Weather & Ocean Code
Restructuring for OpenACC

Carl Ponder, Ph.D.
HPC Applications Performance
NVIDIA Developer Technology
Some interleaving of loops with complex sequential flow:
- Compound conditionals
- Procedure calls
- Recursion
Typical Application Structure

Some interleaving of loops with complex sequential flow:
- Compound conditionals
- Procedure calls
- Recursion

Parallel efficiency depends on dimensions and contents of loops

Code restructuring can improve performance
Typical Application Structure

MPI (process) parallelism suitable at top level of program
- Significant work per task amortizes process startup overhead
- Address spaces are separate.
- Need to minimize communication
- Heterogeneous control-flow ok
- Supported by modern cluster designs
Typical Application Structure

OpenMP (thread) parallelism suitable at most levels of program

- Lightweight threads have little overhead
- Need to balance thread counts with work per thread.
- One memory space visible at all times
- Heterogeneous control-flow ok
- Supported by modern multi-core processors
Multi-core SMP versus GPU

- Different numbers of processing elements (tens versus thousands)
- Memory capacity & bandwidth
OpenMP & OpenACC

Compiler directives as pragma’s or comments
  !$OMP parallel do
  !$acc kernels
Preserves original code
Can be compiled to ignore directives, for testing
Include the following capabilities
  • Managing degree of parallelism
  • Data placement for efficiency
  • Variable scoping
  • Special parallel operations (reductions)

Peak efficiency may require program changes.
GPU Efficiencies

Huge compute capacity
- Thread synchrony
- Coalesced memory access
- Limited private data state per thread
- Separate GPU & CPU memory

Uniquely suited to matrix and grid computations
- DGEMM
- FFT
- Stencils

Works surprisingly well with other problems
- Pattern matching
- Sorting
Loops need large iteration counts.
Control paths need to be reasonably uniform.
GPU data footprint usually smaller than host memory size.
COSMO: Scaling with Mesh Dimensions

- High performance
- High throughput

Wall time/step (s)

Mesh dimensions
Code Restructuring Examples

Applications:
- WRF
- HYCOM
- GEOS-5

Code Changes:
- Loop Splitting
- Rectangularization
- Variable Re-Dimensioning
- Loop distribution
- Loop Interchange
- Loop Fusion
- Data Transposition
- Data Placement
- Arrays-of-Structs
Loop-splitting example (HYCOM)

```fortran
! Compiles as 1-d loop.
do j=1-margin,jj+margin
    dpmin(j)=999.
    ! This may be stored in a register for the duration.
do i=1-margin,ii+margin
    dpold(i,j,k)=dp(i,j,k,n)
    util3(i,j)=util3(i,j)+dp(i,j,k,n)
    dp(i,j,k,n)=dp(i,j,k,n)- ((uflux(i+1,j)-uflux(i,j))+(vflux(i,j+1)-vflux(i,j)))*delt1*scp2i(i,j)
enddo
    dpmin(j)=min(dpmin(j),dp(i,j,k,n))
enddo
```

Good coding for CPU, but limits parallelism

```fortran
! Compiles as 2-d loop.
do j=1-margin,jj+margin
    dpmin(j)=999.
    ! Compiles as 1-d loop.
do i=1-margin,ii+margin
    dpold(i,j,k)=dp(i,j,k,n)
    util3(i,j)=util3(i,j)+dp(i,j,k,n)
    dp(i,j,k,n)=dp(i,j,k,n)- ((uflux(i+1,j)-uflux(i,j))+(vflux(i,j+1)-vflux(i,j)))*delt1*scp2i(i,j)
enddo
do j=1-margin,jj+margin
    dpmin(j)=min(dpmin(j),dp(i,j,k,n))
enddo
```

This must be re-read from memory.
HYCOM Example

Ocean current and temperature model.

Geography is abstracted as a 2D grid.

Boundary arrays are used to mark water intervals.

Most squares are all water or all land.
Rectangularization example (HYCOM)

Dynamic boundaries on i-loop skip redundant work. But processing the boundaries is easier to parallelize:

(This is an artifact of the geographical construction)
The I & k iterations are all independent, so we can rearrange the loops:

\begin{verbatim}
doi=1,m
 .... 
end do

doi=1,m
dok=1,np
 .... 
end do
end do

doi=1,m
 .... 
end do
\end{verbatim}

Requires that variables be re-dimensioned for i.
Variable re-dimensioning (GEOS-5/sorad)

434 RUN_LOOP: do i=1,m
   ...
456   do k=1,np
       ...
461   dp(k) = pl_dev(i,k+1)-pl_dev(i,k)
       ...
486   end do
       ...
645   do k=1,np
       ...
650   taurs=ry_uv(ib)*dp(k)
       ...
728   end do
Generating `dp(k)` for all the i-iterations requires adding an extra dimension `dp(i,k)`. Bundling all the dp arrays in GPU memory reduces the per-thread private space and improves occupancy. CPU can’t cache intermediate values.
Loop Interchange (GEOS-5/sorad)

CPU-friendly loop

RUN_LOOP_2: do i=1,m
    do k=1,np
        dp(k,i) = pl_dev(i,k+1)-pl_dev(i,k)
        dp_pa(k,i) = dp(k,i) * 100. ! dp in pascals
    end do
end do RUN_LOOP_2

Consecutive inner-loop iterations process adjacent elements.
Good cache efficiency.
Loop Interchange (GEOS-5/sorad)

**CPU-friendly loop**

```fortran
RUN_LOOP_2: do i=1,m
   do k=1,np
      ....
      dp(k,i) = pl_dev(i,k+1)-pl_dev(i,k)
      dp_pa(k,i) = dp(k,i) * 100. ! dp in pascals
   ....
   end do
end do RUN_LOOP_2
```

Consecutive inner-loop iterations process adjacent elements.
Good cache efficiency.

**GPU-friendly loop**

```fortran
!$acc parallel
!$acc loop collapse(2) private ( pa, i, k )
RUN_LOOP_2: do k=1,np
   do i=1,m
      ....
      dp(k,i) = pl_dev(i,k+1)-pl_dev(i,k)
      dp_pa(k,i) = dp(k,i) * 100. ! dp in pascals
   ....
   end do
end do RUN_LOOP_2
!$acc end parallel
```

Independent outer-loop iterations processed simultaneously.
Consecutive inner-loop iterations perform new fetches.
Good coalescing and memory bandwidth utilization.
Loop fusion (WRF)

IF( (j >= j_start_f ) .and. (j <= j_end_f) ) THEN ! use full stencil
  DO k=kts,ktf
  DO i = i_start, i_end
    fqv( i, k, jp1 ) = vel*flux5( field(i,k,j-3), field(i,k,j-2), field(i,k,j-1), field(i,k,j ), field(i,k,j+1), field(i,k,j+2), rv(I,k,j) )
  ENDDO
  ENDDO
ELSE IF ( j == jds+1 ) THEN   ! 2nd order flux next to south boundary
  DO k=kts,ktf
  DO i = i_start, i_end
    fqv(i,k, jp1) = 0.5*rv(i,k,j)*(field(i,k,j)+field(i,k,j-1))
  ENDDO
  ENDDO
.....
ENDIF

DO i = i_start, i_end ! This is not more efficient since the loops are either/or.
  DO j = j_start, j_end+1
    IF( (j >= j_start_f ) .and. (j <= j_end_f) ) THEN ! use full stencil
      vel = rv(i,k,j)
      fqv( jp1 ) = vel*flux5( field(i,k,j-3), field(i,k,j-2), field(i,k,j-1), field(i,k,j ), field(i,k,j+1), field(i,k,j+2), vel )
    ELSE IF ( j == jds+1 ) THEN ! 2nd order flux next to south boundary
      fqv(i,k, jp1) = 0.5*rv(i,k,j)*(field(i,k,j)+field(i,k,j-1))
    .....  
  ENDDO
ENDDO
Loop interchange (WRF)

j_loop_y_flux_5 : DO j = j_start, j_end+1 ! Serial outer loop.
   DO i = i_start, i_end
      DO j = j_start, j_end+1
         IF( (j >= j_start_f) .and. (j <= j_end_f) ) THEN ! use full stencil
            vel = rv(i,k,j)
            fqy(jp1) = vel*flux5( field(i,k,j-3), field(i,k,j-2), field(i,k,j-1), field(i,k,j), field(i,k,j+1), field(i,k,j+2), vel )
         ELSE IF ( j == jds+1 ) THEN ! 2nd order flux next to south boundary
            fqy(i,k,jp1) = 0.5*rv(i,k,j)*(field(i,k,j)+field(i,k,j-1))
         ENDIF
      ENDDO
   ENDDO
ENDDO
ENDDO j_loop_y_flux_5

j_loop_y_flux_5 : DO k=kts,ktf ! 2-d loop invoked once. More work per iteration.
   DO i = i_start, i_end
      DO j = j_start, j_end+1! Inner loop is sequential.
         IF( (j >= j_start_f) .and. (j <= j_end_f) ) THEN ! use full stencil
            vel = rv(i,k,j)
            fqy(jp1) = vel*flux5( field(i,k,j-3), field(i,k,j-2), field(i,k,j-1), field(i,k,j), field(i,k,j+1), field(i,k,j+2), vel )
         ELSE IF ( j == jds+1 ) THEN ! 2nd order flux next to south boundary
            ....
         ENDDIF
      ENDDO
   ENDDO
ENDDO j_loop_flux_5
GPU Data Placement (WRF)

wrf_run
  -> integrate
  -> solve_interface
  -> solve_em (DATA PLACEMENT HERE)
    -> microphysics_driver -> mp_gt_driver
      -> first_rk_step_part1
      -> first_rk_step_part2
    -> rk_scalar_tend
  -> advect_scalar_pd
Private Array Scoping (GEOS-5)

!$acc parallel
!$acc loop collapse(2) &
!$acc private ( fdndir, fdndif, rrt, ttt, tdt, tst ) &
!$acc private ( tautob, ssatob, asytob, tautof, ssatof, asytof ) &
!$acc private ( tdaold, ttaold, rsaold, tdanew, ttanew, rsanew ) &
!$acc private ( taurs, tauoz, tauuv, tausto, ssatau, asysto ) &
!$acc private ( denm, yy, xx4, fclr, fsdir, fsdif, flxdn, fall ) &
!$acc private ( fupa, fupc, fupdif, dum )

RUN_LOOP_6: do i=1,m
  do ib=1,nband_uv
type, public :: reactive_transport_auxvar_type
  ! molality
  PetscReal, pointer :: pri_molal(:)   ! mol/kg water

  ! phase dependent totals
  PetscReal, pointer :: total(:, :)   ! mol solute/L water

  type(matrix_block_auxvar_type), pointer :: aqueous

  ! sorbed totals
  PetscReal, pointer :: total_sorb_eq(:) ! mol/m^3 bulk
  PetscReal, pointer :: total_sorb_eq(:, :) ! kg water/m^3 bulk

  ! aqueous species
  ! aqueous complexes
  PetscReal, pointer :: sec_molal(:)
      .....

end type reactive_transport_auxvar_type
      .....

  type(reactive_transport_auxvar_type), pointer :: aux_vars(:)
  type(reactive_transport_auxvar_type), pointer :: aux_vars_bc(:)
  type(reactive_transport_auxvar_type), pointer :: aux_vars_ss(:)
  type(sec_transport_type), pointer :: sec_transport_vars(:)
Arrays-of-Structs (PFLOTRAN)

Spatial & Temporal locality
• First field reference loads whole record into cache
• Subsequent references access other fields in same cached record

Individual fields are not adjacent for coalesced accessing. Arrays-of-records need to be flattened to separate field arrays for GPU.
Code Duplication

Parallel & Serial codes can exist side-by-side:

```c
#ifdef _OPENACC
! GPU-optimized loops
#else
! CPU-optimized loops
#endif
```

Possible future compiler optimizations based on data-dependency analysis:

- Loop-splitting
- Loop interchange
- Data transposition

In some cases code changes may be performed implicitly