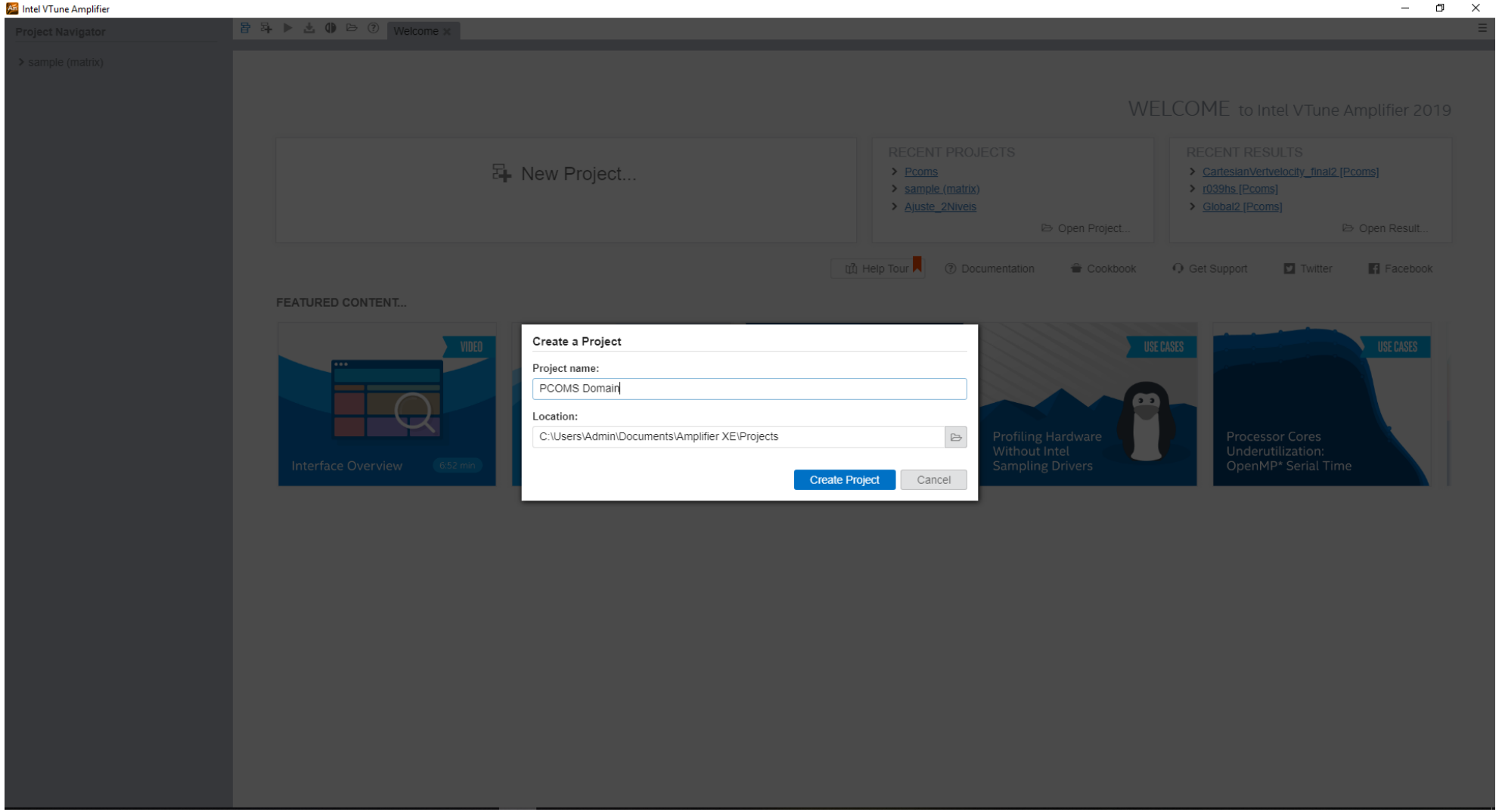
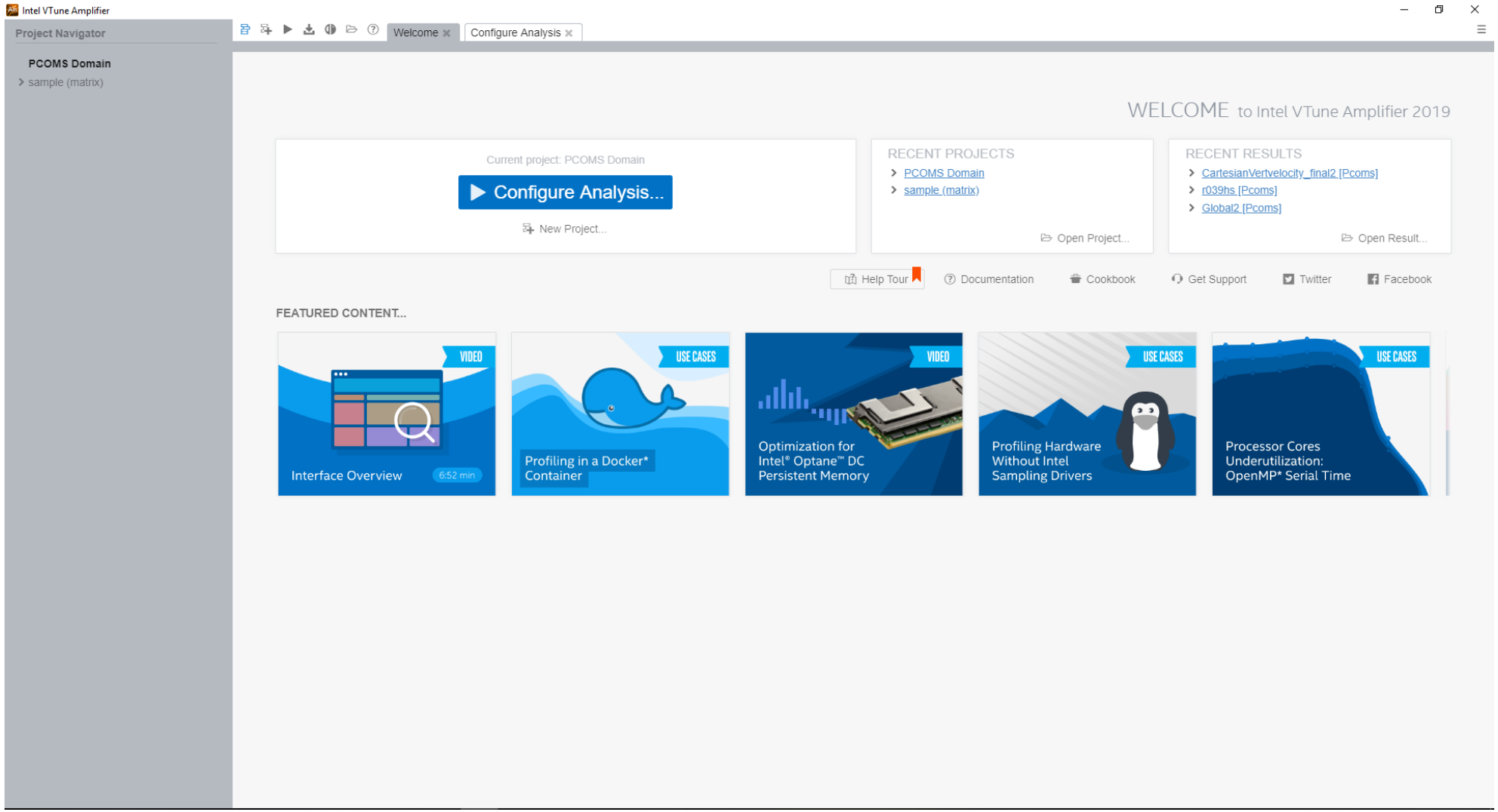
MOHID code optimizations

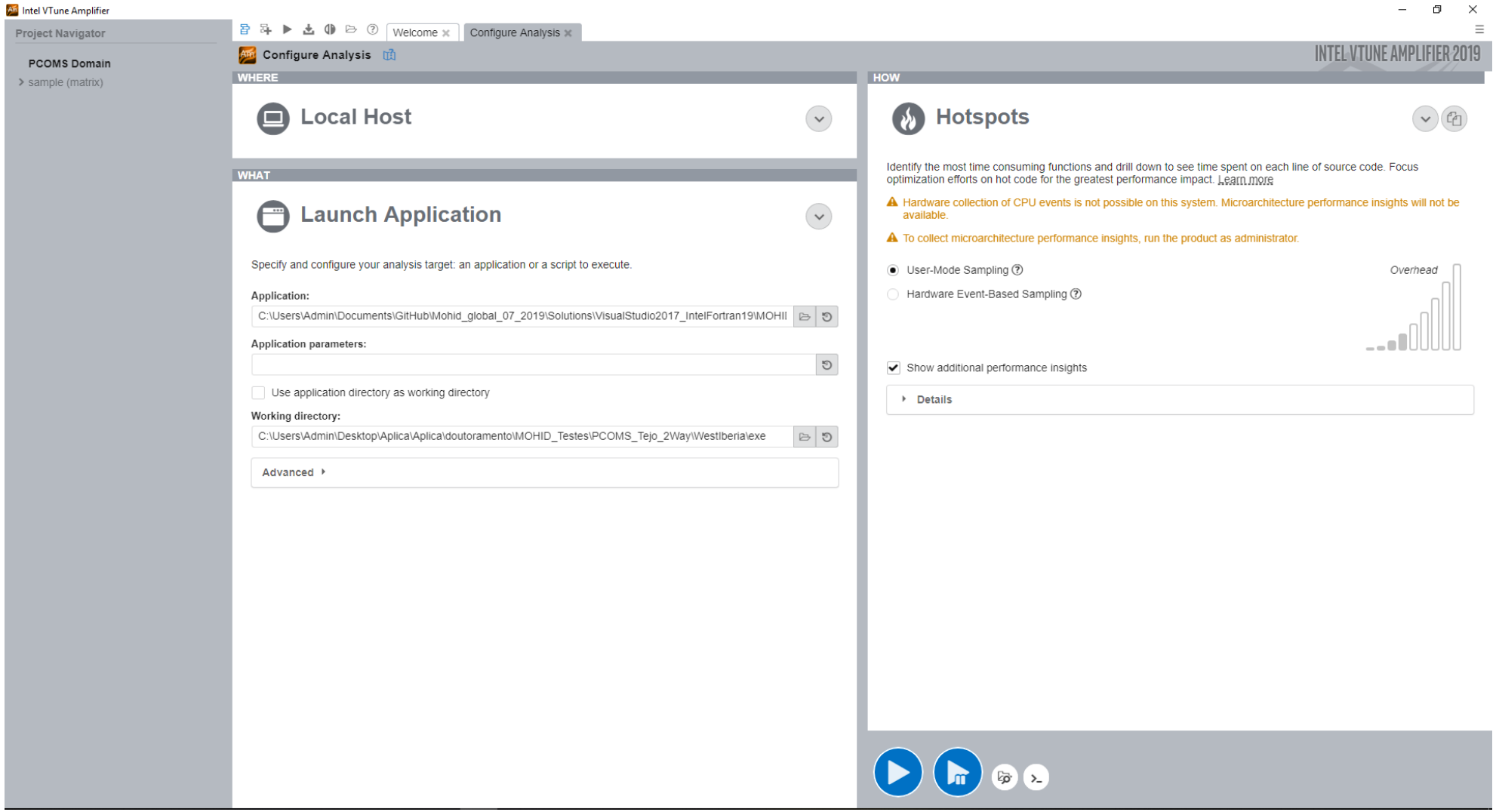
Vtune Amplifier



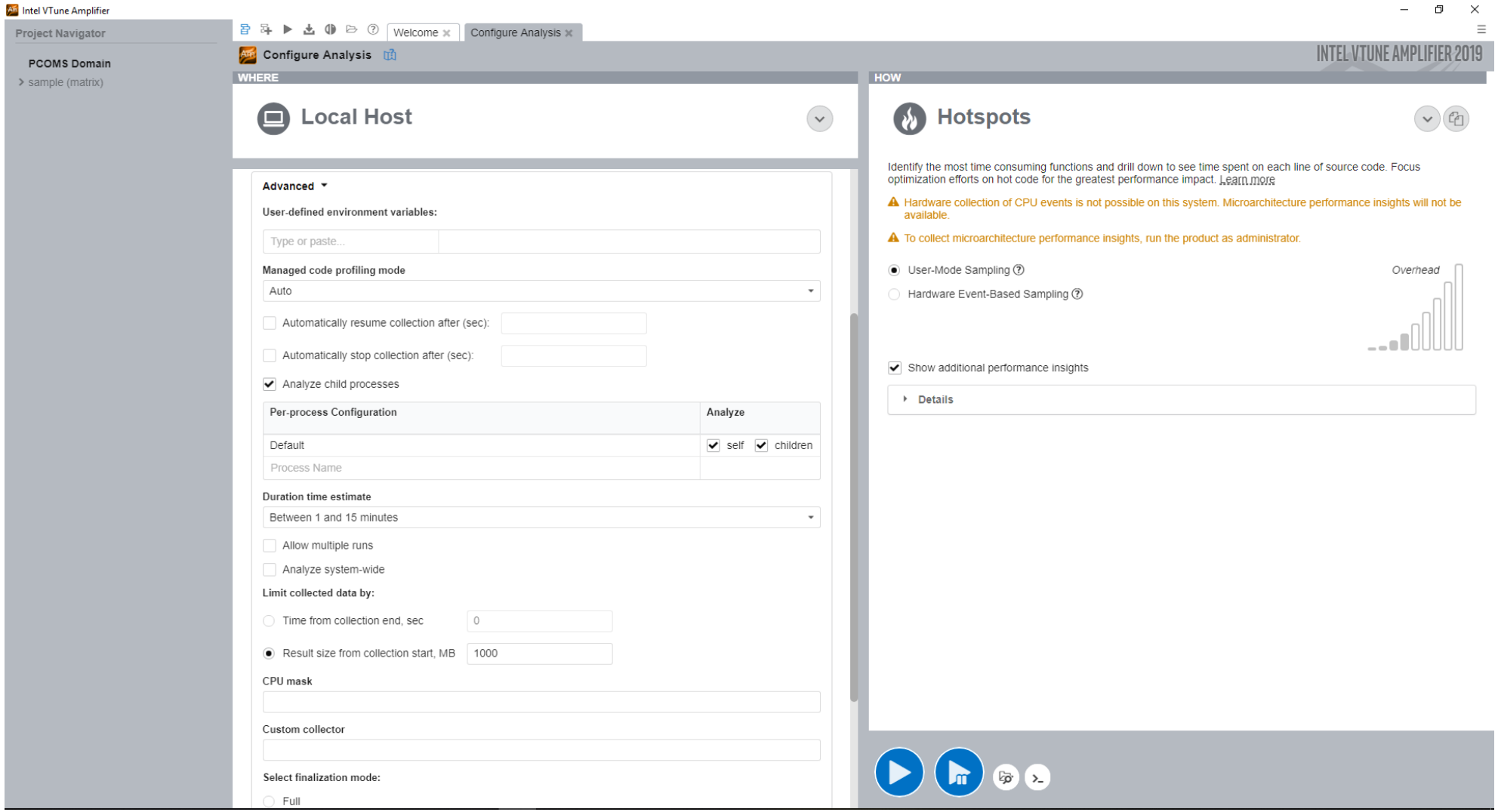
Vtune Amplifier



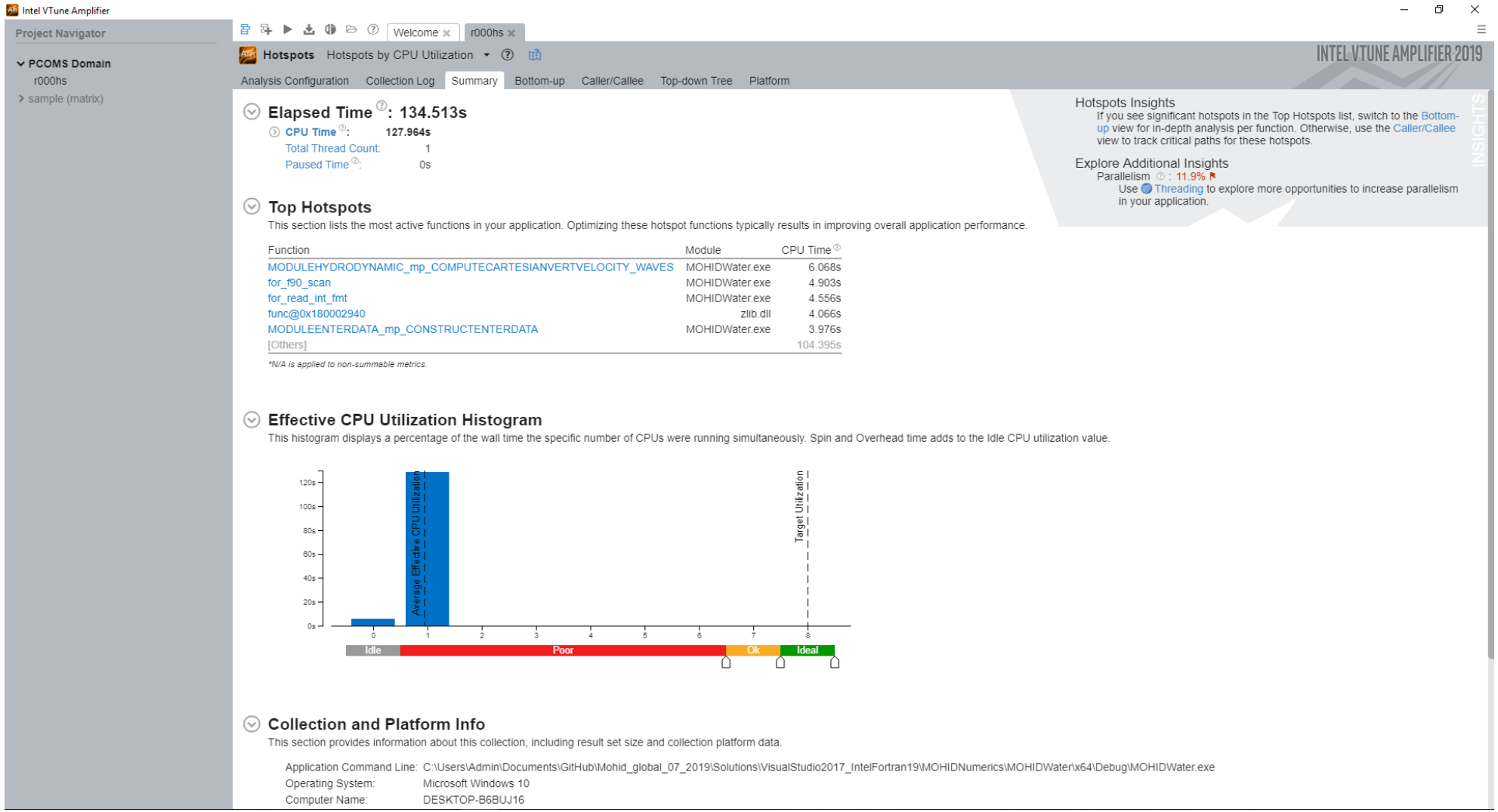
Vtune Amplifier



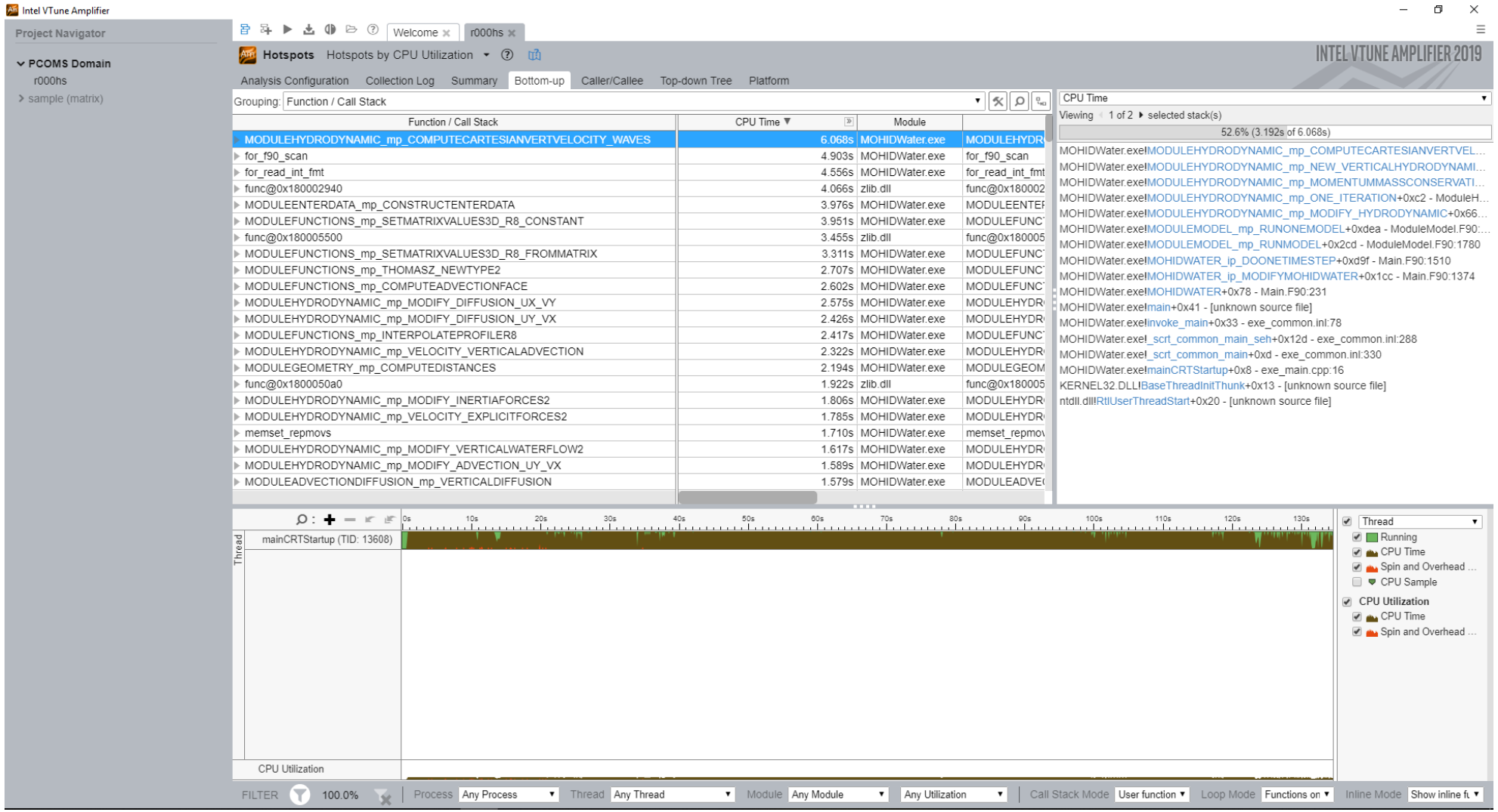
Vtune Amplifier



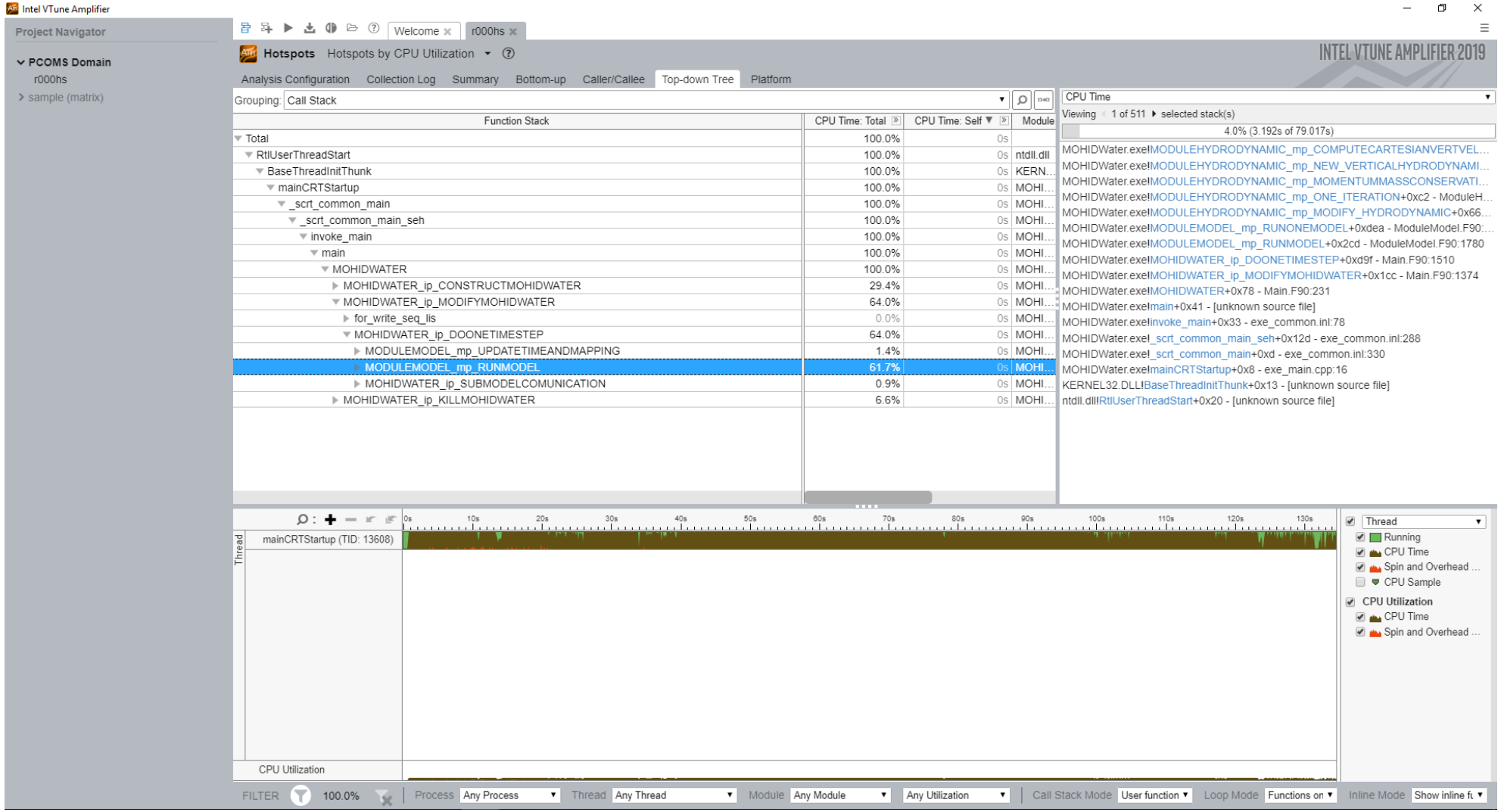
Vtune Amplifier



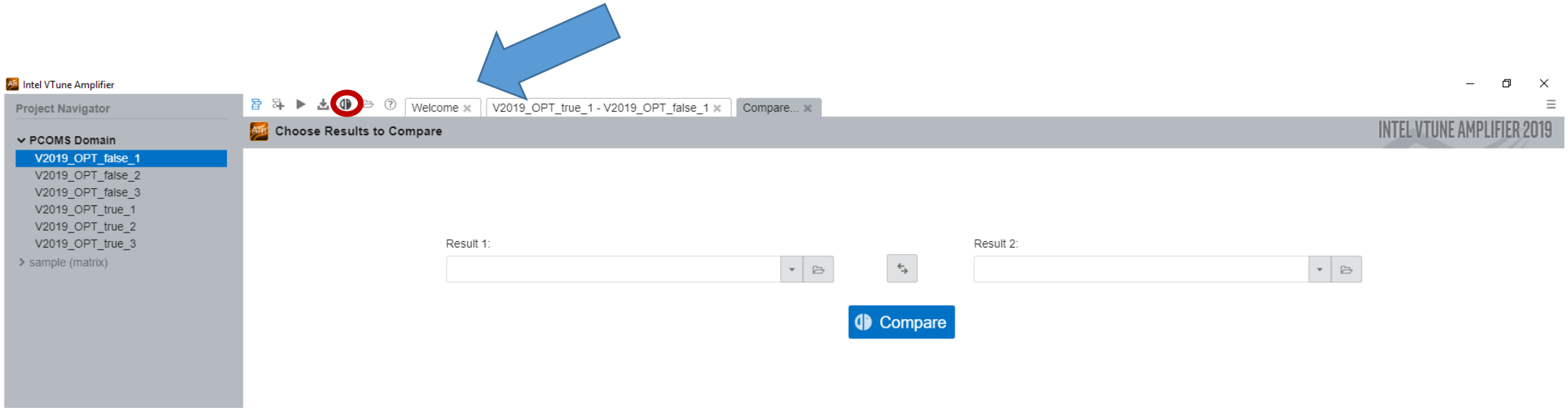
Vtune Amplifier



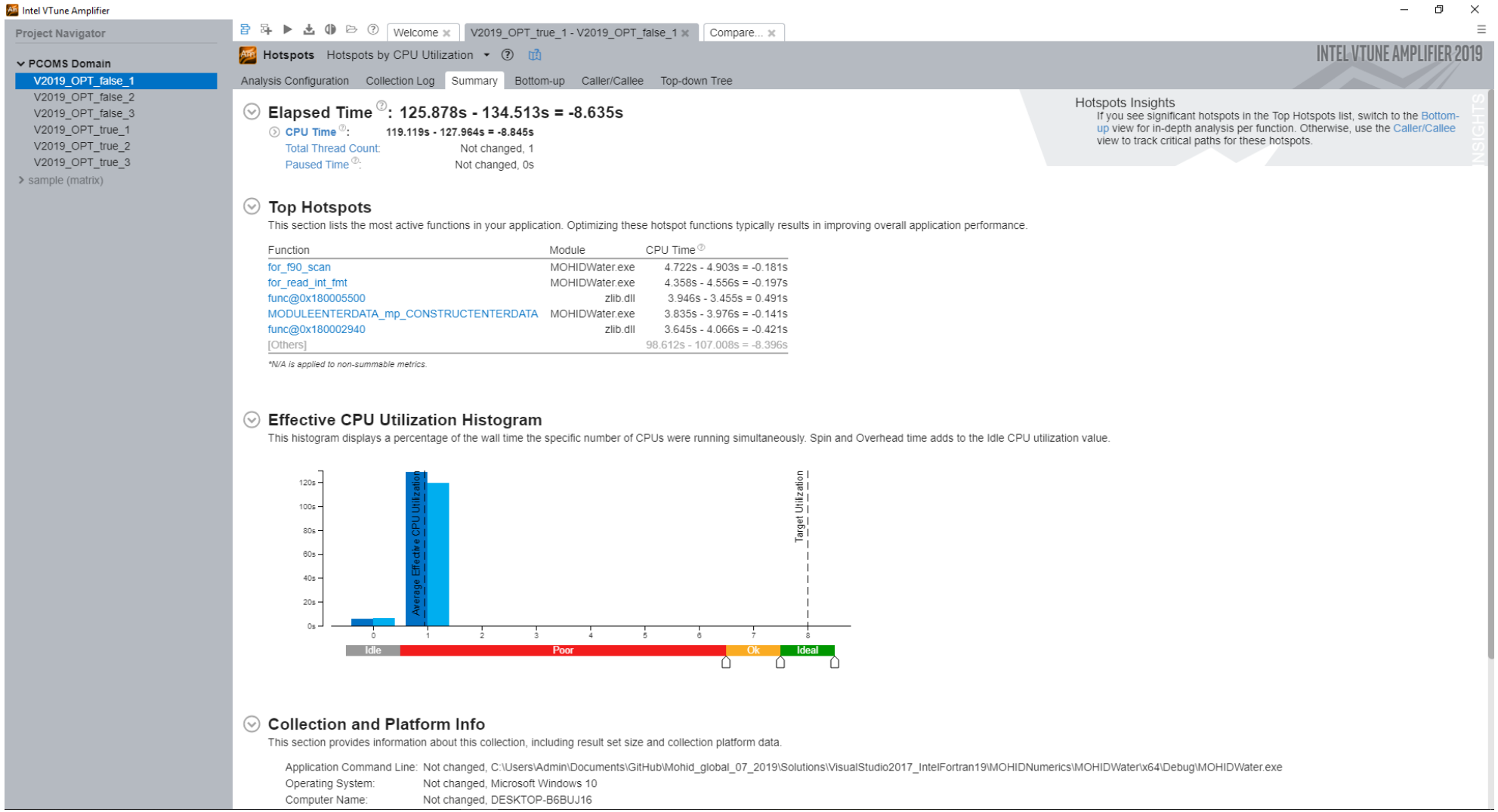
Vtune Amplifier



Vtune Amplifier



Vtune Amplifier



Optimizations – Hydrodynamic module

Me%Velocity%Horizontal%UV%New	Me%Velocity%Horizontal%U%New	Current direction u
	Me%Velocity%Horizontal%V%New	Current direction v
Me%Velocity%Horizontal%VU%New	Me%Velocity%Horizontal%U%New	Current direction u
	Me%Velocity%Horizontal%V%New	Current direction v

```
doj:     do j = JLB, JUB
doi:     do i = ILB, IUB

Cov1:        if (ComputeFaces3D_UV(i, j, KUB) == Covered) then

                 iSouth      = i - di
                 jWest       = j - dj
```

Optimizations – Hydrodynamic module

```
doj:     do j = JLB, JUB
doi:     do i = ILB, IUB
Cov1:       if (ComputeFaces3D_UV(i, j, KUB) == Covered) then
                iSouth     = i - di
                jWest      = j - dj
                kbottom = KFloor_UV(i, j)
                if (Me%ComputeOptions%atmosphereRAMP) TimeCoef2 = TimeCoef

dok:            do  k = kbottom, KUB
                    !Aceleration due the coriolis and centrifugal force
                    ![m/s]          = [m/s]           +     [s]     *     [m/s^2]
                    TiCoef_3D(i, j, k) = TiCoef_3D(i, j, k) + DT_Velocity * Inertial_Aceleration(i, j, k)
                    ![m/s]          = [m/s]           +     [s]     *     [m/s^2]
                    if (Me%Relaxation%Force)   then
                        TiCoef_3D(i, j, k) = TiCoef_3D(i, j, k) + DT_Velocity * Relax_aceleration(i, j, k)
                    endif
                    ![m/s]          = [m/s]           +     [s]     *     [m/s^2]
                    if (Me%ComputeOptions%Obstacle)                                    &
                        TiCoef_3D(i, j, k) = TiCoef_3D(i, j, k) + DT_Velocity * Me%Forces%ObstacleDrag_Aceleration(i, j, k)
                    ![m/s]          = [m/s]           +     [s]     *     [m/s^2]
                    if (Me%ComputeOptions%Turbine)                                     &
                        TiCoef_3D(i, j, k) = TiCoef_3D(i, j, k) + DT_Velocity* Me%Forces%Turbine_Acceleration(i, j, k)
                    if (Me%ComputeOptions%Scraper)                                     &
                        TiCoef_3D(i, j, k) = TiCoef_3D(i, j, k) + DT_Velocity * Me%Forces%Scraper_Aceleration(i, j, k)
                    if (Me%ThinWalls%ON .and. Me%ThinWalls%CloseFlag == 0)             &
                        TiCoef_3D(i, j, k) = TiCoef_3D(i, j, k) + DT_Velocity * Me%Forces%ThinWalls_Dissipation(i,j,k)
                    !Aceleration due to barotropic water Pressure

                    ![m/s^2]                  = [m/s^2] * [m] / [m]
                    WaterPressure_Aceleration  = Gravity * (WaterLevel_New(iSouth, jWest) - &
                                                 WaterLevel_New(i, j)) / DZX_ZY(iSouth, jWest)
                    !Deformation "crosta terrestre" - Tide Potential
                    WaterPressure_Aceleration  = Alpha * WaterPressure_Aceleration
                    if (Me%NonHydrostatic%ON .and. Me%NonHydroStatic%PressureCorrection .and. PressureBackwardInTime) then
                        ![m/s^2]           = [m/s^2]  +     [m^2/s^2] / [m]
                        WaterPressure_Aceleration = WaterPressure_Aceleration +           &
                                                    (PressureCorrect(iSouth, jWest, k) -  &
                                                     PressureCorrect(i     , j    , k))/ DZX_ZY(iSouth, jWest)
```

Optimizations – Hydrodynamic module

```
!------------------------------------------------------------------------------------------------------------
        ![m/s]    = [m/s]     +     [s]     *     [m/s^2]
        !Aceleration due the coriolis and centrifugal force
        !TiCoef_3D = TiCoef_3D + DT_Velocity * Inertial_Aceleration
        call AddMAtrixtimesScalar (TiCoef_3D, Me%Forces%Inertial_Aceleration, DT_Velocity, Me%WorkSize,  &
                                   ComputeFaces3D_UV, Me%Docycle_method, Me%External_Var%KFloor_UV)
!------------------------------------------------------------------------------------------------------------
        if (Me%Relaxation%Force) then
            ![m/s]    = [m/s]     +     [s]     *     [m/s^2]
            call AddMAtrixtimesScalar (TiCoef_3D, Me%Forces%Relax_Aceleration, DT_Velocity, Me%WorkSize,  &
                                       ComputeFaces3D_UV, Me%Docycle_method, Me%External_Var%KFloor_UV)
        endif
!------------------------------------------------------------------------------------------------------------
        if (Me%ComputeOptions%Obstacle) then
            ![m/s]    = [m/s]     +     [s]     *     [m/s^2]
            call AddMAtrixtimesScalar (TiCoef_3D, Me%Forces%ObstacleDrag_Aceleration, DT_Velocity, Me%WorkSize, &
                                       ComputeFaces3D_UV, Me%Docycle_method, Me%External_Var%KFloor_UV)
        endif
!------------------------------------------------------------------------------------------------------------
        if (Me%ComputeOptions%Turbine) then
            ![m/s]    = [m/s]     +     [s]     *     [m/s^2]
            call AddMAtrixtimesScalar (TiCoef_3D, Me%Forces%Turbine_Acceleration, DT_Velocity, Me%WorkSize, &
                                       ComputeFaces3D_UV, Me%Docycle_method, Me%External_Var%KFloor_UV)
        endif
!------------------------------------------------------------------------------------------------------------
        if (Me%ComputeOptions%Scraper) then
            ![m/s]    = [m/s]     +     [s]     *     [m/s^2]
            call AddMAtrixtimesScalar (TiCoef_3D, Me%Forces%Scraper_Aceleration, DT_Velocity, Me%WorkSize, &
                                       ComputeFaces3D_UV, Me%Docycle_method, Me%External_Var%KFloor_UV)
        endif
```

Optimizations – Hydrodynamic module

```
TiCoef_3D => Me%Coef%D3%Ti
ComputeFaces3D_UV    => Me%External_Var%ComputeFaces3D_UV
LandBoundaryFacesUV => Me%External_Var%LandBoundaryFacesUV
Velocity_UV_Old => Me%Velocity%Horizontal%UV%Old
Velocity_UV_New => Me%Velocity%Horizontal%UV%New
WaterPoints3D   => Me%External_Var%WaterPoints3D
```

From 4 if statements and :

```
iSouth  = i -   di
jWest   = j -   dj
i_North = i +   di
j_East  = j +   dj
iSouth2 = i - 2*di
jWest2  = j - 2*dj
iSouth3 = i - 3*di
jWest3  = j - 3*dj
```

To

```
if (Me%Direction%di == 1) then
    call Modify_Advection_UY_VX_Y (Me%External_Var%ComputeFaces3D_V, Me%External_Var%ImposedTangentialFacesV, &
                                   Me%External_Var%BoundaryFacesV, Me%External_Var%KFloor_V,                &
                                   Me%External_Var%DXX, Me%External_Var%Volume_V, Me%WaterFluxes%X,         &
                                   Me%Forces%Horizontal_Transport, Me%Velocity%Horizontal%V%Old)
else
    call Modify_Advection_UY_VX_X (Me%External_Var%ComputeFaces3D_U, Me%External_Var%ImposedTangentialFacesU, &
                                   Me%External_Var%BoundaryFacesU, Me%External_Var%KFloor_U,                &
                                   Me%External_Var%DYY, Me%External_Var%Volume_U, Me%WaterFluxes%Y,         &
                                   Me%Forces%Horizontal_Transport, Me%Velocity%Horizontal%U%Old)
endif
```

Optimizations – Hydrodynamic module

To 1 IF statement and a routine that fits a screen

```
!Momentum normal flux is always compute if at least one of adjacent faces is a face to compute
if (ComputeFaces3D_V(i, j, k) == 1 .or. ComputeFaces3D_V(i-1, j,k) == 1) then

    FaceFlux_WestSouth = (WaterFlux_Y(i-1, j, k) + WaterFlux_Y(i, j, k))/2. ![m^3/s]

    if  ((FaceFlux_WestSouth > 0) .and. (ComputeFaces3D_V(i-2, j, k) == 0)) then
        !NearBoundary = .true. CFace(2) = 1
        Me%Aux3DFlux(i, j, k) = dble(Velocity_V_Old(i-1, j, k)) * FaceFlux_WestSouth ![m/s*m^3/s]
        du4(1) = FillValueReal
    elseif ((FaceFlux_WestSouth <= 0) .and. (ComputeFaces3D_V(i+1,  j,  k) == 0)) then
        !NearBoundary = .true. CFace(3) = 1
        Me%Aux3DFlux(i, j, k) = dble(Velocity_V_Old(i  , j, k)) * FaceFlux_WestSouth ![m/s*m^3/s]
    elseif ((FaceFlux_WestSouth <= 0) .and. (ComputeFaces3D_V(i-2,  j,  k) == 0)) then

        !Use du4(1) obtained from previous iteration
        du4(2) = DZY     (i-2,j  ) ; du4(3) = DZY     (i-1,j  ); du4(4) = DZY(i, j);
        V4 (2) = Volume_V(i-1,j,k) ; V4 (3) = Volume_V(i  ,j,k);

        Vel4(1) = Velocity_V_Old(i-2, j, k); Vel4(2) = Velocity_V_Old(i-1, j, k);
        Vel4(3) = Velocity_V_Old(i  , j, k); Vel4(4) = Velocity_V_Old(i+1, j, k);

        call ComputeAdvectionFace_TVD_Superbee(Vel4, V4, du4, Me%Velocity%DT, FaceFlux_WestSouth, CFace)

        Me%Aux3DFlux(i, j, k) = dble(Vel4(2) * CFace(2)  + Vel4(3) * CFace(3)) *   &
                                FaceFlux_WestSouth ![m/s*m^3/s]
    else

        du4(1) = DZY(i-3, j) ; du4(2) = DZY(i-2, j)      ; du4(3) = DZY(i-1, j)     ; du4(4) = DZY(i, j);
                               V4 (2) = Volume_V(i-1,j,k); V4 (3) = Volume_V(i ,j,k);

        Vel4(1) = Velocity_V_Old(i-2, j, k); Vel4(2) = Velocity_V_Old(i-1, j, k);
        Vel4(3) = Velocity_V_Old(i  , j, k); Vel4(4) = Velocity_V_Old(i+1, j, k);

        call ComputeAdvectionFace_TVD_Superbee(Vel4, V4, du4, Me%Velocity%DT, FaceFlux_WestSouth, CFace)

        Me%Aux3DFlux(i, j, k) = dble(Vel4(2) * CFace(2)  + Vel4(3) * CFace(3)) *   &
                                FaceFlux_WestSouth ![m/s*m^3/s]
    endif
    Horizontal_Transport(i, j, k) = Horizontal_Transport(i, j, k) + Me%Aux3DFlux(i, j, k)
endif
```

Optimizations – Hydrodynamic module

Computation of vertical velocity

```
if (MeshSlope_) then
    if (MeshVelocity_) then

        call CartesianVertVelocity_dszdt (Me%External_Var%Volume_Z_New, Me%External_Var%Volume_Z_Old,  &
                                          Me%External_Var%DUX, Me%External_Var%DVY, Me%Waterlevel%DT,  &
                                          Me%Velocity%Vertical%Cartesian, Me%External_Var%ComputeFaces3D_W, &
                                          Me%External_Var%KFloor_Z)
    endif

    call CartesianVertVelocity_X (Me%External_Var%ComputeFaces3D_W, Me%External_Var%ComputeFaces3D_U,           &
                                  Me%External_Var%BoundaryPoints, Me%External_Var%SZZ, Me%External_Var%DZX,     &
                                  Me%External_Var%DWZ, Me%Velocity%Horizontal%U%New,                           &
                                  Me%Velocity%Vertical%Cartesian, Me%External_Var%KFloor_Z)

    call CartesianVertVelocity_Y (Me%External_Var%ComputeFaces3D_W, Me%External_Var%ComputeFaces3D_V,      &
                                  Me%External_Var%BoundaryPoints, Me%External_Var%SZZ, Me%External_Var%DZY, &
                                  Me%External_Var%DWZ, Me%Velocity%Horizontal%V%New,                       &
                                  Me%Velocity%Vertical%Cartesian, Me%External_Var%KFloor_Z)
endif
```

Optimizations – Hydrodynamic module

Vertical advection

```
MomentumFlux = dble(Vel4(1) * CFace(1)  + Vel4(2) * CFace(2)  +     &
                    Vel4(3) * CFace(3)  + Vel4(4) * CFace(4)) *     &
                    Face_Flux ![m/s*m^3/s]

TiCoef_3D(i, j, k  )  = TiCoef_3D(i, j, k  ) + (1. - ImplicitVertAdvection) * &
                        MomentumFlux * Me%Velocity%DT / V4(3)

TiCoef_3D(i, j, k-1)  = TiCoef_3D(i, j, k-1) - (1. - ImplicitVertAdvection) * &
                        MomentumFlux * Me%Velocity%DT / V4(2)

DCoef_3D (i, j, k  )  = DCoef_3D (i, j, k  ) - ImplicitVertAdvection *  &
                        CFace(2) * Face_Flux *  Me%Velocity%DT / V4(3)

ECoef_3D (i, j, k  )  = ECoef_3D (i, j, k  ) - ImplicitVertAdvection *  &
                        CFace(3) * Face_Flux *  Me%Velocity%DT / V4(3)

ECoef_3D (i, j, k-1)  = ECoef_3D (i, j, k-1) + ImplicitVertAdvection *  &
                        CFace(2) * Face_Flux *  Me%Velocity%DT / V4(2)

FCoef_3D (i, j, k-1)  = FCoef_3D (i, j, k-1) + ImplicitVertAdvection *  &
                        CFace(3) * Face_Flux *  Me%Velocity%DT / V4(2)
```

Optimizations – Hydrodynamic module

Vertical advection

```
if (Me%ComputeOptions%ImplicitVertAdvection == 1) then
    if (Me%Direction%di == 1) then
        call Vel_VertAdv_P2_TVD_SB_Imp_Y(Me%Velocity%Horizontal%V%Old, Me%External_Var%ComputeFaces3D_V,       &
                                         Me%External_Var%BoundaryFacesV, Me%External_Var%KFloor_V,             &
                                         Me%External_Var%WaterColumnV, Me%WaterFluxes%Z, Me%External_Var%DZY,  &
                                         Me%External_Var%Area_V, Me%External_Var%DXX, DT)
    else
        call Vel_VertAdv_P2_TVD_SB_Imp_X(Me%Velocity%Horizontal%U%Old, Me%External_Var%ComputeFaces3D_U,       &
                                         Me%External_Var%BoundaryFacesU, Me%External_Var%KFloor_U,             &
                                         Me%External_Var%WaterColumnU, Me%WaterFluxes%Z, Me%External_Var%DZX,  &
                                         Me%External_Var%Area_U, Me%External_Var%DYY, DT)
    endif
else

    if (Me%Direction%di == 1) then
        call Vel_VertAdv_P2_TVD_SB_Exp_Y(Me%Velocity%Horizontal%V%Old, Me%External_Var%ComputeFaces3D_V,       &
                                         Me%External_Var%BoundaryFacesV, Me%External_Var%KFloor_V,             &
                                         Me%External_Var%WaterColumnV, Me%WaterFluxes%Z, Me%External_Var%DZY,  &
                                         Me%External_Var%Area_V, Me%External_Var%DXX, DT)
    else
        call Vel_VertAdv_P2_TVD_SB_Exp_X(Me%Velocity%Horizontal%U%Old, Me%External_Var%ComputeFaces3D_U,       &
                                         Me%External_Var%BoundaryFacesU, Me%External_Var%KFloor_U,             &
                                         Me%External_Var%WaterColumnU, Me%WaterFluxes%Z, Me%External_Var%DZX,  &
                                         Me%External_Var%Area_U, Me%External_Var%DYY, DT)
    endif
endif
```

Optimizations – Hydrodynamic module

Modify_MatrixesOutput

Computation of all output matrices (ex: interpolation of velocities to the center of the cell), for every dt

Check if it is time to output something (hdf, timeserie, profile, etc)

YES

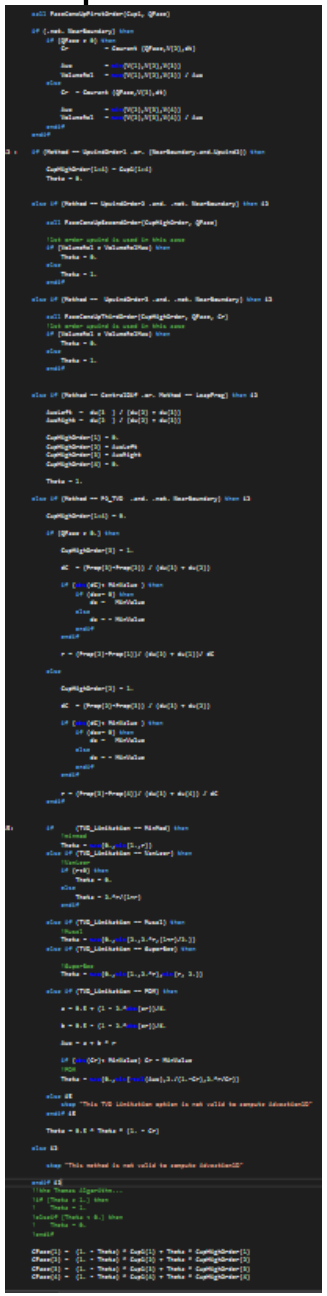
Run computations of output matrices

Optimizations – funtions module

ThomasZ

```
        !$OMP PARALLEL PRIVATE(J,I,K,II,MM,TID,VEC,AUX)
        TID = 1
        !$ TID = 1 + omp_get_thread_num() !
        VEC => Thomas%VEC(TID)
        !$OMP DO SCHEDULE(DYNAMIC,CHUNK)
do2 :   DO J = JLB, JUB
do1 :   DO I = ILB, IUB
            ! JPW: Changed 1 to KLB for consistency. Results should be the same as long as KLB = 1
            !VEC%W(KLB) =-Thomas%COEF3%F (I, J, KLB) / Thomas%COEF3%E(I, J, KLB)
            !VEC%G(KLB) = Thomas%TI(I, J, KLB) / Thomas%COEF3%E(I, J, KLB)
            ! JPW: Original
            if (WaterPoints(I, J, KUB) == 1) then
                VEC%W(KLB) = -Thomas%COEF3%F (I, J, 1) / Thomas%COEF3%E(I, J, 1)
                VEC%G(KLB) = Thomas%TI(I, J, 1) / Thomas%COEF3%E(I, J, 1)

    do3 :       DO K  = KLB+1, KUB+1
                    AUX = Thomas%COEF3%E(I, J, K) + Thomas%COEF3%D(I, J, K) * VEC%W(K-1)
                    IF (abs(AUX) > 0) then
                        VEC%W(K) = -Thomas%COEF3%F(I, J, K) / AUX
                        VEC%G(K) = (Thomas%TI(I, J, K) - Thomas%COEF3%D(I, J, K) * VEC%G(K-1)) / AUX
                    ELSE
                            !write(*,*) 'i, j, k: ', I, J, K
                            !write(*,*) 'ERROR: Instability in THOMASZ - ModuleFunctions - ERR10'
                    END IF
                END DO do3

                RES(I, J, KUB+1) = VEC%G(KUB+1)

    do4 :       DO II = KLB+1, KUB+1
                    MM            = KUB + KLB + 1 - II
                    RES(I, J, MM) = VEC%W(MM) * RES(I, J, MM+1) + VEC%G(MM)
                END DO do4
            endif

        END DO do1
        END DO do2
        !$OMP END DO NOWAIT
        !$OMP END PARALLEL
```

Optimizations – funtions module



Compute advection face

If a flux needs to be computed

Search numerical method

If TVD

Search limitation method

Ex : SUPERBEE

Compute coeficients for fluxes through cells

```
CFace (1:4) = 0.

Cup1        (1:4) = 0.
CupHighOrder(1:4) = 0.

if (QFace > 0) then

    Cup1(2) = 1.
    Cr      = Courant (QFace,V(2),dt)

    CupHighOrder(3) = 1.

    dC  = (Prop(3)-Prop(2)) / (du(3) + du(2))

    if (abs(dC)< MinValue ) then
        if (dc>= 0) then
            dc =   MinValue
        else
            dc = - MinValue
        endif
    endif

    r = (Prop(2)-Prop(1))/ ((du(2) + du(1)) * dC)

    Theta = max(0.,min(1.,2.*r),min(r, 2.))

    Theta = 0.5 * Theta * (1. - Cr)

else

    Cup1(3) = 1.
    Cr      = Courant (QFace,V(3),dt)

    CupHighOrder(2) = 1.

    dC  = (Prop(2)-Prop(3)) / (du(3) + du(2))

    if (abs(dC)< MinValue ) then
        if (dc>= 0) then
            dc =   MinValue
        else
            dc = - MinValue
        endif
    endif

    r = (Prop(3)-Prop(4))/ ((du(3) + du(4)) * dC)

    Theta = max(0.,min(1.,2.*r),min(r, 2.))

    Theta = 0.5 * Theta * (1. - Cr)
endif

CFace(2) =  (1. - Theta) * Cup1(2) + Theta * CupHighOrder(2)
```

Optimizations – Advection diffusion module

- If numerical methods are the same for all properties (usually the case):
  - Setmatrixvalue to 0 -> number of calls reduced by more than half
  - Compute diffusion coefficients on the first property and save into a matrix

- Same optimizations done for advection diffusion routines as for the hydrodynamic module

Comparisons - PCOMS

- Grid: 177 * 125 * 50 (1e6 grid points)
- DT : 60s
- 3D Baroclinic
- OBC from 2D barotropic domain – Tide from Fes2012
- Atmospheric forcing from hourly MM5
- Outputs once per day
- Intel(R) Core(TM) i7-4700MQ CPU @ 2.40GHz, 2401 Mhz, 4 Core(s), 8 Logical Processor(s)
- 8 GB RAM

PCOMS – Hydrodynamics = 190s

Module Hydrodynamic

Function / Call Stack	OLD	NEW	Diference(%)
MODIFYMATRIXESOUTPUT	2.4	0.2	91
MODIFY_DIFFUSION_UX_VY2	5.8	3.4	42
VELOCITY_EXPLICITFORCES	5.2	3.1	41
MODIFY_DIFFUSION_UY_VX2	4.8	4.1	15
COMPUTECARTESIANVERTVELOCITY	6.6	5.6	15
VEL_VERTADV_P2_TVD_SB_IMP_X_Y	6.0	5.2	14
MODIFY_ADVECTION_UY_VX2	4.6	4.0	13
MODIFY_ADVECTION_UX_VY2	2.8	2.5	12

Module Functions

Function / Call Stack	OLD	NEW	Diference(%)
COMPUTEADVECTIONFACE_TVD_SUPERBEE	6.4	3.2	49
COMPUTEADVECTION1D_TVD_SUPERBEE_2	9.9	5.5	45
THOMASZ_NEWTYPE2	8.7	5.7	35
SETMATRIXVALUES3D_R8_CONSTANT	5.7	3.9	32

Module AdvectionDiffusion

Function / Call Stack	OLD	NEW	Diference(%)
HORIZONTALDIFFUSIONYY2	1.8	1.2	36
HORIZONTALDIFFUSIONXX2	2.0	1.4	28
VERTICALDIFFUSION2	3.3	2.6	21
HORIZONTALADVECTIONXX	1.7	1.4	20
ADVECTIONDIFFUSIONITERATION	1.5	1.3	13

Total improvement time : 13%

```
if (.not. Me%ComputeOptions%ConservativeHorDif)then
    if (Me%ComputeOptions%SlippingCondition) then
        if ( .not. Me%CyclicBoundary%ON)  then
            if (.not.Me%ComputeOptions%MomentumDischarge) then
                GoForOptimized = .true.
            endif
        endif
    endif
endif
```

PCOMS – Water Quality = 492s

Module Functions

Function / Call Stack	OLD	NEW	Diference(%)
COMPUTEADVECTION1D_TVD_SUPERBEE_2	79.60	44.87	44
THOMASZ_NEWTYPE2	47.44	30.24	36
SETMATRIXVALUES3D_R8_CONSTANT	33.33	9.84	70

Module AdvectionDiffusion

Function / Call Stack	OLD	NEW	Diference(%)
VERTICALADVECTION	28.2182	27.3944	3
VERTICALDIFFUSION2	25.7763	21.4439	17
HORIZONTALADVECTIONYY	14.2955	11.9312	17
HORIZONTALADVECTIONXX	14.406	11.5727	20
HORIZONTALDIFFUSIONYY2	14.6398	10.1478	31
HORIZONTALDIFFUSIONXX2	15.3964	9.68946	37

Total improvement time : 22%

Tagus 3D

- Grid: 120 * 145 * 50 (9e5 grid points)
- DT : 6s
- 3D Baroclinic
- OBC from PCOMS
- Atmospheric forcing from hourly 3km WRF
- Outputs once per day

Hydrodynamic 3D simulation speedup : 17% **WQ 3D simulation improvement : 26%**

Tagus + Sado 3D

- Grid: 260 * 355 * 42 (4e6 grid points)
- DT : 10s
- 3D Baroclinic
- OBC from CMEMS hourly 3D
- Atmospheric forcing from hourly 2,5km from IPMA
- 3D oututs every 3h; Surface outputs every hour

Hydrodynamic 3D simulation speedup : 21% **WQ 3D simulation improvement : ??**

Activating the optimizations

- In the hydrodynamic.dat -> “GLOBAL_OPT : 1”
- In the WaterProperties.dat -> “OPTIMIZE : 1”

Other Keywords can be used to enable/disable optimization of the different **hydrodynamic** module routines(Defined at the Hydrodynamic.dat):

OPTIMIZE_INERTIAFORCES_OPT
OPTIMIZE_VEL_EXP_FORCES
OPTIMIZE_VERT_WATER_FLOW_OPT
OPTIMIZE_CARTESIAN_VERT_VELOCITY_OPT
OPTIMIZE_ADVECTION_H
OPTIMIZE_ADVECTION_V
OPTIMIZE_DIFFUSION_H
OPTIMIZE_DIFFUSION_V
OPTIMIZE_MATRIXES_OUTPUT

Conclusions

- Avoid if statements inside loops as much as possible : reducing the number of ifs increased loop performance by 15-30%
- Divide big loops which need to acess many 3D matrices – impacts on scalability (vertical velocity example)
- Create routines for each numerical scheeme instead of using ifs - 45% performance increase in advection modules for the case of TVD-SUPERBEE. For the remaing schemes the impact will be bigger.
- Avoid any unnecessary computations (such as in matrices_output routine), as its corresponding impact will grow with matrix size and sizes are increasing more and more.