Extended Computational Kernels in a Massively Parallel Implementation of the Trotter–Suzuki Approximation

ARTICLE *in* COMPUTER PHYSICS COMMUNICATIONS · AUGUST 2015

Impact Factor: 2.41 · DOI: 10.1016/j.cpc.2015.07.017

2 AUTHORS, INCLUDING:

Peter Wittek
ICFO Institute of Photonic Sciences
44 PUBLICATIONS **73** CITATIONS

SEE PROFILE

Extended Computational Kernels in a Massively Parallel Implementation of the Trotter–Suzuki Approximation

Peter Wittek[a,b], Luca Calderaro[c]

[a] *ICFO-The Institute of Photonic Sciences, Av. Carl Friedrich Gauss, 3, 08860 Castelldefels (Barcelona), Spain*
[b] *University of Borås, Allegatan 1, 50190 Borås, Sweden*
[c] *University of Padua, Riviera T. Livio 6, 35123 Padua, Italy*

Abstract

We extended a parallel and distributed implementation of the Trotter–Suzuki algorithm for simulating quantum systems to study a wider range of physical problems and to make the library easier to use. The new release allows periodic boundary conditions, many-body simulations of non-interacting particles, arbitrary stationary potential functions, and imaginary time evolution to approximate the ground state energy. The new release is more resilient to the computational environment: a wider range of compiler chains and more platforms are supported. To ease development, we provide a more extensive command-line interface, an application programming interface, and wrappers from high-level languages.

Keywords: Quantum Evolution, Trotter–Suzuki Algorithm, Parallel and Distributed Computing, GPU Computing

NEW VERSION PROGRAM SUMMARY

Program Title: Trotter–Suzuki-MPI
Catalogue identifier: AEXL_v1_0
Licensing provisions: GNU General Public License, version 3
Programming language: C++, CUDA, Python, MATLAB
Computer: x86-64
Operating system: Linux
RAM: 5 MByte–512 GBytes
Number of processors used: 1–64 in a single node, more in a cluster
Supplementary material:
Keywords: GPU Computing, MPI, Quantum Evolution, Trotter–Suzuki Algorithm, Hybrid Kernel

Classification: 4.12 Computational Methods: Other Numerical Methods
External routines/libraries: OpenMP, MPI, CUDA
Subprograms used:
Journal reference of previous version: Computer Physics Communications 184 (4), 1165–1171 (2013)
Does the new version supersede the previous version?: Yes. The original version is not held in the CPC Program Library but can be obtained from `https://github.com/peterwittek/trotter-suzuki-mpi`.
Nature of problem:
The evolution of a general quantum system is described by the time-dependent Schrödinger equation. The solution of this equation involves calculating a matrix exponential, which is formally simple, but computer implementations must consider several factors to achieve both high performance and high accuracy.
Solution method:
The Trotter–Suzuki approximation leads to an efficient algorithm for solving the time-dependent Schrödinger equation [1, 2]. The implementation relies on high-performance parallel kernels in a distributed environment to maximize the computational power of this algorithm [3, 4].
Reasons for the new version:
The computational kernels were generalized to be able to address a much wider range of physics problems. Furthermore, the code has been modularized to make development easier, providing both a command-line and an application programming interface. High-level wrappers from Python and MATLAB provide further ease of use.
Summary of revisions:

1. The implementation was generalized to include a richer variety of physics problems. The problem can have periodic boundary conditions. Many-body simulations of non-interacting particles became possible extension. We can define an arbitrary stationary potential function. The convenience function `expect_values` helps to obtain expectation values.
2. Imaginary time evolution was implemented to find the ground state before starting the simulation. To avoid imposing the overhead of conditional branching in the most computationally intense parts of the code, some of the core kernel functions were duplicated to include the imaginary time evolution.
3. Most of the functionality is exposed through a command-line interface (CLI) for convenience. This allows specifying the files of the initial state and the potential, the parameters of the Hamiltonian, and further parameters related to the simulation, such as the computational kernel to use and the frequency at which snapshots should be written to the disk.

	CPU	SSE	GPU	Hybrid
old release	6512259	4835126	2566198	1758244
new release	7721790	6634026	3110414	2120868
ratio of slow down	19%	37%	21%	20%

Table 1: Comparison of execution time between the old and the new release. The unit is microseconds.

4. The full functionality of the implementation is exposed as an application programming interface (API) through the 'trotter' function. This allows for integrating the simulation in a larger MPI programme and it is also useful for initializing the state and the potential without having files on the disk. To demonstrate the use of the API, several examples are provided with the code.
5. To further ease development, we redesigned the structure of the implementation, making it more modular. We also introduced a unit testing framework to avoid regression.
6. We improved the testing of MPI dependencies by the configure script and allow compilation without MPI. We also improved the treatment of Intel and Visual C++ compilers.
7. We developed wrappers for Python and MATLAB for the CPU kernel for a high-level interface with the library.

Restrictions:
The vectorized CPU kernel must have a tile width that is divisible by two. This puts a constraint on the possible matrix sizes for this kernel. For instance, running twelve MPI threads in a 4×3 configuration, the dimensions must be divisible by six and eight.
Unusual features:
The library currently only supports the CPU kernel under Windows. The Python and MATLAB wrappers support the CPU and SSE kernels.
Additional comments:
The high-performance kernels were independently extended to study spin dynamics [5]. It remains for future work to include lattice models in this implementation.
Running time:
The generalization slightly altered the memory access patterns of the computational kernels, yielding performance penalty of approximately 20 % compared to the previous version (Table 1). The scaling properties did not change and we see a near-optimal scaling when increasing the number of nodes. The actual running time depends on the system size and the duration to be simulated, and the com-

putational resources. It can range from a few seconds to several days.
Acknowledgment: LC was sponsored by the Erasmus+ programme. Further calculations were sponsored by the Spanish Supercomputing Network (FI-2015-2-0023).

[1] H. Trotter, On the product of semi-groups of operators, Proceedings of the American Mathematical Society 10 (1959) 545–551.

[2] M. Suzuki, Decomposition formulas of exponential operators and Lie exponentials with some applications to quantum mechanics and statistical physics, Journal of Mathematical Physics 26 (1985) 601.

[3] C. Bederián, A. Dente, Boosting quantum evolutions using Trotter–Suzuki algorithms on GPUs, in: Proceedings of HPCLatAm-11, 4th High-Performance Computing Symposium, Córdoba, Argentina, 2011.

[4] P. Wittek, F. Cucchietti, A second-order distributed Trotter–Suzuki solver with a hybrid CPU-GPU kernel, Computer Physics Communications 184 (4) (2013) 1165–1171. doi:10.1016/j.cpc.2012.12.008.

[5] A. D. Dente, C. S. Bederián, P. R. Zangara, H. M. Pastawski, GPU accelerated Trotter–Suzuki solver for quantum spin dynamics, arXiv:1305.0036.