Lattice Simulations using OpenACC compilers

Pushan Majumdar

(Indian Association for the Cultivation of Science, Kolkata)
OpenACC is a programming standard for parallel computing developed by Cray, CAPS, Nvidia and PGI. The standard is designed to simplify parallel programming of heterogeneous CPU/GPU systems.

The OpenACC Application Program Interface describes a collection of compiler directives to specify loops and regions of code in standard C, C++ and Fortran to be offloaded from a host CPU to an attached accelerator, providing portability across operating systems, host CPUs and accelerators.

The directives and programming model defined in this document allow programmers to create high-level host+accelerator programs without the need to explicitly initialize the accelerator, manage data or program transfers between the host and accelerator, or initiate accelerator startup and shutdown.
Programs I have had some experience with:

1. **Staggered fermions with wilson gauge action on**
   (a) single GPU – in some detail
   (b) multi GPU – preliminary

2. **Wilson fermions with Wilson gauge action on**
   single GPU – preliminary

Main bottlenecks is slow data movement between CPU & GPU. Speed is about 5 GB/s.

Impossible to avoid CPU completely as I/O, if-then clause is evaluated on CPU.
BLAS functions are launched from CPU and MPI calls (at least for Fermi GPUs) are launched from CPUs.
subroutine congrad(nitcg)

... All kinds of definitions and declarations ...

!$ACC data copy(nitcg,alpha,betad,betan)
!$ACC+ copyin(nx,iup,idn,u,r)
!$ACC+ copyout(x,y)
!$ACC+ create(ud,ap,atap,p)

* 
  call linkc_acc

!!$OMP parallel do default(shared)
!$ACC parallel loop collapse(2) reduction(+:betan) present(p,r,x)
  do l = 1, mvd2
    do ic=1,nc
      p(l,ic) = r(l,ic) ; x(l,ic) = (0.,0.)
      betan=betan+conjg(r(l,ic))*r(l,ic)
    end do
  end do
end do
betan=real(zdotc(mv3d2,r,1,r,1))

$ACC update host(betan)

if (betan.lt.delit) go to 30

$ACC parallel present(beta,betan,betad,alphan)

beta=betan/betad ; betad=betan ; alphan=betan

$ACC end parallel

do nx = 1, nitrc *Main loop of conjugate gradient begins*

nitcg ← nitcg+1 ;     ap = 0

call fmv(0,mvd2,ap,p) *Matrix-vector multiplication*

alphad=⟨ap,ap⟩ + ⟨p,p⟩ ; alpha=alphan/alphad

atap ← p ;     x ← x + alpha * p
call fmtv(atap,ap) \rightarrow (Matrix-vector multiplication)

\text{r} \leftarrow \text{r} - \alpha \cdot \text{atap} \\
\text{betan}=\langle \text{r},\text{r} \rangle \\
!$\text{ACC}$ \text{update host(betan)} \quad \text{Exit condition evaluated on CPU}

\text{if} \ (\text{betan} \ .lt. \ \text{delit}) \ \text{go to 30} \\
\text{beta}=\text{betan}/\text{betad} \ ; \ \text{betad}=\text{betan} \ ; \ \text{alphan}=\text{betan} \\
\text{p} \leftarrow \text{r} + \beta \cdot \text{p} \\
\text{end do} \quad \text{Main loop of conjugate gradient ends}

30 \text{ continue} \\
* \\
\text{y} = 0 \quad \text{Solution on the second half lattice} \\
\text{call fmv(mvd2,mv,y,x)} \rightarrow (Matrix-vector multiplication)

* \\
!$\text{ACC}$ \text{end data} \\
\text{return}
Matrix-vector multiplication routine

subroutine fmv(noff,nsz,v,w)
... All kinds of definitions and declarations ...
!!$OMP parallel do default(shared)
!!$OMP+ private(nnu,px1,px2,px3,px4,px5,px6)
!!$OMP^ private(v1,v2,v3)
!$ACC parallel loop present(u,ud,v,w,iup,idd)
!$ACC+ private(nnu,px1,px2,px3,px4,px5,px6,v1,v2,v3)
!$ACC+ vector_length(32)
  do  l = noff+1, noff+mvd2
    :
    Routine identical to CPU version
    :
    enddo
return
32^3 x 8 lattice
Staggered fermions with even–odd decomposition

Cray XE6 with cray compiler
Cluster with intel fortran compiler
X2090 GPU with Cray OpenACC compiler
Multi-GPU code

:  
ap_loc ← 0

!$ACC parallel loop present(u, ud, ap_loc, p, iup, idn)
do l = base+1, base+nvd2
  v1 = ap_loc(1, l-base)
:
  \textit{Lines identical to scalar version}
  :
  ap_loc(3, l-base) = v3
enddo

!$ACC update host(ap_loc)
call MPI_ALLGATHER(ap_loc, 3*nvd2, MPI_DOUBLE_COMPLEX,
+        ap, 3*nvd2, MPI_DOUBLE_COMPLEX, MPI_COMM_WORLD, ierr)

!$ACC update device (ap)

Worry about async compiler options.
Summary

- Coding effort is only marginally higher than OpenMP. Almost each OpenMP directive can be replaced with a OpenACC directive. Only additional directive is the creation of a data region with a list of variables (scalars + arrays) so that the compiler knows which variables to copy to the GPU and back again.  
  One data structure I haven’t explored is deviceptr.

- Performance of single GPU staggered fermion code is roughly equivalent to 128 cores of cluster with QDR infiniband interconnect.

- Performance of single GPU Wilson fermion code is roughly equivalent to 96 cores. About 30% difference in performance between hand coded CUDA and OpenACC code.
• GPU with 6GB memory fits in a $32^4$ Wilson fermion lattice
  or a $10 \times 40^3$ staggered fermion lattice.

• Real gain comes only when the whole conjugate gradient
  routine is on the GPU.

• Extremely useful if one does not have access to conventional
  supercomputers.

• For Multi-GPU programs MPI calls on the Fermi GPUs can
  only be made from the CPU so every MPI call involves a data
  transfer from the GPU to CPU and back. Each such copy
  adds a significant ($\sim 15\%$) overhead to the runtime. For
Kepler GPUs the construct host_data use_device cuts this down to a certain extent.

- Further performance gains can be obtained by using mixed-precision routines and improved storage schemes.

Acknowledgments

ILGTI-TIFR for funding the GPU portion of the Cray on which these studies were carried out. IACS for funding the rest of the Cray without which the GPU portion wouldn’t run.

Cray India team for help at various stages during the development of the OpenACC codes.