Scientific Computing WS 2018/2019

Lecture 22

Jürgen Fuhrmann

juergen.fuhrmann@wias-berlin.de

Matrix equation

$$
\begin{aligned}
\frac{1}{\tau^n}\left(Mu^n - Mu^{n-1}\right) + Au^\theta = 0 \\
\frac{1}{\tau^n}Mu^n + \theta Au^n &= \frac{1}{\tau^n}Mu^{n-1} + (\theta-1)Au^{n-1} \\
u^n + \tau^n M^{-1}\theta Au^n &= u^{n-1} + \tau^n M^{-1}(\theta-1)Au^{n-1}
\end{aligned}
$$

$M = (m_{kl})$, $A = (a_{kl})$ with

$$
a_{kl} = \begin{cases} \sum_{l'\in\mathcal{N}_k} \kappa\frac{|\sigma_{kl'}|}{h_{kl'}} & l = k \\ -\kappa\frac{\sigma_{kl}}{h_{kl}}, & l \in \mathcal{N}_k \\ 0, & else \end{cases}
$$

$$
m_{kl} = \begin{cases} |\omega_k| & l = k \\ 0, & else \end{cases}
$$

A matrix norm estimate

Lemma: Assume A has positive main diagonal entries, nonpositive off-diagonal entries and row sum zero. Then, $||(I+A)^{-1}||_\infty \leq 1$

Proof: Assume that $||(I+A)^{-1}||_\infty > 1$. $I+A$ is a irreducible M-matrix, thus $(I+A)^{-1}$ has positive entries. Then for α_{ij} being the entries of $(I+A)^{-1}$,

$$\max_{i=1}^{n} \sum_{j=1}^{n} \alpha_{ij} > 1.$$

Let k be a row where the maximum is reached. Let $e = (1\dots 1)^T$. Then for $v = (I+A)^{-1}e$ we have that $v > 0$, $v_k > 1$ and $v_k \geq v_j$ for all $j \neq k$. The kth equation of $e = (I+A)v$ then looks like

$$\begin{aligned}
1 &= v_k + v_k \sum_{j\neq k} |a_{kj}| - \sum_{j\neq k} |a_{kj}| v_j \\
&\geq v_k + v_k \sum_{j\neq k} |a_{kj}| - \sum_{j\neq k} |a_{kj}| v_k \\
&= v_k \\
&> 1
\end{aligned}$$

This contradiction enforces $||(I+A)^{-1}||_\infty \leq 1$. $\square$

Stability estimate

$$u^n + \tau^n M^{-1}\theta A u^n = u^{n-1} + \tau^n M^{-1}(\theta-1)Au^{n-1} =: B^n u^{n-1}$$
$$u^n = (I + \tau^n M^{-1}\theta A)^{-1} B^n u^{n-1}$$

From the lemma we have $||(I + \tau^n M^{-1}\theta A)^n||_\infty \leq 1$ and $||u^n||_\infty \leq ||B^n u^{n-1}||_\infty$.

For the entries b^n_{kl} of B^n, we have

$$b^n_{kl} = \begin{cases} 1 + \frac{\tau^n}{m_{kk}}(\theta-1)a_{kk}, & k = l \\ \frac{\tau^n}{m_{kk}}(\theta-1)a_{kl}, & else \end{cases}$$

In any case, $b_{kl} \geq 0$ for $k \neq l$. If $b_{kk} \geq 0$, one estimates

$$||B||_\infty = \max_{k=1}^{N} \sum_{l=1}^{N} b_{kl}.$$

But

$$\sum_{l=1}^{N} b_{kl} = 1 + (\theta-1)\frac{\tau^n}{m_{kk}}\left(a_{kk} + \sum_{l\in\mathcal{N}_k} a_{kl}\right) = 1$$

$$||B||_\infty = 1.$$

Stability conditions

- For a shape regular triangulation in $\mathbb{R}^d$, we can assume that $m_{kk} = |\omega_k| \sim h^d$, and $a_{kl} = \frac{|\sigma_{kl}|}{h_{kl}} \sim \frac{h^{d-1}}{h} = h^{d-2}$, thus $\frac{a_{kk}}{m_{kk}} \leq \frac{1}{Ch^2}$
- $b_{kk} \geq 0$ gives

$$(1-\theta)\frac{\tau^n}{m_{kk}} a_{kk} \leq 1$$

- A sufficient condition is

$$C(1-\theta)\frac{\tau^n}{Ch^2} \leq 1$$
$$(1-\theta)\tau^n \leq Ch^2$$

- Method stability:
  - Implicit Euler: $\theta = 1 \Rightarrow$ unconditional stability !
  - Explicit Euler: $\theta = 0 \Rightarrow$ CFL condition $\tau \leq Ch^2$
  - Crank-Nicolson: $\theta = \frac{1}{2} \Rightarrow$ CFL condition $\tau \leq 2Ch^2$
    Tradeoff stability vs. accuracy.

Stability discussion

- $\tau \leq Ch^2$ CFL == "Courant-Friedrichs-Levy"
- Explicit (forward) Euler method can be applied on very fast systems (GPU), with small time step comes a high accuracy in time.
- Implicit Euler: unconditional stability – helpful when stability is of utmost importance, and accuracy in time is less important
- For hyperbolic systems (pure convection without diffusion), the CFL conditions is $\tau \leq Ch$, thus in this case explicit computations are ubiquitous
- Comparison for a fixed size of the time interval. Assume for implicit Euler, time accuracy is less important, and the number of time steps is independent of the size of the space discretization.

	1D	2D	3D
# unknowns	$N = O(h^{-1})$	$N = O(h^{-2})$	$N = O(h^{-3})$
# steps	$M = O(N^2)$	$M = O(N)$	$M = O(N^{2/3})$
complexity	$M = O(N^3)$	$M = O(N^2)$	$M = O(N^{5/3})$

Backward Euler: discrete maximum principle

$$
\begin{aligned}
\frac{1}{\tau^n} M u^n + A u^n &= \frac{1}{\tau} M u^{n-1} \\
\frac{1}{\tau^n} m_{kk} u_k^n + a_{kk} u_k^n &= \frac{1}{\tau^n} m_{kk} u_k^{n-1} + \sum_{k \neq l} (-a_{kl}) u_l^n \\
u_k^n &= \frac{1}{\frac{1}{\tau^n} m_{kk} + \sum_{l \neq k} (-a_{kl})} \left(\frac{1}{\tau^n} m_{kk} u_k^{n-1} + \sum_{l \neq k} (-a_{kl}) u_l^n \right) \\
&\leq \frac{\frac{1}{\tau^n} m_{kk} + \sum_{l \neq k} (-a_{kl})}{\frac{1}{\tau^n} m_{kk} + \sum_{l \neq k} (-a_{kl})} \max(\{u_k^{n-1}\} \cup \{u_l^n\}_{l \in \mathcal{N}_k}) \\
&\leq \max(\{u_k^{n-1}\} \cup \{u_l^n\}_{l \in \mathcal{N}_k})
\end{aligned}
$$

- Provided, the right hand side is zero, the solution in a given node is bounded by the value from the old timestep, and by the solution in the neigboring points.
- No new local maxima can appear during time evolution
- There is a continuous counterpart which can be derived from weak solution
- Sign pattern is crucial for the proof.

Backward Euler: Nonnegativity

$$\begin{aligned} u^n + \tau^n M^{-1} A u^n &= u^{n-1} \\ u^n &= (I + \tau^n M^{-1} A)^{-1} u^{n-1} \end{aligned}$$

- $(I + \tau^n M^{-1} A)$ is an M-Matrix
- If $u_0 > 0$, then $u^n > 0 \; \forall n > 0$

Mass conservation

- Equivalent of $\int_\Omega \nabla \cdot \kappa \nabla u d\mathbf{x} = \int_{\partial\Omega} \kappa \nabla u \cdot \mathbf{n} d\gamma = 0$:

$$\begin{aligned}\sum_{k=1}^{N}\left(a_{kk}u_k + \sum_{l\in\mathcal{N}_k} a_{kl}u_l\right) &= \sum_{k=1}^{N}\sum_{l=1,l\neq k}^{N} a_{kl}(u_l - u_k)\\ &= \sum_{k=1}^{N}\sum_{l=1,l<k}^{N} \left(a_{kl}(u_l - u_k) + a_{lk}(u_k - u_l)\right)\\ &= 0\end{aligned}$$

- $\Rightarrow$ Equivalent of $\int_\Omega u^n d\mathbf{x} = \int_\Omega u^{n-1} d\mathbf{x}$:
  - $\sum_{k=1}^{N} m_{kk} u_k^n = \sum_{k=1}^{N} m_{kk} u_k^{n-1}$

Weak formulation of time step problem

- Weak formulation: search $u \in H^1(\Omega)$ such that $\forall v \in H^1(\Omega)$

$$\frac{1}{\tau^n}\int_\Omega u^n v\,dx + \theta\int_\Omega \kappa\nabla u^n \nabla v\,dx = \frac{1}{\tau^n}\int_\Omega u^{n-1} v\,dx + (1-\theta)\int_\Omega \kappa\nabla u^{n-1}\nabla v\,dx$$

- Matrix formulation

$$\frac{1}{\tau^n}Mu^n + \theta A u^n = \frac{1}{\tau^n}Mu^{n-1} + (1-\theta)Au^{n-1}$$

- M: mass matrix, A: stiffness matrix.
- With FEM, Mass matrix lumping important for getting the previous estimates

Examination dates

Tue Feb 26.
Wed Feb 27.
Wed Mar 14.
Thu Mar 15.
Tue Mar 26.
Wed Mar 27.
Mon Apr 29.(?)
Tue Apr 30.(?)

Time: 10:00-13:00 (6 slots per examination date)

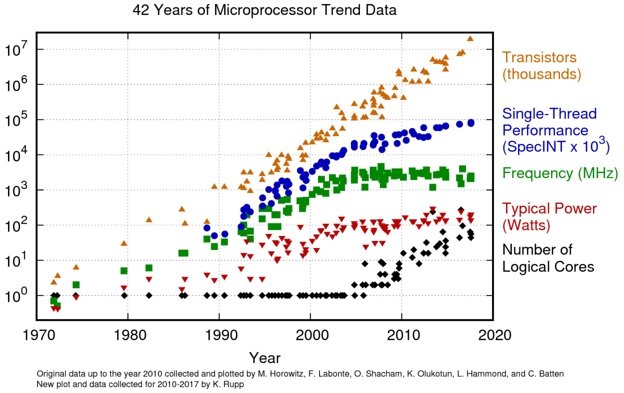
Please inscribe yourself into the corresponding sheets. (See also the back sides).

Room: t.b.a. (MA, third floor)

Prof. Nabben answers all administrative questions.

Please bring your yellow sheets 3 days before the examination to Frau Gillmeister

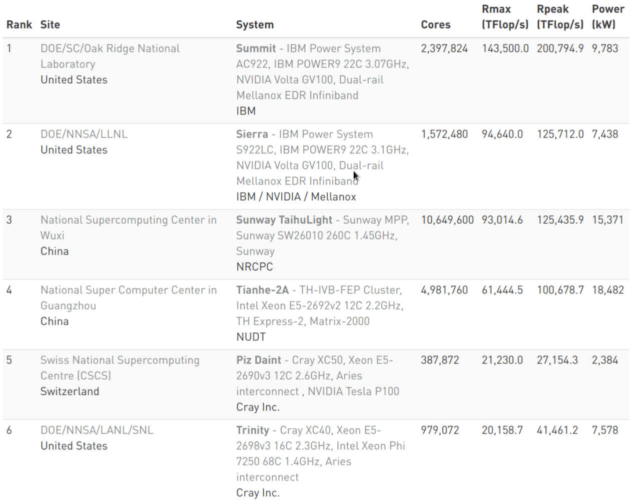
Why parallelization ?



- Clock rate of processors limited due to physical limits
- $\Rightarrow$ parallelization: main road to increase the amount of data processed
- Parallel systems nowadays ubiquitous: even laptops and smartphones have multicore processors
- Amount of accessible memory per processor is limited $\Rightarrow$ systems with large memory can be created based on parallel processors

TOP 500 2018 rank 1-6

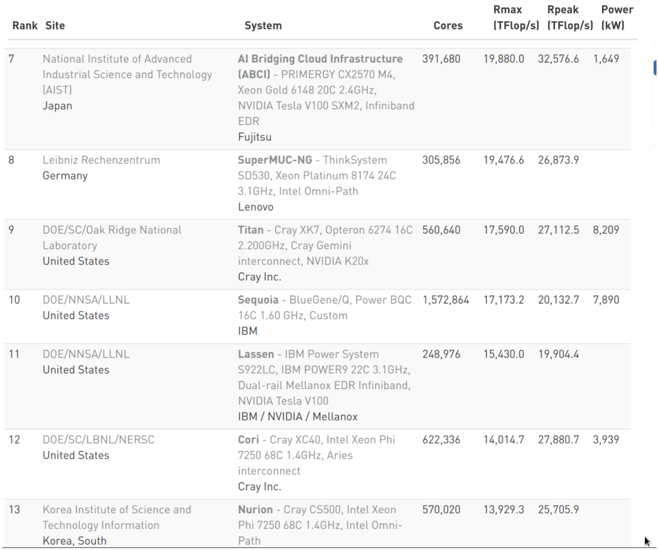
Based on linpack benchmark: solution of dense linear system. Typical desktop computer: $R_{max} \approx 100 \ldots 1000 GFlop/s$



Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	DOE/SC/Oak Ridge National Laboratory United States	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband IBM	2,397,824	143,500.0	200,794.9	9,783
2	DOE/NNSA/LLNL United States	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband IBM / NVIDIA / Mellanox	1,572,480	94,640.0	125,712.0	7,438
3	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
4	National Super Computer Center in Guangzhou China	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 NUDT	4,981,760	61,444.5	100,678.7	18,482
5	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 Cray Inc.	387,872	21,230.0	27,154.3	2,384
6	DOE/NNSA/LANL/SNL United States	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect Cray Inc.	979,072	20,158.7	41,461.2	7,578

[Source:www.top500.org]

TOP 500 2018 rank 7-13

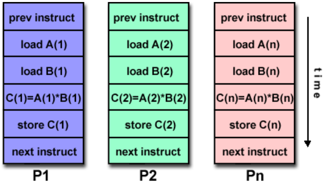


Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
7	National Institute of Advanced Industrial Science and Technology (AIST) Japan	**AI Bridging Cloud Infrastructure (ABCI)** - PRIMERGY CX2570 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR Fujitsu	391,680	19,880.0	32,576.6	1,649
8	Leibniz Rechenzentrum Germany	**SuperMUC-NG** - ThinkSystem SD530, Xeon Platinum 8174 24C 3.1GHz, Intel Omni-Path Lenovo	305,856	19,476.6	26,873.9	
9	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
10	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
11	DOE/NNSA/LLNL United States	**Lassen** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, Dual-rail Mellanox EDR Infiniband, NVIDIA Tesla V100 IBM / NVIDIA / Mellanox	248,976	15,430.0	19,904.4	
12	DOE/SC/LBNL/NERSC United States	**Cori** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect Cray Inc.	622,336	14,014.7	27,880.7	3,939
13	Korea Institute of Science and Technology Information Korea, South	**Nurion** - Cray CS500, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path	570,020	13,929.3	25,705.9	

[Source:www.top500.org]

Parallel paradigms

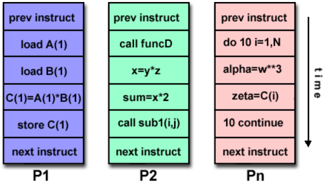
SIMD
Single Instruction Multiple Data



[Source: computing.llnl.gov/tutorials]

- "classical" vector systems: Cray, Convex . . .
- Graphics processing units (GPU)

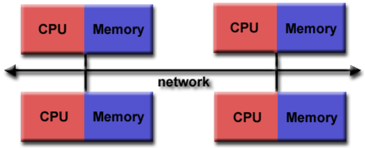
MIMD
Multiple Instruction Multiple Data



[Source: computing.llnl.gov/tutorials]

- Shared memory systems
  - IBM Power, Intel Xeon, AMD Opteron . . .
  - Smartphones . . .
  - Xeon Phi R.I.P.
- Distributed memory systems
  - interconnected CPUs

MIMD Hardware: Distributed memory



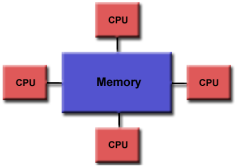
[Source: computing.llnl.gov/tutorials]

- "Linux Cluster"
- "Commodity Hardware"
- Memory scales with number of CPUs interconneted
- High latency for communication
- Mostly programmed using MPI (Message passing interface)
- Explicit programming of communications:
  gather data, pack, send, receive, unpack, scatter

```
    MPI_Send(buf,count,type,dest,tag,comm)
    MPI_Recv(buf,count,type,src,tag,comm,stat)
```

MIMD Hardware: Shared Memory

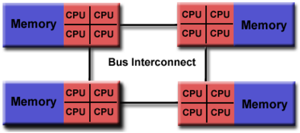
Symmetric Multiprocessing (SMP)/Uniform memory acces (UMA)



[Source: computing.llnl.gov/tutorials]

- Similar processors
- Similar memory access times

Nonuniform Memory Access (NUMA)



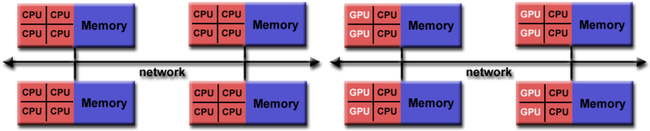
[Source: computing.llnl.gov/tutorials]

- Possibly varying memory access latencies
- Combination of SMP systems
- ccNUMA: Cache coherent NUMA

- Shared memory: one (virtual) address space for all processors involved
- Communication hidden behind memory acces
- Not easy to scale large numbers of CPUS
- MPI works on these systems as well

Hybrid distributed/shared memory

- Combination of shared and distributed memory approach
- Top 500 computers



[Source: computing.llnl.gov/tutorials]

- Shared memory nodes can be mixed CPU-GPU
- Need to master both kinds of programming paradigms

Shared memory programming: pthreads

- Thread: lightweight process which can run parallel to others
- pthreads (POSIX threads): widely distributed
- cumbersome tuning + syncronization
- basic structure for higher level interfaces

```
#include <pthread.h>
void *PrintHello(void *threadid)
{  long tid = (long)threadid;
  printf("Hello World! It's me, thread #%ld!\n", tid);
  pthread_exit(NULL);
}
int main (int argc, char *argv[])
{ pthread_t threads[NUM_THREADS];
  int rc;   long t;
  for(t=0; t<NUM_THREADS; t++){
    printf("In main: creating thread %ld\n", t);
    rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
    if (rc) {printf("ERROR; return code from pthread_create() is %d\n", rc
    }
    pthread_exit(NULL);
  }
}
```

Source: computing.llnl.gov/tutorials

- compile and link with

Shared memory programming: C++11 threads

- Threads introduced into C++ standard with C++11
- Quite late… many codes already use other approaches
- But interesting for new applications

```
#include <iostream>
#include <thread>

void call_from_thread(int tid) {
  std::cout << "Launched by thread " << tid << std::endl;
}

int main() {
  std::thread t[num_threads];
  for (int i = 0; i < num_threads; ++i) {
    t[i] = std::thread(call_from_thread, i);
  }
  std::cout << "Launched from main\n";
  //Join the threads with the main thread
  for (int i = 0; i < num_threads; ++i) {
    t[i].join();
  }
  return 0;
}
```

Source: https://solarianprogrammer.com/2011/12/16/cpp-11-thread-tutorial/

Thread programming: mutexes and locking

- If threads work with common data (write to the same memory address, use the same output channel) access must be synchronized
- Mutexes allow to define regions in a program which are accessed by all threads in a sequential manner.

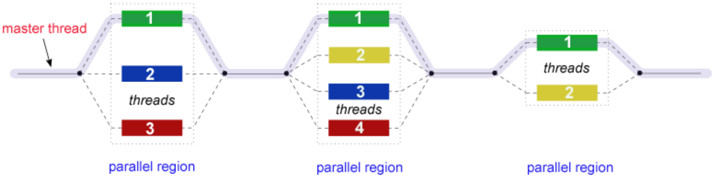
```
#include <mutex>
std::mutex mtx;
void call_from_thread(int tid) {
  mtx.lock()
  std::cout << "Launched by thread " << tid << std::endl;
  mtx.unlock()
}
int main() {
  std::thread t[num_threads];
  for (int i = 0; i < num_threads; ++i) {
    t[i] = std::thread(call_from_thread, i);
  }
  std::cout << "Launched from main\n";
  for (int i = 0; i < num_threads; ++i) t[i].join();
  return 0;
}
```

- *Barrier*: all threads use the same mutex for the same region
- *Deadlock*: two threads block each other by locking two different locks and waiting for each other to finish

Shared memory programming: OpenMP

- Mostly based on pthreads
- Available in C++,C,Fortran for all common compilers
- Compiler directives (pragmas) describe *parallel regions*

```
... sequential code ...
#pragma omp parallel
{
  ... parallel code ...
}
(implicit barrier)
... sequential code ...
```



[Source: computing.llnl.gov/tutorials]

Shared memory programming: OpenMP II

```
#include <iostream>
#include <cstdlib>

void call_from_thread(int tid) {
  std::cout << "Launched by thread " << tid << std::endl;
}

int main (int argc, char *argv[])
{
  int num_threads=1;
  if (argc>1) num_threads=atoi(argv[1]);

  #pragma omp parallel for
  for (int i = 0; i < num_threads; ++i)
  {
    call_from_thread(i);
  }
  return 0;
}
```

- compile and link with

```
      g++ -fopenmp -o cppomp cppomp.cxx
```

Example: $u = au + v$ und $s = u \cdot v$

```
double u[n],v[n];
#pragma omp parallel for
for(int i=0; i<n ; i++)
    u[i]+=a*v[i];

//implicit barrier
double s=0.0;
#pragma omp parallel for
for(int i=0; i<n ; i++)
    s+=u[i]*v[i];
```

- Code can be parallelized by introducing compiler directives
- Compiler directives are ignored if not in parallel mode
- Write conflict with $+ s$: several threads may access the same variable

Preventing conflicts in OpenMP

- Critical sections are performed only by one thread at a time

```
double s=0.0;
#pragma omp parallel for
for(int i=0; i<n ; i++)
#pragma omp critical
{
  s+=u[i]*v[i];
}
```

- Expensive, parallel program flow is interrupted

Do it yourself reduction

- Remedy: accumulate partial results per thread, combine them after main loop
- “Reduction”

```
#include <omp.h>
int maxthreads=omp_get_max_threads();
double s0[maxthreads];
double u[n],v[n];
for (int ithread=0;ithread<maxthreads; ithread++)
   s0[ithread]=0.0;

#pragma omp parallel for
for(int i=0; i<n ; i++)
{
  int ithread=omp_get_thread_num();
  s0[ithread]+=u[i]*v[i];
}

double s=0.0;
for (int ithread=0;ithread<maxthreads; ithread++)
  s+=s0[ithread];
```

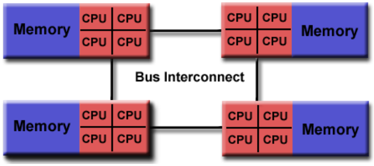
OpenMP Reduction Variables

```
double s=0.0;
#pragma omp parallel for reduction(+:s)
for(int i=0; i<n ; i++)
    s+=u[i]*v[i];
```

- In standard situations, reduction variables can be used to avoid write conflicts, no need to organize this by programmer

OpenMP: further aspects

```
double u[n],v[n];
#pragma omp parallel for
for(int i=0; i<n ; i++)
u[i]+=a*u[i];
```



[Quelle: computing.llnl.gov/tutorials]

- Distribution of indices with thread is implicit and can be influenced by scheduling directives
- Number of threads can be set via `OMP_NUM_THREADS` environment variable or call to `omp_set_num_threads()`
- First Touch Principle (NUMA): first thread which "touches" data triggers the allocation of memory with the processeor where the thread is running on

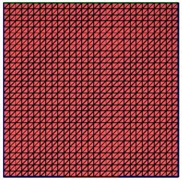
Parallelization of PDE solution

$$\begin{aligned} \Delta u &= f \text{ in}\Omega, & u|_{\partial\Omega} = 0 \\ \Rightarrow u &= \int_\Omega f(y)G(x,y)dy. \end{aligned}$$

- Solution in $x \in \Omega$ is influenced by values of f in all points in Ω
- $\Rightarrow$ global coupling: any solution algorithm needs global communication

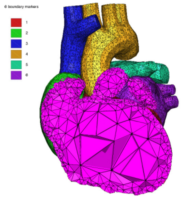
Structured and unstructured grids

Structured grid



- Easy next neighbor access via index calculation
- Efficient implementation on SIMD/GPU
- Strong limitations on geometry

Unstructured grid



[Quelle: tetgen.org]

- General geometries
- Irregular, index vector based access to next neighbors
- Hardly feasible fo SIMD/GPU

Stiffness matrix assembly for Laplace operator for P1 FEM

$$
\begin{aligned}
a_{ij} = a(\phi_i, \phi_j) &= \int_\Omega \nabla\phi_i \nabla\phi_j \, dx \\
&= \int_\Omega \sum_{K\in\mathcal{T}_h} \nabla\phi_i|_K \nabla\phi_j|_K \, dx
\end{aligned}
$$

Assembly loop:
Set $a_{ij} = 0$.
For each $K \in \mathcal{T}_h$:
For each $m, n = 0 \dots d$:

$$
s_{mn} = \int_K \nabla\lambda_m \nabla\lambda_n \, dx
$$

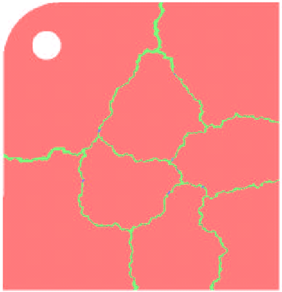
$$
a_{j_{dof}(K,m), j_{dof}(K,n)} = a_{j_{dof}(K,m), j_{dof}(K,n)} + s_{mn}
$$

Mesh partitioning

Partition set of cells in $\mathcal{T}_h$, and color the graph of the partitions.

Result: $\mathcal{C}$: set of colors, $\mathcal{P}_c$: set of partitions of given color. Then: $\mathcal{T}_h = \bigcup_{c\in\mathcal{C}} \bigcup_{p\in\mathcal{P}_c} p$

- Sample algorithm:
  - Subdivision of grid cells into equally sized subsets by METIS (Karypis/Kumar) $\rightarrow$ Partitions of color 1
  - Create separators along boundaries $\rightarrow$ Partitions of color 2
  - "triple points" $\rightarrow$ Partitions of color 3



Parallel stiffness matrix assembly for Laplace operator for P1 FEM

```
Set $a_{ij} = 0$.

For each color $c \in \mathcal{C}$
#pragma omp parallel for
    For each $p \in \mathcal{P}_c$:
        For each $K \in p$:
        For each $m, n = 0 \dots d$:
            $s_{mn} = \int_K \nabla\lambda_m \nabla\lambda_n \, dx$
            $a_{j_{dof}(K,m), j_{dof}(K,n)} += s_{mn}$
```

- Prevent write conflicts by loop organization
- No need for critical sections
- Similar structure for Voronoi finite volumes, nonlinear operator evaluation, Jacobi matrix assembly

Linear system solution

- Sparse matrices
- Direct solvers are hard to parallelize though many efforts are undertaken, e.g. Pardiso
- Iterative methods easier to parallelize
  - partitioning of vectors + coloring inherited from cell partitioning
  - keep loop structure (first touch principle)
  - parallelize
    - vector algebra
    - scalar products
    - matrix vector products
    - preconditioners