Puffin: An Embedded Domain-Specific Language for Existing Unstructured Hydrodynamics Codes

WOLFHPC 2015

Christopher Earl

November 16th, 2015

LLNL-PRES-679383

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Overview

Puffin is an experimental C++98 template meta-programming language, targeting:

Unstructured hydrodynamics calculations

Initially LULESH 2.0, a Lawrence Livermore proxy application

Multiple architectures

Currently, just supports single-thread CPU
Eventually, multi-thread CPU, GPU, and Xeon Phi

Existing (C++) projects / codes

Adding Puffin to an Existing Project

In relevant C++ header and source files, add:

```
#include <Puffin.h>
```

To the existing build system, add dependencies to `Puffin*.h` (16 files)

Future additions may require optional dependencies

For example, NVIDIA's nvcc will be required for GPU execution

Puffin Aspects

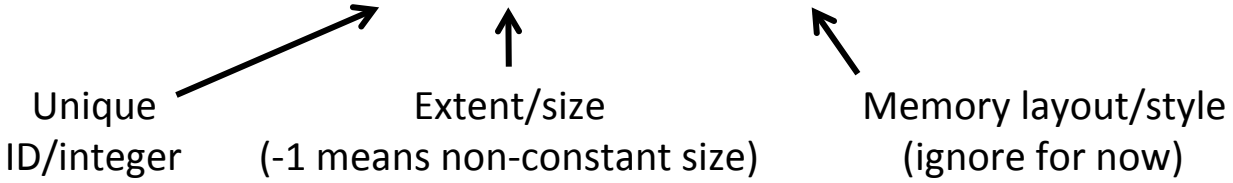
Puffin's basic "space" abstraction is an **aspect**:

Data/memory "space:" dimensions of arrays

Iteration "space:" Loop dimensions

Aspects are user/project defined:

```
typedef PuffinAspect<0,  3, PuffinStyleContainer> DimAspect;
typedef PuffinAspect<1, -1, PuffinStyleArray>     NodeAspect;
typedef PuffinAspect<2, -1, PuffinStyleArray>     ElemAspect;
```

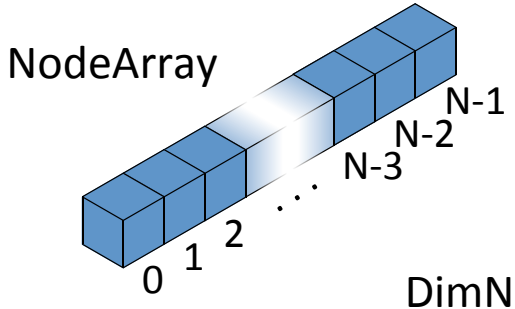


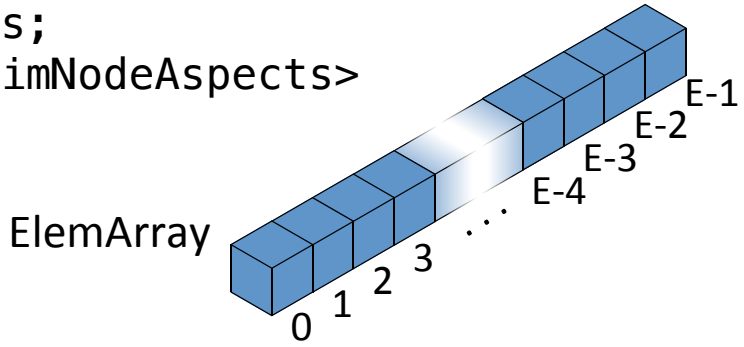
Puffin Arrays

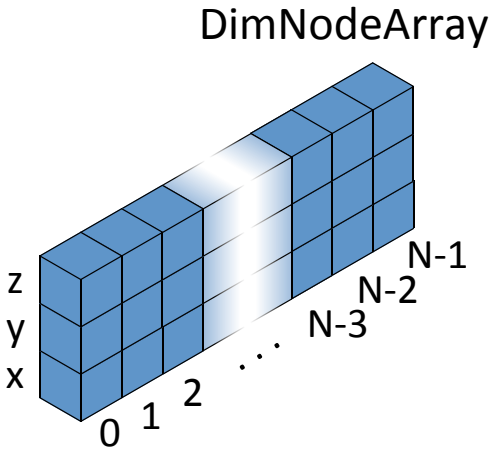
Puffin's Basic Variables

Single- and multiple-aspect arrays:

```
typedef PuffinArray<NodeAspect>
        NodeArray;
typedef PuffinArray<ElemAspect>
        ElemArray;

typedef PuffinAspects<DimAspect, NodeAspect>
        DimNodeAspects;
typedef PuffinArray<DimNodeAspects>
        DimNodeArray;
```





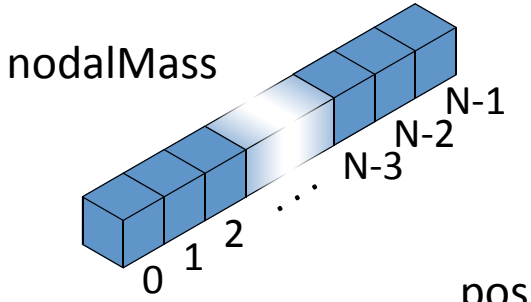


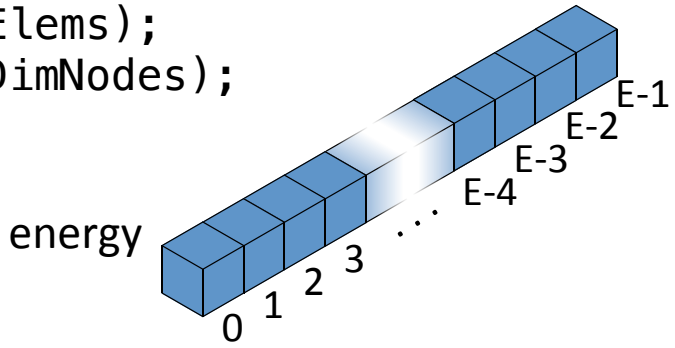
Puffin Arrays

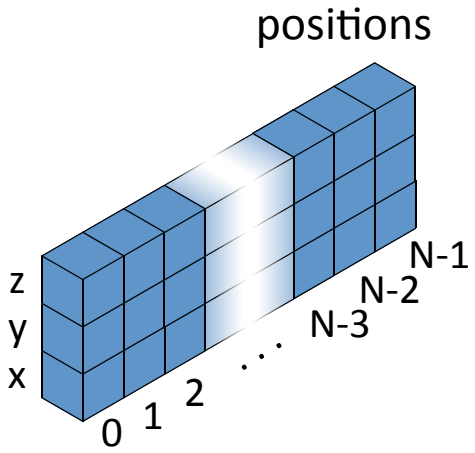
Puffin's Basic Variables

Declaration of Puffin arrays:

```
NodeAspect      Nodes    (N);
ElemAspect      Elem     (E);
DimAspect       Dim;
DimNodeAspects DimNodes(Dim, Nodes);

NodeArray     nodalMass(Nodes);
ElemArray     energy    (Elems);
DimNodeArray positions(DimNodes);
```



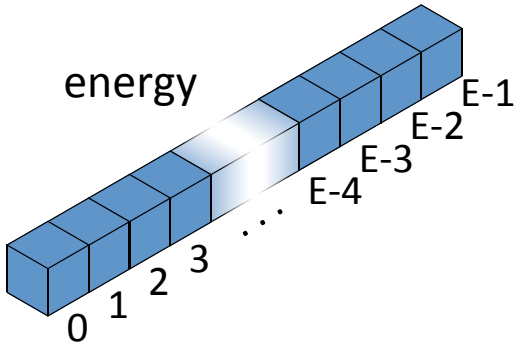




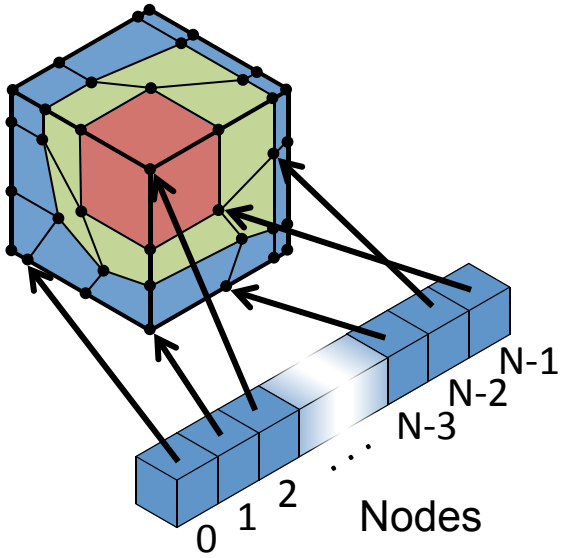
Puffin Provided Arrays

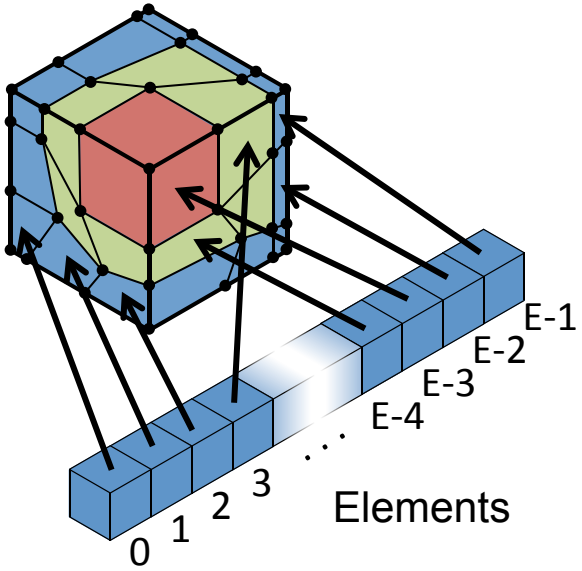
To use existing arrays with Puffin:

```
typedef PuffinProvidedArray<ElemAspect>
        ElemProvidedArray;

double* m_energy = ...; //lulesh allocation

ElemProvidedArray energy(Elems, m_energy);
```



Unstructured Meshes





Order of values (nodes, elements, etc.) in arrays need not correspond to any ordering of simulated space.

Single Assignment Example

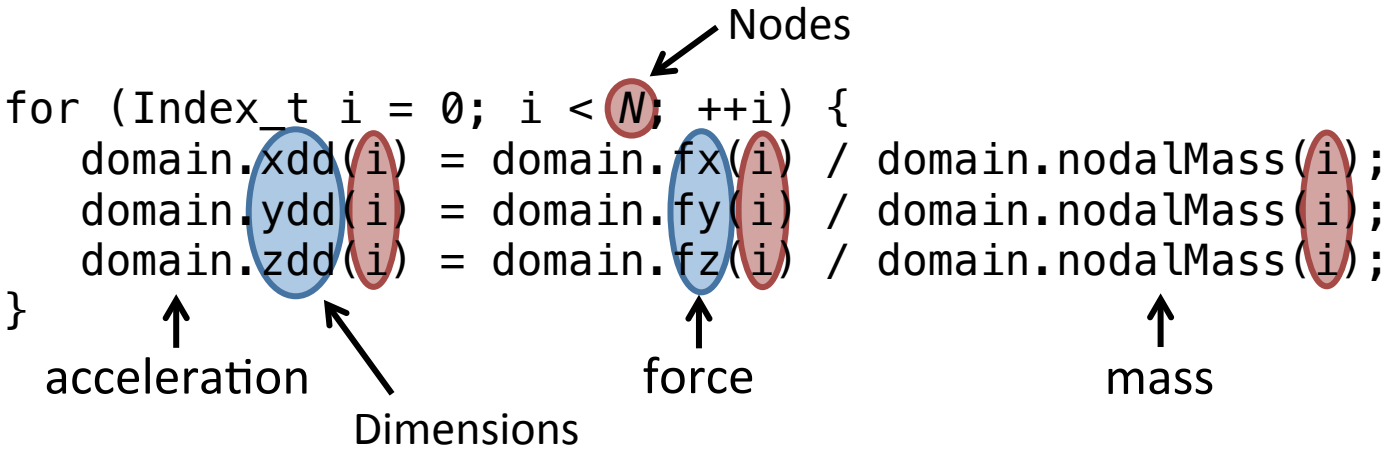
Acceleration calculation in LULESH 2.0:

```
for (Index_t i = 0; i < N; ++i) {
   domain.xdd(i) = domain.fx(i) / domain.nodalMass(i);
   domain.ydd(i) = domain.fy(i) / domain.nodalMass(i);
   domain.zdd(i) = domain.fz(i) / domain.nodalMass(i);
}
```

acceleration force mass

Single Assignment Example

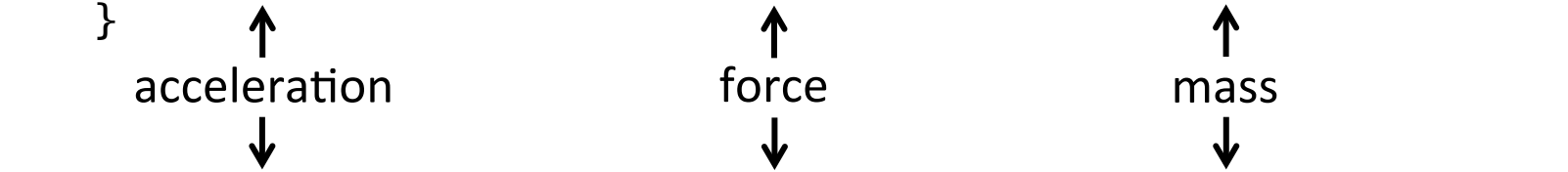
Acceleration calculation in LULESH 2.0:



Single Assignment Example

Acceleration calculation in LULESH 2.0:

```
for (Index_t i = 0; i < N; ++i) {
   domain.xdd(i) = domain.fx(i) / domain.nodalMass(i);
   domain.ydd(i) = domain.fy(i) / domain.nodalMass(i);
   domain.zdd(i) = domain.fz(i) / domain.nodalMass(i);
}
```



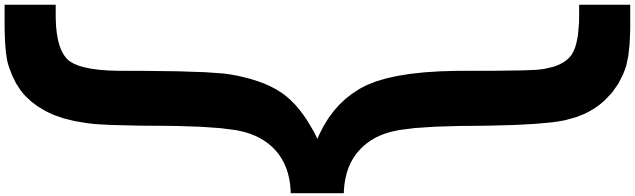
Single Assignment Example

How it works

```
domain.ddd()[Nodes][Dim] <<= domain.fd() / domain.nodalMass();
```

`PuffinLhs<…>`

`PuffinExpression<PuffinDivide<…>, …>`



`PuffinAssignment<…>` (Immediately executes)

Multiple Assignment Example

Plus temporary values and summation

Update of volume derivative and strains in LULESH 2.0:

```
for( Index_t k=0 ; k<E ; ++k ) {
   Real_t vdov = domain.dxx(k) + domain.dyy(k) + domain.dzz(k) ;
   Real_t vdovthird = vdov/Real_t(3.0) ;

   domain.vdov(k) = vdov ;
   domain.dxx(k) -= vdovthird ;
   domain.dyy(k) -= vdovthird ;
   domain.dzz(k) -= vdovthird ;
}
```

Temp value → `Real_t vdovthird = vdov/Real_t(3.0) ;`

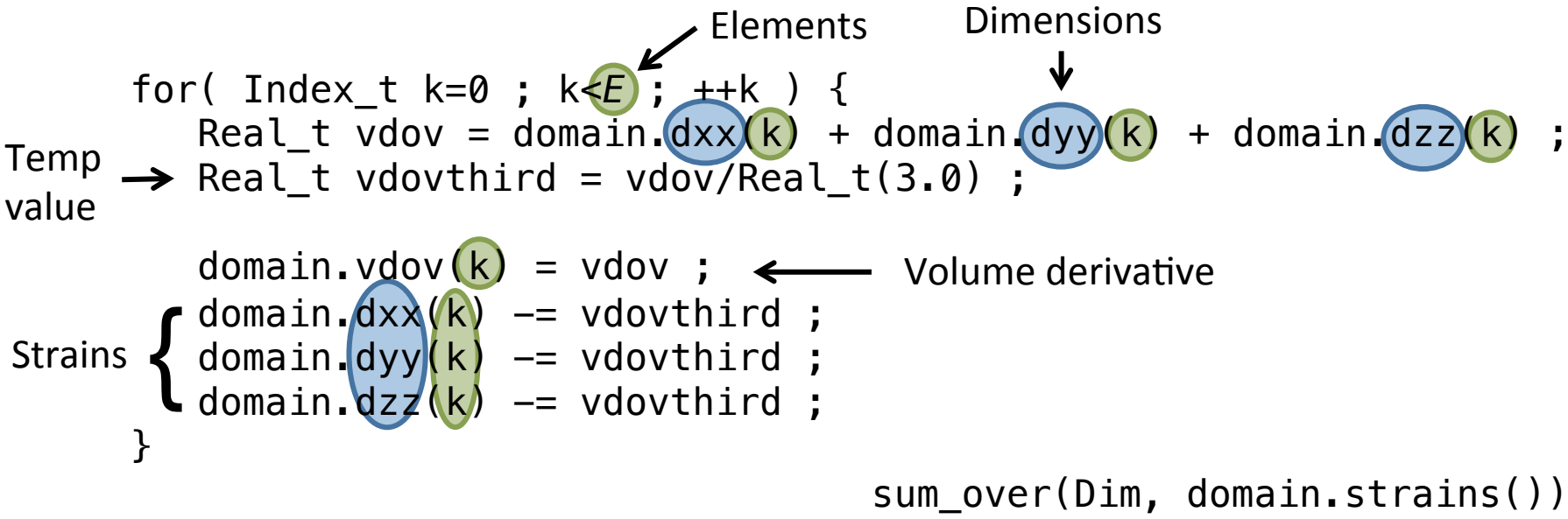
`domain.vdov(k) = vdov ;` ← Volume derivative

Strains { `domain.dxx(k) -= vdovthird ;` `domain.dyy(k) -= vdovthird ;` `domain.dzz(k) -= vdovthird ;`

Multiple Assignment Example

Plus temporary values and summation

Update of volume derivative and strains in LULESH 2.0:



Multiple Assignment Example

Plus temporary values and summation

Update of volume derivative and strains in LULESH 2.0:

```
ScalarValue vdovthird;

puffin_foreach(Elems)
   (domain.vdov()          |= sum_over(Dim, domain.strains()))
   (vdovthird              |= domain.vdov() / 3.0)
   (domain.strains()[Dim] |= domain.strains() – vdovthird)
   .execute();
```

Multiple Assignment Example

How it works

```
puffin_foreach(Elems)  <----------  Foreach Functor
    (domain.vdov()            |= ...) <--  PuffinAssignment<…>
    (vdovthird                |= ...) <--  PuffinAssignment<…>
    (domain.strains()[Dim]    |= ...) <--  PuffinAssignment<…>
    .execute();
```

Multiple Assignment Example

How it works

```
puffin_foreach(Elems)
   (domain.vdov()          |= ...)
   (vdovthird              |= ...)
   (domain.strains()[Dim]  |= ...)
   .execute();
```

} Foreach Functor (1 assignment)

Multiple Assignment Example

How it works

```
puffin_foreach(Elems)
   (domain.vdov()          |= ...)
   (vdovthird              |= ...)
   (domain.strains()[Dim]  |= ...)
   .execute();
```

} Foreach Functor (2 assignments)

Multiple Assignment Example

How it works

```
puffin_foreach(Elems)
   (domain.vdov()          |= ...)
   (vdovthird              |= ...)
   (domain.strains()[Dim]  |= ...)
   .execute();
```

Foreach Functor (3 assignments)

Execution of all assignments happens in a single loop within `execute()` call.

Indirect Array Example

Start of EOS calculation in LULESH 2.0:

```
for (Index_t i = 0; i < domain.regElemSize(r) ; ++i) {
   Index_t elem = domain.regElemlist(r)[i];
   Real_t vchalf ;
   compression[i]  = Real_t(1.)  / vnewc[elem] - Real_t(1.);
   vchalf          = vnewc[elem] - delvc[i]    * Real_t(.5);
   compHalfStep[i] = Real_t(1.)  / vchalf      - Real_t(1.);
}
```

Region indirection

Indirect Array Example

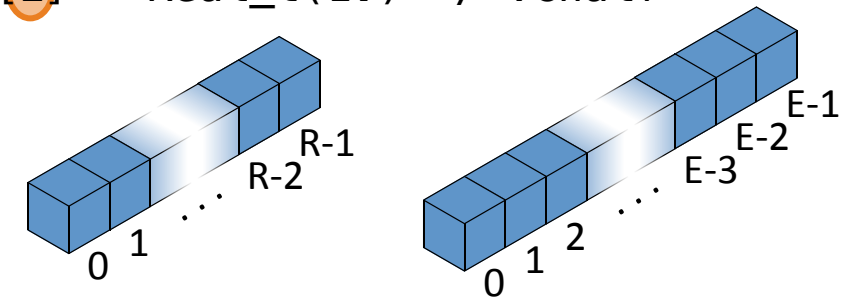
Start of EOS calculation in LULESH 2.0:

Element

Current region

```
for (Index_t i = 0; i < domain.regElemSize(r) ; ++i) {
   Index_t elem = domain.regElemlist(r)[i];
   Real_t vchalf ;
   compression[i]  = Real_t(1.)  / vnewc[elem] - Real_t(1.);
   vchalf          = vnewc[elem] - delvc[elem] * Real_t(.5);
   compHalfStep[i] = Real_t(1.)  / vchalf      - Real_t(1.);
}
```

Region indirection



Indirect Array Example

Definition of region aspect and array, related to ElemAspect:

```
typedef PuffinAspect<3, -1, PuffinStyleRelatedArray<ElemAspect> >
        SingleRegionAspect;

typedef PuffinArray<SingleRegionAspect>
        SingleRegionArray;
```

Based on *r*, an aspect for the current region (CurReg) can be instantiated:

```
SingleRegionAspect CurReg(domain.regElemSize(r),
                          domain.regElemlist(r));
```

Indirect Array Example

Start of EOS calculation in LULESH 2.0:

```
ElemProvidedArray vnewc_p     (Elems, vnewc);
ElemProvidedArray delvc_p     (Elems, delvc);
SingleRegionArray compression (CurReg);
SingleRegionArray compHalfStep(CurReg);
ScalarValue       vchalf;

puffin_foreach(CurReg)
   (compression  |= 1.0     / vnewc_p - 1.0)
   (vchalf       |= vnewc_p - delvc_p * 0.5)
   (compHalfStep |= 1.0     / vchalf  - 1.0)
   .execute();
```

Affiliation Example

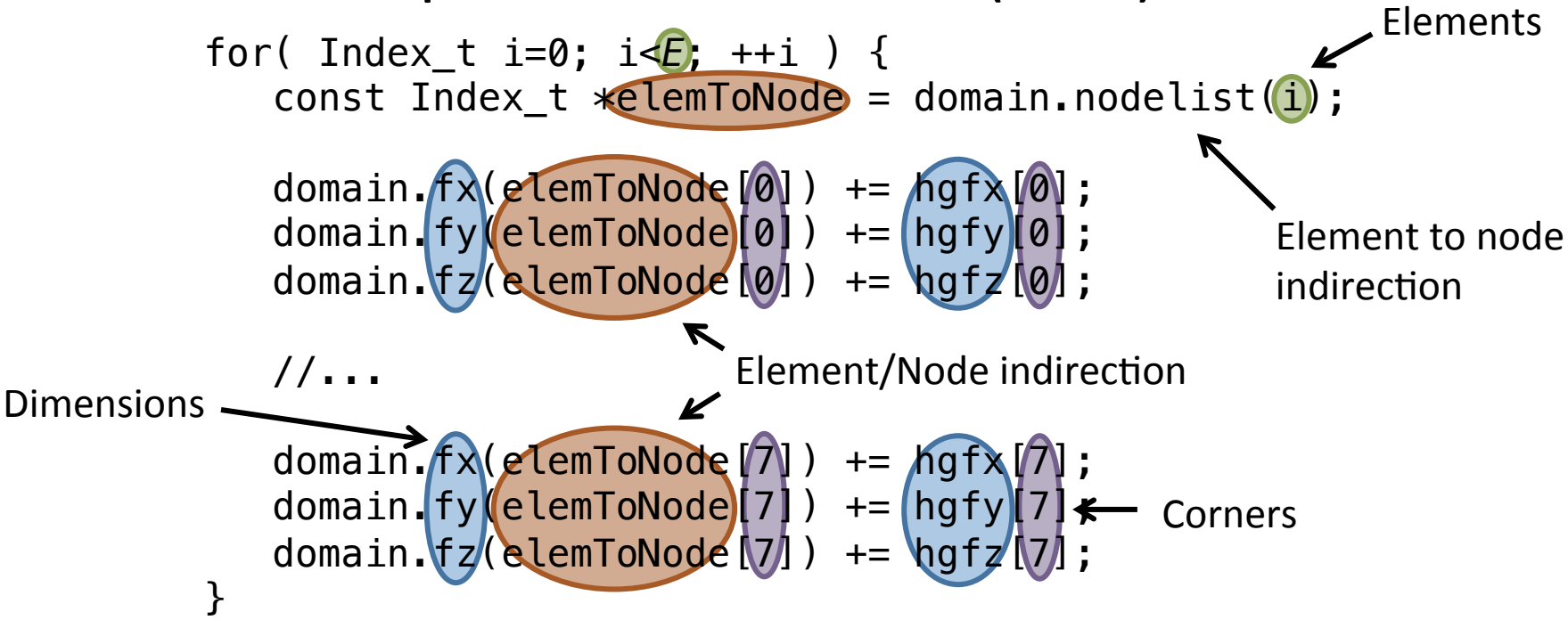
Update of force in LULESH 2.0 (scatter):

```
for( Index_t i=0; i<E; ++i ) {
   const Index_t *elemToNode = domain.nodelist(i);

   domain.fx(elemToNode[0]) += hgfx[0];
   domain.fy(elemToNode[0]) += hgfy[0];
   domain.fz(elemToNode[0]) += hgfz[0];

   //...

   domain.fx(elemToNode[7]) += hgfx[7];
   domain.fy(elemToNode[7]) += hgfy[7];
   domain.fz(elemToNode[7]) += hgfz[7];
}
```

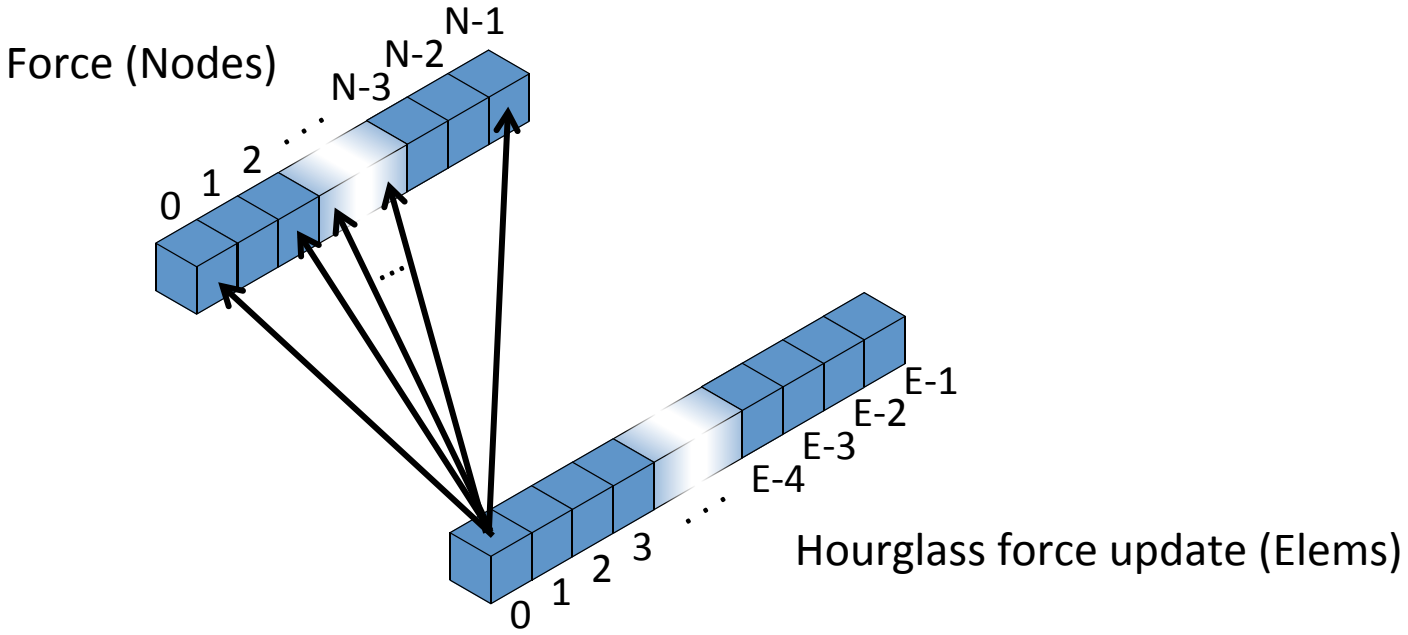
Element to node indirection

Affiliation Example

Update of force in LULESH 2.0 (scatter):

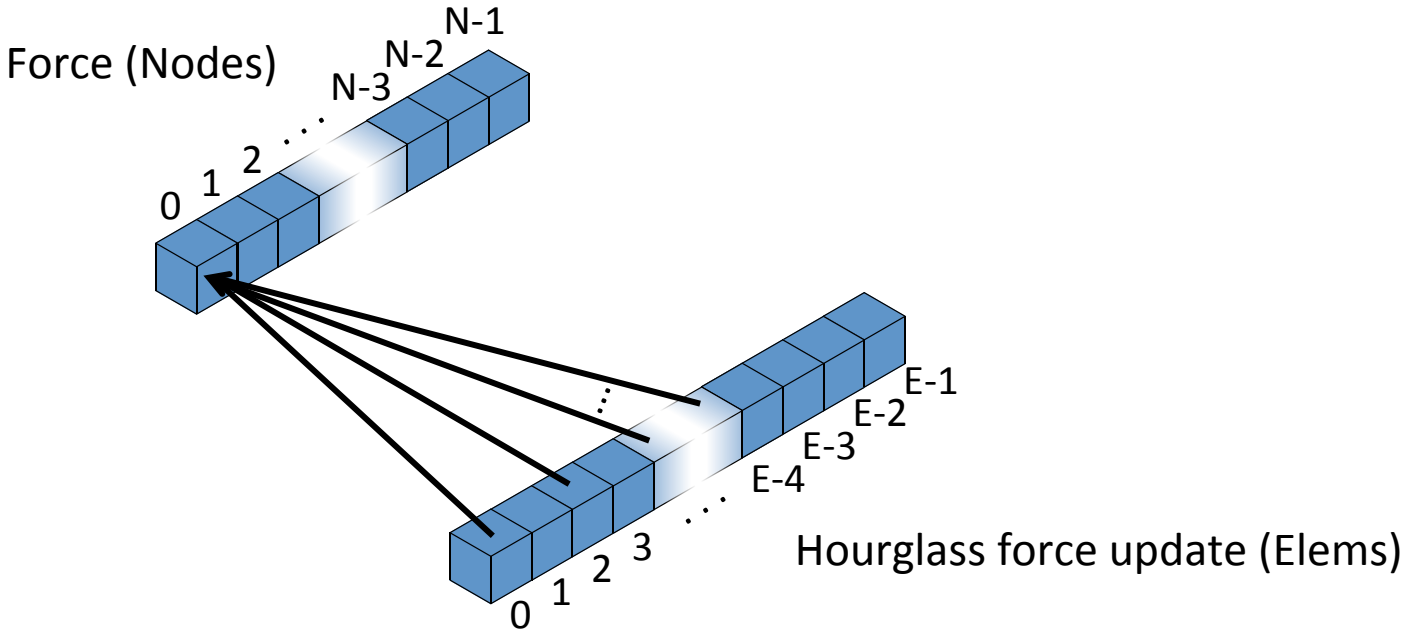


Affiliation Example



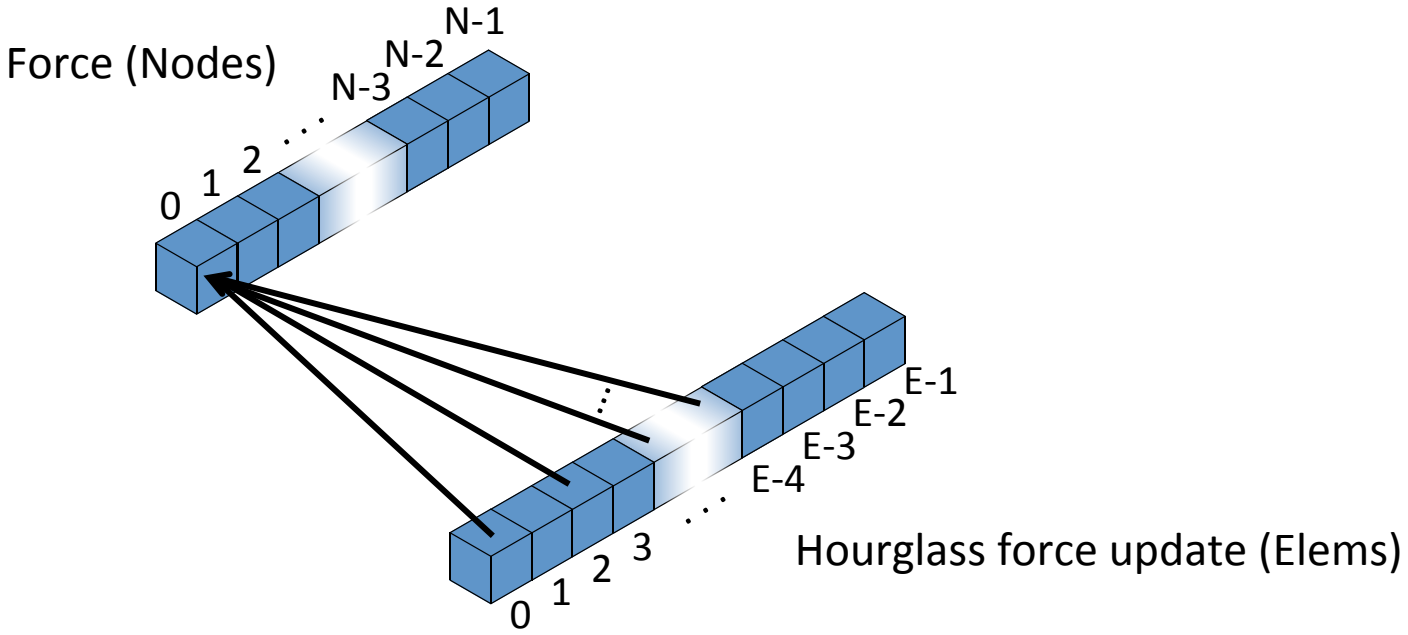
Scatter operation is efficient but not thread-safe.

Affiliation Example



Gather operation is thread-safe.

Affiliation Example



Gather operation is thread-safe.

Affiliation Example

Definition of corner aspect:

```
typedef PuffinAspect<4, 8, PuffinStyleFixedArray>
        CornerAspect;
```

Definition of array type for hgfd:

```
typedef PuffinArray<DimElemCornerAspects>
        DimElemCornerArray;
```

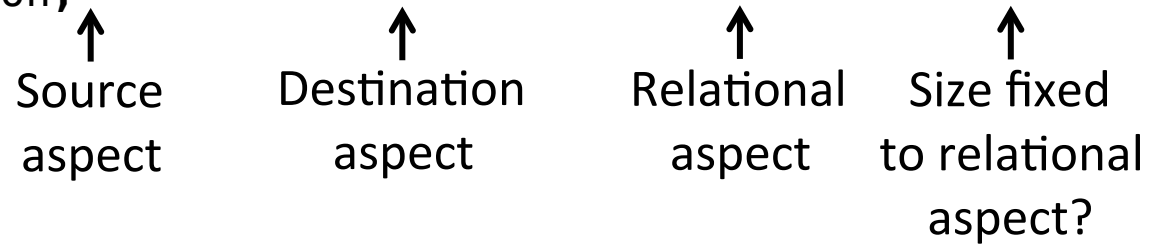
Instantiation of hgfd:

```
DimElemCornerArray hgfd_p(DimElemCorner);
```

Affiliation Example

Definition of affiliations between nodes and elements:

```
typedef PuffinAffiliation<ElemAspect, NodeAspect, CornerAspect, true>
        ElemToNodeAffiliation;
typedef PuffinAffiliation<NodeAspect, ElemAspect, CornerAspect, false>
        NodeToElemAffiliation;
```



An affiliation describes how one aspect relates to another.

Affiliation Example

Scatter version; ElemToNode affiliation used over a statement:

```
puffin_foreach(Elems)
  (ElemToNode(domain.fd()[Dim] |= domain.fd() + hgfd_p))
  .execute();
```

Gather version; NodeToElem affiliation summing over an expression:

```
puffin_foreach(Nodes)(Dim)
   (domain.fd() |= domain.fd() + NodeToElem.sum(hgfd_p))
   .execute();
```

Current syntax is not final, but is sufficient for now.

Puffin version of LULESH 2.0

Puffin supports all but about 10 loops of LULESH 2.0

Remaining loops require “whole-aspect” calculations

Puffin slows compile-time by 1.33x to 3x

Mostly depends upon compile-time options

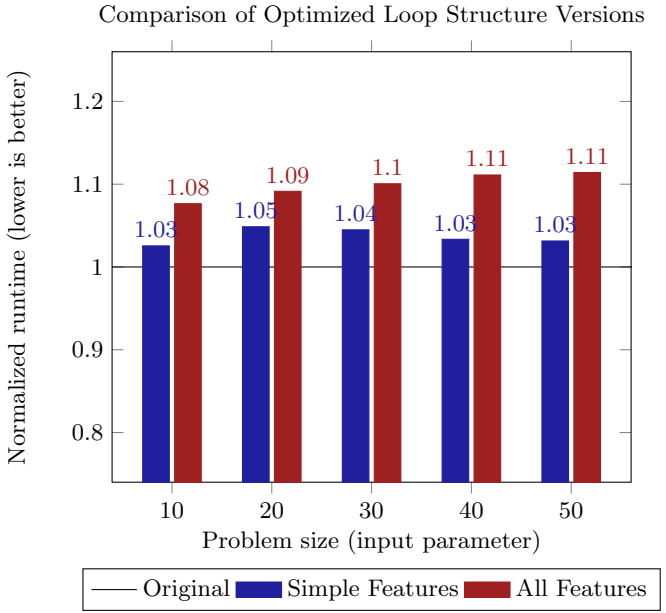
Puffin slows runtime performance by 5-11%

Depends upon problem size, used features, and internal Puffin optimizations

Results can be maintained (down to the bit)

Requires certain compiler flags (to maintain consistent 64-bit precision)

LULESH 2.0 Performance Results



Puffin has roughly 5-11% runtime overhead.

Summary

Puffin is currently a prototype

Puffin can be adopted incrementally

Useful for continuous use and maintaining results

Main benefit is low-overhead portable code with potential for performance

Plan to support single-thread CPU, multi-thread CPU, GPU, and Xeon Phi


Lawrence Livermore
National Laboratory