Towards Exascale Computing with the Atmospheric Model NUMA

Daniel S. Abdi, Andreas Mueller, Lucas Wilcox, Timothy Warburton and Francis Giraldo

2016 Multicore 6 workshop, Sep 14
Boulder, CO, USA
Goal

- NOAA: HIWPP project plan: Goal for 2020:
  ~ 3km – 3.5km global resolution within operational requirements
Goal

- **NOAA: HIWPP project plan: Goal for 2020:**
  ~ 3km – 3.5km global resolution within operational requirements

- **Achieved with NUMA:** baroclinic wave test case at 3.0km within 4.15 minutes per one day forecast on supercomputer Mira
double precision, no shallow atmosphere approx., arbitrary terrain, IMEX in the vertical
NUMA

- NUMA \(^1\) is the dynamical core inside NEPTUNE \(^2\), the U.S. Navy’s next-generation unified global and regional NWP system.
- NEPTUNE was one of five nonhydrostatic atmospheric NWP models involved in a US-wide NOAA program (HIWPP\(^3\)) which aims to compare next-generation NWP models.
- To achieve 3km global resolution in < 8.5 minutes of wallclock time, NUMA must harness the full power of hybrid computers (e.g., Summit and Sierra are IBM/BG with NVIDIA GPUs).

---

\(^1\)NUMA=Nonhydrostatic Unified Model of the Atmosphere is developed and maintained at the Naval Postgraduate School

\(^2\)NEPTUNE=Navy Environmental Prediction sysTem Utilizing the NUMA corE is developed and maintained at the Naval Research Laboratory-Monterey

\(^3\)HIWPP=High Impact Weather Prediction Project (http://hiwpp.noaa.gov/) is a NOAA-run program funded by the Hurricane Sandy Disaster Relief Supplemental Appropriations to help develop new NWP models that track hurricanes better.
## Fastest Supercomputers of the World

according to [top500.org](http://www.top500.org)

<table>
<thead>
<tr>
<th>#</th>
<th>NAME</th>
<th>COUNTRY</th>
<th>TYPE</th>
<th>PEAK (PFLOPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tianhe-2</td>
<td>China</td>
<td>Intel Xeon Phi</td>
<td>54.9</td>
</tr>
<tr>
<td>2</td>
<td>Titan</td>
<td>USA</td>
<td>NVIDIA GPU</td>
<td>27.1</td>
</tr>
<tr>
<td>3</td>
<td>Sequoia</td>
<td>USA</td>
<td>IBM BlueGene</td>
<td>20.1</td>
</tr>
<tr>
<td>4</td>
<td>K computer</td>
<td>Japan</td>
<td>Fujitsu CPU</td>
<td>11.2</td>
</tr>
<tr>
<td>5</td>
<td>Mira</td>
<td>USA</td>
<td>IBM BlueGene</td>
<td>10.0</td>
</tr>
</tbody>
</table>

1 PFlop = $10^{15}$ floating point ops. per sec.
<table>
<thead>
<tr>
<th>#</th>
<th>NAME</th>
<th>COUNTRY</th>
<th>TYPE</th>
<th>PEAK (PFLOPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tianhe-2</td>
<td>China</td>
<td>Intel Xeon Phi</td>
<td>54.9</td>
</tr>
<tr>
<td>2</td>
<td>Titan</td>
<td>USA</td>
<td>NVIDIA GPU</td>
<td>27.1</td>
</tr>
<tr>
<td>3</td>
<td>Sequoia</td>
<td>USA</td>
<td>IBM BlueGene</td>
<td>20.1</td>
</tr>
<tr>
<td>4</td>
<td>K computer</td>
<td>Japan</td>
<td>Fujitsu CPU</td>
<td>11.2</td>
</tr>
<tr>
<td>5</td>
<td>Mira</td>
<td>USA</td>
<td>IBM BlueGene</td>
<td>10.0</td>
</tr>
</tbody>
</table>

1 PFlop = $10^{15}$ floating point ops. per sec.
Titan: GPUs
spectral element and discontinuous Galerkin
Titan

18,688 CPU nodes
(16 cores each)

18,688 NVIDIA GPUs
(2,688 CUDA cores each)
Portability & extensibility: device independent kernel language (or OpenCL / CUDA) and native host APIs.

Available at: [https://github.com/tcew/OCCA2](https://github.com/tcew/OCCA2)
This strategy provides **Portability AND Extensibility**. Here’s what we mean by Extensibility:

1. When new back-ends are available, we change the interface only and **not** the source code.
2. This allows us to insulate ourselves from the uncertainties in the computer-vendor industry.

**Designed for best performance on both GPU and CPU**

1. node-per-thread on the GPU. ($N^3$ gpu-threads are launched to process a 3D element)
2. element-per-thread on the CPU. (1 OpenMP thread is launched to process an element)
Volume and Surface kernels

Figure: Horizontal/Vertical slices for volume integral evaluation

Figure: Coloring of faces for parallel computation of surface integral.
Algorithm for computing $\nabla$, $\nabla \cdot$, and $\Delta$

\begin{verbatim}
procedure GradDiv(q, grad, div, compute)
    Memory fence
    for k,j,i ∈ {0 . . . Nq} do
        sq[k][j][i] = q
    Memory fence
    for k,j,i ∈ {0 . . . Nq} do
        qx=0; qy=0; qz=0;
        for n ∈ {0 . . . Nq} do
            qx += sD[i][n] × sq[k][j][n]
            qy += sD[j][n] × sq[k][n][i]
            qz += sD[k][n] × sq[n][j][i]
        if compute = GRAD then
            grad · x = (qx × Jrx + qy × Jsx + qz × Jtx)
            grad · y = (qx × Jry + qy × Jsy + qz × Jty)
            grad · z = (qx × Jrz + qy × Jsz + qz × Jtz)
        else if compute = DIVX then
            div = (qx × Jrx + qy × Jsx + qz × Jtx)
        else if compute = DIVY then
            div += (qx × Jry + qy × Jsy + qz × Jty)
        else if compute = DIVZ then
            div += (qx × Jrz + qy × Jsz + qz × Jtz)

procedure Grad(q, grad)
call GradDiv(q, grad, -, GRAD)

procedure Div(q, div)
call GradDiv(q·x, -, div, DIVX)
call GradDiv(q·y, -, div, DIVY)
call GradDiv(q·z, -, div, DIVZ)

procedure Lap(q, lap)
call Grad(q, gq)
call Div(gq, lap)
\end{verbatim}

\header{Introduction}{Scalability}{Test cases}{Conclusions}

$\triangleright$ Compute gradient or divergence

$\triangleright$ Load field variables into shared memory

$\triangleright$ Compute local gradients

$\triangleright$ sD are $\nabla \psi$ at LGL nodes preloaded to shared memory.

$\triangleright$ Js are coefficients of the jacobian matrix J

$\triangleright$ Compute gradient of a scalar field

$\triangleright$ Compute divergence of a vector field

$\triangleright$ Compute Laplacian of a scalar field
**Speedup comparison K20X vs 16-core AMD CPU**

<table>
<thead>
<tr>
<th>N</th>
<th>10x10=100 elements</th>
<th>30x30=900 elements</th>
<th>40x40=1600 elements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU</td>
<td>GPU</td>
<td>Speedup</td>
</tr>
<tr>
<td>2</td>
<td>1.46/1.39</td>
<td>0.52/0.47</td>
<td>2.81/2.96</td>
</tr>
<tr>
<td>3</td>
<td>2.68/2.60</td>
<td>0.59/0.49</td>
<td>4.54/5.31</td>
</tr>
<tr>
<td>4</td>
<td>5.30/4.51</td>
<td>0.86/0.54</td>
<td>6.16/8.35</td>
</tr>
<tr>
<td>5</td>
<td>8.12/7.23</td>
<td>1.37/0.77</td>
<td>5.93/9.39</td>
</tr>
<tr>
<td>6</td>
<td>13.89/11.18</td>
<td>2.11/1.07</td>
<td>6.58/10.45</td>
</tr>
<tr>
<td>7</td>
<td>20.49/15.97</td>
<td>2.41/1.31</td>
<td>8.50/12.19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>80x80=6400 elements</th>
<th>120x120=14400 elements</th>
<th>160x160=25600 elements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU</td>
<td>GPU</td>
<td>Speedup</td>
</tr>
<tr>
<td>2</td>
<td>80.72/70.41</td>
<td>13.33/8.94</td>
<td>6.05/7.88</td>
</tr>
<tr>
<td>3</td>
<td>179.07/142.19</td>
<td>18.46/11.18</td>
<td>9.70/12.72</td>
</tr>
<tr>
<td>4</td>
<td>350.54/268.69</td>
<td>32.71/19.02</td>
<td>10.71/14.13</td>
</tr>
<tr>
<td>5</td>
<td>587.17/429.66</td>
<td>67.03/32.38</td>
<td>8.76/13.27</td>
</tr>
<tr>
<td>6</td>
<td>925.25/696.25</td>
<td>110.92/52.91</td>
<td>8.34/13.16</td>
</tr>
<tr>
<td>7</td>
<td>1406.61/968.10</td>
<td>130.16/66.41</td>
<td>10.81/14.58</td>
</tr>
</tbody>
</table>

![Graphs showing performance metrics for surface, horizontal volume, vertical volume, and update kernels.](image-url)

**GFLOPS/s** and **GB/s** plots for different kernels and cores, illustrating performance improvements and limitations.
Semi-implicit time integration on the GPU

(a) 3D-IMEX NoSchur kernels performance

(b) 3D-IMEX Schur form kernels performance

18/32
Semi-implicit time integration on the GPU

Figure: Speedup using high-order Additive Runge-Kutta (ARK) time integrators
A 90% weak scaling efficiency on Titan Supercomputer using 16384 K20x GPUs

Figure: GPU scalability study on Titan supercomputer
Figure: Strong scalability test for 3km, 10km and 13km resolution global simulation on the sphere.
Mira: Blue Gene
spectral elements
Strong scaling with NUMA
1.8 billion grid points (3.0km horizontal, 31 levels vertical)

baroclinic instability, p=3, 3.0km horizontal resolution

Strong scaling with NUMA
1.8 billion grid points (3.0km horizontal, 31 levels vertical)

99.1% strong scaling efficiency

3.14M threads
Optimization of main computations

1 MPI process per core, $p=6$
(January 2015)  
$p=3$  
4 MPI processes per core
(March 2015)

Lapack
rewritten for optimized compiler vectorization
BG/Q vector intrinsics
(May 2015)

OpenMP
optimized storage
merging functions

0 3 6 9 12

0.4 sec.  
1.1 sec.  
2.1 sec.  
3.9 sec.  
4 sec.  
9.4 sec.  
11.8 sec.

12x faster, expect 30x faster

12x faster,
expect 30x faster

done  
future
Roofline Mira: rising bubble

28.5 GB/s memory bandwidth

204.8 GFlops/s peak

5.5x

9.8x

operational intensity (Flops/Bytes)

GFlops per node

attainable timeloop creatorhs

Roofline Mira: rising bubble

28.5 GB/s memory bandwidth

204.8 GFlops/s peak

5.5x

9.8x

operational intensity (Flops/Bytes)

GFlops per node

attainable timeloop creatorhs
Roofline Mira: baroclinic instability

![Diagram showing operational intensity (Flops/Bytes) vs. GFlops per node with 28.5 GB/s memory bandwidth and 204.8 GFlops/s peak performance.](image)

- **28.5 GB/s memory bandwidth**
- **204.8 GFlops/s peak**
- **10.4x and 3.4x** performance ratios

The diagram illustrates the operational intensity and GFlops per node for different applications, demonstrating the attainable performance on the Mira supercomputer.
12.1% of theoretical peak (flops)

Baroclinic instability, p=3, 3.0km horizontal resolution

1.21 PFlops on entire Mira
Where are we heading?
dynamics within 4.5 minutes runtime per one day forecast

baroclinic instability, p=3, 31 vertical layers, 128 columns per node

number of threads

resolution in km

current version
Where are we heading?
dynamics within 4.5 minutes runtime per one day forecast

![Graph showing resolution in km vs. number of threads for Mira and fully optimized versions.](image)

baroclinic instability, p=3, 31 vertical layers, 128 columns per node

current version

fully optimized
Where are we heading?
dynamics within 4.5 minutes runtime per one day forecast

baroclinic instability, $p=3$, 31 vertical layers, 128 columns per node

resolution in km

number of threads

1 2 3 5 10 20 30 50 100

Aurora 2018?
Mira

current version

fully optimized
Knights Landing
(KNL)
Knights Landing (KNL) results

- KNL operates like a standard Intel CPU without the need for code modifications
- No difference between threads and processes observed

<table>
<thead>
<tr>
<th>Processes</th>
<th>Threads per process</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>128</td>
<td>83.2</td>
</tr>
<tr>
<td>64</td>
<td>101</td>
</tr>
<tr>
<td>32</td>
<td>184</td>
</tr>
<tr>
<td>16</td>
<td>101</td>
</tr>
<tr>
<td>8</td>
<td>101</td>
</tr>
<tr>
<td>4</td>
<td>102</td>
</tr>
<tr>
<td>2</td>
<td>104</td>
</tr>
<tr>
<td>1</td>
<td>107</td>
</tr>
</tbody>
</table>
Knights Landing (KNL) results

(a) 3D-IMEX in NoSchur form kernels performance

(b) 3D-IMEX in Schur form kernels performance
Knights Landing (KNL) results

- Number of auto-vectorized loops vs. Polynomial order
- Speedup over non-vectorized version (Base NUMA, float1, float4, float8 vectorization)

Graphs showing the relationship between polynomial order and auto-vectorization as well as speedup over non-vectorized versions.
Knights Landing (KNL) results

![Graph showing scaling efficiency (%) vs. number of KNL nodes]

The graph displays the scaling efficiency (%) for different numbers of KNL nodes. The efficiency decreases slightly as the number of nodes increases, indicating good scaling performance.
Thank you for your attention!