Parallel execution and scriptability in micromagnetic simulations

Thomas Fischbacher,[1] Matteo Franchin,[1,2] Giuliano Bordignon,[1,2]
Andreas Knittel,[1] Hans Fangohr[1]

[1]*School of Engineering Sciences,* [2]*School of Physics and Astronomy,*
University of Southampton, SO17 1BJ Southampton, United Kingdom

We demonstrate the feasibility of an 'encapsulated parallelism' approach towards micromagnetic simulations that combines offering a high degree of flexibility to the user with the efficient utilization of parallel computing resources.

While parallelization is obviously desirable to address the high numerical effort required for realistic micromagnetic simulations through utilizing now widely available multiprocessor systems (including desktop multicore CPUs and computing clusters), conventional approaches towards parallelization impose strong restrictions on the structure of programs: numerical operations have to be executed across all processors in a synchronized fashion. This means that, from the user's perspective, either the structure of the entire simulation is rigidly defined from the beginning and cannot be adjusted easily, or making modifications to the computation sequence requires advanced knowledge in parallel programming.

We explain how this dilemma is resolved in the `Nmag` simulation package in such a way that the user can utilize without any additional effort on his side both the computational power of multiple CPUs and the flexibility to tailor execution sequences for specific problems: simulation scripts written for single processor machines can just as well be executed on parallel machines and behave in precisely the same way, up to increased speed. We provide a simple instructive magnetic resonance simulation example that demonstrates utilizing both custom execution sequences and parallelism at the same time. Furthermore, we show that this strategy of encapsulating parallelism even allows to benefit from speed gains through parallel execution in simulations controlled by interactive commands given at a command line interface.

INTRODUCTION

The recent computer architecture design strategy to improve CPU power by including multiple computational cores within one processor has strongly enhanced the relevance of program parallelization, which hitherto was largely confined to supercomputing centers and expert groups. Utilizing both cores in a dual-core processor (now even available in many notebook PCs) should – in principle – allow one to roughly halve the computing time needed for a lengthy numerical simulation.

While most scientists and engineers have some degree of experience with writing sequential programs, writing – and especially debugging – parallel programs generally is a very different endeavour, as a number of conventions as well as restrictions on the sequence of commands have to be obeyed in a parallel program.

Therefore, it is desirable to have a generic framework for micromagnetic simulations which is both simple enough to be usable by engineers and scientists without special training in parallel programming, yet at the same time utilizes parallel resources when available – which may be a multi-node computational cluster, a supercomputer, or simply a multi-core desktop machine.

The `Nmag` micromagnetic simulation environment has this degree of flexibility and allows the user to set up all simulation scripts as if for sequential computing only. If executed on a multi-processor hardware, the computationally demanding parts of the simulation will automatically be executed in parallel to improve speed.

PARALLELISM UNDERNEATH SEQUENTIAL PROGRAMS

Parallel continuum physics simulations (such as Magpar [1] and `Nmag` [2] for micromagnetism) usually partition the physical region into districts which are then distributed over a number of computing nodes. The underlying rationale is that for large systems, the proportion of surface nodes to volume nodes becomes small, hence even slow communication paths (i.e. a network) can be utilized without serious efficiency penalties to exchange data associated to district interface nodes across computational units. Numerical applications of this type usually employ the Message Passing Interface [3] (MPI) to provide basic parallel communication facilities as well as some parallel linear algebra package such as e.g. PETSc [4] for distributed matrix/vector operations.

The fundamental problem with such data-parallel simulations is that most types of computational steps that involve distributed matrices or vectors have to be executed across all computing nodes synchronously in a fixed order. Usually, this is achieved by running on all computing nodes a single given program in which execution order is completely determined right from the start (cf. figure 1(a)).

Generally, whenever the situation occurs that the type and sequence of parallel operations depends on data not known when the program starts (i.e. external inputs or earlier simulation results), the communication scheme has to be extended to transport not only numerical data,

but also control statements which ensure that all nodes agree on the type and order of future computational command sequences to be executed.

From the perspective of the user, it would be highly desirable to be able to set up simulations in such a way that they can involve (i) arbitrarily complex operations (including using third party software libraries, reading and writing data files, etc.) yet at the same time (ii) employ parallelism in order to improve performance, ideally without having to change simulation scripts for parallel execution. This requires operations not related to the simulation (such as writing results to a file) to be associated to a single computing node – the machine on which the simulation was started, while the role of the other machines 'only' is to provide extra computing power for operations on large distributed data.

This approach has been realised in the `Nmag` micromagnetism package:

(i) `Nmag` (presently) is an extension to the general purpose programming language Python [5], giving the user the freedom to write micromagnetic simulation scripts as simple or as complex as needed.

(ii) The underlying continuum physics simulation core `Nsim` [6] has been extended with capabilities to set up and manage parallel computations in a way that completely hides the technical complexity of parallel execution from the user.

Therefore, the user perceives the package to behave like an ordinary single-processor program which can either be started like any other nonparallel program, or alternatively also in parallel execution mode. The numerical computations required to do simulations are automatically parallelized internally as the system distributes command sequences and data from the master node to the 'slave' nodes as needed, while the behaviour for essentially sequential operations such as writing data remains unchanged, cf. figure 1(b).

Internally, this involves a number of somewhat tricky software engineering issues concerning e.g. synchronizing de-allocation of parallel resources (such as matrices) that ultimately has to be driven by cleanup mechanisms at a higher level which (unfortunately) are intrinsically nondeterministic, or transporting callable functions over the network for matrix setup. These technical implementation aspects are explained in detail in [6].

AN EXAMPLE

The flexibility and utility of this approach is demonstrated by the simulation script shown in figure 2 that integrates parallelizable micromagnetism with optimization routines from the independent software library `SciPy` [7] which was not written for parallel execution

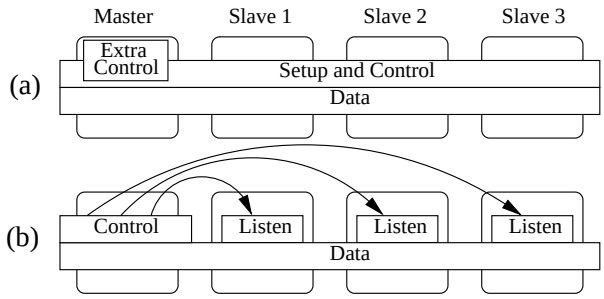

FIG. 1: Design of the control structure of most parallel programs (a) and `Nsim` (b).

(and hence does not care about satisfying the requirements of parallelizable programs). This example determines a resonance frequency of a cylindrical Permalloy nanopillar of dimensions $r = 3\,\mathrm{nm}$, $h = 10\,\mathrm{nm}$ subjected to an electromagnetic wave polarized linearly with $\vec{B}$ perpendicular to the pillar's axis. The EM wave is assumed to interact with the ferromagnet via its $\vec{B}$ component only. In this script, the amplitude-computing function (lines 31ff) is passed on as an objective function to the optimizer (line 39) and internally uses (line 32) another function (defined in lines 15ff) that performs a micromagnetic simulation (line 26) with periodically changing external applied field (line 25). This example can either be started in single-processor mode via `nsim resonance.py`, or alternatively in parallel mode under MPI (here mpich v1) control via: `mpirun -np N nsim resonance.py`, where `N` is the number of CPUs to be used.

There are many other situations beyond optimization problems (as shown in this toy example) where such computational steering of simulations by other code is useful. Indeed, this encapsulation of parallelism even allows to start up the `Nsim` core in interactive mode, set up a simulation, then manually change parameters (such as magnetization) *interactively from a command line prompt*, and interactively extract results on how magnetic fields and energies change. During this interactive session, computations utilize parallelism, but the only way this is noticed by the user is through improved speed (see also the example "Example: IPython" in the `Nmag` manual [8]).

RESULTS AND DISCUSSION

Using a tetrahedral mesh with 214 points and an average node distance of 1.6 nm, we find that this nano-pillar has a resonance frequency around 6 GHz whose exact location is determined (by the script shown in figure 2) to lie at 6.55 GHz. Re-doing the computation with another mesh containing only 138 nodes validates this result.

```
import nmag, math, scipy.optimize
from nmag import SI

Py = nmag.MagMaterial(name = 'Py',
                      Ms = SI(1e6, 'A/m'),
                      llg_damping=0.02,
                      exchange_coupling =
                      SI(13.0e-12, 'J/m'))

A_mag = SI(0.001e6, "A/m") # amplitude
sim = nmag.Simulation()
sim.load_mesh('rod.nmesh.h5',[('rod', Py)],
              unit_length=SI(1e-9,'m'))

def simulate_resonance(freq,
                       amplitude=A_mag,
                       dt=SI(0.5e-12,"s"),
                       time_steps=3000):
    sim.set_m([0,0,1])
    angle_step=(2*math.pi*freq*dt).value
    result=[]
    for i in range(time_steps):
        h_ext=[amplitude.value*
               math.sin(i*angle_step),0,0]
        sim.set_H_ext(h_ext,SI(1,"A/m"))
        sim.advance_time(i*dt)
        avg=sim.get_subfield_average_siv("M","Py")
        result.append((i*dt,h_ext,avg))
    return result

def amplitude(freq_GHz):
    res=simulate_resonance(SI(freq_GHz*1e9,"1/s"))
    osc=[[r[0].value,r[2][0],r[2][1],r[2][2]]
         for r in res]
    (f,params)=nmag.fit_oscillation(osc[2000:])
    a=math.sqrt(sum([p[1]*p[1] for p in params]))
    return -a # we minimize -(amplitude)!

freq=scipy.optimize.fmin(amplitude,
                         [6.0], # start: 6 GHz
                         xtol=0.05,
                         ftol=0.05)
print "Resonance frequency: %.2f GHz" % freq
```

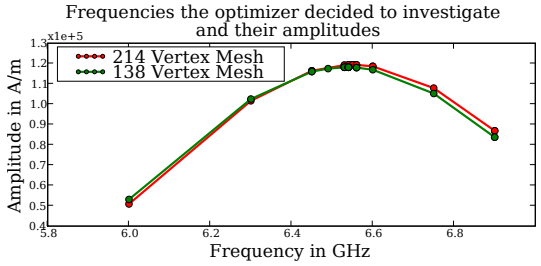

FIG. 2: Above: Script that combines parallel micromagnetic simulation with a nonparallel optimization library to find a resonance. Below: frequencies and corresponding amplitudes for which the optimizer invoked micromagnetic simulations.

While for specific tasks – such as determining resonance frequencies in the linear regime – there sometimes are potent tools available tailored to the problem (such as rkmag [9] for the example chosen here), the point to emphasize is that there are many involved problems for which specialized tools do not exist. In such situations, having access to micromagnetic simulation capabilities which support both parallel computing as well as full integrability with other readily existing sophisticated software components (such as optimization or statistics libraries, or databases) can be a substantial advantage.

While the meshes used above for the sake of convenient fast reproducibility of our results are too small to benefit from parallelization, parallel simulations involving a larger system that are documented in the `Nmag` manual [8] show a 193% speedup of the numerical part of the simulation when run on two processors (i.e. excluding non-parallelizable steps such as writing data), demonstrating the general viability of the approach presented here.

CONCLUSION

The `Nmag` micromagnetic simulation framework has been designed as an extension to a well-established general purpose programming language in order to avoid the otherwise ubiquituous trap of 'growing' an application specific language. The immediate benefit of this design decision is free interoperability with other software components.

This approach allows the user to set up scripts for micromagnetic simulations, which tend to be very simple if common problems are addressed (such as the computation of a hysteresis loop). When investigating less common problems, the user has full freedom to employ as much complexity as needed (which may be partially provided by 3rd party software libraries). Furthermore, any (apparently sequential) script can utilize parallel computing resources in a way that hides all the parallelism-related technological complexity from the user.

[1] W. Scholz, J. Fidler, T. Schrefl, D. Suess, R. Dittrich, H. Forster, V. Tsiantos, *Scalable Parallel Micromagnetic Solvers for Magnetic Nanostructures*, Comp. Mat. Sci. 28 (2003) 366-383.

[2] T. Fischbacher, M. Franchin, G. Bordignon, H. Fangohr, *A Systematic Approach to Multiphysics Extensions of Finite-Element-Based Micromagnetic Simulations: Nmag*, in IEEE Transactions on Magnetics, 43, 6, 2896-2898 (2007)

[3] D. W. Walker, *The design of a standard message passing interface for distributed memory concurrent computers*, in Parallel Computing, 20, 4, 657–673 (1994).

[4] S. Balay, K. Buschelman, L. Dalcin, V. Eijkhout, W. Gropp, D. Karpeev, D. Kaushik, M. Knepley, L. Curfman McInnes, B. Smith, H. Zhang, *PETSc home page, Version 2.3*, `http://www-unix.mcs.anl.gov/petsc/petsc-2/` (2007).

[5] G. van Rossum, Python Website, `http://www.python.org` (2008).

[6] T. Fischbacher, H. Fangohr, *On the automatic translation of classical field theory to parallelized computer simulations and the* `Nsim` *prototype compiler*, in preparation.

[7] E. Jones, T Oliphant, P. Peterson and others, *SciPy: Open Source Scientific tools for Python* 2001 `http://www.scipy.org`

[8] Nmag, manual and code available at `http://nmag.soton.ac.uk`

[9] K. Rivkin, J. B. Ketterson, *Micromagnetic simulations of absorption spectra*, Journal of Magnetism and Magnetic Materials, 306, 204 (2006).