Challenges and Opportunities of Parallelism

Lawrence Snyder

www.cs.washington.edu/homes/snyder

8 April 2009

The Situation Today …

- Consumer and business software cannot achieve further performance improvements without parallelism.
- Is that a problem … or an opportunity?
- My goal today is not to "sell" new and wondrous results, but to give a global perspective on the parallel "problem" answering:
  - Why is it so hard?
  - Where do we go from here?

© Larry Snyder, All Rights Reserved

Outline Of This Talk

- Small Parallelism: today's technology facts
- A Closer Look -- Facts of Life
- Large Parallelism: today's technology facts
- A Closer Look -- Living with Parallelism
- A trivial exercise focuses our attention
- A little "science" can help

© Larry Snyder, All Rights Reserved

Today's Facts

Single Processor

Opportunity

Moore's law continues, so use more gates

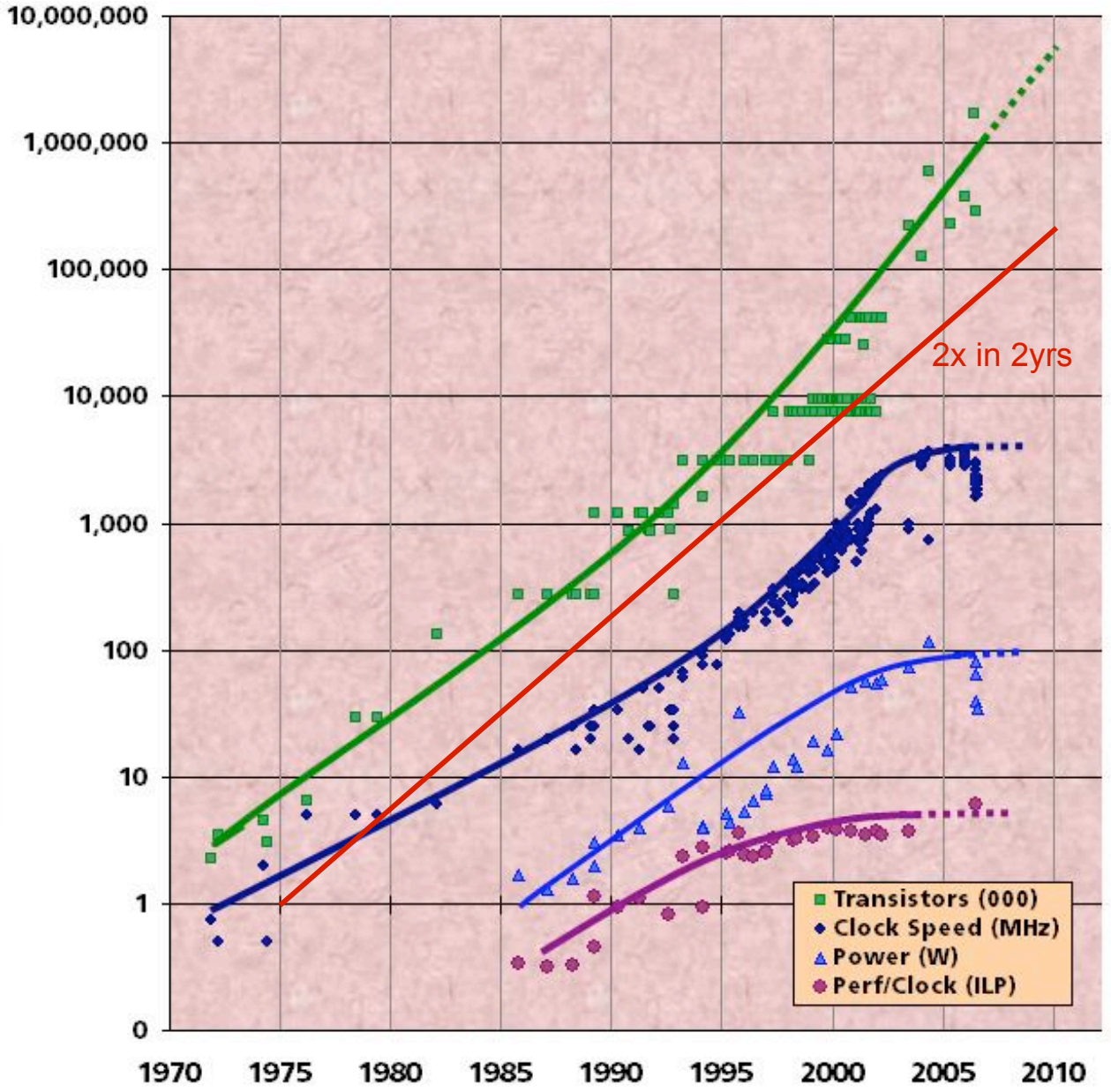


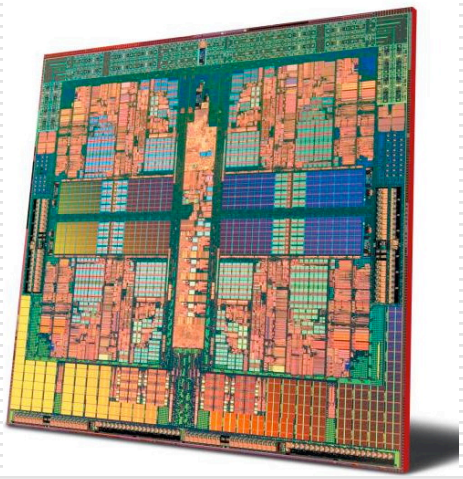
Figure courtesy of Kunle Olukotun, Lance Hammond, Herb Sutter & Burton Smith

© Larry Snyder, All Rights Reserved

Today's Facts

- Laptops, desktops, servers, etc. now are parallel (mulitcore) computers … why?
- Multi-core gives "more" computation and solves difficult hardware design problems
  - What to do with all of the transistors: Replicate
  - Power Issues
  - Clock speeds

… consider each issue



© Larry Snyder, All Rights Reserved

Multi-core Replicates Designs

- Traditionally we have used transistors to make serial processors more aggressive
  - Deeper pipelining, more look-ahead
  - Out of order execution
  - Branch prediction
  - Large, more sophisticated TLBs, caches, etc.
- Why not continue the aggressive design?

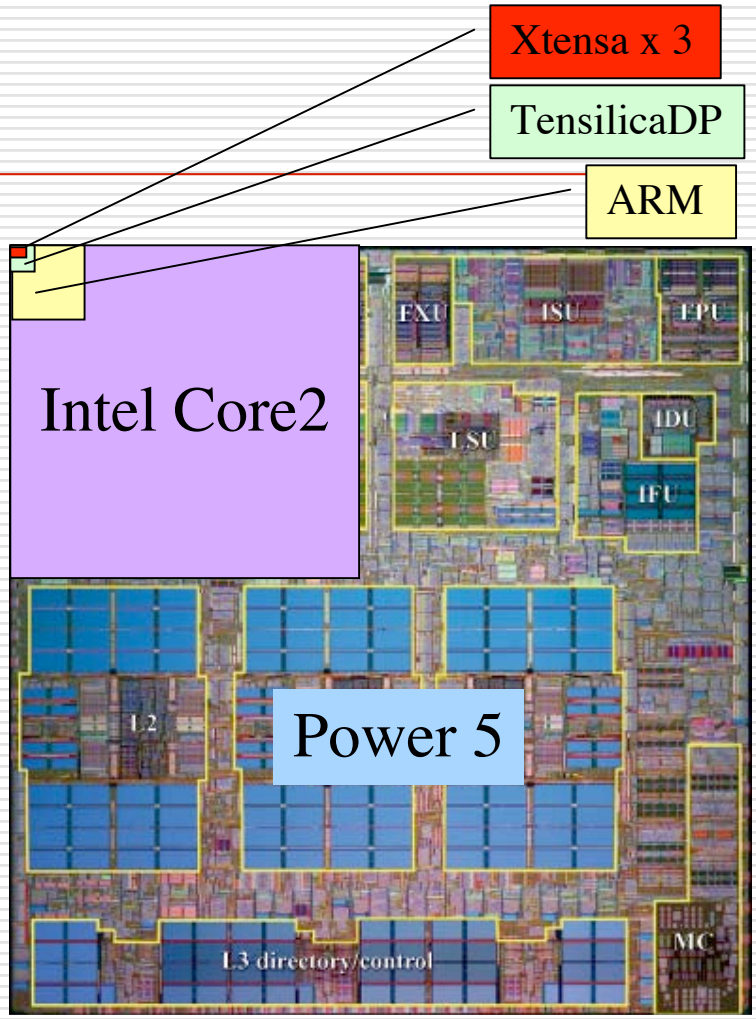
  … Diminishing returns, limit on ILP, few new ideas

Bottom Line: Sequential instruction execution reaching limits

© Larry Snyder, All Rights Reserved

Size vs Power

- Power5 (Server)
  - 389mm^2
  - 120W@1900MHz
- Intel Core2 sc (laptop)
  - 130mm^2
  - 15W@1000MHz
- ARM Cortex A8 (automobiles)
  - 5mm^2
  - 0.8W@800MHz
- Tensilica DP (cell phones / printers)
  - 0.8mm^2
  - 0.09W@600MHz
- Tensilica Xtensa (Cisco router)
  - 0.32mm^2 for 3!
  - 0.05W@600MHz



Each processor operates with 0.3-0.1 efficiency of the largest chip: more threads, lower power

© Larry Snyder, All Rights Reserved

Multi-core Design Space

- Smaller, slower implies more modest design

	Traditional	Core Design	
micro-architecture	out of order	in order	
size	50	10	mm^2
power	37.5	6.25	W
frequency	4	4	GHz
threads	2	4	
single thread	1	0.3	relative perf
vector operations	4	16	words wide
peak throughput	32	128	Gflop/s
area capacity	0.6	13	Gflops/mm
power capacity	0.9	20	Gflops/W

Source: Doug Carmean, Intel

© Larry Snyder, All Rights Reserved

Multi-cores Have Their Problems

- Single threaded computation doesn't run faster (it may run slower than a 1 processor per chip design)
- Few users benefit from m-c ||ism today
  - Existing software is single threaded
  - Compilers don't help and often harm parallelism
  - It's often claimed that OS task scheduling is one easy way to keep processors busy, but there are problems
    - limited numbers of tasks available
    - contention for resources, e.g. I/O bandwidth
    - co-scheduled tasks often have dependences -- no advantage to or prevented from running together

© Larry Snyder, All Rights Reserved

Legacy Code

- Even casual users use applications with a total code base in the 100s of millions LOC … and it's not parallel
- There are not enough programmers to rewrite this code, even if it had || potential
- With single processor speeds flat or falling, this code won't improve

Challenge

How to bring performance to existing programs

© Larry Snyder, All Rights Reserved

What Can Microsoft Do?

- “Parallelism requires adjustments at every level of the stack … the repartitioning of different tasks to different layers … So look for a rebalancing of roles and runtimes. We need to formalize that in the operating system. Expect the first pieces in the next generation of Windows.” Craig Mundy, Microsoft Chief Research & Strategy Officer, 3 October 2008

© Larry Snyder, All Rights Reserved

Legacy Code

- Even casual users use applications with a total code base in the 100s of millions LOC … and it's not parallel
- There are not enough programmers to rewrite this code, even if it had || potential
- With single processor speeds flat or falling, this code won't improve

Challenge
How to bring performance to existing programs

Opportunity
Much legacy code supports backward compatibility -- ignore

© Larry Snyder, All Rights Reserved

Potential of Many Threads - Amdahl

- Maximum performance improvement by parallelism is S-fold if sequential part is1/S

$$T_P = 1/S \cdot T_S + (1 - 1/S) \cdot T_S/P$$

- "Everyone is taught Amdahl's Law in school, but they quickly forget it" -- T. R. Puzak, IBM
- More complex in multi-core case: programs that are 99% parallel get speedup of 72 on 256 cores [Hill & Marty]

© Larry Snyder, All Rights Reserved

Summary So Far

- Opportunities --
  - Moore's law continues to give us gates
  - Multi-core is easy design via replication
  - Replicating small, slower processors fixes power problems & improves ops/second
- Challenges --
  - Smaller, slower designs are smaller, slower on 1 thread
  - Huge legacy code base not improved
  - Parallelism doesn't speed-up sequential code. Period.

© Larry Snyder, All Rights Reserved

Outline Of This Talk

- Small Parallelism: today's technology facts
- A Closer Look -- Facts of Life
- Large Parallelism: today's technology facts
- A Closer Look -- Living with Parallelism
- A trivial exercise focuses our attention
- A little "science" can help

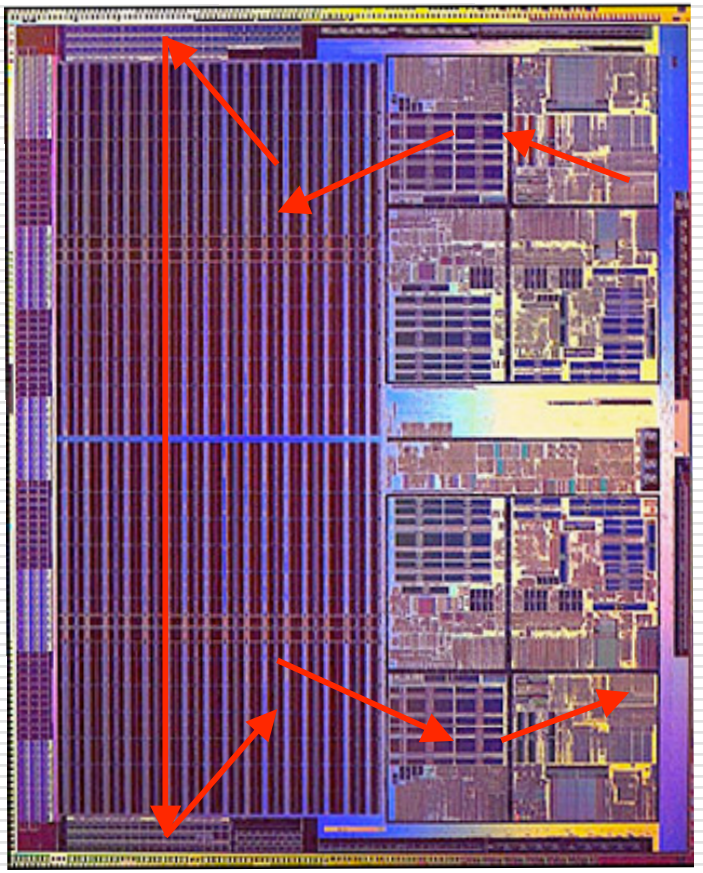
© Larry Snyder, All Rights Reserved

But There's More To Consider

- Two processors may be 'close' on the silicon, but sharing information is still expensive
- r1 ➡ L1 ➡ L2 ➡ coherency protocol ➡ L2 ➡ L1 ➡ r2
- Opteron Dual Core: more than 100 instruction times

Challenge
On Chip Latency

- Latency between cores only gets worse



© Larry Snyder, All Rights Reserved

Latency -- Time To Access Data

- Latencies (relative to instruction execution) have grown steadily over the years, but (transparent) caching has saved us
- No more
  - Interference of multiple threads reduces the benefits of spatial and temporal locality
  - Cache coherence mechanisms are
    - good for small bus-based SMPs
    - slower and complex as size, distance and decentralization increase
- Thus, latency growth is a problem

© Larry Snyder, All Rights Reserved

Bandwidth On/Off Chip

- Many applications that are time-consuming are also data intensive e.g. MPEG compress
- *C* cores share the bandwidth to memory: *available_bandwidth/C*
- Caching traditionally solves BW problems, but Si devoted to cache reduces number of cores
- A problem best solved with better technology

© Larry Snyder, All Rights Reserved

Memory Model Issues

- When we program, we usually think of a single memory image with "one" history of transitions:
  $s_0, s_1, \ldots, s_k, \ldots$
- Not true in the parallel context
  - Deep technical problem
  - Two cases
    - Shared memory
    - Distributed memory
- The consequences of this fact are the largest challenges facing parallel programming

© Larry Snyder, All Rights Reserved

Facts of Life Summary: Challenges

- Latency on chip will increase as core count increases
  - significant already
  - both logic complexity and “electronic” distance
- Bandwidth to L3 and memory shared
- Memory model for programmers
  - Presently broken
  - Extensive research has failed to find alternate
  - May be a “show stopper”

© Larry Snyder, All Rights Reserved

Outline Of This Talk

- Small Parallelism: today's technology facts
- A Closer Look -- Facts of Life
- Large Parallelism: today's technology facts
- A Closer Look -- Living with Parallelism
- A trivial exercise focuses our attention
- A little "science" can help

© Larry Snyder, All Rights Reserved

Large Parallel Facts

- A parallel computer (IBM Roadrunner) has achieved 1 Peta Flop/S (1.1×10^{15} Flop/S)

© Larry Snyder, All Rights Reserved

Large Parallel Facts

- A parallel computer (IBM Roadrunner) has achieved 1 Peta Flop/S (1.1×10^{15} Flop/S)

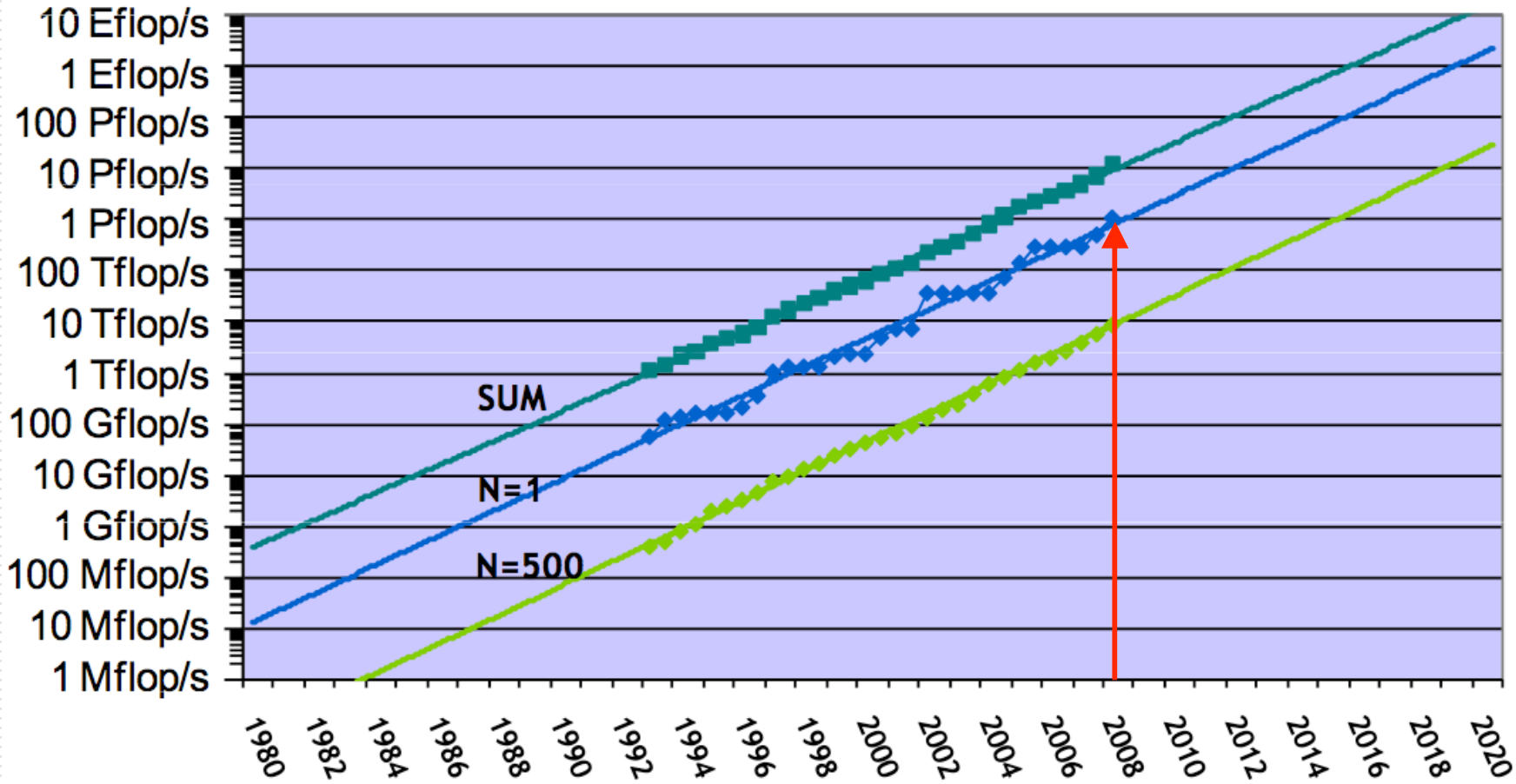
© Larry Snyder, All Rights Reserved

Roadrunner Specs

- 12,562 dual-core AMD Opteron® chips and 12,240 STI PowerXCell 8i chips (25.6 GF SP, 12.8 GF DP per SPE) … each AMD core gets a Cell chip
- 98 terabytes of memory
- 3.8 MW total delivering 376 MFlops/W
- 278 refrigerator-sized racks; 5,200 ft^2 footprint
- Other Data:
  - 10,000 connections – both Infiniband and Gigabit Ethernet
  - 55 miles of fiber optic cable
  - 500,000 lbs.
  - Shipped to Los Alamos in 21 tractor trailer trucks

© Larry Snyder, All Rights Reserved

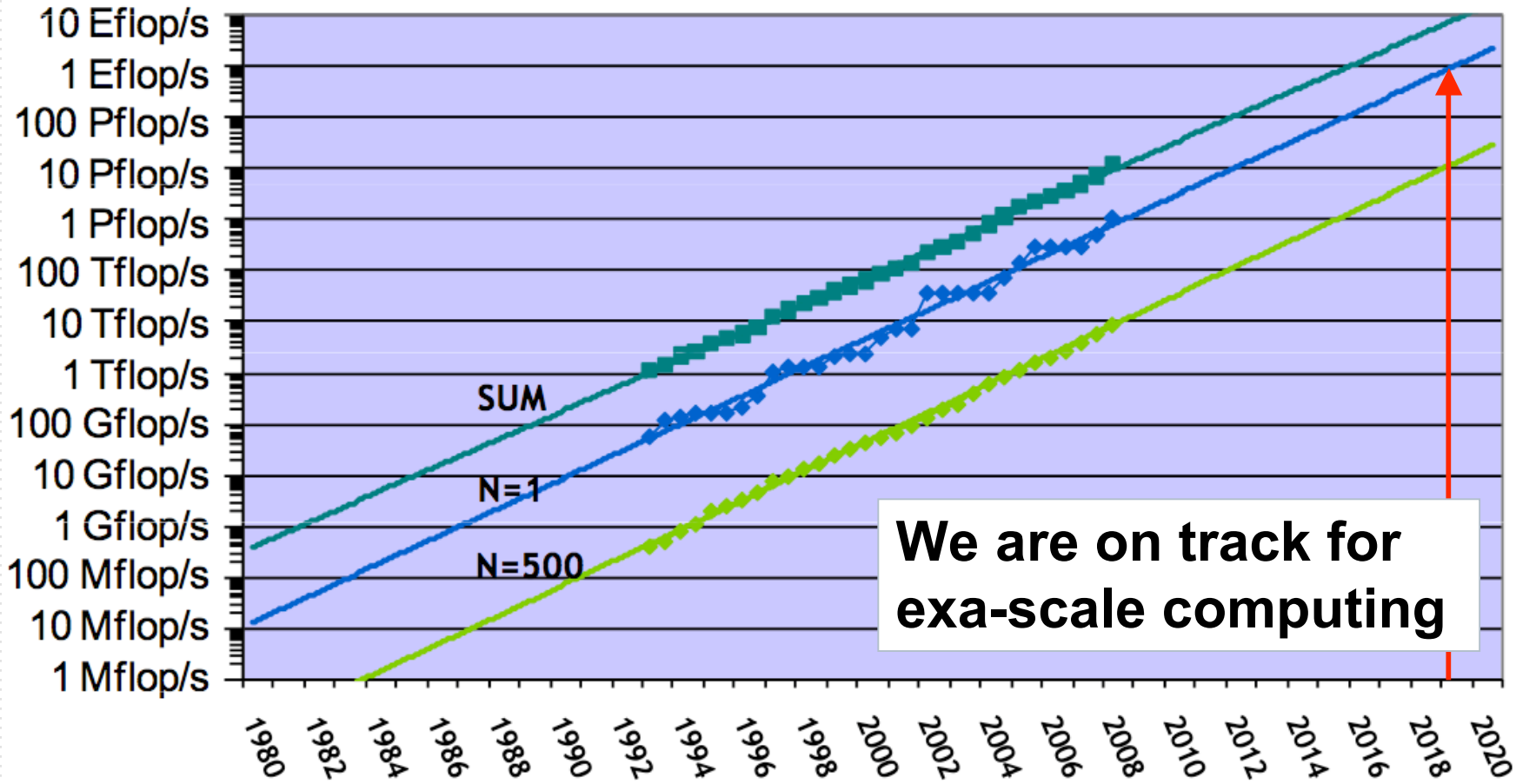
Performance: Top 500



Source: Top 500 Supercomputers

© Larry Snyder, All Rights Reserved

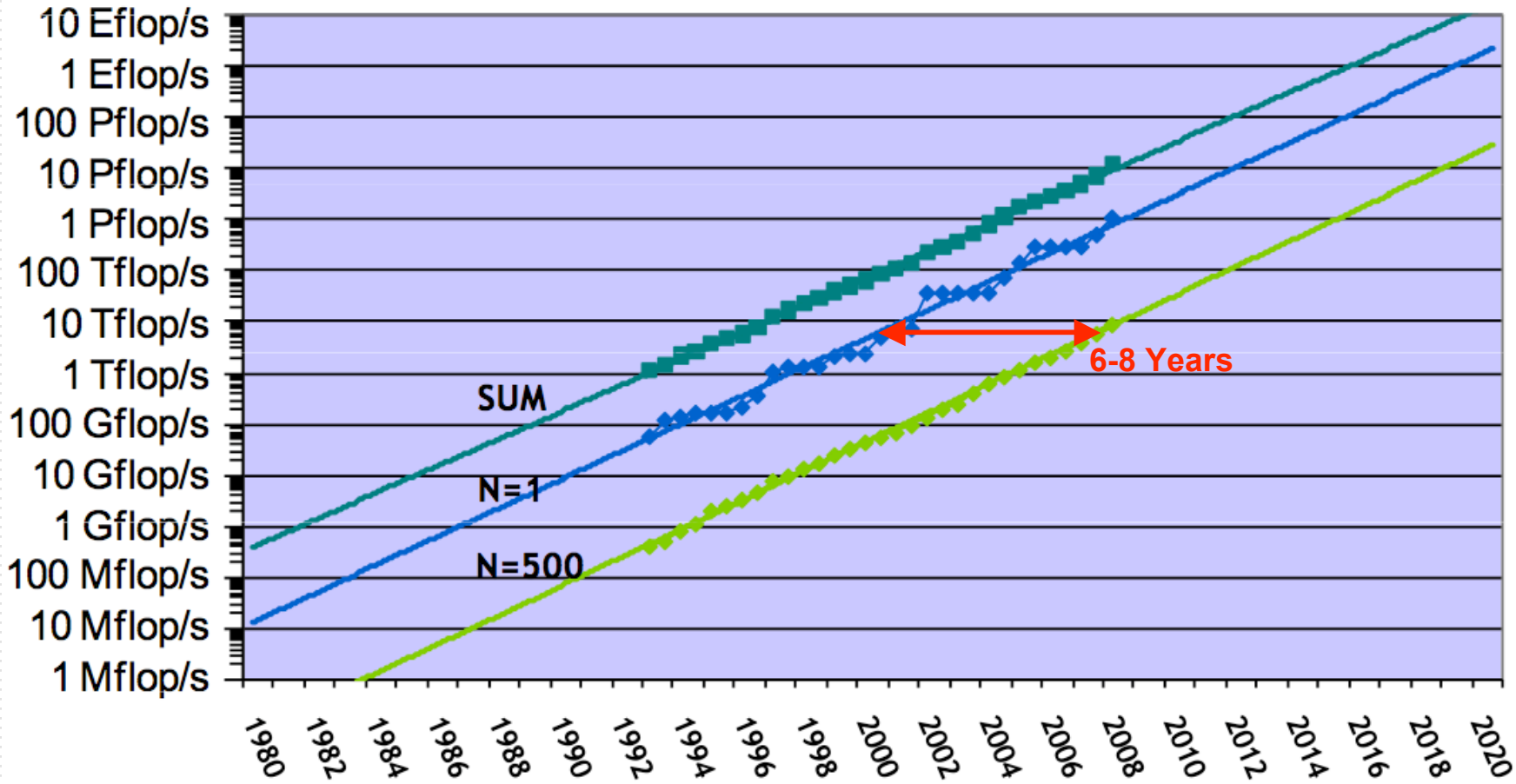
Performance: Top 500



Source: Top 500 Supercomputers

© Larry Snyder, All Rights Reserved

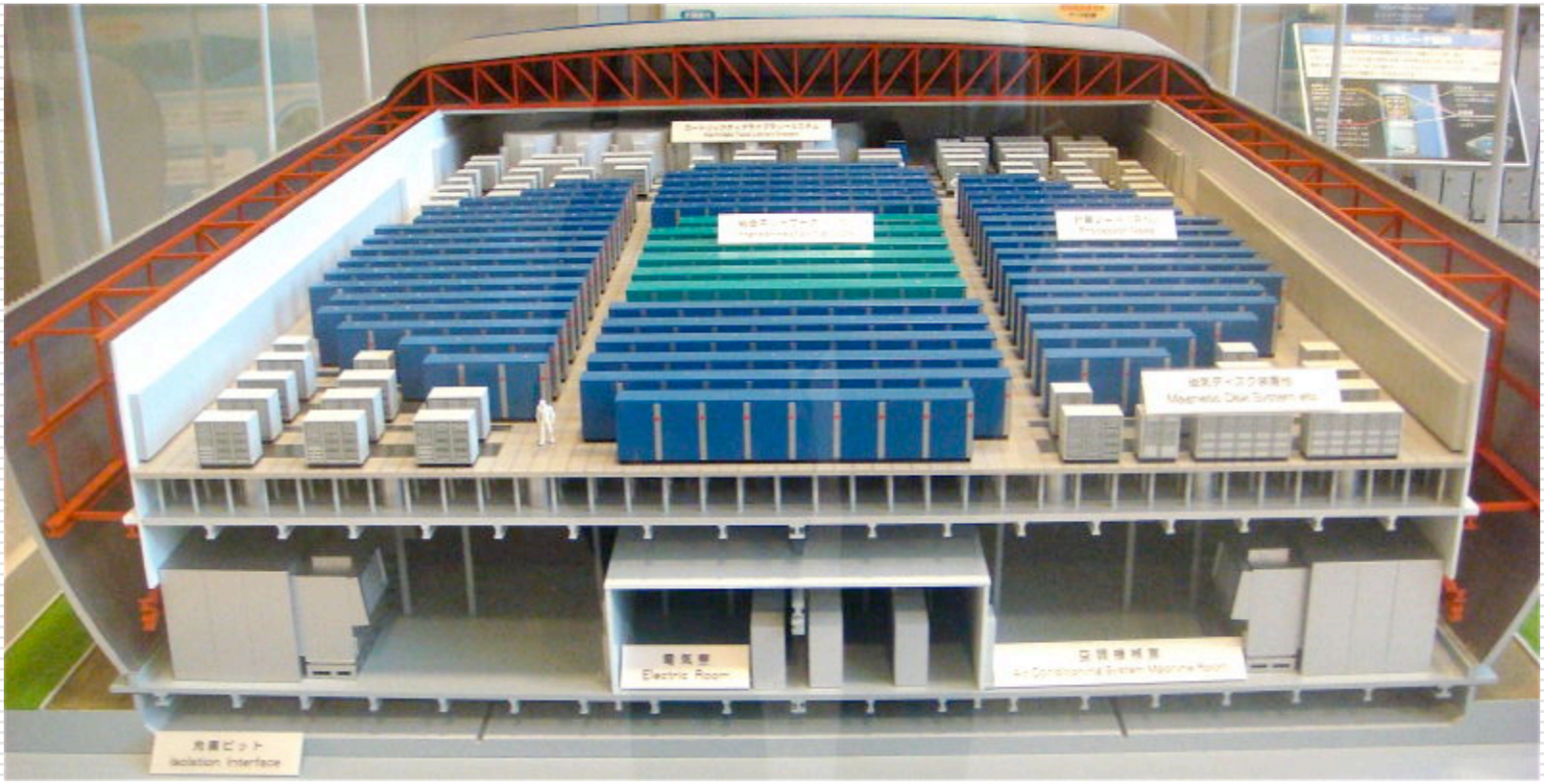
Performance: From 1st To Last



© Larry Snyder, All Rights Reserved

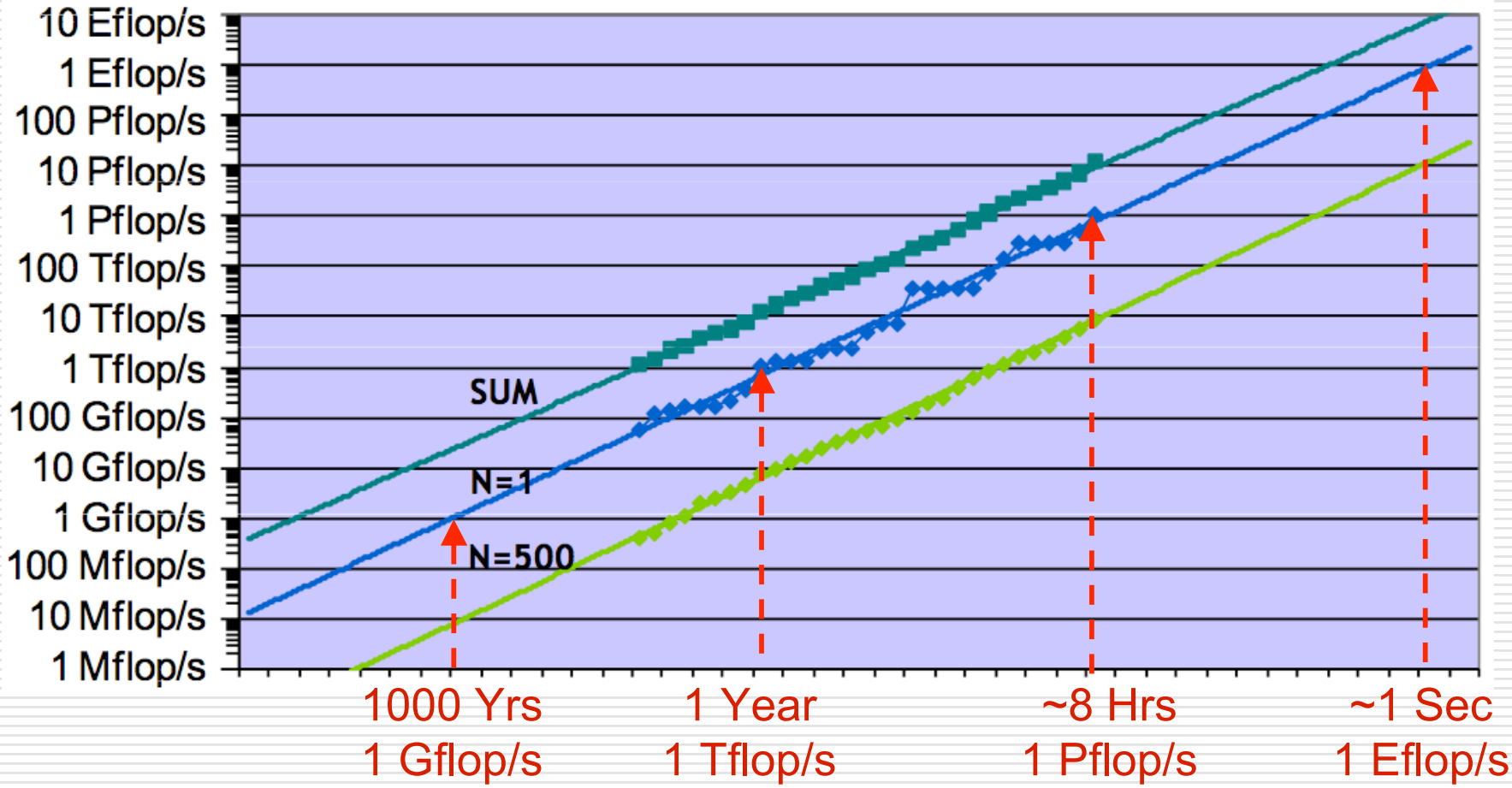
Earth Simulator

- Started at #1 June 2002, 35.8 TF
- Decommissioned Sept 2008; still at #73 #500=12.6TF



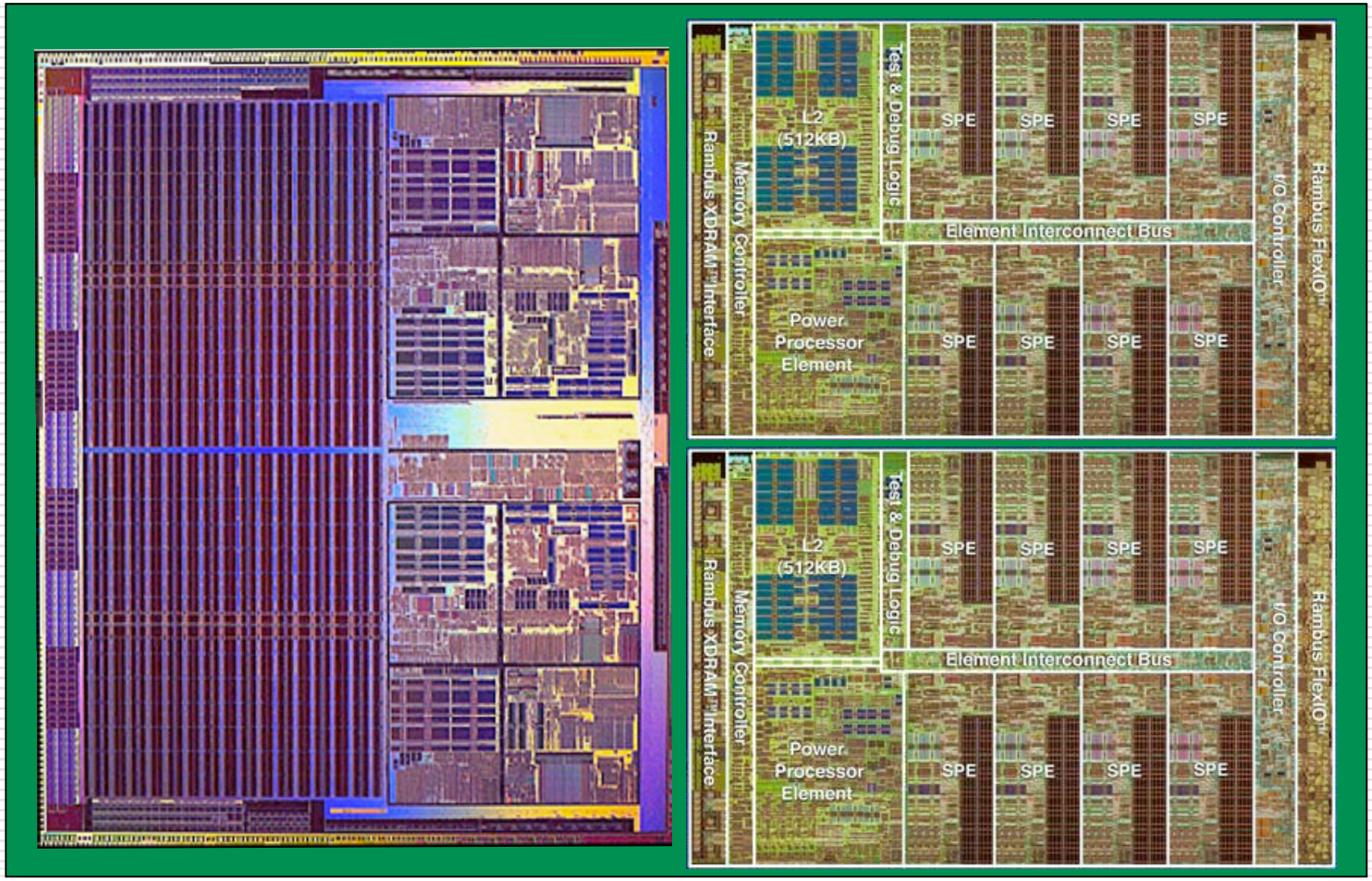
© Larry Snyder, All Rights Reserved

Speed Increase: An Opportunity



© Larry Snyder, All Rights Reserved

Node Structure of RoadRunner

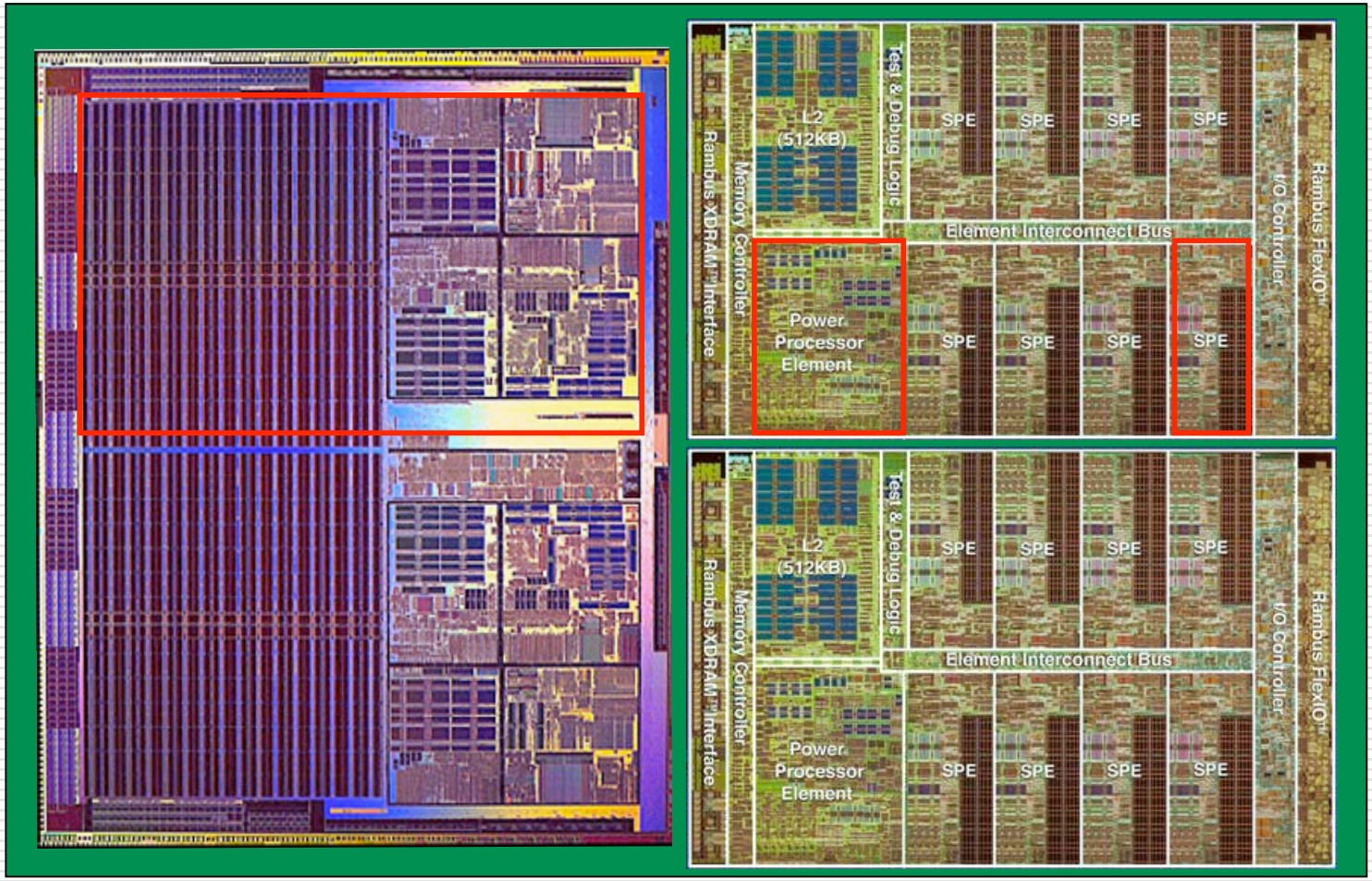


© Larry Snyder, All Rights Reserved

Hybrid Machine

3 ISAs

- **Opteron**
- **PPC**
- **SPE**



© Larry Snyder, All Rights Reserved

Three Programming Levels: Challenge

- Programming Multiple Levels is not simply just 3 compilations
  - SPE is difficult to program -- few HW goodies
  - PPC is mostly orchestrating local data flow
  - Opteron (contributes 3% of performance) mostly orchestrates more global data flow
  - Library support different for each ISA
- LINPAC has very favorable work/word character
- LINPAC benchmark has been developed over many years; writing new HPC programs for this architecture will be time consuming

© Larry Snyder, All Rights Reserved

Interprocessor Communication

- Communication between two arbitrary processors (latency) is a serious problem
- (Not considered today, but in Lecture1)

CMP	AMD	100
SMP	Sun Fire E25K	400-660
Cluster	Itanium + Myrinet	4100-5100
Super	BlueGene/L	5000

Lge range => cannot be ignored

© Larry Snyder, All Rights Reserved

Summary So Far, Large Scale

- Opportunities
  - Moore's law continues
  - Large Scale on track for exa-scale machines by about 2019, though there is much to do
  - The advancement will be significant
    1Kyears --> seconds
- Challenges
  - Hybrid design requires difficult multilevel programming
  - Hardware "lifetimes" are short -- be general
  - Latencies will continue to grow

© Larry Snyder, All Rights Reserved

Outline Of This Talk

- Small Parallelism: today's technology facts
- A Closer Look -- Facts of Life
- Large Parallelism: today's technology facts
- A Closer Look -- Living with Parallelism
- A trivial exercise focuses our attention
- A little "science" can help

© Larry Snyder, All Rights Reserved

Life Times / Development Times

- Hardware -- has a “half-life” measured in years
- Software -- has a “half-life” measured in decades
- Exploiting specific parallel hardware features risks introducing architecture dependences the next platform cannot fulfill
- In parallel programming architecture “shows through” … don’t make the view too clear

© Larry Snyder, All Rights Reserved

Sequential Algorithms No Good Guide

- What makes good sequential and parallel algorithms are different
  - Resources
    - S: Reduce instruction count
    - P: Reduce communication
  - Best practices
    - S: Manipulate references, exploit indirection
    - P: Reduce dependences, avoid interaction
  - Look for algorithms with
    - S: Efficient data structures
    - P: Locality, locality, locality

© Larry Snyder, All Rights Reserved

Good Algorithms Not Well Known

- What is the best way to multiply two dense matrices in parallel? **Ans: In Lecture 1**
- We all know good serial algorithms and sequential programming techniques
- Parallel techniques not widely known, and because they are different from sequential techniques, should we be teaching them to Freshmen in college?

© Larry Snyder, All Rights Reserved

Outline Of This Talk

- Small Parallelism: today's technology facts
- A Closer Look -- Facts of Life
- Large Parallelism: today's technology facts
- A Closer Look -- Living with Parallelism
- A trivial exercise focuses our attention
- A little "science" can help

© Larry Snyder, All Rights Reserved

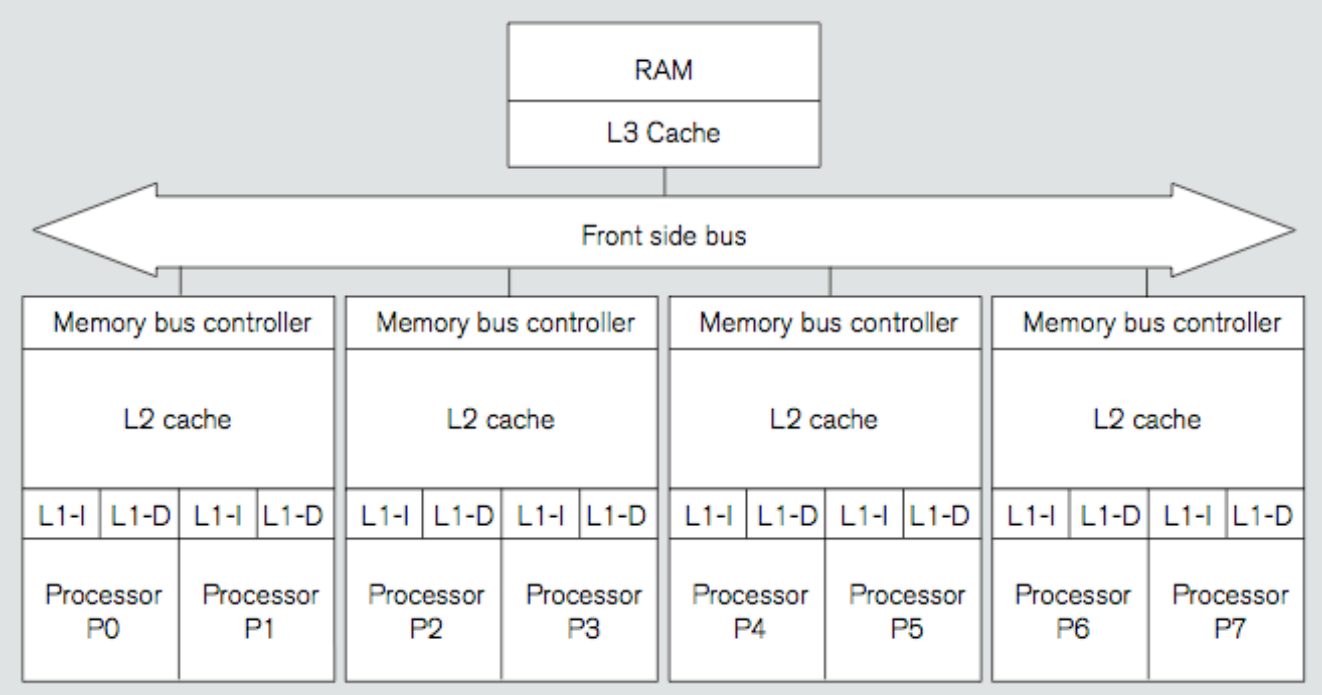
Programmer Challenge: Education

- Most college CS graduates have
  - No experience writing parallel programs
  - No knowledge of the issues, beyond concurrency from OS classes
  - Little concept of standard parallel algorithms
  - No model for what makes a parallel algorithm good
- Where do the programmers come from?

© Larry Snyder, All Rights Reserved

Illustrating the Issues

- Consider the trivial problem of counting the number of 3s in a vector of values on m-c

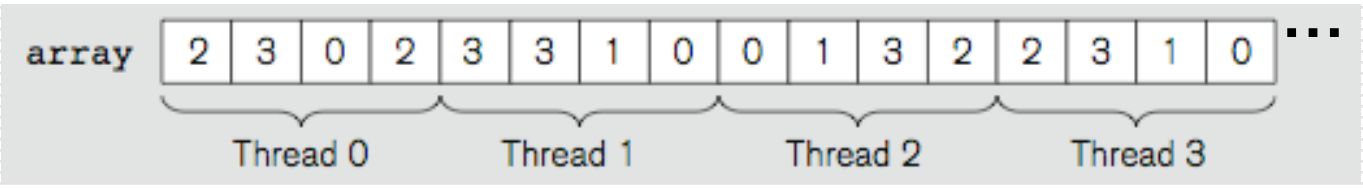


© Larry Snyder, All Rights Reserved

Try # 1

- Assume array in shared memory; 8 way ||-ism

```
void count3s() {
   int count=0; int i=0; /* Create t threads */
   for (i=0;i<t;i++){
      thread_create(count3s_thread,i);
   }
   return count;
}
void count3s_thread(int id){
   int j;
   int length_per_thread=length/t;
   int start=id*length_per_thread;
   for (j=start; j<start+length_per_thread;j++) {
      if (array[j]==3)
         count++;
   }
}
```



© Larry Snyder, All Rights Reserved

Try #1 Assessment

- Try #1 doesn't even get the right answer!
- The count variable is not protected, so there is a race as threads try to increment it

Thread i

```
...
lw   $8,count-off(gp)
addi $8,1
sw   $8,count-off(gp)
...
```

© Larry Snyder, All Rights Reserved

Try #2

- Protect the shared count with a mutex lock

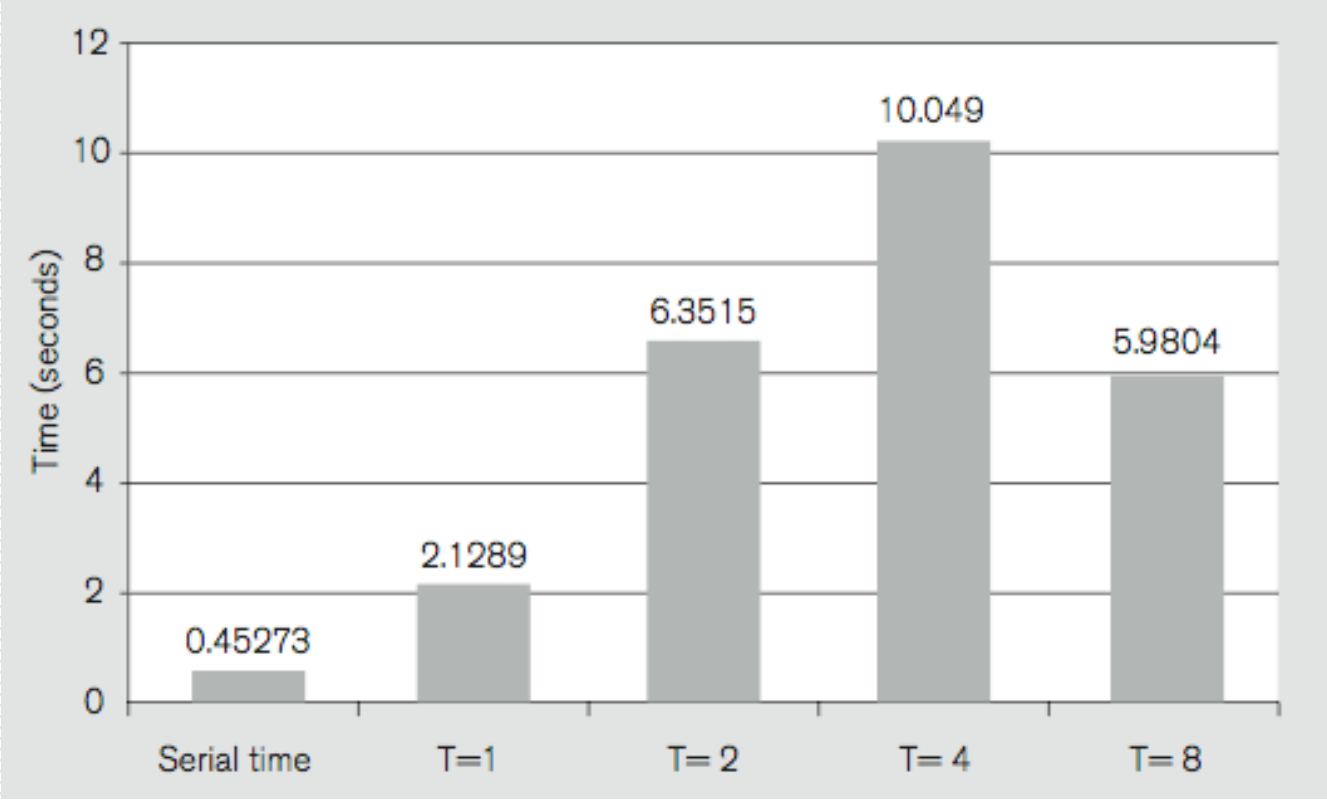
```
void count3s_thread(int id){
   int j;
   int length_per_thread=length/t;
   int start=id*length_per_thread;
   for (j=start; j<start+length_per_thread;j++) {
      if (array[j]==3) {
         mutex_lock(m);
         count++;
         mutex_unlock(m);
      }
   }
}
```

- This solution at least gets the right answer

© Larry Snyder, All Rights Reserved

Try #2 Assessment

- It doesn't perform, however



© Larry Snyder, All Rights Reserved

Try #3

- Include a private variable

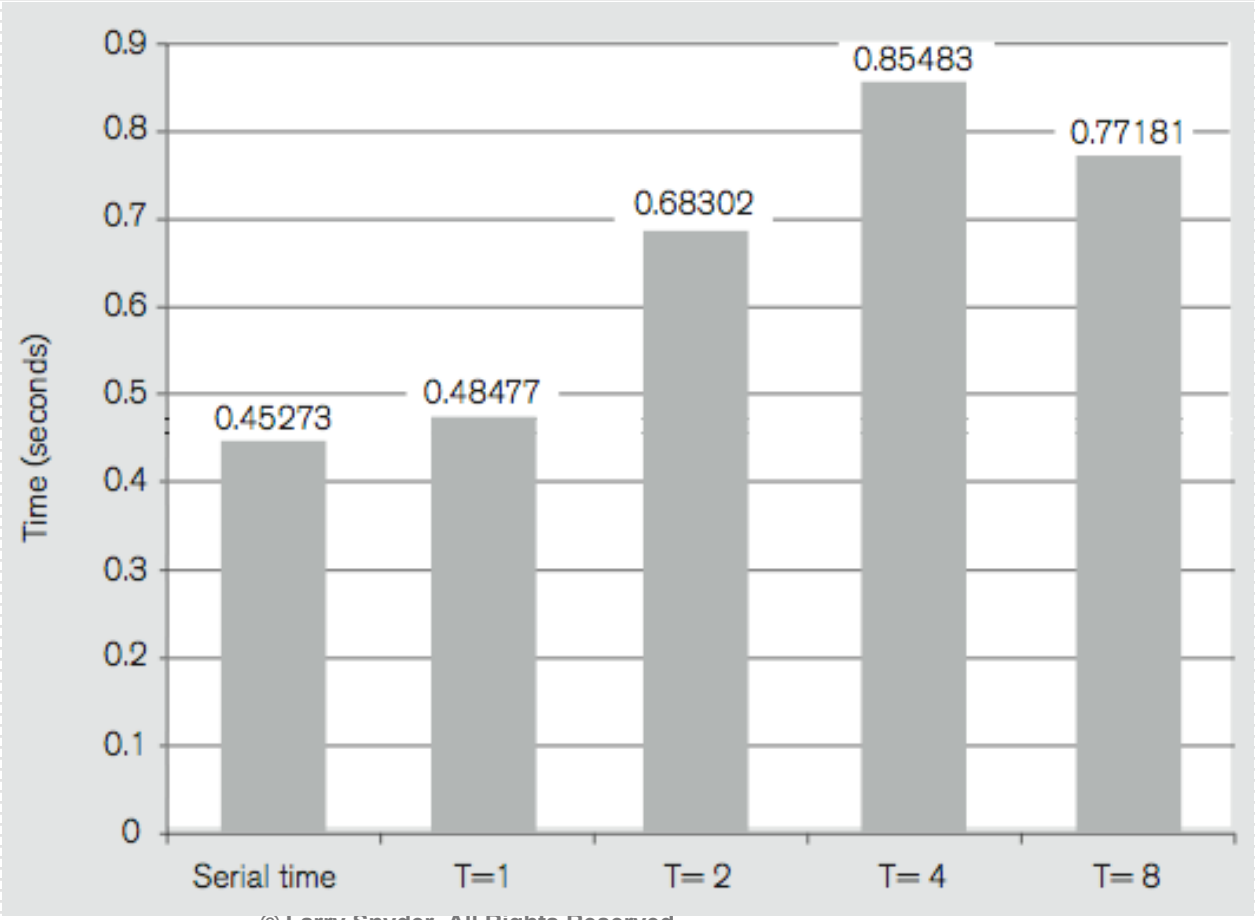
```
void count3s_thread(int id){
   int j;
   int length_per_thread=length/t;
   int start=id*length_per_thread;
   for (j=start; j<start+length_per_thread;j++) {
      if (array[j]==3) {
         private_count[id]++;
      }
   }
   mutex_lock(m);
   count += private_count[id];
   mutex_unlock(m);
}
```

- Contention, if it happens is limited

© Larry Snyder, All Rights Reserved

Try #3 Assessment

- The performance got better, but still no ||-ism



© Larry Snyder, All Rights Reserved

Try #3 False Sharing

- The private variables were allocated to one or two cache lines

- Cache coherency schemes, which keep everyone's cache current operate on the cache-line granularity … still contention
  - Suppose two processors have the cache line
  - When one writes, other is invalidated; refetch

© Larry Snyder, All Rights Reserved

Try #4

- By simply adding padding to give each private count its own cache line Try #3 works

```
struct padded_int {
  int val;
  char padding [60];
} private_count[t];
```

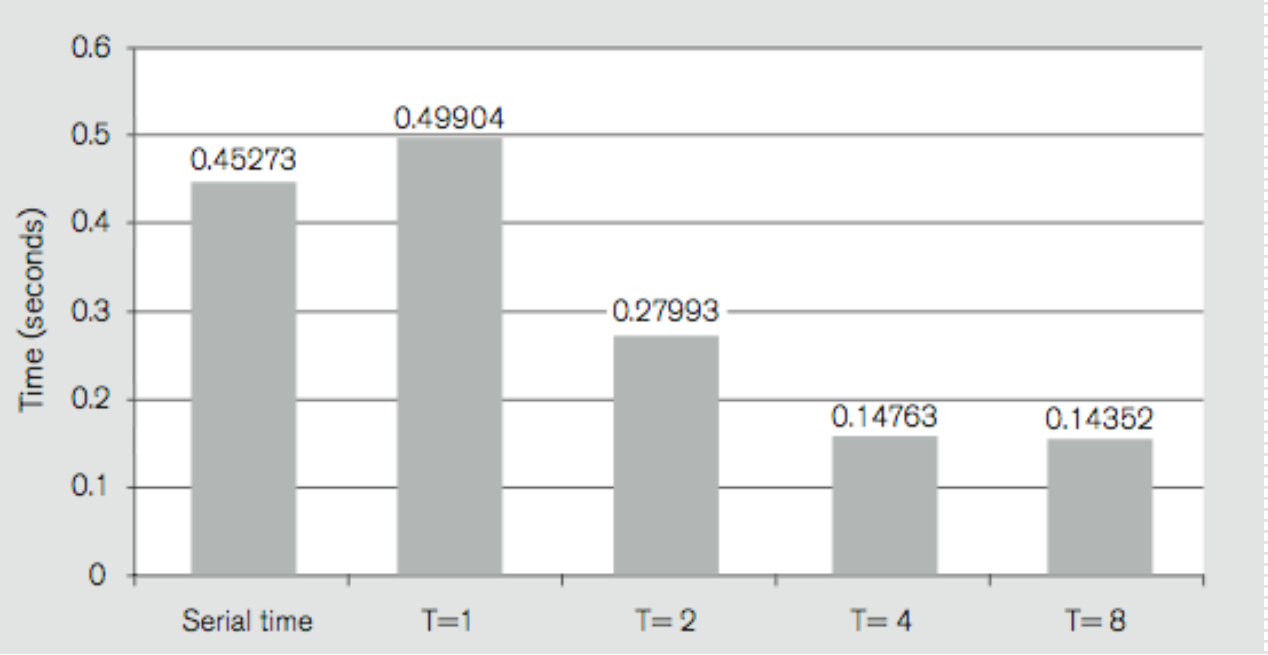
- Notice that false sharing is a sensitivity hardware dependent on f, the cache line size
- Machine features “show through”

© Larry Snyder, All Rights Reserved

Try #4 Assessment

- Finally, speed-up over serial computation

1 processor : 0.91
2 processors: 1.6
4 processors: 3.1
8 processors: 3.2



- It wasn't so easy, and it wasn't so great

© Larry Snyder, All Rights Reserved

Programming Challenges

- Today's programmers not ||-programmers
- Much to worry about
  - Standard abstractions (locks, etc.) too low level
  - Memory Model (mentioned before) busted
  - Parallelism "shows through"
  - Hardware sensitivity (false sharing)
  - Heavy intellectual investment
    - Small task took serious effort
    - Modest performance achieved
    - Not yet a general solution

Houston, We Have Problem

© Larry Snyder, All Rights Reserved

Outline Of This Talk

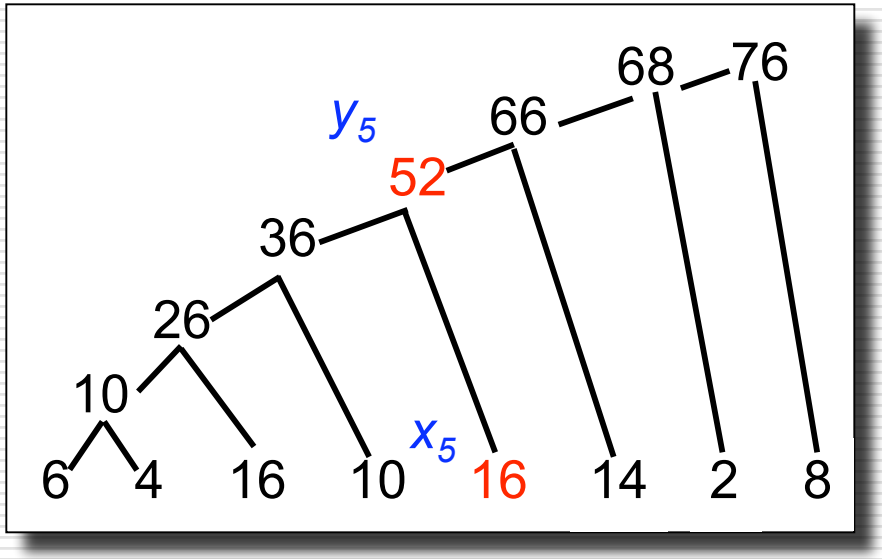
- Small Parallelism: today's technology facts
- A Closer Look -- Facts of Life
- Large Parallelism: today's technology facts
- A Closer Look -- Living with Parallelism
- A trivial exercise focuses our attention
- A little "science" can help

© Larry Snyder, All Rights Reserved

Overcoming Sequential Control

- Many computations on a data sequence seem to be "essentially sequential"
- Prefix sum is an example: for n inputs, the i^{th} output is the sum of the first i items
  - Input: 2 1 5 3 7
  - Output: 2 3 8 11 18
- Given $x_1, x_2, \ldots, x_n$ find $y_1, y_2, \ldots, y_n$ s.t.

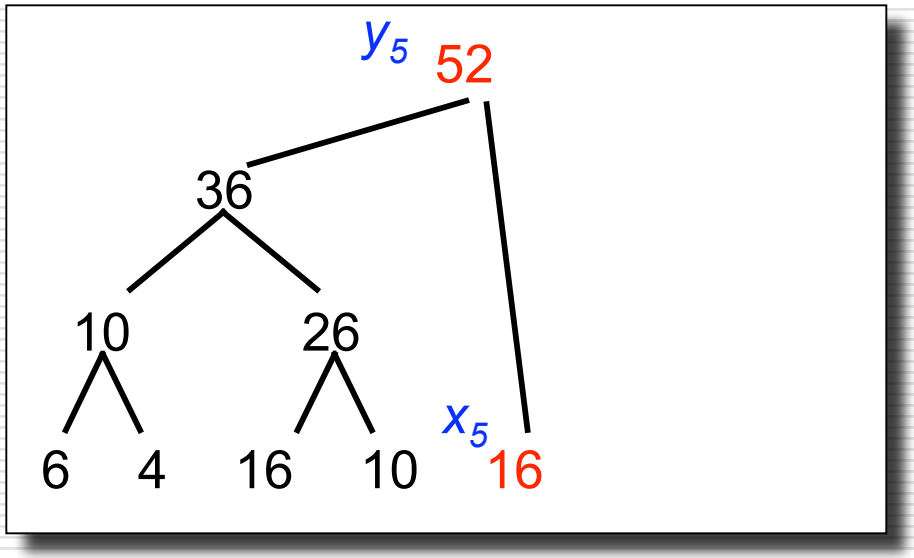
$$y_i = \sum_{j \le i} x_j$$



© Larry Snyder, All Rights Reserved

Naïve Use of Parallelism

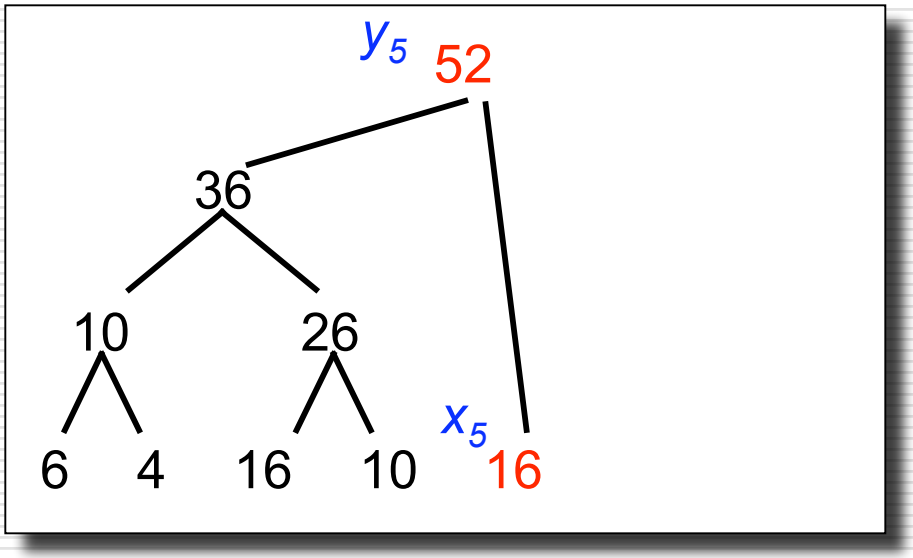
- For any y_i a height *log i* tree finds the prefix: find it, add x_i
  - Much redundant computation
  - Requires O(n^2) parallelism for n prefixes
- Look closer at meaning of tree's intermediate sums

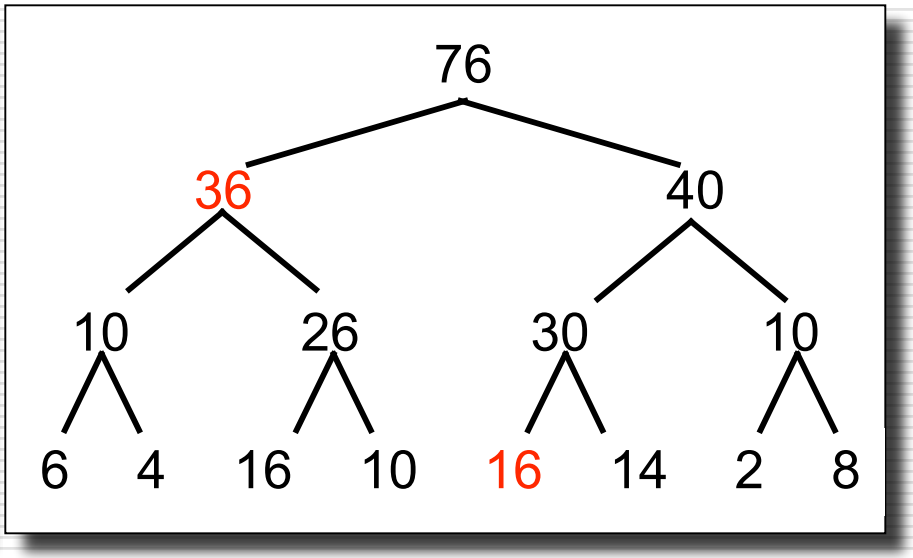


© Larry Snyder, All Rights Reserved

Naïve Use of Parallelism

- For any y_i a height *log i* tree finds the prefix: find it, add x_i
  - Much redundant computation
  - Requires O(n^2) parallelism for n prefixes
- Look closer at meaning of tree's intermediate sums

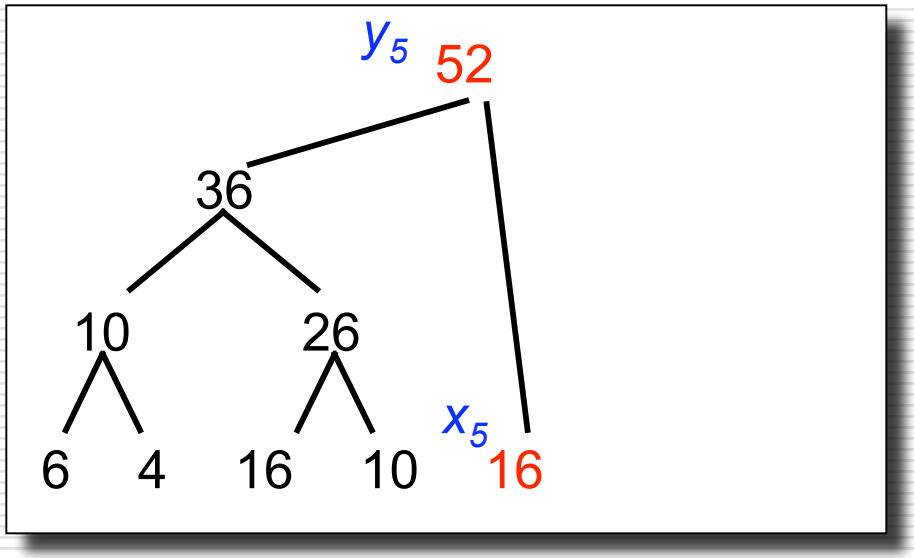


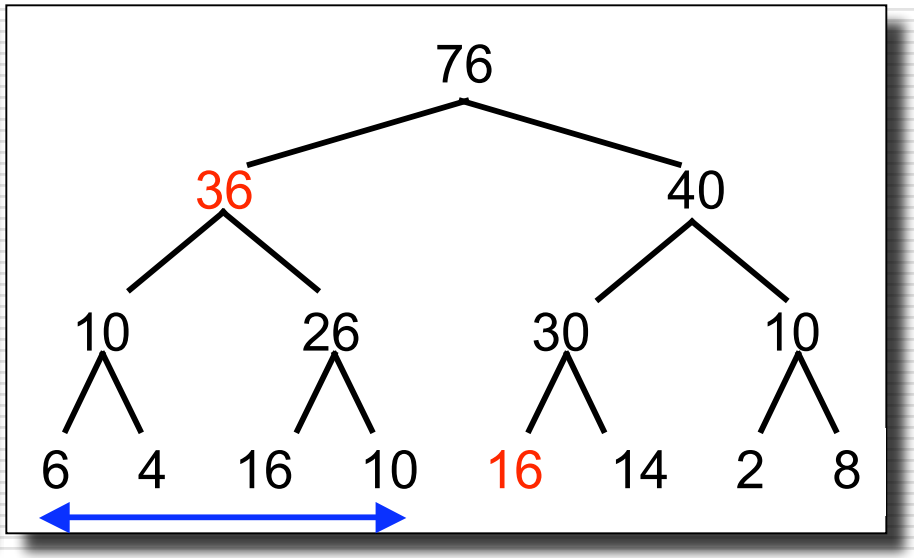


© Larry Snyder, All Rights Reserved

Naïve Use of Parallelism

- For any y_i a height *log i* tree finds the prefix: find it, add x_i
  - Much redundant computation
  - Requires O(n^2) parallelism for n prefixes
- Look closer at meaning of tree's intermediate sums





root summarizes its leaves

© Larry Snyder, All Rights Reserved

Speeding Up Prefix Calculations

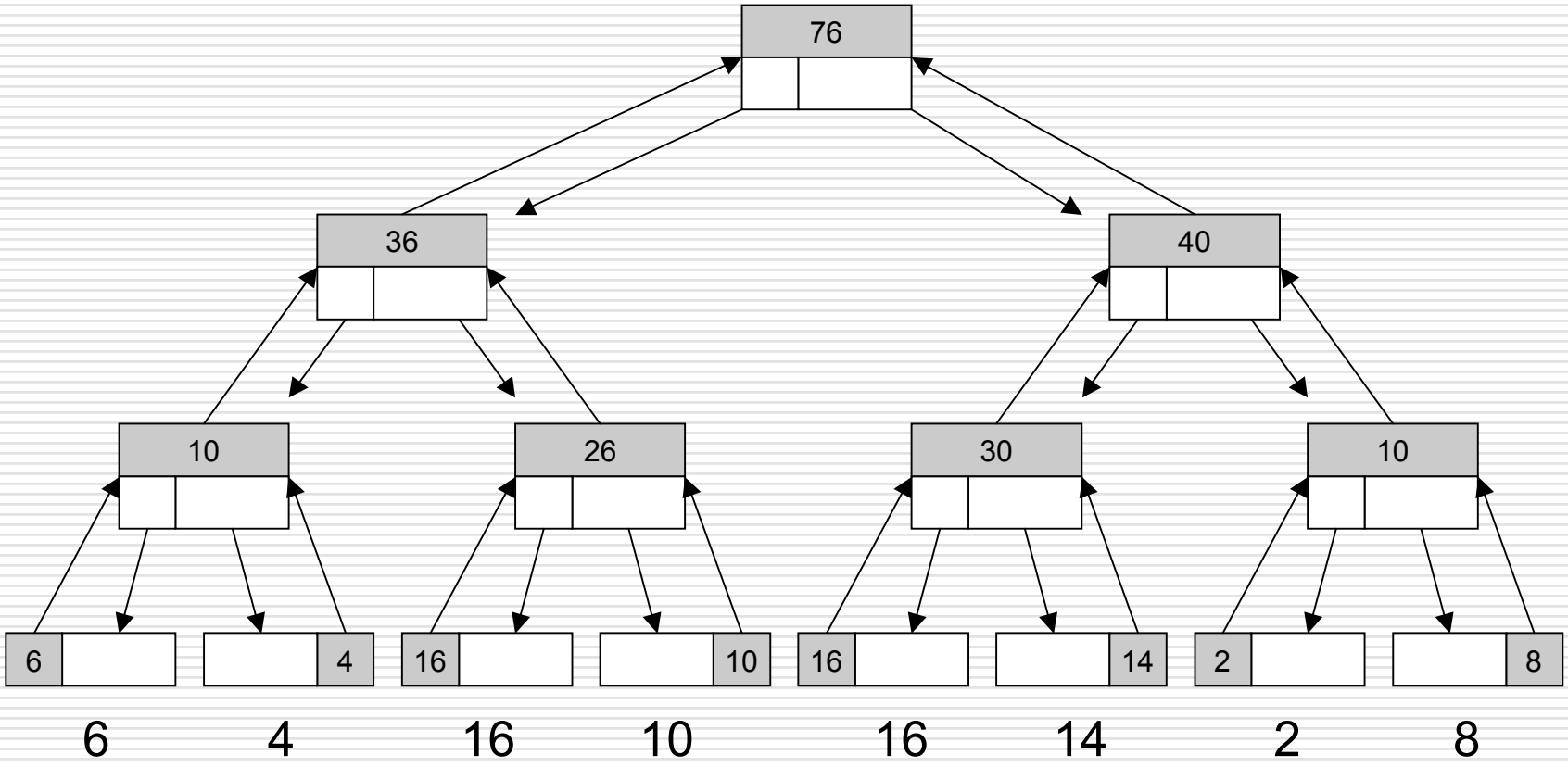
- Putting the observations together
  - One pass over the data computes global sum
  - Intermediate values are saved
  - A second pass over data uses intermediate sums to compute prefixes
  - Each pass will be logarithmic for $n = P$
  - Solution is called: The *parallel prefix algorithm*

R. E. Ladner and M. J. Fischer
Parallel Prefix Computation
Journal of the ACM 27(4):831-838, 1980

© Larry Snyder, All Rights Reserved

Parallel Prefix Algorithm

Compute sum going up

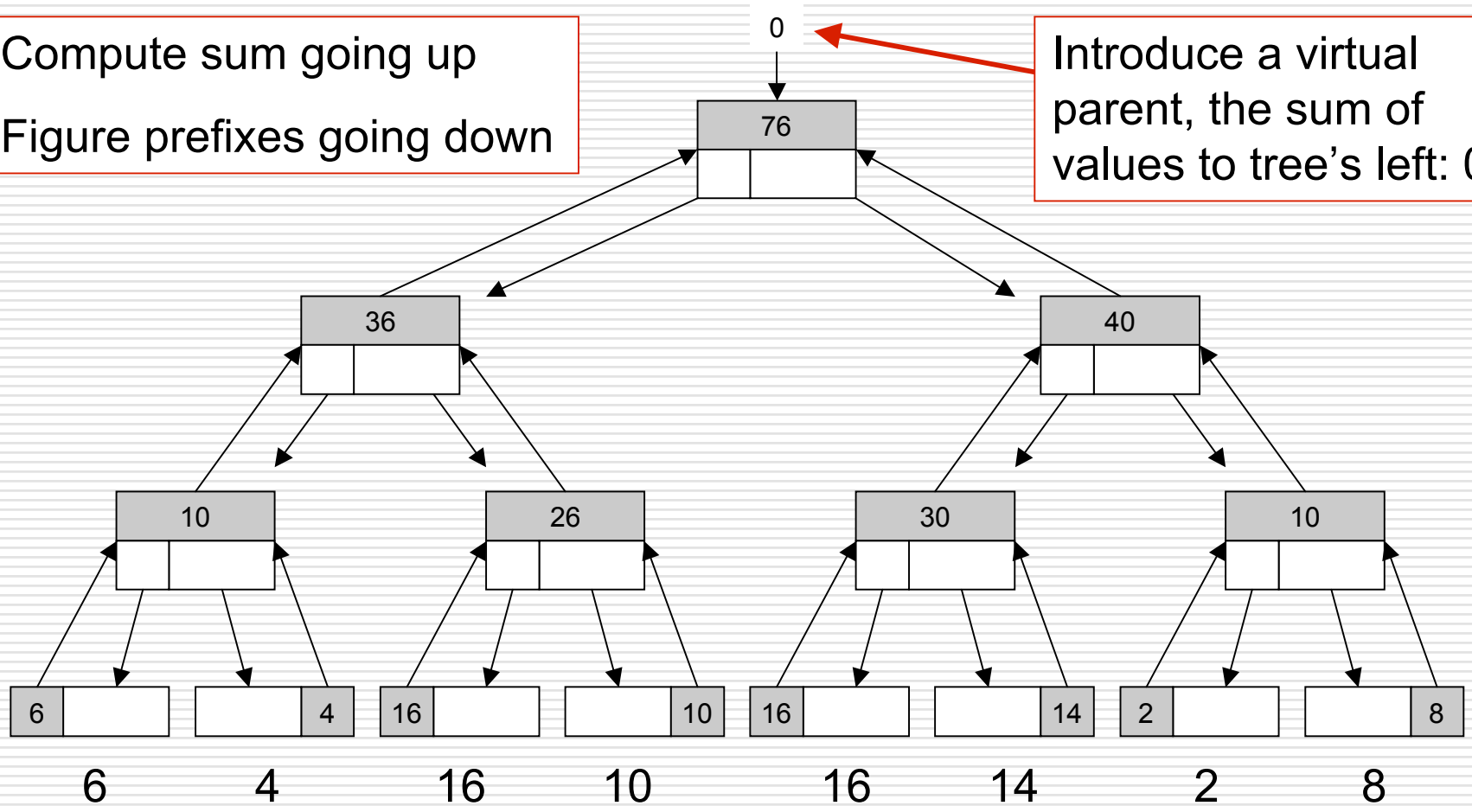


© Larry Snyder, All Rights Reserved

Parallel Prefix Algorithm

Compute sum going up

Figure prefixes going down



Introduce a virtual parent, the sum of values to tree's left: 0

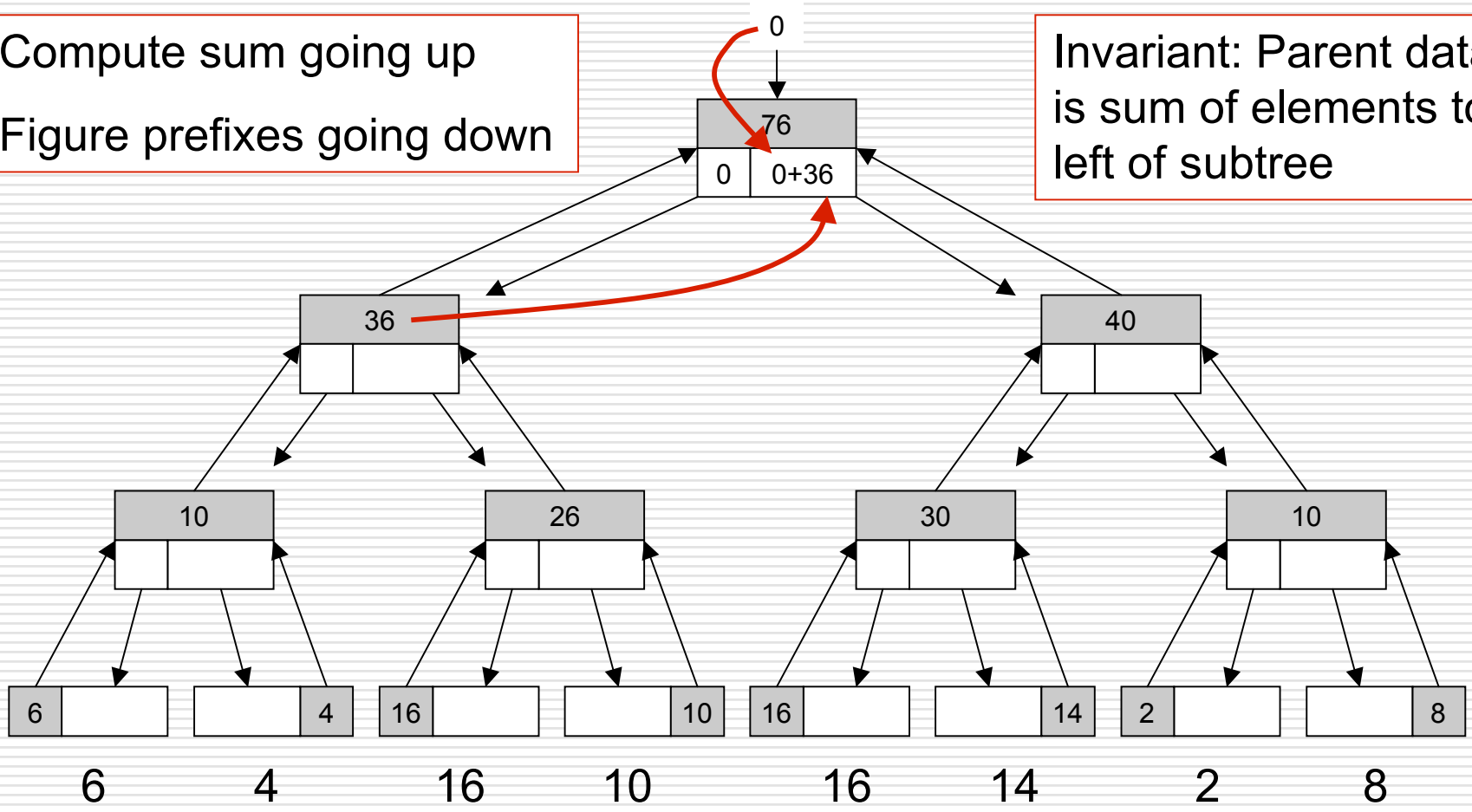
© Larry Snyder, All Rights Reserved

Parallel Prefix Algorithm

Compute sum going up

Figure prefixes going down

Invariant: Parent data is sum of elements to left of subtree

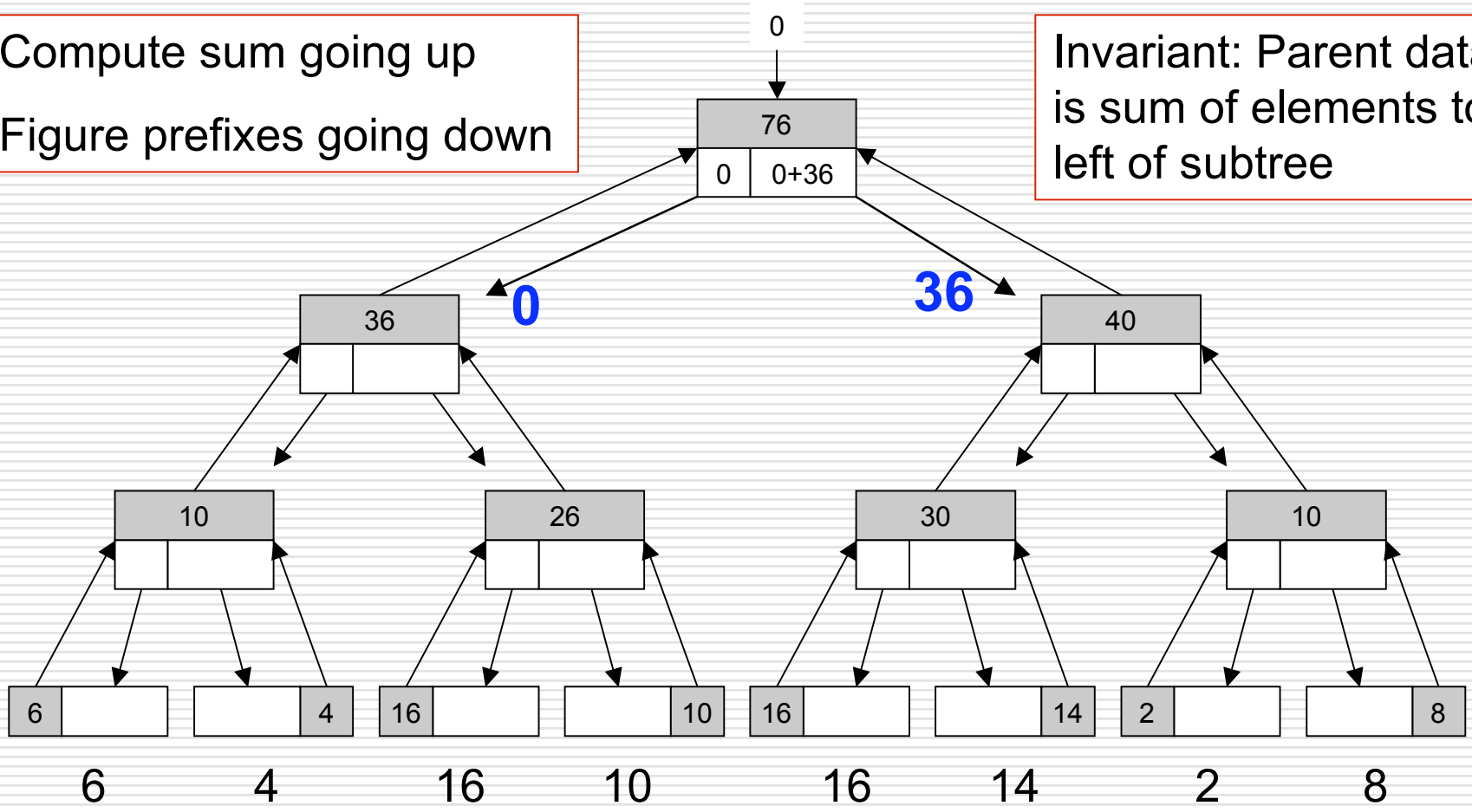


© Larry Snyder, All Rights Reserved

Parallel Prefix Algorithm

Compute sum going up

Figure prefixes going down



Invariant: Parent data is sum of elements to left of subtree

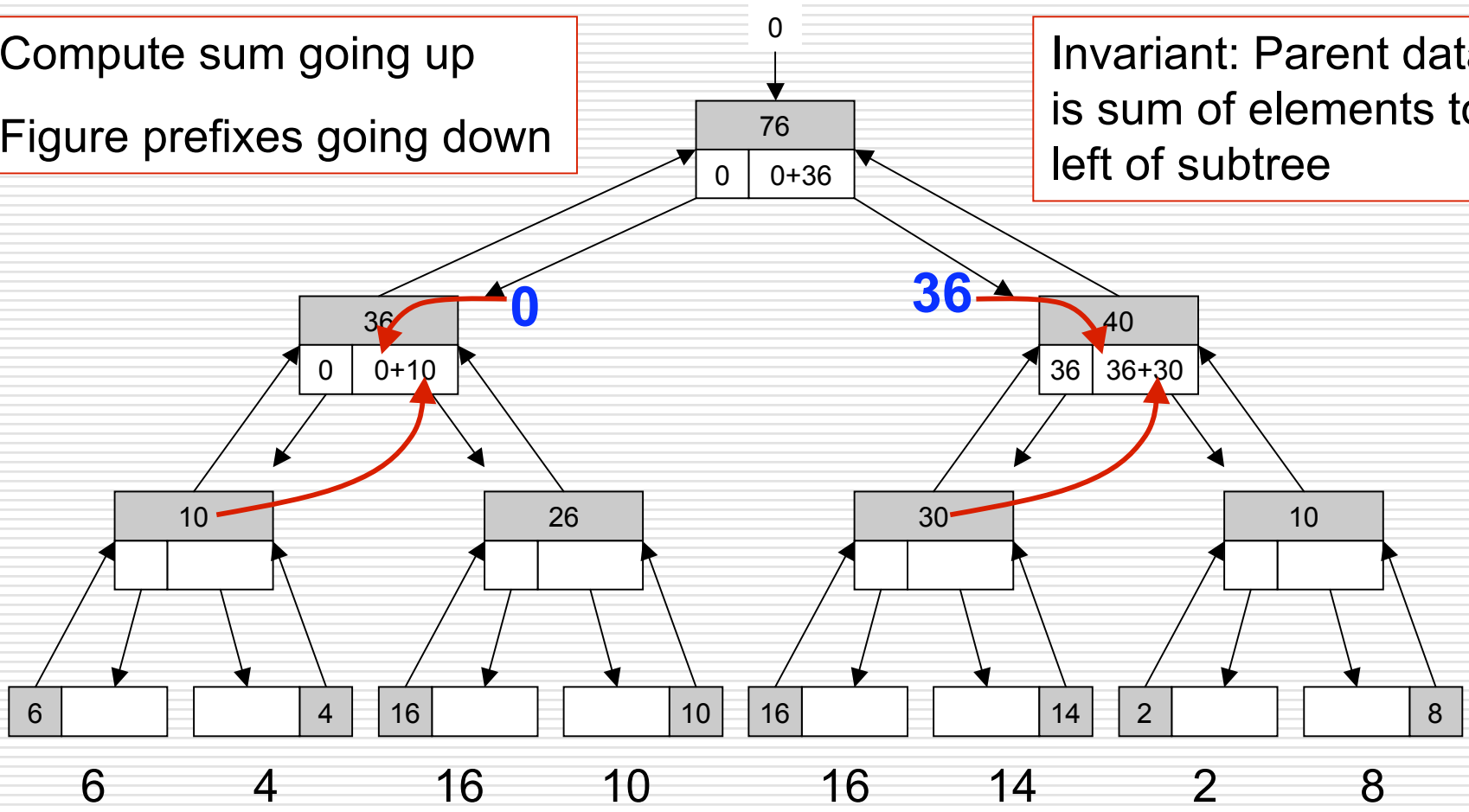
© Larry Snyder, All Rights Reserved

Parallel Prefix Algorithm

Compute sum going up

Figure prefixes going down

Invariant: Parent data is sum of elements to left of subtree

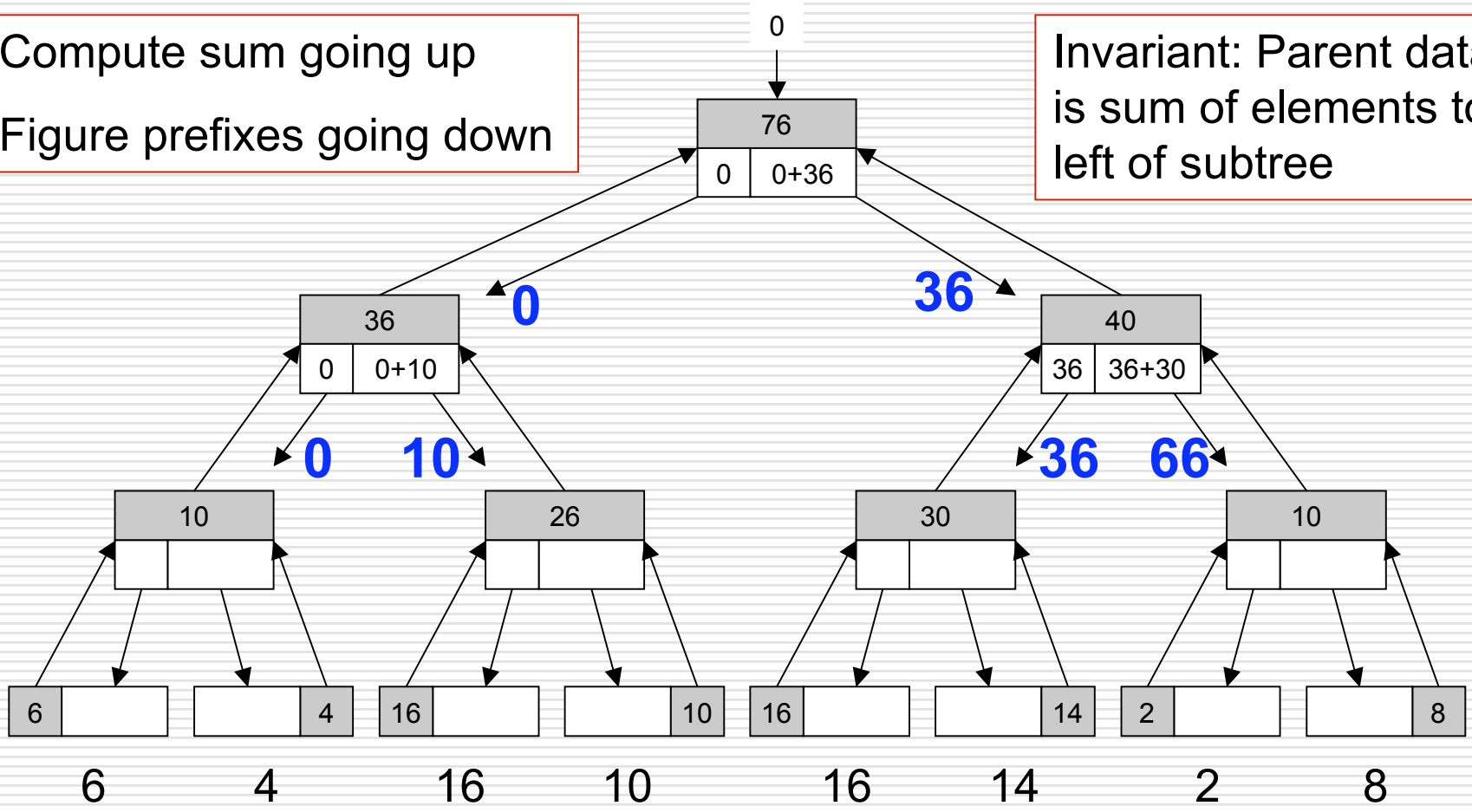


© Larry Snyder, All Rights Reserved

Parallel Prefix Algorithm

Compute sum going up

Figure prefixes going down



Invariant: Parent data is sum of elements to left of subtree

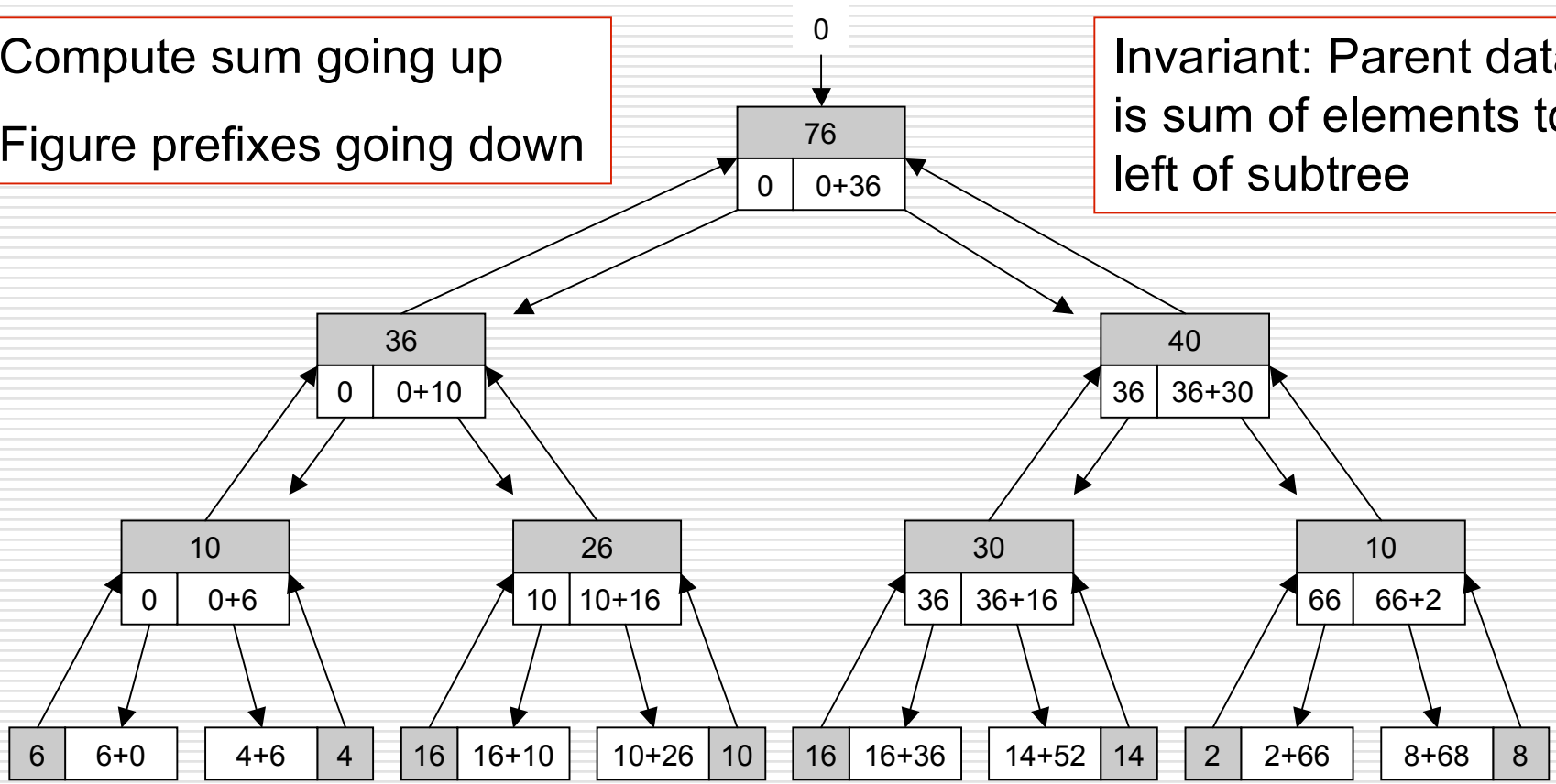
© Larry Snyder, All Rights Reserved

Parallel Prefix Algorithm

Compute sum going up

Figure prefixes going down

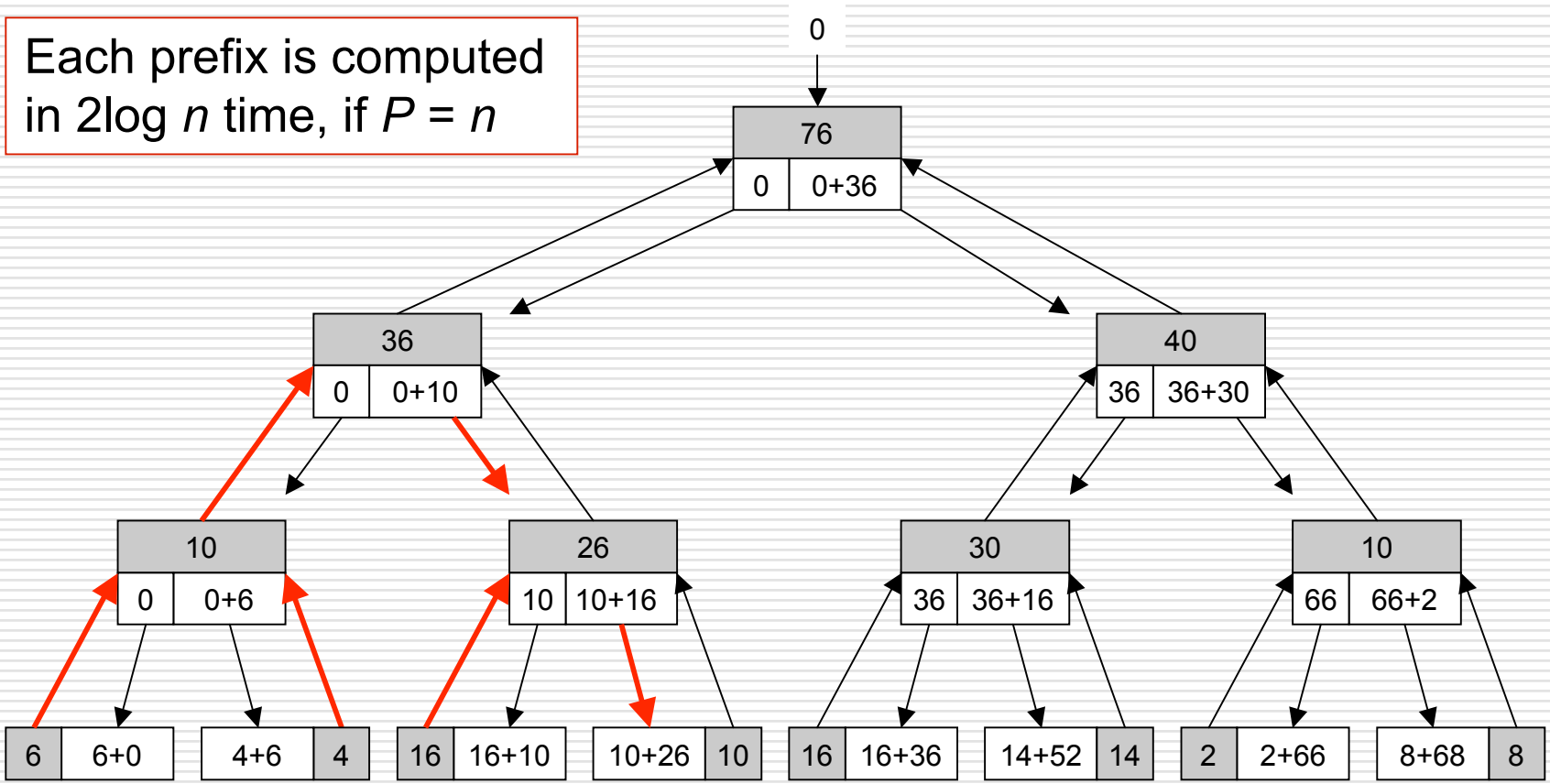
Invariant: Parent data is sum of elements to left of subtree



© Larry Snyder, All Rights Reserved

Parallel Prefix Algorithm

Each prefix is computed in 2log *n* time, if *P* = *n*



© Larry Snyder, All Rights Reserved

Available || Prefix Operators

- Most languages have reduce and scan (|| prefix) built-in for: `+`, `*`, `min`, `max`, `&&`, `||`
- A few languages allow users to define || prefix operations themselves … do they?
- Parallel prefix is MUCH more useful

© Larry Snyder, All Rights Reserved

Available || Prefix Operators

- Most languages have reduce and scan (|| prefix) built-in for: `+`, `*`, `min`, `max`, `&&`, `||`
- A few languages allow users to define || prefix operations themselves … do they?
- Parallel prefix is MUCH more useful

- Length of Longest Run of x
- Number of Occurrences of x
- Histogram
- Mode and Average
- Count Words
- Length of Longest Increasing Run
- Binary String Space Compression
- Run Length Encoding
- Balanced Parentheses
- Skyline

Why is there no standard programming abstraction?

© Larry Snyder, All Rights Reserved

Conclusion +

- Most computers sold today are ||, and nearly all are under-utilized
  - No silver bullet will save us (Lecture 1)
  - Languages, tools, libraries are not ready
- New software is needed to exploit ||ism
  - Naïve parallel programming is difficult, subtle
  - Higher abstractions help (Lecture 1 & 2)
- The “payoff” is to keep riding performance wave

© Larry Snyder, All Rights Reserved

Homework

HW: Use ||-prefix to test for balanced parentheses: ~~(((()))()~~

Further Questions?

© Larry Snyder, All Rights Reserved