Supporting
Machine Independent
Parallel Programming
on
Diverse Parallel Architectures

Wayne Fenton
Balkrishna Ramkumar
Vikram Salekore
Amitabh B. Sinha
Laxmikant V. Kale
Motivation

- Wide range of Parallel machines available

- Each parallel machine has different characteristics; programming them is difficult

- Desirable to write “machine independent” programs

- Machine independent programs must run efficiently on all different types of machines
The

Chare Kernel –

A Machine-Independent

Parallel Programming

Language

Parallel Programming Laboratory
University of Illinois, Urbana
Outline of Talk

• Basic Language Features & Implementation

• Additional Language Features & Implementation

• Performance Data & Future Improvements

• Applications
Basic Language Features

- Types of Processes
  - Chares

- Information Sharing Mechanisms
  - Messages
  - Read Only Variables
Processes

• Chares
  • Medium Grained Processes
  • Data Area
  • Functions
  • Entry Points (activated by messages)
  • Functions and Entry Points share the chare’s data area
Syntax of a Chare

char Example1 {
    /* Local variable declarations */
    entry EP1: (message MESSAGE_TYPE1 *msgPtr)
        C-code-block
    ..
    entry EPn: (message MESSAGE_TYPEn *msgPtr)
        C-code-block
    function1 (<parameter-list>)
        C-code-block
    ..
    functionZ (<parameter-list>)
        C-code-block
}

Parallel Programming Laboratory
University of Illinois, Urbana
Information Sharing Mechanisms

- Messages

- Read Only Variables
  - Initialized in the Init Section of the program
  - Remains unaltered thereafter
Basic System Calls

- **CreateChare**(charename, ep, msg)
  - Creates a chare of type charename
  - Activates created chare by sending message msg at entry point ep

- **SendMsg**(ep, msg, cid)
  - Sends a message msg to chare with ID cid at entry point ep
Implementation of Basic Features

- **Pick Next Message**

  - Shared Machines: messages are picked from the shared queues.

  - Nonshared and NUMA Machines: messages are picked from the local queue, where they are inserted after being picked from the net

- **Initialization Loop**

- **Message Processing Loop**
Implementation of Basic Features

• Initialization Loop

• Pick next initialization message

• For a Read Only initialization message create and initialize the corresponding Read Only variable.

  – On shared machines, a single copy of the variable is maintained.

  – On nonshared and NUMA machines the variable is replicated on each node.
Implementation of Basic Features

• **Message Processing Loop**

  • Pick up next message

• **Process Message**

  – For *CreateChar* messages, allocate data area, and call entry point with data area and creation message as parameters

  – For *SendMsg* messages, determine data area from ID, and call entry point with data area and creation message as parameters
Additional Language Features

• **Types of Processes**
  - Branch-Office Chares

• **Information Sharing Mechanisms**
  - Write Once
  - Accumulators
  - Monotonics
  - Dynamic Tables
Types of Processes

• **Branch-Office Chares (BOC)**

  • A representative branch chara on each node

  • A manager chara on node 0.

  • Branch and Manager charaes have the same syntax as a normal chara.

  • Branches and Manager interact with one another and other charaes through `SendMsg-Branch`, `SendMsgManager` and `BranchCall` system calls.
Syntax of a Branch-Office Chare

BranchOffice Example1 {
  manager {
    /* Syntax of a chare */
  }
  branch {
    /* Syntax of a chare */
  }
}
Information Sharing Mechanisms

- **Write Once Variables**
  - Created once during execution; no subsequent modifications
  - Accesses made through an index

- **Accumulator Variables**
  - *counter variable*
  - an operator to *increment* the counter
  - an operator to *combine* two counter variables
Information Sharing Mechanisms (contd.)

- **Monotonic Variables**
  - monotonically changing variable
  - operator to monotonically update variable

- **Dynamic Tables**
  - Table entries are data items identified by a key
  - Three operations: Insert, Delete, Find.
Implementation of Additional Features

- **Branch-Office Chares**

  - **Initialization** of Branches on the nodes done in the **Initialization Loop** along with Read Only variables – set up data area of branches, and call the initialization entry point with appropriate parameters.

  - Messages communicated between branches and manager are processed in the **Message Processing Loop.** Processing similar to the **SendMsg call.**
Implementation of Additional Features

- **SHARED:** Write Once, Monotonics, Accumulators and Dynamic Tables are implemented as shared variables; access is controlled through locks.

- **NONSHARED AND NUMA:** Write Once, Monotonics, Accumulators and Dynamic tables are implemented as Branch-Office Chares. Branch-Office chares are also used to implement load balancing schemes and quiescence detection.
Implementation of Accumulators as BOCs

- Each branch maintains a local copy of the variable.

- All updates on a node are made to the local copy.

- When the value of the accumulator is "demanded", a collection scheme is initiated on the spanning tree on the nodes, with branches propagating value of the subtree to parent node.

- The manager finally reports the value at specified address.
Example for Performance Data

- A symmetric Traveling SalesPerson Example for 20 cities

- Branch & Bound Algorithm used

- The bound is maintained with a monotonic variable

- Number of nodes in the search tree counted with an accumulator variable

- Search is made more efficient by assigning priorities to nodes
## Performance Data

### Sequent

<table>
<thead>
<tr>
<th>Processors</th>
<th>Without Priority</th>
<th></th>
<th>With Priority</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nodes</td>
<td>Time (ms)</td>
<td>Nodes</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>1</td>
<td>245</td>
<td>6370</td>
<td>360</td>
<td>8490</td>
</tr>
<tr>
<td>4</td>
<td>334</td>
<td>2930</td>
<td>363</td>
<td>2240</td>
</tr>
<tr>
<td>16</td>
<td>703</td>
<td>2010</td>
<td>521</td>
<td>1080</td>
</tr>
</tbody>
</table>

### MultiMax

<table>
<thead>
<tr>
<th>Processors</th>
<th>Without Priority</th>
<th></th>
<th>With Priority</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nodes</td>
<td>Time (ms)</td>
<td>Nodes</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>1</td>
<td>245</td>
<td>14988</td>
<td>360</td>
<td>19942</td>
</tr>
<tr>
<td>4</td>
<td>340</td>
<td>6661</td>
<td>363</td>
<td>5397</td>
</tr>
<tr>
<td>8</td>
<td>579</td>
<td>9173</td>
<td>368</td>
<td>4575</td>
</tr>
</tbody>
</table>
Performance Data (contd.)

<table>
<thead>
<tr>
<th>Processors</th>
<th>Without Priority</th>
<th>With Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nodes</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>4</td>
<td>304</td>
<td>1048</td>
</tr>
<tr>
<td>16</td>
<td>1991</td>
<td>3822</td>
</tr>
<tr>
<td>64</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>128</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Inferences from Performance Data

- Performance is good for shared machines.
- Inadequate load balancing scheme for prioritized scheme.
- Need a better lower bound computation.
Applications

- Othello (Chin-Chau Low)

- Incompressible Viscous Flow Computations (Attila Gursoy)

- Circuit Extraction (Balkrishna Ramkumar)

- N-Body Solver (Celso Mendes)

- Parallel Curve Tracing for Robotics (Darrell Stam)

- High Level Support for Divide-and-Conquer Applications (Attila Gursoy)
Code Size Information

• **Shared**
  
  • Machine Independent Code: 7219 lines of C code
  
  • Average Machine Dependent Code: 239 lines of C code

• **Nonshared**
  
  • Machine Independent Code: 9199 lines of C code
  
  • Average Machine Dependent Code: 849 lines of C code

Parallel Programming Laboratory
University of Illinois, Urbana
Conclusions

• Portable Parallel Programming Possible with Proper Selection of Primitives.

• Chare Kernel is a (MIMD) machine-independent parallel programming language.

• Chare Kernel can serve as a bottom-layer for the construction of application-specific machine-independent high-level languages.