Parallel Performance Optimizations on Unstructured Mesh-Based Simulations

Abhinav Sarje
asarje @ lbl.gov

02 June 2015
International Conference on Computational Science 2015
Reykjavík Iceland
<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abhinav Sarje</td>
<td>Lawrence Berkeley National Laboratory</td>
</tr>
<tr>
<td>Sukhyun Song</td>
<td>University of Maryland College Park</td>
</tr>
<tr>
<td>Douglas Jacobsen</td>
<td>Los Alamos National Laboratory</td>
</tr>
<tr>
<td>Kevin Huck</td>
<td>University of Oregon</td>
</tr>
<tr>
<td>Jeffrey Hollingsworth</td>
<td>University of Maryland College Park</td>
</tr>
<tr>
<td>Allen Malony</td>
<td>University of Oregon</td>
</tr>
<tr>
<td>Samuel Williams</td>
<td>Lawrence Berkeley National Laboratory</td>
</tr>
<tr>
<td>Leonid Oliker</td>
<td>Lawrence Berkeley National Laboratory</td>
</tr>
</tbody>
</table>
Introduction

• *Structured* and *unstructured* meshes in real-world simulations.

• Parallel applications – Not straightforward to partition unstructured meshes.

• Load balance directly related to mesh partitioning quality.

• Data exchange between processes through partition *ghost* or *halo* regions.

• Two key parallelization challenges:
  • Load imbalance across processes – mesh partitioning.
  • Unstructured data access patterns – data organization.
Introduction

- *Structured* and *unstructured* meshes in real-world simulations.
- Parallel applications – Not straightforward to partition unstructured meshes.
- Load balance directly related to mesh partitioning quality.
- Data exchange between processes through partition *ghost* or *halo* regions.
- Two key parallelization challenges:
  - Load imbalance across processes – mesh partitioning.
  - Unstructured data access patterns – data organization.
Introduction

- **Structured** and **unstructured** meshes in real-world simulations.
- Parallel applications – Not straightforward to partition unstructured meshes.
- Load balance directly related to mesh partitioning quality.
- Data exchange between processes through partition *ghost* or *halo* regions.
- Two key parallelization challenges:
  - Load imbalance across processes – mesh partitioning.
  - Unstructured data access patterns – data organization.
Ocean Modeling with MPAS-Ocean

- MPAS = Model for Prediction Across Scales. [Los Alamos]
- A multiscale method.
- Voronoi tessellation-based variable resolution mesh (SCVT).
Ocean Modeling with MPAS-Ocean
Ocean Modeling with MPAS-Ocean

- **Major advantages of such unstructured mesh:**
  - Offers variable resolutions. User defined density functions.
    - Focus on area of interest with high resolution.
    - Avoid unnecessary high-resolution computations in unwanted areas.
  - Smooth resolution transition regions.
  - Locally homogeneous/quasi-uniform coverage of spherical surfaces.
  - Preserve symmetry/isotropic nature of a spherical surface.
  - Naturally allows for discontinuities in the mesh.
  - Straightforward distortion-free mapping to 2D.
- A vertical quantization adds 3rd dimension, representing ocean depths.
Ocean Modeling with MPAS-Ocean

• **Major advantages of such unstructured mesh:**
  • Offers variable resolutions. User defined density functions.
    • Focus on area of interest with high resolution.
    • Avoid unnecessary high-resolution computations in unwanted areas.
  • Smooth resolution transition regions.
  • Locally homogeneous/quasi-uniform coverage of spherical surfaces.
  • Preserve symmetry/isotropic nature of a spherical surface.
  • Naturally allows for discontinuities in the mesh.
  • Straightforward distortion-free mapping to 2D.

• A vertical quantization adds 3rd dimension, representing ocean depths.
Ocean Modeling with MPAS-Ocean

• Major advantages of such unstructured mesh:
  • Offers variable resolutions. User defined density functions.
    • Focus on area of interest with high resolution.
    • Avoid unnecessary high-resolution computations in unwanted areas.
  • Smooth resolution transition regions.
  • Locally homogeneous/quasi-uniform coverage of spherical surfaces.
  • Preserve symmetry/isotropic nature of a spherical surface.
  • Naturally allows for discontinuities in the mesh.
  • Straightforward distortion-free mapping to 2D.

• A vertical quantization adds 3rd dimension, representing ocean depths.
Ocean Modeling with MPAS-Ocean

- Major advantages of such unstructured mesh:
  - Offers variable resolutions. User defined density functions.
    - Focus on area of interest with high resolution.
    - Avoid unnecessary high-resolution computations in unwanted areas.
  - Smooth resolution transition regions.
  - Locally homogeneous/quasi-uniform coverage of spherical surfaces.
  - Preserve symmetry/isotropic nature of a spherical surface.
  - Naturally allows for discontinuities in the mesh.
  - Straightforward distortion-free mapping to 2D.

- A vertical quantization adds 3rd dimension, representing ocean depths.
Ocean Modeling with MPAS-Ocean

SCVT Cells and computed quantities.
Unstructured mesh partitioning ...
Mesh Partitioning and Load Imbalance

- Using a straight-forward graph partitioner, such as Metis.

Computational imbalance across partitions/processes
Mesh Partitioning and Load Imbalance

- High computation-communication imbalance across processes in a run with naive partitioning:
Mesh Partitioning and Load Imbalance

- Need for a better mesh partitioner.

- *Hypergraph* representations are known to model communication more accurately than graphs.

- Available partitioners generate a partitioning by,
  1. balancing the number of cells (or weights) across partitions, and
  2. minimizing the total number of edge cuts.

- Problem:
  - Cost due to halo cells is not considered.
  - Unstructured nature makes halo region costs highly variable across partitions.
  - *Deep* halo regions magnify the effects, making them an important factor for load balancing.
Mesh Partitioning and Load Imbalance

• Need for a better mesh partitioner.

• *Hypergraph* representations are known to model communication more accurately than graphs.

• Available partitioners generate a partitioning by,
  1. balancing the number of cells (or weights) across partitions, and
  2. minimizing the total number of edge cuts.

• Problem:
  • Cost due to halo cells is not considered.
  • Unstructured nature makes halo region costs highly variable across partitions.
  • *Deep* halo regions magnify the effects, making them an important factor for load balancing.
Mesh Partitioning and Load Imbalance

- Need for a better mesh partitioner.

- *Hypergraph* representations are known to model communication more accurately than graphs.

- Available partitioners generate a partitioning by,
  1. balancing the number of cells (or weights) across partitions, and
  2. minimizing the total number of edge cuts.

- Problem:
  - Cost due to halo cells is not considered.
  - Unstructured nature makes halo region costs highly variable across partitions.
  - *Deep* halo regions magnify the effects, making them an important factor for load balancing.
Mesh Partitioning and Load Imbalance

• Need for a better mesh partitioner.

• *Hypergraph* representations are known to model communication more accurately than graphs.

• Available partitioners generate a partitioning by,
  1. balancing the number of cells (or weights) across partitions, and
  2. minimizing the total number of edge cuts.

• Problem:
  • Cost due to halo cells is not considered.
  • Unstructured nature makes halo region costs highly variable across partitions.
  • *Deep* halo regions magnify the effects, making them an important factor for load balancing.
Mesh Partitioning and Load Imbalance

- Input mesh ...
Mesh Partitioning and Load Imbalance

• A partition ...
Mesh Partitioning and Load Imbalance

- 1-Halo region...
Mesh Partitioning and Load Imbalance

- 2-Halo region ...
Mesh Partitioning and Load Imbalance

- 3-Halo region ...
Partitioning-Based Cost Modeling

In a partitioning, for a partition $k$, 

- computation cost, $C_\alpha = \frac{1}{F(p)} \left( \sum_{i \in N_k} w_i + a \sum_{i \in H_k} w_i \right)$
- communication cost, $C_\beta = \frac{1}{F(p)} \frac{h_k}{b_k} + \left( \max_{i \in [1,p]} (c_i) - c_k \right)$
Partitioning-Based Cost Modeling

In a partitioning, for a partition $k$,

\[
\text{computation cost, } C_\alpha = \frac{1}{F(p)} \left( \sum_{i \in N_k} w_i + a \sum_{i \in H_k} w_i \right)
\]

\[
\text{communication cost, } C_\beta = \frac{1}{F(p)} \frac{h_k}{b_k} + \left( \max_{i \in [1,p]} (c_i) - c_k \right)
\]
Partitioning-Based Cost Modeling

In a partitioning, for a partition \( k \),

\[
C_\alpha = \frac{1}{F(p)} \left( \sum_{i \in N_k} w_i + a \sum_{i \in H_k} w_i \right)
\]

communication cost, \( C_\beta = \frac{1}{F(p)} \frac{h_k}{b_k} + \left( \max_{i \in [1,p]} (c_i) - c_k \right) \)
Partitioning-Based Cost Modeling

![Graph showing model vs. actual cost across partitioning instances.](image-url)
A Halo-Aware Partitioning Approach

1: procedure HALOAWAREPARTITION(\(G\))
2: Construct sparse matrix \(A\) representing \(G\)
3: Compute \(A^2, \cdots A^l\), and \(A_1 \cdots A_l = \sum^l A^i\)
4: Construct hypergraph \(H_0\) for \(A_1 \cdots A_l\)
5: while not converged do
6: Compute partitioning \(P_i\) of \(H_i\); construct halos for each partition in \(P_i\)
7: Compute cost prediction for each partition \(k\)
8: Assign weights to the cells, distributing halo cost equally among partition cells
9: Compute total partition weights \(W_k\)
10: Compute imbalance measure, \(f_i = \left(1 - \frac{\min_k (W_k)}{\max_k (W_k)}\right)\)
11: Accept \(P_i\) with probability \(m = \min \left(1, e^{\left(f_i - f_{i-1}\right) / 2}\right)\)
12: if \(P_i\) is accepted then
13: Update \(H_i\) with the new cell weights to construct \(H_{i+1}\)
14: else
15: Reject \(P_i\) by setting \(P_i = P_{i-1}\) and \(f_i = f_{i-1}\)
16: end if
17: end while
18: Output last accepted partitioning as the result
19: end procedure
A Halo-Aware Partitioning Approach

1: procedure HALOAWAREPARTITION($G$)
2:    Construct sparse matrix $A$ representing $G$
3:    Compute $A^2, \ldots A^l$, and $A_1\ldots l = \sum^l A^i$
4:    Construct hypergraph $\mathcal{H}_0$ for $A_1\ldots l$
5: while not converged do
6:    Compute partitioning $P_i$ of $\mathcal{H}_i$; construct halos for each partition in $P_i$
7:    Compute cost prediction for each partition $k$
8:    Assign weights to the cells, distributing halo cost equally among partition cells
9:    Compute total partition weights $W_k$
10:   Compute imbalance measure, $f_i = \left(1 - \frac{\min_k(W_k)}{\max_k(W_k)}\right)$
11:   Accept $P_i$ with probability $m = \min \left(1, e^{\left(f_{i-1}-f_i\right)/2}\right)$
12:   if $P_i$ is accepted then
13:       Update $\mathcal{H}_i$ with the new cell weights to construct $\mathcal{H}_{(i+1)}$
14:   else
15:       Reject $P_i$ by setting $P_i = P_{i-1}$ and $f_i = f_{i-1}$
16:   end if
17: end while
18: Output last accepted partitioning as the result
19: end procedure
A Halo-Aware Partitioning Approach

1: procedure HALOAWAREPARTITION(G)
2:    Construct sparse matrix $A$ representing $G$
3:    Compute $A^2, \cdots A^l$, and $A_{1\ldots l} = \sum^{l} A^i$
4:    Construct hypergraph $\mathcal{H}_0$ for $A_{1\ldots l}$
5:    while not converged do
6:        Compute partitioning $P_i$ of $\mathcal{H}_i$; construct halos for each partition in $P_i$
7:        Compute cost prediction for each partition $k$
8:        Assign weights to the cells, distributing halo cost equally among partition cells
9:        Compute total partition weights $W_k$
10:       Compute imbalance measure, $f_i = \left(1 - \frac{\min_k(W_k)}{\max_k(W_k)}\right)$
11:       Accept $P_i$ with probability $m = \min \left(1, e^{\left(f_i-1-f_i\right)/2}\right)$
12:       if $P_i$ is accepted then
13:           Update $\mathcal{H}_i$ with the new cell weights to construct $\mathcal{H}_{(i+1)}$
14:       else
15:           Reject $P_i$ by setting $P_i = P_{i-1}$ and $f_i = f_{i-1}$
16:       end if
17:    end while
18:    Output last accepted partitioning as the result
19: end procedure
A Halo-Aware Partitioning Approach

1: procedure HALOAWAREPARTITION($G$)
2:   Construct sparse matrix $A$ representing $G$
3:   Compute $A^2, \ldots A^l$, and $A_{1\ldots l} = \sum^l A^i$
4:   Construct hypergraph $\mathcal{H}_0$ for $A_{1\ldots l}$
5: while not converged do
6:   Compute partitioning $P_i$ of $\mathcal{H}_i$; construct halos for each partition in $P_i$
7:   Compute cost prediction for each partition $k$
8:   Assign weights to the cells, distributing halo cost equally among partition cells
9:   Compute total partition weights $W_k$
10: Compute imbalance measure, $f_i = \left(1 - \frac{\min_k(W_k)}{\max_k(W_k)}\right)$
11: Accept $P_i$ with probability $m = \min\left(1, e^{\left(f_i-1-f_i\right)/2}\right)$
12: if $P_i$ is accepted then
13:   Update $\mathcal{H}_i$ with the new cell weights to construct $\mathcal{H}_{i+1}$
14: else
15:   Reject $P_i$ by setting $P_i = P_{i-1}$ and $f_i = f_{i-1}$
16: end if
17: end while
18: Output last accepted partitioning as the result
19: end procedure
A Halo-Aware Partitioning Approach

1: **procedure** HALOAWAREPARTITION(G)
2: Construct sparse matrix $A$ representing $G$
3: Compute $A^2, \ldots A^l$, and $A_1 \ldots l = \sum A^i$
4: Construct hypergraph $\mathcal{H}_0$ for $A_1 \ldots l$
5: **while** not converged **do**
6: Compute partitioning $P_i$ of $\mathcal{H}_i$; construct halos for each partition in $P_i$
7: Compute cost prediction for each partition $k$
8: Assign weights to the cells, distributing halo cost equally among partition cells
9: Compute total partition weights $W_k$
10: Compute imbalance measure, $f_i = \left(1 - \frac{\min_k(W_k)}{\max_k(W_k)}\right)$
11: Accept $P_i$ with probability $m = \min\left(1, e^{\left(f_i - 1 - f_i\right)/2}\right)$
12: **if** $P_i$ is accepted **then**
13: Update $\mathcal{H}_i$ with the new cell weights to construct $\mathcal{H}_{(i+1)}$
14: **else**
15: Reject $P_i$ by setting $P_i = P_{i-1}$ and $f_i = f_{i-1}$
16: **end if**
17: **end while**
18: Output last accepted partitioning as the result
19: **end procedure**
A Halo-Aware Partitioning Approach

1: procedure HALOAWAREPARTITION(G)  
2:     Construct sparse matrix $A$ representing $G$  
3:     Compute $A^2, \cdots A^l$, and $A_1\ldots l = \sum_i^l A^i$  
4:     Construct hypergraph $H_0$ for $A_1\ldots l$  
5:     while not converged do  
6:         Compute partitioning $P_i$ of $H_i$; construct halos for each partition in $P_i$  
7:         Compute cost prediction for each partition $k$  
8:         Assign weights to the cells, distributing halo cost equally among partition cells  
9:         Compute total partition weights $W_k$  
10:        Compute imbalance measure, $f_i = \left(1 - \frac{\min_k(W_k)}{\max_k(W_k)}\right)$  
11:        Accept $P_i$ with probability $m = \min\left(1, e^{(f_i-1-f_i)/2}\right)$  
12: if $P_i$ is accepted then  
13:     Update $H_i$ with the new cell weights to construct $H_{(i+1)}$  
14: else  
15:     Reject $P_i$ by setting $P_i = P_{i-1}$ and $f_i = f_{i-1}$  
16: end if  
17: end while  
18: Output last accepted partitioning as the result  
19: end procedure
Performance Improvements with Better Partitioning

![Chart showing communication time in seconds for different numbers of processes. The x-axis represents the number of processes, ranging from 96 to 12,288, and the y-axis represents communication time in seconds, ranging from $10^2$ to $10^4$. The chart includes a comparison between the original and improved partitioning.](chart.png)
Performance Improvements with Better Partitioning
Performance Improvements with Better Partitioning

![Graph showing communication time vs. number of processes for Original and Hypergraph methods. The x-axis represents the number of processes (96, 192, 384, 768, 1536, 3072, 6144, 12288), and the y-axis represents communication time in seconds (10^2 to 10^4). The graph compares Original and Hypergraph methods, with bars showing the difference in communication time.]
Performance Improvements with Better Partitioning

![Graph showing performance improvements with different partitioning methods.]
Performance Improvements with Better Partitioning
Performance Improvements with Better Partitioning

![Graph showing communication time for different partitioning methods and process counts. The graph compares the original, hypergraph, halo-aware, and depth-halo-aware methods, with communication time on a logarithmic scale and process counts ranging from 96 to 12288. The graph indicates that the depth-halo-aware method consistently reduces communication time across all process counts.]
Performance Improvements with Better Partitioning

![Graph showing computation time vs. number of processes with improved partitioning.]
Performance Improvements with Better Partitioning

![Graph showing computation time improvements with varying number of processes. The graph compares original and hypergraph partitioning methods, indicating significant performance gains with hypergraph partitioning.]
Performance Improvements with Better Partitioning

![Graph showing computation time vs. number of processes for different partitioning strategies: Original, Hypergraph, Halo-aware.]
Performance Improvements with Better Partitioning

The graph illustrates the computation time [s] for different numbers of processes and various partitioning strategies. The x-axis represents the number of processes, ranging from 96 to 12288 in powers of 2 (96, 192, 384, 768, 1536, 3072, 6144, 12288). The y-axis shows the computation time in logarithmic scale, ranging from $10^2$ to $10^4$.

- **Original** (yellow bars): Baseline computation time without partitioning.
- **Hypergraph** (red bars): Improved performance with hypergraph partitioning.
- **Halo-aware** (gray bars): Further enhancement with halo-aware partitioning.
- **Depth-Halo-aware** (dark blue bars): Best performance with depth- halo-aware partitioning.

The graph demonstrates that higher partitioning strategies lead to significant performance improvements as the number of processes increases.
Performance Improvements with Better Partitioning
Performance Improvements with Better Partitioning

![Graph showing time comparison between Original total and Hypergraph total](image-url)

- The graph illustrates the time in seconds for different numbers of processes.
- The x-axis represents the number of processes ranging from 96 to 12288.
- The y-axis represents time in seconds, with logarithmic scaling.
- The graph shows a decrease in time as the number of processes increases, indicating improved performance with better partitioning.

Key Observations:
- Improved data locality with better partitioning.
- Significant performance gains at higher numbers of processes.
Performance Improvements with Better Partitioning

![Graph showing time improvements with different numbers of processes. The graph compares Original total, Hypergraph total, and Halo-aware total times.]
Performance Improvements with Better Partitioning

![Graph showing performance improvements with better partitioning]

- Original total
- Hypergraph total
- Halo-aware total
- Depth-Halo-aware total
Performance Improvements with Better Partitioning
Unstructured data organization ...
Data Locality with Data Ordering

A complete random organization  Original/Reverse Cuthill-McKee
Ordering Unstructured Data with Space Filling Curves

Hilbert SFC

Morton/Z-SFC
Performance Improvements with Data Re-ordering: Cache Usage
Performance Improvements with Data Re-ordering
End Notes

- Overall improved performance by up to $2.2 \times$.
- Improved scaling.
- Enable increased resolution and throughput of high resolution meshes.
- Achieve high SYPD (Simulated Year Per Day.)
- Enable higher accuracy with high resolution.
- Partitioning and ordering methods are generic to apply to other unstructured meshes.
- Collaborations ... ?
Acknowledgements

• Thanks to Aydin Buluc and Umit Catalyurek for discussions on mesh partitioning.

• Authors from LBNL were supported by DOE ASCR under contract number DE-AC02-05CH11231.

• Authors from University of Oregon were supported by DOE SciDAC grant DE-SC0006723.

• D. Jacobsen was supported by DOE Office of BER.

• Resources at NERSC were used, supported by DOE Office of Science under Contract No. DE-AC02-05CH11231.
Thank you!