libxc on GPUs

Introduction

The work below describes a process that allows libxc to run on GPUs, using the natural parallelization over gridpoints. It was done with libxc 1.2.0, but we later also did it (with some relatively minor modifications) for libxc 2.0.0. We obtained speedups of 30x-100x (cpu+gpu vs. cpu).

We did this as an introductory project to learn how to port software to GPUs. The documentation below is "crude" because there is currently not much demand for libxc on GPUs. But perhaps in the future it may be useful for people.

Building libxc 1.2.0 adding additional GPU-compatible functionals

- install CUDA, if necessary (have only tested with CUDA4 and libxc 1.2.0)
- get a fresh libxc-1.2.0 (e.g. tar xfz /nfs/slac/g/suncatfs/sw/package/libxc-1.2.0.tar.gz)
- cd libxc-1.2.0
- ./configure --prefix=`pwd`/install LIBS="-L/opt/CUDA/CUDA41/cuda/lib64 -lcudart" CFLAGS="-O2 -fPIC"
- get libxc-gpu:
    - for SLAC users: svn co svn+ssh://username@suncatls1.slac.stanford.edu/libxc-gpu/tags/1.0 libxc-gpu
    - for offsite users (readonly): svn co svn://suncatls1.slac.stanford.edu/libxc-gpu/tags/1.0 libxc-gpu
- cd libxc-gpu
- modify top two directory names (CUDADIR, XCDIR) in Makefile
- make setup
- cd libxc-1.2.0/src/
- make xc_funcs.h libxc.la
- cd libxc-gpu
- make
- (optional) run xc_gpu_timing executable to compare CPU/GPU timing for several functionals
- cd libxc-1.2.0
- make install

Running a GPAW Calculation with GPU libxc Functionals

- in GPAW there are two ways one can specify a libxc functional, the method shown here with the "+" symbol, or by adding a simpler "short_name" to the top of gpaw/xc/libxc.py.
- generate the setup for the GPU functional you are interested in (search for _GPU in xc_funcs.h in libxc for list of currently ported functionals). For example:

```
gpaw-setup -f GGA_X_RPBE_GPU+GGA_C_PBE_GPU H
```

- add path to the above generated setup to GPAW_SETUP_PATH environment variable
- run the script specifying the GPU functional. For example:

```
#!/usr/bin/env python

from ase import *
from gpaw import GPAW

a = 5.0
H = Atoms([Atom('H',(a/2, a/2, a/2), magmom=1)],
                pbc=False,
                cell=(a, a, a))

H.set_calculator(GPAW(nbands=1, h=0.2, convergence={'eigenstates': 1e-3},txt='H.txt',
                 xc='GGA_X_RPBE_GPU+GGA_C_PBE_GPU'))
e = H.get_potential_energy()
```

For Developers: Porting libxc Functional to GPU

This requires that the functional use the common "work.c" mechanism. Some functionals (e.g. tpss_c seem to not do this yet).

- if one doesn't exist, create an "init" function
- add copy_gga (or copy_lda/copy_mgga as appropriate) to the init function
- declare "static XC(mgga_type) **p_d" and "xc_mgga_type** copy_mgga(xc_mggatype* h, size_t sizeofparams)" at the top of the .cu file
- change "static FLOAT" to "static const *_device_* FLOAT"
- change routines for equation routines from "static void" to "static inline *_device_* void"
- add "extern "C" const" in front of XC(func_info_type) declaration
- add "_gpu" to the name of the XC(func_info_type) declaration
- change the functional number (first parameter in XC(func_info_type)). We typically add _GPU to the end
- change the #define for the functional number at the top of the file. We typically add 1000
- save the functional as a .cu file

- changed "#include "work_mgga_x.c"" to "#include "work_mgga_x.cuh""
- some functionals will call external functions on the GPU like XC(rho2dzeta). May need to cobble together declaration header files for these.
- if a functional has a "params" struct, you must add an "int" first member of that struct, which must be the sizeof(struct). this is used to copy the params struct to the GPU.

For Developers: Porting General "Work" routine to GPU

This need to be done once for LDA/GGA/MGGA x/c for the common "work.c" file (total of 6 of these)

- copy "static void work_mgga_x" declaration (including arguments) to the bottom. this will become cpu wrapper for the gpu kernel.
- in this new "wrapper" routine
    - change "static void" declaration to "extern "C" static void" declaration
    - define block_size and n_blocks (based on number of points)
    - call the gpuwork kernel
    - pass p_d as the first argument
- in the "kernel" routine
    - change kernel "static void work_mgga_x" declaration to "static void *__global__* gpuwork"
    - delete "ip" loop over number of points
    - add "int idx = blockIdx.x*blockDim.x + threadIdx.x"
    - add "if (idx<np)"
    - pointers are normally incremented at the bottom. move all the stuff after "end_ip_loop:" immediately after the above "if" statement
    - add "*idx" at the end of each pointer assignment (pointer arithmetic)
- save work as new copy with .cuh extension
- make sure to set block size large enough so that number of blocks does not exceed 65536 (GPU limit)

libxc on GPUs: Issues

- we really need to move from CUDA4 to CUDA5 so we have a linker. right now all the source for a functional has to show up in one file, so we have to use ugly include files.
- One of the tricky things was to copy the structs xc_lda_type,xc_gga_type,xc_mgga_type from CPU to GPU. A picture of all the structs involved is here. We currently leak the memory for this struct (never deleted)
- mgga_c in libxc 1.2.0 doesn't follow the right pattern in it's "work".c file to be ported to the GPU. we believe this is fixed in 2.0.0.
- we use nvcc for all mixed host/gpu code
- nvcc does C++ mangling. need extern "C" in some cases.
- not sure if we can make kinetic functionals and hybrid functionals (like PBE0) work
- using libxc gets tricky with spin-polarized calculations if spin-indices are not dense in memory (true on CPUs as well)
- libxc is currently unable to add separate X/C contributions "in place" without using extra arrays (true on CPUs as well)

CUDA Problems

- Unique symbol problem, including the same .cuh in multiple files can't set breakpoint. Need cuda5 for linkers.
- Slow startup, 30 sec wait, fixed with "nvidia-smi -pm 1"
- sometimes, we can't step into a function (if it is a file included in a ".cuh"? (multi-layer include))

```
(cuda-gdb)
200        /tmp/tmpxft_00001dc8_00000000-7_gga_c_pbe.cpp3.i: No such file or directory.
       in /tmp/tmpxft_00001dc8_00000000-7_gga_c_pbe.cpp3.i



(cuda-gdb)
func (xs=0xddddddddddddddddd, p=warning: Variable is not live at this point. Value is undetermined.
0x0, order=warning: Variable is not live at this point. Value is undetermined.
0, rs=warning: Variable is not live at this point. Value is undetermined.
0, zeta=-1.4568159901474629e+144, xt=warning: Variable is not live at this point. Value is undetermined.
0, f=warning: Variable is not live at this point. Value is undetermined.
0x0, dfdrs=0xddddddddddddddddd, dfdz=0xddddddddddddddddd, dfdxt=0xddddddddddddddddd, dfdxs=warning: Variable is
not live at this point. Value is undetermined.
0x0, d2fdrs2=0xddddddddddddddddd, d2fdrsz=0xddddddddddddddddd, d2fdrsxt=0xddddddddddddddddd,
d2fdrsxs=0xddddddddddddddddd, d2fdz2=0xddddddddddddddddd, d2fdzxt=0xddddddddddddddddd, d2fdzxs=0xddddddddddddddddd,
d2fdxt2=0xddddddddddddddddd, d2fdxtxs=0xddddddddddddddddd, d2fdxs2=0xddddddddd) at gga_c_pbe.cu:271
271        }
```