Grand Challenge Problems

- A class of problem known as Grand Challenge problems.
- For example, a 10 year simulation of the earth's climate using a state-of-the-art weather model would require 10 days at an execution speed of 10 gigaflops/s.
- The simulation would generate a 100GB or more data.
- There are a number of refinements for greater accuracy of weather prediction that would increase the computational requirements to 10,000 times.
- The refinement such as increased resolution from 100-km cell size to 10-km cell size alone will increase the computation requirement by about 1000 times.
- Other refinement such as simple ocean model, simple atmosphere chemistry to improved versions would introduce further multiplicative factors to speed requirements.

Grand Challenge Problems

- Thus, the need for faster computers is driven by the demands of both
  - data-intensive applications in commerce and
  - computation-intensive applications in science and engineering.
- The requirements of these fields are merging,
  - scientific and engineering applications become more data intensive, whereas
  - commercial applications perform more sophisticated computations.

Trends in Computer Design

- The performance of computer increased exponentially.
- From 1945 to current time the performance has increased at around a factor of 10 per year.
- There are physical limitation to what can be expected in enhancement of performance.
- Enhancing the performance would be utilize internal concurrency in a chip.
- For example, by operating simultaneously on all 64 bits of two numbers that are to be multiplied.

What is the Limitation

- But this strategy is expensive.
- There is a fundamental result in VLSI that states

  For certain transitive computations the chip area A and the time T required to perform this computation are related so that AT^2 must exceed some problem-dependent function of problem size.

- This result is informally considered as follows:

  A computation must move a certain amount of information from one side of a square chip to the other. The amount of information that can be moved in a time unit is limited by the cross section of the chip $\sqrt{A}$. This gives a transfer rate of $\sqrt{A}T$, from which the relation is derived.

- To decrease the time required to move the information by a certain factor, the cross section must be increased by the same factor
- Hence the total area must be increased by the square of that factor.

What is the Limitation

- The AT^2 result means that
  - not only is it difficult to build individual components that operate faster,
  - it may not even be desirable to do so.
- It may be cheaper to use more, slower components.
- For example, if we have an area n^2A of silicon to use in a computer,
  - we can either build n^2 components each of size A and able to perform an operation in time T, or
  - build a single component able to perform the same operation in time T/n.
- The multicomponent system is potentially n times faster.

Parallel Algorithms

- Suppose several instances of the sequential computation model are available and can be used together to work simultaneously.
- How to use such a collection of computing devices to execute a single program faster?
- First of all how a collection many computers can at all be made to work in the control of a single program.
- Second how to ensure that the program can be partitioned efficiently, so that different parts may be assigned to distinct computing devices for execution.
- Third how the outputs of part computations be combined to produce the of the output of the whole.

Parallel Algorithms

- Clearly, the first issue concern architecture of a new computing model different from the classical sequential model.
- What could be this model?
- Is there just one possibility or there are several possibilities?
- Our first task will be answer these questions.

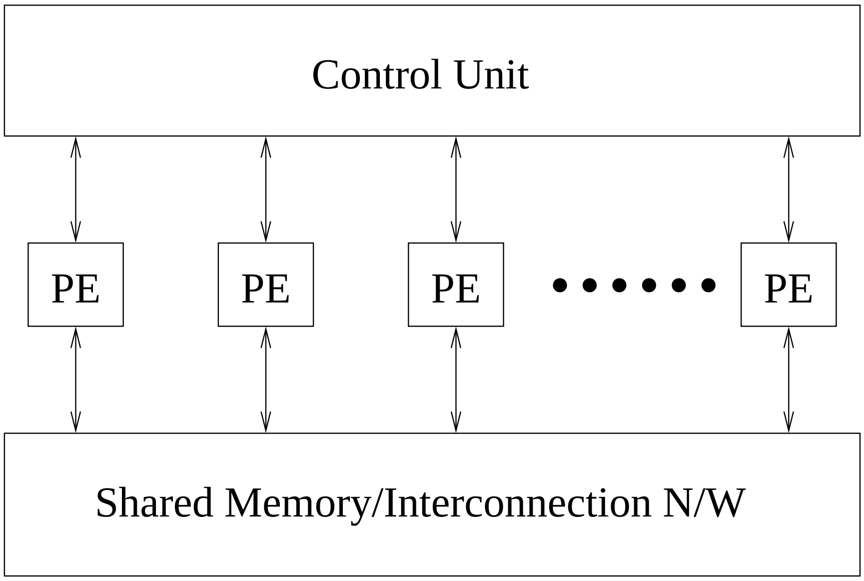
Architecture of a Parallel Model

- The first issue that we have to settle is:
  - whether the model should be viewed as a software model or a hardware model.
- If h/w model is used then there are two possible approaches depending on whether the execution of instructions is driven under a single control or under multiple control.
- If a single control used then we have synchronous model
- In case of single control, only data parallelism can be exhibited.
- In case of multiple control, both data and control parallelism can be exhibited.

Architecture

- We first introduce a classification before discussing further on parallel architectures driven under a single control.
- From point of view of execution control computing systems can be partitioned into 4 classes.
  - SISD – the classical model of sequential computation.
  - SIMD – the model for data parallelism.
  - MISD – the model for pure control parallelism.
  - MIMD – the generalized model.

SIMD Architecture

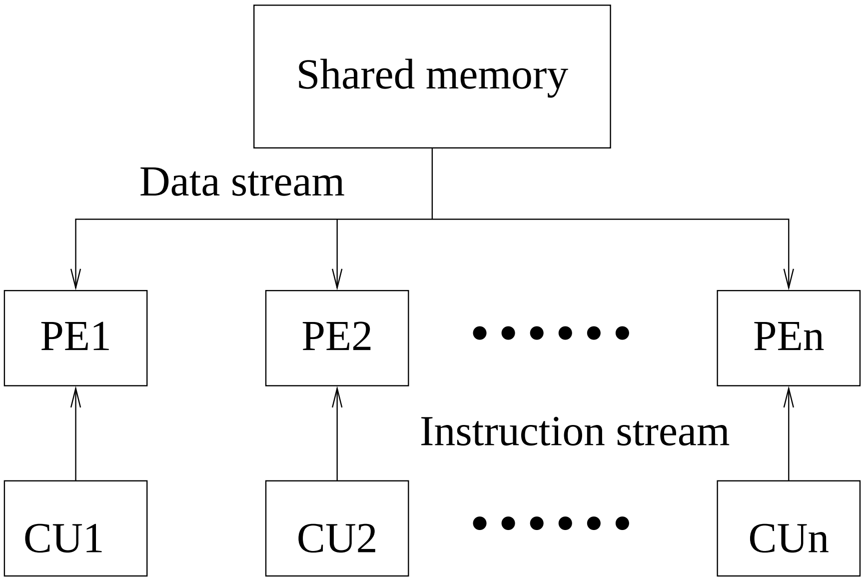


- Inter-processor communication can be done either using a shared memory or by explicit data-links between processors.
- There are problems in each option.

SIMD Architecture

- Shared memory is an attractive option from the point of view of ease in programming.
- But with a large number of processors, a shared memory is difficult put in practice.
- For distributed memory machines full interconnection will not be easy to realize due to similar reasons.
- But sparseness of processor interconnection may cause data communication overheads to dominate the computation time.

MISD Architecture

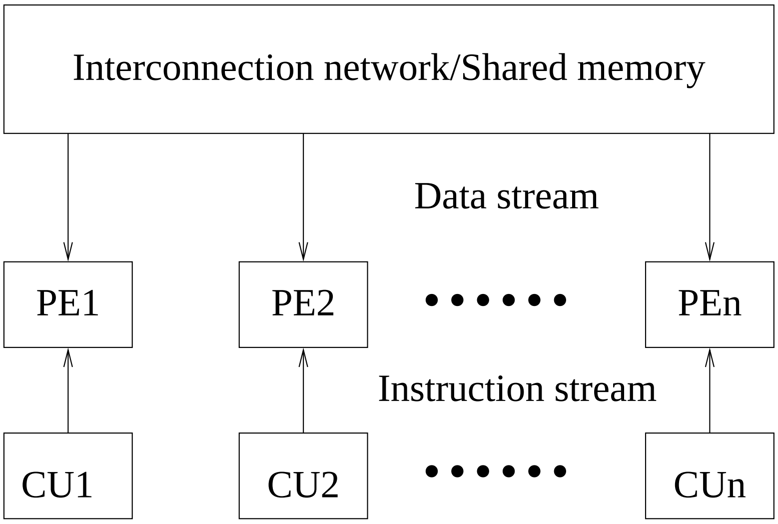


- Looks impractical as it only addresses the control parallelism.
- Different processing elements would work under different control, so MISD is an asynchronous model.

MISD Architecture

- The processor get the data from a single stream.
- So the MISD algorithms has the characteristic that different sets of instructions execute simultaneously on same data.
- It logically means that several algorithms work on same data.
- It does not make much sense to have a distinct class like MISD, because we can always make as many copies of the data as there are processors and run the MISD algorithm as an MIMD computation.

MIMD Architecture

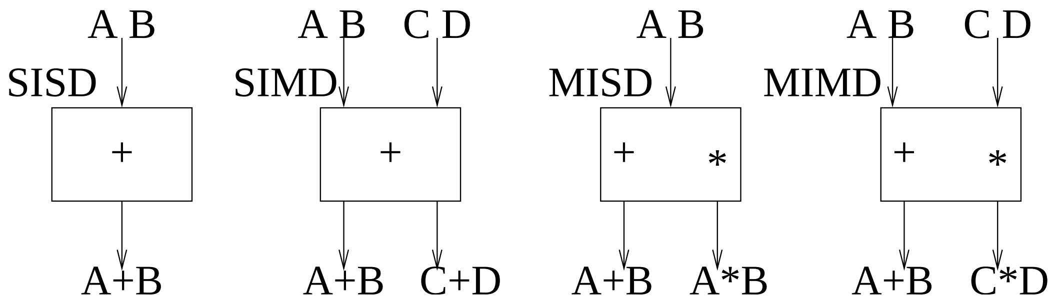


- Most multiprocessor systems and multiple computer systems can be placed in this category.
- A MIMD computer has many interconnected PE, each of which has its own control unit as shown in the figure.
- The processors work on their own data with their own instructions.

MIMD Architecture

- Tasks executed by different processors can start or finish at different times.
- MIMD thus is asynchronous parallel computational model.
- MIMD system is not the same as a distributed system.
- A distributed system can, e.g., consists of a number of PCs connected by a network.
- So even if the number of interconnected PCs can be quite large the communication between any two is very slow to be useful for any close operation on a single problem.

MIMD Architecture



- Above figure sums up the difference between the 4 models in Flynn's classification scheme.

Handler's Classification

- Flynn's taxonomy proved too coarse to express any details. So, more elaborate classification scheme is sought.
- Handler classification is one such scheme.
- It expresses both pipelining and parallelism of computers.
- Handler's taxonomy addresses the computer at three distinct levels:
  - the processor control unit (PCU),
  - the arithmetic logic unit (ALU), and
  - the bit-level circuit (BLC).

Handler's Classification

- It uses three pairs of integers to describe a computer:

$$
\begin{aligned}
\text{Computer} &= (k * k', d * d', w * w'), \text{ where} \\
k &= \text{number of PCUs} \\
k' &= \text{number of PCUs that can be pipelined} \\
d &= \text{number of ALUs controlled by each PCU} \\
d' &= \text{number of ALUs that can be pipelined} \\
w &= \text{number of bits in ALU (or PE) word} \\
w' &= \text{number of pipeline segments on all ALUs or in a single PE}
\end{aligned}
$$

Handler's Classification

- The following rules and operators are used to show the relationship between various elements of the computer.
- The '*' operator is used to indicate that the units are pipelined or macro-pipelined with a stream of data running through all the units.
- The '+' operator is used to indicate that the units are not pipelined but work on independent streams of data.
- The '$\vee$' operator is used to indicate that the computer hardware can work in one of several modes.
- The '$\backsim$' symbol is used to indicate a range of values for any one of the parameters.

Handler's Classification

- Peripheral processors are shown before the main processor using another three pairs of integers.
- If the value of the second element of any pair is 1, it may omitted for brevity.
- Handler's taxonomy is best explained by showing how the rules and operators are used to classify several machines.
- The CDC 6600 has a single main processor supported by 10 I/O processors.
- One control units coordinates one ALU with a 60-bit word length.
- The ALU has 10 functional units which can be formed into a pipeline.
- The 10 peripheral I/O processors may work in parallel with each other and with the CPU.

Handler's Classification

- Each I/O processor contains one 12-bit ALU.
- The description for the 10 I/O processors is:

$$\text{CDC 6600 I/O} = (10, 1, 12)$$

- The description for the main processor is:

$$\text{CDC 6600main} = (1, 1 * 10, 60)$$

Handler's Classification

- The main processor and the I/O processors can be regarded as forming a macro-pipeline so the '*' operator is used to combine the two structures:

$$\begin{aligned} \text{CDC 6600} &= (\text{I/O processors}) * (\text{CPU}) \\ &= (10, 1, 12) * (1, 1 * 10, 60) \end{aligned}$$

- Texas Instrument's Advanced Scientific Computer (ASC) has one controller coordinating four arithmetic units.
- Each arithmetic unit is an eight stage pipeline with 64-bit words.
- Thus we have: ASC = (1, 4, 64 * 8)

Handler's Classification

- The Cray-1 is a 64-bit single processor computer.
- Each ALU has twelve functional units, eight of which can be chained together to from a pipeline.
- Different functional units have from 1 to 14 segments, which can also be pipelined.
- Handler's description of the Cray-1 is:

$$\text{Cray-1} = (1, 12 * 8, 64 * (1 \backsim 14))$$

Handler's Classification

- Another sample system is Carnegie-Mellon University's C.mmp multiprocessor.
- This system was designed to facilitate research into parallel computer architectures and consequently can be extensively reconfigured.
- The system consists of 16 PDP-11 'minicomputers'
- Each of which have a 16-bit word length, interconnected by a crossbar switching network.
- Normally, the C.mmp operates in MIMD mode for which the description is (16, 1, 16).
- It can also operate in SIMD mode, where all the processors are coordinated by a single master controller.
- The SIMD mode description is (1, 16, 16).

Handler's Classification

- Finally, the system can be rearranged to operate in MISD mode.
- Here the processors are arranged in a chain with a single stream of data passing through all of them.
- The MISD modes description is (1 * 16, 1, 16).
- The '$\vee$' operator is used to combine descriptions of the same piece hardware operating in differing modes.
- Thus, Handler's description for the complete C.mmp is:

  C.mmp $= (16, 1, 16) \vee (1, 16, 16) \vee (1 * 16, 1, 16)$

Handler's Classification

- The '*' and '+' operators are used to combine several separate pieces of hardware.
- The '$\vee$' operator is of a different form to the other two in that it is used to combine the different operating modes of a single piece of hardware.
- The ICL DAP is another processor that can be reconfigured.
- It is a mesh connected bit-sliced array processor with a single control unit.
- It can work as 128 ALU's with 32-bit words and 64 Kbits of memory.

Handler's Classification

- Reconfiguring gives a computer with 4,096 ALU's with 1-bit words and 4Kbits of memory.
- Originally this machine was 'front-ended' by an ICL 2900 computer for which the description is (1, 1, 32).
- The DAP was designed to look like a 2MByte block of memory to the ICL 2900.

  ICL DAP $= (1, 1, 32) * [(1, 128, 32) \vee (1, 4096, 1)]$

- Handler's taxonomy does not indicate by what topology the processors are connected, (in this case a mesh network).

ILLIAC-IV

- The ILLIAC-IV array processor was developed at the University of Illinois and fabricated by Burroughs.
- It is made up from a mesh connected array of 64 64-bit ALUs.
- Front-ended by a Burroughs B6700 computer.
- Later designs used two DEC PDP-10's as the front end.
- The ILLIAC-IV can only accept data from one PDP-10 at a time.

$$\begin{aligned}\text{B6700-ILLIAC-IV} &= (1,1,48) * (1,64,64)\\ \text{PDP 10-ILLIAC-IV} &= (2,1,36) * (1,64,64)\end{aligned}$$

ILLIAC-IV

- The ILLIAC-IV can also work in a half word mode where there are 128 32-bit processors rather than the normal 64 64-bit processors.

$$\text{PDP 10-ILLIAC-IV}/2 = (2,1,36) * (1,128,32)$$

- Combining this with the above we get the following:

$$\text{PDP 10-ILLIAC-IV} = (2,1,36) * [(1,64,64) \vee (1,128,32)]$$

ILLIAC-IV

- The OMEN-60 by the Sanders Corporation uses a PDP-11 to front end a block of 64 1-bit ALUs working as an associative processor.
- The PDP-11 interprets the program and forms the control signals for the associative processor.
- If the program can use the associative processor it is run on the associative processor otherwise it is run on the PDP-11.
- The two do not act as a pipeline so the '+' rather than the '*' operator is used to combine them.
- PRAM allows

$$\text{OMEN-60} = (1, 1, 16) + (0, 64, 1)$$

- The '0' indicates that the associative unit has no control unit and cannot interpret its own instructions.

PRAM

- P-RAM is the parallel RAM model.
  - Consists of a finite number of processor numbered 1, $\ldots, p$.
  - All the processors work synchronously.
  - All connected to a large shared memory.
  - Every processor can access its own index number.
- It is a theoretical model.
- Shared memory access protocols for reading and writings define variants of PRAM models.
  - EREW: no simultaneous access to a cell is possible.
  - CREW: allows simultaneous access to a cell only for reading, but no two processors can write simultaneously to a single cell.

CRCW PRAM

- CRCW: allows simultaneous access even for writing. Many variations of PRAM are possible on how the simultaneous writes are resolved.

  common: allowed only if contending processor try to write the same thing.

  arbitrary: any one of the contending processor can successfully write.

  priority: only smallest or largest numbered processor among the contending ones become successful.

  tolerant: if many contending processor try to change the contents of a cell then value remains unchanged.

  collison: a collision symbol gets written in case of current write.

  combining: sum (an associative and commutative operation) of all values that the contending processors try to write gets written.

PRAM

- EREW model is the weakest among the PRAM model. CREW can simulate any EREW algorithm in the same time where CR facility is just ignored.
- Same is also true for simulating any EREW algorithms on the CRCW model.
- Priority CRCW model is stronger than both common and arbitrary CRCW models.
- Any algorithm designed for common model has to elect an arbitrary processor as the `winner` among all the processors that want to write the same value.
- If the algorithm is executed on prioity model then all processor attempting to write same value just allow either the `lowest` or the `highest` indexed processor becomes the `winner`.

CRCW Models

- Any algorithm for tolerant model can be simulated on collision with O(1) slow down, because all we require is replace the collision symbols by previous content of the cell.
- Collision can be simulated on priority model: just let the lowest (highest) indexed processor write the collision symbol.
- So some of the obvious relationships are:

$$\text{common} \leq \text{arbitrary} \leq \text{priority}$$
$$\text{tolerant} \leq \text{collision} \leq \text{arbitrary} \leq \text{priority}$$

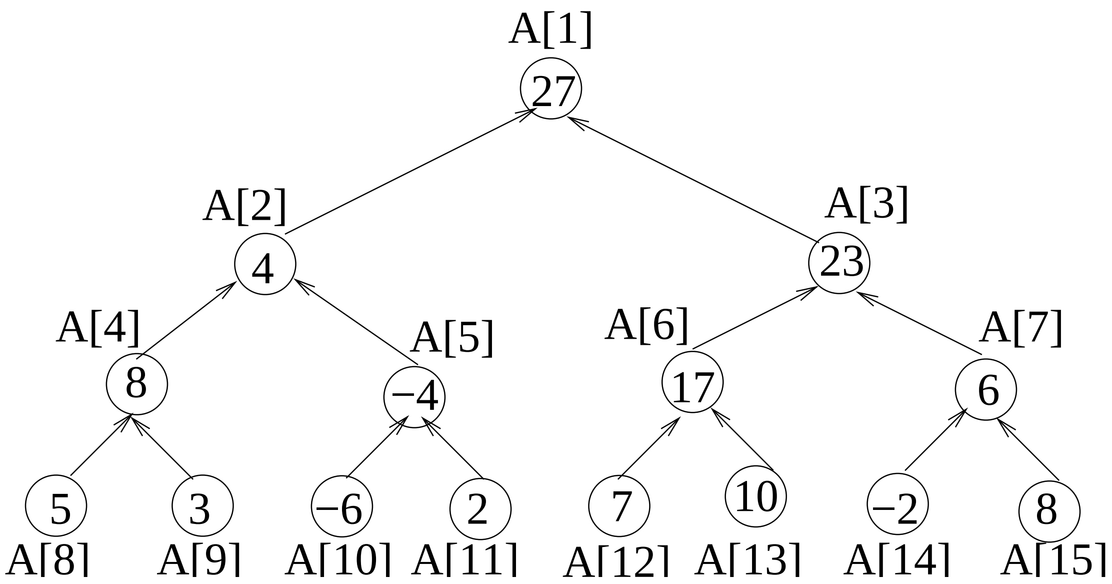
Paradigms

- Most important paradigm is `binary tree`.
- In some algorithms data flows from top to down.
- The type of algorithms that can belong to this are `broadcast` and `divide-and-conquer`.
- The difference between broadcast and divide-and-conquer lies in the way the data sent to the leaves by the root.
  - In first case the root sends the same data to leaves.
  - In the second case the data is sent by recursive subdivision.
- In some algorithms, the data flows from bottom to top, these algorithms are called `fan-in` or `reduction`.

EREW Algorithm for Sum

- Parallel sum is an example of reduction technique.

```
EREWsums() {
   shared var n, h, A[1..2n-1];
   for (h=1; h ≤ log n; h++)
      for all (P_i:  (n/2^h)...(n/2^(h-1)) − 1)
         A[i] = A[2i] + A[2i+1];
}
```



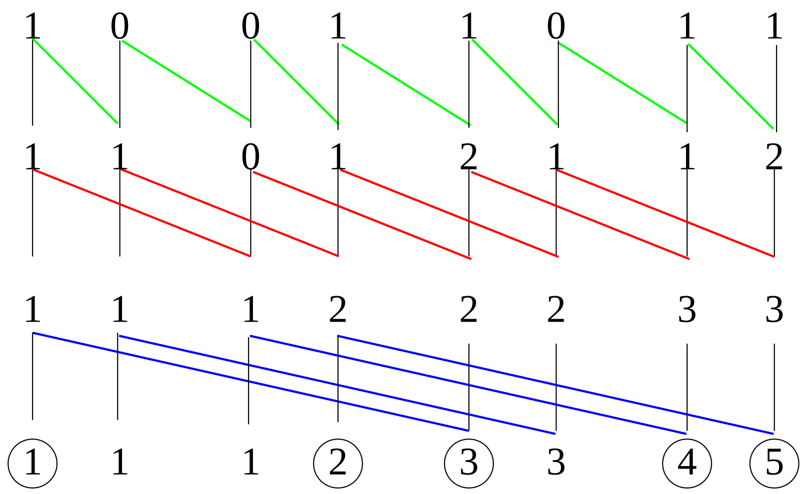
Paradigms

- Parallel prefix sum is an important fundamental step in many parallel algorithms. It is often needed in array compression operation.
- For example, sometimes only selective processors may have to be activated on the basis of the elements of an array of known flag values.
- Each active processors performs the selected operation on the the array element that is sequentially next element of the element in the array on which the adjacent active processor operates.
- So, prefix sum is used to determine the index mappings of the array elements to the active processor indices.

CREW Perfix Sums

```
CREWprefixSum() {
   shared var n, A[0,...,n-1], j;
   for all (P_i: 1...n-1) {
      for (j=0; j≤ ⌈log n⌉; j++)
         if (i-2^j ≥ 0)
            A[i] += A[i-2^j];
   }
}
```

- Concurrent accesses of shared locations is exhibited by this algorithm.

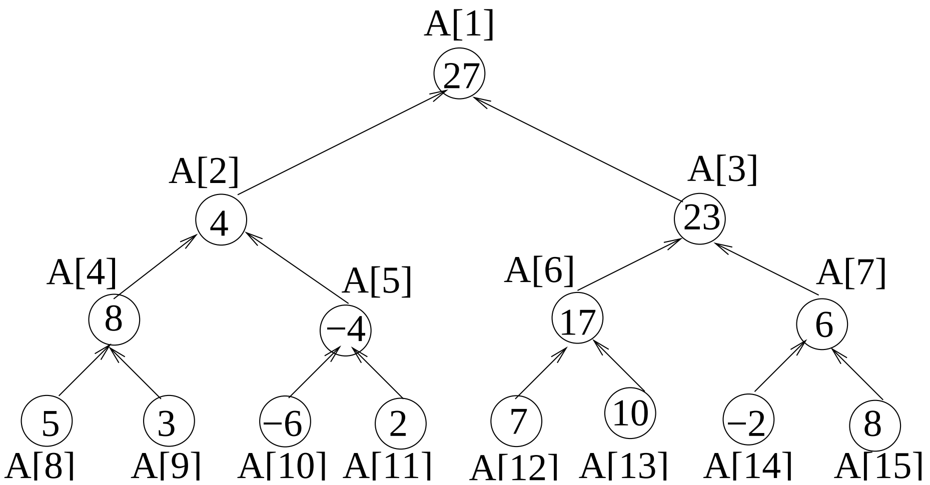


EREW Prefix Sum

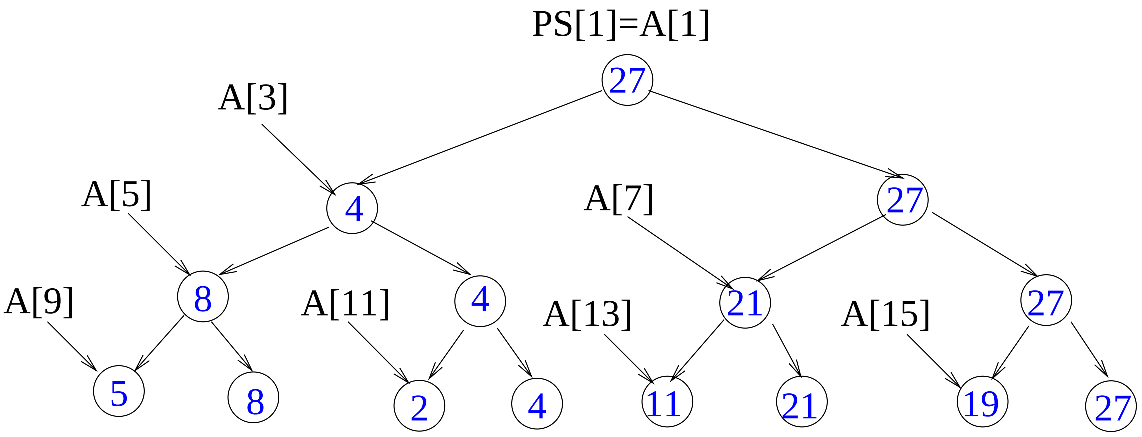
- Prefix sum can be computed on EREW PRAM using an auxiliary array of same size and without any extra premium on time.
- The algorithm consists of two passes: one bottom up and the other top down.
- In the first (bottom up) pass input set of numbers are summed in parallel.
- The second (top down) pass begins from the root, with prefix sum corresponding to the root set to the total sum.
- At any internal node,
  - if the node is the left child of its parent then the prefix sum is the difference of the parent's prefix sum and the right sibling's first pass number.
  - otherwise the prefix sum is same as the prefix sum of the parent.
- Continue the previous two steps until prefix sums for all leaf nodes are computed.

EREW Pefix Sum

- The first pass is same as EREW sum algorithm.



- The second pass progresses like divide-and-conquer.



The Second Pass

```
// Assume PS[.]  to be prefix sums
secondPass() {
   shared var PS[.], A[.], h;
   PS[1] = A[1];
   for (h=log n; h≥1; h--)
      for all (P_i:(n/2^h)...(n/2^(h-1)) - 1) {
            PS[2i+1] = PS[i];
            PS[2i] = PS[i] - A[2i+1];
   }
}
```

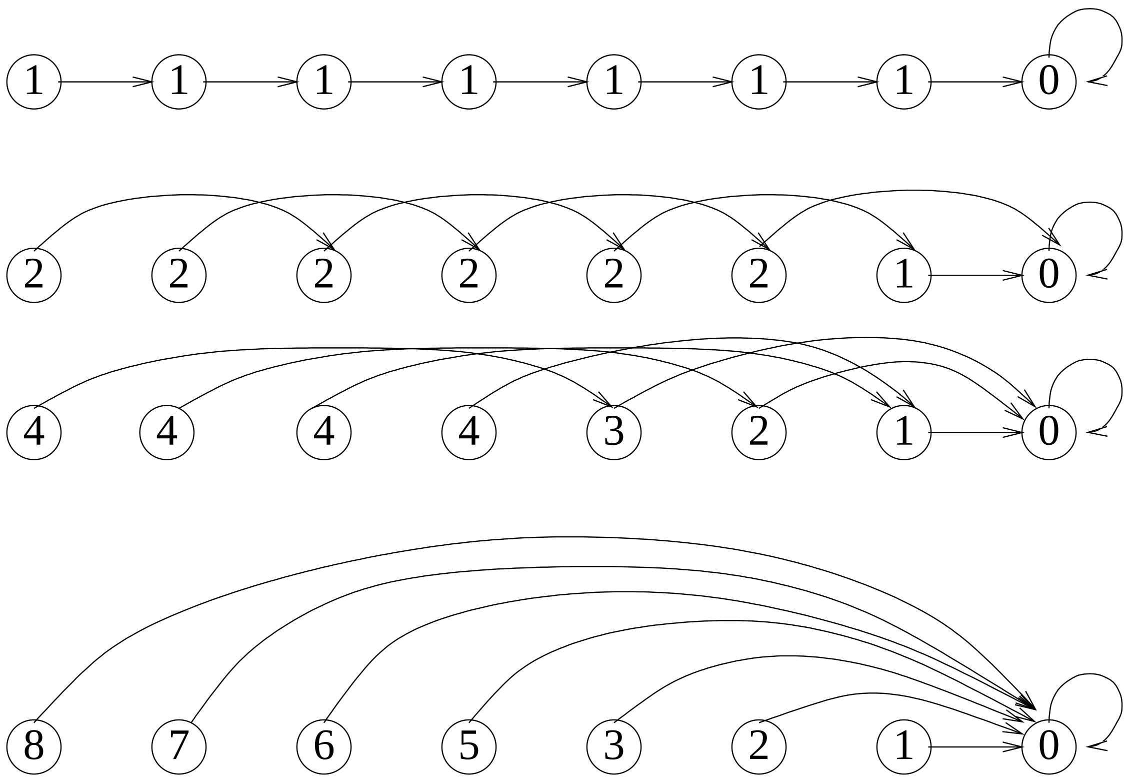

Pointer Jumping

- Inverse of prefix sums problem.
- For each element in a linked list of n elements we are interested to find the suffix sums of last i, $1 \leq i \leq n$, elements.
- Count the number of links from the element to the tail.
- Only one link can be traversed at time.
- So it appears that to traverse from first to tail $n - 1$ steps needed.
- Can we do better than that.

Pointer Jumping

- Associate a processor with each element.
- Jump the pointers in parallel.
- So, every jump would reduce the distance to half of that in the previous step.
- Hence logarithmic jump steps are sufficient to collapse the list so that every element points to the last.
- While doing the pointer jumping, if every processor also adds its link traversal count with that of its successor then list position of the element can be determined.

Pointer Jumping



Pointer Jumping

```
pointerJumping() {
   shared var n, pos[.], next[.], j;
   for all (Pi: 0...n-1) {
      if (next[i] == i)
         pos[i] = 0;
      else
         pos[i] = 1;
      for (j=1; j≤ log n; j++) {
         pos[i] += pos[next[i]];
         next[i] = next[next[i]];
      }
   }
}
```