Advanced Scientific Computing Workshop

System aspects

Christian Herzog
Head of IT, D-PHYS
herzog@phys.ethz.ch

https://indico.phys.ethz.ch/conferenceDisplay.py?confId=5

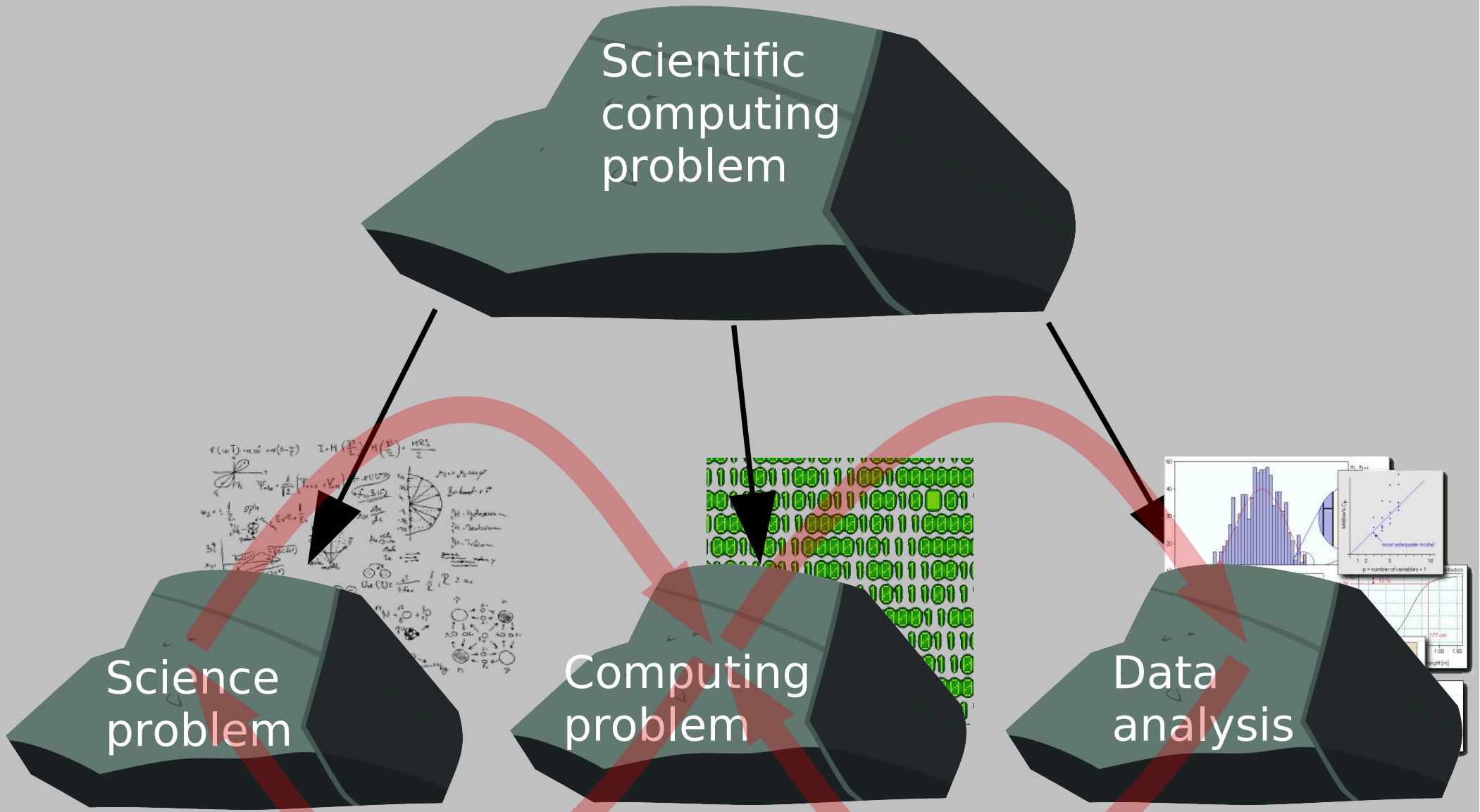
Contents

- Lingo
- Why are we here? What IS scientific computing?
- Scientific computing problems – why are they hard?
- 1st aspect: hardware
- 2nd aspect: type of SC problem
- Hardware architectures
- Implications for your code
- Some words about storage
- Limits revisited
- Programming languages
- Optimizations
- Tips, tricks and anecdotes
- Questions / discussion

Glossary

CPU:	Central Processing Unit → processor, as in 'hardware'
Core:	individual execution unit within a CPU
Thread:	sequence of instructions that may execute in parallel with others, within one process
Cache:	small + fast memory in or close to CPU
Interconnect:	data bus between two units
Throughput:	amount of data transferred per time unit
Latency:	packet delay time
GPGPU:	General-purpose graphics processing units
NUMA:	non-uniform memory access
IPC:	inter-process communication

What is Scientific Computing?



often “easier”

you'll probably spend most of your time here

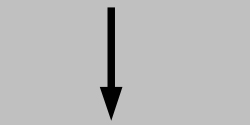
.. and here

The computing problem

Computing problem

easy

you're very lucky



you can still mess up, so don't leave just yet

you missed something

we don't do easy!

hard

implementation / algorithm

software engineering

rest of the workshop

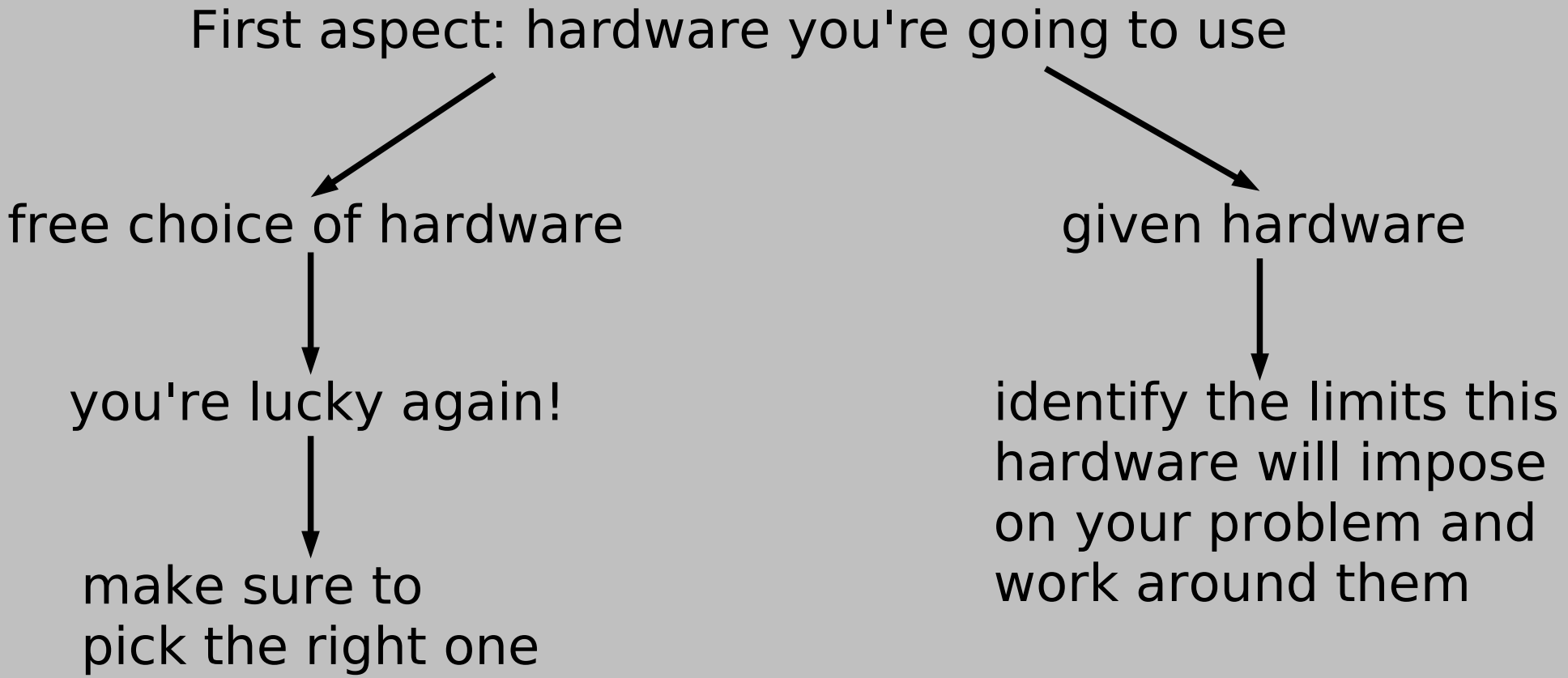
computational limits

- CPU
- Memory
- Interconnect
- Storage

system aspects: our focus

1st aspect: hardware

One of your first tasks when addressing a computing problem is to determine the limiting factors you're going to encounter.



2nd aspect: type of SC problem

Does an easy implementation of your scientific computing problem fit on one machine in terms of CPU and memory?

yes

boy you're so lucky!

node

optimize

limit: CPU or memory

no

can you partition the problem into N independent small parts?

yes

use multithreading or parameter sweeping

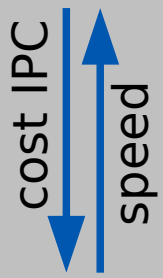
node
NUMA
cluster
cloud

GPGPU

limit: CPU or memory

no

expect heavy IPC/ITC costs (interconnect)

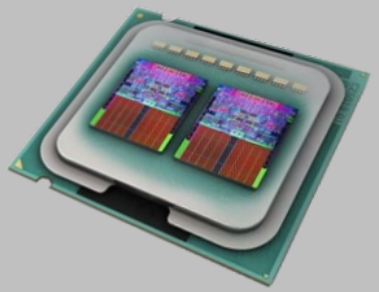
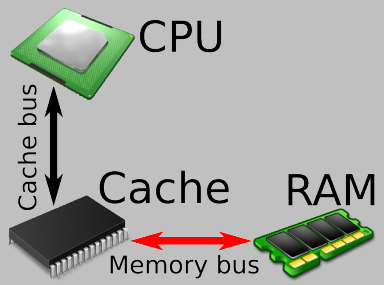
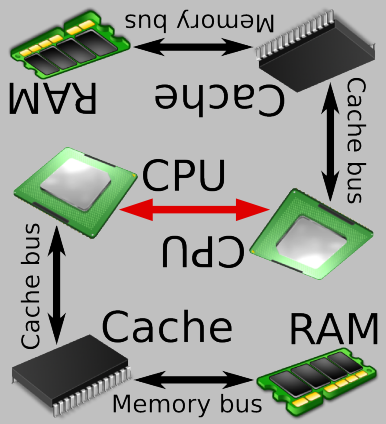
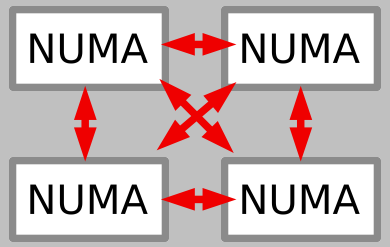
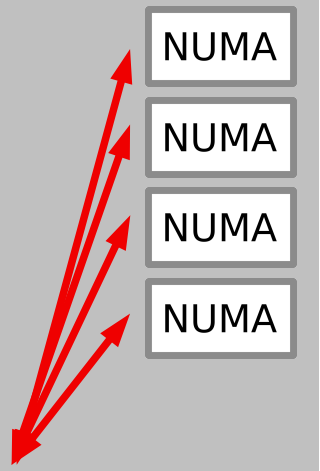


node
NUMA
cluster
cloud

GPGPU

limit: usually mem, partitioning → IPC

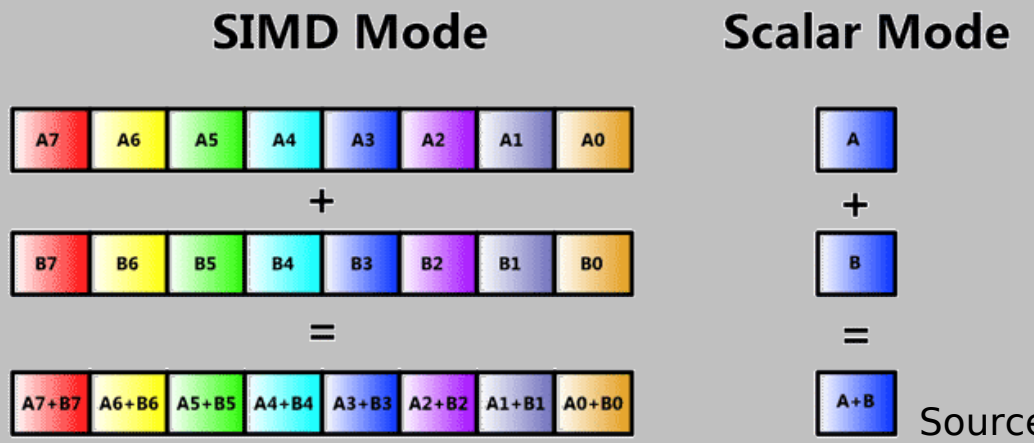
Architectures

CPU	node	NUMA	cluster	cloud
				
M=2..12	M=2..12 * N=1	M=2..12 * N=2..16	M=2..12 * N=2..16 * O=2..10^4	M=2..12 * N=2..16 * O=2..10^3
Limit				
Cache bus	Memory bus	QuickPath, HyperTransp.	Myrinet, Infiniband,	10GBase
100s Gb/s ns latency	>30 Gb/s 10s ns lat.	~20 Gb/s 100s ns lat.	>1 Gb/s µs lat.	1 Gb/s > 10 µs lat.
multithread	multithread	multithread	multithread multiprocess MPI, TIPC...	multithread multiprocess REST, SOAP..

M cores, N CPUs, O nodes

Implications and specialities

- Everything > your laptop is NUMA, so you'll have to use multithreading to get any reasonable performance out of recent (and decent) machines
- You'll have to find the right level of granularity (parallelism vs. IPC costs)
- Additionally, almost all architectures can be enhanced by GPGPU cards (CUDA, OpenCL)
- Another potential 'cheap' performance boost: SIMD (SSE5/AVX)

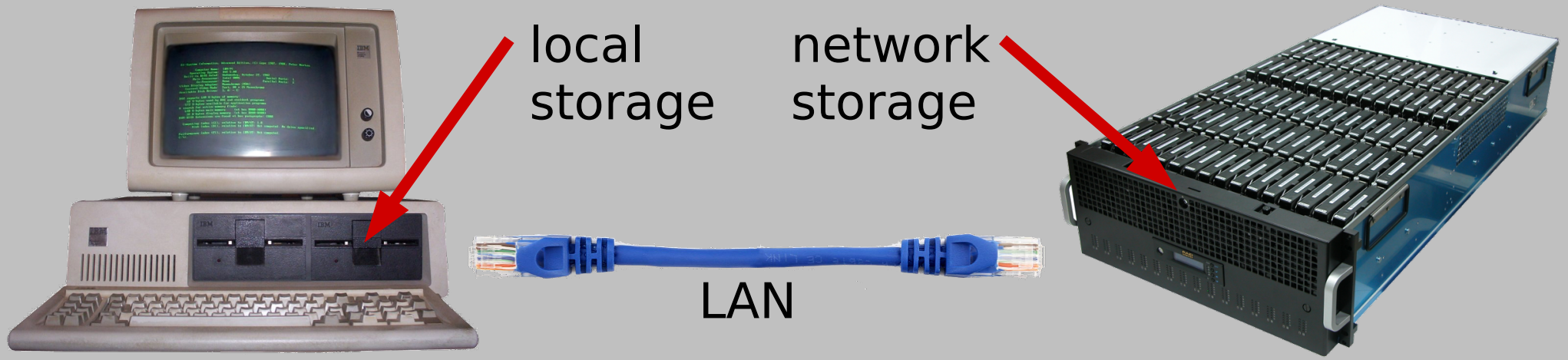


Source: Intel

NVIDIA Kepler

Storage

- Two types of storage:
  - Local storage: per node, temporary space (“scratch”), limited, used for intermediate data dumping and processing
  - Network storage: cross-node, usually much bigger, NFS or cluster file system, used for data processing and archival
- Storage usually is less of a limit, separate from CPU, memory and interconnect
- “But Christian, what about LHC??” - yes, the amount of data was a problem there, but also for CPU and RAM, not only storage
- We'll look at some storage-related optimizations later



Limits revisited

What have we learned so far?

do you have independent or well-partitioned problems with little overlap?

yes

does the problem (code + data) fit into the L1/L2 cache?

yes

probably CPU bound

no

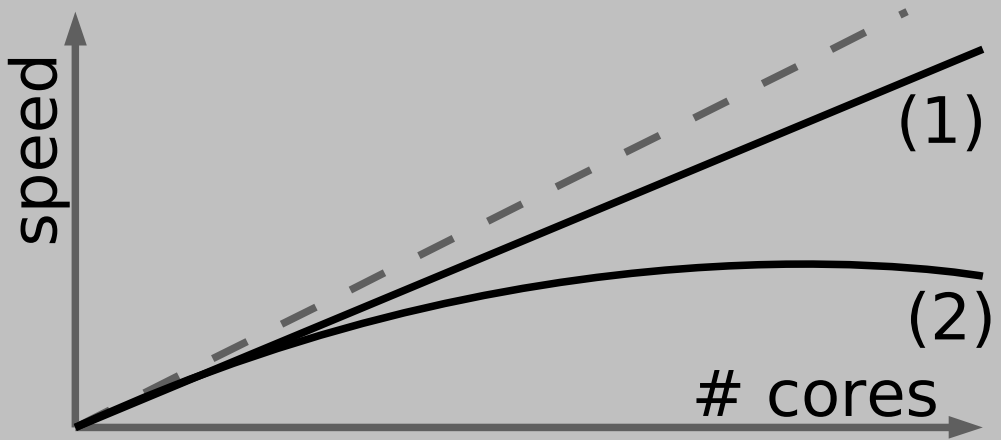
probably memory bound

(1)

no

strong interaction → high IPC costs

limit will probably be the interconnect (2)

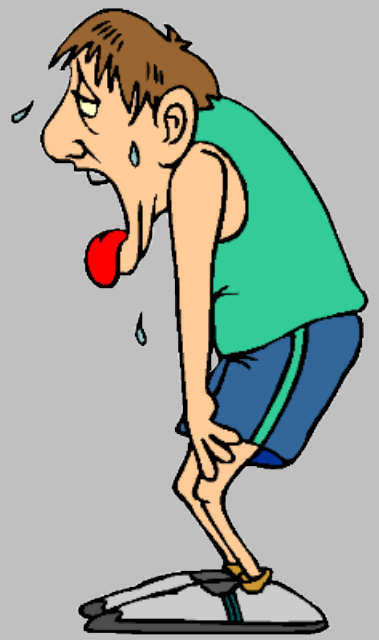


Time to take a deep breath

- Hardware bottlenecks ✔
- SC problem types ✔
- Architectures ✔

Likely limits of your problem ✔

↓ Let's go!



But....

- which programming language should I use?
- how do I get my code to run reallllly fast?

↓

Optimization

Programming languages for SC

- What may you use?
  - not everything might be available on the target machine
  - there might be libraries or existing code you want or have to use
  - you might need to collaborate with others
- What should you use?
  - which language would be the fastest in terms of
    - development time
    - execution speed
- What can you use?
  - Which language(s) are you familiar with?
  - How fast can you learn (and master) another one?

Interpreter language	vs	Compiler language
eg. Perl, Python, PHP		eg. C/C++, Java, Fortran
Code → interpreter → memory		Code → compiler → disk
“fast coding, slow code”		“slow coding, fast code”

Optimization

Software
- Pick the right compiler (gcc, llvm, Intel, PGI..)
- Use optimized libraries (BLAS, LAPACK, BOOST, NumPy..)
- Use the right data types (bit fields, SP/DP..)
- Memory / cache alignment
- Call C / Fortran libraries from scripting languages
- Use a profiler
- Meshing: adaptive, geometry...

Hardware
- Hardware / OS tuning
- Thread pinning

Storage
- File formats (HDF, XML, binary)...
- Mem:disk 100:1

Tips, hints and anecdotes

- Use checkpointing to prevent data loss in case of a crash
- Random numbers – important esp. for Monte Carlo
- Use job queuing systems if available
- Never ever swap, never (mem:disk 100:1)
- Make sure you're using all cores
- Local v. network storage: Don't write 100k files every 2 min
- If you use network storage for long-running jobs, be aware of network interruptions
- Bullshit bingo: MapReduce – just a fancy name for our 'many independent jobs' problem
- On ISG's behalf: please don't move / rename a 4T folder without telling us...

?