A Novel Dynamic Load Balancing Scheme for Parallel Systems

Zhiling Lan, Valerie E. Taylor and Greg Bryan

Journal of Parallel and Distributed Computing
24 June 2002

Outline

- AMR/SAMR

- Dynamic Load Balancing (DLB)

- Experimental Results

- Conclusion
Motivation

- Adaptive structure of (S)AMR results in load imbalance between processors
  - Some processors are overloaded while others are underloaded

- Dynamic load balance solves this issue

- How?
  - Moving-Grid Phase
    - Move grid from overloaded to underloaded processor
  - Splitting-Grid Phase
    - Split grid into smaller grids to achieve coarse granularity

AMR/SAMR - Pros and Cons

- AMR/SAMR – Multiscale Algorithm

  - *Adaptive – To take available information into account*

  - *Mesh/Grid - A topology in which nodes form a regular acyclic d-dimensional grid, and each edge is parallel to a grid axis and joins two nodes that are adjacent along that axis*

  - *AMR - Adaptive mesh-algorithm generating a finer resolution mesh near to discontinuities such as boundaries and corners*

  - Pros
    - Expands tractability
    - Models multiscale phenomena

  - Drawback
    - Load imbalance
Dynamic Load Balancing (DLB)

- Solves load imbalance problem

- Grid-Moving vs. Grid Splitting
  - First move the grid then split it into smaller pieces

- Scratch-and-Remap
  - Repartition
  - Remapped
  - Localized regions

- Diffuse-Based
  - Neighbor information
  - Computational domain

DLB Issues

- Overload vs. Underload processors
  - How do you identify them?

- Amount of data redistributed
  - How is the data distributed among processors?

- Overhead
  - How much overhead does DLB impose on the application?
SAMR Architecture – Grid Layout

- Tree of grids
- Uniform mesh
- Finer grids added
- Subgrids
  - Uniform
  - Rectangular
  - Aligned with parent grid
  - Contained within parent grid

SAMR – Execution

- Step by step execution
- Recursive
- Time step based
Enzo – A cosmological application

- Parallel implementation of SAMR
- Coarse granularity
  - Limit quality of load balancing
- High magnitude of imbalance
  - Scratch and Remap better suited
- Different patterns of imbalance
  - Diffuse-Based better suited
- High frequency of adaptation
  - Diffuse-Based scheme better suited

Current Enzo Implementation

- Global grid hierarchy
- Real grid vs. Fake grid
- Use of previous load balance
- Use of global information
Original DLB Algorithm Used In Enzo

The Original DLB Algorithm

```plaintext
while (MaxLoad > threshold * MinLoad) {
    for (i = 0; i < NumberOfGrids; i++)
        if (grid[i] resides in MaxProc && grid[i] < (MaxLoad - MinLoad)/2) {
            Move grid[i] from MaxProc to MinProc;
            Update load information of MaxProc and MinProc;
            Find new MaxProc (MaxLoad) and MinProc (MinLoad);
            Break;
        }
}
```

Fig. 3. Pseudo-code of the original DLB scheme.

- Load balance ratio – MaxLoad/MinLoad
- Threshold – Invoke load balancing process
- MaxProc vs. MinProc

Load Movement Example

![Load Movement Example Diagram]

Fig. 4. An example of load movements using the original DLB scheme.

- Dash line – Required load
- Processor 0 is overloaded
- Processors 1 and 3 underLoaded
- Drawback:  **Load Is not balanced**
Why Make Changes to Enzo?

- Non efficient implementation of SAMR

  - Drawbacks of SAMR
    - Dynamic nature of memory usage
    - Complicated algorithm
    - Interaction between subgrids

Adaptive Characteristics of SAMR

- Four Characteristics:
  - Granularity
  - Magnitude of imbalance
  - Patterns of imbalance
  - Frequency of refinement
Characteristics of SAMR – Cont.

- Three real data sets
  - AMR64
  - AMR128
  - ShockPool 3D

Table 1
Three experimental datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Initial problem size</th>
<th>Final problem size</th>
<th>Number of adaptations</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMR64</td>
<td>32 × 32 × 32</td>
<td>4096 × 4096 × 4096</td>
<td>2500</td>
</tr>
<tr>
<td>AMR128</td>
<td>64 × 64 × 64</td>
<td>8192 × 8192 × 8192</td>
<td>5000</td>
</tr>
<tr>
<td>ShockPool3D</td>
<td>50 × 50 × 50</td>
<td>6000 × 6000 × 6000</td>
<td>600</td>
</tr>
</tbody>
</table>

Coarse Granularity

- Data movement (10kb-10MB)
- Basic entity is grid
- Computational zone
  - Min number of cell
- Ghost zone
  - Obtain boundary info
  - Default size is 3
- Limits quality of LB
Magnitude of Load Imbalance

- **High load imbalance ratio**
  - Load – Total amount of grids in bytes on a processor
  - Ratio – \( \frac{\text{max}(\text{load})}{\text{average}(\text{load})} \)

- Ratio is **Average** over all adaptations

- Load Balance deteriorates as number of processors increases

Patterns of Imbalance

- **AMR64 vs. ShockPool3D**
  - High refinement = processor is overloaded
  - Regular vs. Irregular behavior
  - High quality of load balance not achieved

Fig. 6. Load imbalance ratio (defined as \( \frac{\text{max}(\text{load})}{\text{average}(\text{load})} \)).

Fig. 7. Percentage of refinement per processor.
High Frequency of Refinement

- Adaptation process is invoked
- Number of adaptations varies
  - ShockPool3D – 600 adaptations
  - AMR64 – 2500 adaptations
  - AMR128 – 5000 adaptations
- High adaptations
  - Fast execution

The New DLB Scheme

- Moving-Grid Phase
  - Redistributes grids
- Splitting-grid phase
  - Splits the grid
- Moving-Grid phase is invoked first
- If no more movement, Splitting-Grid phase occurs
Moving-Grid Phase

- Check if MaxLoad/AvgLoad > threshold
- AvgLoad = Required load
- If suitable sized grid, MaxProc moves its grid to MinProc

Moving Grid-Phase vs. The Old DLB Implementation

- **Old Implementation:**
  - Underloaded processor can be overloaded
  - MaxLoad/MinLoad metrics used

- **New Implementation:**
  - Underloaded processor can **NOT** be overloaded
  - MaxLoad/AvgLoad metrics used, hence fewer invocation steps
  - Lower overhead
Splitting Grid Phase

- MaxProc finds the largest grid it owns (MaxGrid)

If

MaxGrid <= AvgLoad – MinLoad
grid moves from MaxProc to MinProc
Else
MaxProc splits the grid

- After splitting MinProc reaches AvgLoad

Fig. 8. Pseudo-code of our DLB scheme.

DLB Algorithm

```c
DLB Algorithm
Done = 0; LastMax = LastMin;
while (MaxLoad > threshold * AvgLoad && Done == 0) {
    for (i=0;i<NumberofGrids;i++) {
        if(MaxProc == LastMax && MinProc == LastMin)
            Done = 1;
        LastMax = MaxProc; LastMin = MinProc;
        find the largest grid MaxGrid on MaxProc;
        split MaxGrid into two and one of them redistributed to MinProc;
        update MaxProc, MinProc, MaxLoad, MinLoad, and AvgLoad;
    } else {
        Done = 1;
    }
}
```

Fig. 8. Pseudo-code of our DLB scheme.
Load Movement - Revised

- Grid 0 is split into 2 smaller grids
- Direct movement of grid 0 from processor 0 to 1
- Split of grid 1 from processor 0 to processor 3
- Improve load balance

Important Observations

- Use of global info
- No thrashing
- No overloading
- Parallel execution of both phases
Experimental Results

- 250 MHZ R1000 SGI Origin2000 machines used
- Performance counter and timers used
  - I/O overhead eliminated
- Threshold is set to 1.20
- Performances measured by:
  - Execution time
  - Quality of load balancing

Comparison Metrics

- Three metrics used to measure quality of load balancing
  - Imbalance Ratio
  \[
  \text{imbalance}_\text{ratio} = \frac{\sum_{j=1}^{N} \frac{\text{MaxLoad}(j)}{\text{AvgLoad}(j)}}{N}
  \]
  - Standard Deviation
  \[
  \text{avg}_\text{std} = \frac{\sum_{j=1}^{N} \sqrt{\frac{\sum_{i=1}^{p} \frac{\text{MaxLoad}(j) \cdot \text{MaxLoad}(i) \cdot \text{AvgLoad}(j)}{p}}{p-1}}}{N}
  \]
  - Percentage of Idle Processors
  \[
  \text{idle}_\text{procs} = \frac{\sum_{j=1}^{N} \text{idle}(j)}{N}
  \]
Results – Execution Time

Table 2
Relative improvement for three datasets

<table>
<thead>
<tr>
<th>Relative improvement</th>
<th>8 Proc. (%)</th>
<th>16 Proc. (%)</th>
<th>32 Proc. (%)</th>
<th>48 Proc. (%)</th>
<th>64 Proc. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM864</td>
<td>12.60</td>
<td>34.15</td>
<td>57.54</td>
<td>54.19</td>
<td>53.69</td>
</tr>
<tr>
<td>AM8128</td>
<td>-4.84</td>
<td>-2.62</td>
<td>20.17</td>
<td>9.14</td>
<td>7.53</td>
</tr>
<tr>
<td>ShockPoolID</td>
<td>8.07</td>
<td>14.18</td>
<td>23.73</td>
<td>24.15</td>
<td>42.13</td>
</tr>
</tbody>
</table>

- Execution time reduced when number of processor is more than 16

- Two Exceptions because:
  - More computational overhead
  - Fewer processors present

Quality of Load Balancing - Imbalance Ratio and Deviation

- Imbalance ratio increases as number of processor increases
- Using the new DLB imbalance ratio is improved
- Larger improvement for more processors
- Standard deviation less for the new DLB scheme

![Imbalance Ratio](image_url)  
Fig. 12. Imbalance ratio for three datasets.
Quality of Load Balancing Percentage of Idle Processors

- Percentage increases as the number of processor increases
- Percentage of idle processors lower
- Less computer resources wasted

![Average Percentage of Idle Processors](image)

Fig. 13: Average percentage of idle processors.

Sensitivity Analysis - Threshold

- Threshold determines whether a load-balance process should be invoked after each refinement

  - Ideal value should be 1.0 **BUT**
    - More overhead will be introduced
    - More load actions will be taken

- Q: What is the ideal value for the threshold?
  - A: Analyze execution time vs. load imbalance
Performance with different Threshold Values – Execution Time

- Smaller values worse performance
  - More overhead introduced

- Larger values worse performance
  - Poor quality of load balance

- Best performance value is 1.25
Performance with different Threshold Values – Quality of Load Balance

- Small threshold
  - Small imbalance ratio
  - High quality of load balance

- Imbalance ratio gets higher as number of processor increases

- Best threshold value is 1.10

- However small values means more load balancing attempts
Summary

- AMR/SAMR Application
- Dynamic load balancing
  - Grid Moving vs. Grid Splitting technique
  - Resolves the load imbalance problem
- Enzo Application
  - Cosmological Application
- Experimental results
  - Improved quality of load balancing
  - Reduce execution time