GROMACS - Task #2792

Improvement of PME gather and spread CUDA kernels

12/05/2018 12:36 PM - Jonathan Vincent

Status:	New
Priority:	High
Assignee:	
Category:	
Target version:	
Difficulty:	uncategorized

Description

There are some performance issues with the CUDA gather and spread kernels

- The spread kernel is limited by global memory instruction throughput (i.e. the hardware units dealing with global memory are running at capacity)
- There is a memory latency issue for the initial read of the charges and positions into shared memory in the spread kernel. This would be the new problem once the global memory problems are fixed.

The obvious solution to the first part is to recalculate the spline coefficients and derivatives in the gather kernel rather than writing them into global memory and reloading them.

For the second problem, the atom charges and positions are loaded into shared memory because of the way the spline coefficients are calculated. The number of calculations is DIM*atomsPerBlock*Order (3*2*4=24 currently). This is then spread over either 6 threads (with PME_GPU_PARALLEL_SPLINE=0) or 24 threads (with PME_GPU_PARALLEL_SPLINE=1). The way the atoms are allocated to the threads is different to the spread charges which is why the charges and positions need to be in global memory currently.

If the thread ordering of the spline calculations is changed so that the same threads deal with just one atom both when calculating the spline coefficients and when spreading the charges the load to shared memory could be eliminated. This would potentially make going to higher orders more complicated, but assuming order=4 is all we need for a reasonably long time seems like a safe assumption.

Using a larger spline order allows for a smaller PME grid size so would be most appropriate for larger systems, and at that point it should be appropriate to run with fewer threads per atom. STMV with 1,000,000 atoms for example has around 75 waves on a V100 with order*order (16) threads per atom, so instead using 1 thread per atom and 5 waves should be reasonable, and make using higher order splines simpler. For smaller sizes this is less useful because of the tail effects.

In the short term changing the code so that PME_GPU_PARALLEL_SPLINE=1 is turned on (which also requires two additional syncwarps after the shared memory operations with the Volta threading model) helps with the examples I have. It was not obvious to me why PME_GPU_PARALLEL_SPLINE=0 helps in any situation currently. The spline work is gated by localCheck which for PME_GPU_PARALLEL_SPLINE=0 requires orderIndex (= threadWarpIndex / (atomsPerWarp * DIM)) to be 0, so only 6 threads per warp are active with DIM=3 and atomsPerWarp=2. The alternative is 24 active threads in the warp (with order index 0-3). So the penalty for PME_GPU_PARALLEL_SPLINE=1 is you do some redundant work and use 24 threads in the warp rather than just 6, which seems unimportant. The advantage is that all 24 threads write to global memory at the same time, which improves coalescing compared to 6 threads writing 4 times each. This seems like a simple change that for my runs on Cellulose on a V100 decreased the spread kernel time from 470 ns to 400 ns (using a separate PME GPU).

Subtasks:

Task # 3185: Update PME CUDA kernels to allow a different number of threads per atom in...	**New**
Task # 3186: Update Constant/Variable naming in the PME GPU kernels.	**In Progress**
Task # 3187: Template updated PME kernels using threads per atom	**New**
Task # 3188: re-enalble parallel spline calculation for #threads/atoms > 4	**Closed**
Task # 3189: implement heuristics for switching between different spread/gather kernel ...	**In Progress**
Task # 3212: Update regression tests for new kernel flavours	**New**
Task # 3445: create heuristic for c_skipNeutralAtoms	**New**
Task # 3446: apply maintainability updates across all GPU kernels	**New**

Associated revisions

Revision 22118220 - 10/30/2019 01:21 PM - Jonathan Vincent

Update PME CUDA spread/gather

Adds addtional templated kernels to the CUDA spread and
gather kernels. Allowing the use of 4 threads per atom instead of
16 and allowing the spline data to be recalculated in the spread
instead of saved to global memory and reloaded.

The combinations mean we have 4 different kernels that can be called
depending on which is most appropriate for the problem size and
hardware (to be decided heuritically). By default existing method is
used (16 threads per atom, saving and reloading of spline data).

Added an additional option to disable the preloading of charges and
coordinates into shared memory, and instead each thread would
deal with a single atom.

Removed the (currently disabled) PME_GPU_PARALLEL_SPLINE=1 code
path.

Refs #2792 #3185 #3186 #3187 #3188

Change-Id: Ia48d8eb63e38d0d23eefd755dcc228ff9b66d3e6

History

#1 - 01/04/2019 11:32 PM - Jonathan Vincent

Optimisation at https://gerrit.gromacs.org/8908 this is the larger change, removing the saving of the co-efficients in the spline and spread kernel (recalculating in gather), and other changes.

Kernel timings using 4 gv100.
nvprof gmx mdrun -ntmpi 4 -ntomp 10 -pme gpu -nb gpu -pin on -nsteps 1000 -v -npme 1 -notunepme

Note Villin only done on 2 GPUS (one PP one PME) as was not valid DD with 3 PP ranks.

		villin	rnase Dodec	ADH Dodec	Cellulose	STMV
2018.4	Spline and sp	8.8260us	22.638us	111.22us	401.18us	1260.2us
	Gather	3.7310us	7.1570us	35.408us	132.75us	364.36us
Modified	Spline and sp	7.4820us	19.077us	91.102us	317.24us	960.90us
	Gather	6.0960us	8.6500us	41.488us	150.82us	316.12us

The spline and spread kernel is limited by global memory instruction throughput. The gather kernel is limited by memory latency.

Generally improvement is larger for larger problems. Villin gather time likely hurt by using 4 threads per atom rather than 16, which would create a tail effect.

#2 - 01/07/2019 11:56 AM - Jonathan Vincent

Ok added some timings for turning on PME_GPU_PARALLEL_SPLINE=1 as is done in https://gerrit.gromacs.org/#/c/8921/

This is a draft change. For Villin this is better than the above. rnase and ADH it is approximately the same in total time for both kernels. Cellulose and STMV are better with the more complicated change.

		villin	rnase Dodec	ADH Dodec	Cellulose	STMV
2018.4	Spline and sp	8.8260us	22.638us	111.22us	401.18us	1260.2us
	Gather	3.7310us	7.1570us	35.408us	132.75us	364.36us
modified	Spline and sp	7.7690us	20.076us	96.995us	349.99us	1087.7us
	Gather	3.8780us	7.2840us	35.782us	133.10us	363.77us

#3 - 06/27/2019 05:57 PM - Jonathan Vincent

So there are some issues with the PME unit tests with the patch at https://gerrit.gromacs.org/8908.

With the changed method we are breaking some of the assumptions in the unit tests, e.g. the gather tests assume that the grid-line indices and spline coefficients will be read from memory, rather than recalculated from the atom positions and the grid. As the data is not internally consistent this is a problem as we do not reproduce the random splines and derivatives, and so the answer for the tests changes.

Potentially also there are problems with the other tests, in that what is tested for is not output anymore, so we will need a (presumably templated) path that does output the grid-line indices and splines etc to global memory so that they can be checked by the unit tests.

The simplest change for the gather would be to use the scatter values with computed splines as they would then be internally consistent and valid.

#4 - 07/04/2019 12:57 PM - Jonathan Vincent

One other question, is there a redmine for the reordering changes Berk was looking at?

Would be good to look at how they combine with the above better.

Also need to find where the performance crossover is between villin at 5k atoms and rnase at 20k atoms is, and how the reordering changes affect that. Which other hardware is this also important for?

#5 - 07/10/2019 11:36 AM - Szilárd Páll

Jonathan Vincent wrote:

> With the changed method we are breaking some of the assumptions in the unit tests, e.g. the gather tests assume that the grid-line indices and spline coefficients will be read from memory, rather than recalculated from the atom positions and the grid. As the data is not internally consistent this is a problem as we do not reproduce the random splines and derivatives, and so the answer for the tests changes.

I suggest to update unit test to make the data internally consistent. Note that we have the OpenCL codepath too running the same tests so make sure to either change those kernels too or make sure that the tests work with both computed & stored as well as recomputed splines.

> Potentially also there are problems with the other tests, in that what is tested for is not output anymore, so we will need a (presumably templated) path that does output the grid-line indices and splines etc to global memory so that they can be checked by the unit tests.

Sure, I suggest to go ahead with templating and add (back) a conditional store of the spline parameters, calling that flavor of the kernel in tests.

#6 - 07/10/2019 11:46 AM - Szilárd Páll

Jonathan Vincent wrote:

> One other question, is there a redmine for the reordering changes Berk was looking at?
>
> Would be good to look at how they combine with the above better.

There is no new change (Berk was looking into enabling DD sorting without DD), it's the same sorting that we've talked about in the past. There was no redmine I filed one: #3031; should I assign it to you?

> Also need to find where the performance crossover is between villin at 5k atoms and rnase at 20k atoms is, and how the reordering changes affect that. Which other hardware is this also important for?

Same as before + all new hardware.

#7 - 08/14/2019 03:02 PM - Jonathan Vincent

To pass the unit tests it is necessary to write out the theta and grid lines information. Similarly for the gather it is necessary right now to read in the gridlines etc and use those.

The simplest change here is to template the code so that it optionally reads in or writes out similar to the existing version. This allows the potential to fall back to the old way of writing and reading back if it would be advantageous. Need some work on heuristics there though.

The theta is currently stored with AtomsPerWarp values sequentially, so changing from 16 to 4 threads (and the subsequent increase from 2 atoms per warp to 8) changes the data layout. Hence without changing the data layout it is an issue to have different numbers of atoms per warp between the spread and gather.

The simplest change to fix that is to also update the spread to use 4 threads per atom, which similar to the above should be helpful for large systems, but needs to be investigated together with the effect on smaller systems.

Again potentially we could fall back to the old version of 16 threads per atom for smaller systems, but then we would have a lot of code paths, and it would add a lot of complication.

#8 - 08/15/2019 04:12 PM - Jonathan Vincent

Water boxes
Command gmx mdrun -ntmpi 4 -ntomp 10 -pme gpu -nb gpu -pin on -nsteps 1000 -v -npme 1 -notunepme

V100

atoms	960	1533	3000	6000	12000	24000	48000	96000	192000	384000	768000	1536000	3072000
Scatter	4.612	4.555	6.898	9.267	15.072	25.818	45.279	80.346	152.92	293.27	581.64	1133.5	2246.5
Gather	5.247	5.398	5.665	5.932	6.733	10.762	17.571	30.886	60.984	118.01	229.82	447.62	878.08
Total	9.859	9.953	12.563	15.199	21.805	36.58	62.85	111.232	213.904	411.28	811.46	1581.12	3124.58

```
time/atom  0.0102  0.0064  0.0041  0.0025 0.00181  0.00152 0.0013  0.00115  0.00111  0.00107  0.00105    0.001
02    0.00101
```

Master -0c26c550ed55e12b77954dd0e8c5d956421ae501

```
atoms      960     1533    3000     6000     12000   24000    48000    96000     192000    384000   768000 1536000  30720
00
Scatter    3.503   4.261   6.08     9.783   16.978   31.076   56.474   110.47    196.04    379.65   796.98  1510.3   3003.
12
Gather     2.635   2.868   3.22     4.111    6.05     9.409   18.124    33.929    60.512   113.66   227.6    449.61    916.
93
Total      6.138   7.129   9.3     13.894   23.028   40.485   74.598   144.399   256.552   493.31 1024.58 1959.91   3920.
05
time/atom 0.0063 0.0046 0.0031   0.0023   0.0019   0.0016   0.00155   0.00150   0.00133 0.00128 0.00133 0.00127   0.0
0127
```

So for v100 we have a crossover at around 10,000 atoms.

#9 - 08/15/2019 05:03 PM - Szilárd Páll

I still have some uncertainty. There are two different changes in your proposed code: the reduction in intra-warp parallelism and the recomputing of spline parameters in spread and gather (to avoid load/store). You are measuring the impact of *both* changes combined. The question is whether / where (in which regime) does it makes sense to reduce the exposed parallelism to 8 atoms/warp? I do not think we can exclude the option that in fact we see the effect of a large improvement (due to recomputing spline coeffs) and a regression in a wide range of interest (due to reduced amount of parallelism). Also, we do not need to optimize the flush of spline parameters (anymore) as this is needed in testing only, right?

Related to #3031: I assume you are using sorted particle data (it seems you are using DD?); how do this compare to unsorted data?

#10 - 08/16/2019 12:11 AM - Jonathan Vincent

Ok those is a good points.

Yes there is DD on the PP side as we have 3 PP ranks and 1 PME rank.

Re-ran with 16 threads per atom.

```
atoms          960     1533    3000    6000    12000   24000    48000     96000     192000  384000   768000   15
36000  3072000
Scatter        3.374   4.066   5.286   8.124   13.97   25.577   45.168    82.56     161.81   306.53   641.05   11
67.4    2310
Gather         3.467   3.676   4.109   5.03     7.548  12.044   21.138    36.32      73.541  140.29   273.94    5
39.97   1073.1
Total          6.841   7.742   9.395  13.154   21.518  37.621   66.306   118.88    235.351  446.82   914.99   17
07.37   3383.1
time/atom      0.00712 0.00505 0.00313 0.00219  0.00179 0.00156  0.00138   0.00123  0.00122  0.00116  0.00119
 0.00111   0.00110
```

This improves the performance below 10,000 and moves the crossover to around 4,000 atoms. The performance at 24,000 atoms and above is worse than with 4 threads per atom.

Just to confirm the current master branch uses unsorted data without DD? Will re-run with one PME rank and one PP rank.

#11 - 08/16/2019 12:42 PM - Jonathan Vincent

One solution would be to run with 16 threads, and fall back to the save/reload for small sizes.

From my code it would be simple to add an extra logical to the pme_gpu_t struct and look at that when deciding to which version of the kernel to call. The code is already there for the unit tests, so all that would be needed was to add an extra bool and some control code to decide what the value should be.

4 threads/atom works for large sizes, but supporting both is more difficult.

#12 - 08/21/2019 05:10 PM - Jonathan Vincent

OK so I ran some on the RTX 2080 as well.

Looking at the total time for the spline_and_spread kernel plus the gather kernel in various configurations. I have spreadsheet with the individual times, but that is a bit complex to post here.

The command was

```
gmx mdrun -ntmpi 4 -ntomp 10 -pme gpu -nb gpu -pin on -nsteps 1000 -v -npme 1 -notunepme
gmx mdrun -ntmpi 2 -ntomp 10 -pme gpu -nb gpu -pin on -nsteps 1000 -v -npme 1 -notunepme

                                  0.96    1.5    3       6       12     24     48      96      192    384    7
68     1536    3072
2019.2, 4 ranks, RTX 2080         6.055   7.344  11.436  17.398  33.652 62.915 107.11  204.226 397.89 765.08 1
497.4  2860.3  5762.8
2019.2, 2 ranks, RTX 2080         5.991   7.345  11.492  17.531  32.667 61.885 117.638 237.982 476.64 941.64 1
849.64 3502.3  6982.1
Master, 4 ranks, RTX 2080         7.2233  8.5901 11.1816 17.369  31.557 62.42  108.455 212.202 412.4  768.34 1
505.26 2924.16 5918.74
Master, 2 ranks, RTX 2080         6.2285  7.4324 11.2968 17.487  30.904 62.122 118.11  241.019 477.51 942.31 1
853.08 3516.2  7023.8

4 threads, 4 ranks, RTX2080       8.989  10.669  13.148  17.783  29.783 52.76   88.784 162.134 307.28 561.13 1
119.25 2442.23 5119.9
4 threads, 2 ranks, RTX2080       9.2724 10.7427 12.957  17.5215 30.164 52.704  95.529 184.097 352.65 688.31 1
367.78 2675.68 5331.47
16 threads, 4 ranks, RTX2080      6.1631  7.6046 11.1591 17.2047 31.48  62.974 109.468 214.68  409.84 777.16 1
522.51 2909.97 5809.3
16 threads, 2 ranks, RTX2080      6.7712  8.3065 12.4175 18.8315 33.106 64.414 121.744 244.796 488.8  963.91 1
887.86 3577.66 7141.29
16 th, save + L, 4 ranks, RTX2080 6.1631  7.6046 11.1591 17.2047 31.48  62.974 109.468 214.68  409.84 777.16 1
522.51 2909.97 5809.3
16 th, save + L, 2 ranks, RTX2080 6.138   7.588  11.285  17.293  30.717 62.169 118.734 244.632 485.0  960.21 1
888.86 3583.05 7158.33

4 threads, 4 ranks, V100          9.859   9.953  12.563 15.199   21.805 36.58   62.85  111.232 213.90 411.28
811.46 1581.12 3124.58
16 threads, 4 ranks, V100         6.841   7.742   9.395 13.154   21.518 37.621  66.306 118.88  235.35 446.82
914.99 1707.37 3383.1
16 th, save + L, 4 ranks, V100    5.9998  7.056   9.196 13.99    23.529 40.906  73.729 140.706 259.01 497.57 1
007.14 1946.6  3878.48
Master, 4 ranks, V100             6.138   7.129   9.3   13.894   23.028 40.485  74.598 144.399 256.55 493.31 1
024.58 1959.91 3920.05
```

What it looks like right now for those two is we need 4 threads per atom and the new way above 10,000 atoms and to fall back to the old way below. Using 16 threads is quite bad on the RTX2080.

On RTX 2080 we are losing a lot without the DD sorting, this is reduced significantly by the changes as well. Will run the same on the V100.

The master version is 0c26c550ed55e12b77954dd0e8c5d956421ae501 as that was current when I started.

Anyway if having a switch of ways at 10,000 atoms makes sense then probably it should autotune to that, with a command line flag to override?

Also for the unit tests do we need to template a 4 threads per atom version that also writes out the data? I guess so. It would be simpler if that was not needed.

#13 - 09/04/2019 01:51 PM - Szilárd Páll

Jonathan Vincent wrote:

> One solution would be to run with 16 threads, and fall back to the save/reload for small sizes.
>
> 4 threads/atom works for large sizes, but supporting both is more difficult.

There does not seem to be an inherent need to use 4 threads/atom in either spread or gather, is there? What makes it impossible to have 16 (or a flexible) number of threads/atom and still recompute splines?

#14 - 09/04/2019 03:37 PM - Jonathan Vincent

There is not an inherent need for 4 threads per atom.

There is an issue with the data format, where it interleaves the atom data. So if you have a different number of threads per atom (and hence a different number of atoms per warp/block) in the spread and the gather then we would have to re-write how the data is stored. i.e. you end up with data_1_atom_1, data_1_atom_2 ... data_1_atom_x, data_2_atom_1 ... where x is the number of atoms per block, i.e. we have an current need to have the same number of threads in the spread and the gather, but it is not important what that is.

Clearly this could be changed, but it would be a more disruptive change.

Right now it is using order or order*order threads per atom, with only order 4 supported currently. That seems somewhat intrinsic and would be harder to change.

For me the numbers above say that most of the benefit at large sizes is from 4 atoms per thread. On the RTX2080 recalculation the splines does not

do much by itself for example. With large numbers of atoms we have too many threads as only a small proportion can be resident at one time. By going to 4 threads per atom you incrase the effective occupancy and reduce the parallelism. But for large numbers of atoms we have plently of parallelism already so incresing the effective occupancy wins out. At small numbers of atoms it is the reverse.

Anyway yes the most sensible way right now seems to be to have it flexible as either order or order*order, have it flexible as either recalculate or save and reload. Then it can be autotuned or set with a command option.

I am a little concerned that we are getting to have a large number of versions of the code, but hopefully this is still not a huge problem.

My current thinking is that it makes sense to have two bools in the template

- Save+reload vs recalculate
- order threads or order*order threads

Then we should cover most options. That should be easy, and what I have been looking at doing.

Anyway would be good to know if that seems sensible to you, and if you have any suggestions on how the control code which decides which is the best of the 4 combinations to use would look like.

#15 - 09/04/2019 06:18 PM - Szilárd Páll

Jonathan Vincent wrote:

> Looking at the total time for the spline_and_spread kernel plus the gather kernel in various configurations.

Can you please describe the setups that correspond to the first column as well as the way you did the measurements. Does the code on gerrit support all those cases ?

> I have spreadsheet with the individual times, but that is a bit complex to post here.

It would be easier to analyze data if you could share either on google docs or csv if you've already imported/parsed it.

> gmx mdrun -ntmpi 4 -ntomp 10 -pme gpu -nb gpu -pin on -nsteps 1000 -v -npme 1 -notunepme
> gmx mdrun -ntmpi 2 -ntomp 10 -pme gpu -nb gpu -pin on -nsteps 1000 -v -npme 1 -notunepme

To me these seem equivalent from the point of view of PME (the only difference is 3 vs 1 PP). The other comparison (w/o sorting) requires that no separate PME rank is used. Unless of course there are PP ranks sharing a GPU with PME rank in the former case and concurrent kernels are enabled? That would make sense given that the 4 ranks numbers show higher kernel times -- which also means that the 4 ranks data point is not useful. I suggest to gather data with no DD and with 1PP + 1PME, concurrent kernels disabled.

> What it looks like right now for those two is we need 4 threads per atom and the new way above 10,000 atoms and to fall back to the old way below. Using 16 threads is quite bad on the RTX2080.

Before we can conclude that I have a few follow-up questions regarding the benchmark setup and code that I have no access to:

- How is the work laid out on the 16 threads/atom test ("16 threads, N ranks" data points)? In order to avoid wasting 12 of the 16 threads, 3x4 coefficients should be computed concurrently, then shuffled around.
- What other difference is there between the "16 threads, 2 ranks" and "16 th, save + L, 2 ranks" data points? I find it strange that the latter is actually faster than the former on both the V100 and RTX2080 for small cases -- if anything I'd expect that doing more arithmetic instead of memory ops is advantageous in those cases.
- How does atom data prefetching and "

> On RTX 2080 we are losing a lot without the DD sorting, this is reduced significantly by the changes as well. Will run the same on the V100.

Do you have the data on that difference? The above data does not allow such conclusion.

> Anyway if having a switch of ways at 10,000 atoms makes sense then probably it should autotune to that, with a command line flag to override?

It needs to be internal heuritsics -- if any. I'd however like us to take make sure that we are confident that we have the right data to start answering questions:
- 4 threads vs 16 threads per atom with recomputing spline coeff *with* making sure that the calculate_splines() does not let threads idle
- *serialized kernel execution* of single rank and multi rank so we can compare sorting vs no sorting
- PME_GPU_PARALLEL_SPLINE?
- atom data staging vs no staging?
- shared memory use in gather?

#16 - 09/05/2019 12:33 PM - Jonathan Vincent

The spreadsheet is at https://docs.google.com/spreadsheets/d/1Cxu-4KWu8YZDDxE8gjLB7ln3-65aYrBqa0mHjMsWLCE/edit?usp=sharing

Looking at the other points.

#17 - 09/07/2019 01:14 PM - Jonathan Vincent

To answer the more detailed questions

- No the current gerrit code does not support all configurations. I will fix that. I hacked the existing version to cover the other combinations. Need to join them together with some parameters, which should not be hard. Will template them all into a single piece of code and upload it.
- I have started running the correct without sorting stuff and added it to the spreadsheet linked above. Can paste some more here if it is useful.
- One of the effects of setting PME_GPU_PARALLEL_SPLINE is to use 12 threads instead of 4 to calculate the spline coefficients. You can see in the localCheck calculation.

  ```
  const int localCheck  = (dimIndex < DIM) && (orderIndex < (PME_GPU_PARALLEL_SPLINE ? order : 1));
  ```

  Currently just using the existing GPU_PARALLEL_SPLINE=0 implementation here. But you are right we should look at this more carefully.
- In light of the above will run 2019.2 with GPU_PARALLEL_SPLINE=1 (which requires some extra syncs to get around the "threads are threads" issues) on the set of water boxes which should help clarify things. We were seeing inconsistent effects from this previously on different cards, so need more data.
- Note you have a fragment *How does atom data prefetching and "* Not quite sure what you are asking here.

#18 - 09/12/2019 12:05 PM - Szilárd Páll

Jonathan Vincent wrote:

> The spreadsheet is at https://docs.google.com/spreadsheets/d/1Cxu-4KWu8YZDDxE8gjLB7In3-65aYrBqa0mHjMsWLCE/edit?usp=sharing

The data, code versions and run conditions are hard to decipher (in particular what is "new data")? Can you please point at the spread/gather kernel times for master original (16 th/atom with staging), 16 th/atom and 4 th/ atom w/o staging for V100 and 2080 (any data for GP102?). Can you please mark the with/without sorting data in the spreadsheet (also, there are still 4-rank runs there do those use a separate dedicated GPU for PME?)

#19 - 09/12/2019 12:23 PM - Szilárd Páll

Jonathan Vincent wrote:

> - Note you have a fragment *How does atom data prefetching and "* Not quite sure what you are asking here.

Sorry, didn't finish the question: the code in gerrit removes optimizations like the atom data prefetching (pme_gpu_stage_atom_data()). How that that affect performance?

#20 - 10/29/2019 11:20 AM - Jonathan Vincent

Ok so we have updated https://gerrit.gromacs.org/8908 so that it supports using either the spline order threads per atom(4) or the spline order squared (16) threads per atom.

This is templated so we have four combinations.

It is necessary to autotune which combination is used for the final calculation. The updated spreadsheet (and clearer) with the relevant data for this is at https://docs.google.com/spreadsheets/d/1X97MLJ5ys6WQmq5RQPzt000UQRKmbT98d_kCIPpaejE/edit?usp=sharing

Currently remaining

- Finish the code to select which kernel flavour should be used.
- Finish the P102 timing data

So future work

- Update to allow a different number of threads per atom in the gather and spread.
  - Currently this is not supported because of the data layout for the data that is saved in the spread and reloaded in the gather has the data for each atom interleaved, i.e. data_1 atom_1, data_1 atom_2 ... data_1 atom_N, data_2 atom_1 etc, where there are N atoms per block. So the data layout is dependent on the number of atoms per block. So the data ordering most likely has to be changed to allow this. Potentially the unit test code will also have to be updated as well as some unit tests just supply the spline and gridline data, which needs to be in the correct order
- Variable naming. As many of the variables/constants are also used in the OpenCL code they were not renamed. For clarity we should have consistent naming between the 4 threads per atom variables and the 16 threads per atom variables. Which requires changing the OpenCL code as well.
- As suggested by Szilard we could template on threads per atom instead of using a bool to choose between order (4) or order*order(16) threads per atom.

#21 - 10/29/2019 11:37 AM - Szilárd Páll

Please add subtask for the items you list as future work as well as an additional subtask for assessing block size -- assuming we have not tested that the current (is it 256?) is optimal.