Available online at www.sciencedirect.com

SCIENCE DIRECT®

Nuclear Instruments and Methods in Physics Research A 559 (2006) 90–94

NUCLEAR INSTRUMENTS & METHODS IN PHYSICS RESEARCH Section A

www.elsevier.com/locate/nima

The apeNEXT project[☆]

F. Belletti[a], F. Bodin[b], Ph. Boucaud[c], N. Cabibbo[d], A. Lonardo[d], S. de Luca[d], M. Lukyanov[e], J. Micheli[c], L. Morin[e], O. Pene[c], D. Pleiter[f], F. Rapuano[g], D. Rossetti[d], S.F. Schifano[a], H. Simma[e,*], R. Tripiccione[a], P. Vicini[d], The apeNEXT Collaboration

[a]*Dipartimento di Fisica, Università di Ferrara and INFN, Sezione di Ferrara, Italy*
[b]*IRISA/INRIA, Campus Université de Beaulieu, Rennes, France*
[c]*LPT, University of Paris-Sud, Orsay, France*
[d]*Dipartimento di Fisica, Università di Roma "La Sapienza" and INFN, Sezione di Roma, Italy*
[e]*DESY Zeuthen, Germany*
[f]*NIC/DESY Zeuthen, Germany*
[g]*Dipartimento di Fisica, Università di Milano-Bicocca and INFN, Sezione di Milano, Italy*

Available online 20 December 2005

Abstract

Numerical simulations in theoretical high-energy physics (Lattice QCD) require huge computing resources. Several generations of massively parallel computers optimised for these applications have been developed within the APE (array processor experiment) project. Large prototype systems of the latest generation, apeNEXT, are currently being assembled and tested. This contribution explains how the apeNEXT architecture is optimised for Lattice QCD, provides an overview of the hardware and software of apeNEXT, and describes its new features, like the SPMD programming model and the C compiler.
© 2005 Elsevier B.V. All rights reserved.

PACS: 12.38.Gc; 07.05.Bx; 89.20.Ff

Keywords: Computer architecture; Lattice QCD; Massively parallel computing

1. Introduction

apeNEXT is the latest parallel computer developed by the APE Collaboration. The architecture is optimised for numerical simulations of Quantum Chromo Dynamics on the lattice (LQCD). While the previous APE generation, APEmille, has been the workhorse for many LQCD simulations in Europe since 2001, the new machine will contribute to provide the compute power for LQCD during the next years.

Main design goals for apeNEXT are a good *scalability* up to machines with tens of Tflops compute power and an architecture which is *optimised* to deliver around 50% of peak performance for LQCD applications. The project is carried out in the framework of a collaboration between research institutes in Italy (INFN), Germany (DESY) and France (Université Paris-Sud).

Implementation challenges for this project include the integration of all processor and network functionalities in a *single* ASIC chip and the support for the *standard* programming language C.

2. LQCD requirements

Computations in LQCD are based on the Feynman path integral method. The fields are restricted to live on a discretised space-time lattice in order to render the problem mathematically well-defined and suitable for numerical computations on a computer.

The gluons correspond to gauge fields represented on the lattice by SU(3) matrices, $U(x,t,\mu)$, associated to each of

[☆]Talk given by H. Simma at ACAT05.
[*]Corresponding author.
E-mail address: hubert.simma@desy.de (H. Simma).

the four forward links μ between adjacent lattice sites. The fermionic quark fields correspond to 12-component spin-colour vectors, $\psi(x,t)$, associated to each lattice site. On a lattice of e.g. $32^3 \times 64$ lattice sites we need as many as about 2×10^8 complex variables to store the gluon field and one quark flavour. The integration over such a high number of dimensions is carried out by a Monte-Carlo method which generates gauge field configurations according to the distribution

$$P(U) \sim \mathrm{e}^{-S_g(U)} \cdot \int D[\psi]\mathrm{e}^{-\bar{\psi}M(U)\psi}$$

where S_g is the action of the gauge field and $M(U)$ is the Wilson–Dirac operator. The algorithms typically used to compute the integral $D[\psi]$ over the fermion field are based on iterative methods. The computational cost is dominated by the multiplication $M(U)\psi$ of some fermion field with the Wilson–Dirac operator. This is a sparse matrix, but its dimension grows with the number of lattice sites and for each site the computation of $M(U)\psi$ requires 1344 floating-point (FP) operations. The key requirement for the design of a dedicated computer for LQCD applications is therefore the efficient computation of $M(U)\psi$ together with basic linear algebra on the ψ vectors. Obviously, this needs a powerful FP unit, preferably in 64-bit precision to avoid accumulation of rounding errors for global quantities on large lattices. To efficiently feed the arithmetic unit, the FP throughput must be balanced with respect to the bandwidth for local memory accesses and remote communications between the processors (for more details see [1,2]). Further optimisations of the architecture must also include other hardware parameters like latencies and storage space at different levels of the memory hierarchy and of the communication network.

Parallelisation of LQCD codes on a computer can be achieved in a straightforward way by dividing the lattice into equal sublattices and distributing the corresponding variables over a processor grid. In this way, all nodes perform the same operations for most of the time and a simple Single Instruction Multiple Data (SIMD) architecture is sufficient.

Due to the huge computing resources needed for realistic large-scale LQCD simulations, production machines must be scalable up to systems with thousands of processors. These large systems must reliably operate over long time periods, because the execution of just a single LQCD program typically runs for several days.

3. apeNEXT enhancements

While improving on performance as much as possible through the use of more advanced processor architecture and technology, the design of apeNEXT has tried to maintain an architecture, as perceived at the user level, close to its previous generation, APEmille. However, two important features have been added in apeNEXT [3].

First, apeNEXT is a SPMD (as opposed to SIMD) system. Each processing node is a fully independent processor with a full-fledged flow-control unit and, of course, a number-crunching unit. Each node executes its own copy of the program at its own pace. Nodes need to be synchronised only when a data-exchange operation is performed. Internode communications are initiated by a single instruction on the sending node that must be matched by a corresponding instruction on the destination node. The latency associated to these "messages" is short, of the order of 2–3 times the latency of an access to local memory. Therefore, the actual data transfer rate between nodes is bandwidth-limited (as opposed to latency-limited) even for short packets. This is relevant in LQCD applications, where the natural size of data packets transferred to remote processors is not larger than about 200 bytes.

A second important architectural enhancement is the possibility of routing all memory read accesses (to local or remote nodes) through a receiving queue, which can be accessed by the processor with almost zero latency. This mechanism can be used for data pre-fetch in critical kernel loops and allows communications to be executed concurrently with other processor operations.

4. Node architecture

An apeNEXT computing node is based on one processor chip together with its local memory bank (256 MByte of Double Data Rate Dynamic RAM) which are housed on a small daughter board, shown in Fig. 1. One processor delivers up to 1.6 Gflops of processing power in double precision with a power consumption of approximately 7 W, that is about 10 times less than current generation high-end PCs.

The apeNEXT processor is a 64-bit architecture controlled by a very long instruction word (VLIW). All processor functionalities, including the memory and communication interfaces, are integrated in a single custom chip designed to run at a clock frequency of 200 MHz. Some further technical details are explained in the following.

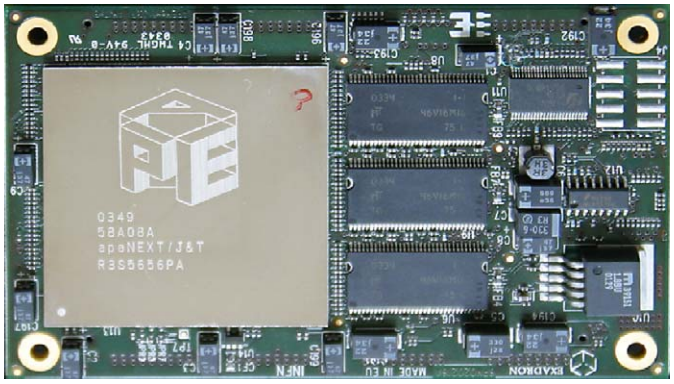

Fig. 1. apeNEXT daughter board with 1.6 Gflops (original size is 65×105 mm).

The arithmetic block performs FP and integer computations. FP operands are either complex values or pairs of real values, represented in 64-bit IEEE double-precision format. At each clock cycle a basic FP operation $a \times b + c$ can be started to provide a maximal throughput of eight FP operations per clock cycle. The arithmetic block also performs integer or bitwise arithmetics on pairs of 64-bit integers and provides support for mathematical functions, like square-roots or exponential of FP numbers.

The large register file with 256 registers, each holding a pair of 64-bit words, is directly connected to the memory and prefetch queues (without any intermediate cache levels).

An address-generation unit allows to compute addresses for memory accesses (and other integer arithmetics) independently and concurrently with the main arithmetic block.

The memory controller interfaces the processor to the external standard DDR-SDRAM which stores both data and instructions. To reduce memory conflicts between data access and program fetch, the VLIW instructions are compressed and pass through an instruction buffer, which allows to keep a block of instructions (e.g. a loop body) in the processor. Instruction de-compression is performed on the fly.

A set of FIFOs implements the queue mechanism, which automatically ensures that data words are delivered to the processor in the same order in which they were requested from local or remote memory (even if accesses have completed in a different order, because a remote access takes longer than a local one).

The network interface controls seven full-duplex LVDS links. Each link transfers 1 byte per clock cycle, i.e. the gross bandwidth is 200 MB/s per link and direction. Due to protocol overhead the effective network bandwidth is $\leqslant 180$ MB/s. The network latency is $O(100\,\text{ns})$, at least one order of magnitude smaller than for today's commercial high performance network technologies. Once a communication request is queued, it is executed under complete hardware control and independently of the rest of the processor.

One of the seven communication links is used for data IO. An additional serial interface, based on the I^2C standard, serves for system initialisation, debugging and exception handling.

A few interrupt and control signals, e.g. to propagate exceptions and to perform synchronisation, are handled by a global tree network with less than 400 ns roundtrip time on large machines.

5. System architecture

The compactness and the low power consumption of the processing nodes are key ingredients to build very compact systems with thousands of nodes. Large apeNEXT systems may have up to 4000 nodes and a peak performance of 6.4 Tflops.

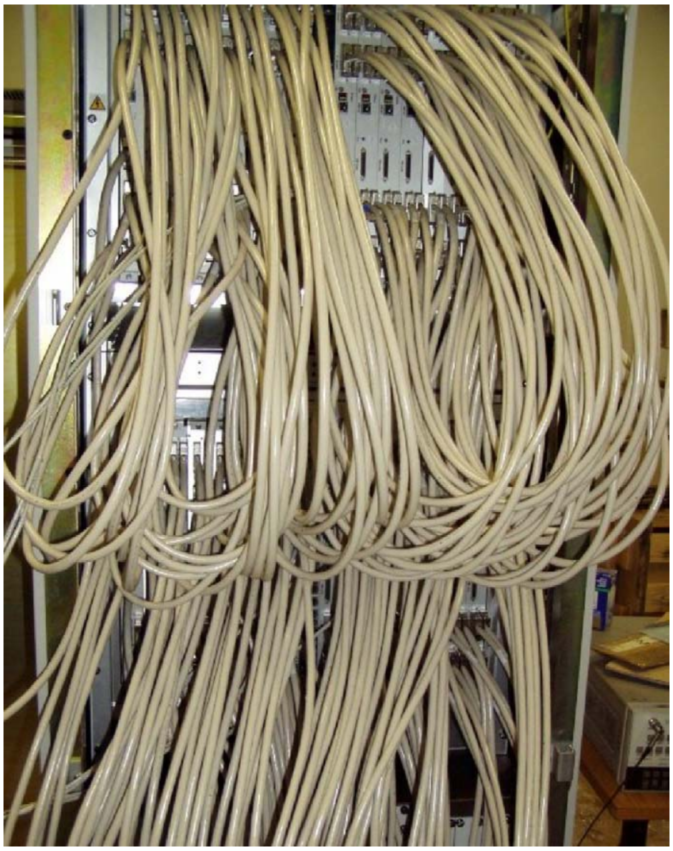
Fig. 2. apeNEXT rack with 512 nodes.

The 6 links of each node are used for fast communication between the processors which are arranged as a three-dimensional torus. The 7th communication link of certain nodes is connected to the host system, which is build up of a cluster of Linux PCs. In addition, all nodes are connected to the host via the slow serial interface and to a tree network for the global signals. All control and communication devices can be configured by software to allow flexible partitioning of the machine into independent sub-systems.

The entire system, including host PCs, power supply units, and air-forced cooling, is hosted in a rack which was custom developed to allow a high-density, robust and reliable system. Sub-systems of 16 apeNEXT nodes ($4 \times 2 \times 2$) are assembled onto a mother board. A set of 16 mother boards is housed within one system crate. All communication links between these nodes run on the crate backplane. Larger systems are assembled connecting together several crates using external cables. An apeNEXT rack, shown in Fig. 2, integrates two fully interconnected crates, i.e. 512 computing nodes, with a peak performance of 0.8 Tflops.

6. Software

apeNEXT machines are hosted and controlled by a cluster of Linux PCs. The cluster contains the slave-PCs directly connected to the apeNEXT core and one (or more) master-PCs where users log-in to run programs. The operating system (OS) allows to control and partition the machine, and supports execution and monitoring of a single-user program on each partition. The OS is distributed over the host PCs and cooperates with system routines running on the apeNEXT nodes to manage the loading of the user program and the handling of IO requests.

The IO and control links of the apeNEXT system are physically connected to Host Interface Boards which are plugged into the host PCs. Typically, the apeNEXT machines have one fast IO channel per 64 nodes, but other configurations, depending on the requirements for partitioning and IO-bandwidth, are possible.

Application programs for apeNEXT are written in the high-level programming languages TAO or C. Since TAO was used by all previous APE machines, a large set of LQCD programs has been developed over the years in TAO and continued TAO support is required for compatibility.

apeNEXT is the first APE machine which can be programmed in C. The compiler is based on the freely available lcc compiler and supports most of the ISO C99 requirements with a few language extensions to exploit specific hardware features and to control parallelism.

For both languages, access to remote data is memory mapped, i.e. the access is performed by specifying the address of the data structure plus a pre-defined constant. No call to library functions is necessary to perform a network data transfer.

The assembly code generated by either compiler is passed to a powerful assembly optimiser which performs a number of important architecture dependent and independent optimisations, like merging multiply-adds into normal operations, removing dead code, removing register copies through register renaming, merging of address generation operations, etc.

The optimised assembly is then translated into corresponding instruction patterns in the VLIW micro-code. In this final compilation step, the fine-grained scheduling of the micro-instructions is performed and optimised, and finally registers and most other hardware resources are allocated.

During micro-code scheduling a number of optimisations can be done at compile-time rather than through complex and power-hungry hardware at execution-time. Since only very few properties of the code execution are handled dynamically at execution-time, static analysis of the micro-code provides a rather accurate estimate of the performance of a given application code and allows to easily identify and understand possible bottlenecks.

7. Benchmarks

The overall performance of LGT applications is dominated by the performance of just a few computational kernels, like dot products and linear combinations of vectors or their multiplication with the Wilson–Dirac operator. Some benchmark results are shown in Table 1.

Table 1
Results for selected benchmarks

Operation	Mflops/s	%Peak
$M[U]\psi$	894	54
Vector norm	592	37
Scalar product	656	41
Linear combinations	464	29

These numbers are obtained after a number of obvious optimisations, like loop unrolling, performing memory access in long bursts, forcing the code of the loop body to be kept in the instruction buffer, or maximising the pipeline filling by performing partial sums. Most of our benchmark programs are written in assembly, but for some of them similar performance has already been reached from high-level language programs.

The benchmark for the Wilson–Dirac operator uses a layout with only $2^3 \times 16$ lattice sites per node. This implies a very high number of remote data accesses and corresponds to the worst case for physical applications. The remarkable efficiency was achievable only by extensive use of the queue mechanism to pre-fetch all data as early as possible and by exploiting concurrent communications along links in different directions.

The performance of the linear algebra benchmarks is mainly limited by the bandwidth of the memory access and not by the FP throughput. The measured performance is close to the maximal theoretical performance that can be achieved when taking into account memory constraints.

8. Conclusions

With apeNEXT a new generation of a special purpose machines is becoming available for LQCD applications. Large prototypes with 512 nodes have been assembled and tested successfully. Benchmarks with LQCD codes achieve excellent efficiencies with up to about 50% of peak performance for the most relevant kernels.

The architecture is scalable up to systems with multi-Tflops performance. Large installations are planned and funded e.g. at INFN in Rome (6656 nodes), DESY Zeuthen (2048 nodes), and University of Bielefeld (3072 nodes) for 2005/2006.

We expect apeNEXT to become a workhorse for LQCD production in Europe similar as it is APEmille today. The availability of the C programming language might also help to port other applications on apeNEXT. It is also remarkable how many architectural features found in dedicated LQCD machines, like APE and QCDOC [4], have been adopted by the latest massively parallel machine developed by IBM [5].

Acknowledgements

We like to warmly thank all people who have contributed to the apeNEXT project in the past, in particular W. Errico, H. Kaldass, N. Paschedag, R. De Pietri, F. Di Renzo, and L. Sartori.

References

[1] apeNEXT Collaboration F. Belletti, et al., Computing in Science and Engineering, IEEE Computer Society, to be published.
[2] R. Tripiccione, Comput. Phys. Commun. 169 (2005) 442.
[3] APE Collaboration, Comput. Phys. Commun. 147 (2002) 402; APE Collaboration, Nucl. Phys. Proc. Suppl. 140 (2005) 176.
[4] QCDOC Collaboration, Nucl. Phys. Proc. Suppl. 94 (2001) 825; QCDOC Collaboration, Nucl. Phys. Proc. Suppl. 129 (2003) 838.
[5] See special issue on BlueGene/L of the IBM J. Res. Dev. 49(2/3) 2005.