Tests of standard and OpenCL versions of ADDA on several Windows machines

Maxim Yurkin and Michel Gross, 18.06.2024

The command lines used are (together with adda.exe or adda_ocl.exe):

```
-grid 128 -size 10 -ntheta 10 -iter bicg -init_field zero
```

and tested additionally ocl version with OCL_BLAS (executable named adda_ocl_blas.exe, "+OB" in the column name), which has optimized Bi-CG iterative solver. The ntheta option limits the number of scattering angles for far-field computations (to be unaffected by https://github.com/adda-team/adda/issues/226). We also turned off "intelligent" initialization of the starting vector in the iterative solver, since it currently employs additional matrix-vector product (with buffer exchanges). This command line results in 42 iterations. We ran each binary three times and selected the one with the smallest time for "Internal fields". This corresponds to unpredictable behavior of Windows – some background processes may slow down the main calculation, still "init OpenCL" time is even less stable.

	i7 - 7700 HQ	HD 630	HD 630 +OB	GTX 1050	GTX 1050 +OB	Ultra 7 165H	Arc 128EU	Arc 128EU + OB	RTX 2000 Ada	RTX 2000 +OB	E5-2696	Titan V	Titan V +OB	E5-2673	GTX 760	GTX 760 +OB
Total wall time	**67.8**	**42.4**	**45.3**	**21.5**	**20.3**	**57.3**	**12.0**	**11.8**	**10.5**	**8.5**	**147.9**	**12.0**	**8.0**	**90.9**	**19.7**	**16.4**
Initialization time	**5.1**	**4.2**	**4.2**	**3.7**	**4.0**	**3.0**	**1.9**	**1.8**	**3.2**	**2.4**	**11.4**	**5.9**	**5.2**	**8.0**	**3.7**	**4.2**
init OpenCL	--	1.1	0.9	1.5	1.5	--	0.3	0.2	1.5	0.8	--	0.8	0.6	--	0.2	0.2
init Dmatrix	2.2	2.4	2.6	2.1	2.4	1.5	1.6	1.6	1.6	1.6	5.5	5.0	4.5	3.2	3.5	4.0
FFT setup	2.8	0.6	0.7	0.0	0.0	1.5	0.0	0.0	0.1	0.0	5.8	0.0	0.0	4.9	0.0	0.0
Internal fields	**62.1**	**38.0**	**40.9**	**17.3**	**16.1**	**54.2**	**10.0**	**9.9**	**7.1**	**6.0**	**135.9**	**5.5**	**2.4**	**82.5**	**15.7**	**11.9**
matvec products	60.6	36.5	0.0	15.8	0.1	53.3	9.0	0.0	6.1	0.0	133.1	2.2	0.1	79.2	12.4	0.1
init solver	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.1	0.1	0.2
FP64, TFLOPs	0.06	0.1		0.06		0.08	N/A		0.2		0.06	7.5		0.03	0.1	
FP32, TFLOPs	0.12	0.4		1.9		0.16	4.6		13		0.12	15		0.06	2.4	
Bandwidth, GB/s	38	38		112		132	132		256		19	650		22	192	
reference	link	link		link		link	link		link		link	link		link	link	

A few details on the used CPUs and GPUs is also given in the table (FP64 and FP32 indicates theoretical max float performance in double and single precision, respectively). CPU flops (of a single core) are estimated based on maximum working frequency and flops/cycles. Memory bandwidth for GPUs integrated with Intel CPUs (HD630 and Arc) is the same as for the CPU (which also depend on the used RAM). The latter values are rough estimates. Moreover, Arc 128EU seems to not support double calculation, so it is not clear how it works (and so fast). Maybe it somehow gets help from the CPU or unofficially supports such calculations.

The conclusions so far:

- Init Dmatrix is currently done on the CPU (https://github.com/adda-team/adda/issues/248), that's why it is slow and becomes especially painful for a combination of simple CPU with powerful GPU.
- Sequential mode spends significant time on FFT setup (approximately 1 iteration), this is known (discussed in the manual) and can be removed. However, such setup (with related optimizations) can be beneficial for large enough number of iterations.
- Initialization of OpenCL have relatively large variability (the values in the table are actually on the lower side), we have no control over that.
- When using OCL_BLAS, the timing of matrix-vector products is completely inadequate, so we can only look at the total times (https://github.com/adda-team/adda/issues/199#issuecomment-2148403620). That's why it is colored orange in the table.
- OCL_BLAS definitely helps, but we have no detailed understanding due to the previous comment. There are two ways, in which it benefits. First, it accelerates the linear-algebra operations (by doing them on GPU instead of CPU) – this corresponds to the difference between times for "Internal fields" and "matvec products". Second, it eliminates buffer load/unload operations around matvec, currently this is included in matvec timing for non-yellow columns. However, even for Titan V the "internal fields" time for OCL_BLAS does not go below the "matvec" time for standard ocl.
- Consequence of the previous one, moving memory between CPU and GPU (at each iteration, as in standard) is NOT a bottleneck. This may be due to two factors: 1) fast GPUs tend to have fast memory speed (including CPU-GPU exchange) and 2) the FFT is a global-exchange operation (more so for 3D FFT), which has to utilize the "outer" memory of the GPU with a speed comparable to CPU-GPU exchange. So, the 3D FFT seems to be the real bottleneck, which is hard to address.
- If there is significant difference between "internal fields" and "matvec" in standard ocl regime, then OCL_BLAS is a recommended approach. In many cases it can save the time comparable to this difference.
- The obvious candidate for GPU characteristic, which defines ADDA performance, is FP64, however, this is not universally so. Larger correlation (with inverse computational time) is observed for memory bandwidth. Specifically, HD630 has comparable FP64 to that of both GTX cards, but much smaller bandwidth – resulting in much poorer performance. Similarly, TITAN V has 37 times larger FP64 than RTX 2000 Ada, but only 2.5 larger bandwidth. And the latter roughly corresponds to the performance improvement. However, a single bandwidth is also misleading, since various levels of cache are important – this is evident from comparing different CPUs (which is further complicated by variable frequency).
- The previous point implies, that it would be hard to get huge improvement from switching to single precision (https://github.com/adda-team/adda/issues/119). Still, two-times acceleration is guaranteed (for the same bandwidth).

Overall, the more powerful is the GPU – the more painful most of the existing issues become. Solution of all (or many) of them is required to obtain remarkable performance. Further, the most suitable GPUs seems those with highest memory bandwidth.