NPS-NRL-Rice-UIUC Collaboration on Navy Atmosphere-Ocean Coupled Models on Many-Core Computer Architectures: Overview and Progress

Lucas C. Wilcox (LCW) ¹
Timothy Campbell (TC)²
Timothy Warburton (TCW) ⁴
Francis X. Giraldo (FXG) ¹
Andreas Klöckner (AK)³
Timothy Whitcomb (TW) ⁵

¹ Naval Postgraduate School
² Naval Research Laboratory Stennis Space Center
³ University of Illinois at Urbana-Champaign
⁴ Virginia Tech
⁵ Naval Research Laboratory Monterey

ESPC AOLI Meeting 2015
Outline

Port OCCA kernels into NUMA

Adaptive Mesh Refinement
Port OCCA kernels into NUMA

- Uses hand rolled OCCA kernels from gNUMA
- Uses the F90 OCCA host interface
- float-by-float right-hand-side verification for random input data
- Various benchmarks were run with the OCCA kernels
Outline

Port OCCA kernels into NUMA

Adaptive Mesh Refinement
Mesh generation in NUMA

Uses $p4est$

$p8est$

$p6est$

Figure 2: An illustration of meshing with the $p6est$ extension of the $p4est$ library. A macro mesh represents the cubed-sphere domain (left); division of the Morton curve creates the partitions for each MPI process; the columns in the $i^{th}$ partition are ordered by a 2D Morton curve (middle); each radial column is stored as a list of “layers” from the bottom to the top (right).

Figure 3: Wallclock time in seconds of the mesh generation with the $p4est$ library up to the entire machine Mira for the simulations in Fig. 4 and 6.

Provides point-to-point communicators for distributed vectors for these discretizations, using the same data structures as the underlying $p4est$ library.

In $p6est$, however, the order of nodes is changed so that radial columns of degrees of freedom are numbered contiguously.

As the elements within a column are defined by recursive bisection, $p6est$ was designed for meshes with $2^k$ elements per column for some $k$. Because NUMA must work with meshes which do not have this property, the $p6est$ format was extended for this work to support an arbitrary number of elements per column. The runtime of the mesh generation is shown in Fig. 3. Even for 43 billion grid points (blue results) it takes always less than 20 seconds runtime.

It should be noted that, although mesh adaptivity is not used in this work, the $p6est$ data structures support bi-modal local mesh adaptivity: elements may be independently refined in the radial direction, and each column can be independently refined into four smaller columns.

V. Code Optimizations

The goal of this section is to optimize our CG storage version of NUMA. To take advantage of the reduced amount of data compared to DG storage we aim at computing as much work as possible on a per grid point basis and try to avoid making computations on a per element basis. The main structure of our code is illustrated in code example 1.

Code example 1: Pseudocode of the main structure of our code NUMA. The blue text needs to be computed element-wise. The rest (black text) can be computed for each grid point separately. MPI communication is highlighted in red.

We tried to optimize all parts of our code. We found `createrhs` to be the only function that contains enough floating point operations to allow significant optimizations. We expect that we need to merge the different parts of our code into one loop over all elements to optimize the entire code. Since this requires us to move to DG storage we focus in this paper on optimizing `createrhs` for CG storage and leave the optimization of the other parts for the DG storage version of NUMA.

Table I and II show performance measurements for different versions of `createrhs`. For the rest of this section we simply refer to the different versions in these tables.

Originally we expected that the performance of our code would improve with increasing polynomial order ($p = 6$), version A). We found however that order $p = 3$ gives significantly better time to solution (version B) without significant impact on the accuracy of our test cases. We use $p = 3$ for the rest of this paper because this order is very well suited for vectorization on Mira (four double precision floating point numbers fit into one register). Another significant
An element based adaptive mesh refinement package.

- Dynamic adaptive mesh refinement, coarsening, load balancing, repartitioning, continuous node numbering
- Adaptivity handled through a forest of octrees
- The space filling curve facilitates efficient, parallel domain decomposition
RECURSIVE ALGORITHMS FOR FORESTS OF OCTREES

Fig. 8. (left) \( Q_n \)-nodes for \( n = 3 \), with one node at each corner, \( (n-1) \) nodes on each edge, \( (n-1)^2 \) nodes on each face, and \( (n-1)^3 \) nodes in each element. (middle, right) For both conformal and nonconformal interfaces, each element node corresponds to exactly one global node. (right) At nonconformal interfaces, an element may reference a global node remotely, as the small element references the top node.

It is important to note that an element node of a leaf may remotely reference a global node that is contained in the domain of a point that is outside the closure \( \text{Dom}(o) \) and that \( o \) is therefore not in the set of adjacent leaves \( \text{leaf supp}(c) \) defined in (5.4).

Formally, a leaf \( o \) remotely references a point \( c \) in the global partition set \( P_\Omega \) if

\[
    c \not\in \text{leaf supp}(c) \quad \text{and} \quad \exists e \quad \text{such that} \quad o \in \text{leaf supp}(e) \quad \text{and} \quad c \in \text{bound}(e).
\]

This relationship is also shown in Figure 8 (right). From this definition, we can see that the global nodes referenced by leaves in \( O_p \) will be contained in the closed locally relevant partition \( P_p \) (5.3). We note that a point \( c \) can be remotely referenced only if \( \dim(c) < d - 1 \).

6.2. Data structures. On process \( p \), we can represent the global nodes using an array \( N_p \) and the element nodes using a double array \( E_p \), where \( E_p[j][k] \) maps the \( k \)th element node of \( O_p[j] \) to its global node in \( N_p \). \( N_p \) and \( E_p \) only reference locally relevant global nodes and thus implement fully distributed parallelism.

In presenting the \( L_{\text{nodes}} \) algorithm, we consider a global node \( g \) to have the following data fields:

- \( g.i \): the globally unique index of this node,
- \( g.p \): the process that owns \( g \) for the purposes of scatter/gather communication of node values,
- \( g.S_{\text{share}} \): the sharer set of all processes that reference this node.

We include the sharer set \( S_{\text{share}} \) so that, in addition to the scatter/gather communication paradigm, the global nodes can also be used in the share-all paradigm, wherein...
p4est scaling of lnodes on JUQUEEN

Isaac, Burstedde, Wilcox, and Ghattas; DOI:10.1137/140970963

Figure 13: Lnodes (n = 1) runtime in seconds versus P

- Lnodes (n = 1) runtime in secs./(N/P)

- P, 16-way: ○ 16 ▲ 128 □ 1024 ◊ 8192 □ 65536 ◊ 458752
- P, 32-way: ○ 32 ▲ 256 □ 2048 ◊ 16384 □ 131072 ◊ 917504
- P, 64-way: ○ 64 ▲ 512 □ 4096 ◊ 32768 □ 262144
**p4est** scaling of $\text{lnodes}$ on JUQUEEN

Isaac, Burstedde, Wilcox, and Ghattas; DOI:10.1137/140970963

- **Nodes runtime /**
  - Lnodes $(n = 1)$ runtime

<table>
<thead>
<tr>
<th></th>
<th>$10^2$</th>
<th>$10^3$</th>
<th>$10^4$</th>
<th>$10^5$</th>
<th>$10^6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^1$</td>
<td>6.77</td>
<td>1.69</td>
<td>1.12</td>
<td>1.22</td>
<td>1.41</td>
</tr>
<tr>
<td>$10^3$</td>
<td>3.55</td>
<td>0.73</td>
<td>0.95</td>
<td>1.22</td>
<td>1.43</td>
</tr>
<tr>
<td></td>
<td>1.21</td>
<td>0.63</td>
<td>0.94</td>
<td>1.26</td>
<td>1.46</td>
</tr>
<tr>
<td></td>
<td>1.4</td>
<td>0.67</td>
<td>0.97</td>
<td>1.3</td>
<td>1.49</td>
</tr>
<tr>
<td></td>
<td>1.58</td>
<td>0.76</td>
<td>1.07</td>
<td>1.34</td>
<td>1.52</td>
</tr>
<tr>
<td></td>
<td>0.86</td>
<td>1.05</td>
<td>1.2</td>
<td>1.35</td>
<td>1.45</td>
</tr>
</tbody>
</table>

- **Lnodes $(n = 7)$ runtime /**
  - Lnodes $(n = 1)$ runtime

<table>
<thead>
<tr>
<th></th>
<th>$10^2$</th>
<th>$10^3$</th>
<th>$10^4$</th>
<th>$10^5$</th>
<th>$10^6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^1$</td>
<td>0.94</td>
<td>2.16</td>
<td>1.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^3$</td>
<td>2.55</td>
<td>4.56</td>
<td>3.84</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.64</td>
<td>3.35</td>
<td>3.98</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.78</td>
<td>5.15</td>
<td>4.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.96</td>
<td>3.63</td>
<td>4.03</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.42</td>
<td>3.85</td>
<td>4.21</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3.44</td>
<td>3.65</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Mini-apps

- **gNUMA**: testbed for accelerating NUMA
  - single device
  - static conforming mesh
- **bigNUMA**: testbed for accelerating adaptive NUMA
  - multi-device using MPI
  - dynamically adaptive meshes
  - **bfam integrated gNUMA**
    - being developed with Jeremy E. Kozdon
    - thin layer on top of **p4est**
    - provides operators for high-order accurate elements
    - splits the mesh into pieces for multi-physics simulations
    - adaptive mesh routines ported to OCCA
    - also used in beardo (bfam Earthquake And Rupture Dynamics with OCCA)
  - **p4est** on the host and everything else on the device