Parallelising Matrix Operations on Clusters for an Optimal Control-Based Quantum Compiler

T. Gradl[1], A. Spörl[2], T. Huckle[1], S.J. Glaser[2], and T. Schulte-Herbrüggen[2]

[1] Department of Mathematics and Computer Science
[2] Department of Chemistry, Technical University Munich, Lichtenbergstrasse 4, D-85747 Garching, Germany
huckle@in.tum.de and tosh@ch.tum.de
http://www5.in.tum.de/~huckle/

Abstract. Quantum control plays a key role in quantum technology, *e.g.* for steering quantum hardware systems, spectrometers or superconducting solid-state devices. In terms of computation, quantum systems provide a unique potential for coherent parallelisation that may exponentially speed up algorithms as in Shor's prime factorisation. Translating quantum software into a sequence of classical controls steering the quantum hardware, *viz.* the quantum compilation task, lends itself to be tackled by optimal control. It is computationally demanding since the classical resources needed grow exponentially with the size of the quantum system. Here we show concepts of parallelisation tailored to run on high-end computer clusters speeding up matrix multiplication, exponentials, and trace evaluations used in numerical quantum control. In systems of 10 spin qubits, the time gain is beyond a factor of 500 on a 128-CPU cluster as compared to standard techniques on a single CPU.

We are currently in the midst of a second quantum revolution. The first one gave us new rules that govern physical reality. The second one will take these rules and use them to develop new technologies.
Dowling and Milburn, 2003 [1]

Scope

For exploiting the power of quantum systems, one has to steer them by classical controls such as voltage gates, radio-frequency pulses, or laser beams. Here the aim is to provide computational infrastructure for doing so in an optimal way, because the shapes of these controls critically determine the performance of the quantum system in terms of overlap of its actual final states with the desired target states. Standard engineering methods solve related problems for systems of classical physics. For quantum control, however, the calculations of optimal shapes become more complicated (on conventional classical computers): the quantum states have to be represented by matrices, the dimensions of which grow exponentially with system size.

Here we present the adaptation of a number of matrix operation routines to high-end parallel computer clusters while using the symmetry of quantum spin

W.E. Nagel et al. (Eds.): Euro-Par 2006, LNCS 4128, pp. 751–762, 2006.

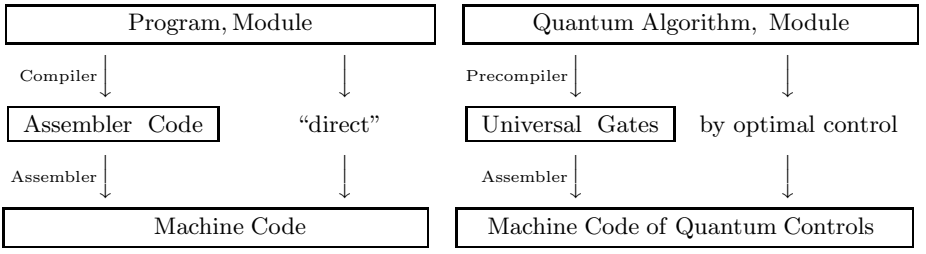


Fig. 1. Compilation in classical computation (left) and quantum computation (right). In the quantum scenario, the machine code has to be time-optimal or dissipation-protected, otherwise decoherence wipes out the coherent superpositions the quantum bits (qubits) build upon. By assembling universal quantum gates timeoptimality is in general not reached, while compilation by quantum control lends itself to minimise losses.

systems to minimise computational and communication effort as well as storage costs.

These methods of using classical computer clusters are tantamount to exploiting the power of present and future quantum resources.

1 Algorithms on Parallel Clusters for Quantum Control

1.1 Quantum Parallelism

Controlling quantum systems also offers a great potential for performing computational tasks or for simulating the behaviour of other quantum systems [2,3]. This is because the complexity of many problems [4] reduces upon going from classical to quantum hardware. It roots in Feynman's observation [2] that the resources required for simulating a quantum system on a classical computer increase exponentially with the system size. In turn, he concluded that using quantum hardware might therefore *exponentially decrease* the complexity of certain classical computation problems. Coherent superpositions of quantum states used as so-called 'qubits' can be viewed as a particularly powerful resource of quantum parallelism unparalleled by any classical system. Important applications are meanwhile known in quantum computation, quantum search and quantum simulation: most prominently, there is the exponential speed-up by Shor's quantum algorithm of prime factorisation [5,6], which relates to the general class of quantum algorithms [7,8] solving hidden subgroup problems in an efficient way [9].

1.2 Quantum Compilation as Control Problem

Moore's Law of increasing classical computing power concomitant to miniaturising microchips is often quoted to make a strong case for predicting that within the next decade computer hardware scales will reach sizes that inevitably have to include quantum effects. Among the generic tools needed for advances in quantum technology (see *e.g.* Ref. [1] for a survey), quantum control plays a major

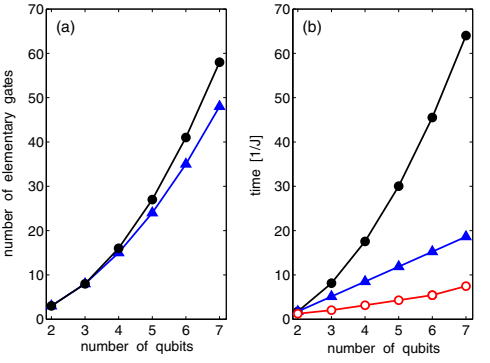
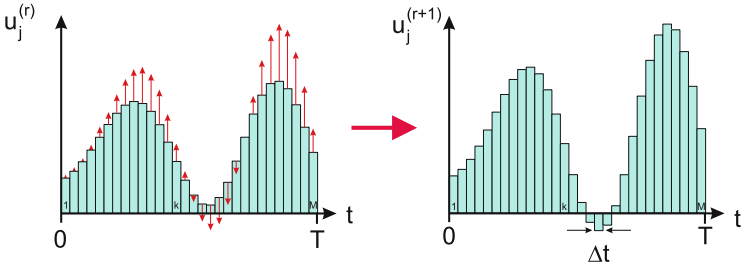


Fig. 2. (a) Gate complexity of the QFT in linear spin chains. Standard-gate decomposition (•) [13] and optimised scalable gate decomposition (▲) [14]. (b) Time complexity of the QFT in linear spin chains. Upper traces give analytical times associated with the decompositions of part (a): standard-gate decompositions (•) [13] and optimised scalable gate decompositions (▲) [14]. Lowest trace: speed-up by time-optimal control with shortest numerical realisations obtained (○) rounded to 0.01 J^{-1}. Details in Ref. [12].

role. As illustrated in Fig. 1, this may be exemplified by envisaging the process of compiling a quantum module into the machine language of a concrete quantum hardware device as an instance of quantum control. To this end, there are two different approaches: one may either use (i) universal elementary quantum gates [10] to synthesise a quantum computational module from prefabricated standard building blocks, the so-called universal quantum gates, or (ii) one may prefer to generate the quantum module directly from the experimentally available controls with the help of gradient-flow based numerical algorithms implementing tools of optimal control [11,12]. While decomposition into universal gates is inspired by discrete level permutations, direct compilation exploits the differential geometry of smooth manifolds for governing unitary quantum dynamics in an optimal way. Recently we have shown that one may thus obtain dramatic speed-ups, *e.g.* for the Quantum Fourier Transform (QFT) in spin systems (see Fig. 2 and [12]) or for realising multiply controlled gates in solid-state devices. Here the speed-up translates into a gain of some two orders of magnitude in approaching the error-correction threshold [15]. The approach is very general and holds for spin and pseudo-spin systems whose dynamics are Lie-algebraically closed.

1.3 Gradient Flow Algorithms for Quantum Control

Our algorithmic tools of optimal quantum control [11] for obtaining these results are based on gradient flows [16,17] tailored to the unitary group of Hamiltonian quantum evolution [18,19,20]. Let U_G denote the unitary representation of a quantum gate, *i.e.* the target matrix. On the other hand, define by $U(T) := e^{-it_M H_M} \dots e^{-it_k H_k} \dots e^{-it_1 H_1}$ the propagator brought about by a sequence of



1. set initial controls $u_j^{(0)}(t_k)$ for all times t_k with $k = 1, 2, \ldots M$ at random or by guess;
2. starting from $U_0 = \mathbb{1}$, calculate the forward-propagation for all $t_1, t_2, \ldots t_k$ (for simplicity $\Delta t := t_{k+1} - t_k$ uniform)

$$U^{(r)}(t_k) \;=\; e^{-i\Delta t H_k^{(r)}} e^{-i\Delta t H_{k-1}^{(r)}} \ldots e^{-i\Delta t H_1^{(r)}} \quad ;$$

3. likewise, starting with $T = t_M$ and $\lambda(T) = \text{const} \cdot U_G$, compute the back-propagation for all $t_M, t_{M-1}, \ldots t_k$;

$$\lambda^{(r)}(t_k) \;=\; e^{i\Delta t H_k^{(r)}} e^{i\Delta t H_{k+1}^{(r)}} \ldots e^{i\Delta t H_M^{(r)}} \lambda(T) \quad ;$$

4. calculate $\frac{\partial h(U(t_k))}{\partial u_j} = \text{Re tr}\{\lambda^\dagger(t_k)(-iH_j)U(t_k)\}$;
5. with $u_j^{(r+1)}(t_k) = u_j^{(r)}(t_k) + \varepsilon \frac{\partial h}{\partial u_j}\Big|_{t=t_k}$ update all the piece-wise constant Hamiltonians to $H_k^{(r+1)}$ and return to step 2.

Fig. 3. Top trace: updating the vector of control amplitudes u_j by the gradients (arrows) evaluated via the iterative scheme, the GRAPE-algorithm [11] in the box below. Gradients are calculated in step 4, while the controls are updated as in step 5.

evolutions of the quantum system under M piece-wise constant Hamiltonians H_k. Then the optimal control problem can be cast into the tasks:

$$\begin{aligned} &\text{maximise} \quad f(U(T)) = \text{Re tr}\{U_G^\dagger U(T)\} \\ &\text{subject to} \quad \dot{U}(t) = -iHU(t) \quad , \end{aligned} \tag{1}$$

where the Hamiltonian H comprises drift and control terms $H = H_{\text{drift}} + \sum_j u_j H_j$ with u_j as element of the (real) control-amplitude vector. As usual, the boundary condition may be included by a Lagrange parameter λ so that one may finally exploit the corner stone of control theory, Pontryagin's maximum principle, in a quantum setting (for details see, *e.g*, [21,11]) to require for the real-valued function h that $\frac{\partial h}{\partial u_j} = \text{Re tr}\{\lambda^\dagger(-iH_j U)\} \to 0$ at all times t_k. So the task can readily be solved by gradient flows iteratively improving the classical controls driving the quantum system into maximal overlap with the target.

1.4 Computational Tasks: Previous Performance and Goals

As sketched in Fig. 3, from a computational point of view, the GRAPE-algorithm makes heavy use of (1) matrix multiplication, (2) matrix exponentials, (3) trace

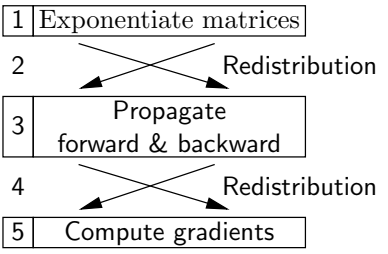


Fig. 4. Redistribution of matrices is needed between steps of the GRAPE algorithm

evaluation, and (4) step-size optimisation in conjugate gradients. The standard C++ code with CBLAS matrix-matrix multiplication [22,23,24] could deal with quantum systems up to 7 spins on an AMD Athlon Processor with 2.13 GHz and 1 MB RAM, taking months of CPU time. With the CPU time roughly growing by a factor of 8 per additional spin qubit, 10-spin systems are clearly out of reach unless one could speed up the calculations some 500 times.

In the present work we set out to reach this benchmark on a high performance cluster. The employed system consists of 128 AMD *Opteron 850* CPU (2.4 GHz), four on each node; the nodes are connected with an *Infiniband* network[1]. For parallel programming the MPI standard was used.

In order to exploit the power [25] of high-end computer clusters, here we address features of distributed matrix multiplication, concepts of broadcasting no more than the necessary information to the nodes of processors, reducing communication effort between different processors as well as exploiting symmetries of the matrix representations of the pertinent quantum mechanical system Hamiltonians for speeding up matrix exponentials. Some of the symmetries are not coincidental: fully controllable quantum systems are Lie-algebraically closed [19]. They are thus largely confined to finite-dimensional spin- or pseudo-spin systems, whose representations in terms of Kronecker or tensor products of Pauli matrices often entail persymmetric matrices (*vide infra*).

2 Parallel Matrix Multiplication

This section compares the implementation of two algorithms for multiplying a series of matrices ("propagation"), as needed in steps 2 and 3 of the GRAPE algorithm. The algorithms differ in run-time and, as will turn out to be most decisive, in memory demand. For comparing performance in terms of run-time, just considering the time required in the propagation step does not suffice; as we will see, it is also important to understand how it is embedded into the whole of the GRAPE algorithm. To this end, consider Fig. 4: the propagation (step 3 in this figure) is preceded by the computation of the exponential matrices $e^{\pm i\Delta t H_k^{(r)}}$ (step 1). The exponentials of all the matrices ($k = 1 \ldots M$) are distributed over

[1] `http://www.lrr.in.tum.de/Par/arch/infiniband/`

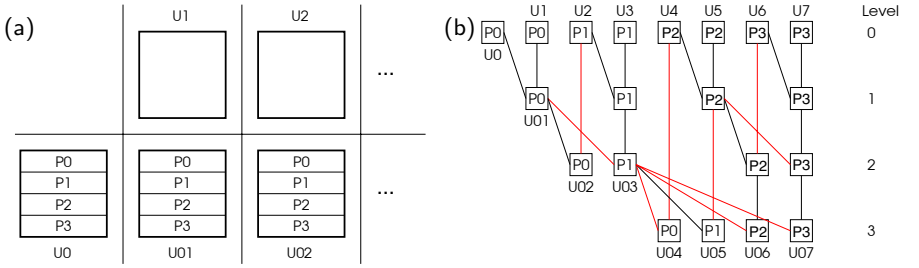


Fig. 5. (a) Slice-wise matrix multiplication provides a simple way of parallelisation. U_{0k} denotes the $(k+1)$-fold product $U_k U_{k-1} \cdots U_0$ according to step 2 in the GRAPE-algorithm (Fig. 3). The resulting complexity is $\mathcal{O}(M \cdot N^3/p)$. Communication between the processors P is needed solely for broadcasting the matrices U_k prior to propagation. (b) Scheme for tree-like propagation. In this example, propagation is carried out in three steps. Red lines indicate communication between processors P_0 through P_3.

all the processors. However, in step 3 it is not granted every processor exactly needs those matrices it computed in step 1. For this very reason an intermediate redistribution of matrices among processors is required not only in step 2 but also upon proceeding from step 3 to step 5 (gradient computation).

2.1 Slice-Wise Propagation

The matrix matrix multiplication AB can most easily be split into jobs distributed to different CPUs by taking say the rows a_ℓ of A separately as

$$AB = (a_1; a_2; \ldots a_N)B = (a_1 B; a_2 B; \ \ldots \ ; a_N B) \quad . \tag{2}$$

This scheme is readily extendible to k out of the M matrices in step 2 and 3 of the GRAPE-algorithm above (see Fig. 5(a)). However, each processor then refers to $k-1$ matrices, which means that they have to be broadcasted in step 2 of figure 4. Also, the workspace required by each processor is of the order of $\mathcal{O}(M \cdot N^2)$. The time complexity in this straightforward scheme can easily be evaluated, because the total number of operations is evenly distributed among the available processors. So the order of operations is $\mathcal{O}(M \cdot N^3/p)$, where $N := 2^n$ denotes the dimension of the matrix and p is the number of processors.

Moreover, here there are no stability concerns, as unitary matrices are known to allow for numerically stable algorithms [26,27]: the computation of the product of two unitary matrices is well-conditioned and this clearly extends to multiple products in general. However, Section 2.2 will reveal this is not necessarily the case in all other schemes.

For further acceleration some simplifying features can be used: step 4 of the GRAPE-algorithm, for instance, takes the trace Re $\mathrm{tr}\{\lambda^\dagger H U\}$, where λ and U are fully occupied, but H is a sparse matrix representation of the spin-control Hamiltonian, which acts by permutation and scalar multiplication on the rows

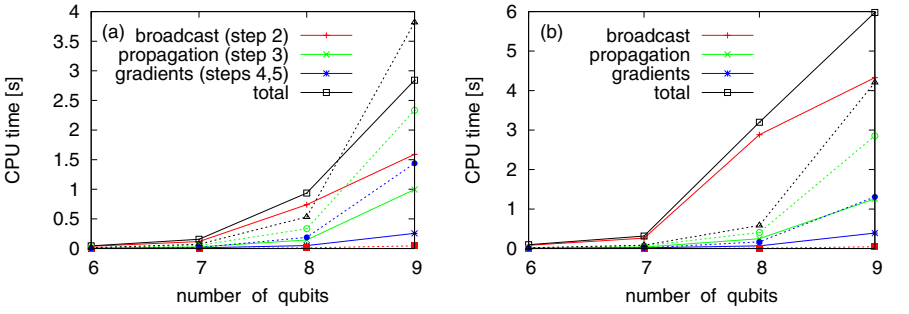


Fig. 6. Comparison of performance against system size with (a) 32 and (b) 64 parallel processors under slicewise (—) and tree-like (- -) propagation. Note the difference in broadcasting time (red lines). Steps 2-5 refer to the numberings as in Fig. 4.

of U. Thus, instead of two matrix-matrix multiplications $\lambda^\dagger H$ and $(\lambda^\dagger H)U$, a multiplication of $\lambda^\dagger$ with a row-transformed U suffices.

As shown in Fig. 6, all these considerations result in valuable speed-ups for quantum systems of up to 9 spin qubits. In larger systems, though, workspace becomes limiting, as every processor requires M matrices. A way of compensation would be to compute the exponential matrices on demand, *i.e.* at forward and backward propagation respectively. However, this would be a really bad solution because computing the exponential takes most of the computation time already. Therefore system sizes beyond 9 spins cannot be computed with the slice-wise propagation and require alternative methods as described next.

2.2 Tree-Like Propagation: The Parallel Prefix Algorithm

A different approach for computing the propagation is the parallel prefix algorithm [28] depicted in Fig. 5(b). In general, it is applicable to arbitrary combinations of number of processors p and digitisation M. Yet in this article we confine ourselves to $p = M/2$, which is also the maximum reasonable number for p. Our code performs forward and backward propagation simultaneously thus increasing the overall number of processes to M. Under the assumption that $p = M/2$, the computation time of the algorithm is $\mathcal{O}(\log_2 M \cdot N^3)$. In contrast to slice-wise propagation, parallel prefix requires communication during the propagation (red lines in Fig. 5(b)): they sum up to $\sum_{l=2}^{\log_2 M}[\mathit{Broadcast}(N^3, p = 2^{l-1}) + (l-1) \cdot \mathit{Send}(N^3)]$, provided the times for *Broadcast* and *Send* are not influenced by other ongoing communication. Recalling the computation time of $\mathcal{O}(M \cdot N^3/p)$ for the slice-wise propagation and assuming $p = M/2$, parallel prefix should never be faster (neglecting effects like memory prefetching). Fig. 6 shows this is indeed true. On the other hand, parallel prefix does not require all the matrices $U(t_k)$ in all processes, which eliminates the broadcast time prior to the propagation step (see Fig. 4). It is this advantage that is large enough to outweigh the slower propagation time.

Even more important is yet another gain from this property: reduced memory demand. In our current implementation the maximum number of matrices stored at a single process is $\mathcal{O}(\log_2 M)$ [P_0 produces one result in every level], which is already much less than the $\mathcal{O}(M)$ of the slice-wise propagation. In case this should be inacceptable, the number can be reduced to $\mathcal{O}(1)$ by implementing a slightly different communication pattern from the one used here.

The stability of parallel prefix-matrix multiplication deserves a closer look [29]: following Mathias [30], in the general case, this multiplication is numerically unstable. He derives the following first-order upper bound to the error

$$\left| \mathrm{rd}(U_1 \cdots U_M) - (U_1 \cdots U_M) \right| \leq 2(N-1)\,\varepsilon \sum_{k=1}^{M-1} |B_k| + \mathcal{O}(\varepsilon^2)\,, \tag{3}$$

where $|B_j|$ stands for the product of absolute values of the largest matrix elements within the factors U_k. However, with all these matrices being unitary in our case, the same error estimate turns into the "well-behaved" form

$$\left| \mathrm{rd}(U_1 \cdots U_M) - (U_1 \cdots U_M) \right| \leq 2(N-1)M\varepsilon + \mathcal{O}(\varepsilon^2) \quad . \tag{4}$$

In coincidence with the linearity in M, we observed $||U_{\mathrm{tree}} - U_{\mathrm{reg}}||_2$ increasing linearly with the propagation step, where at $k = M = 128$, a value of 1.5×10^{-13} was reached in a nine-qubit system (not shown). This underpins the computation is numerically stable for unitary matrices used in quantum dynamics.

3 Optimising Matrix Exponentials

Computing matrix exponentials numerically is a notoriously intricate problem [31,32]. Here we compare the standard Padé-approximation with the generic QR-approach and a symmetry-adapted QR-variant thus allowing optimised LAPACK routines [33,34] to be employed. With controllable qubit systems permitting pseudo-spin representations in terms of Pauli matrices, their Hamiltonian generators of the exponential map often show 'persymmetry' [26,35], *i.e.*, a matrix representation that is symmetric with respect to the anti-diagonal and which may be induced by the Pauli matrices (*vide infra*). Defining J_N as the $N \times N$ reversal matrix (obtained by reversing the columns of the identity matrix), the persymmetry of a matrix A is equivalent to the condition $J_N A J_N = A^T$.

Lemma 1. *(1) A (finite) Kronecker or tensor product of persymmetric matrices is again persymmetric. (2) The same is true in the tensor product of an* even *number* $2r$ *of matrices that are themselves 'anti-persymmetric' due to the sign change* $J_N A J_N = -A^T$.

Proof. Assertion (1) simply follows from the fact that forming the tensor product and taking the transpose with respect to the anti-diagonal commute. Part (2) is due to the construction $J_N = J_2 \otimes J_2 \cdots J_2 \otimes J_2$, since one finds $J_N(A_1 \otimes \cdots \otimes A_k)J_N = (J_2 A_1 J_2 \otimes \cdots \otimes J_2 A_k J_2) = (-1)^{2r}(A_1^T \otimes \cdots \otimes A_k^T)$. ■

Remark. Let $\{\sigma_x, \sigma_y, \sigma_z\} = \left\{\left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 0 & -i \\ i & 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)\right\}$ be the Pauli matrices. By (2), the drift term comprises the persymmetric terms $\sigma_z \otimes \sigma_z + \alpha(\sigma_x \otimes \sigma_x + \sigma_y \otimes \sigma_y)$ for any real α, while the control terms are tensor products of the unit matrix with σ_x, σ_y, which in turn are persymmetric by (1).

Moreover, we can write the Hamiltonian H in the form $H = D + C$ with a real diagonal matrix D and a multilevel matrix C of the recursive form $C = C_1 \otimes \cdots \otimes C_n$ with 2×2 matrices

$$C_k := \begin{pmatrix} 0 & \gamma_k \\ \gamma_k^* & 0 \end{pmatrix} . \tag{5}$$

Each of the small matrices C_k can be transformed into a real circulant matrix by the unitary diagonal matrix $V_k := \mathrm{diag}(1, \gamma_k/|\gamma_k|)$. Therefore, by the Kronecker products of the small matrices V_k we can transform the entire matrix H to a real symmetric persymmetric matrix $\tilde{H}$ of the form

$$\tilde{H} = \begin{pmatrix} A_1 & r\mathbb{1} \\ r\mathbb{1} & A_2 \end{pmatrix} \tag{6}$$

with real r, symmetric A_1 and A_2, and the identity matrix $\mathbb{1}$. Now, the persymmetry leads to the relation $A_2 = JA_1J$ and thereby to the similarity transform

$$\begin{pmatrix} \mathbb{1} & \mathbb{1} \\ J & -J \end{pmatrix} \begin{pmatrix} A_1 & r\mathbb{1} \\ r\mathbb{1} & A_2 \end{pmatrix} \begin{pmatrix} \mathbb{1} & J \\ \mathbb{1} & -J \end{pmatrix} = \begin{pmatrix} A_1 + A_2 + 2r\mathbb{1} & 0 \\ 0 & A_1 + A_2 - 2r\mathbb{1} \end{pmatrix} . \tag{7}$$

Consequently, the computation of the eigenvalues of H for the matrix exponential $\exp(i\tau H)$ can be reduced to solving the same problem for two real matrices of half the size. For the exponentials we have been using two different methods:

1. classical scaling and squaring algorithm based on the Padé approximation;
2. finding the eigendecomposition of persymmetric $\tau H = UDU^{-1}$ for

$$\exp(i\tau H) = \exp(iUDU^{-1}) = U \exp(iD)U^{-1} = U \,\mathrm{diag}\left(\exp(i \cdot d_j)\right)U^{-1} ;$$

with H being hermitian, this method is numerically stable [31,32]; moreover, using persymmetry, the eigendecomposition then reduces to real symmetric matrices of just half the size.

The major disadvantage of the classical scaling and squaring method with full-sized matrices can be circumvented by a series expansion instead of the Padé approximation, because then only sparse matrix matrix products arise. Expanding in terms of Chebychev polynomials is superior to a Taylor expansion [36,37]. Moreover, the error δ for approximating $\exp(i\tau H)$ by a Chebychev series with m terms can asymptotically be estimated by the Bessel function of the first kind $|J(m, ||\tau H||)| \approx \delta$ given the norm of the effective Hamiltonian $||\tau H||$. Thus one can predict the number of steps required for a given accuracy. Unfortunately, due to the control amplitudes sometimes $||\tau H|| \geq 100$; then the Chebychev expansion only pays if low accuracy suffices. Yet in future code this approximation will be included as an option for matrices of small norm or cases not demanding high accuracy.

Table 1. Contributions of Parallelised Matrix Operations to Overall Speed-up

Subroutine	Fraction of CPU Time with 1 CPU	Fraction of CPU Time with 128 CPUs	Weighted Speed-up
maxStepSize	0.9	0.713	521
getGradient	0.091	0.287	52.6
expm	0.075	0.049	43.0
propagation	0.01	0.194	6.0
gradient	0.006	0.044	3.5
optimiseCG	1	1	576

4 Conclusions and Outlook

We have shown how using the potential parallelism inherent in coherent quantum superpositions relies on methods of classical control theory in order to steer the quantum systems. By the nature of quantum matrix mechanics, this requires powerful matrix calculations backed by architecture-adapted redistribution (Fig. 4). Here we demonstrated a speed-up by more than a factor of 500 for a 10 spin system by way of various matrix-operation techniques (see Tab. 1): slice-wise propagation is advantageous in systems up to 9 spin qubits, while tree-like propagation pays for systems from 10 qubits onwards. Moreover, by making use of the symmetry properties induced by the pseudo-spin structure of controllable spin-qubit systems, faster matrix exponentials are feasible. To further improve gradient-flow algorithms, sparse Hamiltonians will be exploited for matrix multiplications along the lines of Ref. [38] or by using Strassen's method.

By the current extensions, larger spin-qubit systems are in reach thus allowing a broader numerical basis for deducing quantum computational complexity measures. Related symmetry properties can be used in wider classes of quantum systems therefore making the presented tools broadly applicable in quantum technology and control. From a general point of view of numerics, the partial problems of computing the exponential $\exp(iH)$ for a hermitian matrix H and furthermore the task of evaluating a sequence of products of unitary matrices are of broader interest and important far beyond the GRAPE-algorithm. Here, the special case of unitary matrices may lead to improved stability and allow algorithms that are fast but usually known to be unstable or only weakly stable.

Acknowledgements

This work was supported in part by the integrated EU project QAP and by *Deutsche Forschungsgemeinschaft*, DFG, in the incentive SPP 1078 *Quanten-Informationsverarbeitung*, QIV. Helpful discussion with Michael Riss on the 'tree-like' matrix multiplication is gratefully acknowledged.

References

1. Dowling, J., Milburn, G.: Quantum technology: The second quantum revolution. Phil. Trans. R. Soc. Lond. A **361** (2003) 1655–1674
2. Feynman, R.P.: Simulating physics with computers. Int. J. Theo. Phys. **21** (1982) 467–488
3. Feynman, R.P.: Feynman Lectures on Computation. Perseus Books, Reading, MA. (1996)
4. Papadimitriou, C.H.: Computational Complexity. Addison Wesley, Reading, MA. (1995)
5. Shor, P.W.: Algorithms for Quantum Computation. In: Proceedings of the Symposium on the Foundations of Computer Science, 1994, Los Alamitos, California, IEEE Computer Society Press, New York (1994) 124–134
6. Shor, P.W.: Polynomial-Time Algorithms for Prime Factorisation and Discrete Logarithm on a Quantum Computer. SIAM J. Comput. **26** (1997) 1484–1509
7. Jozsa, R.: Quantum Algorithms and the Fourier Transform. Proc. R. Soc. A. **454** (1998) 323–337
8. Cleve, R., Ekert, A., Macchiavello, C., Mosca, M.: Quantum Algorithms Revisited. Proc. R. Soc. A. **454** (1998) 339–354
9. Ettinger, M., Høyer, P., Knill, E.: The Quantum Query Complexity of the Hidden Subgroup Problem is Polynomial. Inf. Process. Lett. **91** (2004) 43–48
10. Deutsch, D.: Quantum Theory, the Church-Turing Principle, and the Universal Quantum Computer. Proc. Royal Soc. London A **400** (1985) 97–117
11. Khaneja, N., Reiss, T., Kehlet, C., Schulte-Herbrüggen, T., Glaser, S.J.: Optimal Control of Coupled Spin Dynamics: Design of NMR Pulse Sequences by Gradient Ascent Algorithms. J. Magn. Reson. **172** (2005) 296–305
12. Schulte-Herbrüggen, T., Spörl, A.K., Khaneja, N., Glaser, S.J.: Optimal Control-Based Efficient Synthesis of Building Blocks of Quantum Algorithms: A Perspective from Network Complexity towards Time Complexity. Phys. Rev. A **72** (2005) 042331
13. Saito, A., Kioi, K., Akagi, Y., Hashizume, N., Ohta, K.: Actual Computational Time-Cost of the Quantum Fourier Transform in a Quantum Computer using Nuclear Spins . quant-ph/0001113 (2000)
14. Blais, A.: Quantum Network Optimisation. Phys. Rev. A **64** (2001) 022312
15. Spörl, A.K., Schulte-Herbrüggen, T., Glaser, S.J., Bergholm, V., Storcz, M.J., Ferber, J., Wilhelm, F.K.: Optimal Control of Coupled Josephson Qubits. quant-ph/0504202 (2005)
16. Brockett, R.W.: Dynamical systems that sort lists, diagonalise matrices, and solve linear programming problems. In: Proc. IEEE Decision Control, 1988, Austin, Texas. (1988) 779–803 see also: Lin. Alg. Appl., 146 (1991), 79–91.
17. Helmke, U., Moore, J.B.: Optimisation and Dynamical Systems. Springer, Berlin (1994)
18. Glaser, S.J., Schulte-Herbrüggen, T., Sieveking, M., Schedletzky, O., Nielsen, N.C., Sørensen, O.W., Griesinger, C.: Unitary control in quantum ensembles: Maximising signal intensity in coherent spectroscopy. Science **280** (1998) 421–424
19. Schulte-Herbrüggen, T.: Aspects and Prospects of High-Resolution NMR. PhD Thesis, Diss-ETH 12752, Zürich (1998)
20. U. Helmke, K. Hüper, J.B. Moore, T. Schulte-Herbrüggen.: Gradient Flows Computing the C-Numerical Range with Applications in NMR Spectroscopy. J. Global Optim. **23** (2002) 283–308

21. Butkovskiy, A.G., Samoilenko, Y.I.: Control of Quantum-Mechanical Processes and Systems. Kluwer, Dordrecht (1990)
22. Lawson, C.L., Hanson, R.J., Kincaid, D., Krogh, F.T.: Basic Linear Algebra Subprograms for FORTRAN usage. ACM Trans. Math. Soft. **5** (1979) 308–323
23. Dongarra, J.J., Croz, J.D., Hammarling, S., Hanson, R.J.: An Extended Set of FORTRAN Basic Linear Algebra Subprograms. ACM Trans. Math. Soft. **14** (1988) 1–17
24. Dongarra, J.J., Croz, J.D., Hammarling, S.: A Set of Level 3 Basic Linear Algebra Subprograms. ACM Trans. Math. Soft. **16** (1990) 1–17
25. Dongarra, J., Duff, I., Sørensen, D., van der Vorst, H.: Numerical Linear Algebra on High-Performance Computers. SIAM (1998)
26. Golub, G.H., van Loan, C.F.: Matrix Computations. The Johns Hopkins University Press, Baltimore (1989)
27. Higham, N.J.: Accuracy and Stability of Numerical Algorithms. SIAM (1996)
28. Ladner, R.E., Fischer, M.J.: Parallel Prefix Computation. J. ACM **27** (1980) 831–838
29. Demmel, J., Heath, M., van der Vorst, H.: Parallel Numerical Linear Algebra. Acta Numerica **2** (1993) 111–198
30. Mathias, R.: The Instability of Parallel Prefix Matrix Multiplication. SIAM J. Sci. Comput. **16** (1995) 956–973
31. Moler, C., van Loan, C.: Nineteen Dubious Ways to Compute the Exponential of a Matrix. SIAM Rev. **20** (1978) 801–836
32. Moler, C., van Loan, C.: Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later. SIAM Rev. **45** (2003) 3–49
33. Anderson, E., Bai, Z., Bischof, C., Blackford, L.S., Demmel, J., Dongarra, J., Croz, J.D., Greenbaum, A., Hammarling, S., McKenney, A., Sørensen, D.: LAPACK User's Guide, Third Edition. SIAM (1999)
34. Blackford, L.S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walkerx, D., Whaley, R.C.: ScaLAPACK User's Guide. SIAM (1997)
35. Cantoni, A., Butler, P.: Eigenvalues and Eigenvectors of Symmetric Centrosymmetric Matrices. Lin. Alg. Appl. **13** (1976) 275–288
36. Rivlin, T.J.: The Chebychev Polynomials. Wiley, New York (1974)
37. Veshtort, M., Griffin, R.: SPINEVOLUTION: A Powerful Tool for the Simulation of Solid and Liquid State NMR Spectra. J. Magn. Reson. **178** (2006) 248–282
38. Grote, M., Huckle, T.: Parallel Preconditioning with Sparse Approximate Inverses. SIAM J. Sci. Comput. **18** (1997) 838–853