Karlsruhe Institute of Technology

Paralleles Programmieren mit OpenMP und MPI

MPI-Übungsaufgaben

Vorlesung “Parallelrechner und Parallelprogrammierung”, SoSe 2016

Hartmut Häfner, Steinbuch Centre for Computing (SCC)

STEINBUCH CENTRE FOR COMPUTING - SCC

Summe über alle Prozessoren

Erstellen Sie ein MPI-Programm, in dem Botschaften folgendermaßen verschickt werden:
Prozess 0 sendet eine Botschaft an Prozess 1,
Prozess 1 sendet eine Botschaft an Prozess 2,
…
Prozess N-1 sendet eine Botschaft an Prozess 0.

Die Botschaft soll aus einer 64 Bit Real Größe bestehen, die von der Prozess 0 mit dem Wert 0.0 initialisiert wird. Jede Prozess addiert dann ihre Prozess-Id zu dieser Größe und sendet die Botschaft weiter an die nächste Prozess. Prozess 0 gibt schließlich das Ergebnis aus

```
MPI_Init(&argc, &argv)      MPI_Finalize()
MPI_Comm_rank(MPI_Comm_world, &rank)
MPI_Comm_size(MPI_Comm_world, &size)
MPI_Send(buf, count, datatype, dest, tag, comm)
MPI_Recv(buf, count, datatype, source, tag, comm, status)
```

Kopieren, Kompilieren und Starten von sum_template.c

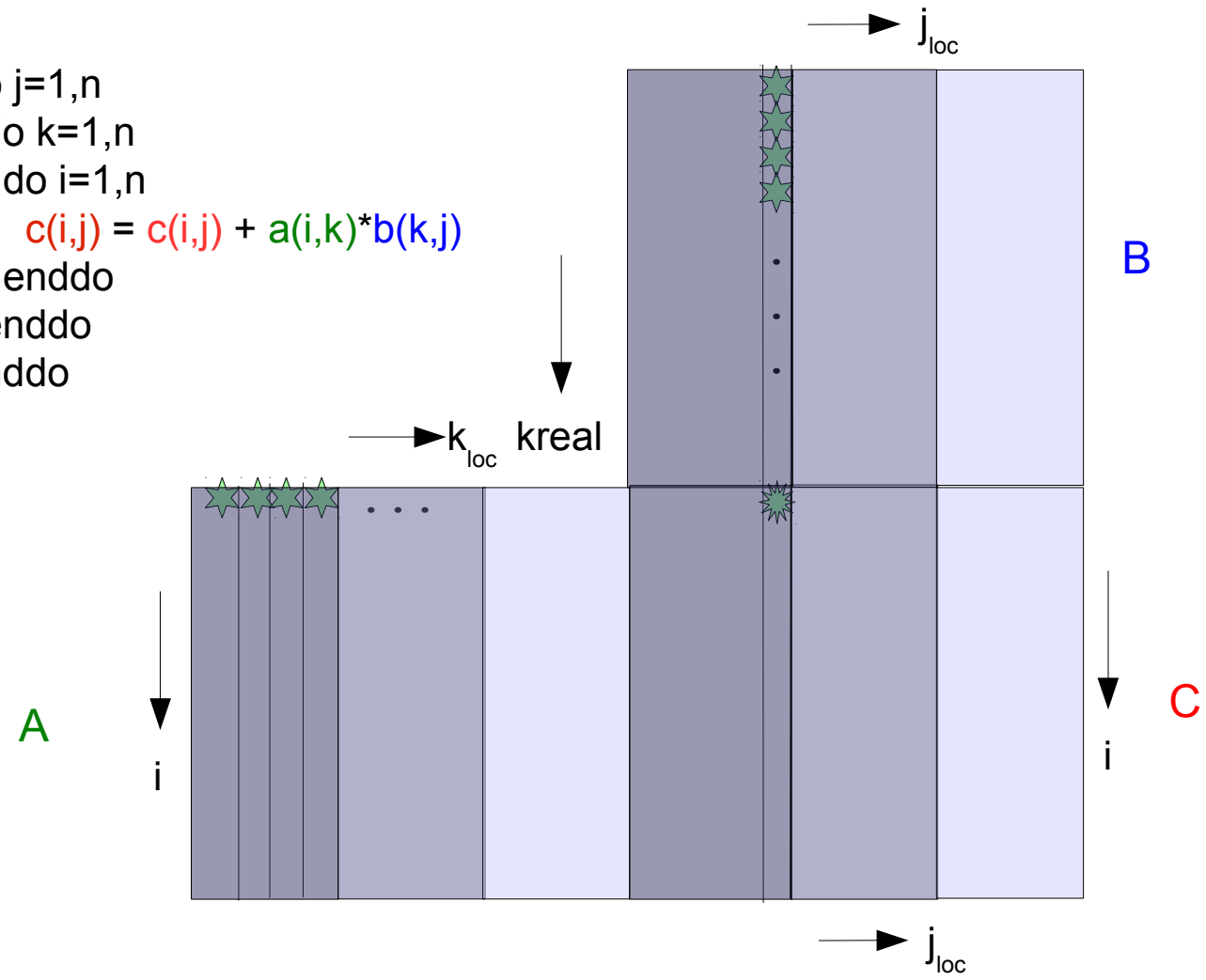
```
module add compiler/intel
module add mpi/openmpi
cp /work/kit/scc/ku8089/MPI-Uebung/mpi_sum_template.c  .
mpicc -O3 -o mpi_sum_simple mpi_sum_template.c
mpirun -np 4 ./mpi_sum_simple
msub -l ADVRES=workshop-kit.201  jobuc_ompi.sh
```

Optimieren Sie das Programm, so dass die Anzahl der benötigten Kommunikationsabschnitte (über die Zeit) auf $O(\log_2 n)$ sinkt. Das Programm muss dabei nur für Prozessorzahlen, die 2er-Potenzen sind, laufen.

Schreiben Sie das Programm so um, dass es auch für alle Prozessorzahlen läuft. (Nicht trivial!)

Die parallele Matrizenmultiplikation C = A * B

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



Kopieren, Kompilieren und Starten von PARMMUL

```
cp /work/kit/scc/ku8089/MPI-Uebung/parmmul_standard.f90  .
cp /work/kit/scc/ku8089/MPI-Uebung/seconds.c  .
icc -c -O -DFTNLINKSUFFIX seconds.c
mpif90 -O3 -o parmmuls parmmul_standard.f90 seconds.o

mpirun -np 4 ./parmmuls          oder
msub -l ADVRES=kit-workshop.201 jobuc_ompi.sh
```

In der parallelen Matrizenmultiplikation PARMMUL wird die blockweise Parallelisierung mit blockierendem Senden und Empfangen verwendet.
a) Schreiben Sie das Programm so um, dass es für alle Matrixgrößen (n) funktioniert.
b) Schreiben Sie das Programm so um, dass es auch auf einem Prozessor läuft.
c) Verwenden Sie in einer weiteren von Ihnen umgeschriebenen Variante nicht-blockierendes Senden und nicht-blockierendes Empfangen, um die Kommunikation hinter der Berechnung (Stichwort: communication hiding) zu verstecken! Hinweis: sie benötigen einen Pufferspeicher für den zu versendenden Matrixblock.

PARMMUL-Code

```
program PARMMUL

integer, parameter ::   n = 100
real*8, parameter ::   one = 1.0, eps = 1.d-10
real*8, pointer ::  a(:,:), ap(:,:), b(:,:), c(:,:)
real*8  t0, t0e, t1, t1e
integer, pointer ::  s(:), kbs(:), kbe(:)
integer ib, lrank, p, rank, totid, frtid, sid1, sid2, rid1, rid2, err
include 'mpif.h'
integer istat(MPI_STATUS_SIZE)

!-------------------------
!**** Communication setup |
!-------------------------
call MPI_INIT(err)
call MPI_COMM_RANK(MPI_COMM_WORLD,rank,err)
call MPI_COMM_SIZE(MPI_COMM_WORLD,p,err)

ns = n/p
allocate(a(n,ns+1), ap(n,ns+1), b(n,ns+1), c(n,ns+1), STAT = err)
allocate(s(p), kbs(p), kbe(p), STAT = err)

!----------------------------------------
!**** Installation of a ring communication |
!----------------------------------------
totid = rank + 1
if (totid == p) totid = 0
```

PARMMUL-Code (2)

```
frtid = rank - 1
if (frtid < 0) frtid = p - 1
lrank = rank + 1                ! 1 <= lrank <= p

do ib=1,MOD(n,p)
  s(ib) = ns + 1                ! s ist Blockbreite
enddo

do ib=MOD(n,p)+1,p
  s(ib) = ns
enddo

kbs(1) = 1                      ! kbs ist Zeiger auf Anfang der Blöcke
kbe(1) = s(1)                   ! kbe ist Zeiger auf Ende der Blöcke

do ib=2,p
  kbs(ib) = kbs(ib-1) + s(ib-1)
  kbe(ib) = kbs(ib) + s(ib) - 1
enddo

do k=1,s(lrank)
  do i=1,n
    a(i,k) = one
    b(i,k) = DBLE(i)
  enddo
enddo
```

PARMMUL-Code (3)

```
call SECONDS(t0,t0e)

do itact=1,p
  ib  = MOD(lrank-itact+p,p) + 1  ! Blocknummer in Takt itakt
  ibn = MOD(ib-2+p,p) + 1         ! Blocknummer der zu empfangenen Daten

!-------------------------------------------------------------------
!**** Standard SEND of A with m-type 10                           |
!-------------------------------------------------------------------
  call MPI_SEND(a(1,1),n*s(ib),MPI_REAL8,totid,10,MPI_COMM_WORLD,err)

  do j=1,s(lrank)                 ! MMUL C = A * B
    do k=1,s(ib)
      kreal = kbs(ib) + k - 1
      if ((k == 1) .and. (itact == 1)) then
        do i=1,n
          c(i,j) = a(i,k)*b(kreal,j)
        enddo
      else
        do i=1,n
          c(i,j) = c(i,j) + a(i,k)*b(kreal,j)
        enddo
      endif
    enddo
  enddo
```

PARMMUL-Code (4)

```
!-------------------------------------------------------
!**** Standard RECV of A with m-type 10                  |
!-------------------------------------------------------
  call MPI_RECV(a(1,1),n*s(ibn),MPI_REAL8,frtid,10,&
                &MPI_COMM_WORLD,istat,err)
enddo
call SECONDS(t1,t1e)

do j=1,s(lrank)
  do i=1,n
    if (ABS(c(i,j)-(n*(n+1)/2.d0)) > eps) then
      print *,' Matrix C is wrong'
    endif
  enddo
enddo
if (lrank == 1) then
  print *,' N=',n,'  Time [sec]=',t1e-t0e,&
          &'  MFLOPS=',2.*n*n*n*1.e-6/(t1e-t0e)
endif

call MPI_FINALIZE(err)
end
```