Introduction to Bioinformatics for Computer Scientists

Lecture 4

Mailing list

- Please send an email to Alexandros.Stamatakis@kit.edu
- Mail of Angelina → misspelled

Terminology introduced last time

- Transcriptome
- Meta-Genome
  - Meta-Genomics
  - Meta-Genetics
- Cromosome
  - Allele
  - Haploid *versus* diploid chromosomes
- Species
- Taxonomy

Terminology introduced last time

- Phylogeny/phylogenetic tree
  - Ingroup *versus* outgroup → tree rooting
  - Clade, Lineage, Subtree, Taxon
  - Extant species
  - Ancestral nodes

Outline for today

- Alexis
  - 1000 insect transcriptome project → how were the sequences extracted
  - Some more slides on the questionnaire
- Tomas/Solon
  - Blast & pair-wise sequence alignment

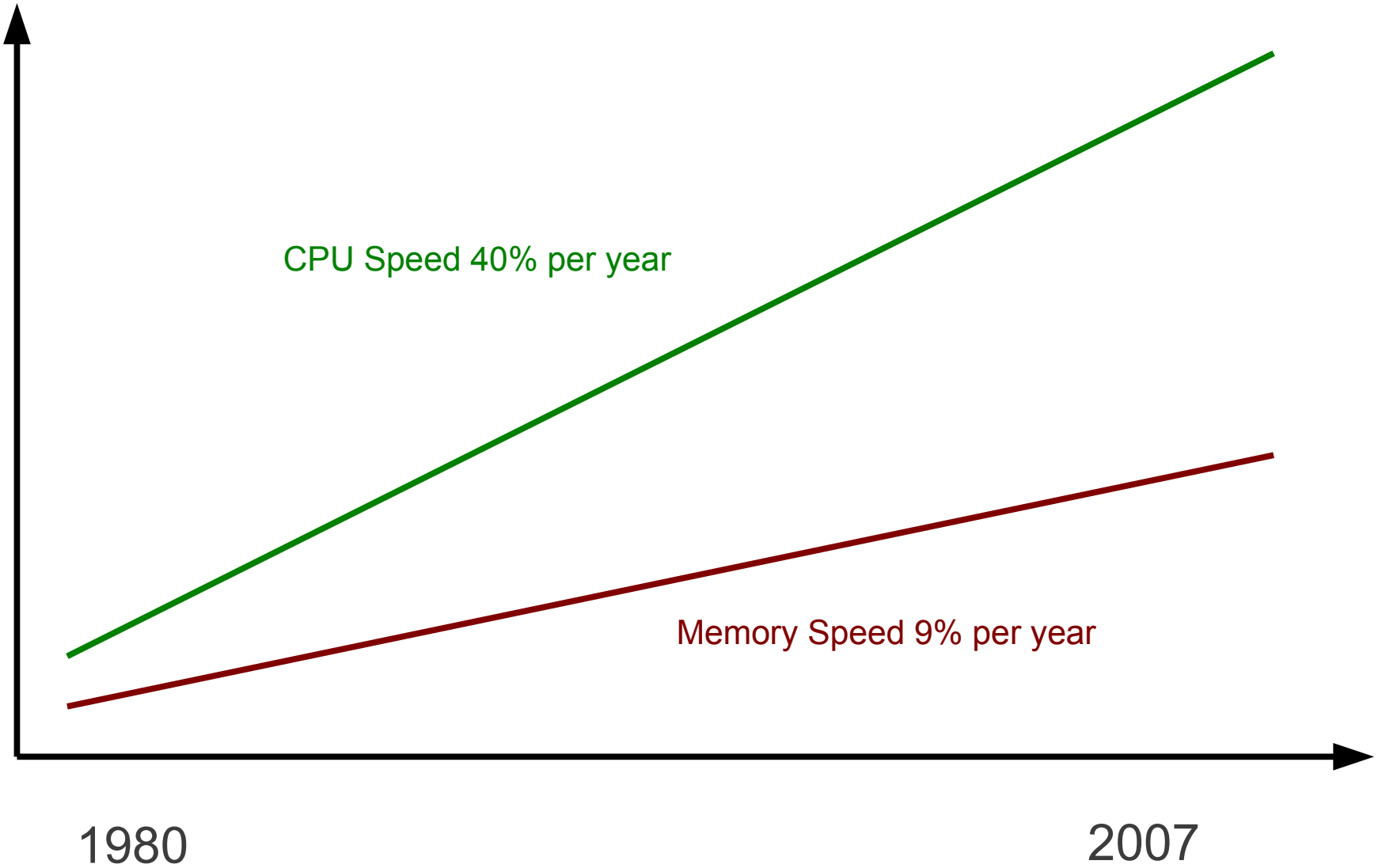
1KITE: Transcriptome Sequencing

- Remember
  - Transcriptome → all transcribed RNA
  - Depends on time & location
- 1KITE www.1kite.org
  - Tried to keep stress levels “normal” such as not to perturb gene *expression*
  - Small insects → the entire animal
    - sometimes used more than one individual to obtain enough RNA
  - Large insects → head & thorax
  - Very large insects → only tissue “Gewebe” to avoid contamination by the gut contents (**Bacterial RNA!**)
  - In general: tried to avoid sampling specific cells only (e.g., muscular cells)
  - First check → apparently good coverage of the transcriptome
- Acknowledgement: Dr. Karen Meusemann (U. Bonn)

Back to the questionnaire

- We got to OpenMP last time

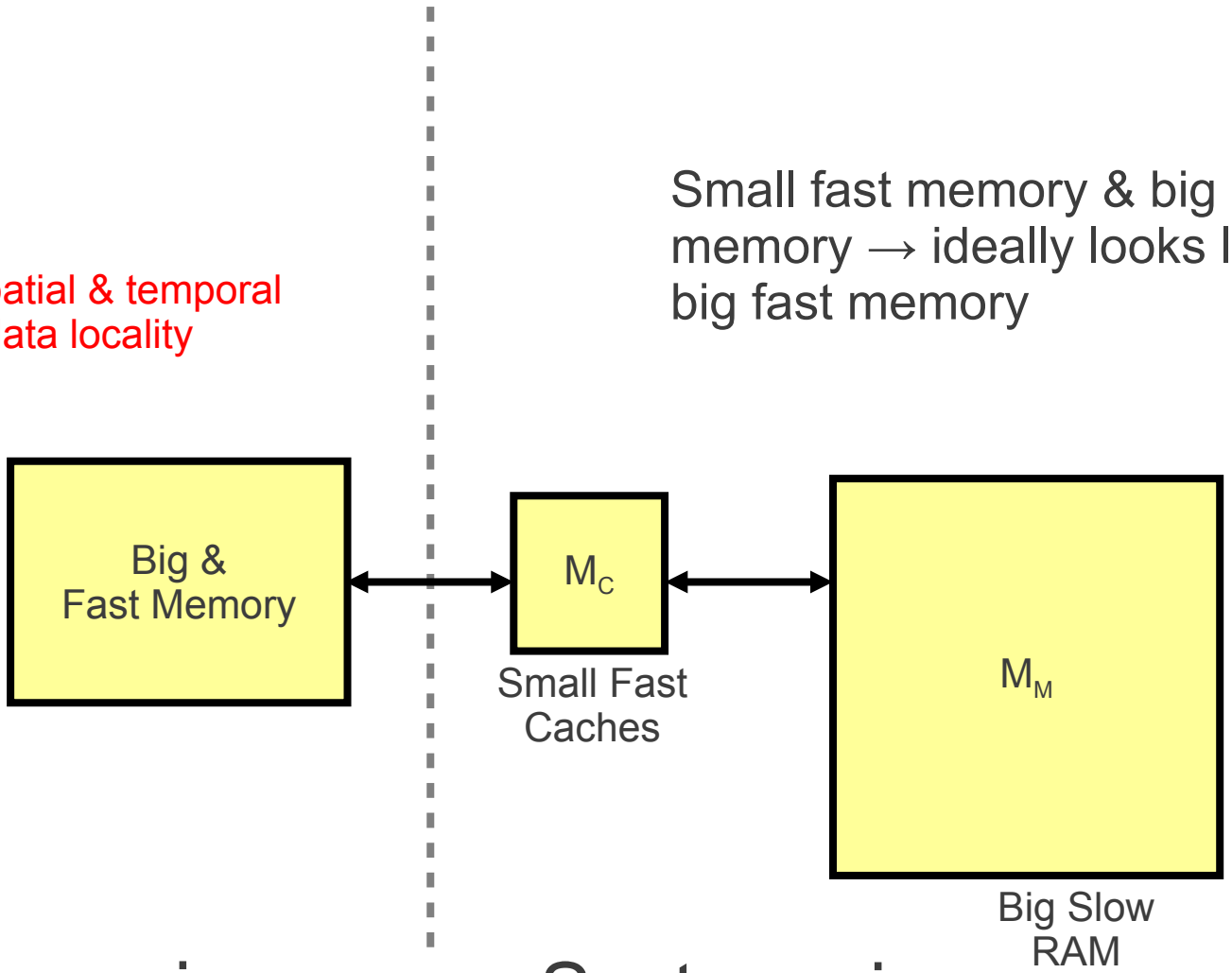
The Memory Gap



What we hope for

Given spatial & temporal code & data locality

Small fast memory & big slow memory → ideally looks like a big fast memory



User view

System view

Principles of Cache Memories & Paging Algorithms

- What I wanted to hear:
  - Temporal locality of data accesses
  - Spatial locality of data accesses
- The term cache comes from the French verb cacher (hiding)

  → Caches hide the memory latency
- If temporal and spatial data locality are not given, caches have no effect

  → random memory access patterns

  → e.g., accessing hash tables!

Cache Coherence

- In a shared memory system with multiple caches
- How do we make sure that copies of the same address *x* in RAM that may reside in two or more caches are consistent when one cache copy is changed?

  → cache coherency protocols
- We often talk about ccNUMA

  → cache-coherent NUMA

Measuring Parallel Performance

- *Speedup*: S = T(1)/T(p)
- It can't be stated frequently enough

  → T(1) actually refers to the fastest sequential implementation/algorithm available, everything else should be reported as **relative** speedup!
- *Scalability*: Up to how many CPUs do we get good speedups?
- We distinguish between:

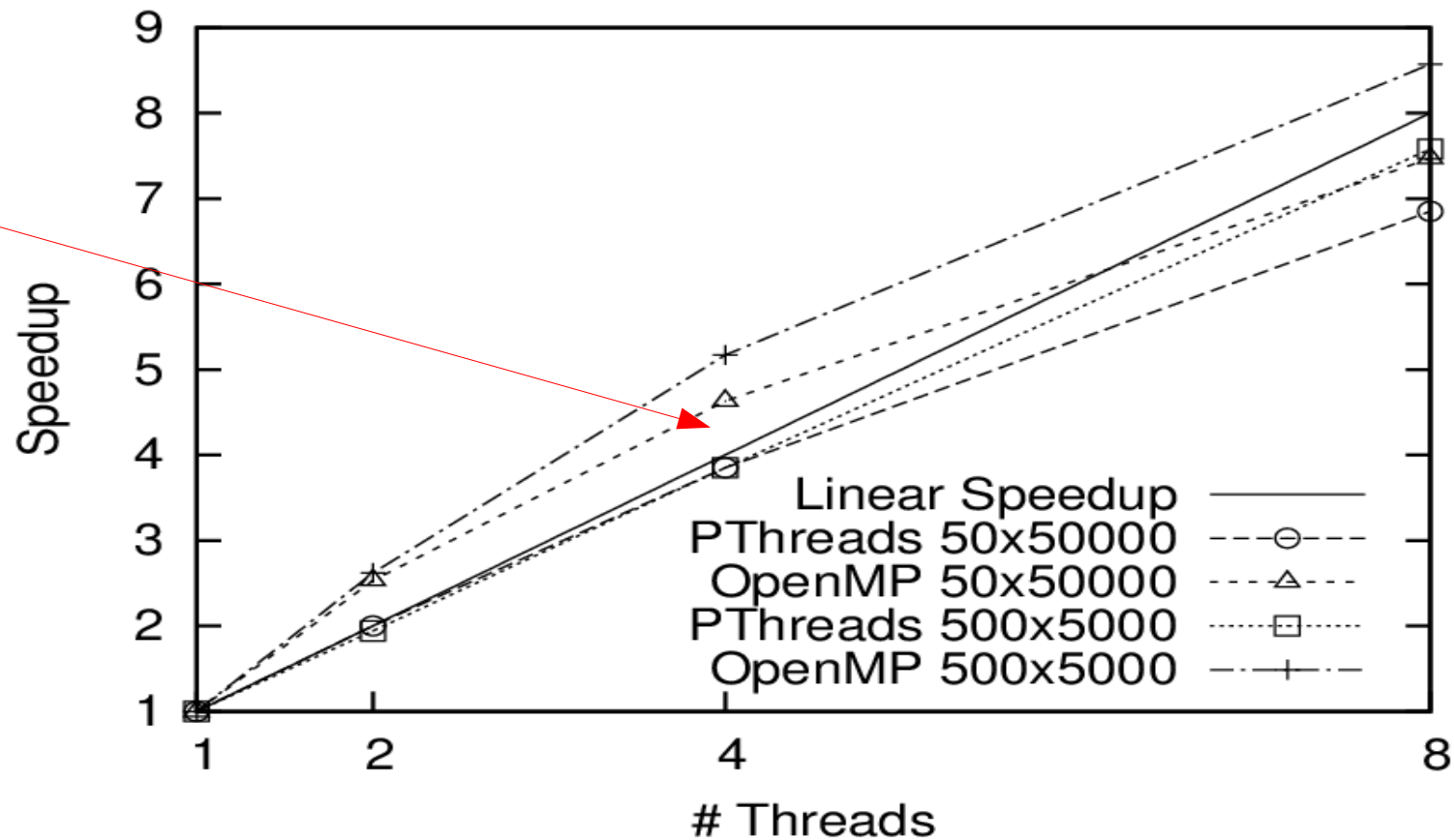
  → *weak scaling*: scale up the problem size as we add cores

  → *strong scaling*: keep problem size fixed as we add cores

Super-linear speedups

- Reducing the per-CPU memory footprint of parallel programs via data distribution can yield super-linear speedups due to increased cache efficiency due to increased total cache size
- Below: RAxML on an AMD Barcelona multi-core system:

Super-linear speedups



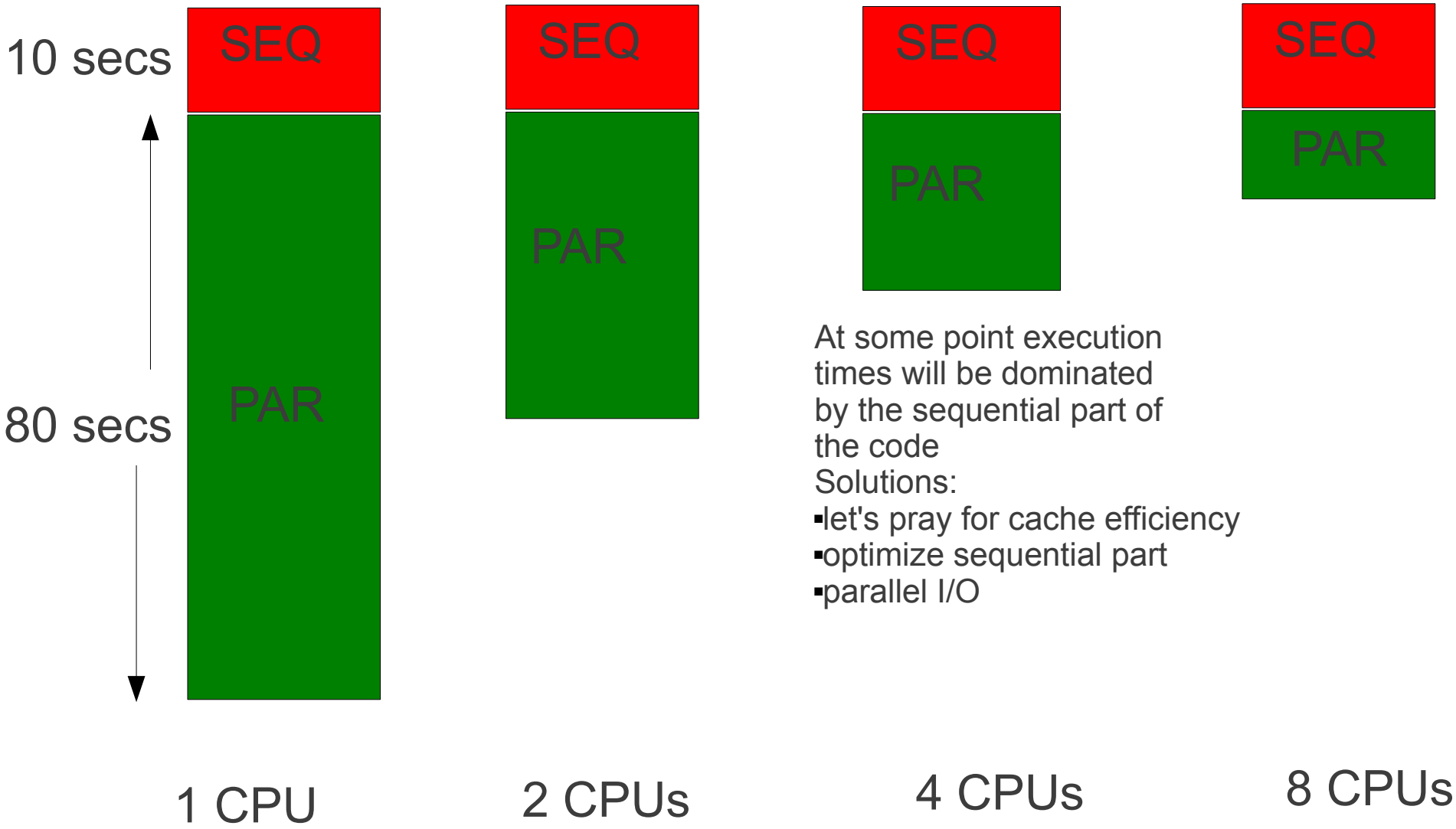
Example

- Sequential code total memory footprint of 20 MB
- Assume a machine with 2 cores and two caches of 10MB each

  → if we use one core, the working set of 20MB will not fit into cache

  → if we use two cores, the working set of 20 MB will entirely fit into the two caches

Amdahl's Law



At some point execution times will be dominated by the sequential part of the code

Solutions:

- let's pray for cache efficiency
- optimize sequential part
- parallel I/O

Amdahl's Law

- Scalability to large number of processors is limited by sequential part of program
- Every program has a sequential portion, even if it is just the time needed to start all the threads or MPI processes!
- More formally: Speedup $\leq 1 /(f + ((1-f)/p)))$ where f is the fraction f with $0 \leq f \leq 1.0$ of the program that must be executed sequentially
- Thus for $p \rightarrow$ infinity the maximum speedup $S\_max \leq 1/f$
- If the fraction f is 0.01 we get $S\_max \leq 100$:-(
- IMPORTANT: here we are assuming linear speedups for the part that can be parallelized!
- Solutions
  - Hope that f is small
  - Make f small
  - May be counter-balanced by cache-efficiency!

Twelve Ways to Fool the Masses when Giving Performance Results on Parallel Computers by David H. Bailey

Who cares about Amdahl's law anyway:

2. Present performance figures for an inner kernel, and then represent these figures as the performance of the entire application.

It is quite difficult to obtain high performance on a complete large-scale scientific application, timed from beginning of execution through completion. There is often a great deal of data movement and initialization that depresses overall performance rates. A good solution to this dilemma is to present results for an inner kernel of an application, which can be souped up with artificial tricks. Then imply in your presentation that these rates are equivalent to the overall performance of the entire application.

Twelve Ways to Fool the Masses when Giving Performance Results on Parallel Computers
by David H. Bailey

Who cares about Amdahl's law anyway:

2. Present performance figures for an inner kernel, and then represent these figures as the performance of the entire application.

It is quite difficult to obtain high performance on a complete large-scale scientific application, timed from beginning of execution through completion. There is often a great deal of data movement and initialization that depresses overall performance rates. A good solution to this dilem is to present results for an inner kernel of an application, whic e souped up with artificial tricks. Then imply in your prese t these rates are equivalent to the overall performan application.

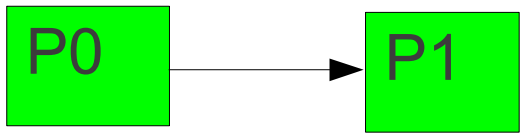
Full list at:
http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/twelve-ways.pdf

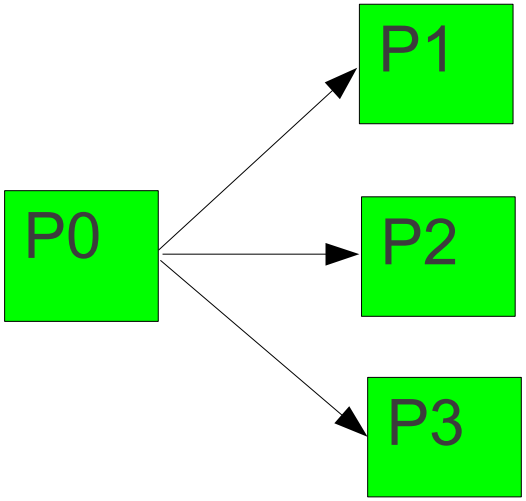
MPI API for parallelizing Code on **Distributed Memory Architectures**

- **M**essage **P**assing **I**nterface
- Send-Receive Paradigm
- Point to Point Communication
- Collective Communication

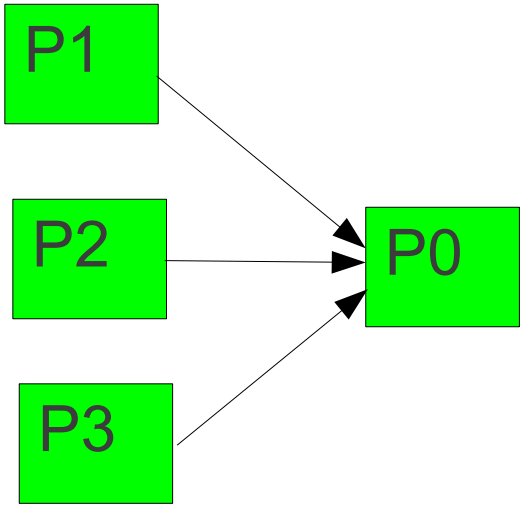
Communication Schemes

Direct/point-to-point





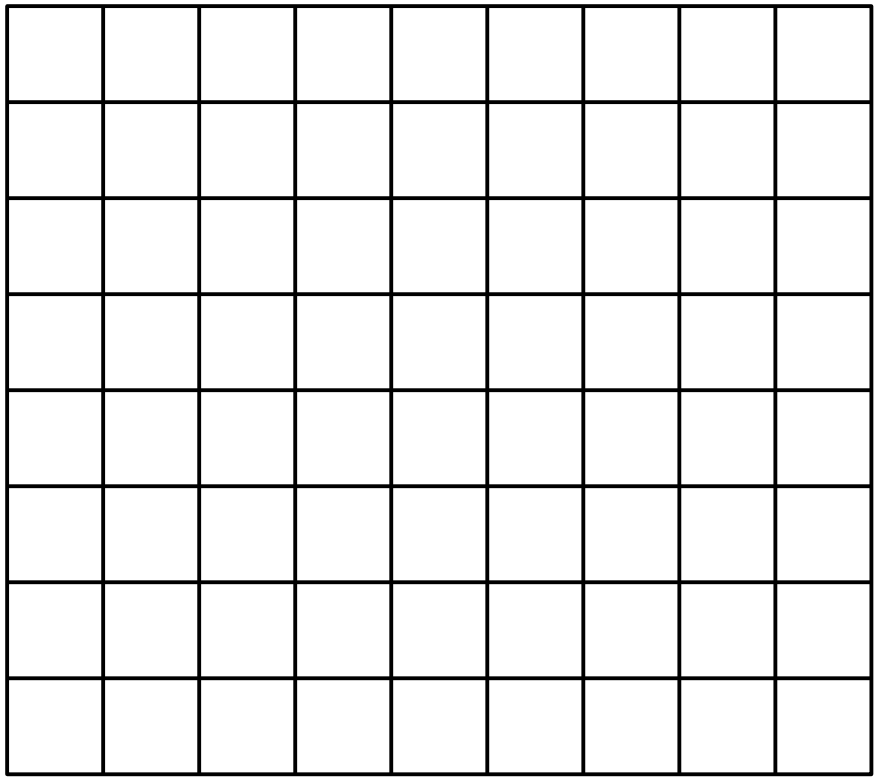
broadcast/multicast



indirect/reduction

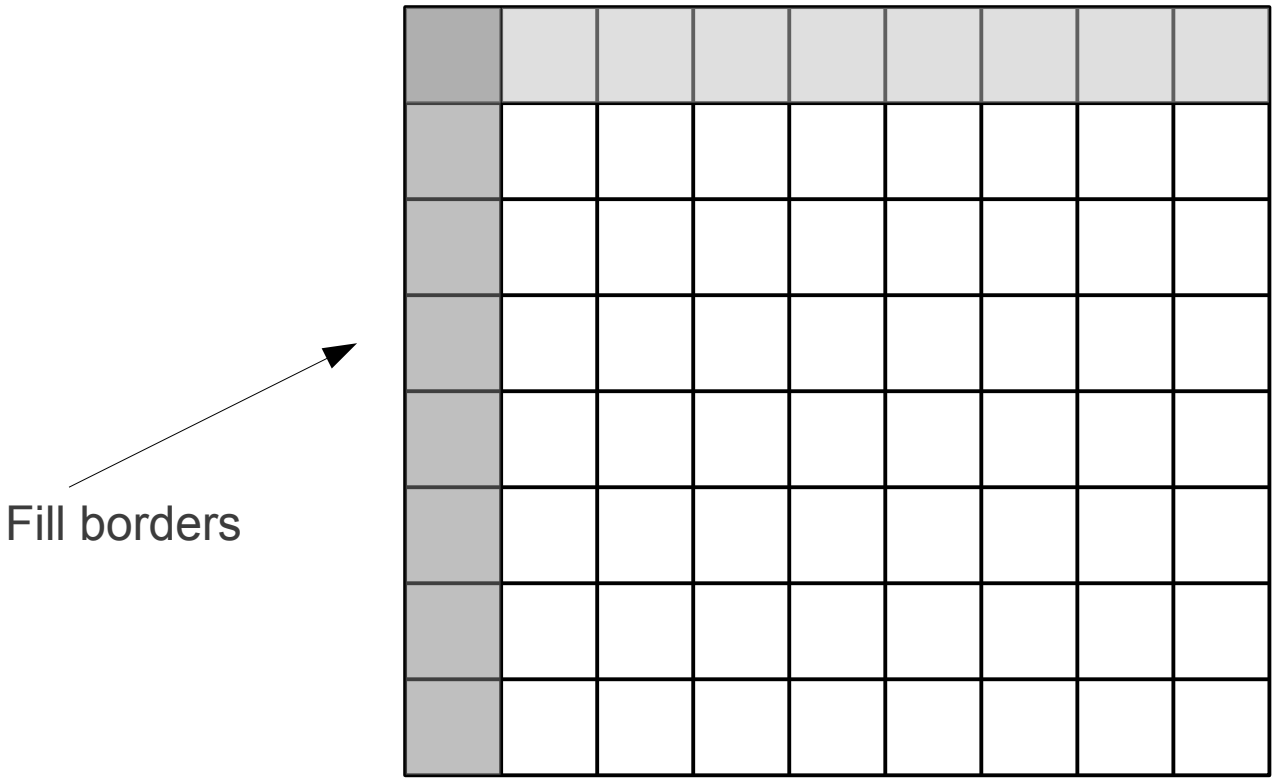
Wave-Front Parallelism

- Dynamic Programming algorithms



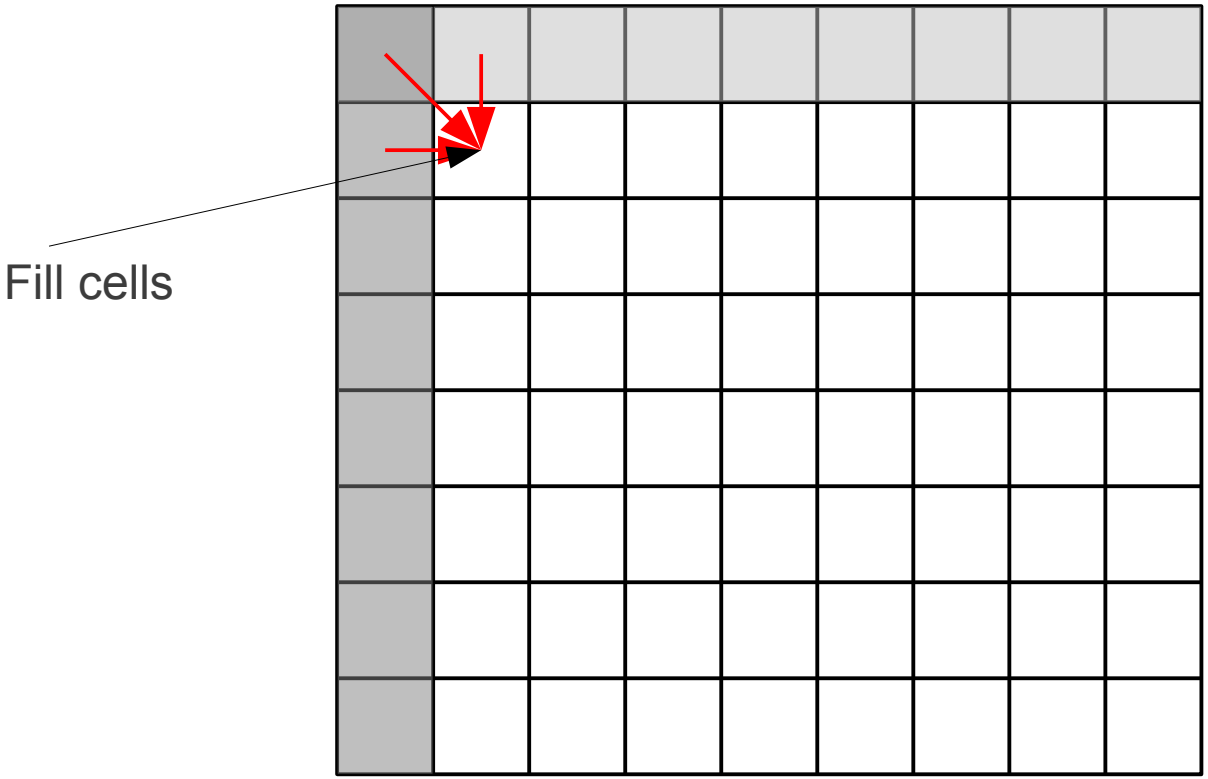
Wave-Front Parallelism

- Dynamic Programming algorithms



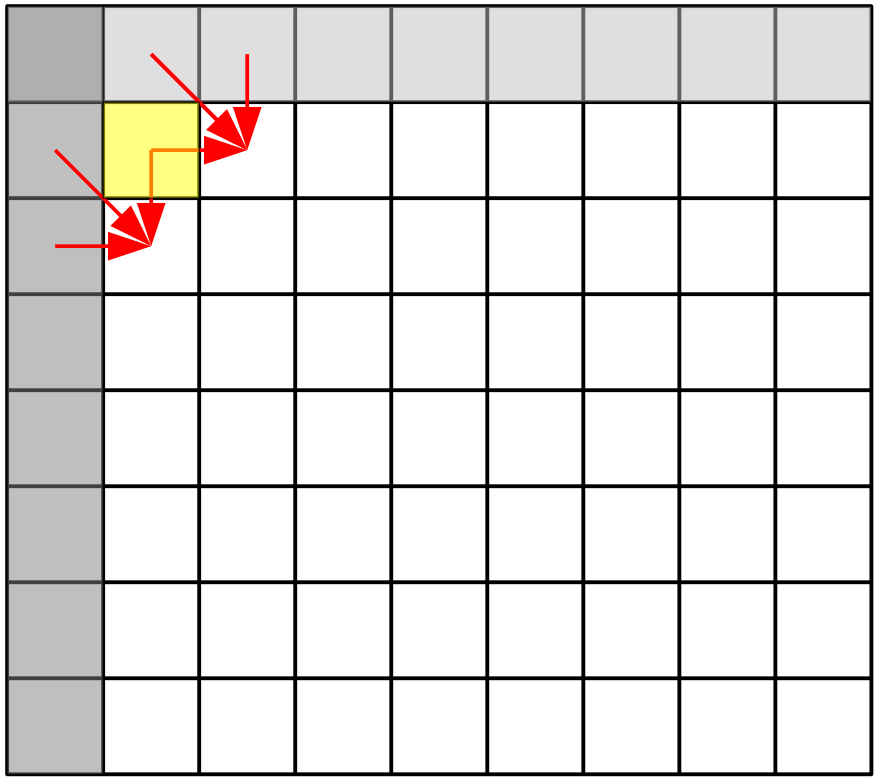
Wave-Front Parallelism

- Dynamic Programming algorithms



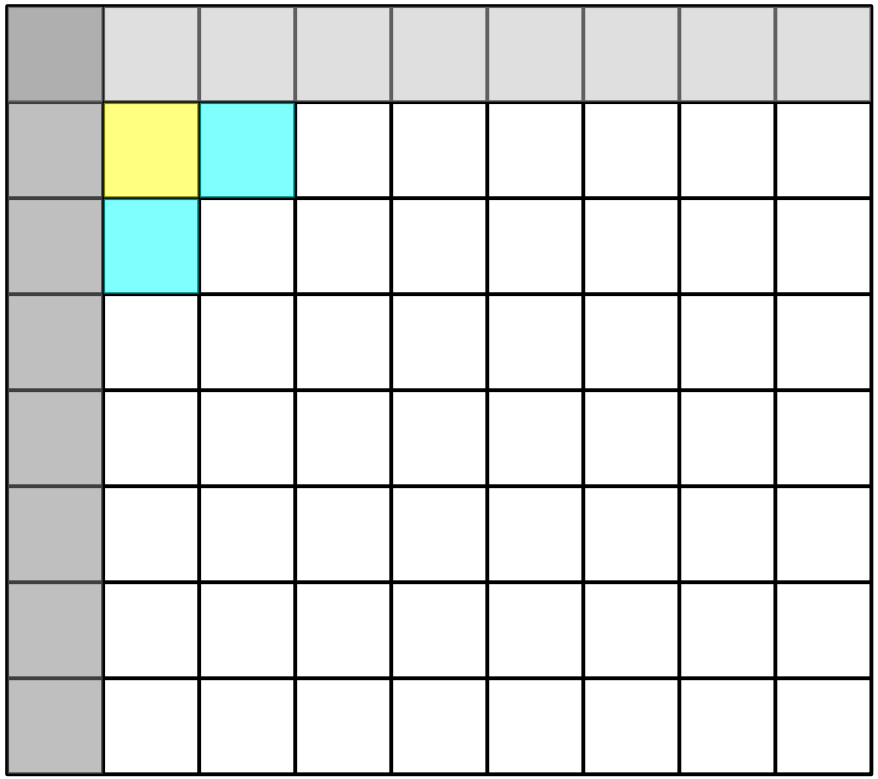
Wave-Front Parallelism

- Dynamic Programming algorithms



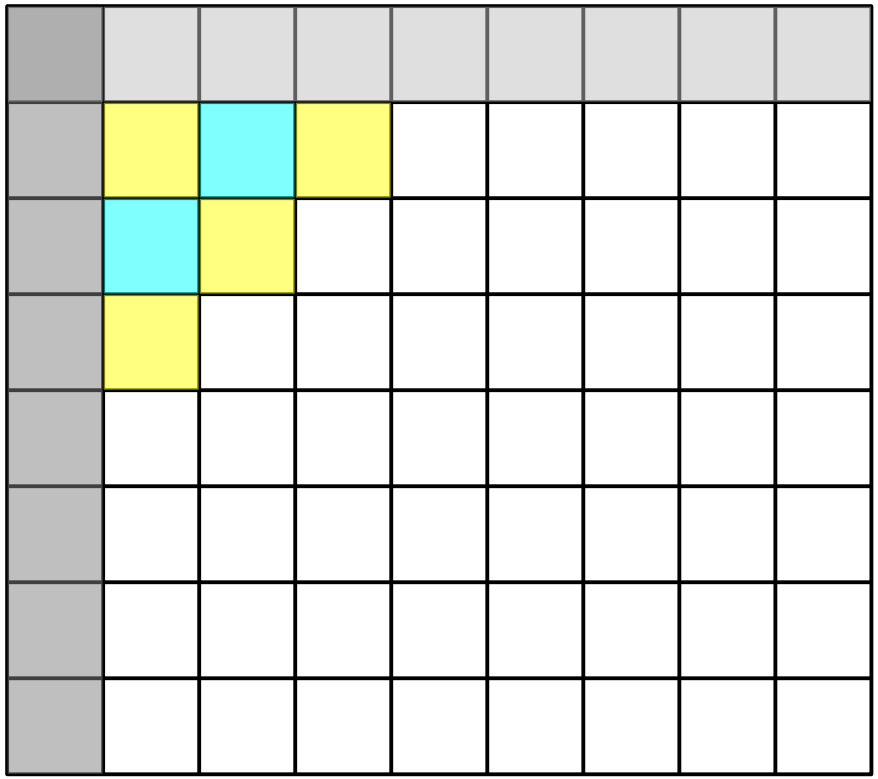
Wave-Front Parallelism

- Dynamic Programming algorithms



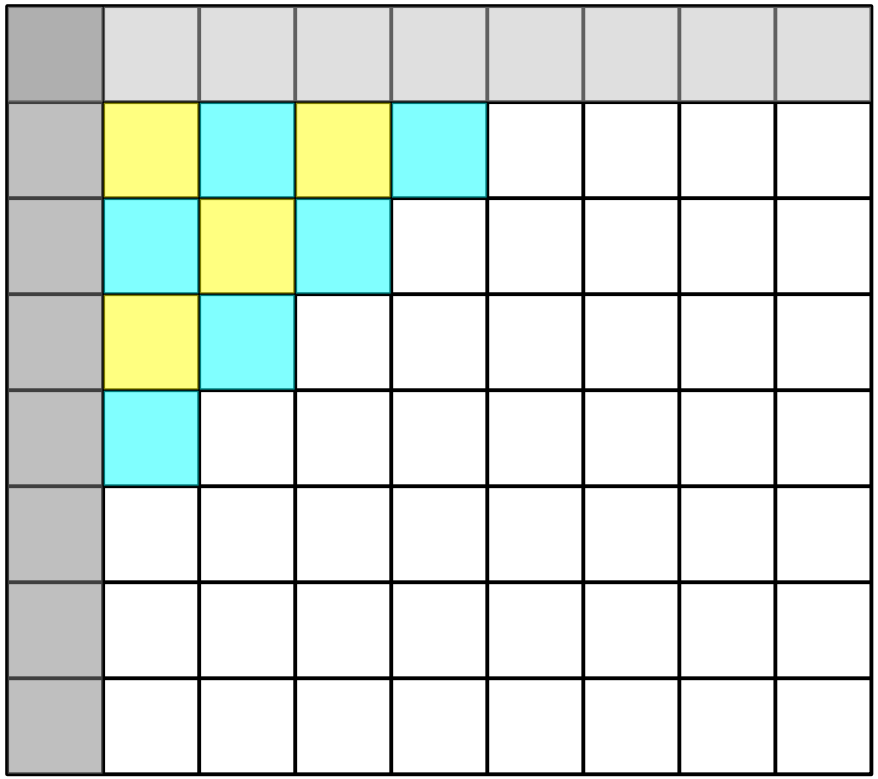
Wave-Front Parallelism

- Dynamic Programming algorithms



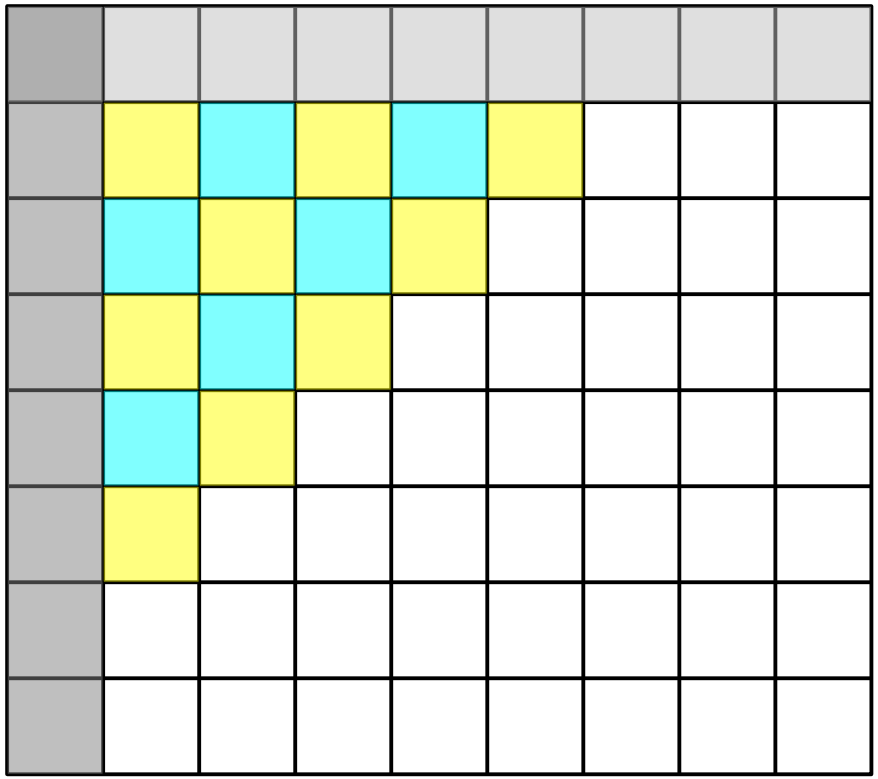
Wave-Front Parallelism

- Dynamic Programming algorithms



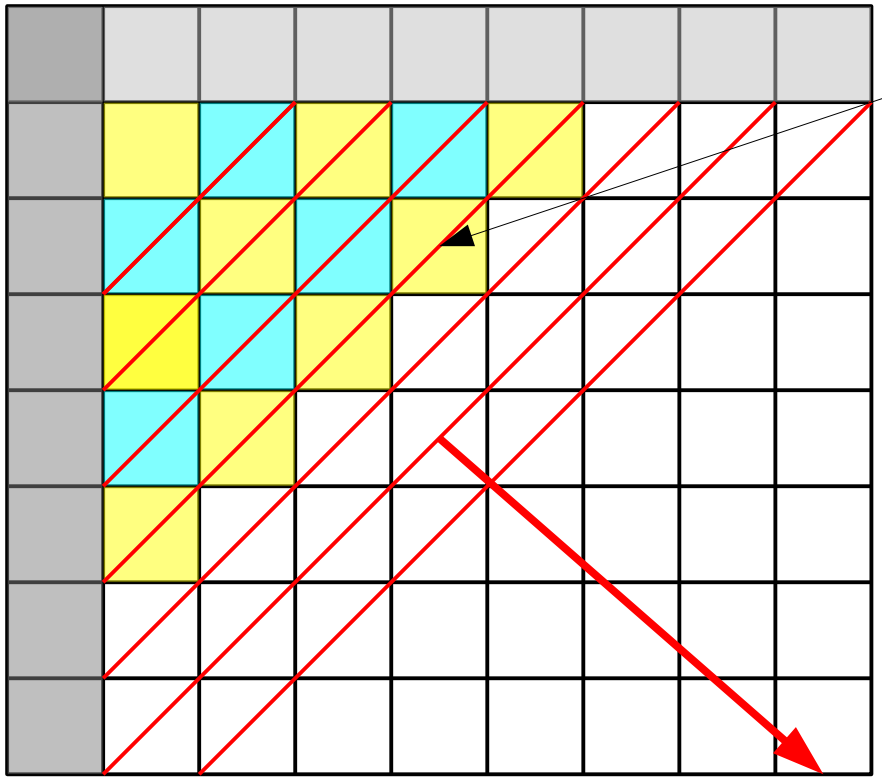
Wave-Front Parallelism

- Dynamic Programming algorithms



Wave-Front Parallelism

- Dynamic Programming algorithms



Number of cells
That can be computed
In parallel proceeds
Like a wave-front through
The matrix

Hash Tables

- Map a universe of keys U to a much smaller integer-based index table
- Lookup of elements in O(1)
- Challenge: define hash function
  - → such that: it is fast to compute
  - → such that: it does not map all keys to the same integer
- Handling collisions: two distinct keys are mapped to the same integer
  - → resolve collisions by chaining
  - → resolve collisions by re-hashing
- A sequence of hash table lookups will generally produce a sequence of random memory accesses

Binary Searches

- Search by comparison & bisection
- Able to find an element in O(log n)

Kruskal & Prim

- Both invented a minimum spanning tree algorithm!

Blast & Sequence Alignment