```
constexpr int NUM_RHS = 9;
constexpr int N = 32;
double A[N][N];
double B[N][N];
double x[NUM_RHS][N];
double y[NUM_RHS][N];
double z[NUM_RHS][N];
// Initialize arrays [...omitted...]
for (int i=0; i<NUM_RHS; ++i) {
  for (int j=0; j<N; ++j) {
    for (int k=0; k<N; ++k) {
      y[i][j] += A[j][k]*x[i][k]
    }
  }
  for (int j=0; j<N; ++j) {
    for (int k=0; k<N; ++k) {
      z[i][j] += B[j][k]*y[i][k]
    }
  }
}
```

$\Longrightarrow$

```
constexpr int NUM_RHS = 9;
constexpr int N = 32;
using Kokkos::parallel_for;
using Kokkos::parallel_reduce;
using TP = Kokkos::TeamPolicy<ExecSpace>;
Kokkos::View<double[N][N]> A("A"), B("B");
Kokkos::View<double[NUM_RHS][N]> x("x"), y("y"), z("z");
// Initialize arrays [...omitted...]
// Create policy: let Kokkos decide team size
TP policy(NUM_RHS, Kokkos::AUTO());
parallel_for(policy,
  KOKKOS_LAMBDA(TP::member_type member) {
    const int i = member.league_rank();
    parallel_for(Kokkos::TeamThreadRange(member,N),
      [=](const int& j){
        parallel_reduce(Kokkos::ThreadVectorRange(member,N),
          [=](const int& k, double& accumulator){
            accumulator += A(j,k)*x(i,k);
        },y(i,j));
    });
    member.team_barrier();
    parallel_for(Kokkos::TeamThreadRange(member,N),
      [=](const int& j){
        parallel_reduce(Kokkos::ThreadVectorRange(member,N),
          [=](const int& k, double& accumulator){
            accumulator += B(j,k)*y(i,k);
        },z(i,j));
    });
});
```