Otto-von-Guericke-University Magdeburg
Max Planck Institute for Dynamics of Complex Technical Systems
Computational Methods for Systems and Control Theory

Dr. Jens Saak, Dipl.-Math. Martin Köhler
Website: http://www.mpi-magdeburg.mpg.de/csc/teaching/19ss/stmsc

Summer Term 2021

Seminar: Scientific Computing
Presentation Topics and Related Literature
March 11, 2021

The order of the topics is completely random so far. If you find a topic you would be interested in, you may propose a week in the term you would like to use for the seminar slot. Note, however, that some topics are better suited for early presentation, while others might benefit from information from part II of the lecture and should be presented rather late in the term.

Open Source Licenses Publishing code is a critical issue in modern scientific computing. One publishes work and especially research ideas and if proper care is taken one can decide how much of the copyright stays with oneself and what others are allowed to do with the codes. This, however, requires a basic knowledge of existing license models. Several open source licenses that allow more or less flexible use of the codes by others. The differences are small but may have critical consequences. A good starting point for the literature study are the two web sites [30, 33].

Reproducibility and Documentation of Computer Experiments The "Reproducibility Crisis" is a standing term across the sciences. It describes the problem of a lack of documentation of any kind of computer experiment and the corresponding lack of availability of the corresponding software, that are used to draw scientific conclusions. A recent literature review and best practice guide is available in [20].

MEX-extensions for MATLAB® and OCT-extensions for Octave using Own Codes MATLAB and its open-source counterpart GNU Octave provide binary interfaces to integrate external libraries into own written codes or to combine MATLAB and Octave scripts with own written C and Fortran codes to improve the performance of your projects. The talk should introduce both the MATLAB [10] and the Octave [14] interface and how they are used and how the data interchange works.

Scientific Computing in Python I & II With the introduction of a proper n-d Array by the NumPy [13] Package the foundation for efficient scientific computations in Python was established. Today a huge number of packages is available including, but not limited to SciPy [16], matplotlib [11], mpi4py [12]. Also sophisticated MATLAB alternatives like the Scientific PYthon Development EnviRonment (spyder) [15] make it possible to write high performance numerical codes avoiding costly licenses and still benefiting from a certain abstraction compared to programming in C, C++, or Fortran.
Taken by: Shaimaa Monem Abdelhafez

Julia — The Basics The Julia Programming language [9] is a fresh approach to develop a new programming language for scientific computing. The authors say:

> "Julia is a high-level, high-performance dynamic programming language for technical computing, with syntax that is familiar to users of other technical computing environments. It provides a sophisticated compiler, distributed parallel execution, numerical accuracy, and an extensive mathematical function library. The library, largely written in Julia itself, also integrates mature, best-of-breed C and Fortran libraries for linear algebra, random number generation, signal processing, and string processing."

The talk should introduce the basic of the programming language and point out some of the unique features which distinguishes it from other scientific programming environments like MATLAB, GNU Octave or NumPy/SciPy.
Taken by: Julius Martensen

Julia — Parallel Features Most modern computers possess more than one CPU, and several computers can be combined together in a cluster. Harnessing the power of these multiple CPUs allows many computations to be completed more quickly. The Julia language provide several construct to parallelize the workload using different techniques like remote execution, parallel loops, shared arrays, coroutines, ... By the design of the language this works on a classical desktop computer as well as on a huge cluster. The talk should present the main parallel programming features of Julia and their differences to "standard" shared and distributed memory parallelization techniques.

Modeling Application Behavior by the Roofline Model Todays largest supercomputers have an energy consumption that requires them to be located next to huge power plants. The ever increasing demand for computing power has raised the energy consumption to a more than critical level. Over the recent few years the Green500 [23] list has introduced a new ranking of supercomputers that takes their energy consumption into account. Energy measurement metric and power saving methodologies [5] today play an important role for many super computing centers. The roofline model is one approach to estimate the performance of an algorithm with respect to a given performance measure. Those measures can be floating point operations, but also energy or combinations of both.

Posit Arithmetic A new data type called a posit is designed as a direct drop-in replacement for IEEE Standard 754 floating-point numbers (floats). Unlike earlier forms of universal number (unum) arithmetic, posits do not require interval arithmetic or variable size operands; like floats, they round if an answer is inexact. However, they provide compelling advantages over floats, including larger dynamic range, higher accuracy, better closure, bitwise identical results across systems, simpler hardware, and simpler exception handling. Posits never overflow to infinity or underflow to zero, and "Not-a-Number" (NaN) indicates an action instead of a bit pattern. A posit processing unit takes less circuitry than an IEEE float FPU. With lower power use and smaller silicon footprint, the posit operations per second (POPS) supported by a chip can be significantly higher than the FLOPS using similar hardware resources. GPU accelerators and Deep Learning processors, in particular, can do more per Watt and per dollar with posits, yet deliver superior answer quality.
Taken by: Aleksey Maleyko

Vector Units of Modern CPUs Since about 8 years it is no longer possible to increase the clock rate of a processor without consuming unacceptably much energy, or getting into trouble with the signal-runtime. One way to increase the performance of a CPU is to do many operations in parallel. The lowest level parallel operations are directly implemented in the CPU as vector units, such as MMX, SEE2, AVX or AltiVec. They can be exploited in optimized program code and compilers [25, 1, 27, 3] or assembly language. Knowing how to use these vector units optimally is a key ingredient to every scientific computing code.
Taken by: Max Jenke

OpenCL Currently there is no clear standard for the programming model in applications involving graphics processing units (GPUs). Nvidia as one of the most important hardware manufacturers is pushing their C language extension CUDA, while AMD/Intel/ARM as their competitor is following the general OpenCL framework that in principle allows to be applied for arbitrary accelerator devices. The presentation should give a concise introduction to the OpenCL programming model and the necessary tools.

OpenACC and OpenMP Offloading If accelerators come into play often people are not using them because of the time consuming process of porting their code to specialized platforms with different programming schemes such as OpenCL or CUDA. The OpenACC framework and the offloading features of OpenMP 4 aim to make this process more and more easy and applicable to many applications by annotating existing code and move the real work to the compiler. Even if no accelerators are available the code can be compiled into a classical CPU-only application without any changes. In this way the programmer can write its programs as usual in its favorite programming language without thinking of the actual way it is transformed to the hardware specific code. This enables people to write codes for various (future) accelerator devices without rewriting code for each new generation of them.

StarPU StarPU is a task programming library for hybrid architectures. It enables the programmer to write task-based algorithms with automatic dependency tracking. The StarPU framework allows to formulate the algorithms once and let the runtime system decide whether the tasks are executed on the CPU, on a GPU using CUDA, on a GPU using OpenCL, or even on a distributed memory system. This decision is made using various performance

models. Furthermore, the necessary data transfers between the CPU and the accelerators are hidden from the user and performed on demand. This makes it easy to turn CPU based implementations into GPU-accelerated ones.A good overview is available in the corresponding research report [18].

Profiling and Debugging Finding memory leaks or analyzing code that has strange/unexpected behavior are two of the most time consuming tasks in software engineering. Debuggers support the programmer in analyzing memory access of a program, running a program step by step or viewing variables and internal data structures during the runtime [2, 31, 7]. Some of them are able to detect problems with respect to parallel parts in the program too. On the other hand, profilers allows to detect correct but slow and badly implemented code. Some of these tools are able to give hints how the programs can be improved to get a more efficient implementation [4, 6, 8]. Both categories of tools are the swiss-army-knifes in scientific programming to get a correctly working and efficient program.

Model Order Reduction Many applications in science and technology today deliver very large systems of differential equations after discretization. Often the states x of these models can be manipulated by certain control inputs u and one is interested in specific observations y that might be measurements of the state in certain positions. The dimensions of u and y are typically much smaller than that of x. Therefore people are searching for a way to compute a good approximation $\hat{y}$ to y for the same input u, by solving a much smaller dynamical system represented by the state $\hat{x}$ with a much smaller dimension than that of x. For linear dynamical systems this problem is mainly solved today. The book [17] by Antoulas is a nice introduction to methods for this type of problems. Basic methods are also described in the Model Order Reduction Wiki [32], where additional references can be found.

$\mathcal{H}$-matrices and Tensor Methods The Hierarchical Matrix format [24, 22, 26] is a so called data sparse storage scheme for a class of densely populated matrices that allows storage and application in linear-poly-logarithmic complexity.

Parallel in Time Methods The solution of ordinary differential equations (ODE) is a key task for many simulations or control process. Normally, it is a strictly sequential scheme which only uses parallelization in the computation of a single time step. If the time horizon gets too large and the single step are very expensive one want to use a distributed cluster to accelerate the whole process. Therefore, the Parallel-in-Time integration is one approach to create an parallel scheme out of the sequential ODE integration by distributing time slices over all participating processors. The presentation should give a basic introduction how these method works and how a parallel implementation could be realized. [28, 19, 29, 21]

References

[1] *Auto-vectorization with gcc 4.7*. http://locklessinc.com/articles/vectorize/.

[2] *GDB: The GNU Project Debugger*. http://www.gnu.org/software/gdb/.

[3] *GNU C Compiler - using vector instructions through built-in functions*. http://gcc.gnu.org/onlinedocs/gcc-4.7.2/gcc/Vector-Extensions.html.

[4] *GPROF Tutorial – How to use Linux GNU GCC Profiling Tool*. http://www.thegeekstuff.com/2012/08/gprof-tutorial/.

[5] *Green 500 metrics and methodologies forum*. http://www.green500.org/forum.

[6] *Intel Advisor*. http://software.intel.com/en-us/intel-advisor-xe.

[7] *Intel Inspector*. http://software.intel.com/en-us/intel-inspector-xe.

[8] *Intel VTune Amplifier*. http://software.intel.com/en-us/intel-vtune-amplifier-xe.

[9] *The julia programming language*. http://julialang.org/.

[10] *Matlab mex-file creation api*. http://www.mathworks.com/help/matlab/call-mex-files-1.html.

[11] *matplotlib*. http://matplotlib.org/.

[12] *Mpi for python (mpi4py)*. http://code.google.com/p/mpi4py/.

[13] *Numpy*. http://www.numpy.org/.

[14] *Octave external code interface*. https://www.gnu.org/software/octave/doc/interpreter/External-Code-Interface.html#External-Code-Interface.

[15] *Scientific python development environment (spyder)*. http://code.google.com/p/spyderlib/.

[16] *Scientific tools for python (scipy)*. http://www.scipy.org/.

[17] A. C. ANTOULAS, *Approximation of Large-Scale Dynamical Systems*, SIAM Publications, Philadelphia, PA, 2005.

[18] C. AUGONNET, S. THIBAULT, AND R. NAMYST, *StarPU: a Runtime System for Scheduling Tasks over Accelerator-Based Multicore Machines*, Research Report RR-7240, INRIA, Mar. 2010.

[19] L. BAFFICO, S. BERNARD, Y. MADAY, G. TURINICI, AND G. ZÉRAH, *Parallel-in-time molecular-dynamics simulations*, Physical Review E, 66 (2002), p. 057701.

[20] J. FEHR, J. HEILAND, C. HIMPE, AND J. SAAK, *Best practices for replicability, reproducibility and reusability of computer-based experiments exemplified by model reduction software*, AIMS Mathematics, 1 (2016), pp. 261–281.

[21] M. J. GANDER AND E. HAIRER, *Nonlinear convergence analysis for the parareal algorithm*, in Domain decomposition methods in science and engineering XVII, U. Langer, M. Discacciati, D. E. Keyes, O. B. Widlund, and W. Zulehner, eds., vol. 60 of Lect. Notes Comput. Sci. Eng., Springer-Verlag, 2008, pp. 45–56.

[22] L. GRASEDYCK, *Theorie und Anwendungen Hierarchischer Matrizen*, Dissertation, University of Kiel, Kiel, Germany, 2001. In German, available at http://e-diss.uni-kiel.de/diss_454.

[23] *The Green500 list*, 2017. Available at http://www.green500.org.

[24] W. HACKBUSCH, *Hierarchische Matrizen. Algorithmen und Analysis*, Springer-Verlag, Berlin, 2009.

[25] G. HAGER AND G. WELLEIN, *Introduction to High Performance Computing for Scientists and Engineers*, CRC Press, Inc., Boca Raton, FL, USA, 1st ed., 2010.

[26] *$\mathcal{H}$lib 1.3*. http://www.hlib.org, 1999–2009.

[27] INTEL, *Intel64 and IA-32 Architectures Optimization Reference Manual*, tech. rep., Intel Corp., 2012. Available at http://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-optimization-manual.pdf.

[28] J.-L. LIONS, Y. MADAY, AND G. TURINICI, *Résolution d'EDP par un schéma en temps "pararéel"*, Comptes Rendus de l'Académie des Sciences. Série I. Mathématique, 332 (2001), pp. 661–668.

[29] Y. MADAY, *The parareal in time algorithm*, tech. rep., Laboratoire Jacques-Louis Lions, 2008. Available from https://www.ljll.math.upmc.fr/publications/2008/R08030.html.

[30] OPENSOURCE.ORG, *About open source licenses*. http://opensource.org/licenses. last visited 2015-03-26.

[31] J. SEWARD, N. NETHERCOTE, J. WEIDENDORFER, AND THE VALGRIND DEVELOPMENT TEAM, *Valgrind 3.3 — Advanced Debugging and Profiling for GNU/Linux applications*, Network Theory Ltd, 2008. More information available at: http://valgrind.org/.

[32] THE MORWIKI COMMUNITY, *MORwiki - Model Order Reduction Wiki*. http://modelreduction.org.

[33] TLDRLEGAL COMMUNITY, *Software licenses in plain english*. https://tldrlegal.com/.