Outline

- Parallel Computers & Computing
- Designing Parallel Algorithms
- **Understanding Performance**
- Parallel Programming Tools
- Application Examples
- Summary and Conclusions

Performance Metrics

- What is “performance”?
  - Execution time, efficiency, throughput, scalability
  - Development cost, maintenance cost
  - Etc.
- Choice of metric depends on application
- We focus on execution time & efficiency

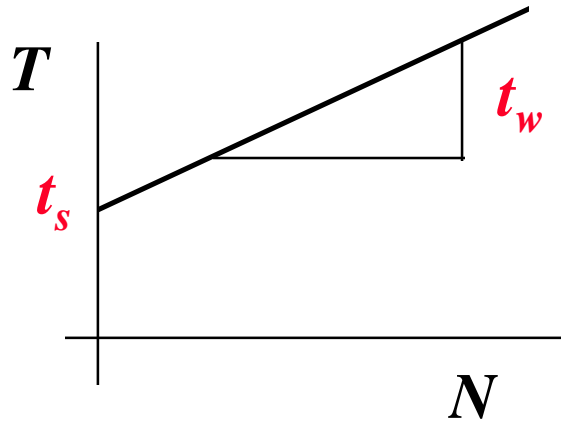
Modeling Performance

- Execution time (sums are over P nodes) is

  $T = (\text{sum}[T_{comp}] + \text{sum}[T_{comm}] + \text{sum}[T_{idle}])/P$

- Computation time comprises both
  - Operations required by sequential algorithm
  - Additional work, replicated work
- Idle time due to
  - Load imbalance, and/or
  - Latency (waiting for remote data)

Bandwidth and Latency

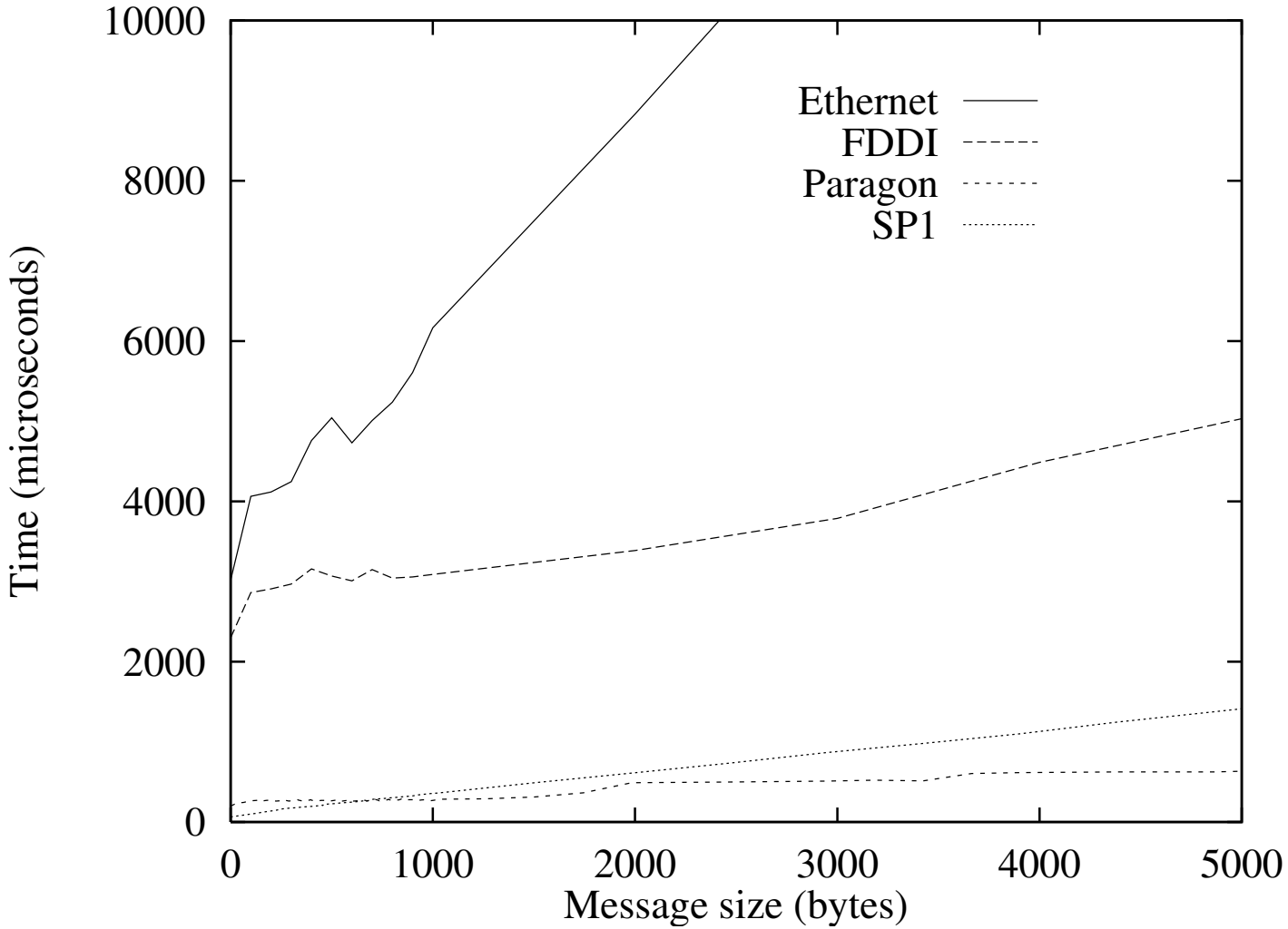


- Recall cost model

$$T = t_s + N\,t_w$$

  - t_s = per-message cost ("latency")
  - t_w = per-word cost ($1/t_w$ = bandwidth)
  - N = message size in words
- Model works well for many algorithms, and on many computers

Measured Costs

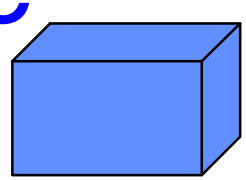


Typical Communication Costs

Computer	t_s	t_w
IBM SP2	40	0.11
Intel Paragon	121	0.07
Meiko CS-2	87	0.08
Sparc/Ethernet	1500	5.0
Sparc/FDDI	1150	1.1

Times in microseconds

Example: Finite Difference



- Finite difference computation on N^2Z grid

$$T_{comp} = t_c N^2 Z \quad (t_c \text{ is compute time/point})$$

- 1-D decomposition, so each node sends $2NZ$ data to 2 neighbors [if ≥ 2 rows/node]

$$T_{comm} = P(t_s 2 \ + t_w 4NZ)$$

- No significant idle time if load balanced, so

$$T = t_c N^2 Z/P + t_s 2 \ + t_w 4NZ$$

Speedup and Efficiency

- Efficiency: a measure of algorithm quality

$$E = T_{seq} / (P\ T_p)$$

  - T_{seq} = Time for best sequential algorithm
  - T_p = Time on P processors
- Speedup: closely related quantity

$$S = P\ E = T_{seq} / T_p$$

- Can also define relative efficiency and speedup, where T_1 replaces T_{seq}

Example: Finite Difference

- In this case, $T_1 = T_{seq}$, and efficiency is

$$E = t_c N^2 Z / [t_c N^2 Z + t_s 2P + t_w 4NZP]$$

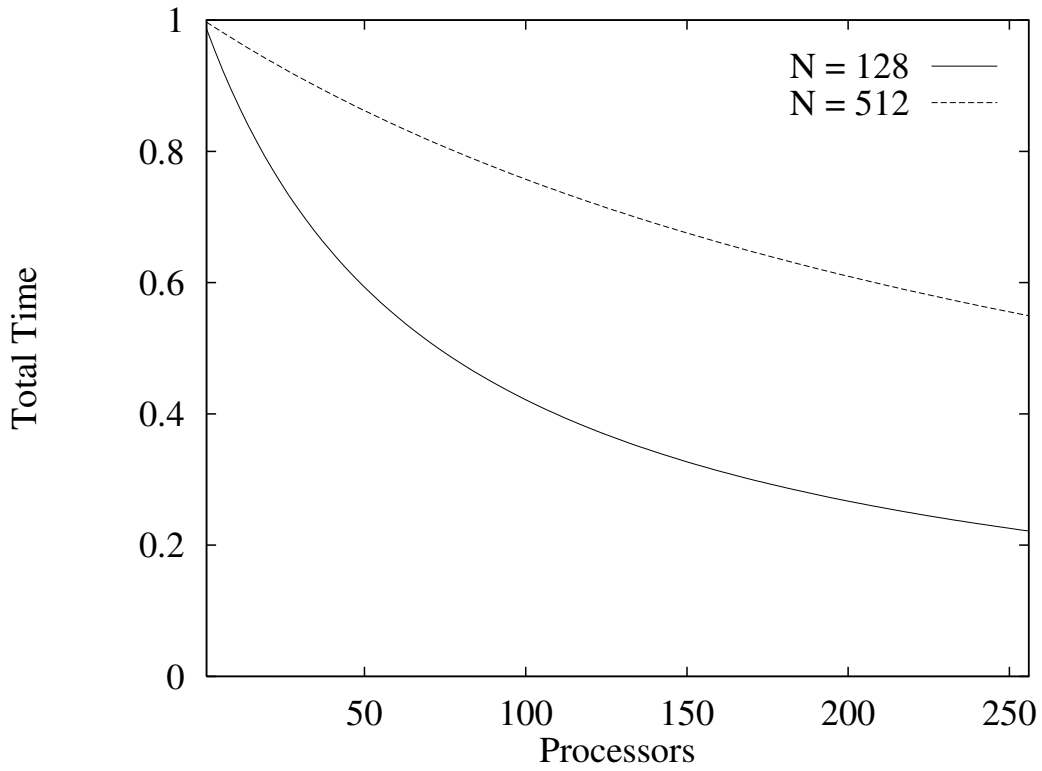
Using Performance Models

- During design
  - Use models to study qualitative behavior
  - Calibrate models by measuring t_c, t_s, t_w, etc.
  - Use calibrated models to estimate performance
- During implementation
  - Compare predictions with observations
  - Relate discrepancies to implementation or model
  - Use models to guide optimization process

Qualitative Analysis: Finite Difference

- Efficiency decreases with increasing P, t_s, t_w
- Efficience increases with increasing N, Z, t_c
- Time decreases with increasing P
- Time bounded from below by comm costs
- Time increases with increasing N, C, t_c, t_s, t_w

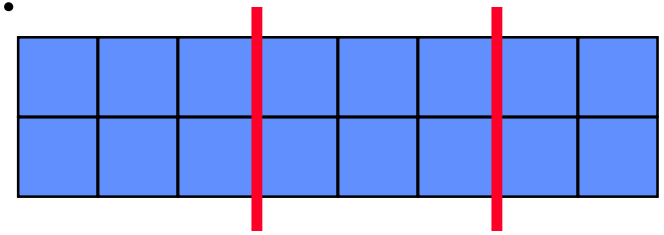
Scalability Predictions: Finite Difference



Efficiency with t_c =2 usec, t_s=100 usec, t_w =0.4 usec, Z=10 (parameters determined in calibration studies)

Model Refinements for Load Imbalance

- P does not always divide N !



$N=8,\ P=3$

- This causes idle time

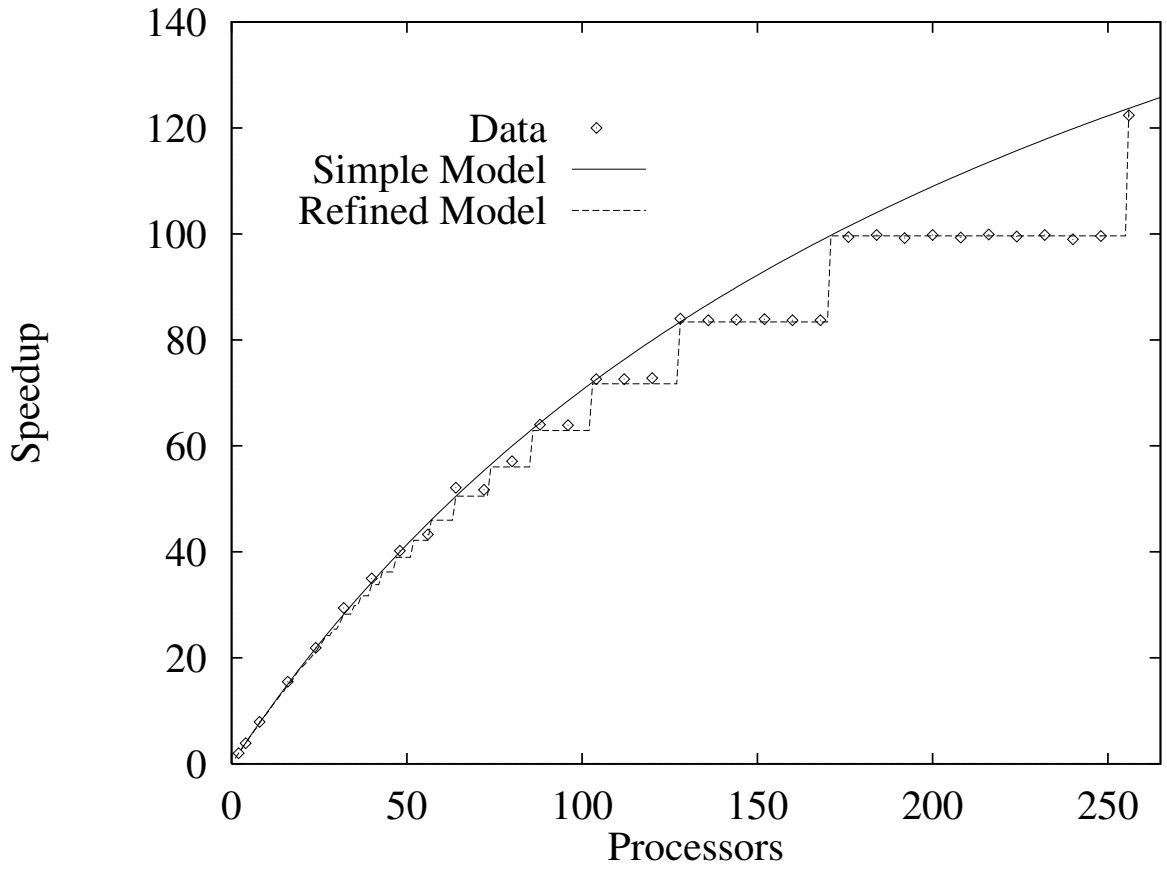
$$T_{max\ comp} = \text{top}(N/P)N\ Z$$

$$T_{idle} = P\ T_{max\ comp} - T_{comp}$$

- Model is easily adjusted to allow for this

$$T = t_c \text{top}(N/P)NZ + t_s 2 \ + t_w 4NZ$$

Refined Model vs. Observations

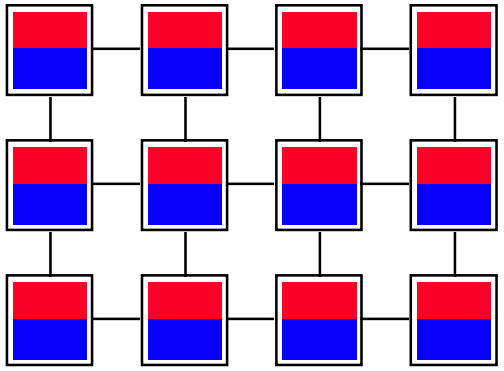


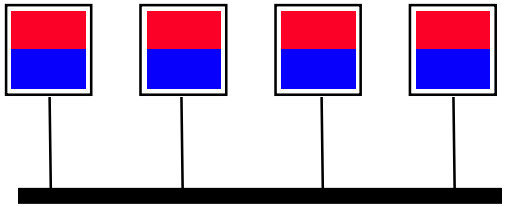
Corrected model matchs observations very closely

Impact of Network Topology

- Multicomputer model assumes comm cost independent of location & other comms
- Real networks are not fully connected

2-D Mesh





Ethernet

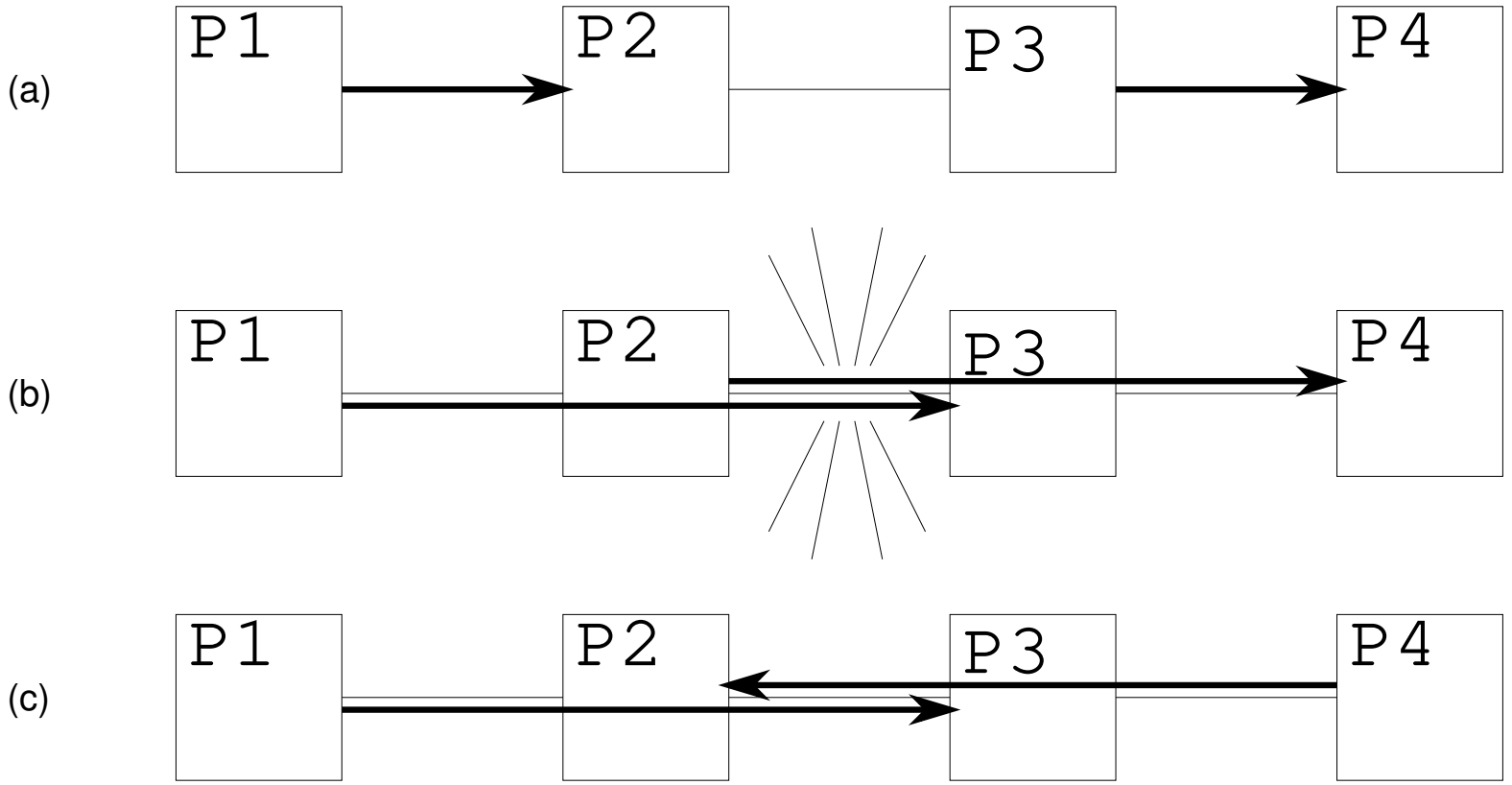
- Multicomputer model can break down

Competition for Bandwidth

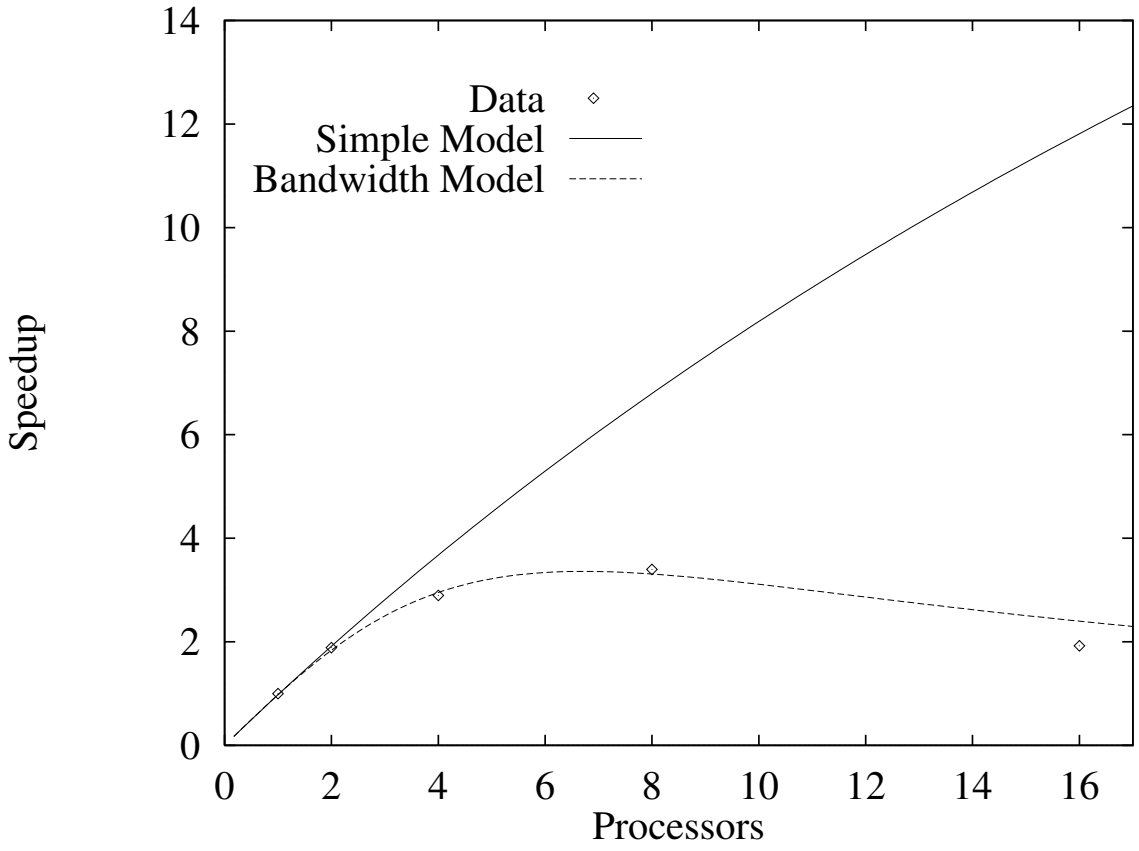
- In many cases, a *bandwidth constrained* model can give sufficiently accurate results
  - "If *S* processes must communicate over same wire at same time, each has *1/S* bandwidth"
- Example: finite difference on Ethernet
  - All processors share single Ethernet
  - Hence bandwidth term scaled by *P*

$$T = t_c N^2 Z/P + t_s 2 + t_w 4NZP$$

Competition in 1-D Mesh



Bandwidth-Constrained Model vs. Observations



Bandwidth-constrained model gives better fit

Performance Models: Summary

- An essential program development tool
- Idealizations of program performance
  - Execution time, efficiency, scalability ...
  - ... as function of N, P, t_s, t_w, etc.
- Useful throughout development process
  - Evaluate, select, and refine designs
  - Identify implementation and model deficiencies

Outline

- Parallel Computers & Computing
- Designing Parallel Algorithms
- Understanding Performance
- **Parallel Programming Tools**
- Application Examples
- Summary and Conclusions

Tool Survey

- High Performance Fortran (HPF)
- Fortran M (FM)
- Compositional C++ (CC++)
- Message Passing Interface (MPI)
- Performance tools

Fortran M

- A small set of extensions to Fortran 77 supporting task/channel model
- Extensions are modeled on Fortran concepts
  - Process = subroutine
  - Communication = file I/O
  - Resource management = processor array
- Compiler manages tasks & communication

FM Example

```
program fm_finite_difference
INPORT  (real x(100)) Li(4), Ri(4)
OUTPORT (real x(100)) Lo(4), Ro(4)
do i = 1,4
   CHANNEL(IN=Li(i), OUT=Ro(mod(i,4)+1))
   CHANNEL(IN=Ri(i), OUT=Lo(mod(i,4)+1))
enddo
PROCESSDO i = 1,4
   PROCESSCALL subdomain(Li(i), Lo(i), Ri(i), Ro(i))
ENDPROCESSDO
end
```

```
subroutine subdomain(Li, Lo, Ri, Ro)
INPORT  (real x(100)) Li, Ri
OUTPORT (real x(100)) Lo, Ro
real X(27, 100), New(25, NP)
X    = exchange(X, Li, Lo, Ri, Ro)
New  = update(X)
diff = maxdiff(New, X, Li, Lo, Ri, Ro)
end
```

FM Analysis

- Advantages
  - Particularly easy to learn and use
  - Support for modular programming
- Disadvantages
  - Relatively low-level
  - Limited support for data-parallel programming
- Good for multi-component programs

Compositional C++

- Small set of extensions to C++
  - Type modifiers, objects, blocks and pointers
  - Communications by extension of C type system
  - Full use of C pointers
- Control of concurrency and locality
- Infrastructure for parallel class libraries

CC++ Example

```
void node_execute() {
   finite_difference(localgrid, localmax);
   global_maximum(localmax, globalmax);
}

void main(int argc, char *argv[]) {
   int P = atoi(argv[1]);
   // Create array of processor objects
   initialize(P);
   // Create concurrent threads
   parfor (int i=0; i<P; i++)
      nodes[i]->node_execute(i);
}
```

CC++ Analysis

- Advantages
  - Support for libraries and reusable code
  - Flexible programming model
- Disadvantages
  - Complex programming model
  - Limited support for data-parallel programming
- Good for development of parallel libraries

Message Passing Interface (MPI)

- A standard message-passing library
  - p4, NX, p4, Express, PARMACS are precursors
- An MPI program defines a set of processes

(usually one process per node)

... that communicate by calling MPI functions

(point-to-point and collective)

... and can be constructed in a modular fashion

(communicators are the key)

MPI Example

```
main(int argc, char *argv[]) {
    MPI_Comm com = MPI_COMM_WORLD;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(com,&np);

    MPI_Send(local+1,1,MPI_FLOAT,lnbr,10,com);
    MPI_Recv(local,1,MPI_FLOAT,rnbr,10,com,&status);
    MPI_Send(local+lsize,1,MPI_FLOAT,rnbr,10,com);
    MPI_Recv(local+lsize+1,1,MPI_FLOAT,lnbr,10,com,&status);

    ldiff = maxerror(local);
    MPI_Allreduce(&ldiff,&diff,1,MPI_FLOAT,MPI_MAX,com);
}
```

MPI Analysis

- Advantages:
  - Wide availability of efficient implementations
  - Support for modular design, code reuse
- Disadvantages:
  - Low level ("parallel assembly code")
  - Less well-suited to shared-memory machines
- Good for performance-critical codes

High Performance Fortran

- A standard data-parallel language
  - CM Fortran, C*, HPC++ are related
- Programmer specifies
  - Concurrency (concurrent operations on arrays)
  - Locality (data distribution)
- Compiler infers
  - Mapping of computation ("owner-computes" rule)
  - Communication

HPF Example

```
      PROGRAM hpf_finite_difference
!HPF$ PROCESSORS pr(4)
      REAL x(100,100), new(100,100)
!HPF$ ALIGN  new(:,:)  WITH x(:,:)
!HPF$ DISTRIBUTE x(BLOCK, *) ONTO pr
      new(2:99,2:99) = (x(1:98,2:99)+x(3:100,2:99)+
     &                  x(2:99,1:98)+x(2:99,3:100)) / 4
      diff = MAXVAL(ABS(new-x))
      end
```

HPF Analysis

- Advantages
  - High level: preserves sequential semantics
  - Standard
- Disadvantages
  - Restricted applicability
  - Requires sophisticated compiler technology
- Good for regular, SPMD problems

Performance Tools

- Performance tools are used to obtain
  - Data used to parameterize performance models
  - Data used to compare programs against models
- Approachs
  - Interval timers: measure time in regions
  - Traces: (semi)automatic logging of events
  - Profiles: automatic collection of profile data
- Many reasonable tools; no standards

Tool Summary

- Choice of tool depends on application
  - Fortran, regular => HPF or MPI
  - Fortran, irregular => FM
  - C/C++, regular => MPI
  - C/C++, irregular => CC++
- Performance tools are an important link to the performance evaluation process

Notes