Production-CL
library for iterative scientific calculations

Petr F. Kartsev
National Research Nuclear University MEPhI (Moscow Engineering Physics Institute)
Dept. No 70 (Physics of Solid-state and Nanosystems),
115409, Kashirskoe sh., 31, Moscow, Russia
PFKartsev@mephi.ru

Motivation

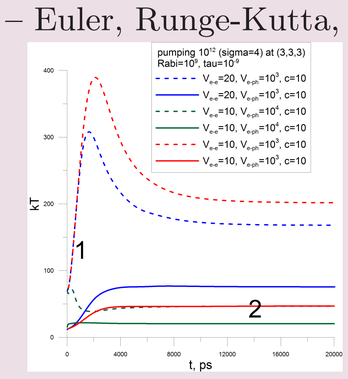
Our research group studies many problems of condensed-matter physics. They include, among others, superconductivity, Bose-Einstein condensation, optoelectronics[2], micromagnetism, atomic clusters. All these problems demand labor-costly calculations in iterative way. For example, (i) to solve a nonlinear problem or to find eigenvalues of the matrix, one typically applies some iteration procedure until the result converges[1]. (ii) To study numerically the evolution in time (molecular dynamics, kinetic equations, nonlinear optics), one repeats steps of implicit/explicit Euler, Runge-Kutta, Crank-Nicolson or another finite difference methods. (iii) Applying the Monte Carlo approach, one uses the law of large numbers, by repeating some stochastic procedure for large number of times. As a result, our typical calculation consists of large amount of calculations made with some repeating procedure.
Typical runtime ranges from several minutes to days and weeks and demands periodic writing of intermediate data, to prevent losing work, for control and debug purposes.
We found OpenCL to be the promising way to increase the simulation performance in our problems. From our point of view, OpenCL is preferable to CUDA, due to wider range of devices. The second cause is the better performance in double precision (required for our type of scientific calculation) on (at least some) AMD devices.
On the other side, the development of OpenCL program proved to be very slow and error-prone, due to long sequence of calls associated with any action, such as running the kernel, or reading/writing data. However the sequence looks almost the same for all kernels in the calculation. This fact allowed us to develop the helper library performing all the typical tasks expected in the iterative calculations. We named it 'Production-CL', meaning its main use in 'production' environment with long runs of calculation and expected level of data safety.
This report is devoted to main ideas and component parts of the PCL library.

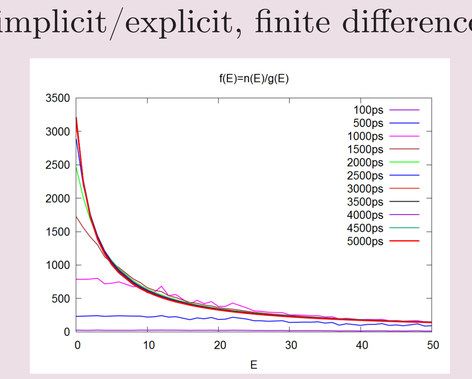
Typical workflow

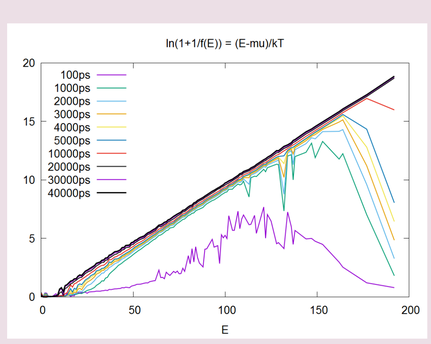
- Long run: from hours to days and weeks. **State save and load** to prevent losing work
- From time to time: **save work data** (arrays, graphs, signals etc. for control and debug)

Condensed-matter problems to solve with iterations

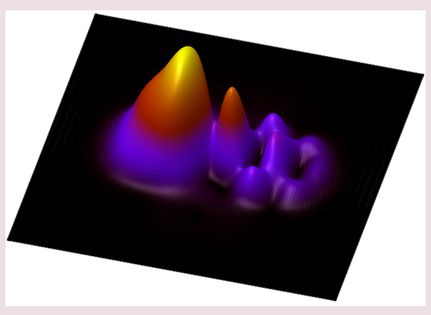
- Study the evolution in time (molecular dynamics, kinetic equations, nonlinear optics)
  – Euler, Runge-Kutta, implicit/explicit, finite difference methods...

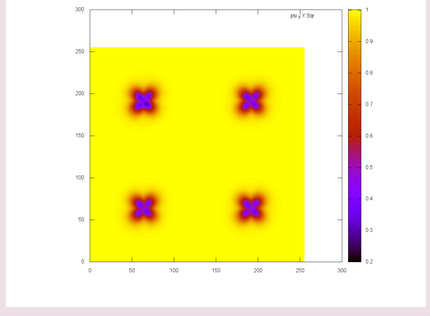


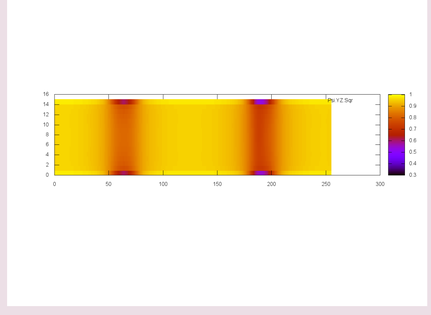




- Solve problem with iterative numerical method (nonlinear equations, eigenvalues, ...)
  – Gross-Pitaevskii, Ginzburg-Landau, Crank-Nicolson, pseudoviscosity method...







Example of PCL code

```
pcl_array x;                                                    // integer handles
pcl_kernel k0;
pcl_batch b;
int startIterations=0;
if (!pcl_loadState("state.dat")) {
        pcl_array y = pcl_createArray( 1048576, 0 );            // 0 = private
        k0 = pcl_createKernel( "init", 1, y);                   // one array: y
        x = pcl_createArray( 1048576, 1 );                      // 1 = public
        pcl_kernel k1 = pcl_createKernel( "calc", 2, x, y);     // two arrays: x,y
        ...
        b = pcl_createBatch();
        pcl_addKernelToBatch( b, k1, 1, 5 );                    // one additional argument=5
        pcl_addKernelToBatch( b, k1, 1, 7 );                    // one additional argument=7
        ...                                                     // more kernels, if needed
} else {
        ... // get x, k0, b, startIterations
}
pcl_runKernel( k0, 1, 2017 );                                   // run kernel k0(y,2017) once
for (i=startIteration; i<numIterations; i++) {
        if ((i%numSave)==0) pcl_saveState("state.dat");         // save everything except private arrays
        pcl_runBatch(b);                                        // run k1(x,y,5), k1(x,y,7), more (if any)
        if ((i%numShow)==0) pcl_saveArray1D(x);                 // dump to file for checking purposes
}
```

Main ideas of PCL

- Consider each array as a super-'variable'
- Call each kernel with a single (simple) line
- Link each kernel with a set of its own arrays: inputs and outputs
- Organize kernels into a batch to be called iteratively (all these kernels with their arguments in predefined order)
- Several levels of abstraction: (i) arrays/kernels, (ii) batch (one step of calculation), (iii) iterations (main flow of calculation)

Main blocks of PCL library

1. Config file
2. Arrays: saved/loaded (public) or created each time (private)
3. Kernels: from source or binary
4. Batch of kernels
5. Save/load state with all arrays contents, kernels and batches. Load and continue
6. Iterations: e.g. 1000000 batch invocations, each 1000 save state, each 100 save main result
7. Save intermediate data files: text array or *.bmp – for debug and control

Comparison and statistics

Standard approach. Solution of Ginzburg-Landau equations for superconducting film

- Number of lines, C: 4346
- Number of files, C: 22
- Number of lines, OpenCL: 1333
- Number of kernels: 17
- Time of development: more than 4 weeks
- Error-prone due to repeating patterns of code
- Hard to implement new features

Production-CL library

- Number of lines, C: 1693

PCL approach. Solution of kinetic equations for excitons in semiconductor

- Number of lines, C: 2132 (excluding PCL)
- Number of files, C: 2 (excluding PCL)
- Number of lines, OpenCL: 1717
- Number of kernels: 18
- Main batch (Runge-Kutta 4th order) consists of 36 kernel calls
- Time of successful development: less than 1 week
- Easy to implement new features, add new terms in equation, etc.

PCL approach. Solution of Gross-Pitaevskii equation for Bose-Einstein condensate in two cavities

- Number of lines, C: 1640 (excluding PCL)
- Number of files, C: 2 (excluding PCL)
- Number of lines, OpenCL: 553
- Number of kernels: 10
- Time of successful development: about 4 days
- Easy to implement new features

Next steps

- Refine and piblish on GitHub
- multi-GPU
- swap to CPU RAM for large problems

References

[1] Richardson W B, Pardhanani A L, Carey G F, and Ardelea A. Int. Journal for Numerical Methods Engineering **59** 1251 (2004).
[2] P.F. Kartsev, I.O. Kuznetsov, J. Phys: Conf. Ser. **737** 012033 (2016)
[3] N.S. Voronova and Yu.E. Lozovik. Phys. Rev. B **86** 195305 (2012)