Seismic Code

- **Given echo data, compute under sea map**
- **Computation model**
  - designed for a collection of workstations
  - uses variation of RPC model
  - workers are given an independent trace to compute
    - requires little communication
    - supports load balancing (1,000 traces is typical)
- **Performance**
  - max mfops = $O((F \cdot nz \cdot B^*)^{1/2})$
  - $F$ - single processor MFLOPS
  - $nz$ - linear dimension of input array
  - $B^*$ - effective communication bandwidth
    - $B^* = B/(1 + BL/w) \approx B/7$ for Ethernet (10msec lat., $w=1400$)
  - real limit to performance was latency **not** bandwidth
Database Applications

- Too much data to fit in memory (or sometimes disk)
  - data mining applications
  - imaging applications
    - use a fork lift to load tapes by the pallet

- Sources of parallelism
  - within a large transaction
  - among multiple transactions

- Join operation
  - form a single table from two tables based on a common field
  - try to split join attribute in disjoint buckets
    - if know data distribution is uniform its easy
    - if not, try hashing
Parallel Search (TSP)

- **may appear to be faster than 1/n**
  - but this is not really the case either

- **Algorithm**
  - compute a path on a processor
    - if our path is shorter than the shortest one, send it to the others.
    - stop searching a path when it is longer than the shortest.
  - before computing next path, check for word of a new min path
  - stop when all paths have been explored.

- **Why it appears to be faster than 1/n speedup**
  - we found the a path that was shorter sooner
  - however, the reason for this is a different search order!
Ensuring a fair speedup

- $T_{\text{serial}} = \text{faster of}$
  - best known serial algorithm
  - simulation of parallel computation
    - use parallel algorithm
    - run all processes on one processor
  - parallel algorithm run on one processor

- If it appears to be super-linear
  - check for memory hierarchy
    - increased cache or real memory may be reason
  - verify order operations is the same in parallel and serial cases
Quantitative Speedup

- Consider master-worker
  - one master and n worker processes
  - communication time increases as a linear function of n
  \[ T_p = T\text{COMP}_p + T\text{COMM}_p \]
  \[ T\text{COMP}_p = \frac{T_s}{P} \]
  \[ \frac{1}{S_p} = \frac{T_p}{T_s} = \frac{1}{P} + \frac{T\text{COMM}_p}{T_s} \]
  \[ T\text{COMM}_p \text{ is } P \times T\text{COMM}_1 \]
  \[ \frac{1}{S_p} = \frac{1}{p} + p \times \frac{T\text{COMM}_1}{T_s} = \frac{1}{P} + \frac{P}{r_1} \]
  where \( r_1 = \frac{T_s}{T\text{COMM}_1} \)
  \[ \frac{d(1/S_p)}{dP} = 0 \rightarrow P_{opt} = r_1^{1/2} \text{ and } S_{opt} = 0.5 \ r_1^{1/2} \]

- For hierarchy of masters
  - \( T\text{COMM}_p = (1+\log P)T\text{COMM}_1 \)
  - \( P_{opt} = r_1 \text{ and } S_{opt} = r_1/(1 + \log r_1) \)
MPI

- **Goals:**
  - Standardize previous message passing:
    - PVM, P4, NX
  - Support copy free message passing
  - Portable to many platforms

- **Features:**
  - point-to-point messaging
  - group communications
  - profiling interface: every function has a name shifted version

- **Buffering**
  - no guarantee that there are buffers
  - possible that send will block until receive is called

- **Delivery Order**
  - two sends from same process to same dest. will arrive in order
  - no guarantee of fairness between processes on recv.
MPI Communicators

- Provide a named set of processes for communication
- All processes within a communicator can be named
  - numbered from 0…n-1
- Allows libraries to be constructed
  - application creates communicators
  - library uses it
  - prevents problems with posting wildcard receives
    - adds a communicator scope to each receive
- All programs start will MPI_COMM_WORLD
Non-Blocking Functions

- **Two Parts**
  - post the operation
  - wait for results

- **Also includes a poll option**
  - checks if the operation has finished

- **Semantics**
  - must not alter buffer while operation is pending
MPI Misc.

- **MPI Types**
  - All messages are typed
    - base types are pre-defined:
      - int, double, real, {,unsigned}{short, char, long}
    - can construct user defined types
      - includes non-contiguous data types

- **Processor Topologies**
  - Allows construction of Cartesian & arbitrary graphs
  - May allow some systems to run faster

- **What’s not in MPI-1**
  - process creation
  - I/O
  - one sided communication
MPI Housekeeping Calls

- Include <mpi.h> in your program
- If using mpich, ...

- First call MPI_Init(&argc, &argv)
- MPI_Comm_rank(MPI_COMM_WORLD, &myrank)
  - Myrank is set to id of this process
- MPI_Wtime
  - Returns wall time
- At the end, call MPI_Finalize()
MPI Communication Calls

- **Parameters**
  - var – a variable
  - num – number of elements in the variable to use
  - type {MPI_INT, MPI_REAL, MPI_BYTE}
  - root – rank of processor at root of collective operation
  - dest – rank of destination processor
  - status - variable of type MPI_Status;

- **Calls (all return a code – check for MPI_Success)**
  - MPI_Send(var, num, type, dest, tag, MPI_COMM_WORLD)
  - MPI_Recv(var, num, type, dest, MPI_ANY_TAG, MPI_COMM_WORLD, &status)
  - MPI_Bcast(var, num, type, root, MPI_COMM_WORLD)
  - MPI_Barrier(MPI_COMM_WORLD)