Potential student projects

Mike Giles

January 10, 2010

Abstract

I think the overall OP2 project has the potential to provide a number of mini-projects which are suitable for undergraduate and MSc projects.

1 Projects

1.1 FORTRAN port

The existing implementation is for programs written in C/C++. The port to FORTRAN should be straightforward in principle, using the PGI FORTRAN CUDA compiler. This might make a good third-year undergraduate project.

1.2 OpenCL and SSE/AVX port

The existing implementation is in CUDA for NVIDIA GPUs. The port to OpenCL will hopefully be fairly straightforward, but the port to SSE/AVX vectors will probably require more experimentation and a good understanding of the Intel hardware and software; it's not clear whether this is best done using Ct or Intel's special SSE/AVX vector datatypes such as `F32vec4`.

1.3 Run-time optimisation of block sizes

Each execution plan for parallel loops with indirection needs to decide on a block size, the number of set elements to be handled by each thread block.

It is not very clear *a priori* what the right size should be. It mustn't be too large or else the shared memory requirements will exceed the amount available. However, it is not clear that the best choice is to maximise the size subject to this constraint. By using a smaller size it will be possible for the run-time system to execute more than one thread block on each multiprocessor, and this can have significant benefits in overlapping the read/write performed at the beginning and end of each thread block, with the execution performed in the middle.

The idea for this project is to use run-time optimisation. A number of plans are constructed for each parallel loop, and the run-time execution tries them all, keeps track of the execution time for each, and optimises over time. The optimisation will have to cope with the fact that there is probably a fair amoint of "noise" in the timing measurements.

I think this project fits in well with an important general theme in scientific computing. Auto-tuning is becoming increasingly common (e.g. FFTW, ATLAS) though usually I think this is done as a one-off optimisation (e.g. when software is first installed) rather than being done at run-time.

1.4 Graph-based partitioning and renumbering

In an OP application, the user specifies a number of sets and associated pointers between the sets. These can be used to create a graph for each set by connecting elements which both point to the same element of a different set, or which are pointed to by the same element of a different set.

Graph-based partitioning can then be used to do the high-level partitioning required for the MPI distributed-memory parallelisation, and the low-level recursive partitioning to renumber sets to maximise data reuse within the thread blocks.

This is a more substantial project, and rather open-ended.

1.5 Improved debugging tools

The current single-threaded CPU implementation performs a number of run-time checks on the user code, but there is a lot more that could be done to check the correctness of the code.

For example, the user declares the way in which each of the arguments to the user's kernels is used, and also declares that the order in which the set elements is processed does not affect the final outcome (except for the limitations of finite precision arithmetic). All of this could be subjected to run-time checks.

1.6 New applications

New applications using the OP library would be welcome.