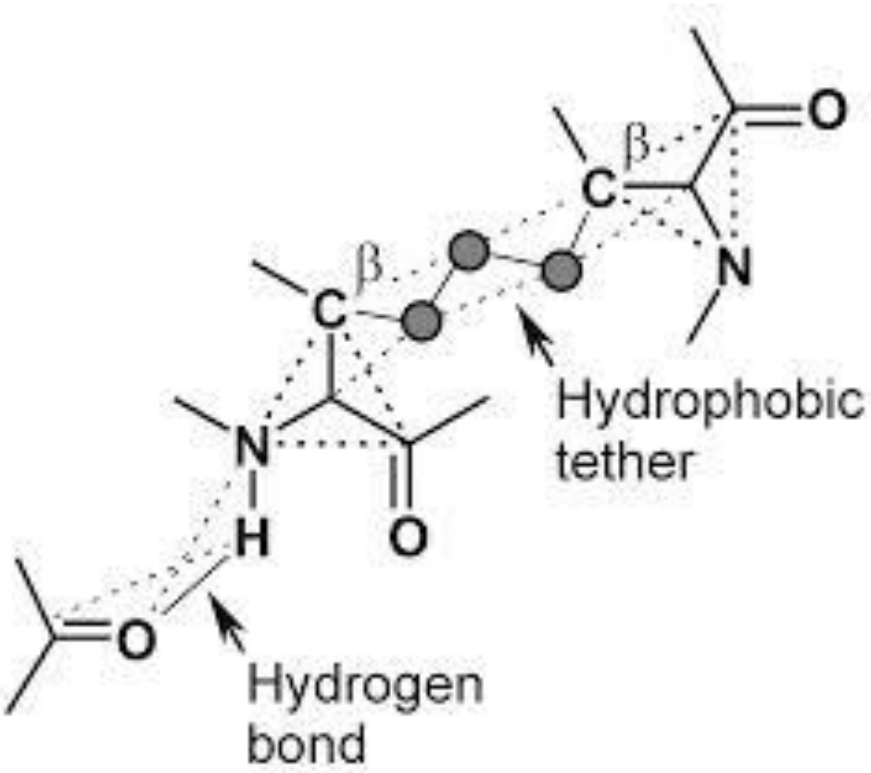
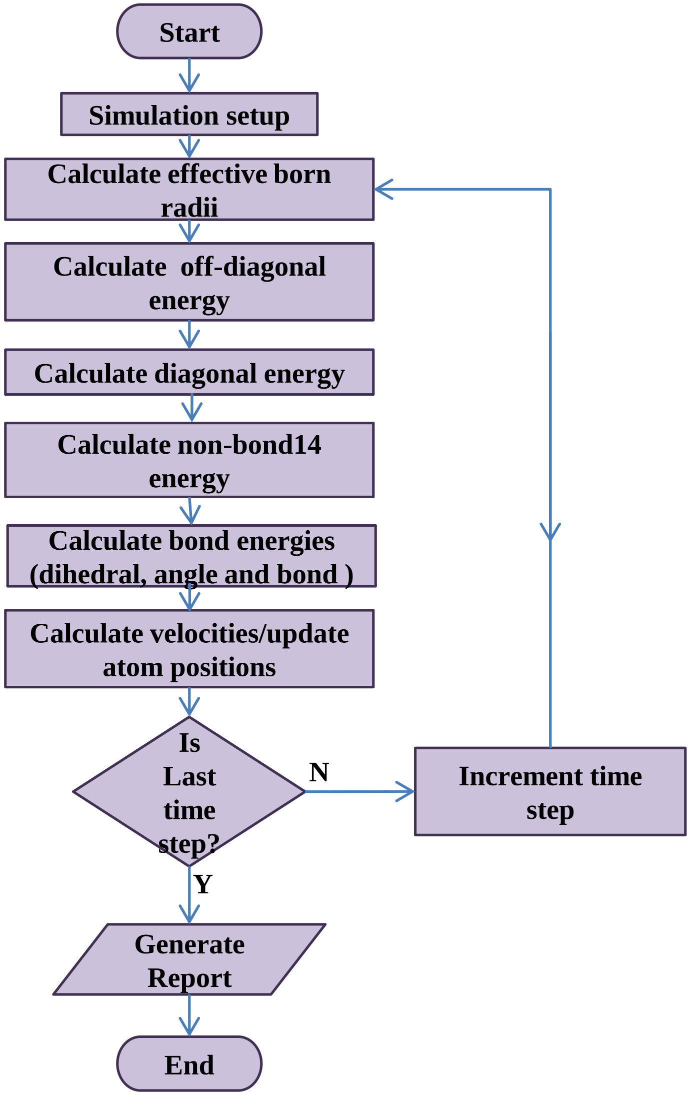
TATA CONSULTANCY SERVICES
Experience certainty.

AMBER – optimizations on Xeon Phi

Manoj Nambiar
Principal Scientist, Innovations Lab – Performance Engg

Amber : Molecular Dynamics Simulation Package





Start

Simulation setup

Calculate effective born radii

Calculate off-diagonal energy

Calculate diagonal energy

Calculate non-bond14 energy

Calculate bond energies (dihedral, angle and bond)

Calculate velocities/update atom positions

Is Last time step?

N

Increment time step

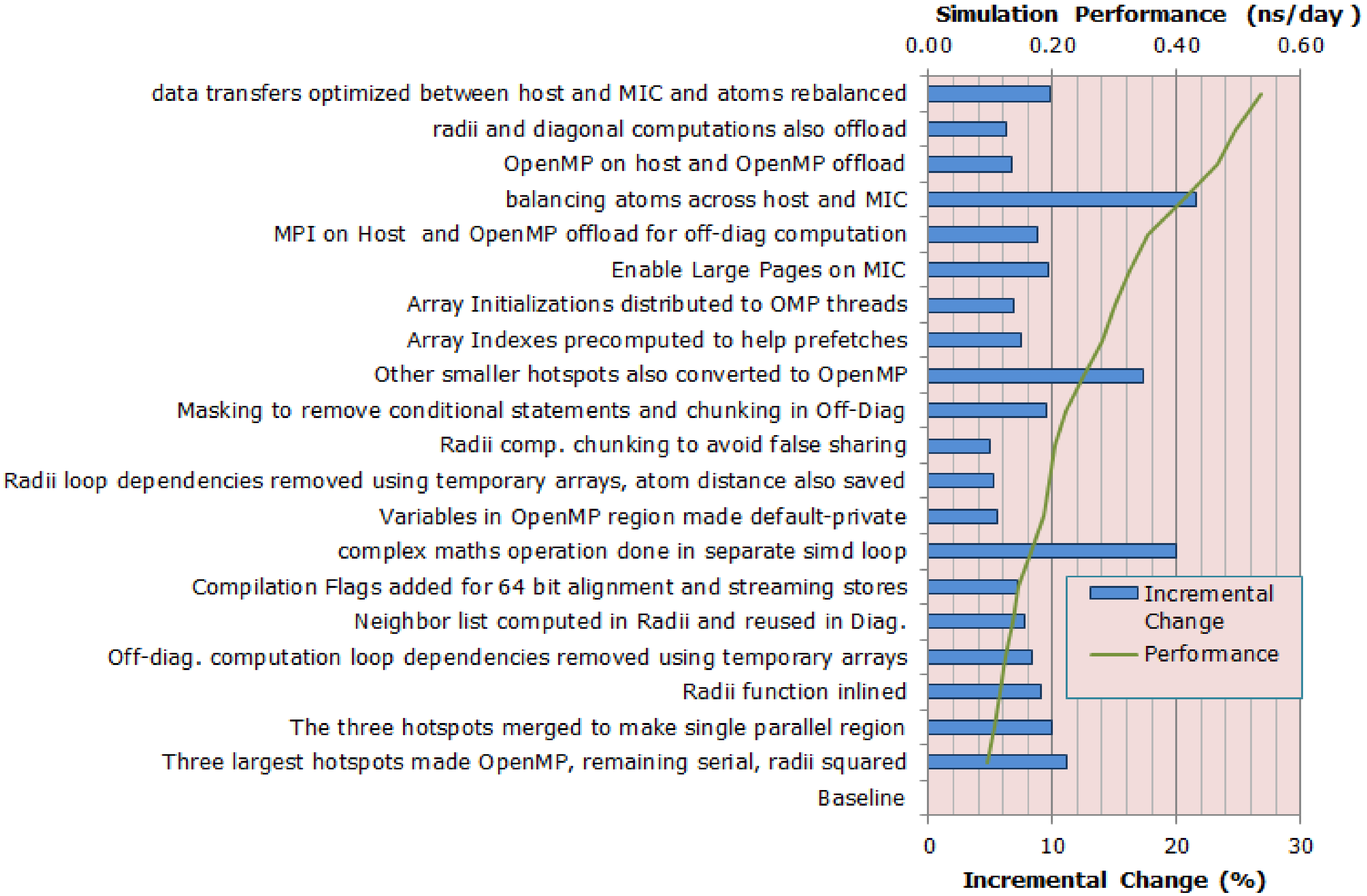
Y

Generate Report

End

Generalized Born Model - Algorithm Flow

Optimization Journey



Optimizations: Conversion from MPI to OpenMP

- In MPI, data is private to each MPI rank (more ranks means more copies of data)

- In OpenMP, data can be shared or private

- Even if 2 MPI ranks work on different parts of same array, copies need to be made. The merging of individual copies with each rank requires inter-process communication which is expensive.

- If an OpenMP thread requires data that has already been read by another thread, there is possibility of getting the data from cache

Results varied from hotspot to hotspot

- off-diagonal time reduced by 33%

- radii computation time doubled

But overall there was a gain by 10%

Radii inlined and three parallel region merged

Energy Calc after converting to OpenMP

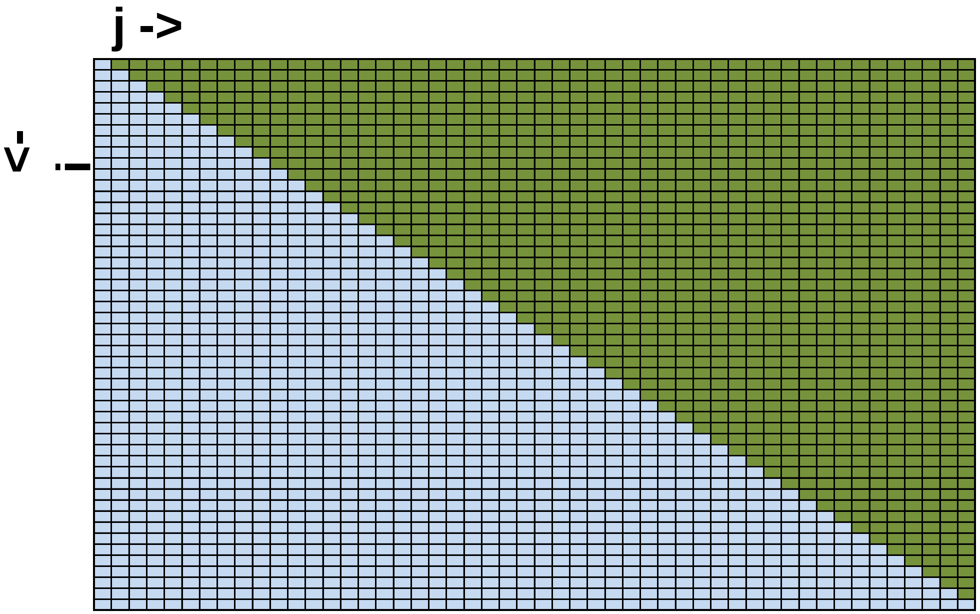
```
gb_energy ()
      call radii_calc() // radii function

     // ---off-diagonal calculation---
     !$omp parallel
     !$omp do
     do i =1 , atm_cnt
         .
     end do
     !$omp end do
     !$omp end parallel

     // ---diagonal calculation ---
     !$omp parallel
     !$omp do
     do i =1 , atm_cnt
        .
      end do
     !$omp end do
     !$omp end parallel
end gb_energy()

radii_calc()
  !$omp parallel
  !$omp do
   do i =1 , atm_cnt
      .
   end do
  !$omp end do
  !$omp end parallel
end radii_calc
```

Inlining of radii and single parallel region

```
gb_energy ()

 omp parallel

 //radii calculation
  !omp do
   do i =1 , atm_cnt
      .
   end do
  !$omp end do


//off-diagonal calculation
    !$omp do
    do i =1 , atm_cnt
       .
    end do
     !$omp end do

    //diagonal calculation
     !$omp do
    do i =1 , atm_cnt
      .
     end do
     !$om p end do
!$omp end parallel

end gb_energy()
```

Shared/private variables need to be declared only once here

Optimizations : Radii calculation loop



When computing j^{th} atoms effect on i^{th} atom, the reverse effect is also calculated

This makes the loop count $N^2/2$ instead of N^2

However since different threads have partially calculated values, reduction is required.

Every atom has to interact with every other atom.

Sum of all interactions contribute to the overall energy

Optimizations : Removing Reduction from radii

Original Code structure

```
!$omp do reduction(reff)
do i =1 , atm_cnt
//create neighbor list
        do j = i+1 , atm_cnt //create neighbor list
           if(condn) then
              jj(icount ) = j
              count = iocount = iocount + 1
           end if
        end do

//gather data for maths operations
         do k = 1 , iocount
             vectmp = f(i,j)
         end do
//the maths operations

//effective radii computation loop
        do k = 1 , iocount

                -
             reff(i) = reff(i) + formula1(i,j)
             reff(j) = reff(j) + formula2(i,j)
        end do
end do
!$omp end do
```

Removal of Reduction from radii

```
!$omp do
do i =1 , atm_cnt
//create neighbor list
        do j = 1 , atm_cnt //create neighbor list
           if(condn) then
              jj(icount ) = j
              count = iocount = iocount + 1
           end if
        end do

//gather data for maths operations
         do k = 1 , iocount
             vectmp = f(i,j)
         end do
//the maths operations

//effective radii computation loop
        do k = 1 , iocount

                -
             reff(i) = reff(i) + formula1(i,j)

        end do
end do
!$omp end do
```

30% reduction in radii computation time

Handling of reduction arrays

Original Code

```
!$omp do reduction (+:energy)
do i =  1 , atm_cnt
     do j = i + 1 , atm_cnt
          .
          .
          .
          energy(i) = energy(i) + formula1
          energy(j) = energy(j) + formula2
     end do
end do
$omp end do
```

Temporary array solution

```
effective radii computation loop

!dir$ omp do reduction (+:energy)
do i =  1 , atm_cnt
   do k = i+1, atm_cnt, 8
   !dir$ ivdep
       do  j = k,  k + 7
              .
              .
              .
           counter = j –k + 1
            tmp_engy_i(counter) =  formula1(i,j)
            tmp_engy_j(counter) =  formula2(i,j)
       end do
       do  j = k,  k + 7
            counter = j –k + 1
            energy(i) = energy(i) + tmp_engy_i(counter)
            energy(j) = energy(j) + tmp_engy_j(counter)
        end do
   end do
end do
$omp end do
```

SIMD reduction solution

```
!dir$ omp do reduction (+:energy)
do i =  1 , atm_cnt
     dir$ simd reduction (+energy)
      do j = i + 1 , atm_cnt
        .
        .
        .
         energy(i) = energy(i) + formula1(i,j)
         energy(j) = energy(j) + formula2(i,j)
      end do
end do
$omp end do
```

25% incremental reduction in off-diagonal computation time

omp do reduction() : creates private copy of variable for each thread and does reduction at the end of loop

simd reduction() : this is a pragma for vector reduction .

Precomputing array indexes

Index computed just before it is required

```
          //off-diag calculation
!$omp do reduction(+:energy)
do i =1 , atm_cnt
.
.
    omp do reduction
     do j = i + 1, atm_cnt
      .
     .
     .
     .
     .
     .
     .
    indx = ioc(i) + ioc(j)
    mydata = iac(indx)
    if(mydata  .eq. 0) then
     .
   end if
    .
    .
    .
    end do
.
.
.

end do
$omp  end do
```

Index computed early

```
//off-diag calculation
!$omp do reduction(+:energy)
do i =1 , atm_cnt
 .
.indx_i = ioc(i)
    omp do reduction
     do j = i + 1, atm_cnt
         indx = indx_i + ioc(j)
        .
       .
       .
       .
       .
       .
        mydata = iac(indx)
        if(mydata  .eq. 0) then
        .
      end if
        .
        .
       .
       .
     end do
 .
 .
 .
end do
$omp end do
```

Compiler inserts prefetch instructions for mydata here

mydata required here

7% incremental reduction in overall time

Using masking variables

Without Masking

```
//off-diagonal calculation
!dir$ simd reduction(energy)
     do j =i +1, atm_cnt.
        .
        .
        if (condition) then
              mydata = ComplexEquation1
        else
              mydata = ComplexEquation2
        end if
        .
       .
       .
       .
        end do
```

With Masking

```
   if(condition)
            C1 = 1
            C2 = 0
   else
           C1 = 0
           C2 = 1
   end if

//off-diagonal calculation
!dir$ simd reduction(energy)
     do j =i +1, atm_cnt.
        .
        .
         mydata =  C1* ComplexEquation1 + C2 * ComplexEquation2
.
       .
       .
       .
        end do
```

9% incremental reduction in overall time

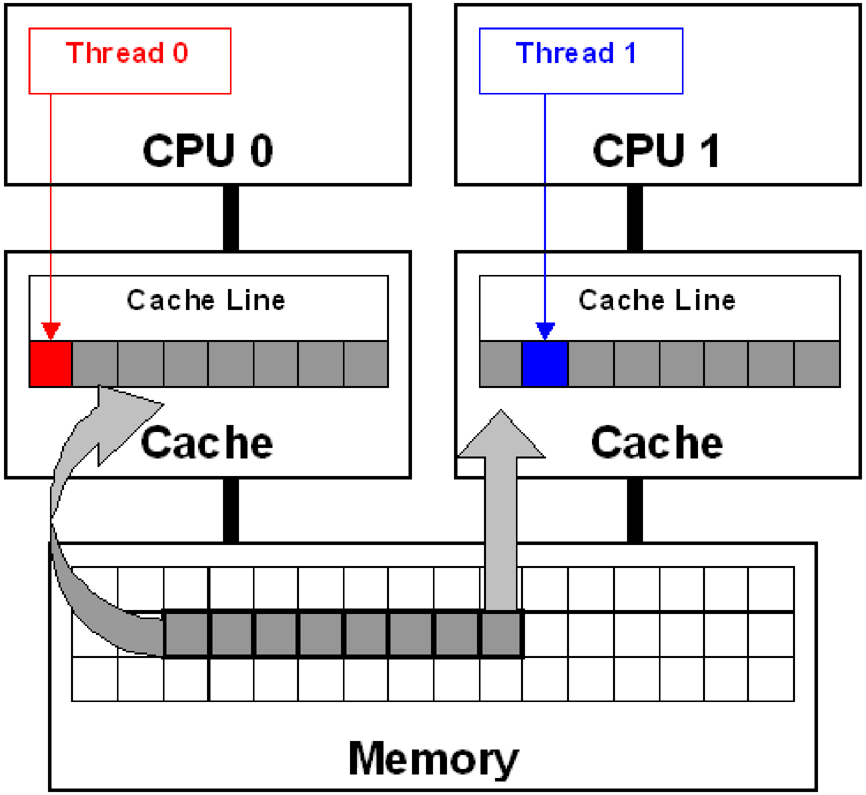
Optimizations : Chunk to prevent false Sharing

ORIGINAL

```
!$omp do
do i =1 , atm_cnt
//create neighbor list
   do j = 1 , atm_cnt
   if(condn) then
     jj(icount ) = j
      count = iocount = iocount + 1
    end if
    end do
.
end  do
!$ omp end do
```

CHANGED CODE

```
!$omp do
do outer_i = 1, atm_cnt, 8
   max_i = MIN(atm_cnt,outer_i + 7)
   do i = outer_i, max_i
//create neighbor list
      do j = 1 , atm_cnt
.
.

.
end  do
!$ omp end do
```



- **Chunk of 8 atoms given to one thread at a time to avoid false sharing**

Summary

- Openmp scaled better than MPI
  - Reduces communication requirements.
- Collapsing openmp regions helped gain performance
- Avoiding Openmp reduction by doubling the computation helps
  - In case where computations are comparable to synchronization overhead
- Use of $dir simd reduction helps enforce vectorization easier
- PreComputing array indexes helps
  - When access pattern is not sequential
- Avoidance of false cache sharing

- Symmetric mode
  - Important to balance workload between Xeon and Xeon Phi for optimal performance

- Higher the problem size better the performance

Overall 5X increase in performance from parallel baseline

TATA CONSULTANCY SERVICES
Experience certainty.

TATA

Thank You

Manoj Nambiar – m.nambiar@tcs.com
Nishant Agrawal – nishant.agrawal@tcs.com