Introduction to Parallel Programming

Joey Bernard

ACEnet Computational Research Consultant

November 2008

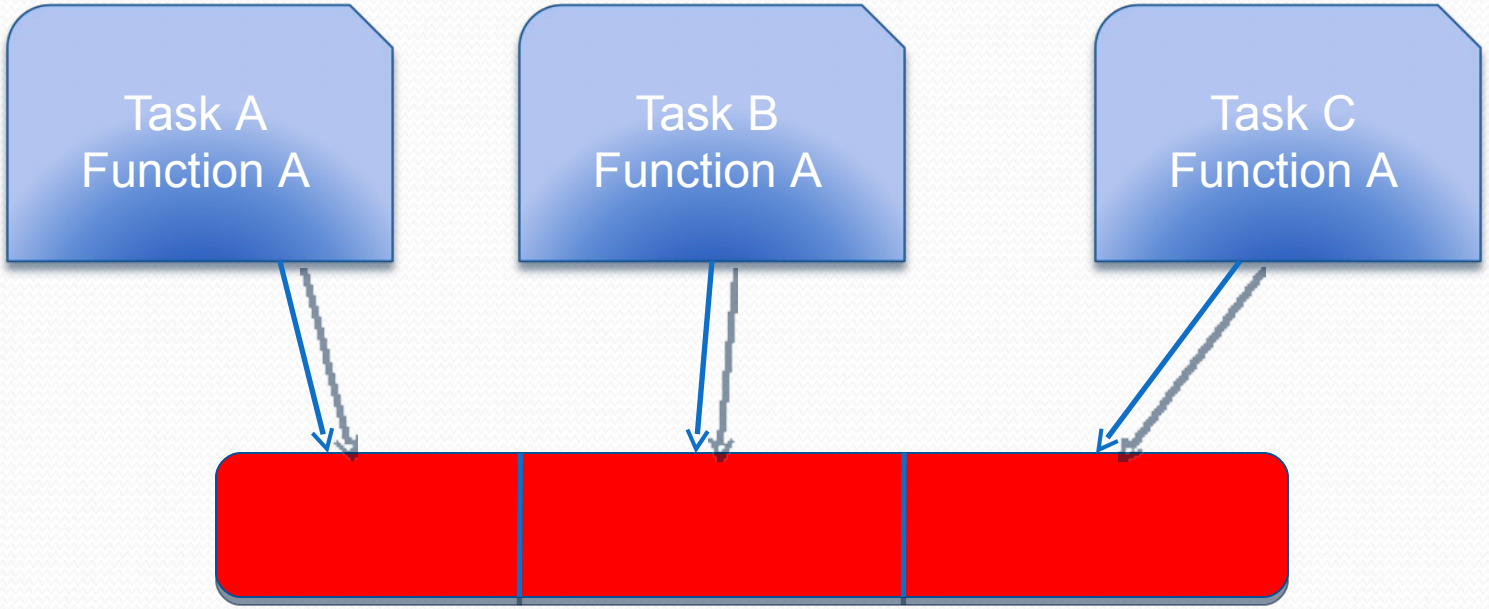
What is Parallel Computing?

- The use of a parallel computer to reduce the time needed to solve a single computational problem
- A parallel computer is a multiple-processor computer
- A multicomputer is a parallel computer made up of multiple computers interconnected
- A symmetrical multiprocessor (SMP) is one in which all CPUs share access to a single global memory
- ACEnet can support both types

Types of Parallelism

- Data Parallelism

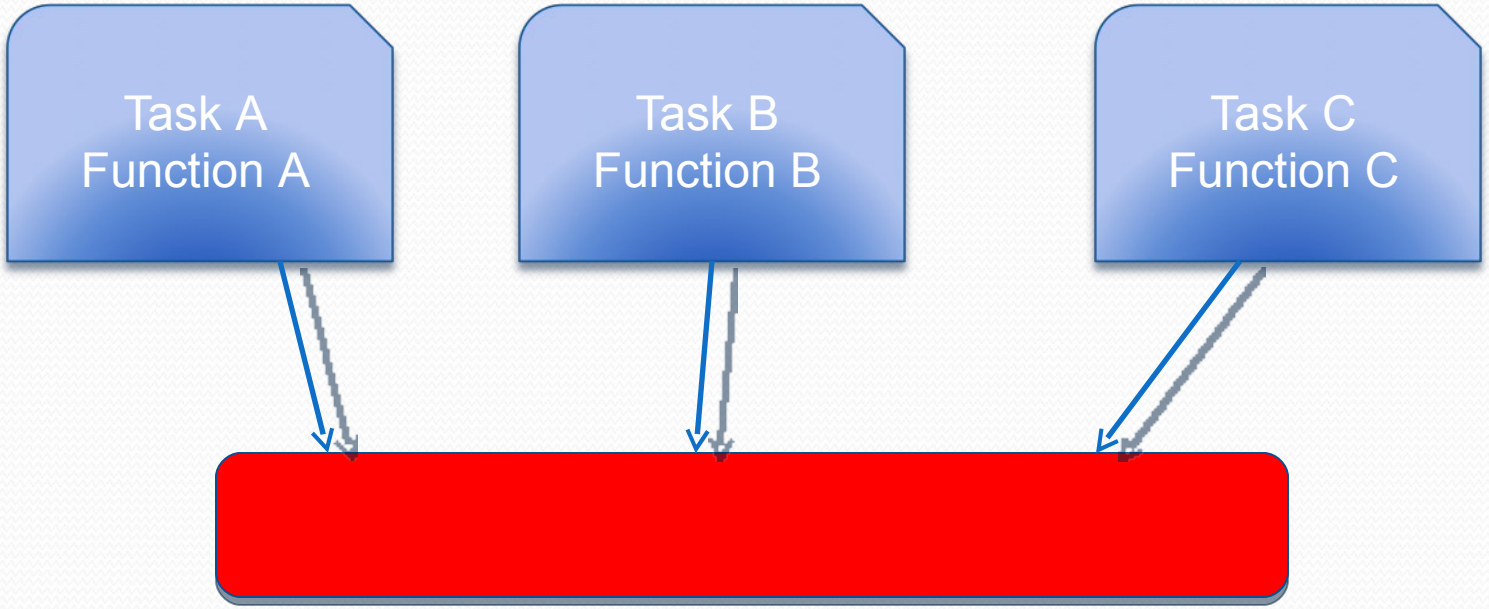
  Independent tasks apply the same operation to different elements of a data set



Types of Parallelism (cont'd)

- Functional Parallelism

  Independent tasks applying different operations to the same data elements



How to Program Parallel Computers

- There are 4 ways to program a parallel computer
  - Extend a compiler
  - Extend a sequential programming language
  - Add a parallel programming layer
  - Create a parallel language

Parallel Architectures

- There are several architectures used to do parallel work
  - Interconnection Networks
  - Processor Arrays
  - Multiprocessors
    - Centralized: unified memory access (UMA) or symmetric multiprocessor (SMP)
    - Distributed: interconnected or nonuniform memory access (NUMA)
  - Multicomputers: asymmetrical or symmetrical

Flynn's Taxonomy

- SISD: Single Instruction Single Data
  - Regular uniprocessor
- SIMD: Single Instruction Multiple Data
  - Processor arrays, pipelined vector processors
- MISD: Multiple Instruction Single Data
  - Systolic arrays
- MIMD: Multiple Instruction Multiple Data
  - Multiprocessors, multicomputers

Parallel Algorithm Design

One methodology is the task/channel model

- Task – a program, its local memory and a collection of I/O ports
  - Local memory contains local data values
  - I/O ports used to send data values among tasks
- Channel – a message queue that connects one task's output port to another task's input port

Foster's Design Methodology

- Partitioning: The first step is the process of dividing the computation and data into pieces
  - Domain decomposition – divide data into pieces and figure out how best to compute
  - Functional decomposition – divide the computation into pieces and figure out how to associate data to it
- Communication: The second step is to determine the communication pattern between the tasks (local or global)

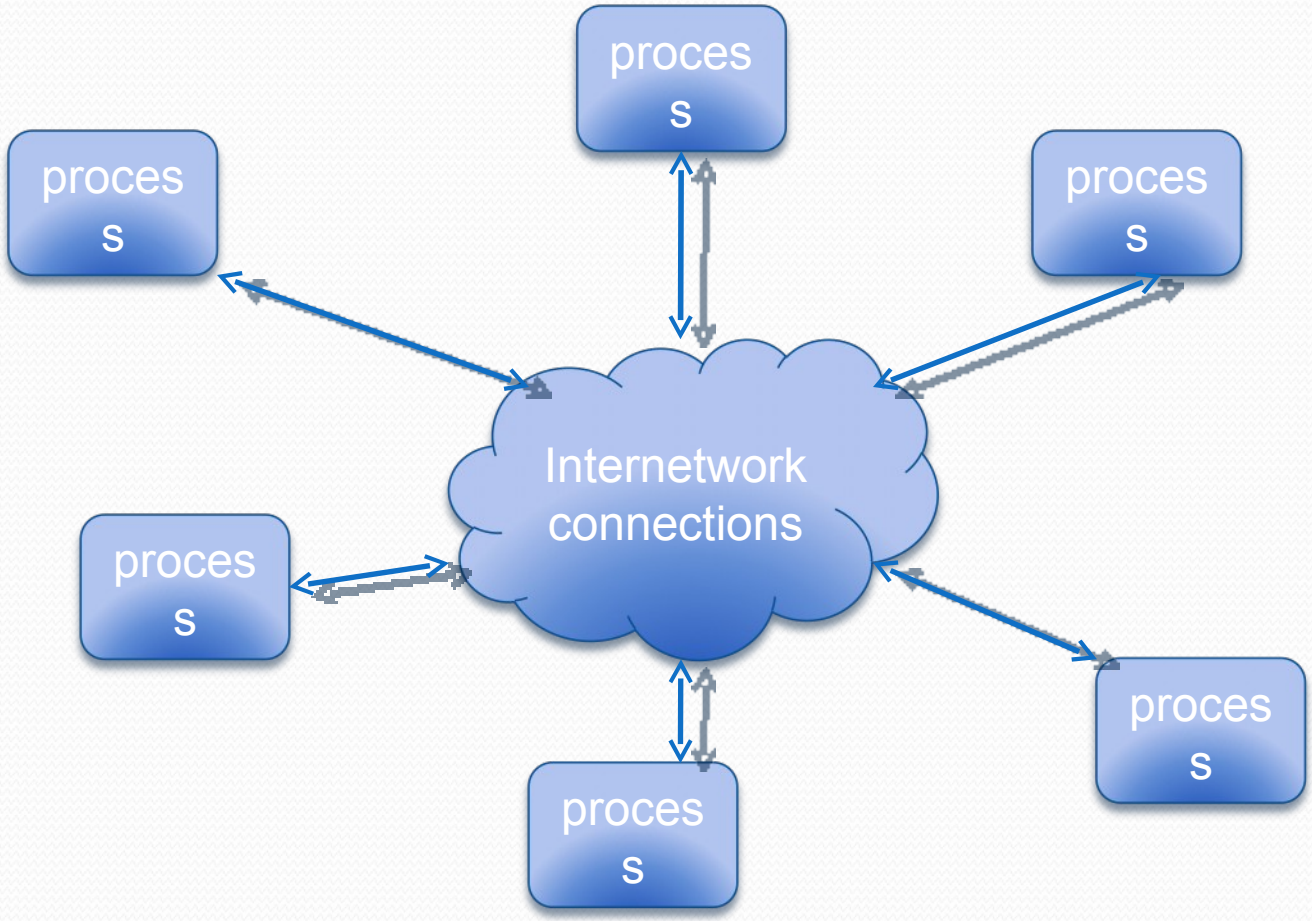
Foster's Design Methodology (cont'd)

- Agglomeration: The third step is the grouping of tasks into larger tasks to improve performance or simplify programming
- Mapping: The fourth and last step is to map these tasks to processors

Message-Passing Programming

- Similar to the task/channel model
  - Task = Process
  - Channel = Interconnect Network
- Every process can communicate with every other process
- Every process runs the same program, but is identified by a unique ID

MPI


process
process
process
Internetwork connections
process
process
process

History of MPI

- Late 1980's – vendors were creating their own versions
- 1989 – first version of PVM (Parallel Virtual Machine)
- 1992 – first draft of MPI
- 1994 – Version 1.0 of the MPI standard
- 1997 – MPI-2 standard adopted

- Implementations include MPICH, openMPI, LAM

Performance Analysis

- Performance analysis helps us decide whether parallel is worth the work, and whether more processors will help
- Speedup – ratio of sequential execution time to parallel execution time
- More processors reduces execution time at the cost of higher communication
- This means speedup usually increases, then decreases

Performance Analysis (cont'd)

- Efficiency – measure of processor utilization

$$\frac{sequential\_execution\_time}{processors\_used \times parallel\_execution\_time}$$

Amdahl's Law (maximum speedup)

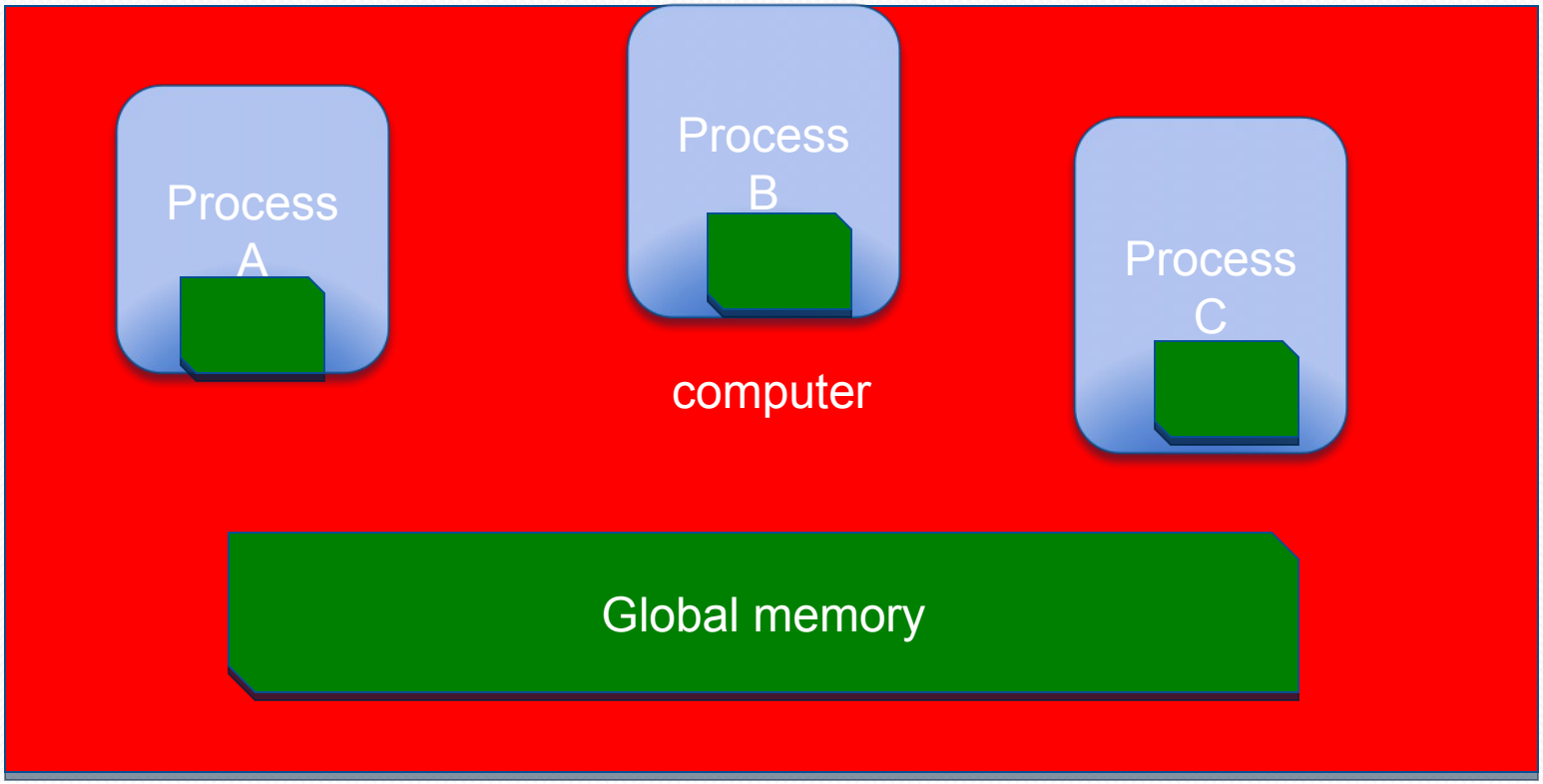
Let f be the fraction of operations in a computation that must be performed sequentially, where $0 \leq f \leq 1$. The maximum speedup ψ achievable by a parallel computer with p processors performing the computation is

$$\psi \leq \frac{1}{f + \frac{1 - f}{p}}$$

Shared-memory Programming

- Abstraction of the generic centralized multiprocessor
- Underlying hardware is a collection of processors having access to the same shared memory
- Processors can interact and synchronize through shared variables

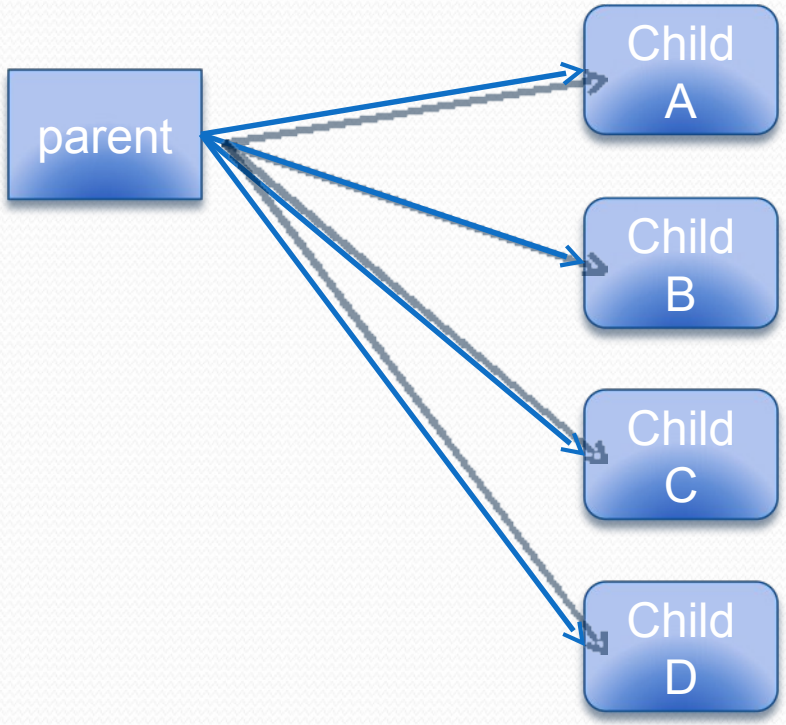
Shared-memory Programs



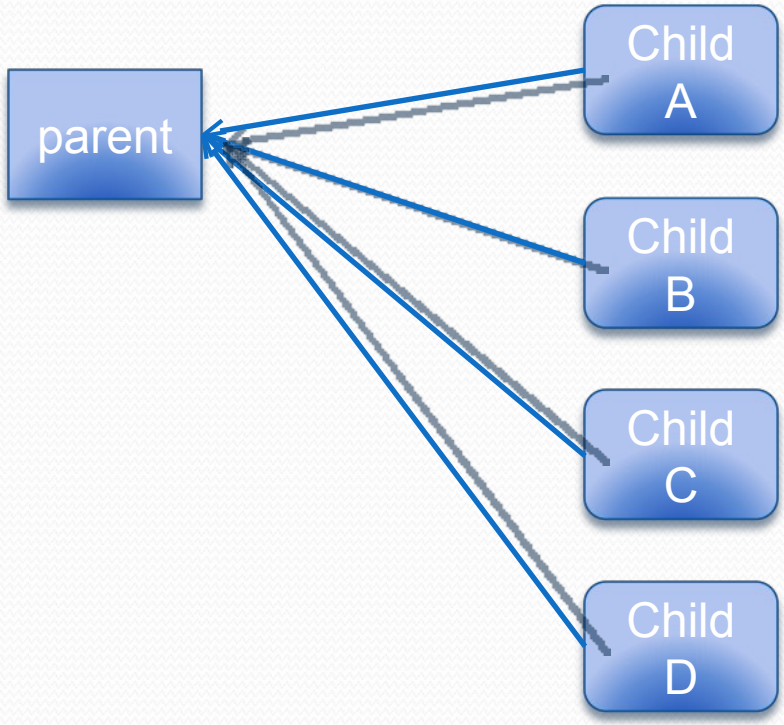
Fork/Join Parallelism

- Program starts with a single thread of execution
- Multiple threads of execution **fork** off to do their work
- They **join** at the end of the calculation
- A sequential program is a parallel program with no forks or joins
- This allows for incremental parallelization

Forking



Joining



Implementations

- OpenMP – a library for C/C++ and Fortran to provide pragmas to ease shared-memory programming
  - Most common form is for loop parallelization
  - Can do general code block parallelization
- Pthreads – a library for setting up multiple threads of execution explicitly
  - Explicitly setup and launch threads
  - Can support separate code for separate threads

Combinations

- Message-passing and shared-memory models can be combined
- Allows for most flexible optimization of tasks and data decomposition
- Much more complex coding and debugging involved