Supplementary Material

The Supplementary Material includes the following data: (1) Table S1 presents the corresponding code for neighbor search implemented using the three different schemes. (2) Fig. S1 shows the variation in GFLOPs/s per watt as the particle number changes across different GPU architectures. (3) Fig. S2 shows the variation in running efficiency as the particle number changes for example 2 (a plate with a circular hole). (4) Fig. S3 shows the variation in running efficiency as the particle scale changes for example 3 (a 2D plate with a pre-existing crack). (5) Fig. S4 shows the variation in running efficiency as the particle scale changes for example 4 (a 3D column). (6) Fig. S5 shows the variation in running efficiency as the particle scale changes for example 5 (Kalthoff-Winkler experiment). (7) Fig. S6 shows the variation of graphics memory required for example 4 with the computation scale.

**Table S1**

|  |
| --- |
| **Table S1** Three different code schemes are used for neighbor search. |
| 1. **//a) Serial code** 2. for (int i = 0; i < N; i++){ 3. for (int j = i + 1; j < N; j++){ 4. if (pow(x[j] - x[i], 2) + pow(y[j] - y[i], 2) < pow(horizon, 2)){ 5. NL[i \* MN + NN[i]++] = j; 6. NL[j \* MN + NN[j]++] = i; 7. } 8. } 9. } 10. **//b) Classical GPU parallel code** 11. unsigned int instr = threadIdx.x + blockIdx.x + blockDim.x; 12. unsigned int stride = gridDim.x \* blockDim.x; 13. for (int i = instr; i < N; i+= stride){ 14. for (int j = i + 1; j < N; j++){ 15. if (pow(x[j] - x[i], 2) + pow(y[j] - y[i], 2) < pow(horizon, 2)){ 16. NL[i \* MN + atomicAdd(NN[i], 1)] = j; 17. NL[j \* MN + atomicAdd(NN[j], 1)] = i; 18. } 19. } 20. } 21. **//c) Subdomain-based GPU parallel code** 22. unsigned int instr = threadIdx.x + blockIdx.x + blockDim.x; 23. unsigned int stride = gridDim.x \* blockDim.x; 24. for (int i = instr; i < cell[aero]; i+= stride){ 25. for (int j = 0; j < (cell[aero] + boundary[aero]); j++){ 26. if (pow(x[j] - x[i], 2) + pow(y[j] - y[i], 2) < pow(horizon, 2)){ 27. NL[i \* MN + atomicAdd(NN[i], 1)] = j; 28. NL[j \* MN + atomicAdd(NN[j], 1)] = i; 29. } 30. } 31. } |
| where, “N” denotes the total number of particles, “MN” is the maximum number of neighbors per particle, “NN” stores the number of neighbors for each particle, and “NL” stores the neighbor indices. “cell[aero]” represents the number of particles within the subdomain, while “boundary[aero]” denotes the number of particles contained in the boundary region of the subdomain. |

In the present study, the adopted neighbor search algorithm is consistently the brute-force method for all simulations. In contrast to traditional cell-linked list method, the subdomains here were designed with built-in boundary regions, so brute-force neighbor search was only performed within a single subdomain. For clarity, three different code schemes, including serial code, classical GPU parallel code, and subdomain-based GPU parallel code, have been given in Table S1. It suggests that the difference between the classical GPU parallel scheme and the subdomain-based GPU parallel scheme mainly lies in the number of particles within the computational domain.

**Figure S1**

|  |
| --- |
|  |
| **Fig. S1.** GFLOPs/s per watt as a function of the computing scale. |

In Section 3.1 of the manuscript, the floating-point performance (GFLOPs/s) during the neighbor search stage was analyzed. Then, we further compared the computational throughput of the proposed scheme across different GPU architectures (A100 80GB PCIe and RTX 4060 Ti 16GB) as the computational scale increases. It can provide a clearer quantitative assessment of the scalability.

The computational throughput is defined as:

|  |  |
| --- | --- |
|  | (S1) |

where, GFLOPs/s denotes the number of floating-point operations per unit time, and *P* represents the power consumption of the GPU device.

Fig. S1 shows the variation of GFLOPs/s per watt with particle count for both GPU architectures. Although the A100 achieves significantly higher absolute throughput due to its superior architecture and memory bandwidth, both devices exhibit the same trend: GFLOPs/s per watt increases with particle number and gradually approaches convergence. This consistent behavior indicates effective utilization of GPU resources on both platforms. Therefore, the agreement between the two architectures demonstrates that the proposed scheme does not depend on a particular hardware design and exhibits good scalability, thereby supporting its general applicability across different GPU architectures.

**Figure S2**

|  |
| --- |
| Example2 附录 |
| **Fig. S2.** (a) Total running time for all scheme and (b) speedups as a function of the computing scale when not considering data transfer. (c) Total running time for all scheme and (d) speedups as a function of the computing scale. |

Using the particle spacing Δ of 0.5mm, 0.2mm, 0.1mm, 0.067mm, and 0.02mm, we constructed five PD models with various particles. Fig. S2 shows the variation in running time and speedup ratio for example 2 as the number of particles increases from 9,684 to 968,572. Due to domain partitioning, the subdomain-based parallel scheme involves more complex data access operations, compared to classical parallel schemes, which leads to increased time consumption, especially for small computation models. Thus, the subdomain-based parallel scheme exhibits lower runtime efficiency for small models, as shown in Fig. S2b. However, as the number of particles increases, the advantages of the subdomain-based scheme in the neighbor search become significant, and thus its computational efficiency gradually surpasses that of the classical parallel scheme.

Figs. S2c,d shows the total runtime of the all schemes and the speedup of each parallel scheme of the computational scale. When the number of particles increases from 9,684 to 968,572, the speedup of the classical parallel scheme increases from 27.6× to 518.9×, while the speedup of the subdomain-based parallel scheme increases from 9.7× to 144.7×. Since the subdomain-based parallel scheme requires the particle information in the GPU to be updated at each time step, the generated data transfer time largely exceeds those of the classical parallel scheme, leading to limited improvements in the total runtime.

**Figure S3**

|  |
| --- |
| Example3 汇总 |
| **Fig. S3.** (a) Total running time for all scheme and (b) speedups as a function of the computing scale when not considering data transfer. (c) Total running time for all scheme and (d) speedups as a function of the computing scale. |

Using the particle spacing Δ of 0.5mm, 0.2mm, 0.1mm, 0.067mm, and 0.02mm, we constructed five PD models with various particles. Fig. S3 shows the variation in running time and speedup ratio for example 3 as the number of particles increases from 10 thousand to 1 million. Due to domain partitioning, the subdomain-based parallel scheme involves more complex data access operations, compared to classical parallel schemes, which leads to increased time consumption, especially for small computation models. Thus, the subdomain-based parallel scheme exhibits lower runtime efficiency for small models, as shown in Fig. S3b. However, as the number of particles increases, the advantages of the subdomain-based scheme in the neighbor search become significant, and thus its computational efficiency gradually surpasses that of the classical parallel scheme.

Figs. S3c,d shows the total runtime of the all schemes and the speedup of each parallel scheme of the computational scale. When the number of particles increases from 10 thousand to 1 million, the speedup of the classical parallel scheme increases from 31.3× to 704.0×, while the speedup of the subdomain-based parallel scheme increases from 10.0× to 142.3×. Since the subdomain-based parallel scheme requires the particle information in the GPU to be updated at each time step, the generated data transfer time largely exceeds those of the classical parallel scheme, leading to limited improvements in the total runtime.

**Figure S4**

|  |
| --- |
| C:\Users\yang\Desktop\图片2.png |
| **Fig. S4.** (a) Total running time for all scheme and (b) speedups as a function of the computing scale when not considering data transfer. (c) Total running time for all scheme and (d) speedups as a function of the computing scale. |

Using the particle spacing Δ of 0.05mm, 0.0375mm, 0.025mm, 0.015mm, and 0.00875mm, we constructed five PD models with various particles. Fig. S4 shows the variation in running time and speedup ratio for example 4 as the number of particles increases from 13,168 to 2,428,284. Due to domain partitioning, the subdomain-based parallel scheme involves more complex data access operations, compared to classical parallel schemes, which leads to increased time consumption, especially for small computation models. Thus, the subdomain-based parallel scheme exhibits lower runtime efficiency for small models, as shown in Fig. S4b. However, as the number of particles increases, the advantages of the subdomain-based scheme in the neighbor search become significant, and thus its computational efficiency gradually surpasses that of the classical parallel scheme.

Figs. S4c,d shows the total runtime of the all schemes and the speedup of each parallel scheme of the computational scale. When the number of particles increases from 13,168 to 2,428,284, the speedup of the classical parallel scheme increases from 98.3× to 368.3×, while the speedup of the subdomain-based parallel scheme increases from 20.6× to 106.1×. Since the subdomain-based parallel scheme requires the particle information in the GPU to be updated at each time step, the generated data transfer time largely exceeds those of the classical parallel scheme, leading to limited improvements in the total runtime.

**Figure S5**

|  |
| --- |
| Example4 汇总 |
| **Fig. S5.** (a) Total running time for all scheme and (b) speedups as a function of the computing scale when not considering data transfer. (c) Total running time for all scheme and (d) speedups as a function of the computing scale. |

Using the particle spacing Δ of 1.25mm, 1mm, 0.75mm, 0.5mm, and 0.375mm, we constructed five PD models with various particles. Fig. S5 shows the variation in running time and speedup ratio for example 5 as the number of particles increases from 89,040 to 3,409,032. Due to domain partitioning, the subdomain-based parallel scheme involves more complex data access operations, compared to classical parallel schemes, which leads to increased time consumption, especially for small computation models. Thus, the subdomain-based parallel scheme exhibits lower runtime efficiency for small models, as shown in Fig. S5b. However, as the number of particles increases, the advantages of the subdomain-based scheme in the neighbor search become significant, and thus its computational efficiency gradually surpasses that of the classical parallel scheme.

Figs. S5c,d shows the total runtime of the all schemes and the speedup of each parallel scheme of the computational scale. When the number of particles increases from 89,040 to 3,409,032, the speedup of the classical parallel scheme increases from 292.2× to 578.4×, while the speedup of the subdomain-based parallel scheme increases from 10.2× to 31.2×. Since the subdomain-based parallel scheme requires the particle information in the GPU to be updated at each time step, the generated data transfer time largely exceeds those of the classical parallel scheme, leading to limited improvements in the total runtime.

**Figure S6**

|  |
| --- |
|  |
| **Fig. S6.** The required graphics memory changes with the computational scale (example 4). |

Statistically, the example 4 contained 26 double-precision real array variables (double type) for displacement, velocity, acceleration, etc. (in particular, two of the arrays have 128 elements for each particle), and 2 integer array variables (int type) for recording the maximum number of family members of particles (1 element for each particle), and the family number of particles (128 elements for each particle), respectively. Therefore, we could obtain the total byte size of . As shown Fig. S6, the intercept of the memory consumption curve for the classical parallel scheme is 469, indicating that the additional graphics memory required by this program is 469MiB (due to extra memory consumption caused by data alignment and other reasons). Thus, the total graphics memory required by the overall program can be estimated using Eq. (9) as: (2756 × 12,822,400) / 10242 + 469 = 34170MiB.