized independently, while the interface-specific blocks will be procured (if available from external sources) or designed in detail at the last moment.

14 Conclusions

This document has described physics requirements 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 an LGT-focused high performance engine. At the engineering level, we plan to use technologies similar to those used in APEmille. We think that this choice reduces development costs and risks.

In the near future, collaboration between groups active in LGT simulations will become tighter and tighter. For this reason we plan to work hard on the development of a software environment allowing easy migration between **apeNEXT** and more traditional computers.

Acknowledgments

Several people have helped shape the basic ideas of **apeNEXT** and contributed several important pieces of information. We would like to thank I. D' Auria, M. Loukyanov, S. Menschikov, D. Pleiter, A. Sapojnikov, T. Sapojnikova, L. Sartori, L. Schur. N. Christ, N. Eicker, Y. Iwasaki, T. Lippert and K. Schilling have provided valuable information on the status and perspectives of their projects.

A Ongoing and planned R/D activities

This section contains a sketchy list of ongoing and planned preliminary activities. A few selected items (which are at a more advanced stage) are covered in specific sections.

- DESY has already built and succesfully tested a data parallel link of moderately high frequency: 11 signal pairs have been operated at 250 Mhz for long periods (50 hours) with no errors. Such a system would move 400 Mbyte/sec (plus check bits + signalling) on 22 LVDS pairs. They are now working to replicate the same system on an APEX FPGA.

- DESY is also working on a second generation FLINK interface, using NSC DS90CR483 / 484 chips. For Flink purposes, it will give 200 Mbyte/sec. For **apeNEXT**, it can be pushed to 112 Mhz (interface frequency) and 48 bits (interface width) corresponding to 672 Mbytes/sec.

- CERN is working on unidirectional multiplexed LVDS and PECL based link technologies. The aim is to test the very high speed capability of these technologies on different cables and. Furthermore the link speed will be optimised to match fast-wide PCI interfaces. The link will be designed so other users can take benefit of this PCI to PCI fast connection.

- The second major effort at CERN should converge into the emulation of the switching mechanism of **apeNEXT** with a moderate to very high speed interconnections. **apeNEXT** requires 6 bi-directional links (or 12 unidirectional). This system can be integrated in the next generation of ALTERA FPGA chips that will have LVDS I/O pins. These chips are a very good test bed for the links and switching required by the project. How many links (bi-directional or not) and at what speed remains to be seen, but again a PCI interface capable to switch on several links is a valuable project also for other applications. This system may cover (although at higher costs and higher power consumption) most of the requirements of the **apeNEXT** network.

- The basic floating point element for a custom processor has been extracted from the APEmille floating point block. Detailed VHDL coding is under progress in Pisa.

B Preliminary Design of the apeNEXT Node

Work has already started on the design of the architecture of the **apeNEXT** node and on preliminary activities tied to the actual hardware implementation. In this section we briefly mention activities already in progress and at a fairly advanced stage. More details are usually found on the relevant www pages of the APEmille/**apeNEXT** intranet (`http://chimera.roma1.infn.it/intranet`).

We are working on the following lines:

- We have defined a preliminary version of the **apeNEXT** assembly language. The defined instructions are, in most cases, suitable extensions of the APEmille assembler. Several new instructions are needed to handle the different structure of the new machine, especially for memory access. Although some of the instructions will probably undergo adjustments as the details of the system are finalized, we think that most instructions are reasonably stable.

- We have started to outline the format and size of the microcode word that controls the processor at each clock cycle. We are currently keeping a few different versions of the microcode, that we use for performance estimations. See (`http://pcape2.pi.infn.it/ fabio/APEnext/instr_next.html`, and the next appendix for details.

- We have slightly modified the APEmille `shaker` program. The `shaker`, a key component of the APEmille compilation chain, schedules and compacts assembly instructions, trying to maximize device usage and filling of pipeline slots. The `shaker` can be configured for a wide range of architectures, by writing a configuration file. The file describes how assembly instructions are mapped on devices and on the microcode word, and the corresponding timing. We have started to prepare configuration files for our target architecture, using pessimistic guesses for a few parameters, so we should be able to produce **apeNEXT** executable code shortly. See also the next appendix.

- We have started the coding of the VHDL description of the processing node. Our first goal is the completion of a first version of the code, in which most blocks are described at rather high level, or only have very partial functionalities. As soon as this version is ready, we will start exercise it with the executable codes produced as described above. We will then proceed to develop more advanced versions of the model, gradually incorporating all details of the architecture.

- We have made a preliminary survey of the silicon technologies that might be used in **apeNEXT**. We have focused on 0.18 μ technologies. These technologies are now in a rather early stage of development, as far cells, pads, Synopsys models are concerned. They should reach a rather stable state in the first part of the year 2001. We have obtained design kits for the 0.18 μ processes of UMC (through IMEC) and of ST (through CMP). We plan to use these design kits early in the design process, to ensure that all components of the VHDL model can be mapped on silicon within the required time and area limits.

- We have completed the design of a prototype register file design, using a standard cell library available with the UMC technology. Our test design has the full size required for **apeNEXT**, namely 2 banks of 256 registers each. Each register has 64 bits. We have also added one additional input and one additional output port, so more complex node architectures can be considered.The target clock frequency of the design is 200 MHz. We are now performing the place&route of the design, in collaboration with IMEC. We are also considering to actually fabricate a test chip, taking part in a Multi-Project Chip (MPC) run, early in May.

C Performance Estimation for the apeNEXT node

We would like to estimate the performance of **apeNEXT** node on tipical QCD program kernel. To evaluate the performance of the node that we are designing we use the shaker tool, a static assembly instruction scheduler for VLIW machines, currently also used to generate executable program for APEmille. The result obtained is a static estimation of the number of machine cycles correpsonding to a kernel section of a physics code, used as benchmark.

C.1 The Shaker

The assembly code producted as output by a compiler or written by hand must be converted into executable code. In tmicro-programmed machines like APE this means converting each assembler instructions in a micro-program of several machine cycles. This is done by the program shaker.

The shaker schedules the assembly instructions of a program in order to optimize resource usage and reduce execution time. In the APE machines we can exploit as many vertical parallelism slots as the number of hardware pipes and as many horizontal parallelism slots as the number of devices that can work in parallel. In order to exploit all kind of parallelism we must keep busy the pipes and the devices for as many machine cycles as possible.

The shaker schedules the start of an instruction as all the nedeed resources, e.g. register operands, are available taking care of logical data dependencies of assembly instructions and hardware constraints.

The shaker uses a configuration file which defines the fields of the micro-word, the resources which are available to be used and the patterns or micro-programs of each assembly instructions. A tipical configuration file contains the following information:

- field name definitions and field value mnemonics
- field location specifications

- micro-word width specification
- micro-program definitions
- usable hardware resource specifications

A configuration file can be defined for each architecture that we want to test. this information is compiled into internal data structures used by the shaker engine.

The input to the shaker is a file containing the assembly instructions and logical dependecies beetween them. The output is a file containing for each assembly instruction the cycle at wich it may be scheduled without breaking logical or hardware dependencies.

C.2 Performance Evaluation

We have defined a shaker configuration file for the architecture of the **apeNEXT** node. Using the output assembly code product by the APEmille compiler or tipical code written by hand, we mapped it onto the **apeNEXT** assembly code supported by the shaker. The result obtained is an estination of the number of machine cycles needed to execute the program under test.

We used as main benchmark the core of the dirac operator. In table 18 we show the measure of efficiency of the new **apeNEXT** node compared with APEmille. The DiracNext3 program is an assembly program written by hand and Dirac and Bolzman are assembly code product by the xtc compiler, one of the available TAO compilers for APEmille. Measured efficiencies are the same as in APEmille or better. The last column shows the efficiency of the same programs in case that we include a second mathematical unit in the node. In this case efficiency drops somewhat, but the sustained performance is still much larger.

See http://pcape2.pi.infn.it/APE under the link **apeNEXT** shaker for more details.

D Performance of LGT kernels on PC's

The theoretical floating point performance provided by off-the-shelf processors has reached an impressive level during the last years. To understand how efficient actual QCD codes perform on a given architecture practical benchmarks

	APEmille	**apeNEXT**	**apeNEXT**II
DiracNext3	58%	72%	48%
Dirac (xtc)	44%	45%	32%
Bolzman (xtc)	55%	60%	42%

Table 18: Expected efficiency of the **apeNEXT** processor on several benchmark kernels.

are usually needed. In particular, the efficiency of memory access and remote communications may be critical for the sustained performance.

In order to disentangle for instance the effects of the multi-layered memory hierachy and the processor itself, we have performed a series of preliminary benchmarks on PC's. They include rather simple tests programs to investigate characteristic quantities such as the bandwidths for data access at the various cache levels, latency and throughput of arithmetic pipelines, concurrency of different functional units and the limits performance increas in case of PC's with multiple processors.

The measurement of pure data access times as a function of the burst length is shown in fig. 11 for a Pentium II at 400 MHz. The clear steps illustrate the different access bandwidths for data residing in different levels of the memory hierarchy (L1 cache, L2 cache, memory). For data access to L1 cache, the bandwidth is determined by the throughput of the load and store units of the processor. For large block sizes the bandwith plateaus at about 70 % of the theoretical value of the memory bus running at 100 MHz. By running the same measurements in parallel on both CPUs of a Dual-Pentium system, one finds that the total memory access rate of the two processors almost saturates the theoretical bandwidth (which may be distributed in a slightly asymmetric way among them).

As a basic benchmark for QCD-like applications, we used a carefully optimized C-code for the kernel of the Wilson-Dirac operator with SU(3) gauge fields on a L^4 lattice. The sustained performance as a function of L is shown in fig. 12. As expected from the above benchmarks for pure memory access, the performance is almost doubled when running an independent program on each of the two CPUs of a Dual-Pentium system. The sustained performance rapidly drops with larger lattice sizes, however it does not show a pronounced step structure as in fig. 11 because some fraction of the data can always be kept in cache. The effect of the cache is also evident from the decrease in performance when the L2 cache is disabled. Hence, it might be possible to increase the sustained performance by a more cache-conscious coding (e.g. by sweeping in a suitable order throught the lattice size). In general, it seems that a sustained performance of around 30 % should be realistic as a rough estimate for typical QCD applications based on the Wilson-Dirac operator.

The effect of remote communications has not yet been studied.

References

[1] for a recent review, see: N. Christ, “Proceedings of Lattice99”, Nucl. Phys. B (Proc. Suppl.), in press.

[2] R. Tripiccione “APEmille”, Parallel Computing 25 (1999) 1297.

[3] F. Jegerlehner et al., “Requirements For High Performance Computing for Lattice QCD: Report of the ECFA Working Panel”, Prepint ECFA/99/200.

[4] F. Aglietti et al., “Proposal for a Multi-Tflops Computing Project”, Rome Preprint 1255/99.

[5] 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.

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

[7] 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;

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

[9] See e.g. C.T. Sachrajda, summary talk at Lattice 99 (Pisa), hep-lat/9911016 and references herein.

[10] Y. Iwasaki, private communication. Also CP-PACS Collaboration, Internal report, 1999.6.20, to be published.

[11] N. H. Christ, private communication.

[12] N. Eicker and T. Lippert, private communication.

[13] F. Csikor et al., hep-lat/9912059

[14] H. Simma, “PC’s for Multi_TFlops LGT Compute Engines?”, Proceedings of the Workshop PC-NETS, INFN/TC-99/09.

[15] AMD Athlon Processor Data Sheet (Publication 21016), March 2000.

[16] see for instance: http://www.trasmeta.com/crusoe

[17] see, for instance: http://www.teradyne.com/prods/bps/vhdm/intro.html

[18] http://apemaia.roma1.infn.it/ betto/C++_apeNEXT.html

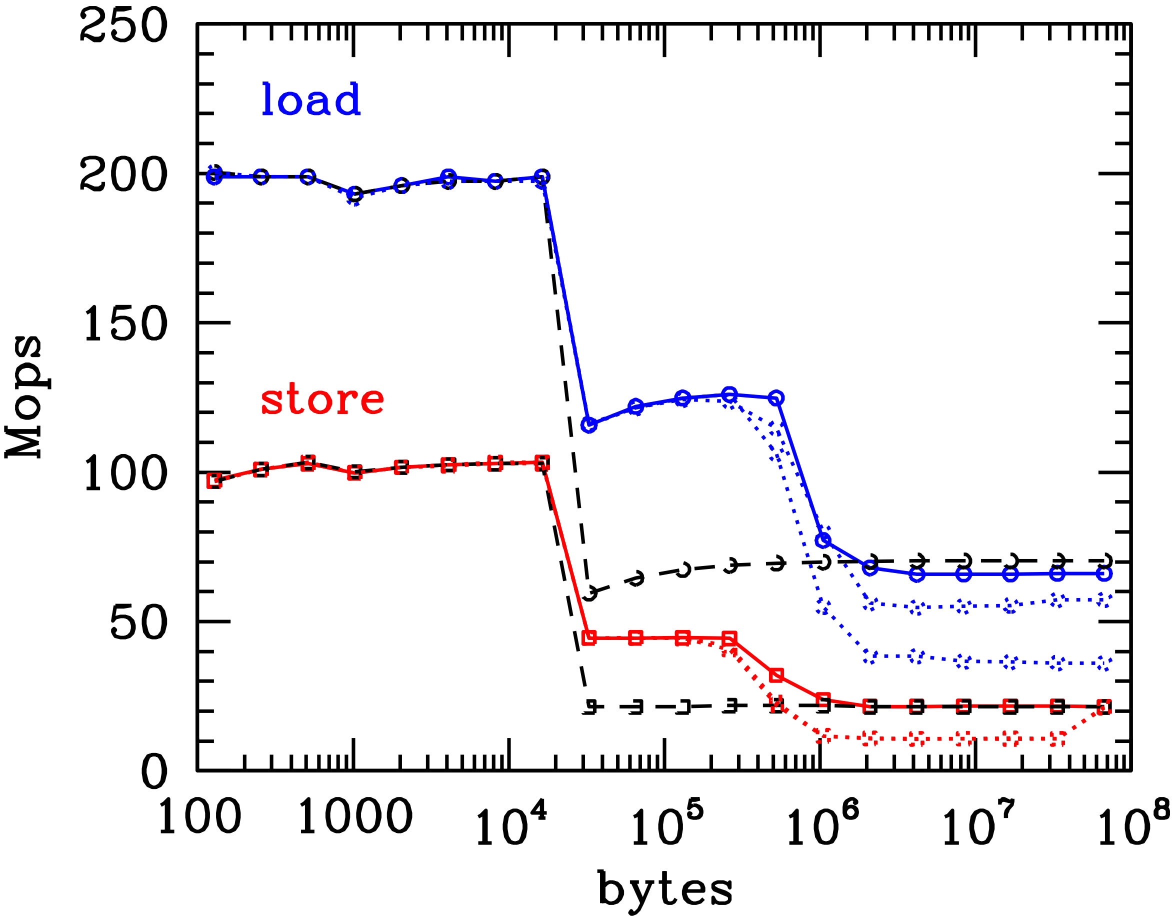

Figure 11: Measurement of cache effects for pure memory accesses with different burst lengths on a Pentiun II at 400 MHz. The upper and lower solid curves are for load and store operations, respectively. The dashed curves show the analogous measurements with the L2 cache disabled. Dotted lines are the results for simultaneous test execution on both CPUs.

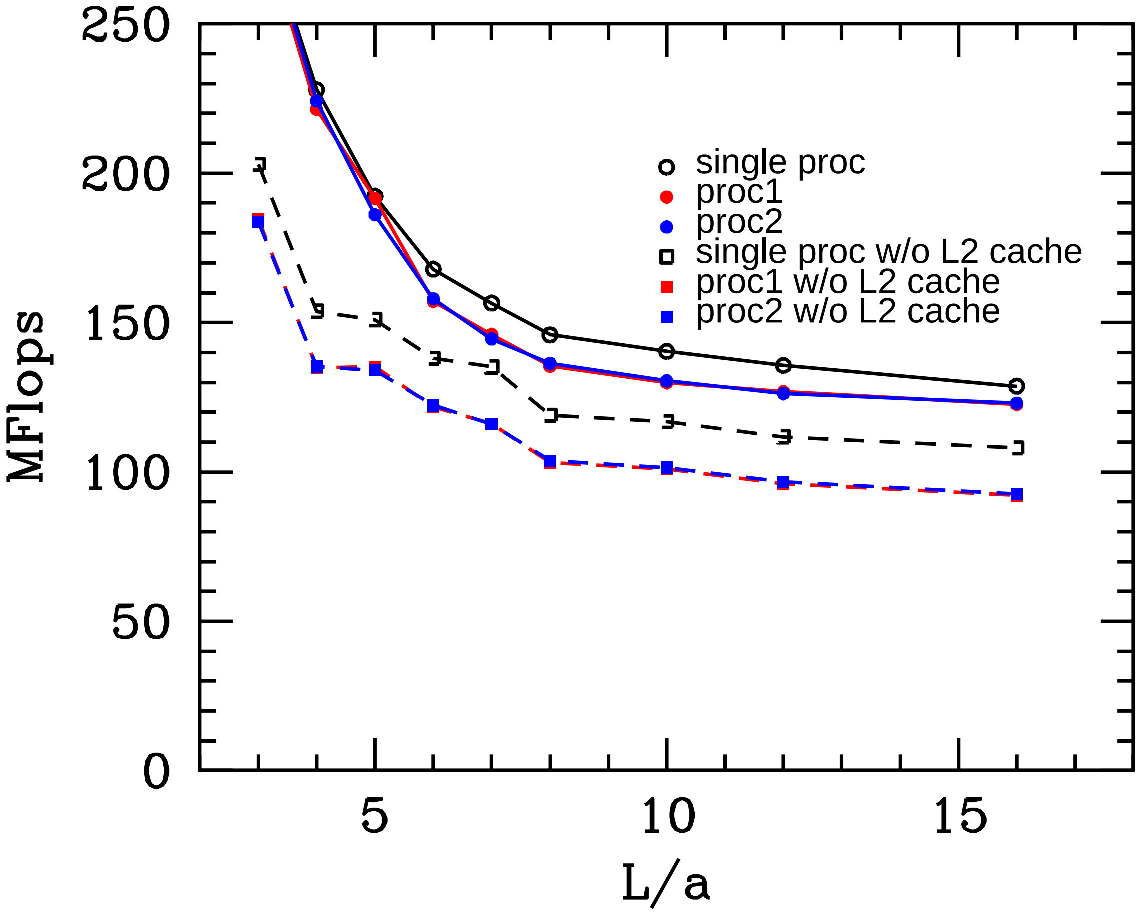

Figure 12: Sustained performance for the Wilson-Dirac operator at various lattice sizes L^4 on single and dual processors with L2 cache enabled (solid curves) and disabled (dashed curves).