```
{
    word alphaScheme("div(phi,alpha)");
    word alpharScheme("div(phirb,alpha)");

    // Standard face-flux compression coefficient
    surfaceScalarField phic(interface.cAlpha()*mag(phi/mesh.magSf()));

    // Add the optional isotropic compression contribution
    if (icAlpha > 0)
    {
        phic *= (1.0 - icAlpha);
        phic += (interface.cAlpha()*icAlpha)*fvc::interpolate(mag(U));
    }

    // Do not compress interface at non-coupled boundary faces
    // (inlets, outlets etc.)
    forAll(phic.boundaryField(), patchi)
    {
        fvsPatchScalarField& phicp = phic.boundaryField()[patchi];

        if (!phicp.coupled())
        {
            phicp == 0;
        }
    }

    tmp<surfaceScalarField> tphiAlpha;

    if (MULESCorr)
    {
        fvScalarMatrix alpha1Eqn
        (
            #ifdef LTSSOLVE
            fv::localEulerDdtScheme<scalar>(mesh, rDeltaT.name()).fvmDdt(alpha1)
            #else
            fv::EulerDdtScheme<scalar>(mesh).fvmDdt(alpha1)
            #endif
          + fv::gaussConvectionScheme<scalar>
            (
                mesh,
                phi,
                upwind<scalar>(mesh, phi)
            ).fvmDiv(phi, alpha1)
        );

        alpha1Eqn.solve();

        Info<< "Phase-1 volume fraction = "
            << alpha1.weightedAverage(mesh.Vsc()).value()
            << "  Min(alpha1) = " << min(alpha1).value()
            << "  Max(alpha1) = " << max(alpha1).value()
            << endl;

        tmp<surfaceScalarField> tphiAlphaUD(alpha1Eqn.flux());
        tphiAlpha = tmp<surfaceScalarField>
        (
            new surfaceScalarField(tphiAlphaUD())
        );

        if (alphaApplyPrevCorr && tphiAlphaCorr0.valid())
        {
            Info<< "Applying the previous iteration compression flux" << endl;
            #ifdef LTSSOLVE
            MULES::LTScorrect(alpha1, tphiAlpha(), tphiAlphaCorr0(), 1, 0);
            #else
            MULES::correct(alpha1, tphiAlpha(), tphiAlphaCorr0(), 1, 0);
            #endif
```

```
            tphiAlpha() += tphiAlphaCorr0();
        }

        // Cache the upwind-flux
        tphiAlphaCorr0 = tphiAlphaUD;

        alpha2 = 1.0 - alpha1;

        interface.correct();
    }

    for (int aCorr=0; aCorr<nAlphaCorr; aCorr++)
    {
        surfaceScalarField phir(phic*interface.nHatf());

        tmp<surfaceScalarField> tphiAlphaUn
        (
            fvc::flux
            (
                phi,
                alpha1,
                alphaScheme
            )
          + fvc::flux
            (
               -fvc::flux(-phir, alpha2, alpharScheme),
                alpha1,
                alpharScheme
            )
        );

        if (MULESCorr)
        {
            tmp<surfaceScalarField> tphiAlphaCorr(tphiAlphaUn() - tphiAlpha());
            volScalarField alpha10(alpha1);

            #ifdef LTSSOLVE
            MULES::LTScorrect(alpha1, tphiAlphaUn(), tphiAlphaCorr(), 1, 0);
            #else
            MULES::correct(alpha1, tphiAlphaUn(), tphiAlphaCorr(), 1, 0);
            #endif

            // Under-relax the correction for all but the 1st corrector
            if (aCorr == 0)
            {
                tphiAlpha() += tphiAlphaCorr();
            }
            else
            {
                alpha1 = 0.5*alpha1 + 0.5*alpha10;
                tphiAlpha() += 0.5*tphiAlphaCorr();
            }
        }
        else
        {
            tphiAlpha = tphiAlphaUn;

            #ifdef LTSSOLVE
            MULES::explicitLTSSolve(alpha1, phi, tphiAlpha(), 1, 0);
            #else
            MULES::explicitSolve(alpha1, phi, tphiAlpha(), 1, 0);
            #endif
        }

        alpha2 = 1.0 - alpha1;

        interface.correct();
    }
```

```
    rhoPhi = tphiAlpha()*(rho1 - rho2) + phi*rho2;

// ADDITION
    rhoCpPhi = tphiAlpha()*(rho1*cp1 - rho2*cp2) + phi*rho2*cp2;
// END of ADDITION

    if (alphaApplyPrevCorr && MULESCorr)
    {
        tphiAlphaCorr0 = tphiAlpha() - tphiAlphaCorr0;
    }

    Info<< "Phase-1 volume fraction = "
        << alpha1.weightedAverage(mesh.Vsc()).value()
        << "  Min(alpha1) = " << min(alpha1).value()
        << "  Max(alpha1) = " << max(alpha1).value()
        << endl;
}
```

rhoCpPhi を追加