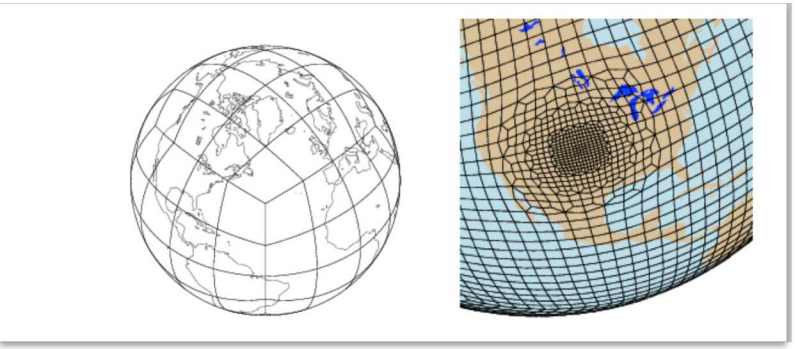
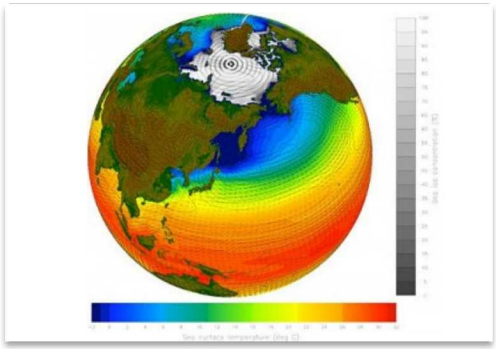
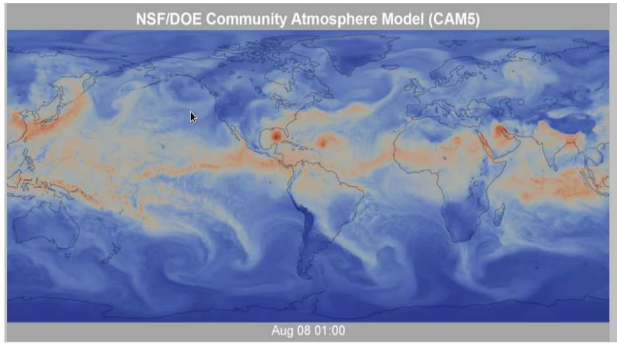
SAND2015-7848C

Exceptional service in the national interest

Sandia National Laboratories







GPU implementations of the CAM-SE advection limiter

Irina Demeshko, Mark A. Taylor, Matthew R. Norman

2015 Heterogeneous Multi-Core 4 Workshop, 09/17/2015

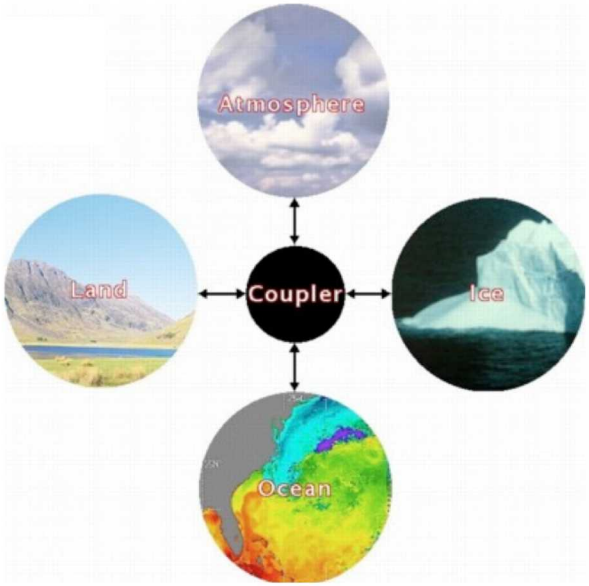
U.S. DEPARTMENT OF ENERGY | NNSA National Nuclear Security Administration

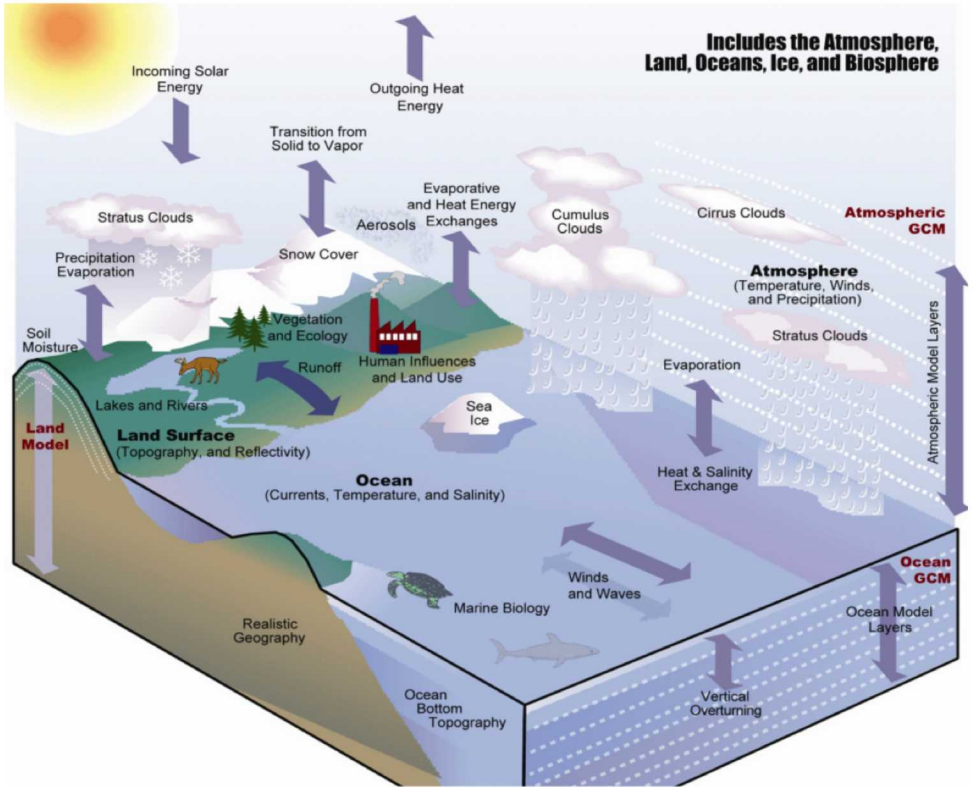
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Outline

- Introduction
- HOMME on GPU
- Advection limiters
- Limiter 8 algorithms
- Performance evaluation results
- Conclusion

CESM





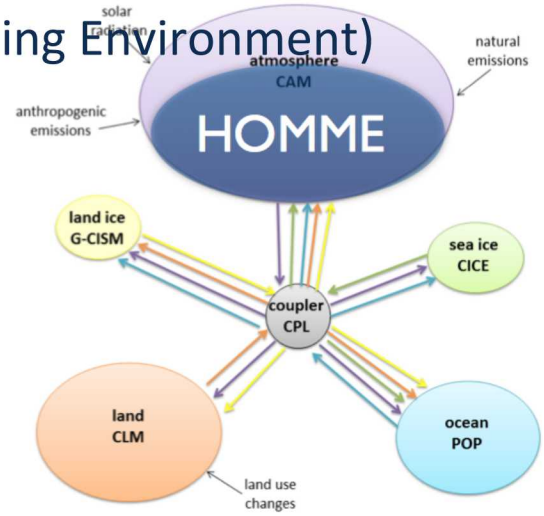
CESM is an IPCC-class model developed by NCAR, National Labs and Universities

Atmosphere, Land, Ocean and Sea ice component models

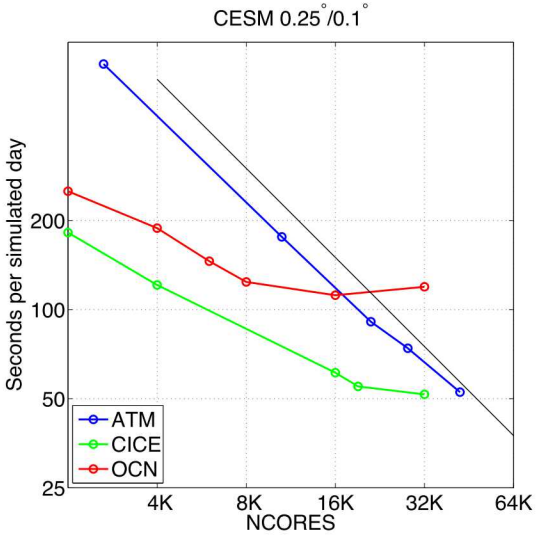
CAM is the atmosphere component model for the CESM

HOMME (High-Order Method Modeling Environment)

- Dynamical core for atmosphere
- Performance scalable
- Support quasi-uniform and variable resolution grids on parallel computers
- Three options:
  Spectral Element (SE) and DG are available, CSLAM coming soon
- **CAM-SE option: advection operator requires limiter**



Vadlamani, S. ,Xscale Conf., Boulder, CO, Aug. 15&16,2013

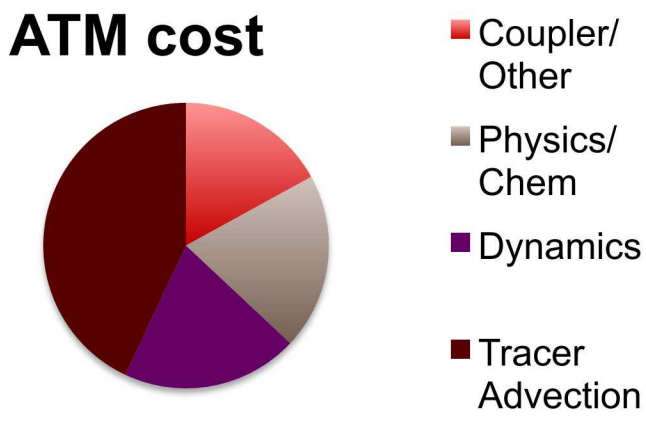


*Mark A. Taylor, DCMIP Summer School, July 30-Aug 10,Boulder

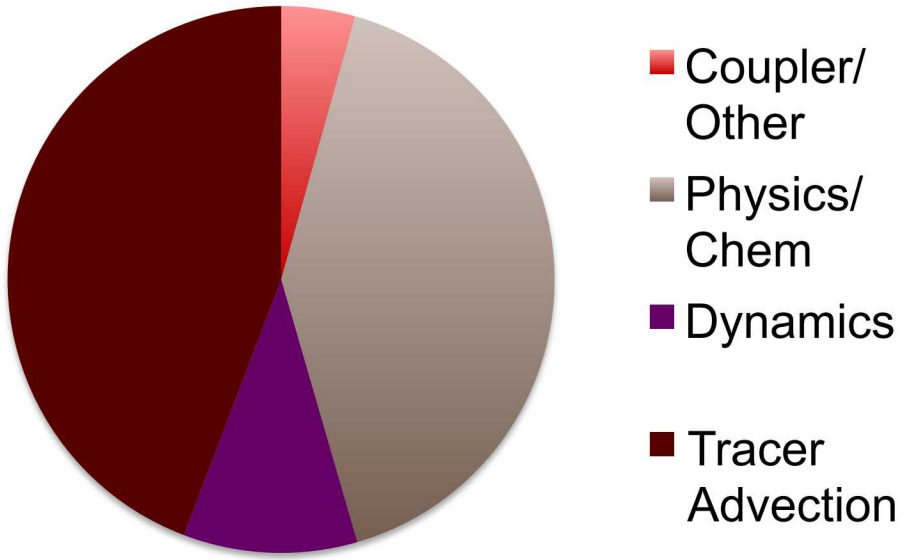
CAM-SE cost estimate

Sandia National Laboratories

CESM (today's configuration)

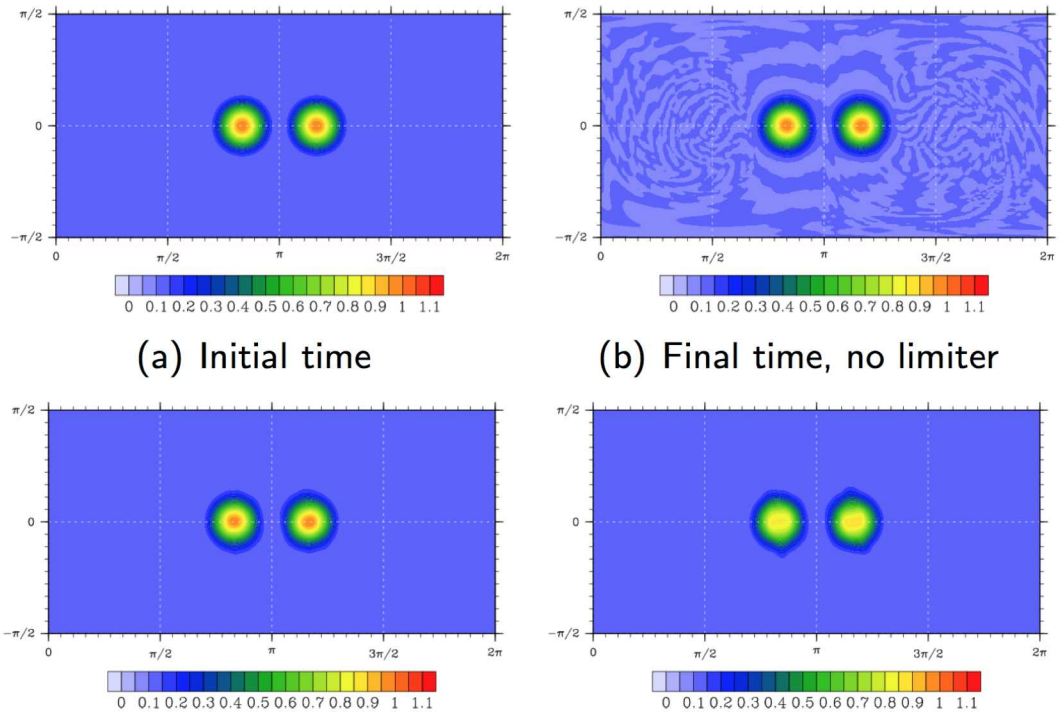


Proposed DOE configuration (in few years)



Advection limiters*

Why we need limiter:



(a) Initial time

(b) Final time, no limiter

(c) Final time, limiter with 1 constraint

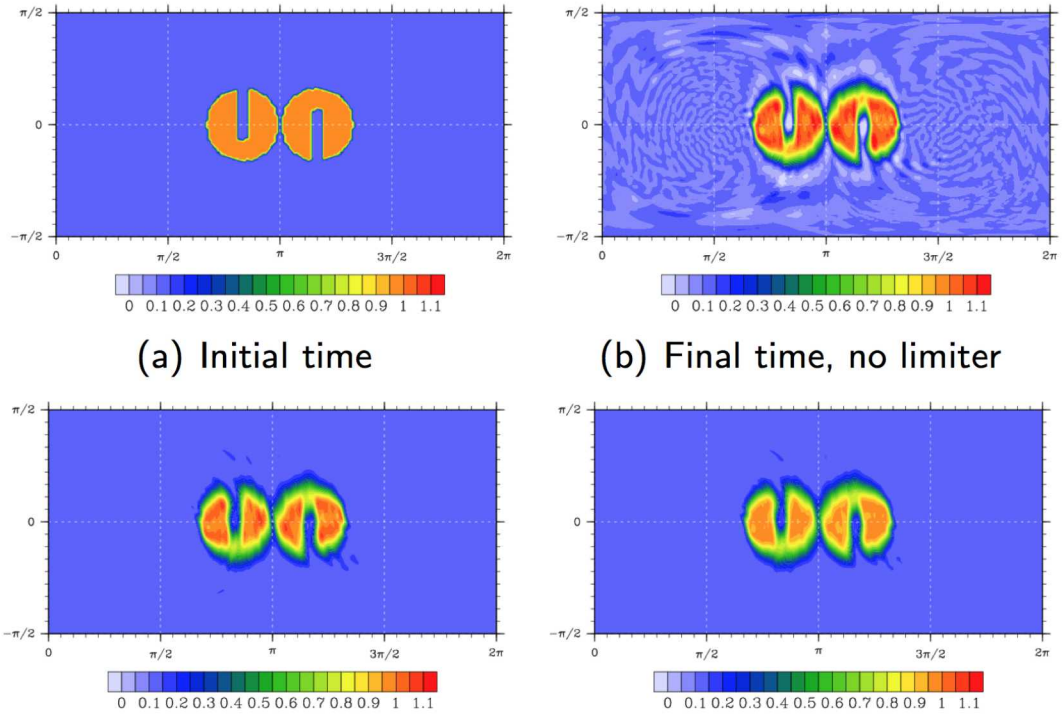
(d) Final time, limiter with 2 constraints

Figure: 1.5° resolution, cosine bells, 7 GLL points

*O.Guba, M. Taylor, A. St.-Cyr. Local Extrema Diminishing Advection for High-Order Finite Element Methods, Transport Workshop, NCAR , 2011

Advection limiters

Why we need limiter (cont.):



(a) Initial time

(b) Final time, no limiter

(c) Final time, limiter with 1 constraint

(d) Final time, limiter with 2 constraints

Figure: 1.5° resolution, slotted cylinders, 7 GLL points

Advection limiters*

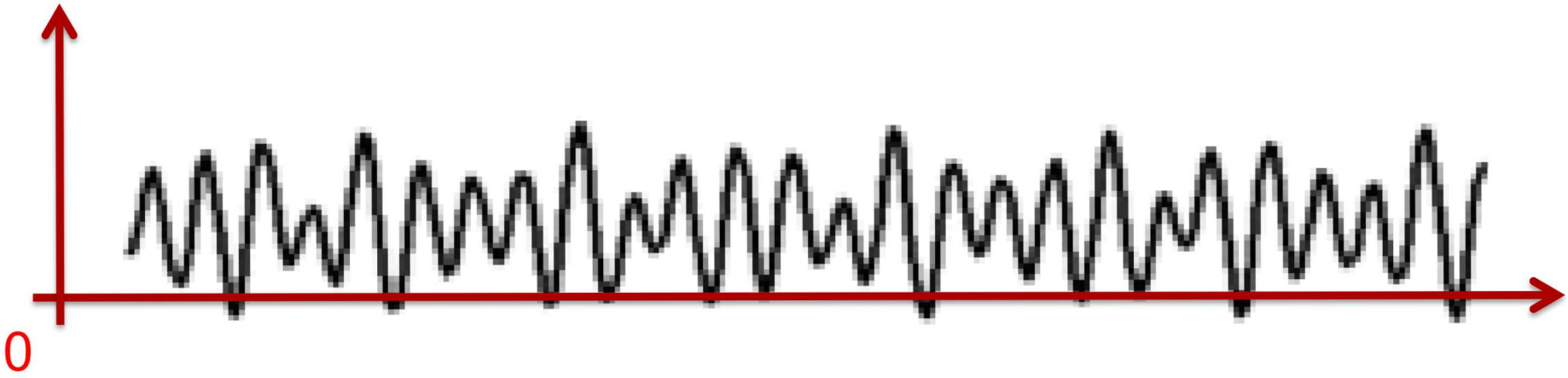
- CAM_SE: 3 optimization-based limiters for h-p spectral element method:
  - Optimal (limiter_optim_iter_full): the most expensive one
  - Min/max limiter (limiter2d_minmax)
  - Sign-preserving limiter (limiter2d_zero): the easiest one

Sign-preserving limiter

Sandia National Laboratories

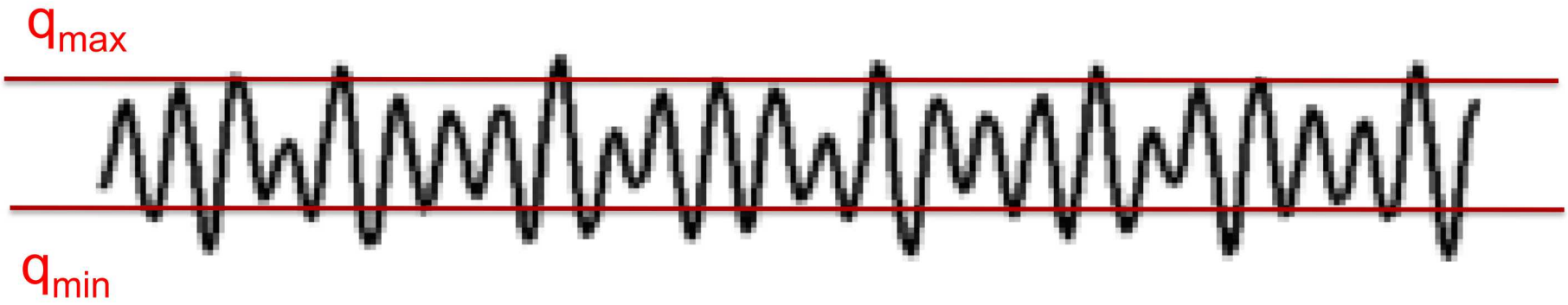
is identical to min/max limiter, but

$$0 \leq \tilde{q}_l$$



Min/max limiter

- Instead of solving a standard quadratic program we find a non-optimal solution in one iteration, which satisfy all the constrains but not necessary the best one.



Optimal limiter

- The idea here is the following: We need to find a grid field($\tilde{q}_i$) which is closest to the unlimited solution (q_i) (in terms of weighted sum), but satisfies the min/max constraints. So, first we find values which do not satisfy constraints and bring these values to a closest constraint. This way we introduce some mass change

$$\min_{\tilde{q}\in\mathbb{R}^I} \sum_{i=1}^{I} w_i\rho_i(\tilde{q}_i - q_i)^2, \quad \sum_{i=1}^{I} w_i\rho_i q_i = \sum_{i=1}^{I} w_i\rho_i\tilde{q}_i,$$

and $q_{min} \le \tilde{q}_i \le q_{max}$

Modification: $0 = q_{min} \le \tilde{q}_i$

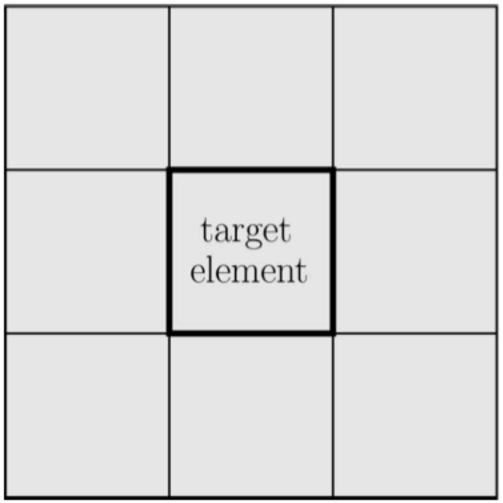


Figure: Stencil for defining constraints q_{min}, q_{max}

- so, we redistribute this mass change in the way that *l2* error is smallest. This redistribution might violate constraints thus, we do a few iterations.

Limiter8 original (indirect addressing)

Sandia National Laboratories

```
subroutine limiter_optim_iter_full(ptens,sphweights,minp,maxp,dpmass)

 ..code ommited

  integer, parameter :: maxiter = 5

  do k = 1 , nlev
    weights(:,k) = sphweights(:) * dpmass(:,k)
    ptens(:,k) = ptens(:,k) / dpmass(:,k)
  enddo

  do k = 1 , nlev
    c = weights(:,k)
    x = ptens(:,k)
    mass = sum(c*x)
    if( (mass / sum(c)) < minp(k) ) then
      minp(k) = mass / sum(c)
    endif
    if( (mass / sum(c)) > maxp(k) ) then
      maxp(k) = mass / sum(c)
    endif

    addmass = 0.0d0
    pos_counter = 0;
    neg_counter = 0;

    do k1 = 1 , np*np
      if ( ( x(k1) >= maxp(k) ) ) then
        addmass = addmass + ( x(k1) - maxp(k) ) * c(k1)
        x(k1) = maxp(k)
        whois_pos(k1) = -1
      else
        pos_counter = pos_counter+1;
        whois_pos(pos_counter) = k1;
      endif
      if ( ( x(k1) <= minp(k) ) ) then
        addmass = addmass - ( minp(k) - x(k1) ) * c(k1)
        x(k1) = minp(k)
        whois_neg(k1) = -1
      else
        neg_counter = neg_counter+1;
        whois_neg(neg_counter) = k1;
      endif
    enddo

    weightssum = 0.0d0
    if ( addmass > 0 ) then
      do i2 = 1 , maxIter
        weightssum = 0.0
        do k1 = 1 , pos_counter
          i1 = whois_pos(k1)
          weightssum = weightssum + c(i1)
          al_pos(i1) = maxp(k) - x(i1)
        enddo

        if( ( pos_counter > 0 ) .and. ( addmass > tol_limiter * abs(mass) ) ) then
          do k1 = 1 , pos_counter
            i1 = whois_pos(k1)
            howmuch = addmass / weightssum
            if ( howmuch > al_pos(i1) ) then
              howmuch = al_pos(i1)
              whois_pos(k1) = -1
            endif
            addmass = addmass - howmuch * c(i1)
            weightssum = weightssum - c(i1)
            x(i1) = x(i1) + howmuch
          enddo
          neg_counter = pos_counter
          whois_neg = whois_pos
          whois_pos = -1
          pos_counter = 0
          do k1 = 1 , neg_counter
            if ( whois_neg(k1) .ne. -1 ) then
              pos_counter = pos_counter+1
              whois_pos(pos_counter) = whois_neg(k1)
            endif
          enddo
        else
          exit
        endif
      enddo
    else
      do i2 = 1 , maxIter
        weightssum = 0.0
        do k1 = 1 , neg_counter
          i1 = whois_neg(k1)
          weightssum = weightssum + c(i1)
          al_neg(i1) = x(i1) - minp(k)
        enddo

        if ( ( neg_counter > 0 ) .and. ( (-addmass) > tol_limiter * abs(mass) ) ) then
          do k1 = 1 , neg_counter
            i1 = whois_neg(k1)
            howmuch = -addmass / weightssum
            if ( howmuch > al_neg(i1) ) then
              howmuch = al_neg(i1)
              whois_neg(k1) = -1
            endif
            addmass = addmass + howmuch * c(i1)
            weightssum = weightssum - c(i1)
            x(i1) = x(i1) - howmuch
          enddo
          !now sort whois_pos and get a new number for pos_counter
          !here pos_counter and whois_pos serve as temp vars
          pos_counter = neg_counter
          whois_pos = whois_neg
          whois_neg = -1
          neg_counter = 0
          do k1 = 1 , pos_counter
            if ( whois_pos(k1) .ne. -1 ) then
              neg_counter = neg_counter+1
              whois_neg(neg_counter) = whois_pos(k1)
            endif
          enddo
        else
          exit
        endif
      enddo
    endif

    ptens(:,k) = x
  enddo

  do k = 1 , nlev
    ptens(:,k) = ptens(:,k) * dpmass(:,k)
  enddo
end subroutine limiter_optim_iter_full
```

Limiter8 original (indirect addressing)

Sandia National Laboratories

```
subroutine limiter_optim_iter_full(ptens,sphweights,minp,maxp,dpmass)

 ..code ommited

  integer, parameter :: maxiter = 5

  do k = 1 , nlev
    weights(:,k) = sphweights(:) * dpmass(:,k)
    ptens(:,k) = ptens(:,k) / dpmass(:,k)
  enddo

  do k = 1 , nlev
    c = weights(:,k)
    x = ptens(:,k)
    mass = sum(c*x)
    if( (mass / sum(c)) < minp(k) ) then
      minp(k) = mass / sum(c)
    endif
    if( (mass / sum(c)) > maxp(k) ) then
      maxp(k) = mass / sum(c)
    endif

    addmass = 0.0d0
    pos_counter = 0;
    neg_counter = 0;

    do k1 = 1 , np*np
      if ( ( x(k1) >= maxp(k) ) ) then
        addmass = addmass + ( x(k1) - maxp(k) ) * c(k1)
        x(k1) = maxp(k)
        whois_pos(k1) = -1
      else
        pos_counter = pos_counter+1;
        whois_pos(pos_counter) = k1;
      endif
      if ( ( x(k1) <= minp(k) ) ) then
        addmass = addmass - ( minp(k) - x(k1) ) * c(k1)
        x(k1) = minp(k)
        whois_neg(k1) = -1
      else
        neg_counter = neg_counter+1;
        whois_neg(neg_counter) = k1;
      endif
    enddo

    weightssum = 0.0d0
    if ( addmass > 0 ) then
      do i2 = 1 , maxIter
        weightssum = 0.0
        do k1 = 1 , pos_counter
          i1 = whois_pos(k1)
          weightssum = weightssum + c(i1)
          al_pos(i1) = maxp(k) - x(i1)
        enddo

        if( ( pos_counter > 0 ) .and. ( addmass > tol_limiter * abs(mass) ) ) then
          do k1 = 1 , pos_counter
            i1 = whois_pos(k1)
            howmuch = addmass / weightssum
            if ( howmuch > al_pos(i1) ) then
              howmuch = al_pos(i1)
              whois_pos(k1) = -1
            endif
            addmass = addmass - howmuch * c(i1)
            weightssum = weightssum - c(i1)
            x(i1) = x(i1) + howmuch
          enddo
          neg_counter = pos_counter
          whois_neg = whois_pos
          whois_pos = -1
          pos_counter = 0
          do k1 = 1 , neg_counter
            if ( whois_neg(k1) .ne. -1 ) then
              pos_counter = pos_counter+1
              whois_pos(pos_counter) = whois_neg(k1)
            endif
          enddo
        else
          exit
        endif
      enddo
    else
      do i2 = 1 , maxIter
        weightssum = 0.0
        do k1 = 1 , neg_counter
          i1 = whois_neg(k1)
          weightssum = weightssum + c(i1)
          al_neg(i1) = x(i1) - minp(k)
        enddo

        if ( ( neg_counter > 0 ) .and. ( (-addmass) > tol_limiter * abs(mass) ) ) then
          do k1 = 1 , neg_counter
            i1 = whois_neg(k1)
            howmuch = -addmass / weightssum
            if ( howmuch > al_neg(i1) ) then
              howmuch = al_neg(i1)
              whois_neg(k1) = -1
            endif
            addmass = addmass + howmuch * c(i1)
            weightssum = weightssum - c(i1)
            x(i1) = x(i1) - howmuch
          enddo
          !now sort whois_pos and get a new number for pos_counter
          !here pos_counter and whois_pos serve as temp vars
          pos_counter = neg_counter
          whois_pos = whois_neg
          whois_neg = -1
          neg_counter = 0
          do k1 = 1 , pos_counter
            if ( whois_pos(k1) .ne. -1 ) then
              neg_counter = neg_counter+1
              whois_neg(neg_counter) = whois_pos(k1)
            endif
          enddo
        else
          exit
        endif
      enddo
    endif

    ptens(:,k) = x
  enddo

  do k = 1 , nlev
    ptens(:,k) = ptens(:,k) * dpmass(:,k)
  enddo
end subroutine limiter_optim_iter_full
```

First, we find values which do not satisfy Constraints (overshoots and undershoots) and store their ID in whois_pos/neg arrays

Limiter8 original (indirect addressing)

Sandia National Laboratories

```
subroutine limiter_optim_iter_full(ptens,sphweights,minp,maxp,dpmass)

 ..code ommited

  integer, parameter :: maxiter = 5

  do k = 1 , nlev
    weights(:,k) = sphweights(:) * dpmass(:,k)
    ptens(:,k) = ptens(:,k) / dpmass(:,k)
  enddo

  do k = 1 , nlev
    c = weights(:,k)
    x = ptens(:,k)
    mass = sum(c*x)
    if( (mass / sum(c)) < minp(k) ) then
      minp(k) = mass / sum(c)
    endif
    if( (mass / sum(c)) > maxp(k) ) then
      maxp(k) = mass / sum(c)
    endif

    addmass = 0.0d0
    pos_counter = 0;
    neg_counter = 0;

    do k1 = 1 , np*np
      if ( ( x(k1) >= maxp(k) ) ) then
        addmass = addmass + ( x(k1) - maxp(k) ) * c(k1)
        x(k1) = maxp(k)
        whois_pos(k1) = -1
      else
        pos_counter = pos_counter+1;
        whois_pos(pos_counter) = k1;
      endif
      if ( ( x(k1) <= minp(k) ) ) then
        addmass = addmass - ( minp(k) - x(k1) ) * c(k1)
        x(k1) = minp(k)
        whois_neg(k1) = -1
      else
        neg_counter = neg_counter+1;
        whois_neg(neg_counter) = k1;
      endif
    enddo

    weightssum = 0.0d0
    if ( addmass > 0 ) then
      do i2 = 1 , maxIter
        weightssum = 0.0
        do k1 = 1 , pos_counter
          i1 = whois_pos(k1)
          weightssum = weightssum + c(i1)
          al_pos(i1) = maxp(k) - x(i1)
        enddo

        if( ( pos_counter > 0 ) .and. ( addmass > tol_limiter * abs(mass) ) ) then
          do k1 = 1 , pos_counter
            i1 = whois_pos(k1)
            howmuch = addmass / weightssum
            if ( howmuch > al_pos(i1) ) then
              howmuch = al_pos(i1)
              whois_pos(k1) = -1
            endif
```

Then we calculate bring these values to a closest constraint. In purpose to reduce Computations we do this only for the values from whois_pos/neg

```
            addmass = addmass - howmuch * c(i1)
            weightssum = weightssum - c(i1)
            x(i1) = x(i1) + howmuch
          enddo
          neg_counter = pos_counter
          whois_neg = whois_pos
          whois_pos = -1
          pos_counter = 0
          do k1 = 1 , neg_counter
            if ( whois_neg(k1) .ne. -1 ) then
              pos_counter = pos_counter+1
              whois_pos(pos_counter) = whois_neg(k1)
            endif
          enddo
        else
          exit
        endif
      enddo
    else
      do i2 = 1 , maxIter
        weightssum = 0.0
        do k1 = 1 , neg_counter
          i1 = whois_neg(k1)
          weightssum = weightssum + c(i1)
          al_neg(i1) = x(i1) - minp(k)
        enddo

        if ( ( neg_counter > 0 ) .and. ( (-addmass) > tol_limiter * abs(mass) ) ) then
          do k1 = 1 , neg_counter
            i1 = whois_neg(k1)
            howmuch = -addmass / weightssum
            if ( howmuch > al_neg(i1) ) then
              howmuch = al_neg(i1)
              whois_neg(k1) = -1
            endif
            addmass = addmass + howmuch * c(i1)
            weightssum = weightssum - c(i1)
            x(i1) = x(i1) - howmuch
          enddo
          !now sort whois_pos and get a new number for pos_counter
          !here pos_counter and whois_pos serve as temp vars
          pos_counter = neg_counter
          whois_pos = whois_neg
          whois_neg = -1
          neg_counter = 0
          do k1 = 1 , pos_counter
            if ( whois_pos(k1) .ne. -1 ) then
              neg_counter = neg_counter+1
              whois_neg(neg_counter) = whois_pos(k1)
            endif
          enddo
        else
          exit
        endif
      enddo
    endif

    ptens(:,k) = x
  enddo

  do k = 1 , nlev
    ptens(:,k) = ptens(:,k) * dpmass(:,k)
  enddo
end subroutine limiter_optim_iter_full
```

Limiter8 original (indirect addressing)

Sandia National Laboratories

```
subroutine limiter_optim_iter_full(ptens,sphweights,minp,maxp,dpmass)

 ..code ommited

  integer, parameter :: maxiter = 5

  do k = 1 , nlev
    weights(:,k) = sphweights(:) * dpmass(:,k)
    ptens(:,k) = ptens(:,k) / dpmass(:,k)
  enddo

  do k = 1 , nlev
    c = weights(:,k)
    x = ptens(:,k)
    mass = sum(c*x)
    if( (mass / sum(c)) < minp(k) ) then
      minp(k) = mass / sum(c)
    endif
    if( (mass / sum(c)) > maxp(k) ) then
      maxp(k) = mass / sum(c)
    endif

    addmass = 0.0d0
    pos_counter = 0;
    neg_counter = 0;

    do k1 = 1 , np*np
      if ( ( x(k1) >= maxp(k) ) ) then
        addmass = addmass + ( x(k1) - maxp(k) ) * c(k1)
        x(k1) = maxp(k)
        whois_pos(k1) = -1
      else
        pos_counter = pos_counter+1;
        whois_pos(pos_counter) = k1;
      endif
      if ( ( x(k1) <= minp(k) ) ) then
        addmass = addmass - ( minp(k) - x(k1) ) * c(k1)
        x(k1) = minp(k)
        whois_neg(k1) = -1
      else
        neg_counter = neg_counter+1;
        whois_neg(neg_counter) = k1;
      endif
    enddo

    weightssum = 0.0d0
    if ( addmass > 0 ) then
      do i2 = 1 , maxIter
        weightssum = 0.0
        do k1 = 1 , pos_counter
          i1 = whois_pos(k1)
          weightssum = weightssum + c(i1)
          al_pos(i1) = maxp(k) - x(i1)
        enddo

        if( ( pos_counter > 0 ) .and. ( addmass > tol_limiter * abs(mass) ) ) then
          do k1 = 1 , pos_counter
            i1 = whois_pos(k1)
            howmuch = addmass / weightssum
            if ( howmuch > al_pos(i1) ) then
              howmuch = al_pos(i1)
              whois_pos(k1) = -1
            endif
            addmass = addmass - howmuch * c(i1)
            weightssum = weightssum - c(i1)
            x(i1) = x(i1) + howmuch
          enddo
          neg_counter = pos_counter
          whois_neg = whois_pos
          whois_pos = -1
          pos_counter = 0
          do k1 = 1 , neg_counter
            if ( whois_neg(k1) .ne. -1 ) then
              pos_counter = pos_counter+1
              whois_pos(pos_counter) = whois_neg(k1)
            endif
          enddo
        else
          exit
        endif
      enddo
    else
      do i2 = 1 , maxIter
        weightssum = 0.0
        do k1 = 1 , neg_counter
          i1 = whois_neg(k1)
          weightssum = weightssum + c(i1)
          al_neg(i1) = x(i1) - minp(k)
        enddo

        if ( ( neg_counter > 0 ) .and. ( (-addmass) > tol_limiter * abs(mass) ) ) then
          do k1 = 1 , neg_counter
            i1 = whois_neg(k1)
            howmuch = -addmass / weightssum
            if ( howmuch > al_neg(i1) ) then
              howmuch = al_neg(i1)
              whois_neg(k1) = -1
            endif
            addmass = addmass + howmuch * c(i1)
            weightssum = weightssum - c(i1)
            x(i1) = x(i1) - howmuch
          enddo
          !now sort whois_pos and get a new number for pos_counter
          !here pos_counter and whois_pos serve as temp vars
          pos_counter = neg_counter
          whois_pos = whois_neg
          whois_neg = -1
          neg_counter = 0
          do k1 = 1 , pos_counter
            if ( whois_pos(k1) .ne. -1 ) then
              neg_counter = neg_counter+1
              whois_neg(neg_counter) = whois_pos(k1)
            endif
          enddo
        else
          exit
        endif
      enddo
    endif

    ptens(:,k) = x
  enddo

  do k = 1 , nlev
    ptens(:,k) = ptens(:,k) * dpmass(:,k)
  enddo
end subroutine limiter_optim_iter_full
```

This way we introduce some mass change (addmass), so, we redistribute addmass in the way that l2 error is smallest.

Limiter8 new algorithm

Sandia National Laboratories

```
subroutine limiter_optim_iter_full(ptens,sphweights,minp,maxp,dpmass)

  integer, parameter :: maxiter = np*np-2

  do k = 1 , nlev
    c = sphweights(:) * dpmass(:,k)
    x = ptens(:,k)/dpmass(:,k)

    mass = sum(c*x)
    sumc= sum(c)
    if (sumc <= 0 ) CYCLE   ! this should never happen, but if it does, doi

    ! relax constraints to ensure limiter has a solution:
    ! This is only needed if runnign with the SSP CFL>1 or
    ! due to roundoff errors
    if( mass  < minp(k)*sumc ) then
      minp(k) = mass / sumc
    endif
    if( mass > maxp(k)*sumc ) then
      maxp(k) = mass / sumc
    endif

   do iter=1,maxiter

     addmass=0.0d0

     do k1=1,np*np
       if((x(k1)>maxp(k))) then
         addmass=addmass+(x(k1)-maxp(k))*c(k1)
         x(k1)=maxp(k)
       endif
       if((x(k1)<minp(k))) then
         addmass=addmass-(minp(k)-x(k1))*c(k1)
         x(k1)=minp(k)
       endif
     enddo !k1

     if(abs(addmass)<=tol_limiter*abs(mass)) exit

     weightssum=0.0d0
     if(addmass>0)then
      do k1=1,np*np
        if(x(k1)<maxp(k))then
          weightssum=weightssum+c(k1)
        endif
      enddo !k1
      do k1=1,np*np
        if(x(k1)<maxp(k))then
           x(k1)=x(k1)+addmass/weightssum
        endif
      enddo
     else
      do k1=1,np*np
        if(x(k1)>minp(k))then
          weightssum=weightssum+c(k1)
        endif
      enddo
      do k1=1,np*np
        if(x(k1)>minp(k))then
          x(k1)=x(k1)+addmass/weightssum
        endif
      enddo
     endif


   enddo!end of iteration

      ptens(:,k)=x(:)
      k1=k1+1

  enddo

  do k=1,nlev
    ptens(:,k)=ptens(:,k)*dpmass(:,k)
  enddo


end subroutine limiter_optim_iter_full
```

Limiter8 new algorithm

```
subroutine limiter_optim_iter_full(ptens,sphweights,minp,maxp,dpmass)

  integer, parameter :: maxiter = np*np-2

  do k = 1 , nlev
    c = sphweights(:) * dpmass(:,k)
    x = ptens(:,k)/dpmass(:,k)

    mass = sum(c*x)
    sumc= sum(c)
    if (sumc <= 0 ) CYCLE   ! this should never happen, but if it does, do

    ! relax constraints to ensure limiter has a solution:
    ! This is only needed if runnign with the SSP CFL>1 or
    ! due to roundoff errors
    if( mass  < minp(k)*sumc ) then
      minp(k) = mass / sumc
    endif
    if( mass > maxp(k)*sumc ) then
      maxp(k) = mass / sumc
    endif

   do iter=1,maxiter

     addmass=0.0d0

     do k1=1,np*np
       if((x(k1)>maxp(k))) then
         addmass=addmass+(x(k1)-maxp(k))*c(k1)
         x(k1)=maxp(k)
       endif
       if((x(k1)<minp(k))) then
         addmass=addmass-(minp(k)-x(k1))*c(k1)
         x(k1)=minp(k)
       endif
     enddo !k1

     if(abs(addmass)<=tol_limiter*abs(mass)) exit

     weightssum=0.0d0
     if(addmass>0)then
      do k1=1,np*np
        if(x(k1)<maxp(k))then
          weightssum=weightssum+c(k1)
        endif
      enddo !k1
      do k1=1,np*np
        if(x(k1)<maxp(k))then
           x(k1)=x(k1)+addmass/weightssum
        endif
      enddo
     else
      do k1=1,np*np
        if(x(k1)>minp(k))then
          weightssum=weightssum+c(k1)
        endif
      enddo
      do k1=1,np*np
        if(x(k1)>minp(k))then
          x(k1)=x(k1)+addmass/weightssum
        endif
      enddo
     endif


   enddo!end of iteration

      ptens(:,k)=x(:)
      k1=k1+1

  enddo

  do k=1,nlev
    ptens(:,k)=ptens(:,k)*dpmass(:,k)
  enddo


end subroutine limiter_optim_iter_full
```

No indirect address: more computations, but less memory access

Limiter 8 optimization

Sandia National Laboratories

```
if ( limiter_option == 8 ) then
  do k = 1 , nlev  ! Loop index added (AAM)
    ! UN-DSS'ed dp at timelevel n0+1:
    dp_star(:,:,k) = dp(:,:,k) - dt * elem(ie)%derived%divdp(:,:,k)
    if ( nu_p > 0 .and. rhs_viss /= 0 ) then
      ! add contribution from UN-DSS'ed PS dissipation
!      dpdiss(:,:) = ( hvcoord%hybi(k+1) - hvcoord%hybi(k) ) * elem(ie)%derived%psdiss_biharmonic(:,:)
      dpdiss(:,:) = elem(ie)%derived%dpdiss_biharmonic(:,:,k)
      dp_star(:,:,k) = dp_star(:,:,k) - rhs_viss * dt * nu_q * dpdiss(:,:) / elem(ie)%spheremp(:,:)
    endif
  enddo
  ! apply limiter to Q = Qtens / dp_star
  call limiter_optim_iter_full( Qtens(:,:,:) , elem(ie)%spheremp(:,:) , qmin(:,q,ie) , &
                                qmax(:,q,ie) , dp_star(:,:,:) )
endif
```

```
  subroutine limiter_optim_iter_full(ptens,sphweights,minp,maxp,dpmass)

    integer, parameter :: maxiter = np*np-2

    do k = 1 , nlev
      c = sphweights(:) * dpmass(:,k)
      x = ptens(:,k)/dpmass(:,k)
```

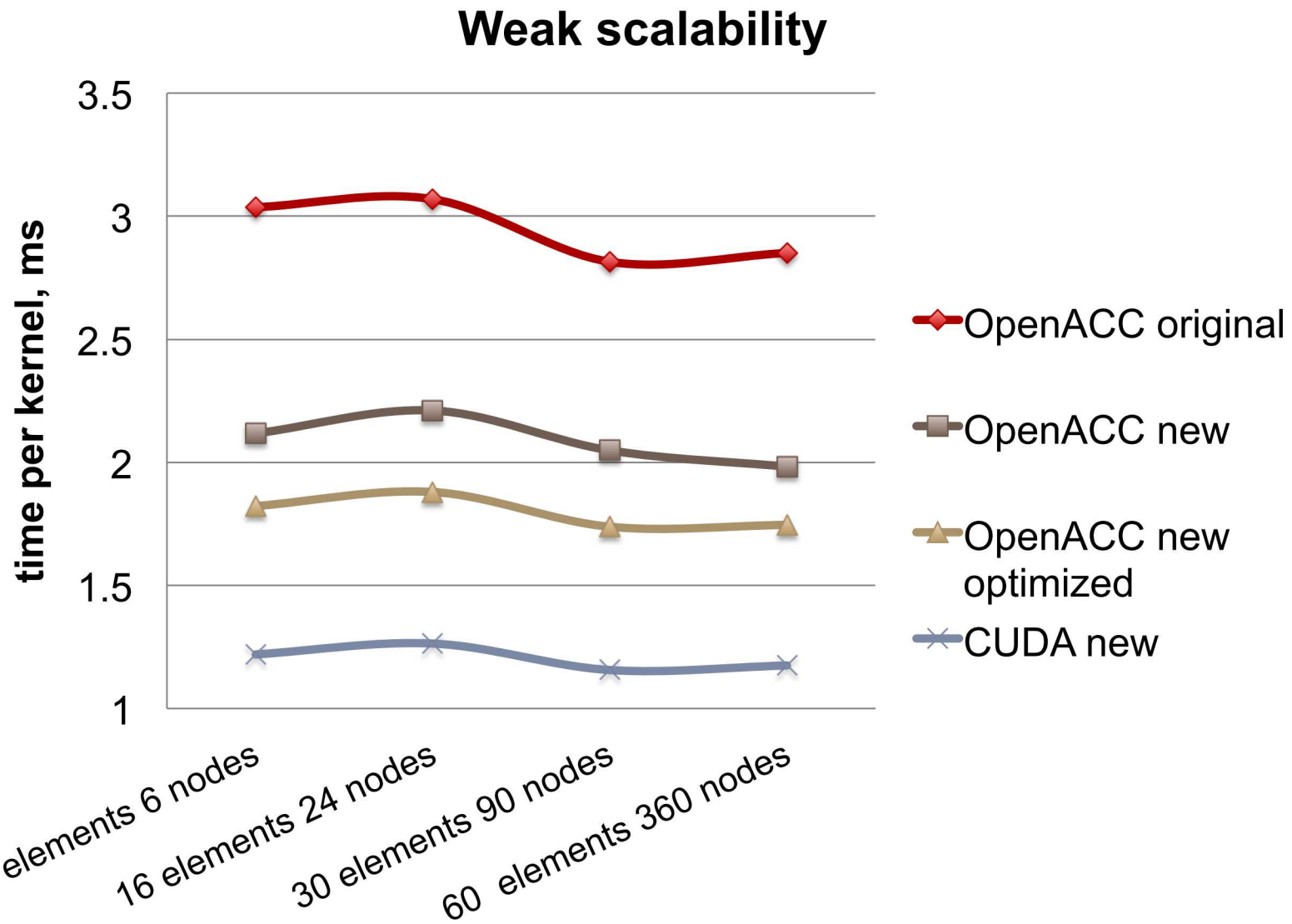
Limiter8 new algorithm optimized

Sandia National Laboratories

```
if ( limiter_option == 8 ) then
  do k = 1 , nlev  ! Loop index added (AAM)
    ! UN-DSS'ed dp at timelevel n0+1:
    dp_star(:,:,k) = dp(:,:,k) - dt * elem(ie)%derived%divdp(:,:,k)
    if ( nu_p > 0 .and. rhs_viss /= 0 ) then
      ! add contribution from UN-DSS'ed PS dissipation
!      dpdiss(:,:) = ( hvcoord%hybi(k+1) - hvcoord%hybi(k) ) * elem(ie)%derived%psdiss_biharmonic(:,:)
      dpdiss(:,:) = elem(ie)%derived%dpdiss_biharmonic(:,:,k)
      dp_star(:,:,k) = dp_star(:,:,k) - rhs_viss * dt * nu_q * dpdiss(:,:) / elem(ie)%spheremp(:,:)
      dp_stari(:,:,k) = 1.0/dp_star(:,:,k)
    endif
  enddo
  ! apply limiter to Q = Qtens / dp_star
  call limiter_optim_iter_full( Qtens(:,:,:) , elem(ie)%spheremp(:,:) , qmin(:,q,ie) , &
                                qmax(:,q,ie) , dp_star(:,:,:), dp_stari(:,:,:) )
endif
```

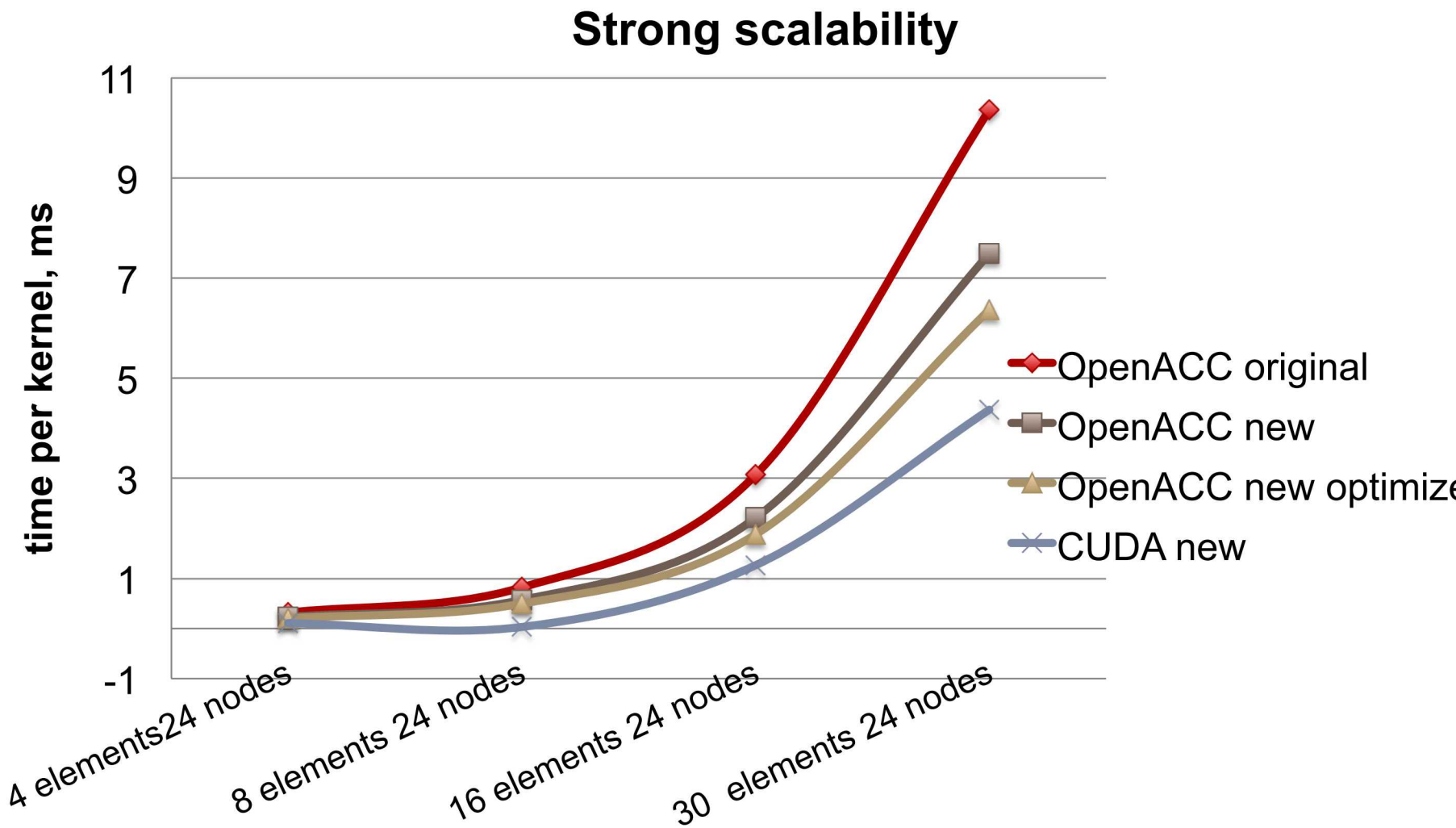
```
  subroutine limiter_optim_iter_full(ptens,sphweights,minp,maxp,dpmass, dpmassi)

    integer, parameter :: maxiter = np*np-2

    do k = 1 , nlev
      c = sphweights(:) * dpmass(:,k)
      x = ptens(:,k)*dpmassi(:,k)
```

Performance evaluation results



Performance evaluation results

Sandia National Laboratories



Conclusion

Sandia National Laboratories

- Limiter 8 has been ported to GPU with CUDA and OpenACC -> tracer advections can be 100% computed on the GPUs
- Code optimizations are necessary
- OpenACC gives reasonable performance
- CUDA is still faster than OpenACC