Programming for Performance

- What limits the performance of parallel programs?
- What factors must be addressed in software?
- What factors can be addressed by the archtiecture?

What Limits Performance?

$$Speedup\ (p) \ = \ \frac{Time\ (1)}{Time\ (P)} < P$$

- Available parallelism
- Load Balance
  - some processors do more work than others
  - some work while others wait
  - remote resource contention
- Communication
- Extra work
  - management of parallelism
  - redundant computation

Parallelism in LU

Replicated Work

Serial Work

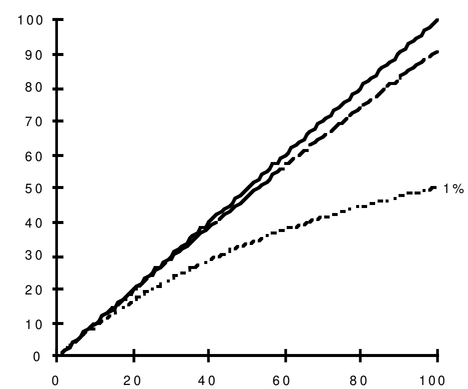
Synchronization

Parallel Portion

```
procedure LU( A[n, n] )
begin
    int i,j,k,pid;
    int rows, mymin, mymax;
    rows <– n / procs;
    mymin <– pid * rows;   mymax <– mymin + rows - 1;
    for k <– 0 to n-1 do
        if (mymin ≤ k ≤ mymax)
            for j <– k+1 to n-1 do
                A[k,j] <– A[k,j] / A[k,k];
        endif
        BARRIER(bar1, nprocs);
        for i <– max(k+1,mymin) to mymax do
            for j <– k+1 to n-1 do
                A[i,j] <– A[i,j] - A[i,k]*A[k,j];
    endfor
```


Available Parallelism and Amdahl's Law

Let S be the fraction of work that is inherently serial or replicated. The speedup on P processors is limited by $Speedup\ (P) \equiv \frac{T\ (1)}{T\ (P)} \le \frac{1}{S + \frac{(1-S)}{P}} \ = \ \frac{P}{1 - S\ (1 + P)}$



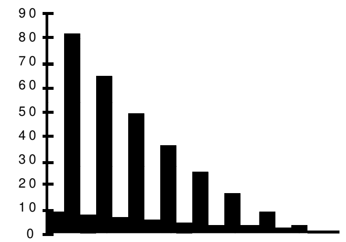
Caution:

- Parallelism Varies with problem size
- Programs are not simply serial / parallel
- Example: LU
  - Total work: $\alpha N^3 + O\left(N^2\right)$
  - Serialization: αN assuming zero-cost communication
  - Serial work is a diminishing fraction

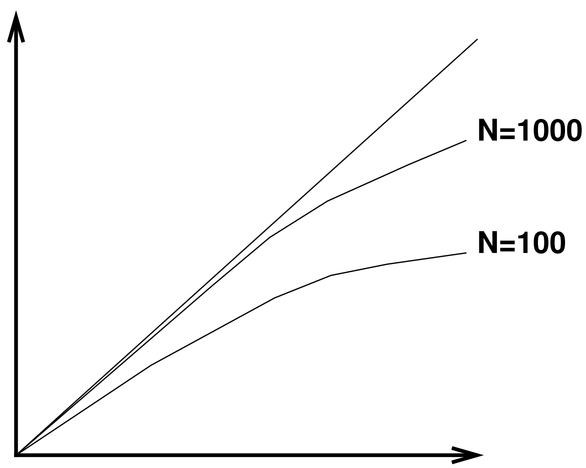
Generalizing Amdahl's Law

Parallism Profile: PP(t) = number of operations at step t

Speedup: $\text{Speedup}(P) \equiv \frac{T(1)}{T(P)} \le \frac{\sum_t pp(t)}{\sum_t \left\lceil \frac{pp(t)}{P} \right\rceil}$



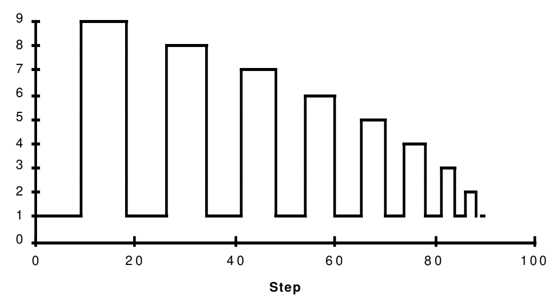
Speedup Limits



Parallelism with Row-wise Decomposition

Step K

- serial: n-k to scale pivot row
- parallel: n-k rows of n-k steps



Avoiding Serialization

Fundamentally a programming / compilation burden

- architecture can reduce motivation for serialization
  - ease of programming (decomposition)
  - support finer grain tasks
  - fast communication
- assuming sufficient parallelism in decomposed problem, what determines performance?
  - Load Balance
  - Communication
  - Extra work

Load Balance

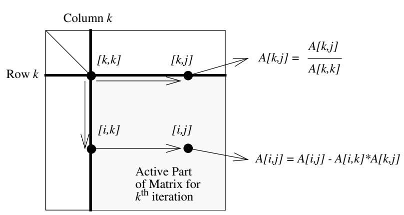
- Each processor should do same amount of work

$$Speedup\ (P) \le \frac{\text{Total Work}}{\underset{p \in P}{\text{MAX}}\ (Work\ (p))}$$

- Should not need to wait for others
- Cost of Load Imbalance = wait time

Load Imbalance in LU

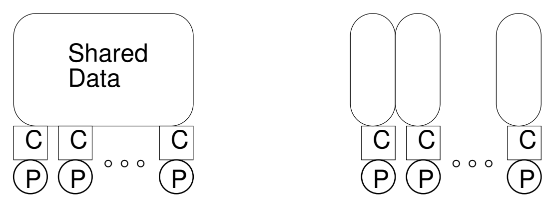
- Block row vs cyclic row



- Scaling Pivot row

Communication

Two Basic Models



Extended Storage Hierarchy

Distributed Address Space

Communication in an Extended Storage Hierarchy

- All shared accesses are expensive
- First access since last update is expensive

```
procedure LU( A[n,n] )
begin
for k <– 0 to n -1 do
    for_all j <– k+1 to n-1 do
        A[k,j] <– A[k,j] / A[k,k];
    BARRIER(bar1, nprocs);
    for_all i <– k+1 to n-1 by rows
        for j <– k+1 to n-1 do
         A[i,j] <– A[i,j] - A[i,k]*A[k,j];
endfor
```

- What is the total communication volume?
    - automatic replication

Communication Cost in a Distributed Address Space

- Remote shared accesses are expensive

```
procedure LU( A[n,n] distribution [cyclic , *])
begin
for k <– 0 to n -1 do
    for_all j <– k+1 to n-1 do
        A[k,j] <– A[k,j] / A[k,k];
    BARRIER(bar1, nprocs);
    for_all i <– k+1 to n-1 by rows
        for j <– k+1 to n-1 do
         A[i,j] <– A[i,j] - A[i,k]*A[k,j];
endfor
```

- What is the total communication volume?

Replication

- Remote shared accesses are expensive

```
procedure LU( A[n,n] distribution [cyclic , *])
begin
for k <– 0 to n -1 do
    for_all j <– k+1 to n-1 do
        A[k,j] <– A[k,j] / A[k,k];
    BARRIER(bar1, nprocs);
    for  j <– k+1 to n-1 do          Broadcast
        rowK[j] <- A[k,j]
    for_all i <– k+1 to n-1 by rows
        for j <– k+1 to n-1 do
         A[i,j] <– A[i,j] - A[i,k]*rowK[j]
endfor
```

- What is the total communication volume?

Cost of Communication

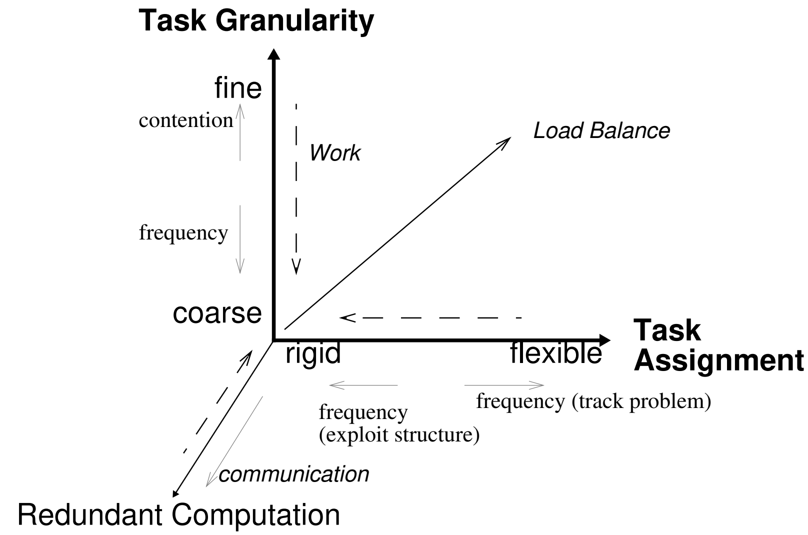
$$Cost = frequency \times \left(Overhead + Latency + \frac{Transfer\ Size}{Bandwidth} - Overlap \right)$$

- Total Volume
- Number of communication operations
    - long messages or bulk transfers
    - long cache lines
- Contention (balance)
    - e.g. all read pivot row
- Overlap

Extra Work

- Management of parallelism
  - loop distribution
  - task queues
  - synchronization
- Redundant computation
- Inefficiencies
  - address calculations
  - less locality

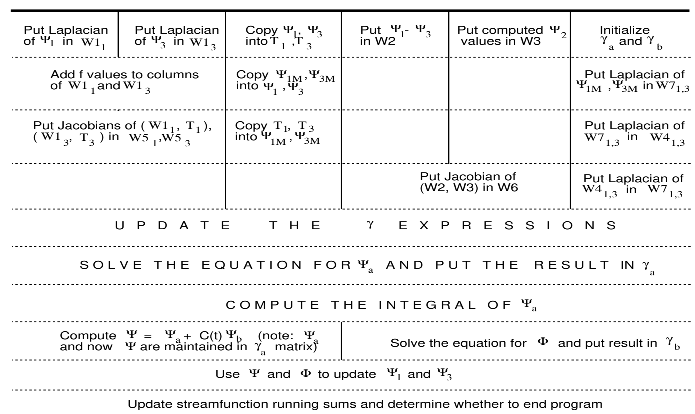
Basic Trade-offs



Trade-offs

- Load Balance
  - fine grain tasks
  - random or dynamic assignment
- Parallelism Overhead
  - coarse grain tasks
  - simple assignment
- Communication
  - decompose to obtain locality
  - recompute from local data
  - big transfers – amortize overhead and latency
  - small transfers – reduce overhead and contention

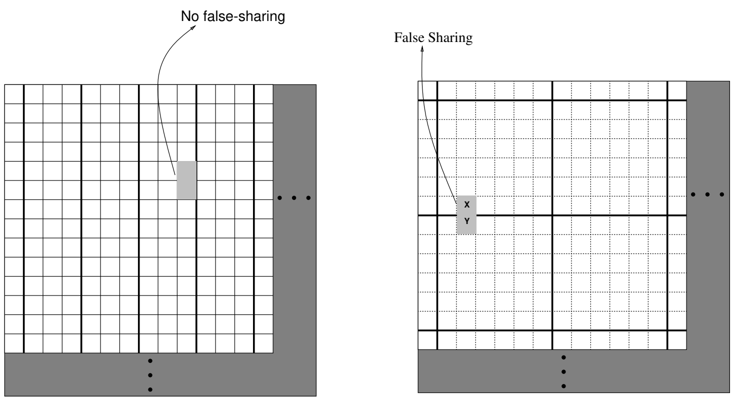
Case Study 1: Ocean Simulation



Note: Every box is a computation on an entire grid(s). Horizontal lines represent synchronization points among all processes, and vertical lines spanning phases demarcate threads of dependence.

Computations in a Time-step

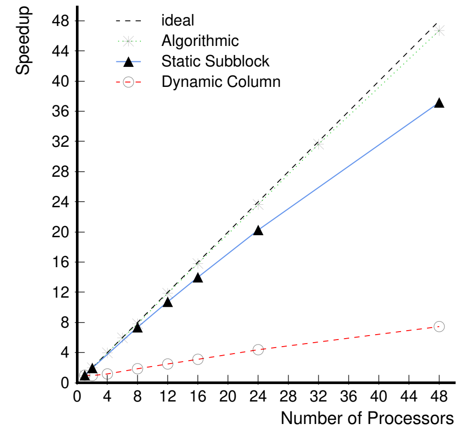
Ocean Simulation



(a) Columnwise (b) Subblock

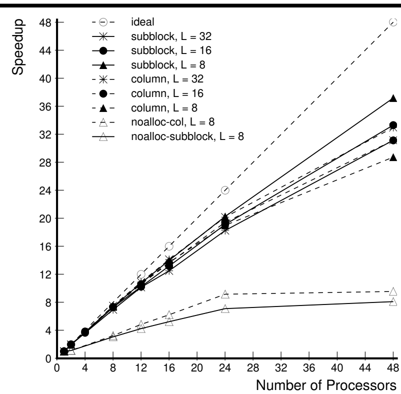
Two Static Partitioning Schemes

Ocean Simulation Performance



- Exploiting locality is critical to performance

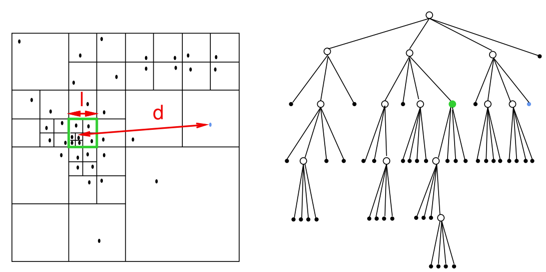
Ocean Simulation Performance



16 KByte Cache, 98-by-98 grids

- Data distribution is important
- Subblock partitioning is better than column

Case Study 2: Barnes-Hut



2–d Spatial Domain Quadtree Representation

Locality Goal:

Particles close together in space should be on same processor

Difficulties: • Nonuniform • Dynamically changing

Load Balancing

- Equal particles ≠ equal work.

  Solution: Assign costs to particles based on the work they do

- Work unknown and changes with time-steps

  Insight : System evolves slowly

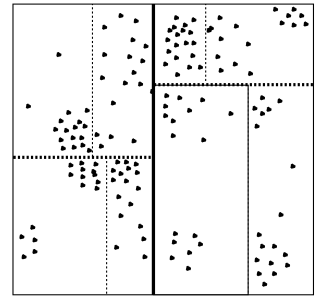
  Solution: *Count* work per particle, and use it as cost for next time-step.

Powerful technique for evolving physical systems

A Partitioning Approach: ORB

Orthogonal Recursive Bisection (*ORB*):

- Recursively bisect space into subspaces with equal work
- Continue until one partition per processor

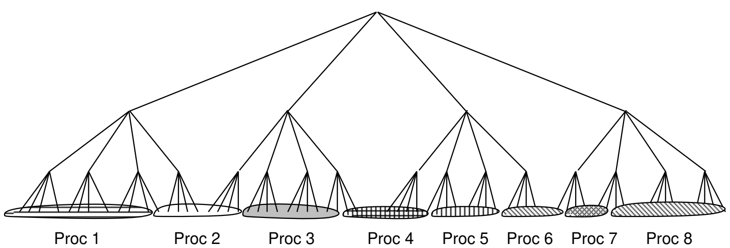


- High overhead for large no. of processors

Another Partitioning Approach: Costzones

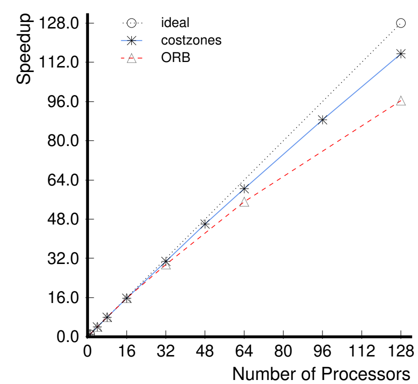
Insight:

Tree already contains an encoding of spatial locality.



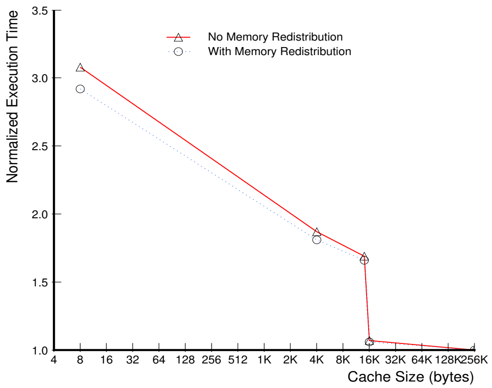
- Low overhead
- Very simple to program

Performance Results (Barnes-Hut)



Speedups on simulated multiprocessor (16K particles)

Key to Performance: Cache Locality



- Caching of communicated data is critical
- Memory allocation/migration is difficult, and doesn't help

Processor-centric view of Execution Time