Molecular Dynamics

Stanford Streaming Computing

Eric Darve
Domain Specific Language

Characteristics of domain specific language:

1. Easy to use for end-user.
   - Machine independent.
   - Complex optimizations & communication protocols hidden.
   - Intuitive interface.

2. Easy to optimize for compiler.
   - Constrain end-user to use given interface.
   - Expose data dependence and parallelism.
Loops

• Adapted to sequential machines which execute one operation at a time.

• Parallel computing: critical information is data dependence.
  Loops imply that operations must be performed in specified order.
  Often not true.
How to perform operations on collections without loops?

Two types of construction:
1. Ordered collection: operations must be performed sequentially.
   Syntax: Iterate { }
2. Unordered collection: operation can be performed in any order.
   Syntax: For all { }
Communications

End-user:
• Knows about data dependence.
• Difficult to optimize communications.

Compiler:
• Knows how to optimize.
• Difficult to analyze data dependence.
Two types of operations:
1. Gather.
2. Scatter.

What should the domain specific language provide to the end-user?
1. Predefined high-level routines which encapsulates communication steps.
   Bill M. : « No user-specified communication »
   or
2. Open MP type of construction.
Molecular Dynamics

Structure of code:
1. Define initial configuration of system.
2. Advance in time:
   1. Given position, compute force.
   2. Given force, advance velocity in time.
   3. Given velocity, advance position in time.

Position and velocity are staggered in time.
Typical data structure

Record: water molecule.
Contains:
• Current position and velocity of three atoms \( \text{O} + \text{H} + \text{H} \).
  \( 3 \times (2 \times 3) = 18 \) floats.
• Values at previous time step: 18 floats.
• Force for each particle: \( 3 \times 3 = 9 \) floats.

Total = 45 floats / record
Computation of force

Two types of forces:

1. Bonded interactions:
   - Interaction list is predefined.
   - Each atom interacts with few other atoms: 4 to 32 atoms.

2. Non bonded interactions:
   - Interaction list is defined by pairs of water molecules closer than threshold.
   - List may change from one time step to the other.
   - Number of interactions is very large: each atom interacts with 400 atoms.
Bonded interactions

Data can be arranged so that most computation is local.

Bonded interactions are between atoms belonging to same molecule:
• Chain + water: few atoms / molecule. Entirely local.
• Solute (e.g. protein): if very large might be split between a few processors.
Bonded interaction record

Contains:

1. Type of interaction potential.
2. List of atoms for the bond.

1. User supplies routine to compute force:

   \[ \text{force record compute\_force(bonded interaction record &);} \]

2. Language implements mapping this kernel to each record.
Fortran code : torsion angle

Gather data for given torsion angle:

```
do 1190 itr = 1, nt
   s1x(itr,iat) = ra(phil(itr,iat),1,1)
   s1y(itr,iat) = ra(phil(itr,iat),2,1)
   s1z(itr,iat) = ra(phil(itr,iat),3,1)
1190 continue
```

Only input is particle position
Compute some cross products:
\[ v_{12}(itr,1) = r_{21}(itr,2)*r_{23}(itr,3) - r_{21}(itr,3)*r_{23}(itr,2) \]
\[ v_{12}(itr,2) = r_{21}(itr,3)*r_{23}(itr,1) - r_{21}(itr,1)*r_{23}(itr,3) \]
\[ v_{12}(itr,3) = r_{21}(itr,1)*r_{23}(itr,2) - r_{21}(itr,2)*r_{23}(itr,1) \]
\[ v_{32}(itr,1) = r_{43}(itr,2)*r_{23}(itr,3) - r_{43}(itr,3)*r_{23}(itr,2) \]
\[ v_{32}(itr,2) = r_{43}(itr,3)*r_{23}(itr,1) - r_{43}(itr,1)*r_{23}(itr,3) \]
\[ v_{32}(itr,3) = r_{43}(itr,1)*r_{23}(itr,2) - r_{43}(itr,2)*r_{23}(itr,1) \]

Dots products and square roots:
\[ v_{12d}(itr) = (v_{12}(itr,1)*v_{12}(itr,1) + v_{12}(itr,2)*v_{12}(itr,2) + v_{12}(itr,3)*v_{12}(itr,3)) \]
\[ v_{32d}(itr) = (v_{32}(itr,1)*v_{32}(itr,1) + v_{32}(itr,2)*v_{32}(itr,2) + v_{32}(itr,3)*v_{32}(itr,3)) \]
\[ r_{v12}(itr) = \sqrt{v_{12d}(itr)} \]
\[ r_{v32}(itr) = \sqrt{v_{32d}(itr)} \]
Intermediate values for the force

dcpdrs(itr,1,1) = r23(itr,2)*dcdr12(3)-r23(itr,3)*dcdr12(2)
dcpdrs(itr,2,1) = r23(itr,3)*dcdr12(1)-r23(itr,1)*dcdr12(3)
dcpdrs(itr,3,1) = r23(itr,1)*dcdr12(2)-r23(itr,2)*dcdr12(1)
dcpdrs(itr,1,4) = dcdr32(2)*r23(itr,3)-dcdr32(3)*r23(itr,2)
dcpdrs(itr,2,4) = dcdr32(3)*r23(itr,1)-dcdr32(1)*r23(itr,3)
dcpdrs(itr,3,4) = dcdr32(1)*r23(itr,2)-dcdr32(2)*r23(itr,1)
dcpdrs(itr,1,2) = r31(itr,2)*dcdr12(3)-r31(itr,3)*dcdr12(2)
dcpdrs(itr,2,2) = r31(itr,3)*dcdr12(1)-r31(itr,1)*dcdr12(3)
dcpdrs(itr,3,2) = r31(itr,1)*dcdr12(2)-r31(itr,2)*dcdr12(1)
dcpdrs(itr,1,3) = r12(itr,2)*dcdr12(3)-r12(itr,3)*dcdr12(2)
dcpdrs(itr,2,3) = r12(itr,3)*dcdr12(1)-r12(itr,1)*dcdr12(3)
dcpdrs(itr,3,3) = r12(itr,1)*dcdr12(2)-r12(itr,2)*dcdr12(1)
Final computation

\[ s1x(itr,1) = \text{dedct}(itr) \times \text{dcpdrs}(itr,1,1) \]
\[ s1y(itr,1) = \text{dedct}(itr) \times \text{dcpdrs}(itr,2,1) \]
\[ s1z(itr,1) = \text{dedct}(itr) \times \text{dcpdrs}(itr,3,1) \]
\[ s1x(itr,2) = \text{dedct}(itr) \times \text{dcpdrs}(itr,1,2) \]
\[ s1y(itr,2) = \text{dedct}(itr) \times \text{dcpdrs}(itr,2,2) \]
\[ s1z(itr,2) = \text{dedct}(itr) \times \text{dcpdrs}(itr,3,2) \]
\[ s1x(itr,3) = \text{dedct}(itr) \times \text{dcpdrs}(itr,1,3) \]
\[ s1y(itr,3) = \text{dedct}(itr) \times \text{dcpdrs}(itr,2,3) \]
\[ s1z(itr,3) = \text{dedct}(itr) \times \text{dcpdrs}(itr,3,3) \]
\[ s1x(itr,4) = \text{dedct}(itr) \times \text{dcpdrs}(itr,1,4) \]
\[ s1y(itr,4) = \text{dedct}(itr) \times \text{dcpdrs}(itr,2,4) \]
\[ s1z(itr,4) = \text{dedct}(itr) \times \text{dcpdrs}(itr,3,4) \]

...
Scatter data once force has been computed:

\[
\text{do 1620 itr = 1, nt} \\
\text{atptr = phil(itr,iat)} \\
\text{ra(atptr,1,5) = ra(atptr,1,5) + s1x(itr,iat)} \\
\text{ra(atptr,2,5) = ra(atptr,2,5) + s1y(itr,iat)} \\
\text{ra(atptr,3,5) = ra(atptr,3,5) + s1z(itr,iat)} \\
1620 \text{ continue}
\]
Non bonded forces

Transfer of data is much more complex.
Example: water molecules.
Three steps:
1. Gather data from other processors.
2. Compute forces for all pair of water molecules.
3. Scatter data to other processors.
Domain decomposition

• To minimize communication between processors, data is organized in cubic clusters.

• **Cutoff distance** is distance beyond which charged particles no longer interacts.

• Size of cluster usually taken greater than or equal to cutoff distance.

• Typical system size: $8 \times 8 \times 8 = 512$ clusters. About 17 molecules per cluster.

• In a given cluster, interaction with molecules in cluster + nearest neighbor clusters only.
Data structure for domain

Domain record:
• Domain number
• Position in space
• List of water molecule records.
Domain connectivity

- Must define list of nearest neighbors for each domain.
- With periodic boundary condition, domains far apart might be considered nearest neighbor.
- Load balancing strategy may lead to complex connectivity…
• User input: function

    boolean are_nearest_neighborDomains(domain record &,
    domain record &) {};

    Defines whether two domains are
    considered close or not

• Language implements physical distribution
  of domains to processors.
  Optimal distribution depends on parallel
  architecture, etc.
Assigning molecules

• Once clusters are assigned to processors, data can be distributed.

• Send information about water molecule collections to each cluster
• End-user: specifies rule of decomposition. User-defined function

    domain record & belongs_to(water molecule record & ) {}

    defines to which domain a water molecule belongs.

• Language provides:

    void domain_decompose(water molecule record *, domain record *);

    Perform domain decomposition and send water molecule records to processors.
Computational step

- End-user: specifies only how force is computed.
- Language: takes care of communication and synchronization of data.
- 2 data structures:
  1. domain record.
  2. List of domain records: nearest neighbor list.
End-user interface

• For all ‘water molecule’ record \( w_1 \) in domain record \( d_0 \) {
  
  For all domain \( d_1 \) in ‘nearest neighbor’ list {
    
    For all ‘water molecule’ record \( w_2 \) in domain record \( d_1 \) perform {
      compute_force(water molecule record & w_1, water molecule record & w_2) ;
    }
  }
}
• ‘For all’ structure: does not specify any order. Easier for compiler to optimize on hardware.

• How do we specify access to data?
Data access specification

Three types of data:
1. Read only: particle position.
2. Write only: no write data at this point.
3. Read/Write access: force.
   By nature, sequential operation rather than parallel.
   • Not convenient for compiler.
   • May result in loss of performance.
Example

• Processor 1 is working on record $w_1$ and $w_2$.
• Processor 2 starts with $w_1$ and $w_2$. But $w_1.force$ and $w_2.force$ are Read/Write accessed by Processor 1.
• Skip this one and goes to next one which is $w_1$ and $w_3$. Same problem…
• Until Processor 2 reaches $w_3$ and $w_4$ and then it starts computing.
Data access for computing forces

• Read: \texttt{w}_1.\texttt{coord} and \texttt{w}_2.\texttt{coord}.
  Read/Write: \texttt{w}_1.\texttt{force} and \texttt{w}_2.\texttt{force}.

• Access is actually very simple: the output of \texttt{compute force()} is simply added to \texttt{w}_1.\texttt{force} and \texttt{w}_2.\texttt{force}.

• In all cases, Read/Write access can be formulated as simple Add, Subtract, Multiply or Divide operation.
Example of OpenMP construct

!$omp parallel do
!$omp& default(shared)
!$omp& private(i,j,k,rij,d)

!$omp& reduction(+ : pot, kin)

do i=1,np
! compute potential energy and forces
    f(1:nd,i) = 0.0
    do j=1,np
        if (i .ne. j) then
            call dist(nd,box,pos(1,i),pos(1,j),rij,d)
            ! attribute half of the potential energy to particle 'j'
            pot = pot + 0.5*v(d)
            do k=1,nd
                f(k,i) = f(k,i) - rij(k)*dv(d)/d
            enddo
        endif
    enddo
enddo
! compute kinetic energy
kin = kin + dotr8(nd,vel(1,i),vel(1,i))
enddo
!$omp end parallel do
Fortran code: water-water

Gather coord of two water molecules.

Computation of cut-off coeff:

if (rsq .gt. wlcut) then
  drsq = rsq – wlcut
  de = drsq * wcuti
  de3 = de * de * de
  dsofp = ((DESS3*de+DESS2) * de + DESS1) * de3 * 2/ drsq
  sof = ((CESS3*de+CESS2) * de + CESS1) * de3+1
else
  dsofp = 0.
  sof = 1.
end if
• Fictitious-fictitious, hydrogen-hydrogen, fictitious-hydrogen, oxygen-oxygen interactions.

• Oxygen-Oxygen:

\[
\begin{align*}
ri &= 1 / rsq \\
r6 &= ri * ri * ri \\
disp &= -TIPB * r6 \\
rep &= TIPA * r6 * r6 \\
wnrg &= wnrg + coul1 + coul2 + coul3 + coul4 + coul5 + coul6 \\
&\quad + coul7 + coul8 + (rep + disp) \\
fco &= sofp * (12*rep + 6*disp) * ri - wnrg * dsofp \\
wnrg &= wnrg * sofp \\
tfxo &= tfxo - fco * delxo \\
tfyo &= tfyo - fco * delyo \\
tfzo &= tfzo - fco * delzo
\end{align*}
\]
Communications between processors

• Gather operation before the loops start.
• Scatter: add contribution to force from all other nearest neighbors domains.
• Can this communication step be handled efficiently by the compiler/language?
  – Efficiency: mask communication with computation?
  – Transparency: detail of hardware hidden from user.
Possible approach

User can provide the following input:

- which data is read or write.
- which data is read/write and requires reduce operation.
  \texttt{OpenMP : reduction(+ / - : force)} for example.
- Can the operation be predefined in language? (high level routines)
Conclusion

• Define minimal set of rules that user must conform to:
  – compiler / language must be able to optimize from there.
  – Sufficiently general that all algorithms in scientific computing can be addressed.
  – Sufficiently intuitive and simple that minimal knowledge and understanding is required from end-user.
Three language levels

• **Domain specific language** : data dependence defined by user.

• **Stream language** : communication and distribution of data among processors optimized at this level.

• **Hardware coding** : low level language.