

Paralleles Programmieren mit OpenMP und MPI

# OpenMP-Übungsaufgaben

Steinbuch Centre for Computing



Forschungszentrum Karlsruhe  
in der Helmholtz-Gemeinschaft



Universität Karlsruhe (TH)  
Forschungsuniversität • gegründet 1825



# Parallele Berechnung von PI

```
program compute_pi
integer                :: i
integer, parameter    :: n=50000000, dp = kind(1.d0)
real(kind=dp)         :: w,x,sum,pi,d

w=1.0/n; sum=0.0
!$OMP PARALLEL PRIVATE(x,d), SHARED(w,sum)
!$OMP DO REDUCTION(+: sum)
do i=1,n
  x  = (i-0.5) * w
  d  = w * SQRT(1.0 - x**2)
  sum = sum + d
enddo
!$OMP END DO
!$OMP END PARALLEL
pi = 4. * sum
print *, 'computed pi = ', pi
end program compute_pi
```

## HP XC3000

### 1 core:

real	2.92s
user	2.93s

### 2 cores:

real	1.50s
user	2.99s

### 4 cores:

real	0.75s
user	2.97s

### 8 cores:

real	0.39s
user	3.10s

```
cp /work/kit/scc/ku8089/OpenMP-Uebung/pi.f90 .  
(cp /work/kit/scc/ku8089/OpenMP-Uebung/seconds.c .)  
(icc -c -O -DFTNLINKSUFFIX seconds.c)  
ifort -O3 -o pi pi.f90 seconds.o      bzw.  
ifort -O3 -openmp -o pi_par pi.f90 seconds.o  
./pi      bzw.  
export $OMP_NUM_THREADS=4  
./pi_par  oder  
msub -l AVRES=kit-workshop.200 ./jobuc_omp.sh
```

Schreiben Sie das parallele Programm PI so um, dass Sie **ohne die Klausel reduction** auskommen!

# Parallele Berechnung von PI mit `atomic (update)`

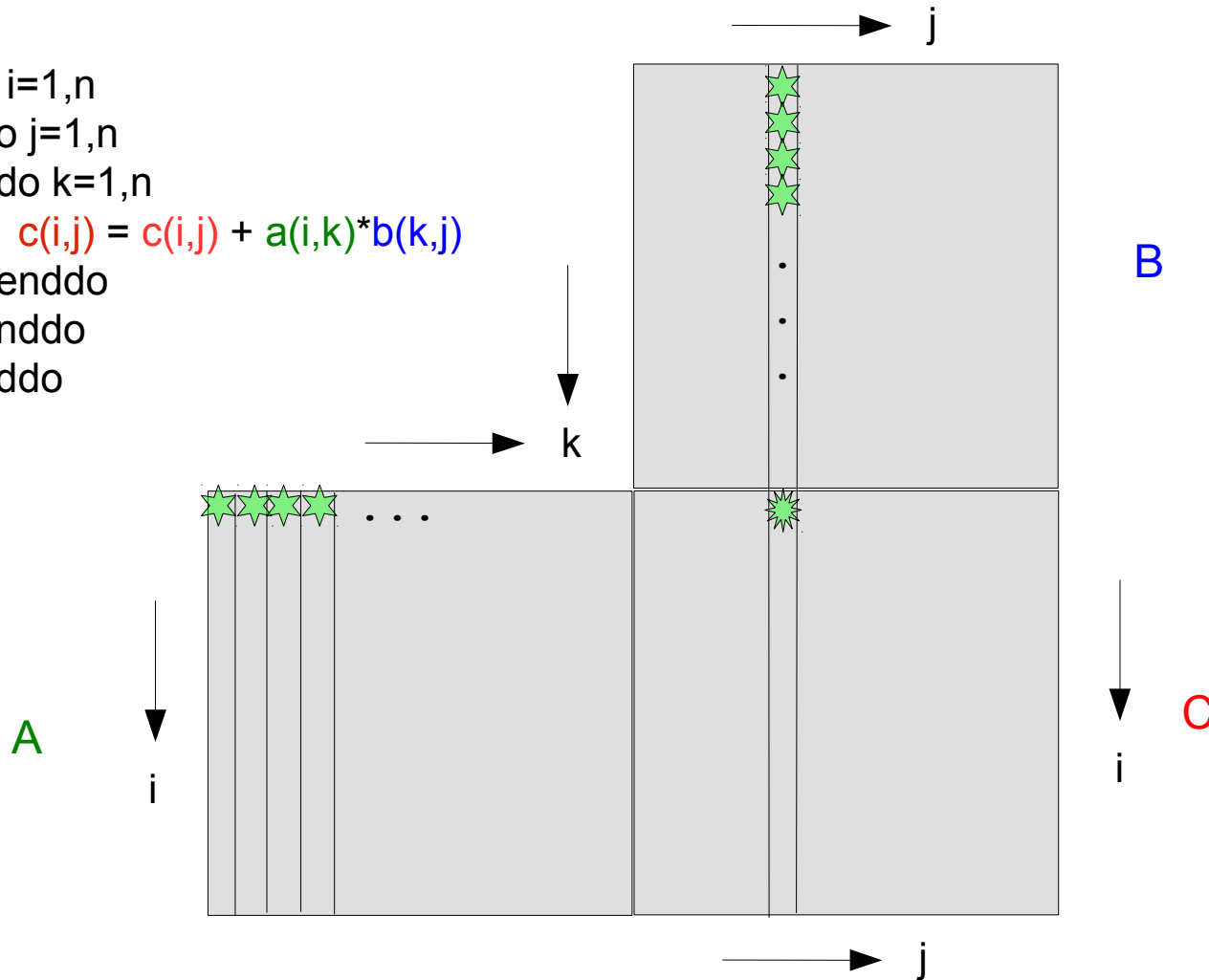
- `atomic` ist ein Spezialfall eines kritischen Abschnitts, der dazu benutzt werden kann, skalaren Variablen in einfachen Zuweisungen einen neuen Wert zuzuweisen.
- Das *assignment statement* muss folgende Syntax aufweisen:  
 $x = x \text{ operator } \text{expr}; \quad x = \text{intrinsic}(x, \text{expr})$

```

program compute_pi
integer                :: i
integer, parameter    :: n=500000000, dp = kind(1.d0)
real(kind=dp)         :: w,x,pi,d,sum,glob_sum
w=1.d0/n; glob_sum=0.d0
!$OMP PARALLEL PRIVATE(x,d,sum)
sum = 0.d0
!$OMP DO
do i=1,n
  x = (i-0.5)*w; d = w*SQRT(1.0-x**2); sum = sum+d
enddo
!$OMP ATOMIC
glob_sum = glob_sum + sum
!$OMP END PARALLEL
pi = 4. * glob_sum
end program compute_pi
  
```

# Die Matrizenmultiplikation $C = A * B$

```
do i=1,n
  do j=1,n
    do k=1,n
       $c(i,j) = c(i,j) + a(i,k)*b(k,j)$ 
    enddo
  enddo
enddo
```



```
ifort -O3 -openmp -o mmul_ijk mmul_ijk.f90 seconds.o  
export OMP_NUM_THREADS=4  
./mmul_ijk      oder  
msub -l ADVRES=kit-workshop.200 jobuc_ompmul.sh
```

**Parallelisieren Sie die Matrizenmultiplikation MMUL\_IJK!**

**Stellen Sie um auf die JKI-Form der Matrizenmultiplikation und parallelisieren Sie diese! (Sie können auch die Umgebungsvariable OMP\_SCHEDULE aktivieren in dem Skript jobuc\_ompmul.sh!)**

```
program parmmul
integer, parameter :: dp=kind(1.d0)
real(kind=dp), dimension(:, :), allocatable:: a, b, c
. . .

!$OMP PARALLEL SHARED(a, b, c)
!$OMP DO
do j = 1, n
  do k = 1, n
    do i = 1, n
      a(i,j) = a(i,j) + b(i,k) * c(k,j)
    end do
  end do
end do
!$OMP END DO
!$OMP END PARALLEL
. . .
```

## HP XC3000 mit n=4000

### 2 MPI-tasks / 1 thread:

CPU (sec) 40.9s

Elapsed (sec) 41.0s

Mflop/s 3124

### 2 MPI-tasks / 2 threads:

CPU (sec) 44.0s

Elapsed (sec) 22.0s

Mflop/s 5817

### 2MPI-tasks / 4 threads:

CPU (sec) 54.1s

Elapsed (sec) 13.5s

Mflop/s 9460

### 2 MPI-tasks / 8 threads:

CPU (sec) 83.4s

Elapsed (sec) 10.7s

Mflop/s 11936

### 16 MPI-tasks:

Elapsed (sec) 8.8s

Mflop/s 14536

# Der Gaußalgorithmus $A*x = b$

```
do j=1,n-1
  pivot = 1/a(j,j)
  do i=j+1,n
    a(i,j) = a(i,j) * pivot
  enddo
```

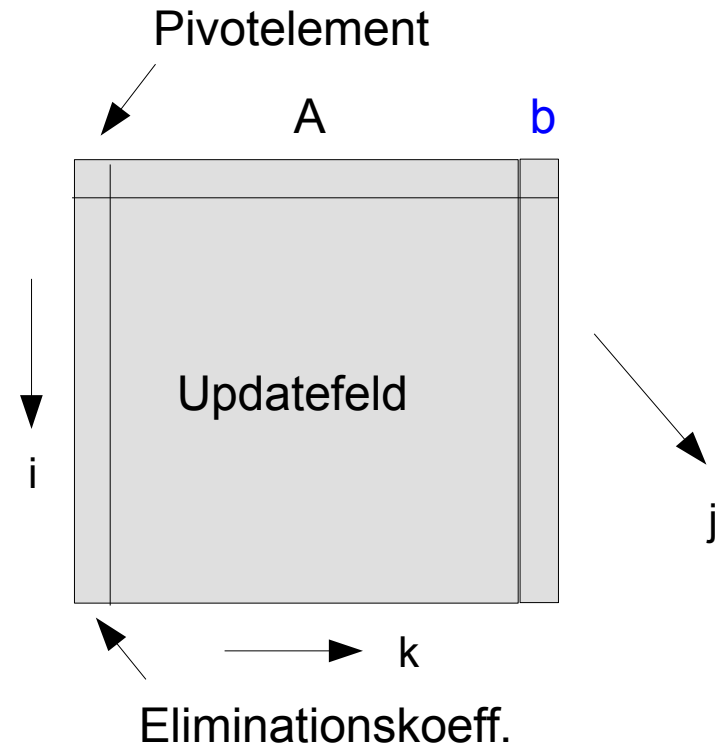
! Berechnung der  
! Elim. koeff.

```
do k=j+1,n
  do i=j+1,n
    a(i,k) = a(i,k) - a(i,j)*a(j,k)
  enddo
enddo
```

! Update  
! der  
! Restmatrix

```
do i=j+1,n
  b(i) = b(i) - a(i,j)*b(j)
enddo
enddo
```

! Update  
! der  
! rechten Seite





```
cp /work/kat/scc/ku8089/OpenMP-Uebung/gauss.f90 .  
(cp /work/kat/scc/ku8089/OpenMP-Uebung/seconds.c .)  
(icc -c -O -DFTNLINKSUFFIX seconds.c)  
ifort -O3 -o gauss gauss.f90 seconds.o      bzw.  
ifort -openmp -O3 -o gauss_par gauss.f90 seconds.o  
./gauss      bzw.  
export OMP_NUM_THREADS=4  
./gauss_par  
msub -l ADVRES=kit-workshop.200 jobuc_ompgauss.sh
```

**Parallelisieren Sie den Gauß-Algorithmus GAUSS!**

**Optimieren Sie den Gauß-Algorithmus für NUMA-Architekturen.**