apeNEXT: a Multi-TFlops Computer for Elementary Particle Physics

apeNEXT-Collaboration: F. Bodin[a], Ph. Boucaud[b], N. Cabibbo[c], F. Di Carlo[c], R. De Pietri[d], F. Di Renzo[d], H. Kaldass[e], A. Lonardo[c], M. Lukyanov[e], S. de Luca[c], J. Micheli[b], V. Morenas[f], O. Pene[b], D. Pleiter[g], N. Paschedag[e], F. Rapuano[c], L. Sartori[h], F. Schifano[h], H. Simma[e], R. Tripiccione[h], P. Vicini[c]

[a]IRISA/INRIA, Campus Université de Beaulieu, Rennes, France

[b]LPT, University of Paris Sud, Orsay, France

[c]INFN, Sezione di Roma, Italy

[d]Physics Department, University of Parma and INFN, Gruppo Collegato di Parma, Italy

[e]DESY Zeuthen, Germany

[f]LPC, Université Blaise Pascal and IN2P3, Clermont, France

[g]NIC/DESY Zeuthen, Germany

[h]Physics Department, University of Ferrara, Italy

We present the apeNEXT project which is currently developing a massively parallel computer with a multi-TFlops performance. Like previous APE machines, the new supercomputer is completely custom designed and is specifically optimized for simulating the theory of strong interactions, quantum chromodynamics (QCD). We assess the performance for key application kernels and make a comparison with other machines used for these kind of simulations. Finally, we give an outlook on future developments.

1. INTRODUCTION

For many problems in modern particle physics lattice gauge field theory offers the only known way to compute various quantities from first principles using numerical simulations. Much progress has been made during recent years, e.g., in calculating the light hadron spectrum, the light quark masses, the running coupling constant α_s or observables in heavy quark physics. Further-on, lattice simulations allow the study of non-perturbative phenomena like chiral symmetry breaking, confinement, or studies of phase transitions in the early universe.[1]

However, progress in this field is severely limited by the tremendous amount of required computing power. In order to make the necessary resources available, various research groups engage in the development of massively parallel computers which are specifically optimized for these kind of applications. One of these projects is APE ("Array Processor experiment") which is currently developing its fourth generation of machines, apeNEXT.[2] This work is carried out within the framework of an European collaboration by INFN (Italy), DESY (Germany) and the University of Paris Sud (France).

The required computing resources critically depend on the physical parameters and the formulation of the theory on the lattice. It remains, for instance, challenging to perform simulations at light quark masses.[3] In now-a-days simulations of full QCD, i.e. calculations which take effects from sea quarks into account, the lightest quarks are usually still significantly heavier than the physical mass of the "up" and "down" quarks. Recently, several research groups started to use chirally improved formu-

[1]See the proceedings of the annual Symposium on Lattice Field Theory [1] for an overview.
[2]See [2] for a review of the APE project.
[3]Estimates for the CPU requirements as a function of the relevant physical parameters are given in [3].

lations of fermions on the lattice. While preserving important properties of the continuum theory, these new formulations, which fulfill the so-called Ginsparg-Wilson relation, lead to a significant raise of the required computing resources. In this paper we will concentrate on large scale applications, but we want to stress that there are also many small and medium scale applications from this field of research.

In simulations of QCD on the lattice, most of the resources are spent in frequent inversions of a huge, but sparse matrix. In commonly used formulations this so-called fermion matrix M has the following structure:

$$M_{x,y} = 1 - \kappa D = 1 - \kappa \sum_{\mu=1}^{4} \Big[(1-\gamma_\mu)\, U_{x,\mu}\, \delta_{x+\hat{\mu},y} + (1+\gamma_\mu)\, U^\dagger_{x-\hat{\mu},\mu}\, \delta_{x-\hat{\mu},y} \Big] \,. \qquad (1)$$

Here x and y label the sites of the 4-dimensional lattice. κ is a real parameter which is directly related to the quark mass. The lattice gauge fields $U_{x,\mu}$ which carry two colour indices are represented by 3×3 complex matrices. The Dirac matrices γ_μ carry two spin indices. Therefore, M is a $(3 \cdot 4 \cdot V)^2$ complex matrix, where V is the lattice volume. Since iterative algorithms are used to calculate the inverse, matrix times vector multiplications are the performance dominating operations.

When developing computers optimized for simulating lattice QCD several features which are basically independent of the specific formulation of the theory can be taken into account. For instance, multiplication by M involves mostly arithmetics with complex numbers and allows for a relatively high data re-use. Appropriate choice of memory hierarchies or register file size therefore have a significant impact on the performance. The fermion matrix is homogeneous in coordinate space and connects – at least in the most often used formulations, e.g. Eq. (1) – only nearest-neighbours. A SIMD programming model is sufficient to parallelize the problem by distributing the lattice sites on a 1 to 4-dimensional mesh of processors. Communication then has to be performed only between nearest-neighbour nodes. These features allow for a significant simplification of the global architecture and particularly the network.

2. apeNEXT HARDWARE ARCHITECTURE

apeNEXT is designed as a massively parallel computer. All processor functionalities, including the network devices, are integrated into one single custom chip running at a clock frequency of 200 MHz. The processor is a 64-bit architecture. However most operations act on $(64+64)$-bit words, which are, e.g., needed to encode an IEEE double precision complex number. The arithmetic unit can at each clock cycle perform the APE normal operation $a \times b + c$, where a, b, and c are complex numbers. The peak floating point performance of each node is therefore 1.6 GFlops.

Each processor is a fully independent processor with access to its private memory bank of 256-1024 MBytes based on standard DDR-SDRAM. In each clock cycle the ECC protected memory bus can sustain one $(64+64)$-bit word with a latency of about 15 clock cycles. The memory controller receives its input from an address generation unit that allows to perform integer operations independently of the arithmetic unit. The memory stores both program and data. A consequence of this organization is that conflicts between data and instruction load-operations are likely to occur, since apeNEXT is a microcoded architecture using 117-bit very long instructions words (VLIW). Two strategies have been employed to avoid these conflicts. First, the hardware supports compression of the microcode. The compression rate depends on the level of optimization, typically values are in the range of 40-70%. Instruction de-compression is performed on-the-fly by dedicated hardware. Second, an instruction buffer allows, under complete software control, pre-fetching of (compressed) instructions and storing performance critical kernels for repeated execution.

Internode communication is done via a high-performance network which has the topology of a three-dimensional torus. The network is able to transfer one byte per clock cycle with an extremely small latency of about 20 clock cycles (i.e. 100 ns). Communication is therefore clearly bandwidth-limited. For the user this has the advantage that almost maximum throughput is already reached when communicating data using rather small bursts, e.g. 12 words or about 200 bytes of data. Transmission of each 128-bit word is protected against single bit errors by a 16-bit cyclic redundancy checksum (CRC). Each node is bi-directionally connected to its six nearest neighbours. However, the hardware supports routing along up to three orthogonal links with only little extra latency. Communication in

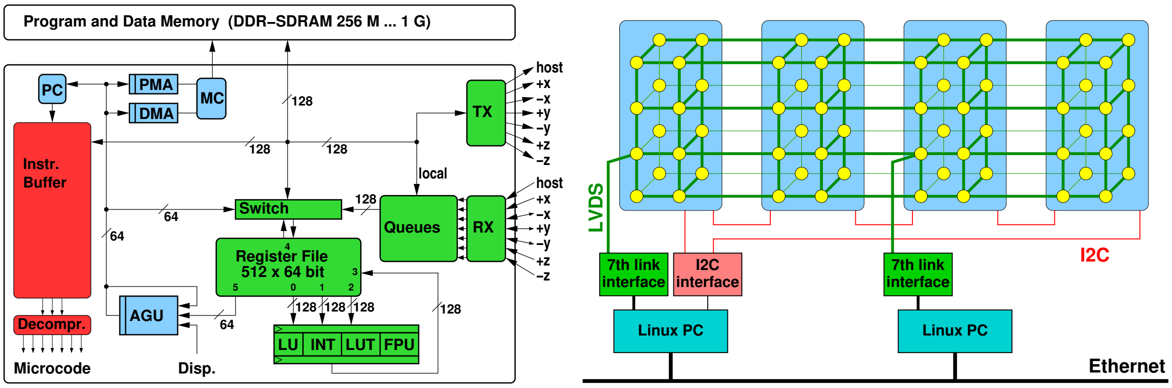

Figure 1. Schematic view of the apeNEXT processor (left) and a sample global configuration with four boards and two front-end PCs (right).

orthogonal directions can be executed independently of each other. apeNEXT systems will contain up to several thousand processors. In order to achieve high scalability it is necessary that network and arithmetic operations overlap. Furthermore, memory and network latencies, although they are small, become nevertheless relevant when aiming for outstanding sustained performance. The apeNEXT processor allows to pre-fetch local or remote data into a receive queue. Later the data can be moved from the queue to the register file with zero latency. This allows to hide memory latencies as well as network bandwidth limitations. The latter is possible because all queued network requests can be executed independently of the rest of the processor. To allow data being re-used a very large register file of 256 $(64 + 64)$-bit registers is provided. Furthermore, data can be written from the register file into the receive queues for later use. Which data is kept inside the processor is therefore completely under user control, who does not have to care about difficult to control cache effects.

All apeNEXT nodes are designed to run asynchronously, which means that apeNEXT follows the single program multiple data (SPMD) programming model. The nodes will implicitly be synchronized by communications. A dedicated logics tree, which allows fast evaluation of global `AND` or `OR` conditions, can also be used for implementing barriers. It can therefore be used for explicit synchronization.

A set of 16 apeNEXT nodes will be assembled onto just one processing board. A set of 16 boards can be attached to one backplane which provides all links for communications between nodes of different boards. Larger systems are assembled by connecting together several crates using external cables. Up to 32 boards can be hosted by one rack. Such a system of 512 nodes provides a peak performance of 0.8 TFlops. It has a footprint of about 1 m^2 only and its estimated power consumption is < 10 kW. Due to this very moderate power dissipation air cooling is still possible.

Accessing an apeNEXT system will be possible via a front-end PC with a custom designed host-interface PCI-board. A simple, I2C-based network is foreseen for bootstrapping the machine and executing simple operating system requests, like program exit or error handling. I/O operations will be handled via high-speed data links connecting one or more boards of a apeNEXT system with one or more host-interface boards. This design concept allows to adjust the overall I/O bandwidth to the needs of the users. The external links are based on the same technology as the internal network and have a gross bandwidth of 200 MBytes/sec. Initial tests with a prototype board indicate that it will be possible to sustain this bandwidth across the 100 MHz, 64-bit PCI-bus.

3. apeNEXT SOFTWARE

On apeNEXT two compilers will be available which allow to write code in the high level languages TAO and C. TAO is a FORTRAN-like programming language based on a dynamical grammar. This grammar allows to define objects and to overload functions in a rather simple way. For the programmer this has the advantage of being able to write its code in a more natural way. For the compiler design using objects makes it easier to do burst memory access, which is crucial for performance reasons. TAO was used to write programs for previous generations of APE machines, which now can be ported to apeNEXT with little effort. The C-compiler is based on the freely available lcc compiler [4]. It supports most of the ANSI 89 standard with a few language extensions. Except for standard data types, variables of type complex and vector (a set of two doubles) are available. Further language extensions are related to the support of parallelization, e.g. the statements `any()`, `all()` and `none()` allow converting local conditions into global conditions. In TAO and C elements of an array stored at a remote memory location can be accessed by specifying a remote offset, which encodes the direction into which local data has to be send and from where data will be received. Whenever the memory controller is provided with a memory address with certain remote-offset bits, it will not load the addressed data into the local queue but write it to a send buffer instead. There is no call to a communication library, e.g. MPI, required to initiate a network transfer. Since the remote-offset bits are within the range of integers, they can be added to index variables used for addressing arrays.

Both compilers produce high level assembly instructions. The high level assembly is also suitable for user programming, since many odd details, like insertion of instruction load operations, jumping to the static instruction cache or executing subroutine calls, are handled by a program which converts high to low level assembly. The low level assembly, which is already very machine specific, can be fed into the optimization toolkit sofan. A number of optimization procedures are postponed until this step since they are difficult to implement on the compiler level in a one-pass procedure. The following optimization steps are foreseen: merging normal operations, removing dead code, eliminating register move operations, merging of address generating operations.

The (optimized) low level assembly is used as input for the microcode generator. During this final compilation step the assembly instructions are translated into (possibly several) microcode instructions. To optimize performance the instructions are scheduled as early as possible taking all dependencies and temporization constraints into account. Finally, the microcode generator allocates the registers, compresses the microcode instructions and patches the jump addresses.

The final piece of software, needed to actually run a program on an apeNEXT machine, is the operating system. The apeNEXT architecture will require a distributed operating system. For handling fast I/O requests system routines running on the custom hardware itself will take care of moving data to or from the nodes which are connected to the external host-interface boards. The latter will be controlled by slave daemons running on the front-end PCs. This daemons poll on the machine's registers accessible via the I2C links to check the machine state, e.g. to detect program halt or exceptions. They will also check for data on the incoming high-speed link to detect I/O requests, which will be handled without halting the machine (unlike in older APE systems). A master process will control the slave daemons running on the one or more front-end PCs connected to a particular apeNEXT system or partition.

4. BENCHMARKS

Functional simulation of the machine design, which is written in VHDL, allows realistic, cycle accurate performance measurements. In the following we report on results for benchmarks which allow to assess the overall performance of lattice QCD simulations on apeNEXT. For all benchmarks the time needed for instruction loading has not been taken into account, since it can reasonably be assumed that performance critical routines are executed many times. These routines will therefore be loaded only once and will be kept in the instruction buffer.

Iterative matrix inversion algorithms, like conjugate gradient or BiCGstab, involve a number of linear algebra operations during each iteration. This includes the calculation of a scalar product of two (complex) vectors x and y, $z = \sum_{i=0}^{N-1} x_i^* y_i$. Since this operation requires for each iteration i loading of 2 words, there is an upper limit for the sustained performance of 50%: the memory read operations require two clock cycles per iteration, while the arithmetic unit could sustain one iteration per clock cycle. This operation is therefore memory bandwidth limited. To get close to the performance limit a few optimization tricks have to be employed. For instance, to hide memory latency burst memory access should be used for loading the vectors x and y. Reducing loop overhead requires loop unrolling. Finally, to hide the latency of the arithmetic unit (in this case 10 clock cycles) the summation should be split into partial sums. With reasonable choices for burst length and unrolling factor a sustained performance of 41% has been measured. The performance limit is not reached due to loop and pipeline overhead (here: about 40 clock cycles). Higher optimization could be achieved by further loop unrolling at the cost of an increased number of required instructions.

The calculation of the squared norm of a complex vector on the other hand requires to load only 1 word per iteration and to perform 4 floating point operations. The apeNEXT processor will nevertheless have to execute a full normal operation, which is equivalent to 8 floating point operations. Therefore, the sustained performance is again limited to 50%. The actual measured performance for this operation is 37%. Another operation typical for lattice QCD applications is the calculation of linear combinations of two complex vectors, $z_i = \alpha\, x_i + y_i$. Keeping the (complex) scalar variable α in the register file, this operation requires for each iteration i loading of 2 words and storing 1 word. This operation is therefore again limited by the memory bandwidth, which restricts the sustained performance to a maximum value of 33%. We have measured a sustained performance of 29%.

Both, calculation of a scalar product of two vectors as well as the squared vector norm require a final global summation of the results of the partial local sums. The time needed for this operation depends on the total number of nodes. To calculate the global sum we shift the local result by one step into direction k ($k = x, y, z$) and add it to the result of that processor. This procedure has to be repeated until the information from all nodes has been propagated to all other nodes. This requires $P_x + P_y + P_z - 3$ iterations, where P_k is the number of nodes in direction k. For achieving bit-identical results on all nodes the result from one node has to be broadcasted to all other nodes, which basically amounts to repeating the summation procedure. Taking network latency and bandwidth into account, each iteration would require about $2 \cdot 35$ clock cycles. In actual simulations with up to $4 \times 2 \times 2$ nodes we measured 71 cycles per iteration (neglecting electrical delays). For typical applications and partition sizes the time needed to calculate a global sum remains nevertheless relatively small.[4]

Let us finally consider the multiplication of a quark field by the hopping part of the fermion matrix, D, as defined in Eq. (1). For each lattice site this operation requires loading of 168 words, storing 12 words and executing 1320 floating point operations. This operation requires read access to remote elements of the quark field.[5] The performance therefore potentially depends on the ratio of the number of remote versus the number of local memory read operations. This ratio depends on the local volume. Assuming $V = 16^3 \times 32$ to be the smallest global lattice size used in simulations on apeNEXT, the smallest local lattice size is $v = 16 \times 2^3$, i.e. all sites are at the boundary and have to be communicated. For this worst case we measured a sustained performance of 55%. This remarkably high sustained performance was only achievable by making efficient use of the queue mechanism and the network. This eventually allows for almost complete overlap of memory load and store operations, communication and arithmetic operations, which means that this most important application scales very well on apeNEXT. To achieve this all data including index fields were prefetched as early as possible. To increase the network throughput consecutive network requests were initiated in orthogonal directions, because these can be executed independently.

5. OTHER ARCHITECTURES

In recent years also other attempts have been started to build machines optimized for simulations of lattice gauge theory, in particular QCD. One of the projects, QCDOC (QCD On a Chip), is also developing a fully custom designed machine [5]. The work is carried out by a collaboration of Columbia University, IBM Research, UKQCD, and the RIKEN-BNL Research Center. The QCDOC chip, similar to apeNEXT, combines all processor functionalities and the network capabilities on one ASIC. It is based on a PowerPC 440 core and its 64-bit floating point unit provides 1 GFlops peak performance. An interesting and performance relevant feature of the QCDOC processor are the 4 MBytes on-chip memory. For key applications in lattice gauge theory the QCDOC collaboration measured sustained performance numbers which are very similar to those of apeNEXT.

A number of groups is exploring the possibilities of cluster computers, typically using commodity PCs with IA32 processors. Significant progress has been achieved in recent years [6]. In particular for Intel Pentium 4 processors is was possible to achieve remarkable sustained performances for relevant benchmarks by making extensive use of the memory-to-cache prefetch capabilities as well as the SSE registers and instructions. However, for currently available network techniques large messages are

[4]For a lattice of size $V = 32^3 \times 64$ the local scalar product of two quark fields (i.e. $N = 12\,V$) requires $O(10^5)$ clock cycles on $8 \times 8 \times 8$ nodes, while $O(10^3)$ clock cycles are needed to calculate the global sum.

[5]In typical simulations the number of updates of the gauge fields is significantly smaller than the number of multiplications with the hopping term. Communication of the gauge fields can therefore be neglected here.

required to obtain high throughput and to reduce latencies. For a number of applications scaling of the performance with an increasing number of nodes therefore becomes a problem. Furthermore, power consumption and space requirements are significantly higher compared to machines like apeNEXT and QCDOC. On the other hand, clusters require less technical know-how, offer more opportunities for developing portable software, and typically allow easy technology upgrades. In case of small and medium sized installations these advantages are likely to prevail.

6. CONCLUSIONS AND OUTLOOK

We have discussed in detail the hardware design of apeNEXT and the software which will be available for this platform. The design has been finished and prototype versions for all hardware components except the processor itself exist and have been tested. This includes board, backplane and host-interface board. Prototype processors are planned to become available in autumn 2003. We expect the final procurement costs to be 0.5 €/MFlops peak performance.

A functional model of the machine design has been used for benchmarking the architecture. We presented a detailed discussion of the results and showed that the actual design goals of the project have been reached. Benchmark programs indicate that it will be possible to run key lattice gauge theory kernels at a sustained performance of $O(50\%)$ or more. While apeNEXT is designed as a special purpose machine optimized for simulating QCD, we expect the architecture to be suitable also for other applications.

During the next years groups doing research in lattice gauge theory in Europe aim for large dedicated installations of massively parallel computers. In Edinburgh (UK) a 10 TFlops QCDOC machine will be installed by 2004. In Italy it is planned to build-up a 10 TFlops apeNEXT installation. In Germany researchers from various groups joined the Lattice Forum (LATFOR) initiative, which proposed a broad research program for the next years [7]. LATFOR assessed the required computing resources and proposed to aim for a 15 and 10 TFlops installation of apeNEXT machines hosted by NIC at DESY (Zeuthen) and GSI (Darmstadt), respectively.

To make efficient use of these vast resources and to push towards development of future high performance computing capabilities an initiative on scientific parallel computing (SciParC) has been launched recently [8]. Aim of this project is to bring together different fields in science, which have in common the need of large computing power on massively parallel machines, for developing portable interfaces and software as well as for exploiting existing hardware architectures and designing future hardware solutions.

ACKNOWLEDGEMENTS

We would like to thank W. Errico (INFN) for his important work in the early phases of the project, and A. Agarwal (DESY/Zeuthen) and T. Giorgino (INFN) for their contributions.

REFERENCES

[1] R.G. Edwards, J.W. Negele, D.G. Richards, "Proceedings of the XXth International Symposium on Lattice Field Theory," Nucl. Phys. B (Proc. Suppl.) 119 (2003).
[2] R. Alfieri *et al.* [APE Collaboration], arXiv:hep-lat/0102011; R. Ammendola *et al.*, arXiv:hep-lat/0211031; F. Bodin *et al.*, arXiv:hep-lat/0306018.
[3] A. Ukawa [CP-PACS and JLQCD Collaborations], Nucl. Phys. Proc. Suppl. **106** (2002) 195; F. Farchioni *et al.* [qq+q Collaboration], arXiv:hep-lat/0209142.
[4] C.W. Fraser, D.R. Hanson, D. Hansen, 1995.
[5] P. A. Boyle *et al.*, Nucl. Phys. Proc. Suppl. **106** (2002) 177 [arXiv:hep-lat/0110124]; P. A. Boyle *et al.*, arXiv:hep-lat/0210034; P. A. Boyle *et al.*, arXiv:hep-lat/0306023.
[6] S. Gottlieb, Comput. Phys. Commun. **142** (2001) 43 [arXiv:hep-lat/0112026]; M. Lüscher, Nucl. Phys. Proc. Suppl. **106** (2002) 21 [arXiv:hep-lat/0110007]; Z. Fodor, S. D. Katz and G. Papp, Comput. Phys. Commun. **152** (2003) 121 [arXiv:hep-lat/0202030].
[7] R. Alkofer *et al.*, http://www-zeuthen.desy.de/latfor.
[8] http://www-zeuthen.desy.de/sciparc.