Today's Plan

Today

- Two questions
  - How should you build a parallel computer?
  - Can we do automatic parallelization at a coarser level?

Parallel architectures

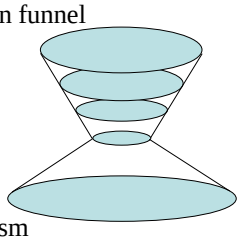
Automatic parallelization

- We've already argued that it's not practical
- A more forceful argument using two parallel algorithms as examples

Parallel Computers

The mismatch becomes worse

- Looking for much more parallelism, often at a larger granularity
- The automatic parallelization funnel



An alternative approach

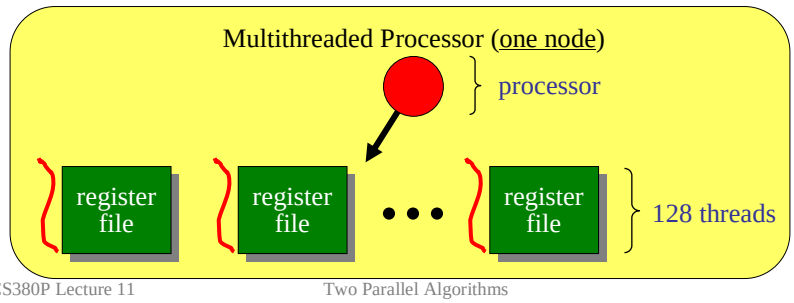
- Start with maximal parallelism
- Define parallel algorithms that operate on a large number of virtual processors
- Map the virtual processors to physical processors
  - Will typically aggregate many virtual processors on onto one physical processors
- **Is this a good idea?**

Case Study: The Tera MTA (*aka* Cray MTA)

The logical extreme in SM computers: Provide the illusion of uniform access to memory even as P scales to large values

The key idea

- Use multithreading to hide latency
- Each processor supports multiple threads. At each clock cycle, the processor switches to another thread. Latency is hidden because by the time a thread executes its next cycle, any expensive memory access had already completed.



The Tera MTA (cont)

Massive parallelism

- How do you get so much parallelism?
- Exploit parallelism at many levels
  - Instruction level
  - Within basic blocks
  - Across different processes
  - Between user code and OS code

Advantage

- Supports hard-to-parallelize applications

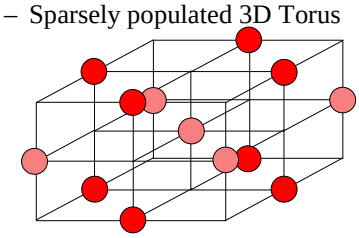
Disadvantage

- Everything was custom designed
- GaAs instead of CMOS technology

The Tera MTA (cont)

Interconnection Topology

- Sparsely populated 3D Torus



Memory

- Randomized memory allocation to reduce contention
- No caches

- Why?
- P processors with latency L to memory ⇒ network must hold P × L messages if each processor will be busy each cycle
  - As L grows, we need to reduce P
  - This is why urban sprawl is bad

The Tera Computer—Epiloque

MTA-1

- Delivered in late 1990's
- Set record for integer sort in 1997

MTA-2

- Follow-on to MTA-1 implemented in CMOS technology
- Impressive speedups on hard problems [Anderson, et al SC2003]

Lessons

- With a good design, good performance can be delivered for a wide variety of application domains

Aside

- Recognizes the importance of good tools
- Large compiler effort with excellent personnel
- In 2000, Tera Computer Co. bought Cray, Inc. from SGI

Distributed Memory Architectures

Goal

- Provide a scalable architecture
- Processes communicate through messages

Disadvantage

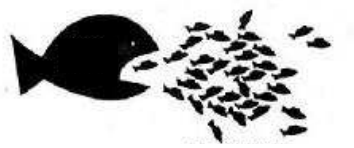
- Often considered more difficult to program
- The distributed memory model is often mistakenly used synonymously with "message passing"
  - This is a short-sighted view, as we can imagine divorcing the programming model from the hardware substrate

Examples

- Most of the larger machines are distributed memory machines

The Law of Nature

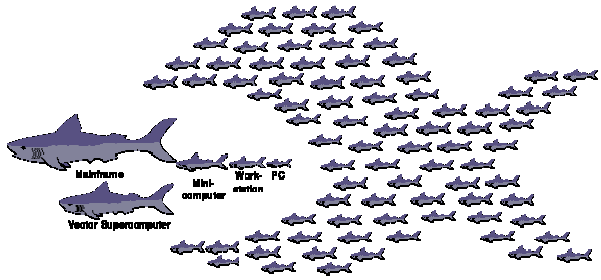
Big fish eat little fish



The Killer Micros

Economies of scale

- Sales of microprocessors took off in the 80's
- Supercomputers with custom-designed processors found it difficult to compete against those with commodity processors



NOW

Networks of Workstations (NOW, COW…)

Use distributed system as a supercomputer

- Don't just reuse the CPU, reuse the entire workstation, including the CPU, memory, and I/O interface
- Views parallel computing as an extension of distributed computing
- Some claim that Networks of Workstations provide parallel computing for free

Problems?

- Software is still not a commodity part
- Moreover, the simpler the hardware, the more the software needs to do
- Workstations typically not designed with NOW's in mind, so some components are not quite right
  - e.g., Need to redesign the network interface

Clusters

Basic idea

- Build distributed memory machines from commodity parts, perhaps with some new redesign
  - e.g., different form factors for rack-mounting
- Connect these workstations with high-speed commodity networks

Advantages

- Scalable price/performance
- Can buy a few nodes or many nodes
- Supports incremental expansion
- Relatively low cost

Disadvantages

- Relatively high communication latency compared to CPU speed

Parallel Programming—the Big Picture

How should we write parallel programs?

- Pthreads
- MPI

} Are these good solutions?

Other alternatives

- Use higher level parallel languages
- Automatic parallelization

Automatic Parallelization

Focus on loops

- Large body of work on loop transformations
- Many loop transformations proposed
- The key question: dependence analysis
  - Can one iteration of a loop execute concurrently with another iteration?

```
for i = 1 to n do
  a[i] = a[i-1]
```

```
i = 1   a[1] = a[0]
i = 2   a[2] = a[1]
i = 3   a[3] = a[2]
        . . .
```

Dependence testing

- Solve system of linear equations to see if a dependence exists across iterations

Automatic Parallelization (cont)

Limitations of this approach?

- Not everything can be expressed as a dense array
- Language semantics interfere
  - Loop transformations are most amenable to Fortran
  - Most modern languages do not have true multi-dimensional arrays, but instead use arrays of arrays
  - These arrays of arrays hinder dependence analysis

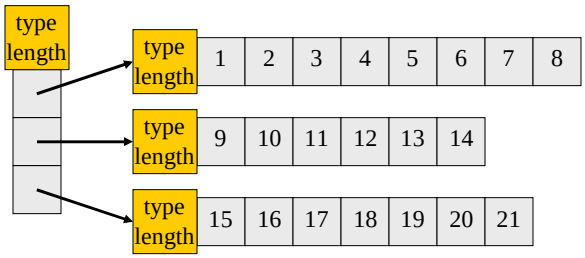
Comparing Arrays

A 2D array in Fortran

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24

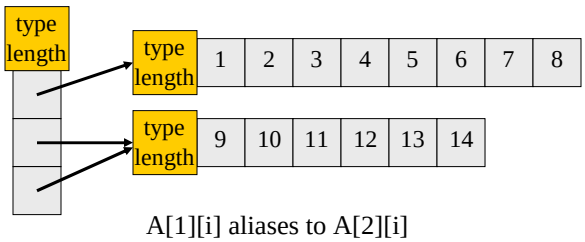
for $i \neq j$ or $k \neq l$,
A[i][k] and A[j][l] refer to distinct memory locations

An array of arrays in Java



Java Arrays

Elements within an array can alias with one another



A[1][i] aliases to A[2][i]

Implications?

- Complicates dependence testing
- Can't simply reason algebraically

Limits of Automatic Parallelization

Perhaps we're just not trying hard enough?

- Parallel algorithms are often fundamentally different from sequential algorithms
- Finding good parallel algorithms is AI-complete [Bill Mark, 2005]

Today

- Two examples that illustrate this point

Image Understanding

Computer identification of images

Example: DHS scans images looking for potential terrorist threats

Image Understanding

Step 1: Convert image to binary image using thresholding

Step 2: Identify connected components

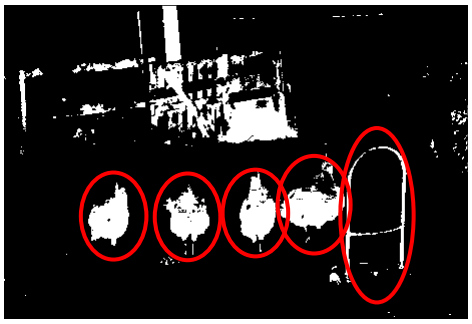
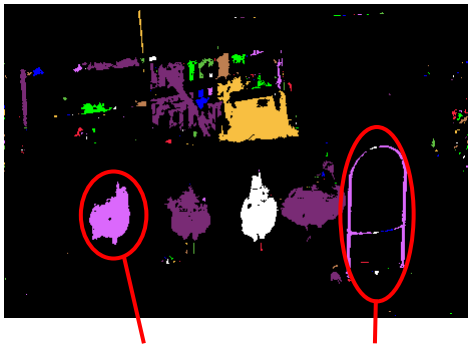


Image Understanding

Step 3: Identify the different components using classification



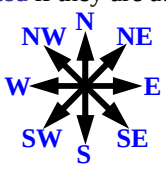
turkey hand cart

Let's focus on Step 2

Counting Connected Components

Connected Components

- Given a binary image, count the number of connected components.
- Two 1's are connected if they are adjacent to each other in any of the 8 compass directions



- Example: The following is a single connected component

0	1	0	0
1	0	1	0
0	0	0	1

- How can we compute the number of connected components quickly?

The Obvious Recursive Approach

Represent each pixel as a node in a graph

- Find pixels that are 1's
- "Bleed out" from these 1's using Breadth First Search
- Continue with unmarked 1's
- Stop when all 1's have been marked

1	1	1	1	1
0	0	0	0	1
1	1	1	0	1
1	0	0	0	1
1	1	1	1	1

What is the running time?

- O(size-of-image)

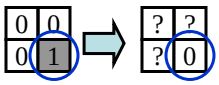
Clever algorithm: O(m+n) for an m×n image and additional hardware

A Parallel Solution

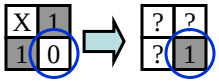
The Amazing Levialdi Shrinking Operator (1972)

- A **morphological operator** that takes a window of pixels as input and modifies a window of pixels as output
- Each pixel simultaneously changes state according to the following rules

(1) A 1 bit becomes a 0 if there are 0's to its West, NW, and North



(2) A 0 bit becomes a 1 if there are 1's to its West and North



(3) All other bits remain unchanged

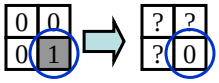
A Parallel Solution

The Amazing Levialdi Shrinking Operator (1972)

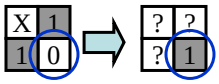
- A **morphological operator** that takes a window of pixels as input and modifies a w
- Each pixel si to the following rules

What do these rules do?

(1) A 1 bit becomes a 0 if there are 0's to its West, NW, and North



(2) A 0 bit becomes a 1 if there are 1's to its West and North

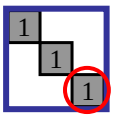


(3) All other bits remain unchanged

Deconstructing the Levialdi Operator

Basic idea

- Each connected component has a well-defined bounding box



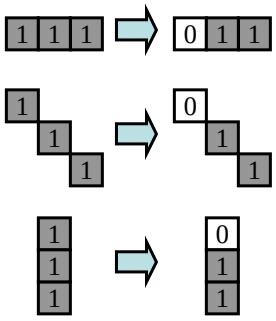
- If we can identify these bounding boxes, we can identify the connected components
- Note that each bounding box has a well-defined lower-right corner

Deconstructing the Levialdi Operator II

Basic idea (cont)

- These rules cause a connected component to shrink towards the lower right hand corner of its bounding box, until it eventually disappears (poof!)

- Rule (1) causes the upper-left most 1 to disappear



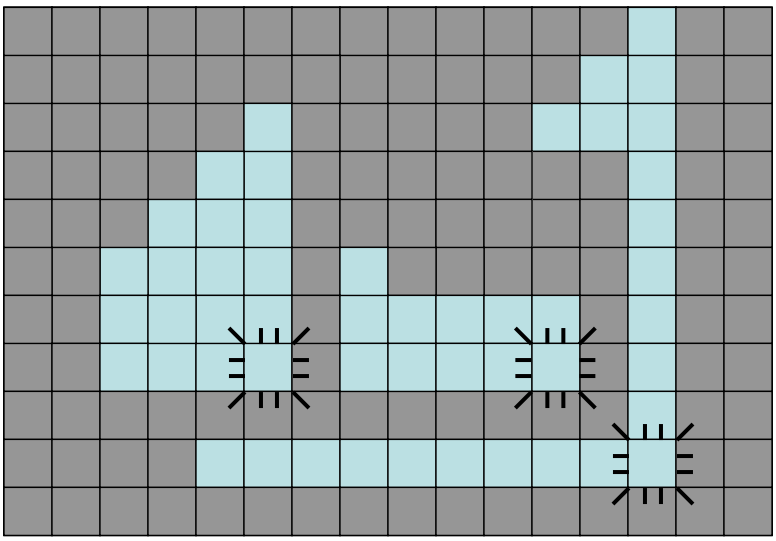
Deconstructing the Levialdi Operator III

Basic idea (cont)

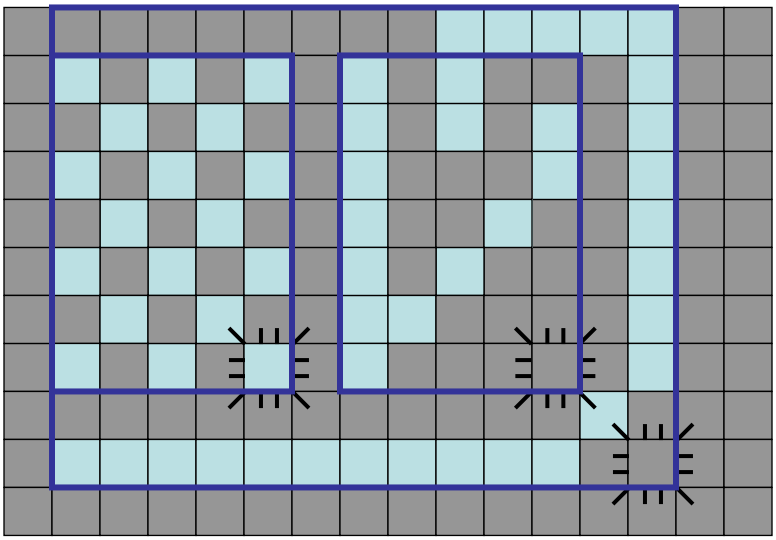
– Rule (2) creates a 1 if a 0 sits at the bottom-right of two 1's

– Thus, holes in the bounding box are filled as the connected component shrinks.

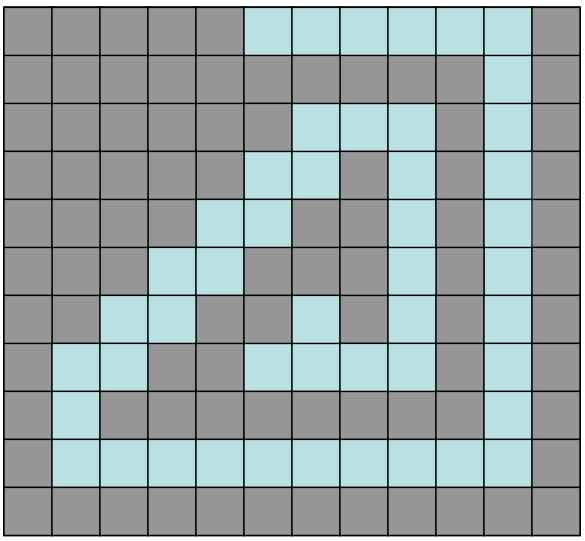
The Algorithm in Action



The Algorithm in Action



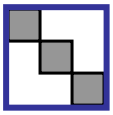
The Dreaded Spiral



Does This Algorithm Work?

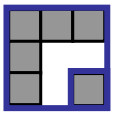
Problem

- What if two components have the same bounding box?
- This cannot happen, because then the two would be connected

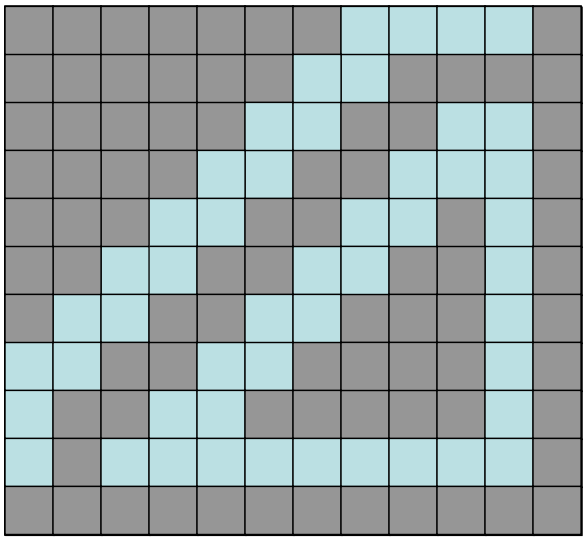


Problem

- What if two components have bounding boxes with the same lower right corner?
- This can happen, but the algorithm will detect the poofs at different times



Overlapping Bounding Boxes



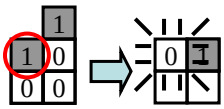
How Do We Recognize the Disappearing Components?

Identify the lower right corner

– To check for the disappearance of a connected component

For each [1→0] transition,

check to see if the bit's East, SE, and South bits are 0.



Q: What if the bit to the NE is a 1?

Won't we mistakenly identify a poof when we shouldn't?

A: The bit to the NE would have set the bit to the East to a 1, so our check above would have failed

The Levialdi Algorithm

```
Boolean Image[n][n];       // Initialize boundaries to 0's
Boolean Next[n][n];
int     count = 0;
Boolean moreOnes = False;

do {
   shrink();
   countAndTest();
   Swap (Next, Image);
} while (moreOnes);
```

The Levialdi Algorithm (cont)

```
shrink()
{
   for (int i=1; i<=n; i++)
   {
      for (int j=1; j<=n; j++)
      {
         Next[i][j] = Image[i][j] && (Image[i][j-1]   ||
                                      Image[i-1][j-1] ||
                                      Image[i-1][j]);
         Next[i][j] = Image[i][j] || (Image[i-1][j] &&
                                      Image[i][j-1] &&
                                     !Image[i][j]);
      }
   }
}
```

Rule 1

Rule 2

The Levialdi Algorithm (cont)

```
countAndTest()
{
   for (int i=1; i<=n; i++)
   {
      for (int j=1; j<=n; j++)
      {
         if (Image[i][j] && !Next[i][j] &&
                           !(Next[i][j+1] ||
                             Next[i+1][j+1] ||
                             Next[i+1][j]))
            count++;
         if (Next[i][j])
            moreOnes = True;
      }
   }
}
```

count poofs

Time Complexity

Q: How long does it take to shrink a connected component?

A: O(w+h), where w and h are the width and height of the connected component's bounding box.

Q: How large can a bounding box be?

A: No bigger than m × n, the size of the image.
So the running time of the algorithm is O((m+n) × iter), where iter is the time it takes to iterate over the 2 for loops

What's Neat About this Algorithm?

Plenty of parallelism

- The algorithm uses only local information at each pixel, so we can implement the algorithm in parallel.
- We can use a 2D array of processors, with one processor per pixel.

The larger point

- The algorithm is completely different from the sequential one

The Solar System

Isaac Newton (17th Century)

- A planet orbiting the Sun follows an elliptical path
- This 2-body problem is easy to solve mathematically

Is the solar system stable?

Laplace failed

- Made simplifying assumptions
- Solvable but inaccurate

Still an unsolved problem

Computer Simulations of the Solar System [Saha, et al, 1997]

This is an obvious N-body problem

- 1 sun
- 9 planets
- 1 moon

Largest solution to date

- 100M years
- Took about a year to simulate on a uni-processor workstation (c. 1996)

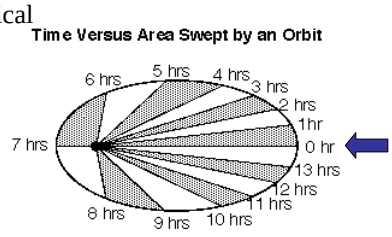
Can we parallelize this?

- Obvious answer: Assign one processor to each planet
- How can we get significantly more parallelism?

Integration-Based Approach

Leverage Newton's work

- Without the gravitational effects of the other planets, each planet's orbit would be elliptical

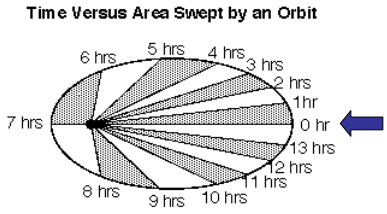


- We can estimate where each planet will be t time steps in the future
- Given estimates for all of the planets, we can then adjust each individual estimate to produce an accurate simulation

Integration-Based Approach (cont)

Parallelize across time

- Each processor computes planet locations for a different time step in the future



- Since there are so many time steps, we have massive parallelism!
- We can use the parallel prefix operation to make corrections for each estimate

Parallel Prefix Revisited

Prefix Sum

Original array

2	7	1	2	4	5	2	3

Parallel prefix with + operator

2	9	10	12	16	21	23	26

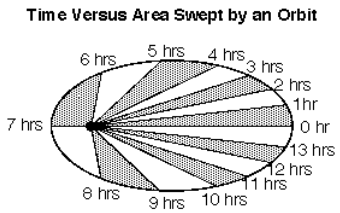
→ Compute partial sums in this direction

Solar System

- Instead of accumulating partial sums, accumulate partial errors in the original approximation

Integration-Based Approach (cont)

A few details



- Can use thousands of processors
- Largest feasible timestep is about one week
- Requires quadruple precision floating-point precision for a few iterations
- Goal: simulate 10B years

Summary

Parallel algorithms

- Often fundamentally different from sequential algorithms
- Limits the attractiveness of automatic parallelization
- Are there intermediate solutions? Stay tuned . . .

Next Time

Reading #9

- Anderson and Snyder paper

Assignment 4

- Available on Friday

Meet in GDC?

- Stay tuned