The Charm++ Programming Model

Abhishek Kumar

Contents

- Introduction
- Design Philosophy
- Challenges
- Technical Approach
- Capabilities
- Extensions
- Other Languages

Introduction

Charm++ is an object-oriented asynchronous message passing parallel programming paradigm.

Object-oriented: Program is broken down into a logical collection of objects that interact with each other.

Asynchronous message passing: Messages are sent in a manner that is asynchronous to the code execution

Parallel programming paradigm: Is not a programming language but a way of writing a program

https://charmplusplus.org/tutorial/CharmConcepts.html
https://www.researchgate.net/publication/327527690_Parallel_science_and_engineering_applications_The_charm_approach

Design Philosophy

- Optimal division of labor between the programmer and the system
  - Let the programmers do what they can do best
  - Automate aspects that are tedious for the programmer but relatively easy for a system
- Develop features only in an application-driven manner
  - abstractions or features were added when the application use cases suggested them

Challenges

- Automatic communication/computation overlap
- Load balancing
- Resilience

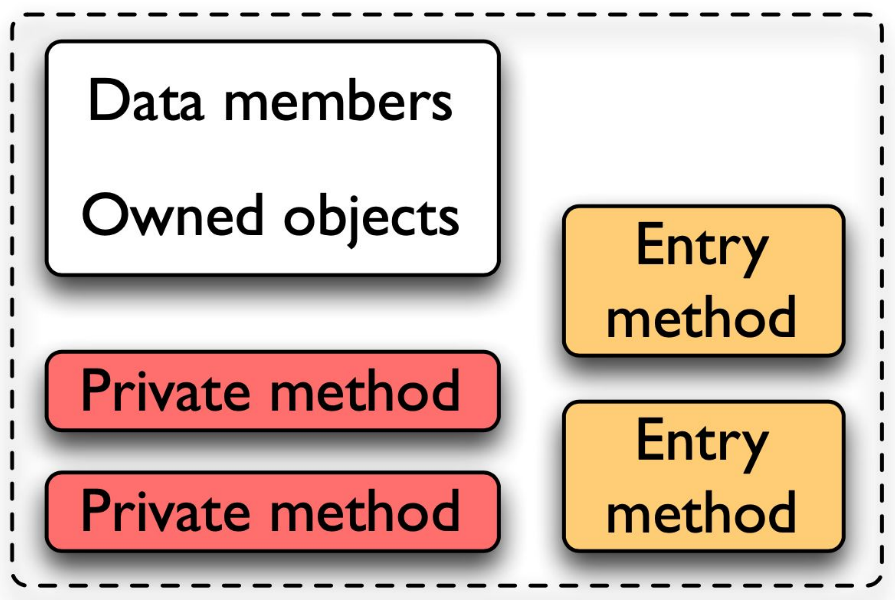
Technical Approach

- Computation is broken down by the programmer into a large number of objects, independent of the number of processors
- Separation of application logic and resource management
- ‘Chares’ are the units of decomposition in Charm++
- Programmer views the overall computation as many chares interacting.

Technical Approach

A chare:

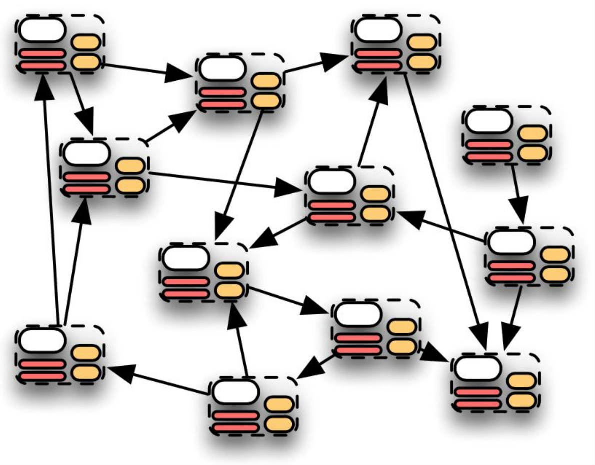
- Has data elements and private and public methods.
- Has public methods that can be remotely invoked and are called ‘entry’ methods.
- Cannot directly access data elements from other chares
  - Processor virtualization

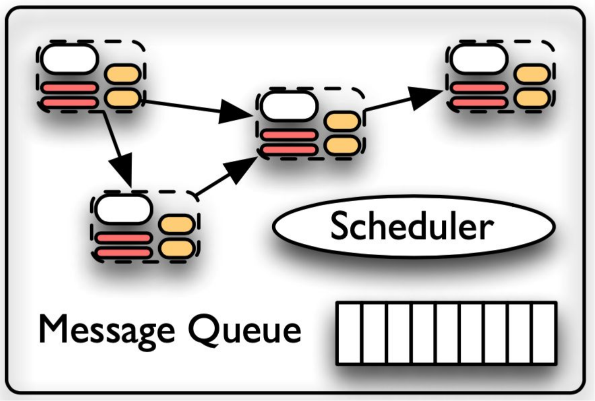


Technical Approach

Programmers point of view:

- Computation consists of collection of chare objects and entry method
- Computation begins with construction of a ‘main chare’
  - Initialize read-only variables
  - RTS copies these on each processor
  - Constructor creates chares and collections of chares
- On each processor
  - scheduler selects one entry method invocations
  - unpacks the parameters
  - executes the entry method with the parameters





Technical Approach

- An abstraction:
    - Many chares can be organized together into a collection (chare arrays)
    - Individual chares can be accessed by an index
    - Chare arrays support reductions and broadcasts over all its elements
        - These are are both asynchronous non-blocking operations
    - RTS assigns the chares belonging to the chare array to processors

Capabilities

- Dynamic load balancing
  - Supports many load balancing strategies
  - Two-phase process
    - Programmer decomposes the work (and data) into chares
    - At runtime, the RTS assigns and reassigns chares to individual processors, to attain such goals as better load balancing, and/or minimization of communication volume
- Automatic checkpointing
  - Handles hardware failures without losing much computation
- Fault tolerance
  - Can run in spite of node crashes in the middle of execution
- Power management
  - Can monitor core temperatures and power draw , and automatically changing frequencies and voltages

Extensions

- Support ‘blocking’ calls
  - Structured Dagger methods
  - Threaded methods
- Processor-awareness
  - Some situations require processor information
    - Libraries or performance oriented optimizations
  - Chare-group: collection of chares
    - exactly one member mapped to each processor

Other Languages

- Adaptive MPI
  - Implementation of the MPI standard on top of Charm++
- MSA
  - Mini-language on top of Charm++
- Charisma
- Charj

The Charm++ Programming Model

Thank you!