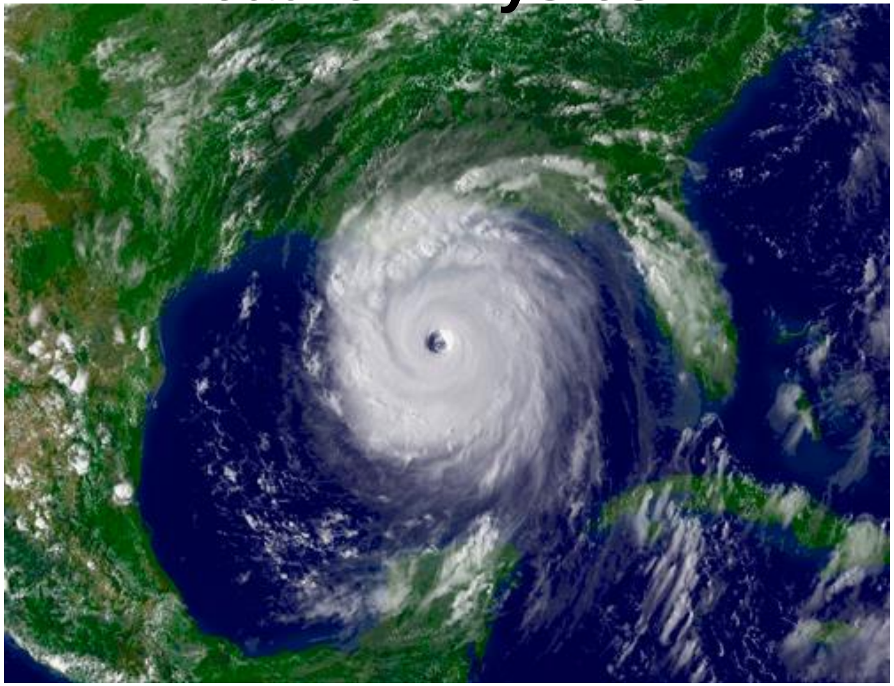
Optimization Strategies for WSM6

Weather Physics



1. Motivation
2. Overview of NEPTUNE and WSM6
3. Overview of KNL architecture
4. Methodology
5. Stand alone Experiments
6. WSM6 results
7. Discussion and Conclusion

Presented by:
T.A.J.Ouermi, Aaron Knoll, Mike Kirby, Martin Berzins

www.sci.utah.edu

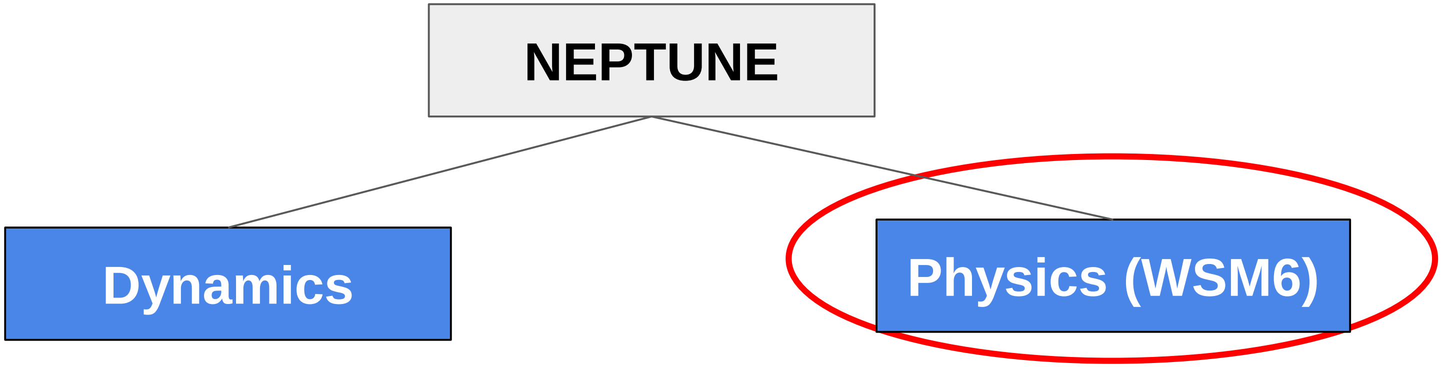
User Productivity Enhancement, Technology Transfer, and Training (PETTT)

Intel Parallel Computing Center

THE UNIVERSITY OF UTAH

Motivation

- Optimizing Numerical Weather Prediction (NWP) codes leads to faster forecast.
- “Navy Environmental Prediction sysTem Utilizing the NUMA corE”(NEPTUNE)
- This optimization targets intel KNL and potential future architectures because NWP codes port easily to Intel MIC as opposed to GPUs
- Understand how to effectively use OpenMP for portable shared memory parallelism in the context of NEPTUNE.



NEPTUNE

Dynamics

Physics (WSM6)

- Uses spectral elements --> high scalability because of small communication.

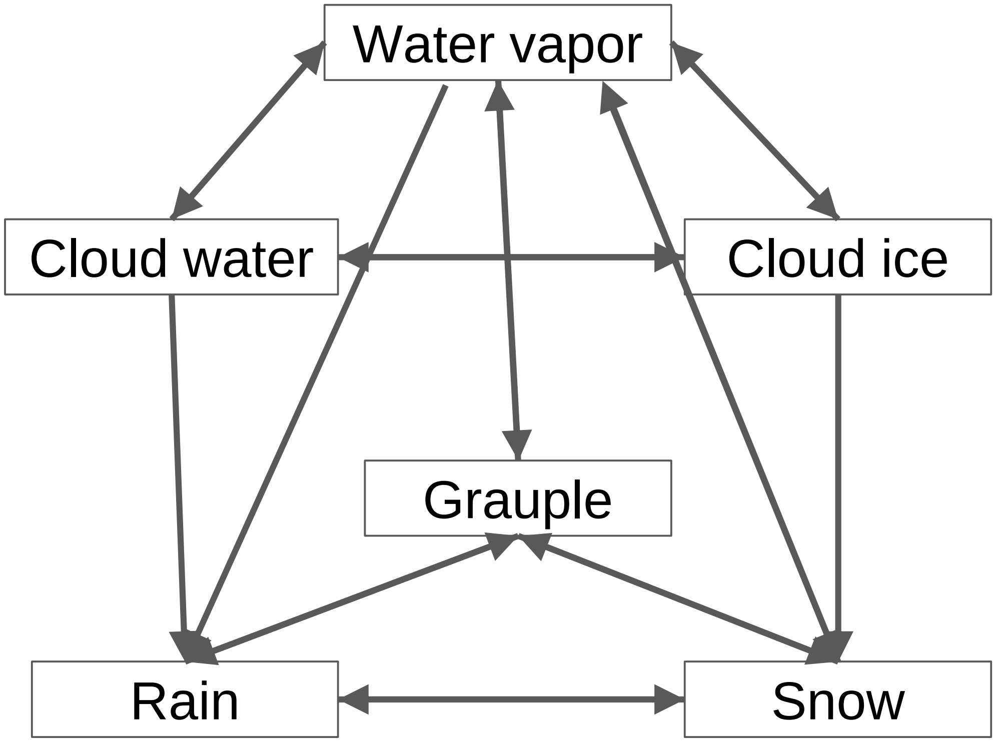
- Non-hydrostatic Unified Model of the Atmosphere

- Does not Scale Well

- Comprised of surface flux, boundary layer, shallow convection, warm-rain microphysics, and radiation processes

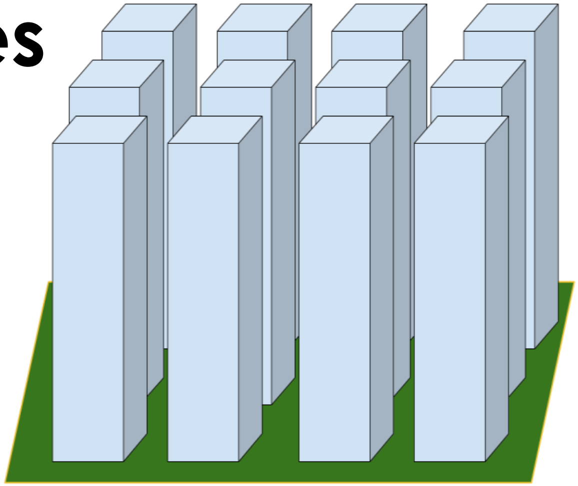
- WSM6 is a components of the physics part of NEPTUNE

Physics Optimization Challenges

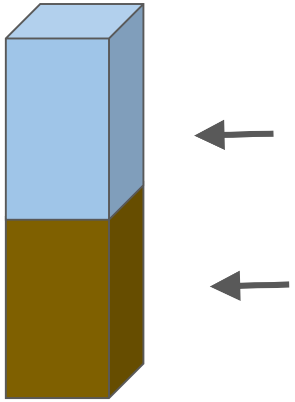

Water vapor
Cloud water
Cloud ice
Grauple
Rain
Snow

WRF single-moment 6-class Microphysics Scheme (WSM6)

- WSM6 models various precipitation phenomena within vertical columns, exchanged through dynamics
    - 27 loops over 39 arrays with conditionals, array copies, and subroutine calls.
- Irregularity and complexity of physics between various states makes optimization challenging.



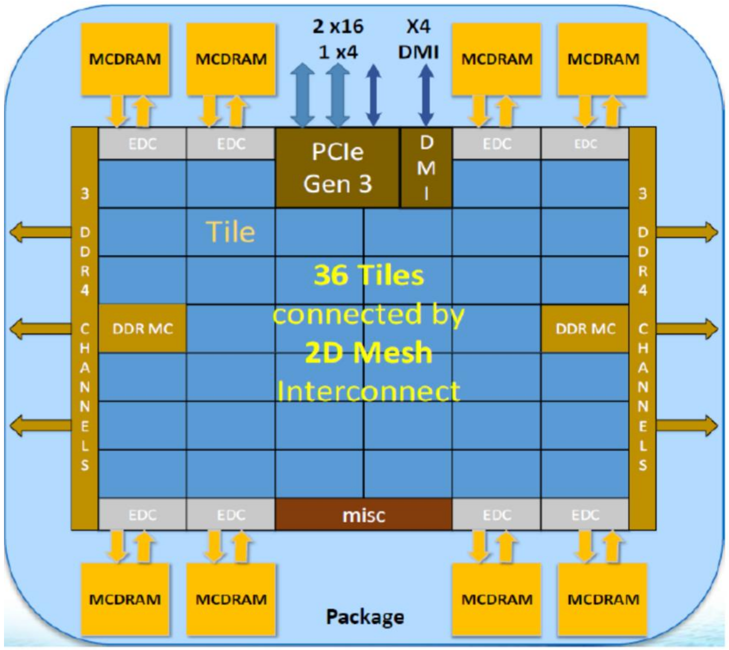
Vertical Physics representation


Sea
Land

Like soft hail and about 2-5mm in diameter

Grauple Particles

Overview of KNL Architecture

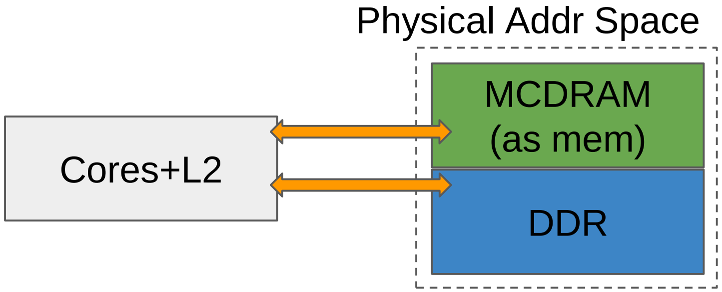


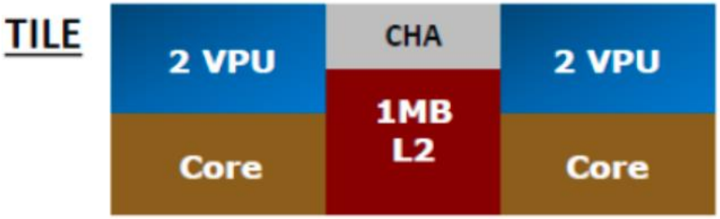
Chip: 36 Tiles interconnected by **2D Mesh**
Tile: 2 Cores + 2 VPU/core + 1 MB L2

Memory: MCDRAM: 16 GB on-package; High BW
DDR4: 6 channels @ 2400 up to 384 GB
IO: 36 lanes PCIe* Gen3. 4 lanes of DMI for chipset
Node: 1-Socket only
Fabric: Intel® Omni-Path Architecture on-package (not shown)

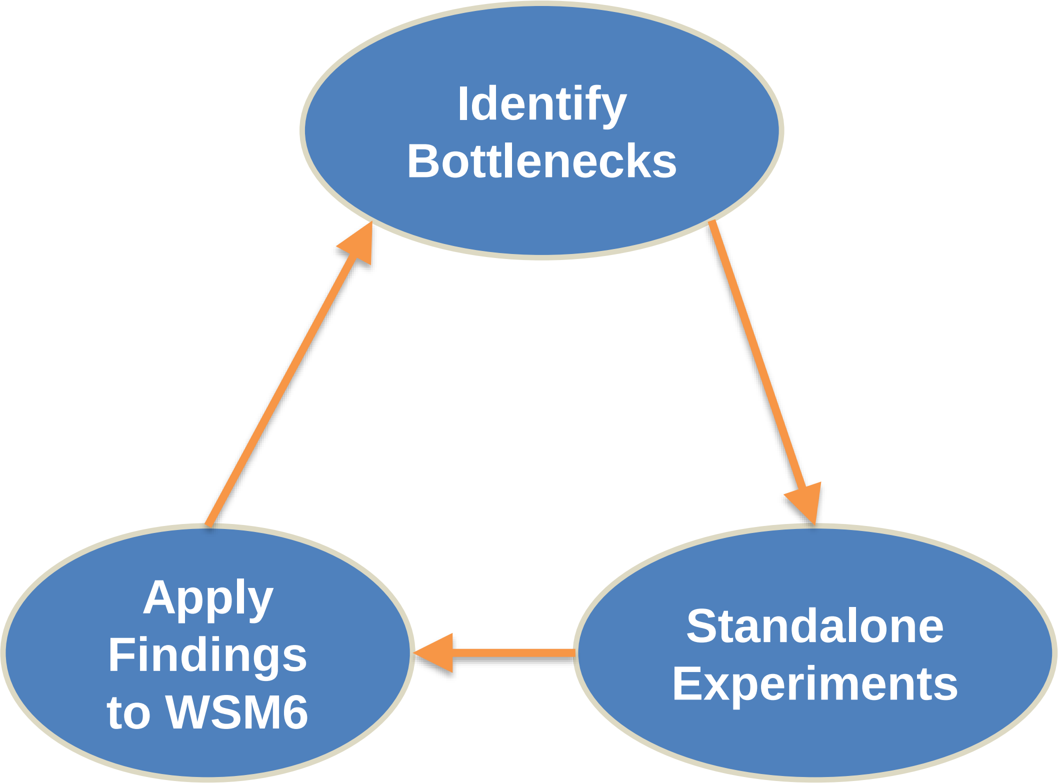
Vector Peak Perf: 3+TF DP and 6+TF SP Flops
Scalar Perf: ~3x over Knights Corner
Streams Triad (GB/s): MCDRAM : 400+; DDR: 90+

MCDRAM ~5X Higher BW than DDR





Methodology



Identify Bottlenecks

- Wall clock time (at each loop)
  Vtune profiler
- Adviser, compiler optrpt. output

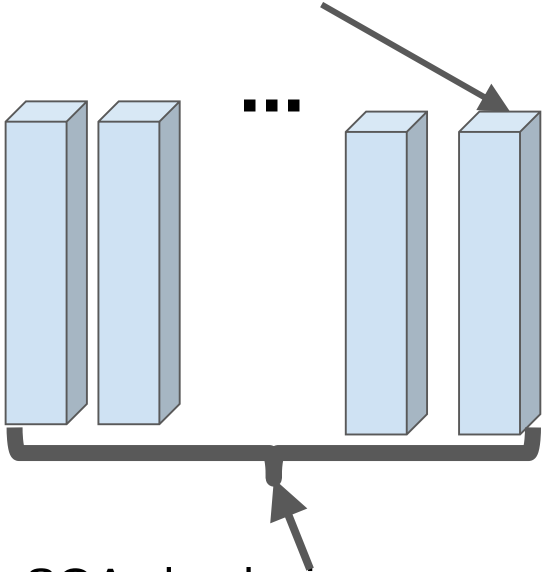
Standalone Experiments

- Examine OpenMP, and structures of arrays (SOA) behavior on code's subsets in controlled setting

Apply Findings to WSM6

- Threads (OMP PARALLEL, DO)
- SIMD (OMP SIMD, DO SIMD)

Structure of Arrays (SOA)



Physics column

...

SOA chunk size. Corresponds to parts of i loop.

AOS

```
struct aos{
  double x, y, z;
};
struct aos
points[1024];
```

SOA

```
struct soa{
  double x[1024], y[1024];
  double z[1024];
};
struct soa points;
```

Basic AOS to SOA

```
!$OMP DO
for k=1, km
for i=1,im
  dza(i,k)=zi(i,k)-za(i,k)
enddo
Enddo
!$OMP END DO
```

```
!$OMP DO
for i=1,im
for k=1, km
  dza(k,i)=zi(k,i)-za(k,i)
enddo
Enddo
!$OMP END DO
```

Transpose example

- Simple example of SOA.
- Figure to the right shows actual SOA used in WSM6 optimization.
- Chunk size is chosen to be multiple of vector unit length.
- Top down optimization approach = From “high-level” to “low-level”

Standalone Experiments

- **OpenMP functionality with a non-trivial WSM6 loop**
  - OMP PARALLEL and OMP DO constructs Using WSM6 loop 12
  - Functionality of conditionals and nested conditionals
  - Functionality of subroutine calls
- **Add OpenMP directives to individual loops**
  - OMP SIMD, OMP DO SIMD constructs
  - On sample code with conditional and subroutine call
- **SOA with 1D and 2D arrays**
  - Size and structure of SOA.

Pthreads vs OpenMP (in C)

	Pthread		OpenMP	
Threads	Thread creation (us)	Context switch (us)	Thread creation (us)	Context Switch (us)
2	199	0.6	14311	6.0
8	121	1.2	4209	1.4
32	102	1.0	1654	0.8
64	99	**1.0**	1417	**0.9**

- OpenMP thread creation time is significant
  - The first PARALLEL section costs a lot
  - Subsequent PARALLEL sections (with small KMP_BLOCKTIME), or DO constructs within one section – not much worse than pthreads.
- Context switches are in fact slightly cheaper than with Pthreads

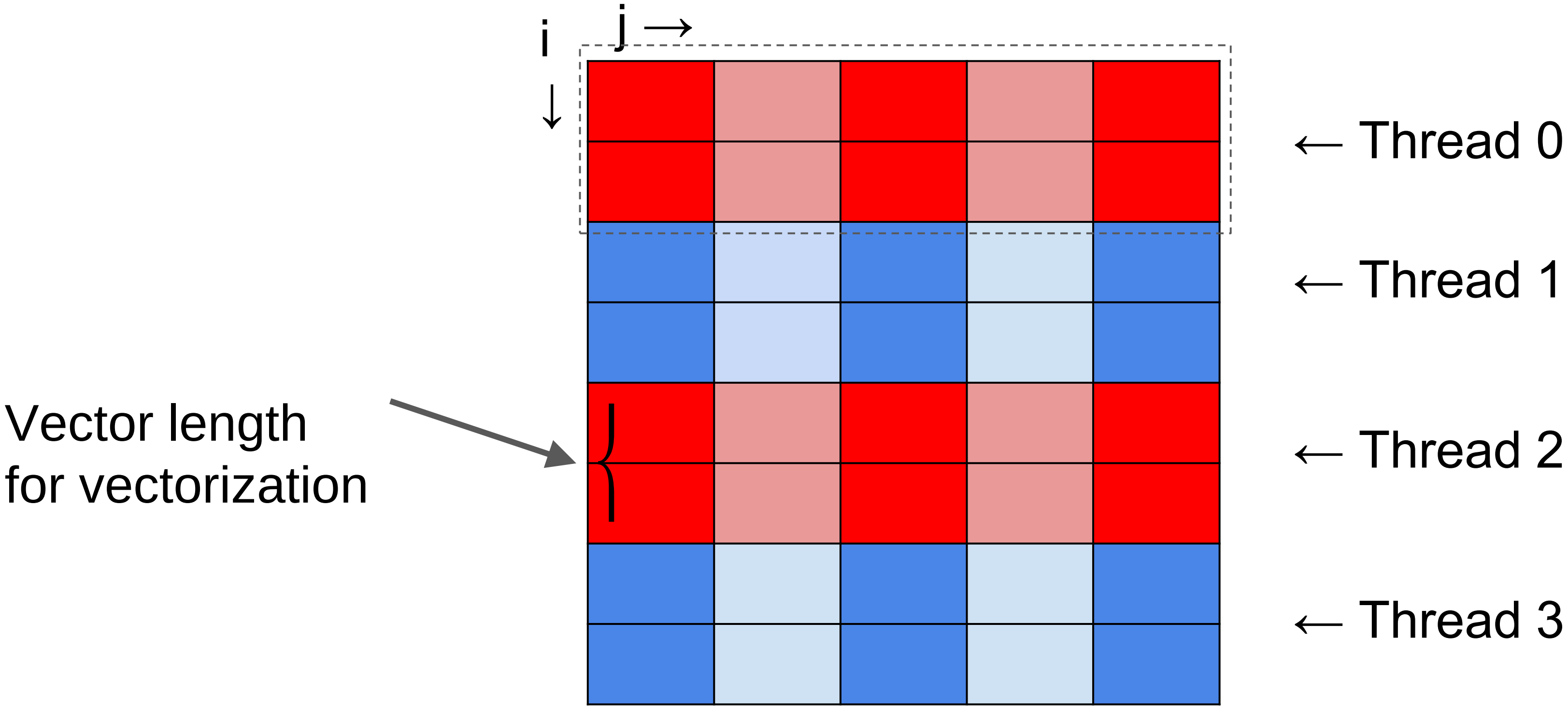
Impact of function calls and conditionals in OMP DO construct

- OMP DO **With** conditionals and subroutine calls: **9.7x speedup** at 64 cores
- OMP DO **Without** conditionals and subroutine calls: **30x speedup** at 64 cores
- Conditionals and function calls hurt scalability
- This behaves better with SIMD in actual WSM6 results:
  - **Loops 12-14: 41x speedup with OMP DO SIMD**

Loop 12 from WSM6

```
do k=kte,kts,-1
do i=its,ite
  ....
  if(t(i,k).gt.t0c)then
    ....
    work2(i,k)=venfac(p(i,k), t(i,k),den(i,k))
    if(qrs(i,k,2).gt.0)then
      ....
      psmlt(i,k)= xka(t(i,k),den(i,k)) ....
      ....
    endif
    if(qrs(i,k,3).gt.0)then
      ....
      pgmlt(i,k)= xka(t(i,k),den(i,k)) ....
      ....
    endif
  endif
enddo
enddo
```

OMP DO, SIMD, DO SIMD



i ↓ j →

← Thread 0

← Thread 1

← Thread 2

← Thread 3

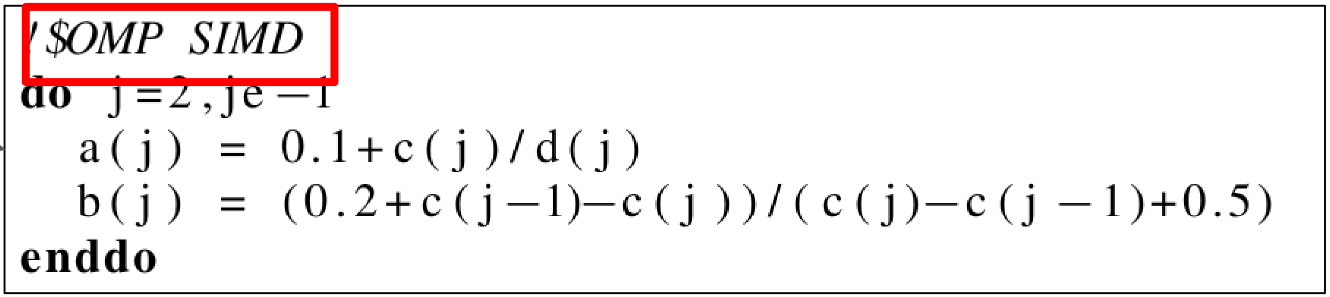
Vector length
for vectorization

	Time	Speedup
OMP DO	2.88	24
Manual binding	0.63	109
OMP DO SIMD	0.61	112
manual binding + OMP SIMD	0.23	229

- OMP SIMD is beneficial for both conditionals and subroutines
  - Works fine with nested subroutines, when OMP DECLARE SIMD is used
- OMP PARALLEL + OMP SIMD (manual binding) was fastest
  - OMP DO SIMD slower than manual binding, but less so with conditionals

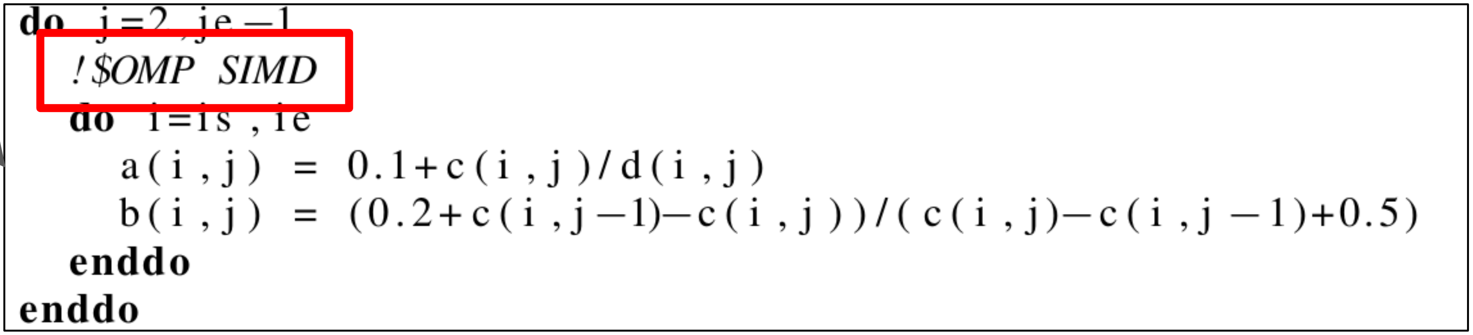
1D vs 2D Standalone Experiments

1D Case



```
!$OMP SIMD
do j=2,je-1
   a(j) = 0.1+c(j)/d(j)
   b(j) = (0.2+c(j-1)-c(j))/(c(j)-c(j-1)+0.5)
enddo
```

2D Case

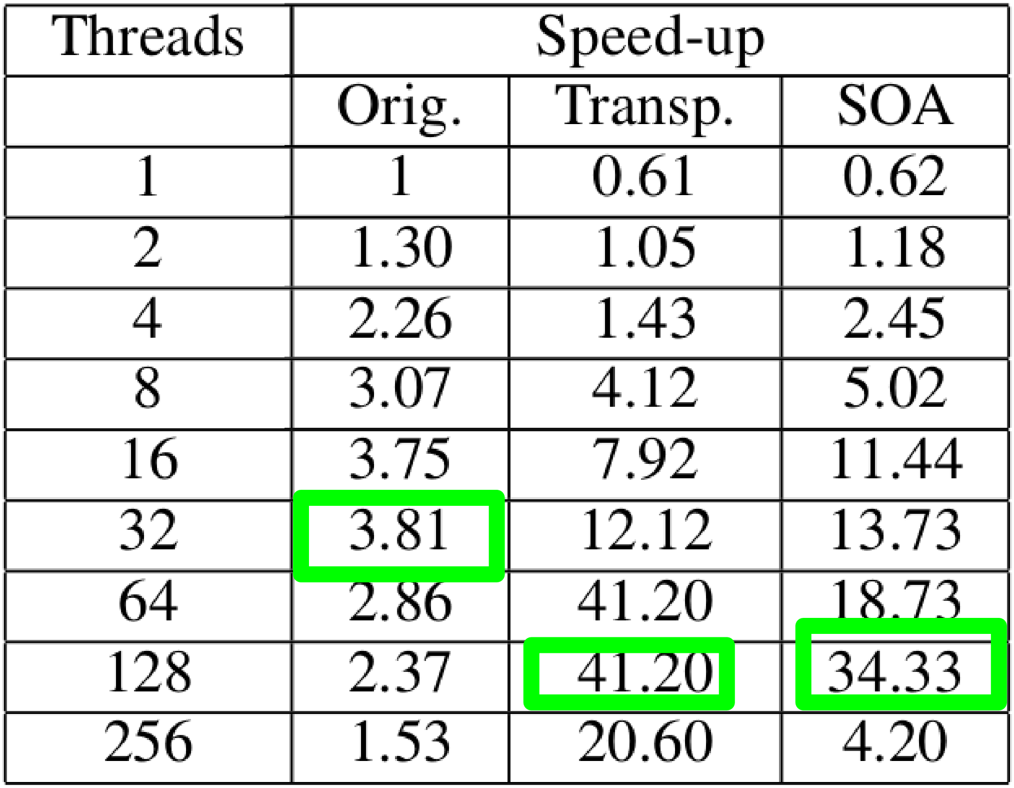


```
do j=2,je-1
  !$OMP SIMD
  do i=is,ie
     a(i,j) = 0.1+c(i,j)/d(i,j)
     b(i,j) = (0.2+c(i,j-1)-c(i,j))/(c(i,j)-c(i,j-1)+0.5)
  enddo
enddo
```

- Computation similar to some of the computation in WSM6.
- 1D Case: SIMD is applied on the j loop. Along j the access pattern are more involved than 2D.
- 2D Case: SIMD is applied on the i loop. No dependencies even in the case of WSM6.

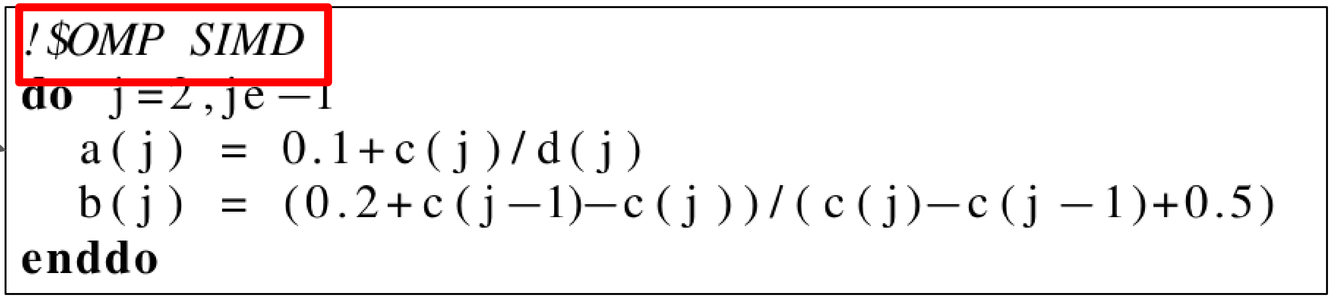
SOA Results 1D

- The table show results from a standalone experiment with 1D arrays as SOA.
- SOA yield good result but not the best.
- The transpose approach performs better
  - uses more thread than original.



Threads	Speed-up		
	Orig.	Transp.	SOA
1	1	0.61	0.62
2	1.30	1.05	1.18
4	2.26	1.43	2.45
8	3.07	4.12	5.02
16	3.75	7.92	11.44
32	3.81	12.12	13.73
64	2.86	41.20	18.73
128	2.37	41.20	34.33
256	1.53	20.60	4.20

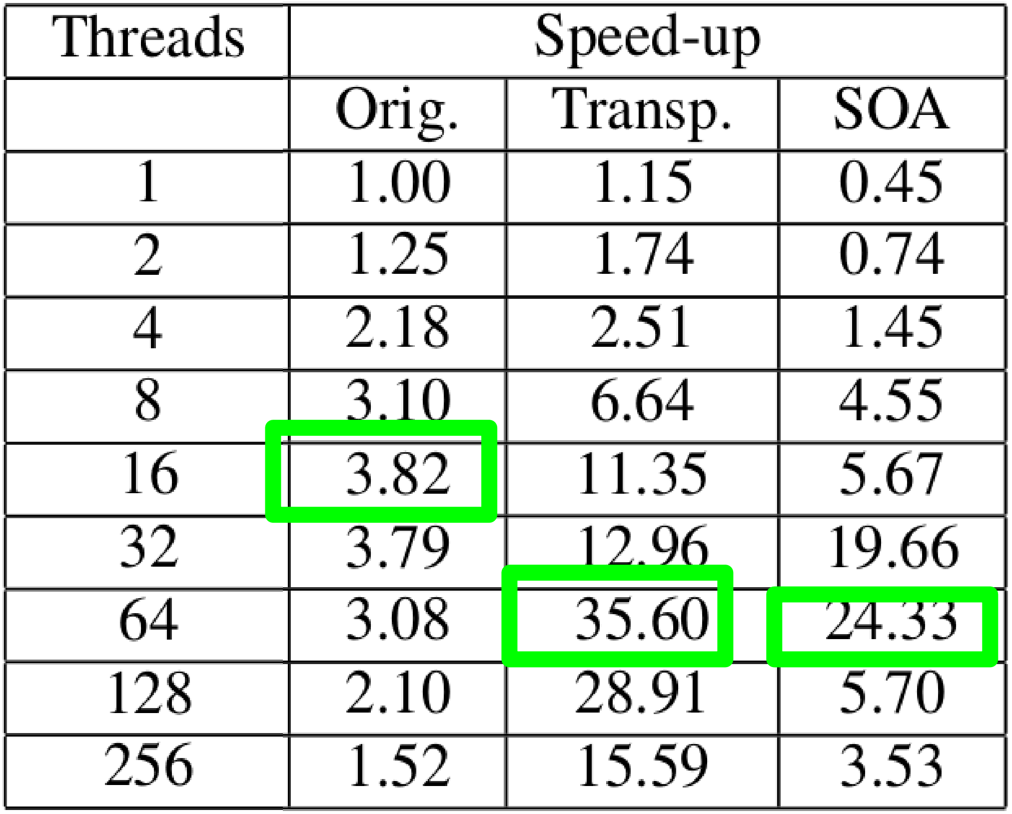
1D Case



```
!$OMP SIMD
do j=2,je-1
   a(j) = 0.1+c(j)/d(j)
   b(j) = (0.2+c(j-1)-c(j))/(c(j)-c(j-1)+0.5)
enddo
```

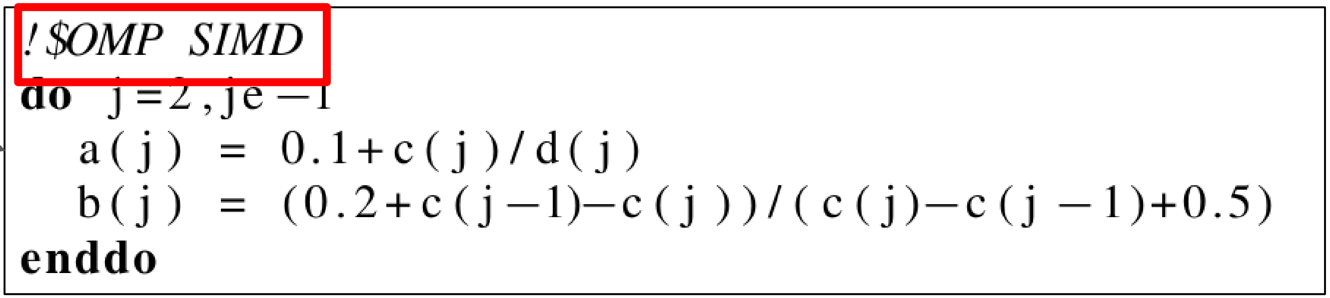
SOA Results 1D with Large Arrays

- The Table shows result a standalone experiment with increase size of the 1D arrays 16x previous experiment with 1D arrays.
- Transpose still outperforms SOA.
- This indicate that the structure of SOA plays a role in performance.



Threads	Speed-up		
	Orig.	Transp.	SOA
1	1.00	1.15	0.45
2	1.25	1.74	0.74
4	2.18	2.51	1.45
8	3.10	6.64	4.55
16	3.82	11.35	5.67
32	3.79	12.96	19.66
64	3.08	35.60	24.33
128	2.10	28.91	5.70
256	1.52	15.59	3.53

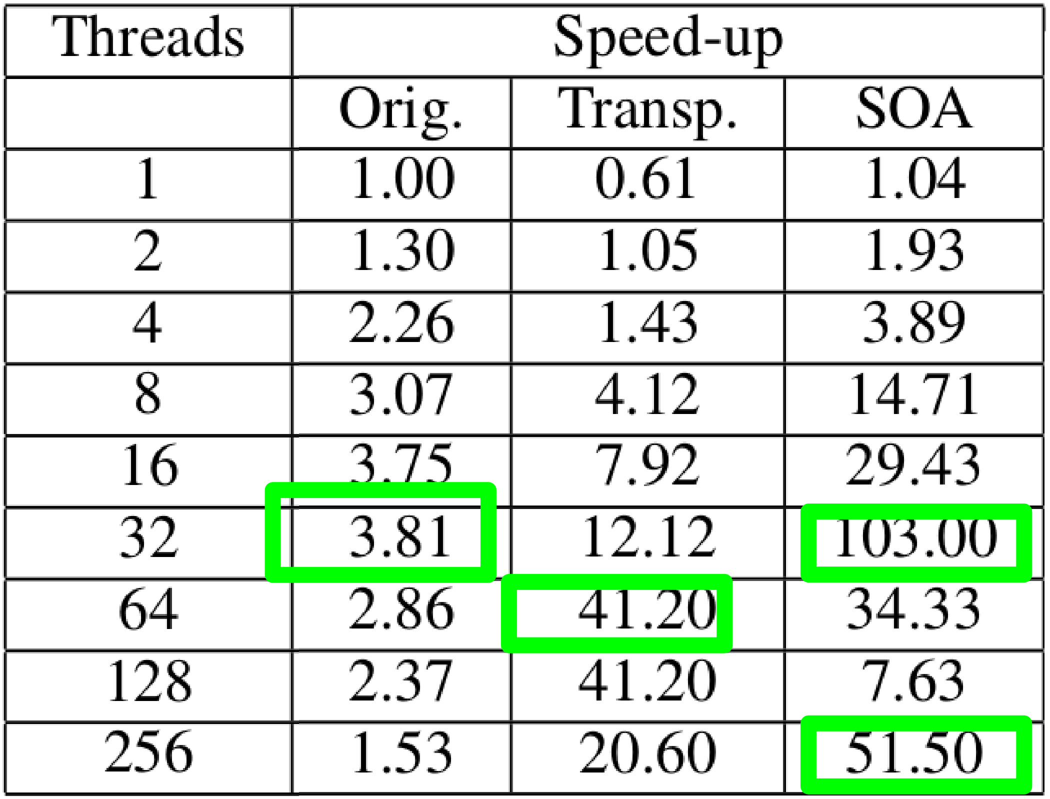
1D Case



```
!$OMP SIMD
do  j=2,je−1
   a(j)  =  0.1+c(j)/d(j)
   b(j)  =  (0.2+c(j−1)−c(j))/(c(j)−c(j −1)+0.5)
enddo
```

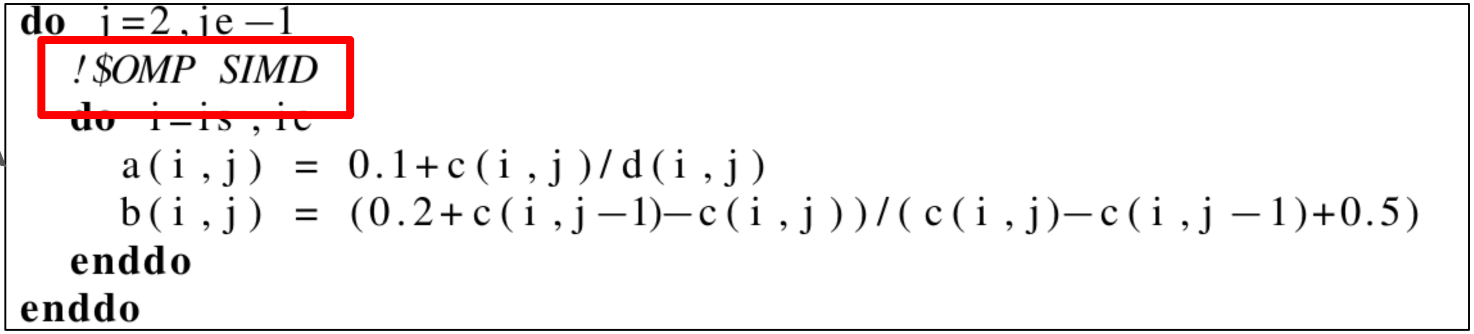
SOA Results 2D with Large Arrays

- The table shows results from a standalone experiment with 1D arrays as SOA.
- SOA outperforms here because we are able to better leverage vector units.



Threads	Speed-up		
	Orig.	Transp.	SOA
1	1.00	0.61	1.04
2	1.30	1.05	1.93
4	2.26	1.43	3.89
8	3.07	4.12	14.71
16	3.75	7.92	29.43
32	3.81	12.12	103.00
64	2.86	41.20	34.33
128	2.37	41.20	7.63
256	1.53	20.60	51.50

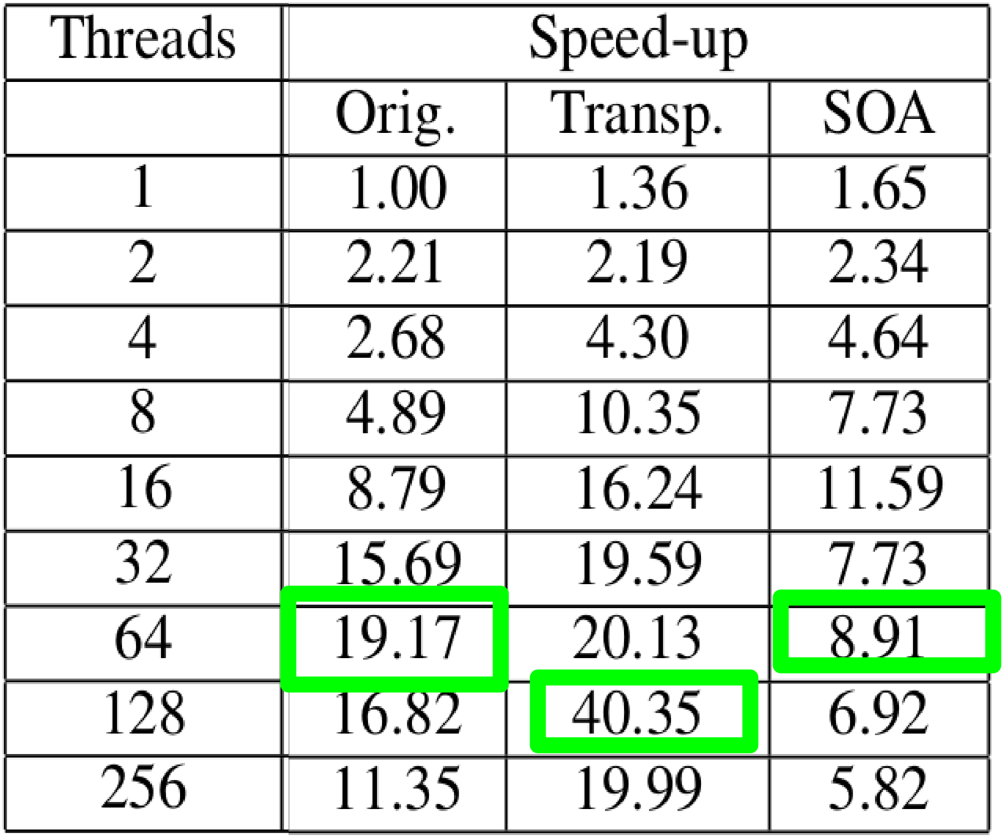
2D Case



```
do  j=2,je−1
  !$OMP  SIMD
  do  i=is,ie
     a(i,j)  =  0.1+c(i,j)/d(i,j)
     b(i,j)  =  (0.2+c(i,j−1)−c(i,j))/(c(i,j)−c(i,j −1)+0.5)
  enddo
enddo
```

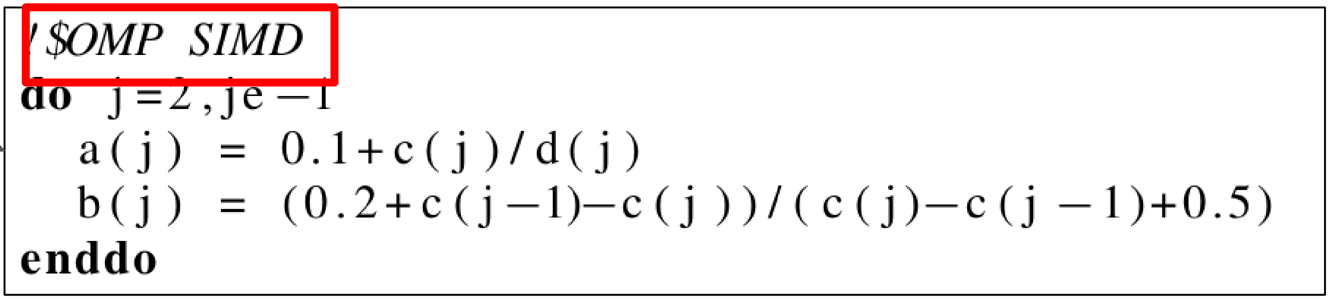
SOA Results 2D

- The Table shows result a standalone experiment with increase size of the 1D arrays 16x previous experiment with 2D arrays
- Transpose outperform the others
- SOA performs poorly because of cache missing. The size of arrays in the SOA doe not fit in cache. This memory access penalty.
- The key to using SOA are structure and size.



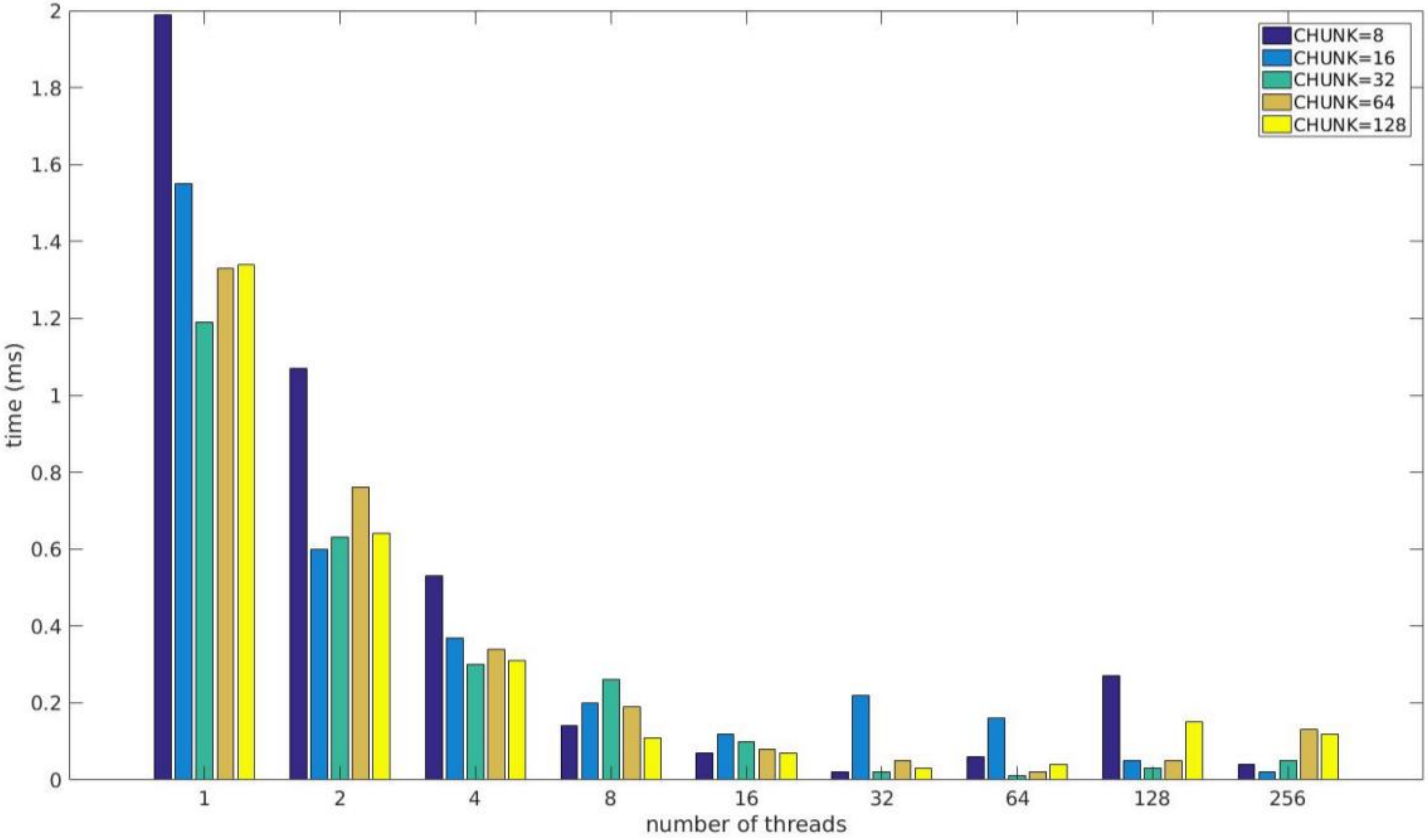
Threads	Speed-up		
	Orig.	Transp.	SOA
1	1.00	1.36	1.65
2	2.21	2.19	2.34
4	2.68	4.30	4.64
8	4.89	10.35	7.73
16	8.79	16.24	11.59
32	15.69	19.59	7.73
64	19.17	20.13	8.91
128	16.82	40.35	6.92
256	11.35	19.99	5.82

1D Case



```
!$OMP  SIMD
do  j=2,je−1
   a(j)  =  0.1+c(j)/d(j)
   b(j)  =  (0.2+c(j−1)−c(j))/(c(j)−c(j −1)+0.5)
enddo
```

Chunk sizes



- Chunk size chunk = 32 provides better results.
- The chunk size have to be large enough to provide sufficient work to minimize the overhead related to thread usage.

WSM6 Optimization Effort

- **“Low-level” OpenMP approach based on standalone experiments:**
  - Initialize all threads in an OMP PARALLEL section in wsm6init()

  In main WSM6 body wsm62d()
  - OMP PARALLEL, and DO SIMD directives
  - Merged loops to hide latency, removed redundant assignments

- **Compare KNL and Haswell**
  - Haswell = Intel Xeon 2.5 GHz, 72 cores with 2 threads per core
  - Haswell has a higher clock frequency than KNL
  - KNL has high higher bandwidth and larger L3/MCDRAM

- **“Low level and high-level” Optimization with OpenMP and SOA**
  - SOA at a higher level in call stack
  - SIMD at the lower level for vectorization
  - Merged loops, and removed keywords (exit, cycle goto)

Speedup per Loop

- slope_wsm6 has different speed-ups for the same routine. Thread invocation time and memory access impact runtime.
- Loop 22 and 23 are simple with no complex logic.
- Overall, good scalability to 64 cores on KNL.
- Includes loop 12 (from standalone example). OMP SIMD enables 41x speedup, including nested conditionals, subroutines
- Final copy of the result arrays shows significant thrashing

	Speedup	
Loop	**KNL**	**Haswell**
1	14	4.9
2-4	19	4.5
5-6	36	10
7	15	4.2
slope_wsm6	**55**	**8.7**
8	14	3.7
9-11	6.9	3.7
12-14	**41**	**3.0**
15-17	**74**	**19**
18-19	3.5	4.2
slope_wsm6	**45**	**5.6**
20-21	34	2.8
22	**98**	**13**
23	**100**	**5.5**
24-26	57	12
27	**.77**	**0.80**

Removal of Fortran Keywords

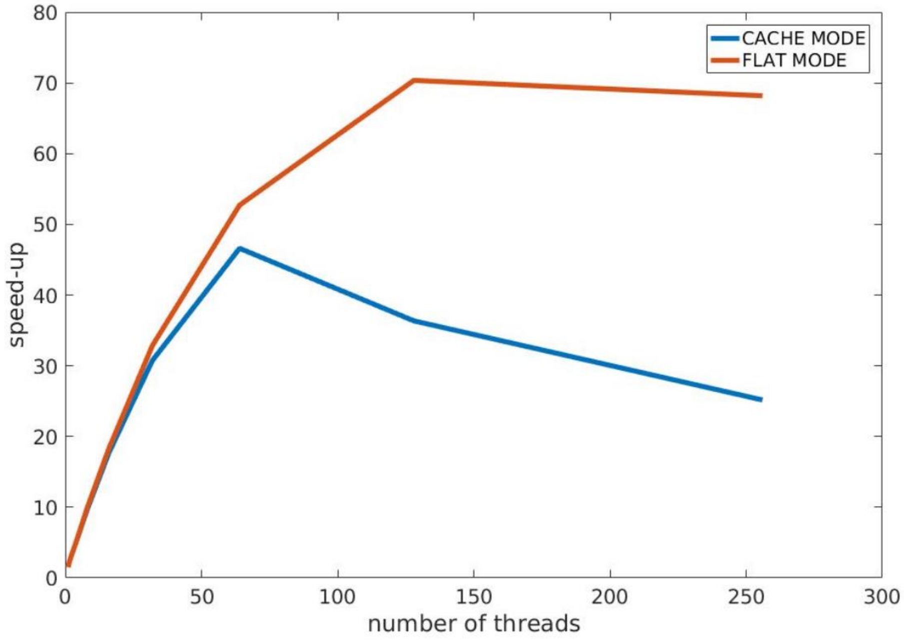
```
sum_precip: do k=1,km
        if(condition1)
            update precip
            cycle sum_precip
        elseif(condition2)
            update precip
            exit sum_precip
        end if
        exit sum_precip
end do sum_precip
```

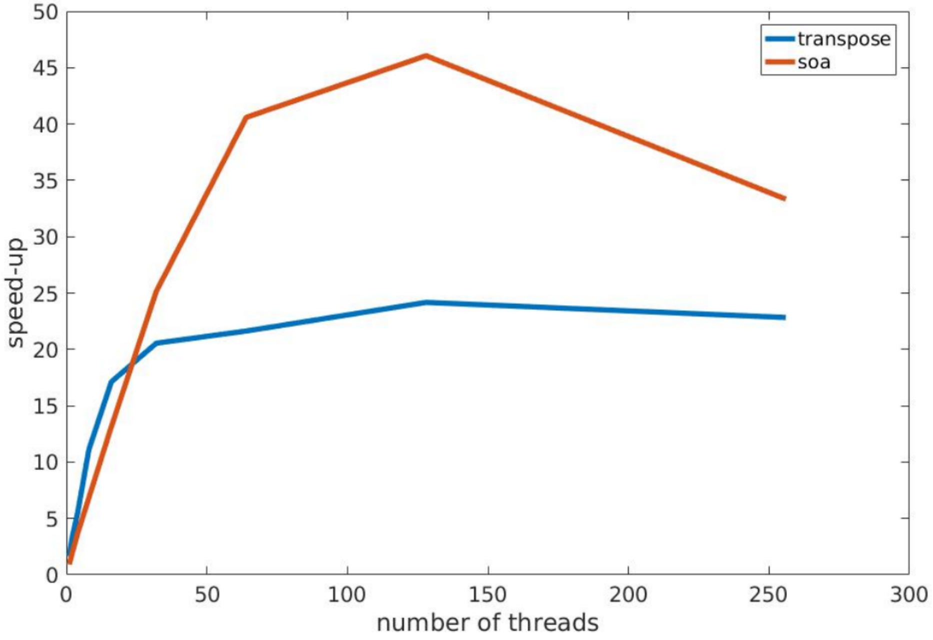
```
sum_mask = 1

sum_precip: do k=1,km
        if(condition1 .and sum_mask)
            update precip
        elseif(condition2 .and. sum_mask)
            update precip
            sum_mask = 0
        else
        sum_mask = 0
end do sum_precip
```

```
100     continue
              .
              .
              .
        if(n .le. inter)
        goto 100
        end if
```

```
Do n=1,iter
      .
      .
      .
end do
```

WSM6 Results





- The plot to the left clearly show that using flat mode outperform the cache mode.
- These results are similar to the what we observe in the community.
- **70x Speed**-up on WSM6

- The plot to the right clearly show that SOA outperform the Transpose approach.
- The Rain routines are part of the WSM6 Module itself.

Conclusion

- **“Low-level” OpenMP with OMP DO SIMD achieves ~50x speedup over the 25 parallelized loops of WSM6.**
    - Restructure non-trivial logic with nested conditionals, subroutine calls, and unaligned memory access to enable performance
    - Vtune suggests 5.6% of peak in these sections
    - Including bottlenecks, WSM6 within NEPTUNE is **3x** faster than serial on KNL

- **“High-level and low-level”**
    - Restructure of non-trivial loops
    - SOA at top level call in WSM6 and SIMD at the lower level
    - This approach led to **70x** on WSM6

Future Work

- **Apply these methodologies to GFS operational physics in NEPTUNE**
- **Investigate the impact from translation between dynamics and physics**
- **Investigate behavior and scalability on large system i.e OpenMP + MPI**
- **Investigate other optimization Strategies**
    - lightweight runtime system for weather physics codes
    - Other approaches for data reorganizations

Acknowledgements

DOD PETTT sponsors:

- DOD HPCMP PP-KY07-CWO-001-P3
- Alex Reinecke, Kevin Viner (NRL), Rajiv Bendale, Hugh Thornburg (Engility)

Additional consulting and hardware support:

- John Michelakes (UCAR), Lars Koesterke (TACC), and Intel Parallel Computing Centers program
- This work continues under IPCC

THANK YOU

Questions?