Is MPI Suitable for a Generative Design-Pattern System?

Paras Mehta, Nelson Amaral, and Duane Szafron

May 6, 2005
Outline

• **Correct Object-Oriented Pattern-Based Parallel Programming System (CO₂P₃S)**

• **MPI/C Advanced Pattern-Based Parallel Programming System (MAP₃S)**
  - Comparison with CO₂P₃S
  - Patterns
  - Results
  - Future work
CO$_2$P$_3$S
Motivation: CO$_2$P$_3$S

- **Correct Object-Oriented Pattern-Based Parallel Programming System**
  1) select a pattern
  2) set parameters
  3) generate framework
  4) insert application-specific code
CO$_2$P$_3$S

- Patterns implemented in Java
  - Five shared-memory patterns using multi-threaded Java
  - Four distributed memory patterns using RMI and Jini
CO$_2$P$_3$S

• **Strengths**
  - object-oriented code lends well to frameworks
  - “write once, run anywhere”
  - user writes only sequential, application-specific code

• **Shortcomings**
  - as a parallel programming system, Java not widely used
  - performance issues
MAP$_3$S

- **MPI/C Advanced Pattern-Based Parallel Programming System**

- **Goals:**
  1) Recreate CO$_2$P$_3$S' usability
  2) Appeal to broader user base
  3) Improve performance
  4) Improve user control of performance
Mesh Pattern

• Mesh: multi-dimensional arrangement of data elements

• Problems involving iteration over matrices
  - Conway's Game of Life
  - Morphogen Pair simulation

• Data parallel problems
Mesh Pattern

- Example
  - 8x8 mesh
Mesh Pattern

• Example
  – 8x8 mesh
  – Iterate over all elements
Mesh Pattern

• Example
  - 8x8 mesh
  - Iterate over all elements
  - Next value depends on neighbours' values
Mesh Pattern

• Example
  - 8x8 mesh
  - Iterate over all elements
  - Next value depends on neighbours' values
Mesh Pattern

- Example
  - 8x8 mesh
  - Divide into 4x4 blocks
Mesh Pattern

- Example
  - 8x8 mesh
  - Divide into 4x4 blocks
  - Distribute round robin to four processes
Mesh Pattern

• Example
  - Need to synchronize at borders between iterations
Mesh Pattern

• Example
  - Need to synchronize at borders between iterations
  - Eight neighbours for each element requires more communication
Mesh Pattern

- Example
  - Toroidality requires still more communication
Mesh Pattern - CO$_2$P$_3$S

- Parameters:
  - neighbours, toroidality, iteration type

- Main Hooks:
  - iteration methods
  - termination condition
  - initialize
  - reduce
Mesh Pattern - $\text{CO}_2\text{P}_3\text{S}$

- Main loop:

```cpp
initialize();
while(notDone()){
    prepare();
    iterate();
}
reduce();
```
Mesh Pattern - $\text{CO}_2\text{P}_3\text{S}$

- Main loop:

```java
initialize();
while(notDone()){
    prepare();
    iterate();
}
reduce();
```
Mesh Pattern - $\text{CO}_2\text{P}_3\text{S}$

- Main loop:

```java
initialize();
while(notDone()){
    prepare();
    iterate();
}
reduce();
```
Mesh Pattern - $\text{CO}_2\text{P}_3\text{S}$

- **Main loop:**

```c
initialize();
while(!notDone()){
    prepare();
    iterate();
}
reduce();
```
Mesh Pattern - CO$_2$P$_3$S

• Main loop:

```
initialize();
while(notDone()){
    prepare();
    iterate();
}
reduce();
```
Mesh Pattern - CO$_2$P$_3$S

- Main loop:

```plaintext
initialize();
while(notDone()){
    prepare();
    synchronize();
    iterate();
}
reduce();
```
Mesh Pattern - MAP₃S

- Same hooks, parameters as CO₂P₃S
- C structures instead of user-defined classes
- explicit communication for structures
- macros instead of hook methods
  - explicit inlining
Search-Tree Pattern

• Problems involving tree traversal
  – Optimization and heuristic search
  – Alpha-Beta search

• Task parallel problems
Search-Tree Pattern

• Start from root
Search-Tree Pattern

- Start from root
- Generate children
Search-Tree Pattern

- Start from root
- Generate children
- Distribute children to multiple processes and compute sequentially
Search-Tree - \( \text{CO}_2 \text{P}_3 \text{S} \)

- \( \text{CO}_2 \text{P}_3 \text{S} \)
  - tasks on a shared work queue
    - tasks represent subtrees to be searched
    - processes pull tasks off of shared queue as necessary
Search-Tree - CO₂P₃S

• parameters:
  - traversal technique (BFS vs. DFS)
  - early termination

• Main Hooks:
  - parallel
  - divide
  - conquer
if(parallel())
    divide();
else
    conquer();
Search-Tree - MAP$_3$S

- divide and conquer methodology
- same parameters and hooks as CO$_2$P$_3$S
- more specific pattern for alpha-beta search
- dynamic load balancing
Search-Tree - MAP$_3$S

- Dynamic Load Balancing
  - multiple queues
  - load balancing by work-stealing
Search-Tree - MAP$_3$S

- Work stealing

Process 0  Process 1  Process 2
Search-Tree - $\text{MAP}_3$S

- Work Stealing

Process 0  Process 1  Process 2
Search-Tree - MAP₃S

- Work Stealing

Process 0  Process 1  Process 2

Diagram of a tree structure with multiple processes.
Search-Tree - MAP$_3$S

- Work Stealing

- Process 0
- Process 1
- Process 2
Search-Tree - MAP\textsubscript{3}S

- Work Stealing
Search-Tree - MAP₃S

- Work Stealing
Search-Tree - MAP₃S

- Work Stealing
Search-Tree - MAP_3S

• Work Stealing
Search-Tree - MAP^3S

- Dynamic Load Balancing
  - comprises a task farm
  - useful for many task-parallel problems
  - adapted for use with the search-tree
Search-Tree - MAP₃S

- Computation Thread:

```java
if(!workListEmpty()){
    getWork();
    processTask();
}
else{
    requestTask();
    wait(request);
}
```
Search-Tree - MAP₃S

• Computation Thread:

```java
if(!workListEmpty()){
    getWork();
    processTask();
}
else{
    requestTask();
    wait(request);
}
```
Search-Tree - MAP₃S

• Computation Thread:

```java
if(!workListEmpty()){
    getWork();
    processTask();
}
else{
    requestTask();
    wait(request);
}
```
Search-Tree - MAP₃S

• Computation Thread:

```java
if(!workListEmpty()){
    getWork();
    processTask();
}
else{
    requestTask();
    wait(request);
}
```
Message Processing Thread:

```c
while(!taskFarmDone){
    MPI_Probe();
    processMessage();
}
```
• Message Processing Thread:

```c
while(!taskFarmDone) {
    MPI_Probe();
    processMessage();
}
```
Search-Tree - MAP₃S

- Message Processing Thread:

```c
while(!taskFarmDone){
  MPI_Probe();
  processMessage();
}
```
Search-Tree - MAP$_3$S

• Message Processing Thread:

```c
while(!taskFarmDone){
    MPI_Probe();
    processMessage();
}
```
• Message Processing Thread:

```c
while(!taskFarmDone){
  MPI_Probe();
  processMessage();
  // receive requests
}
```
Search-Tree - MAP$_3$S

• Message Processing Thread:

```c
while(!taskFarmDone){
    MPI_Probe();
    processMessage();
    // receive requested tasks
}
```
• Message Processing Thread:

```c
while(!taskFarmDone){
    MPI_Probe();
    processMessage();
    // termination cycle
}
```
Results

• Usability
  - For the implemented patterns, MAP₃S recreates the hooks of CO₂P₃S
  - Pattern-user needs write only sequential, computational code
  - Generated communication framework handles the rest
Results

- Speedup – Mesh Pattern:
Results

- Speedup – Search-Tree Pattern:
Results

- **Comparison – \( CO_2 P_3 S \) vs. \( MAP_3 S \)**

Conway's Game of Life - Shared Memory

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>COPS</th>
<th>MAPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>600</td>
<td>500</td>
</tr>
<tr>
<td>1</td>
<td>1000</td>
<td>800</td>
</tr>
<tr>
<td>2</td>
<td>600</td>
<td>400</td>
</tr>
<tr>
<td>3</td>
<td>300</td>
<td>200</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>100</td>
</tr>
</tbody>
</table>
Immediate-Future Work

• Current Patterns
  - test more problems

• New Patterns
  - continue Wavefront
  - develop Pipeline
Future Work

• Current Patterns
  – test with more users
  – performance tuning

• More Patterns

• Pattern-Framework Generator

• Standardize pattern hooks
  – allow for completely different frameworks
Questions?
Search-Tree - MAP₃S

- Start from root
- Generate to given depth
Search-Tree - MAP$_3$S

- Start from root
- Generate to given depth
Search-Tree - MAP$_3$S

- Start from root
- Generate nodes to given depth
- Processes steal leaf nodes from generating process
Search-Tree - MAP$_3$S

- Leaf nodes represent subtrees to be processed sequentially
Search-Tree - MAP$_3$S

- Completed leaf nodes return computational result to parent
Search-Tree - MAP$_3$S

- Completed leaf nodes return computational result to parent
- Updating parent may require message