Macros abstracting memory region decorators (`__global`, `__shared`, etc.)

Support calling map from C++ with overloaded math function `sqrt` being resolved for real_type via `ADL`

Store "frequently" used attributes in thread-local memory to avoid multiple reads from `__global` memory

Consistency Checks -> `Disabled / NO-OP` in Release Mode

Update attributes for $p_{ii} \equiv (\mathbf{Q})\,[ii]$ -> Write back to `SIXTRL_PARTICLE_ARGPTR_DEC` memory

Macros abstracting language features

```
NS(track_result_type) NS(Track_particle_drift_exact)(
    SIXTRL_PARTICLES_ARGPTR_DEC NS(Particles)* SIXTRL_RESTRICT p,
    NS(particles_num_type) const ii,
    SIXTRL_BE_ARGPTR_DEC const NS(DriftExact) *const
        SIXTRL_RESTRICT drift ) SIXTRL_NOEXCEPT
{
    typedef NS(particle_real_type) real_type;

    #if defined( __cplusplus )
    using std::sqrt; /* ADL */
    #endif /* defined( __cplusplus ) */

    real_type const length = NS(DriftExact_length)( drift );
    real_type const px     = NS(Particles_px)( p, ii );
    real_type const py     = NS(Particles_py)( p, ii );

    real_type const one_plus_delta =
            NS(Particles_delta)( p, ii ) + ( real_type )1.;

    real_type const lpzi = length / sqrt(
            one_plus_delta * one_plus_delta -
            ( px * px + py * py ) );

    SIXTRL_ASSERT( p != SIXTRL_NULLPTR );
    SIXTRL_ASSERT( drift != SIXTRL_NULLPTR );
    SIXTRL_ASSERT( ( one_plus_delta * one_plus_delta ) >
                   ( px * px + py * py ) );

    NS(Particles_add_to_x)( p, ii, px * lpzi );
    NS(Particles_add_to_y)( p, ii, py * lpzi );
    NS(Particles_add_to_s)( p, ii, length );
    NS(Particles_add_to_zeta)( p, ii,
            NS(Particles_rvv)( p, ii ) * length -
            one_plus_delta * lpzi );

    return SIXTRL_TRACK_SUCCESS;
}
```