Řešení lineárních systémů a paralelizace v CFD

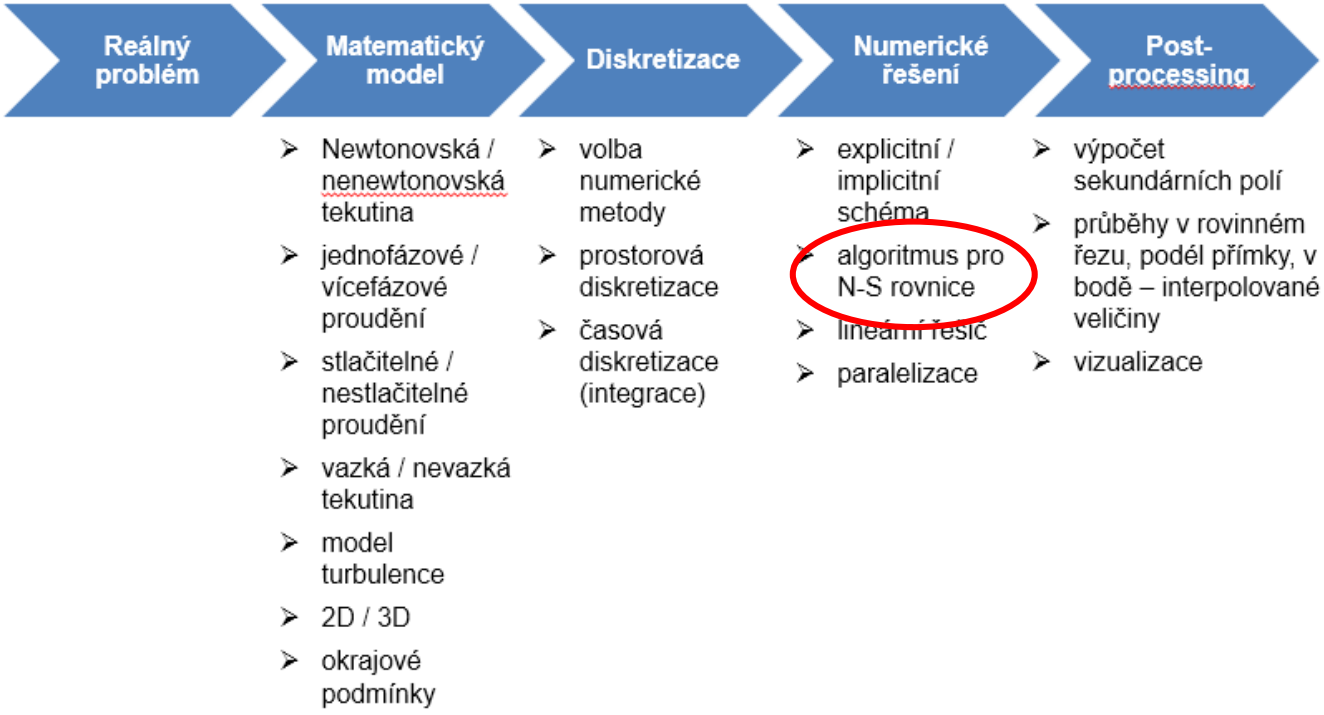
Petr Šidlof

Česko-anglický slovník termínů v CFD

Česky	Anglicky
monolitický / segregovaný řešič	monolithic / segregated solver
paměťové nároky	memory requirements
řídká matice	sparse matrix
přímý řešič	direct solver
iterativní řešič	iterative solver
Krylovovské metody	Krylov subspace methods
metoda konjugovaných gradientů	conjugate gradient method
víceúrovňové metody	multigrid methods
paralelní škálování	parallel scaling
výpočetní výkon	computational power
sdílená paměť	shared memory
distribuovaná paměť	distributed memory

Česky	Anglicky
dekompozice oblasti	domain decomposition
grafická karta	graphics processing unit (GPU)
počítačový cluster	computer cluster
počítač se sdílenou pamětí	shared-memory computer
vícejádrový systém	multicore system

Typy CFD řešičů založených na metodě konečných objemů (1)



1. Pressure-based řešiče

1.1 Segregované

1.2 Monolitické (coupled)

2. Density-based řešiče

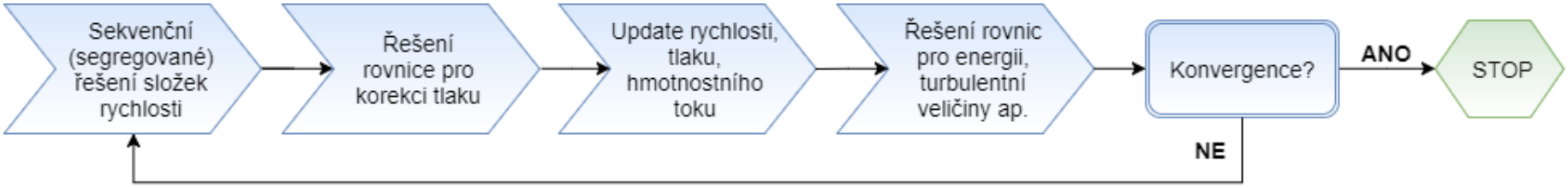
Typy CFD řešičů založených na metodě konečných objemů (2)

1. Pressure-based řešiče

- původně vyvinuté pro nízkorychlostní nestlačitelné proudění
- rychlostní pole řešením rovnic pro hybnost
- tlakové pole řešením Poissonovy tlakové rovnice (divergence rovnice pro hybnost, rovnice kontinuity)

1.1 Segregované řešiče – algoritmy PISO, SIMPLE

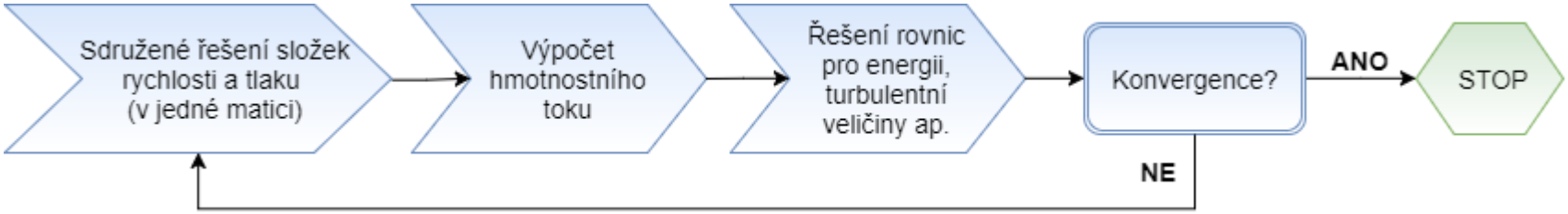
- segregované (sekvenční) řešení jednotlivých složek rychlosti, tlaku, energie, turbulentních veličin
- nízké nároky na paměť, rychlé řešení
- pomalá konvergence – iterativní proces



Typy CFD řešičů založených na metodě konečných objemů (3)

1.2 Monolitické (coupled) řešiče

- řešení všech složek rychlosti a tlaku v jediné matici
- větší a hůře podmíněná matice – řešení jedné iterace je výrazně výpočetně náročnější
- silné sdružení rychlosti a tlaku – robustnější
- oproti segregovanému přístupu menší počet iterací, ale každá z nich je náročnější

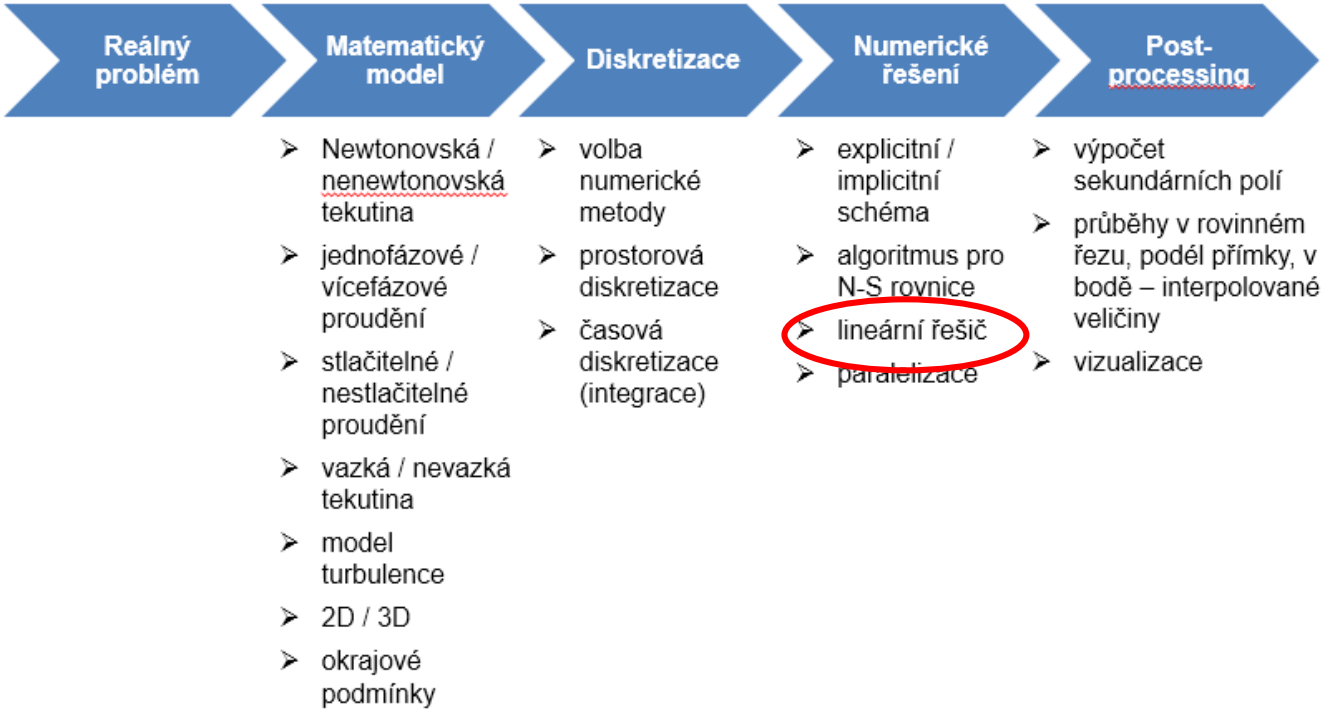


Typy CFD řešičů založených na metodě konečných objemů (4)

2. Density-based řešiče

- původně vyvinuté pro vysokorychlostní, stlačitelné proudění
- rychlostní pole řešením rovnic pro hybnost
- pole hustoty řešením rovnice kontinuity
- tlakové pole ze stavové rovnice

Řešení lineárních systémů



- s výjimkou plně explicitních řešičů CFD výpočet vždy vede na numerické řešení soustavy lineárních rovnic
- ve většině případů zabírá řešení lineárního systému největší část zdrojů – CPU času a RAM

Lineární systémy v CFD

- v případě CFD problémů (MKD, MKO a částečně i MKP) diskrétní rovnice pro daný uzel nebo prvek závisí pouze na hodnotách sousedních uzlů / prvků
- matice jsou řídké (sparse) – počet nenulových prvků je řádu O(n), nikoliv O(n^2)

koeficient – příspěvky z časové derivace, konv. a dif. členu, lin. část zdroj. členu

koeficient – odpovídající příspěvky sousedních elementů

$$a_P \phi_P^n + \sum_N a_N \phi_N^n = R_P$$

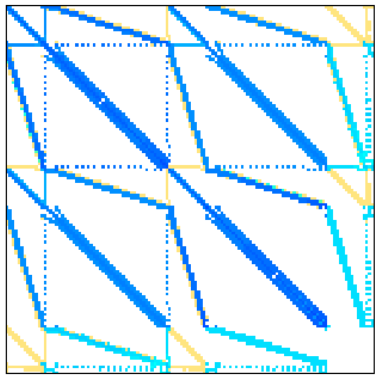
členy, které lze vypočítat z minulého časového kroku

hodnota v daném prvku

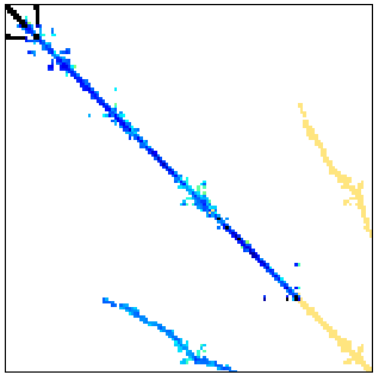
hodnoty v sousedních prvcích prvku

SuiteSparse Matrix Collection
Formerly the University of Florida Sparse Matrix Collection

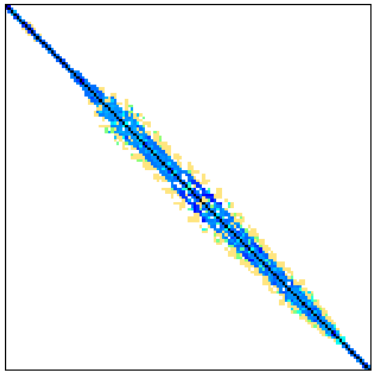
https://sparse.tamu.edu



2D FEM – Navier-Stokes, square with inlet/outlet



3D CFD – Charleston harbor



CFD – symmetric presure matrix

Metody pro řešení (řídkých) soustav lineárních rovnic v CFD (1)

1. Přímé řešiče

- obsahují explicitní faktorizaci matice A (například LU faktorizace .. A = L U)
- spolehlivé, umožňují v **konečném** počtu kroků vypočítat **přesné** řešení lineárního systému
- pro velké systémy rovnic neúnosně výpočetně a paměťové náročné

Příklady

- (Gaussova eliminace) – husté matice, $O(n^3)$
- TDMA – tridiagonal matrix algorithm: MKD, O(n)
- LU faktorizace
- UMFPACK – multifrontal LU factorization (řídké matice, Matlab)

Metody pro řešení (řídkých) soustav lineárních rovnic v CFD (2)

2. Iterační řešiče

- výpočet iterační sekvence, která postupně konverguje k přesnému řešení lineárního systému
- výrazně nižší paměťová náročnost - pro velké systémy lineárních rovnic obvykle jediná volba

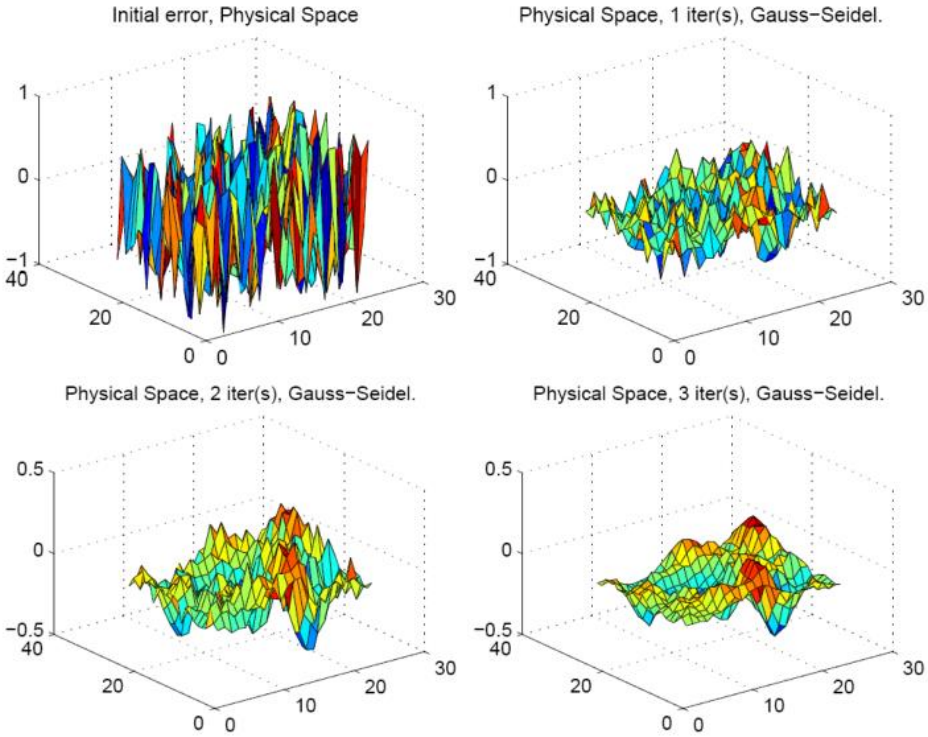
Příklady

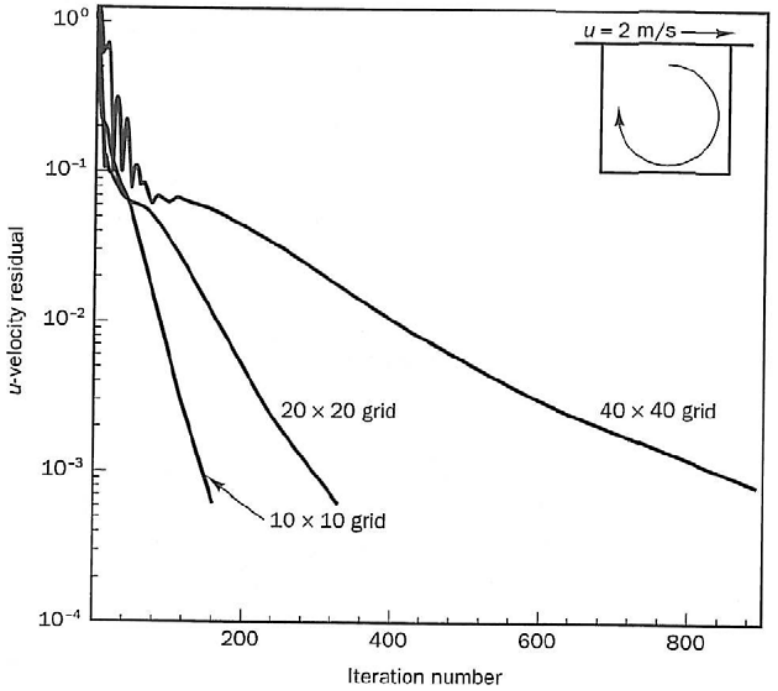
- Jacobiho metoda – prakticky se nevyužívá, pedagogické účely
- Gauss-Seidelova metoda
- Relaxační metody – zrychlení konvergence G-S pomocí relaxačního parametru
- Krylovovské metody (Krylov subspace methods)
  - Konjugované gradienty (Conjugate Gradient Method – CG)
  - Biconjugate Gradient Stabilized – BiCGSTAB
  - Generalized Minimum Residual – GMRES
  - Minimal Residual – MINRES
- Víceúrovňové metody (multigrid methods)

Vlastnosti klasických iteračních metod

Rychlost konvergence iteračních metod (např. G-S) **klesá** se zjemňující se sítí

Problém: G-S vyhlazuje rychle vysokofrekvenční chyby, pomalé tlumení nízkofrekvenčních chyb



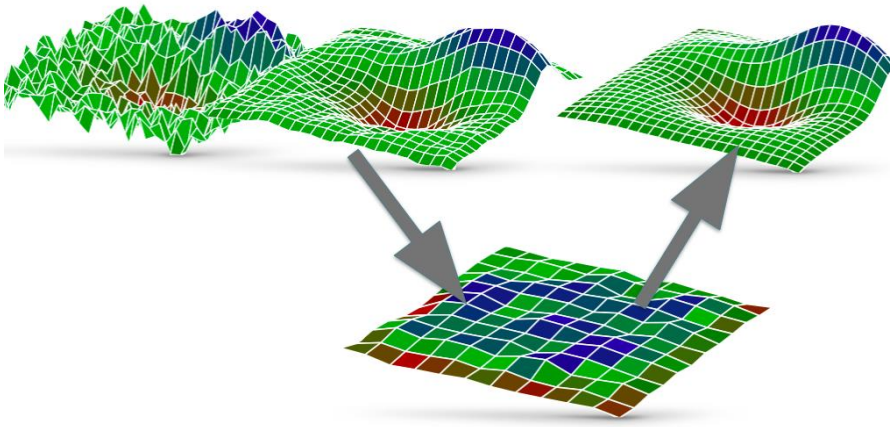


Versteeg & Malalasekera, An Introduction to Computational Fluid Dynamics, Pearson / Prentice Hall 2007

MULTIGRIDY

NTNU Trondheim, Engineering CFD course

Víceúrovňové metody (multigrid methods)



PRACE summer school of HPC, University of Illinois

Motivace

zrychlení konvergence iteračních metod
zejména pro velké problémy

Základní myšlenka

- běžné iterační metody vyhlazují rychle vysokofrekvenční chyby, pomalé tlumení nízkofrekvenčních chyb
- použití několika různě velikých sítí (matic) – hrubá pro rychlé odstranění nízkofrekvenčních chyb, jemná pro detailní řešení a odstranění vysokofrekvenčních chyb

Postup – základní V-cyklus

1. **pre-smoothing:** několik kroků iterační metody (např. Gauss-Seidel) na jemné síti pro vyhlazení vysokofrekvenčních chyb
2. **restrikce:** projekce na hrubou síť
3. **řešení na hrubé síti:** několik iterací pro vyhlazení nízkofrekvenčních chyb
4. **interpolace (prolongace):** projekce zpět na jemnou síť
5. **post-smoothing:** několik iterací pro finální vyhlazení vysokofrekvenčních chyb

Víceúrovňové metody (multigrid methods)

Varianty

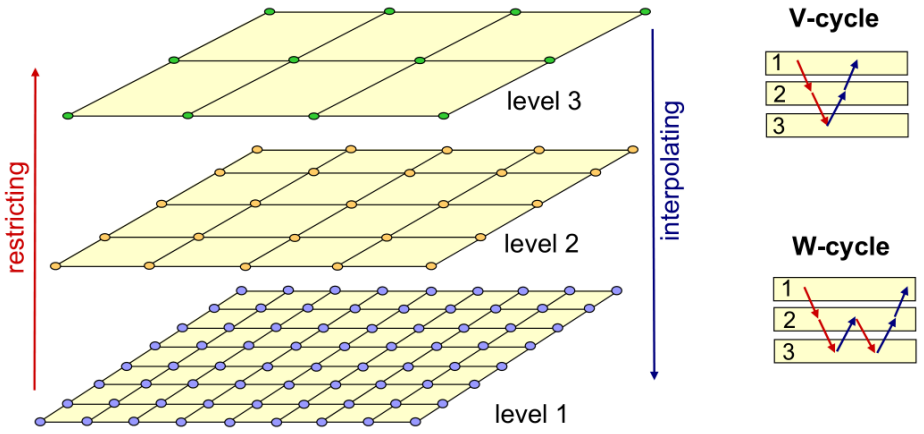
- Algebraické multigridy (AMG) – na úrovni matice. Univerzální, bez znalosti geometrie a bez nutnosti opakované diskretizace. Lze použít jako „blackbox“ pro libovolný problém a síť
- Geometrické multigridy (GMG) – na úrovni sítě. Lze dobře využít a-priori informací o sousedních elementech – zejména vhodné pro strukturované sítě. V těchto případech je GMG výrazně efektivnější (CPU i RAM)

Vlastnosti

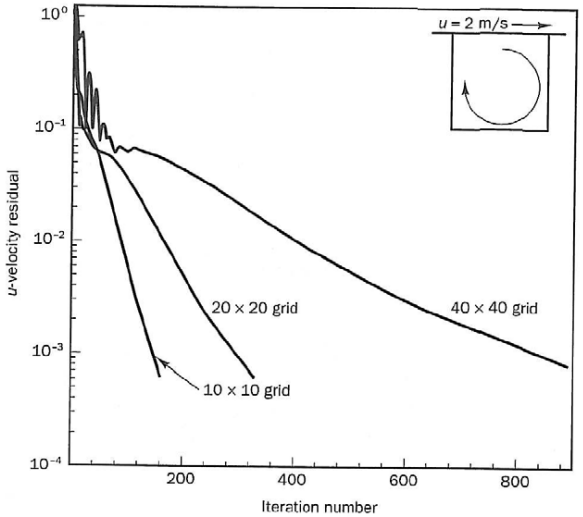
- pro mnoho typů problémů (zejména velké úlohy) se jedná o nejefektivnější řešič
- konvergence téměř nezávislá na velikosti problému, škáluje lineárně s počtem uzlů
- algoritmus musí být přizpůsoben na daný typ problému (zejména GMG)
- dobré pro eliptické problémy, pro parabolické a hyperbolické pomalejší konvergence

Technické detaily

- základní iterační řešič – obvykle Gauss-Seidelova metoda nebo neúplný LU rozklad (ILU)
- na nejhrubší úrovni lze použít i přímý řešič
- V-cykly, W-cykly, F-cykly



Konvergence iteračních metod



Rekapitulace

- přímé řešiče → přesný výsledek v konečném počtu kroků
- iterační řešiče .. postupné přibližování k přesnému výsledku (který ale neznáme)

Důležitá otázka: kdy zastavit iterace?

Možnosti

➢ **Rozdíl dvou následujících iterací**
Problém: rozdíl je malý

a) když metoda konverguje k přesnému řešení

b) když je konvergence příliš pomalá

➢ **Reziduál**

- kvantitativní míra, jak dobře výsledek dané iterace splňuje diskrétní formu rovnice
- iterační řešič zastaví ve chvíli, kdy reziduál klesne pod předem definovanou hodnotu

Reziduál

1. Diskrétní rovnice pro element P: $a_P \, \Phi_P^n = -\sum_N a_N \Phi_N^n + R_P$

   - finální zkonvergované řešení splňuje tuto rovnici přesně
   - iterace číslo k: mezi pravou a levou stranou rovnice je rozdíl ..

   .. lokální reziduál $F_P^{\Phi,k} = \left| -\sum_N a_N \Phi_N^n + R_P - a_P \, \Phi_P^n \right|$

2. Indikátor konvergence přes celou výpočetní oblast

   .. globální reziduál $G^{\Phi,k} = \sum_P F_P^{\Phi,k}$

3. Normalizace (aby reziduál nezávisel na velikosti veličiny Φ)

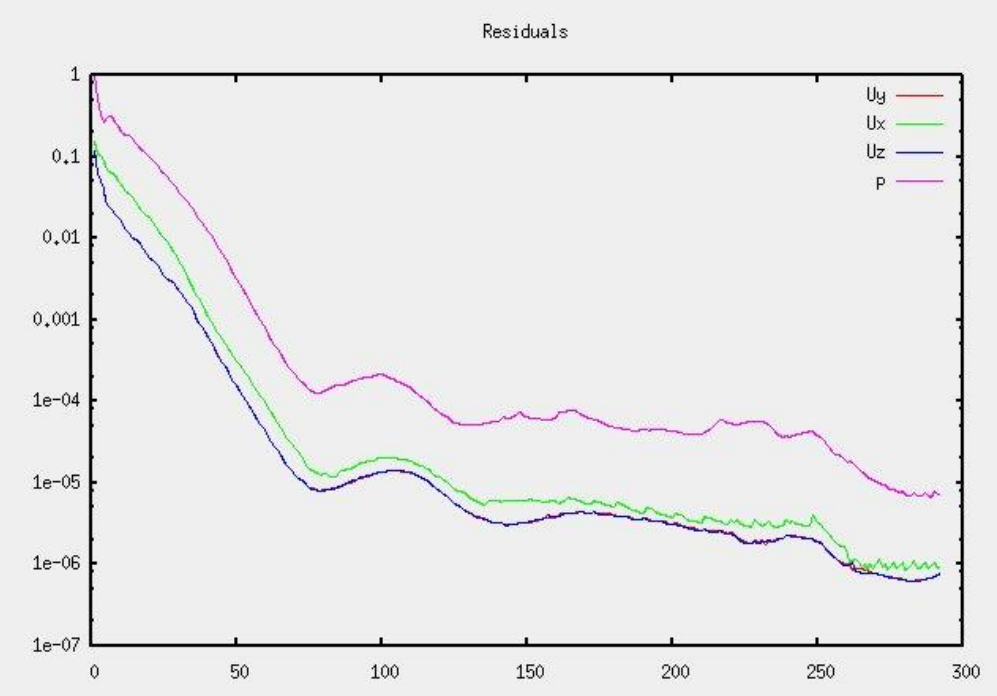
   .. normalizovaný globální reziduál $H^{\Phi,k} = \frac{G^{\Phi,k}}{Z}$

Volba normalizačního faktoru:

- reziduál na dané iteraci k_0: $Z = G^{\Phi,k_0}$
- absolutní hodnota levé strany diskrétní rovnice: $Z = \sum_P |a_P \Phi_P^n|$
- tok veličiny Φ do oblasti: $Z = \left((\rho \, \mathbf{v} \, \Phi)_f \cdot \boldsymbol{n}_f \; S_f \right)_{in}$

Konvergence - závěrečné poznámky

Konvergence metody neznamená, že jsme došli ke správnému výsledku!!



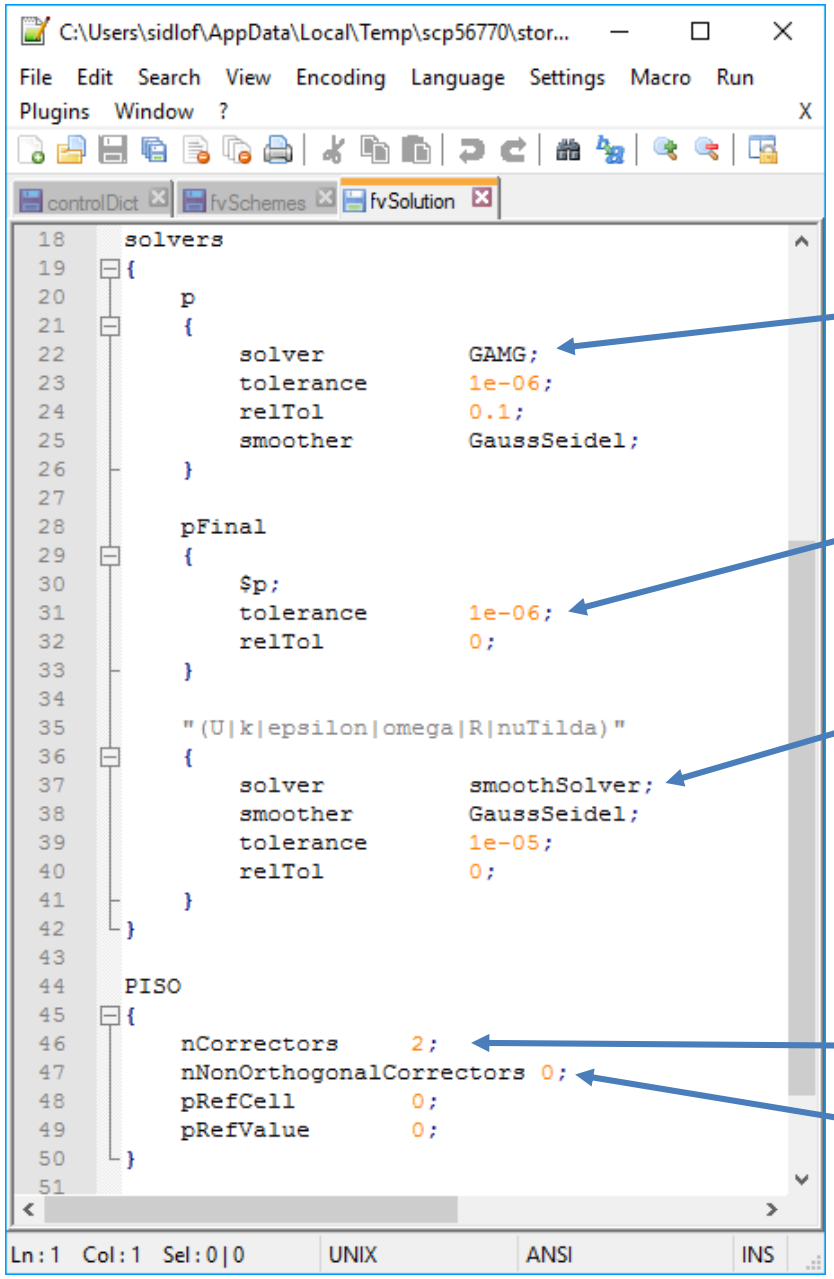
Iterační metoda nekonverguje – řešení dost dobře nemůže být správné

Iterační metoda konverguje – řešení **může** být správné (v souladu s realitou – experimentem)

Příklady konvergujícího, ale nesprávného řešení

- špatný matematický model (rovnice, okrajové podmínky, model turbulence..)
- špatný numerický model (nevhodné diskretizační schémata, výpočetní síť)

Příklad - OpenFOAM 4.1 tutorials: incompressible/pisoFoam/ras/cavity



geometrický-algebraický multigridní řešič pro tlak, absolutní reziduál < 1e-6, relativní reziduál (aktuální / počáteční reziduál) < 0.1

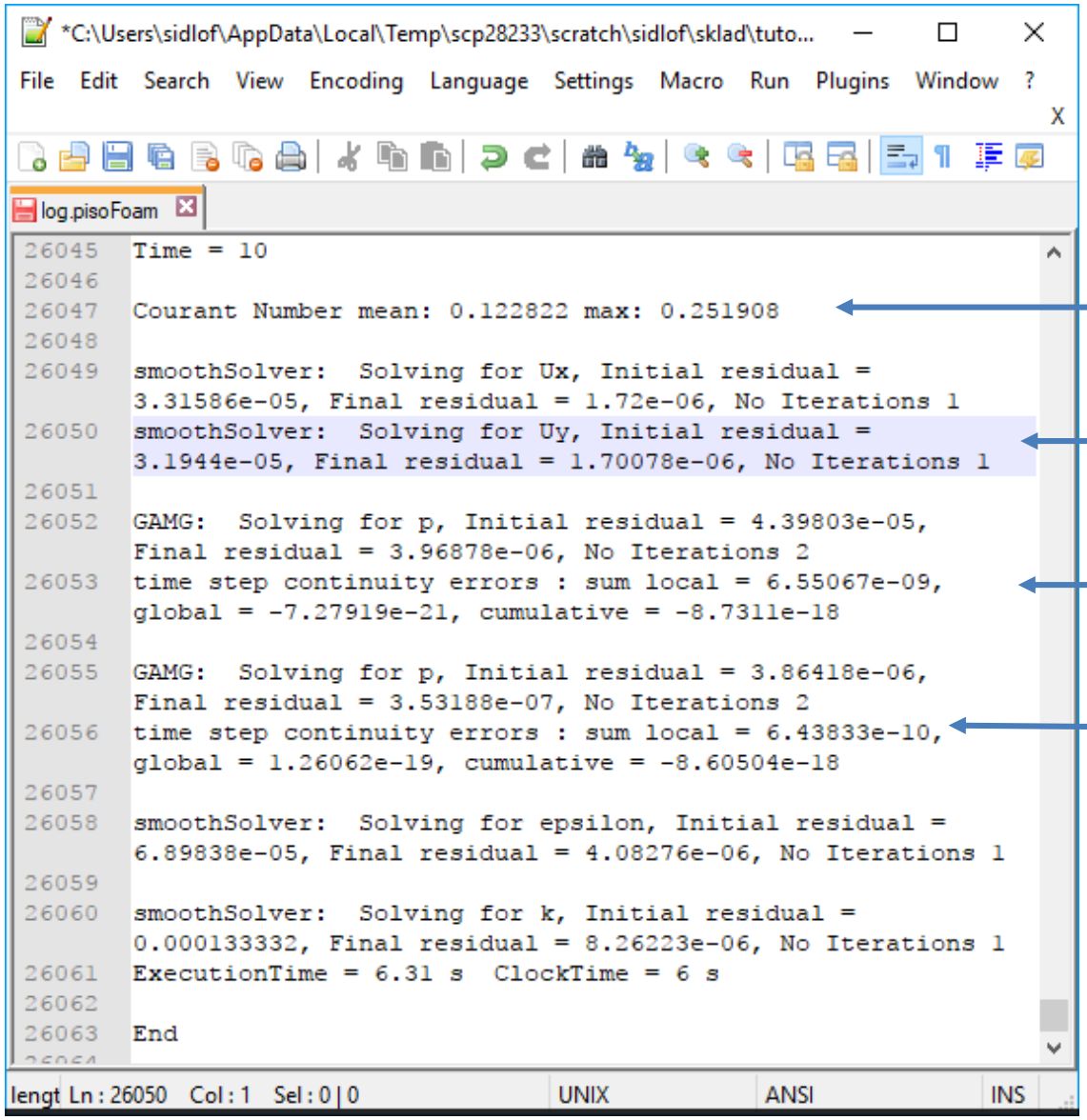
totéž pro finální iteraci tlaku v PISO algoritmu – musí dosáhnout abs. reziduálu < 1e-6

řešič na bázi Gauss-Seidel vyhlazovače pro složky rychlosti

počet iterací ve smyčce PISO

počet iterací pro neortogonální korektor

Příklad - OpenFOAM 4.1 tutorials: incompressible/pisoFoam/ras/cavity



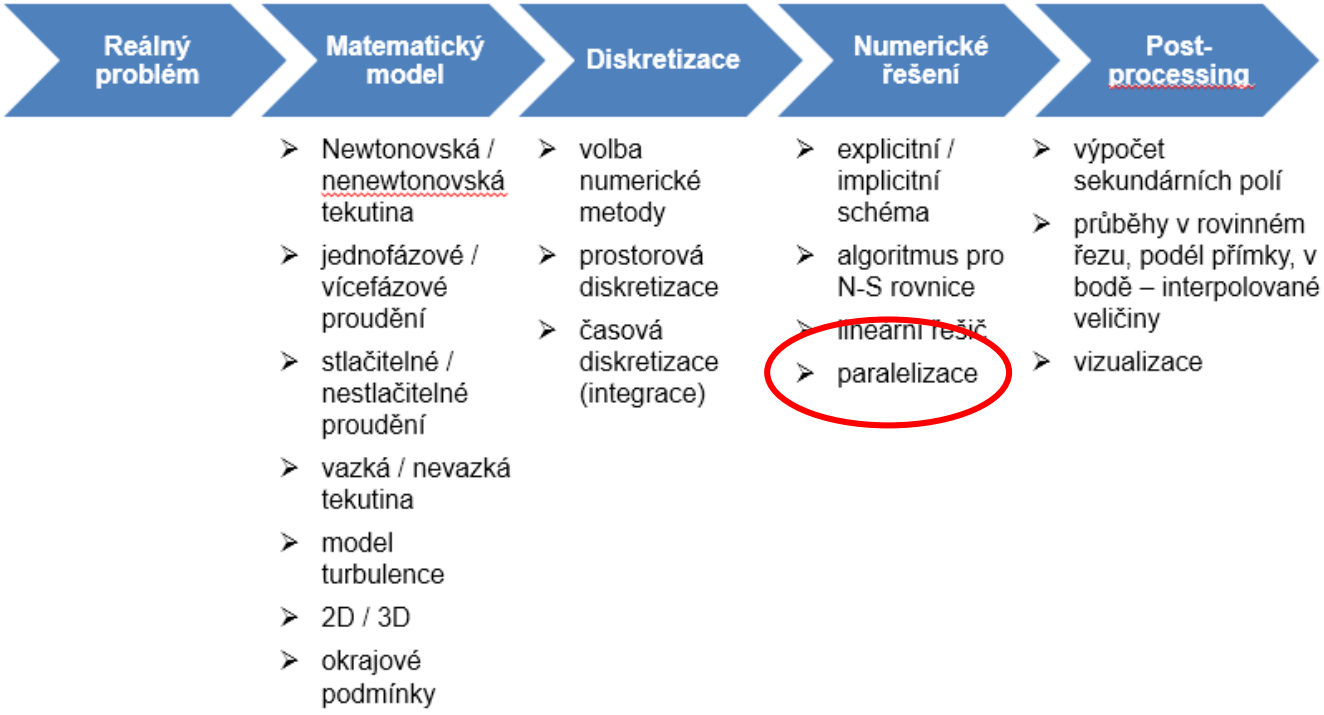
střední a maximální Courantovo číslo

segregovaný prediktor pro složky rychlosti (Gauss-Seidel)

první korektor – relativní tolerance 0.1 (multigrid)

druhý korektor – absolutní reziduál < 1e-6 (multigrid)

Paralelizace CFD výpočtů



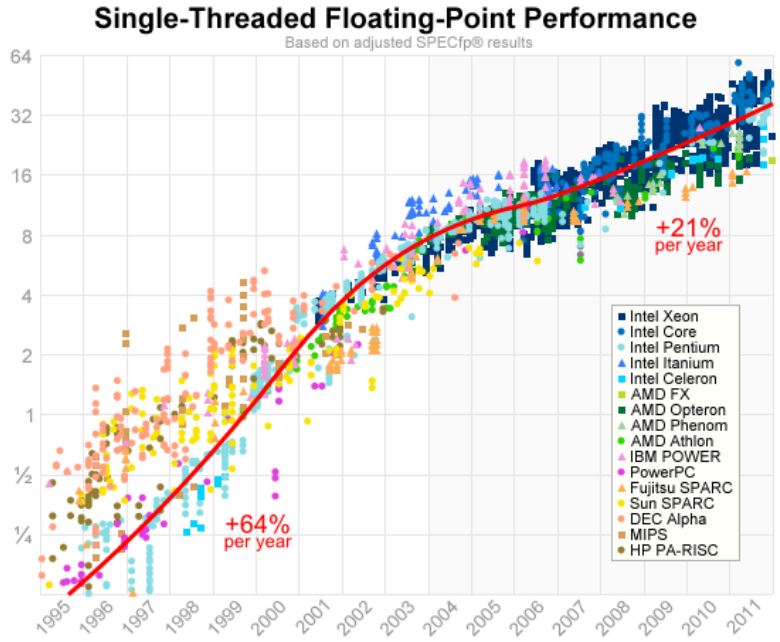
- diskretizace PDR vede na soustavu lineárních rovnic
- přesnější a detailnější řešení – jemnější síť – větší lineární systém
- problémy o velikosti >1M elementů není praktické řešit na jediném CPU

→ **nutnost paralelizace**

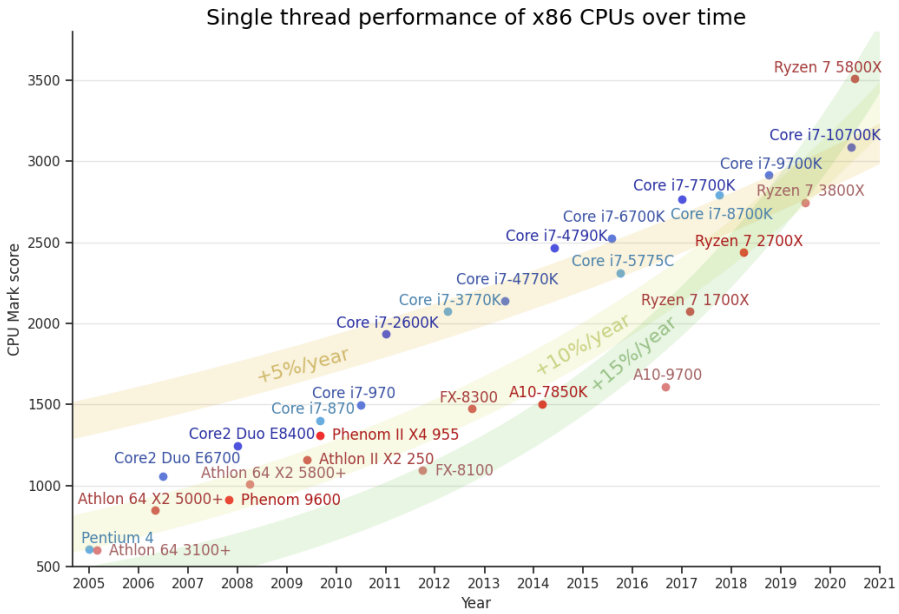
Paralelizace CFD výpočtů - motivace

Mooreův zákon: počet tranzistorů v procesoru se každé 1.5 – 2 roky zdvojnásobí

Nikoliv ovšem sériový výkon procesoru!



http://preshing.com/20120208/a-look-back-at-single-threaded-cpu-performance/



https://mlech26l.github.io/pages/2020/12/17/cpus.html

Poznámka

Na úrovni HW už v dnešní době není limitující rychlost zpracování aritmetických operací (FLOPS), ale rychlost přístupu k paměti

Amdahlův zákon

Ne všechny části algoritmu lze paralelizovat. Zrychlení algoritmu *S* při použití *n* CPU:

$$S(n) = \frac{1}{(1-P) + \frac{P}{n}}$$

P .. paralelizovatelná část algoritmu

n .. počet procesorů

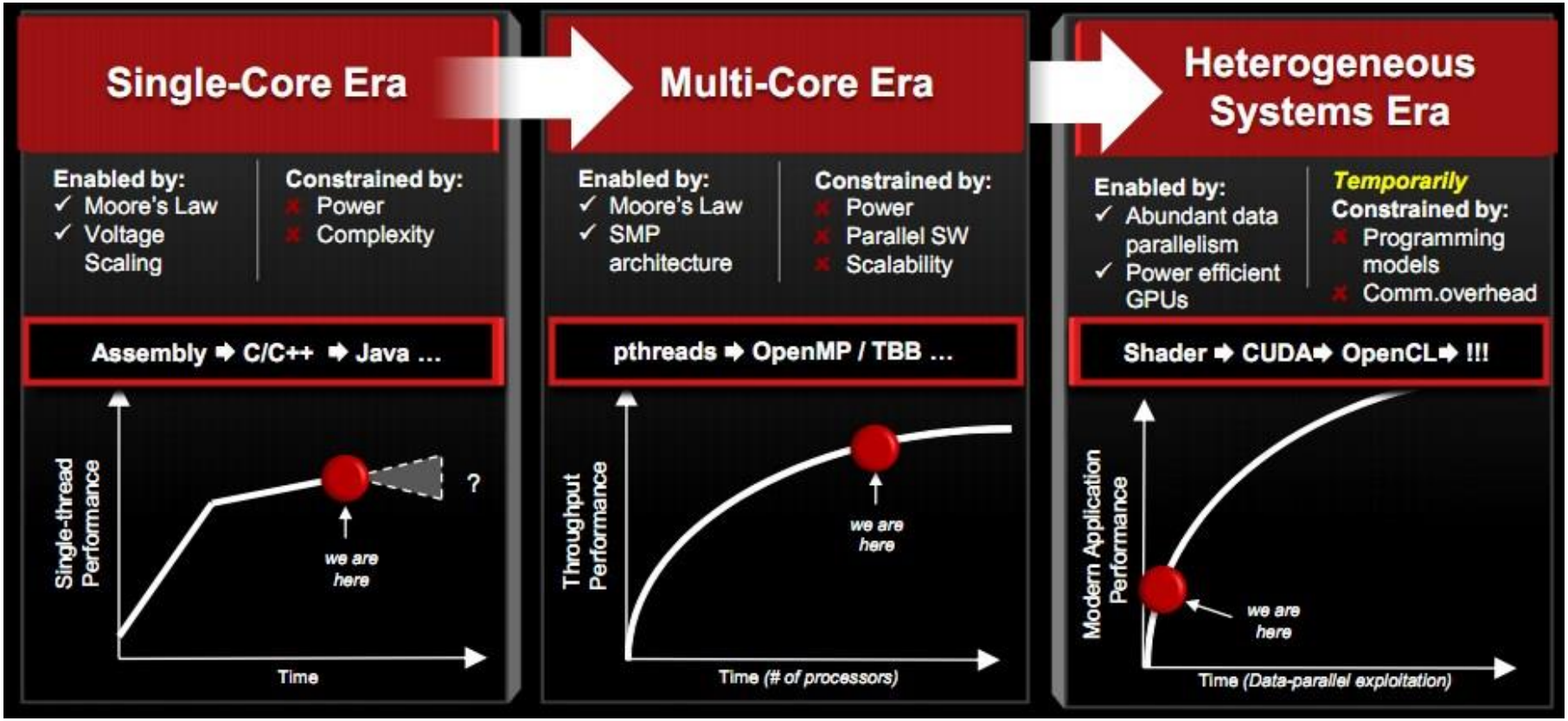
Příklad

- sériová část 30s
- paralelizovatelná část 300s
- sériová část 30s

Počet CPU	Celkový čas	Zrychlení
1	30 + 300 + 30 = 360s	1.0x
2	30 + 150 + 30 = 210s	1.7x
10	30 + 30 + 30 = 90s	4.0x
100	30 + 3 + 30 = 63s	5.7x
∞	30 + 0 + 30 = 60s	6.0x

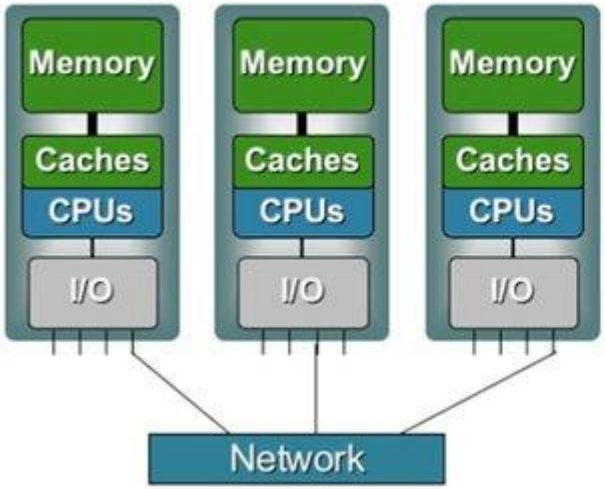
Paralelizace CFD výpočtů - přístupy

- sdílená paměť – OpenMP (nízkoúrovňový paralelismus na úrovni smyček), TBB (Intel)
- distribuovaná paměť – MPI (dekompozice oblasti)
- GPGPU – CUDA (nVidia), OpenCL, …

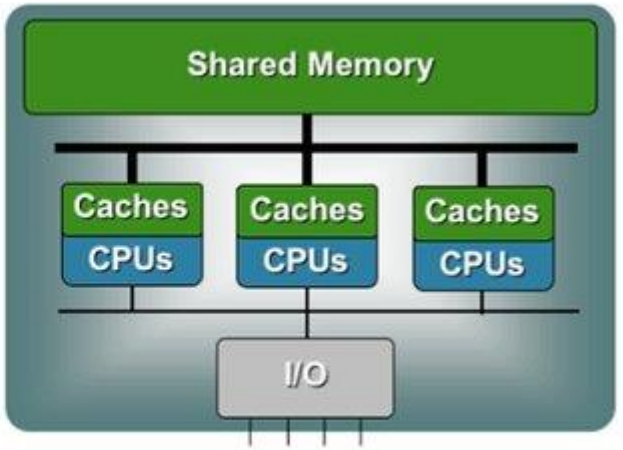


https://www.extremetech.com/computing/116561-the-death-of-cpu-scaling-from-one-core-to-many-and-why-were-still-stuck/3 (2012)

Hardware pro paralelní CFD



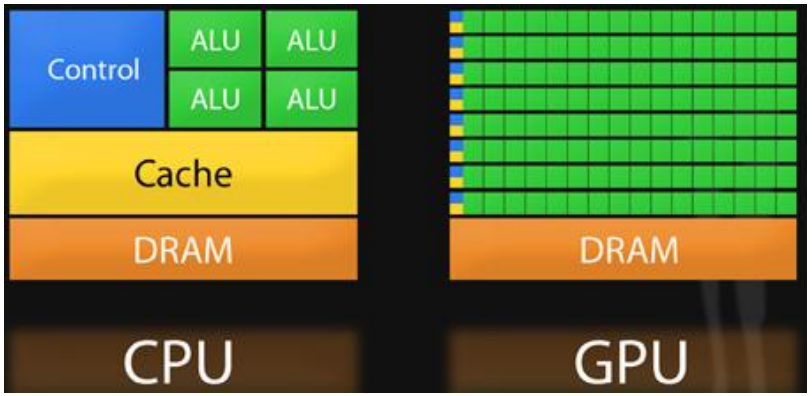
počítačový cluster



superpočítač se sdílenou pamětí (SMP)

GPU (Graphics processing unit)

- CPU – rychlé sekvenční zpracování dat
- GPU – masivně paralelní zpracování dat – velký počet ALU



Nejvýkonnější počítače světa (06/2017)

https://www.top500.org/lists/2017/06/

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
2	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
3	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 Cray Inc.	361,760	19,590.0	25,326.3	2,272
4	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
5	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
78	IT4Innovations National Supercomputing Center, VSB-Technical University of Ostrava Czech Republic	**Salomon** - SGI ICE X, Xeon E5-2680v3 12C 2.5GHz, Infiniband FDR, Intel Xeon Phi 7120P HPE	76,896	1,457.7	2,011.6	1,538

Nejvýkonnější počítače světa (09/2019)

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
2	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
3	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
4	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
5	**Frontera** - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR , Dell EMC Texas Advanced Computing Center/Univ. of Texas United States	448,448	23,516.4	38,745.9	
6	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray/HPE Swiss National Supercomputing Centre (CSCS) Switzerland	387,872	21,230.0	27,154.3	2,384
375	IT4Innovations National Supercomputing Center, VSB-Technical University of Ostrava Czech Republic	**Salomon** - SGI ICE X, Xeon E5-2680v3 12C 2.5GHz, Infiniband FDR, Intel Xeon Phi 7120P HPE	76,896	1,457.7	2,011.6	4,806

Nejvýkonnější počítače světa (12/2023)

https://www.top500.org/lists/2019/09/

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	1,679.82	22,703
2	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	4,742,808	585.34	1,059.33	24,687
3	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Microsoft Azure United States	1,123,200	561.20	846.84	
4	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
5	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	531.51	7,107

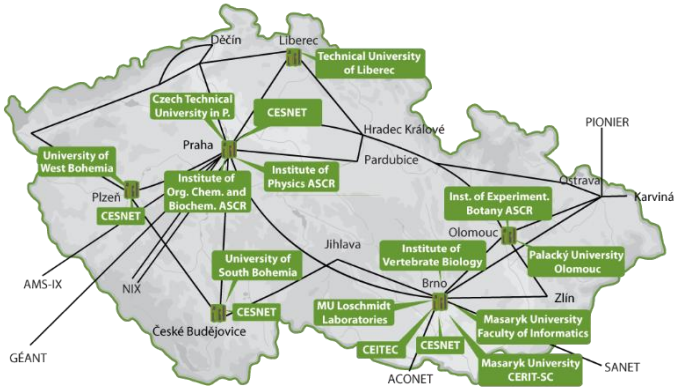
Paralelní počítače dostupné na NTI FM (12/2023)

charon

- vlastní výpočetní cluster NTI FM (pořízen 09/2017), fyzicky v budově A
- 24 uzlů s konfigurací 2x10-core Intel Xeon Silver 4114 2.2GHz, 96GB DDR4 2400 ECC, SSD 480 GB, Intel Omnipath. Frontend s diskovým polem 64TB
- zařazeno do infrastruktury Metacentrum

metacentrum

- virtuální sdružení akademických výpočetních zdrojů ČR – desítky superpočítačů, cca 20000 CPU jader
- centrální instalace aplikací, centrální plánovač úloh pbs-pro
- volně dostupné pro akademické použití

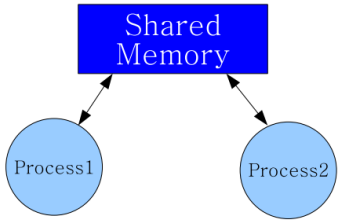


Karolina – IT4I

- 3.8 PFlop/s
- CPU část: 720x 2x AMD 7H12 (64 cores), celkem 92 160 jader
- GPU část: 72x 2x AMD 7763 + 8x Nvidia A100 GPU
- Inifiniband HDR

OpenMP (Open Multiprocessing)

- specifikace direktiv překladače, knihoven a proměnných prostředí pro paralelizaci s využitím sdílené paměti

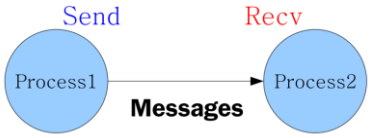


- komunikace pouze přes sdílenou paměť – nelze použít na clusterech (IT4I, metacentrum – charon), pouze na vícejádrové architektuře nebo superpočítači se sdílenou pamětí

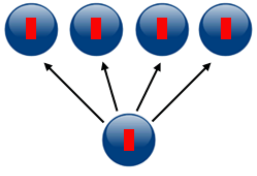
```
1  /*****************************************************
2  *   In this simple example, the master thread forks
   a parallel region.
3  *   All threads in the team obtain their unique
   thread number and print it.
4  *   The master thread only prints the total number
   of threads.  Two OpenMP
5  *   library routines are used to obtain the number
   of threads and each
6  *   thread's number.
7  * AUTHOR: Blaise Barney  5/99
8  *****************************/
9  #include <omp.h>
10 #include <stdio.h>
11 #include <stdlib.h>
12
13 int main (int argc, char *argv[])
14 {
15 int nthreads, tid;
16
17 /* Fork a team of threads giving them their own
   copies of variables */
18 #pragma omp parallel private(nthreads, tid)
19   {
20   /* Obtain thread number */
21   tid = omp_get_thread_num();
22   printf("Hello World from thread = %d\n", tid);
23
24   /* Only master thread does this */
25   if (tid == 0)
26     {
27     nthreads = omp_get_num_threads();
28     printf("Number of threads = %d\n", nthreads);
29     }
30
31   }  /* All threads join master thread and disband */
32 }
```

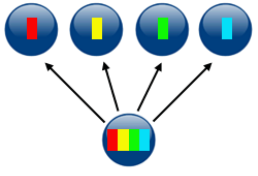
MPI (Message Passing Interface)

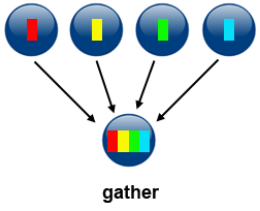
- standard pro komunikaci pomocí posílání zpráv

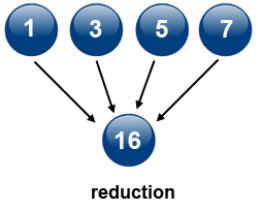


- komunikace pomocí sítě (ethernet, infiniband) – lze použít na počítačových clusterech
- implementace MPI: MPICH, Open MPI, MPT (SGI MPI), …









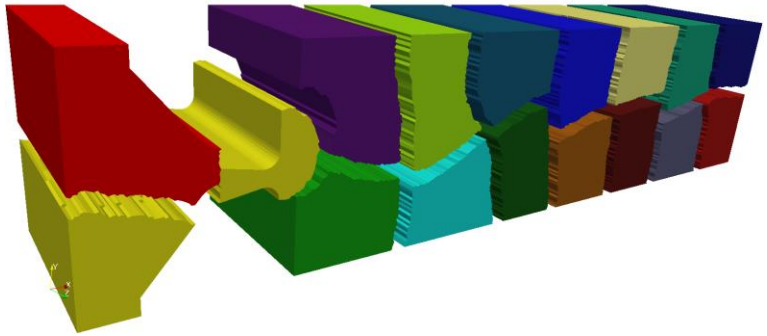

```c
#include "mpi.h"
#include <stdio.h>

int main( int argc, char *argv[] )
{
    int  numprocs, myrank, namelen, i;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    char greeting[MPI_MAX_PROCESSOR_NAME + 80];
    MPI_Status status;

    MPI_Init( &argc, &argv );
    MPI_Comm_size( MPI_COMM_WORLD, &numprocs );
    MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
    MPI_Get_processor_name( processor_name, &
    namelen );

    sprintf( greeting, "Hello, world, from
    process %d of %d on %s",
             myrank, numprocs, processor_name );

    if ( myrank == 0 ) {
        printf( "%s\n", greeting );
        for ( i = 1; i < numprocs; i++ ) {
            MPI_Recv( greeting, sizeof( greeting
             ), MPI_CHAR,
                      i, 1, MPI_COMM_WORLD, &
                      status );
            printf( "%s\n", greeting );
        }
    }
    else {
        MPI_Send( greeting, strlen( greeting ) +
         1, MPI_CHAR,
                  0, 1, MPI_COMM_WORLD );
    }

    MPI_Finalize( );
    return 0;
}
```

MPI vs OpenMP – shrnutí

MPI	OpenMP
distribuovaná paměť	sdílená paměť
počítačové clustery	vícejádrové systémy / SMP superpočítače
MPI zprávy	direktivy překladače
flexibilní, univerzální	jednodušší na programování a debugging

OpenFOAM

- paralelizace: domain decomposition (simple, metis, scotch)
- komunikace: MPI
- počet procesorů: omezeno výpočetními zdroji a škálovatelností algoritmu

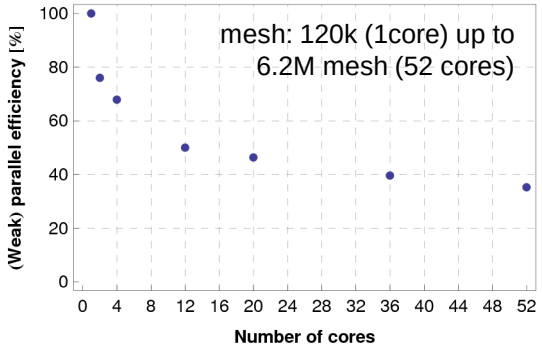


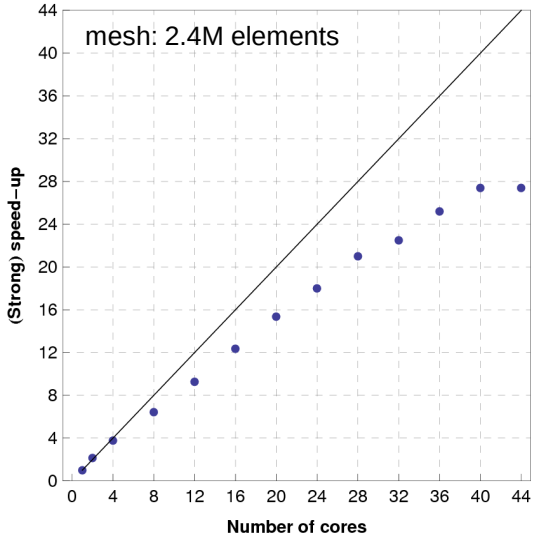
Subdomains	Min/max cells per processor ratio Scotch	Metis	Max / total processor faces Scotch	Metis
2	0.988	0.999	3326 / 3326	3881 / 3881
4	0.981	0.998	6642 / 9859	7929 / 11841
8	0.963	0.942	6378 / 21177	7914 / 25194
16	0.954	0.959	5384 / 34569	6634 / 39620
32	0.953	0.943	4946 / 56953	5600 / 64718

Paralelní škálování

fox: SGI Altix UV 100 (shared-memory) – Centrum intenzivních výpočtů ČVUT

- cc-NUMA architektura
- 12 6-core Intel Xeon Nehalem CPU
- 8GB RAM na jádro (576GB alokovatelných kterýmkoliv jádrem)
- SGI NUMAlink5 interconnect (15GB/s)





ANSYS Fluent

- paralelizace: domain decomposition, možnost akcelerace AMG na GPGPU
- komunikace: MPI
- počet procesorů: omezeno licencí

Závěrečné poznámky

- podstatná výhoda vlastních kódů a GNU/GPL balíků – nulové náklady na licenci SW (u komerčních platba roste s počtem jader)
- pro dobré paralelní škálování je nutné mít vždy na paměti:
  - load balance
  - čas na výpočet vs. čas na komunikaci
  - optimální programovací model a knihovny pro daný hardware