Accelerating NEMO with OpenACC

Maxim Milakov, Peter Messmer, Thomas Bradley
NVIDIA
NEMO

- Modeling framework for oceanographic research, operational oceanography, seasonal forecast and climate studies

- 240 projects in 27 countries

- Fortran, 90,000 lines of code
3 ways to Accelerate Applications

Applications

Libraries

Directives

Programming languages

“Drop-In” Acceleration

Easily Accelerate Applications

Maximum Flexibility
Flat profile
Directives

- Directives is a good solution for flat profile codes

- Memory transfers between host and device might limit the performance when individual subroutines are accelerated

- We need to keep data on the accelerator between subroutines

- OpenACC enables it
OpenACC Directives

- Simple compiler hints, similar to OpenMP
- Compiler Parallelizes the code
- Portable code

Program myscience...

... serial code ...

\$\text{acc kernels}

do k = 1, n1

do i = 1, n2

... parallel code ...

endo
do

...\$\text{acc end kernels}

...

End Program myscience
Kernels directive

```
!$acc kernels

! 3. monotonic flux in the i & j direction (paa & pbb)
!  -------------------------------

DO jk = 1, jpkml
    DO jj = 2, jpjml
        DO ji = fs 2, fs_jpiml ! vector opt.
            zau = MIN( 1.0e0, zbetdo(ji,jj,jk), zbetup(ji+1,jj,jk) )
            zbu = MIN( 1.0e0, zbetup(ji,jj,jk), zbetdo(ji+1,jj,jk) )
            zcu = ( 0.5 + SIGN( 0.5 , paa(ji,jj,jk) ) )
            paa(ji,jj,jk) = paa(ji,jj,jk) * ( zcu * zau + ( 1.0e0 - zcu) * zbu )
            zav = MIN( 1.0e0, zbetdo(ji,jj,jk), zbetup(ji,jj+1,jk) )
            zbv = MIN( 1.0e0, zbetup(ji,jj,jk), zbetdo(ji,jj+1,jk) )
            zcv = ( 0.5 + SIGN( 0.5 , pbb(ji,jj,jk) ) )
            pbb(ji,jj,jk) = pbb(ji,jj,jk) * ( zcv * zav + ( 1.0e0 - zcv) * zbv )
        ! monotonic flux in the k direction, i.e. pcc
        !  -------------------------------
            za = MIN( 1.0e0, zbetdo(ji,jj,jk), zbetup(ji,jj,jk) )
            zb = MIN( 1.0e0, zbetup(ji,jj,jk), zbetdo(ji,jj+1,jk) )
            zc = ( 0.5 + SIGN( 0.5 , pcc(ji,jj,jk+1) ) )
            pcc(ji,jj,jk+1) = pcc(ji,jj,jk+1) * ( zc * za + ( 1.0e0 - zc) * zb )
    END DO
  END DO
END DO

!$acc end kernels

CALL lbc_lnk( paa, 'U', -1.e0, pbb, 'V', -1.e0 ) ! lateral boundary condition (changed sign)
```
Data directive

```fortran
!$acc data pcopy(umask, vmask, tmask, tmask i, tsn, un, vn, wn, avt, avmu) &
!$acc pcopy(eu, elv, elt, elf, e2u, e2v, e2t, e2f, e3u, e3v, e3t, e3f, e3w, e3vw) &
!$acc pcopy(fmask, hdinv, hdinv b, rotb, rotn, ua, va, ub, vb, ff, mbkt, rhop, hmlt) &
!$acc pcopy(spgu, spgy, gcdbrc, gcra, gcx, gcxb, sshn, bmask, gcden) &
!$acc pcopy(almv, bfrua, bfrva, mbku, mbkv, utau, utau b, utau b, eta) &
!$acc pcopy(tsa, tsb, rhd, gdept, gdegw, mre, gru, grv, hmlp, hmlpt) &
!$acc pcopy(nmin, omlmask, uslpml, wslpml, wslpml, uslp, vslp, wslp, wslp) &
!$acc pcopy(ems, emsp, qns, qsr, gphi, gphi, gphi, gphi, gphi, gphi, gphi, gphi, gphi, gphi, gphi) &
!$acc pcopy(qns b, emps b, fr i, rnf, hu, hv, gcp, gccd, gced, gcr) &
!$acc pcopy(ssu m, ssv m, ssd m, ssh m, gtsu, gtsv, ahtu, ahtv, ahtv) &
!$acc pcopy(ssh, sshb, etot3, qsr hc b, qsr hc, ah wslp2, bfrcoef2d) &
!$acc pcopy(r2dtra, rdtrdta, dissl, htau, en, avmb, avtb, avtb 2d, avm) &
!$acc pcopy(bh tsc, sbc_tsc b, rnf tsc, rnf tsc b, nk rnf, h rnf, rnfmsk)

#if defined key_mpp_mpi
!$acc data pcreate(t3ew, t3we, t3ns, t3sn, t2ew, t2we, t2ns, t2sn, pt2d_lbcnfd, tr2ew, tr2we, tr2ns, tr2sn)
#endif

DO WHILE ( istp <= nitend .AND. nstop == 0 )
  #if defined key_agrif
    CALL Agrif_Step( stp ) ! AGRIF: time stepping
  #else
    CALL stp( istp ) ! standard time stepping
  #endif
  istp = istp + 1
  IF( lk_mpp ) CALL mpp_max( nstop )
END DO

#if defined key_mpp_mpi
!$acc end data
#endif

!$acc end data
```
Running on GPU - timeline in Vampir
MPI communication

Sending data with arrays on CPU:
- Pack
- Send

Arrays on GPU:
- Pack
- D2H
- Send

Receiving data:
- Receive
- H2D
- Unpack

!$acc update host

!$acc update device
MPI communication - timeline

Pack
MPI communication - timeline

Pack - D2H
MPI communication - timeline

Pack - D2H - MPI Send
MPI communication - timeline

Pack - D2H - MPI Send, MPI Receive
MPI communication - timeline

Pack - D2H - MPI Send, MPI Receive - H2D
MPI communication - timeline

Pack - D2H - MPI Send, MPI Receive - H2D - Unpack
Reducing MPI communication overhead

- According to initial tests MPI sync took significant amount of time

- 2 improvements:
  - Batch sequential independent array synchronization into a single MPI send/receive
  - Enable extra halo for the solver to reduce MPI sync

- Beneficial both for GPU and CPU
SOR solver - timeline
Status of the code

- Output is switched off
  - NEMO 3.5 will have IO functionality redesigned
  - We postponed adapting IO to GPU-located arrays
  - Correctness of the code is tracked by checking statistics of the solver at each time step

- GYRE only

- There is a small, acceptable difference in the results
  - IEEE 754-2008 doesn’t dictate correct rounding for SIN/COS/EXP
  - SIN/COS/EXP are implemented differently in different architectures
Build configuration

- HDF5 1.8.9, NETCDF 4.1.3, NEMO 3.4, double precision

- Intel Fortran compiler 13.0, Intel MPI 4.1, -fast -O3

- PGI Fortran compiler 12.10, MVAPICH2 1.9a2, -fast -O3 -Mipa=fast,inline -Mcray=pointer

- The same as above plus -acc -ta=nvidia -Minfo=all -Mneginfo=all
Benchmarking - hardware

- “Sandy Bridge + Kepler” nodes, each having:
  - CPU: 2 sockets * Xeon E5-2670 (Sandybridge), 2.6GHz (3.3GHz Turbo Boost), 8 cores, 64 GB RAM
  - GPU: 2x Tesla K20X, ECC off, 6GB RAM each
  - 4x FDR Infiniband (56 Gb/s)

- Running configuration is GYRE_50 (1/4 degree), requires about 23GB of total RAM, fits 4 K20X

- The code is running on 2 nodes

- The performance is measured by running 1000 time steps, startup and shutdown overheads are not included in figures
Benchmarking - results

GPU vs. CPU - 3.1x speedup

Time steps per second

- Intel, CPU
- PGI, CPU
- GPU
Vampir - analysis

- 84% CUDA Kernels
- 5% GPU<->Host memory transfer
- 6% Other GPU overhead
- 2% CPU code
- 1% MPI synchronization
- 1% Profiling overhead
CUDA-aware MPI will enable us passing device pointers to MPI calls thus leveraging GPU Direct.

- GPU Direct RDMA will enable us to get rid of D2H/H2D steps completely.
Next steps

- Implement writing output files functionality
- Port ORCA2-LIM configuration to GPU
- Build NEMO with other OpenACC compilers (CAPS and Cray)
- Start using CUDA-aware MPI implementations
References

- NEMO: http://www.nemo-ocean.eu
- OpenACC: http://www.openacc-standard.org
- PGI Accelerator: http://www.pgroup.com/resources/accel.htm
- CAPS OpenACC compiler: http://www.caps-enterprise.com
- Cray Compiler Environment: http://www.cray.com/Products/XK6/Software.aspx
- VampirTrace: http://goo.gl/aCVMy
- Vampir: http://www.vampir.eu
- MVAPICH2: http://mvapich.cse.ohio-state.edu
Thank you!

Q & A