21st Summer School of PARALLEL COMPUTING

July 2 - 13, 2012 (Italian)
September 3 - 14, 2012 (English)

Hybrid programming: an example

Carlo Cavazzoni

CINECA

Outline

- Motivation
- Application test case: QuantumESPRESSO
- Mixed paradigm and FFT
- ScalaPACK and Multithread library
- Typical loops
- Perspective and Conclusions

Dennard scaling law

new gen.

old gen.

$L' = L / 2$

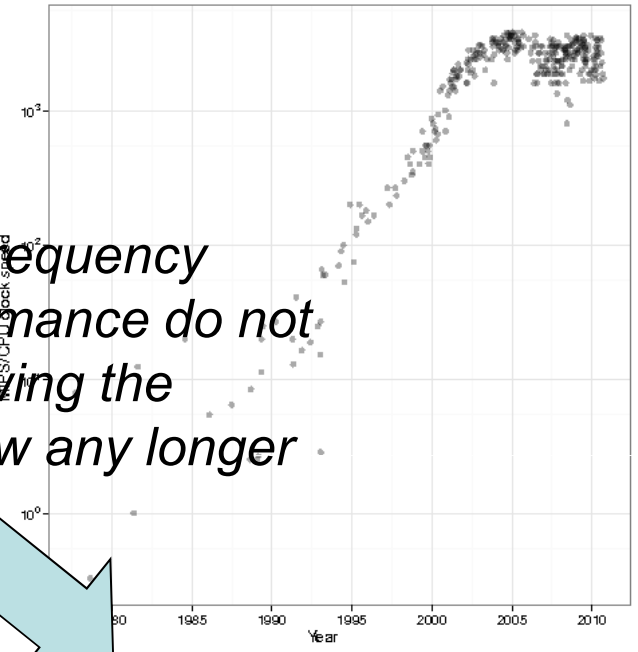
$V' = V / 2$

$F' = F * 2$

$D' = 1 / L^2 = 4D$

$P' = P$

do not hold anymore!



The core frequency and performance do not grow following the Moore's law any longer

$L' = L / 2$

$V' = \sim V$

$F' = \sim F * 2$

$D' = 1 / L^2 = 4 * D$

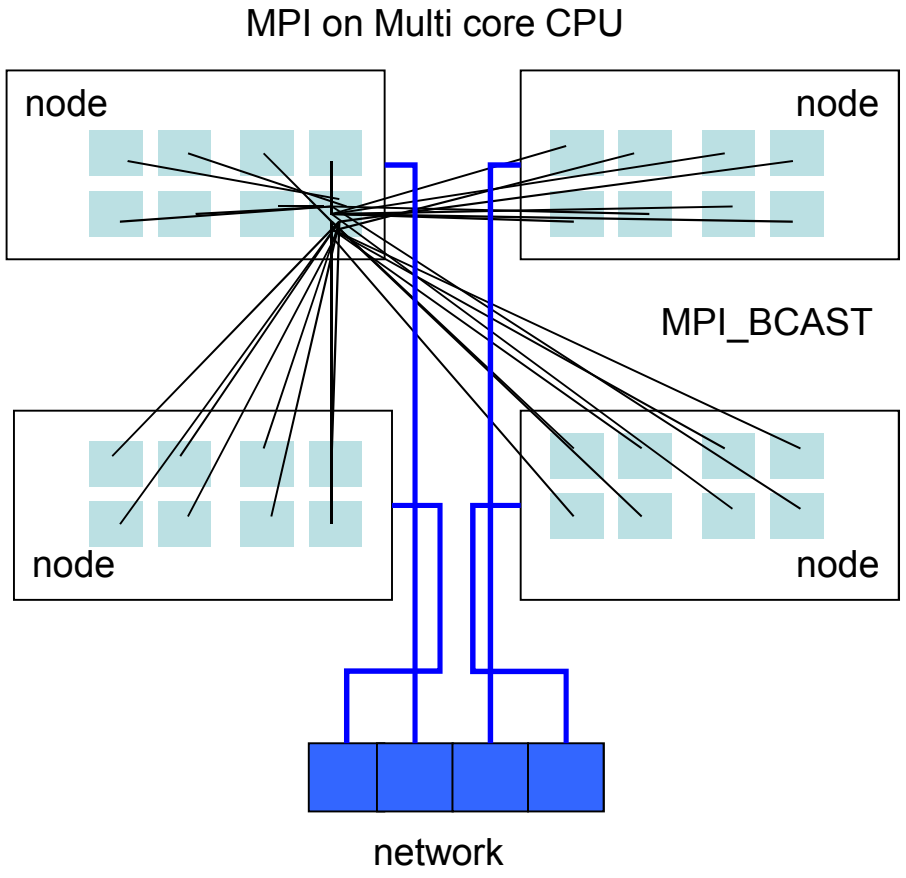
$P' = 4 * P$

The power crisis!

Increase the number of cores to maintain the architectures evolution on the Moore's law

Programming crisis!

MPI inter process communications



1 MPI proces / core
Stress network
Stress OS

Many MPI codes (QE) based on ALLTOALL
Messages = processes * processes

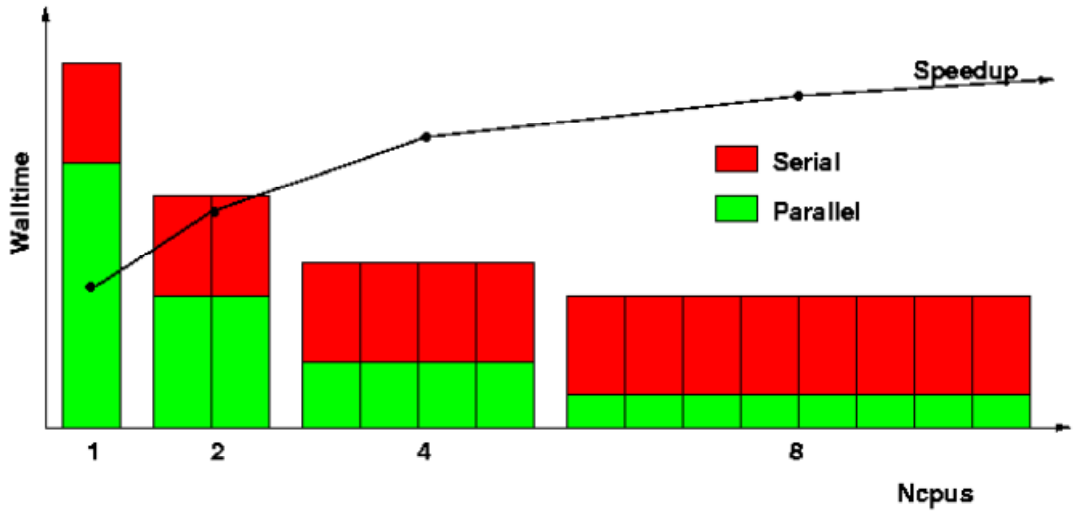
We need to exploit the hierarchy

Re-design applications

Mix message passing And multi-threading

What about Applications?

In a massively parallel context, an upper limit for the scalability of parallel applications is determined by the fraction of the overall execution time spent in non-scalable operations (Amdahl's law).



maximum speedup tends to
$1 / (1 - P)$
P= parallel fraction

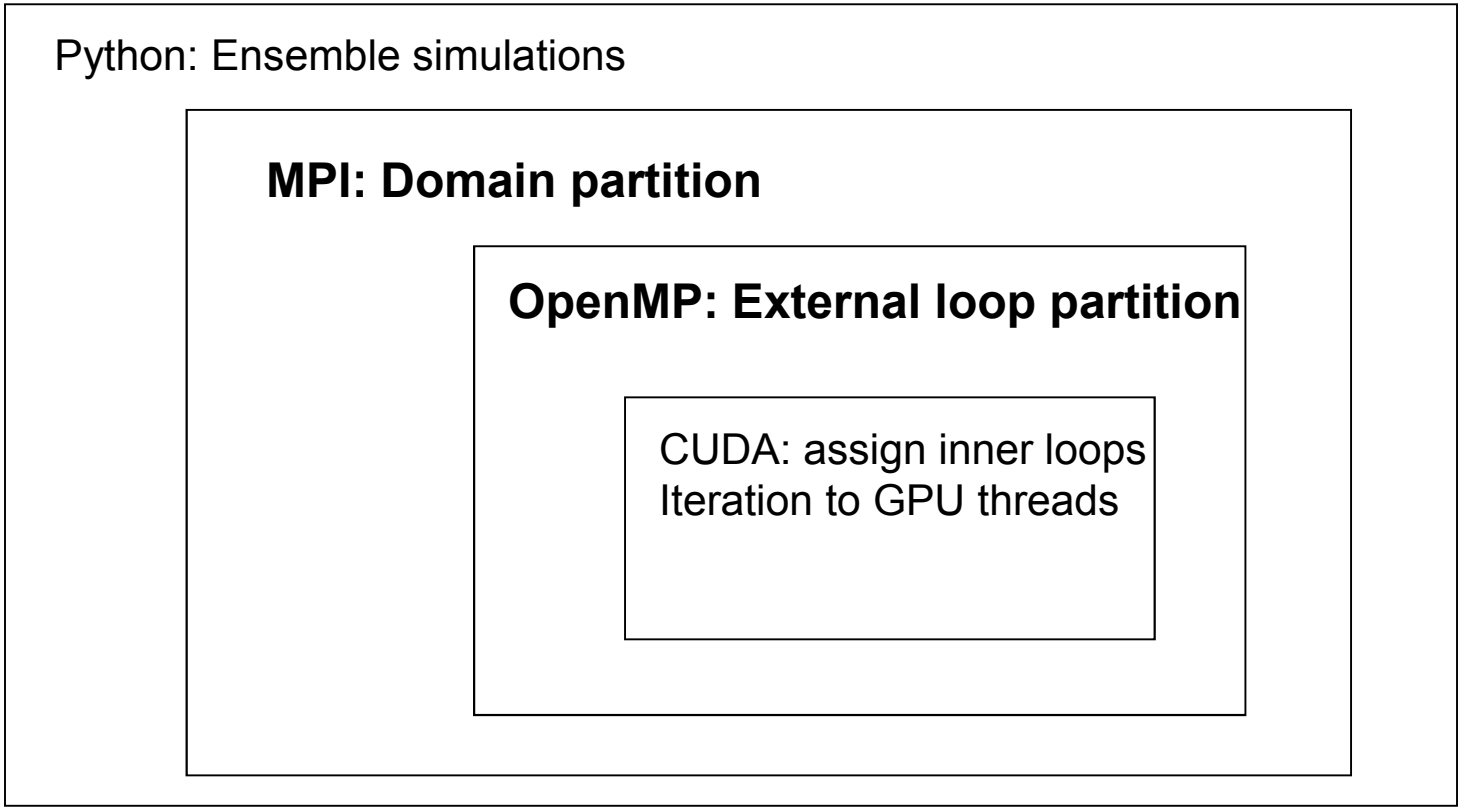
1000000 core

$P = 0.999999$

serial fraction= 0.000001

FERMI (BGQ) total concurrency: 655360 (65536/rack)

Use Hierarchy & Hybrid Programming



MPI + OpenMP

MPI (inter node)

Distributed memory systems
message passing
data distribution model
Version 2.1 (09/08)
API for C/C++ and Fortran

OpenMP (intra node)

Shared memory systems
Threads creations
relaxed-consistency model
Version 3.0 (05/08)
Compiler directive C and Fortran

Mixed Paradigm

PROS:

- Better use of memory hierarchy
- Better use of interconnection
- Improve scalability

CONS:

- Overhead in thread management
- Greater attention to memory access
- Worse performances

Quantum ESPRESSO is an open-source suite of computer codes for electronic-structure calculations and materials modeling. It is based on density-functional theory, plane waves, and pseudopotentials.

P. Giannozzi, et al J.Phys.:Condens.Matter, 21, 395502 (2009) http://dx.doi.org/10.1088/0953-8984/21/39/395502 .

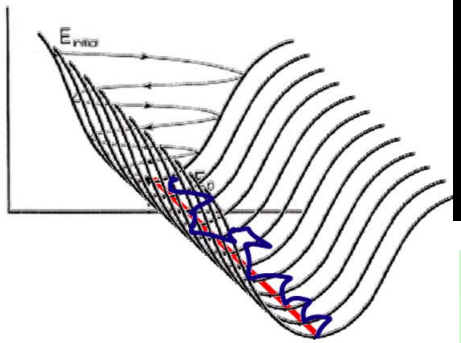
www.quantum-espresso.org

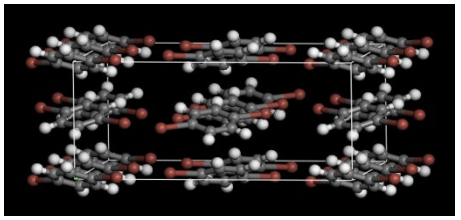
www.qe-forge.org

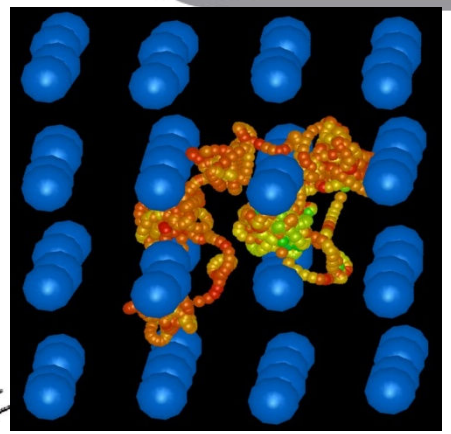
CP (Car-Parrinello)

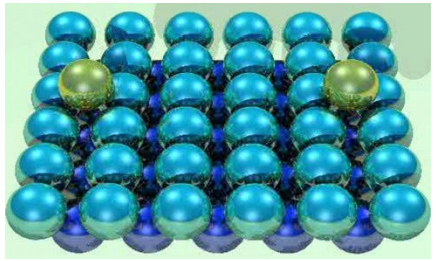
Code for Car-Parrinello MD
Disordered systems
Liquids
Finite temperature

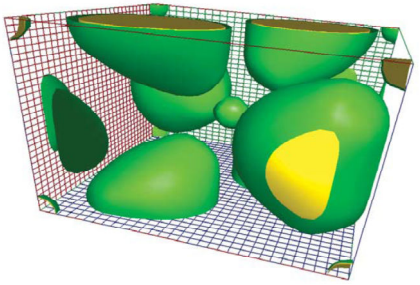
!!! Roberto Car & Michele Parrinello develop the CP method on a CRAY machine Installed at CINECA !!!







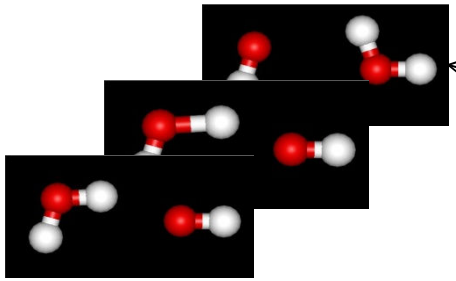




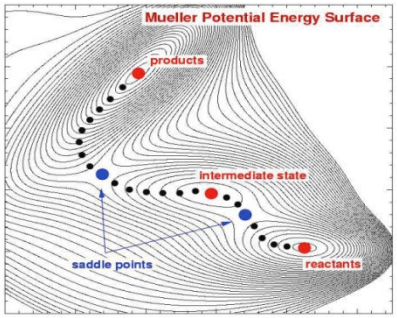
More scalable

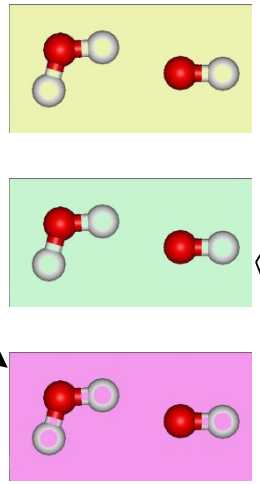
Less scalable

Parallelization (before OpenMP)



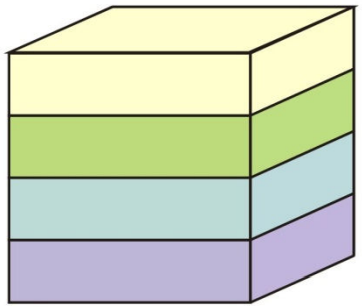
Parallelization over Images
Energy barrier evaluation





Parallelization over Pools:
K-points sempling of
Brillouin Zone.
Electronic band structure

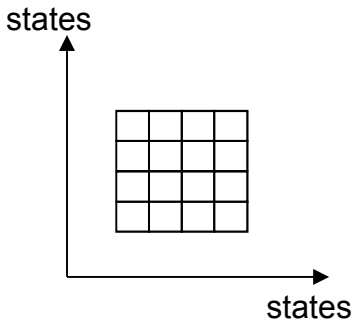
1x1xN proc grid



Charge density

Parallelization of the FFT
Real and Reciprocal Space
Decomposition.
One FFT for each electron

√N x √N proc grid



Hamiltonian

Parallelization of
Bands structure.
Matrix Diagonalization
At least one state for
each electron

Loosely coupled ⟷ **Tightly coupled**

TABLE I: Summary of parallelization levels in QUANTUM ESPRESSO.

group	distributed quantities	communications	performance
image	NEB images	very low	linear CPU scaling, fair to good load balancing; does not distribute RAM
pool	**k**-points	low	almost linear CPU scaling, fair to good load balancing; does not distribute RAM
plane-wave	plane waves, **G**-vector coefficients, **R**-space FFT arrays	high	good CPU scaling, good load balancing, distributes most RAM
task	FFT on electron states	high	improves load balancing
linear algebra	subspace Hamiltonians and constraints matrices	very high	improves scaling, distributes more RAM

Main Algorithms in QE

3D FFT

Linear Algebra

- Matrix Matrix Multiplication
- less Matrix-Vector and Vector-Vector
- Eigenvalues and Eigenvectors computation

Space integrals

Point function evaluations

Parallelization Strategy

3D FFT	ad hoc MPI & OpenMP driver
Linear Algebra	ScalaPACK + blas multithread
Space integrals	MPI & OpenMP loops parallelization and reduction
Point function evaluations	MPI & OpenMP loops parallelization

Mixed QE: Implicit *vs* Explicit approach

Implicit

Linking multi-threading libraries

No control of thread creation overhead

Relatively simple

Explicit

OpenMP syntax knowledge

Manage code flow and thread

More efficient

In QE we use both multi-thread parallelization

Multithread libraries (mkl, acml, essl)

No explicit OpenMP directive

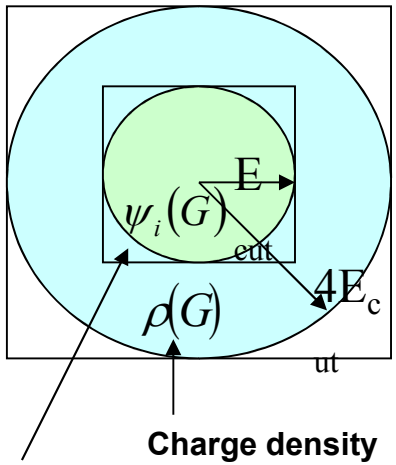
Very efficient (in most cases)

No possibility to mix multithread and non multithread version of the same library (selective call)

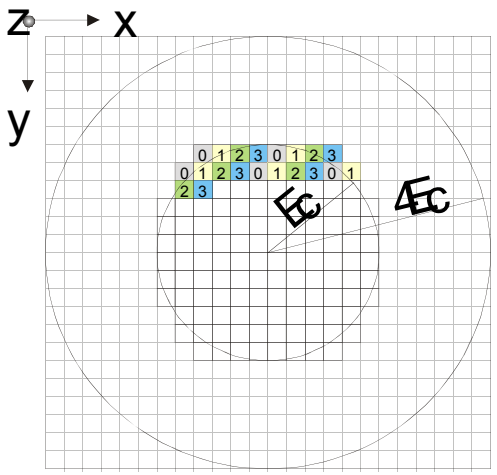
Workaround using **omp_set_num_threads() omp_get_max_threads()** not always possible

Understanding QE 3DFFT, Parallelization of Plane Wave

Reciprocal Space
G ⇔Plane Wave vectors

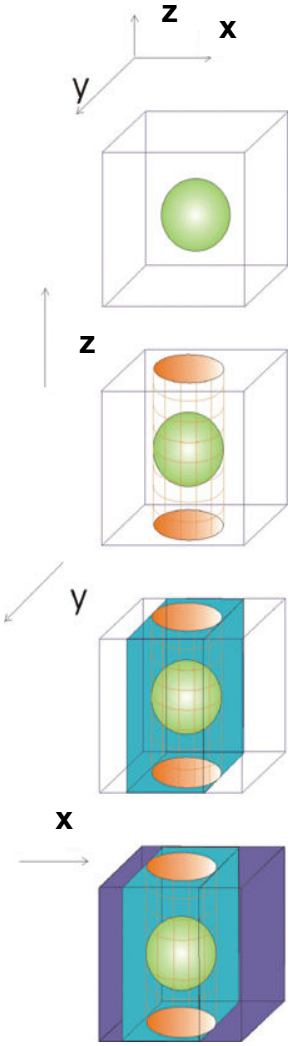


Single state electronic wave function



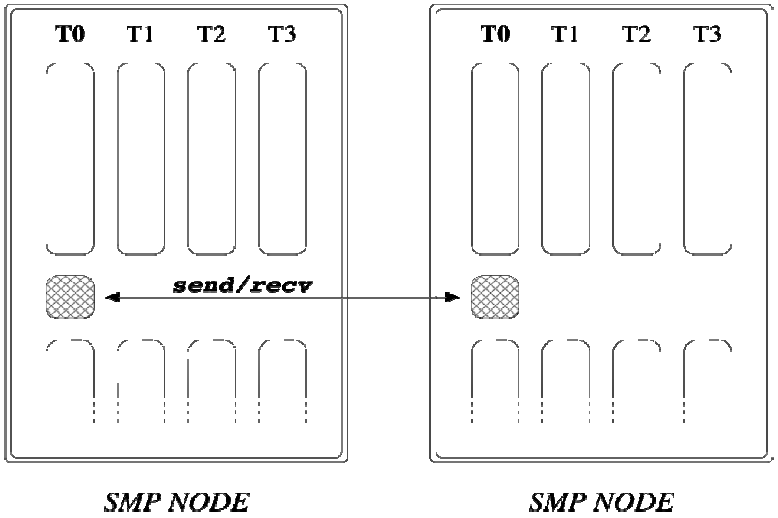
Column
0 PE 0
1 PE 1
2 PE 2
3 PE 3

~ Nx Ny / 5 FFT along z



Similar 3DFFT are present in most ab-initio codes like CPMD

Understanding QE 3DFFT: *master-only*




```
# begin OpenMP region
   do i = 1, nsl    in parallel
      call 1D-FFT along z ( f[offset] )
   end do
# end OpenMP region

   call fw_scatter( ... )

# begin OpenMP region
   do i = 1, nzl    in parallel
      do j = 1, Nx
            if ( dofft[j] ) then
               call 1D-FFT along y  ( f[offset] )
      end do
      call 1D-FFT along x ( f[offset] )  Ny-times
   end do
# end OpenMP region
```

Local (to each MPI process) number of "z" stick distributed to threads

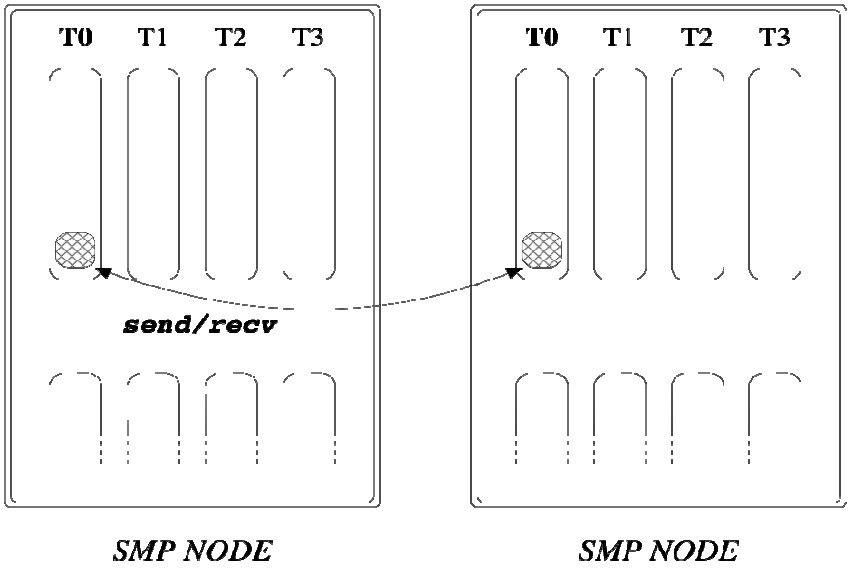
Thread private

Parallel transpose (implemented with isend/irecv)

Local (to each MPI process) number of "xy" plane Distributed to threads

Thread private

QE 3DFFT: *funneled*

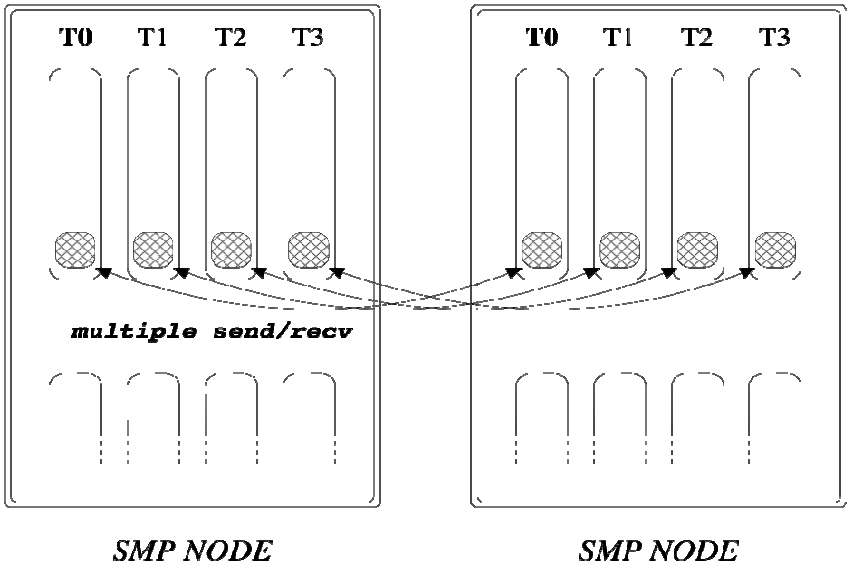



```
# begin OpenMP region
   do i = 1, nsl    in parallel
      call 1D-FFT along z ( f[offset] )
   end do

# begin of OpenMP MASTER section
   call fw_scatter( ... )
# end of OpenMP MASTER section
# force synchronization with OpenMP barrier

   do i = 1, nzl    in parallel
      do j = 1, Nx
            if ( dofft[j] ) then
               call 1D-FFT along y ( f[offset] )
      end do
      call 1D-FFT along x ( f[offset] )  Ny-times
   end do
# end OpenMP region
```

QE 3DFFT: *multiple*



Pros:

- Overlap communication and computation
- Less synchronizations between threads and memory flush

- Cons:
  - Do not exploit memory and network hierarchy
  - Stress the network like plain MPI

Not yet completed…

Space Integrals: Electrostatic potential
simple loop parallelization

```
!$omp parallel do default(shared), private(rp,is,rhet,rhog,fpibg), reduction(+:eh,ehte,ehti)
      DO ig = gstart, ngm
        rp   = (0.D0,0.D0)
        DO is = 1, nsp
          rp = rp + sfac( ig, is ) * rhops( ig, is )
        END DO
        rhet  = rhoeg( ig )
        rhog  = rhet + rp
        IF( tscreen ) THEN
          fpibg     = fpi / ( g(ig) * tpiba2 ) + screen_coul(ig)
        ELSE
          fpibg     = fpi / ( g(ig) * tpiba2 )
        END IF
        vloc(ig) = vloc(ig)   +  fpibg *         rhog
        eh       = eh         +  fpibg *         rhog * CONJG(rhog)
        ehte     = ehte       +  fpibg *   DBLE(rhet * CONJG(rhet))
        ehti     = ehti       +  fpibg *   DBLE(  rp * CONJG(rp))
      END DO
```

Ionic density (reciprocal space)

Electronic density (reciprocal space)

Hartree potential

Hatree Energy

Ionic contribution

Electronic contribution

! IBM xlf compiler does not like name of function used for OpenMP reduction

Space Integral: Non local energy
less simple loop parallelization

Larger parallel region to reduce
The fork/join overhead

```
!$omp parallel default(shared), &
!$omp private(is,iv,ijv,isa,ism,ia,inl,jnl,sums,i,iss,sumt), reduction(+:ennl)
      do is = 1, nsp
         do iv = 1, nh(is)
            do jv = iv, nh(is)
               ijv = (jv-1)*jv/2 + iv
               isa = 0
               do ism = 1, is - 1
                  isa = isa + na(ism)
               end do
!$omp do
               ...
!$omp end do
            end do
         end do
      end do
!$omp end parallel
```

```
!$omp do
      do ia = 1, na(is)
         inl = ish(is)+(iv-1)*na(is)+ia
         jnl = ish(is)+(jv-1)*na(is)+ia
         isa = isa+1
         sums = 0.d0
         do i = 1, n
            iss = ispin(i)
            sums(iss) = sums(iss) + f(i) * bec(inl,i) * bec(jnl,i)
         end do
         sumt = 0.d0
         do iss = 1, nspin
            rhovan( ijv, isa, iss ) = sums( iss )
            sumt = sumt + sums( iss )
         end do
         if( iv .ne. jv ) sumt = 2.d0 * sumt
         ennl = ennl + sumt * dvan( jv, iv, is)
      end do
!$omp end do
```

Point Function evaluation: Exchange and Correlation energy

```
!$omp parallel do private( rhox, arhox, ex, ec, vx, vc ), reduction(+:etxc)
     do ir = 1, nnr
        rhox = rhor (ir, nspin)
        arhox = abs (rhox)
        if (arhox.gt.1.d-30) then
           CALL xc( arhox, ex, ec, vx(1), vc(1) )
           v(ir,nspin) = e2 * (vx(1) + vc(1) )
           etxc = etxc + e2 * (ex + ec) * rhox
        endif
     enddo
!$omp end parallel do
```

Real space electronic charge density

XC functional (external subroutine)

XC Potential

XC Energy

Gram-Schmidt kernel:
dealing with thread private allocatable array

```
!$omp parallel default(shared), private( temp, k, ig )

      ALLOCATE( temp( ngw ) )

!$omp do
      DO k = 1, kmax
         csc(k) = 0.0d0
         IF ( ispin(i) .EQ. ispin(k) ) THEN
            DO ig = 1, ngw
               temp(ig) = cp(1,ig,k) * cp(1,ig,i) + cp(2,ig,k) * cp(2,ig,i)
            END DO
            csc(k) = 2.0d0 * SUM(temp)
            IF (gstart == 2) csc(k) = csc(k) - temp(1)
         ENDIF
      END DO
!$omp end do

      DEALLOCATE( temp )

!$omp end parallel
```

Example of non trivial loop parallelization

```
 DO nt = 1, ntyp
     IF ( upf(nt)%tvanp ) THEN         !
        DO ih = 1, nh(nt)            !
           DO jh = ih, nh(nt)!
              CALL qvan2( ngm, ih, jh, nt, qmod, qgm, ylmk0 )              !
!$omp parallel default(shared), private(na,qgm_na,is,dtmp,ig,mytid,ntids)
#ifdef __OPENMP
              mytid = omp_get_thread_num()
              ntids = omp_get_num_threads()
#endif
              ALLOCATE(  qgm_na( ngm ) )                !
              DO na = 1, nat                    !
#ifdef __OPENMP
                IF( MOD( na, ntids ) /= mytid ) CYCLE
#endif
                IF ( ityp(na) == nt ) THEN
                   qgm_na(1:ngm) = qgm(1:ngm)*eigts1(ig1(1:ngm),na)*eigts2(ig2(1:ngm),na)*eigts3(ig3(1:ngm),na)
                   DO is = 1, nspin_mag
                       dtmp = 0.0d0
                       DO ig = 1, ngm
                          dtmp = dtmp + aux( ig, is ) * CONJG( qgm_na( ig ) )
                       END DO
                       deeq(ih,jh,na,is) = fact * omega * DBLE( dtmp )
                       deeq(jh,ih,na,is) = deeq(ih,jh,na,is)
                    END DO
                 END IF
              END DO
              DEALLOCATE( qgm_na )
!$omp end parallel
           END DO
        END DO
     END IF
  END DO
```

take the thread ID

take the number of threads

distribute atoms round-robin to threads

Conclusion

Number of cores double every two years

MPI and OpenMP exploit multi-core nodes

Both implicit and explicit multi-threading

Mixed paradigm: not big effort, good compilers and libraries

Next challenge: beyond Parallel Computing (reliable, hybrid)