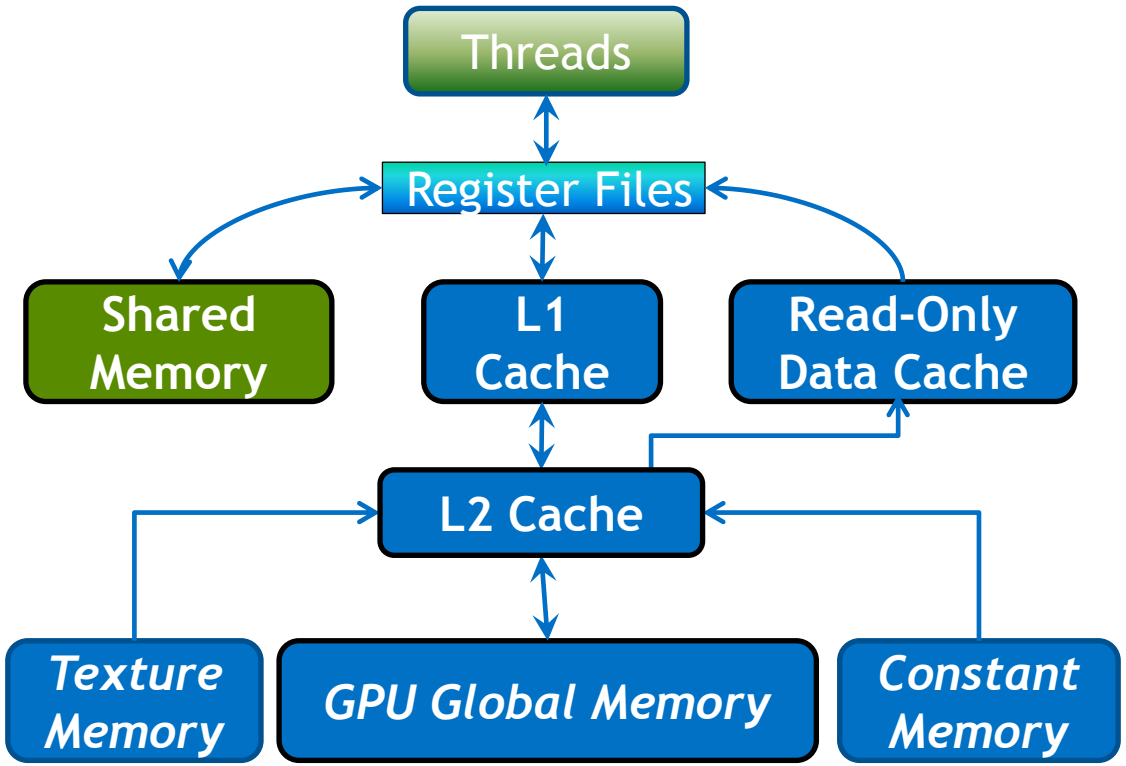
CACHE DIRECTIVE OPTIMIZATION IN THE OPENACC PROGRAMMING MODEL

Xiaonan (Daniel) Tian, Brent Leback, and Michael Wolfe

PGI

GPU ARCHITECTURE



USING SHARED MEMORY WITH CUDA

Creating Shared Memory:

Static Shared Memory	`__shared__ int s[64];`
Dynamic Shared Memory	`extern __shared__ int s[];`

Handling Data Race: `__syncthreads();`

PROS AND CONS OF CUDA APPROACH

Pros:

- Better control over hardware

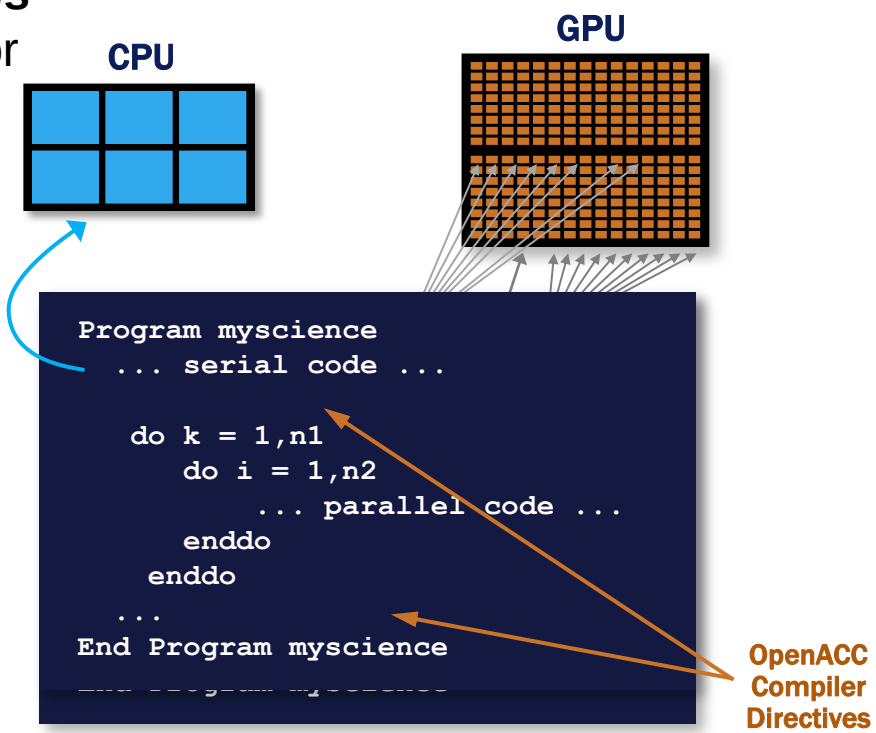
Cons:

- Familiar with CUDA and GPU
- Redesign the algorithm
- Thread Synchronization
- Bank Conflicts

OPENACC: A DIRECTIVE-BASED APPROACH

Rich Set of Data Directives
Two Offload Region Constructs: ***parallel*** and ***kernels***
Three Levels of Parallelism: gang, worker and vector

```
#pragma acc data copyin(a[0:n], b[0:n]),
copyout(c[0:n])
{
 #pragma acc kernels loop independent
  for(i=0; i<n; i++){
        c[i]=a[i] + b[i];
   }
}
```



CACHE DIRECTIVE CONSTRUCT

C/C++

```
#pragma acc cache (a[lower1: length1] [lower2: length2])
```

Fortran

```
!$acc cache (a(lower1:upper1, lower2: upper2))
```

Examples:

```
#pragma acc cache (a[i-1: 3] [j]) // i and j as loop index
!$acc cache (a(J, :))     ! cache the entire dimension
!$acc cache (a)           ! cache the entire array
```

CASE STUDIES

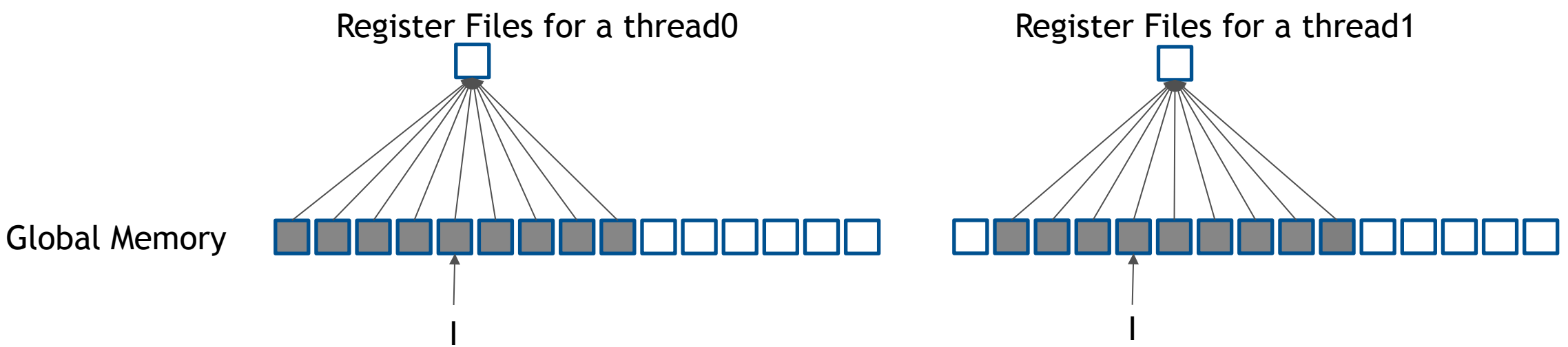
Partial Array Cached

Entire Array Dimension Cached

Entire Array Cached

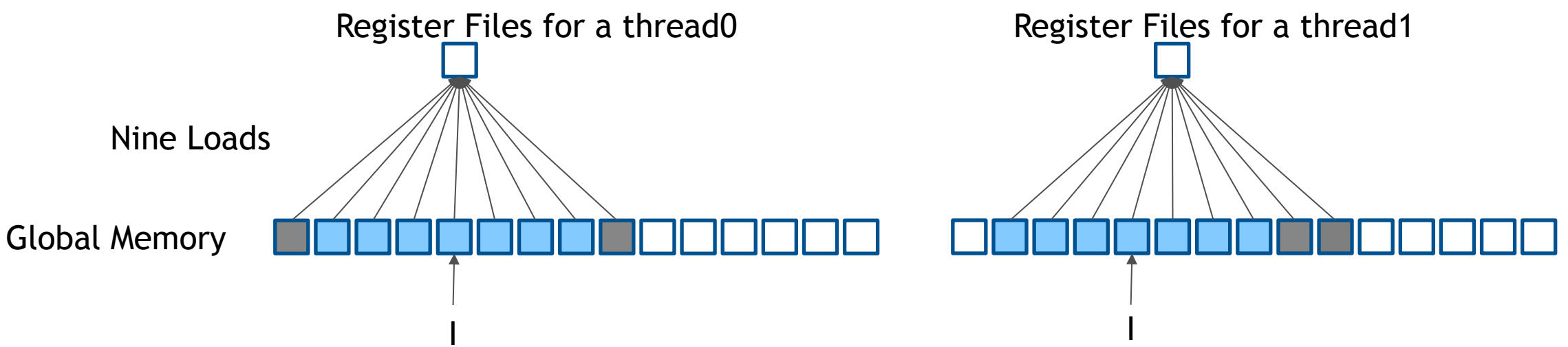
PARTIAL ARRAY CACHED: 1D CACHE

```
!$acc loop gang vector
DO i=4, M
C(i) = (A(i-4) + A(i-3) + A(i-2) +  A(i-1) +
    A(i) + A(i+1) + A(i+2) + A(i+3) + A(i+4))/9.0
```



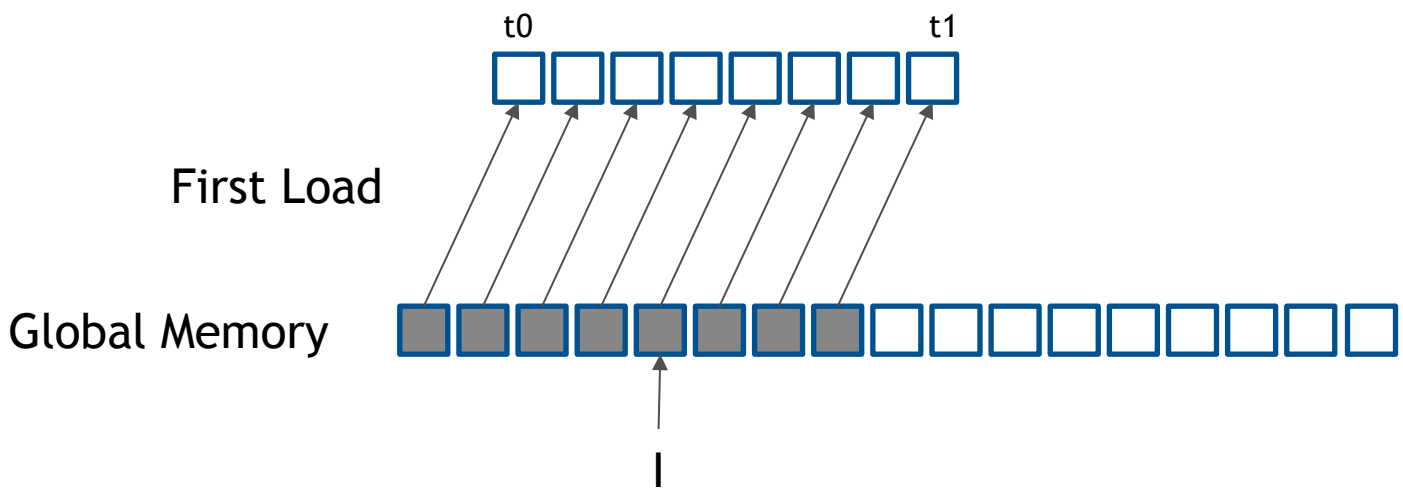
PARTIAL ARRAY CACHED: 1D CACHE

```
!$acc loop gang vector
DO i=4, M
C(i) = (A(i-4) + A(i-3) + A(i-2) +  A(i-1) +
    A(i) + A(i+1) + A(i+2) + A(i+3) + A(i+4))/9.0
```



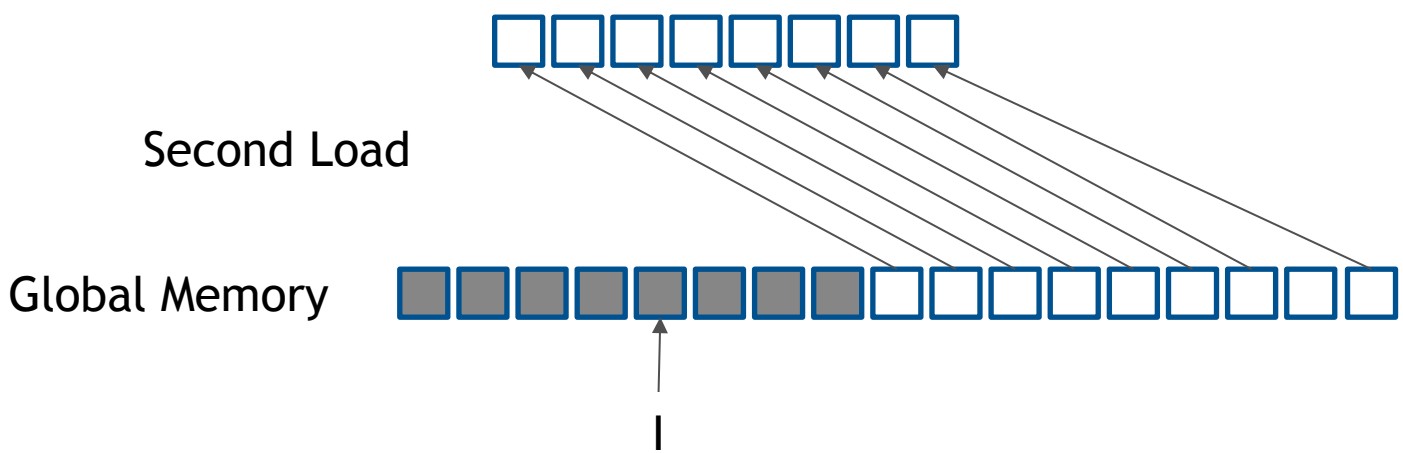
PARTIAL ARRAY CACHED: 1D CACHE

```
!$acc loop gang vector
DO i=4, M
    !$acc cache (A(i-4:i+4))
    C(i) = (A(i-4) + A(i-3) + A(i-2) +  A(i-1) +
A(i) + A(i+1) + A(i+2) + A(i+3) + A(i+4))/9.0
```



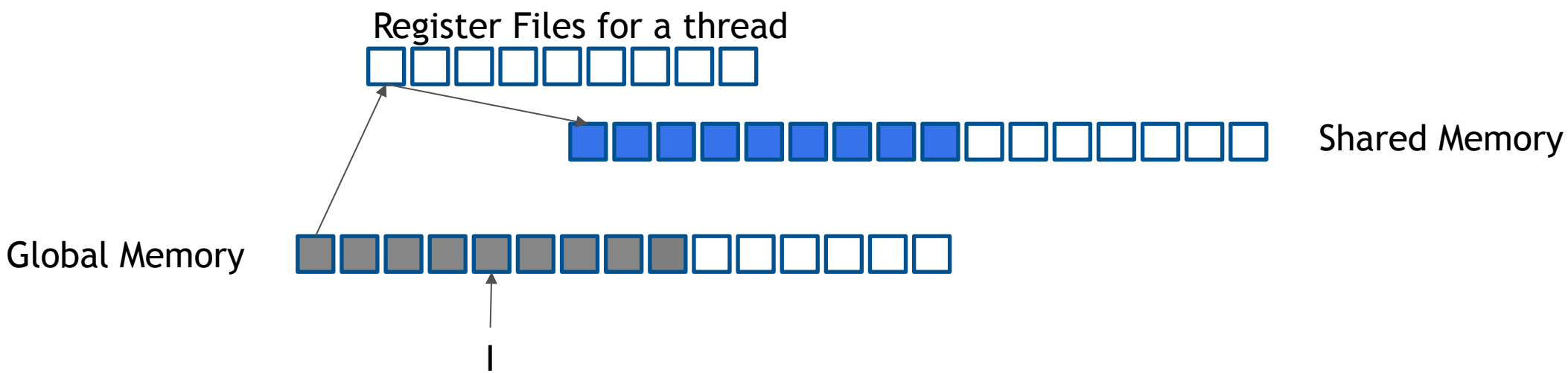
PARTIAL ARRAY CACHED: 1D CACHE

```
!$acc loop gang vector
DO i=4, M
    !$acc cache (A(i-4:i+4))
    C(i) = (A(i-4) + A(i-3) + A(i-2) +  A(i-1) +
A(i) + A(i+1) + A(i+2) + A(i+3) + A(i+4))/9.0
```



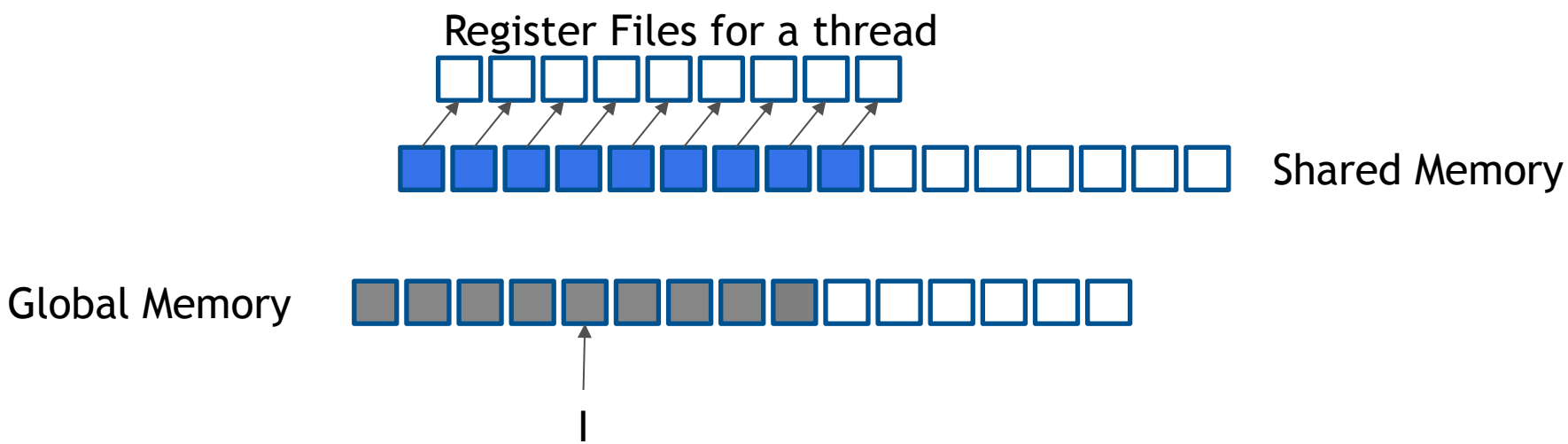
PARTIAL ARRAY CACHED: 1D CACHE

```
!$acc loop gang vector
DO i=4, M
    !$acc cache (A(i-4:i+4))
    C(i) = (A(i-4) + A(i-3) + A(i-2) +  A(i-1) +
A(i) + A(i+1) + A(i+2) + A(i+3) + A(i+4))/9.0
```



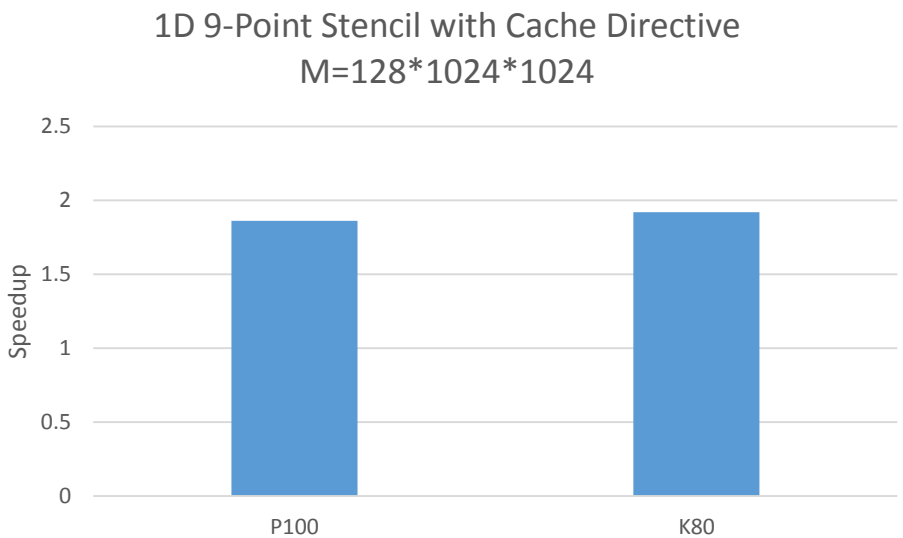
PARTIAL ARRAY CACHED: 1D CACHE

```
!$acc loop gang vector
DO i=4, M
    !$acc cache (A(i-4:i+4))
    C(i) = (A(i-4) + A(i-3) + A(i-2) +  A(i-1) +
A(i) + A(i+1) + A(i+2) + A(i+3) + A(i+4))/9.0
```



PARTIAL ARRAY CACHED: 1D CACHE

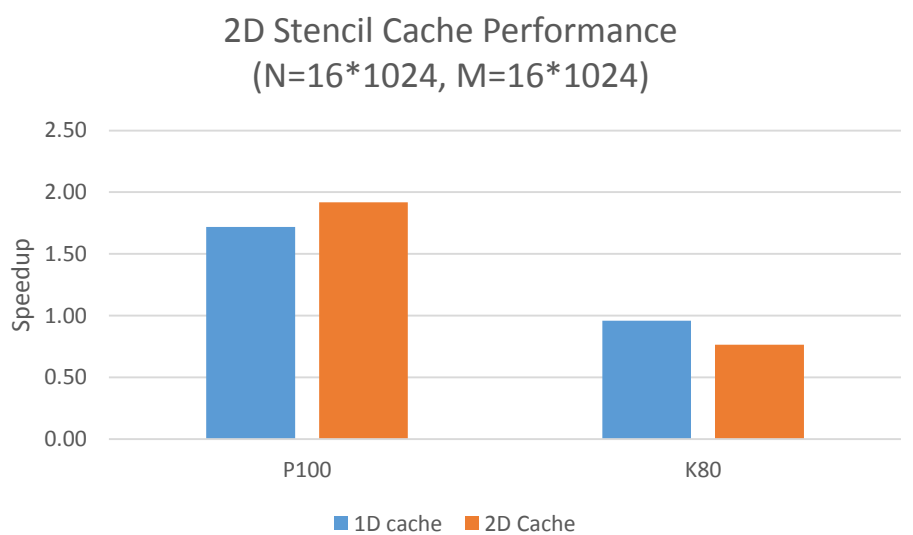
```
!$acc loop gang vector
DO i=1, M
    !$acc cache (A(i-4:i+4))
    C(i) = (A(i-4) + A(i-3) + A(i-2) +  A(i-1) +
A(i) + A(i+1) + A(i+2) + A(i+3) + A(i+4))/9.0
```



PARTIAL ARRAY CACHED: 1D VS 2D

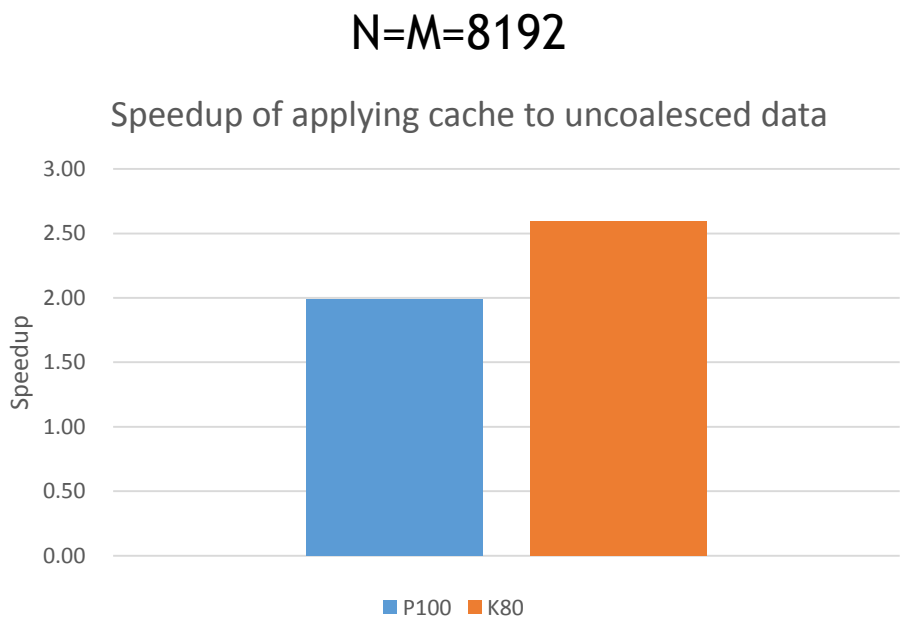
```
!$acc loop gang
DO j=1,N
!$acc loop vector
DO i=1, M
   !$acc cache (A(i-4:i+4, j))
   C(i, j) = (A(i-4, j) + A(i-3, j)  + A(i-2, j) +  A(i-1, j)  + A(i, j)
     + A(i+1, j) + A(i+2, j) + A(i+3, j) + A(i+4, j)
         + A(i, j-4) + A(i, j-3)  + A(i, j-2) +  A(i, j-1)
     + A(i, j+1) + A(i, j+2) + A(i, j+3) + A(i, j+4)) * coeff
```

```
!$acc loop gang
DO j=1,N
!$acc loop vector
DO i=1, M
   !$acc cache (A(i-4:i+4, j-4:j+4))
   C(i, j) = (A(i-4, j) + A(i-3, j)  + A(i-2, j) +  A(i-1, j)  + A(i, j)
     + A(i+1, j) + A(i+2, j) + A(i+3, j) + A(i+4, j)
         + A(i, j-4) + A(i, j-3)  + A(i, j-2) +  A(i, j-1)
     + A(i, j+1) + A(i, j+2) + A(i, j+3) + A(i, j+4)) * coeff
```



PARTIAL ARRAY CACHED: UNCOALESCED

```
!$acc loop gang
DO j=1,N
!$acc loop vector
DO i=1, M
    !$acc cache (B(j, i-1:i+1))
    C(i, j) = (A(i-1, j)  + A(i, j) + A(i+1, j) +
              B(j, i+1) + B(j, i) + B(j, i+1)) * coeff
```



ENTIRE ARRAY DIMENSIONS CACHED

ORNL CAAR ACME

```
!$acc parallel loop gang collapse(3)
do ie = 1 , nelemd
      do q = 1 , qsize
        do ks = 1 , nlev, kchunk
          !$acc cache(s(:,:,ks:ks+kchunk-1,q,ie))
          !$acc loop vector collapse(3)
          do k = 1 , kchunk
            do j = 1 , np
              do i = 1 , np
                do l = 1 , np
                  dsdx00 = dsdx00 + deriv_dvv(l,i)*s(l,j,ks+k-1,q,ie)
                  ...
```

ENTIRE ARRAY CACHED

Nonhydrostatic Icosahedral Model: NIM

```
!$acc parallel acc loop gang private(fu0, sumu, …)
do ipn=IPS,IPE
!$acc cache(fu0, sumu, …)
!$acc loop vector
  do k=1,NZ
    fu0(k) = 0.0
    …
  enddo
!$acc loop vector
 do k=1,NZ
    fu0(k) = fu0(k) + sumu(k)
    ….
  end do
```

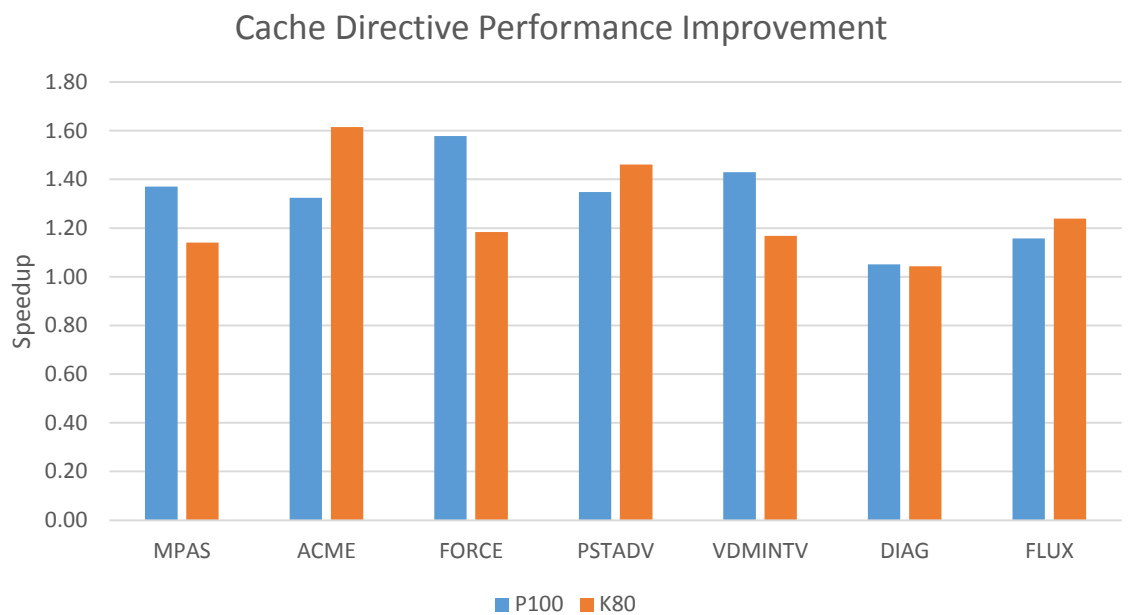
VARIABLE-LENGTH ARRAY

```
real :: a(NX)
...
!$acc loop gang private(a)
DO j=1,N
  !$acc cache (a)
  !$acc loop vector
  DO i=1, M
...
```

pgfortran -acc -ta=tesla:safecache a.f90 -Minfo

PERFORMANCE DATA

Kernels from Real-World Apps



DISCUSSION

Recommendation often given:

If there is data reuse within the thread-block, then use shared memory to cache such data and then access latency is reduced.

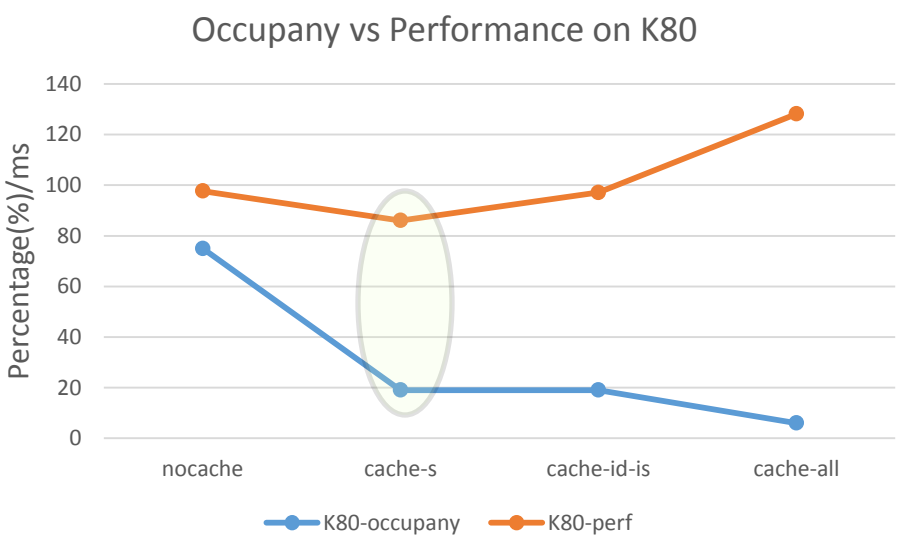
Recommendation $\neq$ Better Performance

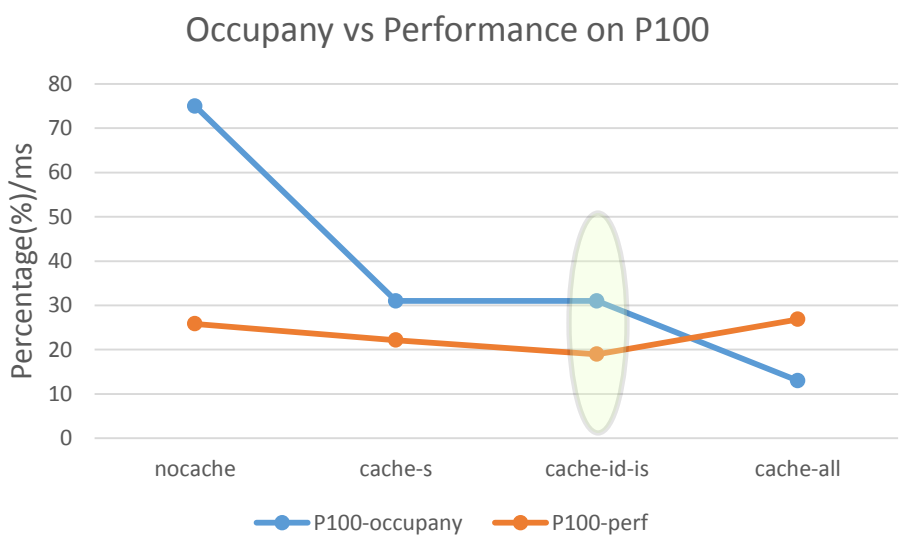
Performance Factors:

Thread Occupancy Memory Access Latency Hardware Platforms Others

CASE STUDY: ORNL DIRAC

Array	S (double)	Is(int)	Id(int)
Size	1599*8	1599*4	1599*4





CONCLUSION

Summary:

Cache directive does improve the performance in real world applications

Pros:

Help reduce uncoalesced memory access

Combining with gang-level private, avoid data fetch from global memory

Cons:

No performance improvement guarantee, if the shared memory is overly used