§36. Nuclear Fusion Simulation at Exascale

Boku, T. (Univ. Tsukuba), Idomura, Y. (JAEA), Yagi, M. (JAEA), Naitou, H. (JAEA), Nakajima, N., Todo, Y., Watanabe, T., Sakagami, H., Nuga, H. (Univ. Tsukuba), Fujita, N. (Univ. Tsukuba)

i) Introduction In fusion plasmas, there are a lot of physical phenomena should be described by the kinetic theory, such as the effect of fast particles generated by fusion reactions or RF waves and the turbulence phenomena driven by the drift wave. This is because that the conventional fluid simulation, which requires less computational resources than the kinetic simulation, can not describe the physics accurately. Because of this reason, a number of kinetic simulation codes, which require huge computational resources, have been developed. Moreover, since there are a lot of phenomena that have various time and spatial scale in fusion plasmas, the analysis of the fusion plasma requires several simulation codes and the analysis should be done self-consistently. This process also requires extremely huge computational resources. Therefore massively parallel and massively large-scale simulation codes are required for the fusion simulation. The aim of our project is the development and enhancement of the fusion simulation code to be suitable for the coming exa-scale generation.

ii) Porting As the first step of this project, we tried to port the conservative global gyro-kinetic toroidal full-f 5D Vlasov code, GT5D[1)], which is already performed on JAEA Altix3700Bx2 system, into NIFS SR16000 system and Univ. of Tsukuba HA-PACS system.

iii) Diagnostic At the next step, we analyzed the computation performance of GT5D, which is parallelized by MPI and OpenMP. Table: I denotes the part of the flat profile of GT5D with operation check parameters. It is found that only two function, MAIN and l4dx_s, cost large portion of calculation time, around 90% of the whole calculation time. In the case of MAIN, since the MAIN function of GT5D includes much of processing (it includes about 2000 lines.), it should be divided into some functions in order to ease to tune and accelerate the code. Therefore the function which is a largest consumer of computational resources is l4dx_s, and it should be tuned at first.

% time	cumulative seconds	self seconds	calls	self Ks/call	total Ks/call	name
47.73	6416.68	6416.68	793804	0.00	0.00	l4dx_s_
40.87	11910.81	5494.14	4	1.37	3.36	MAIN__
10.05	13261.84	1351.03	794604	0.00	0.00	bcdf_

Table I: The part of the flat profile of GT5D is shown. This test calculation is executed with 4 MPI processes and single thread.

msec/call	CPU 1 Thr.	CPU 8 Thr.	1 GPU
l4dx_r_	134.3	23.4	16.6
l4dx_s_	163.7	28.4	33.2
l4dx_l_	306.0	39.7	102.7
l4dx_nl_	517.2	67.9	120.7

Table II: The comparison of calculation time among single CPU, multi-thread CPU, and single GPU on HA-PACS. HA-PACS includes 8 CPUs in one node.

msec/call	CPU 8 Thr.	1 GPU
timedev_1	18.0	5.5
timedev_2	17.5	8.7
timedev_3	21.1	9.8
timedev_4	21.3	11.2
timedev_5	21.5	11.3
timedev_6	22.7	7.2
timedev_7	17.9	5.5
timedev_8	27.5	8.9
timedev_9	20.2	10.6

Table III: The comparison of calculation time between 8-thread CPU, and single GPU.

iv) Acceleration Here we implement the GPU acceleration for l4dx_s and related functions by using PGI Fortran compiler. At this step, we did no tuning for speed up, but we only confirm the GPU calculation is executed precisely. Table II shows the comparison of calculation time among three cases, using single CPU, 8 thread CPU, and 1 GPU on HA-PACS, Univ. of Tsukuba, for each function. It is found that GPU execution is about 3 $\sim$ 8 times faster than 1 CPU execution. On the other hand, it is slower than 8 threads CPU execution. This is because we still do not tune the code for GPU calculation and it is expected that the code can get the higher performance due to appropriate tuning for GPU.

Next, we divided MAIN function into one MAIN function and 9 sub functions, which are named as timedev_1 $\sim$ timedev_9 and we implemented GPU acceleration on each sub functions. Table III shows the speed up of each sub functions by introducing GPU acceleration. Nevertheless the sub functions are not tuned appropriately, GPU calculation performance become higher than that of 8 threads CPU.

v) Future work In order to achieve further acceleration and obtain good scalability, we proceed to implement GPU accelerator, to search the possibility of hiding communication and *etc.*

1) Y. Idomura, M. Ida, N. Aiba, S. Tokuda, Computer Physics Communications, **179** (2008)