CLUBB GPUization and Performance Portability

Gunther Huebler

In Collaboration with:
Vincent Larson, Supreeth Suresh, Jian Sun, Sheri Voelz,
John Dennis, Brian Dobbins, NCAR and DOE

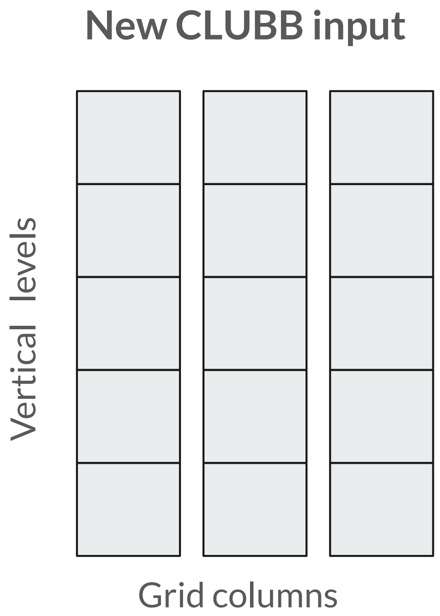
CLUBB - Cloud Layers Unified by Binormals

Designed to operate over a single column of vertical levels

Majority of calculations in the vertical are independent

+ Grid columns are completely independent

= Lots of parallelism to exploit



Initial loop pushing - Top Down

```
do i = 1, ngrdcol
    call advance_clubb
        call advance_xm
            ...
            z(:) = x(:) * y(:)
```

```
call advance_clubb
    do i = 1, ngrdcol
        call advance_xm
            ...
            z(i,:) = x(i,:) * y(i,:)
```

```
call advance_clubb
    do i = 1, ngrdcol
        call advance_xm
            ...
    do i = 1, ngrdcol
        z(i,:) = x(i,:) * y(i,:)
```

```
call advance_clubb_core
    call advance_xm
        do i = 1, ngrdcol
            ...
    do i = 1, ngrdcol
        do k = 1, nz
            z(i,k) = x(i,k) * y(i,k)
```

Push loop into procedure, creating one massive loop - requires adding a dimension to fields

Separate computation and procedure calls by breaking up big loop

Replace vector notation with loops and push another loop down

Loop Pushing Challenges

Global variables

- Loop pushing should always produce identical output
- Breaking up loops may cause errors if global variables are used

Duplicate procedures for different sized data are needed

- 1D/2D/scalar procedures wrapped with and interface works well
- using `**acc routine**` is the other option (but not recommended)

GPUization

```
call advance_clubb_core
    call advance_xm
        !$acc copyin() copyout() create()
        !$acc parallel loop
        do i = 1, ngrdcol
            ...
    do i = 1, ngrdcol
        do k = 1, nz
            z(i,k) = x(i,k) * y(i,k)
```

Start at lowest levels
- add parallel loop directives
- add copyin / copyout / create
  for inputs / outputs / locals

```
call advance_clubb_core
    !$acc copyin() copyout() create()
    call advance_xm
        !$acc create()
        !$acc parallel loop
        do i = 1, ngrdcol
            ...
    !$acc parallel loop
    do i = 1, ngrdcol
        do k = 1, nz
            z(i,k) = x(i,k) * y(i,k)
```

- move data statements up a level, leaving local allocations (create)

- add loop directives

```
!$acc copyin() copyout()
call advance_clubb_core
    !$acc create()
    call advance_xm
        !$acc create()
        !$acc parallel loop
        do i = 1, ngrdcol
            ...
    !$acc parallel loop collapse(2)
    do i = 1, ngrdcol
        do k = 1, nz
            z(i,k) = x(i,k) * y(i,k)
```

Repeat until all computations are in parallel loops and all data copies are at top level

Challenges

Bug hunting

- Finding bugs can require lots of iteration, GPUizing small sections at a time

Strange bugs

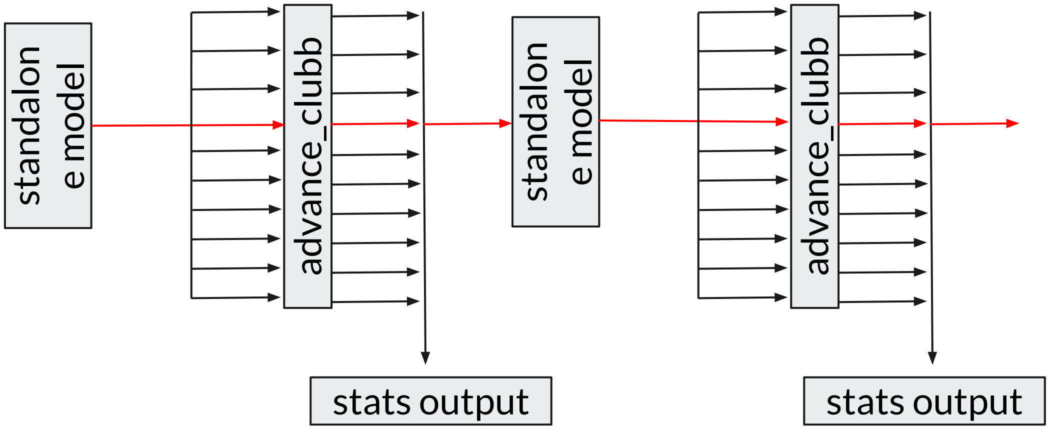
- Size 0 array allocation (create) causes memory errors for implicitly structured statements

Testing

- Output won't be identical, can be hard to differentiate bit-differences from errors

Testing

- Modified standalone model to create and output fake columns
- Measured output discrepancies at different optimization levels
- Measured output discrepancies when adding an intentional error
- Determined a threshold to distinguish error from bit-changes



GPUization - Linear Algebra

Historically CLUBB has used Lapack

We created custom LU decomp solvers
- no need for external library

Also 4x faster than Lapack on CPUs
- solvers are tailored to 3 and 5 band matrices
- no copying matrix into a standard form
- pivoting doesn't seem necessary for our use case

Converting from OpenACC to OpenMP

Converting from ACC to OMP directives is entirely automated
- total of 2640 lines of acc directives in CLUBB
- https://github.com/intel/intel-application-migration-tool-for-openacc-to-openmp

Most ACC directives have direct OMP equivalents
- **acc parallel loop = omp parallel loop**
- **acc data** = **omp data** / **acc enter data** = **omp enter data**
- no **gang/vector** in OMP
- no **default(present)** in OMP

Performance Results

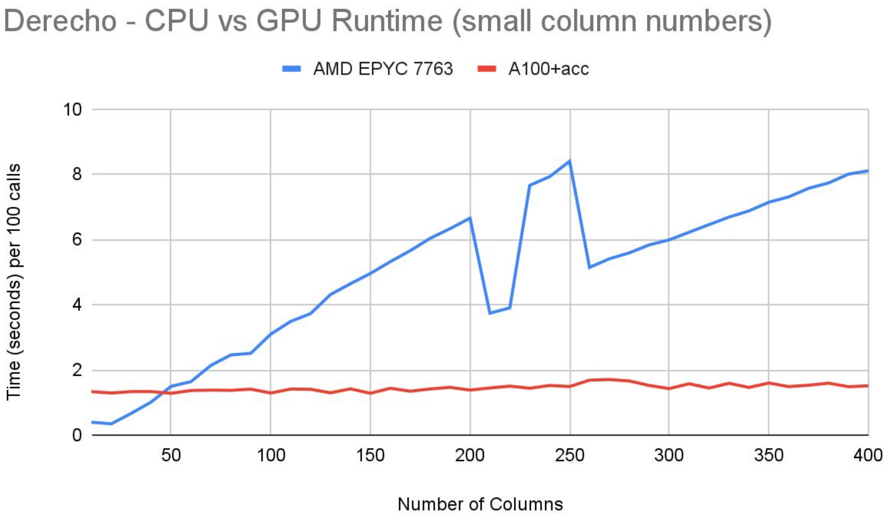
Numbers were gathered using the standalone “fake” column method

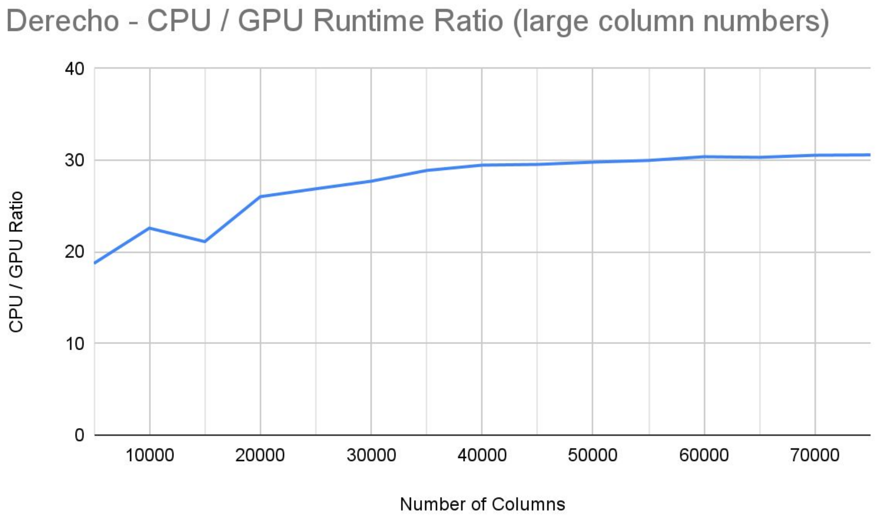
Times shown refer to 100 calls to **advance_clubb_core**

Times were gathered using a case with 134 vertical levels

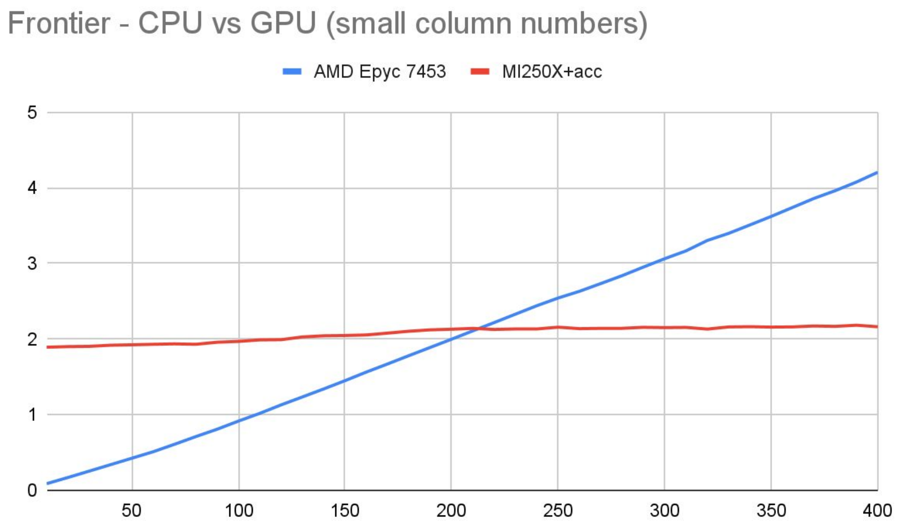
Results from Derecho use NVHPC on Nvidia A100 + AMD EPYC 7763
Results from Frontier use CRAY on AMD MI250X + “optimized” AMD EPYC 7453

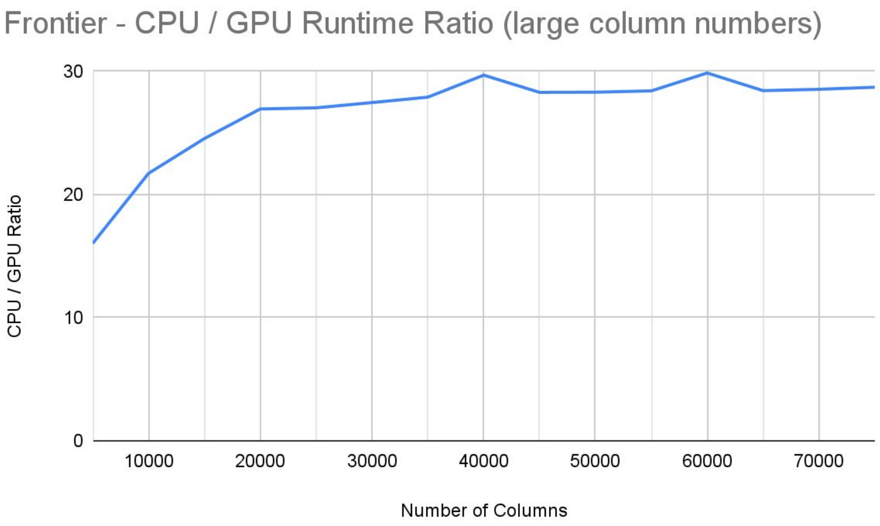
CPU (single core) vs GPU
Derecho -- AMD EPYC 7763 vs NVIDIA A100





CPU (single core) vs GPU
Frontier -- AMD EPYC 7453 vs AMD MI250X





CPU Multicore vs GPU?

Depends on hardware

- Derecho - 32 CPU cores per GPU
- Frontier - 8 CPU cores per GPU

Difficult finding the best configuration

- 32 threads on a 32 core CPU is NOT optimal
- optimal thread number depends on columns

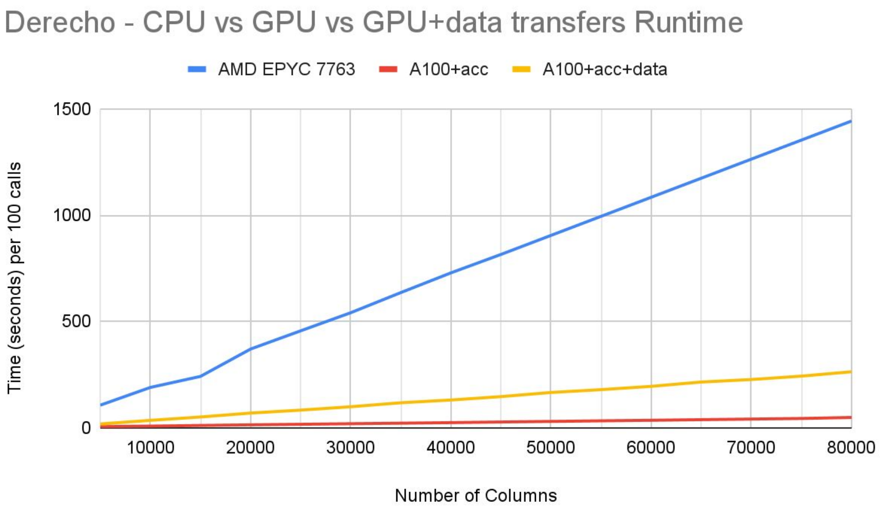
Different ways of parallelizing on CPU

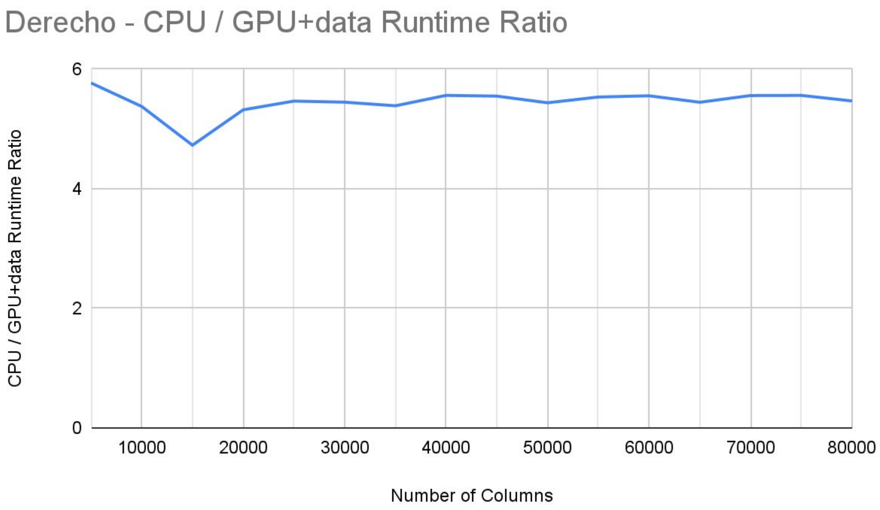
- openmp cpu threads
- mpi processes
- acc on CPU cores (-acc=multicore)

columns	1 thread	2 threads	4 threads	6 threads	8 threads	10 threads	12 threads	16 threads	20 threads	24 threads	28 threads	32 threads	36 threads	40 threads	48 threads	56 threads	64 threads
10	0.1739199	0.3609052	0.4587281	0.507148	0.572598	0.5639191	0.616662	0.635942	0.6184812	0.6244719	0.6767859	0.6896939	0.679534	0.7116499	0.7057769	0.760777	0.8124919
20	0.2850699	0.4317009	0.506021	0.5210421	0.5755758	0.6459279	0.6289639	0.704123	0.8138809	0.72914	0.8017609	0.8190539	0.811753	0.8236041	0.856842	0.9077468	0.975316
30	0.5440869	0.68348	0.6988549	0.738045	0.791532	0.8275349	0.8654101	0.931721	1.024775	1.092733	1.175297	1.223504	1.280154	1.347786	1.382214	1.45085	1.62734
40	0.7485809	0.7966721	0.769341	0.8572559	0.9250832	0.8949299	0.9855399	1.108908	1.125388	1.311437	1.3832	1.499716	1.523771	1.607126	1.6931	1.780439	1.970042
50	0.9657118	1.029333	0.95718	1.005182	1.086549	1.130519	1.183012	1.301324	1.466663	1.598719	1.682908	1.783628	1.939221	2.044103	2.284439	2.432778	2.659965
60	1.101505	1.112052	1.033386	1.070996	1.143341	1.143475	1.231714	1.409922	1.523065	1.680631	1.806796	1.905102	2.051246	2.189286	2.39743	2.649514	2.969754
70	0.879859	0.8947921	0.8099699	0.7895629	0.7784898	0.8570728	0.9018562	0.969584	1.065801	1.149292	1.220721	1.302529	1.381789	1.459519	1.627085	1.744545	1.937921
80	1.429252	1.393182	1.207159	1.226081	1.252189	1.263199	1.323522	1.444937	1.563762	1.7164	1.789174	1.914269	2.010888	2.099603	2.375337	2.56461	2.971143
90	1.815763	1.686949	1.438051	1.468617	1.56028	1.588825	1.613196	1.775803	1.927514	2.104071	2.267392	2.43217	2.627018	2.766356	3.109337	3.415899	3.860078
100	2.024152	1.726192	1.477904	1.578195	1.671714	1.645127	1.823313	1.981456	2.011258	2.278621	2.508089	2.70891	2.962937	3.158046	3.4951	3.906998	4.510304
200	4.128615	3.482141	2.694397	2.672514	2.676385	2.716888	2.725997	2.900885	3.181673	3.326735	3.528017	3.75147	3.982535	4.332677	4.739707	5.286816	6.015594
300	4.638511	3.636836	2.737644	2.511585	2.474988	2.610872	2.597033	2.761736	3.042367	3.078614	3.28276	3.538825	3.772051	3.949828	4.413426	4.922977	5.665914
400	6.215342	4.821171	3.457494	3.19754	3.006222	3.120345	3.410393	3.346479	3.625428	3.857721	3.961379	4.385865	4.43636	4.664742	5.142087	5.782276	6.254519
500	11.02653	5.71147	4.113483	3.755767	3.830324	3.772901	3.670212	3.86019	4.154749	4.288191	4.466737	4.927687	5.054929	5.190607	5.75955	6.231572	6.810253
600	12.81196	8.162413	5.971601	5.546429	5.236706	5.354739	5.220313	5.522388	5.728132	5.79775	5.990475	6.260918	6.602459	6.82615	7.253183	7.909901	8.524744
700	15.44185	9.575266	6.834738	6.290747	6.27914	6.111324	5.88725	6.217782	6.415081	6.534464	6.86054	7.077579	7.113693	7.521591	7.891208	8.415764	9.338909
800	18.02168	11.00163	7.918591	7.539292	7.1904	7.037462	6.950239	7.066599	7.22304	7.401031	7.80851	7.919706	8.642665	8.801287	8.941633	9.782146	10.95606
900	17.86609	12.20995	8.821412	7.949412	7.908643	7.780753	7.306122	7.530461	7.740519	8.152866	8.648833	9.084002	9.390467	9.605153	10.15662	10.85245	11.54268
1000	20.3161	10.13947	6.952821	6.007382	5.809196	5.918099	5.465507	6.094188	5.682955	5.86698	5.962132	6.308026	6.613776	6.571036	7.391406	7.778223	8.170792
2000	43.04111	20.1945	14.32025	11.68917	10.66884	11.80393	11.76958	9.677084	10.22121	10.14732	10.77386	10.66382	11.39746	11.2523	12.07893	12.95129	12.6793
4000	74.21236	42.17818	26.56903	22.76451	22.92345	20.67723	20.72448	21.32133	21.31335	20.46025	21.80457	23.73321	23.1146	23.66351	25.14736	26.35736	27.09214
6000	107.0354	62.10591	38.5945	32.61211	31.46306	31.94279	32.89522	30.87162	30.59357	31.79996	31.3295	32.12423	32.74764	33.60072	34.6663	35.57525	36.77738
8000	142.1556	84.524	51.0033	44.72787	41.07072	38.31995	38.98942	38.2856	36.83006	40.625	40.07935	41.47214	42.63329	42.13686	43.88255	47.65929	48.64026
10000	174.8267	104.4793	63.28668	54.25824	54.42431	48.32491	49.0793	45.96922	48.52158	48.68888	50.378	50.78878	49.34952	51.65981	52.68114	57.44966	59.80953
15000	235.0483	158.7512	93.65284	85.01671	77.30313	73.57104	70.07619	72.52956	71.49947	73.25378	73.58936	75.8078	77.22366	76.25075	78.42762	81.82963	83.87129
20000	307.0153	216.9319	139.5679	116.1917	105.3447	100.6086	108.5596	96.1376	98.33346	101.3841	98.72761	104.6584	103.9692	109.4046	109.2813	114.5127	115.042
25000	386.1227	273.4608	161.0923	141.1918	140.1698	136.5088	121.822	121.7375	125.7266	120.81	124.2464	128.3898	130.0912	127.7298	133.827	143.267	147.3376
30000	460.952	329.6489	195.1177	159.6235	166.8193	152.3523	147.9067	151.6026	152.9902	153.4277	153.7256	152.0176	151.2462	160.6244	160.5072	166.4504	169.7961
35000	613.299	417.1245	248.6876	213.9641	201.563	195.0272	199.2155	198.2973	193.2568	200.1044	196.5808	201.2268	207.2881	214.0457	215.7966	225.9367	239.2946
40000	704.7343	480.2349	280.2366	245.3722	232.6216	225.0685	222.5005	221.6313	233.1876	226.4225	226.5018	230.4022	241.0962	237.9516	253.3036	262.6156	274.3885
45000	789.2264	536.4769	318.4565	273.8456	262.0843	251.9659	247.5313	255.3855	249.7669	251.3868	252.5438	256.6631	261.7682	276.8968	274.1662	291.6109	316.0999
50000	884.4467	602.5971	353.4376	307.2018	293.7679	280.8431	276.5077	276.1331	277.6115	277.6332	284.4915	292.1806	297.3101	300.2007	312.0501	325.4406	335.6428
60000	1061.662	725.6881	429.8362	372.9828	375.8567	361.0919	337.1772	349.7416	337.9184	339.4349	344.6499	350.0494	355.3468	364.4255	369.5547	384.3665	406.5944
70000	1253.746	857.3915	506.304	437.4329	428.3044	403.6089	395.542	398.5212	399.2153	404.4551	409.2749	414.7741	423.7222	425.9652	436.9974	452.156	481.944
80000	1447.946	988.3656	594.9959	509.1799	485.8095	497.1215	463.203	480.148	462.9361	478.6432	482.7906	483.7289	485.9384	488.9766	501.6219	514.2358	549.3961

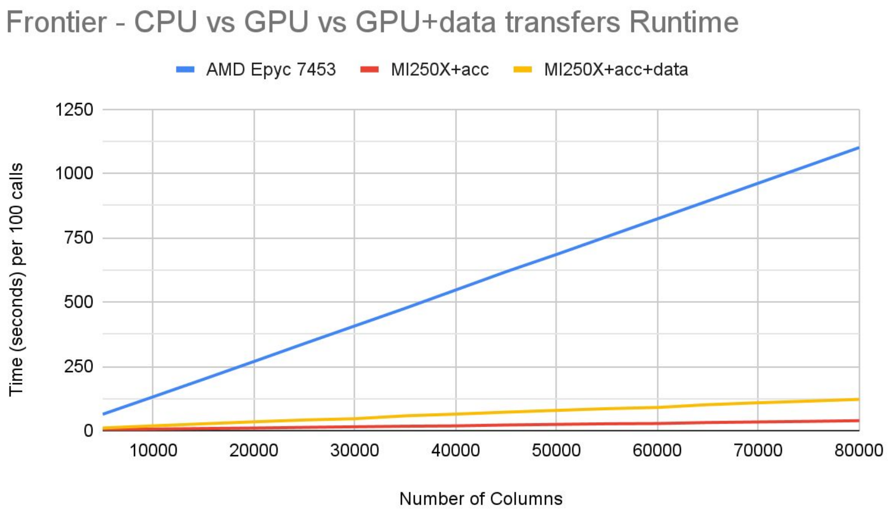
Using acc multicore (acc on CPU cores) on Derecho. The largest speedup observed was 4.4x over single core runtime using 16 threads and 2000 columns

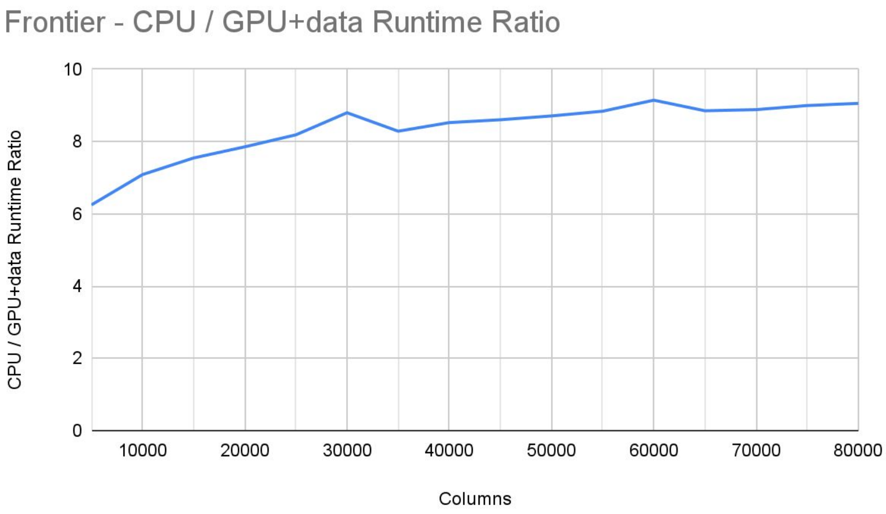
Cost of Data Transfers --- Derecho A100+acc+nvhpc





Cost of Data Transfers --- Frontier MI250X+acc+cray





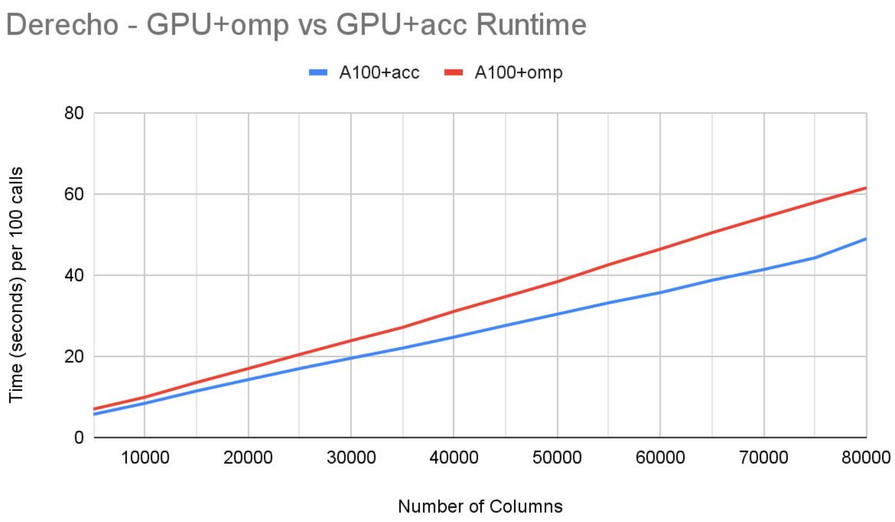
Cost of Data Transfers

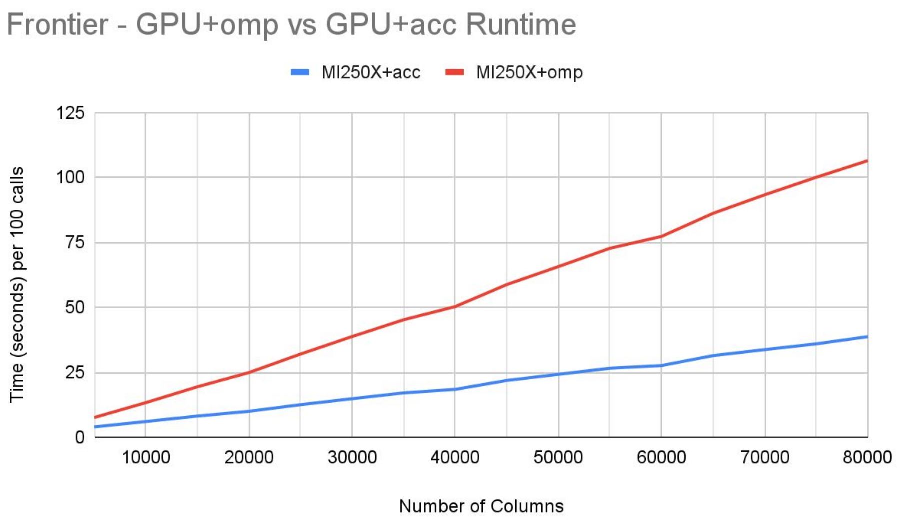
Frontier GPU+data transfers: ~8x speedup vs single CPU core

Derecho GPU+data transfers: ~5x speedup vs single CPU core

Recall: The largest CPU multicore speedup vs a single CPU core was 4.4x
(only tested on Derecho using -acc=multicore)

OpenACC vs OpenMP --- Runtime Comparison





OpenACC vs OpenMP --- What's Slower?

Loops in kernels perform worse with OMP, but not consistently

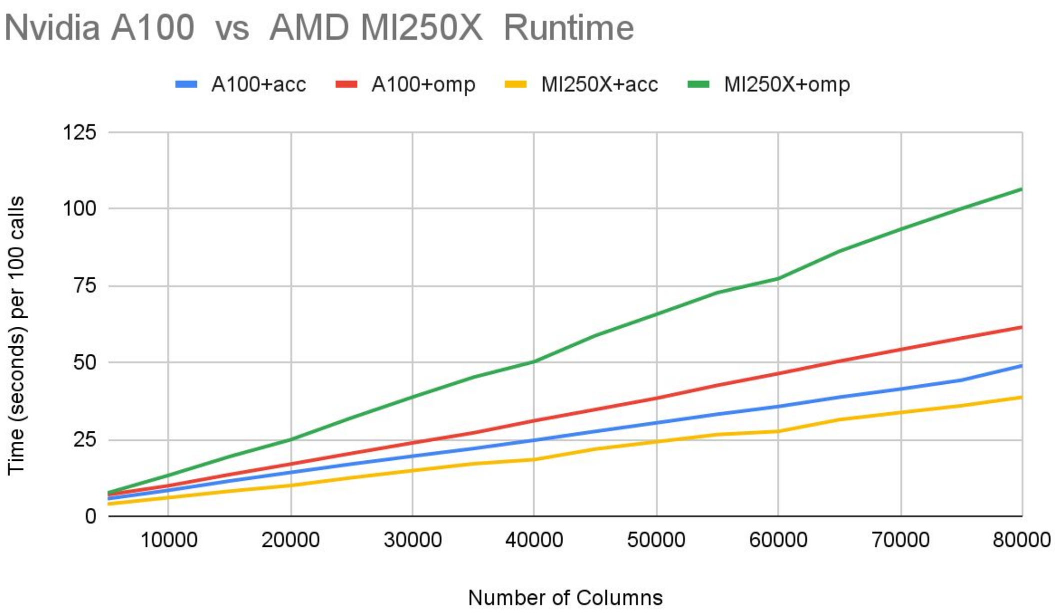
```
!$acc parallel loop gang vector collapse(2) default(present)
!$omp target teams loop collapse(2)
do i = 1, ngrdcol
  do k = 2+num_draw_pts, upper_hf_level-num_draw_pts
    k_start = k - 2
    k_end   = k + 2
    invrs(i,k) = one / sum( field(i,k_start:k_end) )
  end do
end do
```

Derecho+NVHPC - ACC: 181us   OMP: 6.4ms (35x slower)
OMP is NOT slower here with Frontier+CRAY

```
!$acc parallel loop gang vector collapse(2)
!$omp target teams loop collapse(2)
do k = 2, nz-1
  do i = 1, ngrdcol
    low_lev  = max( low_levs(i,k), 2 )
    high_lev = min( high_levs(i,k), nz )
    max_x(i,k) = max_x_lev(i,low_lev)
    do j = low_lev, high_lev
      max_x(i,k) = max( max_x(i,k), max_x_lev(i,j) )
    end do
  end do
end do
```

Frontier+CRAY - ACC: 161us   OMP: 27ms (165x slower)
OMP is NOT slower here with Derecho+NVHPC

A100 vs MI250x



Tips and Tricks - ACC + OMP

No issue with OpenMP and OpenACC directives on the same loops

Define explicitly which directives to obey in FFLAGS and LDFLAGS

- NVHPC: -[no]acc -mp=gpu
- CRAY: -h [no]omp -h [no]acc

Tips and Tricks - NVHPC Function Bug?

Arguments cannot appear as input and output
- no issue in subroutines as **inputs**

```
field = func( field )
```

✗

This is a workaround

```
tmp = func( field )

!$acc parallel loop
do k = 1, nz
    field(k) = tmp(k)
end do
```

✓

Tips and Tricks --- Different Data Directives

Explicitly structured

- lifetime of data region is until **end**
- explicit regions can be nested

```
subroutine something
    !$acc data copyin() copyout() create()
    ...
    !$acc end data
```

Implicitly structured

- lifetime of data region is procedure
- least customizable

```
subroutine something
    !$acc declare copyin() copyout() create()
    ...
```

Unstructured

- no regions, data lives until **delete**
- most customizable

```
subroutine something
    !$acc enter data copyin() copyout() create()
    ...
    !$acc exit data delete()
```

Tips and Tricks - Cray vs NVHPC

CRAY - **default(present)** will prevent scalars from being copied in before kernels

NVHPC - **default(present)** only prevents automatic array copies

- Using **default(present)** with cray will require explicit copyins for scalars

CRAY - **acc create()** on a previously allocated variable will overwrite the allocation

NVHPC - **acc create()** on a previously allocated variable will have no effect

- This is a way a bug might sneak by if you only test with NVHPC

Tips and Tricks - reduction

Goal - GPUized version of “**any()**”

Problem - “**any**” is a serial operation

Solution - mimic with “**reduction**” clause

Behavior in order

1. copyin boolean value set by CPU to GPU (light blue)
2. check threshold in parallel (first blue)
3. perform reduction calculation (second blue)
4. copyout final boolean from GPU to CPU (pink)

```
l_field_below_threshold = .false.
!$acc parallel loop gang vector collapse(2) default(present) &
!$acc          reduction(.or.:l_field_below_threshold)
do k = 1, nz
  do i = 1, ngrdcol
    if ( field(i,k) < threshold ) then
      l_field_below_threshold = .true.
    end if
  end do
end do
```

Source code using OpenACC

Memory Operation / Kernel execution over time (nsys-profile)

Tips and Tricks - CRAY + Lapack

We encountered a strange error when compiling Lapack using CRAY at optimization level -O1 or above

- Symptom: **undefined reference to _ismin_ / _idmin_ / _ismax_ / ...**
- Fix: compile with secret flag "**-hnopattern**"
- https://github.com/OpenMathLib/OpenBLAS/issues/3651

Tips and Tricks - Profiling

Omnitrace is the AMD equivalent of Nvidia Nsight Systems

- https://github.com/AMDResearch/omnitrace

- https://developer.nvidia.com/nsight-systems