

The Optimal Circuit Layout

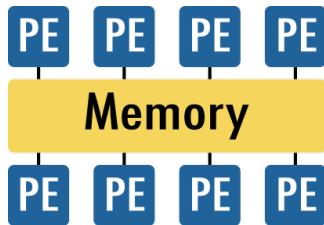
Nikola Samardzic

2020-04-11

Introduction

- We are looking for specific use-cases for our theoretical result and hope you can help!
- For more on the implementation details, see [?]

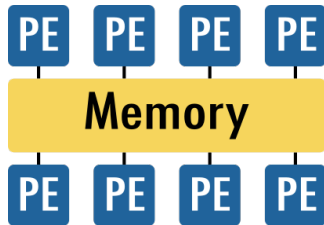
The PRAM Model



- PRAM = Parallel Random-Access Machine
- All processors share a big memory
- Memory access is constant time for each processor



The PRAM Model



- PRAM = Parallel Random-Access Machine
- All processors share a big memory
- Memory access is constant time for each processor

This requirement is physically unfeasible



The PRAM Model

- All $O(\log N)$ under PRAM: sorting, merging, ranking, FFT, matrix mul., poly. mul, longest repeated substring, transitive closure, connected components, etc.
- A lot of research on PRAM model (mostly from the '80s) [1]

Key Question

Can we make the PRAM model more realistic without losing these amazing results?

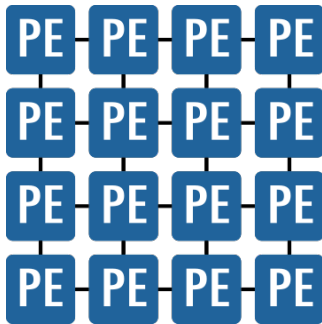
Preliminaries: The AT^2 -Interconnect Model

- Goal: Minimize AT^2 of a circuit (A is area; T is time)
- Signal delay over a wire is proportional to the wire's length ($\Theta(L)$)
- Computational gates take some constant, finite area
- The circuit lives in a 2D plane

Preliminaries: Transitive Functions

- **Transitive functions** are $\Omega(N)$ area and $\Omega(\sqrt{N})$ time.
- Transitive functions include: sorting, cyclic shift, product, convolution, linear transform, matrix multiply, etc.
- The **Bitonic Mesh-Sort** achieves this lowerbound for both area and time! [2]

Bitonic-Mesh Sort

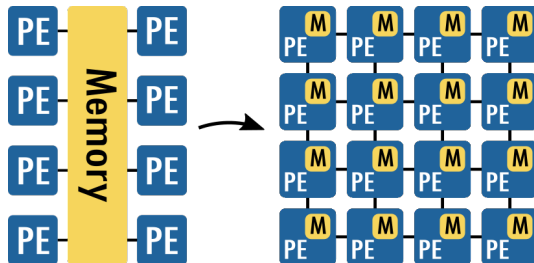


- Bitonic network-based sorter, but runs on a mesh of processors

Key Insight

Key Insight

We can use bitonic mesh sorting to implement global memory in the PRAM model.



Key Insight

- For all i , the i -th PE in the mesh:
 - contains a small memory block and computational unit
 - stores the contents of the i -th address of PRAM's global memory
 - The i -th PE in the mesh corresponds to the i -th PE in the PRAM model

The Algorithm

- We can use the mesh to run an arbitrary PRAM algorithm:
- By definition, each PRAM algorithm alternates between a memory access and compute step
 - During the computation step: each PE runs one cycle of the PRAM algorithm as normal
 - During the memory access step: we use bitonic mesh-sort to get the requested data to each processor in $\Theta(\sqrt{N})$ time (which is optimal)

Result

This approach immediately gives optimal circuits for all transitive functions ...all based on the same architecture.

Advantages

Advantages of our approach:

- Theoretically optimal
- Deterministic execution time no matter the input (major advantage compared to how networks are implemented in datacenters now!)
- Very easy to write circuits for this model: Just write the PRAM algorithm in C and a general framework can take care of the rest.

Disadvantage

- The total number of cycles we use is $\sim 14\sqrt{N}$

Key Issue

Constants matter! We need to find specific technological domains where the constants our approach has actually give great results.

We need help finding specific applications of this approach.
Some ideas:

- distributed computing using peer-to-peer networks

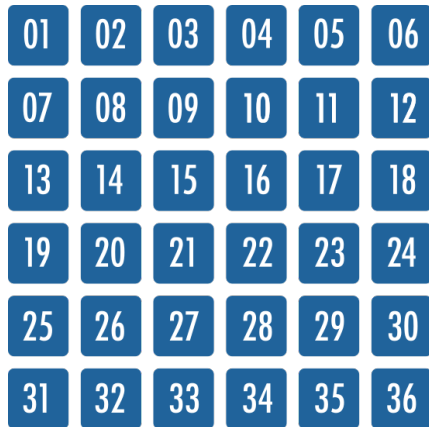
Implementing the memory access step

How do we implement N PE's reading data from each other in $O(\sqrt{N})$ time?

- Each PE specifies a destination PE it wants to read from (or 'nop' if it doesn't want to read).
- The tuple <source PE, destination PE> is sent to the destination PE.
- The destination PE creates a tuple <data, source PE>, with data equal to the contents of its memory.
- The <data, source PE> tuple is returned to the source.
- Similar process for concurrent reads, exclusive / concurrent writes [?]
- Details follow...

N-to-N Processor Communication Details (I)

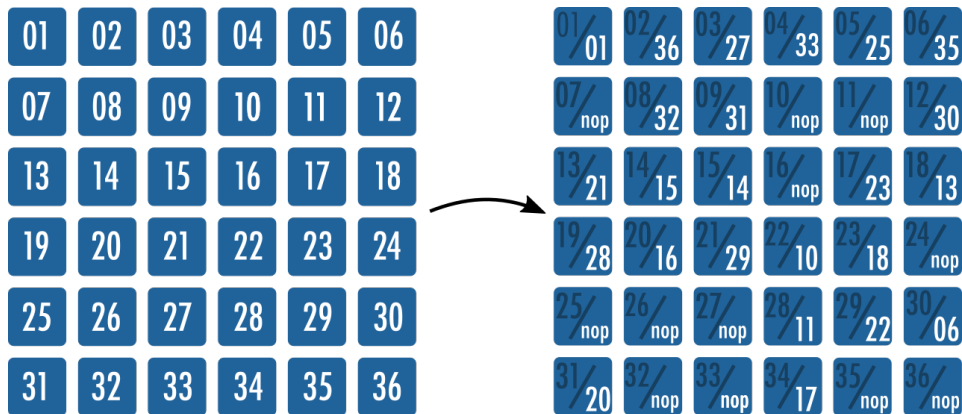
- Each PE has an index:



01	02	03	04	05	06
07	08	09	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

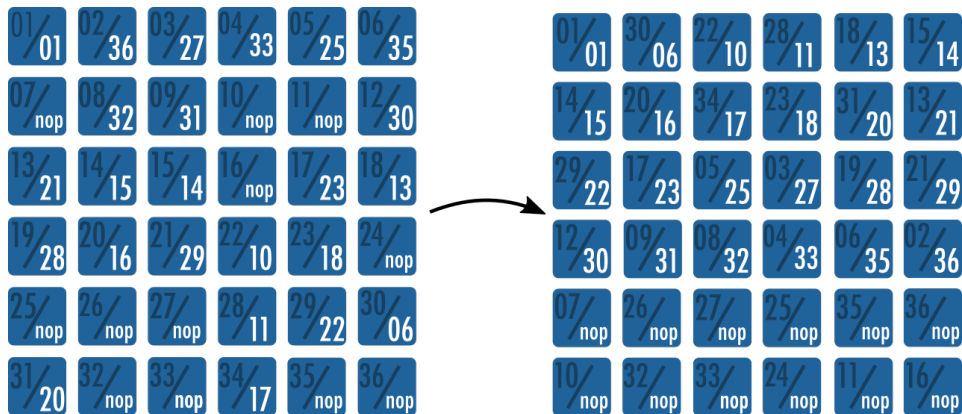
N-to-N Processor Communication Details (II)

- Each PE creates a tuple containing its index and the index of its destination PE (e.g., PE 2 wants to read from PE 36):



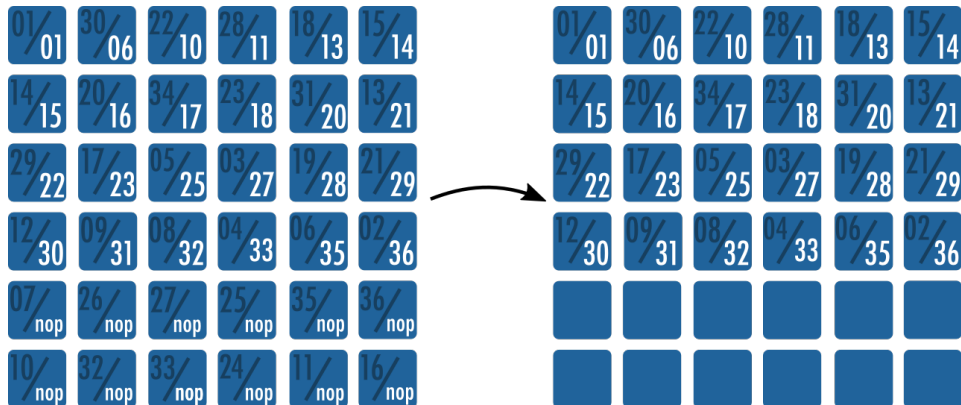
N-to-N Processor Communication Details (III)

- Now sort the tuples based on destination PE using bitonic mesh-sort (consider nop bigger than any integer) [2]:



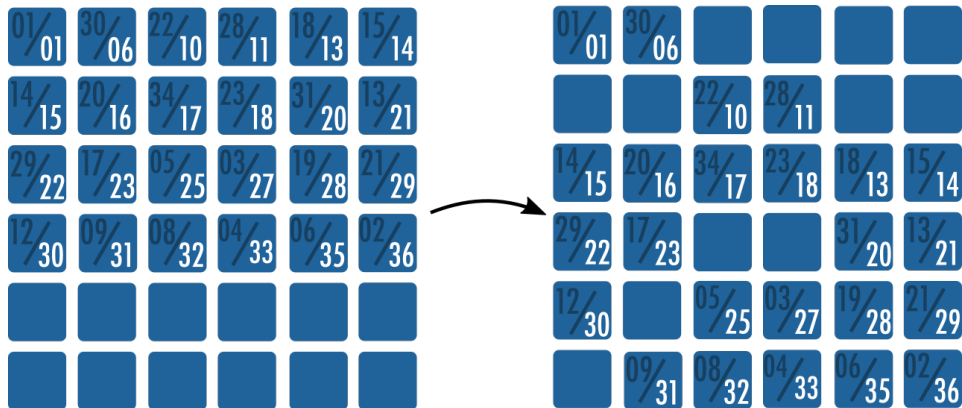
N-to-N Processor Communication Details (IV)

- Remove nop tuples:



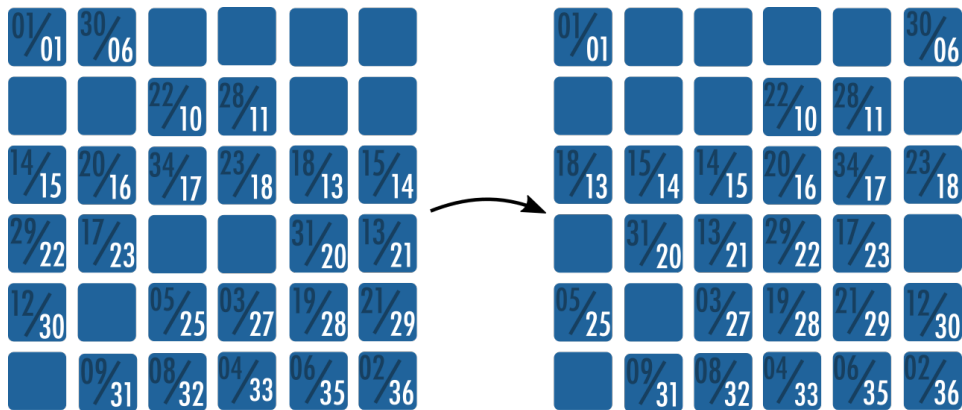
N-to-N Processor Communication Details (V)

- Put each tuple into its destination row (without changing any tuple's column):



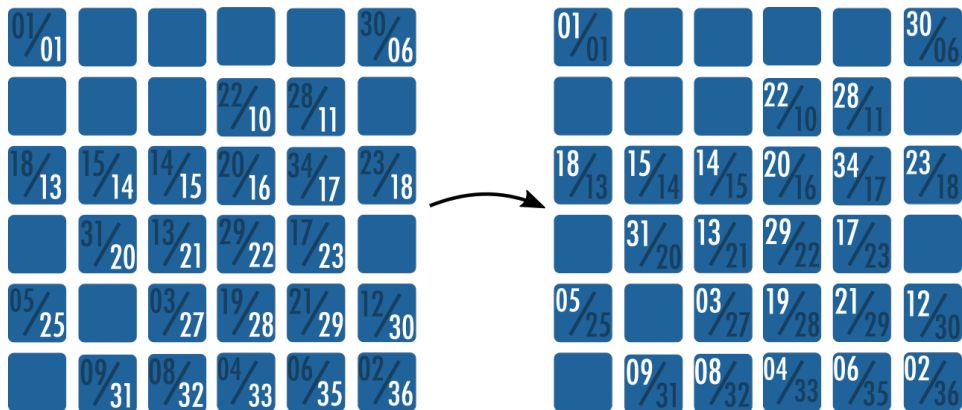
N-to-N Processor Communication Details (VI)

- Put each tuple into its destination PE (without changing any tuple's row):



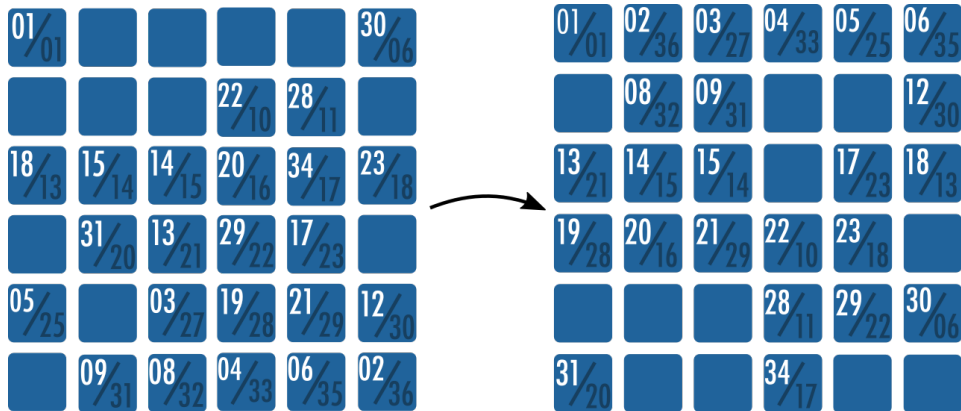
N-to-N Processor Communication Details (VII)

- The destination PE creates the <data, source PE> tuple:



N-to-N Processor Communication Details (VIII)

- The new tuples are moved back to the source PE's in the same way they were moved to the destination PE's:



- The source PE's read the data in the tuples they received.

References I



J. Jája.

An Introduction to Parallel Algorithms.
Addison-Wesley, 1992.



D. Nassimi and S. Sahni.

Bitonic sort on a mesh-connected parallel computer.
1979.