Computer Science in Ocean and Climate Research

Lecture 5: Increasing Computational Performance

Prof. Dr. Thomas Slawig

CAU Kiel
Dep. of Computer Science

Summer 2020

Contents

Increasing model performance

- In ocean and climate science, we have computationally expensive models and computer codes.
- We thus need to use all options to make the (our) models faster.
- This is a main task for computers scientists working in this research area.
- How can we make a model run faster???

Contents

High Performance Computing (HPC): Hardware

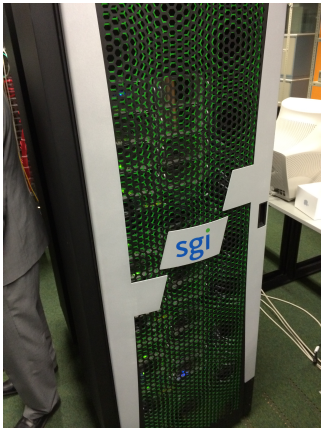
Single work station, laptop

- faster nowadays
- but still too slow for big 3-D problems

High Performance Computing (HPC): Hardware

HP Computers of single groups with restricted access

- fast
- easy access
- expensive to buy
- ... and to maintain
- personnel for maintainance needed
- lifetime $\approx 5 - 7$ years



High Performance Computing (HPC): Hardware

Mid-size Clusters (as available e.g. at CAU)

- several nodes with 16, 24, 32 ... cores
- e.g.: easy to get 128 processes
- purchase price ...
- ... and maintainance cost shared by many groups
- maintained and ...
- installation of additional software by university personnel
- access via batch system
- small job (fast access) vs. big jobs (optionally longer waiting time)

NEC HPC-Linux-Cluster

✗ Studierende | ✓ Beschäftigte | ✓ Einrichtungen

Der NEC HPC-Linux-Cluster des Rechenzentrums ist Teil eines hybriden NEC Hc
einem ⇒ NEC SX-ACE Vektorrechnersystem mit einer theoretischen Peakperfo
Linux-Cluster mit insgesamt 6192 Cores und einer theoretischen Peakperform
über die gleichen Vorrechner zu erreichen, haben Zugriff auf ein gemeinsames **5I**
Batchberechnungen werden über ein **Batchsystem (NQSII)** verwaltet.

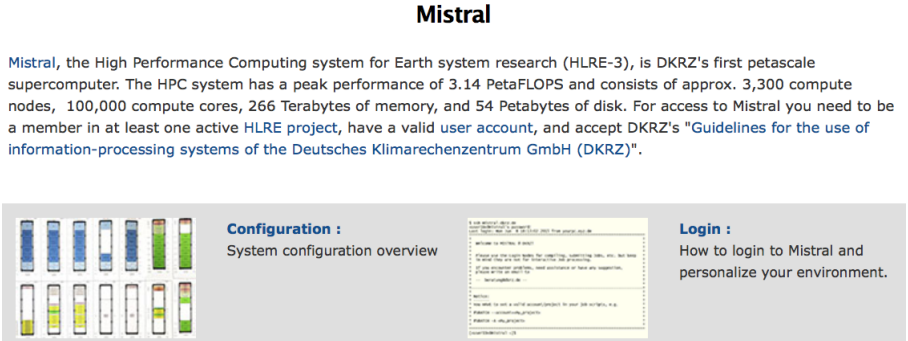
High Performance Computing (HPC): Hardware

HPC centers:

- DKRZ (German Climate Computing Center) Hamburg, Jülich, HLRN Göttingen & Berlin, Stuttgart, München
- Necessary to write application for project

Cloud services:

- e.g., AWS Amazon Web Services
- Alternative to own single-group HPC ?



Contents

Parallelization issues

- Parallelization on the level of the model and algorithm:
  - in space
  - in time
  - in sub-models: structural (operator splitting)
  - for ensembles of parameters, initial values ...
- Realization on the computer
  - using one process with several threads, which share their memory
  - $\rightsquigarrow$ technology (e.g.): `OpenMP` (Open Message Passing)
  - access on common data possible (has to be taken into account)
  - using several processes which have separated, distributed memory
  - $\rightsquigarrow$ technology (e.g.): `MPI` (Message Passing Interface)
  - communication of the common data necessary

Contents

Better numerical methods or algorithms

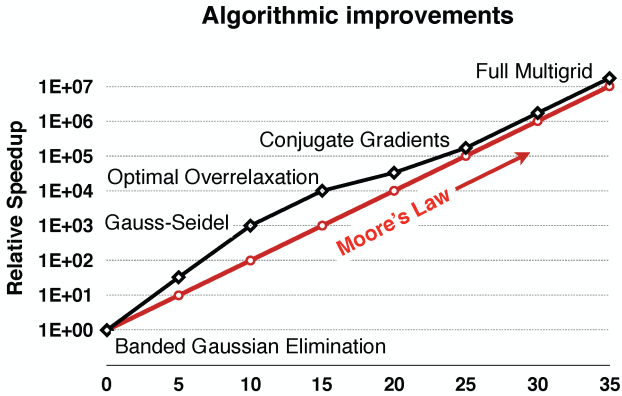


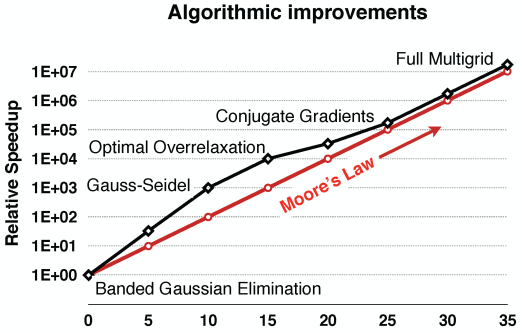
Figure: Moore's law for algorithms to solve the 3D Poisson equation (black) plotted with Moore's law for transistor density (red), each showing 24 doublings (factor of approximately 16 million) in performance over an equivalent period. Source: Rüde et al.: *Research and Education in Computational Science and Engineering* arXiv:1610.02608.

Better numerical methods or algorithms

Typical example: Linear solver

$$Ax = b$$

- From standard Gaussian
- to Banded Gaussian
- iterative solvers (Gauss-Seidel, Overrelaxation)
- Conjugate Gradients
- to Multigrid.



Source: Rüde et al.: *Research and Education in Computational Science and Engineering* arXiv:1610.02608.

Contents

Compiled vs. scripting languages

- Usually compiled languages (C, Fortran) are faster.
- Compiler of hardware vendor (e.g., Intel® `ifort`) might be more efficient
- ... than general compiler (as GNU).
- True for own implementations.
- But: Interpreted languages like Python, Matlab® use built-in functions, (e.g. for linear solvers) that are very efficient.
- Examples:
  - Matlab/octave matrix functions, e.g., linear solver:
    ```
    x = A \ b;
    ```
  - Python `numpy`, `scipy` modules, linear solver:
    ```
    x = numpy.linalg.solve(A, b)
    ```
  - R: similar functions exist

Intelligent use of language features

- Especially in Fortran: (See predator-prey model)

```
real(8) :: x(:), y(:), A(:,:)
...
x = x*y
y = MATMUL(A,x)
```

  rather than do loops.
- (Hopefully) Optimized versions for the actual hardware.
- Parallelizable ?

Contents

Using appropriate data types

- 3-D climate models: spatial discretization often leads to sparse matrices
- Sparse data structures: *csr(compressed sparse row)* and other formats
- Numerical libraries:
  - LAPACK for band matrices
  - Intel® Math Kernel Library (MKL)
- Also available in languages like Python, Matlab®, Octave:
  - Matlab®, Octave sparse matrix functions

```
A = ...;
A = sparse(A);
x = A \ b;
```

  - Python `numpy`, `scipy` modules

```
import scipy.sparse.linalg
x = scipy.sparse.linalg.spsolve(A, b, ...);
```

Contents

Use of numerical libraries

- Often: expensive computations are matrix-vector or vector-vector operations:

$$\begin{aligned} & y = Ax \\ \text{solve} \quad & Ax = b \\ & z = ax + y \end{aligned}$$

  for $a, b \in \mathbb{R}, x, y, z, b \in \mathbb{R}^n, A \in \mathbb{R}^{n \times n}$.

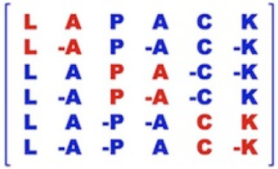
- BLAS: Basic Linear Algebra Subprograms:

Presentation:

The BLAS (Basic Linear Algebra Subprograms) are routines that provide standard building blocks for performing basic vector and matrix operations. The Level 1 BLAS perform scalar, vector and vector-vector operations, the Level 2 BLAS perform matrix-vector operations, and the Level 3 BLAS perform matrix-matrix operations. Because the BLAS are efficient, portable, and widely available, they are commonly used in the development of high quality linear algebra software, LAPACK for example.

Use of numerical libraries

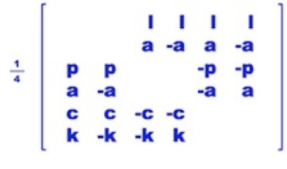
- LAPACK: Linear Algebra PACKage:



Version 3.8.0

Browse the LAPACK User Forum
Contact the LAPACK team

Get the latest LAPACK News



- precompiled versions available, optimized for actual hardware
- ScaLAPACK (distributed-memory implementation)
- PLASMA (Parallel Linear Algebra for Scalable Multi-core Architectures)
- MAGMA (Matrix Algebra on GPU and Multicore Architectures)
- On Intel® machines/processors: Intel® Math Kernel Library.

Sometimes: problems in existing climate models

- Lack of clear interfaces, necessary to replace existing subroutines by maybe more efficient libraries:

```
!-----------------------------------------------------------------------
!     now invert
!-----------------------------------------------------------------------
        do 500 k=km,1,-1
            if (k .le. kmax) then
            g     = c1/(b(k)-c(k)*e(k))
            e(k-1) = a(k)*g
            f(k-1) = (d(k)+c(k)*f(k))*g
            endif
        500   continue
```

Contents

Using reduced precision: IEEE standards

Representation of floating point numbers: $m = (1.m_1 \dots m_k)_2, e = (e_0 \dots e_{\ell-1})_2 - |e_{min}|$.

$$z = (-1)^{\nu} \left(1 + \sum_{i=1}^{k} m_i 2^{-i}\right) 2^{e}, \quad \nu, m_i \in \{0,1\}$$

$$e = \sum_{i=0}^{\ell-1} e_i 2^{i} - (2^{\ell-1} - 1), \quad e_i \in \{0,1\}, i = 0, \dots, \ell - 1$$

`single`: (32 bit)

```
V EEEEEEEE MMMMMMMMMMMMMMMMMMMMMMM
0 1       8 9                     31
```

`double`: (64 bit)

```
V EEEEEEEEEEE MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
0 1          11 12                                                63
```

Using reduced precision: IEEE standards

`single`: (32 bit)

```
V EEEEEEEE MMMMMMMMMMMMMMMMMMMMMMM
0 1      8 9                     31
```

Beispiele:

`0 11111111 00000000000000000000000`	$=$	$+\infty$
`1 11111111 00000000000000000000000`	$=$	$-\infty$
`0 11111111 00000100000000000000000`	$=$	`NaN`
`1 11111111 00100010001001010101010`	$=$	`NaN`
`0 10000000 00000000000000000000000`	$=$	$+1.0 * 2^{128-127} = 2$
`0 10000001 10100000000000000000000`	$=$	$+1.101 * 2^{129-127} = 6.5$
`1 10000001 10100000000000000000000`	$=$	$-1.101 * 2^{129-127} = -6.5$
`0 00000001 00000000000000000000000`	$=$	$+1.0 * 2^{1-127} = 2^{-126} = x_{\min}$
`0 00000000 10000000000000000000000`	$=$	$+0.1 * 2^{-126} = 2^{-127}$
`0 00000000 00000000000000000000001`	$=$	$+0.0\ldots01 * 2^{-126} = 2^{-149}$
	$=$	kleinste darstellbare Zahl

IEEE standards: characteristic numbers

- Smallest number on the computer:

$$\text{single} : \approx 10^{-45}, \quad \text{double} : \approx 10^{-324}$$

- Biggest number on the computer:

$$\text{single} : \approx 10^{37}, \quad \text{double} : \approx 10^{308}$$

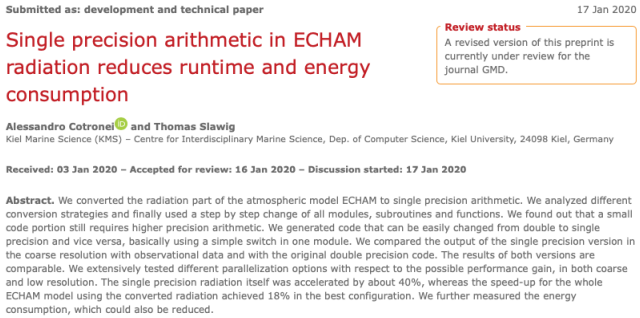
- Machine precision *epsilon* $:= \min_{x \in \mathbb{M}} \{1 + x >_{\mathbb{M}} 1\}$ in machine numbers $\mathbb{M}$, where also the comparison $>_{\mathbb{M}}$ is done in machine arithmetic.

$$\text{single} : \approx 10^{-8}, \quad \text{double} : \approx 10^{-16}.$$

- Usually this accuracy is not needed everywhere (!).
- There are also now computations which use half precision only ($\rightsquigarrow$ Machine Learning).

Single Precision might save time

- Results differ slightly $\rightsquigarrow$ models have to be calibrated again.
- Motivated by: Vana et al.: *Single Precision in Weather Forecasting: An Evaluation with the IFS*, Monthly Weather Review 2017.
- Radiation part of an atmospheric model transformed to single precicion $\rightsquigarrow -40\%$ runtime, -15% energy.



Submitted as: development and technical paper 17 Jan 2020

Review status
A revised version of this preprint is currently under review for the journal GMD.

Single precision arithmetic in ECHAM radiation reduces runtime and energy consumption

Alessandro Cotronei and Thomas Slawig
Kiel Marine Science (KMS) – Centre for Interdisciplinary Marine Science, Dep. of Computer Science, Kiel University, 24098 Kiel, Germany

Received: 03 Jan 2020 – Accepted for review: 16 Jan 2020 – Discussion started: 17 Jan 2020

Abstract. We converted the radiation part of the atmospheric model ECHAM to single precision arithmetic. We analyzed different conversion strategies and finally used a step by step change of all modules, subroutines and functions. We found out that a small code portion still requires higher precision arithmetic. We generated code that can be easily changed from double to single precision and vice versa, basically using a simple switch in one module. We compared the output of the single precision version in the coarse resolution with observational data and with the original double precision code. The results of both versions are comparable. We extensively tested different parallelization options with respect to the possible performance gain, in both coarse and low resolution. The single precision radiation itself was accelerated by about 40%, whereas the speed-up for the whole ECHAM model using the converted radiation achieved 18% in the best configuration. We further measured the energy consumption, which could also be reduced.

What is important

- Performance is a crucial point in ocean and climate simulation.
- There are different options for increasing computational performance:
  - Hardware
  - Choice of programming language
  - Compilers optimized for the used hardware
  - Efficient libraries, compiled for used hardware
  - Parallel algorithms
  - State-of-the-art mathematical methods
  - Reduced arithmetic precision
- Software Design: Modular structure enables flexibility to include new and faster algorithms and code parts easily.
- In operational climate models, this is not necessarily the case.