Xcompact3D - OpenMP and CUDA implementation

Semih Akkurt and Sylvain Laizet

Imperial College London

ExCALIBUR Turbulence Meeting
Sep 29, 2022

Overview

- This talk is about recent progress on OpenMP and CUDA implementation
- Started in June in Sylvain's group at Imperial
- First task is to implement OpenMP support for Xcompact3D
- Investigated two separate approaches for the GPU implementation

OpenMP implementation

Potential benefits of OpenMP + MPI implementation in Xcompact3D

- Reduced number of ranks
- Fewer MPI messages (though the total size is fixed)

Current Limitations in Xcompact3D

- Derivative functions are memory bandwidth bound
- Vectorisation is not possible in X pencil arrangement
- Y and Z pencil derivations suffer from scattered data access

Xcompact3D - Current Implementation X-pencil

```
do concurrent(k=1:xsize(3), j=1:xsize(2), i=1:xsize(1))
   ta1(i, j, k) = ux1(i, j, k)*ux1(i, j, k)
   tb1(i, j, k) = ux1(i, j, k)*uy1(i, j, k)
   tc1(i, j, k) = ux1(i, j, k)*uz1(i, j, k)
end do

call derx(td1, ta1, x3d_op_derxp, xsize(1), xsize(2), xsize(3))
call derx(te1, tb1, x3d_op_derx, xsize(1), xsize(2), xsize(3))
call derx(tf1, tc1, x3d_op_derx, xsize(1), xsize(2), xsize(3))
call derx(ta1, ux1, x3d_op_derx, xsize(1), xsize(2), xsize(3))
call derx(tb1, uy1, x3d_op_derxp, xsize(1), xsize(2), xsize(3))
call derx(tc1, uz1, x3d_op_derxp, xsize(1), xsize(2), xsize(3))

! Convective terms of x-pencil are stored in tg1,th1,ti1
do concurrent(k=1:xsize(3), j=1:xsize(2), i=1:xsize(1))
   dux1(i, j, k, 1) = -half*(td1(i, j, k) + ux1(i, j, k)*ta1(i, j, k))
   duy1(i, j, k, 1) = -half*(te1(i, j, k) + ux1(i, j, k)*tb1(i, j, k))
   duz1(i, j, k, 1) = -half*(tf1(i, j, k) + ux1(i, j, k)*tc1(i, j, k))
end do
```

Xcompact3D - Cache Blocking Support X-pencil

```
! allocate xdu, xdv, xdw, xd2u, xd2v, xd2w, xud, xvd, xwd, xdud, xdvd, xdwd
! all x pencil sized (nx length)
do k = 1, nz
   do j = 1, ny
      call derx(xdu, ux1(:, j, k), x3d_op_derxp, xsize(1), xsize(2), xsize(3))
      call derx(xdv, uy1(:, j, k), x3d_op_derx, xsize(1), xsize(2), xsize(3))
      call derx(xdw, uz1(:, j, k), x3d_op_derx, xsize(1), xsize(2), xsize(3))

      xud = ux1(:, j, k)*ux1(:, j, k)
      xvd = ux1(:, j, k)*uy1(:, j, k)
      xwd = ux1(:, j, k)*uz1(:, j, k)

      call derx(xdud, xud, x3d_op_derx, xsize(1), xsize(2), xsize(3))
      call derx(xdvd, xvd, x3d_op_derxp, xsize(1), xsize(2), xsize(3))
      call derx(xdwd, xwd, x3d_op_derxp, xsize(1), xsize(2), xsize(3))

      dux1(:, j, k, 1) = -half*(xdud + ux1(:, j, k)*xdu)
      duy1(:, j, k, 1) = -half*(xdvd + ux1(:, j, k)*xdv)
      duz1(:, j, k, 1) = -half*(xdwd + ux1(:, j, k)*xdw)
   end do
end do
```

Pencil Grouping Approach - Vectorised Data Structure

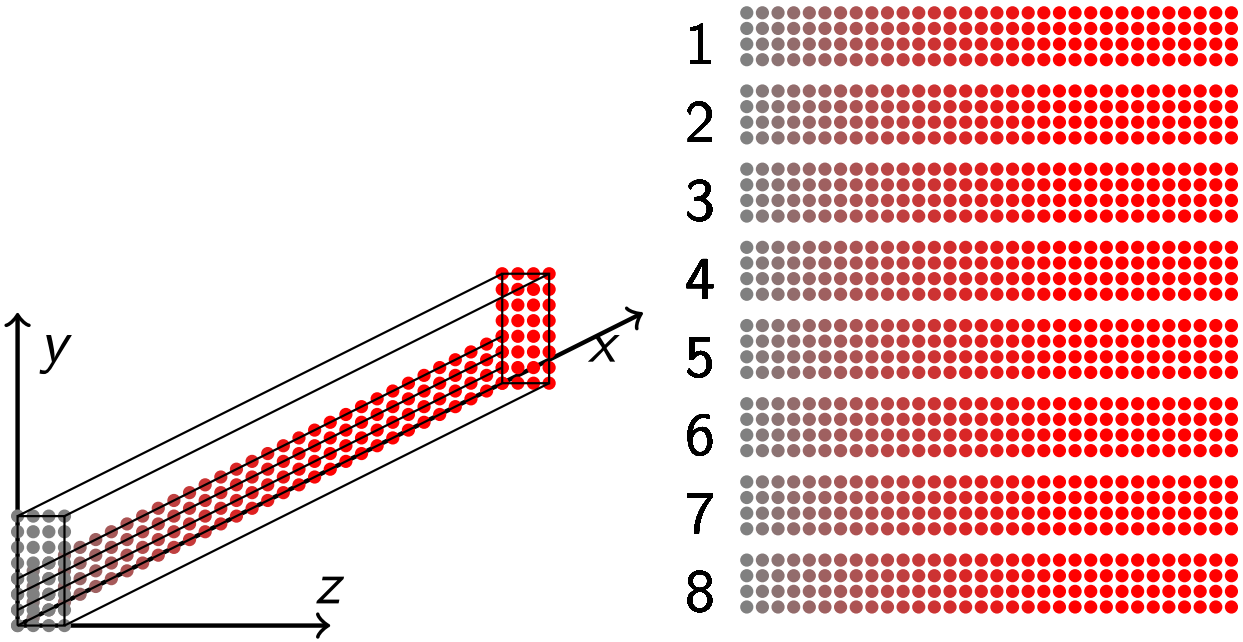

Figure: Pencil grouping data structure for vectorisation support.

Xcompact3D - Vectorised Derivation Function

```
! RHS stuff for Dirichlet BC both ends
!$omp simd
do i = 1, SZ
   tx(i, 1) = derps%af1*ux(i, 1) + derps%bf1*ux(i, 2) + derps%cf1*ux(i, 3)
   tx(i, 2) = derps%af2*(ux(i, 3) - ux(i, 1))
end do
!$omp end simd
do j = 3, nx - 2
   !$omp simd
   do i = 1, SZ
      tx(i, j) = derps%afi*(ux(i, j + 1) - ux(i, j - 1)) &
                 + derps%bfi*(ux(i, j + 2) - ux(i, j - 2))
   end do
   !$omp end simd
end do
! nx-1 <= j <= nx
!$omp simd
do i = 1, SZ
   tx(i, nx - 1) = derps%afm*(ux(i, nx) - ux(i, nx - 2))
   tx(i, nx) = -derps%afn*ux(i, nx) - derps%bfn*ux(i, nx - 1) - derps%cfn*ux(i, nx
end do
!$omp end simd
```

Summary of OpenMP implementation

- Focused entirely on the transport equation
- Reduced bandwidth use via cache blocking and introduced vectorisation suppport
- Memory requirement is significantly reduced
- Switched to 1D decomposition with asynchronous communication
- Fewer ranks thus fewer MPI messages
- Data transposes are required when switching between pencil arrangements

OpenMP - Benchmarks on ARCHER2

n	Xcompact3D Full code	Xcompact3D Transport Eq.	New Code Transport Eq.
512^3 @2 Nodes	95.8	57.1	8.0
1024^3 @4 Nodes	444.2	259.1	39.2

GPU Implementation - Two strategies

Strategy 1
Pencil group

Strategy 2
Single pencil[1]

1. László, Giles, and Appleyard. 2016. Manycore Algorithms for Batch Scalar and Block Tridiagonal Solvers. ACM Trans. Math. Softw.

Feasibility of the Pencil Groups Strategy on GPUs

- CPUs have enough cache to store the intermediate result in the cache.
- GPU cache size is the main limitation for the pencil groups approach.

	Group size (SZ)			
n	1	8	32	64
256	2 KiB	16 KiB	64 KiB	128 KiB
512	4 KiB	32 KiB	128 KiB	256 KiB
1024	8 KiB	64 KiB	256 KiB	512 KiB
2048	16 KiB	128 KiB	512 KiB	1024 KiB
4096	32 KiB	256 KiB	1024 KiB	2048 KiB
8192	64 KiB	512 KiB	2048 KiB	4096 KiB

	NVIDIA A100	AMD MI 250X	NVIDIA H100	Intel Ponte Vecchio
# of SM/CU/XeC	108	220	132	128
# of DP cores per CU	32	32	64	32
Total L1 Cache	192KiB	80KiB	264KiB	512KiB
Shared L1 (Scratch)	164KiB	64KiB	228KiB	512KiB
L2 Cache (shared)	40MiB	16MiB	50MiB	288MiB
Memory BW [GB/s]	1555GiB/s	3277GiB/s	3000GiB/s	3277GiB/s

Bandwidth comparison of the two strategies on GPUs

Strategy 1

Fuse derivative functions
Fuse RHS and Forward Thomas

- Read u
- Apply RHS and Forward Thomas
- Write du, dud, d2u
- Read du, dud, d2u
- Apply Backward Thomas
- Write u*du+dud+d2u

29 or 25

Strategy 2
Single Pencil

```
temp1 = u
temp2 = u*v
tr = d(u*v)
temp = d(u)
tr += temp*u
temp = d2(u)
tr += temp
```

8

```
temp1 = u*v
tr = d(u*v)
temp1 = u
temp = d(u)
tr += temp*u
temp = d2(u)
tr += temp
```

11

```
tr = d(u*v)
temp = d(u)
tr += temp*u
temp = d2(u)
tr += temp
```

17