A Parallel Array Abstraction for Data-Driven Objects

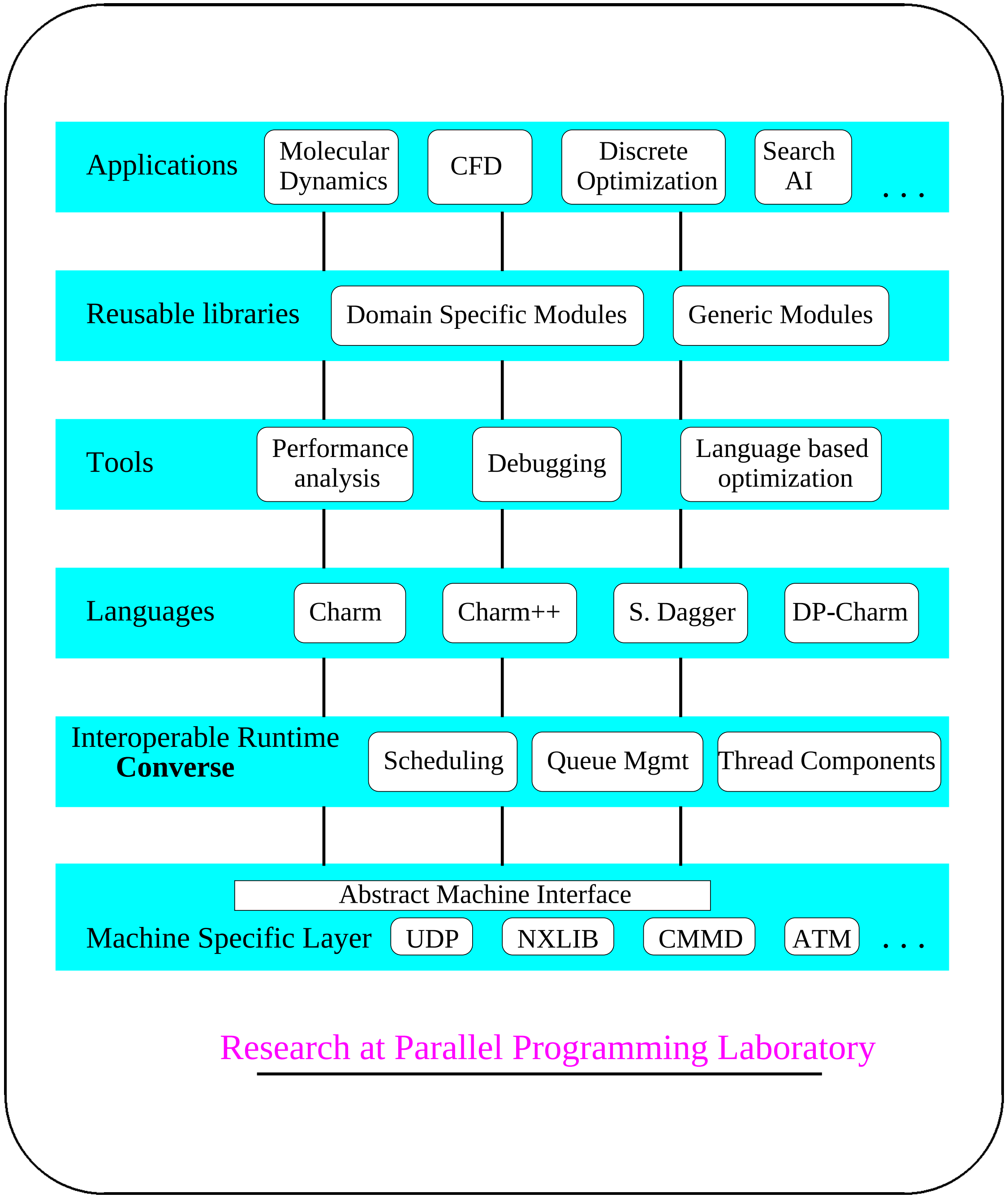
Sanjeev Krishnan and Laxmikant V. Kalé

Department of Computer Science

University of Illinois

kale@cs.uiuc.edu

http://charm.cs.uiuc.edu


Applications
Molecular Dynamics
CFD
Discrete Optimization
Search AI
. . .
Reusable libraries
Domain Specific Modules
Generic Modules
Tools
Performance analysis
Debugging
Language based optimization
Languages
Charm
Charm++
S. Dagger
DP-Charm
Interoperable Runtime
Converse
Scheduling
Queue Mgmt
Thread Components
Abstract Machine Interface
Machine Specific Layer
UDP
NXLIB
CMMD
ATM
. . .
Research at Parallel Programming Laboratory

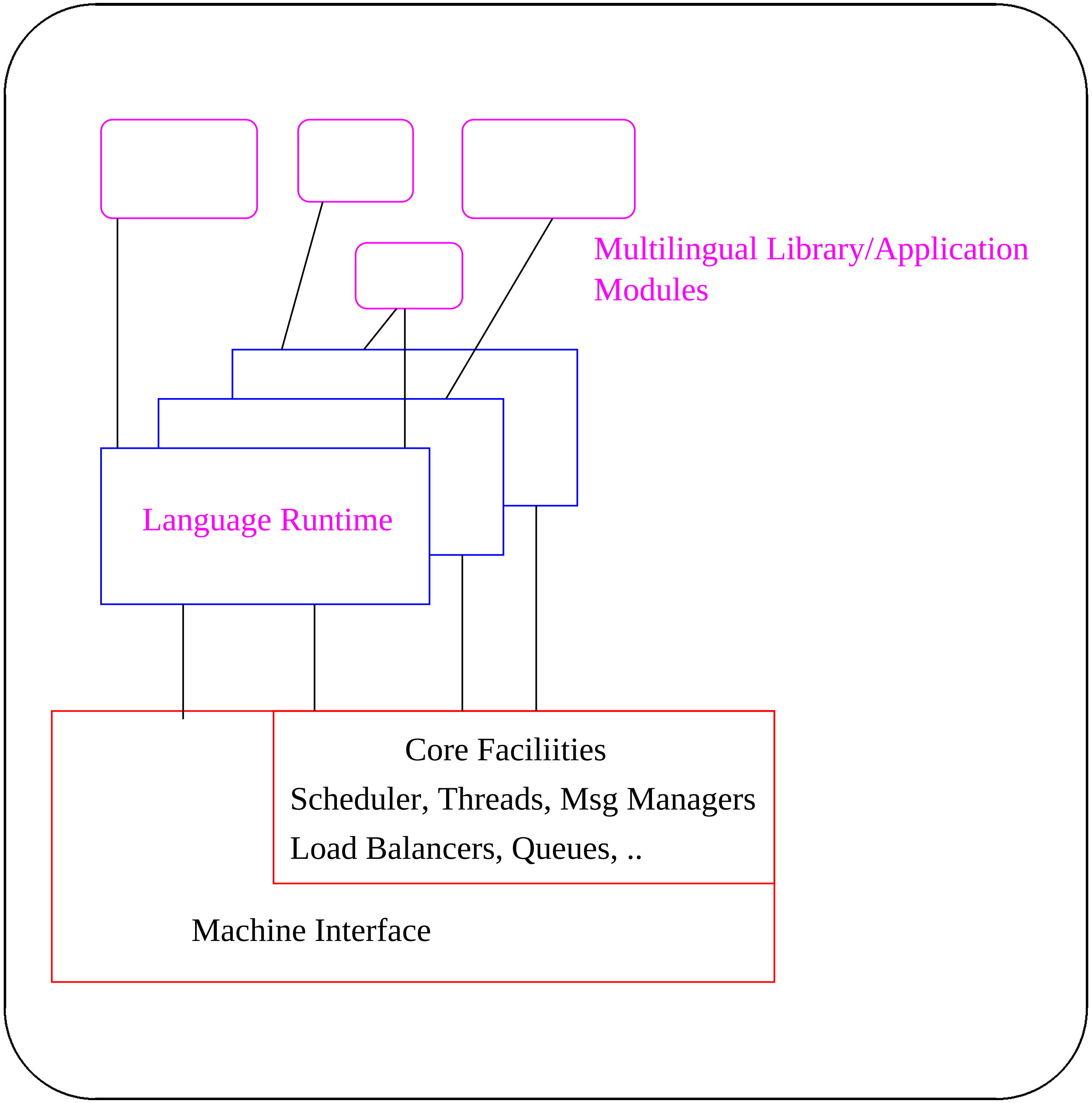
The Parallel Programming Framework

- Converse: multilingual interoperability
- Languages: Charm++, Charm, DP (HPF), tSM, tPVM, POL, ..
- Libraries:
- Applications:

What is CONVERSE?

CONVERSE is an *interoperable parallel runtime system* that is designed to support execution of programs with *modules written in different parallel languages*.

- Facilitates development of runtime systems for new languages.
- Supports coexistence of multilingual modules, including scheduling.

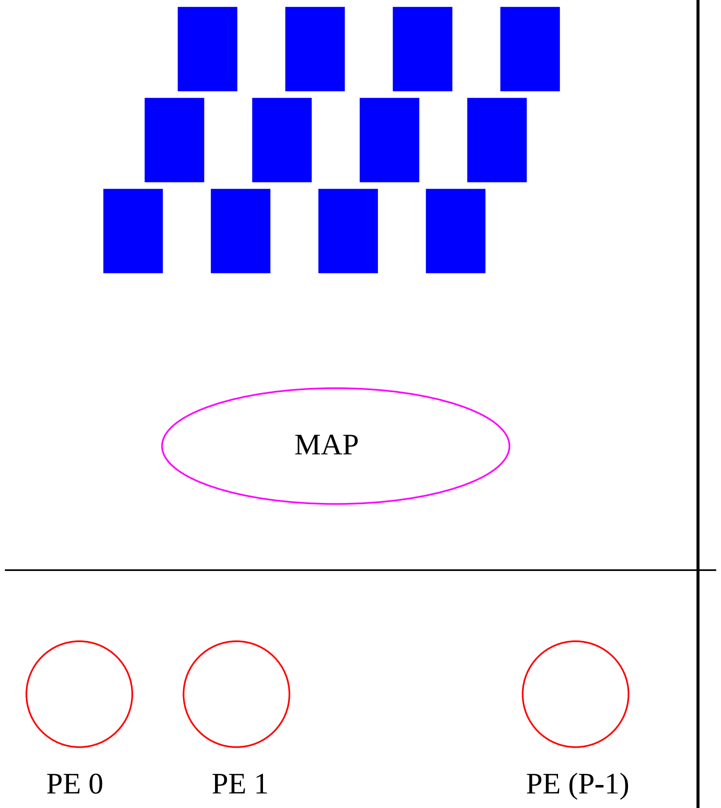

Multilingual Library/Application Modules
Language Runtime
Core Faciliities
Scheduler, Threads, Msg Managers
Load Balancers, Queues, ..
Machine Interface

Charm++ : Review

- Separation of sequential and parallel objects.
- Chares: message driven objects, dynamically load balanced.
- Asynchronous method invocation. (message driven execution)
- Branch office chare: and object with a branch on every processor
- Information sharing abstractions: specifically shared objects.

The Parallel Object Array Abstraction

- Multi-dimensional array
- Each element is an active object
- Allows multiple elements / processor
- Arbitrary mappings of objects to processors
- Map function may be user defined
- Mechanics of method invocation, multicast, remap, etc. handled automatically.



Parallel Array Definition

```
chare class MyArray : public array {
        // list of private and public data and function members
    entry:
        // list of "entry functions" where messages are received
        MyArray(MessageType *m) ;  // constructor
        void EntryFunction(MessageType *m) ;
} ;
```

Allows array behaviors to be composed with object behaviors.

Inherited fields :

- thishandle : unique handle (global pointer) of array element.
- thisgroup : global id by which the whole array is known.
- thisi, thisj, thisk : coordinates of the array element.

Parallel Array Creation

```
MapFunctionType mymapfn ;
MessageType *msgptr ;
MyArray  group arrayid1, arrayid2 ;

arrayid1 =  newgroup MyArray[XSize][Ysize](msgptr) ;
arrayid2 =  newgroup ( mymapfn) MyArray[XSize][Ysize](msgptr) ;
```

Mapping Function

- Returns processor which owns an array element.
- Can find the owner using a mapping expression.
- Can implement arbitrary mapping using a fully enumerated list.

```
int MyMapFn(int aid, int i, int j, int k)
{       // Multi-partition
        return ( XArraySize*((i-k+XArraySize)%XArraySize) +
                 (j-k+XArraySize)%XArraySize );
}
```

Parallel Array Messaging

Point-to-point :

```
arrayid[i][j]=>EntryFunction(msgptr) ;
```

Multicast :

```
arrayid[i1..i2][j1..j2]=>EntryFunction(msgptr) ; // sub-array
arrayid[ALL][j]=>EntryFunction(msgptr) ;         // column
arrayid[i][ALL]=>EntryFunction(msgptr) ;         // row
arrayid[ALL][ALL]=>EntryFunction(msgptr) ;       // whole array
```

Local method invocation :

```
arrayid[i][j]->datamember ;
arrayid[i][j]->f(x,y,z..) ;
```

Remapping and migration

Synchronous :

```
arrayid->remap((MapFunctionType)newmapfn, return_chare_handle,
                         &(ReturnChareType::ReturnFunction));
```

Asynchronous :

```
array::migrate((MapFunctionType)newmapfn)
```

called by each individual array element.

Implementation

- Array library implemented using Converse runtime system.
- Implemented as extension to Charm++
- Translator support in progress.
- Can be used in other languages/systems:

Utility

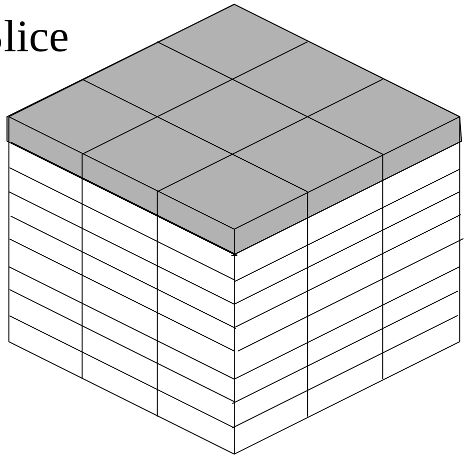
- Implementing distributed services, distributed data structures, global operations. Benefit : encapsulation of complex concurrent operations.
- Multidimensional parallel object arrays can represent a computational space in scientific applications.
- Distributed data exchange between modules
- Encapsulation of processor-specific information.

Example : NAS SP Benchmark

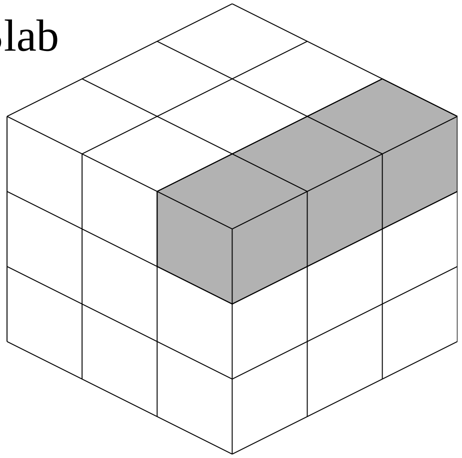
- 3-D computational space
- Each iteration has sweeps along X, Y, Z axes
- Several mappings possible : need to balance load and reduce communication.

NAS SP Benchmark

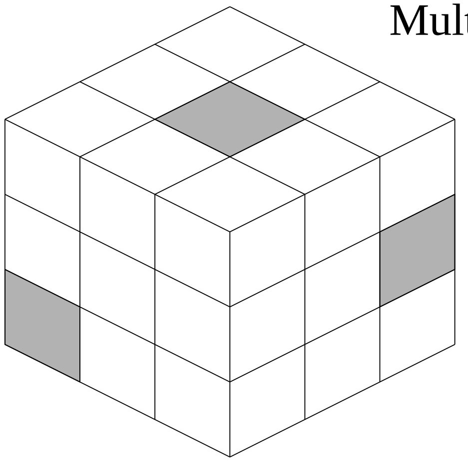
Slice



Slab



Multi-partition



0	1	2	3
3	0	1	2
2	3	0	1
1	2	3	0

2-D multi-partition

NAS SP Benchmark

- Parallel object arrays allow flexible, reusable code
- Different mappings can be easily experimented with
- Synchronous remapping supports transpose
- Asynchronous migration overlaps communication and computation during transpose.

Related Work

- Charm++ : branch office chares
- Concurrent aggregates
- High Performance Fortran
- PC++
- ICC++

Summary

- Multi-dimensional arrays of message driven objects
- User controllable mapping function.
- Asynchronous method invocation, multicast, and broadcast.
- Synchronous and asynchronous remapping.
- Useful for experimenting with alternate mappings, performance tuning..
- Can be used in a multi-lingual program
- More information: http://charm.cs.uiuc.edu