Porting a Fortran Oceanographic code to GPUs; the gNEMO Project

Andrew Porter, Stephen Pickles & Mike Ashworth
Computational Science & Engineering Dept.
STFC Daresbury Laboratory
Contents

• Oceanography – NEMO
• Accelerator Directives
• Moving a Routine to a GPU
• Performance Results
• Conclusions and Future Prospects
• Acknowledgements
Contents

• Oceanography – NEMO
• Accelerator Directives
• Moving a Routine to a GPU
• Performance Results
• Conclusions and Future Prospects
• Acknowledgements
NEMO

- Widely-used European ocean model
- Fortran90 and MPI
- Highly portable
- Memory-bandwidth bound
- ~20 years of development

http://www.nemo-ocean.eu/
Contents

• Oceanography – NEMO
• Accelerator Directives
• Moving a Routine to a GPU
• Performance Results
• Conclusions and Future Prospects
• Acknowledgements
Accelerator Directives - Motivation

• CUDA & OpenCL are C based
• NEMO core is \( \sim 100K \) lines of Fortran90
• Performance
  – GPUs have \( \sim 10x \) peak memory bandwidth of a CPU
  – Maintain single code base but add option to use GPU if available
• Portability
  – not every computer has a GPU attached
## Accelerator ‘Directives’ - Options

<table>
<thead>
<tr>
<th>Approach</th>
<th>Notes</th>
<th>Fortran support</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGI Accelerator Directives</td>
<td>Currently NVIDIA specific, basis for OpenAcc</td>
<td>Yes</td>
</tr>
<tr>
<td>HMPP Workbench</td>
<td>Can generate CUDA and OpenCL code, will support Intel MIC in 2012.</td>
<td>Yes</td>
</tr>
<tr>
<td>PGI CUDA Fortran</td>
<td>NVIDIA specific</td>
<td>Yes</td>
</tr>
<tr>
<td>OpenCL</td>
<td>Portable, open standard</td>
<td>No</td>
</tr>
<tr>
<td>CUDA C</td>
<td>Widely used, mature and low-level but NVIDIA specific</td>
<td>No</td>
</tr>
</tbody>
</table>
• Oceanography – NEMO
• Accelerator Directives
• Moving a Routine to a GPU
• Performance Results
• Conclusions and Future Prospects
• Acknowledgements
Porting a Routine I

• Mark-up region to accelerate
  – (Move region into a separate ‘codelet’)
• (In-line any routine calls in region)
• Make all loops explicit
  – no $array(:, :, jk)$ notation permitted
• Mark-up the loops to parallelize
• Permute loops for memory coalescing
  – Want consecutive threads to work on consecutive memory addresses
DO jn = 1, kjpt

    zdit(1,:,:,:)=0.e0_wp

DO jk = 1, jpkm1
    DO jj = 1, jpjml
        DO ji = 1, jpiml
            zdit(ji,jj,jk) = (ptb(ji+1,jj,jk,jn) - ptb(ji,jj,jk,jn)) * umask(ji,jj,jk)
        END DO
    END DO
END DO
END DO
END DO
END DO
DO jn = 1, kjpt

zdit(1,:,:,:) = 0.e0_wp

DO jk = 1, jpkml 
    DO jj = 1, jpjml 
        DO ji = 1, jpiml 
            zdit(ji,jj,jk) = (ptb(ji+1,jj,jk,jn) - ptb(ji,jj,jk,jn)) * umask(ji,jj,jk)
        END DO 
    END DO 
END DO 
END DO 
END DO 
END DO
DO jk = 1, jpkml
  DO jj = 1, jpjml
    zdit(1,jj,jk)=0.e0
  END DO
END DO

END DO
DO  jn = 1, kjpt

zdit(1,:,:,:) = 0.e0_wp

DO  jk = 1, jpkml
  DO  jj = 1, jpjml
    DO  ji = 1, jpiml
      zdit(ji,jj,jk) = (ptb(ji+1,jj,jk,jn) - ptb(ji,jj,jk,jn)) * umask(ji,jj,jk)
    END DO
  END DO
END DO
END DO
!$hmppcg permute jj,ji,jk
DO jk = 1, jpkml
!$hmpp parallel
  DO jj = 1, jpjm1
    zdit(ji,jj,jk,1) = (ptb(ji+1,jj,jk,1) -
                       ptb(ji,jj,jk,1)) * umask(ji,jj,jk)
    zdit(ji,jj,jk,2) = (ptb(ji+1,jj,jk,2) -
                       ptb(ji,jj,jk,2)) * umask(ji,jj,jk)
  END DO
END DO
END DO
END DO
Porting a Routine II

• Analyse data transfers & work to reduce:
  – Keep constant arrays on the device
  – Asynchronous data transfer & kernel execution
  – For halo swaps, transfer halo regions only
  – Overlap transfers of halos to/from GPU with halo packing/unpacking on host
  – `#include` halo pack/unpack code as can’t call subroutines on GPU
END DO
END DO
END DO

CALL halo_swap( zwi(:,:,:,:,1))
CALL halo_swap( zwi(:,:,:,:,2))

DO jk = 1, jpk, 1
  DO jj = 1, jpj, 1
    DO ji = 1, jpi, 1

  END DO
END DO

Code Example II

 !$hmpp <traadv_tvd> kernel1 waitstore, args[zhaloswi]
 CALL unpack_halos(zhaloswi, zwi, 1)
 CALL unpack_halos(zhaloswi, zwi, 2)
 CALL halo_swap( zwi(:, :, :, 1))
 CALL halo_swap( zwi(:, :, :, 2))
 CALL pack_halos(zhaloswi, zwi, 1)
 CALL pack_halos(zhaloswi, zwi, 2)

 !$hmpp <traadv_tvd> kernel3 advancedload, args[zhaloswi], asynchronous
Porting a Routine III

The diagram shows the number of lines of code for various routines:
- traldf_iso
- traadv_tvd
- trazdf_imp
- ldf_slp

The routines contain halo swaps.

Legend:
- Blue: Original
- Orange: Directives
Example – Tracer Advection

- Originally ~400 lines; GPU version ~1000 lines!
- One child routine (80 lines)
  - Contains one halo swap => splits routine into two codelets
  - Called twice => in-lined twice
- Six separate codelets
  - Six lots of routine interface descriptions
- 16 halo swaps, all for 3D arrays
Contents

• Oceanography – NEMO
• Accelerator Directives
• Moving a Routine to a GPU
• Performance Results
• Conclusions and Future Prospects
• Acknowledgements
Results - Hardware

- ‘cseht’ & ‘SiD’ machines at Daresbury
- Quad-core Intel Nehalem processor
  - E5540 @ 2.53GHz
- NVIDIA S1070 server
  - contains four M1060 cards
  - ‘Tesla’
- NVIDIA M2050
  - ‘Fermi’
Optimising data transport
- transfer halo regions only

- Nehalem
- Tesla (whole arrays)
- Tesla sm_20, Fermi
- sm_20, Fermi
- sm_13, Fermi

Tracer advection kernel with ORCA1 grid

<table>
<thead>
<tr>
<th></th>
<th>Compute</th>
<th>Transport</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nehalem</td>
<td>0.7</td>
<td>0.0</td>
</tr>
<tr>
<td>Tesla (whole arrays)</td>
<td>0.4</td>
<td>0.3</td>
</tr>
<tr>
<td>Tesla</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>sm_20, Fermi</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>sm_13, Fermi</td>
<td>0.1</td>
<td>0.1</td>
</tr>
</tbody>
</table>
Comparison with OMP (ORCA2, traldf_iso compute only)

Sub-optimal scaling due to problem size and memory bandwidth
Integration with NEMO

![Graph showing integration with NEMO](image)
Contents

• Oceanography – NEMO
• Accelerator Directives
• Moving a Routine to a GPU
• Performance Results
• Conclusions and Future Prospects
• Acknowledgements
Conclusions

• Successfully ported four routines to GPU using HMPP Workbench
• No speed-up for the sea-ice routine
• Basic porting is fairly straightforward
  – Have to in-line subroutines
  – MPI calls must be on host
  – Can also end up restructuring for performance
• Must work hard to reduce data transfers
• Fragile code
Future Prospects I

• hmpp currently the most mature directives option
  – Also supports asynchronous data transfers
  – Soon to support Intel MIC
• OpenACC announced at SC11
  – Based on PGI’s model
  – PGI, nvidia, Cray & CAPS
  – Initial spec. quite basic
Future Prospects II

- GPUDirect & multi-GPU codes
  - Share pinned memory with Infiniband interconnect
  - DMA between GPUs
  - Avoids doing a copy in system memory for MPI calls

- Move the GPU onto the motherboard
  - Nvidia’s Denver, AMD’s Fusion, Intel’s MIC
  - A single memory address space?
Acknowledgments

• NERC for funding gNEMO
• DiSCO at Daresbury for systems
• Igor Kozin, Xiaohu Guo & Stephen Pickles for advice/discussions
• PGI technical support & forum
• CAPS technical support
Extras...
Scaling of OMP version of lim_rhg