Supporting Machine Independent Parallel Programming on Diverse Parallel Architectures

Wayne Fenton

Balkrishna Ramkumar

Vikram Saletore

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

```
chare 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
}
```

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 *CreateChare* 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 chare** on each node
  - A **manager chare** on node 0.
  - Branch and Manager chares have the same syntax as a normal chare.
  - Branches and Manager interact with one another and other chares through *SendMsgBranch*, *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

- **DynamicTables**
  - 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 alongwith 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 Messsage 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				
Processors	*Without Priority*		*With Priority*	
	Nodes	Time (ms)	Nodes	Time (ms)
1	245	6370	360	8490
4	334	2930	363	2240
16	703	2010	521	1080

MultiMax				
Processors	*Without Priority*		*With Priority*	
	Nodes	Time (ms)	Nodes	Time (ms)
1	245	14988	360	19942
4	340	6661	363	5397
8	579	9173	368	4575

Performance Data (contd.)

NCUBE				
Processors	*Without Priority*		*With Priority*	
	Nodes	Time (ms)	Nodes	Time (ms)
4	304	1048	494	3007
16	1991	3822	1166	1623
64	-	-	4146	1615
128	-	-	7974	1418

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

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.