

# Split Molecular Dynamics

---

**Aiichiro Nakano**

*Collaboratory for Advanced Computing & Simulations  
Department of Computer Science  
Department of Physics & Astronomy  
Department of Chemical Engineering & Materials Science  
Department of Quantitative & Computational Biology  
University of Southern California*

**Email: [anakano@usc.edu](mailto:anakano@usc.edu)**

***Goal:*** Learn MPI communicator concept using *in situ* data analysis of molecular dynamics simulation



# MPI\_Comm\_split()

---

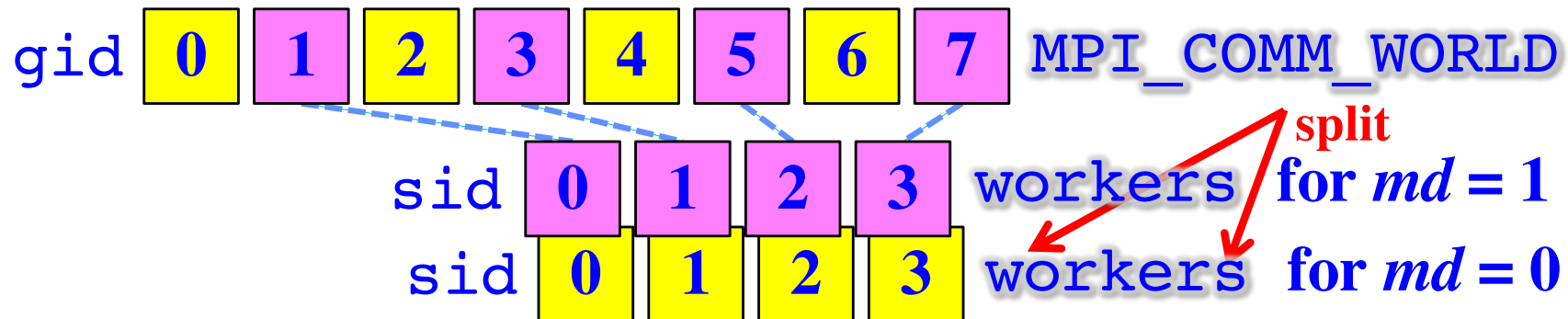
```
MPI_Comm mothercomm, daughtercomm;  
int color, key;  
MPI_Comm_split(mothercomm, color, key, &daughtercomm);
```

- **MPI\_Comm\_split()** subdivides a communicator, **mothercomm**, into a set of daughter communicators, where processes of the same **color** belong to the same daughter communicator. Processes within each **color** are ranked according to **key**, or if **key** is the same, according to the rank in **mothercomm**. It returns a pointer to a daughter communicator, **daughtercomm**, to which the process belongs.
- **MPI\_Comm\_split()** is a simpler, higher-level function to construct communicators, instead of using **MPI\_Comm\_create()** combined with **MPI\_Group\_excl()** or **MPI\_Group\_incl()**.

# MD & Analysis Communicators

- Split `MPI_COMM_WORLD` into two communicators; one performs molecular dynamics (MD) simulation, whereas the other analyzes simulation data on the fly in background.

```
int gid,sid,md;  
MPI_Comm workers;  
MPI_Comm_rank(MPI_COMM_WORLD,&gid); //Global rank  
md = gid%2; // = 1 (MD workers) or 0 (analysis workers)  
MPI_Comm_split(MPI_COMM_WORLD,md,0,&workers);  
MPI_Comm_rank(workers,&sid); // Rank in workers
```



Run as `mpirun -n 2×nproc`

# of processes needed for MD (specified in `pmd.h`)

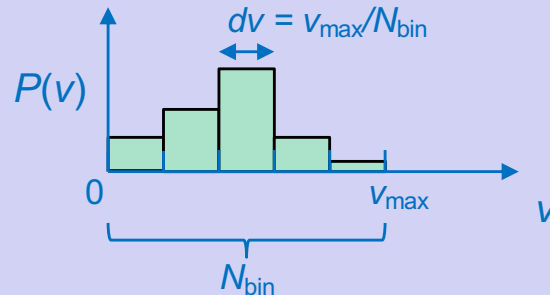
# Analysis: Velocity Probability Density

**$P(v)$ : Probability density function of atom velocity  $v$**

```
#define VMAX 5.0 // Maximum velocity value to construct a velocity histogram
#define NBIN 100 // # of bins in the histogram

void calc_pv() {
    double lpv[NBIN],pv[NBIN],dv,v;
    int i;

    dv = VMAX/NBIN; // Bin size
    for (i=0; i<NBIN; i++) lpv[i] = 0.0; // Reset local histogram
    for (i=0; i<n; i++) {
        v = sqrt(pow(rv[i][0],2)+pow(rv[i][1],2)+pow(rv[i][2],2));
        lpv[v/dv < NBIN ? (int)(v/dv) : NBIN-1] += 1.0; // Increment histogram
    }
    MPI_Allreduce(lpv,pv,NBIN,MPI_DOUBLE,MPI_SUM,workers);
    MPI_Allreduce(&n,&nglob,1,MPI_INT,MPI_SUM,workers);
    for (i=0; i<NBIN; i++) pv[i] /= (dv*nglob); // Normalization
    if (sid == 0) {
        for (i=0; i<NBIN; i++) fprintf(fpv,"%le %le\n",i*dv,pv[i]);
        fprintf(fpv,"\n");
    }
}
```



$v = |\vec{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2}$

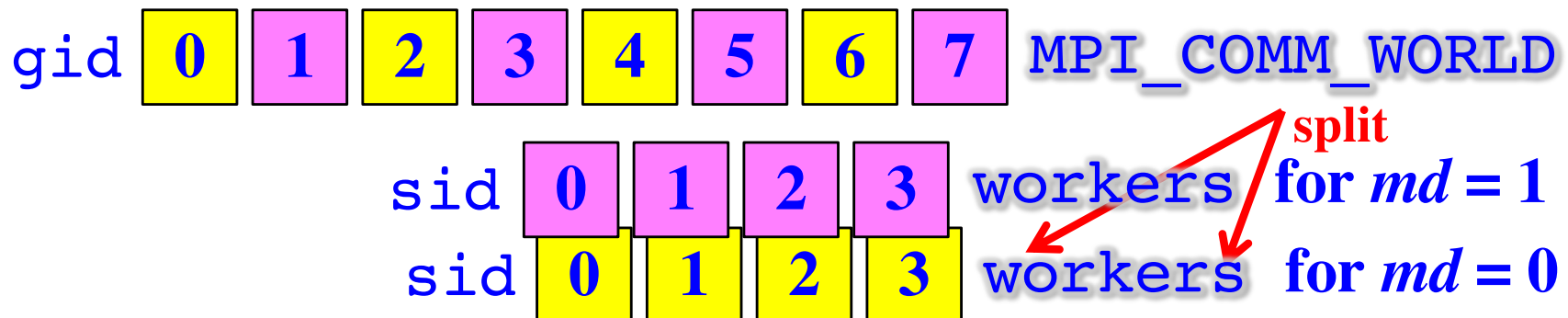
$\int dv p(v) = 1$

# Main Program: Initialization

```
init_params();
if (md) {
    set_topology();
    init_conf();
    atom_copy();
    compute_accel();
}
else
    if (sid == 0) fpv = fopen("pv.dat", "w");
```

Define FILE \*fpv;

- All processes read input parameters, `init_params()`. The `nproc` processes of MD workers (`md == 1`) perform MD initialization tasks, whereas only rank 0 among the other `nproc` analysis workers (`md == 0`) opens a file to output the calculated velocity probability density function.



# Main Program: Main MD Loop

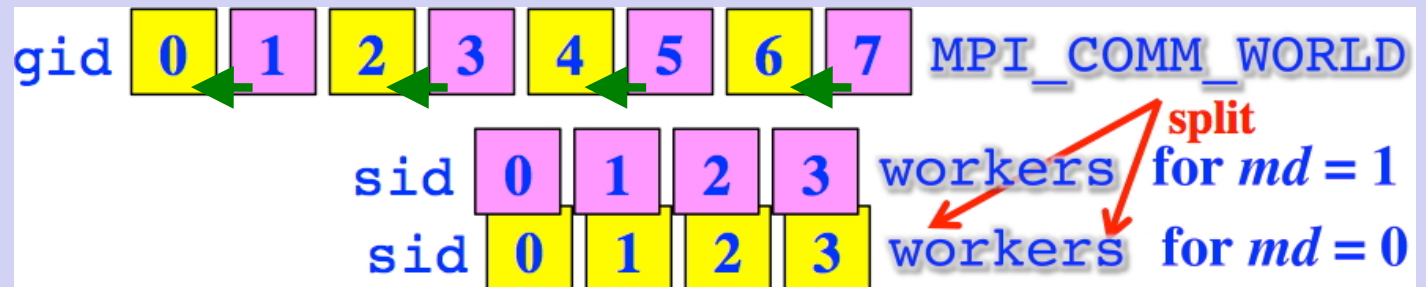
```

for (stepCount=1; stepCount<=StepLimit; stepCount++) {
  if (md) single_step();
  if (stepCount%StepAvg == 0) {
    if (md) {
      Send # of atoms, n, to rank gid-1 in MPI_COMM_WORLD
      Send velocities of n atoms to rank gid-1 in MPI_COMM_WORLD
      eval_props();
    }
    else {
      Receive # of atoms, n, from rank gid+1 in MPI_COMM_WORLD
      Receive velocities of n atoms from rank gid+1 in MPI_COMM_WORLD
      calc_pv();
    }
  }
}

```

	0	1	2	3	4	5	
<i>dbuf</i>	$v_{0x}$	$v_{0y}$	$v_{0z}$	$v_{1x}$	$v_{1y}$	$v_{1z}$	...

$dbuf[3*i+a] \leftarrow rv[i][a] \ (i = 0, \dots, n-1; a = 0,1,2)$

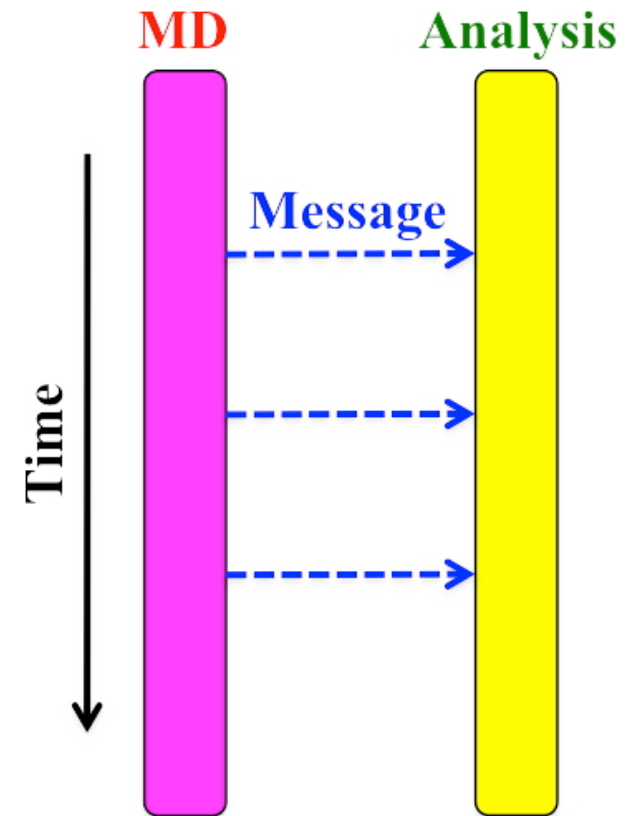


- MD workers perform MD simulation. Every **stepAvg** steps, MD workers send their atom velocities to corresponding analysis workers (*i.e.*, those with the same ranks in respective daughter communicators). Upon receiving the velocities, analysis workers calculate the velocity probability density function.

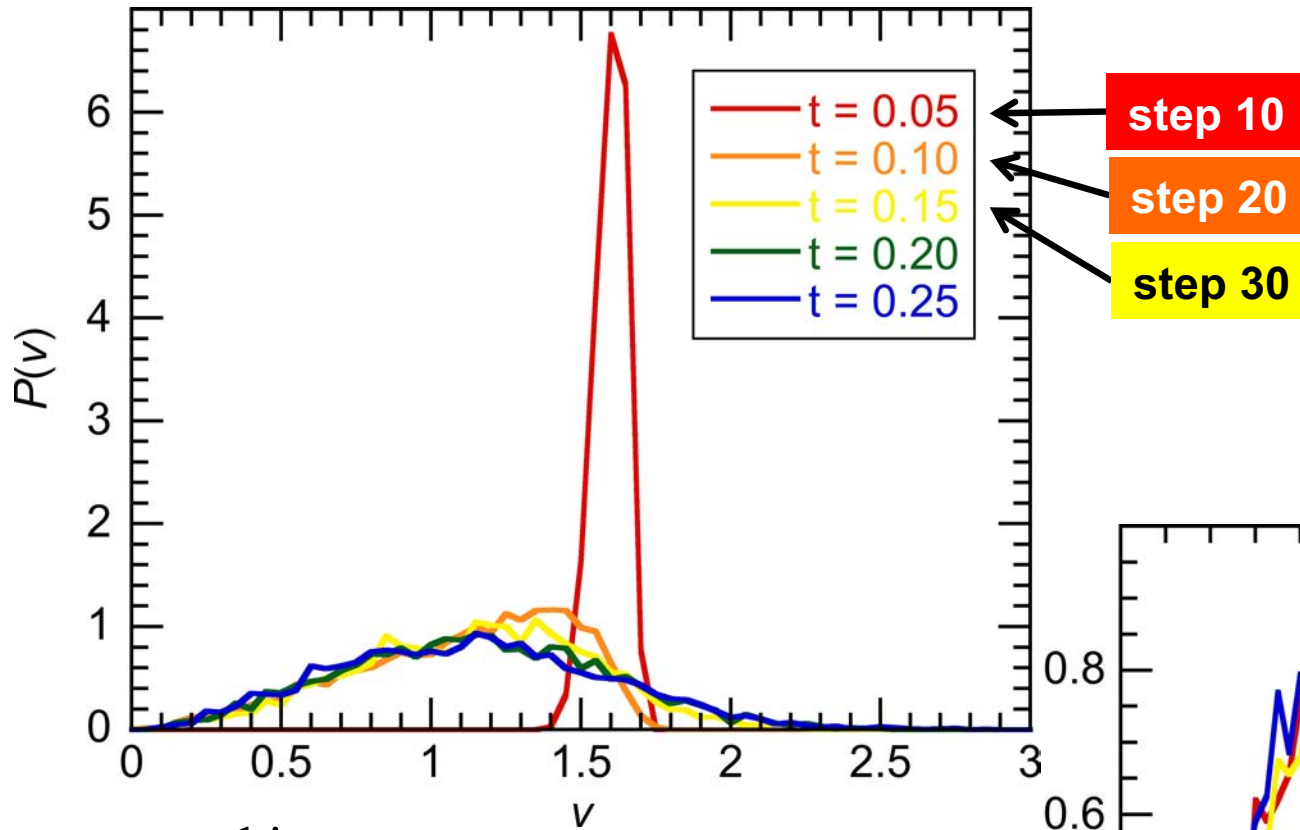
# Main Program: Finalization

```
if (md && sid == 0)
    printf("CPU & COMT = %le %le\n",cpu,comt);
if (!md && sid == 0)
    fclose(fpv);
```

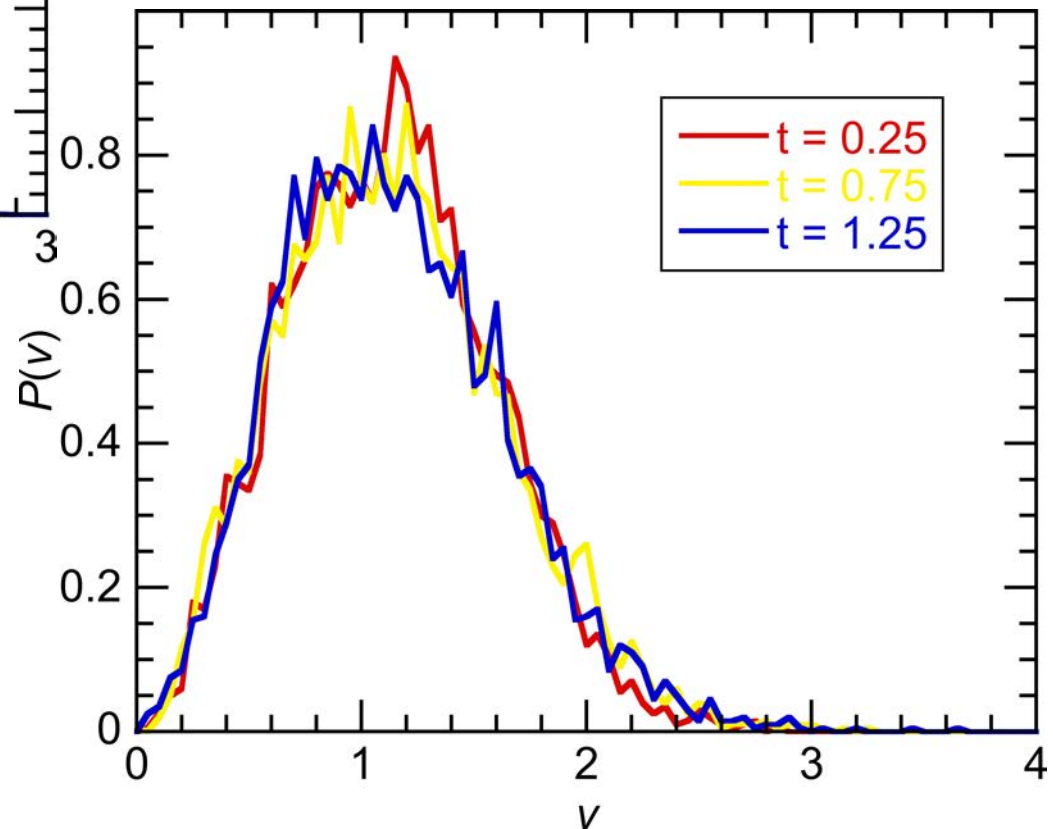
- Rank 0 of MD workers reports the computing & communication times, whereas rank 0 of analysis workers closes the probability density output file.
- **Finally:** Change all MPI\_COMM\_WORLD's in the original MD functions to workers ~ it's only a matter in the small MD world!



# Results

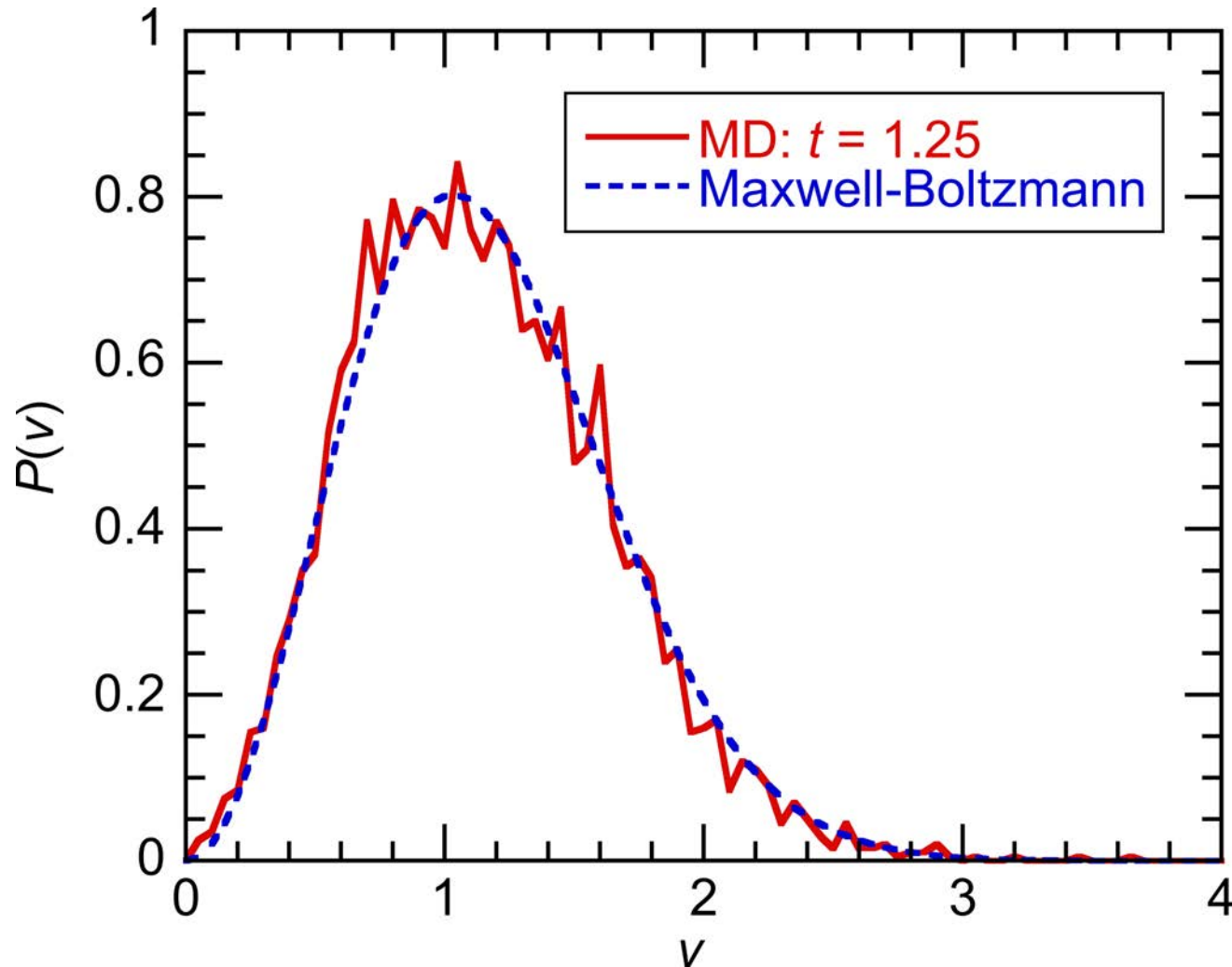


```
pmd.in
5 5 5 // InitUcell[0|1|2]
0.8   // Density
1.0   // InitTemp
0.005 // DeltaT
30    // StepLimit
10    // StepAvg
```





# Maxwell-Boltzmann Distribution

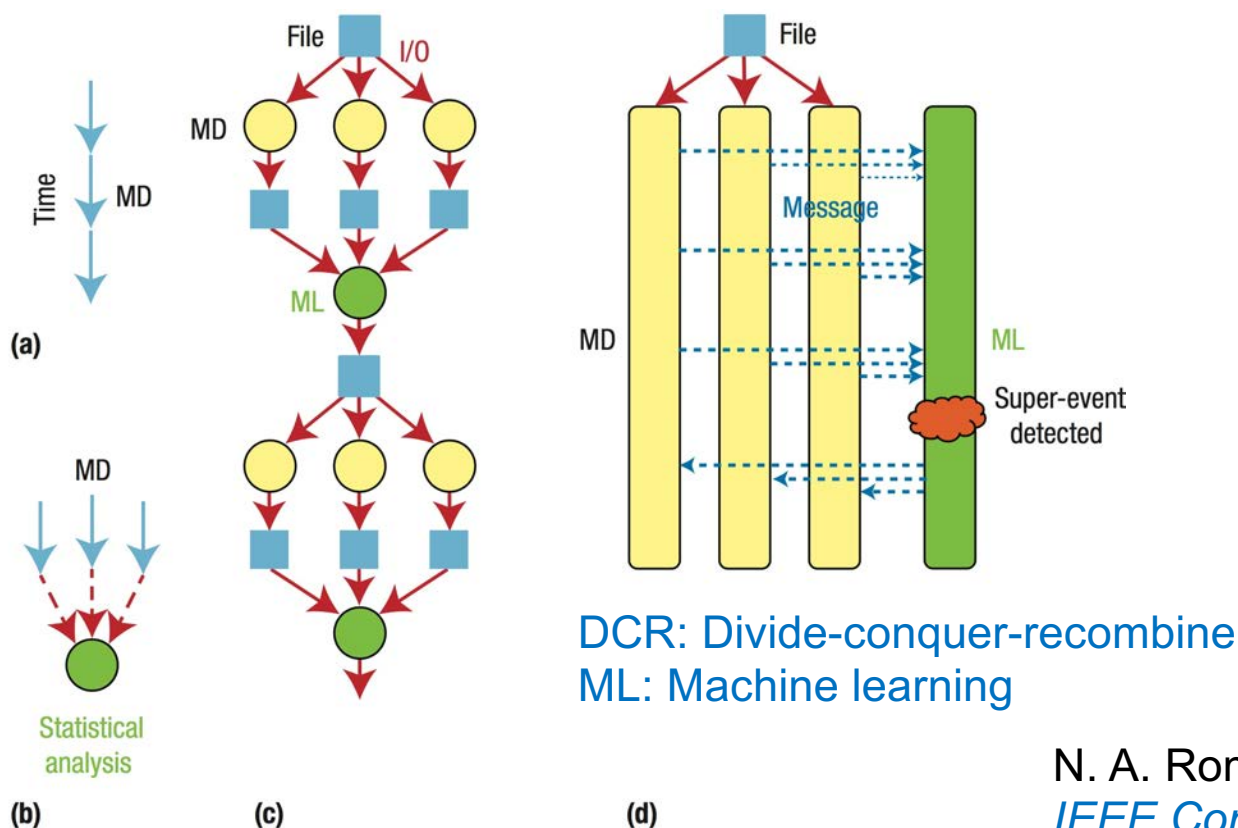


$$P_{\text{Maxwell-Boltzmann}}(v) = \frac{4}{\sqrt{\pi}} \left( \frac{m}{2k_B T} \right)^{3/2} v^2 \exp\left( -\frac{mv^2}{2k_B T} \right)$$

K. Shimamura *et al.*, [Appl. Phys. Lett. 107, 231903 \('15\)](#)

# *In Situ* Data Analysis

Use communicators to add data analytics & extra logic to parallel simulations



N. A. Romero *et al.*,  
[\*IEEE Computer\* \*\*48\*\*\(11\), 33 \('15\)](#)

**FIGURE 2.** DCR in time. (a) Molecular dynamics (MD) simulations have sequential time dependence. (b) Parallel replica dynamics (PRD) predicts long-time behavior through statistical analysis of multiple parallel MD trajectories. (c) Conventional file-based and (d) new in situ PRD simulations. ML represents machine-learning tasks.

See also T. Do *et al.*, [A lightweight method for evaluating \*in situ\* workflow efficiency](#), *J. Comput. Sci.* **48**, 101259 ('21)