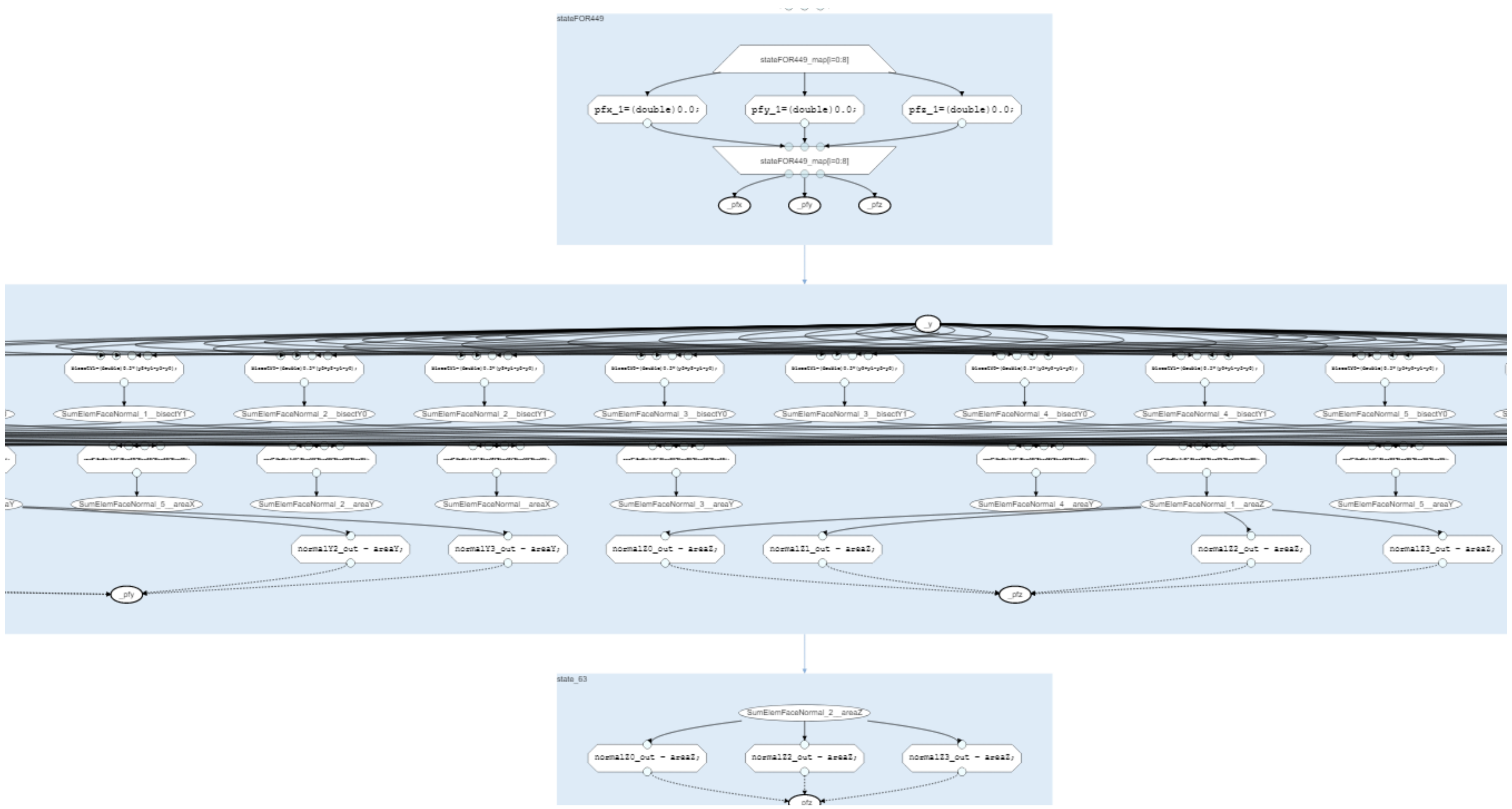
Alexandru Calotoiu

DaFlEx

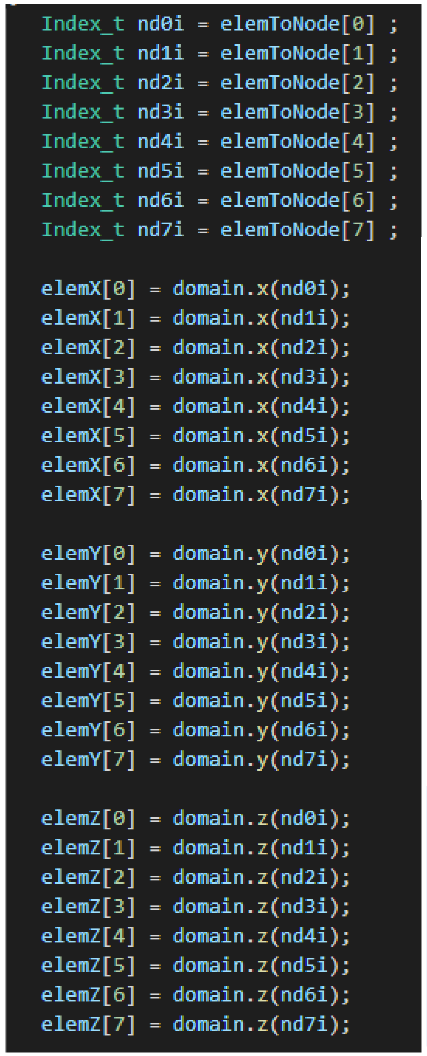
SPCL

Lulesh

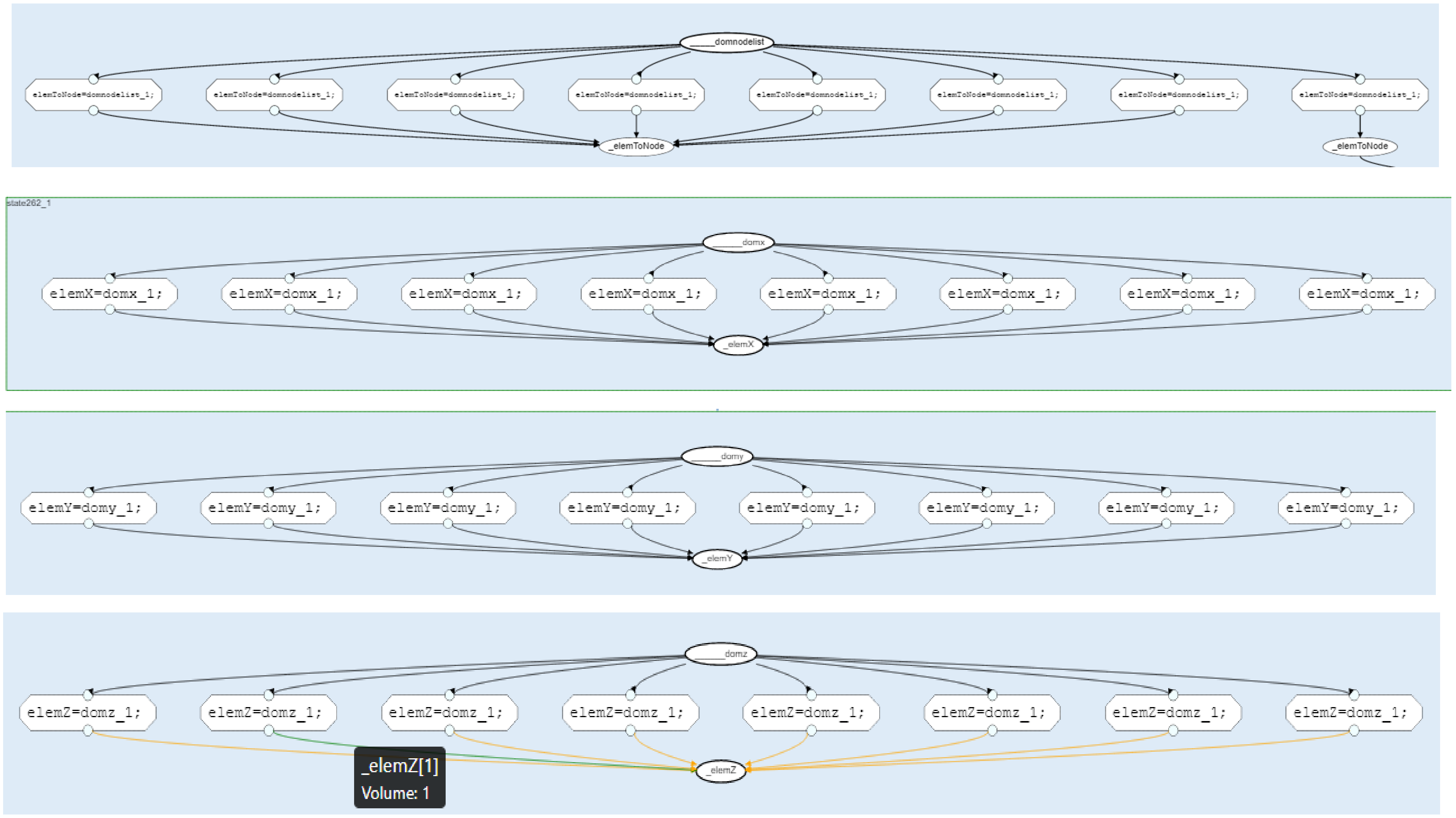
- SEDOV Blast problem solver
- Focus on ~1000 lines of code
- Nominally C++


stateFOR449
stateFOR449_map[i=0:8]
pfx_1=(double)0.0;
pfy_1=(double)0.0;
pfz_1=(double)0.0;
stateFOR449_map[i=0:8]
_pfx
_pfy
_pfz
SumElemFaceNormal_1__bisectY1
SumElemFaceNormal_2__bisectY0
SumElemFaceNormal_2__bisectY1
SumElemFaceNormal_3__bisectY0
SumElemFaceNormal_3__bisectY1
SumElemFaceNormal_4__bisectY0
SumElemFaceNormal_4__bisectY1
SumElemFaceNormal_5__bisectY0
SumElemFaceNormal_5__areaX
SumElemFaceNormal_2__areaY
SumElemFaceNormal__areaX
SumElemFaceNormal_3__areaY
SumElemFaceNormal_4__areaY
SumElemFaceNormal_1__areaZ
SumElemFaceNormal_5__areaY
normalY2_out = areaY;
normalY3_out = areaY;
normalZ0_out = areaZ;
normalZ1_out = areaZ;
normalZ2_out = areaZ;
normalZ3_out = areaZ;
_pfy
_pfz
state_63
SumElemFaceNormal_2__areaZ
normalZ0_out = areaZ;
normalZ2_out = areaZ;
normalZ3_out = areaZ;
_pfz

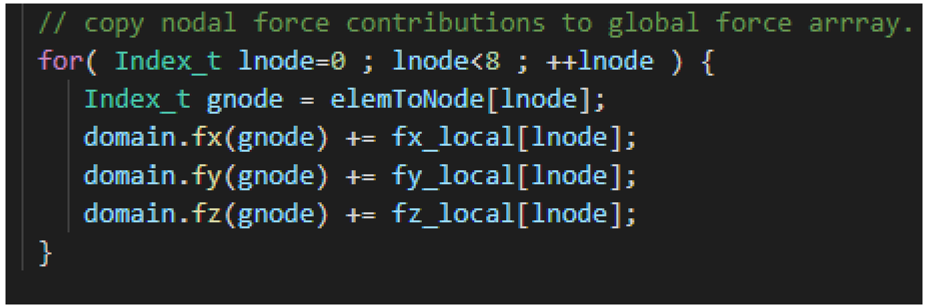
Indirect accesses



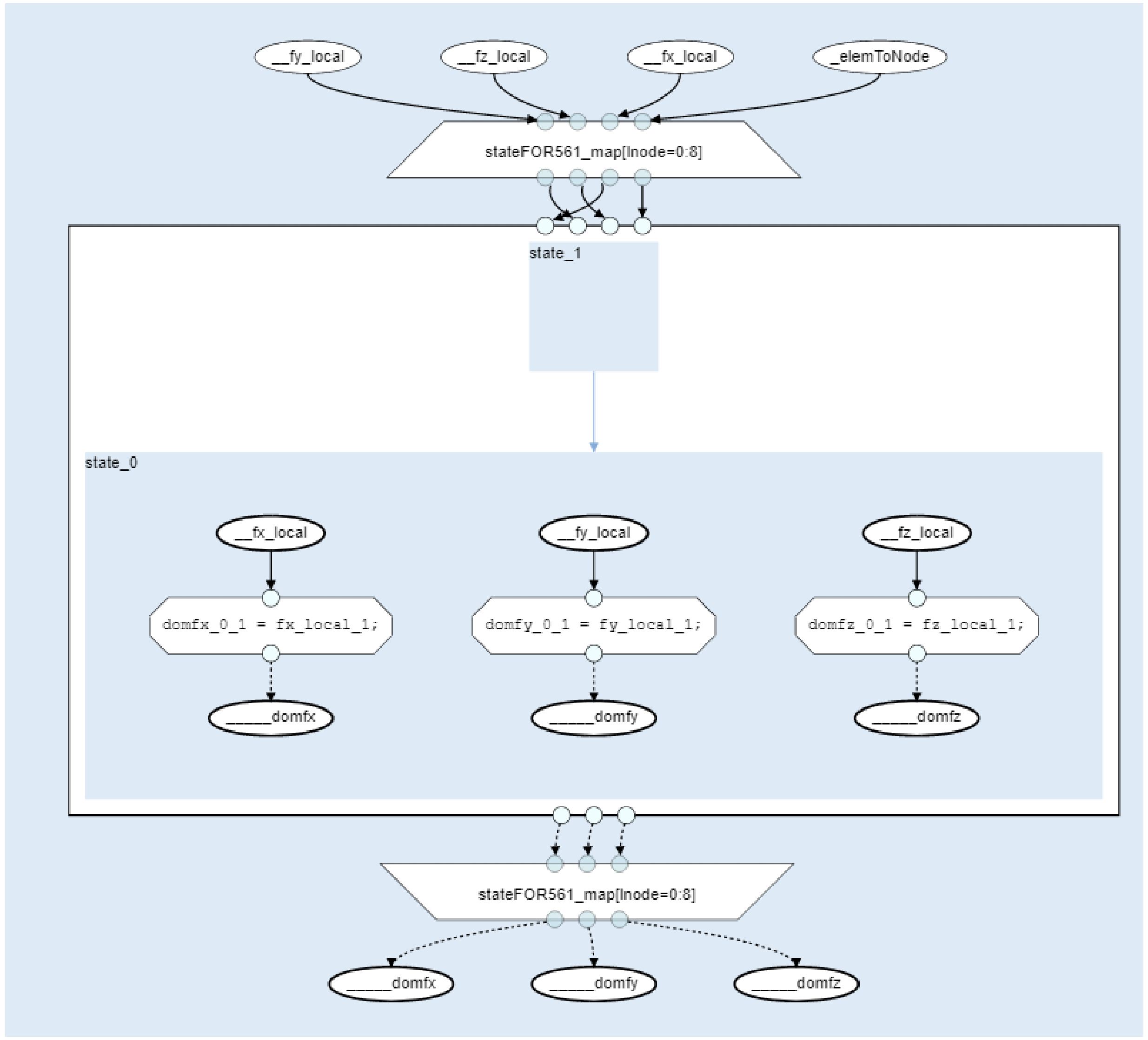
```
Index_t nd0i = elemToNode[0] ;
Index_t nd1i = elemToNode[1] ;
Index_t nd2i = elemToNode[2] ;
Index_t nd3i = elemToNode[3] ;
Index_t nd4i = elemToNode[4] ;
Index_t nd5i = elemToNode[5] ;
Index_t nd6i = elemToNode[6] ;
Index_t nd7i = elemToNode[7] ;

elemX[0] = domain.x(nd0i);
elemX[1] = domain.x(nd1i);
elemX[2] = domain.x(nd2i);
elemX[3] = domain.x(nd3i);
elemX[4] = domain.x(nd4i);
elemX[5] = domain.x(nd5i);
elemX[6] = domain.x(nd6i);
elemX[7] = domain.x(nd7i);

elemY[0] = domain.y(nd0i);
elemY[1] = domain.y(nd1i);
elemY[2] = domain.y(nd2i);
elemY[3] = domain.y(nd3i);
elemY[4] = domain.y(nd4i);
elemY[5] = domain.y(nd5i);
elemY[6] = domain.y(nd6i);
elemY[7] = domain.y(nd7i);

elemZ[0] = domain.z(nd0i);
elemZ[1] = domain.z(nd1i);
elemZ[2] = domain.z(nd2i);
elemZ[3] = domain.z(nd3i);
elemZ[4] = domain.z(nd4i);
elemZ[5] = domain.z(nd5i);
elemZ[6] = domain.z(nd6i);
elemZ[7] = domain.z(nd7i);
```



Update detection

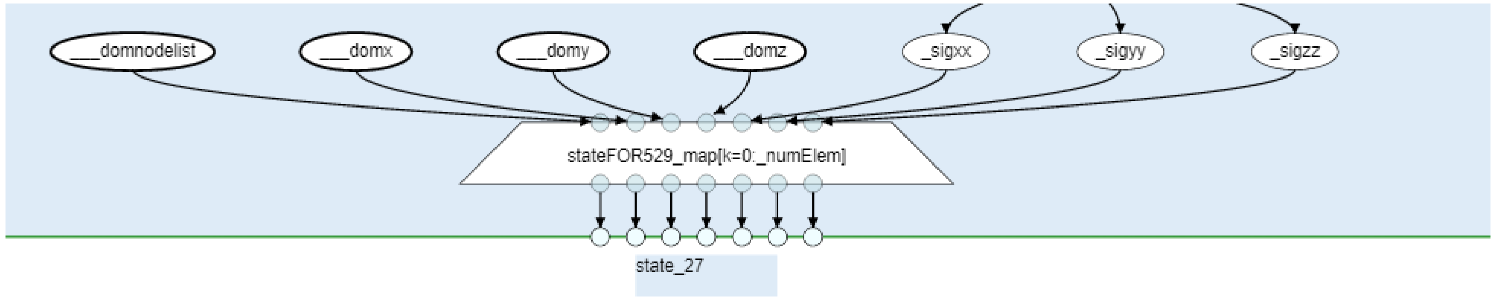


```
// copy nodal force contributions to global force arrray.
for( Index_t lnode=0 ; lnode<8 ; ++lnode ) {
    Index_t gnode = elemToNode[lnode];
    domain.fx(gnode) += fx_local[lnode];
    domain.fy(gnode) += fy_local[lnode];
    domain.fz(gnode) += fz_local[lnode];
}
```



Complete autoparallelization!

```
#pragma omp parallel for firstprivate(numElem)
  for( Index_t k=0 ; k<numElem ; ++k )
  {
    const Index_t* const elemToNode = domain.nodelist(k);
    Real_t B[3][8] ;// shape function derivatives
    Real_t x_local[8] ;
    Real_t y_local[8] ;
    Real_t z_local[8] ;

    // get nodal coordinates from global arrays and copy into local arrays.
    CollectDomainNodesToElemNodes(domain, elemToNode, x_local, y_local, z_local);

    // Volume calculation involves extra work for numerical consistency
    CalcElemShapeFunctionDerivatives(x_local, y_local, z_local,
                                     B, &determ[k]);

    CalcElemNodeNormals( B[0] , B[1], B[2],
                         x_local, y_local, z_local );

    if (numthreads > 1) {
       // Eliminate thread writing conflicts at the nodes by giving
       // each element its own copy to write to
       SumElemStressesToNodeForces( B, sigxx[k], sigyy[k], sigzz[k],
                                    &fx_elem[k*8],
                                    &fy_elem[k*8],
                                    &fz_elem[k*8] ) ;
    }
    else {
       SumElemStressesToNodeForces( B, sigxx[k], sigyy[k], sigzz[k],
                                    fx_local, fy_local, fz_local ) ;

       // copy nodal force contributions to global force arrray.
       for( Index_t lnode=0 ; lnode<8 ; ++lnode ) {
          Index_t gnode = elemToNode[lnode];
          domain.fx(gnode) += fx_local[lnode];
          domain.fy(gnode) += fy_local[lnode];
          domain.fz(gnode) += fz_local[lnode];
       }
    }
  }
```



SDFG too large for slide…