GPU porting of the bottleneck kernel InteractionForcesFluid in DualSPHysics

Waku Hasegawa†, Aoto Abe†, Dai Wada†, and Takaaki Miyajima†, †Meiji University (Japan),
e-mail:takaaki.miyajima@cs.meiji.ac.jp

明治大学 MEIJI UNIVERSITY

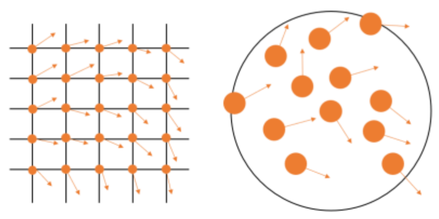
Overview

Physics simulations are utilized across various fields such as engineering and computational sciences, significantly contributing to the reduction of prototyping costs in product development.The open-source fluid analysis program DualSPHysics, based on the Smoothed Particle Hydrodynamics (SPH) method, has seen expanding applications in engineering.
In the SPH method, increasing the number of particles improves accuracy, but also increases the computational load.GPGPU (General-Purpose computing on Graphics Processing Units) has gathered attention as a potential solution to this issue; however, when the codebase is large, porting to GPUs becomes difficult even when using OpenACC. In this study, we performed performance optimization on the InteractionForcesFluid function, which is the primary computational bottleneck in the entire DualSPHysics system.
Through several optimizations such as sync avoidance, formula replacement, and templating particle attributes, a maximum speedup of 9.39 times compared to the CPU version was achieved.

Smoothed Particle Hydrodynamics (SPH)

Comparison of Particle Methods and Grid Methods

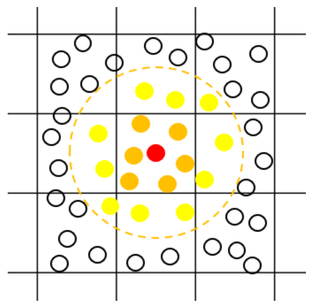
- Particle methods are Lagrangian approaches that discretize matter as particles
- Particle methods can flexibly handle complex physical behaviors such as free surfaces and deformations



Grid Method　　Particle Method

Bottleneck

- In particle methods, a significant portion of computation time is spent searching for “neighbor particles” that influence the target particle and calculating the effects they exert

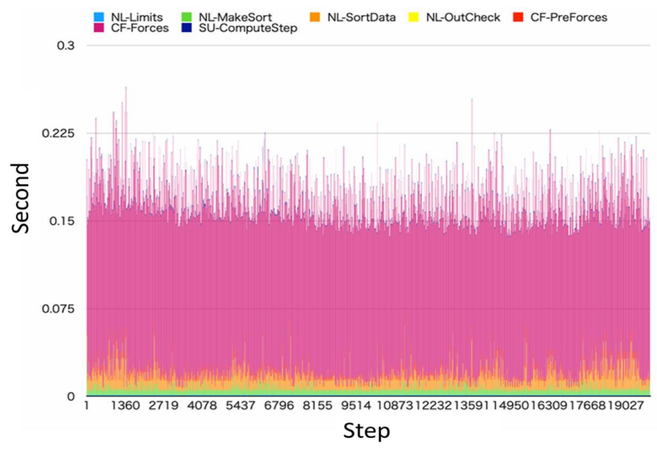


neighbor particles and affct particles

DualSPHysics

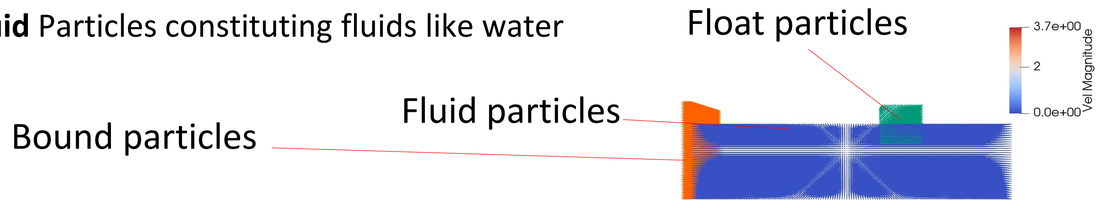
Open-source fluid analysis software based on the SPH method

- Supports parallel processing in OpenMP and CUDA
- Does not support process parallel processing via MPI
- Neighborhood particle search processing time accounts for the majority of execution time (pink portion)



Particle Attributes

- **Bound** Particles representing immovable objects such as walls and obstacles
- **Float** Particles constituting movable rigid bodies such as wood or blocks floating on water
- **Fluid** Particles constituting fluids like water



Four Optimizations of Interaction Calculation Functions for GPU using OpenACC

V01: DualSPHysics’s OpenMP implementation (baseline)

V02: Naive GPU porting and avoiding unnecessary data transfer

- Lifetime of transfer target data and transfer data
- Read-only variables: Use copyin clause
- Modifiable variables: Use copy specification
- Conditionally used variables: Use if-conditional data regions to prevent redundant data transfers

```
#pragma acc data copyin(this,FtObjs[0:FtCount],t.divdata, ... ) \
                copy(viscdt,t.ar[0:Np],t.ace[0:Np])
#pragma acc data copy(t.shiftposfs[0:Np]) if(shift)
#pragma acc data copy(t.spsgradvel[0:Np]) if(tvisco==VISCO_LaminarSPS)
#pragma acc data copy(t.delta[0:Np]) if(t.delta)
{
   //-Interaction Fluid-Fluid.
   InteractionForcesFluid<tker,ftmode,tvisco,tdensity,shift>
      (t.npf,t.npb,false,Visco,t.divdata,t.dcell,t.spstau,t.spsgradvel, ... );


   // There is no need to write the data back here.

   //-Interaction Fluid-Bound.
   InteractionForcesFluid<tker,ftmode,tvisco,tdensity,shift>
      (t.npf,t.npb,true,Visco*ViscoBoundFactor,t.divdata,t.dcell, ... );
}
```

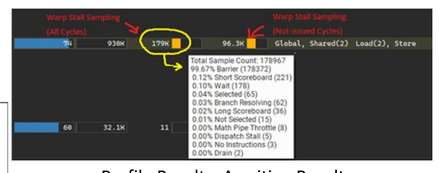
- Consecutive calls to the same function for different particle attributes (Bound / Float / Fluid)

V03: Avoiding synchronization on reduction with loop division

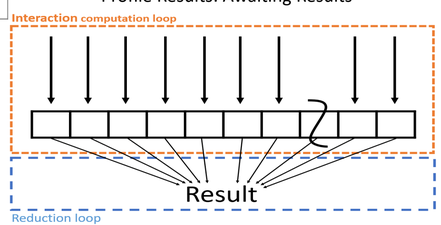
- OpenACC provides reduction directives, which are convenient but carry the risk of performance degradation due to synchronization.

```
#pragma acc parallel loop reduction(+:sum)
for (int i = 0; i < N; i++) {
    sum += data[i];
}
```

- Avoid synchronization by using temporary arrays and post-reduction aggregation, accelerating the main loop.
- Disadvantage: Increased global memory access.

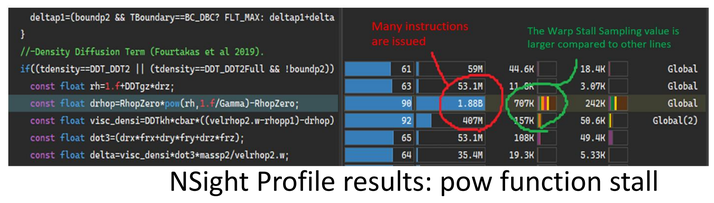


Profile Results: Awaiting Results



V04: Reducing instruction latency through formula replacement

- The pow function is used for the density diffusion term.
- Compared to other instructions, it had approximately five times more stall cycles.



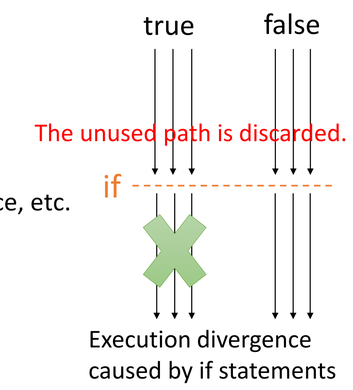
NSight Profile results: pow function stall

- Replacing pow with operations using log and exp reduces the number of instructions; precision requires further consideration.

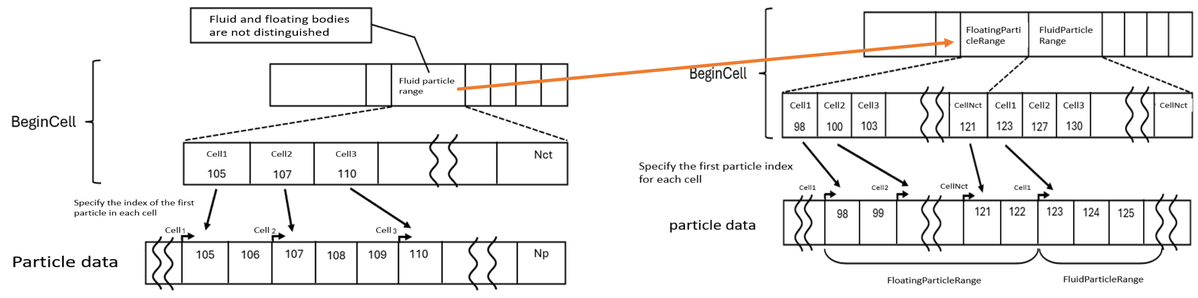
$$\sqrt[n]{x} = \exp(\frac{1}{n} \cdot \log x)$$

V05: Static resolution of particle attribute if statements via templates

- In thread-level parallelism, if statements cause execution divergence.
- On GPUs, both the taken and non-taken paths of an if statement are executed.
- Conditions inside loops
  - Execution configuration: kernel structure and the algorithms used
  - Particle attributes: Fluid, Bound, Float
  - Computational branching: skipping computations based on distance, etc.
- Previous Research: Template-Based Solution for Access Patterns in Adaptive Grid Partitioning [5]
- Conditional Branches Solved by Templates: Particle Attributes... Fluid, Boundary, Floating



Execution divergence caused by if statements



Evaluation of four optimizations

Evaluation Environment

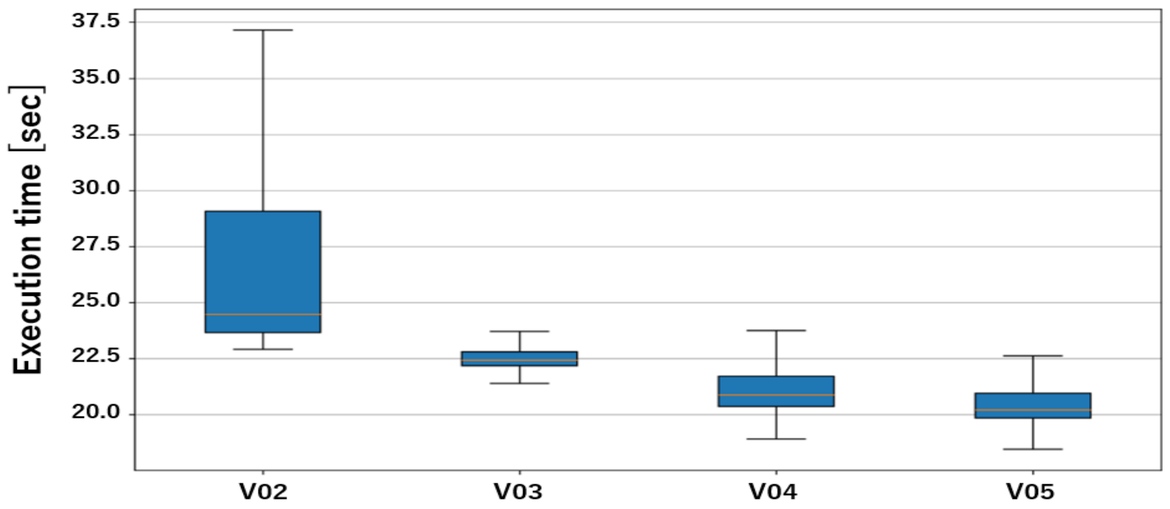
- Measure processing time and evaluate speed up of four optimizations over the baseline
- Two different cases are used to assess the generality

OS	Ubuntu 20.04.6 LTS
CPU	AMD EPYC 7343 3.2GHz
GPU	NVIDIA A100 PCIE 40GB
Mem	HMA82GR7CJR8N-XN DDR4 16GB 3200MHz x8

Evaluation Machine

GPU Driver	565.57.01
CUDA Driver API	12.7
CUDA	12.0
NVIDIA HPC SDK	23.1
OpenACC	2.7
Compiler Flags	-O3 –std=c++0x –fopenmp –acc –Minfo=accel -Minline

Compilation Environment



Kernel computation time per time step for each optimization

Name	01_Dambreak		11_Floating	
	Time[s]	Speedup Ratio	Time[s]	Speedup Ratio
V01	7946.77	1.00	16789.40	1.00
V02	1096.15	7.25	2051.15	8.19
V03	953.09	8.34	1872.79	8.96
V04	903.85	8.79	1883.14	8.92
V05	846.70	9.39	1968.31	8.53

Total computation time and speedup ratio

References

[1]高田貴正、新田知生、 大野和彦：“Sph法による流体解析のgpu上での高速化”, ハイパフォーマンスコンピューティングと計算科学 シンポジウム論文集,第2017巻, pp. 26–35 (2017).
[2] J. J. Monaghan: “Smoothed particle hydrodynamics”, Reports on Progress in Physics, 68, 8, p. 1703 (2005).
[3] J. M. Domınguez, G. Fourtakas, C. Altomare, R. B. Canelas, A. Tafuni, O. Garcıa-Feal, I. Martınez-Estevez, A. Mokos, R. Vacondio, A. J. C. Crespo, B. D. Rogers, P. K. Stansby and M. G ´ omez-Gesteira: “Dualsphysics: from fluid dynamics to multiphysics problems”, Compu tational Particle Mechanics, 9, 5, p. 867–895 (2021).
[4] S. Long, X.-W. Guo, X. Fan, C. Li, K. Wong, R. Zhao, Y. Liu, S. Zhang and C. Yang: “Paralleldualsphysics: sup porting efficient parallel fluid simulations through mpi enabled sph method”, Proceedings of the 51st Interna tional Conference on Parallel Processing, ICPP ’22, ACM, p. 1–11 (2022).
[5] 長谷川雄太, 青木尊之：“ステンシル計算の高速化のため のc++テンプレートによるgpuカーネル生成”,Technical Report 13, 東京工業大学 (2015)