Performance Optimization Techniques for Accelerating WRF Physics Codes on Micro-architectures

Presented by:

T.A.J.Ouermi, Mike Kirby, Martin Berzins

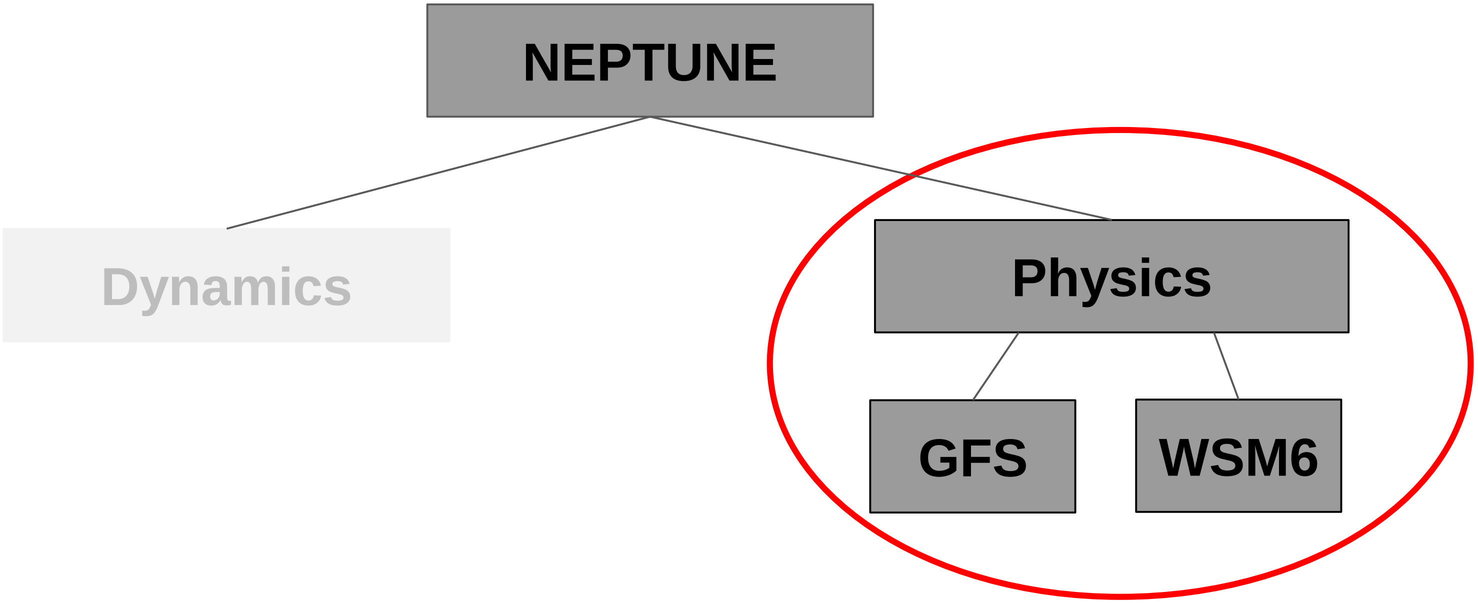
Motivation

- Faster weather physics for operational Navy Environmental Prediction sysTem Utilizing the NUMA corE (NEPTUNE)

- Target architectures: Micro-acrchitectures
  - Intel Knights Landing (KNL),
  - Intel Haswell

- Portability with OpenMP

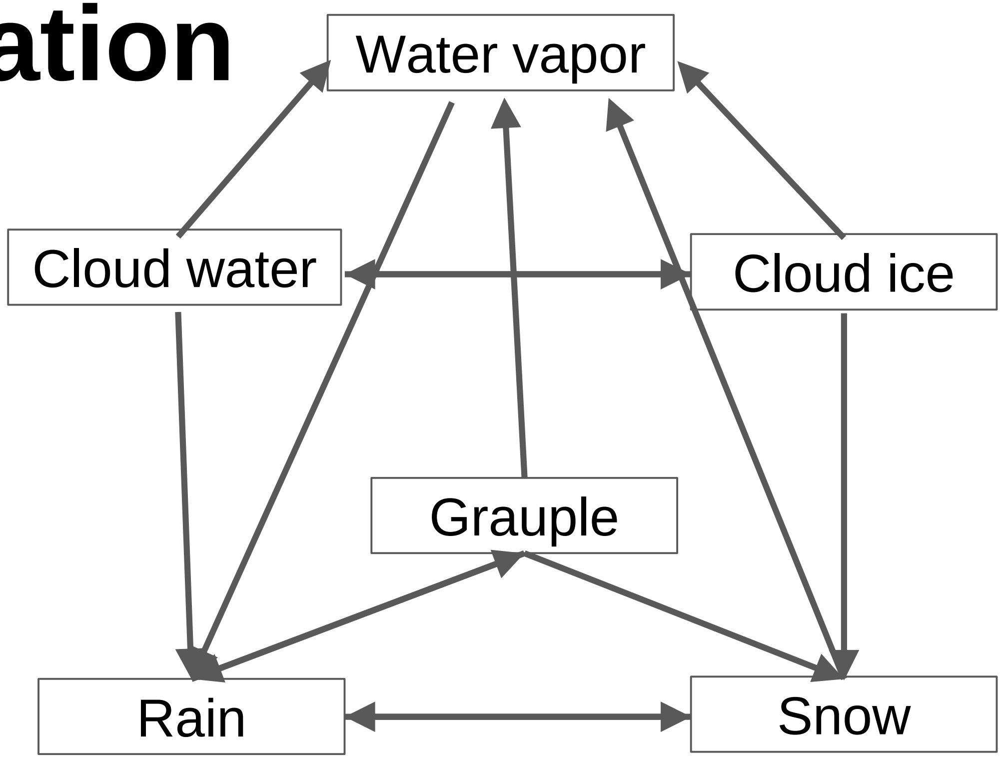
NEPTUNE



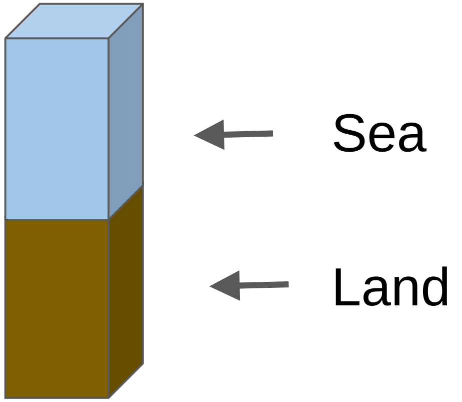
Physics Optimization Challenges

- ➢Large loops with many conditional not favorable for parallelism.

- ➢Difficult to optimize with transition between many regimes.

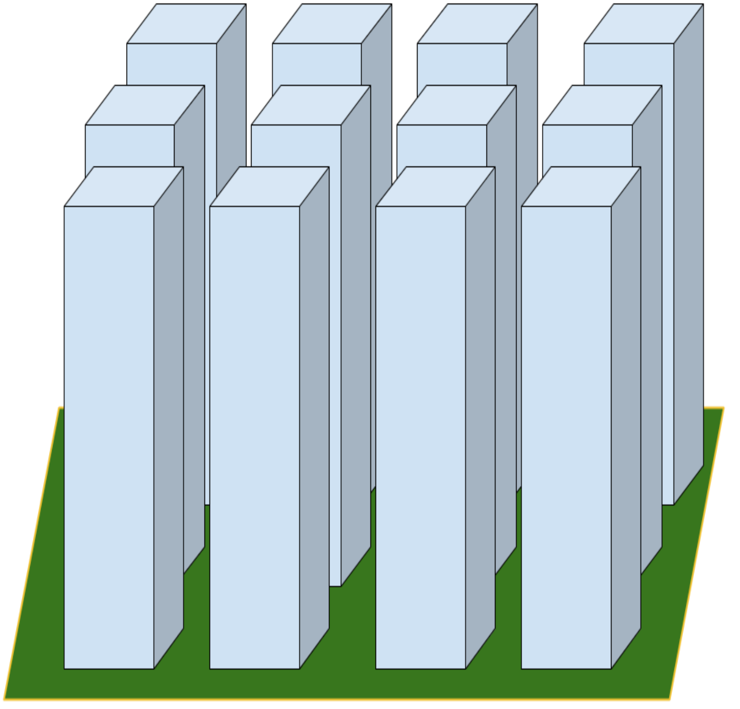


<u>WRF single-moment 6-class Microphysics Scheme (WSM6)</u>



Vertical Physics Advantage

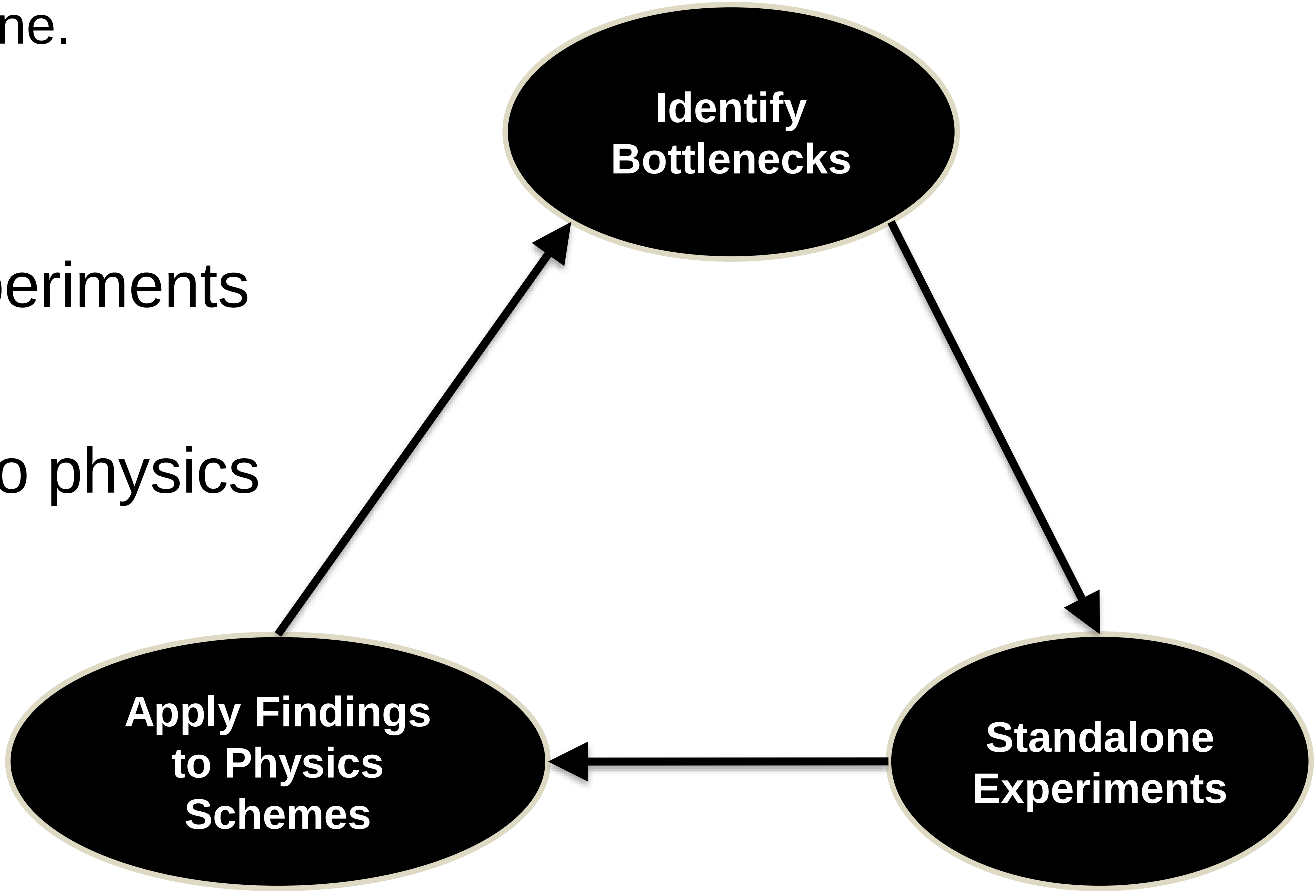
- ➢Dependencies within columns.
- ➢No dependencies between columns.



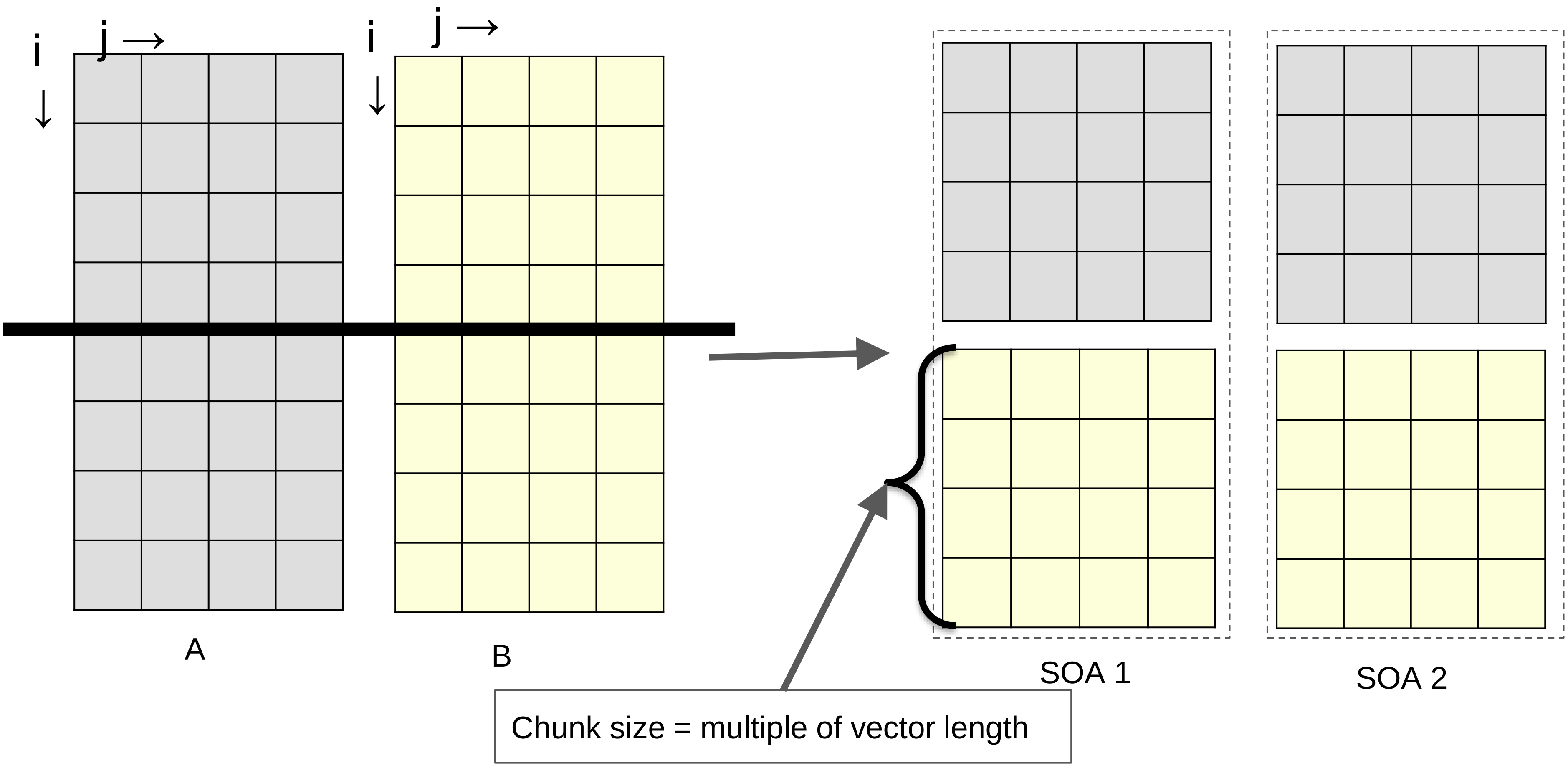
Vertical Physics representation

Methodology

- Identify bottlenecks
  - Wall Clock, Vtune.
  - Advisor, optrpt.
- Standalone experiments
- Apply findings to physics



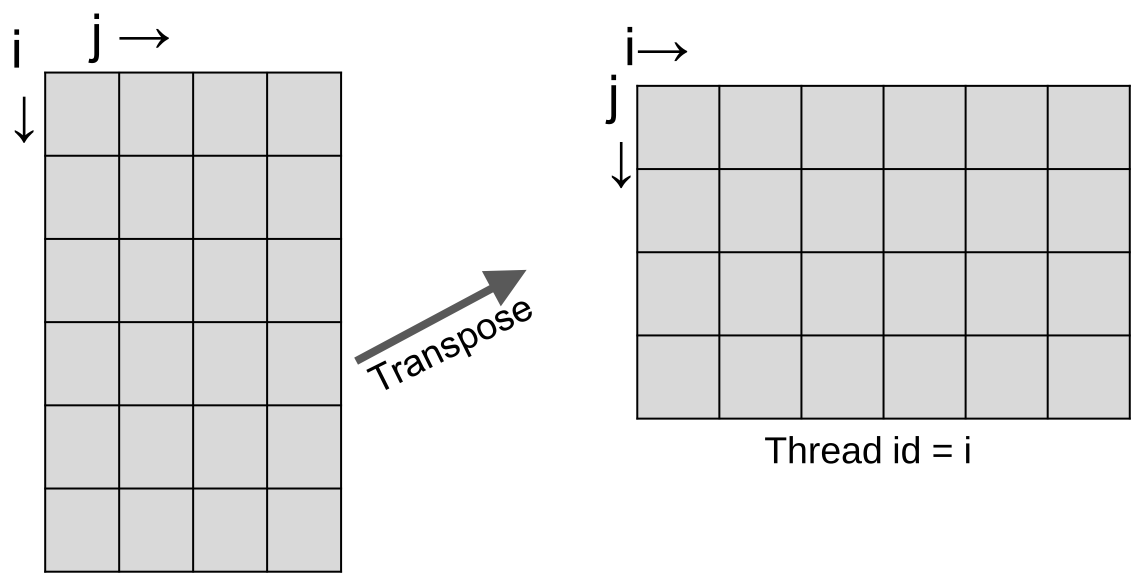
Structures of Arrays (SOA)



Transpose

```
!$OMP DO
do j=1,km
do i=1,im
 a(i,k)=b(i,k)-c(i,k)
end do
end do
```

```
!$OMP DO
do i=1,im
do j=1,km
 a(k,i)=b(k,i)-c(k,i)
end do
end do
```



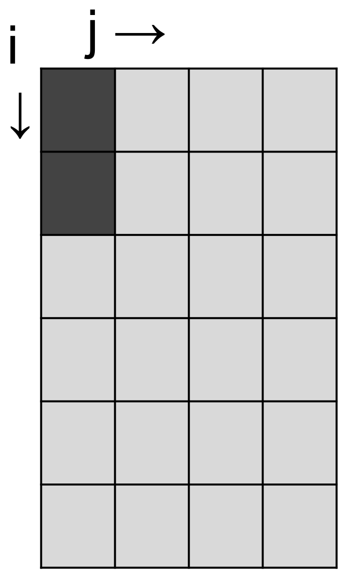
Thread id = j

Thread id = i

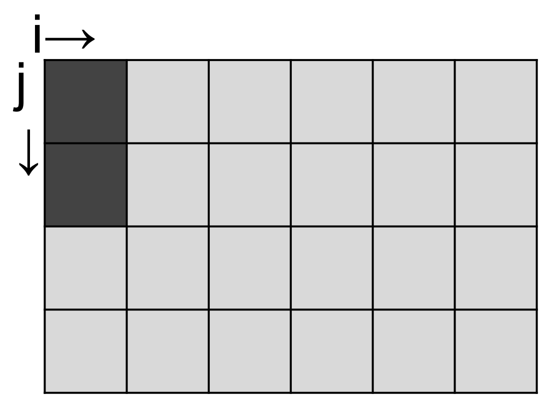
Vectorization

```
!$OMP DO
do j=1,km
!$OMP SIMD
do i=1,im
 a(i,k)=b(i,k)-c(i,k)
end do
end do
```

```
!$OMP DO
do i=1,im
!$OMP SIMD
do j=1,km
 a(k,i)=b(k,i)-c(k,i)
end do
end do
```



Thread id = j



Thread id = i

Architectures

Intel Knights Landing(KNL)	Intel Xeon CPU E-7-8890 (Haswell)
1 socket	4 sockets
64 cores	18 cores per socket
4 threads per core	2 threads per core
2VPU per core (AVX-512)	VPU (AVX-2)
Clock of 1.5 Ghz	Clock of 2.5 Ghz
L1 32k	L1 32k
L2 1024k	L2 256K
MCDRAM 16GB	L3 46MB

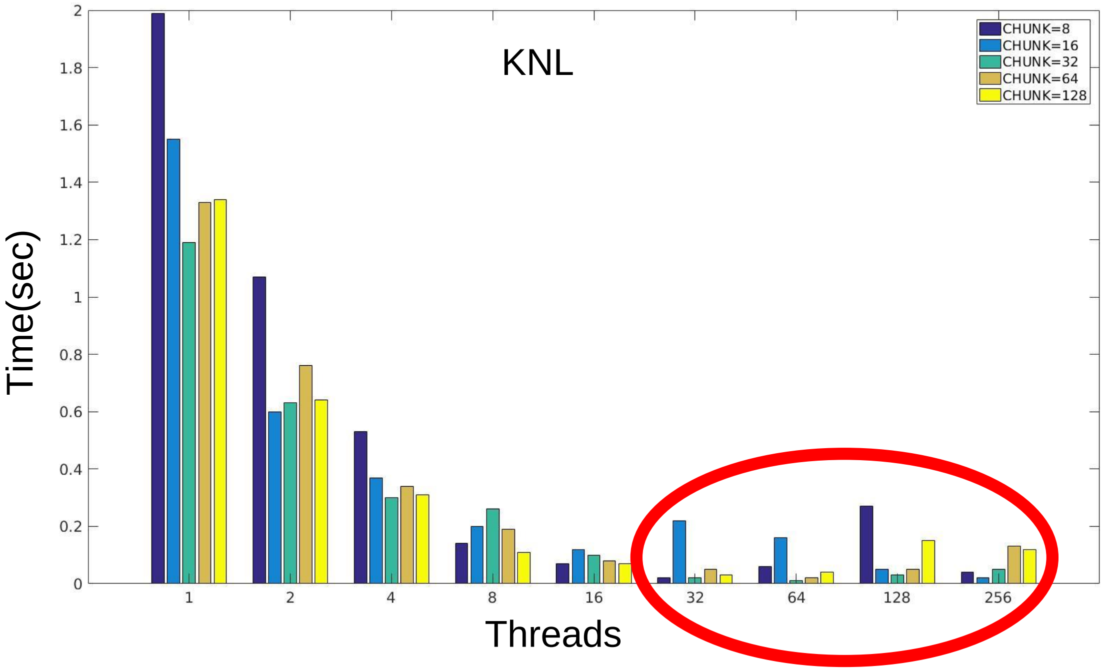
Experiments

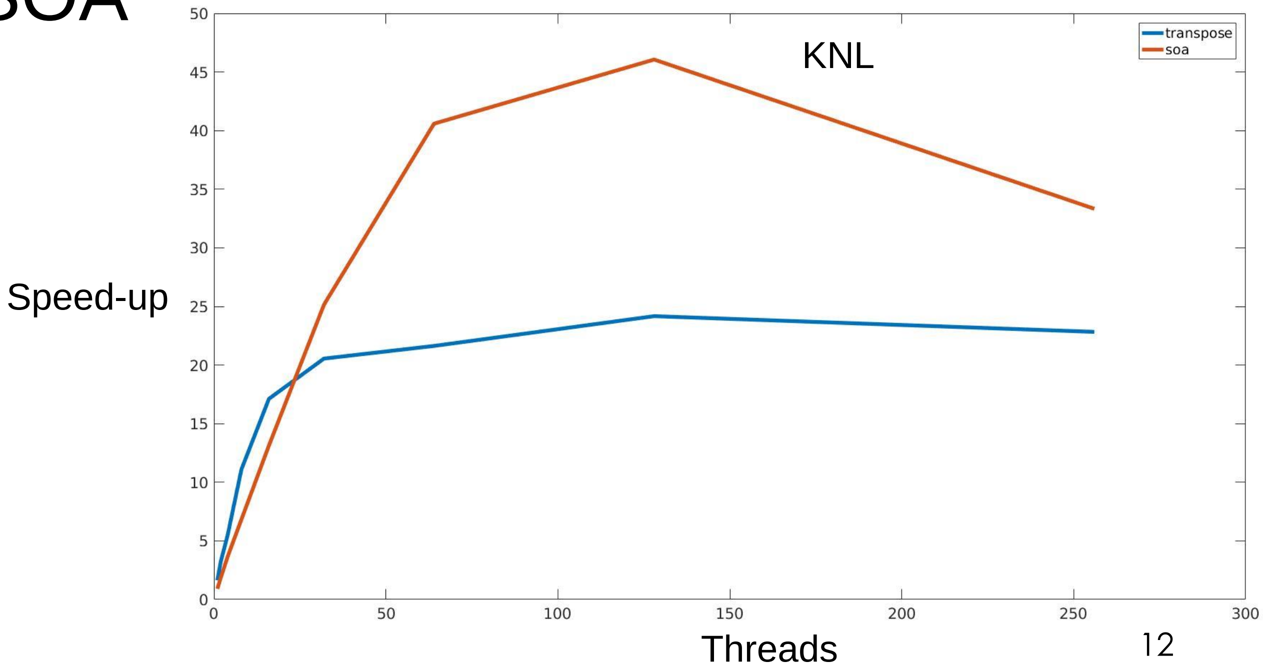
- Transpose data.
- Thread-local SOA with different chunk sizes.
- Scheduling: dynamics vs static.
- Thread configurations.
- Simplify complex code by removing conditionals and nested code for vectorization.

Transpose vs SOA

➢Identify suitable chunk size.

➢Thread-local SOA **2x** faster than transpose.

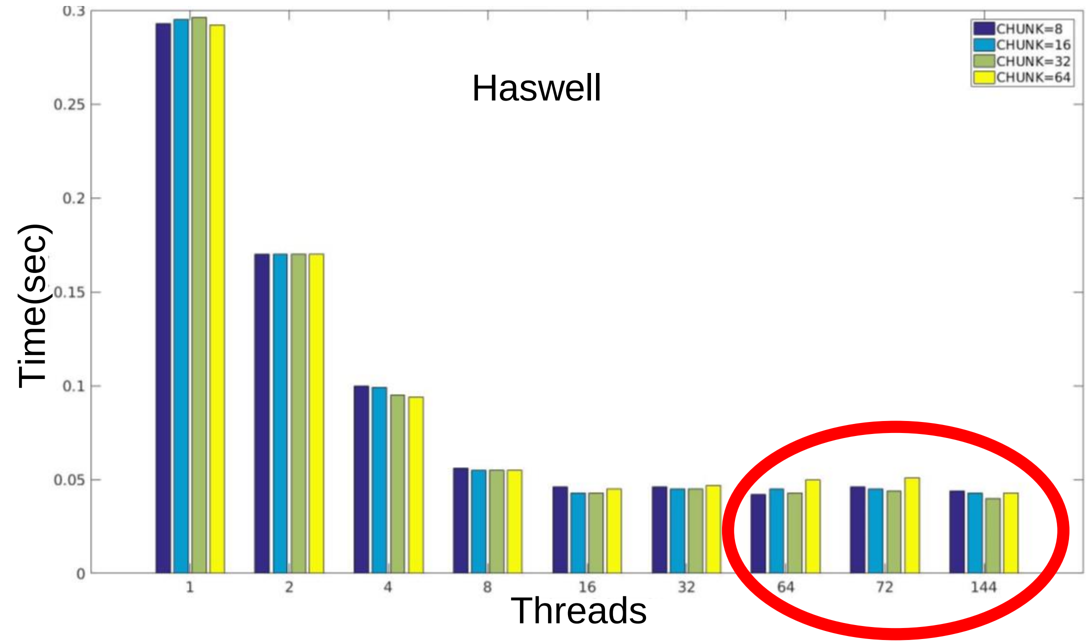


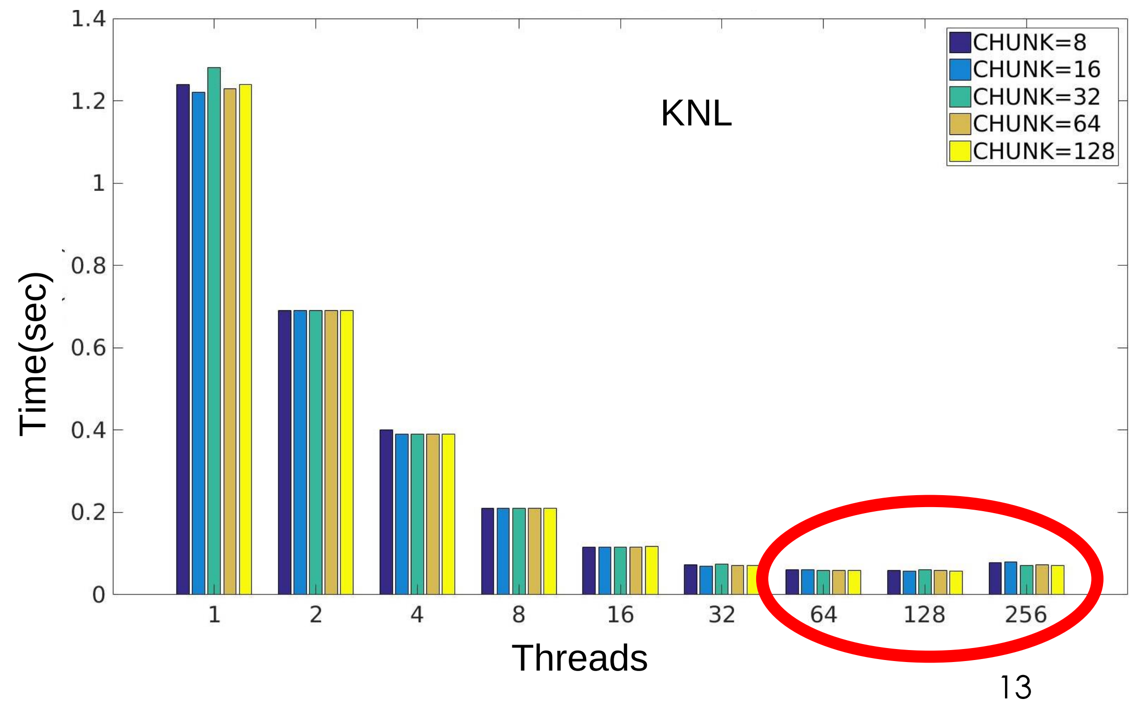


WSM6 Chunk Size

➢Chunk = 32 for haswell.

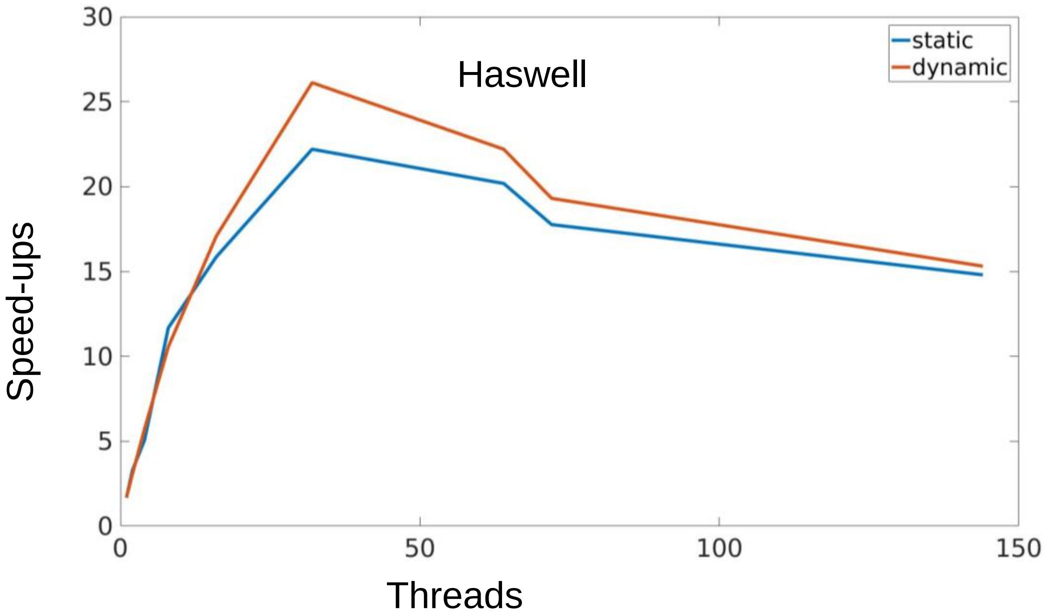
➢Chunk = 64 for KNL.

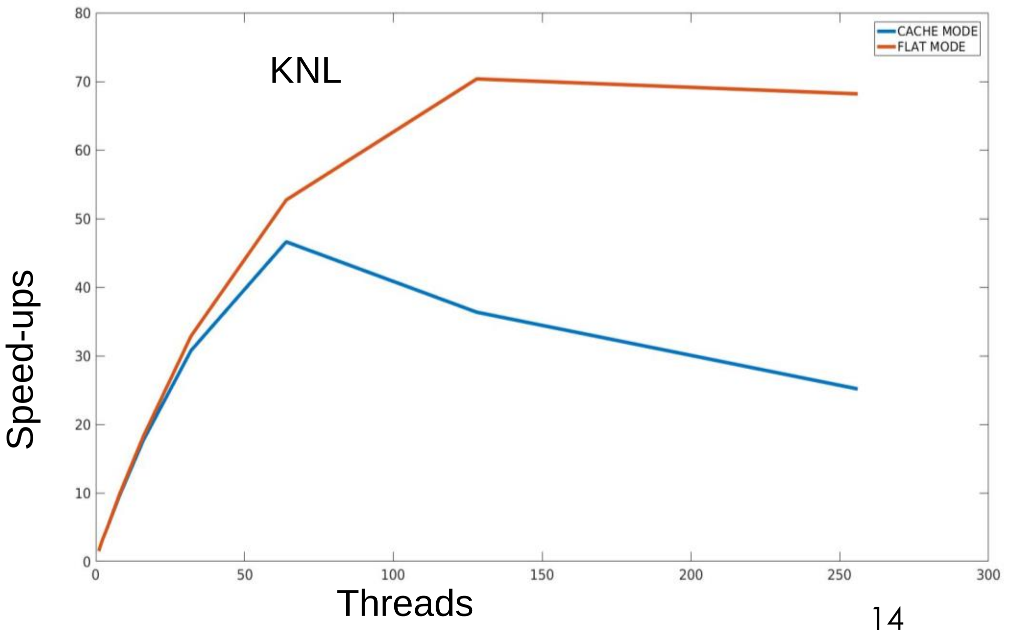




WSM6 Results

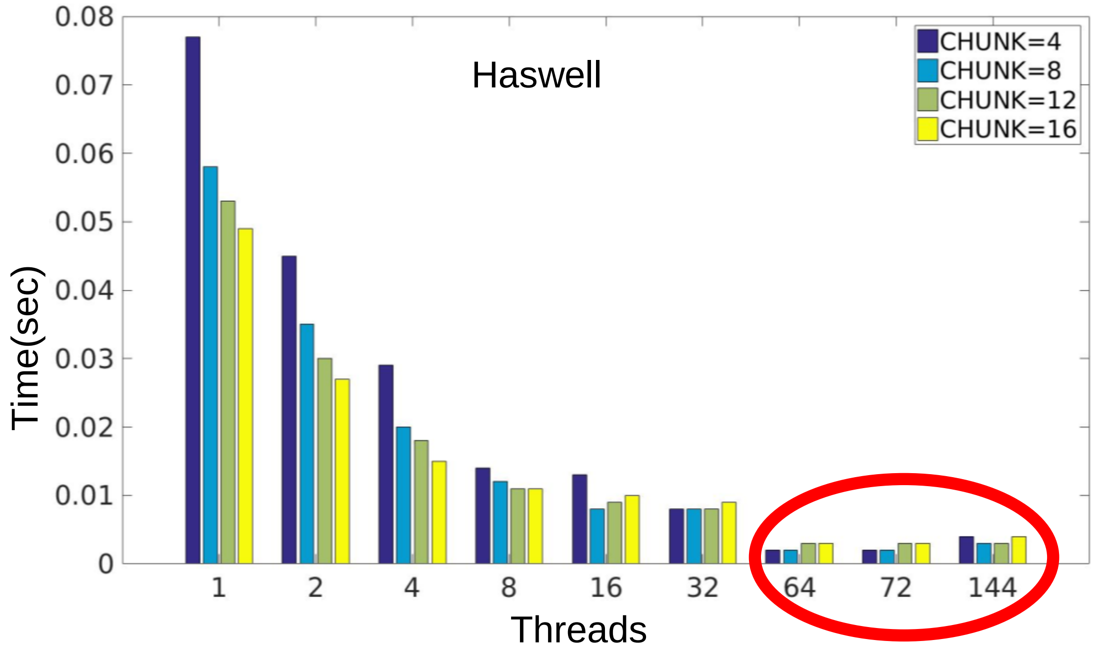
- ➢Dynamic scheduling better in both cases.
- ➢**70x** on KNL and **26x** on Haswell.
- ➢FLAT better results than CACHE on KNL.
- ➢Haswell peak at 32 threads and KNL at 64 threads

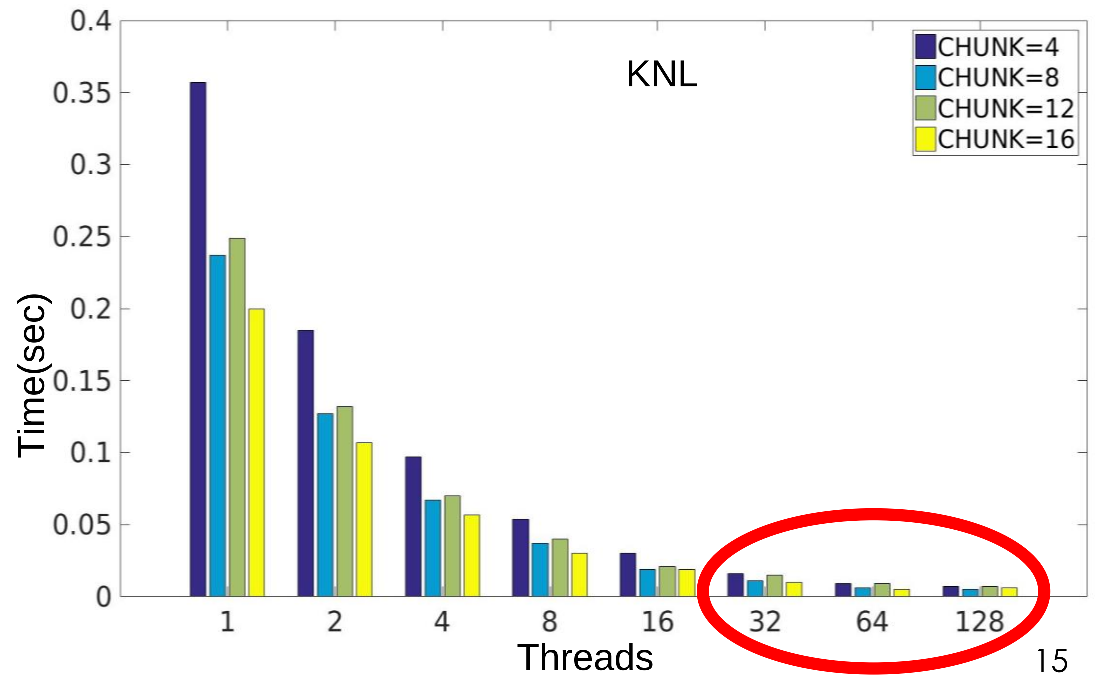




GFS Phys. Chunk Size

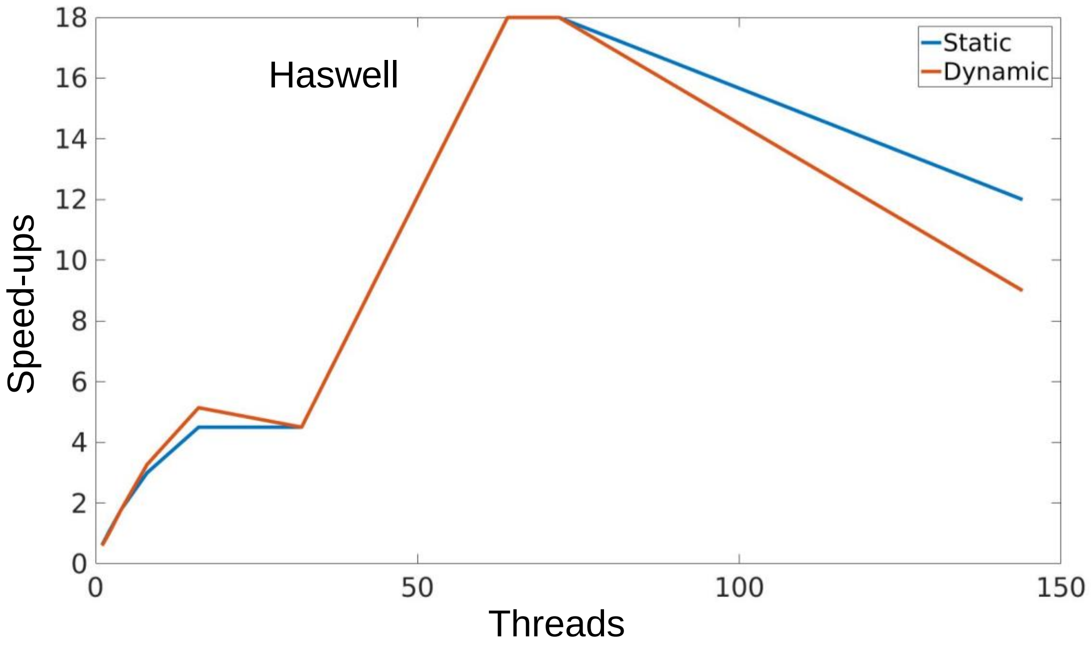
- ➢Chunk = 8, 12 for haswell.
- ➢Chunk = 16,8 for KNL.

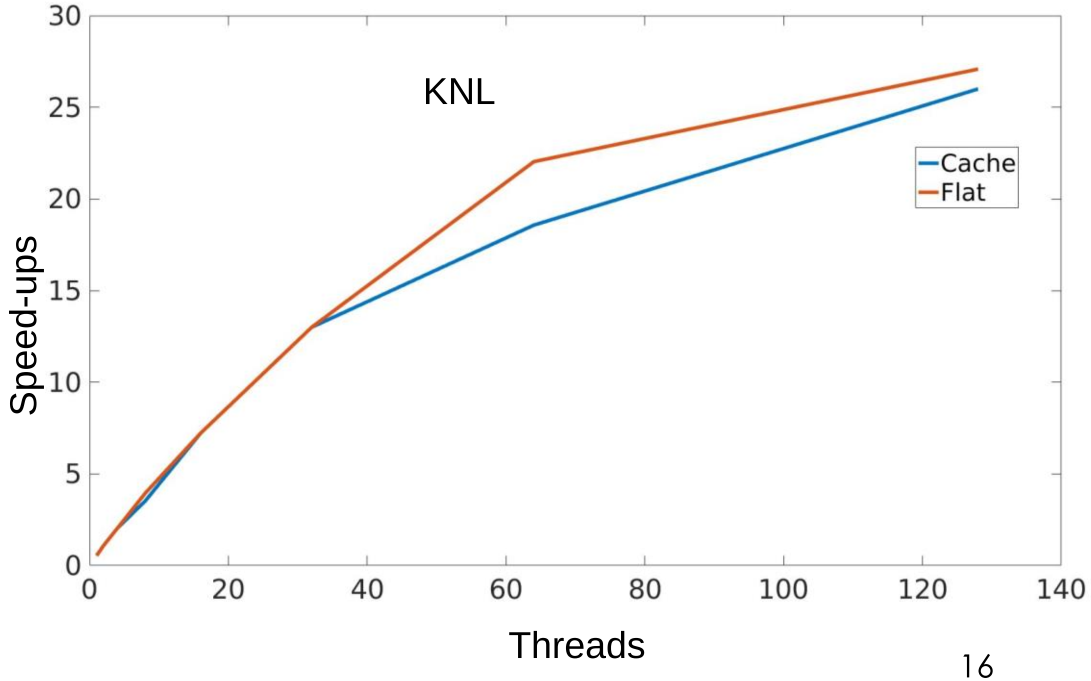




GFS Phys. Results

- ➢Scale up to **18x** with 72 threads on Haswell.
- ➢Scale up to **27x** with 128 threads on KNL.
- ➢Static scheduling performs better than dynamics.

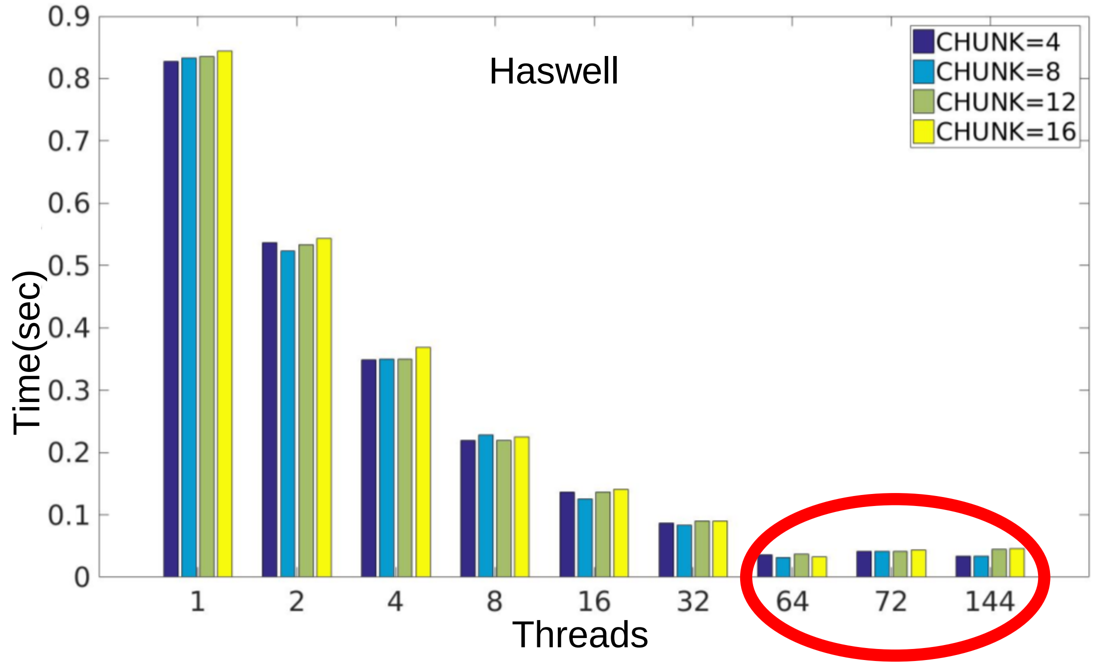


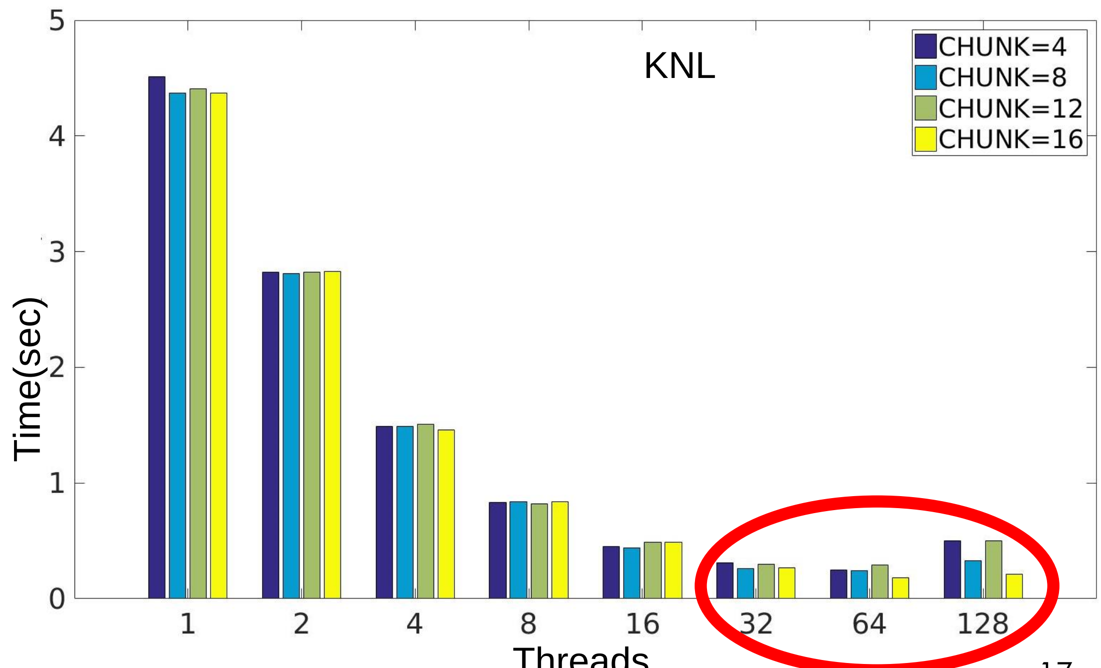


GFS Rad. Chunks

➢Chunk = 8, 12 for haswell.

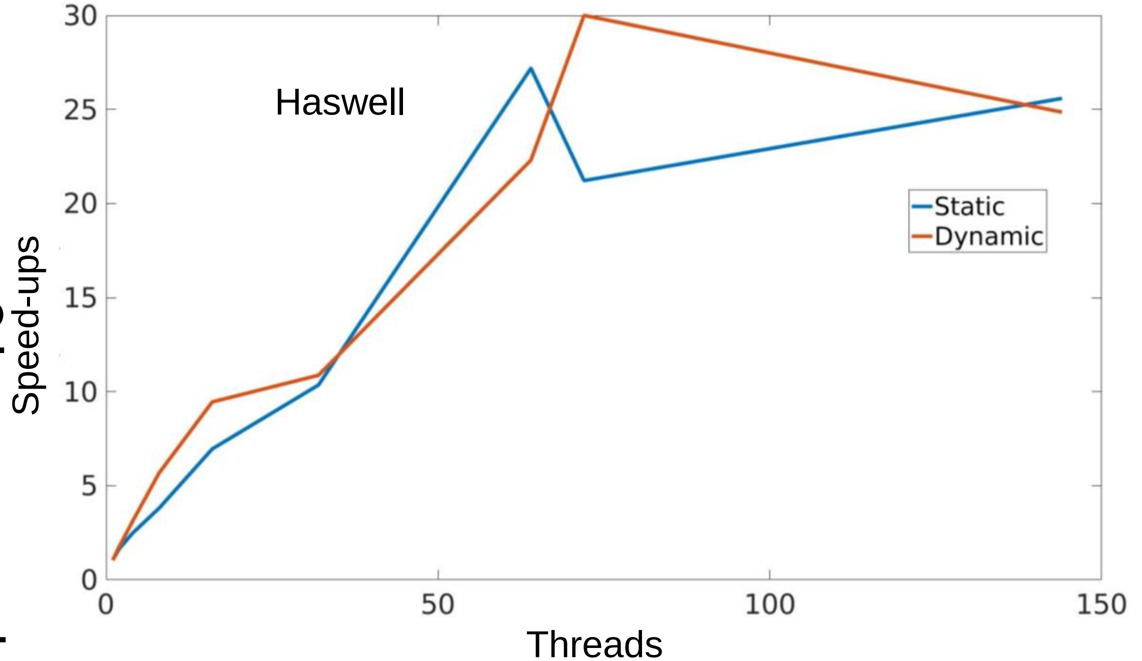
➢Chunk = 16,8 for KNL.

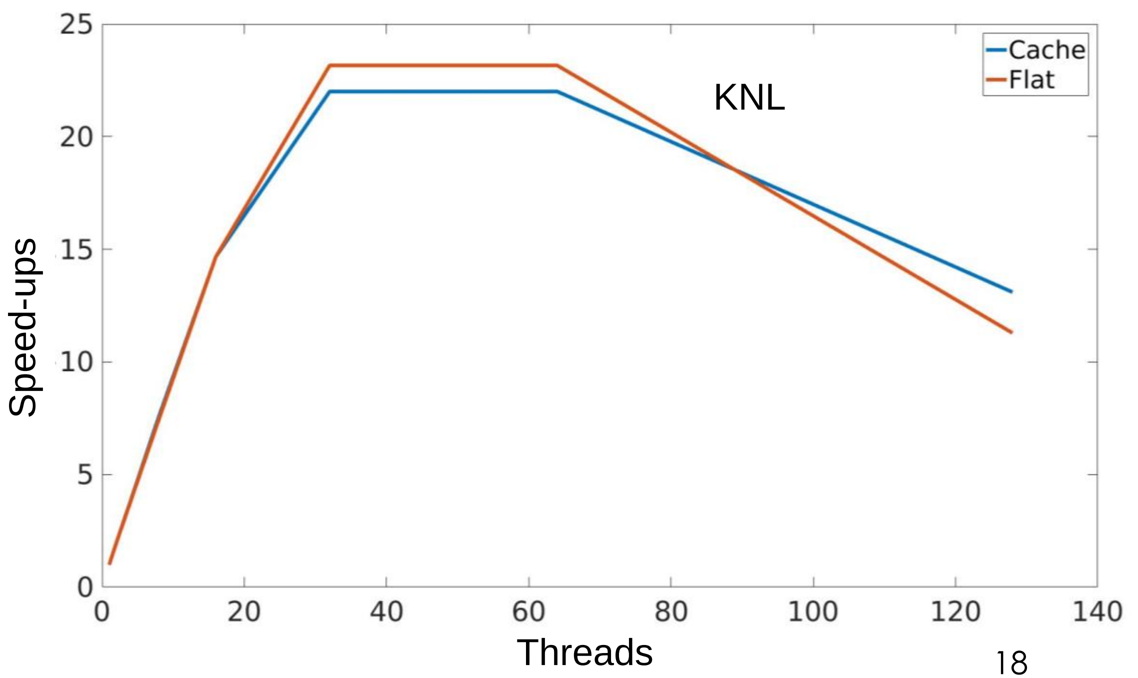




GFS Rad. Results

- ➢Scale up to **30x** 72 threads on Haswell.
- ➢Scale up to **23x** 64 threads on KNL.
- ➢Dynamics scheduling performs better.





Discussion

physics schemes		WSM6	GFS physics	GFS radiation
KNL	best time (ms)	23.0	4.8	190.0
	speed-up	70	27	23
	threads	64	128	64
	configuration	dynamic+flat	static+flat	dynamic+flat
Haswell	best time (ms)	17.0	2.0	29.0
	speed-up	26	18	30
	threads	32	72	72
	configuration	dynamic	static	dynamic

- Better runtimes with haswell because more cores and faster clock.

Discussion

physics schemes		WSM6	GFS physics	GFS radiation
KNL	best time (ms)	23.0	4.8	190.0
	speed-up	70	27	23
	threads	64	128	64
	configuration	dynamic+flat	static+flat	dynamic+flat
Haswell	best time (ms)	17.0	2.0	29.0
	speed-up	26	18	30
	threads	32	72	72
	configuration	dynamic	static	dynamic

- Better runtimes with haswell because more cores and faster clock.

- Better speed-ups with KNL because better utilization of threads.

Conclusion and Future Work

- Code modification to use thread-local SOA.
- Identifying the appropriate chunk size to maximize work per thread and locality.

- Future Directions
  - Better understanding of how to improve peak performance.
  - Study of MPI+OpenMP on larger test cases in context of NEPTUNE.

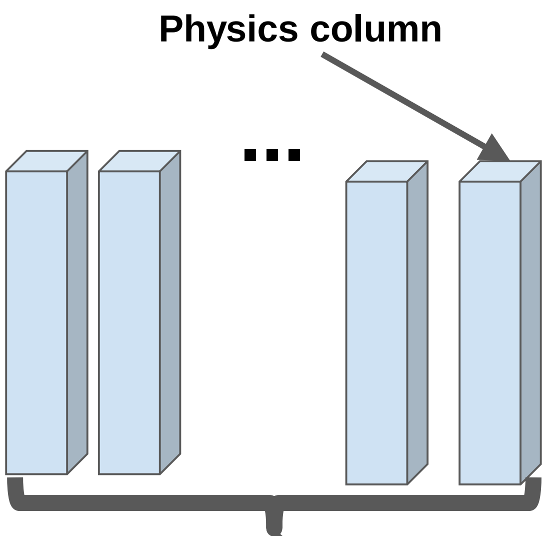
- Acknowledgements:
  - Intel Parallel Computing Center.
  - Alex Reinecke, Kevin Viner (NRL), John Michelakes (UCAR)

Thank you!!

Questions?

E-mail: touermi@sci.utah.edu

Structure of Arrays (SOA)



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”

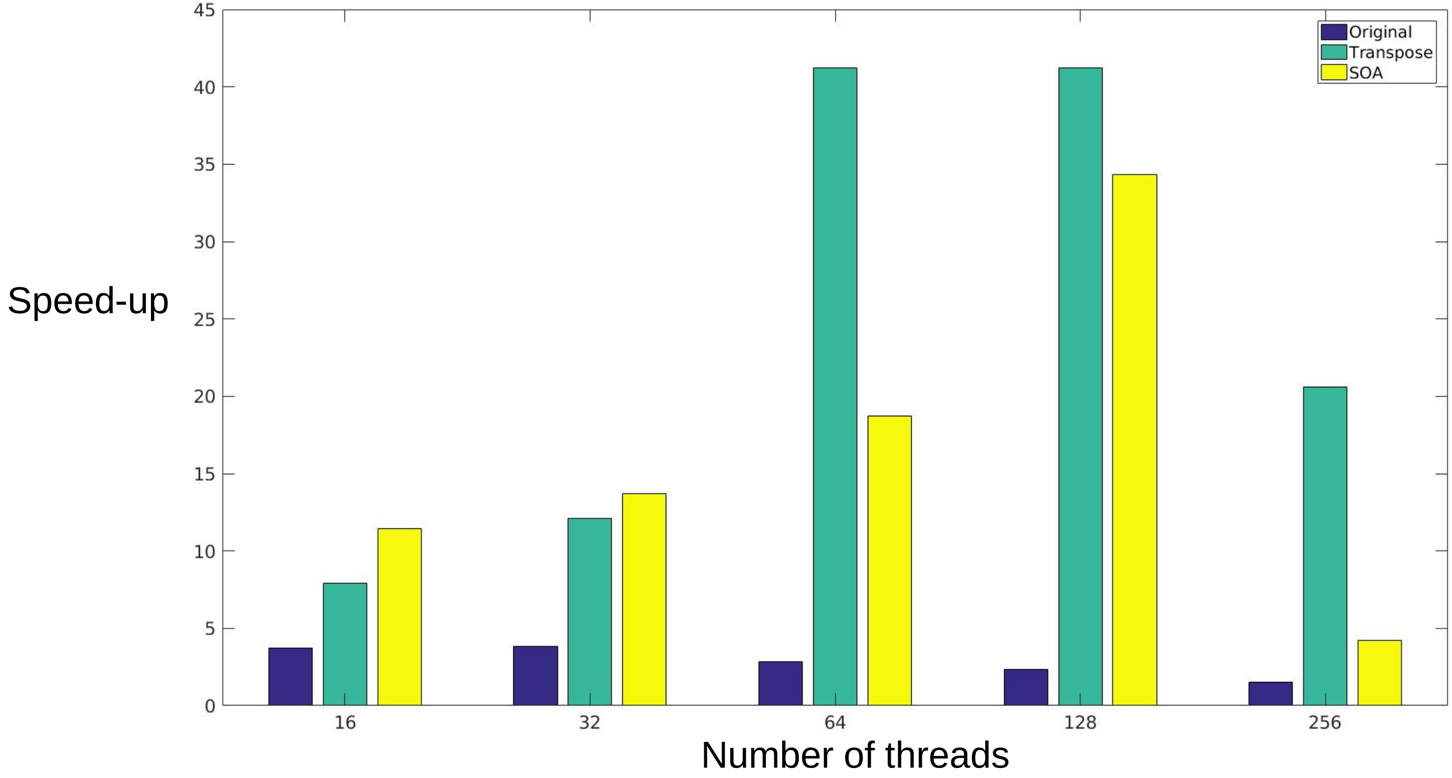
Complex Loop Parallelization

- No conditional 9.7x
- No function calls 30x
- Vectorization 41x

```
do k=kte,kts-1
do i=its,ite
 …
 if(t(i,k).gt.t0c)then
  …
  w(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)…
  end if
  if(qrs(i,k,2).gt.0)then
  psmlg(i,k)=xka(t(i,k), den(i,k)…
  …
  end if
 end if
end do
end do
```

Loop 12 from WSM6

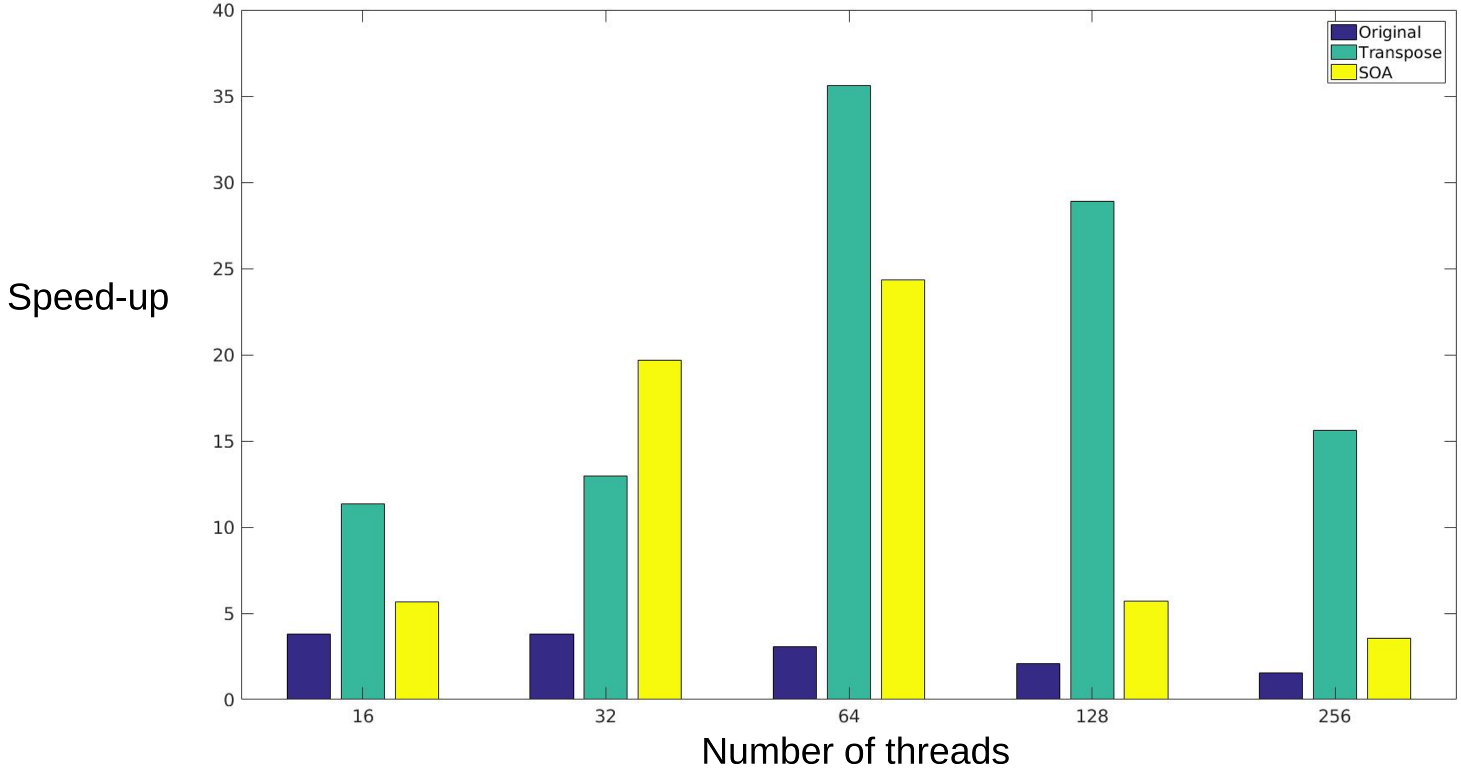
1D Arrays Experiments




```
!$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)
end do
```

1D case

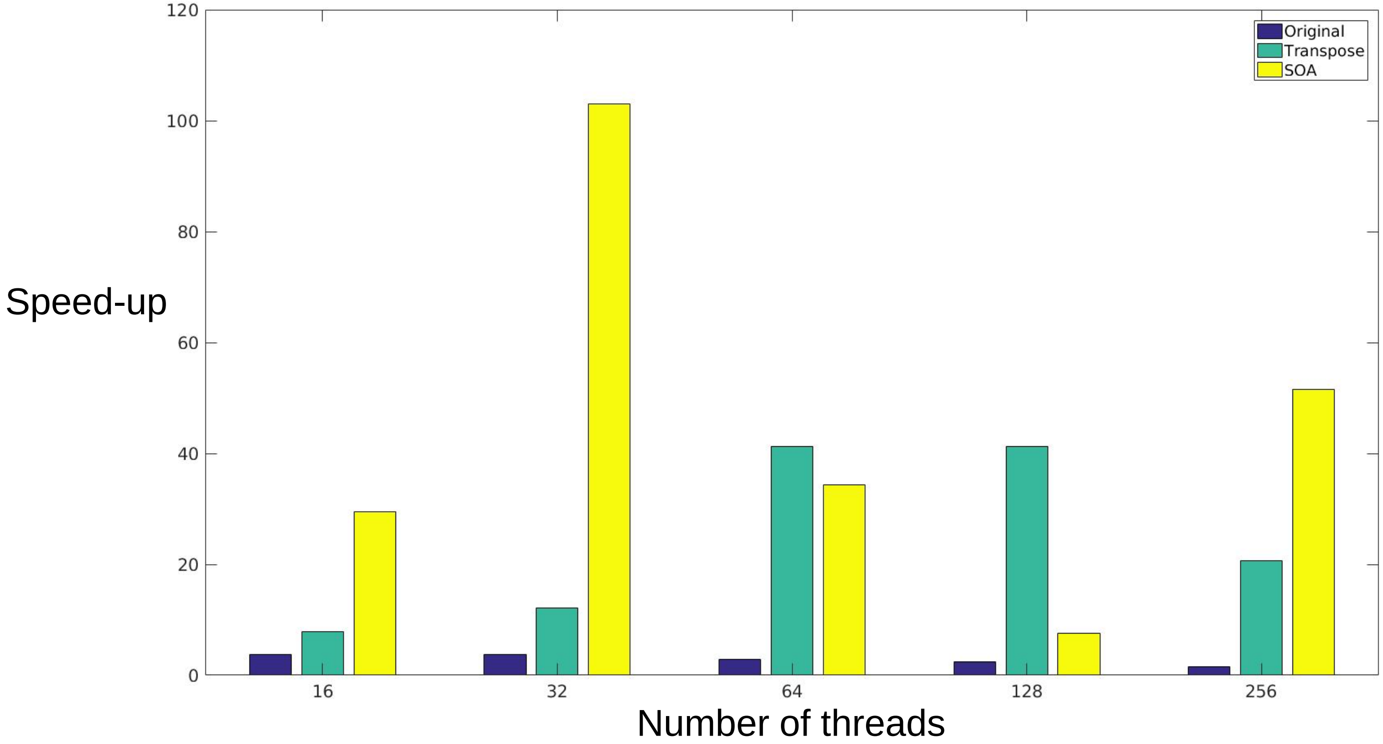
1D Arrays Experiments




```
!$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)
end do
```

1D case with large array sizes

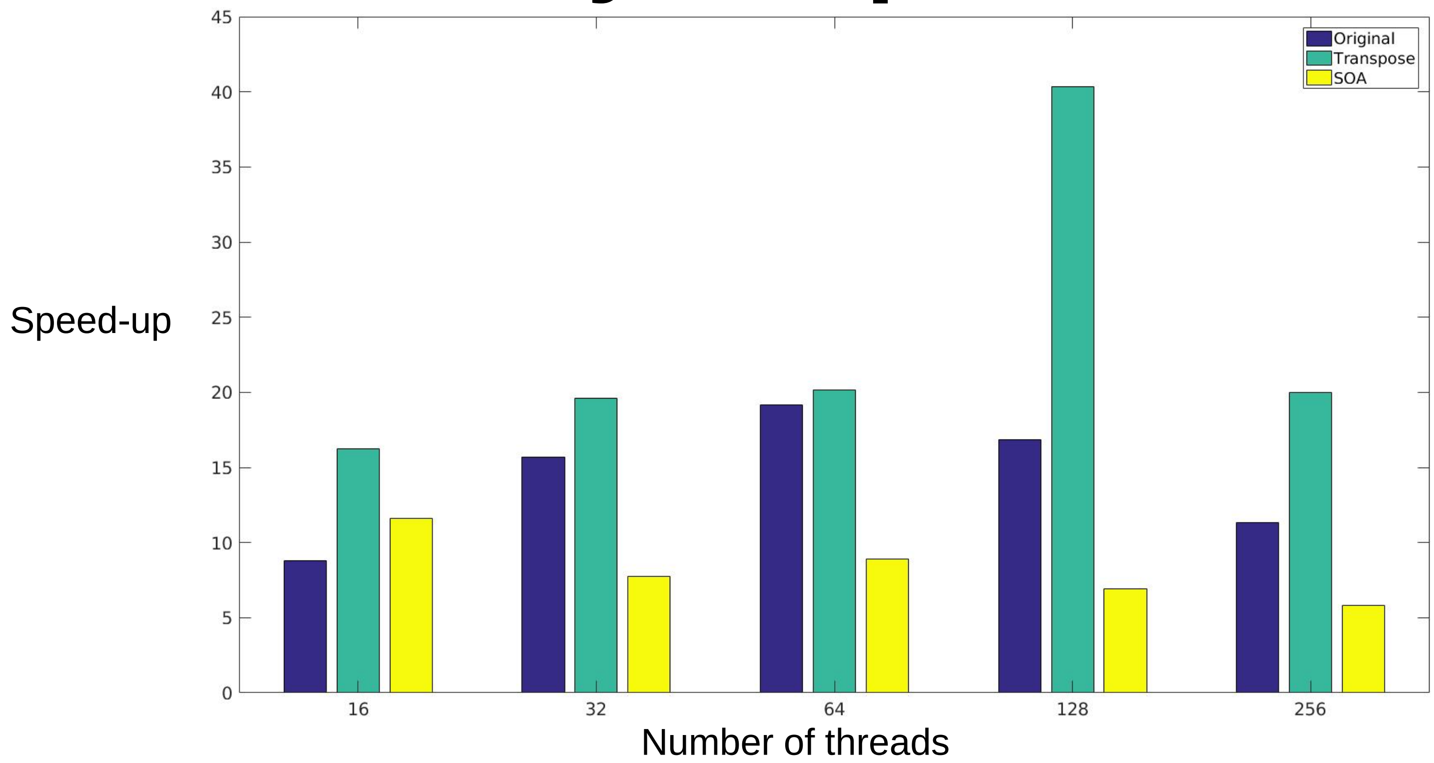
2D Arrays Experiments




```
do j=2,je-1
!$OMP SIMD
do i=1,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)
end do
```

2D case

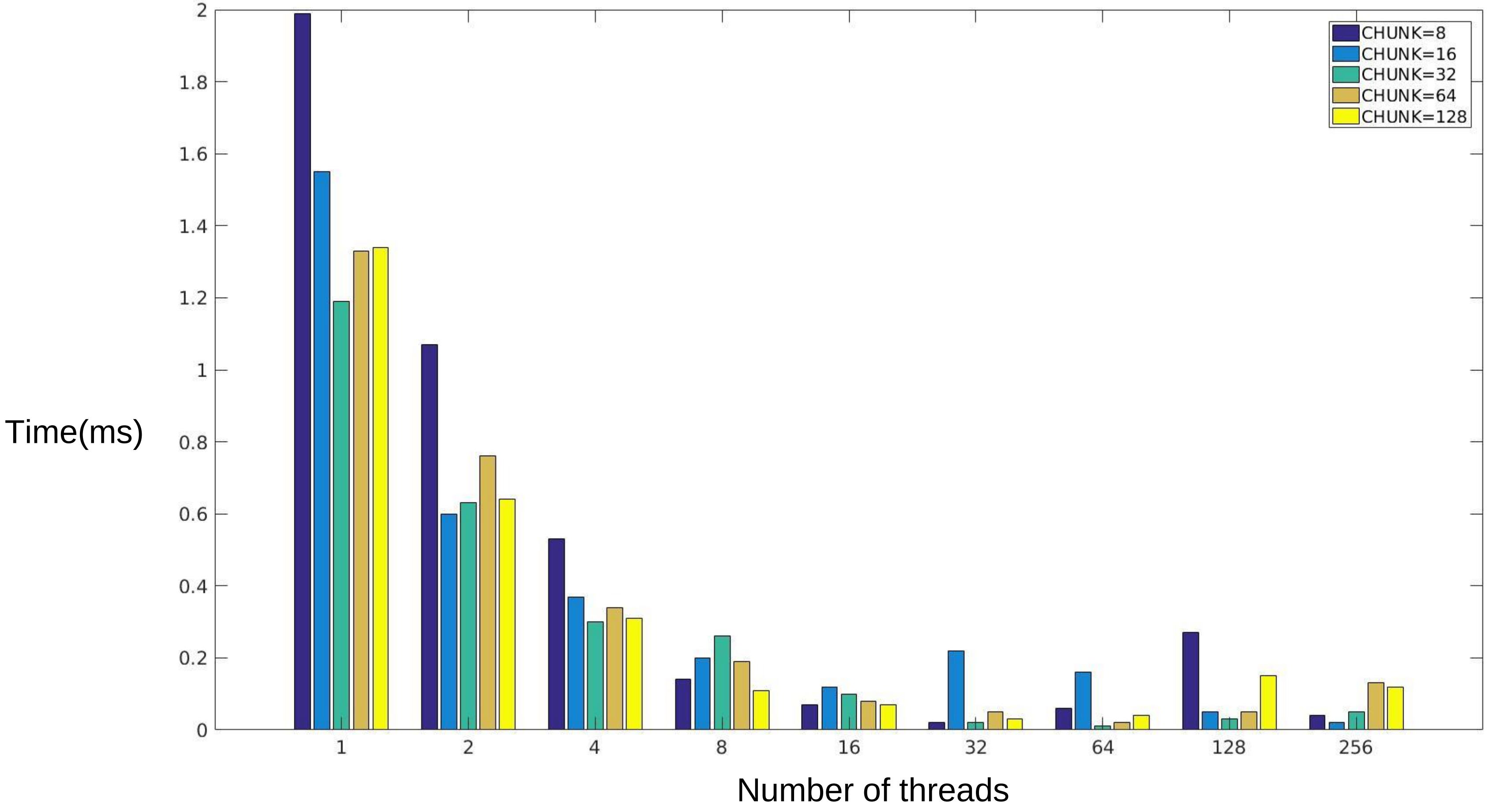
2D Arrays Experiments




```
do j=2,je-1
!$OMP SIMD
do i=1,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)
end do
```

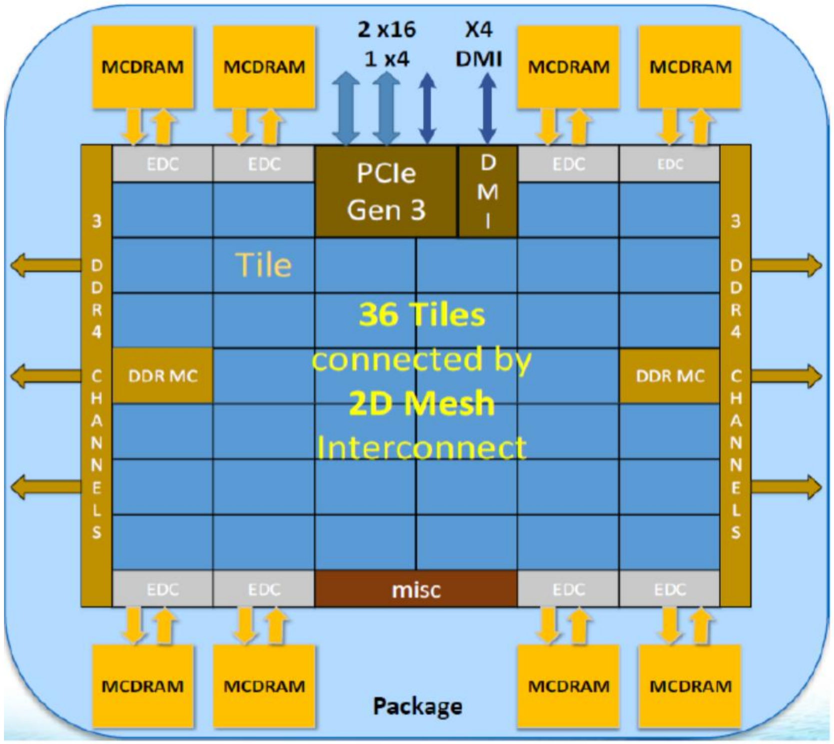
<u>2D case with large array sizes</u>

Chunk Size



Time(ms)
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
CHUNK=8
CHUNK=16
CHUNK=32
CHUNK=64
CHUNK=128
1
2
4
8
16
32
64
128
256
Number of threads

KNL Architecture

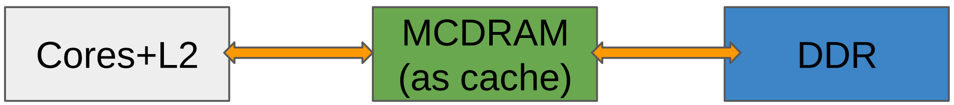


Tile

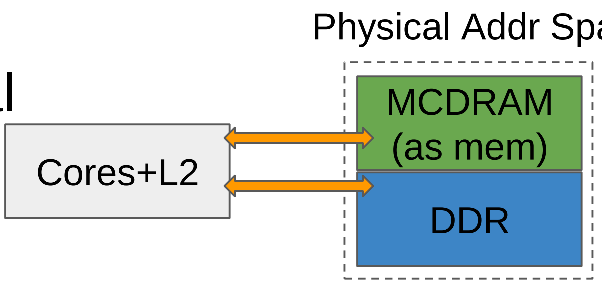
- MCDRAM:16GB, High BW
- Peak 3 teraflops double precision
- 512 bit vectors

MCDRAM & Configurations

- **Cache Mode**
  - No source changes needed
  - Misses are expensive (higher latency)



- **Flat Mode**
  - MCDRAM mapped to physical address
    - use numactl -- for configuration
  - Exposed as NUMA node



- **Hybrid Mode**
  - Combination of flat and cache mode
    - eg: 8GB cache and 8GB flat