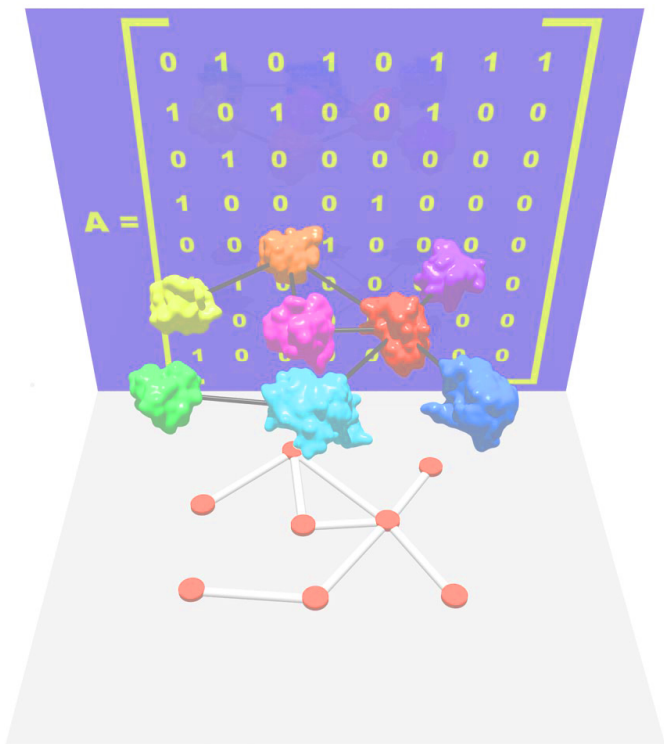
Statistical analysis of network data and evolution on GPUs: High-performance statistical computing



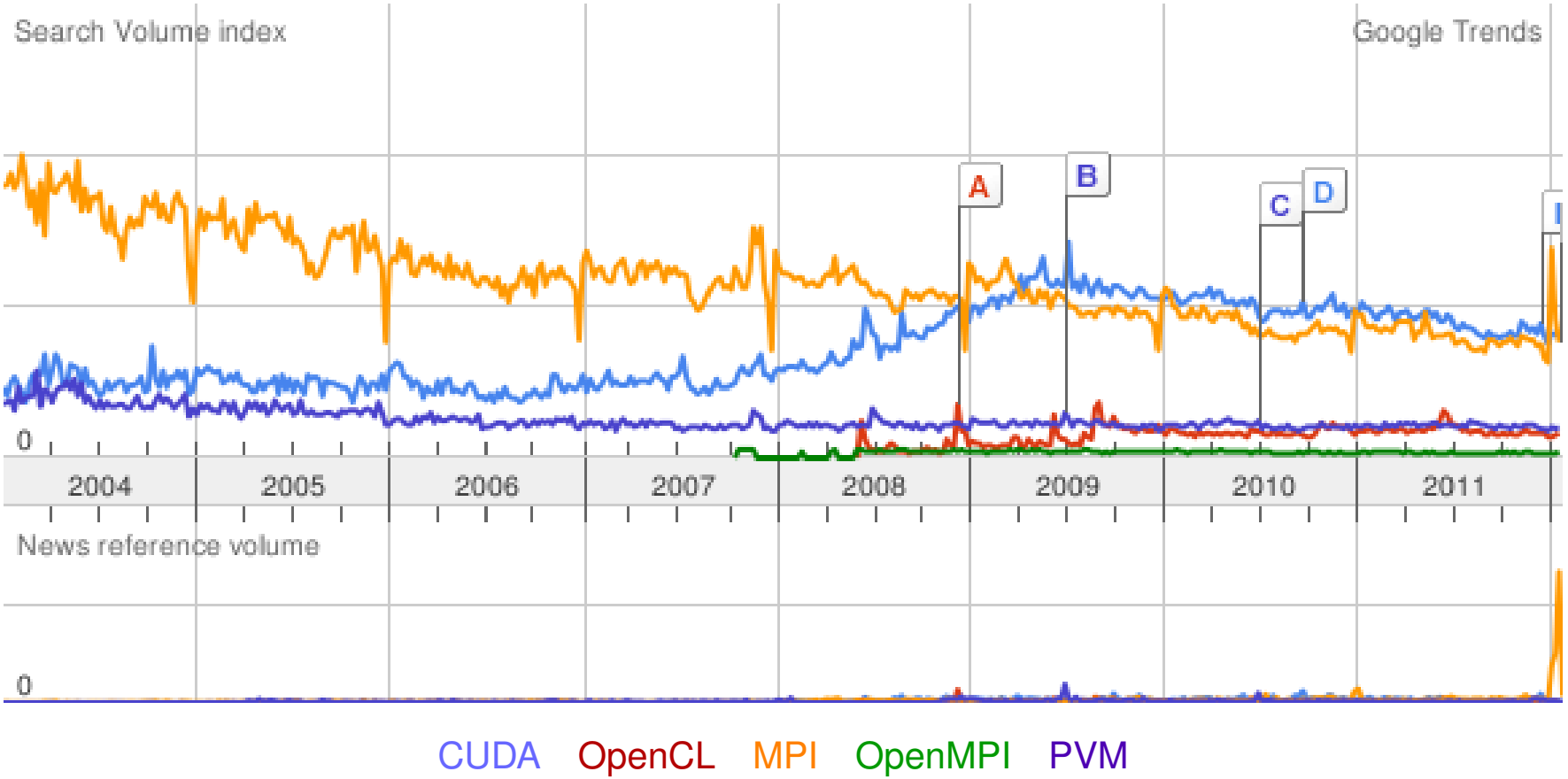
Thomas Thorne & Michael P.H. Stumpf

Theoretical Systems Biology Group

25/01/2012

Imperial College London

Trends in High-Performance Computing



CUDA OpenCL MPI OpenMPI PVM

(Hidden) Costs of Computing

Prototyping/Development Time is typically reduced for scripting languages such as R or Python.

(Hidden) Costs of Computing

Prototyping/Development Time is typically reduced for scripting languages such as R or Python.

Run Time on single threads C/C++ (or Fortran) have better performance characteristics. But for specialized tasks other languages, e.g. Haskell, can show good characteristics.

(Hidden) Costs of Computing

Prototyping/Development Time is typically reduced for scripting languages such as R or Python.

Run Time on single threads C/C++ (or Fortran) have better performance characteristics. But for specialized tasks other languages, e.g. Haskell, can show good characteristics.

Energy Requirements Every Watt we use for computing we also have to extract with air conditioning.

(Hidden) Costs of Computing

Prototyping/Development Time is typically reduced for scripting languages such as R or Python.

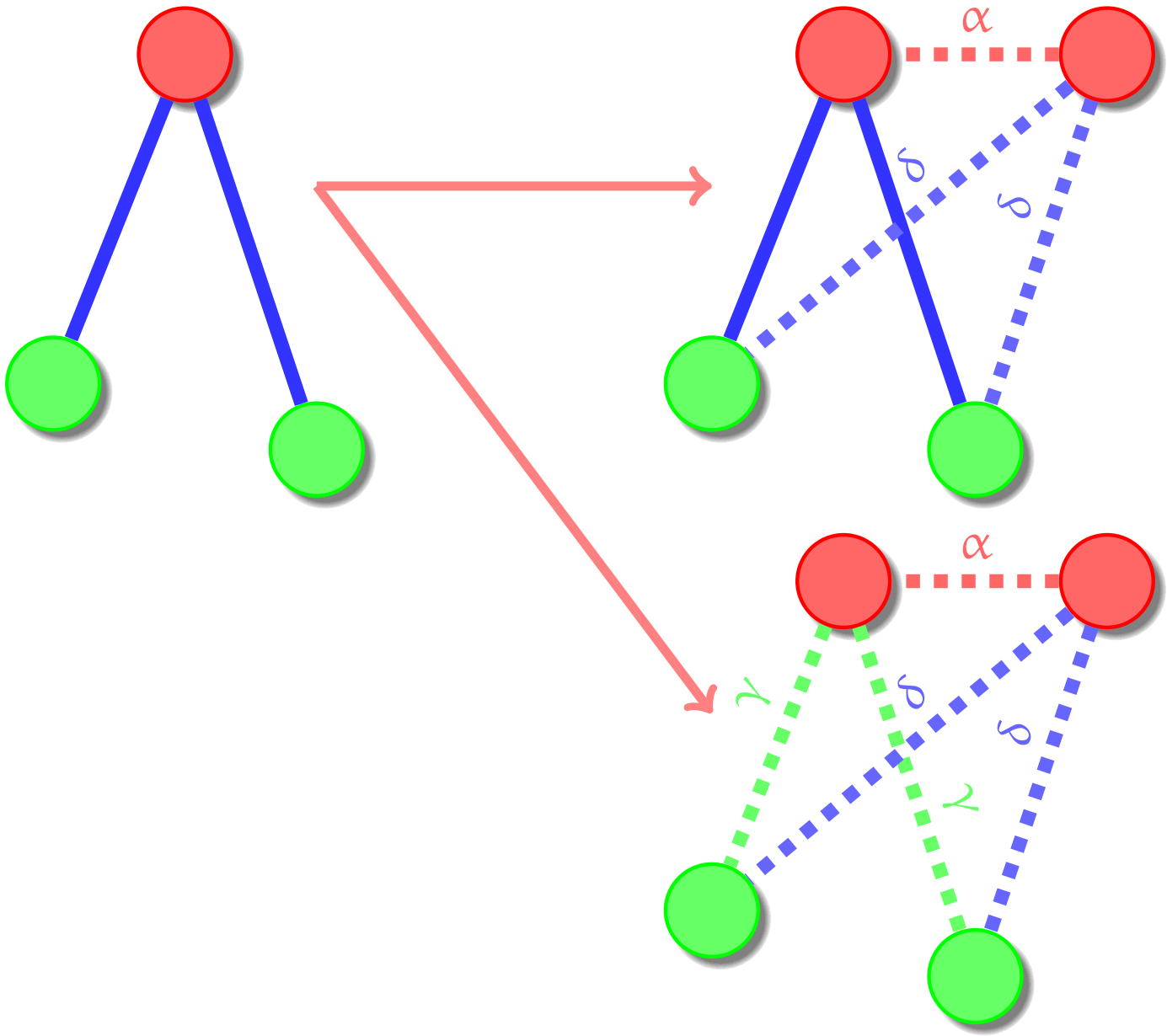
Run Time on single threads C/C++ (or Fortran) have better performance characteristics. But for specialized tasks other languages, e.g. Haskell, can show good characteristics.

Energy Requirements Every Watt we use for computing we also have to extract with air conditioning.

The role of GPUs

- GPUs can be accessed from many different programming languages (e.g. PyCUDA).
- GPUs have a comparatively small footprint and relatively modest energy requirements compared to clusters of CPUs.
- GPUs were designed for consumer electronics: computer gamers have different needs from the HPC community.

Evolving Networks

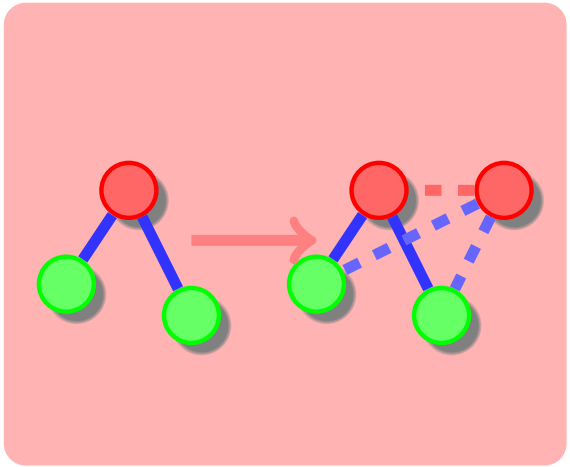


Evolving Networks

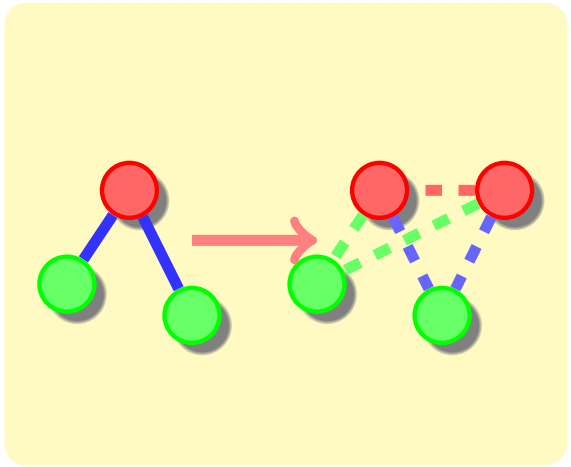
Model-Based Evolutionary Analysis

- For sequence data we use models of nucleotide substitution in order to infer phylogenies in a likelihood or Bayesian framework.
- None of these models — even the general time-reversible model — are particularly realistic; but by allowing for complicating factors e.g. rate variation we capture much of the variability observed across a phylogenetic panel.
- Modes of network evolution will be even more complicated and exhibit high levels of contingency; moreover the structure and function of different parts of the network will be intricately linked.
- Nevertheless we believe that modelling the processes underlying the evolution of networks can provide useful insights; in particular we can study how functionality is distributed across groups of genes.

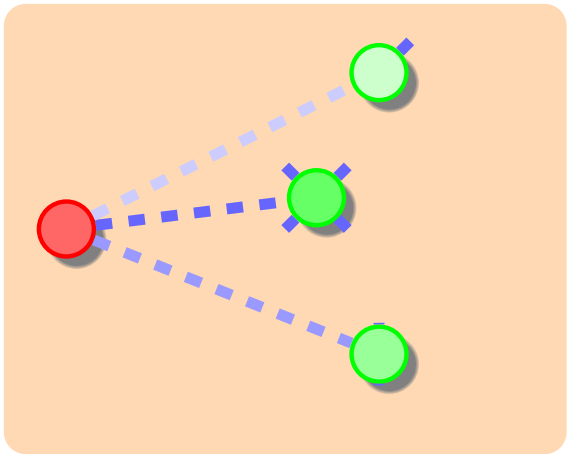
Network Evolution Models



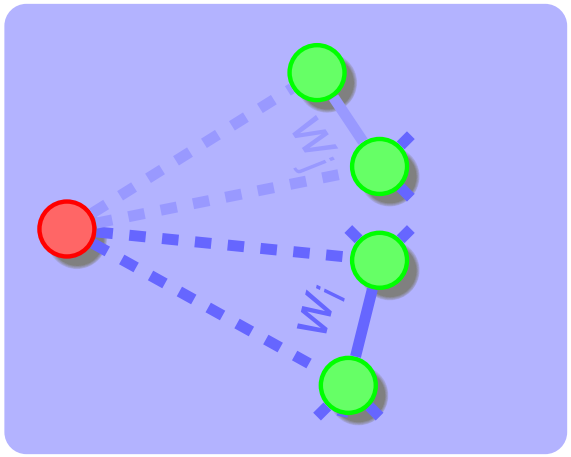
(a) Duplication attachment



(b) Duplication attachment with complimentarity



(c) Linear preferential attachment



(d) General scale-free

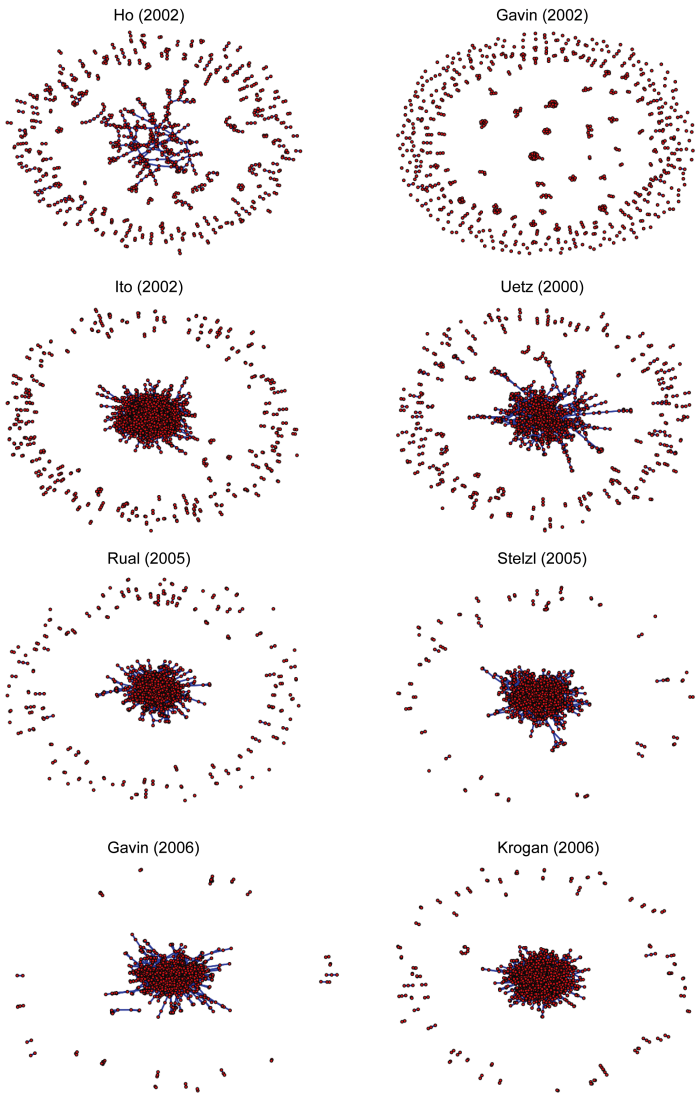
ABC on Networks

Summarizing Networks

- Data are noisy and incomplete.
- We can simulate models of network evolution, but this does not allow us to calculate likelihoods for all but very trivial models.
- There is also no sufficient statistic that would allow us to summarize networks, so ABC approaches require some thought.
- Many possible summary statistics of networks are expensive to calculate.

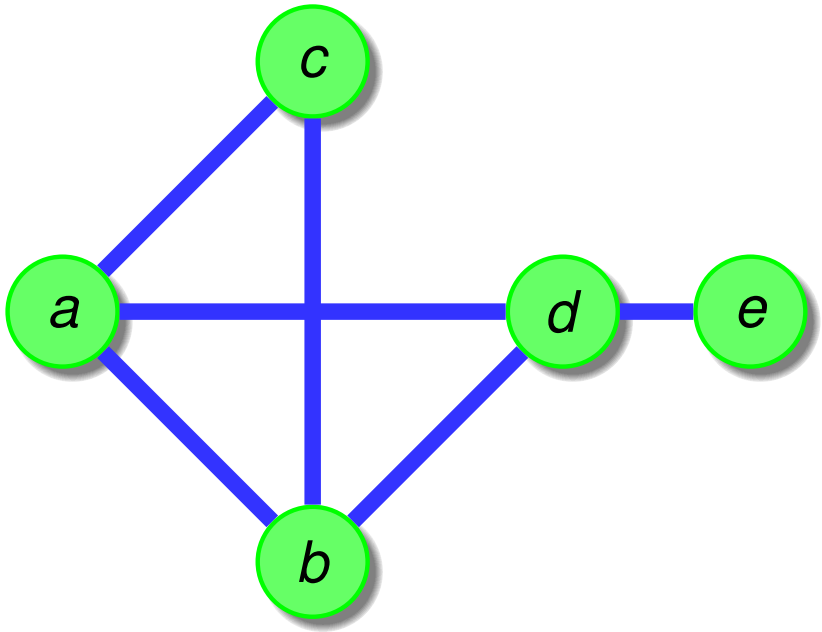
Full likelihood: Wiuf et al., PNAS (2006).

ABC: Ratman et al., PLoS Comp.Biol. (2008).



Stumpf & Wiuf, J. Roy. Soc. Interface (2010).

Graph Spectrum



$$A = \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \begin{array}{c} a \\ b \\ c \\ d \\ e \end{array}$$

Graph Spectra

Given a graph G comprised of a set of nodes N and edges $(i, j) \in E$ with $i, j \in N$, the adjacency matrix, A, of the graph is defined by

$$a_{i,j} = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

The eigenvalues, λ, of this matrix provide one way of defining the graph spectrum.

Spectral Distances

A simple distance measure between graphs having adjacency matrices A and B, known as the edit distance, is to count the number of edges that are not shared by both graphs,

$$D(A, B) = \sum_{i,j} (a_{i,j} - b_{i,j})^2.$$

Spectral Distances

A simple distance measure between graphs having adjacency matrices A and B, known as the edit distance, is to count the number of edges that are not shared by both graphs,

$$D(A,B) = \sum_{i,j} (a_{i,j} - b_{i,j})^2.$$

However for unlabelled graphs we require some mapping h from $i \in N_A$ to $i' \in N_B$ that minimizes the distance

$$D(A,B) \geqslant D'_h(A,B) = \sum_{i,j} (a_{i,j} - b_{h(i),h(j)})^2,$$

Spectral Distances

A simple distance measure between graphs having adjacency matrices A and B, known as the edit distance, is to count the number of edges that are not shared by both graphs,

$$D(A,B) = \sum_{i,j} (a_{i,j} - b_{i,j})^2.$$

However for unlabelled graphs we require some mapping h from $i \in N_A$ to $i' \in N_B$ that minimizes the distance

$$D(A,B) \geqslant D'_h(A,B) = \sum_{i,j} (a_{i,j} - b_{h(i),h(j)})^2,$$

Given a spectrum (which is relatively cheap to compute) we have

$$D'(A,B) = \sum_{l} \left(\lambda_l^{(\alpha)} - \lambda_l^{(\beta)}\right)^2$$

ABC using Graph Spectra

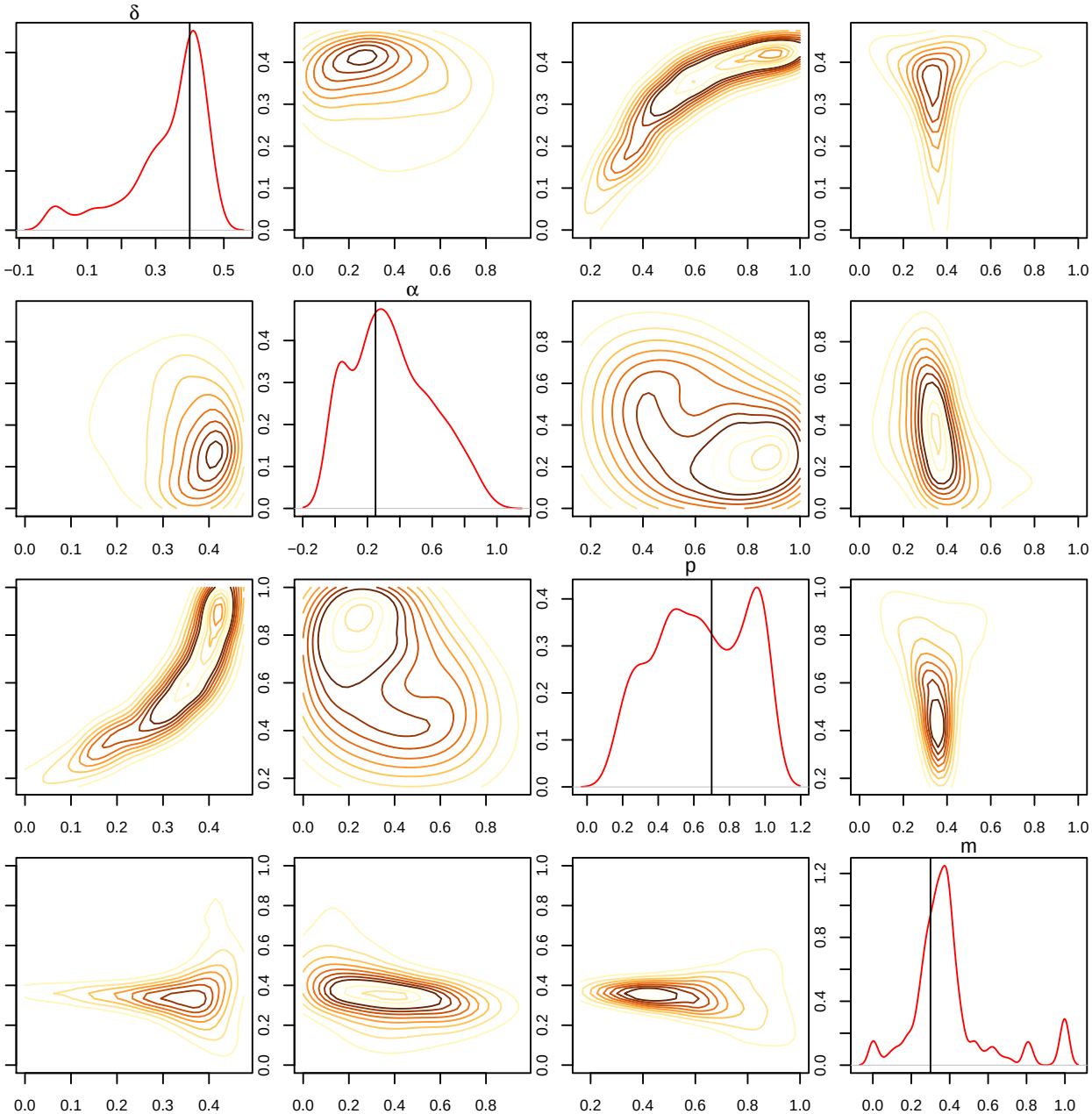
For an observed network, $\mathcal{N}$, and a simulated network, $\mathcal{S}_\theta$, we use the distance between the spectra

$$D'(\mathcal{N}, \mathcal{S}_\theta) = \sum_l \left(\lambda_l^{(\mathcal{N})} - \lambda_l^{(\mathcal{S})}\right)^2,$$

in our ABC SMC procedure. Note that this distance is a close lower bound on the distance between the raw data; we therefore do not have to bother with summary statistics.

Also, calculating graph spectra costs as much as calculating other $O(N^3)$ statistics (such as *all shortest paths*, the *network diameter* or the *within-reach distribution*).

Simulated Data

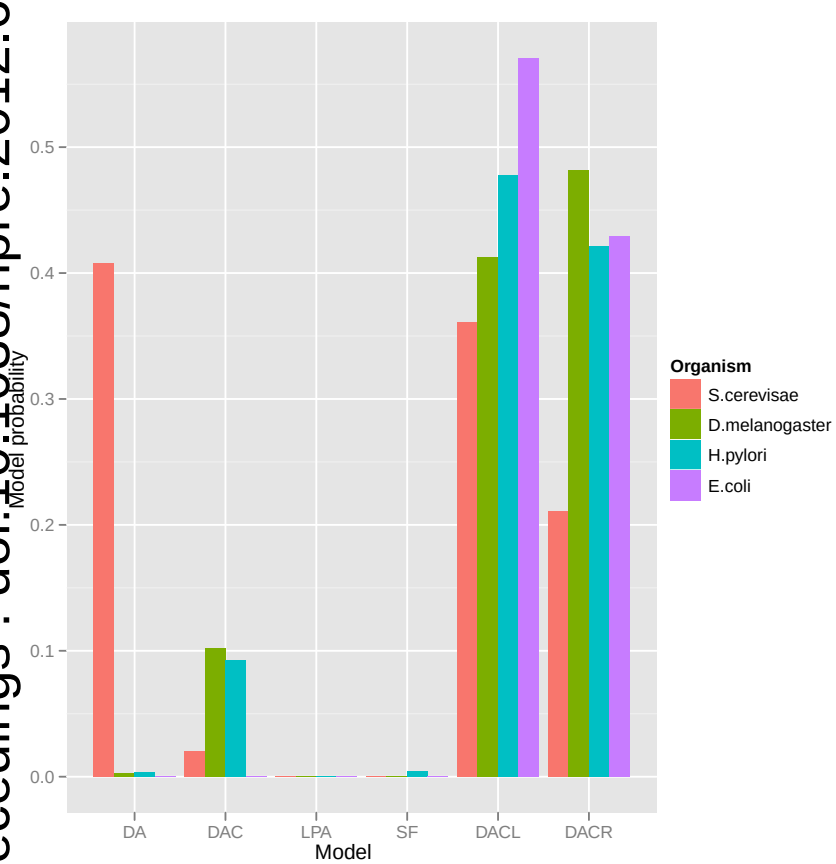


Protein Interaction Network Data

Species	Proteins	Interactions	Genome size	Sampling fraction
S.cerevisiae	5035	22118	6532	0.77
D. melanogaster	7506	22871	14076	0.53
H. pylori	715	1423	1589	0.45
E. coli	1888	7008	5416	0.35

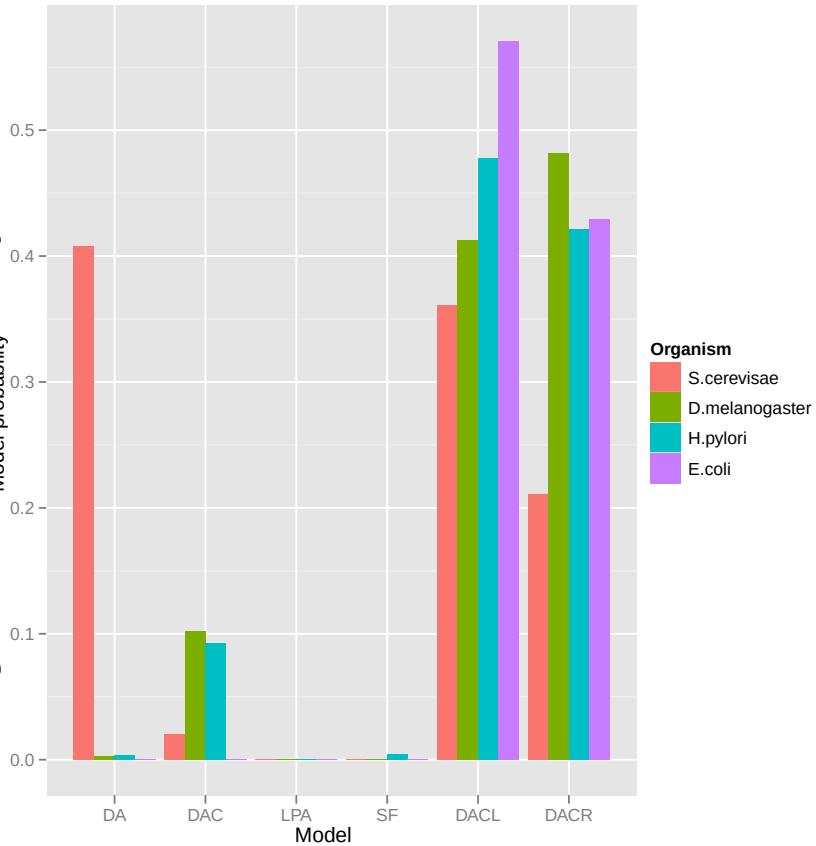
Protein Interaction Network Data

Species	Proteins	Interactions	Genome size	Sampling fraction
S.cerevisiae	5035	22118	6532	0.77
D. melanogaster	7506	22871	14076	0.53
H. pylori	715	1423	1589	0.45
E. coli	1888	7008	5416	0.35



Protein Interaction Network Data

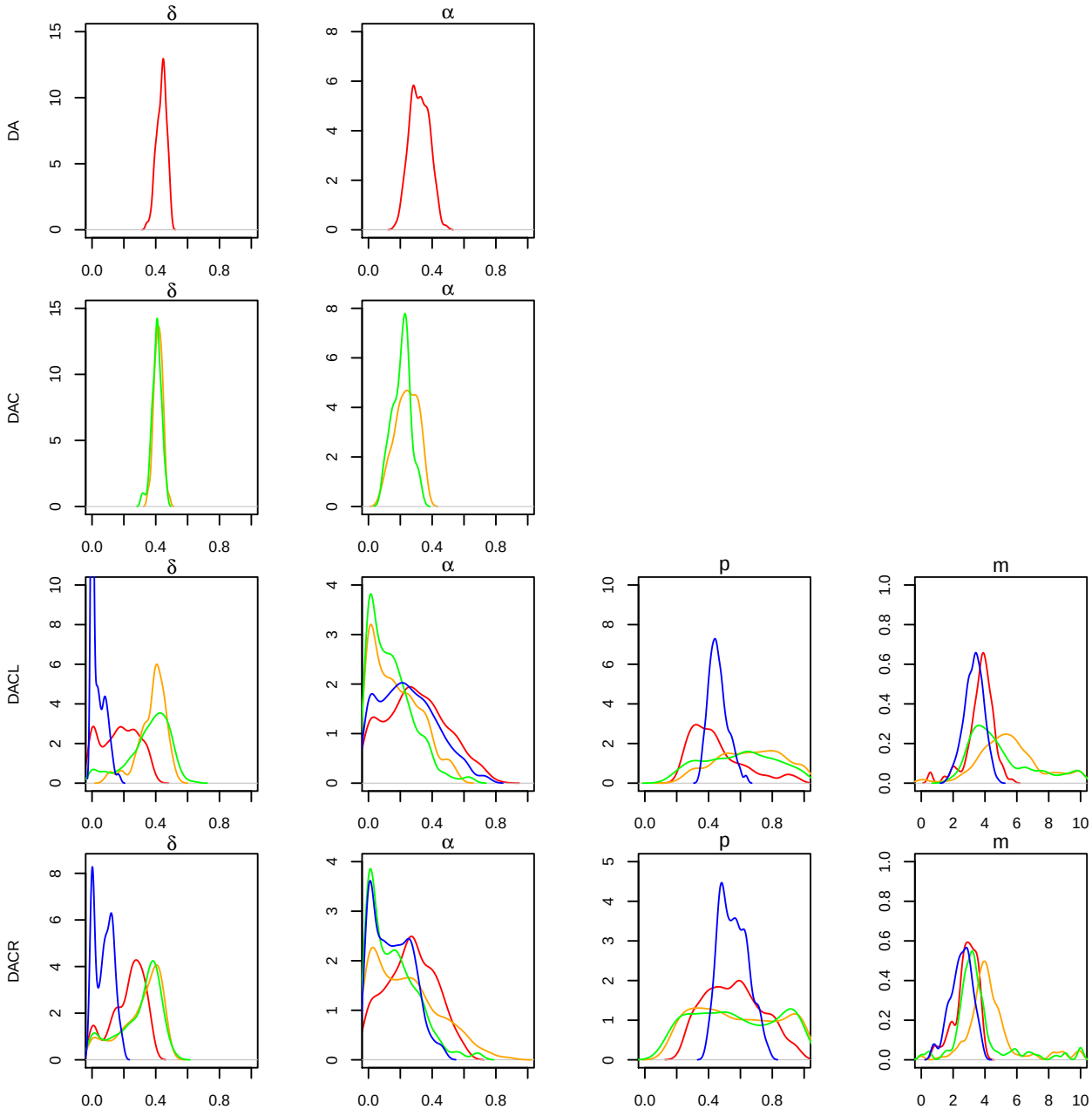
Species	Proteins	Interactions	Genome size	Sampling fraction
S.cerevisiae	5035	22118	6532	0.77
D. melanogaster	7506	22871	14076	0.53
H. pylori	715	1423	1589	0.45
E. coli	1888	7008	5416	0.35



Model Selection

- Inference here was based on all the data, not summary statistics.
- Duplication models receive the strongest support from the data.
- Several models receive support and no model is chosen unambiguously.

Protein Interaction Network Data



S.cerevisiae
D. melanogaster
H. pylori
E. coli

GPUs in Computational Statistics

Performance

- With highly optimized libraries (e.g. BLAST/ATLAS) we can run numerically demanding jobs relatively straightforwardly.

GPUs in Computational Statistics

Performance

- With highly optimized libraries (e.g. BLAST/ATLAS) we can run numerically demanding jobs relatively straightforwardly.
- Whenever poor-man's paralellization is possible, GPUs offer considerable advantages over multi-core CPU systems (at comparable cost and energy requirements).

GPUs in Computational Statistics

Performance

- With highly optimized libraries (e.g. BLAST/ATLAS) we can run numerically demanding jobs relatively straightforwardly.
- Whenever poor-man's paralellization is possible, GPUs offer considerable advantages over multi-core CPU systems (at comparable cost and energy requirements).
- Performance depends crucially on our ability to map tasks onto the hardware.

GPUs in Computational Statistics

Performance

- With highly optimized libraries (e.g. BLAST/ATLAS) we can run numerically demanding jobs relatively straightforwardly.
- Whenever poor-man's paralellization is possible, GPUs offer considerable advantages over multi-core CPU systems (at comparable cost and energy requirements).
- Performance depends crucially on our ability to map tasks onto the hardware.

Challenges

- GPU hardware was initially conceived for different purposes — computer gamers need fewer random numbers than MCMC or SMC procedures. The address space accessible to single-precision numbers also suffices to kill zombies etc .

GPUs in Computational Statistics

Performance

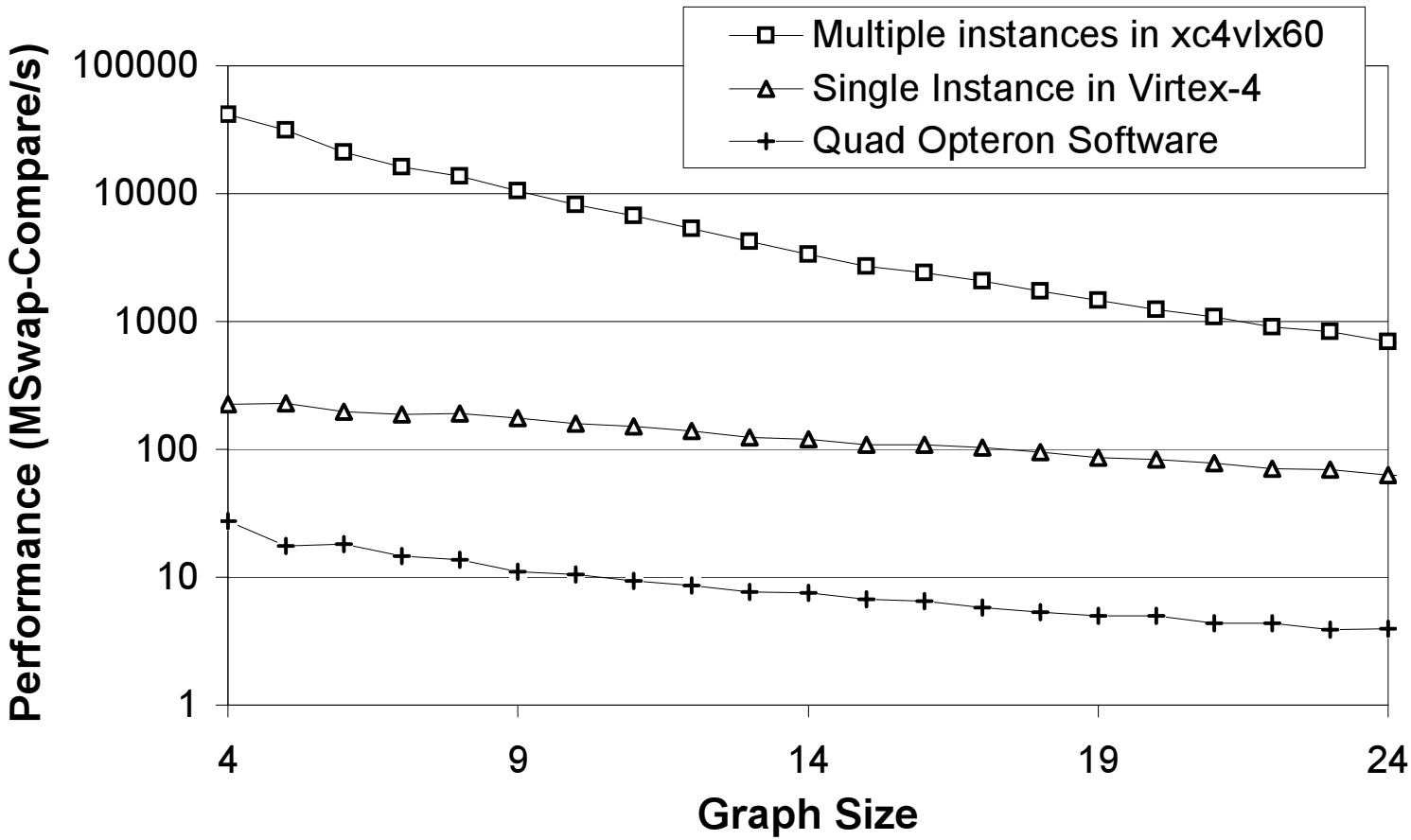
- With highly optimized libraries (e.g. BLAST/ATLAS) we can run numerically demanding jobs relatively straightforwardly.
- Whenever poor-man's paralellization is possible, GPUs offer considerable advantages over multi-core CPU systems (at comparable cost and energy requirements).
- Performance depends crucially on our ability to map tasks onto the hardware.

Challenges

- GPU hardware was initially conceived for different purposes — computer gamers need fewer random numbers than MCMC or SMC procedures. The address space accessible to single-precision numbers also suffices to kill zombies etc .
- Combining several GPUs requires additional work, using e.g. MPI.

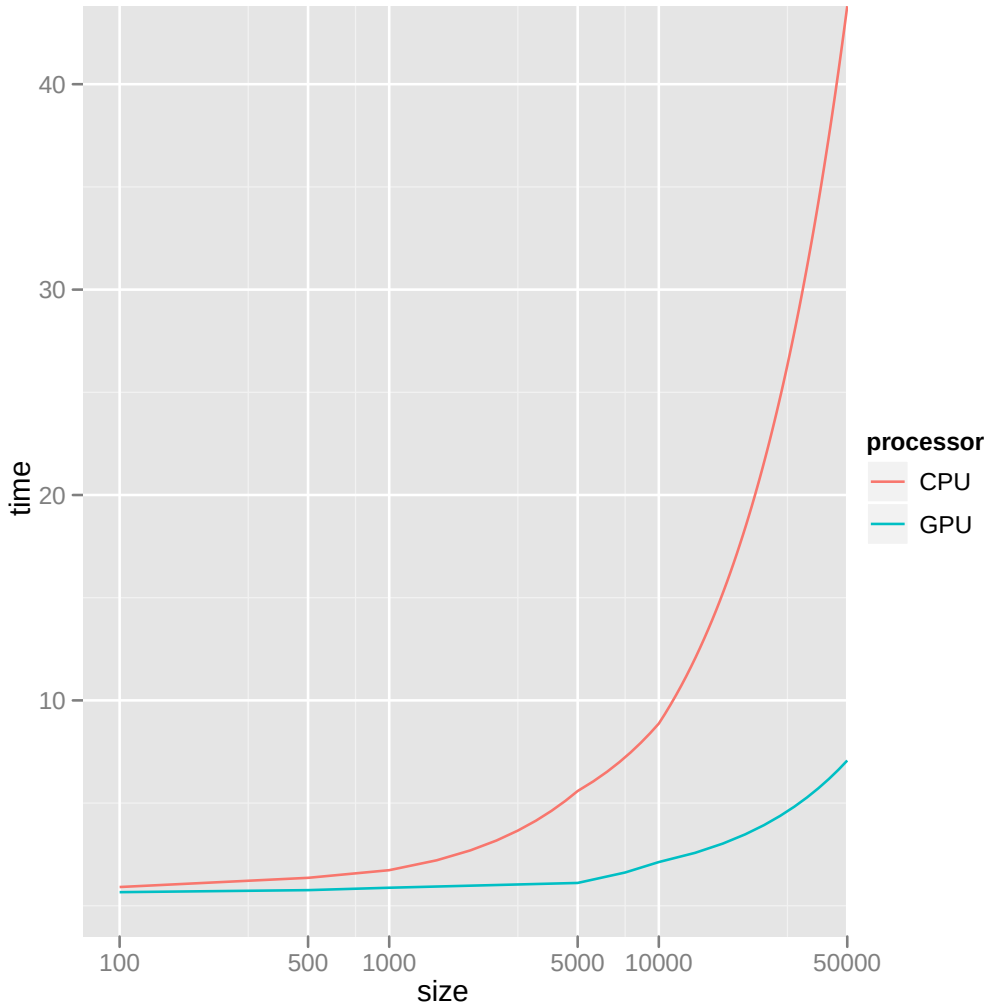
Alternatives to GPUs: FPGAs

Field Programmable Gate Arrays are configurable electronic circuits that are partly (re)configurable. Here the hardware is adapted to the problem at hand and encapsulates the programme in its programmable logic arrays.



Alternatives to GPUs: CPUs (and MPI. . .)

CPUs with multiple cores are flexible, have large address spaces and a wealth of flexible numerical routines. This makes implementation of numerically demanding tasks relatively straightforward. In particular there is less of an incentive to consider how a problem is best implemented in software that takes advantage of hardware features.



6-core Xeon vs M2050 (448 cores) programmed in OpenCL

(Pseudo) Random Numbers

The *Mersenne-Twister* is one of the standard random number generators for simulation. MT19937 has a period of $2^{19937} - 1 \approx 4 \times 10^{6001}$.

(Pseudo) Random Numbers

The *Mersenne-Twister* is one of the standard random number generators for simulation. MT19937 has a period of $2^{19937} - 1 \approx 4 \times 10^{6001}$.

But MT does not have cryptographic strength (once 624 iterates have been observed, all future states are predictable), unlike *Blum-Blum-Shub*,

$$x_{n+1} = x_n \text{mod} M,$$

where $M = pq$ with p, q large prime numbers. But it is too slow for simulations.

(Pseudo) Random Numbers

The *Mersenne-Twister* is one of the standard random number generators for simulation. MT19937 has a period of $2^{19937} - 1 \approx 4 \times 10^{6001}$.

But MT does not have cryptographic strength (once 624 iterates have been observed, all future states are predictable), unlike *Blum-Blum-Shub*,

$$x_{n+1} = x_n \text{mod} M,$$

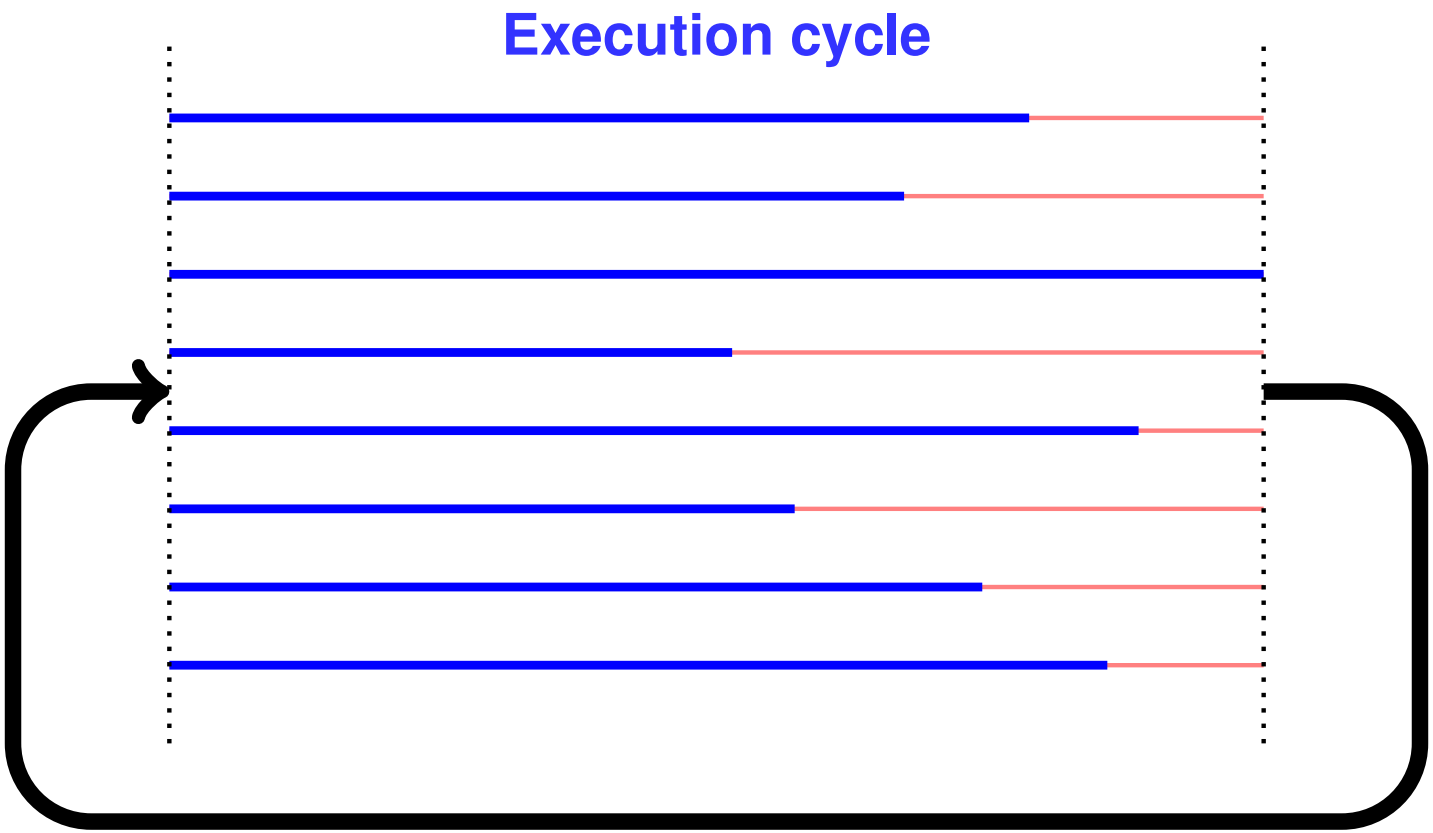
where $M = pq$ with p, q large prime numbers. But it is too slow for simulations.

Parallel Random Number Generation

- Running RNGs in parallel does not produce reliable sets of random numbers.
- We need algorithms produce large numbers of parallel streams of "good" random numbers.
- We also need better algorithms for weighted sampling.

What GPUs are Good At

- GPUs are good for linear threads involving mathematical functions.
- We should avoid loops and branches in the code.
- In an optimal world we should aim for all threads to finish at the same time.



Generating RNGs in Parallel

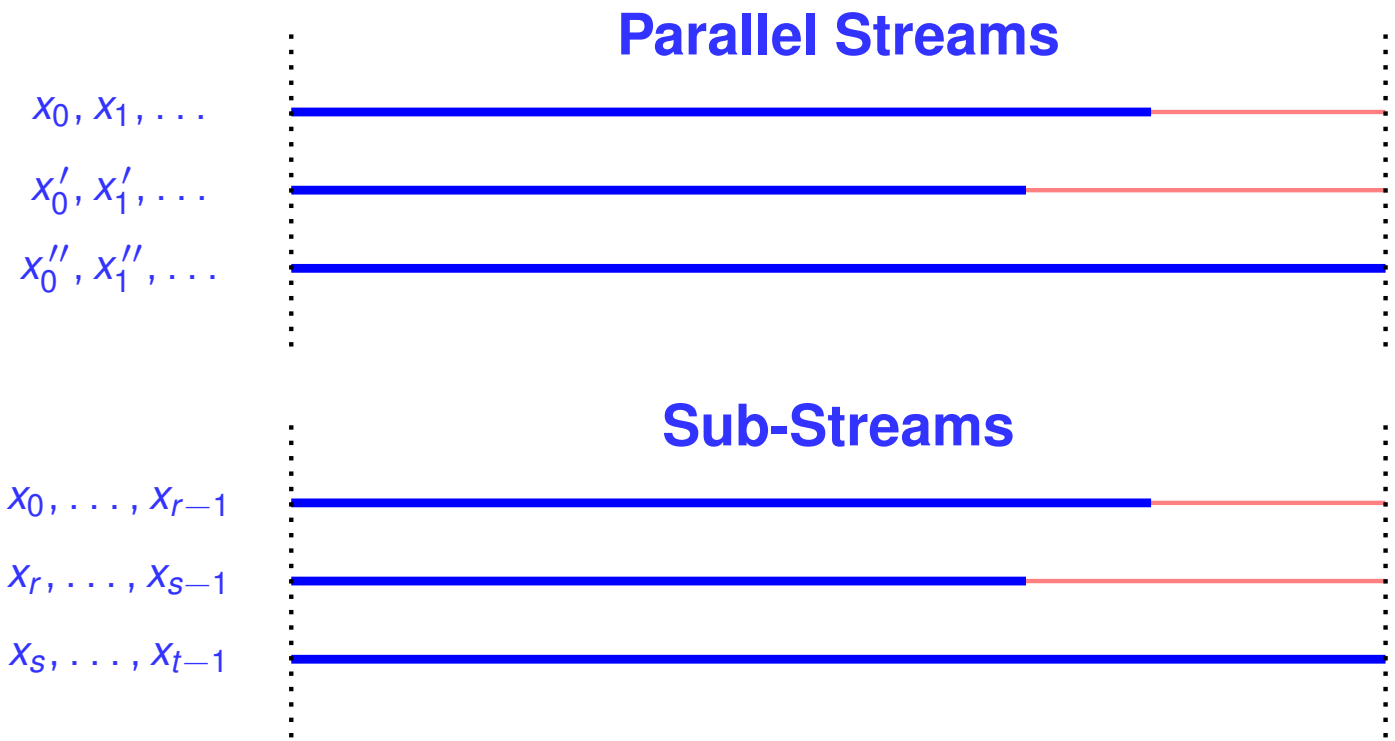
We assume that we have a function $x_{n+1} = f(x_n)$ which generates a stream of (pseudo) random numbers

$$x_0, x_1, x_2, \ldots, x_r, \ldots, x_s, \ldots, x_t, \ldots, x_z$$

Generating RNGs in Parallel

We assume that we have a function $x_{n+1} = f(x_n)$ which generates a stream of (pseudo) random numbers

$$x_0, x_1, x_2, \ldots, x_r, \ldots, x_s, \ldots, x_t, \ldots, x_z$$



Counter-Based RNGs

We can represent RNGs as state-space processes,

$$y_{n+1} = f(y_n) \quad \text{with} \quad f : Y \longrightarrow Y$$
$$x_{n+1} = g_{k,n \bmod J}(y_{\lfloor n+1/J \rfloor}) \quad \text{with} \quad g : Y \times \mathcal{K} \times \mathbb{Z}_J \longrightarrow X,$$

where x are essentially behaving as $x \sim \mathcal{U}_{[0,1]}$; here $\mathcal{K}$ is the key space, J the number of random numbers generated from the internal state of the RNG.

Counter-Based RNGs

We can represent RNGs as state-space processes,

$$y_{n+1} = f(y_n) \quad \text{with} \quad f : Y \longrightarrow Y$$
$$x_{n+1} = g_{k,n \bmod J}(y_{\lfloor n+1/J \rfloor}) \quad \text{with} \quad g : Y \times \mathcal{K} \times \mathbb{Z}_J \longrightarrow X,$$

where x are essentially behaving as $x \sim \mathcal{U}_{[0,1]}$; here $\mathcal{K}$ is the key space, J the number of random numbers generated from the internal state of the RNG.
Salmon et al.(SC11) propose to use a simple form for $f(\cdot)$ and

$$g_{k,j} = h_k \circ b_k$$

Counter-Based RNGs

We can represent RNGs as state-space processes,

$$y_{n+1} = f(y_n) \quad \text{with} \quad f : Y \longrightarrow Y$$
$$x_{n+1} = g_{k,n \bmod J}(y_{\lfloor n+1/J \rfloor}) \quad \text{with} \quad g : Y \times \mathcal{K} \times \mathbb{Z}_J \longrightarrow X,$$

where x are essentially behaving as $x \sim \mathcal{U}_{[0,1]}$; here $\mathcal{K}$ is the key space, J the number of random numbers generated from the internal state of the RNG.
Salmon et al.(SC11) propose to use a simple form for $f(\cdot)$ and

$$g_{k,j} = h_k \circ b_k$$

For a sufficiently complex bijection b_k we can use simple updates and leave the randomization to b_k. Here cryptographic routines come in useful.

RNG Performance on CPUs and GPUs

Method	Max. input	Min. state	Output size	Intel CPU cpB	Intel CPU GB/s	Nvidia GPU cpB	Nvidia GPU GB/s	AMD GPU cpB	AMD GPU GB/s
Counter-based, Cryptographic									
AES(sw)	(1+0)×16	11×16	1×16	31.2	0.4	–	–	–	–
AES(hw)	(1+0)×16	11×16	1×16	**1.7**	**7.2**	–	–	–	–
Threefish (Threefry-4×64-72)	(4+4)×8	0	4×8	7.3	1.7	51.8	15.3	302.8	4.5
Counter-based, Crush-resistant									
ARS-5(hw)	(1+1)×16	0	1×16	0.7	17.8	–	–	–	–
ARS-7(hw)	(1+1)×16	0	1×16	**1.1**	**11.1**	–	–	–	–
Threefry-2×64-13	(2+2)×8	0	2×8	2.0	6.3	13.6	58.1	25.6	52.5
Threefry-2×64-20	(2+2)×8	0	2×8	2.4	5.1	15.3	51.7	30.4	44.5
Threefry-4×64-12	(4+4)×8	0	4×8	1.1	11.2	9.4	84.1	15.2	90.0
Threefry-4×64-20	(4+4)×8	0	4×8	**1.9**	**6.4**	15.0	52.8	29.2	46.4
Threefry-4×32-12	(4+4)×4	0	4×4	2.2	5.6	9.5	83.0	12.8	106.2
Threefry-4×32-20	(4+4)×4	0	4×4	3.9	3.1	15.7	50.4	25.2	53.8
Philox2×64-6	(2+1)×8	0	2×8	2.1	5.9	8.8	90.0	37.2	36.4
Philox2×64-10	(2+1)×8	0	2×8	4.3	2.8	14.7	53.7	62.8	21.6
Philox4×64-7	(4+2)×8	0	4×8	2.0	6.0	8.6	92.4	36.4	37.2
Philox4×64-10	(4+2)×8	0	4×8	3.2	3.9	12.9	61.5	54.0	25.1
Philox4×32-7	(4+2)×4	0	4×4	2.4	5.0	3.9	201.6	12.0	113.1
Philox4×32-10	(4+2)×4	0	4×4	3.6	3.4	**5.4**	**145.3**	**17.2**	**79.1**
Conventional, Crush-resistant									
MRG32k3a	0	6×4	1000×4	3.8	3.2	–	–	–	–
MRG32k3a	0	6×4	4×4	20.3	0.6	–	–	–	–
MRGk5-93	0	5×4	1×4	7.6	1.6	9.2	85.5	–	–
Conventional, Crushable									
Mersenne Twister	0	312×8	1×8	2.0	6.1	43.3	18.3	–	–
XORWOW	0	6×4	1×4	1.6	7.7	5.8	136.7	16.8	81.1

Taken from Salmon et al.(see References).

Computational Challenges of GPUs

- Address spaces are a potential issue: using single precision we are prone to run out of numbers for challenging MCMC/SMC applications very quickly.

Computational Challenges of GPUs

- Address spaces are a potential issue: using single precision we are prone to run out of numbers for challenging MCMC/SMC applications very quickly.
- Memory is an issue. In particular, registers/memory close to the computing cores are precious.

Computational Challenges of GPUs

- Address spaces are a potential issue: using single precision we are prone to run out of numbers for challenging MCMC/SMC applications very quickly.
- Memory is an issue. In particular, registers/memory close to the computing cores are precious.
- Bandwidth is an issue — coordination between GPU and CPU is much more challenging in statistical applications than e.g. for texture mapping of graphics rendering tasks.

Computational Challenges of GPUs

- Address spaces are a potential issue: using single precision we are prone to run out of numbers for challenging MCMC/SMC applications very quickly.
- Memory is an issue. In particular, registers/memory close to the computing cores are precious.
- Bandwidth is an issue — coordination between GPU and CPU is much more challenging in statistical applications than e.g. for texture mapping of graphics rendering tasks.
- Programming has to be much more hardware aware than for CPUs — more precisely, non-hardware adapted programming will be more obviously less efficient than for CPUs.

Computational Challenges of GPUs

- Address spaces are a potential issue: using single precision we are prone to run out of numbers for challenging MCMC/SMC applications very quickly.
- Memory is an issue. In particular, registers/memory close to the computing cores are precious.
- Bandwidth is an issue — coordination between GPU and CPU is much more challenging in statistical applications than e.g. for texture mapping of graphics rendering tasks.
- Programming has to be much more hardware aware than for CPUs — more precisely, non-hardware adapted programming will be more obviously less efficient than for CPUs.
- There are some differences from conventional ANSI standards for mathematics (e.g. rounding).

References

GPUs and ABC

- Zhou, Liepe, Sheng, Stumpf, Barnes (2011) *GPU accelerated biochemical network simulation.* Bioinformatics **27**:874-876.
- Liepe, Barnes, Cule, Erguler, Kirk, Toni, Stumpf (2010) *ABC-SysBioapproximate Bayesian computation in Python with GPU support.* Bioinformatics **26**:1797-1799.

GPUs and Random Numbers

- Salmon, Moraes, Dror, Shaw (2011) *Parallel Random Numbers: As Easy as 1, 2, 3.* in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ACM, New York; `http://doi.acm.org/10.1145/2063384.2063405`.
- Howes, Thomas (2007) *Efficient Random Number Generation and Application Using CUDA* in *GPU Gems 3*, Addison-Wesley Professional, Boston; `http://http.developer.nvidia.com/GPUGems3/gpugems3_ch37.html`.

Acknowledgements

bbsrc biotechnology and biological sciences research council
MRC Medical Research Council
EPSRC Engineering and Physical Sciences Research Council
HFSP
The Wellcome Trust
The Leverhulme Trust
EMBO
THE ROYAL SOCIETY