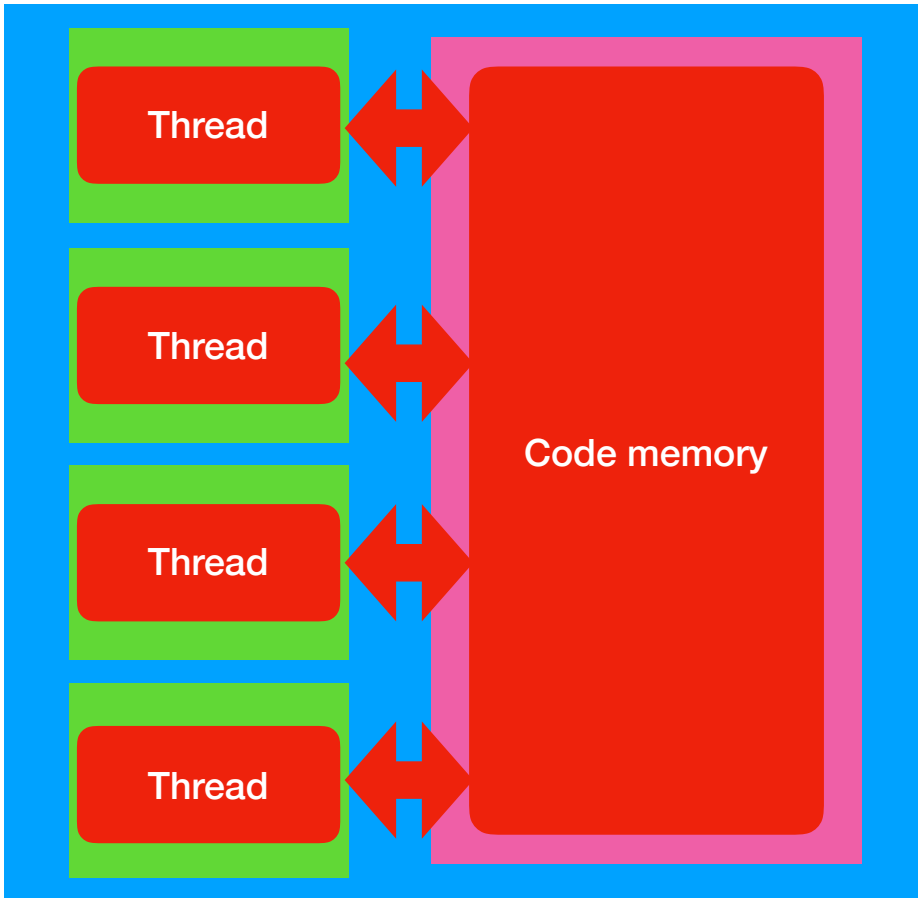
Parallelisation

Tim Harries

openmp versus MPI

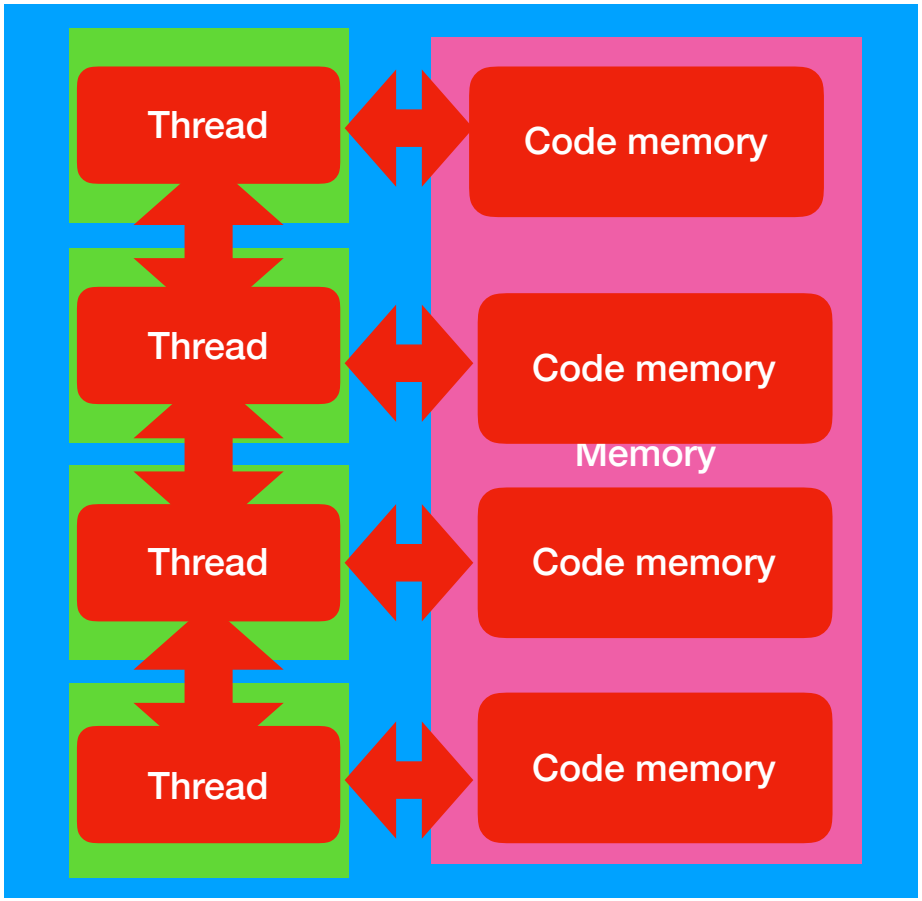
- OpenMP works on shared memory, so each thread can read and write to the same memory
- Implemented as OMP directives in the code (that look like comments)
- Easy to get working
- In MPI each thread has its own memory, and communication is done via subroutine calls
- More complex to implement but potentially more scalable - you can use distributed computers (many CPUs working together and communicating over a network)

Openmp



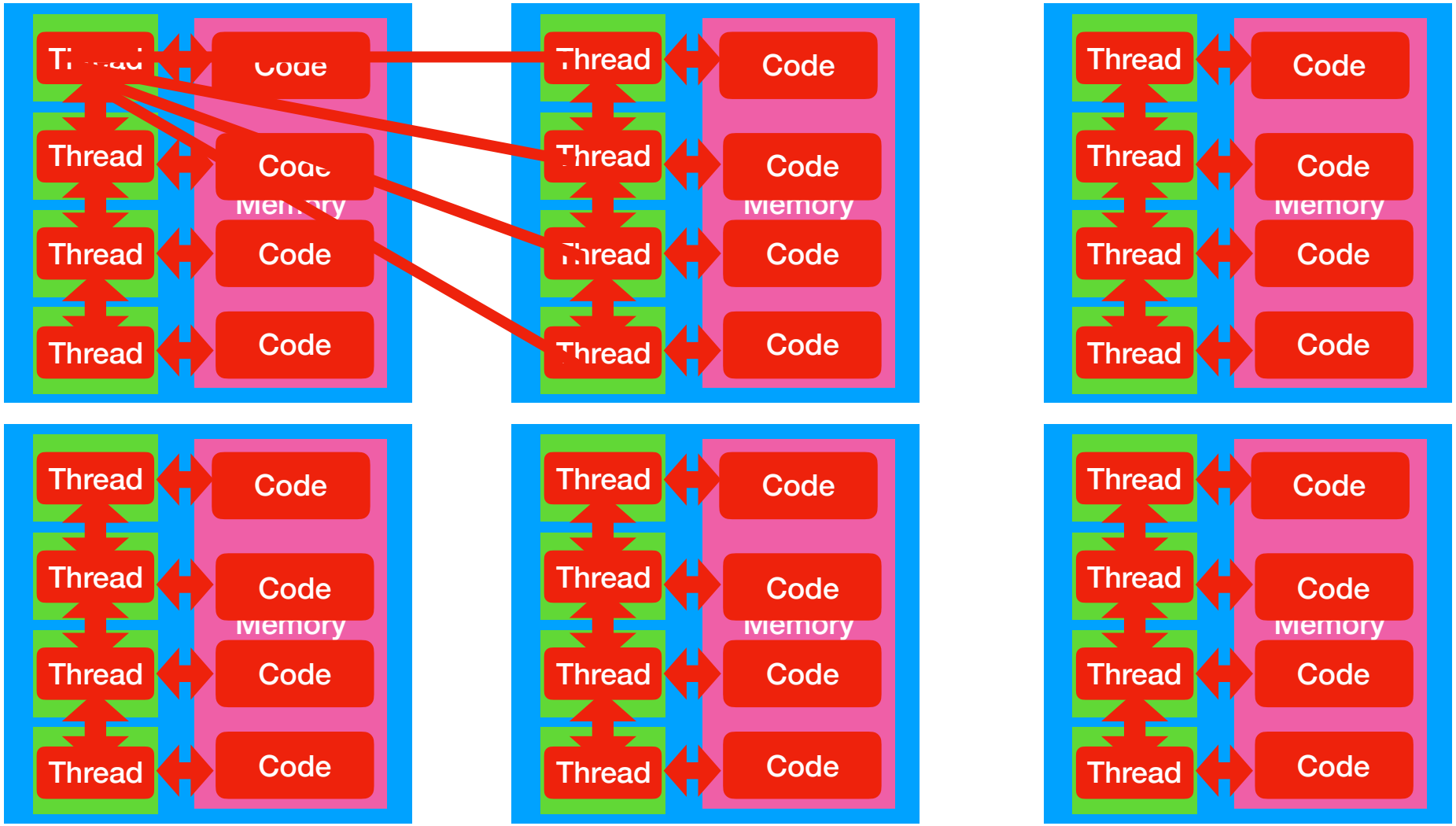
A cartoon of a node

MPI

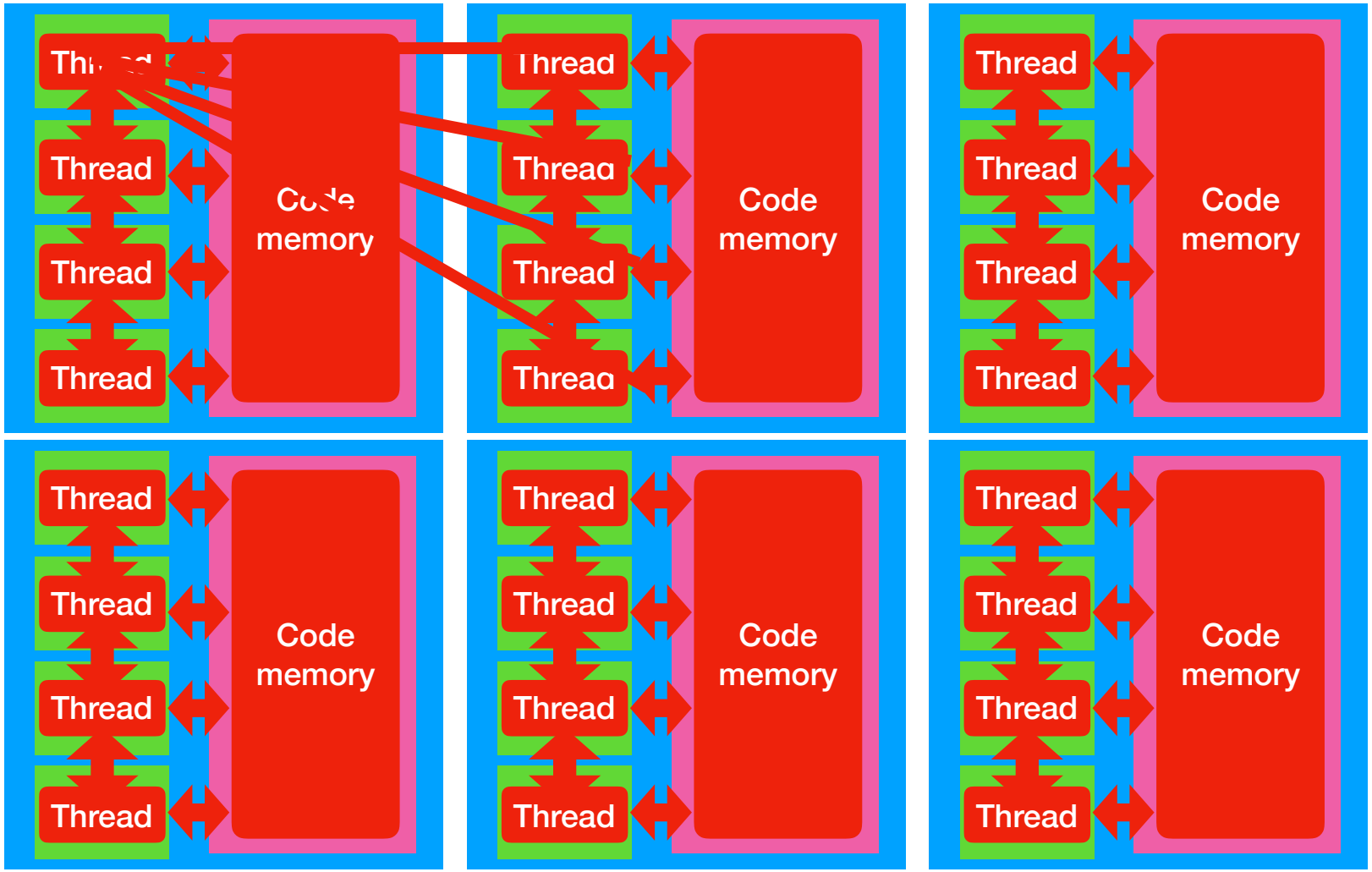


A cartoon of a node

MPI



Hybrid MPI/openmp



Scattering in a uniform density sphere

```
iseed = -23456
r = ran3(iseed) ! seed the random number generator

nScat = 0

do i = 1, nPhotons ! start the photon loop

   x = 0.; y = 0.; z = 0. ! photon position is on the origin
   radius = 0.

   call randomDirection(u,v,w) ! choose an isotropic direction

   do while(radius < tauMax) ! start the scattering loop

      r = ran3(iSeed)
      tau = -log(r) ! select a random tau


      x = x + tau * u; y = y + tau * v; z = z + tau * w ! move the photon that tau

      radius = sqrt(x**2 + y**2 + z**2) ! work out the new radial position

      if (radius < tauMax) then ! if we are still inside the sphere

         nScat = nScat + 1 ! increment the scattering counter

         call randomDirection(u,v,w) ! find a new isotropic direction

      endif
   enddo ! end of scattering loop
enddo ! end of photon loop
```

OMP parallelisation

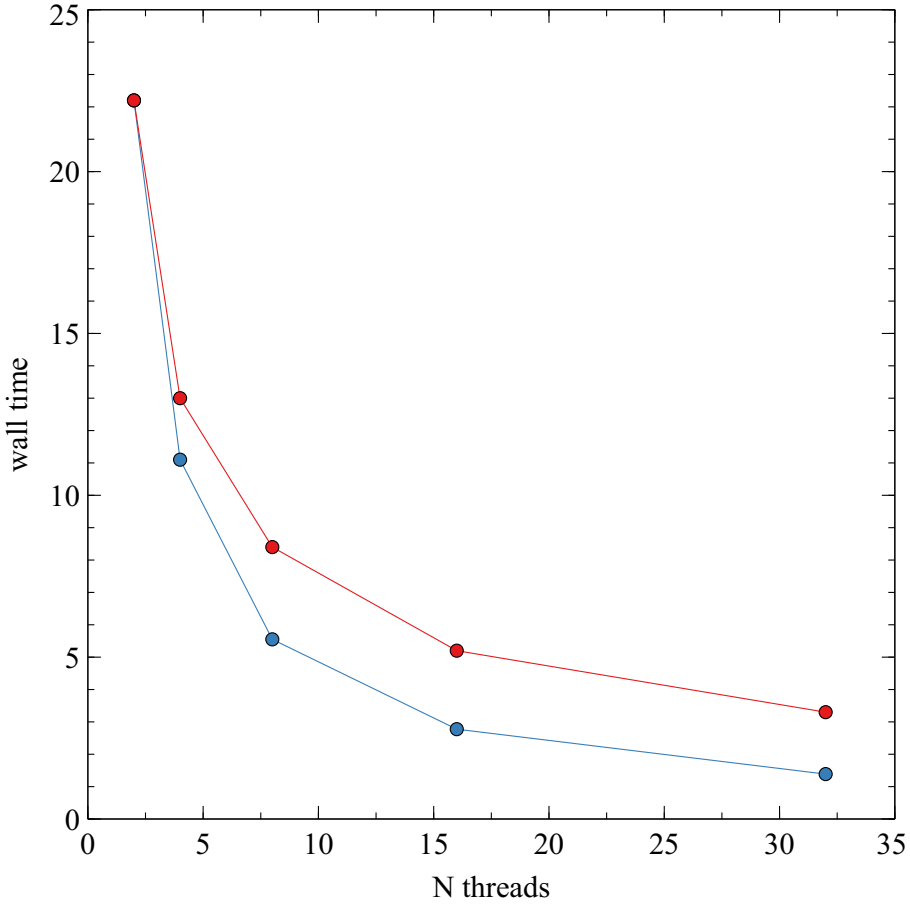
```
!$OMP PARALLEL DEFAULT (NONE) &
!$OMP PRIVATE (i, j, x, y, z, u, v, w, radius, r, tau, iSeed, ithread) &
!$OMP REDUCTION(+: nScat)

  iseed = -23456
  r = ran3(iseed)

!$OMP DO SCHEDULE (STATIC)
  do i = 1, nPhotons ! start the photon loop

     x = 0.; y = 0.; z = 0. ! photon position is on the origin
     radius = 0.

     call randomDirection(u,v,w) ! choose an isotropic direction

     do while(radius < tauMax) ! start the scattering loop

        r = ran3(iSeed)
        tau = -log(r) ! select a random tau


        x = x + tau * u; y = y + tau * v; z = z + tau * w ! move the photon that tau

        radius = sqrt(x**2 + y**2 + z**2) ! work out the new radial position

        if (radius < tauMax) then ! if we are still inside the sphere

           nScat = nScat + 1 ! increment the scattering counter

           call randomDirection(u,v,w) ! find a new isotropic direction

        endif
     enddo ! end of scattering loop
  enddo ! end of photon loop
!$OMP ENDDO
!$OMP END PARALLEL
```

OMP parallelisation

```
!$OMP PARALLEL DEFAULT (NONE) &
!$OMP PRIVATE (i, j, x, y, z, u, v, w, radius, r, tau, iSeed, ithread) &
!$OMP REDUCTION(+: nScat)

  ithread = omp_get_thread_num() ! this is the thread number
  iseed = -23456 + iThread ! set a individual seed for the thread
  r = ran3(iseed)


!$OMP DO SCHEDULE (STATIC)
  do i = 1, nPhotons ! start the photon loop

     x = 0.; y = 0.; z = 0. ! photon position is on the origin
     radius = 0.

     call randomDirection(u,v,w) ! choose an isotropic direction

     do while(radius < tauMax) ! start the scattering loop

        r = ran3(iSeed)
        tau = -log(r) ! select a random tau


        x = x + tau * u; y = y + tau * v; z = z + tau * w ! move the photon that tau

        radius = sqrt(x**2 + y**2 + z**2) ! work out the new radial position

        if (radius < tauMax) then ! if we are still inside the sphere

           nScat = nScat + 1 ! increment the scattering counter

           call randomDirection(u,v,w) ! find a new isotropic direction

        endif
     enddo ! end of scattering loop
  enddo ! end of photon loop
!$OMP ENDDO
!$OMP END PARALLEL
```

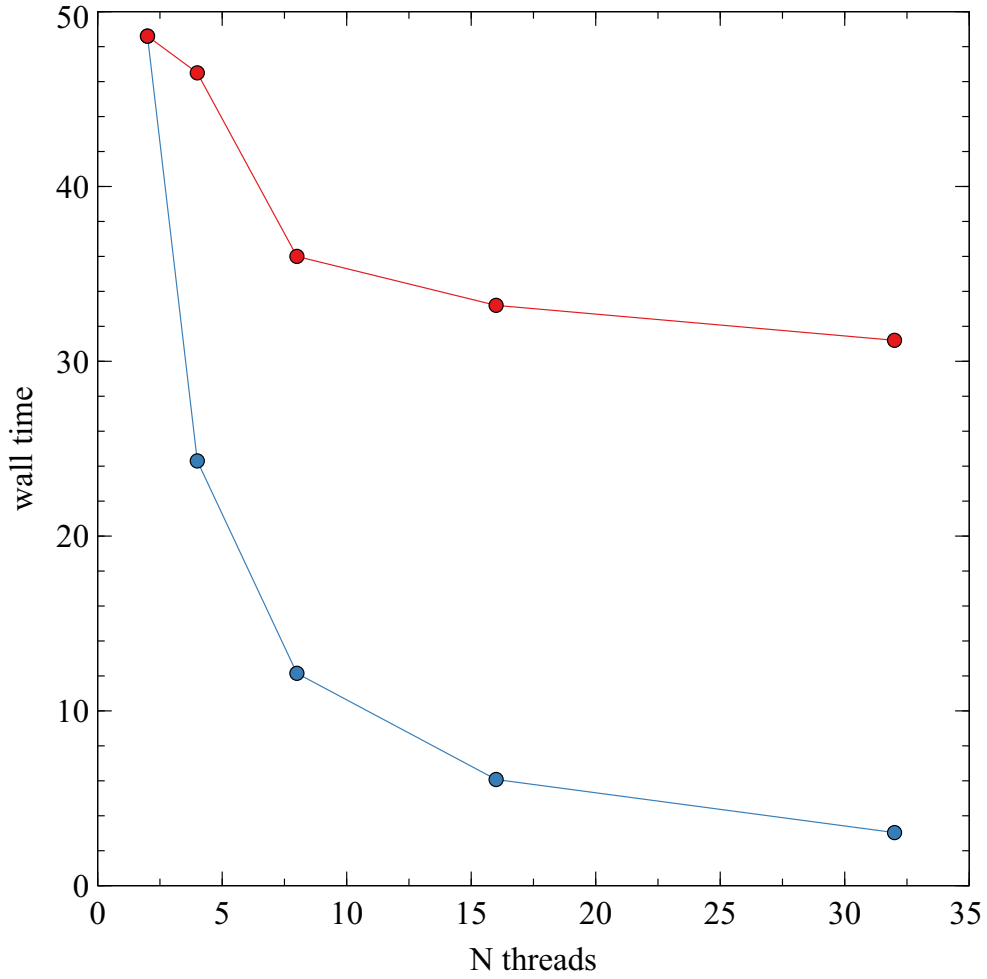
OMP parallelisation

```
REAL FUNCTION RAN3(IDUM)
  implicit none
  integer :: mbig,mseed,mz,idum,mj,ii, k, mk
  real :: fac
  PARAMETER (MBIG=1000000000,MSEED=161803398,MZ=0,FAC=1.E-9)
  real,save ::  MA(55)
  integer, save :: iff = 0
  integer, save :: inext,inextp
!$OMP THREADPRIVATE(MA, IFF, INEXT, INEXTP)
  IF(IDUM.LT.0.OR.IFF.EQ.0)THEN
     IFF=1
     MJ=MSEED-IABS(IDUM)
     MJ=MOD(MJ,MBIG)
     MA(55)=MJ
     MK=1
```


25
20
15
10
5
0
wall time
0
5
10
15
20
25
30
35
N threads

OMP parallelisation

```
!$OMP PARALLEL DEFAULT (NONE) &
!$OMP PRIVATE (i, j, x, y, z, u, v, w, radius, r, tau, iSeed, ithread) &
!$OMP SHARED (iArray) &
!$OMP REDUCTION(+: nScat)

  ithread = omp_get_thread_num() ! this is the thread number
  iseed = -23456 + iThread ! set a individual seed for the thread
  r = ran3(iseed)


!$OMP DO SCHEDULE (STATIC)
  do i = 1, nPhotons ! start the photon loop

     x = 0.; y = 0.; z = 0. ! photon position is on the origin
     radius = 0.

     call randomDirection(u,v,w) ! choose an isotropic direction

     do while(radius < tauMax) ! start the scattering loop

        r = ran3(iSeed)
        tau = -log(r) ! select a random tau


        x = x + tau * u; y = y + tau * v; z = z + tau * w ! move the photon that tau

        radius = sqrt(x**2 + y**2 + z**2) ! work out the new radial position

        if (radius < tauMax) then ! if we are still inside the sphere

           nScat = nScat + 1 ! increment the scattering counter

           j = int(real(nbins)*real(radius)/real(tauMax)) + 1
!$OMP ATOMIC
           iArray(j) = iArray(j) + 1

           call randomDirection(u,v,w) ! find a new isotropic direction

        endif
     enddo ! end of scattering loop
  enddo ! end of photon loop
!$OMP ENDDO
!$OMP END PARALLEL
```

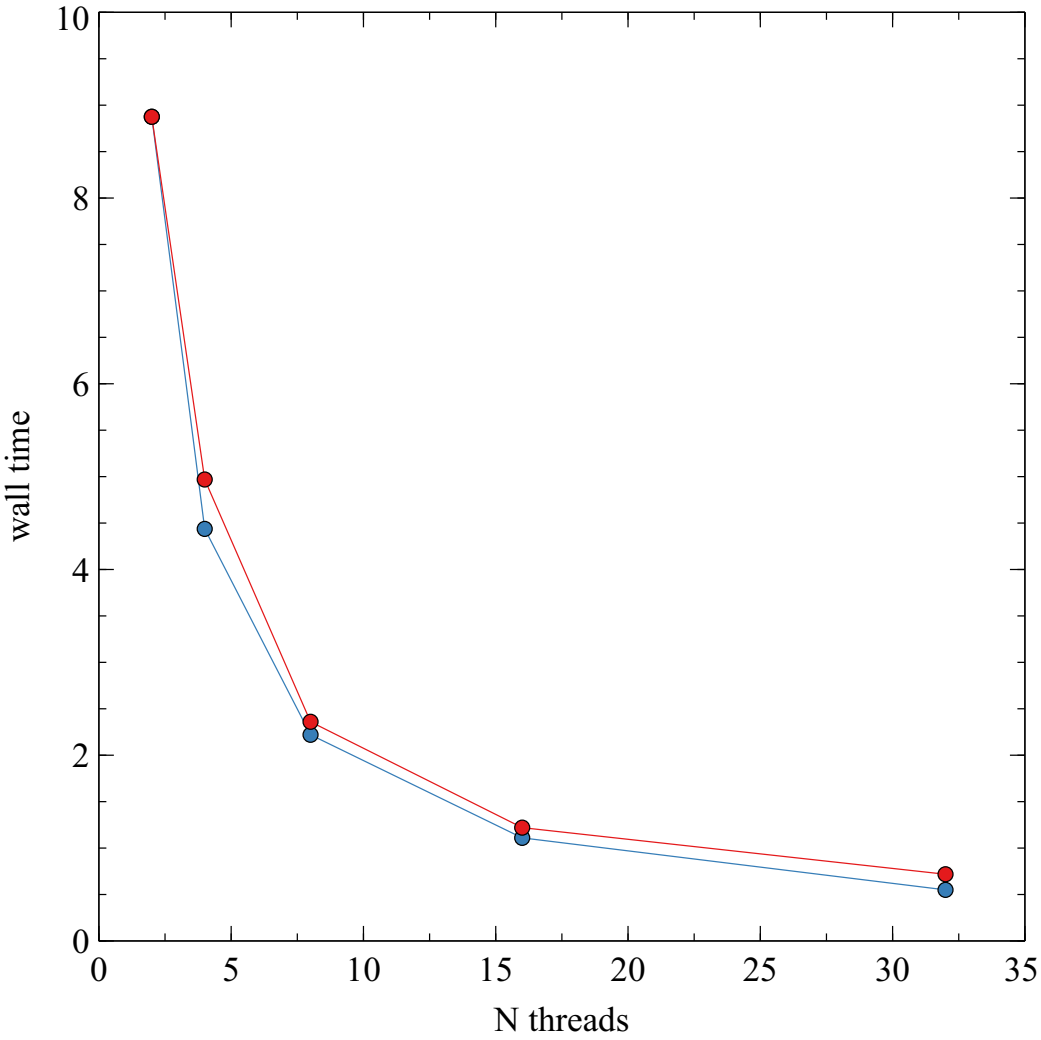

50
40
30
20
10
0
wall time
0
5
10
15
20
25
30
35
N threads

MPI parallelisation

```
call MPI_INIT(ierr) ! initialize MPI
call MPI_COMM_SIZE(MPI_COMM_WORLD, nMPIThreads, ierr) ! sets nMPIThreads to the number of MPI threads
call MPI_COMM_RANK(MPI_COMM_WORLD, myRank, ierr) ! sets myRank to the rank of the MPI thread

call wallTime(startTime)

iseed = -23456 + myRank ! set a individual seed for the thread
r = ran3(iseed)


ibeg = myRank * (nPhotons/nMPIThreads) + 1
iend = (myRank+1) * (nPhotons/nMPIThreads)
if (myRank == (nMPIThreads-1)) iend = nPhotons


do i = iBeg, iEnd ! start the photon loop

   x = 0.; y = 0.; z = 0. ! photon position is on the origin
   radius = 0.

   call randomDirection(u,v,w) ! choose an isotropic direction

   do while(radius < tauMax) ! start the scattering loop

      r = ran3(iSeed)
      tau = -log(r) ! select a random tau


      x = x + tau * u; y = y + tau * v; z = z + tau * w ! move the photon that tau

      radius = sqrt(x**2 + y**2 + z**2) ! work out the new radial position

      if (radius < tauMax) then ! if we are still inside the sphere
```

MPI parallelisation

```
        call randomDirection(u,v,w) ! find a new isotropic direction

      endif
   enddo ! end of scattering loop
enddo ! end of photon loop

call MPI_REDUCE(nScat, iTemp, 1, MPI_INTEGER, MPI_SUM, 0, MPI_COMM_WORLD, ierr) ! sums nScat over all threads and sends it to rank 0
nScat = iTemp

call wallTime(endTime)

if (myRank == 0) write(*,*) "Time taken (seconds) ", endTime-startTime

if (myRank == 0) write(*,*) "Average number of scatterings is ",real(nScat)/real(nPhotons) ! write out the result


call MPI_FINALIZE(ierr)
```


10
8
6
4
2
0
wall time
0
5
10
15
20
25
30
35
N threads

Hybrid parallelisation

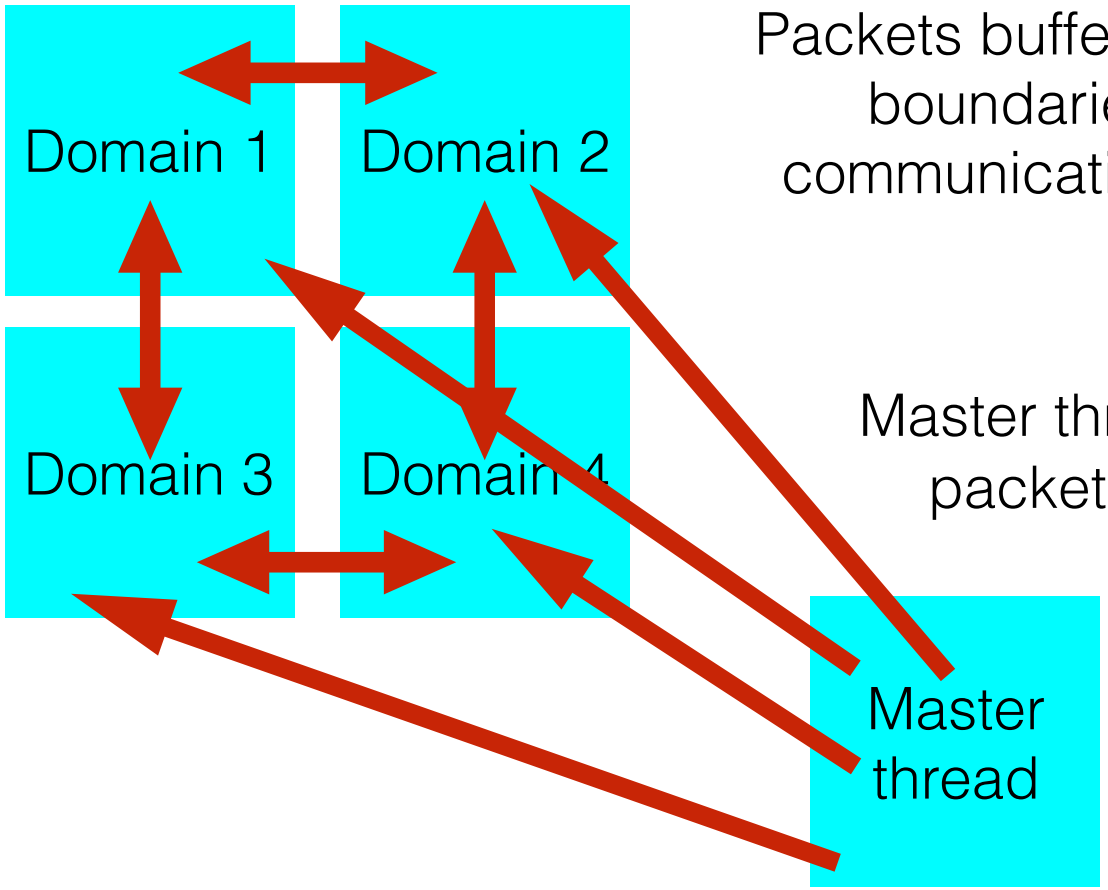
```
  call MPI_INIT(ierr) ! initialize MPI
  call MPI_COMM_SIZE(MPI_COMM_WORLD, nMPIThreads, ierr) ! sets nMPIThreads to the number of MPI threads
  call MPI_COMM_RANK(MPI_COMM_WORLD, myRank, ierr) ! sets myRank to the rank of the MPI thread

  call wallTime(startTime)

!$OMP PARALLEL DEFAULT (NONE) &
!$OMP PRIVATE (i, j, x, y, z, u, v, w, radius, r, tau, iSeed, ithread) &
!$OMP SHARED (ibeg, iend, nMPIThreads, myRank) &
!$OMP REDUCTION(+: nScat)

  ithread = omp_get_thread_num() ! this is the thread number
  iseed = -23456 + 100 * myrank + iThread ! set a individual seed for the thread
  r = ran3(iseed)


  ibeg = 1
  iend = nPhotons

  ibeg = myRank * (nPhotons/nMPIThreads) + 1
  iend = (myRank+1) * (nPhotons/nMPIThreads)
  if (myRank == (nMPIThreads-1)) iend = nPhotons


!$OMP DO SCHEDULE (STATIC)
  do i = iBeg, iEnd ! start the photon loop
```

Hybrid parallelisation

```
        if (radius < tauMax) then ! if we are still inside the sphere

           nScat = nScat + 1 ! increment the scattering counter

           call randomDirection(u,v,w) ! find a new isotropic direction

        endif
     enddo ! end of scattering loop
  enddo ! end of photon loop
!$OMP ENDDO
!$OMP END PARALLEL


  call MPI_REDUCE(nScat, iTemp, 1, MPI_INTEGER, MPI_SUM, 0, MPI_COMM_WORLD, ierr) ! sums nScat over all threads and sends it to rank 0
  nScat = iTemp


  call wallTime(endTime)

  if (myRank == 0) write(*,*) "Time taken (seconds) ", endTime-startTime

  if (myRank == 0) write(*,*) "Average number of scatterings is ",real(nScat)/real(nPhotons) ! write out the result


  call MPI_FINALIZE(ierr)
```

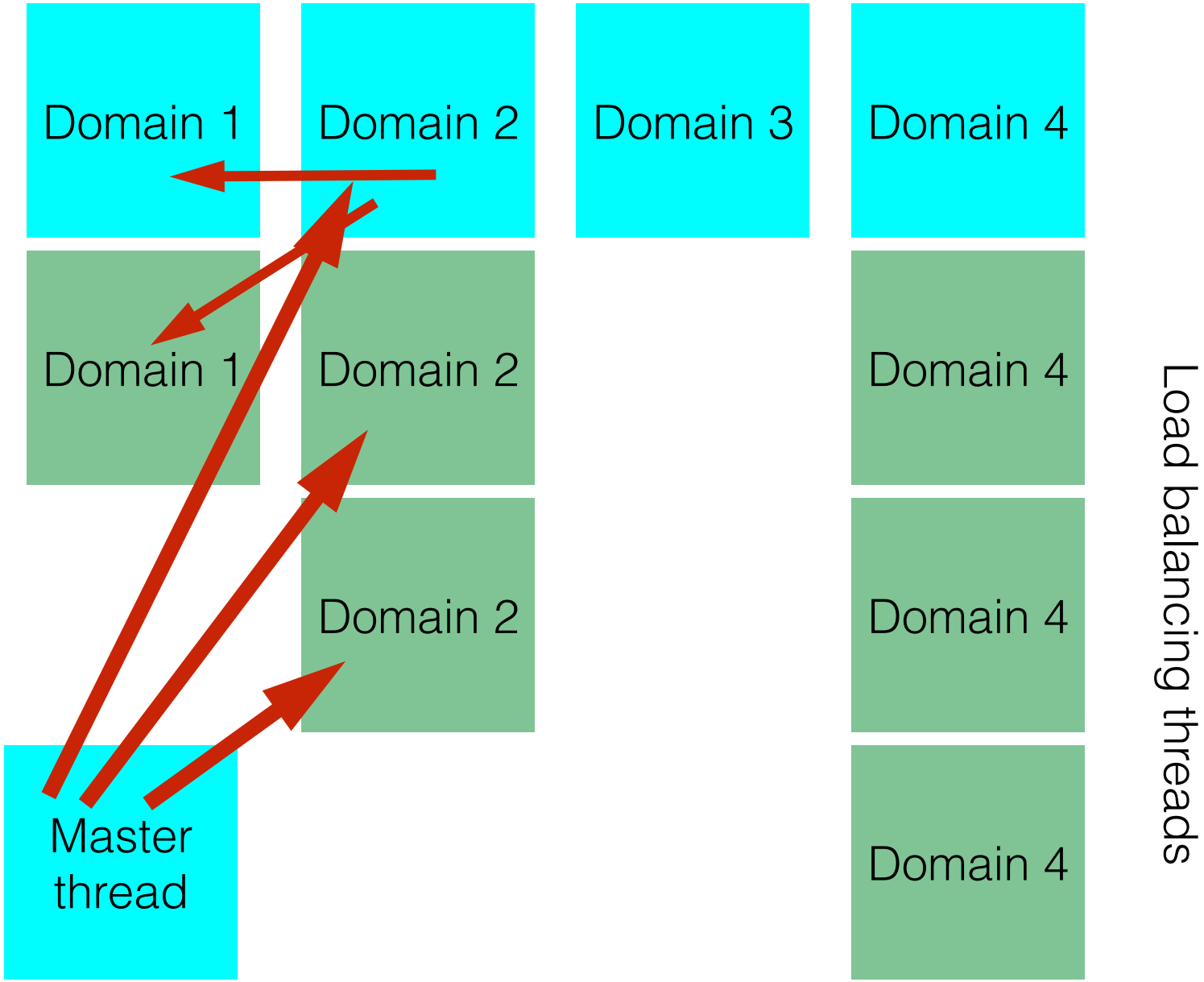
Domain decomposition

Each domain corresponds to a thread



Packets buffered at domain boundaries reduce communication overhead

Master thread distributes packets to domains


Domain 1
Domain 2
Domain 3
Domain 4
Domain 1
Domain 2
Domain 4
Domain 2
Domain 4
Domain 4
Master thread
Load balancing threads

Ahmdal's law

The serial parts of your code (in our case typically those parts outside the main photon loop) won't benefit from parallelisation.

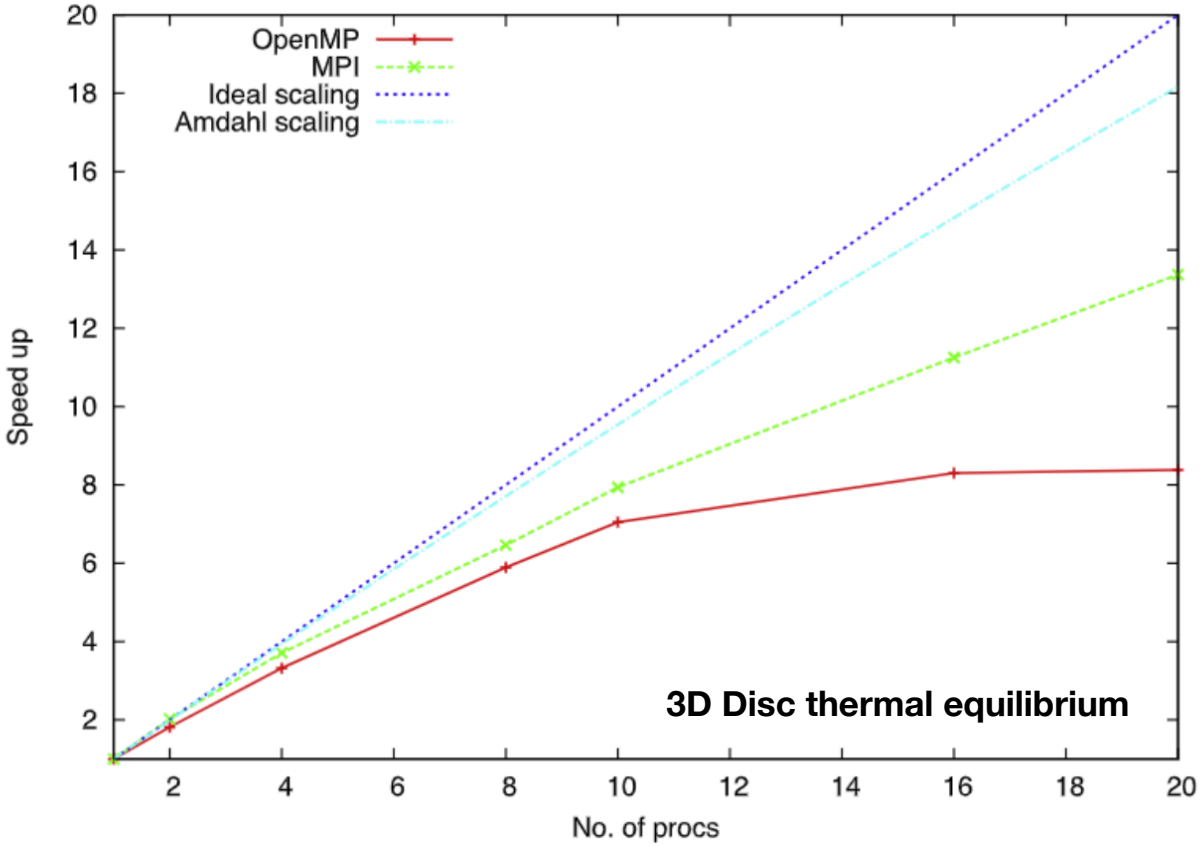
This provides a fundamental limit to the possible speed up of your code, which is given by Ahmdal's law:

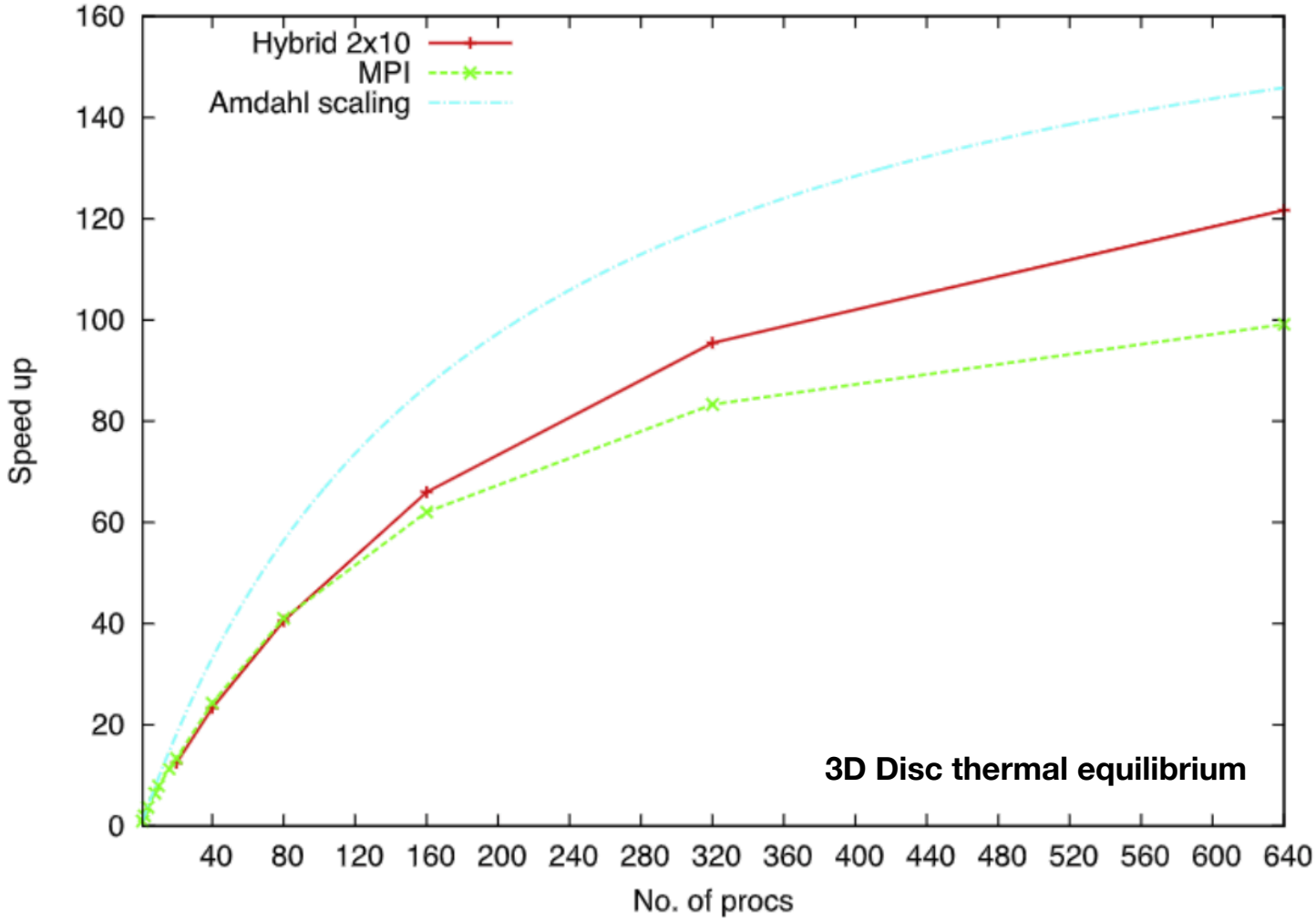
$$S(N) = \frac{1}{\frac{p}{N} + s}$$

S(N) is the speed up for N threads
p is the fraction of parallelisable code
s (=1-p) is the fraction of serial code

From a serial run we determine that the parallel fraction is for TORUS thermal equilibrium is 0.99467 which limits the maximum possible speed up to 188 (i.e. in the absence of any other factors scaling is limited to around 9 nodes with 20 cores per node).

Scaling of TORUS on a single node



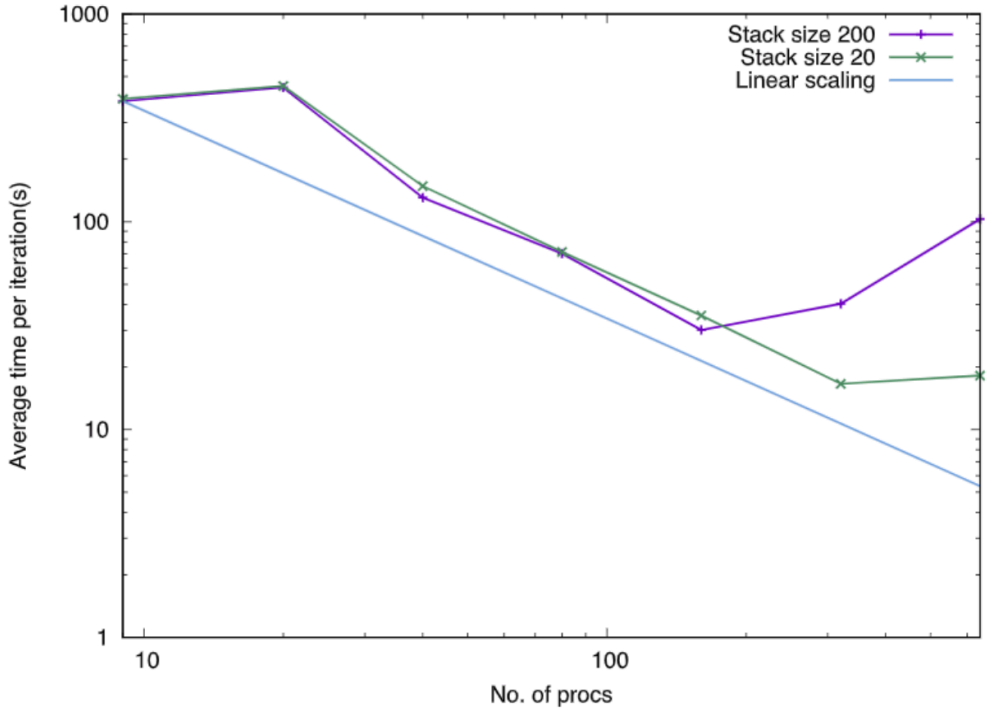

Hybrid 2x10
MPI
Amdahl scaling
Speed up
No. of procs
0
20
40
60
80
100
120
140
160
40
80
120
160
200
240
280
320
360
400
440
480
520
560
600
640
3D Disc thermal equilibrium

Comparison of single node performance for OpenMP, MPI and hybrid configurations. The OpenMP time is taken from the fastest 20-core run. Hybrid (2 × 10) uses 2 MPI processes and 10 OpenMP threads per MPI process. Hybrid (4 × 5) uses 4 MPI processes and 5 OpenMP threads per MPI process. The memory use is the maximum resident size reported by the GNU time command.

Configuration	Memory use (GB)	Time (s)	Normalised time
OpenMP	1.73	179.9	1.59
MPI	34.9	112.8	1
Hybrid (2 × 10)	3.47	120.8	1.07
Hybrid (4 × 5)	6.92	122.5	1.09

Table 8
Timings for a radiation hydrodynamics calculation of a D-type ionisation front expansion. The average time is the time taken for all iterations of the photoionisation loop excluding the first iteration. The fastest run for a given N_{stack} is shown in bold.

Hydro procs	Balancing procs	Total procs	No. of nodes	N_{stack}	Av. time (s)
8	0	9	1	200	380.2
8	11	20	1	200	442.9
8	31	40	2	200	130.8
8	71	80	4	200	70.3
8	**151**	**160**	**8**	**200**	**30.2**
8	311	320	16	200	40.3
8	631	640	32	200	103.1
8	0	9	1	20	390.2
8	11	20	1	20	449.8
8	31	40	2	20	148.5
8	71	80	4	20	71.6
8	151	160	8	20	35.4
8	**311**	**320**	**16**	**20**	**16.6**
8	631	640	32	20	18.2



Exercises

- The source code for the serial, openmp, mpi, and hybrid versions are on google drive directory linked to from summer school schedule
- There is a readme file in there to tell you how to compile and run the code
- You will need gfortran and an MPI implementation such as Open MPI (www.open-mpi.org)
- Try running the codes with different thread numbers and look at the speed up. Also try uncommenting the OMP ATOMIC code in the openmp version and see how this affects the speed up…