Parallel Quantum Dynamics

Aiichiro Nakano

Collaboratory for Advanced Computing & Simulations
Department of Computer Science
Department of Physics & Astronomy
Department of Chemical Engineering & Materials Science
Department of Biological Sciences
University of Southern California

Email: anakano@usc.edu

Self-centric parallelization (easy spatial decomposition) of a partial-differential-equation solver as a ‘boundary condition’
Self-Centric (SC) Parallelization

• SC is the easiest serial-to-parallel migration path via single-program multiple-data (SPMD) programming
  1. Take a serial code
  2. Each MPI rank only works on a spatial subsystem
  3. Boundary information obtained from neighbor ranks
  4. Long-range interaction by real-space multigrids; scalability behavior is similar to short-ranged (see slides 7-8 in [link](http://cacs.usc.edu/education/cs653/02-04DC-slide.pdf))

S. C. Tiwari et al., *ACM HPCAsia2020*, Best Paper (’20)
K. Nomura et al., *IEEE/ACM Supercomputing, SC14* (’14)
Quantum Dynamics Program: qd1.c

for step = 1 to NSTEP
    pot_prop(): \( \psi_j \leftarrow \exp(-iV_j \Delta t/2) \psi_j \) (\( j \in [1,NX] \))
    kin_prop(\( \Delta t/2 \))
    kin_prop(\( \Delta t \))
    kin_prop(\( \Delta t/2 \))
    pot_prop(): \( \psi_j \leftarrow \exp(-iV_j \Delta t/2) \psi_j \) (\( j \in [1,NX] \))

\[
\psi(t + \Delta t) \leftarrow \exp(-iV\Delta t/2) \exp(-iT_x \Delta t) \exp(-iV\Delta t/2) \psi(t)
= e^{-iV\Delta t/2} U^{(\text{half})}_x U^{(\text{full})}_x U^{(\text{half})}_x e^{-iV\Delta t/2} \psi(t)
\]

\[
\begin{align*}
\psi_j \leftarrow & blx(\Delta)_j \psi_{j-1} + al(\Delta)_j \psi_j + bux(\Delta)_j \psi_{j+1} \\
\exp(-i\Delta T_x) & = U^{(\text{half})}_x U^{(\text{full})}_x U^{(\text{half})}_x
\end{align*}
\]

\[
\begin{align*}
\epsilon_n^+ &= \frac{1}{2} \left[ \exp\left( -\frac{i\Delta t}{n} (a + b \right) \right] + \exp\left( -\frac{i\Delta t}{n} (a - b \right) \right] \\
\epsilon_n^- &= \frac{1}{2} \left[ \exp\left( -\frac{i\Delta t}{n} (a + b \right) \right] - \exp\left( -\frac{i\Delta t}{n} (a - b \right) \right]
\end{align*}
\]

\( \text{cf. velocity-Verlet half-time acceleration} \)
Quantum Dynamics Computation

- **Essence:** Keep updating wave-function values mesh point-by-point as a function of those on the nearest-neighbor mesh points

  $$\psi_j(t+1) \leftarrow f(\psi_{j-1}(t), \psi_j(t), \psi_{j+1}(t))(j \in [1,N_x])$$

- **Periodic boundary condition via augmentation**

  ```c
  void periodic_bc() {
    int s;
    for (s=0; s<=1; s++) {
      psi[0][s] = psi[NX][s];
      psi[NX+1][s] = psi[1][s];
    }
  }
  ```

- **Often sufficient just to understand computational characteristics for parallelizing a serial code**

  [Diagram]

  http://cacs.usc.edu/education/cs596/src/qd
SC Parallelization

• Self-centric spatial decomposition

\[ x_j = j\Delta x \]

\[ x_j^{(global)} = j\Delta x + pL_x \]

offset

• Local & global coordinates

• Global coordinates only in \( \text{init\_prop}() \) & \( \text{init\_wavefn}() \)
Boundary Wave Function Caching

- **Parallelized periodic_bc()**

  ```c
  int myid, nproc;
  plw = (myid-1+nproc)%nproc; /* Lower partner process */
  pup = (myid+1)%nproc; /* Upper partner process */
  
  /* Cache boundary wave function value at the lower end */
  dbuf[0:1] ← psi[NX][0:1];  
  Send dbuf to pup;
  Receive dbufr from plw;
  psi[0][0:1] ← dbufr[0:1];
  
  /* Cache boundary wave function value at the upper end */
  dbuf[0:1] ← psi[1][0:1];
  Send dbuf to plw;
  Receive dbufr from pup;
  psi[NX+1][0:1] ← dbufr[0:1];
  ```

I. Message (1D array) composition

II. Message passing

III. Message storing
• Parallelized periodic\_bc()

for \forall \text{directions}

send front row \psi(...,1 \text{ or } \text{N}_\alpha,...) to forward neighbor
receive back appendage \psi(...,\text{N}_\alpha+1 \text{ or } 0,...) from back neighbor
Multidimensional Parallelization

- Message composition

\[
dbuf \leftarrow psi(i_b : i_e, j_b : j_e, k_b : k_e) \\
psi(i'_b : i'_e, j'_b : j'_e, k'_b : k'_e) \leftarrow dbuf^r
\]

(Example) x-low direction

\[
i_b = 1, \ i_e = 1, \ j_b = 1, \ j_e = N_y, \ k_b = 1, \ k_e = N_z \\
i'_b = N_x + 1, \ i'_e = N_x + 1, \ j'_b = 1, \ j'_e = N_y, \ k'_b = 1, \ k'_e = N_z
\]
Parallel QD Results

Strong scaling results

- Wall-clock time (s) vs. Number of processors
- Speedup vs. Number of processors
- Parallel efficiency vs. Number of processors

Measured speedup vs. Ideal speedup
Parallel QD Communications

\[ \psi \leftarrow K\psi \]

**kinetic propagator**

**spectral method**

\[ \psi \leftarrow \exp(-iK\Delta)\psi \]

**space-splitting method**

\[ \psi \leftarrow \exp(-iK\Delta)\psi \]

\[ O(M\log N) \]

Up to 1,000 processors

\[ O(N) \]

Metascaleable to future architectures
Parallel QD Algorithms

- Not all algorithms are scalable on parallel computers
- Implicit solvers (e.g. Crank-Nicholson method) are numerically stable but less scalable due to sequential dependence

\[
\psi(t + \Delta t) \leftarrow \exp \left( -\frac{i}{\hbar} \hat{H} \Delta t \right) \psi(t) \approx \frac{1 - \frac{i}{2\hbar} \hat{H} \Delta t}{1 + \frac{i}{2\hbar} \hat{H} \Delta t} \psi(t) + O((\Delta t)^3)
\]

\[
\left(1 + \frac{i}{2\hbar} \hat{H} \Delta t\right) \psi(t + \Delta t) = \left(1 - \frac{i}{2\hbar} \hat{H} \Delta t\right) \psi(t)
\]

\[
\alpha x_{i-1} + \beta x_i + \alpha x_{i+1} = b_i
\]

\[
x_{i+1} \leftarrow \frac{1}{\alpha} b_i - \frac{\beta}{\alpha} x_i - x_{i-1}
\]

- Sequential recursion needs be converted to divide-&-conquer (recursive doubling) for parallelization