Parallel Architecture
[High Performance (HPC) Architecture]

Categories based on Flynn's Taxonomy

- Single instruction stream, single data stream (SISD):
  - A sequential computer which exploits no parallelism in either the instruction or data streams
  - One instruction pool acting on one data pool

2

Categories based on Flynn's Taxonomy

- Single instruction stream, multiple data streams (SIMD):
  - A single instruction operates on multiple different data streams
  - SIMD exploits data-level parallelism by applying the same operation to multiple items of data in parallel
  - Data parallelism for
    - Matrix-oriented scientific computing
    - Media-oriented image and sound processors
  - Each processor has its own data memory, but there is a single instruction memory and control processor
  - Types of SIMD architecture:
    - Vector (array) processors
    - Graphics processor units (GPUs)

3

Categories based on Flynn's Taxonomy

- Multiple instruction streams, multiple data streams (MIMD)
  - Multiple autonomous processors simultaneously executing different instructions on different data
  - Types of MIMD architectures:
    - Tightly-coupled MIMD: All the processing elements have shared memory model
    - Loosely-coupled MIMD: All the processing elements have separate memory model
  - Multiprocessors
    - Computers consists of single chip with multiple cores (multicore)
    - Computers consisting of several chips, each may be multicore design

4

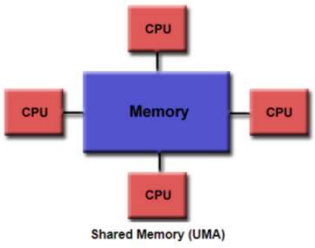
Multiprocessors: Parallel Architecture

- Shared memory architecture
- Distributed memory architecture
- Hybrid architecture

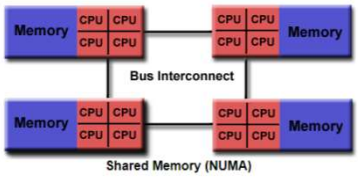
5

Shared Memory Architecture

- Centralised multiprocessors:
  - Multiple processors attached to a bus
  - all the processors share the same primary memory
    - Uniform Memory Access (UMA) or Symmetric Multiprocessor (SMP)
- Distributed multiprocessors:
  - Processors are distributed and connected by bus interconnect
  - Distributed collection of memories forms one logical address space
    - Same address on different processors refers to the same memory location
    - Nonuniform Memory Access (NUMA)



Shared Memory (UMA)



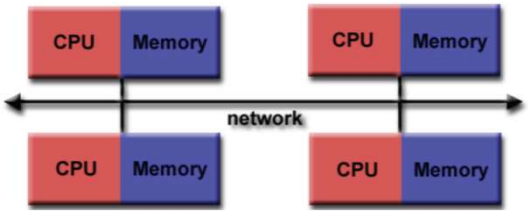
Shared Memory (NUMA)

Support of **Cache Coherence**

6

Distributed Memory Architecture

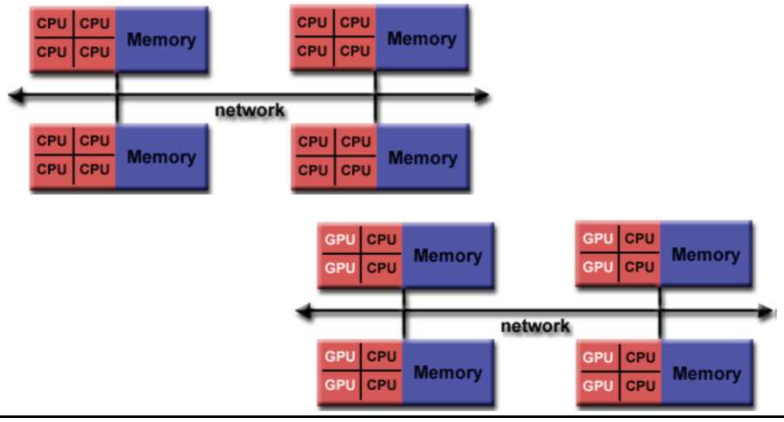
- Distributed multiprocessors:
  - Each processor has its local memory with the address space available only for this processor
  - Processors can exchange data through the interconnection network by means of communication by passing messages
  - Required a communication network to connect inter-processor memory
  - Each processor operated independently
  - No concept of cache coherency



7

Hybrid Memory Architecture

- Distributed multiprocessors:
  - Employ both shared and distributed memory architectures
  - Shared memory could be shared memory machine or with GPUs



8

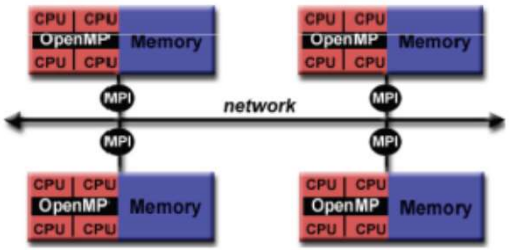
Parallel Programming Models

- Shared memory: All tasks have visibility to a global address space
  - **OpenMP**
- Distributed memory: All tasks have their own private address space and communicate via messages
  - **Message Passing Interface (MPI)**

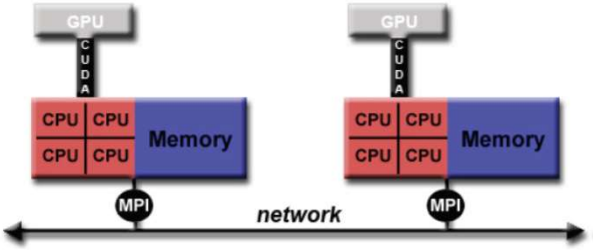
9

Parallel Programming Models

- Hybrid memory: Combination of shared and distributed memory
  - **OpenMP + MPI**



  - **CUDA + MPI**

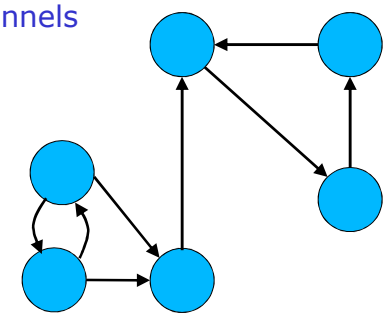


10

Parallel Algorithm Design

Task/Channel Model

- Described by Ian Foster [1]
- Facilitates the development of efficient parallel algorithms, particularly distributed memory architecture
- It represents a parallel computation as a set of tasks that may interact with each other by sending messages through channels

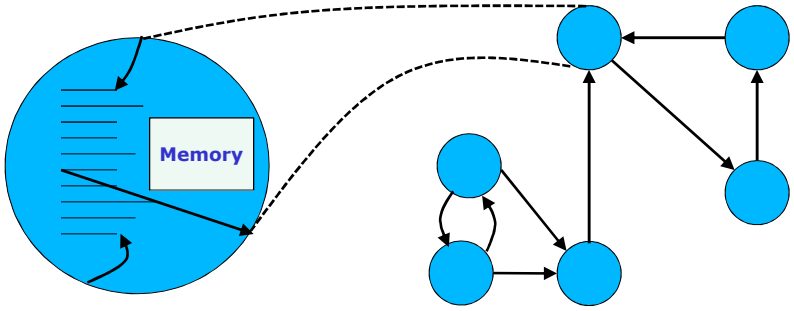


[1] Ian Foster, Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering, Addison-Wesley, 1995.

12

Task/Channel Model

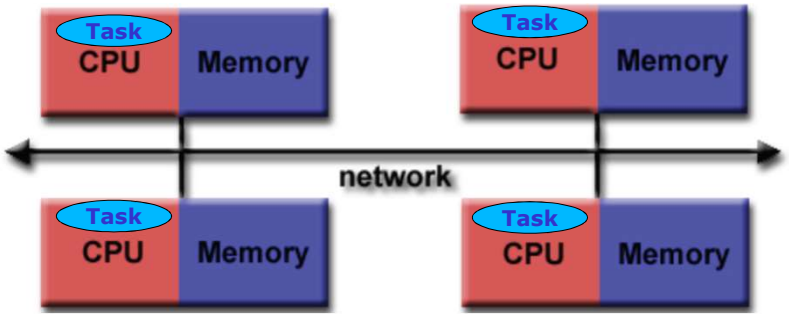
- **Task**: A task is a program, its local memory and a collection of I/O ports
  - The local memory contains the program's instruction and its private data
  - A task can send local data values to other tasks via output ports
  - A task can receive local data values from other tasks via input ports



13

Task/Channel Model

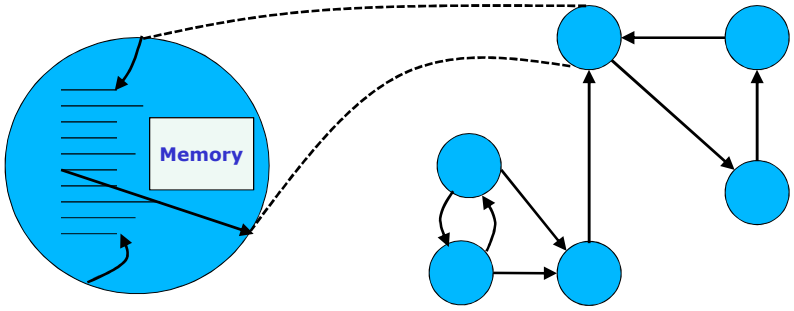
- **Task**: A task is a program, its local memory and a collection of I/O ports
  - The local memory contains the program's instruction and its private data
  - A task can send local data values to other tasks via output ports
  - A task can receive local data values from other tasks via input ports



14

Task/Channel Model

- **Task**: A task is a program, its local memory and a collection of I/O ports



- **Channel**: Message queue that connects one task's output port with another task's input port

15

Foster's Design Methodology

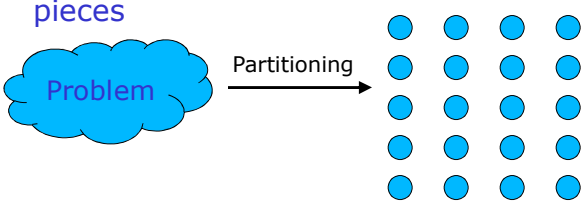
- Four-step process for designing parallel algorithm [1]
  - Partitioning
  - Communication
  - Agglomeration
  - Mapping

[1] Ian Foster, Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering, Addison-Wesley, 1995.

16

Partitioning

- Process of dividing the computation and data into pieces

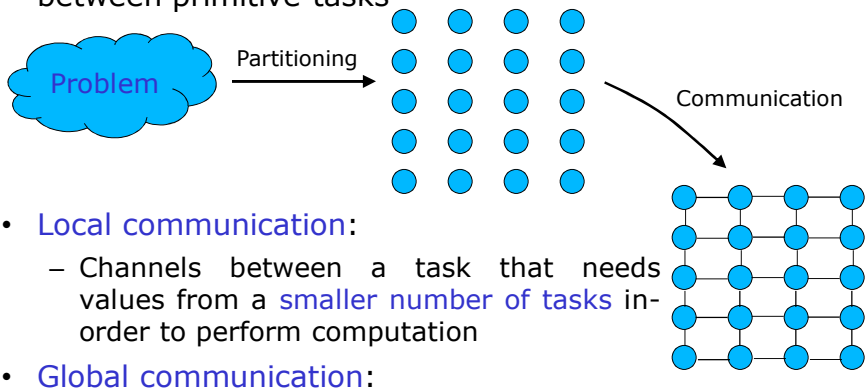


- Domain decomposition (data-centric):
  - First divide the data into pieces and then determine how to associate computations with the data
  - Focus on largest or frequently accessed data structure
- Functional decomposition (computation-centric):
  - First divide the computation into pieces and then determine how to associate data item into individual computations
  - Collection of tasks and running concurrently
- ***Primitive task***

17

Communication

- Process of determining the communication pattern between primitive tasks

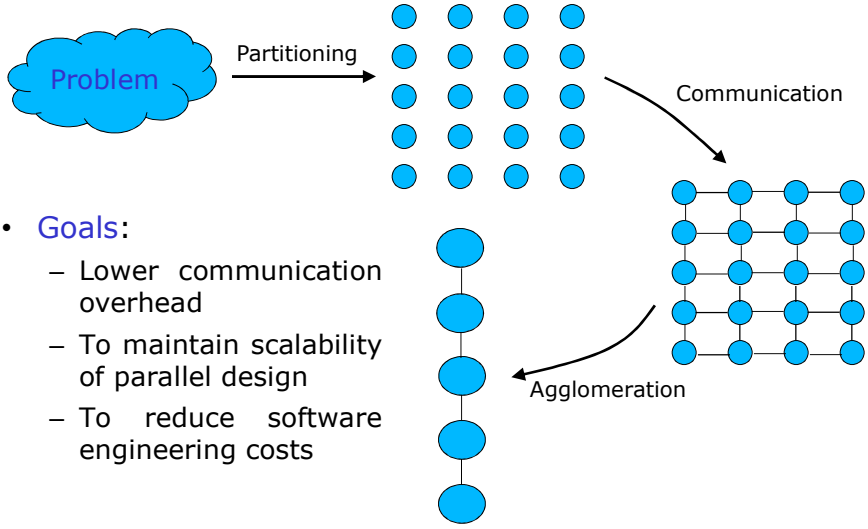


- Local communication:
  - Channels between a task that needs values from a smaller number of tasks in-order to perform computation
- Global communication:
  - Channels between a task and a significant number of primitive tasks in-order to perform computation

18

Agglomeration

- Process of grouping tasks into larger tasks in-order to improve performance or simplify programming

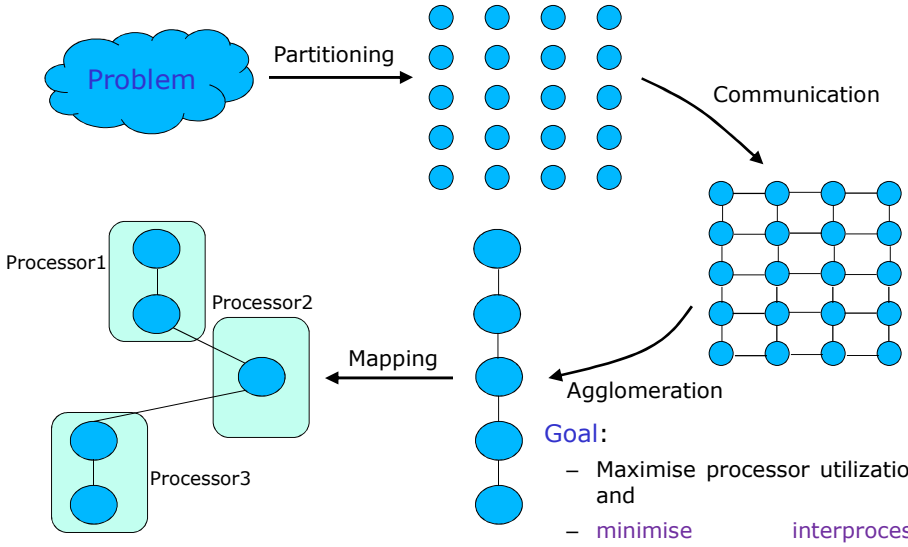


- Goals:
  - Lower communication overhead
  - To maintain scalability of parallel design
  - To reduce software engineering costs

19

Mapping

- Process of assigning tasks to processors



Goal:
- Maximise processor utilization and
- minimise interprocess communication

20

Example: Finding Minimum in a Matrix

21

Text Books

- M. J. Quinn, *Parallel Programming in C with MPI and OpenMP*, McGraw-Hill, 2004.
- Ian Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*, Addison-Wesley, 1995.

22