SOLPS-ITER

Gaurav Saxena (HLST)
David Vicente Dorca (Head, User Support)

BSC-ACH Meeting | 27th Nov. 2024

This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 — EUROfusion). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

Presentation Code

RGB Presentation Color Code

Bad / Alarm

Good / Improvement

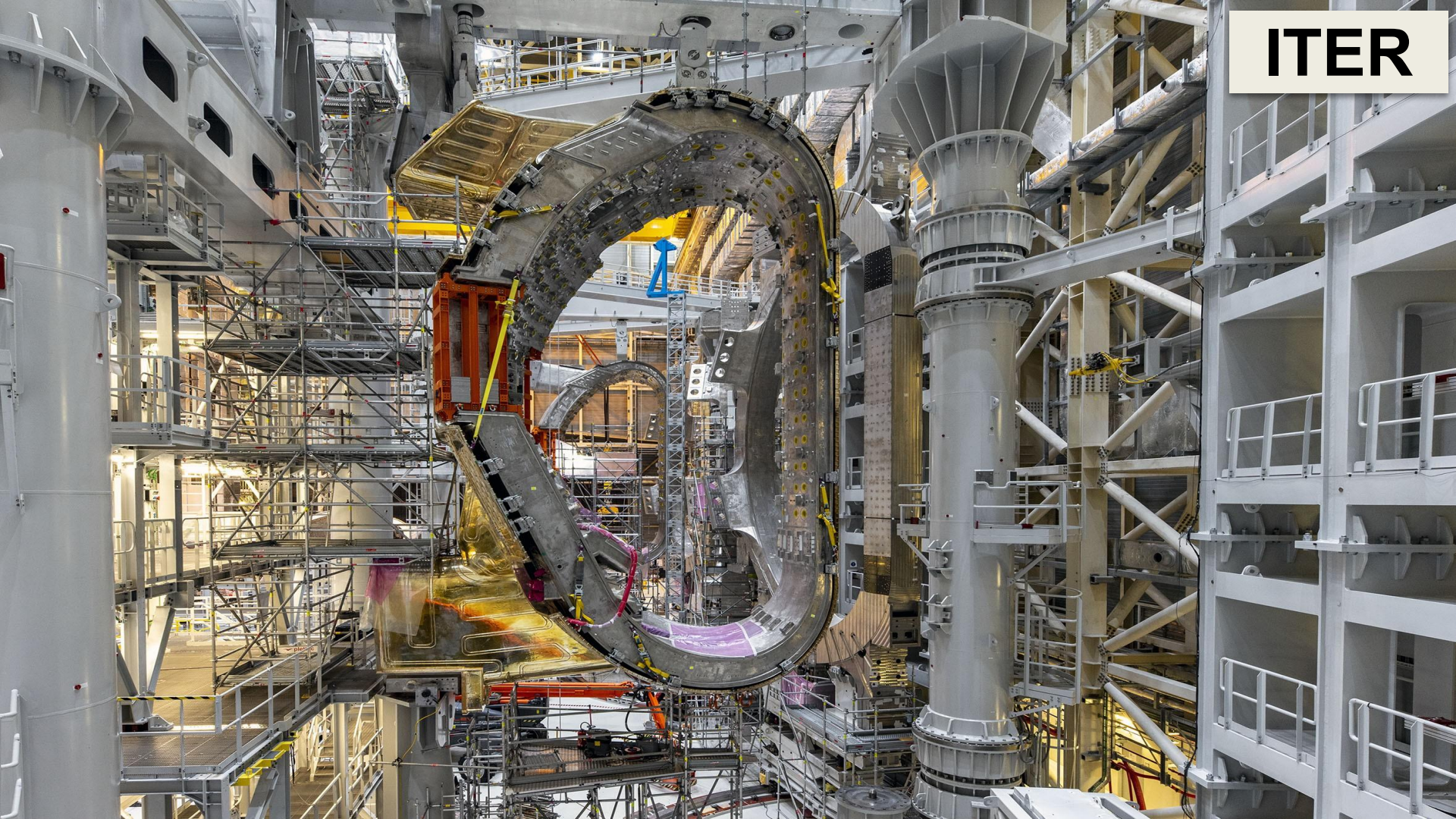
Keyword / Code

File name (can ignore)

What is SOLPS-ITER ?

SOLPS- ITER

- **S**crape-**O**ff **L**ayer **P**lasma **S**imulation (boundary plasma)
- Monte Carlo code **Eirene** (MPI parallelized)
- **B2.5** Plasma Fluid solver (OpenMP parallelized) - Improve scaling (Our Focus)
- Fortran 77 (fixed form), Fortran 90
- ITER = International Thermonuclear Experimental Reactor


ITER

Recap (2023)

(Very) Brief Recap (on MN4 in 2023)

- Re-wrote get_num_threads() function (5.6 - 17% time reduction at 12 & 48 threads, respectively).
- Added: -xHost -align-array-64byte -qopt-zmm-usage=high (Intel 18.x), enabled KMP_AFFINITY (≈ 36% time reduction at 48 threads)
- Explored “hot cache” effect in b2xpfe.F by removing n > 16384 condition in parallel sfill() function and removing indirect accesses by a pattern (24.5% time reduction for this loop but retracted as pattern not general)
- Removed unneeded critical sections (4.45x Speed-up Vs 1.23x original)
- Loop in fka() vectorized using !$OMP SIMD (as -fp-model=precise prevented vectorization), copied 3rd dimensions of 2 arrays to auxiliary array for unit strides (new function fka_new()).

ITER_2171_D+He+Be+Ne Scaling

'b2mndr_ntim' '100', standalone OpenMP, **21 species**, #ifndef NO_OPENMP_B2SIFRTF, *critical sections removed*

Threads	Before Time (sec)	Before S_p	After Time (sec)	After S_p	Contig Time	Contig S_p
1	957	1.00	937	1.00	890	1.00
2	596	1.60	577	1.62	555	1.60
4	388	2.46	373	2.51	367	2.42
12	251	3.80	241	3.88	242	3.67
24	210	4.55	203	4.61	208	4.27
48	215	4.45	208	4.50	211	4.21

- **After** vectorization of loop in function fka() [although non-unit stride]
- ACTUAL speed-ups: **957/t_x** i.e. 1.72, 2.60, 3.95, 4.60 and 4.53

Overview 2024

2024 overview

- Continued working in fka_new() subroutine
- Concentrated on Zhdanov closure scheme
- Identified multiple hotspots: b2sifr_ , b2tfnb, and ma30bd
- … (specifically) serial bottlenecks b2txcy, b2stbr_phys and b2sihs
- Migration to MN5
- Assistance for porting code to *Red-Hat Linux* nodes at ITER.
- Assistance for removing extremely critical bugs when using gfortran.
- ⭐ All changes incorporated in SOLPS-ITER official release *3.0.9/3.1.1*.

Work in 2024

Loop inside fka_new()

```
!$OMP SIMD REDUCTION(+:fka_new)                                    0.682s
      do r = 0, ns-1                                               46.319s
        fka_new = fka_new + rz2_temp(index+r+1)*                   3.077s
     &             na_temp(index+r+1)* sqrt(mp)*                   46.309s
     &            sqrt(am(a)*am(r)/(am(a)+am(r)))
      enddo
      fka_new = fka_new*rz2_temp(index+a+1)                        4.400s
      return
      end function fka_new                                         5.252s
```

(1) sqrt(am(isb)*am(is)/(am(isb)+am(is)) is calculated again and again, takes ~ 46/109 seconds

(2) Calculate all combinations *outside* fka_new like:

```
do j = 0, ns-1
   do i = 0, ns-1
      am_sqrt(i,j) = sqrt(am(j)*am(i)/(am(j)+am(i))
   end do
end do
```

Look up table !

Modifying fka_new()

```
      function fka_new(ix, iy, a, rz2_temp, na_temp)
      implicit none
      integer, intent(in) :: ix, iy, a
      real (kind=R8) :: fka_new
      real (kind=R8) :: rz2_temp(1:ns*(nx+2)*(ny+2))
      real (kind=R8) ::  na_temp(1:ns*(nx+2)*(ny+2))
      integer r, index
      fka_new = 0.0_R8
      index = (ix+1)*ns+(iy+1)*ns*(nx+2)
!$OMP SIMD REDUCTION(+:fka_new)
      do r = 0, ns-1
        fka_new = fka_new + rz2_temp(index+r+1)*
     &        na_temp(index+r+1)* sqrt(mp)*
     &            sqrt(am(a)*am(r)/(am(a)+am(r)))
      enddo
      fka_new = fka_new*rz2_temp(index+a+1)
      return
      end function fka_new
```

fka_new(ix,iy,a,rz2_temp,na_temp,am_sqrt)

real (kind=R8) :: am_sqrt(0:ns-1,0:ns-1)

. . .

. . .

*fka_new = fka_new + … * am_sqrt(r,a)*

. . .

. . .

(1) Removes repeated calculation
(2) 2 memory accesses → 1 access

Before and After - I

```
!$OMP SIMD REDUCTION(+:fka_new)                                    
      do r = 0, ns-1                                               
        fka_new = fka_new + rz2_temp(index+r+1)*                   
     &             na_temp(index+r+1)* sqrt(mp)*                   
     &           sqrt(am(a)*am(r)/(am(a)+am(r)))                   
      enddo                                                        
      fka_new = fka_new*rz2_temp(index+a+1)                        
      return                                                       
      end function fka_new                                         
```

0.682s	
46.319s	
3.077s	
46.309s	
4.400s	
5.252s	

Before: 46.309 sec

702	!$OMP SIMD REDUCTION(+:fka_new_opt)		
703	do r = 0, ns-1	0.682s	5,922,000,000
704	fka_new_opt = fka_new_opt + rz2_temp(index+r+1)*	14.033s	79,569,000,000
705	& na_temp(index+r+1)* sqrt(mp)*	8.309s	50,316,000,000
706	& am_sqrt(r,a)	0.792s	6,006,000,000
707	enddo		
708	fka_new_opt = fka_new_opt*rz2_temp(index+a+1)	2.135s	10,752,000,000
709	return		
710	end function fka_new_opt	3.538s	19,887,000,000

After: 0.792 sec

Before and After - II

Grouping: Function / Call Stack

Function / Call Stack	CPU Time — Effective Time by Utilization (Idle Poor Ok Ideal Over)	Instructions Retired	CPI Rate
b2sifrtf_IP_fka_new_	109.727s	372,729,000,000	0.628
ma30bd	102.540s	536,277,000,000	0.403
b2tfnb	100.606s	405,888,000,000	0.506
b2sifrtf	95.454s	371,742,000,000	0.527
__intel_skx_avx512_memset	83.265s	10,962,000,000	15.757
ma30bdcopy3	74.996s	328,545,000,000	0.473
b2sifrtf_ompparallel_for@447	40.535s	100,170,000,000	0.813
ma28dd	34.942s	86,037,000,000	0.855
ma28ddcopy3	22.423s	51,681,000,000	0.902
b2sifrtf_ompparallel_for@279	21.941s	114,345,000,000	0.387
_f90_reduction_init_array	17.912s	15,540,000,000	2.320
sfill_ompparallel_for@41	16.699s	3,234,000,000	11.156
_f90_reduction_final_strided	16.338s	56,658,000,000	0.591

fka_new → Before: 109.73 sec
fka_new_opt→ After: 34.86 sec
But …
b2sifrtf → Before: 95.45 sec
b2sifrtf → After: 99.203

Net gain = 70 sec

Function / Call Stack	CPU Time — Effective Time by Utilization (Idle Poor Ok Ideal Over)	Instructions Retired	CPI Rate
_INTERNAL_25_____src_kmp_barrier_cpp_2e	0s	2,115,624,000,000	3.600
ma30bd	103.052s	539,574,000,000	0.408
b2tfnb	100.977s	404,817,000,000	0.508
b2sifrtf	99.203s	383,502,000,000	0.532
__intel_skx_avx512_memset	84.538s	12,096,000,000	14.309
ma30bdcopy3	74.304s	329,553,000,000	0.479
b2sifrtf_ompparallel_for@455	42.941s	119,574,000,000	0.757
_INTERNAL_25_____src_kmp_barrier_cpp_2e	0s	22,176,000,000	3.505
ma28dd	36.305s	84,504,000,000	0.873
b2sifrtf_IP_fka_new_opt_	34.862s	214,788,000,000	0.350

ITER_2171_D+He+Be+Ne Scaling

'b2mndr_ntim' '100', standalone OpenMP, 21 species, #ifndef NO_OPENMP_B2SIFRTF, *critical sections removed*

Threads	Before Time (sec)	Before S_p -20	After Time (sec)	After S_p -47	Contig Time	Contig S_p -70	Sqrt Opt Time	Sqrt Opt S_p
1	957	1.00	937	1.00	890	1.00	820	1.00
2	596	1.60	577	1.62	555	1.60	513	1.6
4	388	2.46	373	2.51	367	2.42	345	2.37
12	251	3.80	241	3.88	242	3.67	232	3.53
24	210	4.55	203	4.61	208	4.27	203	4.03
48	215	4.45	208	4.50	211	4.21	210	3.90

(1) **After** vectorization of loop in function fka() [although non-unit stride]
(2) (1) + **Contiguous** stride optimization
(3) (1) + (2) + **Square Root** optimization

2024

Zhdanov Closure scheme

Zhdanov scheme: ITER_Be-W_D+T+He+Ne/run_Zhdanov

- 98 species (quite high)
- In run_Zhdanov directory, some switches in b2mn.dat
  - b2mndr_ntim ‘50’
  - b2mndr_elapsed ‘0.0’
  - b2mndr_eirene ‘0’
- Hotspots (1 thread):
  - b2sifr_ = 31.36 sec
  - b2tfnb = 28.28 sec
  - ma30bd = 16.86 sec

b2sifr_() in b2sifr_.F

```
 do is = 0, ns-1
..consider distinct cases
  if (is.ne.isb.and.
&    (.not.is_neutral(is)).and.(.not.is_neutral(isb))) then
 ..contribution from friction between charged species
   am_sqrt = sqrt(am(isb)*am(is)/(am(isb)+am(is)))
   do iy = 0, ny-1
     do ix = 0, nx-1
      coef = phm0*(lnlam(ix,iy)*inv_ctaup)*rz2(ix,iy,isb)*
&            rz2(ix,iy,is)*mp*am_sqrt
```

Calculate outside loop.

Time of "*statement*" reduces from:

12 sec → 5.66 sec (i.e. ≈ 50%)

Single thread b2sifr_() i.e. complete subroutine timings

- b2sifr_() previous time: 31.36 sec
- b2sifr_() current time: 26.12 sec
- Reduction : (31.36 - 26.12)/31.36 ≈ 16%

b2sifr_() loop not vectorized → Vectorized

```
 do is = 0, ns-1
..consider distinct cases
  if (is.ne.isb.and.
&    (.not.is_neutral(is)).and.(.not.is_neutral(isb))) then
 ..contribution from friction between charged species
   am_sqrt = sqrt(am(isb)*am(is)/(am(isb)+am(is)))
   do iy = 0, ny-1
     do ix = 0, nx-1
      coef = phm0*(lnlam(ix,iy)*inv_ctaup)*rz2(ix,iy,isb)*
&           rz2(ix,iy,is)*mp*am_sqrt
      if(leftix(ix,iy).ne.-2 .and. rightix(ix,iy).ne.nx+1) then
       t0 = na(ix,iy,isb)*na(ix,iy,is)*vti32(ix,iy)
       if (styl0.eq.0) then
        smbch(ix,iy,0) = smbch(ix,iy,0)+coef*t0*ua(ix,iy,is)
        smbch(ix,iy,1) = smbch(ix,iy,1)-coef*t0
       else if (styl0.eq.1) then
        smbch(ix,iy,0) = smbch(ix,iy,0)+coef*t0*ua(ix,iy,is)
        smbch(ix,iy,3) = smbch(ix,iy,3)-coef*t0/roxa(ix,iy,isb)
       else if (styl0.eq.2) then
        smbch(ix,iy,2) = smbch(ix,iy,2)+coef*t0*ua(ix,iy,is)/
&                        roxa(ix,iy,isb)
        smbch(ix,iy,3) = smbch(ix,iy,3)-coef*t0/
&                        roxa(ix,iy,isb)
       endif
       smfrb(ix,iy) = smfrb(ix,iy) +
&       coef*t0*(ua(ix,iy,is)-ua(ix,iy,isb))
      endif
     enddo
   enddo
  else
 ..contribution from friction involving neutral species
   (We ignore such a term, except for charge exchange friction
   between H0 and H1. That term is calculated via the atomic
   physics routines.)
  endif
 enddo
```

```
..loop over all other species
 do is = 0, ns-1
..consider distinct cases
  if (is.ne.isb.and.
&    (.not.is_neutral(is)).and.(.not.is_neutral(isb))) then
 ..contribution from friction between charged species
   am_sqrt = sqrt(am(isb)*am(is)/(am(isb)+am(is)))

    if (styl0.eq.0) then
     do iy = 0, ny-1
     !$OMP SIMD
       do ix = 0, nx-1
        a_coef(ix,iy) = phm0*(lnlam(ix,iy)*inv_ctaup)*
&            rz2(ix,iy,isb)*
&            rz2(ix,iy,is)*mp*am_sqrt
       if(leftix(ix,iy).ne.-2 .and. rightix(ix,iy).ne.nx+1) then
        a_t0(ix,iy) = na(ix,iy,isb)*na(ix,iy,is)*vti32(ix,iy)

        smbch(ix,iy,0) = smbch(ix,iy,0)+a_coef(ix,iy)*
&                        a_t0(ix,iy)*ua(ix,iy,is)
        smbch(ix,iy,1) = smbch(ix,iy,1)-a_coef(ix,iy)*a_t0(ix,iy)

        smfrb(ix,iy) = smfrb(ix,iy) +
&       a_coef(ix,iy)*a_t0(ix,iy)*(ua(ix,iy,is)-ua(ix,iy,isb))
       endif
      enddo
     enddo
    endif
```

b2sifr_() vectorization attempt

- Similar code for styl0.eq.1 (loop 2) and styl0.eq.2 (loop 3)
- Loop is vectorized !
- Timing (sec) for b2sifr_():

  31.36 → 26.12 → 7.26

- Time Reduction b2sifr_(): (31.36 - 7.26)/31.36 x 100 = 76.85%
- roxa() in loops 2 and 3 is a function ! (Loop with function call never a candidate for vectorization)

Total run time ITER_Be*/run_Zhdanov (b2mndr_ntim '100')

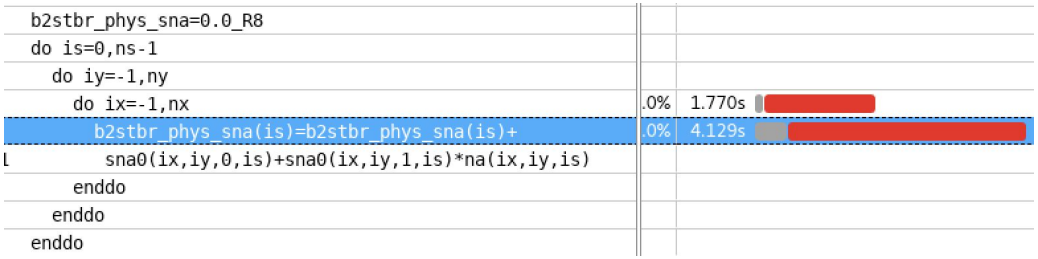
Threads	Time (sec)	S_p -25	Time(sec) b2sifr_() OPT	S_p b2sifr_() OPT
1	247	1	222	1
2	170	1.45	157	1.41
4	133	1.85	125	1.78
8	113	2.19	108	2.05
16	104	2.45	101	2.20
24	101	2.45	99	2.24
48	101	2.45	97	2.29

Serial bottlenecks

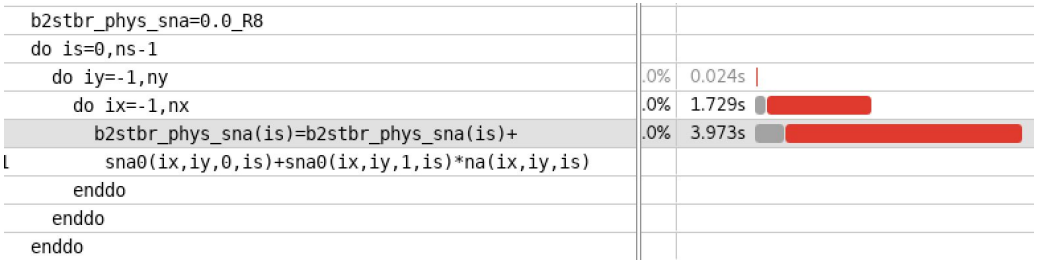
- No parallel part in b2txcy, b2stbr_phys and b2sihs
- ***Now*** parallelized 4 loops in b2txcy
  - b2txcy() Speed-up at 24 threads = S_1/S_P = 12.442/0.72 = 17.28x
- Function call hy1() in b2txcy() loops prevents vectorization
  - Declaring function as pure *does not inline it*.
  - !DIR$ ATTRIBUTES FORCEINLINE :: hy1 - does not forcefully inline hy1() [option only for Intel Fortran Compiler]
  - [Later] solved using -ipo compiler option

b2stbr_phys() subroutine in b2stbr_phys.F

- Total time for b2stbr_phys() 11.926 sec when b2mndr_ntim '100'.



Loop: 331 - 339,
approx 5.8 secs



Loop: 517 - 524
approx 5.6 secs

- Total 4 instances, only 2 within another do loop
- 2 instances outside outer do loop are NOT time intensive
- 2 instances parallelized, b2stbr_phys() takes 0.69 secs (24 threads) i.e. S = 11.93/0.69 = 17.28x (but vectorization broken !)

* Add !$OMP SIMD to innermost loop

```
525         b2stbr_phys_sna=0.0_R8
526  !$OMP PARALLEL DO COLLAPSE(3)
527  !$OMP& DEFAULT(NONE)
528  !$OMP& SHARED(ns,ny,nx,sna0,na)
529  !$OMP& PRIVATE(is,iy,ix)
530  !$OMP& REDUCTION(+:b2stbr_phys_sna)
531         do is=0,ns-1
532           do iy=-1,ny
533  !$OMP SIMD
534             do ix=-1,nx
535               b2stbr_phys_sna(is)=b2stbr_phys_sna(is)+
536       1        sna0(ix,iy,0,is)+sna0(ix,iy,1,is)*na(ix,iy,is)
537             enddo
538           enddo
539         enddo
540  !$OMP END PARALLEL DO
```

```
332         b2stbr_phys_sna=0.0_R8
333  !$OMP PARALLEL DO COLLAPSE(3)
334  !$OMP& DEFAULT(NONE)
335  !$OMP& SHARED(ns,ny,nx,sna0,na)
336  !$OMP& PRIVATE(is,iy,ix)
337  !$OMP& REDUCTION(+:b2stbr_phys_sna)
338         do is=0,ns-1
339           do iy=-1,ny
340  !$OMP SIMD
341             do ix=-1,nx
342               b2stbr_phys_sna(is)=b2stbr_phys_sna(is)+
343       1        sna0(ix,iy,0,is)+sna0(ix,iy,1,is)*na(ix,iy,is)
344             enddo
345           enddo
346         enddo
347  !$OMP END PARALLEL DO
```

***Note:** **MN5** ifort produced compilation error with !$OMP SIMD,

- Canonical structure of loop disrupted
- Later can use COLLAPSE(2)
- Then use !$OMP SIMD REDUCTION(+:b2stbr_phys_sna) in the innermost loop.

Total run time ITER_Be/run_Zhdanov with b2mndr_ntim '100'

Threads	Time (sec)			$S_{p\ (3rd\ column)}$
	1	2	3	
1	424	442	416	1
2	285	289	276	1.50
4	216	217	207	2
8	182	177	173	2.40
16	166	158	158	2.63
24	164	154	155	2.68
48	160	152	152	2.73

smin() and smax()

- Both serial subroutines
- Not vectorized due to -fp-model=precise
- Parallelization increases time rather than decrease time !
- Vectorized by adding !$OMP SIMD combined with reduction clause
- smin() decrease at ‘b2mndr_ntim’ 100, (8.71-2.71)/8.71 x 100 = 68.89%
- smax() decrease at ‘b2mndr_ntim 100’, (4.58 - 1.14)/4.58 x 100 = 75.11%

Total run time ITER_Be*/run_Zhdanov with b2mndr_ntim ‘100’

Threads	Time (sec)		S_p
	prev	now	
1	416	408	1
2	276	269	1.52
4	207	201	2.02
8	173	168	2.42
16	158	149	2.73
24	155	147	2.77
48	152	144	2.83

Migration to Marenostrum5

Migrating to Marenostrum 5 (MN5)

- Compiled with oneapi/2024.1 but since libcilkrtl.so.* missing, switched to oneapi/2023.2
- MSCL compiled using ifx/ifort, a function prototype in date2.c changed.
- The core files compiled using ifort
- Very consistent 38 - 40% time reduction for corresponding core (thread) count between MN4 and MN5 (when thread affinity respected).
- Turned on -ipo flag to aid inlining of small functions (which help with vectorization as well) → time reduction
- Loops in b2sihs_.F parallelized

Developers Experiments

Comparison Table (Multiple Examples)

[Intermediate performance improvements courtesy ***Xavier Bonnin***]

Test Case	AUG_16151_D+C+He standalone 100 timesteps		AUG_16151_D+C+He coupled 10 timesteps		ITER_535_D+He+Ar standalone 100 timesteps		ITER_2171_D+He+Be+Ne standalone 100 timesteps	
	1 thread	6 threads	1 thread	4 threads	1 thread	6 threads	1 thread	7 threads
After 2023	10.298s	7.615s (35%)	8m58s	4m09s (116%)	2m43s	1m24s (94%)	16m13s	6m12s (162%)
In 2024	6.240s	3.508s (78%)	5m30s	2m30s (120%)	1m48s	0m45s (140%)	10m22s	3m17s (216%)
Improv.	65%	117%	63%	66%	32%	86%	56%	88%

Further …

- Provided assistance for porting code to *Red-Hat Linux* nodes in ITER.
- Provided assistance for removing critical bugs when using gfortran.
- All changes incorporated in release *3.0.9/3.1.1.*
- Presented work in SOLPS-ITER *61st, 62nd, 63rd*, and *66th* User-Forum Meetings.

Many thanks to Xavier Bonnin & David Coster. We are humbled and honored by their kind words and encouragement 🙏*.*

Thank you !
For any questions:

Gaurav Saxena

(gaurav.saxena@bsc.es)

&

David Vicente Dorca

(david.vicente@bsc.es)