

Introduction to Bioinformatics for Computer Scientists

Knowledge Quiz
&
Current Research

A review of the Questionnaire

- HPC
- Algorithms
- And some comments on programming in general

HPC: some remarks

- HPC is nice
 - better algorithms are nicer
 - ... and more energy-efficient
- Before parallelizing
 - optimize the sequential code first
 - frequently not done when comparing x86 code to some GPU or FPGA implementation
 - learn the basics, before playing around with parallel HW

HPC: some remarks

- HPC is nice
 - better algorithms are nicer
 - ... and more energy-efficient
- **BUT:** There is the Jevons Paradox

Innovations

New Results

Comment on this paper

A Fast and Memory-Efficient Implementation of the Transfer Bootstrap

Sarah Lutteropp, Alexey M. Kozlov, Alexandros Stamatakis

doi: <https://doi.org/10.1101/734848>

480x speedup

CORRECTED PROOF

RAxML-NG: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference

Alexey M Kozlov, Diego Darriba, Tomáš Flouri, Benoit Morel, Alexandros Stamatakis

Bioinformatics, btz305, <https://doi.org/10.1093/bioinformatics/btz305>

Published

4x speedup

EPA-ng: Massively Parallel Evolutionary Placement of Genetic Sequences

Pierre Barbera, Alexey M Kozlov, Lucas Czech, Benoit Morel, Diego Darriba, Tomáš Flouri, Alexandros Stamatakis

Systematic Biology, Volume 67, Issue 1, 1 June 2018, Pages 167–177, <https://doi.org/10.1093/sysbio/syy054>

30x speedup

Multi-rate Poisson tree processes for single-locus species delimitation under maximum likelihood and Markov chain Monte Carlo

P Kapli, S Lutteropp, J Zhang, K Kobert, P Pavlidis, A Stamatakis, T Flouri

Bioinformatics, Volume 33, Issue 11, 1 June 2017, Pages 1627–1638, <https://doi.org/10.1093/bioinformatics/btx025>

Published: 20 January 2017

~1000x speedup

Oh, wow, this will help save a lot of energy!

The Jevons Paradox

New Results

Comment on this paper

A Fast and Memory-Efficient Implementation of the Transfer Bootstrap

Sarah Lutteropp, Alexey M. Kozlov, Alexandros Stamatakis

doi: <https://doi.org/10.1101/734848>

480x speedup

EPA-ng: Massively Parallel Evolutionary Placement of Genetic Sequences

Pierre Barbera, Alexey M Kozlov, Lucas Czech, Benoit Morel, Diego Darriba, Tomáš Flouri, Alexandros Stamatakis

Systematic Biology, Volume 67, Issue 1, 1 June 2018, Pages 1093–1104, <https://doi.org/10.1093/sysbio/syy054>

30x speedup

CORRECTED PROOF

RAxML-NG: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference

Alexey M Kozlov, Diego Darriba, Tomáš Flouri, Benoit Morel, Alexandros Stamatakis

Bioinformatics, Volume 33, Issue 11, 1 June 2017, Pages 1637–1638, <https://doi.org/10.1093/bioinformatics/btx025>

4x speedup

Multi-rate Poisson tree processes for single-locus species delimitation under maximum likelihood and Markov chain Monte Carlo

P Kapli, S Lutteropp, J Zhang, K Kobert, P Pavlidis, A Stamatakis, T Flouri

Bioinformatics, Volume 33, Issue 11, 1 June 2017, Pages 1637–1638, <https://doi.org/10.1093/bioinformatics/btx025>

Published: 20 January 2018

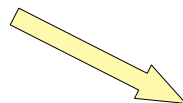
~1000x speedup

W. S. Jevons „The Coal Question“ (1865)

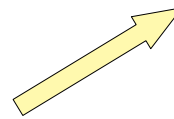
Improved efficiency



Increased consumption rate



Lower cost



The Jevons Paradox

New Results

Comment on this paper

A Fast and Memory-Efficient Implementation of the Transfer Bootstrap

Sarah Lutteropp, Alexey M. Kozlov, Alexandros Stamatakis

doi: <https://doi.org/10.1101/734848>

480x speedup

EPA-ng: Massively Parallel Evolutionary Placement of Genetic Sequences

Pierre Barbera, Alexey M Kozlov, Lucas Czech, Benoit Morel, Diego Darriba, Tomáš Flouri, Alexandros Stamatakis

Systematic Biology, Volume 66, Issue 1, 1 January 2017, Pages 107–116, <https://doi.org/10.1093/sysbio/syy054>

30x speedup

CORRECTED PROOF

RAxML-NG: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference

Alexey M Kozlov, Diego Darriba, Tomáš Flouri, Benoit Morel, Alexandros Stamatakis

Bioinformatics, Volume 33, Issue 11, 1 June 2017, Pages 1677–1681, <https://doi.org/10.1093/bioinformatics/btz305>

Published

4x speedup

Multi-rate Poisson tree processes for single-locus species delimitation under maximum likelihood and Markov chain Monte Carlo

P Kapli, S Lutteropp, J Zhang, K Kobert, P Pavlidis, A Stamatakis, T Flouri

Bioinformatics, Volume 33, Issue 11, 1 June 2017, Pages 1637–1638, <https://doi.org/10.1093/bioinformatics/btx025>

Published: 20 January 2017

~1000x speedup

We need an alternative solution!

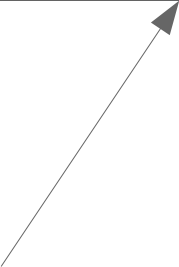
Energy monitoring: RAxML - NG

- New in RAxML - NG v1.0: energy usage report

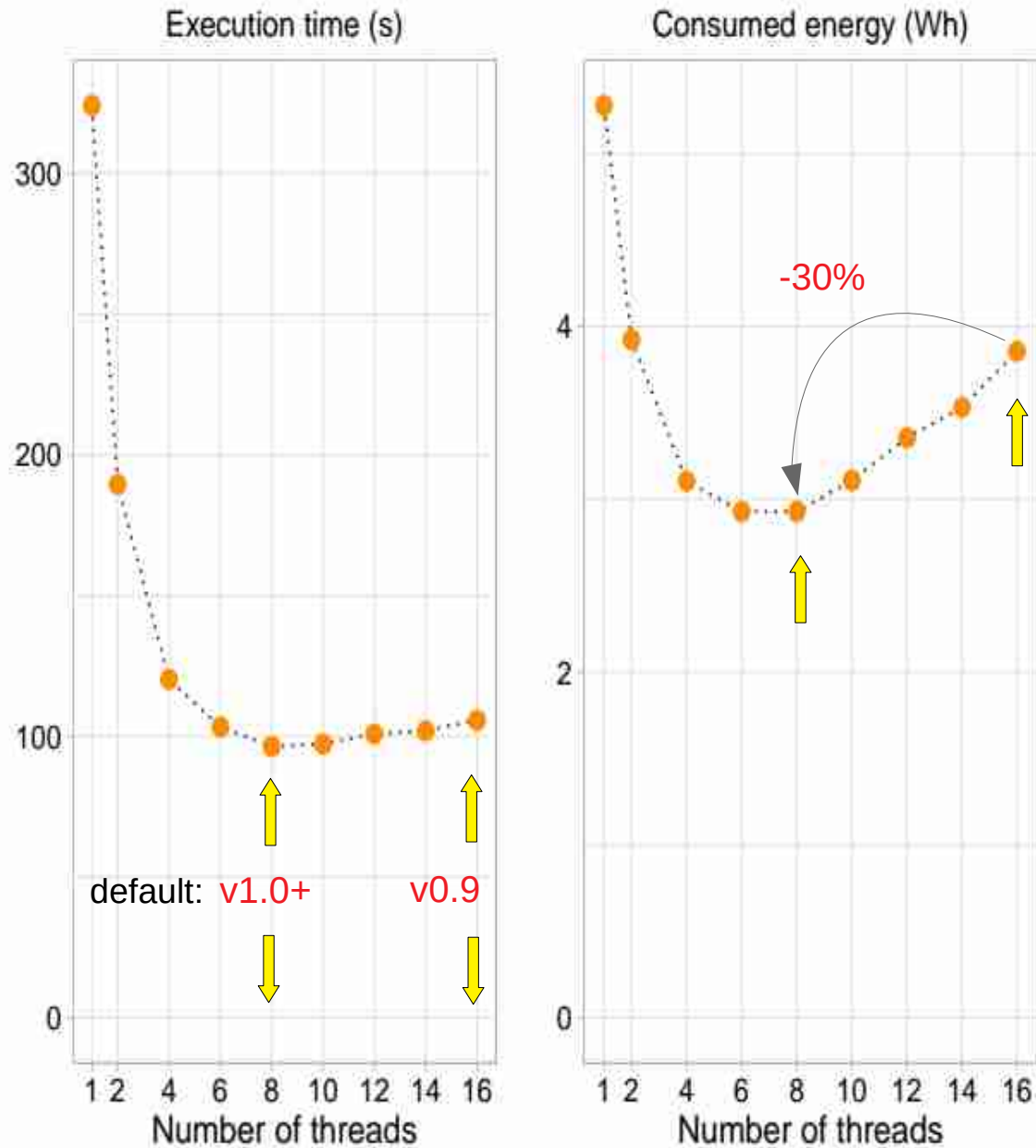
Elapsed time: 42846.287 seconds

Consumed energy: 162370.469 Wh (= 812 km in an electric car, or 4059 km with an e-scooter!)

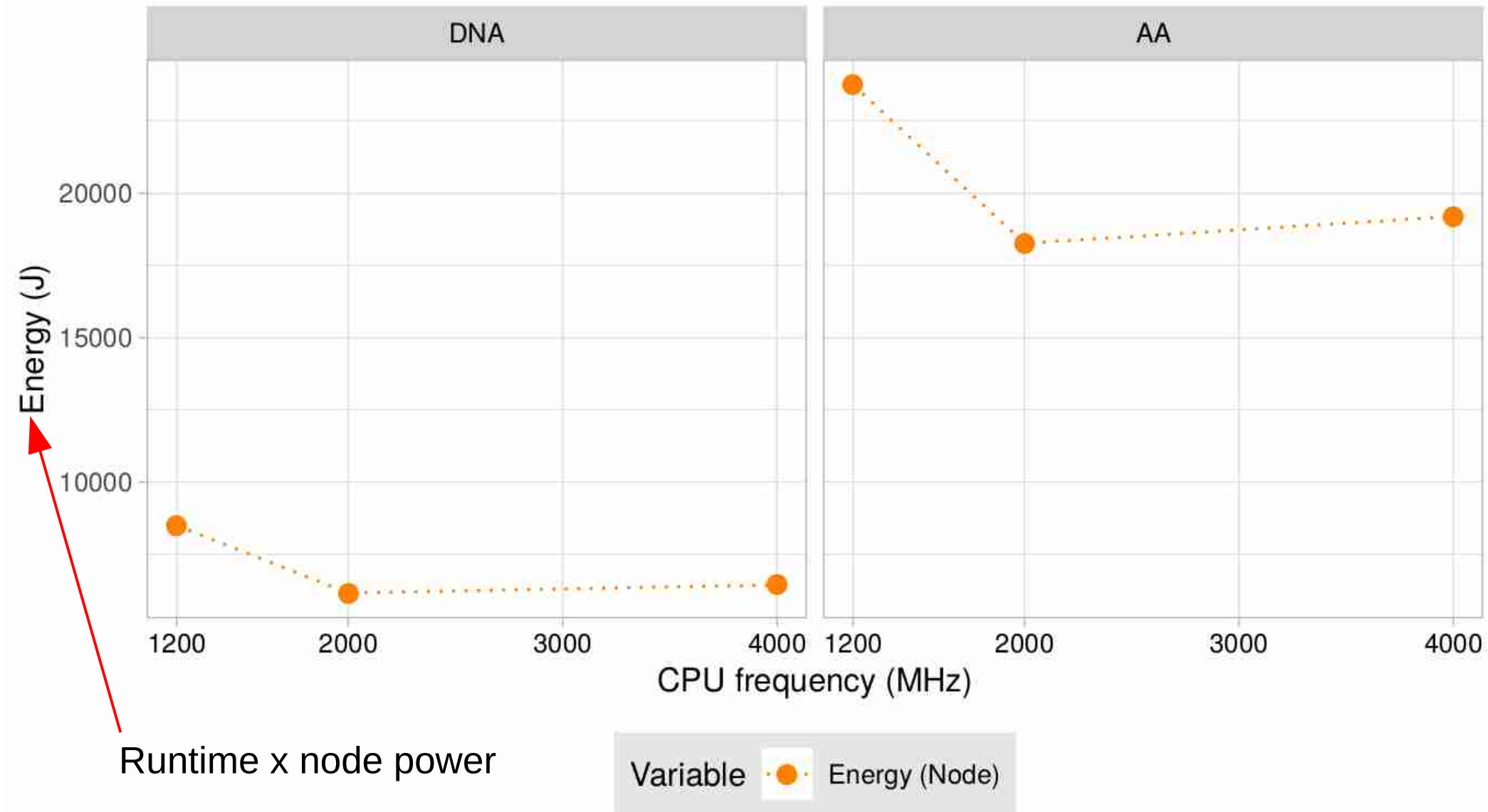
Single tree search (96 nodes x 12h):
>160 kWh



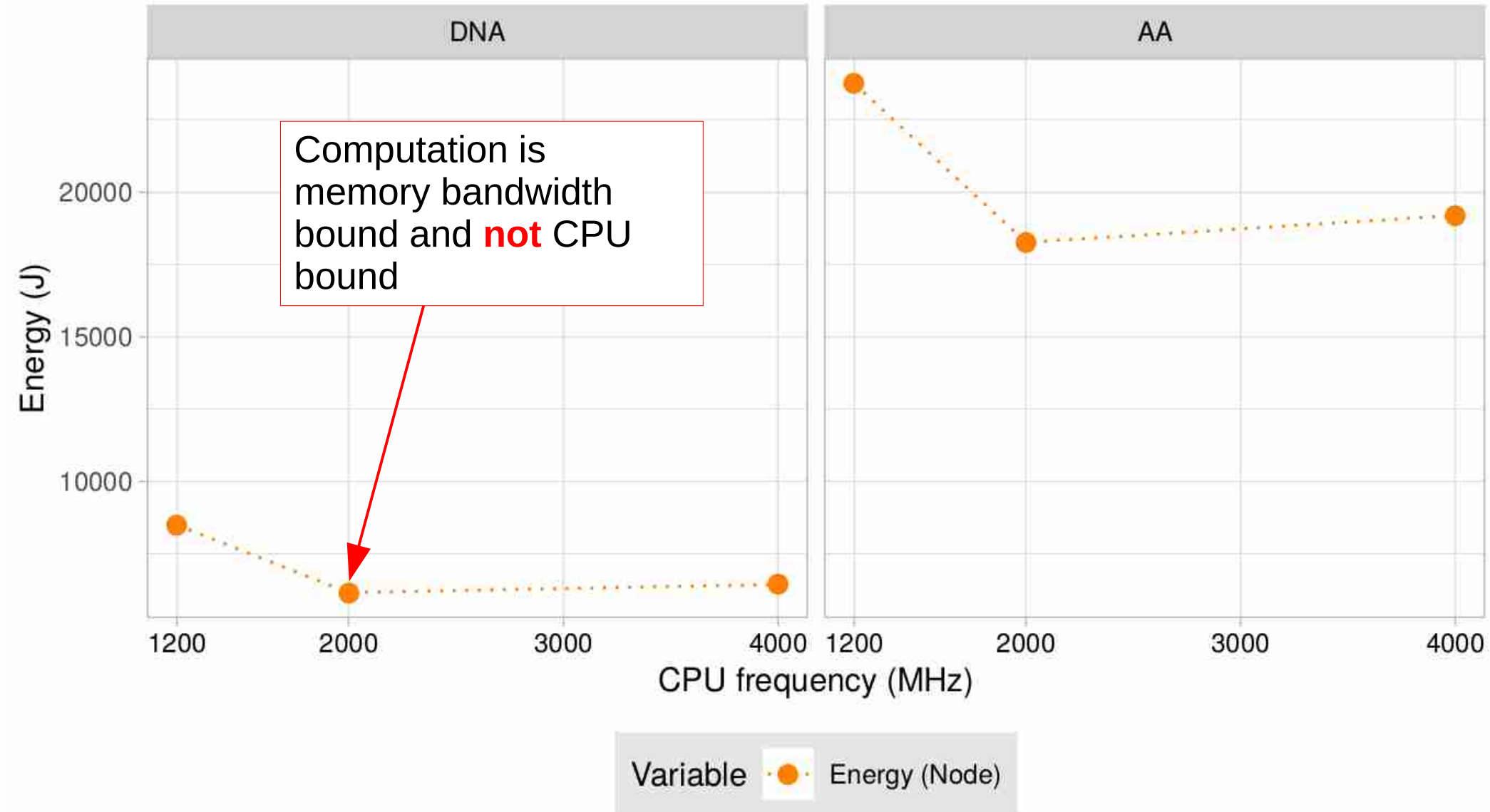
Energy Saving Mode in RAxML-NG v1.0



Phylogenetic Inference: Energy as a function of CPU clock frequency



Phylogenetic Inference: Energy as a function of CPU clock frequency



Back to Programming: Program Optimization

- *“The First Rule of Program Optimization: Don't do it. The Second Rule of Program Optimization (for experts only!): Don't do it yet.”*
- *“Premature optimization is the root of all evil”*

Why do we care about Code Quality?

- In bioinformatics software is potentially used by thousands of people
- It's not just proof of concept as frequently in CS
- Production level codes
 - Bugs & conceptual errors → thousands of papers could be wrong
 - Who maintains the software?
 - Who supports the software?

Code Quality

- A lot of people don't care about code quality
- How can we quantify code quality?
 - difficult
- We can take some simple measures to improve code quality

Compiler Warnings

- gcc → activate all warnings you can think of

- Doesn't catch a lot of issues
- Classic malloc mistake:

```
int a = 2000000, b = 3000000;  
double c = (double *)malloc(a * b * sizeof(double));
```

- Clang compiler

use `-Weverything -Wno-padded`

→ catches the above `malloc()` issue and many more issues

Architecture Analyzers

- Tools to assess SW architecture quality
 - tell you which parts to redesign
- For instance, pmccabe: function complexity and line counting for C and C++
- See <https://people.debian.org/~bame/pmccabe>

Use Assertions!

- Assertions

```
assert(logical op);
```

in C allow to partially implement some simple (and incomplete) form of Hoare-Logic

- Grep a code for assertions and see how many there are
→ good programmers use a lot of assertions!
- Assertions also facilitate debugging
- Users often provide incomplete bug reports, but, when an assertion fails, you directly know the line in the code that failed
- Good example for using assertions?

Switch Statement in C

```
switch(a)
{
    case 1:
        doSomtehing;
        break;
    case 2:
        doSomethingElse;
        break;
    default:
        assert(0);
}
```

The 'crappy' software project

- Analyzed 15 widely-used evolutionary biology tools \approx 65,000 citations
- Analyses performed
 - Compiled with gcc and clang with all warnings enabled
 - Memory check with valgrind
 - Checked if assertions are used via assert ()
 - Analyzed degree of code duplication
- **Caution:** “bad” quality does not induce that a tool is faulty, but the probability of it being faulty is higher!

The State of Software for Evolutionary Biology

Diego Darriba, Tomáš Flouri, Alexandros Stamatakis 

Molecular Biology and Evolution, Volume 35, Issue 5, May 2018, Pages 1037–1046, <https://doi.org/10.1093/molbev/msy014>

Published: 29 January 2018

SoftWipe

- Discussion with Science Journalist - *“Can this process be automated?”*
- Development of SoftWipe - An automated tool and benchmark for **relative** quality ranking of scientific software
- Ranking of *51* open source tools written in C or C++ from a wide range of research areas
 - Astrophysics
 - Computer Science

New Results

[Comment on this paper](#)

SoftWipe - a tool and benchmark to assess scientific software quality

Adrian Zapletal, Dimitri Hoehler, Carsten Sinz, Alexandros Stamatakis

doi: <https://doi.org/10.1101/2020.10.07.330621>

SoftWipe Benchmark

| program name | absolute score | relative score |
|------------------|----------------|----------------|
| genesis | 8.6 | 8.8 |
| hyperphyllo | 8.6 | 8.6 |
| kahypar | 8.4 | 8.5 |
| candy-kingdom | 8.2 | 8.2 |
| bindash-1.0 | 8.0 | 7.9 |
| fastspar | 7.8 | 7.9 |
| repeatscounter | 7.5 | 7.7 |
| axe-0.3.3 | 7.5 | 7.5 |
| virulign-1.0.1 | 7.4 | 7.4 |
| naf-1.1.0/unnaf | 7.4 | 7.5 |
| naf-1.1.0/ennaf | 7.4 | 7.4 |
| ExpansionHunter | 7.3 | 7.5 |
| glucose-3-drup | 7.1 | 7.0 |
| raxml-ng | 7.0 | 7.0 |
| dawg | 6.8 | 6.9 |
| ntEdit-1.2.3 | 6.4 | 6.2 |
| defor | 6.3 | 6.4 |
| swarm | 6.2 | 6.2 |
| lemon | 6.1 | 6.0 |
| treerecs | 6.1 | 6.1 |
| IQ-TREE-2.0-rc1 | 6.1 | 5.7 |
| BGSA_CPU-1.0 | 5.9 | 5.4 |
| emerald | 5.8 | 5.5 |
| dr_sasa_n | 5.7 | 6.0 |
| copmem-0.2 | 5.7 | 5.7 |
| samtools | 5.6 | 5.6 |
| seq-gen | 5.6 | 5.6 |
| dna-nn-0.1 | 5.3 | 5.2 |
| sf | 5.2 | 5.2 |
| cryfa-18.06 | 5.1 | 5.1 |
| ngsLD | 5.1 | 5.0 |
| HLA-1.A | 4.9 | 4.5 |
| iqtree1.6.10 | 4.9 | 4.9 |
| vsearch | 4.6 | 4.6 |
| prank | 4.6 | 4.5 |
| prequal | 4.5 | 4.4 |
| minimap | 4.5 | 4.4 |
| phym1 | 4.4 | 4.4 |
| clustal | 4.2 | 4.3 |
| mrBayes | 4.1 | 4.1 |
| tcoffee | 4.1 | 4.2 |
| gadget | 4.1 | 4.0 |
| crisflash | 4.0 | 4.0 |
| PopLDdecay | 3.8 | 3.8 |
| cellcoal | 3.8 | 3.6 |
| bpp | 3.8 | 3.6 |
| ms | 3.7 | 3.7 |
| mafft | 3.3 | 3.1 |
| athena | 2.9 | 2.8 |
| covid-sim-0.13.0 | 2.5 | 2.4 |
| indelible | 1.4 | 1.0 |

SoftWipe Benchmark

| program name | absolute score | relative score |
|------------------|----------------|----------------|
| genesis | 8.6 | 8.8 |
| hyperphyllo | 8.6 | 8.6 |
| kahypar | 8.4 | 8.5 |
| candy-kingdom | 8.2 | 8.2 |
| bindash-1.0 | 8.0 | 7.9 |
| fastspar | 7.8 | 7.9 |
| repeatscounter | 7.5 | 7.7 |
| axe-0.3.3 | 7.5 | 7.5 |
| virulign-1.0.1 | 7.4 | 7.4 |
| naf-1.1.0/unnaf | 7.4 | 7.5 |
| naf-1.1.0/ennaf | 7.4 | 7.4 |
| ExpansionHunter | 7.3 | 7.5 |
| glucose-3-drup | 7.1 | 7.0 |
| raxml-ng | 7.0 | 7.0 |
| dawg | 6.8 | 6.9 |
| ntEdit-1.2.3 | 6.4 | 6.2 |
| defor | 6.3 | 6.4 |
| swarm | 6.2 | 6.2 |
| lemon | 6.1 | 6.0 |
| treerecs | 6.1 | 6.1 |
| IQ-TREE-2.0-rc1 | 6.1 | 5.7 |
| BGSA_CPU-1.0 | 5.9 | 5.4 |
| emerald | 5.8 | 5.5 |
| dr_sasa_n | 5.7 | 6.0 |
| copmem-0.2 | 5.7 | 5.7 |
| samtools | 5.6 | 5.6 |
| seq-gen | 5.6 | 5.6 |
| dna-nn-0.1 | 5.3 | 5.2 |
| sf | 5.2 | 5.2 |
| cryfa-18.06 | 5.1 | 5.1 |
| ngsLD | 5.1 | 5.0 |
| HLA-1.A | 4.9 | 4.5 |
| iqtree1.6.10 | 4.9 | 4.9 |
| vsearch | 4.6 | 4.6 |
| prank | 4.6 | 4.5 |
| prequal | 4.5 | 4.4 |
| minimap | 4.5 | 4.4 |
| phym1 | 4.4 | 4.4 |
| clustal | 4.2 | 4.3 |
| mrBayes | 4.1 | 4.1 |
| tcoffee | 4.1 | 4.2 |
| gadget | 4.1 | 4.0 |
| crisflash | 4.0 | 4.0 |
| PopLDdecay | 3.8 | 3.8 |
| cellcoal | 3.8 | 3.6 |
| bpp | 3.8 | 3.6 |
| ms | 3.7 | 3.7 |
| mafft | 3.3 | 3.1 |
| athena | 2.9 | 2.8 |
| covid-sim-0.13.0 | 2.5 | 2.4 |
| indelible | 1.4 | 1.0 |

Does not change over time as more tools are added →
can easily be referenced

SoftWipe Benchmark

| program name | absolute score | relative score |
|------------------|----------------|----------------|
| genesis | 8.6 | 8.8 |
| hyperphyllo | 8.6 | 8.6 |
| kahypar | 8.4 | 8.5 |
| candy-kingdom | 8.2 | 8.2 |
| bindash-1.0 | 8.0 | 7.9 |
| fastspar | 7.8 | 7.9 |
| repeatscounter | 7.5 | 7.7 |
| axe-0.3.3 | 7.5 | 7.5 |
| virulign-1.0.1 | 7.4 | 7.4 |
| naf-1.1.0/unnaf | 7.4 | 7.5 |
| naf-1.1.0/ennaf | 7.4 | 7.4 |
| ExpansionHunter | 7.3 | 7.5 |
| glucose-3-drup | 7.1 | 7.0 |
| raxml-ng | 7.0 | 7.0 |
| dawg | 6.8 | 6.9 |
| ntEdit-1.2.3 | 6.4 | 6.2 |
| defor | 6.3 | 6.4 |
| swarm | 6.2 | 6.2 |
| lemon | 6.1 | 6.0 |
| treerecs | 6.1 | 6.1 |
| IQ-TREE-2.0-rc1 | 6.1 | 5.7 |
| BGSA_CPU-1.0 | 5.9 | 5.4 |
| emerald | 5.8 | 5.5 |
| dr_sasa_n | 5.7 | 6.0 |
| copmem-0.2 | 5.7 | 5.7 |
| samtools | 5.6 | 5.6 |
| seq-gen | 5.6 | 5.6 |
| dna-nn-0.1 | 5.3 | 5.2 |
| sf | 5.2 | 5.2 |
| cryfa-18.06 | 5.1 | 5.1 |
| ngsLD | 5.1 | 5.0 |
| HLA-1.A | 4.9 | 4.5 |
| iqtree1.6.10 | 4.9 | 4.9 |
| vsearch | 4.6 | 4.6 |
| prank | 4.6 | 4.5 |
| prequal | 4.5 | 4.4 |
| minimap | 4.5 | 4.4 |
| phym1 | 4.4 | 4.4 |
| clustal | 4.2 | 4.3 |
| mrBayes | 4.1 | 4.1 |
| tcoffee | 4.1 | 4.2 |
| gadget | 4.1 | 4.0 |
| crisflash | 4.0 | 4.0 |
| PopLDdecay | 3.8 | 3.8 |
| celloal | 3.8 | 3.6 |
| bpp | 3.8 | 3.6 |
| ms | 3.7 | 3.7 |
| mafft | 3.3 | 3.1 |
| athena | 2.9 | 2.8 |
| covid-sim-0.13.0 | 2.5 | 2.4 |
| indelible | 1.4 | 1.0 |

Does change over time as more tools are added →
Difficult to be referenced

| program name | absolute score | relative score |
|------------------|----------------|----------------|
| genesis | 8.6 | 8.8 |
| hyperphyllo | 8.6 | 8.6 |
| kahypar | 8.4 | 8.5 |
| candy-kingdom | 8.2 | 8.2 |
| bindash-1.0 | 8.0 | 7.9 |
| fastspar | 7.8 | 7.9 |
| repeatscounter | 7.5 | 7.7 |
| axe-0.3.3 | 7.5 | 7.5 |
| virulign-1.0.1 | 7.4 | 7.4 |
| naf-1.1.0/unnaf | 7.4 | 7.5 |
| naf-1.1.0/ennaf | 7.4 | 7.4 |
| ExpansionHunter | 7.3 | 7.5 |
| glucose-3-drup | 7.1 | 7.0 |
| raxml-ng | 7.0 | 7.0 |
| dawg | 6.8 | 6.9 |
| ntEdit-1.2.3 | 6.4 | 6.2 |
| defor | 6.3 | 6.4 |
| swarm | 6.2 | 6.2 |
| lemon | 6.1 | 6.0 |
| treerecs | 6.1 | 6.1 |
| IQ-TREE-2.0-rc1 | 6.1 | 5.7 |
| BGSA_CPU-1.0 | 5.9 | 5.4 |
| emerald | 5.8 | 5.5 |
| dr_sasa_n | 5.7 | 6.0 |
| copmem-0.2 | 5.7 | 5.7 |
| samtools | 5.6 | 5.6 |
| seq-gen | 5.6 | 5.6 |
| dna-nn-0.1 | 5.3 | 5.2 |
| sf | 5.2 | 5.2 |
| cryfa-18.06 | 5.1 | 5.1 |
| ngsLD | 5.1 | 5.0 |
| HLA-1.A | 4.9 | 4.5 |
| iqtree l.6.10 | 4.9 | 4.9 |
| vsearch | 4.6 | 4.6 |
| prank | 4.6 | 4.5 |
| prequal | 4.5 | 4.4 |
| minimap | 4.5 | 4.4 |
| phym1 | 4.4 | 4.4 |
| clustal | 4.2 | 4.3 |
| mrBayes | 4.1 | 4.1 |
| tcoffee | 4.1 | 4.2 |
| gadget | 4.1 | 4.0 |
| crisflash | 4.0 | 4.0 |
| PopLDdecay | 3.8 | 3.8 |
| cellcoal | 3.8 | 3.6 |
| bpp | 3.8 | 3.6 |
| ms | 3.7 | 3.7 |
| mafft | 3.3 | 3.1 |
| athena | 2.9 | 2.8 |
| covid-sim-0.13.0 | 2.5 | 2.4 |
| indelible | 1.4 | 1.0 |

Written by computer scientists

SoftWipe Benchmark

SoftWipe Benchmark

My lab

| program name | absolute score | relative score |
|------------------|----------------|----------------|
| genesis | 8.6 | 8.8 |
| hyperphylo | 8.6 | 8.6 |
| kahypar | 8.4 | 8.5 |
| candy-kingdom | 8.2 | 8.2 |
| bindash-1.0 | 8.0 | 7.9 |
| fastspar | 7.8 | 7.9 |
| repeatscounter | 7.5 | 7.7 |
| axe-0.3.3 | 7.5 | 7.5 |
| virulign-1.0.1 | 7.4 | 7.4 |
| naf-1.1.0/unnaf | 7.4 | 7.5 |
| naf-1.1.0/ennaf | 7.4 | 7.4 |
| ExpansionHunter | 7.3 | 7.5 |
| glucose-3-drup | 7.1 | 7.0 |
| raxml-ng | 7.0 | 7.0 |
| dawg | 6.8 | 6.9 |
| ntEdit-1.2.3 | 6.4 | 6.2 |
| defor | 6.3 | 6.4 |
| swarm | 6.2 | 6.2 |
| lemon | 6.1 | 6.0 |
| treerecs | 6.1 | 6.1 |
| IQ-TREE-2.0-rc1 | 6.1 | 5.7 |
| BGSA_CPU-1.0 | 5.9 | 5.4 |
| emerald | 5.8 | 5.5 |
| dr_sasa_n | 5.7 | 6.0 |
| copmem-0.2 | 5.7 | 5.7 |
| samtools | 5.6 | 5.6 |
| seq-gen | 5.6 | 5.6 |
| dna-nn-0.1 | 5.3 | 5.2 |
| sf | 5.2 | 5.2 |
| cryfa-18.06 | 5.1 | 5.1 |
| ngsLD | 5.1 | 5.0 |
| HLA-1.A | 4.9 | 4.5 |
| iqtree1.6.10 | 4.9 | 4.9 |
| vsearch | 4.6 | 4.6 |
| prank | 4.6 | 4.5 |
| prequal | 4.5 | 4.4 |
| minimap | 4.5 | 4.4 |
| phym1 | 4.4 | 4.4 |
| clustal | 4.2 | 4.3 |
| mrBayes | 4.1 | 4.1 |
| tcoffee | 4.1 | 4.2 |
| gadget | 4.1 | 4.0 |
| crisflash | 4.0 | 4.0 |
| PopLDdecay | 3.8 | 3.8 |
| cellcoal | 3.8 | 3.6 |
| bpp | 3.8 | 3.6 |
| ms | 3.7 | 3.7 |
| mafft | 3.3 | 3.1 |
| athena | 2.9 | 2.8 |
| covid-sim-0.13.0 | 2.5 | 2.4 |
| indelible | 1.4 | 1.0 |

SoftWipe Benchmark

| program name | absolute score | relative score |
|------------------|----------------|----------------|
| genesis | 8.6 | 8.8 |
| hyperphyllo | 8.6 | 8.6 |
| kahypar | 8.4 | 8.5 |
| candy-kingdom | 8.2 | 8.2 |
| bindash-1.0 | 8.0 | 7.9 |
| fastspar | 7.8 | 7.9 |
| repeatscounter | 7.5 | 7.7 |
| axe-0.3.3 | 7.5 | 7.5 |
| virulign-1.0.1 | 7.4 | 7.4 |
| naf-1.1.0/unnaf | 7.4 | 7.5 |
| naf-1.1.0/ennaf | 7.4 | 7.4 |
| ExpansionHunter | 7.3 | 7.5 |
| glucose-3-drup | 7.1 | 7.0 |
| raxml-ng | 7.0 | 7.0 |
| dawg | 6.8 | 6.9 |
| ntEdit-1.2.3 | 6.4 | 6.2 |
| defor | 6.3 | 6.4 |
| swarm | 6.2 | 6.2 |
| lemon | 6.1 | 6.0 |
| treerecs | 6.1 | 6.1 |
| IQ-TREE-2.0-rc1 | 6.1 | 5.7 |
| BGSA_CPU-1.0 | 5.9 | 5.4 |
| emerald | 5.8 | 5.5 |
| dr_sasa_n | 5.7 | 6.0 |
| copmem-0.2 | 5.7 | 5.7 |
| samtools | 5.6 | 5.6 |
| seq-gen | 5.6 | 5.6 |
| dna-nn-0.1 | 5.3 | 5.2 |
| sf | 5.2 | 5.2 |
| cryfa-18.06 | 5.1 | 5.1 |
| ngsLD | 5.1 | 5.0 |
| HLA-1.A | 4.9 | 4.5 |
| iqtree1.6.10 | 4.9 | 4.9 |
| vsearch | 4.6 | 4.6 |
| prank | 4.6 | 4.5 |
| prequal | 4.5 | 4.4 |
| minimap | 4.5 | 4.4 |
| phyml | 4.4 | 4.4 |
| clustal | 4.2 | 4.3 |
| mrBayes | 4.1 | 4.1 |
| tcoffee | 4.1 | 4.2 |
| gadget | 4.1 | 4.0 |
| crisflash | 4.0 | 4.0 |
| PopLDdecay | 3.8 | 3.8 |
| celloal | 3.8 | 3.6 |
| bpp | 3.8 | 3.6 |
| ms | 3.7 | 3.7 |
| mafft | 3.3 | 3.1 |
| athena | 2.9 | 2.8 |
| covid-sim-0.13.0 | 2.5 | 2.4 |
| indelible | 1.4 | 1.0 |

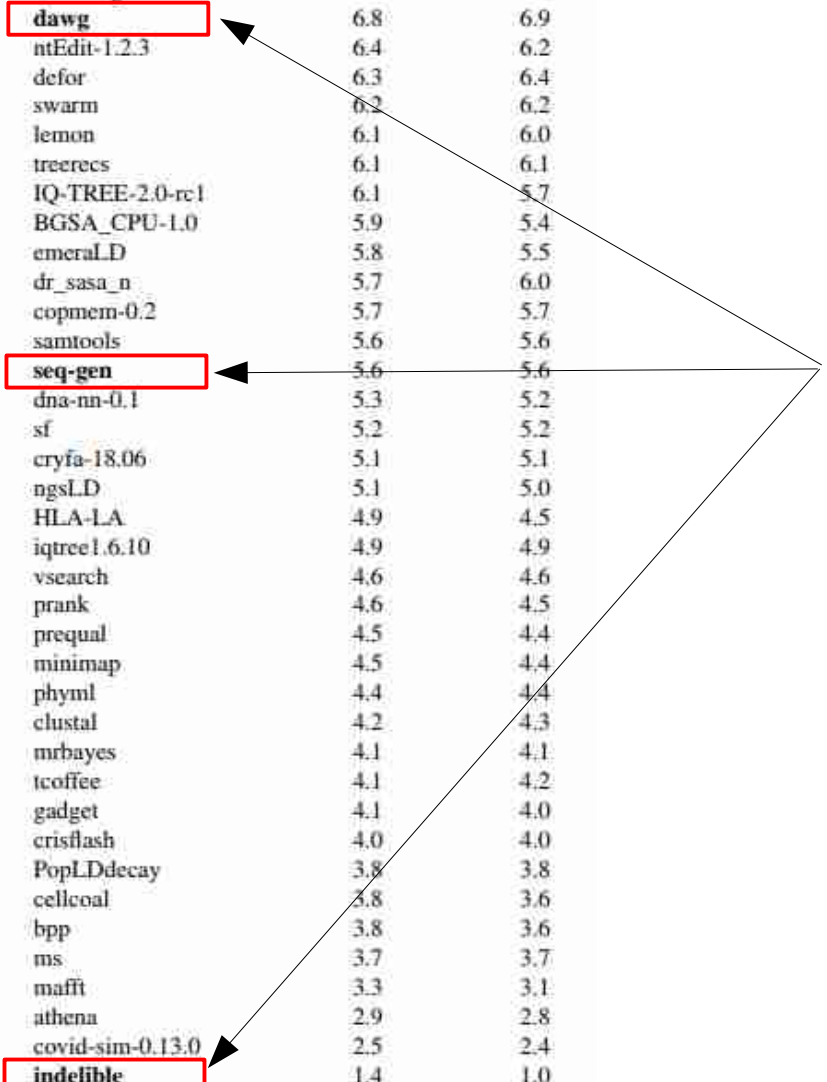
Astrophysics



SoftWipe Benchmark

| program name | absolute score | relative score |
|------------------|----------------|----------------|
| genesis | 8.6 | 8.8 |
| hyperphyllo | 8.6 | 8.6 |
| kahypar | 8.4 | 8.5 |
| candy-kingdom | 8.2 | 8.2 |
| bindash-1.0 | 8.0 | 7.9 |
| fastspar | 7.8 | 7.9 |
| repeatscounter | 7.5 | 7.7 |
| axe-0.3.3 | 7.5 | 7.5 |
| virulign-1.0.1 | 7.4 | 7.4 |
| naf-1.1.0/unnaf | 7.4 | 7.5 |
| naf-1.1.0/ennaf | 7.4 | 7.4 |
| ExpansionHunter | 7.3 | 7.5 |
| glucose-3-drup | 7.1 | 7.0 |
| raxml-ng | 7.0 | 7.0 |
| dawg | 6.8 | 6.9 |
| ntEdit-1.2.3 | 6.4 | 6.2 |
| defor | 6.3 | 6.4 |
| swarm | 6.2 | 6.2 |
| lemon | 6.1 | 6.0 |
| treerecs | 6.1 | 6.1 |
| IQ-TREE-2.0-rc1 | 6.1 | 5.7 |
| BGSA_CPU-1.0 | 5.9 | 5.4 |
| emerald | 5.8 | 5.5 |
| dr_sasa_n | 5.7 | 6.0 |
| copmem-0.2 | 5.7 | 5.7 |
| samttools | 5.6 | 5.6 |
| seq-gen | 5.6 | 5.6 |
| dna-nn-0.1 | 5.3 | 5.2 |
| sf | 5.2 | 5.2 |
| cryfa-18.06 | 5.1 | 5.1 |
| ngsLD | 5.1 | 5.0 |
| HLA-1.A | 4.9 | 4.5 |
| iqtree1.6.10 | 4.9 | 4.9 |
| vsearch | 4.6 | 4.6 |
| prank | 4.6 | 4.5 |
| prequal | 4.5 | 4.4 |
| minimap | 4.5 | 4.4 |
| phym1 | 4.4 | 4.4 |
| clustal | 4.2 | 4.3 |
| mrBayes | 4.1 | 4.1 |
| tcoffee | 4.1 | 4.2 |
| gadget | 4.1 | 4.0 |
| crisflash | 4.0 | 4.0 |
| PopLDdecay | 3.8 | 3.8 |
| celloal | 3.8 | 3.6 |
| bpp | 3.8 | 3.6 |
| ms | 3.7 | 3.7 |
| mafft | 3.3 | 3.1 |
| athena | 2.9 | 2.8 |
| covid-sim-0.13.0 | 2.5 | 2.4 |
| indelible | 1.4 | 1.0 |

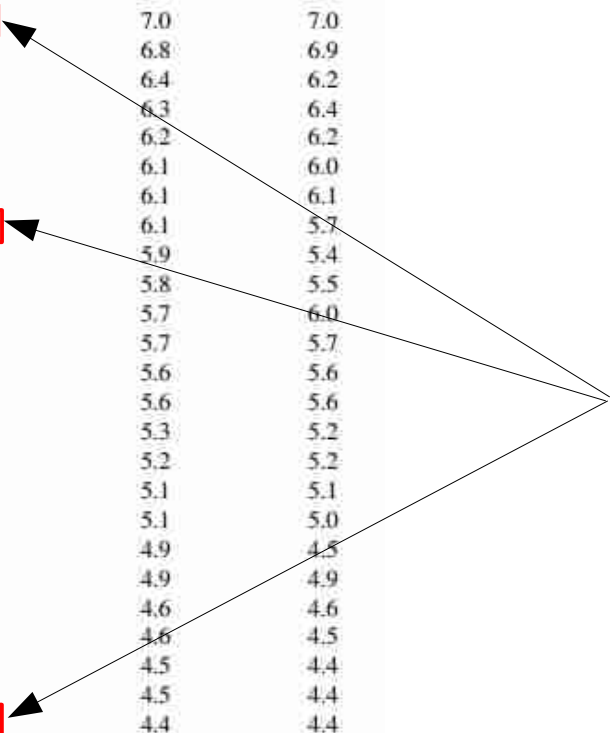
Tools with highly similar functionality



SoftWipe Benchmark

| program name | absolute score | relative score |
|------------------|----------------|----------------|
| genesis | 8.6 | 8.8 |
| hyperphyllo | 8.6 | 8.6 |
| kahypar | 8.4 | 8.5 |
| candy-kingdom | 8.2 | 8.2 |
| bindash-1.0 | 8.0 | 7.9 |
| fastspar | 7.8 | 7.9 |
| repeatscounter | 7.5 | 7.7 |
| axe-0.3.3 | 7.5 | 7.5 |
| virulign-1.0.1 | 7.4 | 7.4 |
| naf-1.1.0/unnaf | 7.4 | 7.5 |
| naf-1.1.0/ennaf | 7.4 | 7.4 |
| ExpansionHunter | 7.3 | 7.5 |
| glucose-3-drup | 7.1 | 7.0 |
| raxml-ng | 7.0 | 7.0 |
| dawg | 6.8 | 6.9 |
| ntEdit-1.2.3 | 6.4 | 6.2 |
| defor | 6.3 | 6.4 |
| swarm | 6.2 | 6.2 |
| lemon | 6.1 | 6.0 |
| treerecs | 6.1 | 6.1 |
| IQ-TREE-2.0-rc1 | 6.1 | 5.7 |
| BGSA_CPU-1.0 | 5.9 | 5.4 |
| emerald | 5.8 | 5.5 |
| dr_sasa_n | 5.7 | 6.0 |
| copmem-0.2 | 5.7 | 5.7 |
| samtools | 5.6 | 5.6 |
| seq-gen | 5.6 | 5.6 |
| dna-nn-0.1 | 5.3 | 5.2 |
| sf | 5.2 | 5.2 |
| cryfa-18.06 | 5.1 | 5.1 |
| ngsLD | 5.1 | 5.0 |
| HLA-1.A | 4.9 | 4.5 |
| iqtree1.6.10 | 4.9 | 4.9 |
| vsearch | 4.6 | 4.6 |
| prank | 4.6 | 4.5 |
| prequal | 4.5 | 4.4 |
| minimap | 4.5 | 4.4 |
| phym1 | 4.4 | 4.4 |
| clustal | 4.2 | 4.3 |
| mrBayes | 4.1 | 4.1 |
| tcoffee | 4.1 | 4.2 |
| gadget | 4.1 | 4.0 |
| crisflash | 4.0 | 4.0 |
| PopLDdecay | 3.8 | 3.8 |
| cellcoal | 3.8 | 3.6 |
| bpp | 3.8 | 3.6 |
| ms | 3.7 | 3.7 |
| mafft | 3.3 | 3.1 |
| athena | 2.9 | 2.8 |
| covid-sim-0.13.0 | 2.5 | 2.4 |
| indelible | 1.4 | 1.0 |

Tools with highly similar functionality



SoftWipe Benchmark

| program name | absolute score | relative score |
|-------------------------|----------------|----------------|
| genesis | 8.6 | 8.8 |
| hyperphylo | 8.6 | 8.6 |
| kahypar | 8.4 | 8.5 |
| candy-kingdom | 8.2 | 8.2 |
| bindash-1.0 | 8.0 | 7.9 |
| fastspar | 7.8 | 7.9 |
| repeatscounter | 7.5 | 7.7 |
| axe-0.3.3 | 7.5 | 7.5 |
| virulign-1.0.1 | 7.4 | 7.4 |
| naf-1.1.0/unnaf | 7.4 | 7.5 |
| naf-1.1.0/ennaf | 7.4 | 7.4 |
| ExpansionHunter | 7.3 | 7.5 |
| glucose-3-drup | 7.1 | 7.0 |
| raxml-ng | 7.0 | 7.0 |
| dawg | 6.8 | 6.9 |
| ntEdit-1.2.3 | 6.4 | 6.2 |
| defor | 6.3 | 6.4 |
| swarm | 6.2 | 6.2 |
| lemon | 6.1 | 6.0 |
| treerecs | 6.1 | 6.1 |
| IQ-TREE-2.0-rc1 | 6.1 | 5.7 |
| BGSA_CPU-1.0 | 5.9 | 5.4 |
| emerald | 5.8 | 5.5 |
| dr_sasa_n | 5.7 | 6.0 |
| copmem-0.2 | 5.7 | 5.7 |
| samttools | 5.6 | 5.6 |
| seq-gen | 5.6 | 5.6 |
| dna-nn-0.1 | 5.3 | 5.2 |
| sf | 5.2 | 5.2 |
| cryfa-18.06 | 5.1 | 5.1 |
| ngsLD | 5.1 | 5.0 |
| HLA-1.A | 4.9 | 4.5 |
| iqtree1.6.10 | 4.9 | 4.9 |
| vsearch | 4.6 | 4.6 |
| prank | 4.6 | 4.5 |
| prequal | 4.5 | 4.4 |
| minimap | 4.5 | 4.4 |
| phym1 | 4.4 | 4.4 |
| clustal | 4.2 | 4.3 |
| mrBayes | 4.1 | 4.1 |
| tcoffee | 4.1 | 4.2 |
| gadget | 4.1 | 4.0 |
| crisflash | 4.0 | 4.0 |
| PopLDdecay | 3.8 | 3.8 |
| celloal | 3.8 | 3.6 |
| bpp | 3.8 | 3.6 |
| ms | 3.7 | 3.7 |
| mafft | 3.3 | 3.1 |
| athena | 2.9 | 2.8 |
| covid-sim-0.13.0 | 2.5 | 2.4 |
| indelible | 1.4 | 1.0 |

NEWS WEBSITE OF THE YEAR

The Telegraph Coronavirus News Politics Sport Business Money Opinion Tech Life Style Travel Culture

Gadgets ▾ Innovation ▾ Big tech ▾ Start-ups ▾ Politics of tech ▾ Gaming ▾

Coding that led to lockdown was 'totally unreliable' and a 'buggy mess', say experts

The code, written by Professor Neil Ferguson and his team at Imperial College London, was impossible to read, scientists claim

Covid simulation tool



SoftWipe in Practice

- Leads to healthy competition among lab members → everyone wants to write the cleanest code
- Used by researchers inside and outside of the lab during the development process → potential bugs identified and avoided
- Used as teaching tool in programming practicals
- SoftWipe score already used by us and others in Bioinformatics software paper submissions
- **Vision:** Establish software quality indicators as a necessary prerequisite for software paper submissions

Software Quality and Maintainability

- The Next Generation (**-NG**) projects:
 - Re-design, re-factoring, from scratch re-implementation of flagship tools to ensure maintainability, sustainability, and extensibility & increase scalability/performance
 - ModelTest-NG – model testing of evolutionary models for phylogenetic inference
 - RAxML-NG – phylogenetic inference
 - EPA-NG – phylogenetic placement of environmental reads

Profiling

- Profiling: collecting statistics from example executions
 - Provides an idea which routines/functions are critical
 - Common profiling approaches:
 - Instrument manually/automatically all procedure call/return points (potentially expensive)
 - Sampling PC every X milliseconds -- so long as program run is significantly longer than the sampling period, the accuracy of profiling is pretty good
 - Profiling output

| <u>Routine</u> | <u>% of Execution Time</u> |
|----------------|----------------------------|
| function_a | 60% |
| function_b | 27% |
| function_c | 4% |
| ... | |
| function_zzz | 0.01% |

- Often over 80% of the time spent in less than 20% of the code (80/20 rule)
 - Code locality principle
 - Data locality principle
- More accurate profiling is possible with on-chip built-in HW counters and analysis tools

Using Gprof

- GNU profiler
- Uses PC sampling technique
- Add the **-pg** flag to the compiling step!
- Add the **-pg** flag to the linking step!
- Then just run the program as before
 - It will execute slower though!
- Slowdown for dense matrix-matrix multiplication (**mmult**):
 - Less than 1%
- Then type: `gprof mmult`

Gprof on mmult

Flat profile:

Each sample counts as 0.01 seconds.

| % time | cumulative seconds | self seconds | self calls | total ns/call | ns/call | name |
|-----------|-----------------------|-----------------|---------------|------------------|---------|---------|
| 99.83 | 156.90 | 156.90 | | | | main |
| 0.17 | 157.17 | 0.27 | 11184640 | 24.14 | 24.14 | randum |
| 0.00 | 157.17 | 0.00 | 18 | 0.00 | 0.00 | gettime |

%
time the percentage of the total running time of the
 program used by this function.

.....

Using Valgrind

- Open source tool to find C/C++ memory leaks and a lot of other stuff!
- When compiling add **-g** to the compiler flags such that you can see the source lines where valgrind detects errors

- To look for memory leaks type:

```
valgrind --tool=memcheck --leak-check=yes -v
```

... and then just the executable as you called it so far

- Also works with PThreads-based programs
- Valgrind **slowdown** for **mmult almost factor 10 (188 -> 1451 secs)!**

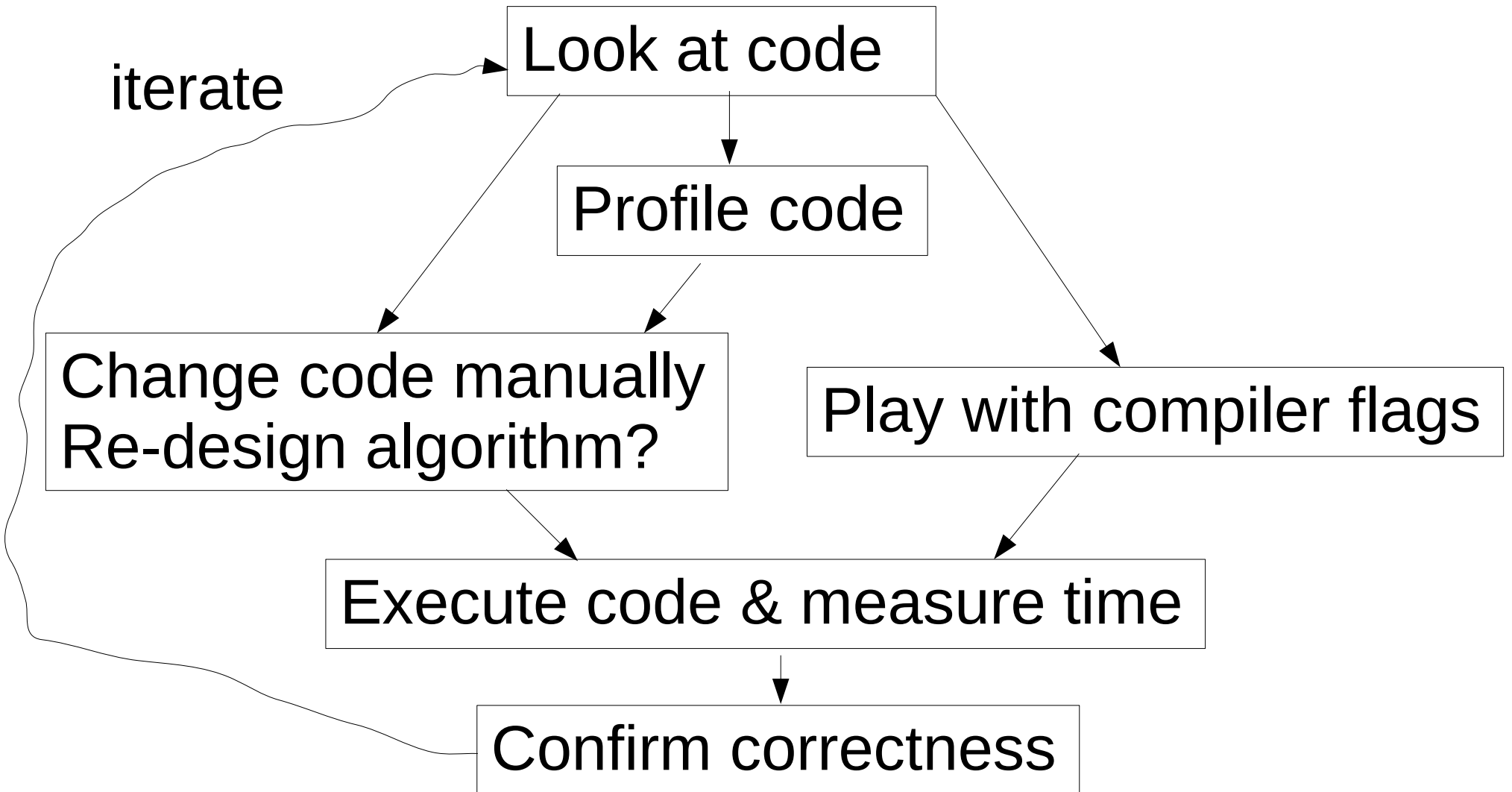
Valgrind on mmult

```
Total time 1451.554931
--13304-- REDIR: 0x40cd4b0 (memset) redirected to 0x4023d50 (memset)
==13304==
==13304== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 13 from 1)
--13304--
--13304-- supp: 13 dl-hack3-1
==13304== malloc/free: in use at exit: 0 bytes in 0 blocks.
==13304== malloc/free: 12,264 allocs, 12,264 frees, 134,264,640 bytes allocated.
```

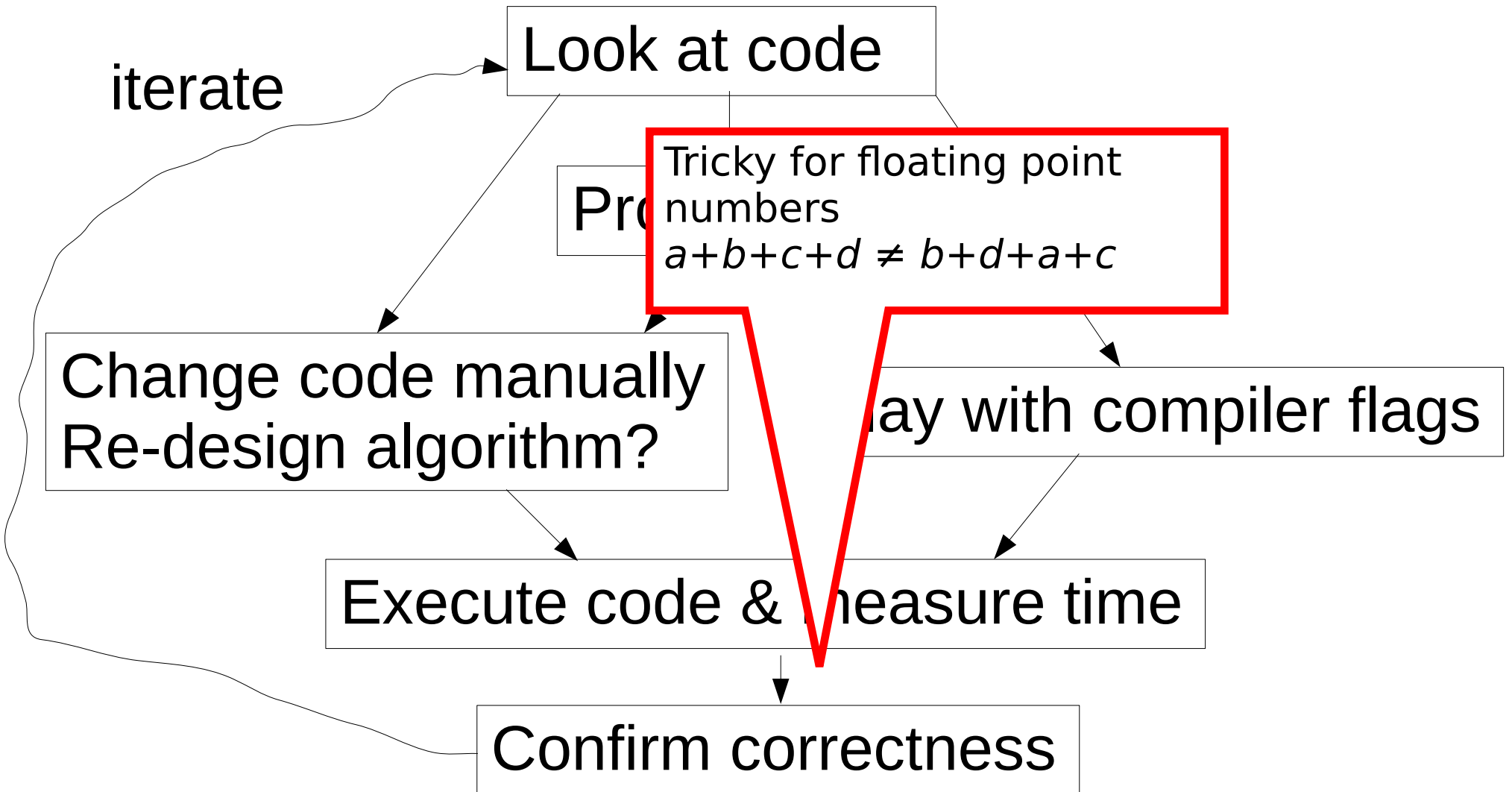
Deliberate bug: `c[matrixSize][matrixSize-1] = 0.0;`

```
==14585== Invalid read of size 4
==14585==   at 0x80488AC: main (mmult.c:124)
==14585== Address 0x41aa148 is 0 bytes after a block of size 64 alloc'd
==14585==   at 0x4022AB8: malloc (vg_replace_malloc.c:207)
==14585==   by 0x8048625: main (mmult.c:73)
==14585==
==14585== Invalid write of size 8
==14585==   at 0x80488BC: main (mmult.c:124)
==14585== Address 0x78 is not stack'd, malloc'd or (recently) free'd
```

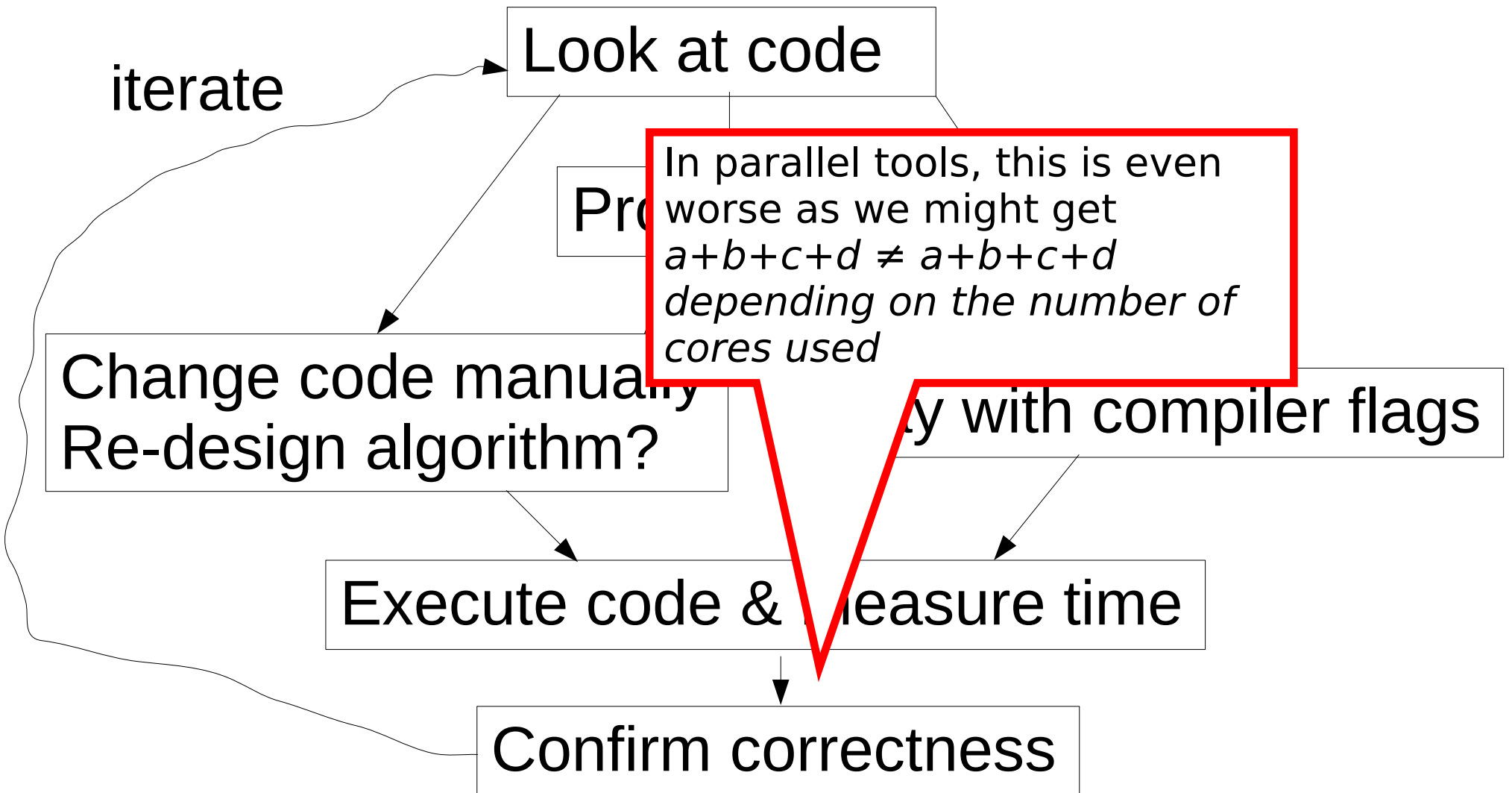
Optimization Cycle



Optimization Cycle



Optimization Cycle



BLAS: Basic Linear Algebra Subprograms

- Interface definition for linear algebra routines
- Several highly optimized implementations exist!
- Code optimization by using optimized libraries!
- Level 1 routines
 - Vector-vector ops
- Level 2 routines
 - Vector-matrix ops
- Level 3 routines
 - Matrix-matrix ops
- C-interface:
 - Functions preceded by **cb`las_`**, e.g., **cb`las_dgemm()`**
 - Arrays need to be contiguous in memory
 - Passed as pointers, **not** pointers of pointers

BLAS

- Implementations
 - GOTO-BLAS → usually the fastest one
 - ATLAS-BLAS → easier to install
 - Intel MKL Math Kernel Library
 - also offers fast implementations of $\exp()$, $\log()$ and other functions
 - AMD AMCL
- Interfaces: matrix linearization:
 - by rows or
 - by columns?
- ATLAS has a C interface

De-Normalized Floating Point Numbers

- On-line demo of my benchmark
- Micro-Benchmark available at:
<https://github.com/stamatak/denormalizedFloatingPointNumbers>

De-Normalized Floating Point Numbers

- Internal representation > 64 bit, that deviates from IEEE 754 standard
- Allows for de-normalized floating point numbers to represent values near 0 → directly done by the hardware
- CPU issues floating point exception when using de-normalized floats
→ this exception delays CPU by several cycles!
- J. Björndalen and O. Anshus: "Trusting floating point benchmarks-are your benchmarks really data-independent?"
http://www.hpc2n.umu.se/sites/default/files/events/para06/papers/paper_195.pdf

Floating Point Numbers are the Root of all Evil!

- Read the classic paper: *What Every Computer Scientist Should Know About Floating Point Arithmetic*
- Link:
<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.6768>
- What can happen when we do parallel numerics?

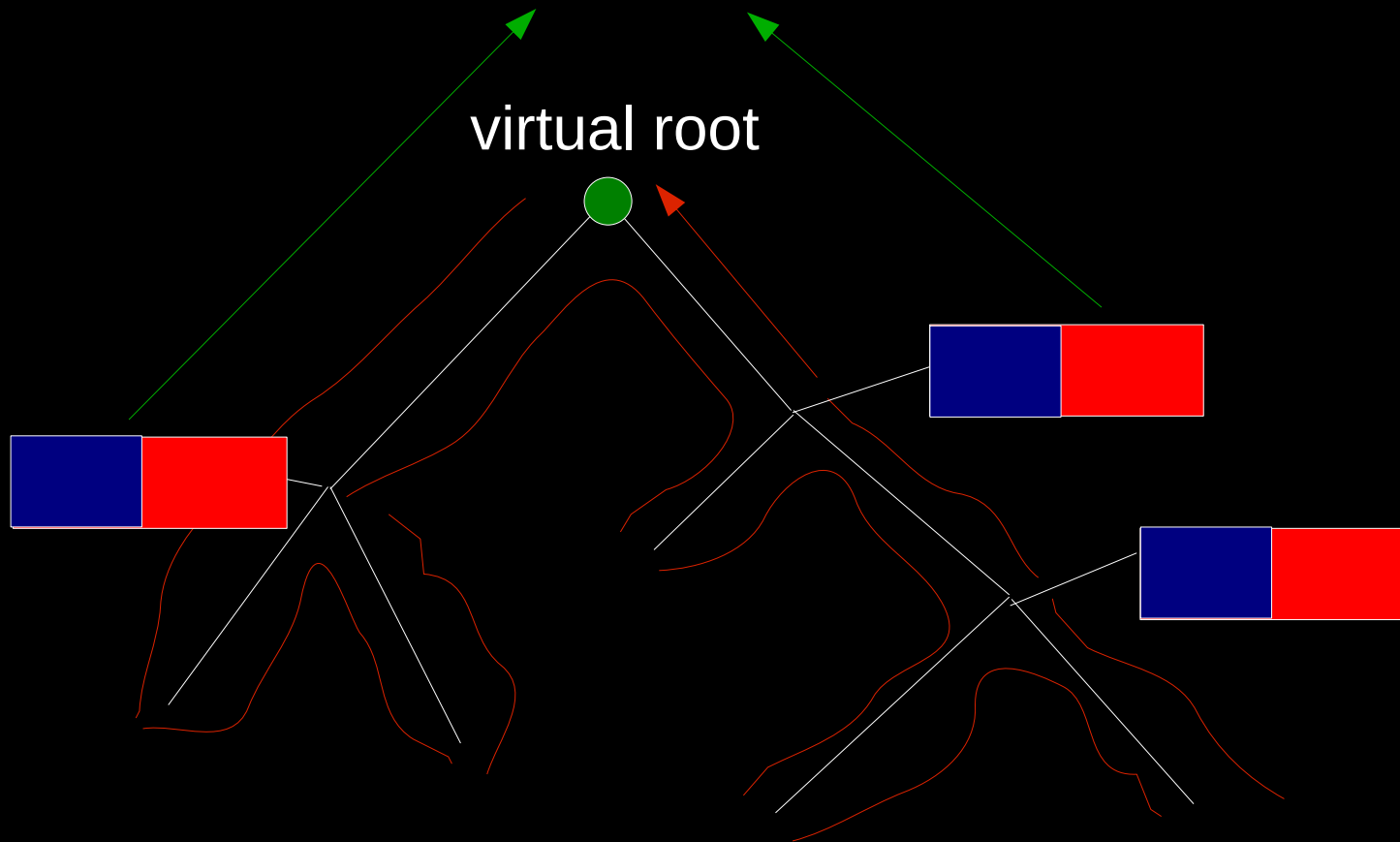
Floating Point Numbers are the Root of all Evil!

- Read the classic paper: *What Every Computer Scientist Should Know About Floating Point Arithmetic*
- Link:
<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.6768>
- What can happen when we do parallel numerics?
- For instance if we do likelihood calculations and run the same code twice with 2 and 4 cores?
- What could happen?

An Example

Parallel Numerics

$$\sum \log(l_j) + \sum \log(l_j)$$



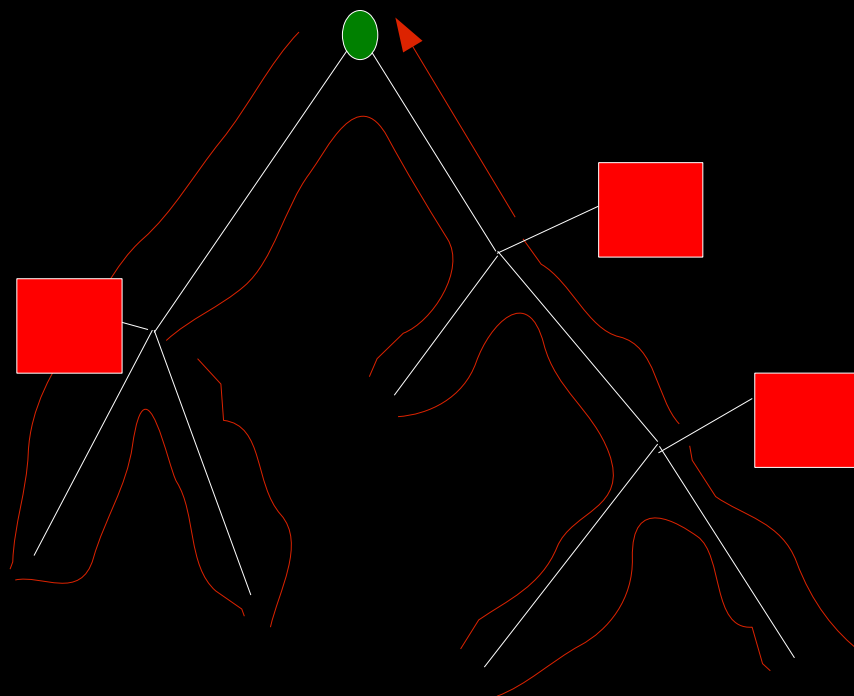
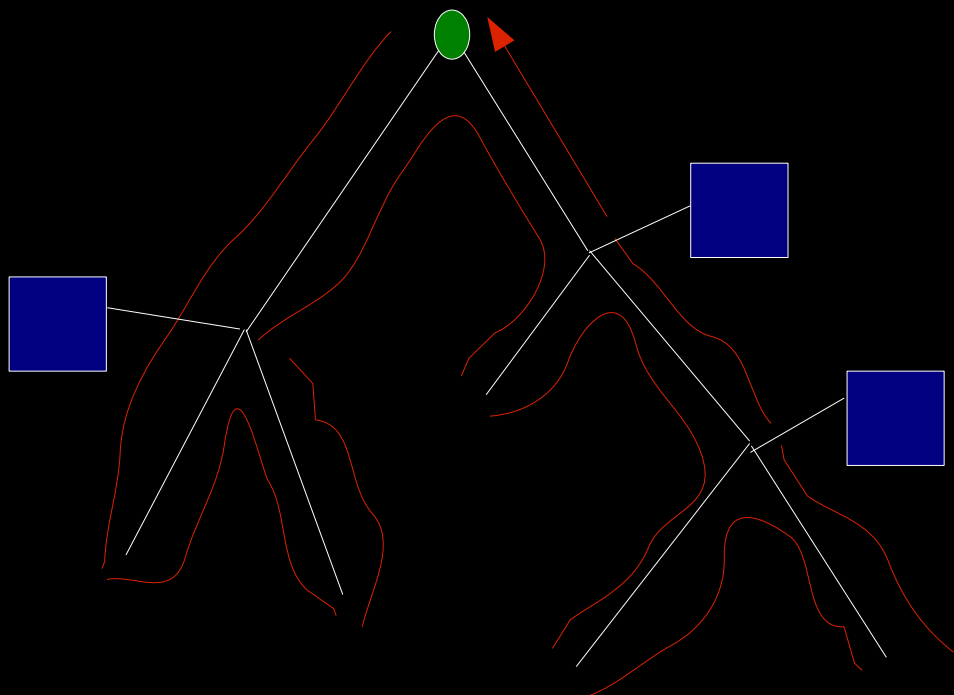
Parallel Numerics

2 cores

Overall score

$\Sigma \log(l_i)$

$\Sigma \log(l_i)$

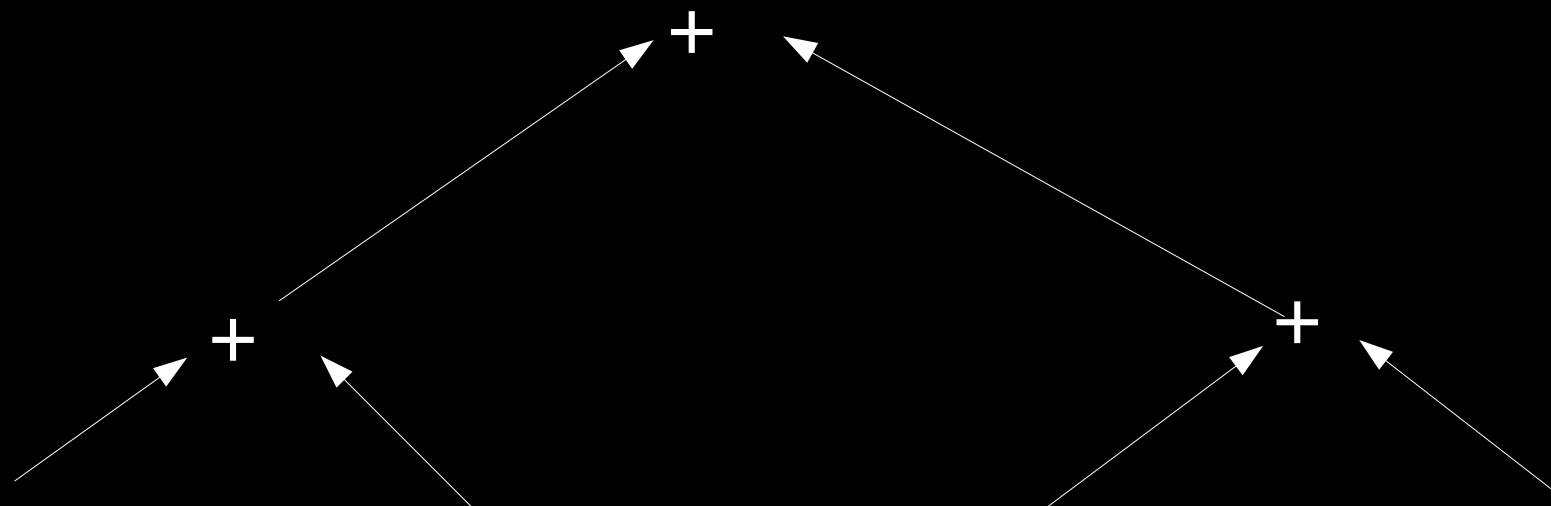


Parallel Numerics

4 cores

Overall score

Can be different!

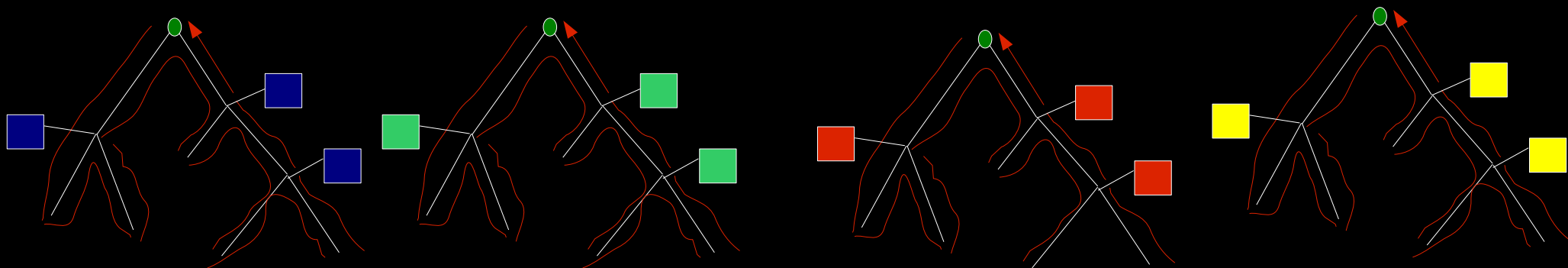


$\Sigma \log(l_i)$

$\Sigma \log(l_i)$

$\Sigma \log(l_i)$

$\Sigma \log(l_i)$

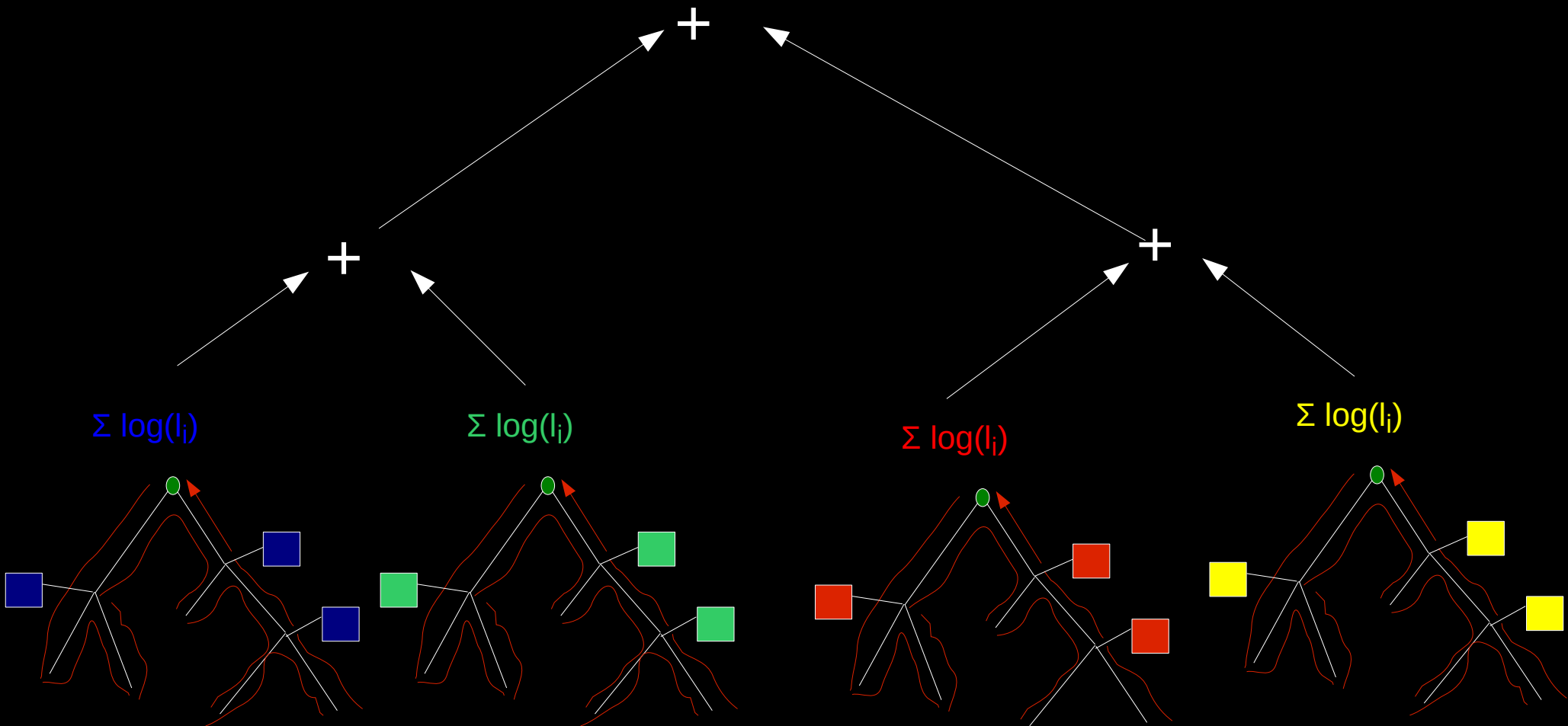


Parallel Numerics

4 cores

Overall score

Can be different \rightarrow
may return different tree

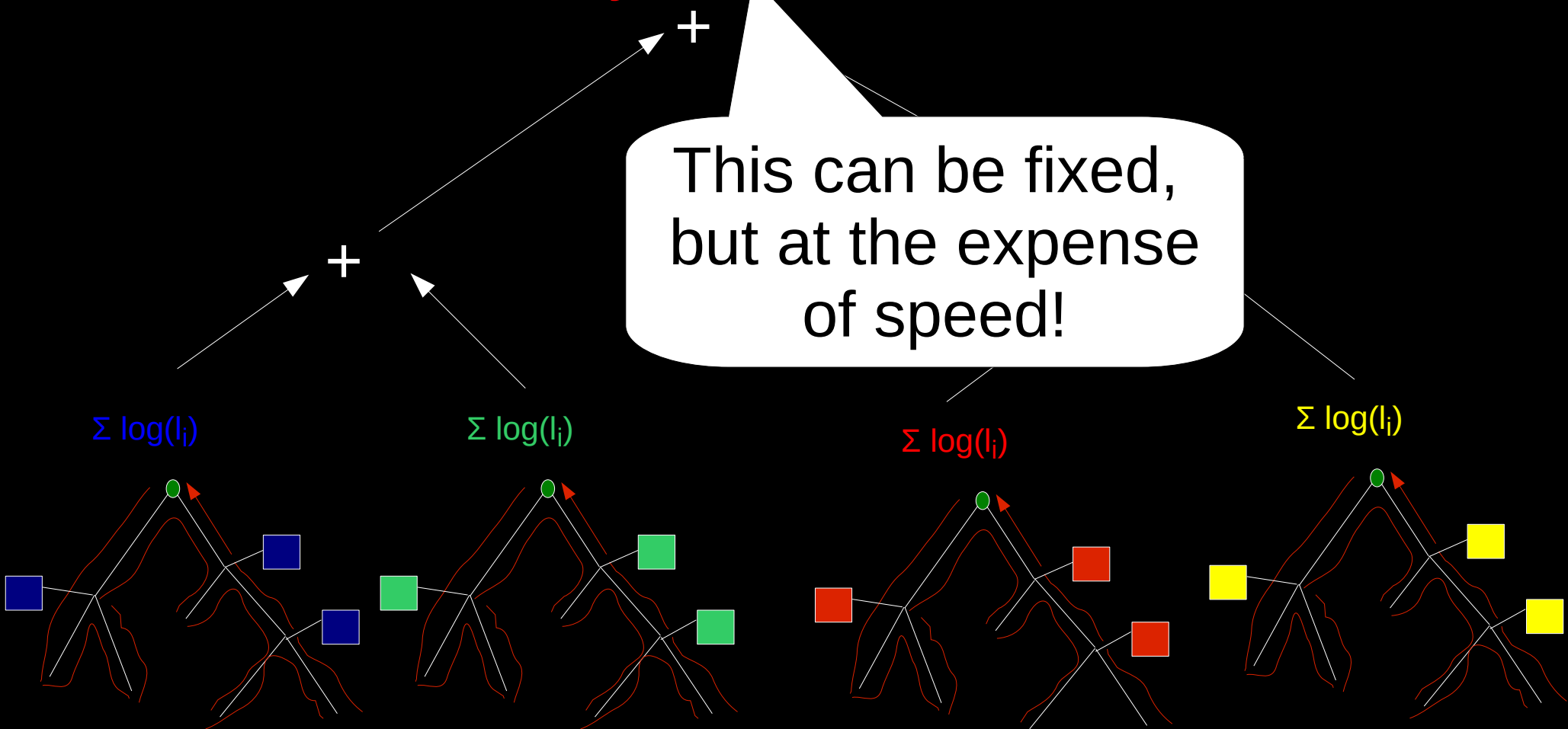


Parallel Numerics

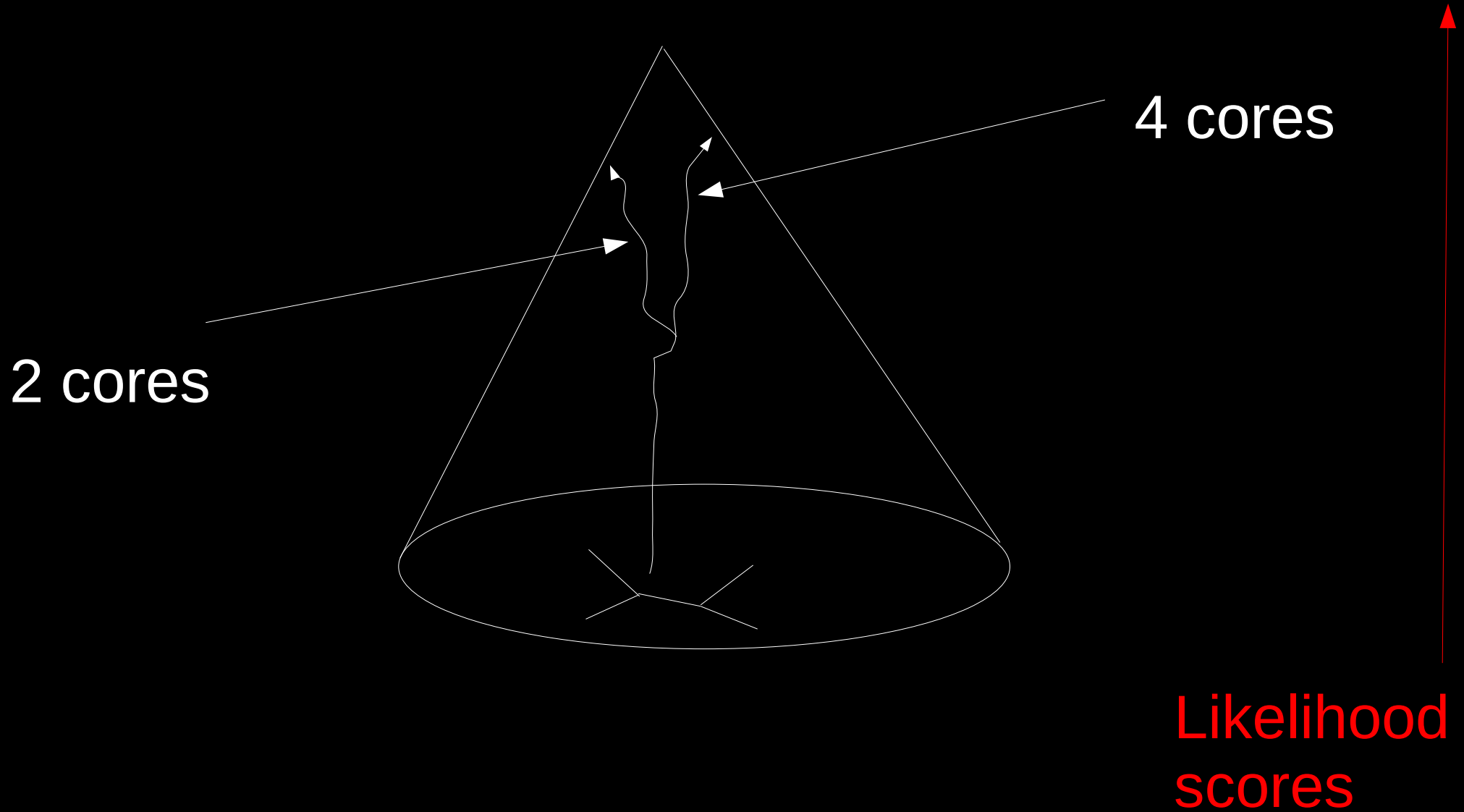
4 cores

Overall score

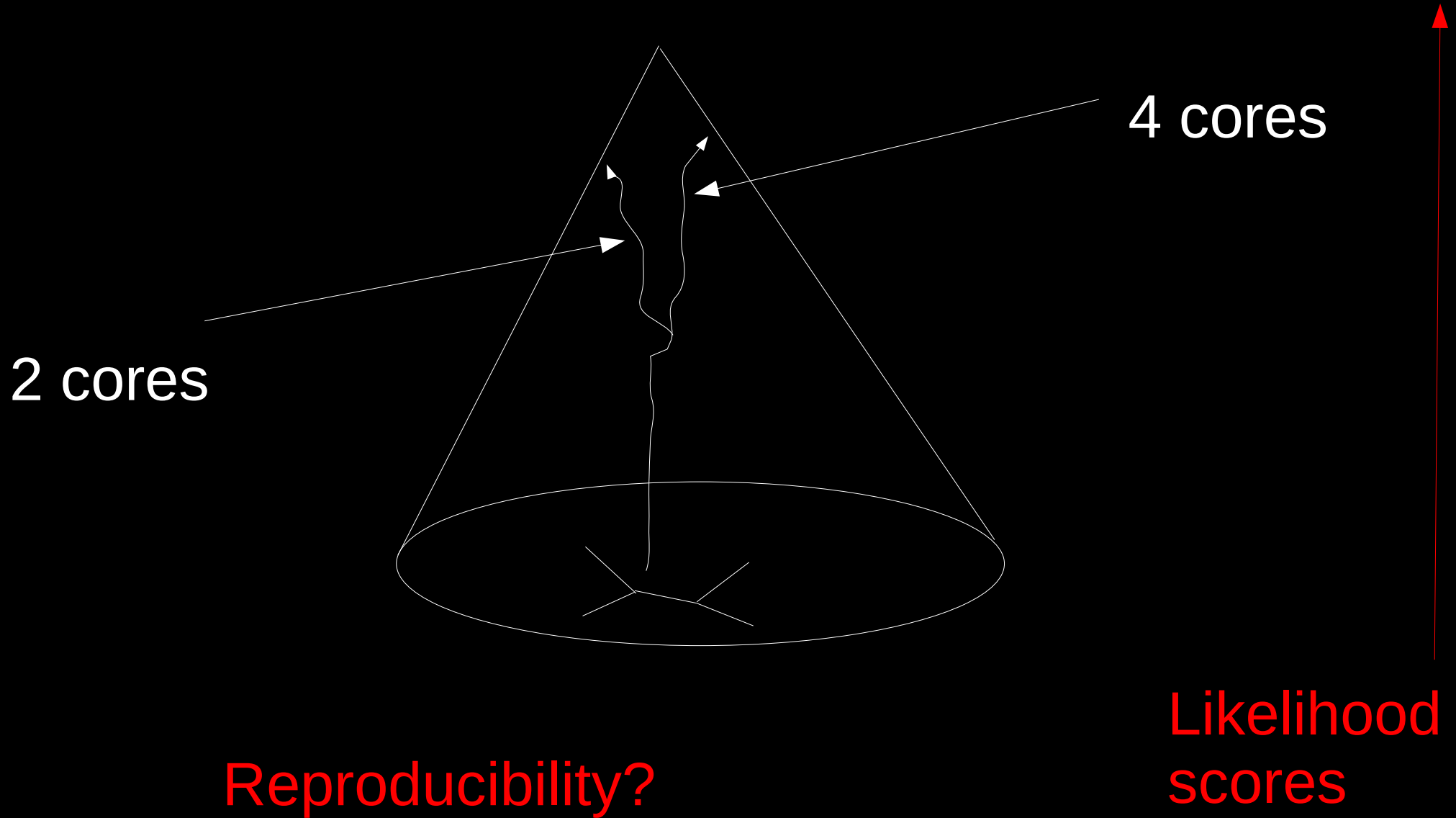
Can be different →
may return different tree



Diverging tree Searches



Diverging tree Searches




Instruction Level Parallelism

- All processors since around 1985 use pipelining to overlap (and hence parallelize and accelerate) the execution of instructions
- Goal: increase amount of parallelism and hence speed at a low (mostly HW) level
- Two distinct approaches:
 - HW-based & dynamic
 - RISC: **R**educed **I**nstruction **S**et **C**omputer/**C**omputing
 - SW (compiler)-based very long instruction words VLIW & static branch prediction
 - Intel Itanium processor
- ILP represents very fine-grained low-level parallelism
- When tuning performance you must keep in mind that there is a pipelined processor!

Levels of Parallelism

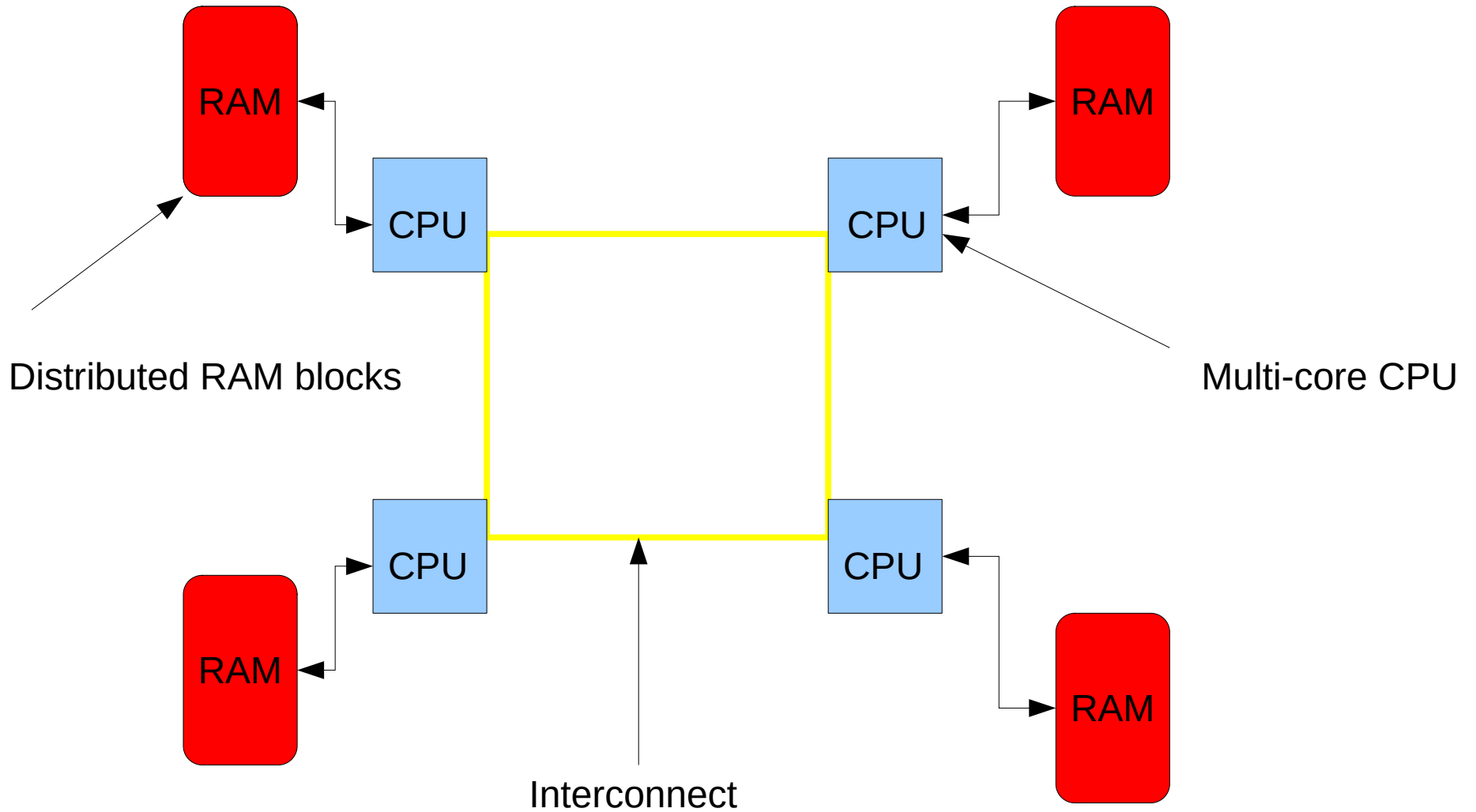
bottom-up

- 
- ILP in a single pipeline
 - SSE & AVX vector instructions on a single core
 - has anybody ever used vector intrinsics?
 - compilers are supposed to vectorize automatically
 - often this doesn't work
 - our experience: plain, good C code versus manually vectorized code: speedup factor 4.5 (with 256 bit AVX instructions)
 - Simultaneous multi-threading/Hyper-threading
 - Thread-based (OpenMP/Pthreads) on shared memory
 - multi-cores
 - mostly
 - loop-level parallelism
 - Fork-join paradigm
 - shared memory supercomputers
 - MPI across nodes
 - high bandwidth/low latency network interconnect
 - current state-of-the art: Infiniband technology
 - closely coupled parallel codes
 - Distributed/Cloud computing

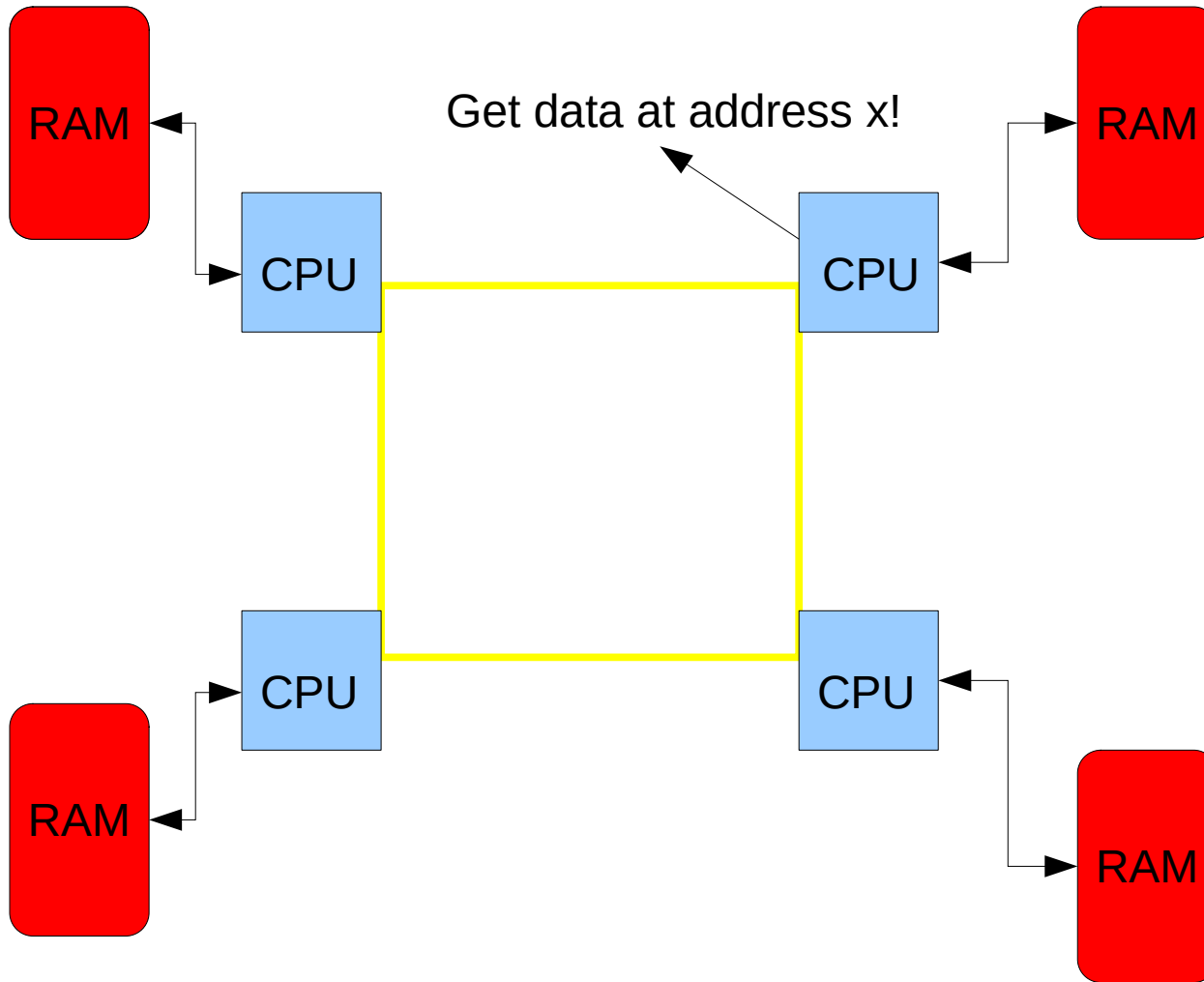
NUMA

- Taxonomy of shared memory computers based on RAM address access times
- UMA: Uniform Memory Access
- NUMA: Non-Uniform Memory Access
 - all multi-cores
 - we actually have distributed shared memory!

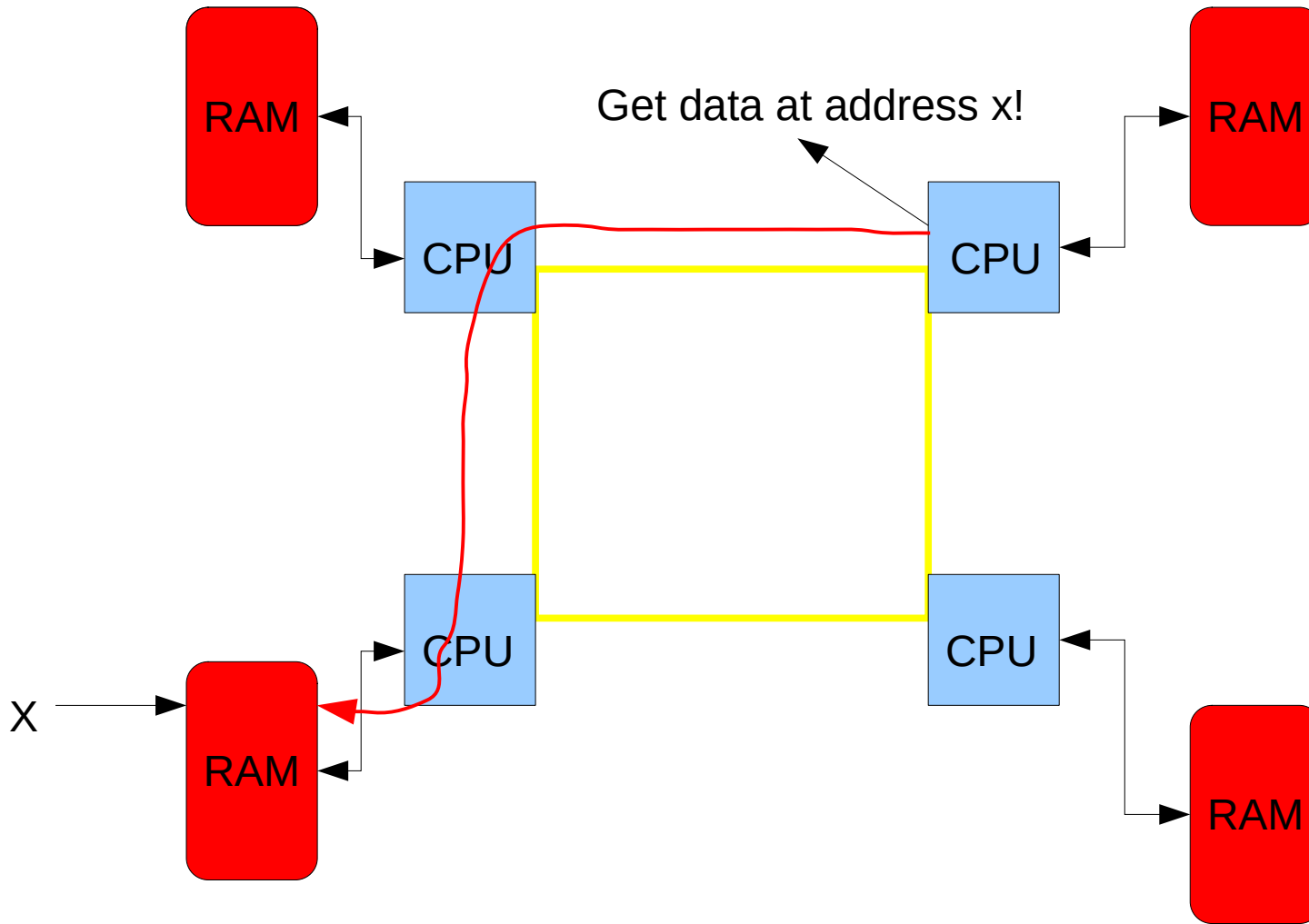
NUMA



NUMA



NUMA



NUMA

- We need to think about data locality when programming on a NUMA architecture!
- What happens when a thread on a NUMA system allocates memory?

NUMA

- We need to think about data locality when programming on a NUMA architecture!
- What happens when a thread on a NUMA system allocates memory?
 - first touch policy!
- Page gets allocated to RAM block closest to the first core that does a writing access
 - there also exist performance issues associated to *thread-to-core* pinning
 - can you imagine what could be the problem here

Top500 list

- List of top 500 supercomputer systems in the world
- Benchmarking based on LINPACK linear algebra package

| Rank | System | Cores | Rmax (TFlop/s) | Rpeak (TFlop/s) | Power (kW) |
|------|--|------------|-------------------|--------------------|---------------|
| 1 | Supercomputer Fugaku - Supercomputer Fugaku, 464FX44C 2.20Hz, Tolu interconnect D, Fujitsu RIKEN Center for Computational Science Japan | 7,630,848 | 442,010.0 | 537,212.0 | 29,899 |
| 2 | Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta BV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Dak Ridge National Laboratory United States | 2,414,592 | 148,400.0 | 200,794.9 | 10,896 |
| 3 | Sierra - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta BV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States | 1,572,480 | 94,640.0 | 125,712.0 | 7,438 |
| 4 | Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China | 10,649,600 | 93,014.6 | 125,435.9 | 15,371 |
| 5 | Perlmutter - HPE Cray EX23Sn, AMD EPYC 7742 64C 2.45GHz, NVIDIA A100 56MB 40 GB, Samsung 10, HPE DOE/SC/LBNL/NERSC United States | 761,856 | 70,870.0 | 93,750.0 | 2,589 |
| 6 | Sesame - NVIDIA DGX A100, AMD EPYC 7742 64C 2.45GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States | 950,320 | 63,460.0 | 79,215.0 | 2,446 |
| 7 | Tianhe-2A - TH-1VD-FEP Cluster, Intel Xeon E5-2692v2 12C 2.20Hz, TH Exstream-2, Mellanox-3000, NUDT National Super Computer Center in Guangzhou China | 4,761,760 | 61,444.5 | 100,678.7 | 18,482 |
| 8 | JUWELS Booster Module - Bull Sequana XH2000, AMD EPYC 7402 24C 2.80GHz, NVIDIA A100, Mellanox HDR InfiniBand/ParTec ParaStation Cluster/Swiss, Alza Forschungszentrum Juelich (FZJ) Germany | 649,280 | 44,120.0 | 70,980.0 | 1,764 |
| 9 | HPDC - PowerEdge C4140, Xeon Gold 6252 24C 2.10Hz, NVIDIA Tesla V100, Mellanox HDR InfiniBand, DELL EMC Eni S.p.A. Italy | 669,740 | 35,450.0 | 51,720.8 | 2,252 |
| 10 | Voyager-EU52 - NO9Kamer_A100_v4, AMD EPYC 7V12 48C 2.45GHz, NVIDIA A100 80GB, Mellanox HDR Infiniband, Microsoft Azure Azure East US 2 United States | 253,440 | 30,050.0 | 39,531.2 | |



Top500 list

- List of top 500 supercomputer systems in the world
- Benchmarking based on LINPACK linear algebra package

| Rank | System | Cores | Rmax (TFlop/s) | Rpeak (TFlop/s) | Power (kW) |
|------|---|------------|----------------|-----------------|------------|
| 1 | Supercomputer Fugaku 48C 2.20Hz, Tolu Interco RIKEN Center for Compu Japan | 7,630,848 | 442,010.0 | 537,212.0 | 29,899 |
| 2 | Summit IBM PowerPC A100 United States | 4,454,400 | 384,000.0 | 200,794.9 | 10,896 |
| 3 | Frontier AMD EPYC 7742 64C 2.255Hz, NVIDIA A100 Mellanox HDR InfiniBand United States | 4,454,400 | 344,640.0 | 125,712.0 | 7,438 |
| 4 | Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China | 10,649,600 | 93,014.6 | 125,435.9 | 15,371 |
| 5 | Perlmutter - HPE Cray EX23Sn, AMD EPYC 7742 64C 2.45GHz, NVIDIA A100 5xMiB 40 GB, Samsung 10, HPE DOE/SC/LLBNL/NERSC United States | 761,856 | 70,870.0 | 93,750.0 | 2,589 |
| 6 | Sesame - NVIDIA DGX A100, AMD EPYC 7742 64C 2.255Hz, NVIDIA A100, Mellanox HDR InfiniBand, Nvidia NVIDIA Corporation United States | 950,320 | 63,460.0 | 79,215.0 | 2,446 |
| 7 | Tianhe-2A - TH-10B-FEP Cluster, Intel Xeon E5-2692v2 12C 2.20Hz, TH Exstream-2, Mellanox-3000, NUDT National Super Computer Center in Guangzhou China | 4,791,760 | 61,444.5 | 100,678.7 | 18,482 |
| 8 | JUWELS Booster Module - Bull Sequana X92000, AMD EPYC 7402 24C 2.80Hz, NVIDIA A100, Mellanox HDR InfiniBand/ParTec ParaStation Cluster/Swiss, Alz Forschungszentrum Juelich (FZJ) Germany | 649,280 | 44,120.0 | 70,980.0 | 1,764 |
| 9 | HPDC - PowerEdge C4140, Xeon Gold 6352 24C 2.10Hz, NVIDIA Tesla V100, Mellanox HDR InfiniBand, DELL EMC Eni S.p.A. Italy | 669,740 | 35,450.0 | 51,720.8 | 2,252 |
| 10 | Voyager-EU52 - NVIDIA A100, AMD EPYC 7V12 48C 2.45GHz, NVIDIA A100 80GB, Mellanox HDR InfiniBand, Microsoft Azure Azure E54 US 2 United States | 253,440 | 30,050.0 | 39,531.2 | |

Who knows what weak scaling is?



OpenMP

- **Open** specifications for **Multi-Processing**
- Easy semi-automatic parallelization of codes for shared memory systems
- Just insert so-called **pragmas** into the code to parallelize loops (mostly)
- Based upon PThreads
- Needs a dedicated OpenMP-enabled compiler
- In comparison to direct usage of PThreads
 - faster implementation
 - provides the programmer less control over what is happening!

OpenMP Hello World

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

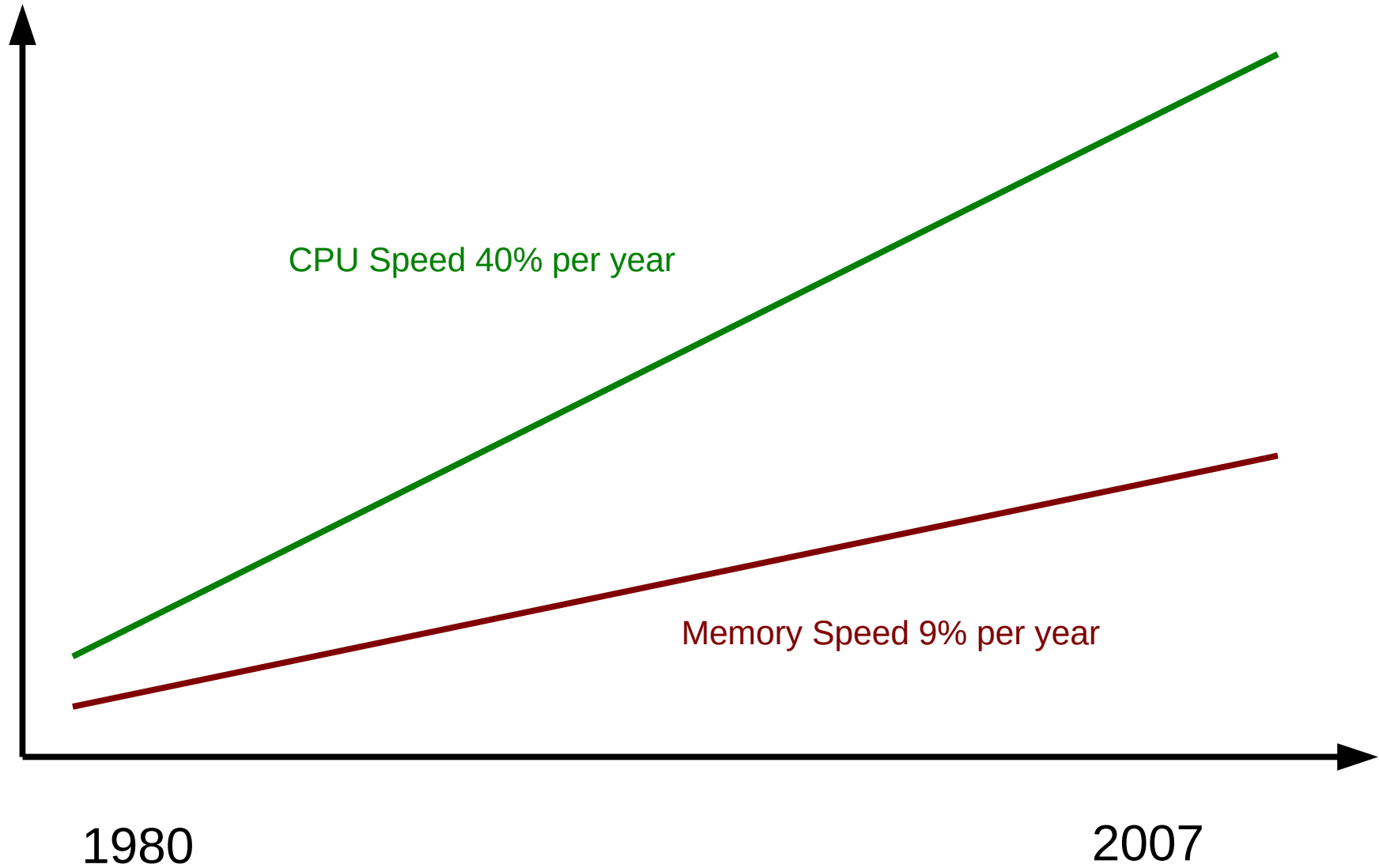
int main(void)
{
    int id,i;
    omp_set_num_threads(4);

    #pragma omp parallel for private(id)
    for (i = 0; i < 4; ++i)
    {
        id = omp_get_thread_num();

        printf("Hello World from thread %d\n", id);
    }

    return 0;
}
```

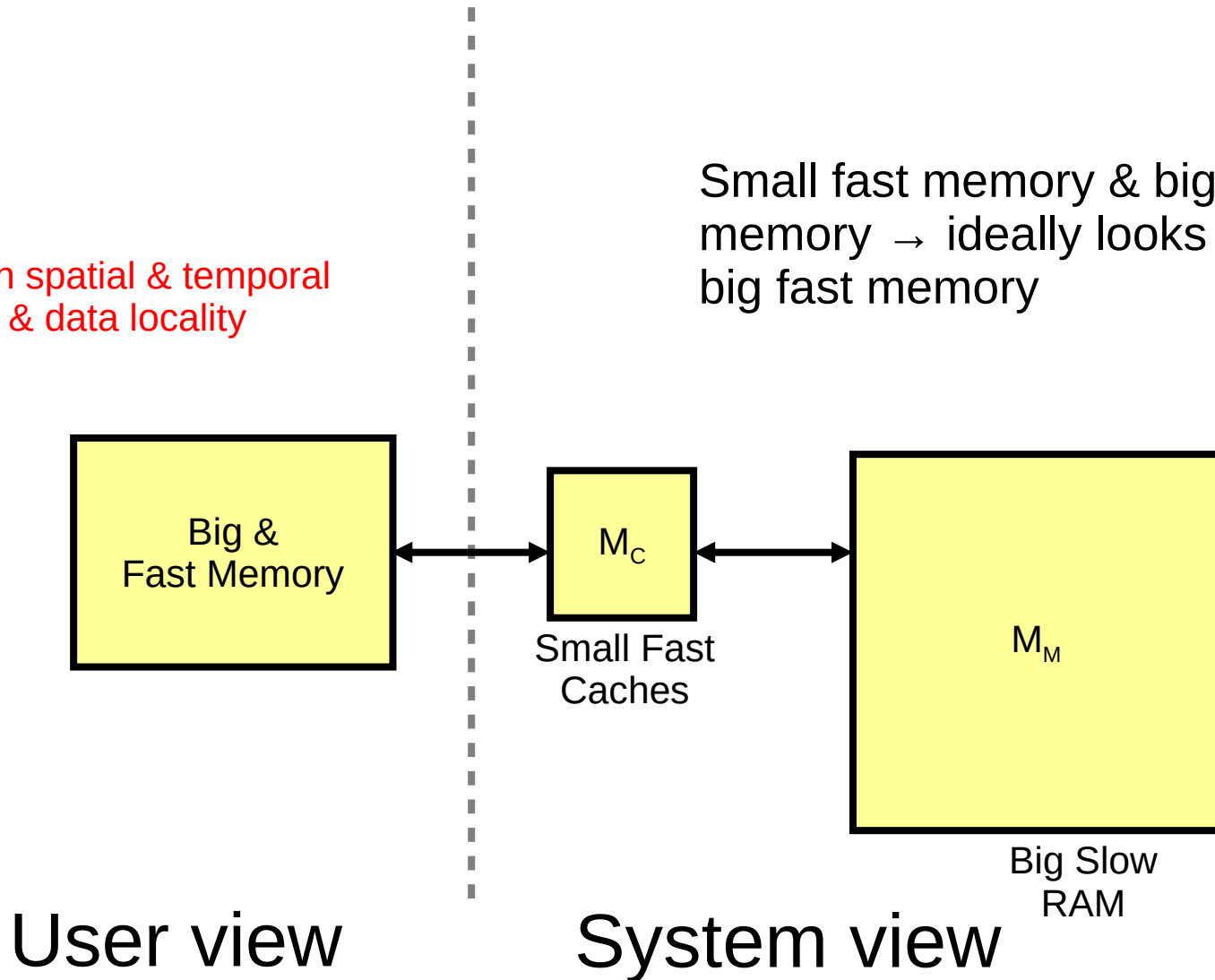

The Memory Gap



What we hope for

Given spatial & temporal
code & data locality

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



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

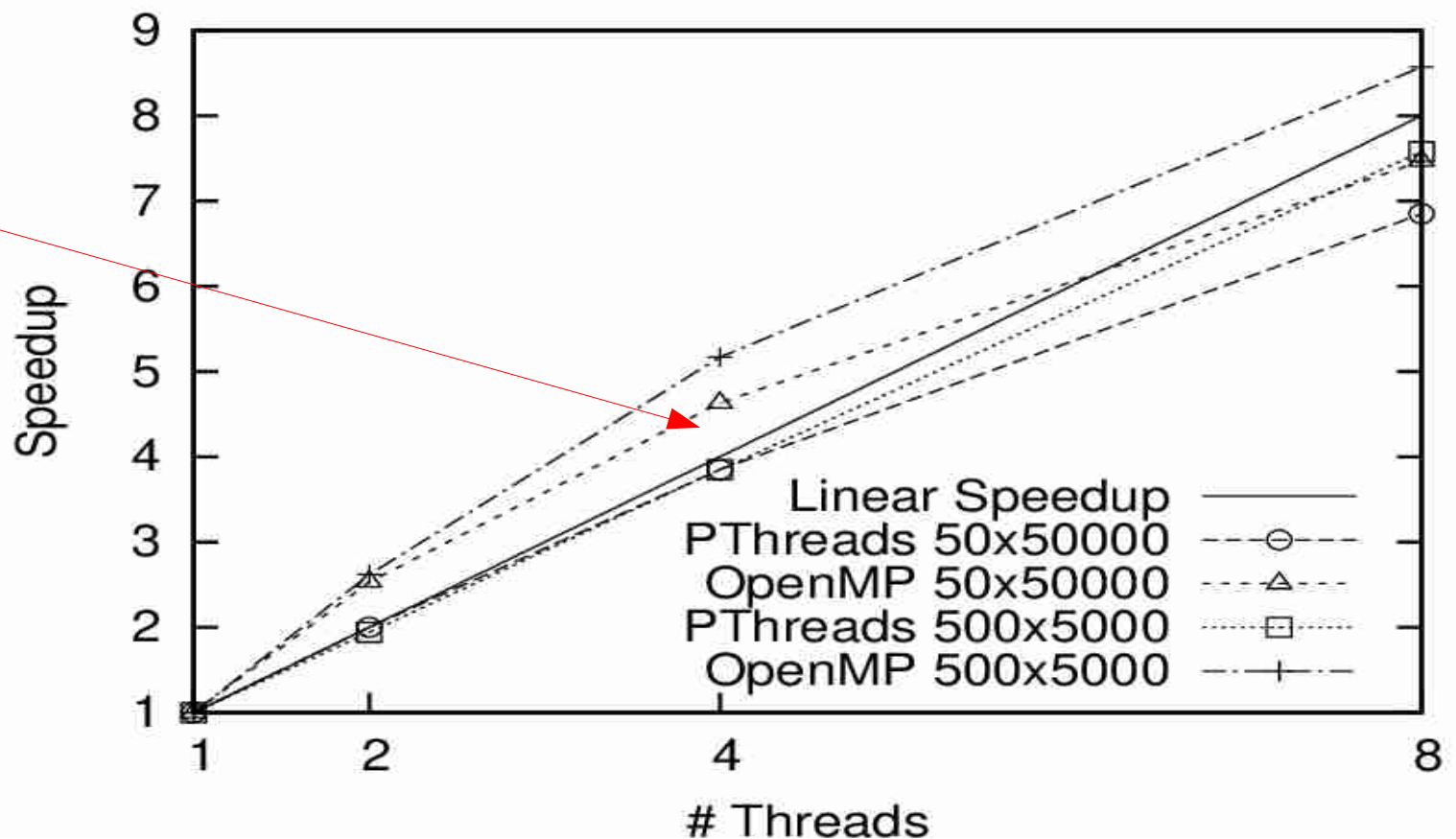
Measuring Parallel Performance

- *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:

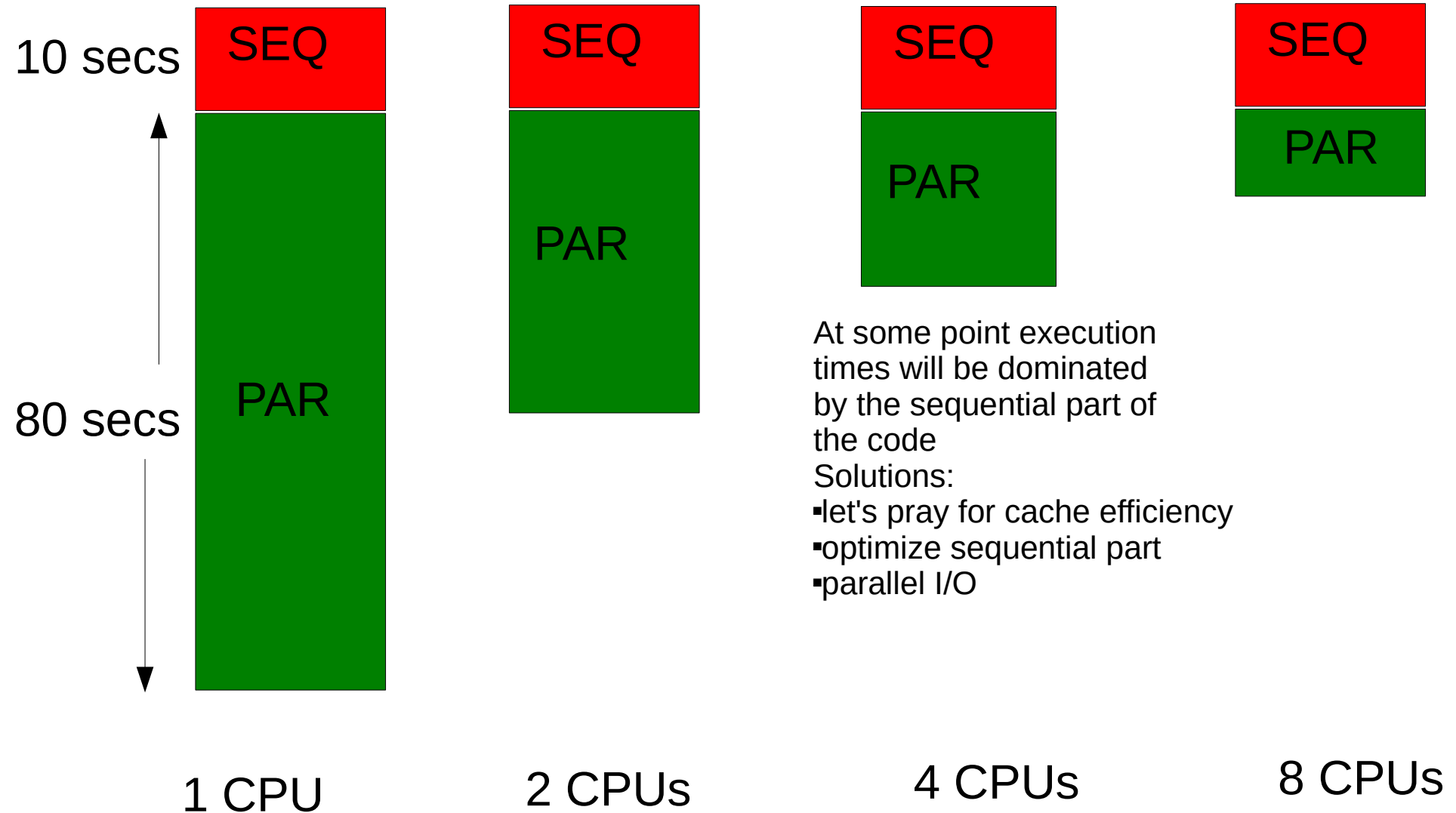
Super-linear speedups



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



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:

$$\text{speedup} \leq 1 / (f + ((1-f)/p))$$

where f is the fraction f with $0 \leq f \leq 1.0$ of the program that must be executed sequentially

- Thus for $p \rightarrow \text{infinity}$ the maximum speedup $S_{max} \leq 1/f$
- If the $f:=0.01 \rightarrow S_{max} \leq 100$:-)
- 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!

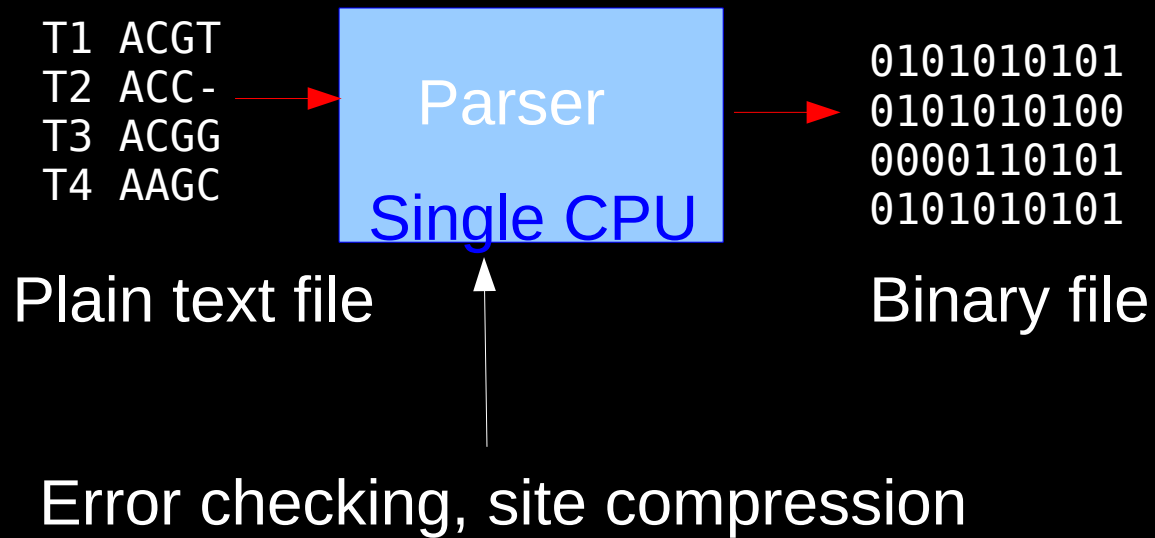
Amdahl's law

- This is a real issue
- Encountered it while developing our codes
ExaBayes and ExaML
- Reduced parallel I/O times at start-up from 15 to 1 minute on about 4000 cores!

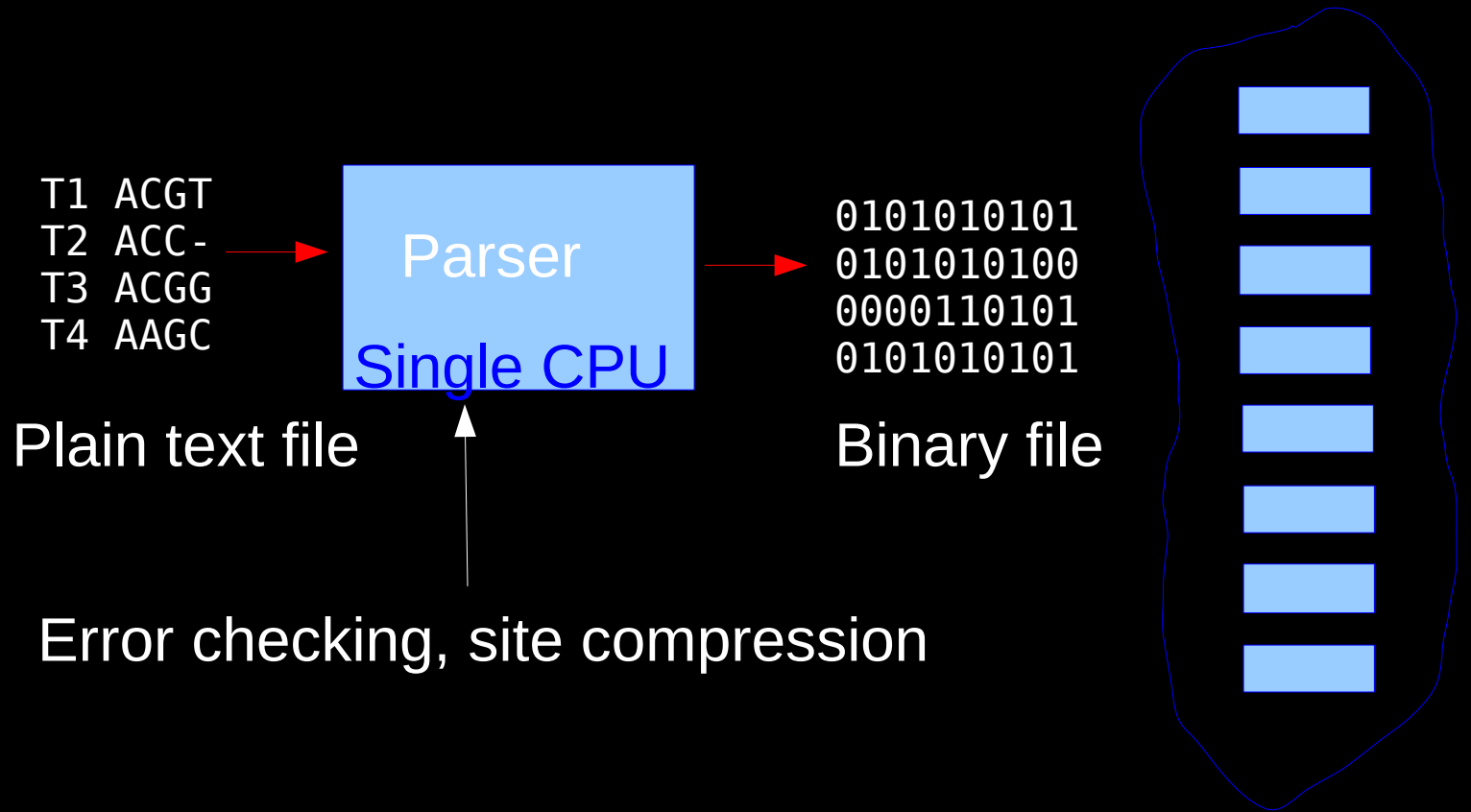
ExaBayes and ExaML

- Exascale Bayesian and Maximum Likelihood Phylogenetic Inference
- Codes for phylogenetic inference on supercomputers
- Scale up to 32,000 cores on SuperMUC x86 system
- Analyze datasets with 200 taxa and 100,000,000 sites

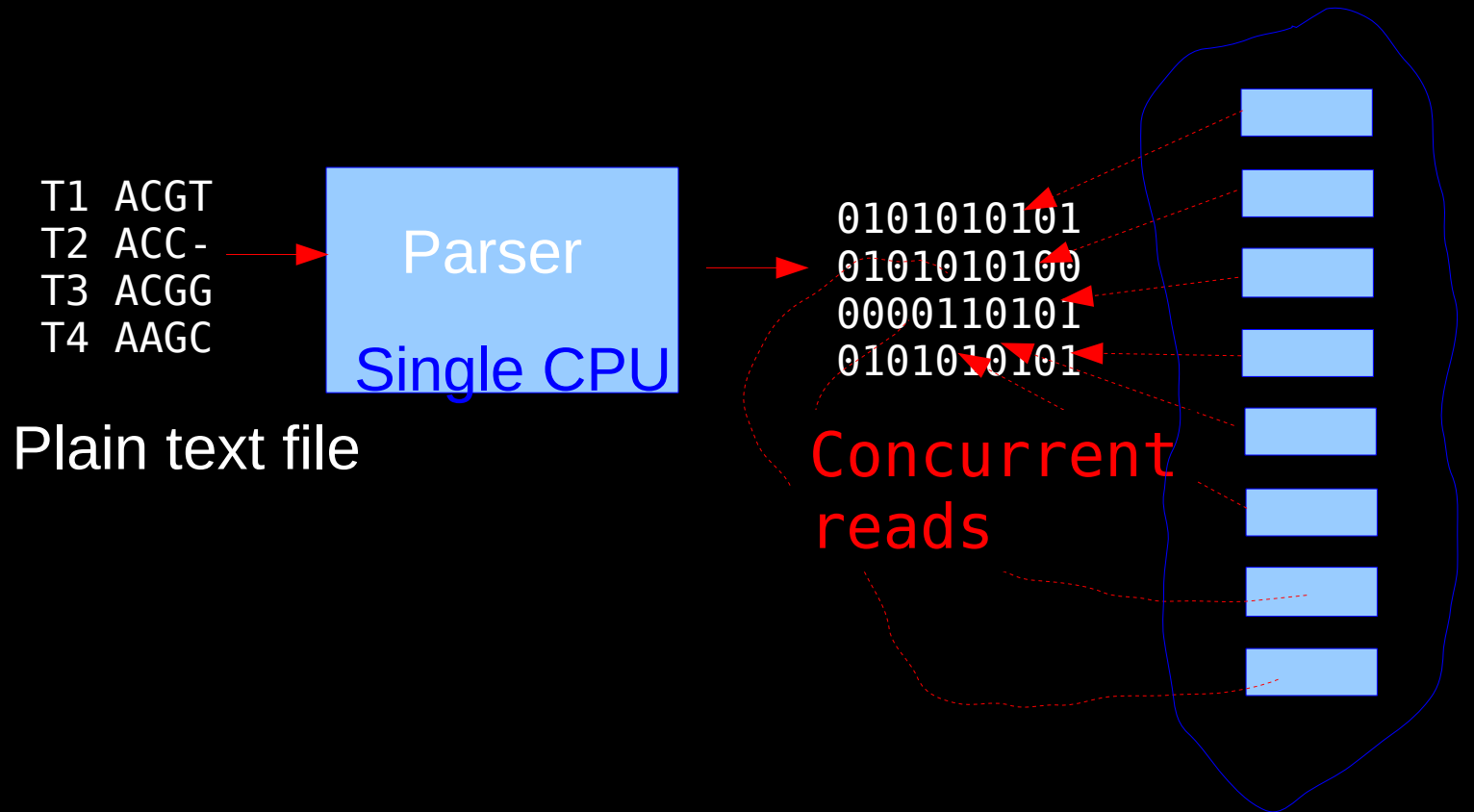
Handling I/O in ExaBayes and ExaML



Handling I/O in ExaBayes and ExaML



Handling I/O in ExaBayes and ExaML



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 dilemma is to present results for an inner kernel of an application, which is souped up with artificial tricks. Then imply in your presentation that these rates are equivalent to the overall performance of the application.

Full list at:

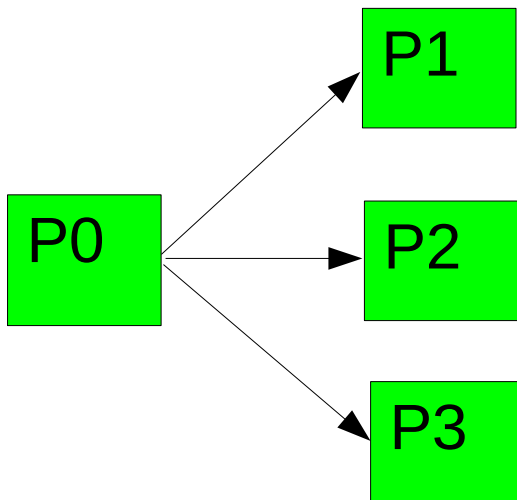
<http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/twelve-ways.pdf>

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

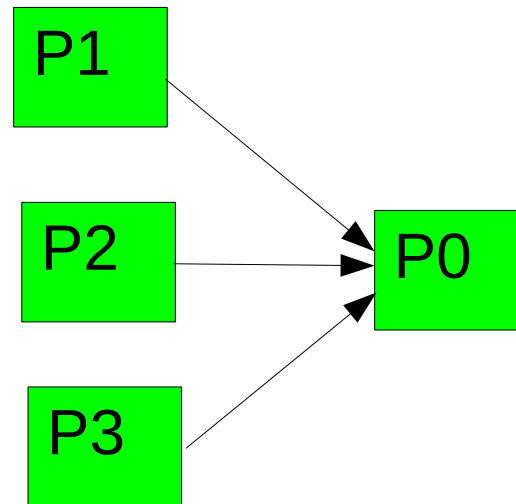
- **Message Passing Interface**
- **Send-Receive Paradigm**
- **Point to Point Communication**
- **Collective Communication**

Communication Schemes

Direct/point-to-point



broadcast/multicast



indirect/reduction

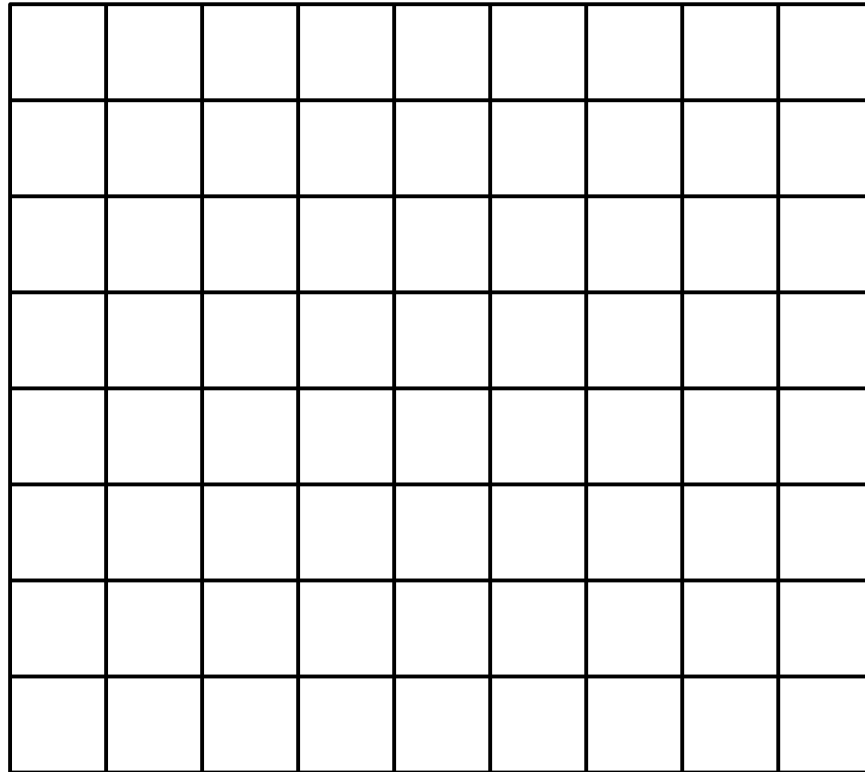
Classic MPI Programming Error

```
MPI_Init (&argc, &argv);  
/* starts MPI */  
MPI_Comm_rank (MPI_COMM_WORLD, &rank);  
/* get current process id */  
MPI_Comm_size (MPI_COMM_WORLD, &size);  
/* get number of processes */
```

- Those three commands must be the very first at the beginning of `main()` !

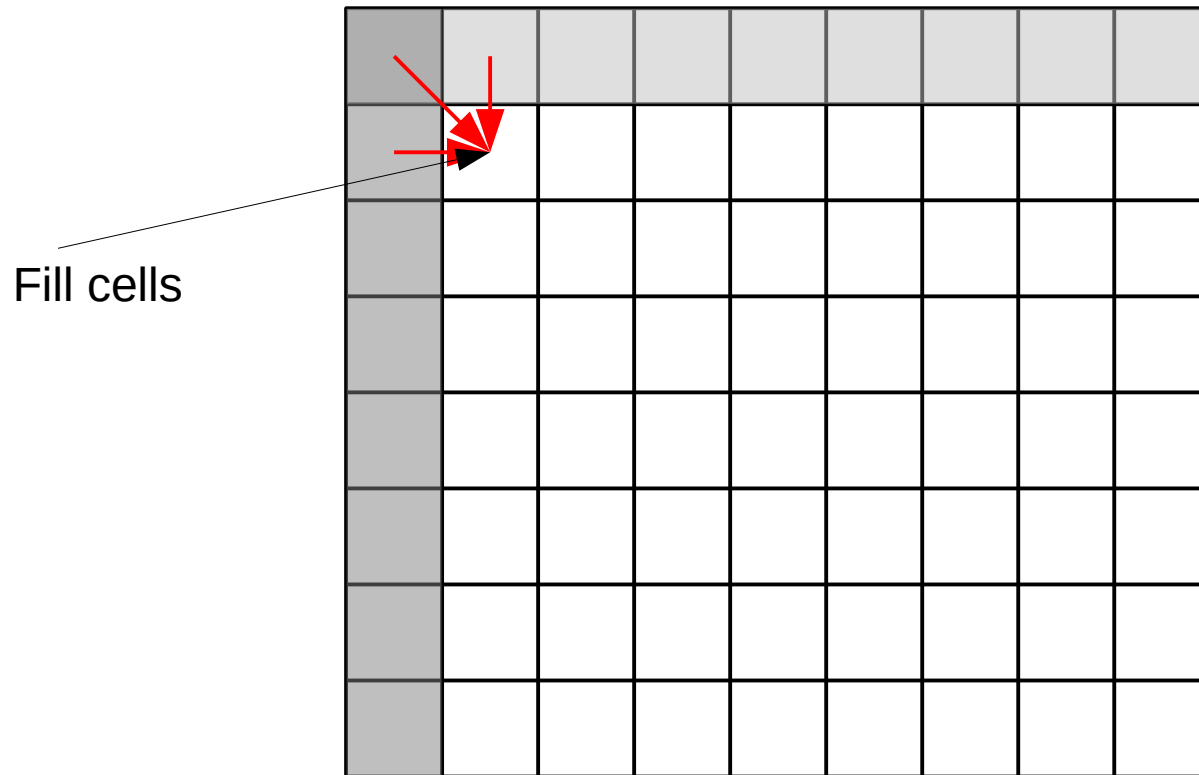
Wave-Front Parallelism

- Dynamic Programming algorithms



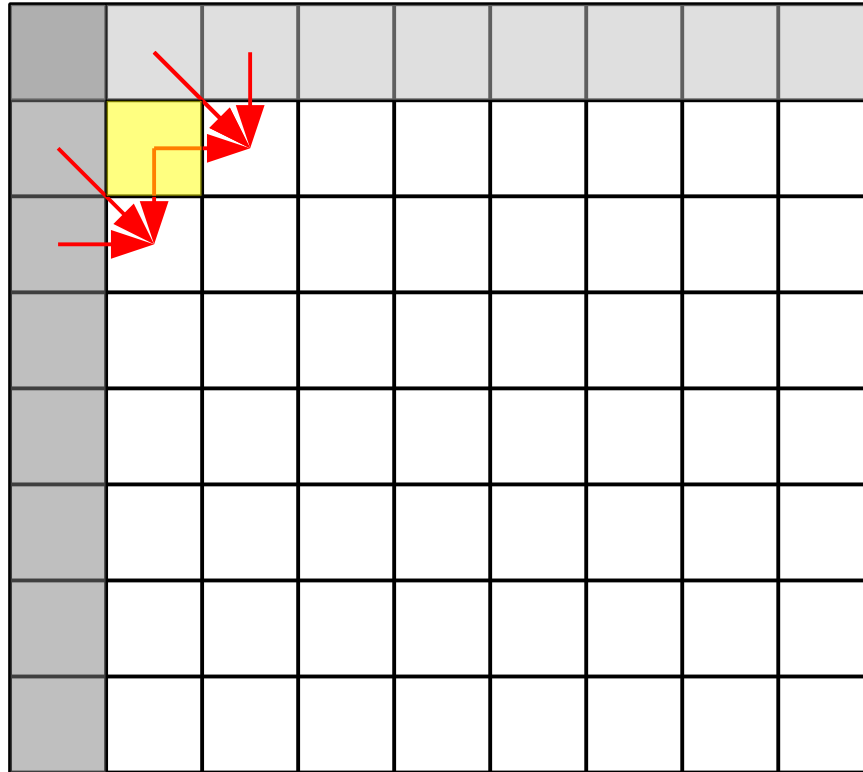
Wave-Front Parallelism

- Dynamic Programming algorithms



Wave-Front Parallelism

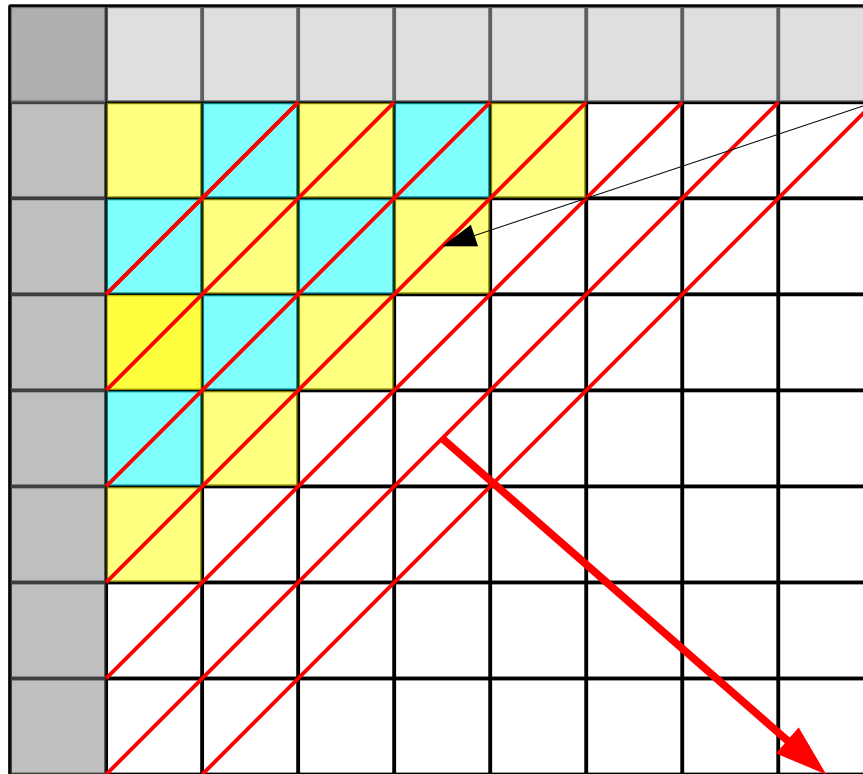
- Dynamic Programming algorithms



Wave-Front Parallelism

- Dynamic Programming algorithms

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



Other ways to vectorize a Dynamic Programming Matrix?

- Assume we have a reference sequence r and a lot of query sequences q_1, \dots, q_n that we want to align to r
- Is there an option, other than wavefront, to parallelize this?

Other ways to vectorize a Dynamic Programming Matrix?

- Assume we have a reference sequence r and a lot of query sequences q_1, \dots, q_n that we want to align to r
- Is there an option, other than wavefront, to parallelize this?
 - yes, it's called intra-sequence vectorization :-)

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!