



KNOSOS and SOLPS-ITER

Gaurav Saxena & David Vicente Dorca



BSC-ACH Meeting | 16th Nov. 2023



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.

RGB Presentation Color Code



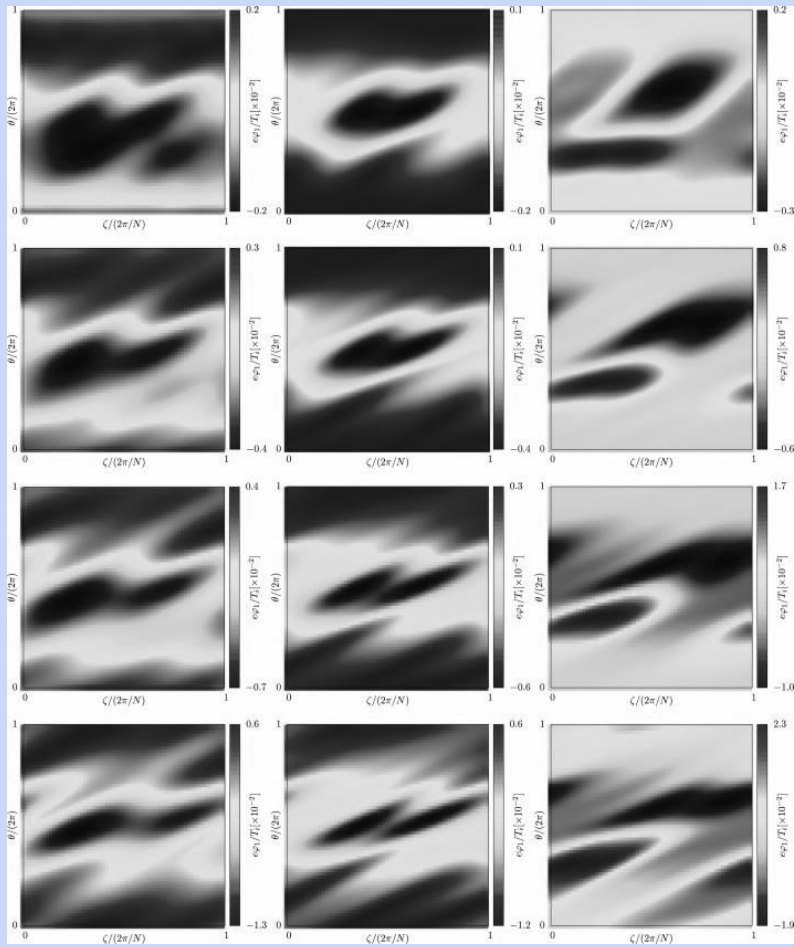
Bad / Alarm



Good / Improvement



Keyword / Code



KNOSOS

(Ricard Zarco Badia, Helena Vella Beltran, Laia Julio Plana, David Vicente, Gaurav Saxena)

KNOSOS

- **Ki**Netic **O**rbit-averaging **S**olver for **S**tellarators
- Related to Plasma in 3-D magnetic confinement
- **Multiple particle species** being solved on their **own surface**
- Written in **Fortran90**, **MPI** parallelized
- Open source, available at: <https://github.com/joseluisvelasco/KNOSOS>
- **Problems**: One surface-One MPI, a Notorious process taking much longer, parallel efficiency, others.

Basics: Division by Zero

- Pass `-fp_trap` flag to **PETSc** (Portable Extension Toolkit for Scientific Computing)
- `srun ./knosos.x -fp_trap`
- Run DDT: `ddt ./knosos.x -fp_trap` to locate *exact error point*
 - All 22 Processes stop in `fill_3dgrid (configuration.f90:1617)` with signal `SIGFPE` (Arithmetic exception).

NaN in $[A][x] = [vecb]$ MPI Rank 15

- Linear system of equations $Ax = b$...
- Code equivalent $[A][vecx] = [vecb]$
- But $vecb$ contains a NaN

```
CALL VecSetValues(vecb,npoint_ps,indx,c,INSERT_VALUES,ierr)
CALL VecAssemblyBegin(vecb,ierr)
CALL VecAssemblyEnd(vecb,ierr)
!Solve
CALL CPU_TIME(tstart2)

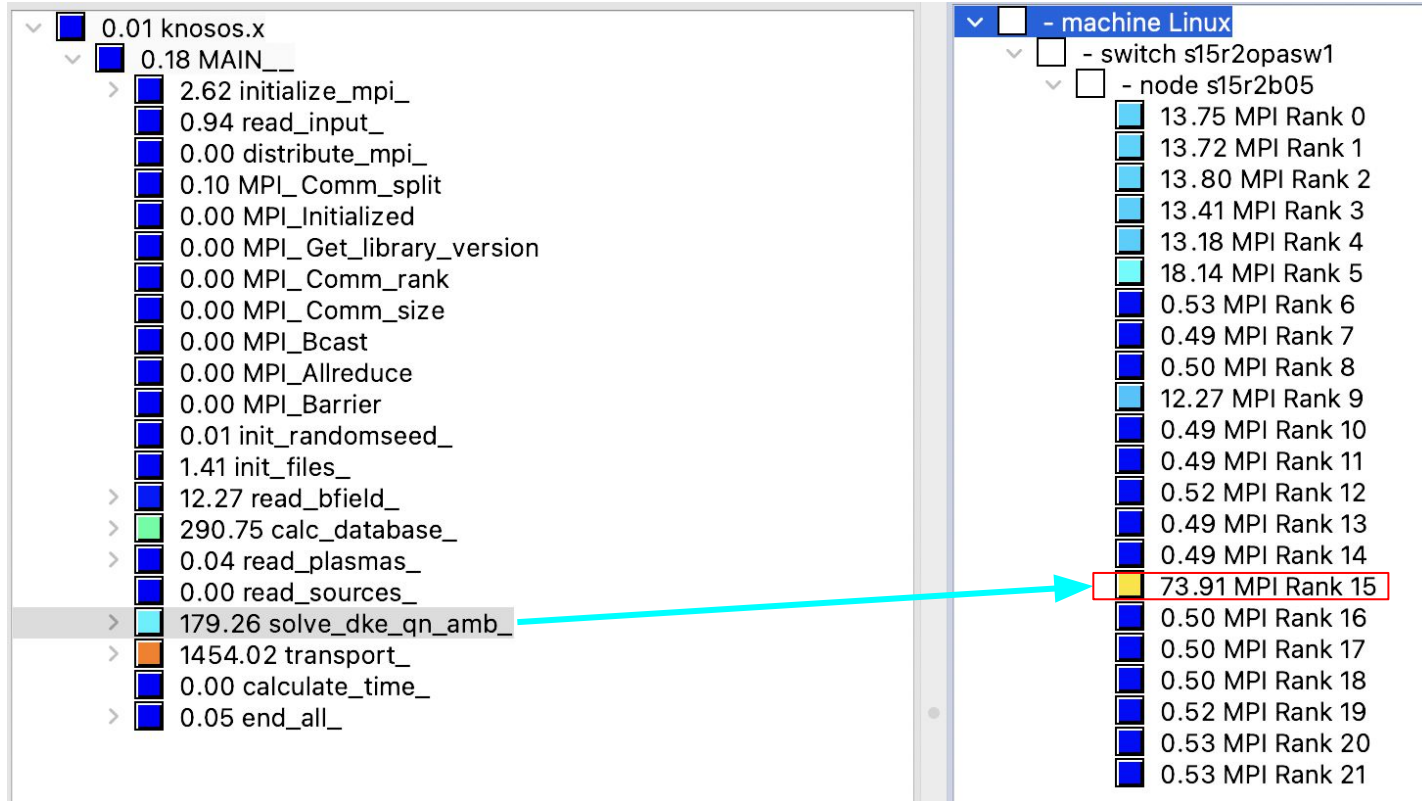
!CALL VecRestoreArrayReadF90(x,xx_v,ierr)
CALL KSPSolve(ksp,vecb,vecx,ierr)
```

```
serr
xx_v
(1)
(2)
(3)
(4)
(5)
(6)
(7)
```

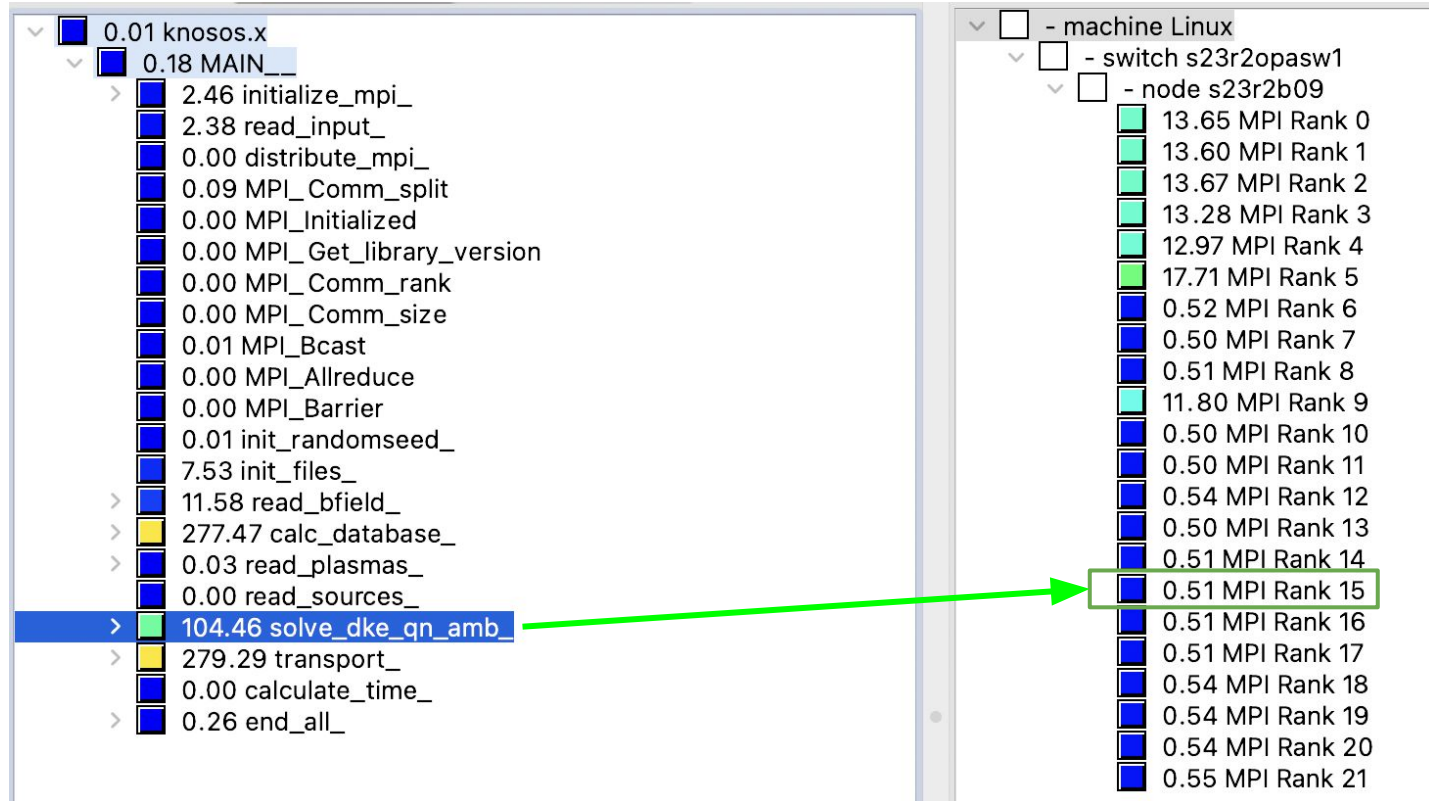
```
001|000|000|000|000|000|000|000|00...
(0, 0.041105771488678021, -nan(0...
0
0.041105771488678021
-nan(0x8000000000000000)
0.044991712773031232
-2.1565856624709245
-0.097111904168478161
0.047302123011333047
```

- $vecb$ has type Vec of PETSc
- Convert to simple array using $VecGetArrayReadF90(vecb,xx_v,ierr)$ where xx_v is $PetscScalar$, pointer $:: xx_v(:)$

Before replacing NaN at `vecb(3)` with random value

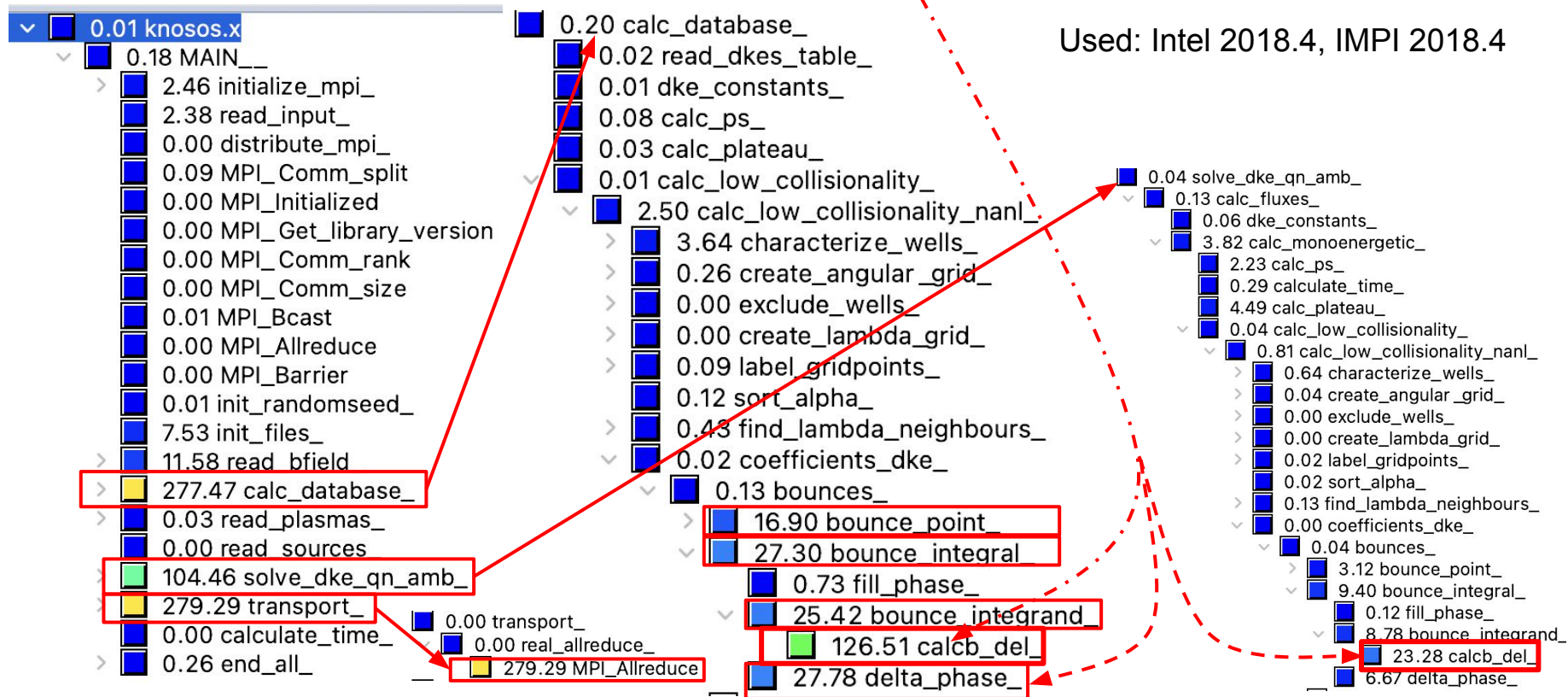


After replacing NaN at `vecb(3)` with random value



HotSpots

Used: Intel 2018.4, IMPI 2018.4



Attempt Optimization - 1(a), CALCB_DEL in coefficients.f90

```

DO nm=1,Nnm
  n=np(nm)
  m=mp(nm)
  IF(STELL_ANTISYMMETRIC) THEN


    IF(flag.EQ.0.OR.flag.EQ.2) THEN
      ! B_0=B_0+bnmc0(nm)*cosnm(nm)+bnms0(nm)*sinnm(nm) <---- original
      GS_TEMP_1=GS_TEMP_1+bnmc0(nm)*cosnm(nm)+bnms0(nm)*sinnm(nm)
      !IF(flagB1) B_1=B_1+bnmc1(nm)*cosnm(nm)+bnms1(nm)*sinnm(nm) <----original
      IF(flagB1) GS_TEMP_4=GS_TEMP_4+bnmc1(nm)*cosnm(nm)+bnms1(nm)*sinnm(nm)
      !IF(TANG_VM.AND.flag.GT.1) dBdpsi=dBdpsi+dbnmcdpsi(nm)*cosnm(nm)+dbnmsdpsi(nm)*sinnm(nm) <----original
      IF(TANG_VM.AND.flag.GT.1) GS_TEMP_6=GS_TEMP_6+dbnmcdpsi(nm)*cosnm(nm)+dbnmsdpsi(nm)*sinnm(nm)
    END IF

    IF(flag.NE.0) THEN
      qnmsinnm=bnmc0(nm)*sinnm(nm)
      !dBdz_0=dBdz_0-qnmsinnm*n*nzperiod <----original
      GS_TEMP_8=GS_TEMP_8-qnmsinnm*n*nzperiod
      !dBdt_0=dBdt_0-qnmsinnm*m <---- original
      GS_TEMP_10=GS_TEMP_10-qnmsinnm*m
      qnmcosnm=bnms0(nm)*cosnm(nm)
      !dBdz_0=dBdz_0+qnmcosnm*n*nzperiod <---- original
      GS_TEMP_8=GS_TEMP_8+qnmcosnm*n*nzperiod
      !dBdt_0=dBdt_0+qnmcosnm*m <----original
      GS_TEMP_10=GS_TEMP_10+qnmcosnm*m
    IF(flagB1) THEN
      qnmsinnm=bnmc1(nm)*sinnm(nm)
      !dBdz_1=dBdz_1-qnmsinnm*n*nzperiod <---- original
      GS_TEMP_12=GS_TEMP_12-qnmsinnm*n*nzperiod
      !dBdt_1=dBdt_1-qnmsinnm*m <----original
      GS_TEMP_14=GS_TEMP_14-qnmsinnm*m
      qnmcosnm=bnmc1(nm)*cosnm(nm)
      !dBdz_1=dBdz_1+qnmcosnm*n*nzperiod <----original
      GS_TEMP_12=GS_TEMP_12+qnmcosnm*n*nzperiod
      !dBdt_1=dBdt_1+qnmcosnm*m <----original
      GS_TEMP_14=GS_TEMP_14+qnmcosnm*m
    END IF
  END IF

ELSE

  IF(flag.EQ.0.OR.flag.EQ.2) THEN
    ! B_0=B_0+bnmc0(nm)*cosnm(nm) <---- original
    GS_TEMP_2=GS_TEMP_2+bnmc0(nm)*cosnm(nm)
    !IF(flagB1) B_1=B_1+bnmc1(nm)*cosnm(nm) <--original
    IF(flagB1) GS_TEMP_5=GS_TEMP_5+bnmc1(nm)*cosnm(nm)
    !IF(TANG_VM.AND.flag.GT.1) dBdpsi=dBdpsi+dbnmcdpsi(nm)*cosnm(nm) <----original
    IF(TANG_VM.AND.flag.GT.1) GS_TEMP_7=GS_TEMP_7+dbnmcdpsi(nm)*cosnm(nm)
  END IF

```

 Time recorded for *instance of CALCB_DEL* taking maximum time

22 Processes (Total time)	Original (NaN)	No NaN (expected behaviour)	Optimized=Vectorized
CALCB_DEL	138.31 sec	126.51 sec	47.28 sec
Total App Time	1941.66 sec	685.77	551.22

Speed-up for CALCB_DEL = $126.51/47.28 = 2.67x$

Total Speed-up = $685.77/551.22 = 1.24x$

Attempt Optimization - 2, DELTA_PHASE in coefficients.f90

```

LOOP BEGIN at /gpfs/home/bsc99/bsc99102/KNOSOS/knosos/mn4/Sources/coefficients.f90(354,3)
remark #15388: vectorization support: reference cosnm_temp(:) has aligned access
remark #15389: vectorization support: reference cosnm(:) has unaligned access
remark #15389: vectorization support: reference cosnm_del(:) has unaligned access
remark #15389: vectorization support: reference sinnm(:) has unaligned access [ /gpfs
remark #15389: vectorization support: reference sinnm_del(:) has unaligned access [ /
remark #15389: vectorization support: reference sinnm(:) has unaligned access [ /gpfs
remark #15389: vectorization support: reference cosnm(:) has unaligned access [ /gpfs
remark #15389: vectorization support: reference sinnm_del(:) has unaligned access [ /
remark #15389: vectorization support: reference sinnm(:) has unaligned access [ /gpfs
remark #15389: vectorization support: reference cosnm_del(:) has unaligned access [ /
remark #15381: vectorization support: unaligned access used inside loop body
remark #15305: vectorization support: vector length 4
remark #15399: vectorization support: unroll factor set to 4
remark #15309: vectorization support: normalized vectorization overhead 0.156
remark #15301: FUSED LOOP WAS VECTORIZED
remark #15321: Compiler has chosen to target XMM/YMM vector. Try using -qopt-zmm-usage=
remark #15449: unmasked aligned unit stride stores: 1
remark #15450: unmasked unaligned unit stride loads: 4
remark #15451: unmasked unaligned unit stride stores: 1
remark #15475: --- begin vector cost summary ---
remark #15476: scalar cost: 23
remark #15477: vector cost: 6.000
remark #15478: estimated potential speedup: 3.410
remark #15488: --- end vector cost summary ---
LOOP END
    
```

```

LOOP BEGIN at /gpfs/home/bsc99/bsc99102/KNOSOS/knosos/mn4/Sources/coefficients.f90(355,3)
remark #15388: vectorization support: reference cosnm_temp(:) has aligned access
remark #15388: vectorization support: reference cosnm(:) has aligned access
remark #15388: vectorization support: reference cosnm_del(:) has aligned access
remark #15388: vectorization support: reference sinnm(:) has aligned access [ /gpfs/h
remark #15388: vectorization support: reference sinnm_del(:) has aligned access [ /gp
remark #15305: vectorization support: vector length 4
remark #15399: vectorization support: unroll factor set to 4
remark #15300: LOOP WAS VECTORIZED
remark #15321: Compiler has chosen to target XMM/YMM vector. Try using -qopt-zmm-usage=
remark #15448: unmasked aligned unit stride loads: 4
remark #15449: unmasked aligned unit stride stores: 1
remark #15475: --- begin vector cost summary ---
remark #15476: scalar cost: 12
remark #15477: vector cost: 2.250
remark #15478: estimated potential speedup: 4.540 |
remark #15488: --- end vector cost summary ---
LOOP END
    
```

22 Processes	No NaN	Aligned + zmm
DELTA_PHASE	35.99 sec	29.03 sec

Speed-up = 1.23x

This is another loop ! Needs vector aligned separately !

```

!dir$ vector aligned
cosnm_temp=cosnm*cosnm_del-sinnm*sinnm_del
sinnm=cosnm*sinnm_del+sinnm*cosnm_del
cosnm=cosnm_temp
    
```

Add at compile time

`-align array64byte -qopt-zmm-usage=high`

Attempt Optimization - 1(b), **CALCB_DEL** in **coefficients.f90**

```

DO nm=1,Nnm
  n=np(nm)
  m=mp(nm)
  IF(STELL_ANTISYMMETRIC) THEN

    IF(flag.EQ.0.OR.flag.EQ.2) THEN
      ! B_0=B_0+bnmc0(nm)*cosnm(nm)+bnms0(nm)*sinnm(nm) <---- original
      GS_TEMP_1=GS_TEMP_1+bnmc0(nm)*cosnm(nm)+bnms0(nm)*sinnm(nm)
      !IF(flagB1) B_1=B_1+bnmc1(nm)*cosnm(nm)+bnms1(nm)*sinnm(nm) <----original
      !IF(flagB1) GS_TEMP_4=GS_TEMP_4+bnmc1(nm)*cosnm(nm)+bnms1(nm)*sinnm(nm)
      !IF(TANG_VM.AND.flag.GT.1) dBdpsi=dBdpsi+dbnmc1(nm)*cosnm(nm)+dbnms1(nm)*sinnm(nm) <----original
      !IF(TANG_VM.AND.flag.GT.1) GS_TEMP_6=GS_TEMP_6+dbnmc1(nm)*cosnm(nm)+dbnms1(nm)*sinnm(nm)
    END IF

    IF(flag.NE.0) THEN
      qnmsinnm=bnmc0(nm)*sinnm(nm)
      !dBdz_0=dBdz_0-qnmsinnm*nzperiod <----original
      GS_TEMP_8=GS_TEMP_8-qnmsinnm*nzperiod
      !dBdt_0=dBdt_0-qnmsinnm*m <---- original
      GS_TEMP_10=GS_TEMP_10-qnmsinnm*m
      qnmcosnm=bnms0(nm)*cosnm(nm)
      !dBdz_0=dBdz_0+qnmcosnm*nzperiod <---- original
      GS_TEMP_8=GS_TEMP_8+qnmcosnm*nzperiod
      !dBdt_0=dBdt_0+qnmcosnm*m <----original
      GS_TEMP_10=GS_TEMP_10+qnmcosnm*m
    IF(flagB1) THEN
      qnmsinnm=bnmc1(nm)*sinnm(nm)
      !dBdz_1=dBdz_1-qnmsinnm*nzperiod <---- original
      GS_TEMP_12=GS_TEMP_12-qnmsinnm*nzperiod
      !dBdt_1=dBdt_1-qnmsinnm*m <----original
      GS_TEMP_14=GS_TEMP_14-qnmsinnm*m
      qnmcosnm=bnms1(nm)*cosnm(nm)
      !dBdz_1=dBdz_1+qnmcosnm*nzperiod <----original
      GS_TEMP_12=GS_TEMP_12+qnmcosnm*nzperiod
      !dBdt_1=dBdt_1+qnmcosnm*m <----original
      GS_TEMP_14=GS_TEMP_14+qnmcosnm*m
    END IF
  END IF

ELSE

  IF(flag.EQ.0.OR.flag.EQ.2) THEN
    ! B_0=B_0+bnmc0(nm)*cosnm(nm) <---- original
    GS_TEMP_2=GS_TEMP_2+bnmc0(nm)*cosnm(nm)
    !IF(flagB1) B_1=B_1+bnmc1(nm)*cosnm(nm) <--original
    !IF(flagB1) GS_TEMP_5=GS_TEMP_5+bnmc1(nm)*cosnm(nm)
    !IF(TANG_VM.AND.flag.GT.1) dBdpsi=dBdpsi+dbnmc1(nm)*cosnm(nm) <----original
    !IF(TANG_VM.AND.flag.GT.1) GS_TEMP_7=GS_TEMP_7+dbnmc1(nm)*cosnm(nm)
  END IF

```

22 Processes (Total time)	Original (NaN)	No NaN (expected behaviour)	Optimized=Vectorized + aligned + zmm
CALCB_DEL	138.31 sec	126.51 sec	35.16 sec
Total App Time	1941.66 sec	685.77	514.02

Speed-up for **CALCB_DEL** = $126.51/35.16 = 3.59x$

Total Speed-up = $685.77/514.02 = 1.33x$

Multiple MPI Procs: → Surfaces

[Parallelizing `COEFFICIENTS_DKE`]

- `COEFFICIENTS_DKE` takes 41% of total time (after removing NaN)
- `COEFFICIENTS_DKE` takes 33% of total time in Vectorized code
- `COEFFICIENTS_DKE_GS` new function which divides work between MPI processes (*takes about 13% of total time with 96 processes*)

Parallelizing COEFFICIENTS_DKE_GS

- MPI processes that collectively handle a surface placed in SURFACE_COMM_WORLD
- Execution path:

```
IF (surface_comm_numprocs .GT. 1) THEN
    CALL COEFFICIENTS_DKE_GS(...)
ELSE
    CALL COEFFICIENTS_DKE(...)
END IF
```
- Divides main loop **almost equally** between MPI processes (in SURFACE_COMM_WORLD)

```
! DO ipoint=1,npoint <--- original
WRITE(iout,"(a,I7,a,I7)") 'Loop bounds =', 1+cumulative_work(surface_comm_myrank+1), ' to =
DO ipoint=1+cumulative_work(surface_comm_myrank+1), cumulative_work(surface_comm_myrank+2)
```


Results for 22 surfaces

Processes	-03 (Round-Robin process assign.)	-03 (Compact process assign.)
22	25.984 (base)	24.709 (base)
44	24.476 (5.8%)	21.482(13.06%)
48	22.307 (14.15%)	23.348(5.5%)
66	19.675 (24.28)	21.953(11.15%)
88	19.133 (26.36%)	19.053 (22.89%)
96	19.470 (25.06%)	21.139 (14.44%)

22 processes Vs 88 processes

- With 22 MPI `coefficients_dke` 5.43 sec per process (=119.58/22)
- With 88 MPI `coefficients_dke_gs` takes 1.57 sec per process
(=138.69/88)
- Speed-up per process = $5.43/1.57 = 3.45$ (\approx ideal value 4)
- Majority of the difference due to `MPI_Allgatherv` ~ 0.15 secs per process (=19.11/88)

Summary


- Summary
 - Identification of NaN
 - Vectorization + alignment + zmm registers: 33% improvement
 - Parallelization with 88 processes: 25% improvement (only `COEFFICIENTS_DKE` parallelized)
 - Total 58% improvement
- Current status: Completed

SOLPS-ITER

(David Vicente Dorca, Cristian Morales, Gaurav Saxena)



SOLPS-ITER

- **Scrape-Off Layer Plasma Simulation** (boundary plasma)
 - Monte Carlo code **Eirene** (MPI parallelized)
 - **B2.5 Plasma Fluid solver (OpenMP parallelized) - bad scaling**
 - Extremely difficult to install (... *but* ...courtesy David Vicente Dorca in 6 hours ...)
 - Fortran 77 (fixed form), Fortran 90
- 

get_num_threads() function

[solps-iter/modules/B2.5/src/modules/b2mod_openmp.F]

```
C    integer function get_num_threads()
C    get_num_threads = 1
C!$OMP parallel
C!$OMP single
C!$    get_num_threads = omp_get_num_threads()
C!$OMP end single
C!$OMP end parallel
C    end function
```



- ❖ **Ideally** should emulate `omp_get_num_threads()` (*does not*)
 - Returns `OMP_NUM_THREADS` when called from a non-OpenMP non-parallel region (*should return 1 because there are no threads*).
 - Returns 1 when called from OpenMP parallel region (*should return `OMP_NUM_THREADS` value, but no nested parallelism hence 1 returned*).

Replace `get_num_threads()`

❖ Why ?

- **Spawning threads** expensive.
- **Called by every subroutine/function every-time !**
- **Not scalable** as threads increase
- **Threads wait** in barrier at the end of *single region AND parallel region* i.e. 2 barriers

```
integer function get_num_threads()
logical :: in_parallel
in_parallel = .false.
!$ in_parallel = OMP_IN_PARALLEL()
if(in_parallel) then
!$   get_num_threads = omp_get_num_threads()
else
   get_num_threads = 1
end if
end function
```

Version 1

```
C   integer function get_num_threads()
C#ifdef _OPENMP
C   if(OMP_IN_PARALLEL()) then
C!$       get_num_threads = omp_get_num_threads()
C   else
C!$       get_num_threads = omp_get_max_threads()
C   end if
C#else
C   get_num_threads = 1
C#endif
C   end function
```

Version 2

Preliminary Experiment

- ❖ Marenostrom 4 (MN4), Intel 2017.4, IMPI 2017.4
- ❖ `AUG_16151_D+C+He/16151_1.6MW_2.0e19_D=0.4_chi=1.6_pump=0.90`
- ❖ MPI + OpenMP coupled version of Eirene + B2.5
- ❖ `KMP_AFFINITY=disabled` ▶
- ❖ `b2mndir_ntim=20, b2mndir_elapsed=0`
 - Each run has 21 Eirene iterations
 - Each Eirene iteration gives its run-time
 - $B2.5_Run_Time = (Real_Time) - (Total_Eirene_Run_Time)$

Results

Original (LHS) Vs Optimized Version 2 (RHS)

Threads	Eirene (sec)	Real time (sec)	B2.5 (sec)	B2.5 Speed-up (from 1 thread)
48	85.33	259	173.67	1.36
24	82.46	222	139.54	1.69
12	83.15	229	145.85	1.62
6	82.46	235	152.54	1.55
4	80.02	241	160.98	1.47
2	80.98	273	192.02	1.23
1	80.13	316	235.87	1

Threads	Eirene (sec)	Real Time (sec)	B2.5 (sec)	B2.5 Speed-up (from 1 thread)
48	82.83	227	144.17	1.73
24	82.29	222	139.71	1.79
12	81.31	219	137.69	1.81
6	81.05	228	146.95	1.70
4	80.25	239	158.75	1.57
2	82.98	274	191.02	1.31
1	81.51	331	249.49	1.00

- ❖ At 48 threads, B2.5 time reduction: $(173.67 - 144.17)/173.67 = 17\%$ (recommended)
- ❖ At 12 threads, B2.5 time reduction: $(145.85 - 137.69)/145.85 = 5.6\%$

Compiler Additional Flags

```
FCOPTS    = -g -O2 -fPIC -assume no2underscore -fp-model precise  
FCOPTS    += -xHost -align array64byte -qopt-zmm-usage=high #added
```

- ❖ `-xHost` should generate AVX-512 instructions for Intel
- ❖ `-align array64byte` should allocate 64 byte aligned arrays (except `COMMON` block)
- ❖ `-qopt-zmm-usage=high` maximizes use of `zmm` register but Intel/2017.4 takes it as an unknown option

ifort: command line warning #10006: ignoring unknown option '-qopt-zmm-usage=high'

⇒ Warning with Intel/2017.4

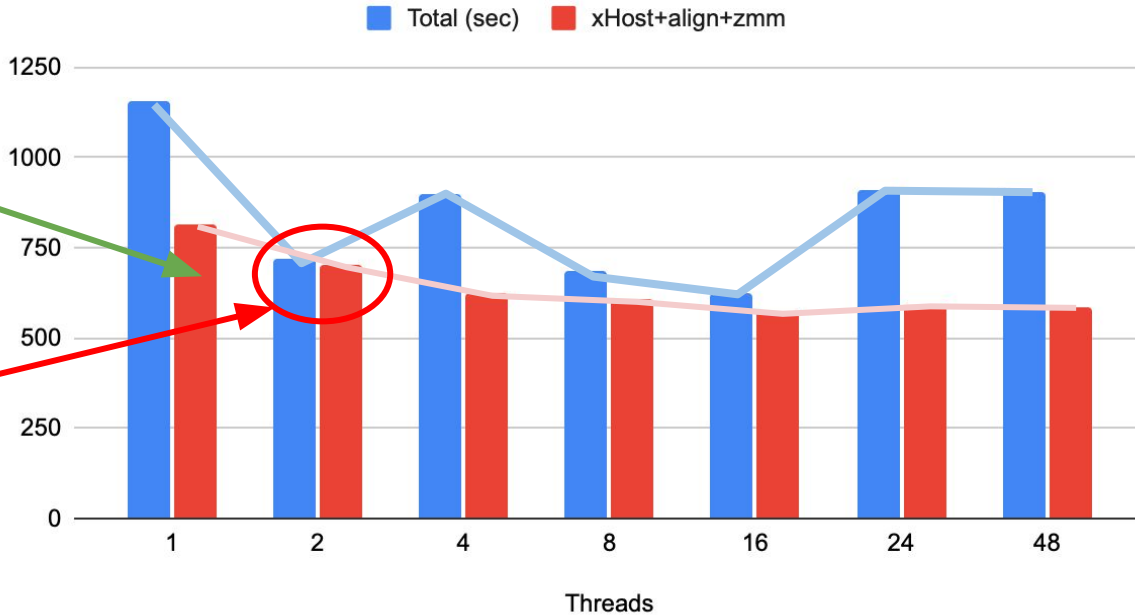
⇒ **available only with Intel/18.x :**

- ❖ <https://www.intel.com/content/www/us/en/developer/articles/technical/the-intel-advanced-vector-extensions-512-feature-on-intel-xeon-scalable.html>

Total run-time Vs threads

Intel 2018.4, IMPI/2018.4, **KMP_AFFINITY=disabled**

Total (sec) and Flags_Total(sec)



Vector length increased to 8 at several places

Anomaly !

Remember:

KMP_AFFINITY=disabled
else error with < 48 threads.

Final comparison

Intel/2018.4, KMP_AFFINITY=disabled

Intel/2018.4,
KMP_AFFINITY=scatter,verbose,norespec
t
xHost+align+zmm

Threads	Total (sec)	xHost+align+zmm(sec)	Total(sec)
1	1156	815	816
2	722	701	665
4	900	626	597
8	686	610	583
16	625	573	564
24	908	589	553
48	904	586	575

Lesson of the day: **KMP_AFFINITY** is extremely important for performance in SOLPS-ITER

At 'b2mndr_ntim' '500'

Threads	Exp. Time (min & sec)	Exp. Time (sec)	Ideal time (sec)	Full B2.5 Speed-up (S_p)
1	6 m 49 sec	409	–	–
2	4 m 34 sec	274	204.5	1.49
4	3 m 32 sec	212	102.25	1.92
12	2 m 45 sec	165	51.125	2.47
24	2 m 46 sec	166	25.56	2.46
48	2 m 50 sec	170	12.75	2.40

b2xpfe.F loops - problems in Vectorization

```
!$OMP PARALLEL DO DEFAULT(none) COLLAPSE(3)
!$OMP& PRIVATE(ix, iy, is)
!$OMP& SHARED(fne, fna, fa_ext, rza, za_ext, ns, ns_ext, nx, ny)
!$OMP& SHARED(leftix, leftiy, vol, bottomix, bottomiy)
#endif /* NO_OPENMP_B2XPFE */
do iy = -1, ny
do is = 0, ns-1
do ix = -1, nx
if (leftix(ix,iy).ne.-2) then
fne(ix,iy,0) = fne(ix,iy,0) + fna(ix,iy,0,is)*
(rza(ix,iy,is)*vol(leftix(ix,iy),leftiy(ix,iy))+
rza(leftix(ix,iy),leftiy(ix,iy),is)*vol(ix,iy))/
(vol(ix,iy)+vol(leftix(ix,iy),leftiy(ix,iy)))
endif
if (bottomiy(ix,iy).ne.-2) then
fne(ix,iy,1) = fne(ix,iy,1) + fna(ix,iy,1,is)*
(rza(ix,iy,is)*vol(bottomix(ix,iy),bottomiy(ix,iy))+
rza(bottomix(ix,iy),bottomiy(ix,iy),is)*vol(ix,iy))/
(vol(ix,iy)+vol(bottomix(ix,iy),bottomiy(ix,iy)))
endif
enddo
enddo
enddo
#endif
#endif
```

If condition

Indirect access

(array index = another array element)

E.g. `vol(leftix(ix,iy),leftiy(ix,iy))`

Substitute:

- `leftix(ix,iy) = ix - 1`
- `leftiy(ix,iy) = iy`

But does not hold in general !

Vectorization Report

```
remark #15415: vectorization support: irregularly indexed load was generated for the variable <vol_(leftix_(ix,iy),leftiy_(ix,iy))>, masked, 64-bit indexed, part
remark #15415: vectorization support: irregularly indexed load was generated for the variable <rza>, masked, 64-bit indexed, part of index is read from memory |
remark #15415: vectorization support: irregularly indexed load was generated for the variable <vol_(leftix_(ix,iy),leftiy_(ix,iy))>, masked, 64-bit indexed, part
remark #15415: vectorization support: irregularly indexed load was generated for the variable <vol_(bottomix_(ix,iy),bottomiy_(ix,iy))>, masked, 64-bit indexed, p
remark #15415: vectorization support: irregularly indexed load was generated for the variable <rza>, masked, 64-bit indexed, part of index is read from memory |
remark #15415: vectorization support: irregularly indexed load was generated for the variable <vol_(bottomix_(ix,iy),bottomiy_(ix,iy))>, masked, 64-bit indexed, p
remark #15305: vectorization support: vector length 8
remark #15309: vectorization support: normalized vectorization overhead 0.157
remark #15300: LOOP WAS VECTORIZED
remark #15450: unmasked unaligned unit stride loads: 2
remark #15456: masked unaligned unit stride loads: 16
remark #15457: masked unaligned unit stride stores: 2
remark #15458: masked indexed (or gather) loads: 6
remark #15475: --- begin vector cost summary ---
remark #15476: scalar cost: 128
remark #15477: vector cost: 58.250
remark #15478: estimated potential speedup: 2.000
remark #15486: divides: 2
remark #15488: --- end vector cost summary ---
OP END
```

Original

```
remark #15381: vectorization support: unaligned access used inside loop body
remark #15305: vectorization support: vector length 8
remark #15399: vectorization support: unroll factor set to 2
remark #15309: vectorization support: normalized vectorization overhead 0.129
remark #15301: PARTIAL LOOP WAS VECTORIZED
remark #15448: unmasked aligned unit stride loads: 1
remark #15449: unmasked aligned unit stride stores: 1
remark #15450: unmasked unaligned unit stride loads: 5
remark #15475: --- begin vector cost summary ---
remark #15476: scalar cost: 46
remark #15477: vector cost: 12.120
remark #15478: estimated potential speedup: 3.530
remark #15486: divides: 1
remark #15488: --- end vector cost summary ---
```

Original + Substitution

Advantage: Estimated Potential
Speed = 3.53 instead of 2.00

Hot Cache Effect in b2xpfe.F

[disable n > 16384 condition in sfill(...)]

Threads	Time (original)(s _p)	Time (sec) (original + Subs)	Time (original + Subs + sfill)(s _p)
1	6.08 (1)	6.05	5.89 (1)
2	3.20 (1.9)	3.21	3.07 (1.91)
4	1.97 (3.08)	1.99	1.86 (3.16)
12	1.16 (5.24)	1.17	1.04 (5.66)
24	1.02 (5.96)	1.00	0.77 (7.65)
48	1.22 (4.98)	1.11	0.92 (6.40)

- ❖ At 24 threads, time reduction ~ 24% (from original)
- ❖ At 48 threads, time reduction ~ 24.5% (from original)
- ❖ Speed-up of OpenMP loop region ~ 7.65 at 24 threads 😊

Critical Section: ITER_2171_D+He+Be+Ne Scaling

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

Threads	Time (sec)		S _p	
1	1045	957	1	1.00
2	787	596	1.32	1.60
4	856	388	1.22	2.46
12	848	251	1.23	3.80
24	834	210	1.25	4.55
48	847	215	1.23	4.45

- ❖ #ifndef NO_OPENMP_B2SIFRTF enables OpenMP
- ❖ **Removed ! $\$$ OMP CRITICAL as no need** (Serial or Parallel invocation both)

Function `fka()` in `b2sifrtf.F`

- ❖ With **24 threads**, takes a total of **193 sec**
- ❖ Loop in `fka()` not vectorized as `-fp-model=precise` used
- ❖ We use `!$OMP SIMD reduction(+:fka)`
- ❖ `fka()` vectorized (**non-unit strides**): takes ~ **165 sec** with **24 threads**
- ❖ Finally, 3rd dimensions of arrays `rz2(ix,iy,is)` and `na(ix,iy,is)` copied to auxiliary contiguous arrays to remove jumps (**unit-stride**).

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

Summary: Mail/Meeting minutes (excerpts) from Xavier Bonin

- “Following the presentation from the Barcelona Supercomputing Center Advanced Computing Hub..., ***several of their recommendations were implemented in the code.***”
- “...This includes the proper setting on the OMP_STACKSIZE variable, correcting the ***get_num_threads*** utility routine to avoid creating unnecessary threads, removing the superfluous ***critical sections in b2sifrtf***, and providing better compiler optimization flags.”
- “...some more measurements of code speed-up from the ***removal of the b2sifrtf critical regions***, and gave a guide for ***optimization flags*** for the GCC compiler ... next target for code speed-up is the ***fka utility routine...***”
- “...***good news is that the BSC ACH team will be able to continue working with us to improve the code performance in 2024***, as we have received some renewed EUROfusion funding for this activity.”

Thank you.

Gaurav Saxena (gaurav.saxena@bsc.es)

&

David Vicente Dorca (david.vicente@bsc.es)