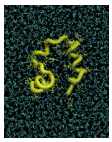
New Constraint algorithms for GPU-Accelerated Molecular Dynamics: Accuracy-Performance Trade-offs Across GPU Architectures

Jaewoon Jung[1,2], Diego Ugarte La Torre[1,3], Chigusa Kobayashi[1], Katsuhisa Ozaki[4], and Yuji Sugita[1,2,3]
[1]RIKEN R-CCS [2]RIKEN PRI [3]Univ. Tokyo [4]Shibaura Inst. Tech.

Molecular Dynamics (MD)

1. Energy/forces are described by classical molecular mechanics force field.

2. Update state according to equations of motion



$$\dot{\mathbf{r}}_i = \frac{\mathbf{p}_i}{m_i}, \quad \dot{\mathbf{p}}_i = \mathbf{F}_i$$

Equation of motion

$$\mathbf{p}_i\left(t+\frac{1}{2}\Delta t\right) = \mathbf{p}_i\left(t-\frac{1}{2}\Delta t\right) + \mathbf{F}_i(t)\Delta t$$
$$\mathbf{r}_i(t+\Delta t) = \mathbf{r}_i(t) + \frac{\mathbf{p}_i\left(t+\frac{1}{2}\Delta t\right)\Delta t}{m_i}$$

Integration

MD trajectory
=> Ensemble generation

- **Trajectory should be accurate**
- **Long time MD trajectories are important to obtain thermodynamic quantities of target systems.**

Constraints in MD

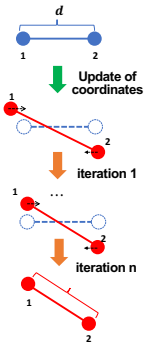
1. **Constraints in MD : freeze the chemical bonds involving hydrogen**
2. **Purpose of constraints: Increase the speed by enabling large time step**
   - Without constraint => Time step is 0.5 to 1.0 fs
   - With constraint => Time step is 2.0 fs
3. How to solve constraints: Iteration with Lagrange multiplier
   - We first assume unknown multiplier in constraint force, and the unknown multiplier is obtained with iterative way.

$$\mathbf{F}_i(t) \rightarrow \mathbf{F}_i(t) + \sum_{k\in K}\lambda_k \nabla_i \sigma_k(t), \sigma_k = |\mathbf{r}_{k_1} - \mathbf{r}_{k_2}|^2 - {d_k}^2 = 0$$

   - Lagrange multiplier λ_k is evaluated as

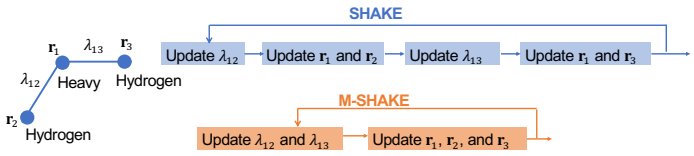
$$\lambda_k \approx \frac{{d_k}^2 - |\mathbf{r}_{k_1} - \mathbf{r}_{k_2}|^2}{\Delta t^2 (\mathbf{r}_{k_1} - \mathbf{r}_{k_2})\cdot(\mathbf{r}_{\mathrm{old},k_1} - \mathbf{r}_{\mathrm{old},k_2})\left(\frac{1}{m_{k_1}} + \frac{1}{m_{k_2}}\right)}$$

   - Lagrange multiplier λ_k is updated until $|\mathbf{r}_{k1} - \mathbf{r}_{k2}|$ is close to d_k.



Constraints Algorithm: M-SHAKE

For XHn case, we consider Lagrange multipliers at the same time instead of considering each one by one.



- We can reduce the iteration by considering the coupling of 1-2 and 1-3.
- In each step, we need matrix inverse calculation for 2x2 (XH_2 case) and 3x3 (XH_3 case)

In M-SHAKE of XH_2 and XH_3, the Lagrange multipliers are obtained by

$$XH_2: \begin{bmatrix}\lambda_{12}\\ \lambda_{13}\end{bmatrix} = \mathbf{A}_{XH_2}^{-1}\begin{bmatrix}{d_{12}}^2 - |\mathbf{r}_1 - \mathbf{r}_2|^2\\ {d_{13}}^2 - |\mathbf{r}_1 - \mathbf{r}_3|^2\end{bmatrix}, \quad XH_3: \begin{bmatrix}\lambda_{12}\\ \lambda_{13}\\ \lambda_{14}\end{bmatrix} = \mathbf{A}_{XH_3}^{-1}\begin{bmatrix}{d_{12}}^2 - |\mathbf{r}_1 - \mathbf{r}_2|^2\\ {d_{13}}^2 - |\mathbf{r}_1 - \mathbf{r}_3|^2\\ {d_{14}}^2 - |\mathbf{r}_1 - \mathbf{r}_4|^2\end{bmatrix}$$

$$\mathbf{A}_{XH_2} = \begin{bmatrix}\left(\frac{\Delta t^2}{m_1} + \frac{\Delta t^2}{m_2}\right)(\mathbf{r}_1 - \mathbf{r}_2)\cdot(\mathbf{r}_{\mathrm{old},1} - \mathbf{r}_{\mathrm{old},2}) & \frac{\Delta t^2}{m_1}(\mathbf{r}_1 - \mathbf{r}_2)\cdot(\mathbf{r}_{\mathrm{old},1} - \mathbf{r}_{\mathrm{old},3})\\ \frac{\Delta t^2}{m_1}(\mathbf{r}_1 - \mathbf{r}_3)\cdot(\mathbf{r}_{\mathrm{old},1} - \mathbf{r}_{\mathrm{old},2}) & \left(\frac{\Delta t^2}{m_1} + \frac{\Delta t^2}{m_3}\right)(\mathbf{r}_1 - \mathbf{r}_3)\cdot(\mathbf{r}_{\mathrm{old},1} - \mathbf{r}_{\mathrm{old},3})\end{bmatrix}$$

$$\mathbf{A}_{XH_3} = \begin{bmatrix}\left(\frac{\Delta t^2}{m_1} + \frac{\Delta t^2}{m_2}\right)(\mathbf{r}_1 - \mathbf{r}_2)\cdot(\mathbf{r}_{\mathrm{old},1} - \mathbf{r}_{\mathrm{old},2}) & \cdots & \frac{\Delta t^2}{m_1}(\mathbf{r}_1 - \mathbf{r}_2)\cdot(\mathbf{r}_{\mathrm{old},1} - \mathbf{r}_{\mathrm{old},4})\\ \frac{\Delta t^2}{m_1}(\mathbf{r}_1 - \mathbf{r}_3)\cdot(\mathbf{r}_{\mathrm{old},1} - \mathbf{r}_{\mathrm{old},2}) & \ddots & \frac{\Delta t^2}{m_1}(\mathbf{r}_1 - \mathbf{r}_3)\cdot(\mathbf{r}_{\mathrm{old},1} - \mathbf{r}_{\mathrm{old},4})\\ \frac{\Delta t^2}{m_1}(\mathbf{r}_1 - \mathbf{r}_4)\cdot(\mathbf{r}_{\mathrm{old},1} - \mathbf{r}_{\mathrm{old},2}) & \cdots & \left(\frac{\Delta t^2}{m_1} + \frac{\Delta t^2}{m_4}\right)(\mathbf{r}_1 - \mathbf{r}_4)\cdot(\mathbf{r}_{\mathrm{old},1} - \mathbf{r}_{\mathrm{old},4})\end{bmatrix}$$

Because coordinates are updated during iteration, we need to evaluate inverse matrix at every iteration.

Special treatment of XH_1 case

By considering nonlinearity, we can obtain analytic solution of the Lagrange multiplier

$$\mathbf{r}_1^c = \mathbf{r}_1 + \frac{\Delta t^2}{2m_1}(\mathbf{r}_{\mathrm{old},1} - \mathbf{r}_{\mathrm{old},2})\lambda_{12} = \mathbf{r}_1 + \frac{1}{m_1}\mathbf{b}\lambda_{12}$$
$$\mathbf{r}_2^c = \mathbf{r}_2 - \frac{\Delta t^2}{2m_2}(\mathbf{r}_{\mathrm{old},1} - \mathbf{r}_{\mathrm{old},2})\lambda_{12} = \mathbf{r}_2 - \frac{1}{m_2}\mathbf{b}\lambda_{12}$$
$$|\mathbf{r}_1^c - \mathbf{r}_2^c|^2 = |\mathbf{r}_1 - \mathbf{r}_2|^2 + 2\left(\frac{1}{m_1} + \frac{1}{m_2}\right)(\mathbf{r}_1 - \mathbf{r}_2)\cdot\mathbf{b}\lambda_{12} + \left(\frac{1}{m_1} + \frac{1}{m_2}\right)^2 b^2{\lambda_{12}}^2 = d^2$$

Quadratic equation of λ_{12} and we can obtain the solution from quadratic formula !!

How to improve accuracy with FP32?

- Accuracy of coordinate is limited to 10^{-7} ~ 10^{-6} by FP32.
- Accordingly, the accuracy of velocity is limited by 10^{-6} ~ 10^{-4} and we cannot obtain the maxium accuracy of FP32.
- This can limit the usage of commercial GPU.

Our suggestion to obtain the same level of velocity accuracy as coordinate accuracy

- We use mixed precision in coordinates by considering temporay coordinates as FP64.
- Velocity update is performed at the same time of coordinate update.

1. Initialization:
1.1. Load inverse masses: $\frac{1}{m_x}$ and $\frac{1}{m_H}$ (FP32)
1.2. Load the reference relative positions: $\mathbf{r}_{0i} = \mathbf{r}_0(t) - \mathbf{r}_{hi}(t)$ (FP32)
1.3. Load predicted positions: $\mathbf{r}_0^*(t+\Delta t)$ and $\mathbf{r}_{hi}^*(t+\Delta t)$ **(FP64)**
1.4. Load predicted velocities: $\mathbf{v}_0^*\left(t+\frac{\Delta t}{2}\right)$ and $\mathbf{v}_{hi}^*\left(t+\frac{\Delta t}{2}\right)$ (FP32)
1.5. Load target of bond length: d_{0i} (FP32)

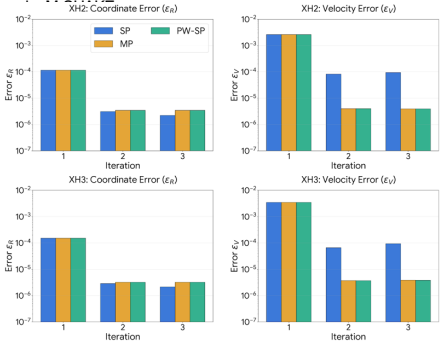
2. Iterative Loop:
for iter = 1, N:
2.1. Compute predicted bond vectors **(FP64)**:
$\mathbf{r}_{0i}^* = \mathbf{r}_0^*(t+\Delta t) - \mathbf{r}_{hi}^*(t+\Delta t)$
2.2. Cast bond vectors to lower precision:
$\mathbf{r}_{0i}^*$ (FP64) $\rightarrow \mathbf{r}_{0i}^*$ (FP32)
2.3. Evaluate the constraint matrix $\mathbf{A}_{XH_n}$ (FP32)
2.4. Invert the matrix $\mathbf{A}_{XH_n}^{-1}$ (FP32)
2.5. Compute constraint violations (FP32):
$\delta_i = d_{0i}^2 - |\mathbf{r}_{0i}^*|^2$
2.6. Solve for Lagrange multipliers (FP32):
$\begin{bmatrix}\lambda_{01}\\ \vdots\\ \lambda_{03}\end{bmatrix} = \boldsymbol{A}_{XH_n}^{-1}\begin{bmatrix}\delta_1\\ \vdots\\ \delta_n\end{bmatrix}$
2.7. Update $\mathbf{r}_0^*(t+\Delta t)$ and $\mathbf{r}_{hi}^*(t+\Delta t)$ **(FP64)**
2.8. Update s $\mathbf{v}_0^*\left(t+\frac{\Delta t}{2}\right)$ and $\mathbf{v}_{hi}^*\left(t+\frac{\Delta t}{2}\right)$ (FP32)

3. Set the constrained coordinates and velocities (FP32):
$\mathbf{r}_0^*(t+\Delta t) \rightarrow \mathbf{r}_0(t+\Delta t)$ and $\mathbf{r}_{hi}^*(t+\Delta t) \rightarrow \mathbf{r}_{hi}(t+\Delta t)$
$\mathbf{v}_0^*\left(t+\frac{\Delta t}{2}\right) \rightarrow \mathbf{v}_0\left(t+\frac{\Delta t}{2}\right)$ and $\mathbf{v}_{hi}^*\left(t+\frac{\Delta t}{2}\right) \rightarrow \mathbf{v}_{hi}\left(t+\frac{\Delta t}{2}\right)$

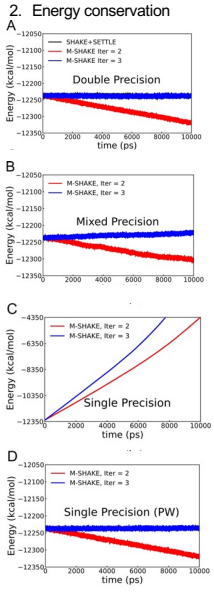
We further apply pairwise arithmetic instead of using FP64 for coordinates

Result

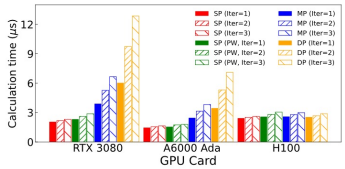
1. Error of coordinate (ε_R (Å)) and velocity ε_V (Å/ps*)



2. Energy conservation



3. Performance of constraints



Summary

1. We presented a constraint algorithm in MD to enhance the computational speed while maintaining accuracy
   - Mixed-precision for coordinate
   - Single-precision with a pairwise sum algorithm
2. The new algorithms has much better accuracy than using single precision only.
3. The new algorithms improve performance on consumer-grade GPUs with limited FP64 throughput.

Acknowledgements

This work was supported in part by MEXT JSPS KAKENHI (Grant Nos. 19H05645 and 21H05249 (to Y.S.), 21H05282 (to J.J.), and 23K11328 (to C.K.)). This work used computational resources of the TSUBAME4.0 provided by Institute of Science Tokyo through the HPCI System Research Project (Project ID: hp250065).