Table of Contents

Parallel Programming Models

Many languages and libraries exist for creating parallel applications.

Each presents a programming model to its users.

During this lecture, we'll discuss criteria for evaluating a parallel model and use them to explore various approaches.

OpenMP	Charm++	Linda
Pthreads	UPC	MapReduce
Cilk	STAPL	Matlab DCE
TBB	X10	OpenCL
HPF	Fortress	
MPI	Chapel	

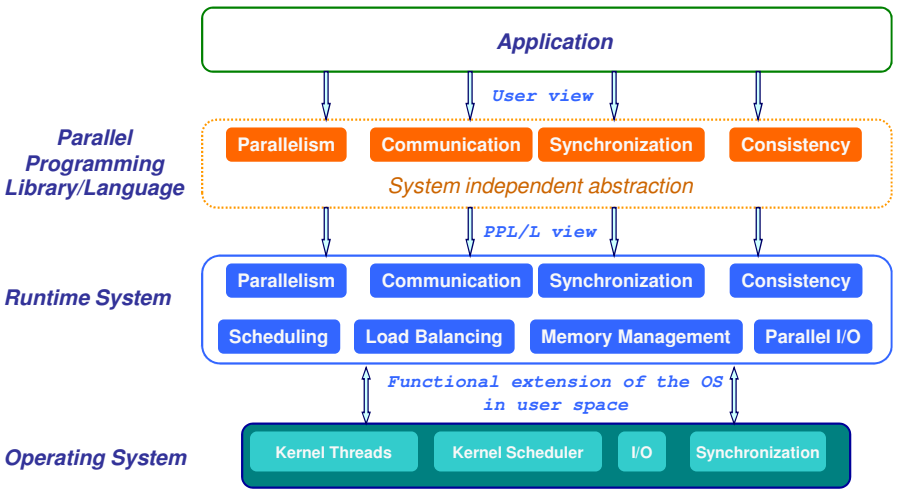
Programming Models Evaluation

What should we consider when evaluating a parallel programming model?

- Parallel Execution Model
- Productivity
- Performance
- Portability

Table of Contents

Parallel Execution Model



Exec Model Productivity Performance Portability

Parallel Execution Model

- Parallel Programming Model (user view)
  - Parallelism
  - Communication
  - Synchronization
  - Memory consistency
- Runtime System (RTS)
  - Introduction, definition and objectives
  - Usual services provided by the RTS
  - Portability / Abstraction

Parallel Programming Model (user view)

- Parallelism
- Communication
- Synchronization
- Memory consistency

PPM – Implicit Parallelism

Implicit parallelism (single-threaded view)

- User not required to be aware of the parallelism
  - User writes programs unaware of concurrency
    - Possible re-use previously implemented sequential algorithms
    - Often minor modifications to parallelize
  - User not required to handle synchronization or communication
    - Dramatic reduction in potential bugs
    - Straightforward debugging (with appropriate tools)
- Productivity closer to sequential programming
- Performance may suffer depending on application
- E.g. Matlab DCE, HPF, OpenMP*, Charm++*

* at various levels of implicitness

PPM – Explicit Parallelism

Explicit parallelism (multi-threaded view)

- User required to be aware of parallelism
  - User required to write parallel algorithms
    - Complexity designing parallel algorithms
    - Usually impossible to re-use sequential algorithms (except for embarrassingly parallel ones)
  - User responsible for synchronization and/or communication
    - Major source of bugs and faulty behaviors (e.g. deadlocks)
    - Hard to debug
    - Hard to even reproduce bugs
- Considered low-level
  - Productivity usually secondary
  - Best performance when properly used, but huge development cost
  - E.g. MPI, Pthreads

PPM – Mixed Parallelism

Mixed view

- Basic usage does not require parallelism awareness
- Optimization possible for advanced users

- Benefits from the two perspectives
  - High productivity for the general case
  - High performance possible by fine-tuning specific areas of the code
- E.g. STAPL, Chapel, Fortress

Table of Contents

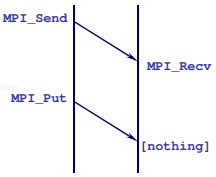
PPM – Implicit Communication

Implicit Communication

- Communication through shared variables
- Synchronization is primary concern
  - Condition variables, blocking semaphores or monitors
  - Full/Empty bit
- Producer/consumer between threads are expressed with synchronizations

- Increases productivity
  - User does not manage communication
  - Reduced risk of introducing bugs

PPM – Explicit Communication



Explicit Communication

- Message Passing (two-sided communication, P2P)
  - User explicitly sends/receives messages (e.g., MPI)
  - User required to match every Send operation with a Receive
  - Implicitly synchronizes the two threads
    - Often excessive synchronization (reduces concurrency)
    - Non-blocking operations to alleviate the problem (e.g., MPI_Isend/Recv)
- One-sided communication
  - User uses get/put operations to access memory (e.g., MPI-2, GASNet, Cray T3D)
  - No implicit synchronization (i.e., asynchronous communication)

PPM – Explicit Communication

Explicit Communication – Active Message, RPC, RMI

- Based on Message Passing
- Messages activate a handler function or method on the remote side
- Asynchronous
  - No return value (no **`get`** functions)
  - Split-phase programming model (e.g. Charm++, GASNet)
    - Caller provides a callback handler to asynchronously process “return” value
- Synchronous
  - Blocking semantic (caller stalls until acknowledgement/return is received)
  - Possibility to use **`get`** functions
- Mixed (can use both)
  - E.g., ARMI (STAPL)

Table of Contents

PPM – Implicit Synchronization

Implicit Synchronization

- Hidden in communication operations (e.g., two-sided communication)
- Data Dependence Graph (DDG)
  - PPL synchronizes where necessary to enforce the dependences
  - E.g., STAPL
- Distributed Termination Detection
  - When implemented as background algorithm (e.g., in Charm++)
- Improved productivity
  - Less bugs from race conditions, deadlocks …
- E.g., STAPL, Charm++, MPI-1 and GASNet (to a certain extent)

PPM – Explicit Synchronization

Explicit Synchronization

- Critical section / locks
  - One thread allowed to execute the guarded code at a time
- Condition variables / blocking semaphores
  - Producer/consumer synchronization
  - Introduces order in the execution
- Monitors / counting semaphores
  - Shared resources management
- Barrier / Fence (global synchronization)
  - Threads of execution wait until all reach the same point
- E.g., Pthreads, TBB, OpenMP

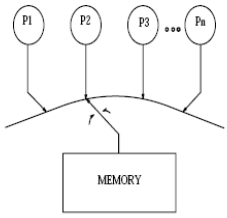
Table of Contents

PPM – Memory Consistency

Introduction to Memory Consistency

- Specification of the effect of Read and Write operations on the memory
- Usual user assumption: Sequential Consistency

Definition: [A multiprocessor system is sequentially consistent if] the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.

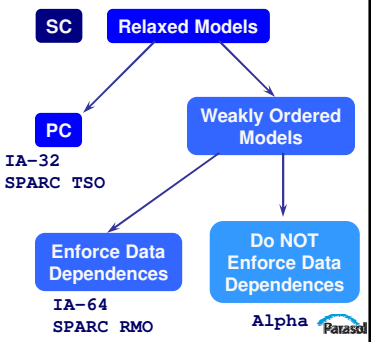


PPM – Memory Consistency

Introduction to Memory Consistency

Sequential Consistency: Don't assume it !

- Sequential Consistency (SC)
  - MIPS/SGI
  - HP PA-RISC
- Processor Consistency (PC)
  - *Relax write→read dependencies*
  - Intel x86 (IA-32)
  - Sun TSO (Total Store Order)
- Relaxed Consistency (RC)
  - *Relax all dependencies, but add fences*
  - DEC Alpha
  - IBM PowerPC
  - Intel IPF (IA-64)
  - Sun RMO (Relaxed Memory Order)



Material from: Hill, M. D. 2003. Revisiting "Multiprocessors Should Support Simple Memory Consistency Models", http://www.cs.wisc.edu/multifacet/papers/dagstuhl03_memory_consistency.ppt

PPM – Memory Consistency

Introduction to Memory Consistency

Example :

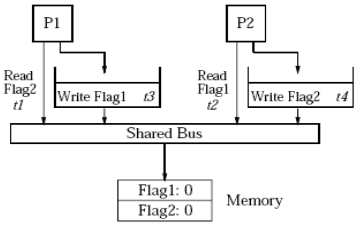
```
// Dekker's algorithm for critical sections
// Initially Flag1 = Flag2 = 0

P1                              P2

Flag1 = 1;        W(Flag1)      Flag2 = 1;        W(Flag2)
If (Flag2 == 0) R(Flag2)        if (Flag1 == 0) R(Flag1)
// critical section             // critical section
...                             ...
```

Correct execution if a processor's Read operation returns 0 iff its Write operation occurred before both operations on the other processor.

- Relaxed consistency: buffer write operations
  - Breaks Sequential Consistency
  - Invalidates Dekker's algorithm
  - Write operations delayed in buffer



Material from & further reading: Adve, S. V. and Gharachorloo, K. 1996. Shared Memory Consistency Models: A Tutorial. Computer 29, 12 (Dec. 1996), 66-76. DOI= http://dx.doi.org/10.1109/2.546611

Exec Model Productivity Performance Portability

PPM – Memory Consistency

Relaxed Memory Consistency Models

- Improve performance
  - Reduce the ordering requirements
  - Reduce the observed memory latency (hides it)
- Common practice
  - Compilers freely reorder memory accesses when there are no dependencies
  - Prefetching
  - Transparent to the user

Parasol

Table of Contents

Runtime System (RTS)

- Introduction
  - Definition
  - Objectives
- Usual services provided by the RTS
- Portability / Abstraction

RTS – Introduction

- Software layer
  - Linked with the application
  - Executes in user space
- Provides applications with functionalities
  - Missing in the Operating System and drivers
  - More advanced/specialized than the OS counterpart

RTS – Definition*

Functional extension of the Operating System in user space

- No precise definition available
- Fuzzy functional boundary between RTS and OS
  - Services are often a refined or extended version of the OS
  - Functional redundancy with OS services
    - Avoid entering Kernel space
    - Provide reentrancy
    - E.g., threading, synchronization, scheduling …
- Widely variable set of provided services
  - No minimum requirements
  - No limit on the amount of functionality

Non-formal, short definition

RTS – Objectives

Objectives of RTS for Parallel Programming Languages/Libraries:

- Enable portability
  - Decouple the PPL from the system
  - Exploit system-specific optimized features (e.g., RDMA, Coprocessor)
- Abstract complexity of large scale heterogeneous systems to enable portable scalability
  - Provide uniform communication model
  - Manage threading, scheduling and load-balancing
  - Provide parallel I/O and system-wide event monitoring
- Improve integration between application and system
  - Use application runtime information
    - Improve RTS services (e.g., scheduling, synchronization)
    - Adaptive selection of specialized code