Parallel Programming with MPI

- motivation and terminology
- kinds of parallel processing
- how much speedup to expect
- the Message Passing Interface Library
- example: dot product
- exercise: matrix-vector product
- sources of further information

conducted by
Mike Kupferschmid
Scientific Programming Consultant
VCC 322, x6558, kupfem@rpi.edu
Motivation and Terminology

“My program runs too long!”

improve the algorithm
improve the program
use a faster computer
divide the work between 2 or more computers
domain decomposition (SIMD)
   Same Instructions, Multiple Data
   each does same calculation on different data
functional decomposition (MIMD)
   Multiple Instructions, Multiple Data
   processors do calculations that might differ
SIMD is a subset of MIMD

dividing the work
   might also distribute the memory
Kinds of Parallel Processing

vectorization
  very fine-grained SIMD
  arithmetic on multiple vector elements at once
easiest to use but usually very expensive
used in supercomputers made by Cray, NEC

message passing
  coarse-grained MIMD
  = SIMD if processes happen to be the same
  cluster of computers connected by network

high-performance FORTRAN
  data-parallel, special case of SIMD
  compiler automates message passing
The Message Passing Interface Library

MPI

eys several different versions are available
we use public-domain MPIch
can be used with FORTRAN or C
subroutine calls do inter-processor communication
same program runs on each processor
   SPMD, Same Program, Multiple Data
each copy finds out processor it is running on
and does appropriate part of the calculation
only a small subset of MPI is illustrated here
for more information see References
How Much Speedup to Expect

for most problems speedup is in range $\times 1$ to $\times 10$
provided enough processors are used
higher for “embarrassingly parallel” problems
can be less than 1 (slow-down)

Amdahl’s Law:

$$S_n = \frac{1}{(1-p)+r+p/n}$$

$S_n$ is speedup using $n$ processors

$p$ is fraction parallelizable
want $p$ close to 1

$r$ is communicate/compute ratio
want $r$ close to 0
Speedup vs Communicate/Compute Ratio
for parallel fraction $p = 0.7$
Example: Dot Product

\[ x^\top y = \sum_{j=1}^{100} x_j y_j = \sum_{j=1}^{50} x_j y_j + \sum_{j=51}^{100} x_j y_j \]

\[ x^\top y \quad \text{processor 1} \quad + \quad \text{processor 0} \]
References


