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/mpcsc/lehre/2013_SS_STMSC/

Summer Term 2013

Selected Topics in Modern Scientific Computing

Presentation Topics and Related Literature

February 22, 2013

Hierarchical matrices The Hierarchical Matrix forma [18, 17, 19] 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.

GPU Computing: CUDA vs. 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/ATI as their competitor is following the general OpenCL framework that in principle allows to be applied for arbitrary accelerator devices.

Sparse Matrices on GPUs I: The ELLR-T storage format The storage of sparse matrices on GPUs is especially difficult. The ELLR-T storage format[24, 25] is one possible way to overcome the drawbacks of the classic sparse storage formats.

Sparse Matrices on GPUs II: The Interleaved CRS storage format The interleaved CRS format[21] is another sparse storage format tailored for the special structure of the hardware model of modern GPUs.

MATLAB® I: The GPU computing toolbox GPU computing has also arrived in MATLAB. This talk is giving an overview of the capabilities and limitations of the corresponding toolbox.

MATLAB II: The parallel computing toolbox The parent toolbox of all parallel computation approaches that go beyond multicore computing in MATLAB. Again features and limitations will be presented.

Scientific Computing in Python I & II With the introduction of a proper n-d Array by the NumPy[12] 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[14], matplotlib[10], mpi4py[11]. Also sophisticated MATLAB alternatives like the Scientific PYthon Development EnviRonment (spyder)[13] 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.

Cache coherent Non-Uniform Memory Access (architecture/difficulties/tools) Non-uniform memory access[22, 16] has become an important topic in modern scientific computing with the introduction of multicore CPUs. Those CPU often share certain levels in the Cache hierarchy among a number of Cores and others are exclusive to single cores. To ensure data efficient implementation of algorithms additional measures have to be taken.

GreenHPC: Environmentally Responsible Supercomputing 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[6] 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.

Vector units on modern CPUs Since about 5 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 [16, 1, 20, 3] or assembly language. Knowing how to use these vector units optimally is a key ingredient to every scientific computing code.

Parallel Finite Elements Many tasks in modern finite element method (FEM) analysis of physical processes in the sciences can be treated on single desktop machines. However the increasing power of FEM software is accompanied with an ever increasing demand of detail knowledge on the scientists side. Therefor, specialized techniques for the treatment of modern scientific problems on distributed parallel machines, such as Linux clusters, have been pursued over the years. [15]

Profiling & 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, 23, 8]. 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, 7, 9]. Both categories of tools are the swiss-army-knifes in scientific programming to get a correctly working and efficient program.

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] *The green500 list*. http://www.green500.org/.

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

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

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

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

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

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

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

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

[15] C. C. DOUGLAS, G. HAASE, AND U. LANGER, *A Tutorial on Elliptic PDE Solvers and Their Parallelization*, SIAM, 2003.

[16] G. W. GEORG HAGER, *Introduction to High Performance Computing for Scientists and Engineers*, Chapman & Hall, 2010.

[17] 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.

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

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

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

[21] M. LIEBMANN, *Efficient PDE solvers on modern hardware with applications in medical and technical sciences*, phd in natural sciences,, Institute of Mathematics and Scientific Computing – Karl Franzens University Graz, 2009.

[22] T. RAUBER AND G. RÜNGER, *Parallel Programming for Multicore and Cluster Systems*, Springer, 1st edition ed., 2010. ISBN 978-3-642-04817-3.

[23] 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/.

[24] F. VÁZQUEZ, J. J. FERNÁNDEZ, AND E. M. GARZÓN, *A new approach for sparse matrix vector product on nvidia gpus*, Concurrency and Computation: Practice and Experience, 23 (2011), pp. 815–826.

[25] F. VÁZQUEZ, J. J. FERNÁNDEZ, AND E. M. GARZÓN, *Automatic tuning of the sparse matrix vector product on gpus based on the ellr-t approach*, Parallel Computing, 38 (2012), pp. 408 – 420.