Parallel Programming

The preferred parallel algorithm is generally different from the preferred sequential algorithm

- Compilers cannot transform a sequential algorithm into a parallel one with adequate consistency
- Legacy code must be rewritten to use ||ism
- Your knowledge of sequential algorithms is not that useful for parallel programming
- There is no silver bullet

1

Easy Cases: Data Parallelism

Iteration body for a given index is independent of all others … can be performed in parallel

```
void
array_add(int A[], int B[], int C[], int length) {
  int i;
  for (i = 0 ; i < length ; ++ i) {
   C[i] = A[i] + B[i];
  }
}
```

The standard programming abstraction would be

```
for_all i in [0..length-1]{C[i]=A[i]+ B[i];}
```

2

Is it always that easy?

Not always… a more challenging example:

```
unsigned
sum_array(unsigned *array, int length) {
  int total = 0;
  for (int i = 0 ; i < length ; ++ i) {
         total += array[i];
  }
  return total;
}
```

Is there parallelism here?

3

We first need to restructure the code

```
unsigned
sum_array2(unsigned *array, int length) {
  unsigned total, i;
  unsigned temp[4] = {0, 0, 0, 0};
  for (i = 0 ; i < length & ~0x3 ; i += 4) {
    temp[0] += array[i];
    temp[1] += array[i+1];
    temp[2] += array[i+2];
    temp[3] += array[i+3];
  }
  total = temp[0] + temp[1] + temp[2] + temp[3];
  return total;
}
```

4

Then generate SIMD code for hot part

SIMD == Single Instruction, Multiple Data

```
unsigned
sum_array2(unsigned *array, int length) {
  unsigned total, i;
  unsigned temp[4] = {0, 0, 0, 0};
  for (i = 0 ; i < length & ~0x3 ; i += 4) {
    temp[0] += array[i];
    temp[1] += array[i+1];
    temp[2] += array[i+2];
    temp[3] += array[i+3];
  }
  total = temp[0] + temp[1] + temp[2] + temp[3];
  return total;
}
```

5

Intel SSE/SSE2 as an example of SIMD

SSE == X86 Streaming SIMD Extensions

- Added new 128 bit registers (XMM0 – XMM7), each can store
  - ❖ 4 single precision FP values (SSE) 4 * 32b
  - ❖ 2 double precision FP values (SSE2) 2 * 64b
  - ❖ 16 byte values (SSE2) 16 * 8b
  - ❖ 8 word values (SSE2) 8 * 16b
  - ❖ 4 double word values (SSE2) 4 * 32b
  - ❖ 1 128-bit integer value (SSE2) 1 * 128b

	4.0 (32 bits)	4.0 (32 bits)	3.5 (32 bits)	−2.0 (32 bits)
+	−1.5 (32 bits)	2.0 (32 bits)	1.7 (32 bits)	2.3 (32 bits)
	2.5 (32 bits)	6.0 (32 bits)	5.2 (32 bits)	0.3 (32 bits)

6

Many Computations Have Dependences

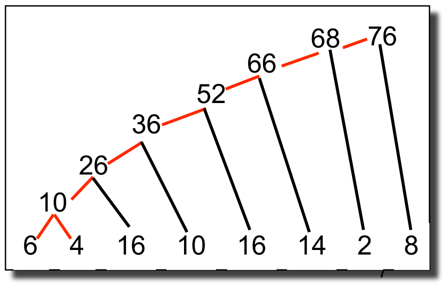
Aggregate or Reduction Operations

```
unsigned
sum_array(unsigned *array, int length) {
  int total = 0;
  for (int i = 0 ; i < length ; ++ i) {
        total += array[i];
  }
  return total;
}
```

Standard abstraction is

```
  total = sum(array);
```

which allows ||-solution



Overcoming Sequential Control

Many computations on a data sequence seem to be "essentially sequential"

Prefix sum is an example: for *n* inputs, the *i*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.

$$y_i = \Sigma_{j \le i}\, x_j$$

Sequential Computation

Consider computing the prefix sums

```
for (i=1; i<n+1; i++) {
  A[i] += A[i-1];
}
```

A[i] is the sum of the first i elements

Semantics ...

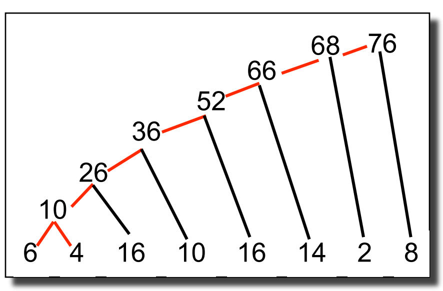
- A[1] is unchanged
- A[2] = A[2] + A[1]
- A[3] = A[3] + (A[2] + A[1])

  ...
- A[n] = A[n] + (A[n-1] + (... (A[2] + A[1]) …)

What advantage can ||ism give?

Illustrating The Semantics

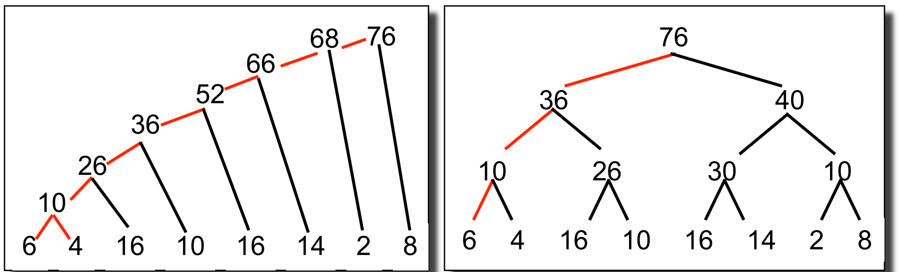
The computation of the items can be described pictorially
The picture illustrates the dependences of the sequential code



Addition, of course, is associative

Restructuring the Computation

Express the computation as a tree

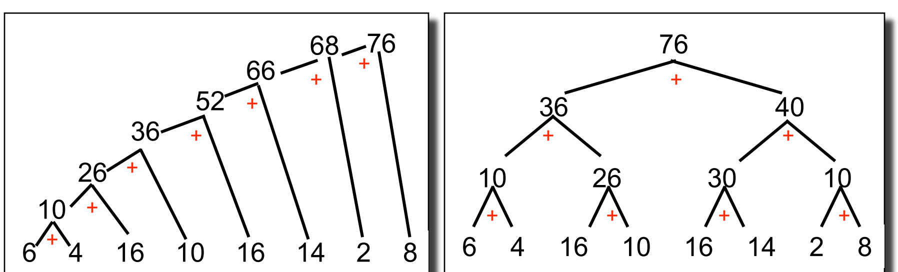


Dependence chain shallower -- faster

- ❖Sequential: 7, Tree: 3

Restructuring the Computation

Express the computation as a tree



Dependence chain shallower -- faster

- ❖Sequential: 7, Tree: 3

Operation count is unchanged: 7 each

Naïve Use of Parallelism

For any y_i a height *log i* tree finds the prefix

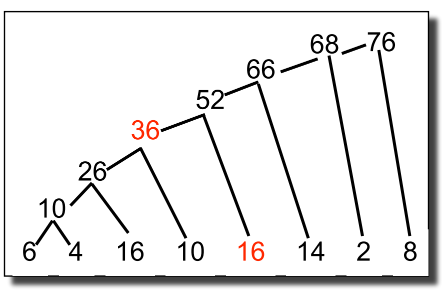
- ❖ Much redundant computation
- ❖ Requires $O(n^2)$ parallelism for *n* prefixes
- ❖ It may be parallel but it is unrealistic

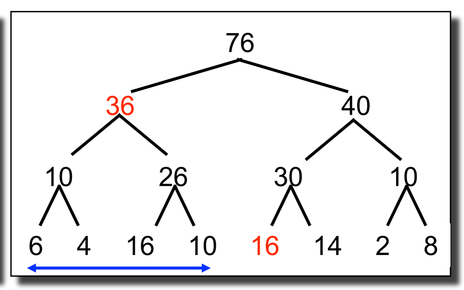
Naïve Use of Parallelism

For any y_i a height *log i* tree finds the prefix

- ❖ Much redundant computation
- ❖ Requires $O(n^2)$ parallelism for *n* prefixes

Look closer at meaning of tree's intermediate sums





root summarizes its leaves

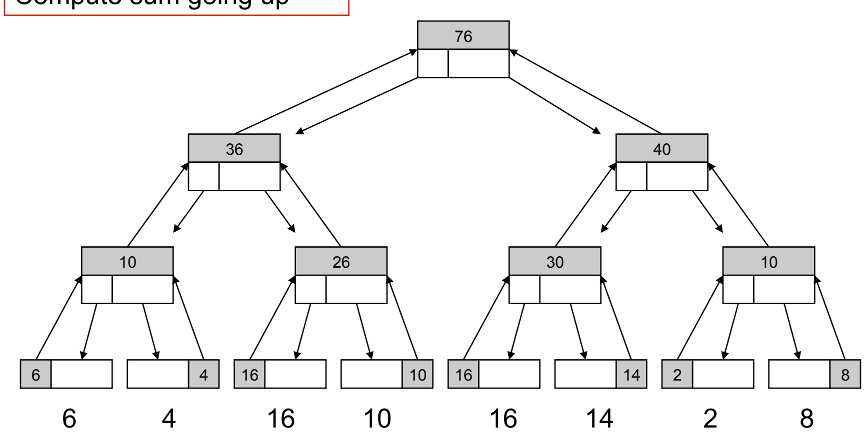
Speeding Up Prefix Calculations

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*

Parallel Prefix Algorithm

Compute sum going up

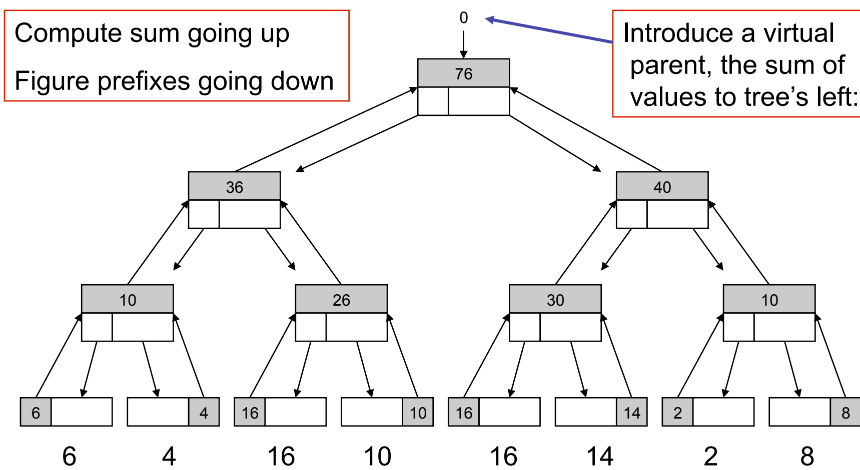


Parallel Prefix Algorithm

Compute sum going up

Figure prefixes going down

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

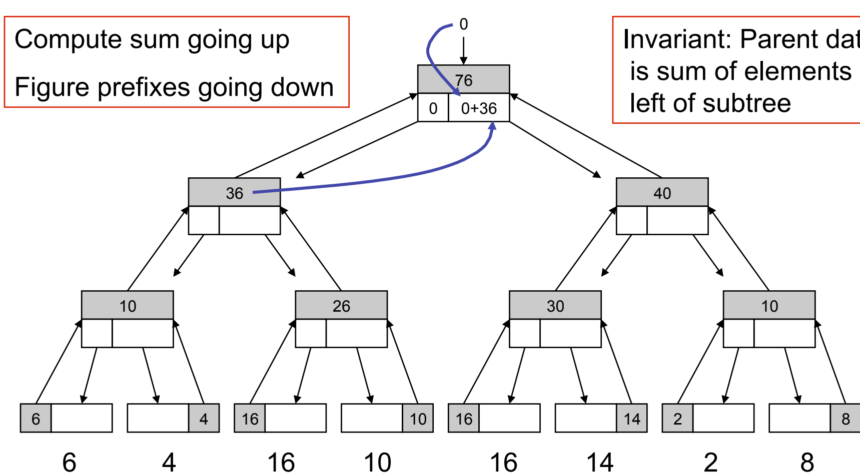


Parallel Prefix Algorithm

Compute sum going up

Figure prefixes going down

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

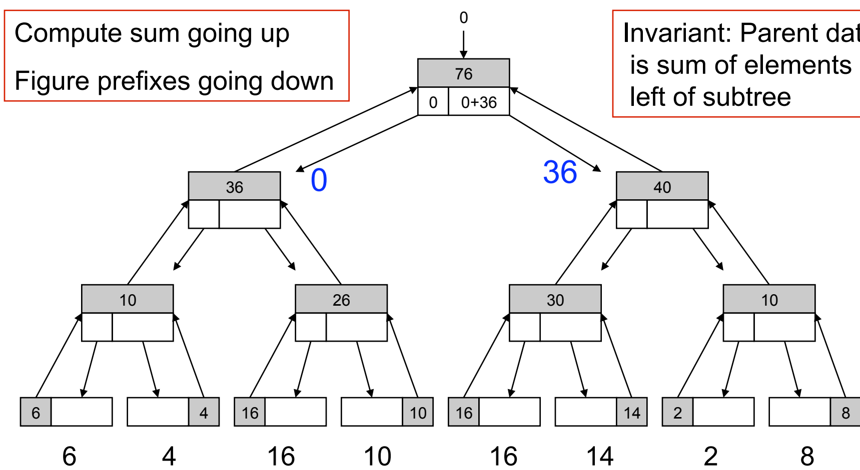


Parallel Prefix Algorithm

Compute sum going up

Figure prefixes going down

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

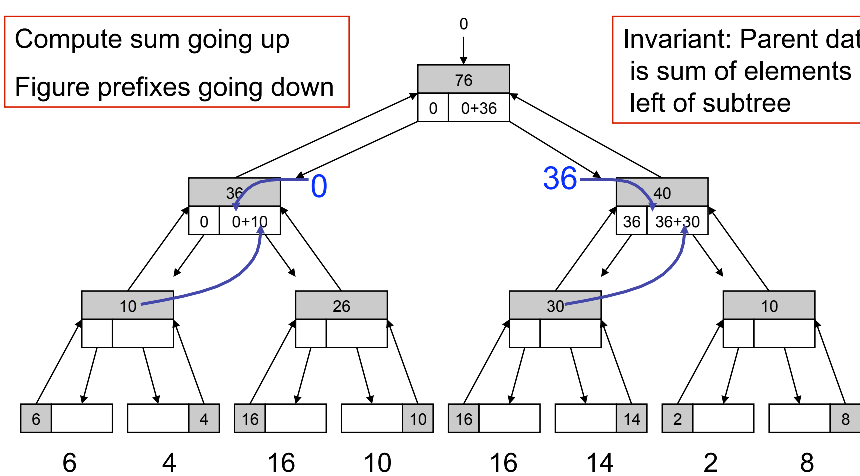


Parallel Prefix Algorithm

Compute sum going up

Figure prefixes going down

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

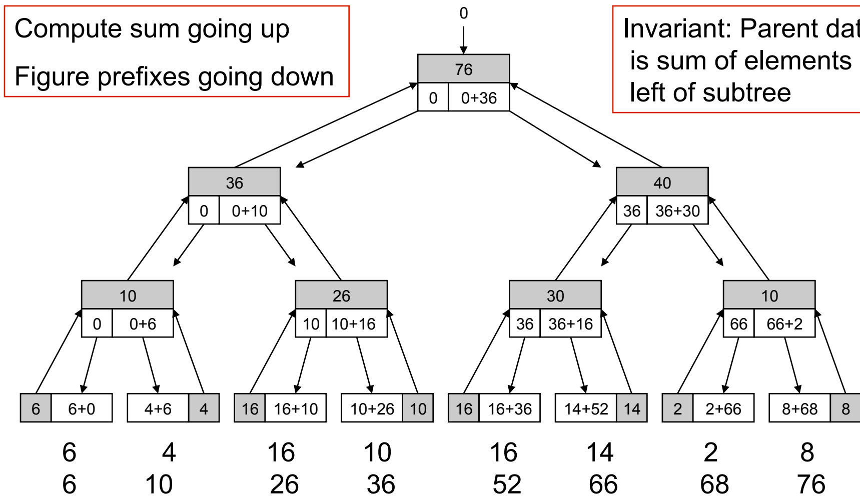


Parallel Prefix Algorithm

Compute sum going up

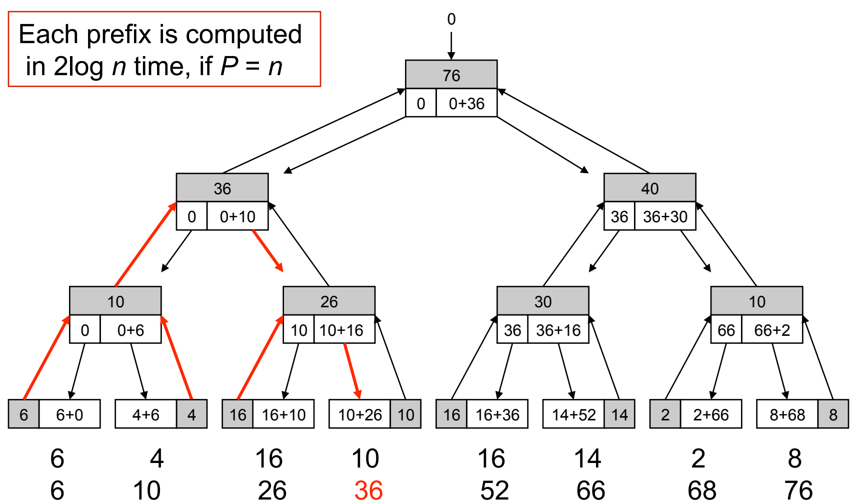
Figure prefixes going down

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



Parallel Prefix Algorithm

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



Fundamental Tool of || Pgmming

Original research on parallel prefix algorithm published by

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

The Ladner-Fischer algorithm requires *2log n* time, twice as much as simple tournament global sum, not linear time

Applies to a wide class of operations

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

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

Summary

Sequential computation is a special case of parallel computation (P==1)

Generalizing from sequential computations usually arrives at the wrong solution … rethinking the problem to develop a parallel algorithm is the only real solution

It's a good time to start acquiring parallel knowledge