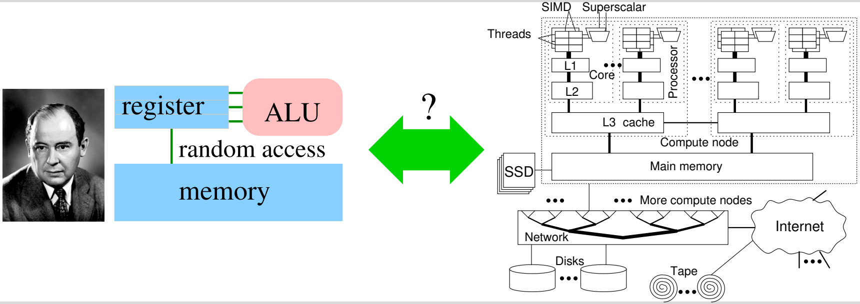
Karlsruhe Institute of Technology

Taming the Zoo of Parallel Machine Models

SPAA Tutorial

Peter Sanders | Jun 17, 2024



Overview

From von Neumann to modern machines

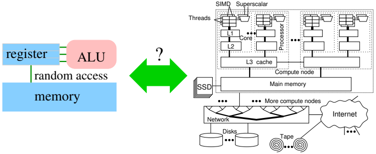
- RAM
- PRAM
- Memory hierarchies
- Distributed memory & communication
- Relations between models

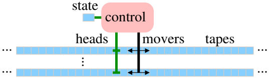
More abstract models

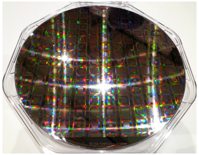
- MapReduce
- Distributed graph algorithms

Wrap Up

- GPU, Quantum, and more
- Open problems
- Future models

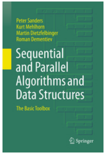






Sources

- Chapter Machine Models in 1st volume of my book Algorithm Engineering Draft:
  https://ae.iti.kit.edu/documents/people/sanders/aeModels.pdf
- My (parallel) algorithms textbook [SMDD19]
- My lecture on parallel algorithms



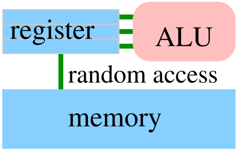
What is a (good) Machine Model

Abstraction of computations that allow (asymptotic) analysis of the resource consumption of algorithms

- Preferably simple
- Preferably close to reality

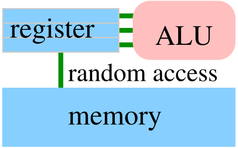
$\neq$ programming model
(which lacks performance aspect)

$\neq$ performance model
(which is much more detailed,
often more specific to a particular problem)



Random Access Machines (RAM)

(modern variant of the *von Neumann Model*) [SS63, SMDD19]



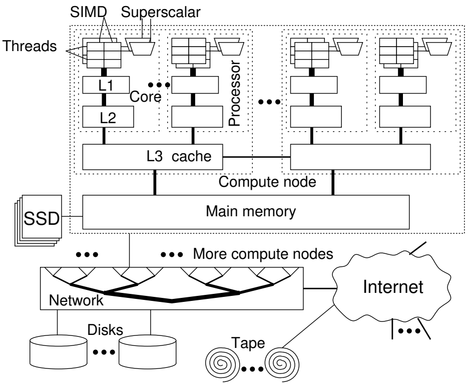
Sequential algorithms

- Count machine instructions
- on words of size $\mathrm{O}(\log n)$ for input size n, i.e. allow word/bit parallelism.

Still the basis for much of algorithmics.

The Problem

Modern machines are vastly different:



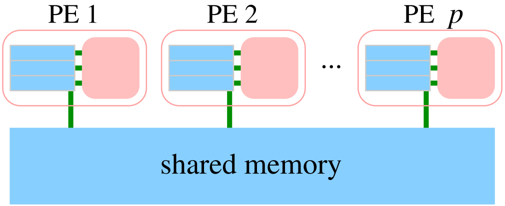
many forms of parallelism, complex memory hierarchies,...

Challenge: reconcile precision and simplicity

A Simple Parallel Model: PRAM
Parallel Random Access Machine

Idea: change RAM as little as possible.

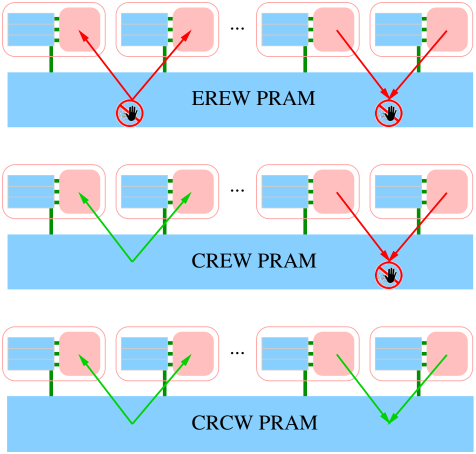
- p PEs (**P**rocessing **E**lements); numbered $1..p$.
  Every PE knows p and its own number.
- One machine instruction per clock cycle and PE – synchronous
- Shared global memory



Basic PRAM Variants

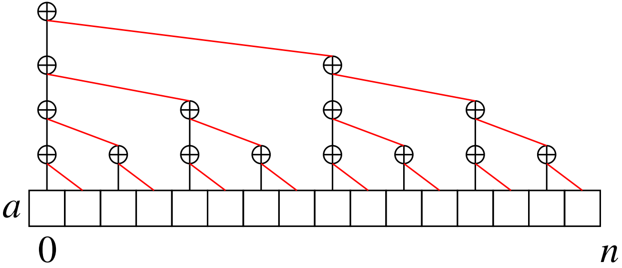
Concurrent access: Allow access by multiple PEs to the same memory cell in the same step?
Concurrent=yes, Exclusive=no.
Read or Write.

- EREW PRAM:
  Exclusive-Read, Exclusive-Write PRAM
  most restrictive
- CREW PRAM:
  Concurrent-Read, Exclusive-Write PRAM.
  Kind of default since
  caches approximate concurrent read well.
- CRCW PRAM:
  Concurrent-Read, Concurrent-Write PRAM.
  further variants regarding how conflicts are resolved (common, arbitrary, priority, combine)



Example: Reduction of Array $a[0..n)$

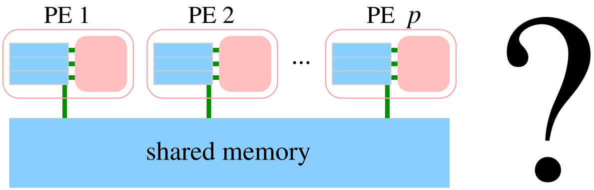
PE index $i \in \{0, \ldots, n-1\}$
active := 1
for $0 \leq k < \lceil \log n \rceil$ **do**
    if active **then**
        if bit k of i **then**
            active := 0
        else if $i + 2^k < n$ **then**
            $a[i] := a[i] + a[i + 2^k]$



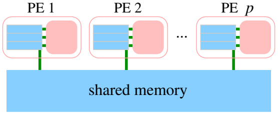
Criticism of PRAM

PRAMs were THE parallel model in the 1980s.
Parallel processing fell “out of favor” in the mid 1990s
(many bankruptcies of companies, era of “just wait for faster sequential processors”).
The PRAM model was seen as part of the problem.

- Unrealistic assumptions? Too fine-grained, hard to realize physically, lockstep execution is unrealistic,. . .
- Lack of transfer into applications
- Theoretical algorithms often look very different from what is done in applications
- Research ran out of interesting problems?



In Defence of PRAMs



!

In retrospect the criticism was largely unfair.

- PRAMs are very natural first step to express parallelism in computations.
- Many efficient PRAM algorithms can be further developed into efficient realistic algorithms.
- Todays many-core CPUs and GPUs are much closer to PRAMs than the first parallel computers (e.g., #cores, memory channels).
- Other now popular models such as MRC seem much farther removed from reality than PRAM.
- Surprisingly many open problems are left.

The actual problems were

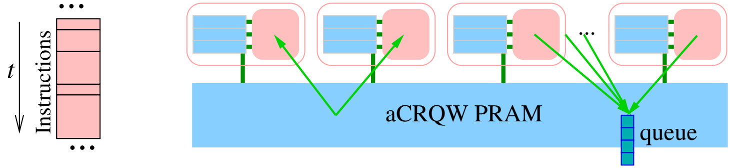
- Coarse complexity theoretic goal of polylogarithmic time and polynomial work ($\Rightarrow$algorithms can be highly inefficient)
- Lack of algorithm engineering

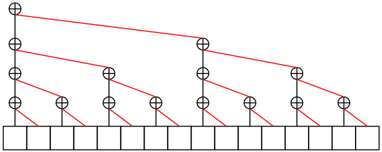
For example, my results on shortest paths [MS03b], matchings [BOS$^+$13], sampling [SLH$^+$18, HS22], or suffix arrays [KSB06], can be naturally explained using PRAM models.

Asynchronous PRAMs

aCRQW PRAM:

asynchronous Concurrent-Read, Queued-Write PRAM [GMR98, GMR99, SMDD19].
Programs must be correct regardless how long it takes to execute an instruction.
Write access takes time k when k PEs concurrently access a cell.
Uses atomic operations like fetch-and-add or compare-and-swap (CAS).
$\Rightarrow$ adequate modelling of contention.

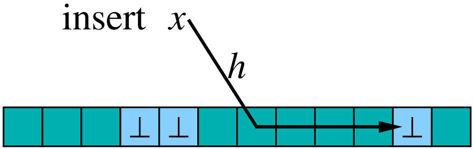




Exercise: What happens with our reduction example?

Example: Concurrent Hash Tables

Linear Probing [MSD19, SMDD19] largely does the job.

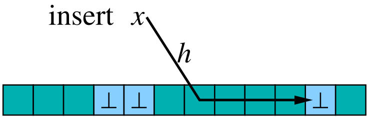


- Constant expected time for find – contention does not appear in practice as concurrent accesses are covered by cached copies.
- Constant expected time for insert – contention is unlikely.
- Update can suffer from contention.
- Maintaining a global variable for size kills performance of insert.

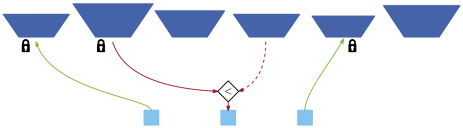
Open Problem:
Scalable Concurrent Algorithms and Data Structures

Karlsruhe Institute of Technology

There is a large body of very important work on concurrent algorithms and data structures (in particular lock-free/wait-free) that is lacking a scalability analysis (or provably scalable variants).
I believe aCRQW-PRAM or related models can be a good basis here.



hash table



priority queue

The Work-Span Model

Abstract from the actual number of processors p [AB16, BFGS20] used for a computation.
Only look at the

work W performed and the

span (or depth) T_∞ i.e., the longest sequence of dependent operations.
$\approx$ time with unbounded # of PEs.

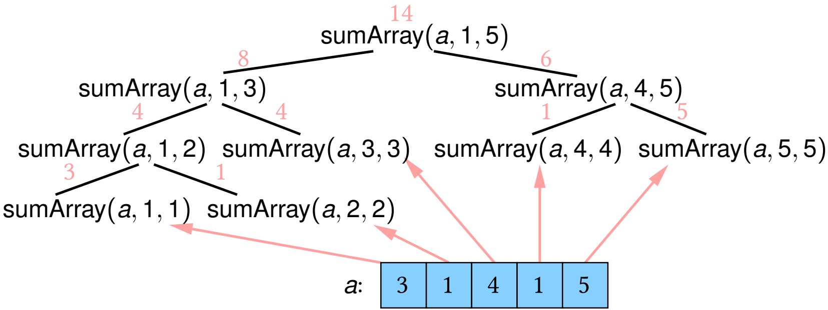
Computation $=$ graph of dependencies.
Unfolds at running time ($\neq$ circuit models)
Use fork operations to spawn tasks and atomic shared-memory operations
(e.g. CAS on aCRQW PRAM).

Example

Function sumArray$(a : \text{Array}, i, j)$ // compute $\sum_{k=i}^{j} a[k]$

  if $i = j$ **then return** $a[i]$

  else return sumArray$(a, i, \lfloor \frac{i+j}{2} \rfloor)$+ sumArray$(a, \lfloor \frac{i+j}{2} \rfloor + 1, j)$



Load Balancing for Forking Programs

Roughly: Using a work-stealing load balancer [BL99, SMDD19], the aCRQW PRAM can run programs in the work span model in expected time

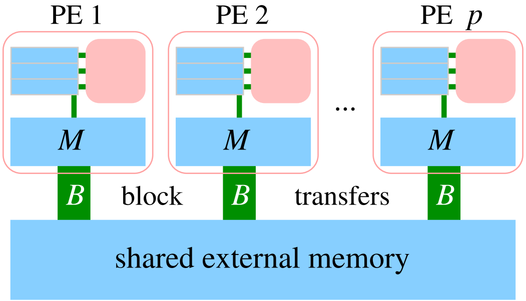
$$T(p) = \frac{\overbrace{W}^{\text{work}}}{p} + \overbrace{T_\infty}^{\text{span}}$$

(But check details of the model, e.g., what kind of forking)

Test-of-Time Award SPAA 2024

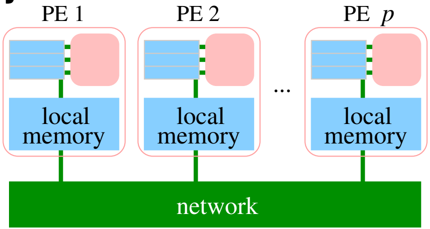
Parallel External Memory (PEM)

Any PRAM can be augmented to model a 2-level memory hierarchy [AGNS08].

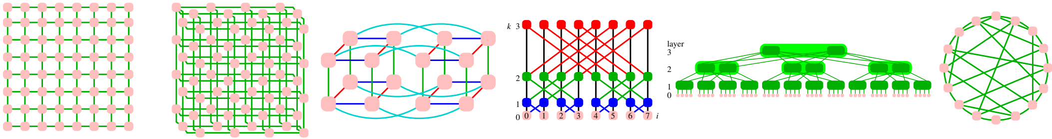


Count (parallel) I/O steps (in addition to internal work).
Example: cache-efficient parallel sorting [AFSW22].

Distributed-Memory Models



- Conceptually as simple and natural as PRAM
- We need a cost model for communication.
- Here: Abstract away the concrete network topology
  which was highly popular in the early days of parallel computing [Lei92].



Bulk Synchronous Parallel (BSP)

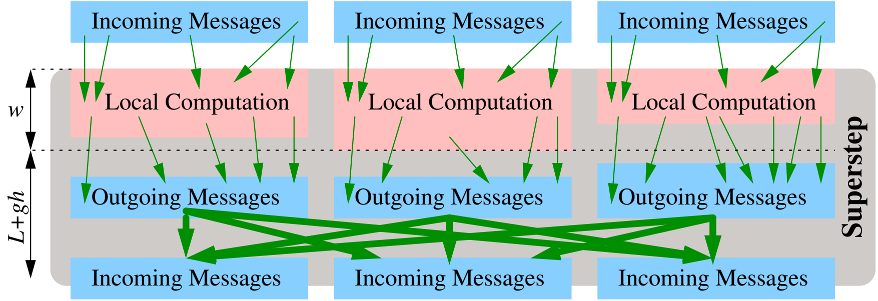
Decompose computation into synchronized supersteps [Val94]

L Latency (includes global synchronization)

g gap – measures network throughput

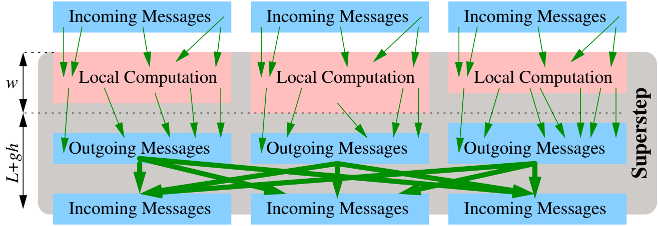
h bottleneck communication volume (*h*-relation)

w max. local work



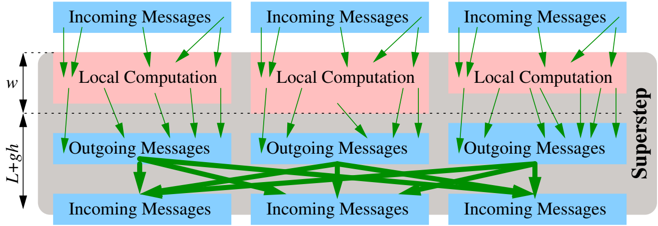
BSP: Strengths and Weaknesses

- + widely used
- + supported by some libraries
- – global synchronization is as expensive as other common primities such as broadcast, reduction, prefix sum which all take a logarithmic number of supersteps
- – too little differentiation of communication patterns, e.g., wrt average message size, locality
- – In practice, think of L as a huge value growing linearly with the number of PEs p!
- Special case CGM (Coarse Grained Multicomputer) – only count rounds



BSP$^+$: Fixing Some Weaknesses [SS24]

- Define h in terms of packets of size B rather than using machine words (BSP* [BDadH95]).
- Allow the collectives broadcast, reduce, prefix sum within a single superstep (anyway part of BSP libraries).



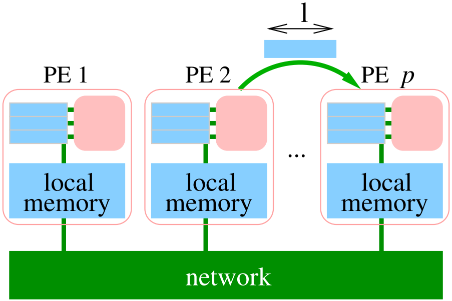
Asynchronous Distributed Memory: Point-to-Point Communication (P2P)

Time for sending a message of length ℓ [FL94, CHPvdG07, SMDD19]:

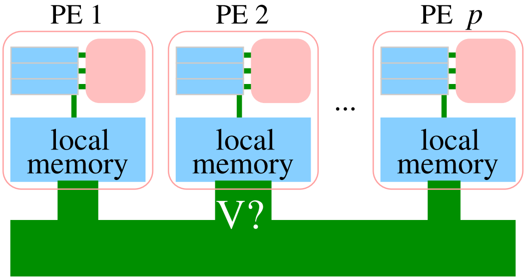
$$T_{\text{comm}} := \alpha + \beta\ell$$

Each PE can progress on one send and one receive at any point in time (full duplex).
(Several variants, e.g., either or = half duplex)

+ simple
+ asynchronous
+ defacto standard in practical distributed-memory computing
+ more useful parameters than the related LogP [CKP+93] model (see also LogGP model [AISS97])
− less known in theory community



Communication Efficient Algorithms

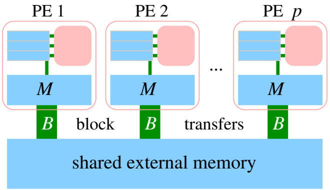


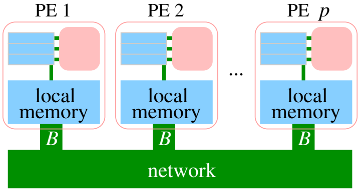
Owner computes paradigm:
Minimize bottleneck communication volume V.
Can we achieve volume sublinear in the local computation time?
Can we achieve polylogarithmic span?

Examples: duplicate detection [SSM13], linear programming [SSM13], top-k problems [HS16], graph generation [FLM$^+$19]

Relations Between Models

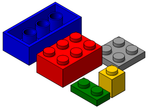
- All PRAM variants as well as BSP, P2P can emulate each other with at most logarithmic slow-down (assuming $L \subseteq \mathrm{O}(\log p)$, $\{g, \beta, \alpha\} \subseteq \mathrm{O}(1)$).
- BSP$^+$, P2P with $\ell = B = \alpha/\beta$ and PEM seem closely related.
  However, studying communication efficient algorithms with PEM would require you to assume that the input resides in the local memories.





Thus, decision for a particular model often is a matter of taste or you are interested in looking at parameters like α as variables rather than constants.

Model Lego



Use different models for different aspects of your algorithm.
Deal with hierarchical and heterogeneous architectures.
For example, parallel sorting [AFSW22]:

- BSP for a distributed memory sample sort
- PEM for a node-local shared memory sorter and partitioner
- specialized modelets to deal with branch mispredictions [KS06, SW04] or associative caches [MS03a]

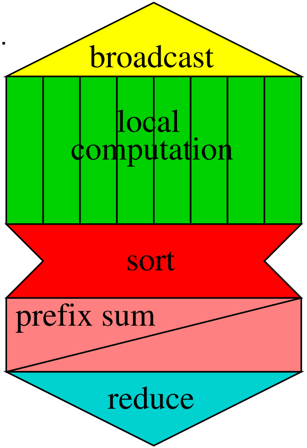
Much cleaner than one all-encompassing model.

Model-Agnostic Algorithm Design

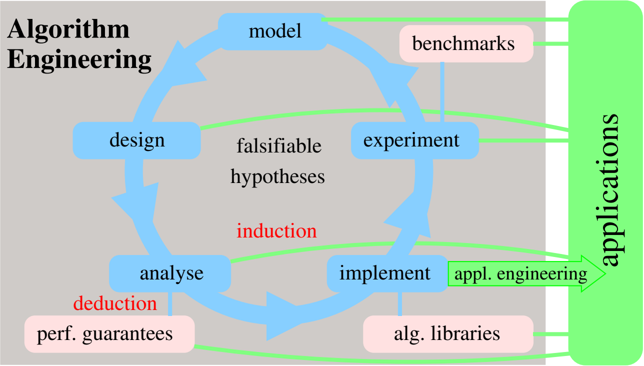
Construct your algorithm from building blocks like
broadcast, reduction, prefix sum, permutation, sorting, hash tables,. . .

Then plug in analysis for different models.

Model-Bingo in examples from my work:
random permutation [San98],
matching [BOS+13],
sampling [SLH+16]



Machine Models in Algorithm Engineering



pick initial model(s)
design
implementation takes care of unmodelled features
experiments may cause changing the model

Overview

From von Neumann to modern machines

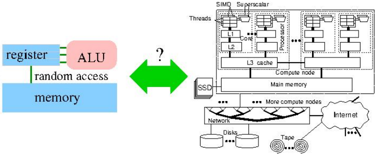
- RAM
- PRAM
- Memory hierarchies
- Distributed memory
- Relations between models

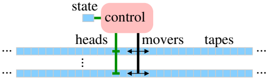
More abstract models

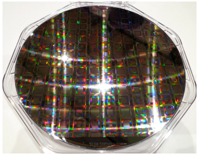
- MapReduce
- Distributed graph algorithms

Wrap Up

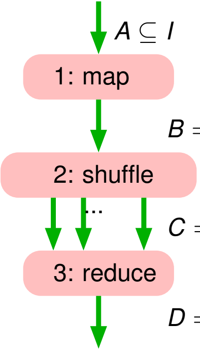
- GPU, Quantum, and more
- Open problems
- Future models







MapReduce



$$A \subseteq I$$

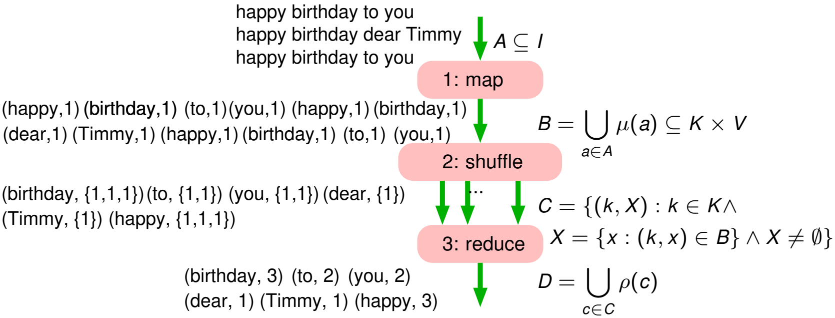
$$B = \bigcup_{a \in A} \mu(a) \subseteq K \times V$$

$$C = \{(k, X) : k \in K \wedge X = \{x : (k, x) \in B\} \wedge X \neq \emptyset\}$$

$$D = \bigcup_{c \in C} \rho(c)$$

[DG08]

MapReduce Example: Word Count



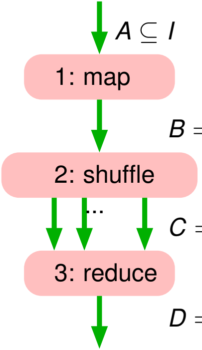
MapReduce Discussion

$+$ Abstracts away difficult issues

- parallelization
- load balancing
- fault tolerance
- memory hierarchies
- ...

$-$ Large overheads

$-$ Limited functionality



$A \subseteq I$

$$B = \bigcup_{a \in A} \mu(a) \subseteq K \times V$$

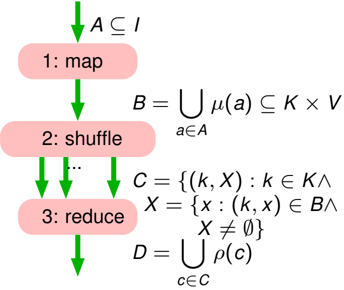
$$C = \{(k, X) : k \in K \wedge X = \{x : (k, x) \in B\} \wedge X \neq \emptyset\}$$

$$D = \bigcup_{c \in C} \rho(c)$$

MapReduce MRC Model

A problem is in MRC [KSV10] iff for input of size n:

- solvable in $\mathrm{O}(\mathrm{polylog}(n))$ MapReduce steps
- μ and ρ evaluate in time $\mathrm{O}(\mathrm{poly}(n))$
- μ and ρ use space $\mathrm{O}\left(n^{1-\epsilon}\right)$ ("substantially sublinear")
- overall space for B $\mathrm{O}\left(n^{2-\epsilon}\right)$ ("substantially subquadratic")



Roughly: count steps, (good for coarse grained complexity theory)
very loose constraints on everything else

MapReduce MRC Model **Criticism**

A problem is in MRC iff for input of size n:

- solvable in $\mathrm{O}(\mathrm{polylog}(n))$ MapReduce steps
- μ and ρ evaluate in time $\mathrm{O}(\mathrm{poly}(n))$ — n^2? n^{42}? “big” data?
- μ and ρ use space $\mathrm{O}\left(n^{1-\epsilon}\right)$ (“substantially sublinear”)
- overall space for B $\mathrm{O}\left(n^{2-\epsilon}\right)$ (“substantially subquadratic”) — “big” data?

Roughly: count steps,
very loose constraints on everything else — speedup? efficiency?

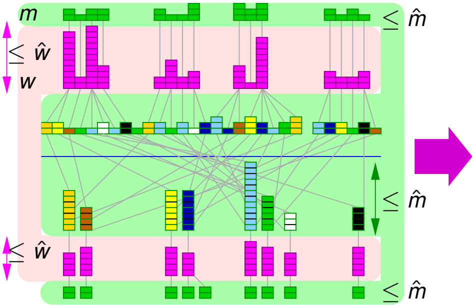
Sounds a bit like the problem that afflicted the complexity theoretical view of PRAMs

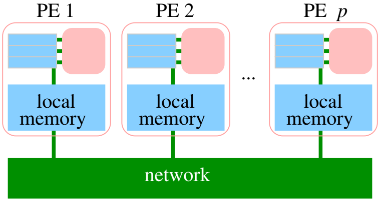
MapReduce – MRC$^+$ Model

Connecting MapReduce to Realistic Machine Models [San20]

internal work

$$\Theta\left(\frac{w}{p} + \hat{w} + \log p\right)$$



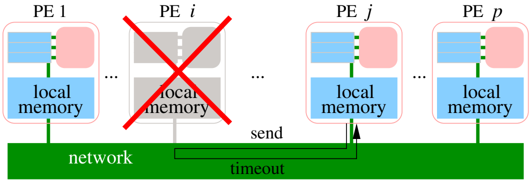


$$\Theta\left(\frac{m}{p} + \hat{m} + \log p\right)$$

bottleneck communication volume

Open Problems

- Apply MRC$^+$ Model to algorithms/problems previously studied for MRC. Are these still looking good? Do they yield new PRAM, BSP, P2P algorithms? Other algorithms now look better?
- Develop sth like MRC$^+$ for Big Data Tools with more functionality like Spark [ZCF$^+$10] or Thrill [BAJ$^+$16]
- Use the algorithms from [San20] to obtain more scalable MapReduce implementations. Possibly further developed with equally scalable fault tolerance.

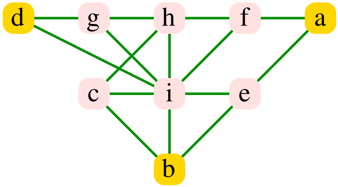


Distributed Graph Algorithms

- "The network is the computer" [Pel00, AW04, Ray13]
- Count rounds of data exchange with all neighbors

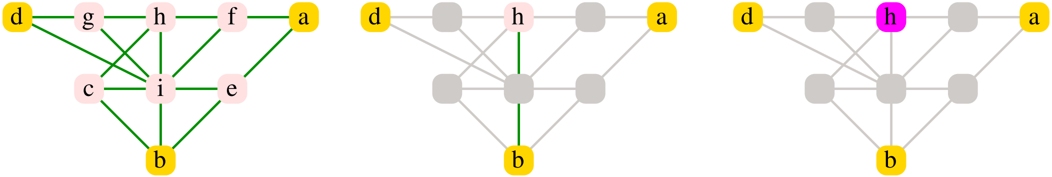
  Local Model: unbounded message lengths

  Congest Model: $\mathrm{O}(\log|V|)$ message lengths
- Quite different from processing graphs $G = (V, E)$ on distributed-memory computers, usually with $p \ll |V|$



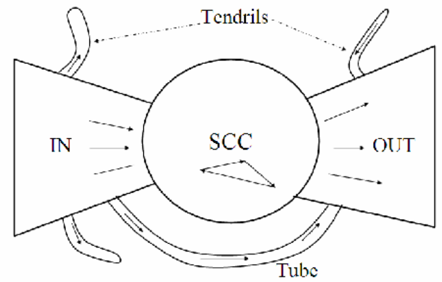
Example:
Luby's Maximal Independent Set Algorithm [Lub86]

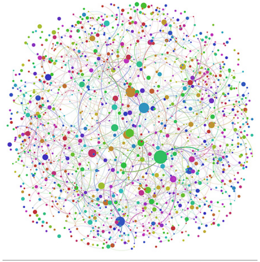
Iteratively select nodes with locally smallest (random) label.
Remove neighbors and incident edges.
whp $\mathrm{O}(\log|V|)$ rounds



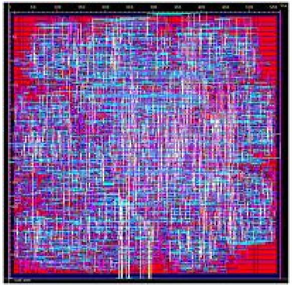
Open Problem

Clarify relation to realistic parallel machine models when processing large graphs, e.g., how to emulate an algorithm in the Congest model on BSP?
How to deal with high-degree nodes?
Which distributed graph algorithms then translate into efficient parallel algorithms?





Data: AER, JPE, Econometrica, RES, QJE (2000-present)
By Cloudly From: www.cloudlychen.net



Overview

From von Neumann to modern machines

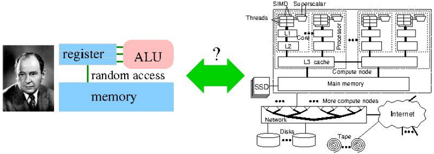
- RAM
- PRAM
- Memory hierarchies
- Distributed memory
- Relations between models

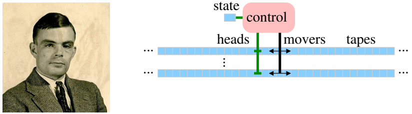
More abstract models

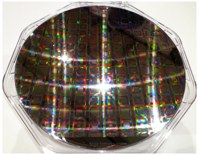
- MapReduce
- Distributed graph algorithms

Wrap Up

- GPU, Quantum, and more
- Open problems
- Future models







Steve Jurvetson
`creativecommons.org/licenses/by/2.0/`

GPU

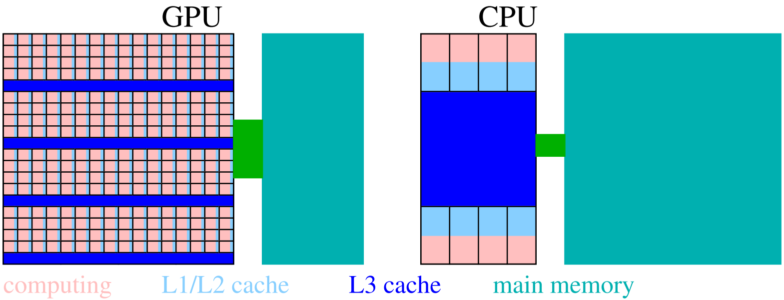
- Hierarchy of thread groups with increasingly closer cooperation
- Collective memory access patterns

Do we need a model for it?

Open problem (?)

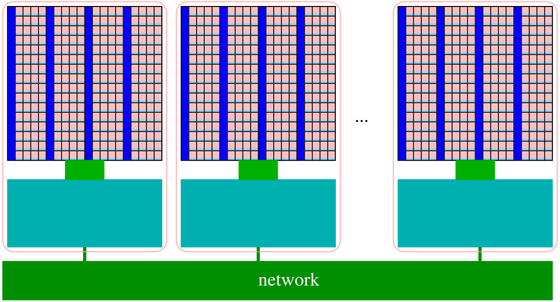
Or not?

- Differences to shared-memory CPUs is more quantitative than qualitative – more threads, smaller caches
- Not that different from a many-core CPU with powerful SIMD-units?
- CUDA is more a programming model than a machine model (and proprietary)
- Ideosyncrasies of warps not fitting to an asymptotic model
- Moving target (and converging to CPU models?)



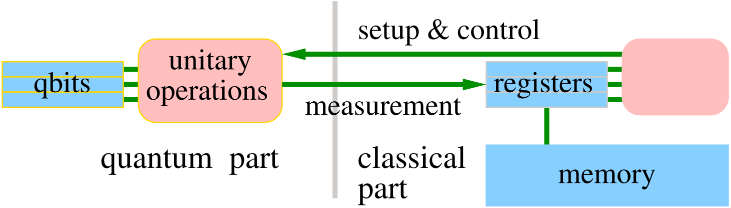
Model LEGO for GPU?

- distributed-memory models for compute nodes
- aCRQW PRAM on each GPU
- broad-word/SIMD parallelism, constant factor things on warp level



Quantum Computing

[Gru99, NC10, Kni96]

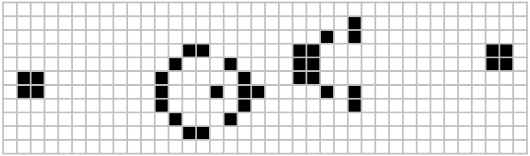


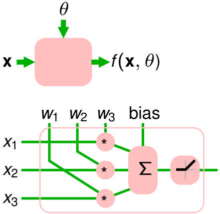
Have size of quantum registers as a parameter?

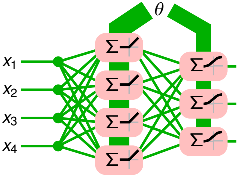
Very different: Quantum annealing – Assume a solver for an NP-hard problem like quadratic unconstrained binary optimization (QUBO)

More from the Zoo

- Reversible computing
- Private computations
- Cellular automata [B+66]
- Molecular computing
- Neural processing
- Peer-to-Peer networks and fault tolerance

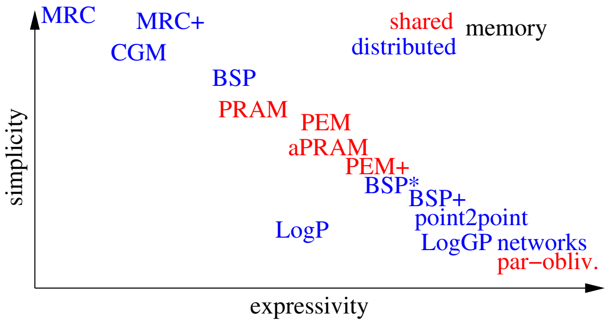






Conclusions

- The zoo is big
- Many models have their point
- But often specifics do not matter
- Tradeoff expressivity versus simplicity



Big Open Problems

Karlsruhe Institute of Technology

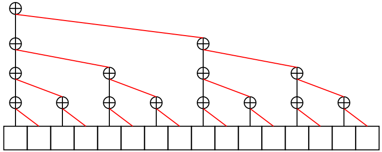
- Classical complexity theory leads to rather coarse grained results.
  Do we need fine-grained complexity of parallel algorithms?
  For example, what about efficient algorithms with span $\mathrm{O}\left(\sqrt{n}\right)$?
- Asymptotics meets hardware design. The quantitative approach [HP17] was highly successful but is hard to extrapolate.
- The asymptotics of physical constraints
  (energy consumption, cooling, wire delays, fault tolerance,...) There was a lot of work on VLSI in the 1980s [Ull84] but that only scratched the surface.
- GPU, quantum, neural
- Would superhuman AIs care about our abstractions?

Exercise: Reduction on aCRQW

Karlsruhe Institute of Technology

Implement $\mathrm{O}(\log p)$ reduction using for the aCRQW model. Perhaps using CAS operations or fetch-and-add.

PE index $i \in \{0, \ldots, n-1\}$
active := 1
for $0 \leq k < \lceil \log n \rceil$ **do**
    (* avoid global barrier here ***)**
    if active **then**
        if bit k of i **then**
            active := 0
        else if $i + 2^k < n$ **then**
            (* ensure that $a[i + 2^k]$ contains the right subtree sum ***)**
            $a[i] := a[i] + a[i + 2^k]$



Exercise: MRC $\rightarrow$ MRC$^+$

Analyze your favorite MRC algorithm using MRC$^+$.
(simple examples: word-count, page-rank).

Sum over all MapReduce steps:

w Total work for μ, ρ

$\hat{w}$ Maximal work for μ, ρ

m Total data volume for $A \cup B \cup C \cup D$

$\hat{m}$ Maximal object size in $A \cup B \cup C \cup D$,

References I

[AB16] Umut A. Acar and Guy Blelloch.
Algorithm design: Parallel and sequential.
2016.
draft.

[AFSW22] Michael Axtmann, Daniel Ferizovic, Peter Sanders, and Sascha Witt.
Engineering in-place (shared-memory) sorting algorithms.
ACM Transaction on Parallel Computing, 9(1):2:1–2:62, 2022.

[AGNS08] Lars Arge, Michael T Goodrich, Michael Nelson, and Nodari Sitchinava.
Fundamental parallel algorithms for private-cache chip multiprocessors.
In *20th Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 197–206. ACM, 2008.

[AISS97] Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman.
LogGP: Incorporating long messages into the LogP model for parallel computation.
Journal of Parallel and Distributed Computing, 44(1):71–79, July 1997.

[AW04] Hagit Attiya and Jennifer Welch.
Distributed computing: fundamentals, simulations, and advanced topics, volume 19.
John Wiley & Sons, 2004.

[B+66] Arthur W Burks et al.
Theory of self-reproducing automata.
IEEE Transactions on Neural Networks, 5(1):3–14, 1966.

[BAJ+16] T. Bingmann, M. Axtmann, E. Jöbstl, S. Lamm, H. Chau Nguyen, A. Noe, S. Schlag, M. Stumpp, T. Sturm, and P. Sanders.
Thrill: High-Performance Algorithmic Distributed Batch Data Processing with C++.
ArXiv e-prints, August 2016.

[BDadH95] Armin Bäumker, Wolfgang Dittrich, and Friedhelm Meyer auf der Heide.
Truly efficient parallel algorithms: c-optimal multisearch for an extension of the bsp model.
In *European Symposium on Algorithms*, pages 17–30. Springer, 1995.

References II

[BFGS20] Guy E Blelloch, Jeremy T Fineman, Yan Gu, and Yihan Sun.
Optimal parallel algorithms in the binary-forking model.
In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 89–102, 2020.

[BL99] R. D. Blumofe and C. E. Leiserson.
Scheduling multithreaded computations by work stealing.
Journal of the ACM, 46(5):720–748, 1999.

[BOS+13] Marcel Birn, Vitaly Osipov, Peter Sanders, Christian Schulz, and Nodari Sitchinava.
Efficient parallel and external matching.
In *Euro-Par*, volume 8097 of *LNCS*, pages 659–670. Springer, 2013.

[CHPvdG07] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert van de Geijn.
Collective communication: theory, practice, and experience.
Concurrency and Computation: Practice and Experience, 19(13):1749–1783, 2007.

[CKP+93] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. v. Eicken.
LogP: Towards a realistic model of parallel computation.
In *Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 1–12, San Diego, CA, 19–22 May, 1993. ACM, New York.

[DG08] Jeffrey Dean and Sanjay Ghemawat.
Mapreduce: simplified data processing on large clusters.
Commun. ACM, 51:107–113, January 2008.

[FL94] Pierre Fraigniaud and Emmanuel Lazard.
Methods and problems of communication in usual networks.
Discrete Applied Mathematics, 53(1–3):79–133, 1994.

[FLM+19] Daniel Funke, Sebastian Lamm, Ulrich Meyer, Manuel Penschuck, Peter Sanders, Christian Schulz, Darren Strash, and Moritz von Looz.
Communication-free massively distributed graph generation.
J. of Parallel and Distributed Computing, 131:200–217, 2019.
Conference version at IPDPS 2018, best paper award.

References III

[GMR98] P. B. Gibbons, Y. Matias, and V. Ramachandran.
The queue-read queue-write pram model: Accounting for contention in parallel algorithms.
SIAM J. Comput., 28(2):733–769, 1998.

[GMR99] Phillip B Gibbons, Yossi Matias, and Vijaya Ramachandran.
Can a shared-memory model serve as a bridging model for parallel computation?
Theory of Computing Systems, 32(3):327–359, 1999.

[Gru99] Jozef Gruska.
Quantum computing, volume 2005.
McGraw-Hill London, 1999.

[HP17] J. L. Hennessy and D. A. Patterson.
Computer Architecture a Quantitative Approach.
Morgan Kaufmann, 6th edition, 2017.

[HS16] Lorenz Hübschle-Schneider and Peter Sanders.
Communication efficient algorithms for top-k selection problems.
In *30th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2016.

[HS22] Lorenz Hübschle-Schneider and Peter Sanders.
Parallel weighted random sampling.
ACM Transaction on Mathematical Sofware, 48(3):29:1–29:40, 2022.

[Kni96] Emmanuel Knill.
Conventions for quantum pseudocode.
Technical report, Los Alamos National Lab., NM (United States), 1996.

[KS06] K. Kaligosi and P. Sanders.
How branch mispredictions affect quicksort.
In *14th European Symposium on Algorithms (ESA)*, volume 4168 of *LNCS*, pages 780–791, 2006.

References IV

[KSB06] J. Kärkkäinen, P. Sanders, and S. Burkhardt.
Linear work suffix array construction.
Journal of the ACM, 53(6):1–19, 2006.

[KSV10] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii.
A model of computation for MapReduce.
In *21st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 938–948, 2010.

[Lei92] T. Leighton.
Introduction to Parallel Algorithms and Architectures.
Morgan Kaufmann, 1992.

[Lub86] M. Luby.
A simple parallel algorithm for the maximal independent set problem.
SIAM Journal on Computing, 15(4):1036–1053, 1986.

[MS03a] K. Mehlhorn and P. Sanders.
Scanning multiple sequences via cache memory.
Algorithmica, 35(1):75–93, 2003.

[MS03b] U. Meyer and P. Sanders.
Δ-stepping: A parallelizable shortest path algorithm.
Journal of Algorithms, 49(1):114–152, 2003.
also in ESA 99, test-of-time award 2019.

[MSD19] Tobias Maier, Peter Sanders, and Roman Dementiev.
Concurrent hash tables: Fast and general(?)!
ACM Trans. Parallel Comput., 5(4):16:1–16:32, 2019.

[NC10] Michael A Nielsen and Isaac L Chuang.
Quantum Computation and Quantum Information.
Cambridge University Press, 2010.

References V

[Pel00] David Peleg.
Distributed computing: a locality-sensitive approach.
SIAM, 2000.

[Ray13] Michel Raynal.
Distributed algorithms for message-passing systems, volume 500.
Springer, 2013.

[San98] P. Sanders.
Random permutations on distributed, external and hierarchical memory.
Information Processing Letters, 67(6):305–310, 1998.

[San20] Peter Sanders.
Connecting mapreduce computations to realistic machine models.
In *IEEE Conference on Big Data*, pages 84–93, 2020.

[SLH+16] Peter Sanders, Sebastian Lamm, Lorenz Hübschle-Schneider, Emanuel Schrade, and Carsten Dachsbacher.
Efficient random sampling – parallel, vectorized, cache-efficient, and online.
CoRR, abs/1610.05141, 2016.

[SLH+18] Peter Sanders, Sebastian Lamm, Lorenz Hübschle-Schneider, Emanuel Schrade, and Carsten Dachsbacher.
Efficient random sampling – parallel, vectorized, cache-efficient, and online.
ACM Transaction on Mathematical Sofware, 44(3):29:1–29:14, 2018.

[SMDD19] Peter Sanders, Kurt Mehlhorn, Martin Dietzfelbinger, and Roman Dementiev.
Sequential and Parallel Algorithms and Data Structures – The Basic Toolbox.
Springer, 2019.

[SS63] J. C. Shepherdson and H. E. Sturgis.
Computability of recursive functions.
Journal of the ACM, 10(2):217–255, 1963.

References VI

[SS24] Peter Sanders and Darren Strash.
Algorithm Engineering, chapter Machine Models.
2024.
to appear, draft at https://ae.iti.kit.edu/documents/people/sanders/aeModels.pdf.

[SSM13] Peter Sanders, Sebastian Schlag, and Ingo Müller.
Communication efficient algorithms for fundamental big data problems.
In *IEEE Int. Conf. on Big Data*, pages 15–23, 2013.

[SW04] P. Sanders and S. Winkel.
Super scalar sample sort.
In *12th European Symposium on Algorithms*, volume 3221 of *LNCS*, pages 784–796. Springer, 2004.

[Ull84] J. D. Ullman.
Computational Aspects of VLSI.
Computer Science Press, 1984.

[Val94] L. Valiant.
A bridging model for parallel computation.
Communications of the ACM, 33(8):103–111, 1994.

[ZCF+10] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica.
Spark: Cluster computing with working sets.
In *2nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, 2010.