

Exploring Co-Design in Chapel Using LULESH

SIAM CSE13, MS79
February 26, 2013

Greg Titus
Principal Engineer
Chapel Team, Cray Inc.



Chapel

What is Chapel?

- **An emerging parallel programming language**
 - Design and development led by Cray Inc.
 - in collaboration with academia, labs, industry
 - Initiated under the DARPA HPCS program
- **Overall goal: Improve programmer productivity**
 - Improve the programmability of parallel computers
 - Match or beat the performance of current programming models
 - Support better portability than current programming models
 - Improve the robustness of parallel codes
- **A work-in-progress**
- **<http://chapel.cray.com/>**

Chapel's Implementation

- Being developed as open source at SourceForge
- Licensed as BSD software
- **Target Architectures:**
 - Cray architectures
 - multicore desktops and laptops
 - commodity clusters
 - systems from other vendors
 - *in-progress*: CPU+accelerator hybrids, manycore, ...

Motivating Chapel Themes

- 1) General Parallel Programming
- 2) Global-View Abstractions
- 3) Multiresolution Design
- 4) Control over Locality/Affinity
- 5) Reduce HPC \leftrightarrow Mainstream Language Gap

1) General Parallel Programming

With a unified set of concepts...

...express any parallelism desired in a user's program

- **Styles:** data-parallel, task-parallel, concurrency, nested, ...
- **Levels:** model, function, loop, statement, expression

...target all parallelism available in the hardware

- **Types:** machines, nodes, cores, instructions



1) General Parallel Programming

With a unified set of concepts...

...express any parallelism desired in a user's program

- **Styles:** data-parallel, task-parallel, concurrency, nested, ...
- **Levels:** model, function, loop, statement, expression

...target all parallelism available in the hardware

- **Types:** machines, nodes, cores, instructions

| Style of HW Parallelism | Programming Model | Unit of Parallelism |
|-------------------------|-------------------|---------------------|
| Inter-node | MPI | executable/process |
| Intra-node/multicore | OpenMP | iteration/task |
| GPU/accelerator | CUDA | SIMD function/task |



1) General Parallel Programming

With a unified set of concepts...

...express any parallelism desired in a user's program

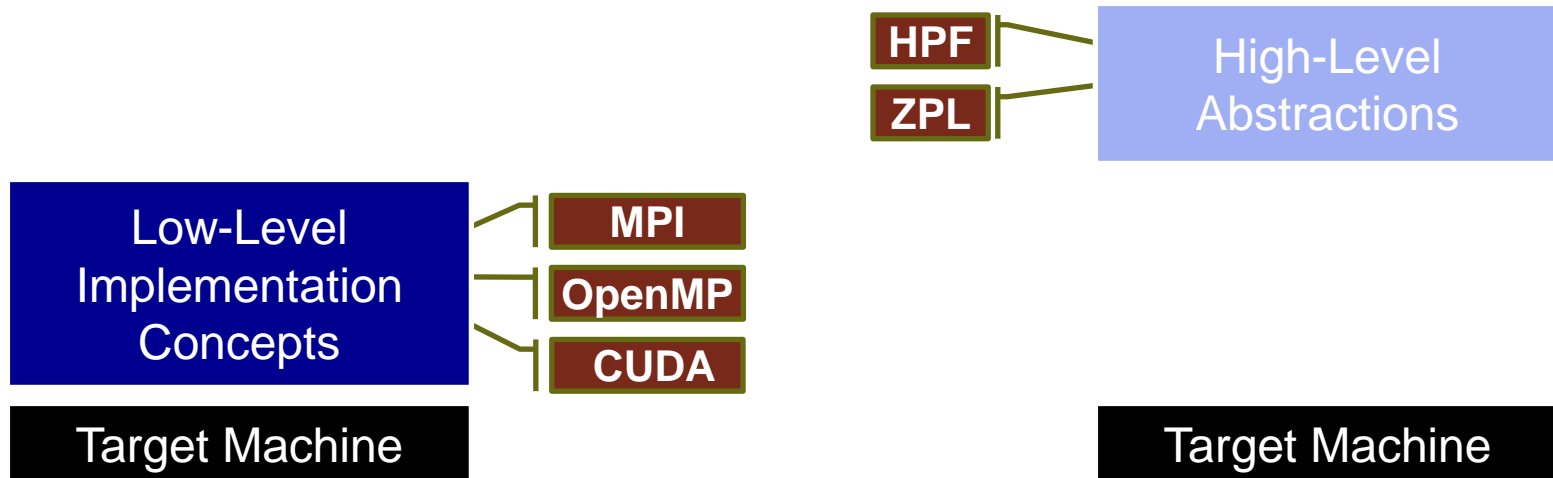
- **Styles:** data-parallel, task-parallel, concurrency, nested, ...
- **Levels:** model, function, loop, statement, expression

...target all parallelism available in the hardware

- **Types:** machines, nodes, cores, instructions

| Style of HW Parallelism | Programming Model | Unit of Parallelism |
|-------------------------|-------------------|---------------------|
| Inter-node | Chapel | executable/task |
| Intra-node/multicore | Chapel | iteration/task |
| GPU/accelerator | Chapel | SIMD function/task |

3) Multiresolution Design: Motivation



“Why is everything so tedious/difficult?”
“Why don’t my programs port trivially?”

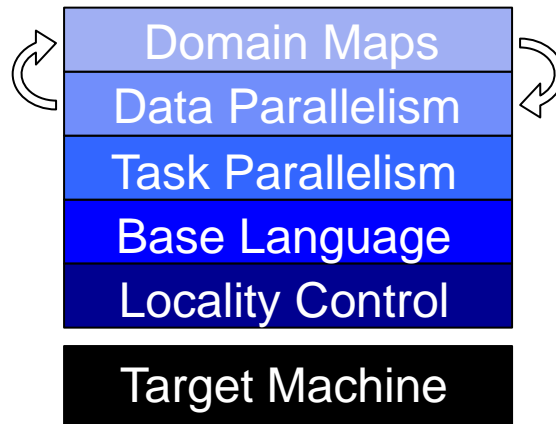
“Why don’t I have more control?”

3) Multiresolution Design

Multiresolution Design: Support multiple tiers of features

- higher levels for programmability, productivity
- lower levels for greater degrees of control

Chapel language concepts



- build the higher-level concepts in terms of the lower
- permit the user to intermix layers arbitrarily

LULESH (in Chapel)

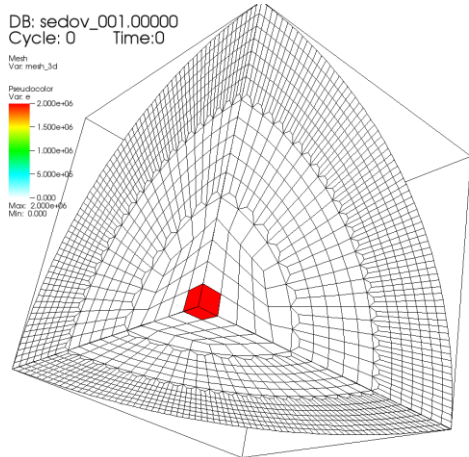


What is LULESH?

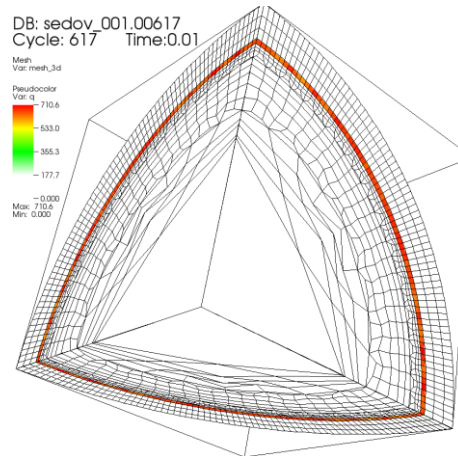
- ***Livermore Unstructured Lagrange Explicit Shock Hydrodynamics* challenge problem**
 - Developed as a *proxy application* at LLNL under DARPA UHPC
 - Includes computations and algorithms used in production codes
- **<https://computation.llnl.gov/casc/ShockHydro/>**
- **There are reference implementations in many languages**
 - Serial C
 - C + OMP
 - C + OMP + MPI (not publically available yet)
 - CUDA
 - Loci (logic programming)
 - A++ (C++ class library)
 - Chapel

What Does LULESH Do?

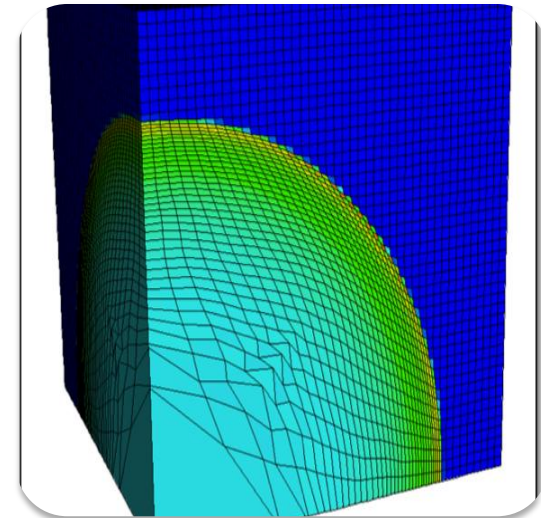
- Solve one octant of the spherical Sedov problem (blast wave) using Lagrangian hydrodynamics for a single material



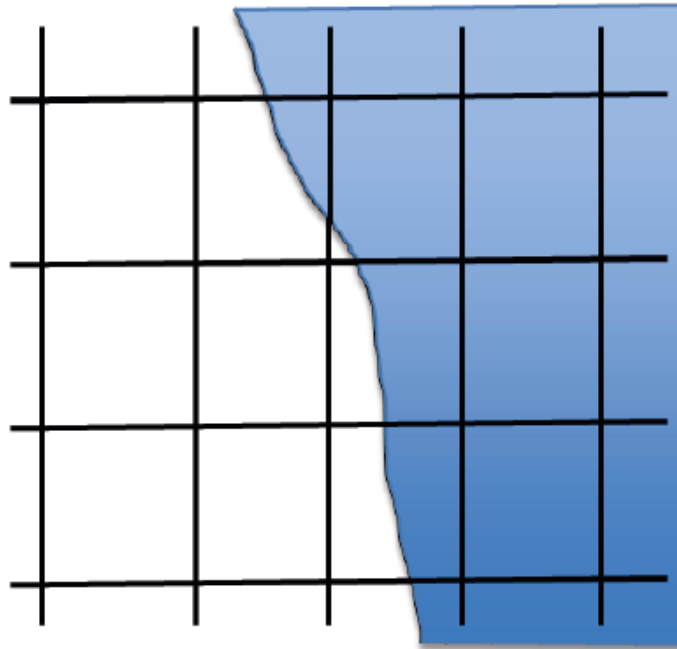
user: keasler
Thu Apr 12 11:56:04 2012



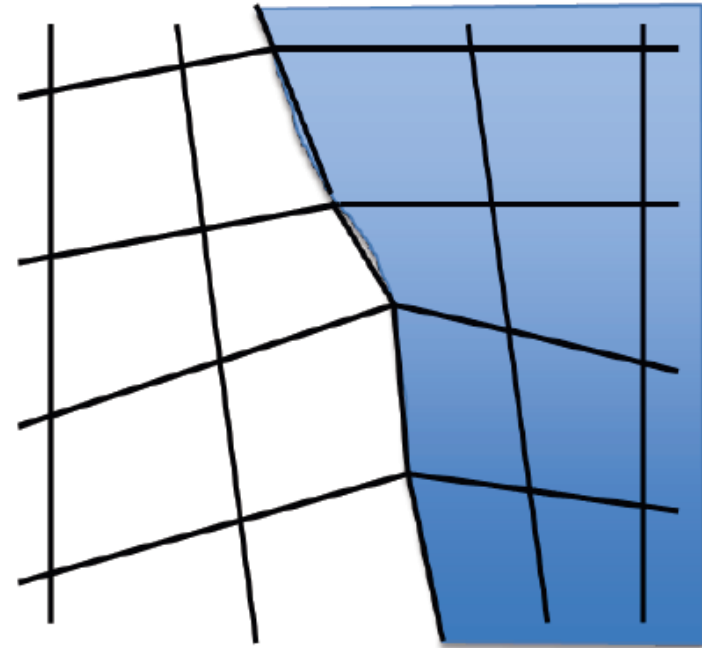
user: keasler
Thu Apr 12 11:57:44 2012



Eulerian vs. Lagrangian Meshes



Eulerian mesh
(grid stays fixed)



Lagrangian mesh
(grid adapts to materials)

Image Source: LULESH specification, LLNL-TR-490254

<https://computation.llnl.gov/casc/ShockHydro/>

LULESH Compared to a Real Hydrocode

- **LULESH**

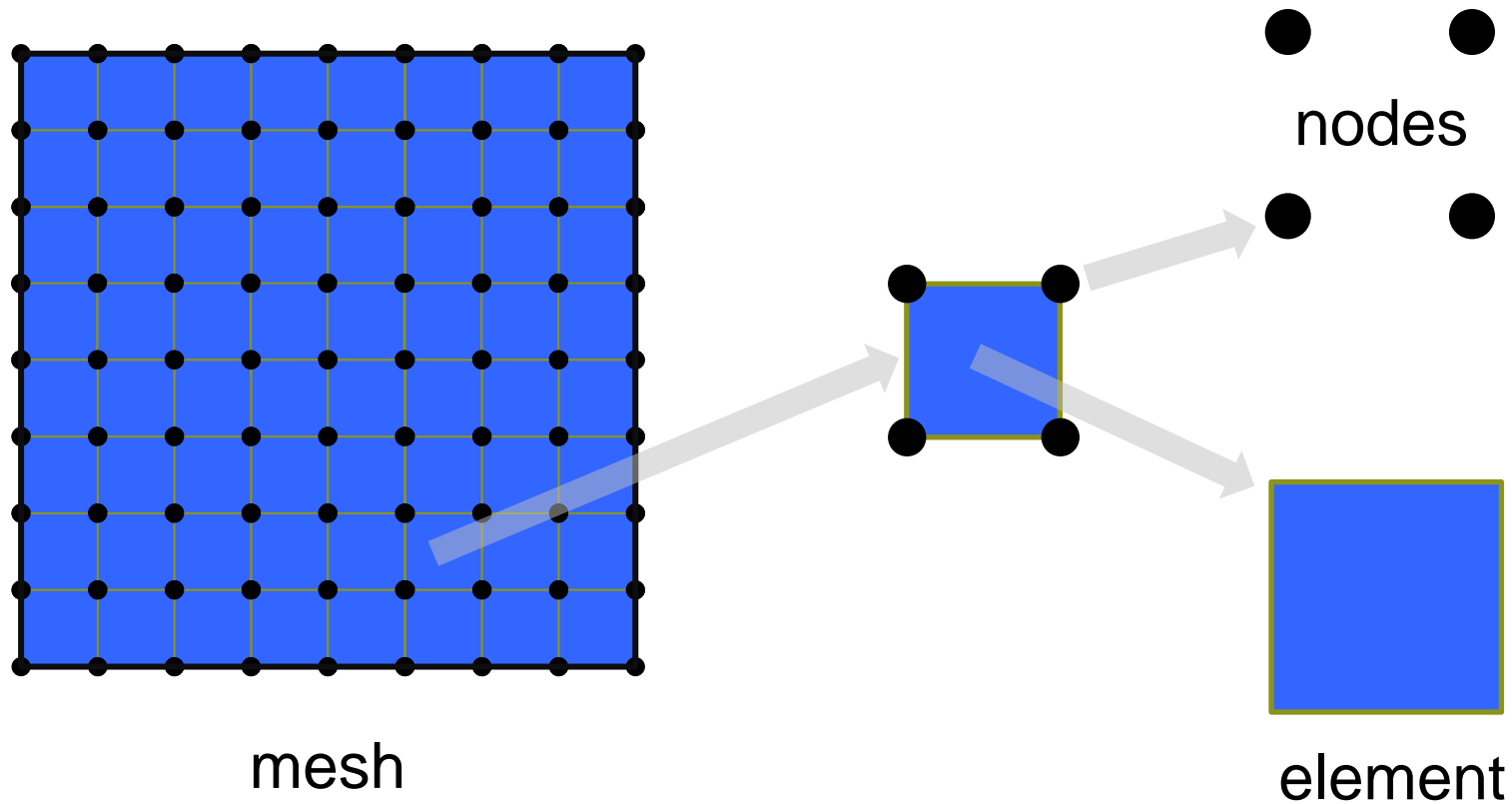
- Structured input provided (3D regular)
- Single material per cell

- **Real Hydrocodes**

- Unstructured input (compact, irregular)
- Could have multiple materials in a cell

- **But: LULESH uses code idioms similar to those in a real code, so as to stress compilation and execution similarly**

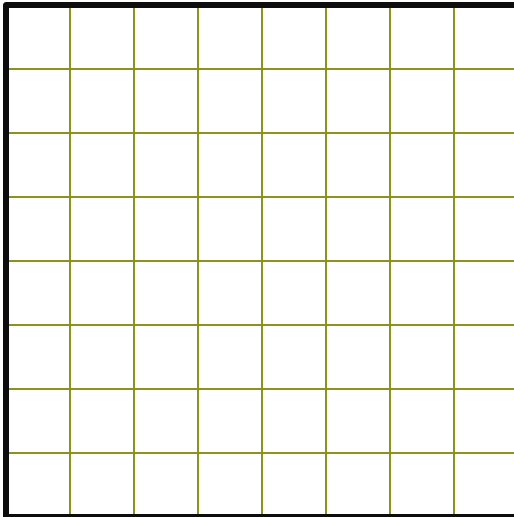
Fundamental LULESH Concepts/Terminology



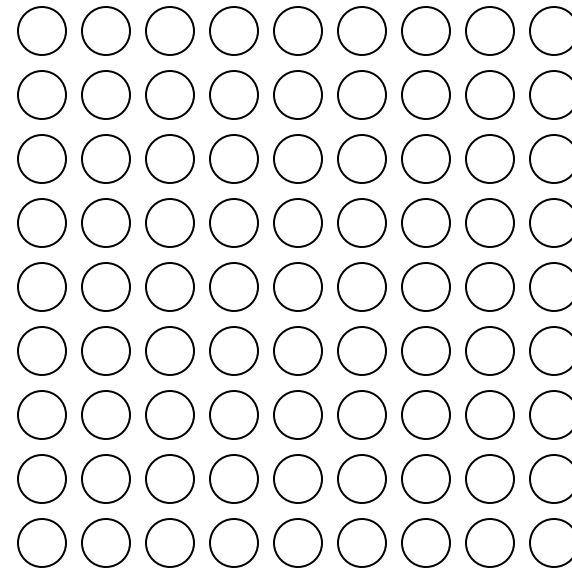
Chapel Representation (Structured)

- **Abstract Element and Node *Domains*:**

```
const nodesPerEdge = elemsPerEdge+1;
const ElemSpace = {0..#elemsPerEdge, 0..#elemsPerEdge},
      NodeSpace = {0..#nodesPerEdge, 0..#nodesPerEdge};
```



ElemSpace



NodeSpace

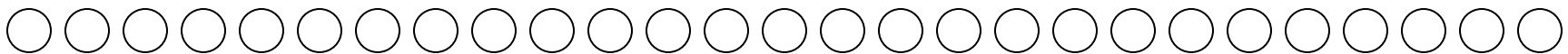
Chapel Representation (Unstructured)

- **Abstract Element and Node *Domains*:**

```
const ElemSpace = {0..#numElems},  
      NodeSpace  = {0..#numNodes};
```



ElemSpace

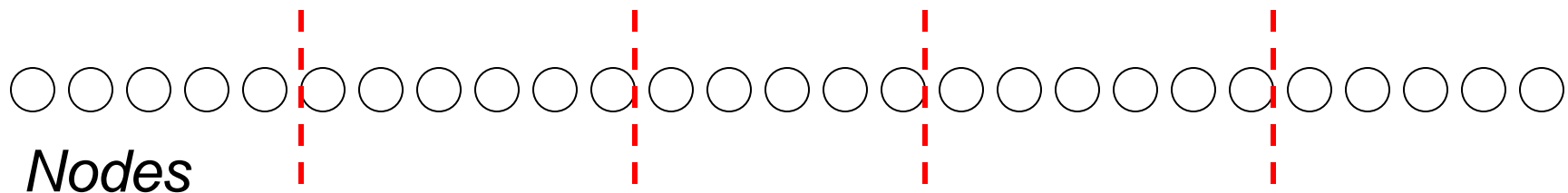
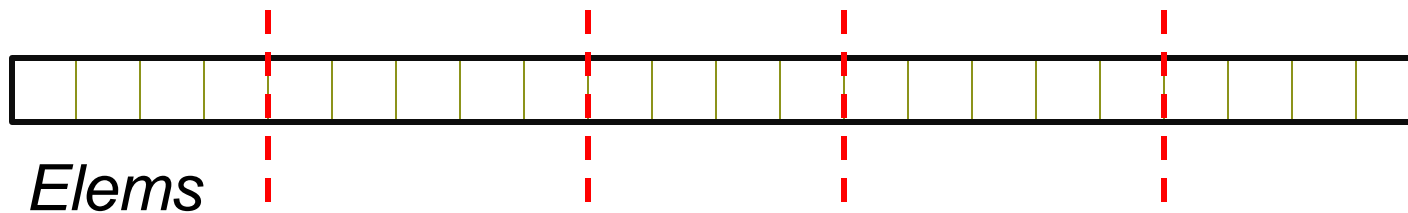


NodeSpace

Chapel Representation (Multi-locale)

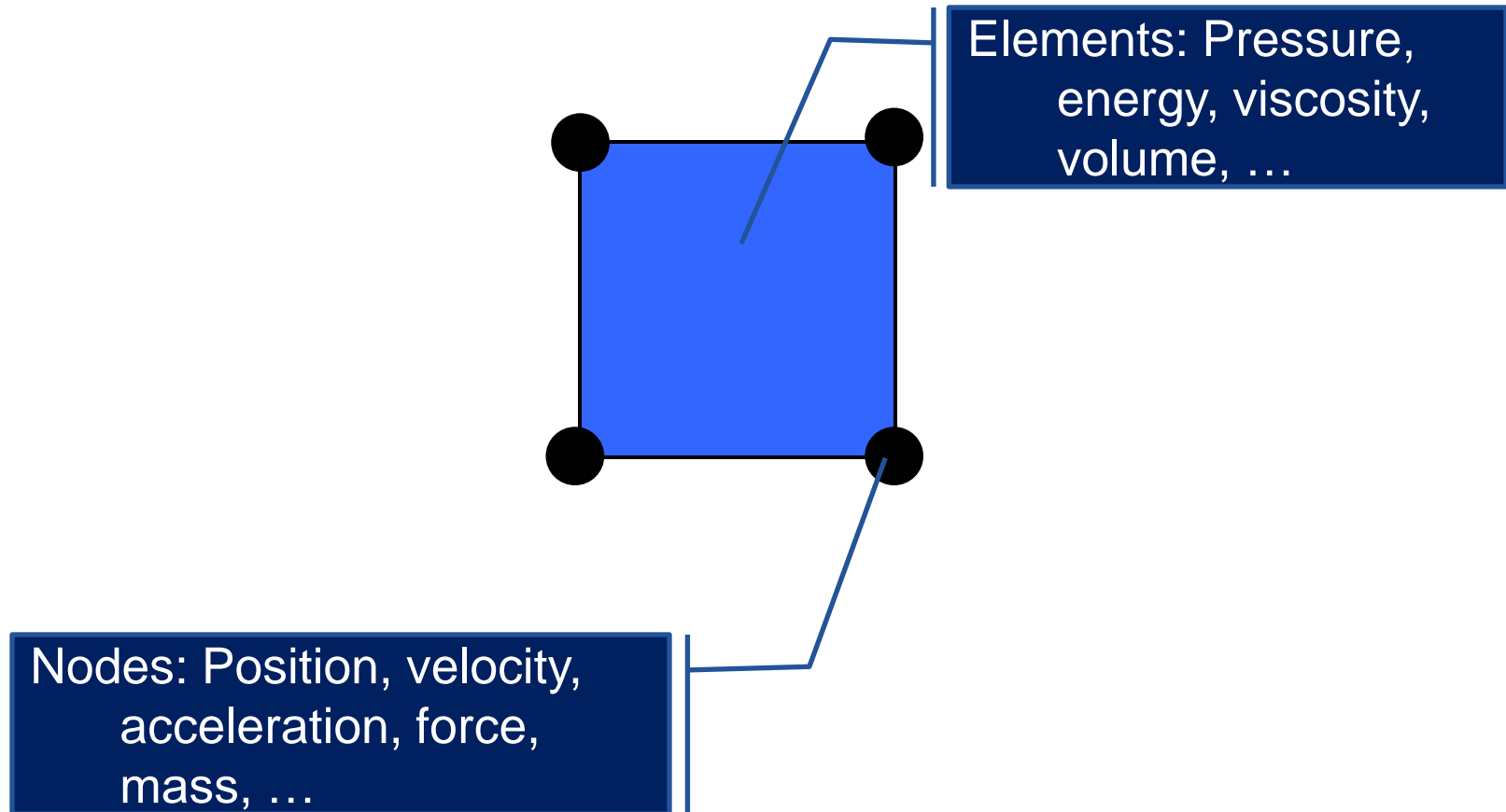
- **Distributed Element/Node Domains:**

```
const Elms = ElemSpace dmapped Block(ElemSpace),
      Nodes = NodeSpace dmapped Block(NodeSpace);
```



Element and Node Fields

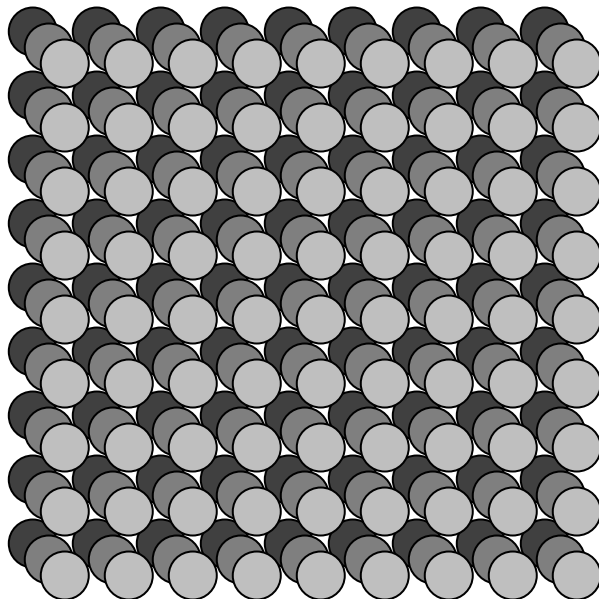
- Some variables (*fields*) are associated with elements, others with nodes.



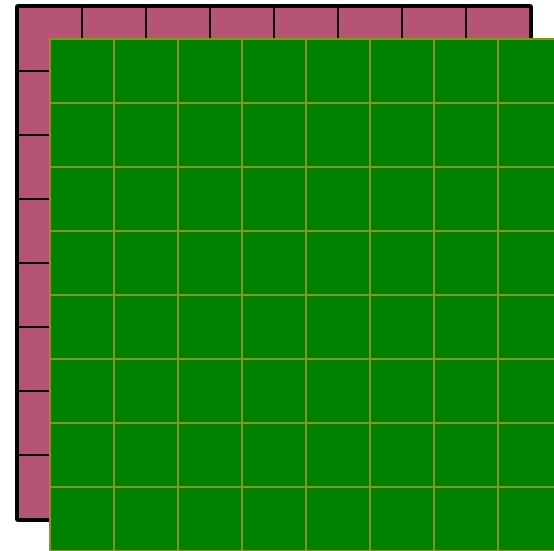
Representation of Fields in Chapel

- Sample field declarations:

```
var x, y, z: [Nodes] real;
var e, p: [Elms] real;
```



x, y, z

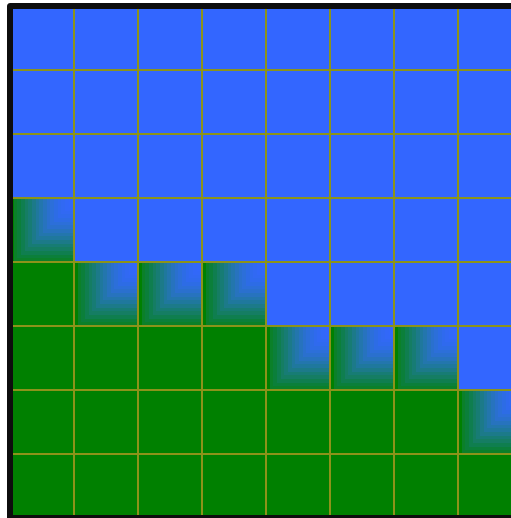


e, p

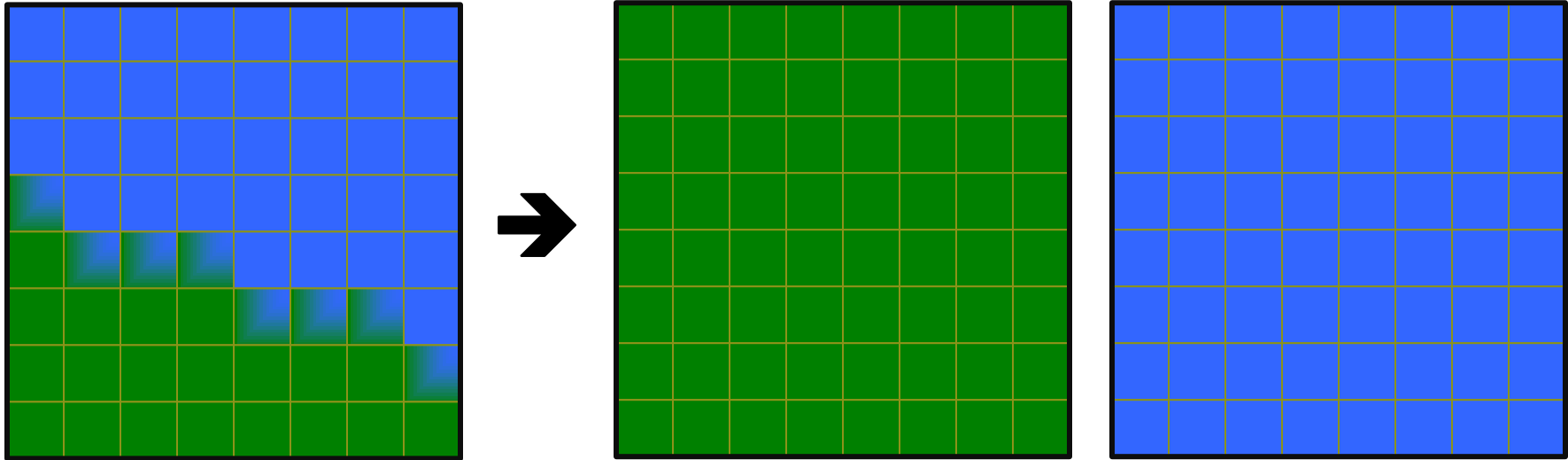
(Conceptual representation)

Materials Representation

- Not all elements will contain all materials, and some will contain combinations



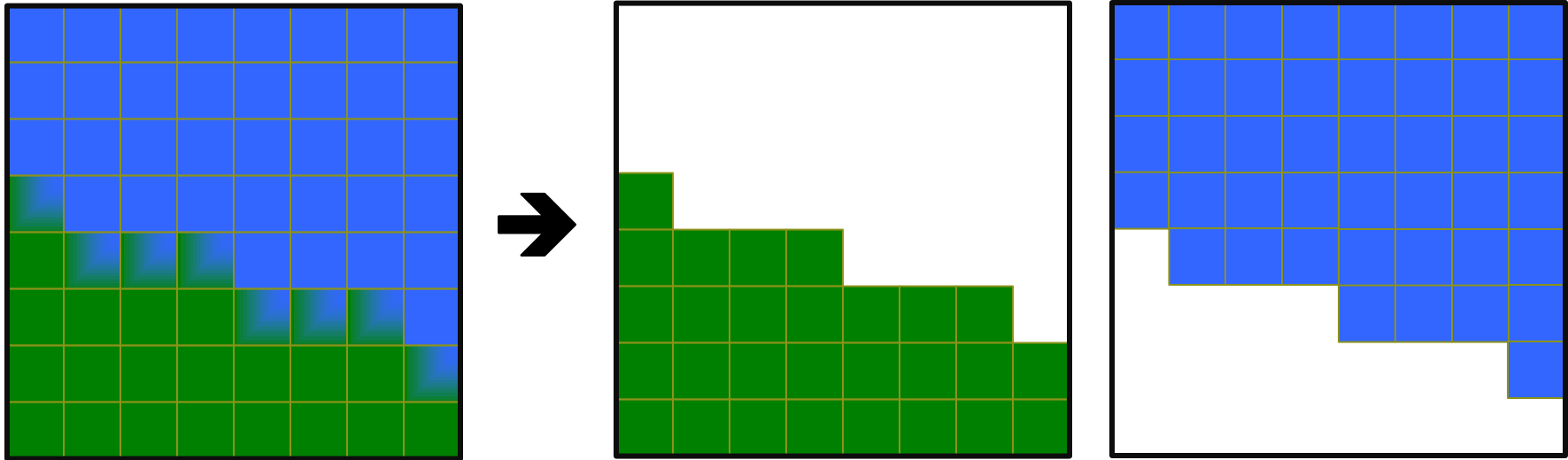
Materials Representation (Dense)



naïve approach: store all materials everywhere
(reasonable for LULESH, but not in practice)

```
const Mat1Elems = Elems,  
      Mat2Elems = Elems;
```

Materials Representation (Sparse)



improved approach: use sparse subdomains to
only store materials where necessary

```
var Mat1Elems: sparse subdomain(Elems) = enumerateMat1Locs(),  
    Mat2Elems: sparse subdomain(Elems) = enumerateMat2Locs();
```


1. **Introduction**
 2. **Background**
 3. **Methodology**
 4. **Results**
 5. **Discussion**
 6. **Conclusion**
 7. **References**
 8. **Appendix**
 9. **Figure 1**
 10. **Figure 2**
 11. **Figure 3**
 12. **Figure 4**
 13. **Figure 5**
 14. **Figure 6**
 15. **Figure 7**
 16. **Figure 8**
 17. **Figure 9**
 18. **Figure 10**
 19. **Figure 11**
 20. **Figure 12**
 21. **Figure 13**
 22. **Figure 14**
 23. **Figure 15**
 24. **Figure 16**
 25. **Figure 17**
 26. **Figure 18**
 27. **Figure 19**
 28. **Figure 20**
 29. **Figure 21**
 30. **Figure 22**
 31. **Figure 23**
 32. **Figure 24**
 33. **Figure 25**
 34. **Figure 26**
 35. **Figure 27**
 36. **Figure 28**
 37. **Figure 29**
 38. **Figure 30**
 39. **Figure 31**
 40. **Figure 32**
 41. **Figure 33**
 42. **Figure 34**
 43. **Figure 35**
 44. **Figure 36**
 45. **Figure 37**
 46. **Figure 38**
 47. **Figure 39**
 48. **Figure 40**
 49. **Figure 41**
 50. **Figure 42**
 51. **Figure 43**
 52. **Figure 44**
 53. **Figure 45**
 54. **Figure 46**
 55. **Figure 47**
 56. **Figure 48**
 57. **Figure 49**
 58. **Figure 50**
 59. **Figure 51**
 60. **Figure 52**
 61. **Figure 53**
 62. **Figure 54**
 63. **Figure 55**
 64. **Figure 56**
 65. **Figure 57**
 66. **Figure 58**
 67. **Figure 59**
 68. **Figure 60**
 69. **Figure 61**
 70. **Figure 62**
 71. **Figure 63**
 72. **Figure 64**
 73. **Figure 65**
 74. **Figure 66**
 75. **Figure 67**
 76. **Figure 68**
 77. **Figure 69**
 78. **Figure 70**
 79. **Figure 71**
 80. **Figure 72**
 81. **Figure 73**
 82. **Figure 74**
 83. **Figure 75**
 84. **Figure 76**
 85. **Figure 77**
 86. **Figure 78**
 87. **Figure 79**
 88. **Figure 80**
 89. **Figure 81**
 90. **Figure 82**
 91. **Figure 83**
 92. **Figure 84**
 93. **Figure 85**
 94. **Figure 86**
 95. **Figure 87**
 96. **Figure 88**
 97. **Figure 89**
 98. **Figure 90**
 99. **Figure 91**
 100. **Figure 92**
 101. **Figure 93**
 102. **Figure 94**
 103. **Figure 95**
 104. **Figure 96**
 105. **Figure 97**
 106. **Figure 98**
 107. **Figure 99**
 108. **Figure 100**
 109. **Figure 101**
 110. **Figure 102**
 111. **Figure 103**
 112. **Figure 104**
 113. **Figure 105**
 114. **Figure 106**
 115. **Figure 107**
 116. **Figure 108**
 117. **Figure 109**
 118. **Figure 110**
 119. **Figure 111**
 120. **Figure 112**
 121. **Figure 113**
 122. **Figure 114**
 123. **Figure 115**
 124. **Figure 116**
 125. **Figure 117**
 126. **Figure 118**
 127. **Figure 119**
 128. **Figure 120**
 129. **Figure 121**
 130. **Figure 122**
 131. **Figure 123**
 132. **Figure 124**
 133. **Figure 125**
 134. **Figure 126**
 135. **Figure 127**
 136. **Figure 128**
 137. **Figure 129**
 138. **Figure 130**
 139. **Figure 131**
 140. **Figure 132**
 141. **Figure 133**
 142. **Figure 134**
 143. **Figure 135**
 144. **Figure 136**
 145. **Figure 137**
 146. **Figure 138**
 147. **Figure 139**
 148. **Figure 140**
 149. **Figure 141**
 150. **Figure 142**
 151. **Figure 143**
 152. **Figure 144**
 153. **Figure 145**
 154. **Figure 146**
 155. **Figure 147**
 156. **Figure 148**
 157. **Figure 149**
 158. **Figure 150**
 159. **Figure 151**
 160. **Figure 152**
 161. **Figure 153**
 162. **Figure 154**
 163. **Figure 155**
 164. **Figure 156**
 165. **Figure 157**
 166. **Figure 158**
 167. **Figure 159**
 168. **Figure 160**
 169. **Figure 161**
 170. **Figure 162**
 171. **Figure 163**
 172. **Figure 164**
 173. **Figure 165**
 174. **Figure 166**
 175. **Figure 167**
 176. **Figure 168**
 177. **Figure 169**
 178. **Figure 170**
 179. **Figure 171**
 180. **Figure 172**
 181. **Figure 173**
 182. **Figure 174**
 183. **Figure 175**
 184. **Figure 176**
 185. **Figure 177**
 186. **Figure 178**
 187. **Figure 179**
 188. **Figure 180**
 189. **Figure 181**
 190. **Figure 182**
 191. **Figure 183**
 192. **Figure 184**
 193. **Figure 185**
 194. **Figure 186**
 195. **Figure 187**
 196. **Figure 188**
 197. **Figure 189**
 198. **Figure 190**
 199. **Figure 191**
 200. **Figure 192**
 201. **Figure 193**
 202. **Figure 194**
 203. **Figure 195**
 204. **Figure 196**
 205. **Figure 197**
 206. **Figure 198**
 207. **Figure 199**
 208. **Figure 200**
 209. **Figure 201**
 210. **Figure 202**
 211. **Figure 203**
 212. **Figure 204**
 213. **Figure 205**
 214. **Figure 206**
 215. **Figure 207**
 216. **Figure 208**
 217. **Figure 209**

LULESH in Chapel



trunk/test/release/examples/benchmarks/lulesh/*.chpl
in the **SourceForge** repository, as of r21020 (2/14/13)

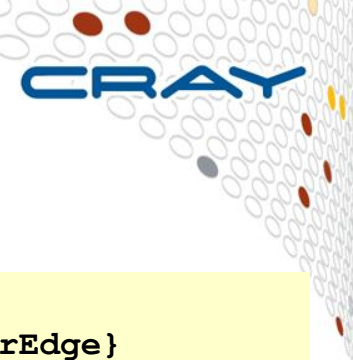
There are: 1288 lines of code
266 lines of comments
487 blank lines

(the C+MPI+OpenMP version is nearly 4x bigger)

LULESH in Chapel



← this is all of the representation dependent code →



The Representation Dependent Code

```
const ElemSpace = if use3DRepresentation
    then {0..#elemsPerEdge, 0..#elemsPerEdge, 0..#elemsPerEdge}
    else {0..#numElems},
NodeSpace = if use3DRepresentation
    then {0..#nodesPerEdge, 0..#nodesPerEdge, 0..#nodesPerEdge}
    else {0..#numNodes};
```

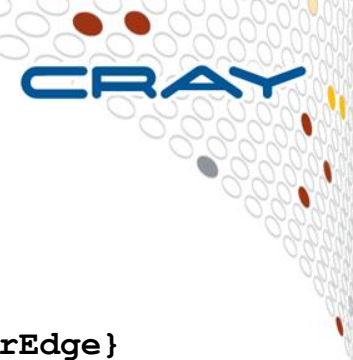
```
const Elems = if useBlockDist then ElemSpace dmapped Block(ElemSpace)
    else ElemSpace,
Nodes = if useBlockDist then NodeSpace dmapped Block(NodeSpace)
    else NodeSpace;
```

```
var elemToNode: [Elems] nodesPerElem*index(Nodes);
```

```
var XSym, YSym, ZSym: sparse subdomain(Nodes);
```

```
const MatElems: MatElemsType = if sparseMaterials then enumerateMatElems()
    else Elems;
```

domains for elements and nodes



The Representation Dependent Code

```
const ElemSpace = if use3DRepresentation
    then {0..#elemsPerEdge, 0..#elemsPerEdge, 0..#elemsPerEdge}
    else {0..#numElems},
NodeSpace = if use3DRepresentation
    then {0..#nodesPerEdge, 0..#nodesPerEdge, 0..#nodesPerEdge}
    else {0..#numNodes};
```

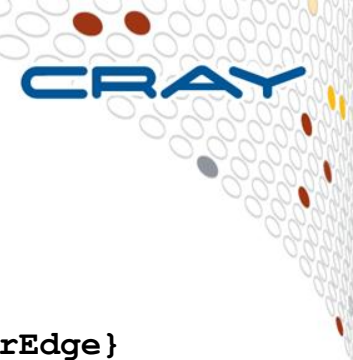
```
const Elems = if useBlockDist then ElemSpace dmapped Block(ElemSpace)
    else ElemSpace,
Nodes = if useBlockDist then NodeSpace dmapped Block(NodeSpace)
    else NodeSpace;
```

```
var elemToNode: [Elems] nodesPerElem*index(Nodes);
```

```
var XSym, YSym, ZSym: sparse subdomain(Nodes);
```

```
const MatElems: MatElemsType = if sparseMaterials then enumerateMatElems()
    else Elems;
```

potentially distributed domains for
elements and nodes



The Representation Dependent Code

```
const ElemSpace = if use3DRepresentation
    then {0..#elemsPerEdge, 0..#elemsPerEdge, 0..#elemsPerEdge}
    else {0..#numElems},
NodeSpace = if use3DRepresentation
    then {0..#nodesPerEdge, 0..#nodesPerEdge, 0..#nodesPerEdge}
    else {0..#numNodes};

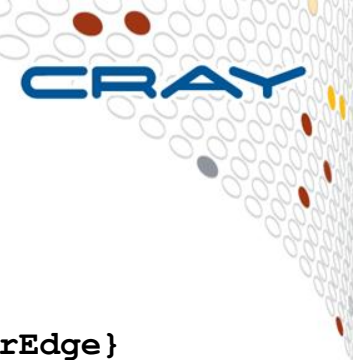
const Elems = if useBlockDist then ElemSpace dmapped Block(ElemSpace)
    else ElemSpace,
Nodes = if useBlockDist then NodeSpace dmapped Block(NodeSpace)
    else NodeSpace;

var elemToNode: [Elems] nodesPerElem*index(Nodes);

var XSym, YSym, ZSym: sparse subdomain(Nodes);

const MatElems: MatElemsType = if sparseMaterials then enumerateMatElems()
    else Elems;
```

nodes adjacent to each element



The Representation Dependent Code

```
const ElemSpace = if use3DRepresentation
    then {0..#elemsPerEdge, 0..#elemsPerEdge, 0..#elemsPerEdge}
    else {0..#numElems},
NodeSpace = if use3DRepresentation
    then {0..#nodesPerEdge, 0..#nodesPerEdge, 0..#nodesPerEdge}
    else {0..#numNodes};

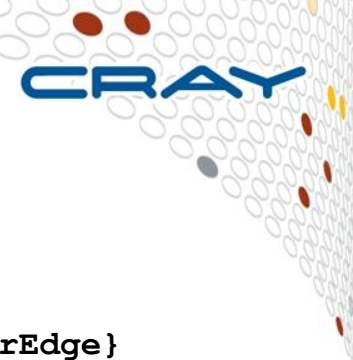
const Elems = if useBlockDist then ElemSpace dmapped Block(ElemSpace)
    else ElemSpace,
Nodes = if useBlockDist then NodeSpace dmapped Block(NodeSpace)
    else NodeSpace;

var elemToNode: [Elems] nodesPerElem*index(Nodes);

var XSym, YSym, ZSym: sparse subdomain(Nodes);

const MatElems: MatElemsType = if sparseMaterials then enumerateMatElems()
    else Elems;
```

symmetry planes



The Representation Dependent Code

```
const ElemSpace = if use3DRepresentation
    then {0..#elemsPerEdge, 0..#elemsPerEdge, 0..#elemsPerEdge}
    else {0..#numElems},
NodeSpace = if use3DRepresentation
    then {0..#nodesPerEdge, 0..#nodesPerEdge, 0..#nodesPerEdge}
    else {0..#numNodes};

const Elems = if useBlockDist then ElemSpace dmapped Block(ElemSpace)
    else ElemSpace,
Nodes = if useBlockDist then NodeSpace dmapped Block(NodeSpace)
    else NodeSpace;

var elemToNode: [Elems] nodesPerElem*index(Nodes);

var XSym, YSym, ZSym: sparse subdomain(Nodes);

const MatElems: MatElemsType = if sparseMaterials then enumerateMatElems()
    else Elems;
```

domain describing elements that
contain the material

The Representation Dependent Code

```
proc MatElemsType type {
  if sparseMaterials {
    if (printWarnings && useBlockDist && numLocales > 1) then
      writeln("WARNING: The LULESH Material Elements (MatElems) are not yet\n",
        "          distributed, so result in excessive memory use on,\n",
        "          and communication with, locale 0\n");
    return sparse subdomain(Elems);
  } else
    return Elems.type;
}
```

```
iter elemToNodes(elem) {
  for param i in 1..nodesPerElem do
    yield elemToNode[elem][i];
}
```

```
iter elemToNodesTuple(e) {{
  for i in 1..nodesPerElem do
    yield (elemToNode[e][i], i);
}}
```

the type of the domain describing
elements that contain the material



The Representation Dependent Code

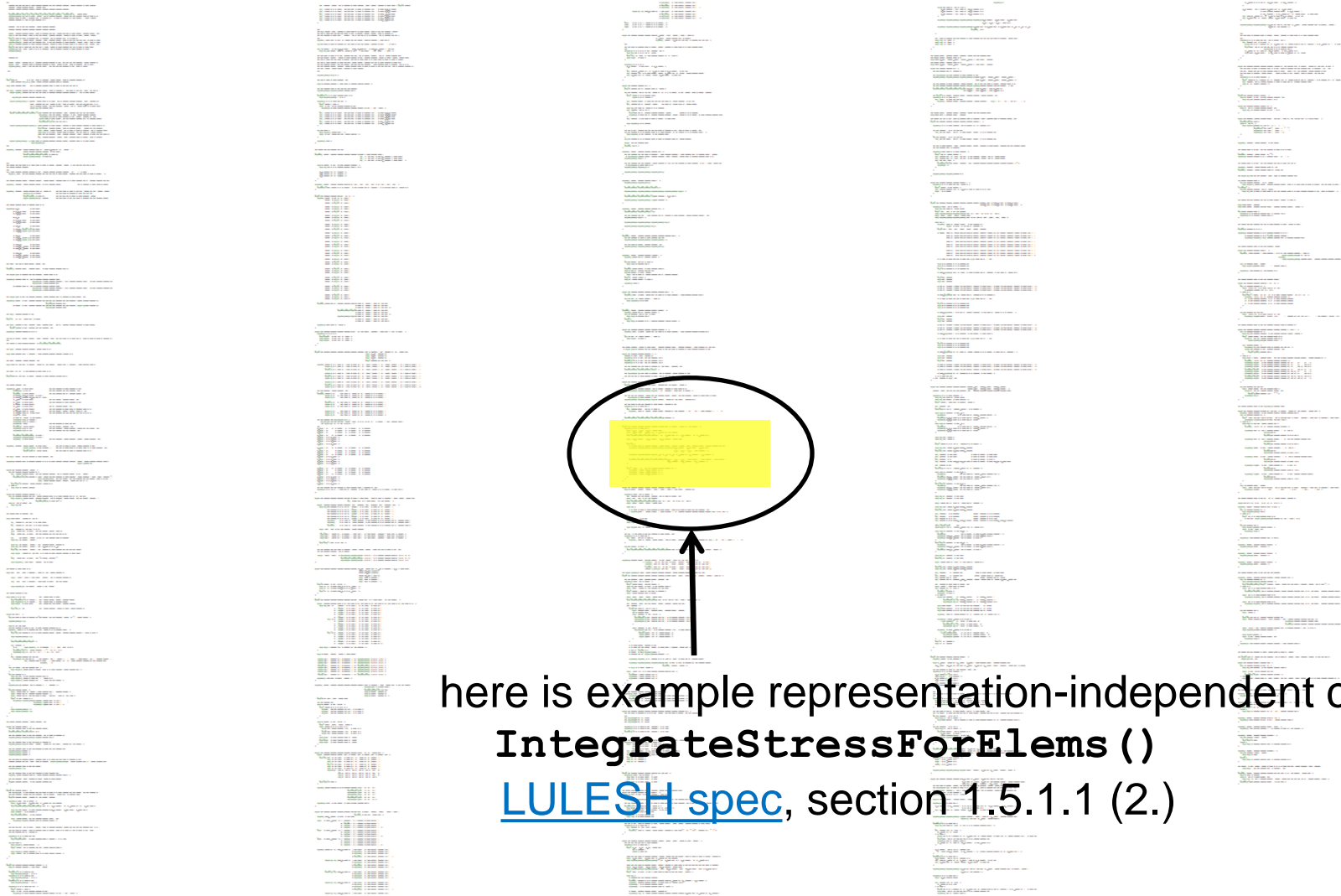
```
proc MatElemsType type {  
  if sparseMaterials {  
    if (printWarnings && useBlockDist && numLocales > 1) then  
      writeln("WARNING: The LULESH Material Elements (MatElems) are not yet\n",  
              "          distributed, so result in excessive memory use on,\n",  
              "          and communication with, locale 0\n");  
    return sparse subdomain(Elems);  
  } else  
    return Elems.type;  
}
```

```
iter elemToNodes(elem) {  
  for param i in 1..nodesPerElem do  
    yield elemToNode[elem][i];  
}
```

```
iter elemToNodesTuple(e) {{  
  for i in 1..nodesPerElem do  
    yield (elemToNode[e][i], i);  
}}
```

iterators mapping elements to their adjacent nodes

LULESH in Chapel



here is example representation-independent code:
IntegrateStressForElems()
[LULESH spec](#), section 1.5.1.1 (2.)

Representation Independent Physics!

```

proc IntegrateStressForElems(sigxx, sigyy, sigzz, determ) {
  forall k in Elems {
    var b_x, b_y, b_z: 8*real;
    var x_local, y_local, z_local: 8*real;
    localizeNeighborNodes(k, x, x_local, y, y_local, z, z_local);

    var fx_local, fy_local, fz_local: 8*real;

    local {
      /* Volume calculation involves extra work for numerical consistency. */
      CalcElemShapeFunctionDerivatives(x_local, y_local, z_local,
                                       b_x, b_y, b_z, determ[k]);

      CalcElemNodeNormals(b_x, b_y, b_z, x_local, y_local, z_local);

      SumElemStressesToNodeForces(b_x, b_y, b_z, sigxx[k], sigyy[k], sigzz[k],
                                  fx_local, fy_local, fz_local);
    }

    for (noi, t) in elemToNodesTuple(k) {
      fx[noi].add(fx_local[t]);
      fy[noi].add(fy_local[t]);
      fz[noi].add(fz_local[t]);
    }
  }
}

```

loop over all elements

collect nodes neighboring this element; localize node fields

update node forces from element stresses

All of this is independent of:

- structured vs. unstructured mesh
- shared vs. distributed data
- sparse vs. dense representation

Codesign

LULESH in Chapel, Codesign Timeline

Apr 2011: LLNL expresses interest in Chapel at Salishan Conference

- Introduced us to the LULESH benchmark

LULESH in Chapel, Codesign Timeline

Apr 2011: LLNL expresses interest in Chapel at Salishan Conference

- Introduced us to the LULESH benchmark

Summer 2011: Cray intern ports LULESH to Chapel

- *caveat*: used structured mesh to represent data arrays



LULESH in Chapel, Codesign Timeline

Apr 2011: LLNL expresses interest in Chapel at Salishan Conference

- Introduced us to the LULESH benchmark

Summer 2011: Cray intern ports LULESH to Chapel

- *caveat*: used structured mesh to represent data arrays

Nov 2011: Chapel team tunes LULESH for single-node performance

Dec 2011: Chapel team visits LLNL (talk, tutorial, 1-on-1 sessions)



LULESH in Chapel, Codesign Timeline

Apr 2011: LLNL expresses interest in Chapel at Salishan Conference

- Introduced us to the LULESH benchmark

Summer 2011: Cray intern ports LULESH to Chapel

- *caveat*: used structured mesh to represent data arrays

Nov 2011: Chapel team tunes LULESH for single-node performance

Dec 2011: Chapel team visits LLNL (talk, tutorial, 1-on-1 sessions)

Mar 2012: Jeff Keasler (LLNL) visits Cray to pair-program

- in one afternoon, converted from structured to unstructured mesh
- impact on code minimal (mostly in declarations)



LULESH in Chapel, Codesign Timeline

Apr 2011: LLNL expresses interest in Chapel at Salishan Conference

- Introduced us to the LULESH benchmark

Summer 2011: Cray intern ports LULESH to Chapel

- *caveat*: used structured mesh to represent data arrays

Nov 2011: Chapel team tunes LULESH for single-node performance

Dec 2011: Chapel team visits LLNL (talk, tutorial, 1-on-1 sessions)

Mar 2012: Jeff Keasler (LLNL) visits Cray to pair-program

- in one afternoon, converted from structured to unstructured mesh
- impact on code minimal (mostly in declarations)

Apr 2012: LLNL reports on collaboration at Salishan

Apr 2012: Chapel 1.5.0 release includes LULESH as an example code



LULESH in Chapel, Codesign Timeline

Apr 2011: LLNL expresses interest in Chapel at Salishan Conference

- Introduced us to the LULESH benchmark

Summer 2011: Cray intern ports LULESH to Chapel

- *caveat*: used structured mesh to represent data arrays

Nov 2011: Chapel team tunes LULESH for single-node performance

Dec 2011: Chapel team visits LLNL (talk, tutorial, 1-on-1 sessions)

Mar 2012: Jeff Keasler (LLNL) visits Cray to pair-program

- in one afternoon, converted from structured to unstructured mesh
- impact on code minimal (mostly in declarations)

Apr 2012: LLNL reports on collaboration at Salishan

Apr 2012: Chapel 1.5.0 release includes LULESH as an example code

Sep-Nov 2012: performance tuning

Nov 2012: SC12

- Chapel HPC Challenge entry
- LLNL talk at the Chapel Lightning Talks BoF
- Cray talk at Proxy Apps BoF

Dec 2012: Multi-institution LULESH paper accepted to IPDPS '13

Next Steps

- **Performance Optimizations and Tuning**
 - Reductions
 - Communication optimizations
 - Aggregation
 - Overlap
 - Atomics
- **Explore array-of-structs vs. struct-of-arrays ideas**
- **Identify funding to dedicate focus on DOE proxy apps**



Codesign Takeaways for Chapel Team

- Improved comprehension of the science behind the code and data structures
- Deeper understanding of array-of-struct vs. struct-of-arrays tensions
- Awareness of performance issues based on past LLNL experience

“The opportunity to work on LULESH with computational scientists at LLNL has been incredibly valuable. In part, this is due to the level of expertise that we’ve had access to. In part it’s due to LULESH’s design: it is compact enough to be manageable for our team to understand while being realistic enough to carry weight with actual users.”

Brad Chamberlain, Chapel Technical Lead, Cray Inc.



Codesign Takeaways for LULESH Team

- Impact of representation-independent features made evident firsthand
- Saw value of using global-view sparse domains to avoid local \leftrightarrow global index translation
- View Chapel as an opportunity for code that ports across next-gen architectures

“Chapel is a maintainable future-proof language. With additional back-end performance enhancements, we would be using it to develop science codes, with an eye towards multiphysics production codes.”

Jeff Keasler, ASC code developer, LLNL



Summary of the LULESH Effort in Chapel

- **Evidence that Chapel's language design is solid**
 - Not just an HPCS technology demonstrator
- **Evidence that people are getting serious about Chapel**
 - LLNL sees Chapel as a serious contender for hydrocodes
- **Co-design success story**
 - Access to experts for a code that people actually care about
 - Feedback on the language and implementation
 - New challenges for the language and implementation

Questions?

