**Maintaining Load Balance and Locality**

### Problem of Load Imbalance
- Load balance necessary to make efficient use of computational resources.
- Load imbalance stems from application (generally coarse-grained, persistent) and architecture (generally fine-grained, transient).

### Problem with a Runtime-based Solution
- Load balance necessary to make effective use of computational resources.
- Need to carefully balance for load balance and locality for application and architecture.

### Key Idea of Solution
1. Correct fine-grained static allocations, and then dynamic allocation.
2. Ratio of static iterations to all iterations is the static fraction (denoted by r).
3. Tune static fraction to maintain both load balance and locality.

**Within-node Results and Analysis**

### Figure 1: Communication-avoiding LU timeline with a static scheduling approach, with performance loss due to load imbalance shown by white spaces.

### Figure 2: Timelines along with L2 and L3 cache misses.

**Implementation and Strategy Optimization**

### Figure 3: Performance of CALU using Mixed Static/Dynamic Scheduling performs better compared to widely known numerical linear algebra libraries.

### Table 1: Percent gains of mixed static/dynamic scheduling over OpenMP static scheduling for NAS benchmarks on Intel Westmere multi-core node.

<table>
<thead>
<tr>
<th>SP</th>
<th>BT</th>
<th>LU</th>
<th>FT</th>
<th>CG</th>
<th>MG</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.14%</td>
<td>5.42%</td>
<td>5.57%</td>
<td>5.31%</td>
<td>14.67%</td>
<td>9.48%</td>
</tr>
</tbody>
</table>

### Table 2: Percent gains with NAS benchmarks on BG/Q multi-core node.

<table>
<thead>
<tr>
<th>SP</th>
<th>BT</th>
<th>LU</th>
<th>FT</th>
<th>CG</th>
<th>MG</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.09%</td>
<td>-1.05%</td>
<td>-1.62%</td>
<td>-1.59%</td>
<td>7.93%</td>
<td>5.04%</td>
</tr>
</tbody>
</table>

**Across-node Results and Scalability**

### Figure 4: Performance analysis of CALU on Intel Westmere 16-core cluster.

**Results Summary**
- At full-scale, mesh obtains good gains of 25.52% over static sched. with slack-conscious sched. (callsite) and has reasonable gains of 10.15% over static sched. with constrained staggered sched. (callsite).
- At full-scale, sched. gets limited gains of 7.56% over mixed static/dynamic (Static Hybrid) with callsite/CG, but gets significant gains of 18.53% over mixed static/dynamic with vSched.
- The full scheduling, which combines callsite/CG and vSched, provides overall gains of 28.16% for mesh, and 46.4% for n-body.

**Conclusions and Future Work**
- Important to tune the balance between load balance and locality.
- Using our approach, we achieve significant gains with a relatively low programmer effort, and perform competitively to guided sched.
- Additional gains can be achieved in full strategy, through more intelligent tuning of the parameters of multiple schedulers.