Generic parallel algorithms for quantum lattice models

Matthias Troyer

Theoretische Physik, ETH Zürich, CH-8093 Zürich

Abstract. The investigation of strong quantum effects in condensed matter physics is currently one of the major research areas in physics with important application aspects, like the high temperature superconductors. The only reliable approach to these systems are large scale numerical simulations, made difficult by the exponential scaling of most algorithms. We present efforts in developing a generic and parallel program package for the simulation the full quantum of such systems using generic programming techniques of the C++ language, including expression template techniques for quantum operators. We identify the problems and performance bottlenecks encountered in implementing these generic programs.

1 Introduction

Electron correlation effects in solids, caused by strong quantum effects are the cause for a number of interesting phenomena – like high temperature superconductivity – with potentially very important technical consequences. Analytic as well as quantum chemistry approaches fail for these materials since they rely on quantum effects being small. In order to calculate the properties of such materials the full quantum effects must be treated! This makes the simulations exponentially hard. In contrast to quantum chemistry algorithms, which scale polynomially, typically like N^4, the exact algorithms for quantum models scale exponentially, typically like 4^N. This makes these simulations a grand challenge.

In the past decade a variety of algorithms have been developed, each of which is useful for the calculation of certain properties. The programs are usually written and optimized in Fortran specifically for one model. Such a specific implementation contrasts sharply to the generality of the algorithms. The question arises: is it possible to write generic programs using these algorithms, similar to the generic algorithms in the C++ standard template library.

In this paper we report on the PALM++ project (Parallel Algorithms for Lattice Models), an effort to implement a generic and parallel toolkit for these algorithms using generic and object oriented programming features of C++, including expression template techniques for quantum operators. The parallelization of the algorithms allows much larger systems to be simulated and more accurate results to be obtained. The genericity will allow new models to be studied quickly, without having to spend several months on writing a new program. In the long run we hope that these programs will provide all researchers in this field with the possibility of easily performing these essential simulations.

In this paper we will first introduce quantum lattice models for fermions and how they can be represented on a computer using bit strings. Then we discuss the parallelization of the various algorithms. Finally we present benchmarks and feasibility studies which are the basis for the PALM++ project.

2 Quantum lattice models and bit string representations

2.1 Quantum lattice models

As the complete and reliable solution of the full Schrödinger equation for a material is a task which is beyond the power of any computer, approximations have to be made. Quantum chemistry assumes weak quantum effects and works with a nearly-independent electron picture. In strongly correlated materials, where quantum effects are strong, another approach is taken. The quantum effects are treated exactly, but the atoms are simplified. Instead of treating all orbitals, effective models containing only a few of the most relevant orbitals are derived. For these simplified models, the *full quantum* problem is then treated exactly. In that way, a large number of different models are derived for various materials, but they all share common properties which can be used for generic algorithms.

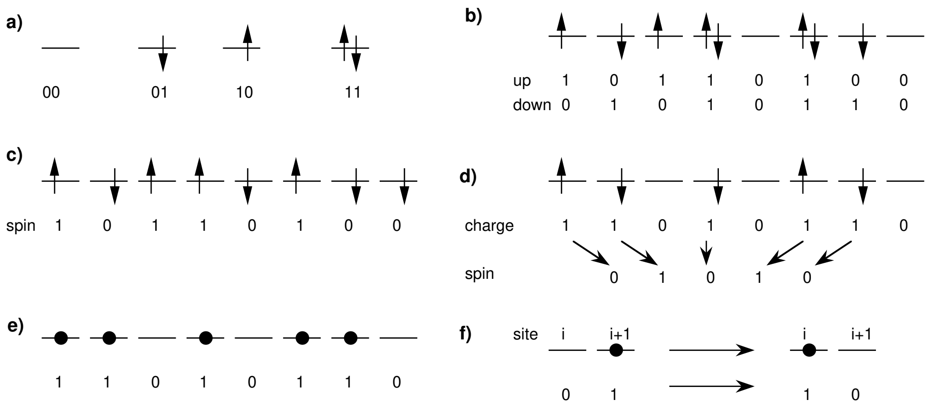

Fig. 1. Quantum mechanical states and their representation by bit strings for a) the four states of a single orbital, b) a linear chain of orbitals, c) a Heisenberg model, d) a t-J-type model, e) a spinless fermion model; f) operation of the quantum operator $c_i^\dagger c_{i+1}$ on a quantum state and on the bit string representation.

We first look at a single orbital. Due to the Pauli exclusion principle, there can be only two electrons in an orbital. One has its spin pointing "up", and one has its spin pointing "down". Thus four states are possible and shown in figure 1a. In a system with L orbitals the total number of states is 4^L, which makes these problems of intrinsically exponential complexity. These states can be represented on a computer using bit strings. The four states of a single orbital can be represented by two bits, one for the up-spin and one for the down-spin. A "1" bit indicates the presence of a particle and a "0" bit its absence. This bit string representation can be extended to the case of L orbitals. There one bit string of L bits is used for the up-spin electrons and another bit string for the down-spin electrons, as is shown in figure 1b. Other models are more restricted. In a Heisenberg model there is always one electron in an orbital, but its spin can be either up or down. In this model a bit string is used, where a "0" bit represents a down spin and a "1" bit an up spin, as is shown in figure 1c. The t-J model is one of the favorite models, as it contains only three instead of four states. A site can contain either a hole (empty), or an up-spin, or a down spin, but not both an up- and a down spin. The usual bit string representation is one bit string of length L where "1" bits represent the positions of the N particles, and another bit string of length N, where the n-th bit represents the spin state (up or down) of the n-th particle, as is shown in figure 1d.

2.2 Quantum operators

We now focus on a simpler model of spinless fermions, where there is only one type of electrons. A "1" bit indicates the presence of an electron and a "0" bit its absence, as shown in figure 1e. The Hamilton (energy) operator describes the physics of a quantum model. Its eigenvalues are the energy levels, and the eigenvectors the wave functions, from which physical properties can be calculated. This Hamilton operator is usually written in the second quantized form using operator notation. For a chain of non-interacting spinless fermions it is

$$H = -t \sum_{i=0}^{L-1} c_i^\dagger c_{i+1} + c_{i+1}^\dagger c_i \tag{1}$$

The symbol $c_i^\dagger$ is an operator which adds an electron on site i. The operator c_i is the adjoint operator and removes an electron from site i. The product $c_i^\dagger c_{i+1}$ attempts to remove an electron from site $i+1$ and to add one on site i. If this is possible – there needs to be an electron on site $i+1$ and no electron on site i – an electron hops from site $i+1$ to site i, as is illustrated in figure 1f. In that case there is a matrix element $-t$ connecting the two states. Using a bit string representation of the quantum states the matrix representation of this operator $c_i^\dagger c_{i+1}$ is thus a very sparse matrix with at most one nonzero matrix element per row! The total Hamilton operator is a sum of about $O(N)$ such terms and is also represented by a sparse matrix in this bit string representation. The matrix has dimension scaling like $O(\exp(N))$ but only $O(N)$ non-zero matrix elements per row. This sparse matrix can easily be calculated using bit operations.

2.3 Calculating matrix elements

H is blockdiagonal, consisting of blocks for each particle number N. The basis for the N particle subspace consists of all bit strings of length L (number of orbitals), with N bits (number of particles) set to one. The bit strings corresponding to the basis elements are usually stored in an array (which we will call `state`).

The Hamilton operator H describes the matrix in quantum operator form. To obtain the matrix elements we need to apply these operators in turn to each of the states. For a term like $-tc_i^\dagger c_{i+1}$ (see figure 1f) this is done as follows:

loop over all indices $n = 1\ldots$dimension of basis
    set $s \leftarrow \texttt{state}(i)$
    apply the quantum operator c_{i+1} by clearing bit $i+1$
        $s' \leftarrow s\&\tilde{}(1 << (i+1))$
    apply the quantum operator $c_i^\dagger$ by setting bit i
        $s'' \leftarrow s'|(1 << i)$
    if s'' is one of the basis elements
        determine the index m such that $s'' = \texttt{state}(m)$
        set the matrix element $H_{m,n} \leftarrow -t$

The bit operations can be optimized by combining the terms $c_i^\dagger c_{i+1} + c_{i+1}^\dagger c_i$. The effect of this sum can be expressed by an **exclusive or** (^) with a bit mask that has bits i and $i+1$ set.

The time consuming part of this algorithm is the determination of the index m of a state s. A binary search in the sorted list of states is possible, but perfect hashing schemes are preferred. For bit strings of small sizes L the representation s can act as key for a hash table. For longer L perfect two-step hashing techniques have been developed for some models.

Translational symmetry can be used to further reduce the number of basis states. Translations are performed through circular bit shifts of the bit string. Fermionic signs arise when hopping terms like $c_i^\dagger c_j$ appear in the model. The matrix elements then are $-t \times (-1)^{\text{number of “1” bits between bits } i \text{ and } j}$. For the calculation of these signs fast *bit count* operations are essential.

3 Algorithms for quantum lattice models and their parallelization

Several types of algorithms have been developed for the simulation of quantum lattice models. Each of these algorithms has its shortcomings but they complement each other.

3.1 Exact diagonalization

Already for a small system of only $L = 40$ spinless fermion orbitals the dimension of the largest subspace is more than 3×10^9. The extreme eigenvalues of such a matrix can be calculated using iterative methods, like the Lanczos algorithm [2]. Parallelization of exact diagonalization calculations can be done by parallelizing the sparse matrix-vector multiplications. The matrices are usually too large to be stored (1 Terabyte for $L = 40$ and $N = 20$). Matrix elements have to be calculated "on-the-fly" at each iteration. This makes the following points essential for optimal performance

- Perfect multi-step hashing functions for the indices of bit strings. This problem is hard when translational symmetries are used.
- Optimized bit operations, such as bit counts, perferably supported by specialized hardware. Benchmarks for the typical bit operations used in these calculations were performed and are reported section 4.1.

3.2 Series expansions

Series expansion techniques express the properties of *infinitely large* systems as a power series in a small parameter. One example is the high temperature series, an expansion in the inverse temperature $1/T$:

$$A = \sum_n a_n T^{-n} \tag{2}$$

The calculation of the coefficient a_n requiring calculating $\mathrm{Tr}H^n$, the trace of the n-th power of H. This needs to be done on finite clusters with n orbitals, represented again by bit strings of length n. The trace $\mathrm{Tr}H^n$ is calculated by looping over all basis vector s_i and calculating $(s_i, H^n s_i)$ using the same sparse matrix-vector multiplication functions as for exact diagonalization. The calculation of the trace can be easily parallelized as the calculations of $(s_i, H^n s_i)$ for different basis vectors s_i are independent, only the final reduction is serial.

As the dimension of the vector spaces grow exponentially with the order n, series are possible up to about 20-th order. This is sufficient to obtain results at intermediate and high temperatures but not at low temperatures.

3.3 Quantum Monte Carlo

Monte Carlo methods for quantum systems map the quantum system to a classical system in a higher dimensional space, which is then simulated using standard Monte Carlo methods. In the mapping to the classical systems again quantum operators are used. Parallelization of Monte Carlo simulations is straight-forward. Independent simulations can be run in parallel and the results averaged at the end. A library for the automatic parallelization of Monte Carlo simulations was developed by the authors and presented last year at this conference [1].

Classical Monte Carlo algorithms for lattice models scale polynomially with the number of lattice sites N. This is also the case for some quantum models, including unfrustrated magnetic and bosonic models. For most models of electrons however, there is a severe cancellation problem, the so-called "negative sign problem" in two or more dimensions. This causes the algorithms to scale exponentially with the number of orbitals N and inverse temperature $1/T$, again restricting its use.

3.4 Density Matrix Renormalization Group

The density matrix renormalization group method [3] is a recent development, which reduces the exponentially large number of basis states by using just a small number $M = m^2$ of important basis states. The algorithm iteratively increases the lattice size by adding two new orbitals at each iteration. In the case of fermions there are four states for one orbital. The two additional orbitals increase the dimension of the basis to $16M$. Out of these $16M$ states the M eigenvectors of the density matrix with the largest eigenvalues are selected as the important basis states, hence the name density matrix renormalization group method. The procedure is then repeated, growing the lattice by two orbitals each iteration.

Numerically, the Hamiltonian matrix H in this truncated basis is no longer sparse, but still block-sparse. A matrix-vector multiplication of this block-sparse matrix can be parallelized by distributing the blocks over the nodes of a parallel computer. We need to investigate whetherthe POOMA-II array library [4] can be used for these purposes.

The DMRG method scales only polynomially in the number of orbitals N and basis states M. In one dimensional systems, the basis size M needed for good accuracy is independent of N. The DMRG algorithm then scales like $O(N)$ complexity and is often the method of choice. In higher dimensional models, the basis size M needed for reliable results unfortunately grows exponentially with N, thus the computational effort again scales exponentially.

4 Are generic algorithms feasible?

In many cases the algorithms discussed in the previous section scale exponentially with the size of the lattice (number of orbitals N). Parallel implementations are thus essential to be able to simulate reasonable sized lattices and to obtain good accuracy. The implementation of the algorithms is made harder by the fact that there is a large number of different models. As we discussed in section 2.1 each model class needs a different bit string representation of quantum states. Up till now, programs have been written specifically for each model.

The aim of the PALM++ project is to develop parallel and generic programs for the four main algorithms used in the simulation of strong quantum correlation effects in solid state physics. The parallelization strategies for the algorithms have been outlined in the previous section. Here we wish to discuss whether a generic implementation is feasible. Of special concern are three issues:

– Are bit operations in C++ as efficient as their FORTRAN counterparts?
– Can these algorithms be implemented in a generic and still efficient way using C++ template techniques?
– Can the implementation of a model be simplified using expression templates for quantum operators, similar to the expression templates technique for array operations developed by Todd Veldhuizen [7]?

We have performed three feasibility studies to answer these questions and report the results in the next sections.

4.1 Benchmarks of bit operations

Table 1. Benchmark of bit operations. Typical bit operations needed for quantum lattice models have been timed on a Cray SV1 and a 333 MHz UltraSparc. Timings are given for 200 iterations over loops over arrays with 5 million elements. 64 bit integers were used on the Cray and 32 bit integers on the UltraSparc. Notes: (1) using Cray intrinsic functions. (2) using array notation instead of loops. (3) could not be done as bit shift operations are not implemented for arrays.

Compiler	array type	exclusive or	circular shift	extract bits	bit count
Cray SV1 C++	C arrays	19 sec	20 sec	16 sec	
Cray SV1 C++	std::vector	19 sec	20 sec	16 sec	
Cray SV1 C++$^{(1)}$	C arrays			16 sec	16 sec$^{(1)}$
Cray SV1 Fortran 90		19 sec	26 sec$^{(1)}$		16 sec $^{(1)}$
UltraSparc KAI C++	C arrays	58 sec	44 sec		247 sec
UltraSparc KAI C++	std::vector	83 sec	71 sec		
UltraSparc KAI C++	std::valarray	74 sec	64 sec		
UltraSparc KAI C++	std::valarray$^{(2)}$	100 sec	$^{(3)}$	$^{(3)}$	$^{(3)}$
UltraSparc KAI C++	blitz::Array$^{(2)}$	64 sec	$^{(3)}$	$^{(3)}$	$^{(3)}$
UltraSparc Fortran 77		60 sec	66 sec		
UltraSparc Fortran 90		66 sec	66 sec		

To answer the first question we have performed benchmarks for a number of important bit operations, such as "exclusive or", circular shifts, extraction of two bits, and bit counts. Timing results for loops over one billion (10^9) iterations are listed in table 1. On the Cray SV1 C++ and Fortran were nearly identical in performance. Intrinsic functions do not lead to any measurable speedup compared to C++ bit operations due to the perfect vectorization of the code. The Fortran intrinsic circular shift of the lower n bits of an integer is slower than the C++ version since the intrinsic function keeps the higher bits unchanged, while our C++ version does not care about them. Use of the `std::vector` class has no performance penalty on the Cray SV1.

On the UltraSparc large variations can be seen, depending on the compiler. The Sun Fortran compilers seem to be not as optimized as the C compilers for out-of-cache problems. We find that array classes – with the exception of the Blitz array class – need some additional tuning to be comparable to hand coded loops using C arrays

Compiler and hardware support for bit counts can be essential for optimal codes, but are not available on most workstations. A large performance drop was seen on the UltraSparc for bit count operations. Bit count operations are used extensively in cryptographic decoding sieves, hence the Cray has good hardware support. On the workstations, without such hardware support, bit counts need to be done using about a dozen bit operations. Then the CPU performance and not memory bandwidth limit the performance. Note that all the problems sizes were chosen on purpose to be much larger than any available cache, as this is the typical case for quantum lattice models.

4.2 Generic algorithms for classical simulations

Generic algorithms, as used in the standard template library, are well known to be similarly efficient as specialized algorithms. For array operations the Blitz++ library[8] was shown to be on average about as efficient as optimized Fortran programs. There is thus hope that also more complex algorithms for lattice models can be implemented as efficient in C++ as in Fortran.

As a benchmark case we have chosen the Wolff cluster update algorithm for classical Monte carlo simulations. This algorithms was originally derived in a generic form [5], but until now only implemented for specific models. We have implemented this algorithm generically, using the `std::vector` and `std::queue` templates [6]. We have compared the performance with an optimized Fortran implementation for a classical Heisenberg spin model. For this model the classical spins are represented by N-vectors, which we implement using the `TinyVector<double,N>` class of the Blitz++ library [8].

Table 2. Benchmark runs for 12000 Monte Carlo sweeps using the Wolff cluster algorithm on a 20^3 lattice at the critical temperature. As C++ compiler the KAI C++ compiler with optimization options `+K3 -O3` was used. As fortran compiler the vendor supplied native Fortran 77 compiler was used, with highest optimizations.

	333 MHz UltraSparc 5	IBM SP2
KAI C++	12.3 sec	21 sec
Native FORTRAN 77	9.51 sec	28 sec
Performance C++/FORTRAN	77 %	133 %

In table 2 we compare the performance of the generic C++ program and the specific Fortran program. We find that the codes perform comparably. Depending on the architecture and compiler, either the Fortran or the C++ program perform better. This raises our confidence that generic C++ programs are feasible also for the complex algorithms that are needed for quantum lattice models.

More details, including source codes, will be published in the near future and can be found on the web page `http://www.itp.phys.ethz.ch/compphys/`.

4.3 Expression templates for quantum operators

The bit string representation of quantum states and operators is very efficient but it is a big task to implement a new operator or model. Ideally we want to be able to code an operation such as

$$|\phi\rangle = (-tc_i^\dagger c_j + \mu(n_i + n_j) + Vn_i n_j)|\psi\rangle \tag{3}$$

as

```
phi = (-t*cdag(i)*c(j) + mu*(n(i)+n(j)) + V*n(i)*n(j))*psi
```

and not as several dozen lines of code, looping over all states, applying bit operations and using model-specific hash functions.

This is possible, in principle, using the expression template (ET) technique developed by Todd Veldhuizen for array expressions. By now we have succeeded in implementing ETs for such quantum operators for the simple case of spinless fermions. This test implementation has identified the following critical issues which will need to be dealt with in a full ET implementation for general quantum lattice models:

1. In C++ operators bind from left to right, while in the standard quantum mechanical notations operators act to the wave function on the right. `cdag(i)*c(j)*psi` is parsed as `(cdag(i)*c(j))*psi`. The quantum ET implementation rearranges this expression to `cdag(i)*(c(j)*psi)` before evaluating it.[1]
2. The vectors `phi` and `psi` can have vastly different dimensions. If `phi` is much smaller than `psi` the matrix elements should not be calculated as in section 2.3. In this case instead of applying the operators to `psi`, the adjoints of the operators should be applied to `phi`. In some cases it is necessary to delegate this decision to runtime.
3. Efficient hashing functions have to be found for the various bit string representations for quantum mechanical states.
4. There are many tricks that can speed up these operator expression. One that was mentioned in section 2.3 is the optimized evaluation of $c_i^\dagger c_{i+1} + c_{i+1}^\dagger c_i$ by one exclusive or operation. The quantum ET implementation has to apply all these optimizations in order to be competitive with hand-coded programs.
5. In contrast to array operations, the quantum operators are actually sparse matrices, and the quantum ET implementation performs a sparse matrix-vector multiplication. This makes the implementation more complex. Note that similar problems have to be dealt with if one wishes to implement stencil operators using ETs. Such an example would be array stencils that allow $(\boldsymbol{v} \cdot \nabla)u$ to be coded as `(v*grad)*u`. It should be possible to use an ET implementation for quantum operators also for array stencil operators.

[1] We want to be able to use the standard notation and not to force the user to set extra parentheses.

5 Conclusions and outlook

A combination of object oriented and generic programming techniques, including extension of the expression template ideas to quantum operators allows the implementation of generic programs for quantum lattice models. We have performed feasibility studies showing that such generic programs are possible and can perform with similar efficiency as specialized Fortran programs.

The PALM++ project (Parallel Algorithms for Lattice models) is an effort lead by the authors in collaboration with researchers at other theoretical physics institutes. It aims at developing a generic parallel toolkit for such simulations.

These parallel programs will allow new models to be investigated with minimal effort, without requiring several weeks to months of coding . They will make numerical simulations, which are essential for current research in solid state physics accessible to all physicists and not just to a small group of computational experts.

Until now a library for the parallelization of Monte Carlo simulations [1], as well as quantum Monte Carlo programs for magnetic systems have been developed. Current research and development efforts focus on parallel and generic implementations of exact diagonalization and series expansion algorithms using expression templates for quantum operators.

C++ is at the moment the only language which permits generic but still optimal implementations. The effort involved in programming expression templates makes it clear that C++ is by no means the ideal language for these efforts. We have identified some of the major problems encountered and hope that computer scientists will develop languages that are better suited for generic scientific computing and in which generic programming and expression templates are more naturally implementable. Bit operations in C++ were found to be efficient, with the exception of bit counts, which need to be supported by hardware and compilers to be efficient.

References

1. M. Troyer, B. Ammon and E. Heeb, in the Proceedings of ISCOPE '98, Lecture Notes in Computer Science **1505**, (Springer Verlag, 1998).
2. C. Lanczos, J. Res. Natl. Bur. Stand. **45**, 225 (1950).
3. S. R. White, Phys. Rev. Lett. 69, 2863 (1992).
4. See e.g. S. Karmesin *et al.* in the Proceedings of ISCOPE '98, Lecture Notes in Computer Science **1505**, (Springer Verlag, 1998) or the web page http://www.acl.lanl.gov/pooma.
5. U. Wolff, Phys. Rev. Lett **62**, 361 (1989).
6. E. Heeb and M. Troyer, Bull. Am. Phys. Soc. **44**, 1, YB13.07.
7. T. Veldhuizen, C++ Report **7**, 26(1995); *ibid.* **7**, 36(1995).
8. See e.g. T. Veldhuizen in the Proceedings of ISCOPE '98, Lecture Notes in Computer Science **1505**, (Springer Verlag, 1998) or the web page http://monet.uwaterloo.ca/blitz.