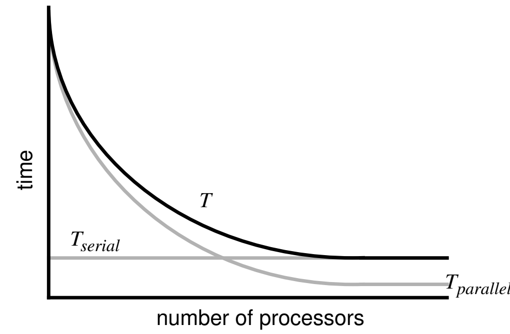
Thinking Parallel

Timothy J. Williams

NATIONAL ENERGY RESEARCH SUPERCOMPUTER CENTER

Amdahl's Law

- Any parallel program has a parallel part and a serial part.
- Total compute time is $T = T_{parallel} + T_{serial}$
- No matter how many processors you use, $T \geq T_{serial}$



- If the serial fraction is α, the maximum possible speedup is,

$$\lim_{P \to \infty} \frac{T_1}{T_P} = \frac{1}{\alpha}$$

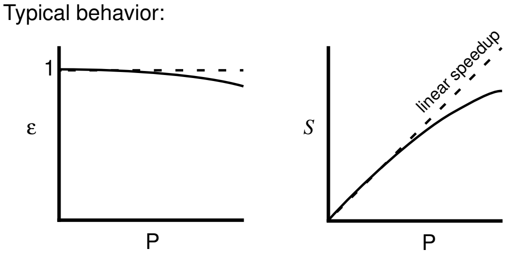
where P is the number of processors.

Amdahl's Law (cont'd)

- N.B. (Sandia): Amdahl's fraction, α, may not be constant wrt problem size N.
  - If α decreases as N increases, max. speedup with P processors is greater for larger N.

Parallel Efficiency

- $\varepsilon = \frac{1}{P}\frac{T_1}{T_P}$
  - P = number of processors
  - T_1 = time for *optimal serial algorithm* on one processor
  - T_P = time for *parallel algorithm* on P processors
- Using T_1 = time for parallel algorithm on 1 processor is cheating!
- Measures *scalability* of parallel algorithm: higher is better.
- Higher $\varepsilon \Rightarrow$ use more procs. effectively on fixed problem size.
  - Speedup: $S = P\varepsilon$
  - Typical behavior:



What *Is* Your Problem?

What are the answers you seek?

- Single number?
  - Data reduction ⇒
    - processor *syncronization* (wait for all PE's)
    - interprocessor communication
- Many single numbers?
  - Truly embarassingly parallel: Independent parameter-sweep runs
    - Entire problem fits on one node
      - Generalization: use only a few nodes per problem, to get high parallel efficiency.
    - If the parameter sweep is essential to the physics, than running all the cases is really running a single problem. Just think of it as a single program.
- Profiles in subset of dimensions?
  - Data reduction (see above)

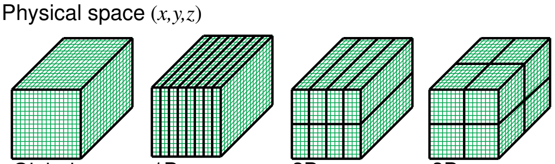
What *Is* Your Problem? (cont'd)

- Pictures?
  - Runtime graphics ⇒data reduction
    - processor syncronization
    - interprocessor communication
  - Postprocessing for graphics
    - Parallel I/O (at least O)
      - this is a research area
    - Access pieces small enough to handle on postprocessing machine
      - For MPP-sized problems, this is important.
        - 256-node Cray T3D: 16GB
        - SGI Onyx, largish: 500MB
    - Canned graphics packages will not work on distributed data.

In general, anything non-local in the space of computation requires special handling.

Domain Decomposition

- Useful concept in parallel scientific computing, even non-message-passing.
- Divide a spatial domain into smaller pieces (*subdomains*)
  - Physical space (x,y,z)



  - Fourier space (k_x, k_y, k_z) , phase space $(\boldsymbol{x}, \boldsymbol{v})$, parameter space
- Use maximum practical dimensionality of DD
  - Max #procs on fixed size problem (minimum subd.size = 1)
- Not older numerical analysis meaning of domain decomp.:
  - Solving *entire* problem, not independent sub-problems.
- Global/local view of data
  - Data parallel: local view implicit
  - Message passing: local view explicit
  - OO: Local view implicit at scientific programmer level,

Analyzing for Parallelism

When you look at an operation/(sub-)algorithm, ask these questions:

- Where are the data?
- Which parts of the calculation are parallel?

Analyzing for Parallelism (cont'd)

- Where are the data?
  - How to maximize computation/communication ratio?
    - computation rates (Cray T3D):
      - From peak 150 MFLOPS $\Rightarrow$ 1 FLOP=7×10^{-9} sec
      - From typical 20 MFLOPS $\Rightarrow$ 1 FLOP=5×10^{-8} sec
    - communication rates (Cray T3D)
      - Peak comm. latency (`shmem_put`) $2\mu\mathrm{sec}$ $\Rightarrow$ 1 Xfer = 2×10^{-6} sec
      - Peak comm. B/W (`shmem_put`) 120MB/s $\Rightarrow$ 1 Word =1×10^{-9} sec
    - Choice of domain decomposition important
      - Surface-to-volume ratio of subdomains

Analyzing for Parallelism (cont'd)

- Which parts of the calculation are parallel?
  - "Data dependencies"
    (Like data dep. in loop breaking vectorization on C90.)
    - Iterative method: all PE's must wait until global test parameter computed (requires communication)
    - Timestepping: each PE must wait until all data it needs for next time step are ready.

Examples

- Finite difference
- Boundary conditions
- Scatter-add
- Global average
- Unstructured mesh storage
- Tridiagonal matrix problem solution
- FFT
- Matrix-matrix multiply $AB = C$

Finite difference

Example stencil for 2^{nd} derivative:

$$\frac{\partial^2}{\partial x^2} f(x_0) = \frac{1}{12\Delta x^2}\left[-f(x_0+2\Delta x) + 16 f(x_0+\Delta x) - 30 f(x_0) + 16 f(x_0-\Delta x) - f(x_0-2\Delta x) + O(\Delta x^4)\right]$$

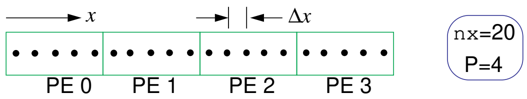
- Where are the data?
  - Data parallel: specified in layout directives (or default)
    - HPF code:

```
real f(1:nx),fdprime(1:nx)
!HPF DISTRUBITE f(BLOCK)
!HPF DISTRUBITE fdprime(BLOCK)
fdprime(3:nx-2) = c0*(-f(5:nx) +
                      16*f(4:nx-1) -
                      30*f(3:nx-2) +
                      16*f(2:nx-3) -
                         f(1:nx-4))
```

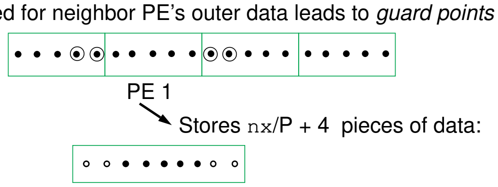
    - `f(i-2)`, `f(i-1)`, `f(i+1)`, and/or `f(i+2)` may be on different PE(s) than `f(i)` (and `fdprime(i)`).
    - Worst choice: all 5 pieces of data on different PEs
      - One comm. for each term (for each grid point)
    - Comm. may be specially optimized (`CSHIFT`)

Finite difference (cont'd)

- Where are the data? (cont'd)
  - Message passing with domain decomposition



    - $x_0 \pm \Delta x$, $x_0 \pm 2\Delta x$ may be on different PE(s) than x_0
    - Need for neighbor PE's outer data leads to *guard points*



    - Communicate all guard layers in one message, then use local values for all F.D. calculations.

```
parameter (nx=20,p=4,ng=2)
real f(1:nx/p+2*ng)
real fdprime(1:nx/p+2*ng)
do i=(ng+1),(nx/p+2*ng),1
 fdprime(i)= c0*(-f(i+2)+ 16*f(i+1)
                 30*f(i)   + 16*f(i-1) -
                    f(i-2))
enddo
```


Boundary conditions

- Boundary conditions (global) often cause special problems for parallel programming with distributed data.
- Periodic boundary conditions for finite difference example:
  - Data parallel
    - In absence of CSHIFT, need extra serial statements

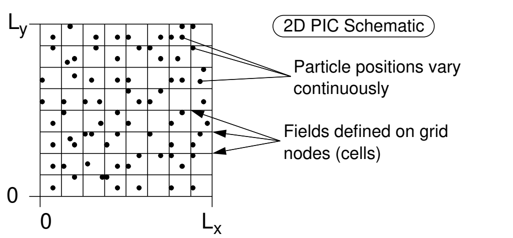
```
fdprime(2) = c0*(-f(nx-1) + 16*f(nx) -
               30*f(2) + 16*f(3) - f(4))
fdprime(1) = c0*(-f(nx-1) + 16*f(nx) -
               30*f(2) + 16*f(3) - f(4))
fdprime(nx-1) = c0*(-f(nx-3) +16*f(nx-2) -
                 30*f(nx-1) +16*f(nx) -
                    f(1))
fdprime(nx) = c0*(-f(nx-2) +16*f(nx-1) -
                30*f(nx) +16*f(1) - f(2))
```

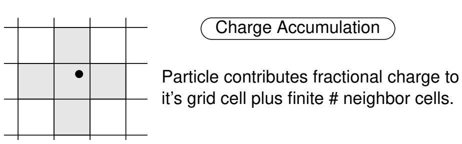
  - Message passing
    - PE's at ends of global box have guard points that wrap around periodically.
      - No extra statements needed in loop.

- Which parts of calculation are parallel?
  - Data parallel: non-boundary points.
  - Message passing: all points

Scatter-add

- From particle-in-cell simulation





- Message passing
  - Domain decomp with guard cells: *local* scatter-add
  - Pass accumulated grid data for overlap (guards)
- Data parallel
  - `FORALL`, if no special gather/scatter functions
  - Unpredictable, randomized communication between particle-owner procs and grid-owner procs

Global average

For distributed data, averaging ⇒ communication.

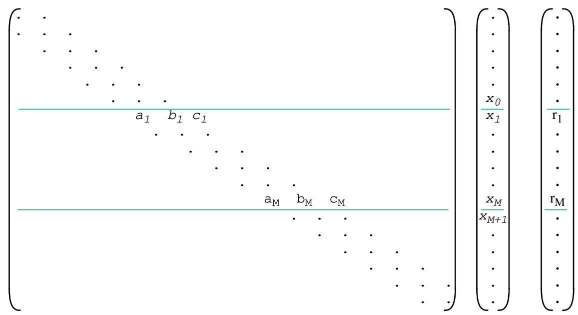
- For example, average over whole grid.
  - Which parts of the calculation are parallel?
    - Message passing:
      - Partial sums of local data only
      - Send partial sums to every other PE
      - Add up local copies of partial sums to get local copy of global sum.
    - Data parallel:
      - Similar; efficiency is implementation-dependent
        - HPF (F90) `SUM` intrinsic

Unstructured mesh storage

- Unstructured mesh data stored as:
  - ❍ Ordered list of coordinate values for each grid node
    `real x(1:n),y(1:n),z(1:n)`
  - ❍ Vector field values at each node
    `real ex(1:n),ey(1:n),ez(1:n)`
    `real ax(1:n),ay(1:n),az(1:n)`
  - ❍ Connectivity data for each node
    - ■ Integer value M_i and array of M_i indices for each node: The other nodes it connects with.
- Global view of data structure: global numbering 1,2,...,`n`
- Local view
  - ❍ Message-passing
    - ■ Explicit, special handling of connections to nodes on other processor's memory
    - ■ Special algorithms to partition of mesh among processors (spectral bisection, *e.g.*)
  - ❍ Data parallel (pure)
    - ■ Implicit communications for connections to nodes in other processor's memory

Tridiagonal matrix problem solution

- Each PE owns sub-piece of matrix problem (shown for P=3):



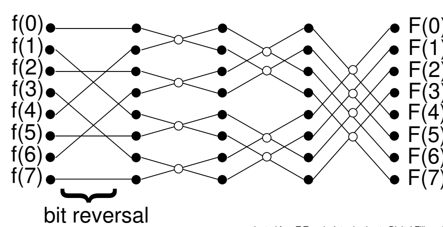
  - ❍ $\boldsymbol{x} = \boldsymbol{x}^R + c^{UH}\boldsymbol{x}^{UH} + \boldsymbol{c}^{LH}\boldsymbol{x}^{LH}$, with $\{\boldsymbol{x}^R, \boldsymbol{x}^{UH}, \boldsymbol{x}^{LH}\}$ solutions to three tridiagonal systems (per PE) (no communication).
    - ■ $\{c^{UH}, c^{LH}\}$ solutions to $2P$x$2P$ *reduced matrix* problem
      - ❑ Matrix elements from all PEs (communication)

Tridiagonal matrix problem solution (cont'd)

- Serial algorithm (LU decomposition): $O(8N)$ operations
- Parallel algorithm: $O(17N)$ operations
  - ❍ Parallel efficiency limit of 8/17=0.47
  - ❍ But, $T_P = \frac{1}{P}\frac{T_1}{\varepsilon} \sim \frac{0.47}{P}T_1$, so we do benefit from parallism.

FFT

- 1D fast Fourier transforms require interchanges of widely-separated elements in the data sequence

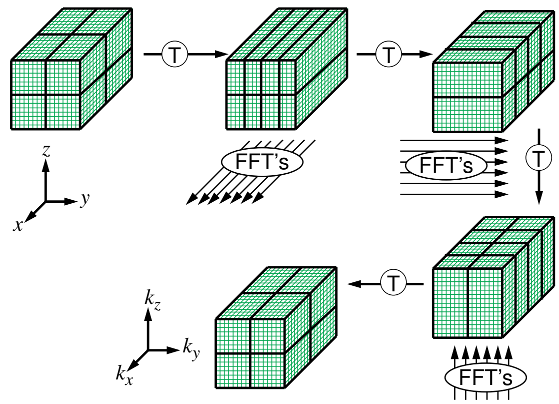


adapted fromT.Terrel, *Introduction to Digital Filters 2nd ed.*(Wiley, 1988)

  - ❍ Communication intensive for distributed data
    - ■ Data parallel: may use special fast comm. hardware with appropriate data layout
      - ❑ *E.G.*, CMSSL routines on CM-200, CM-5
    - ■ Typically avoided in message passing programs

FFT (cont'd)

- Multidimensional
  - ❍ Data parallel: same considerations as 1D
  - ❍ Message passing:
    Transposes, multiple serial FFT's on local 1D strips:

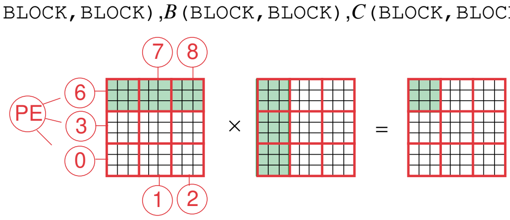


Matrix-matrix multiply

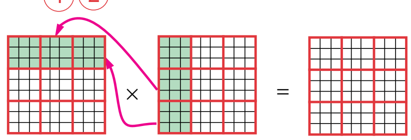
- $AB = C$
- For example, consider 9x9 matrix on 9 PEs
  - ❍ column-major
  - ❍ at most two copies of subdomains' data fits on 1 PE
- Compare combinations of domain decompositions
  - ❍ Like HPF `DISTRIBUTION's`
- For message-passing time estimates, use:
  - ❍ Message latency = $10\mu\mathrm{sec}$
  - ❍ Transfer speed for 1W (64-bit word) = 10nsec

Matrix-matrix multiply (cont'd)

❍ A `(BLOCK,BLOCK)`, B `(BLOCK,BLOCK)`, C `(BLOCK,BLOCK)`

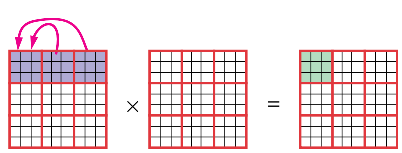


Send operands
(2 msg*9W)



------Multiply & add to get partial sum-products....------

Send partial products, then add
(2 msg*9W)

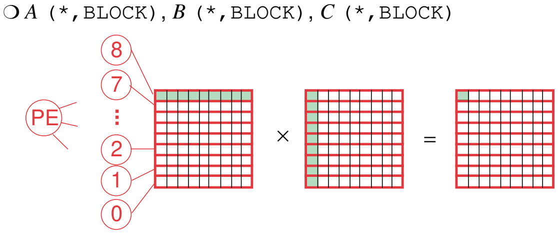


■ Total 30 messages, total 234W transferred ⇒

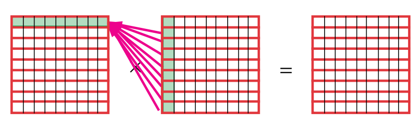
$T_{comm} = 3\times10^{-4}$ sec

Matrix-matrix multiply (cont'd)

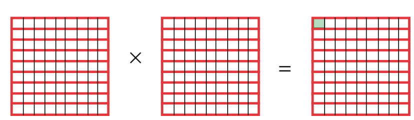
❍ A `(*,BLOCK)`, B `(*,BLOCK)`, C `(*,BLOCK)`



Send operands
(8 msg*1W)



Compute dot product



■ Total 72 messages, total 72W transferred ⇒

$T_{comm} = 7.2\times10^{-4}$ sec

Matrix-matrix multiply (cont'd)

- Exercise for the reader:
  - A `(*,BLOCK)`, B `(*,BLOCK)`, C `(BLOCK,*)`
  - A `(*,BLOCK)`, B `(BLOCK,*)`, C `(BLOCK,*)`
  - A `(BLOCK,BLOCK)`, B `(BLOCK,*)`, C `(*,BLOCK)`
  - A `(BLOCK,BLOCK)`, B `(BLOCK,*)`, C `(BLOCK,*)`
  - A `(BLOCK,BLOCK)`, B `(BLOCK,*)`, C `(BLOCK,BLOCK)`