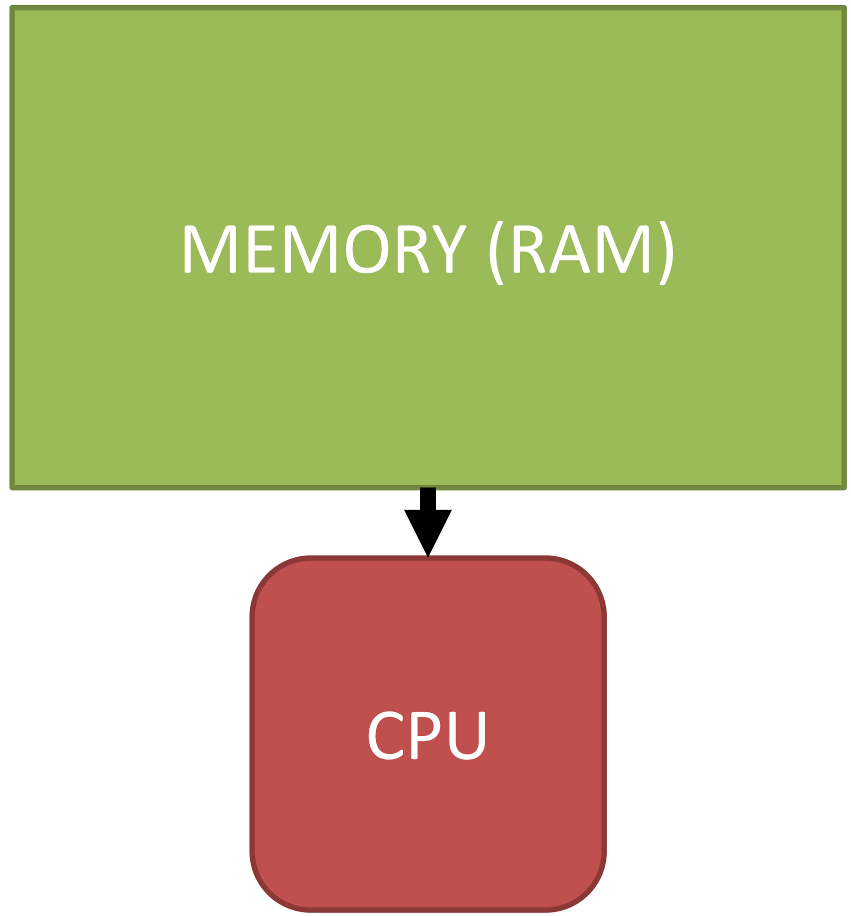
University of St Andrews | FOUNDED 1413

Computers, supercomputers and how to use them

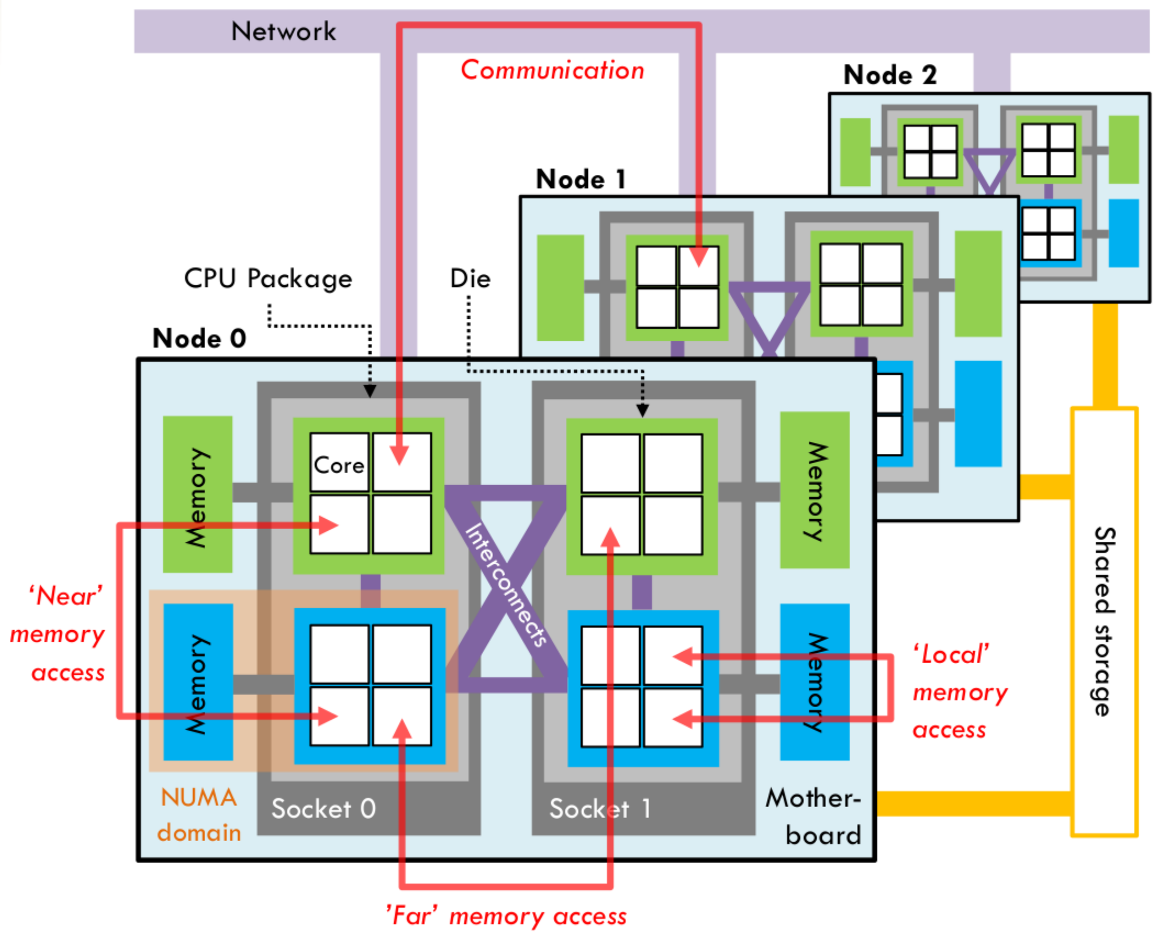
Bert Vandenbroucke

bv7@st-andrews.ac.uk

An old computer



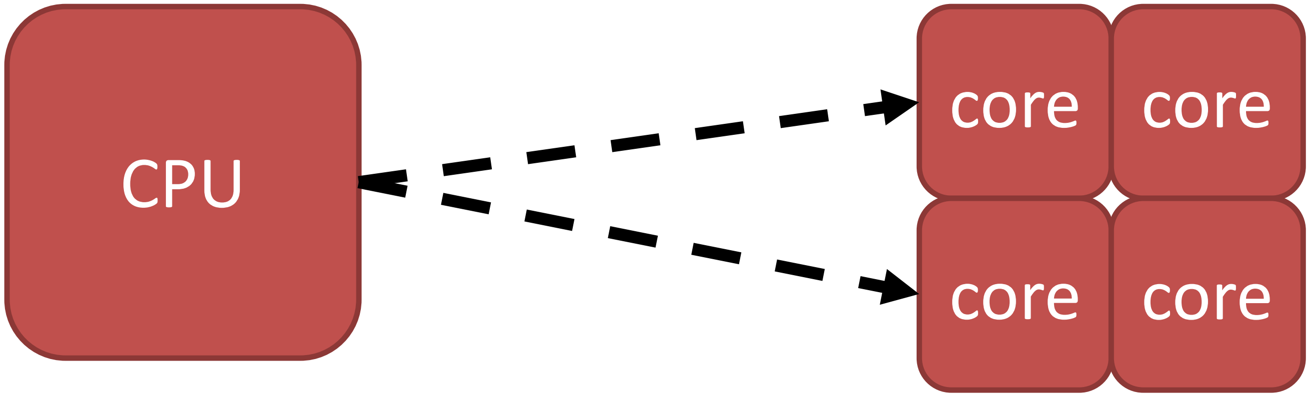
A new computer



Verstocken et al., 2017

Wait what?

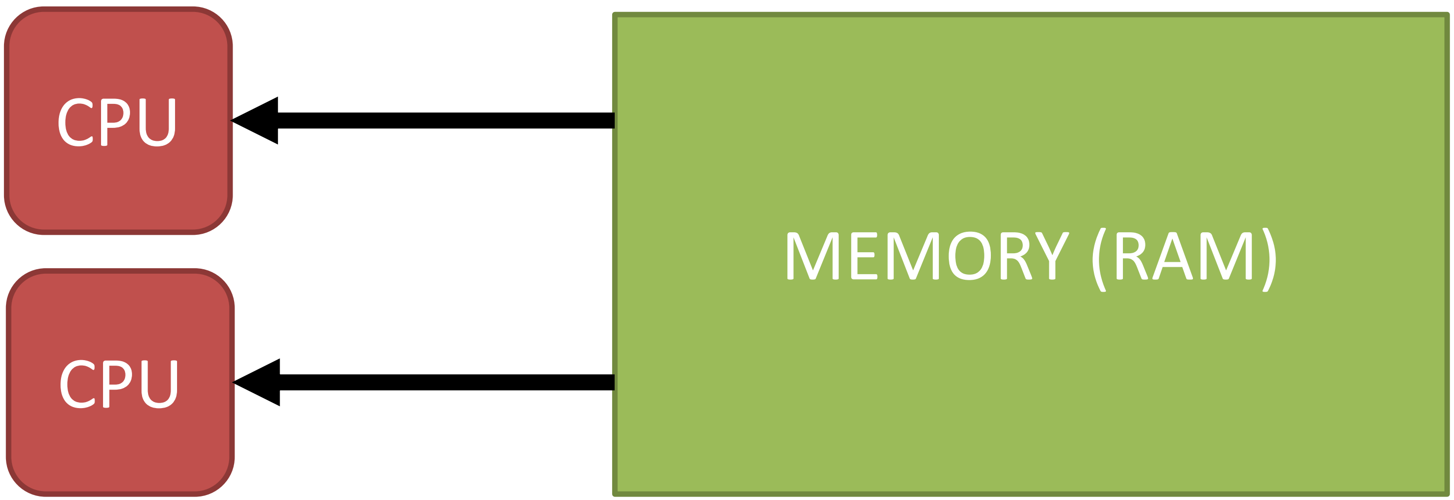
Multicore CPUs



Multiple individual computing units that share the same memory connection

Wait what?

MultiCPU machines



Multiple individual CPUs that share the same memory (but with a different memory connection)

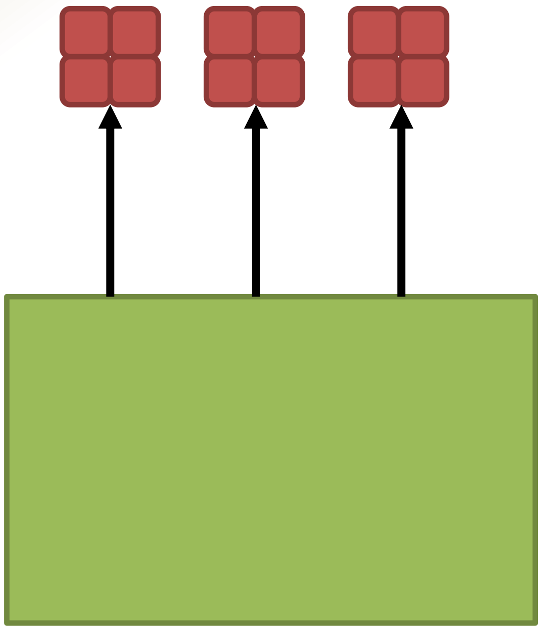
Wait what?

Multinode machines

Multiple individual computers linked together using a fast communication network

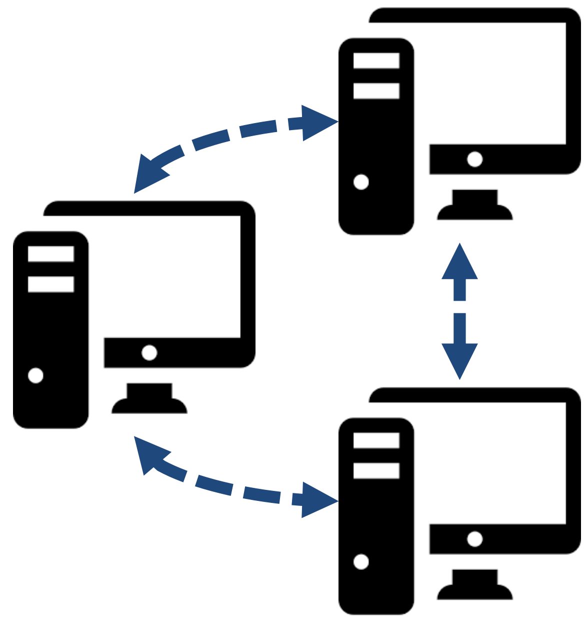
Parallelization

Shared memory



Locking

Distributed memory



Communication

Parallelization

Shared memory

Single program that executes some tasks using multiple cores/CPUs

We need to specify which tasks/instructions can be done in parallel

We need to make sure memory is accessed in a *thread-safe* way

Distributed memory

Multiple program *instances* that can communicate with each other

We need to make sure each instance only does part of the work

We need to add instructions to communicate relevant data from one instance to another

Parallelization

Shared memory

Usually easy to implement

Various standards:

- OpenMP
- PThreads
- C++ threads (since C++11)
- Intel TBB
- ...

Distributed memory

Always hard to do

Only one standard: MPI

Various implementations of MPI:

- OpenMPI (≠OpenMP!!)
- MPICH
- IntelMPI
- ...

Parallelization

Shared memory

Only suitable for shared memory systems

= small systems

= limited by amount of available memory

Distributed memory

Works on both types of systems

Only possibility when running on large systems

Limited by amount of communication, speed of network...

Parallelization

Shared memory

OpenMP example

```
#pragma omp parallel for
for(int i = 0; i < 100; ++i){
  a[i] = b[i] + c[i];
}
```

```
!$OMP PARALLEL DO
do j=1,100
  a(j) = b(j) + c(j)
end do
!$OMP END PARALLEL DO
```

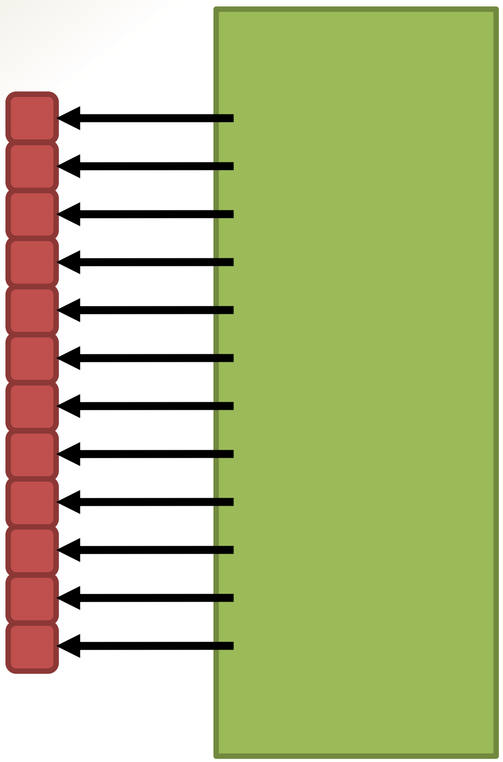
Distributed memory

MPI example

```
MPI_Allgather(a, 25, MPI_DOUBLE, a, 25,
   MPI_DOUBLE, MPI_COMM_WORLD)
```

```
CALL MPI_ALLGATHER(a, 25
   MPI_DOUBLE, a, 25, MPI_DOUBLE,
   MPI_COMM_WORLD, ierr)
```

GPUs



Individual computing unit with
- small amount of memory
- huge amount of special cores

Cores work using single-instruction-multiple-data (SIMD) paradigm: all cores HAVE TO execute the same operation

Huge speedup IF your algorithm can be rewritten using SIMD

Copying data from and to GPU memory usually very slow

GPUs

No standard yet (OpenACC part of OpenMP4?)

Generally three approaches:

- CUDA (NVIDIA)
- OpenCL (Apple)
- OpenGL (free, limited support)

GPU clusters still need MPI to do communication

Vectorization

Modern CPU cores have support for SIMD instructions

In principle, the compiler should automatically identify eligible code and optimize

Unfortunately, compilers are very bad at this

There are special instructions (AVX) to manually improve vectorization

Terminology

Thread: shared memory parallel unit

Process: distributed memory parallel unit

Core: smallest individual computing unit

Node: largest computing unit with single memory

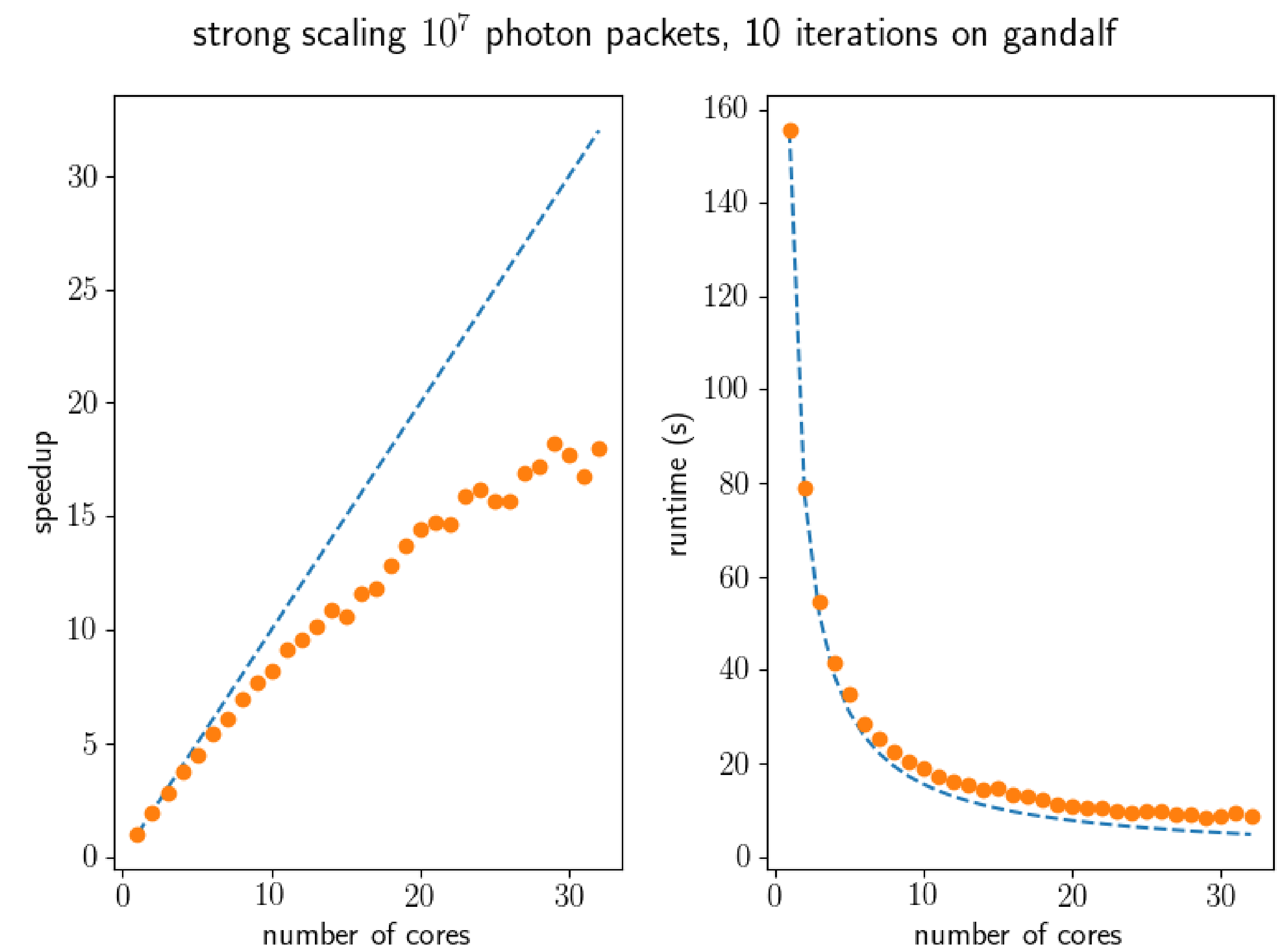
CPU: not generally used as computing term

Terminology

Strong scaling: how much faster your code runs when using more cores for the same problem size
(ideally: cores x2 = time / 2)

Weak scaling: how much slower your code runs when using more cores for the same load per core
(ideally: cores x2 + problem x2 = same time)

Examples



BV, in prep. (future Code & Cake talk?)

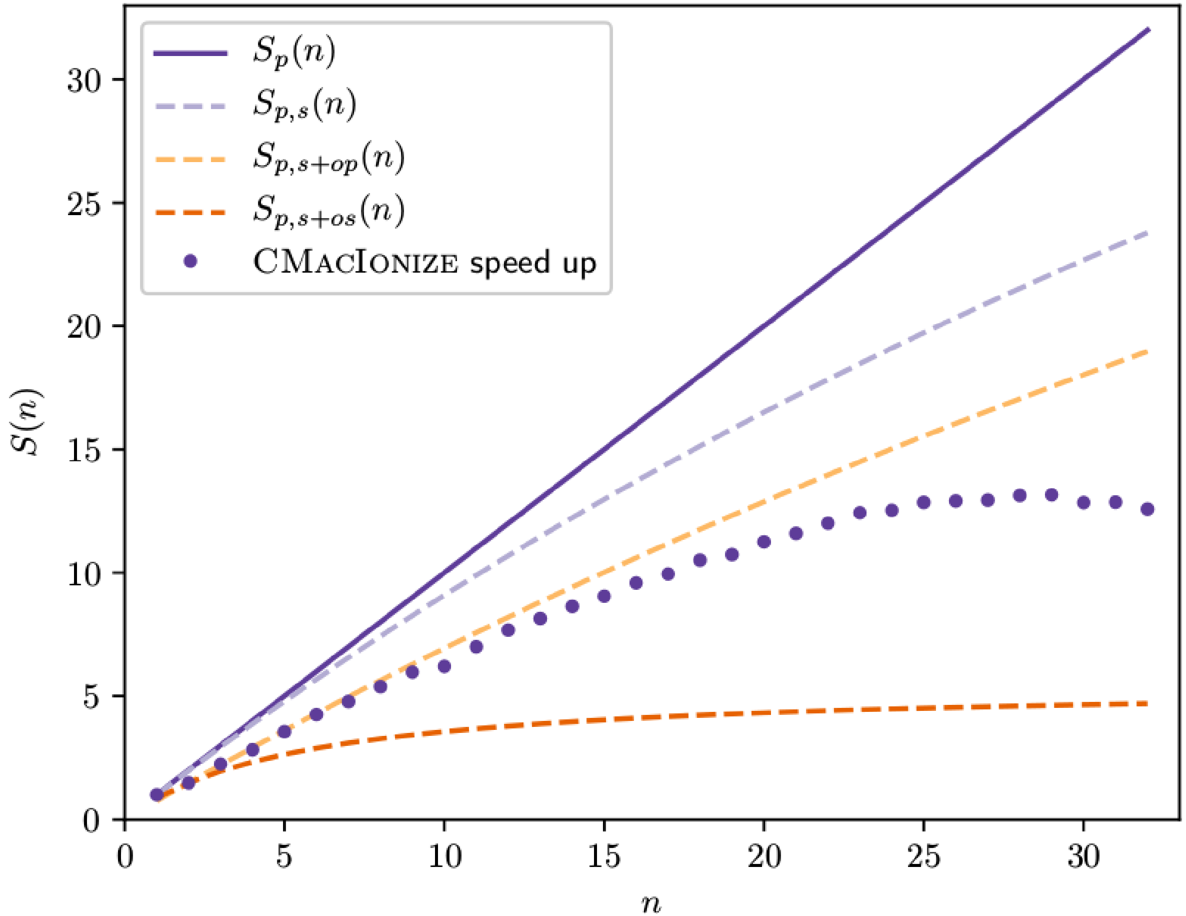
Terminology

Serial fraction: fraction of algorithm that cannot be done in parallel

Overhead: extra work that needs to be done to run in parallel

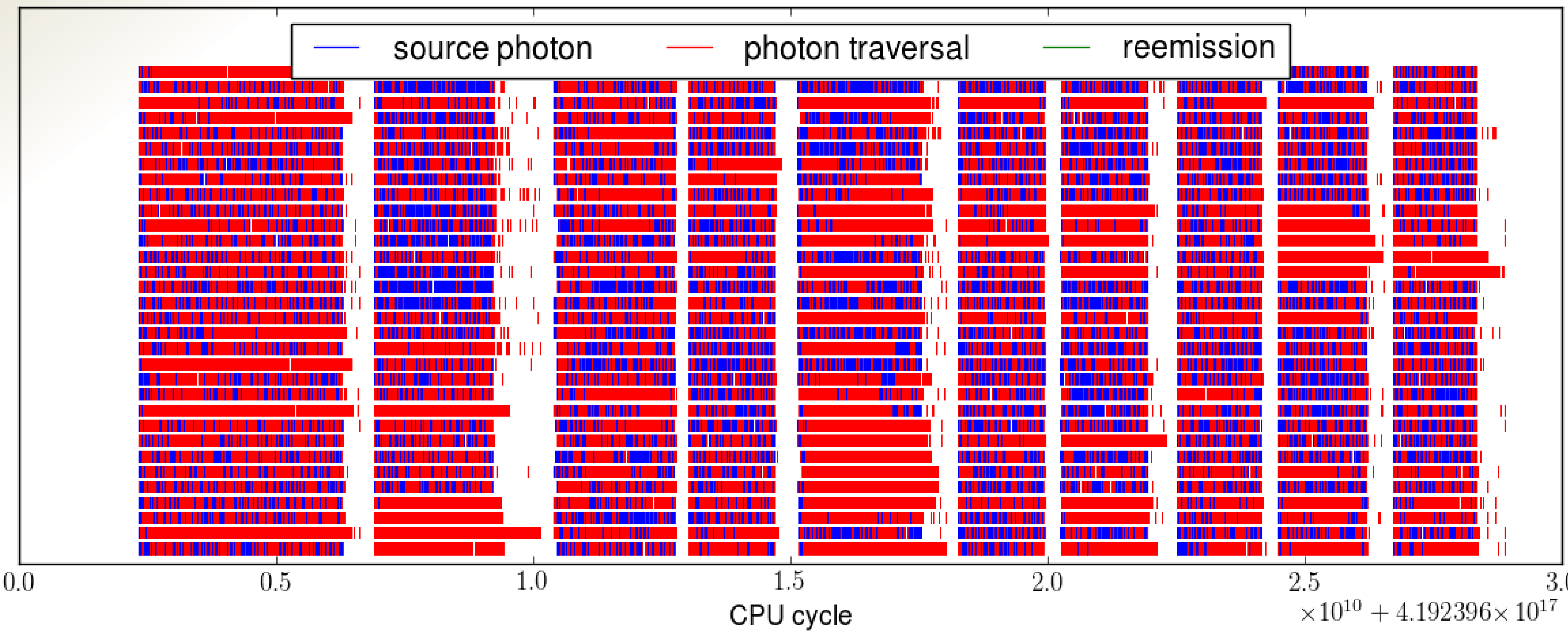
Load imbalance: overhead due to cores/nodes waiting for other cores/nodes to finish

Example



BV & Wood, 2018

Example



BV, in prep. (future Code & Cake talk?)

Terminology

Computation bound: algorithm is limited by how fast computations can be executed

Memory bound: algorithm is limited by how fast data flows from memory to core

Communication bound: (parallel) algorithm is limited by how fast data is communicated

General remarks

- OpenMP is a very easy way to get a reasonable speed up for programs that run on your computer/a single node remote machine
- MPI can be more efficient than OpenMP on a shared memory system (depends on memory layout, very hard to predict)
- Hybrid algorithms use a combination of OpenMP + MPI

General remarks

- Hyperthreading: OS runs 2 threads on a single core: sometimes more efficient (depends on algorithm)
- Clock speed: speed of cores can depend on how many cores are being used

What can go wrong?

Shared memory parallelization:

- race condition: multiple threads writing to the same memory block at the same time
- deadlock: thread locks a variable and does not unlock it

Distributed memory parallelization:

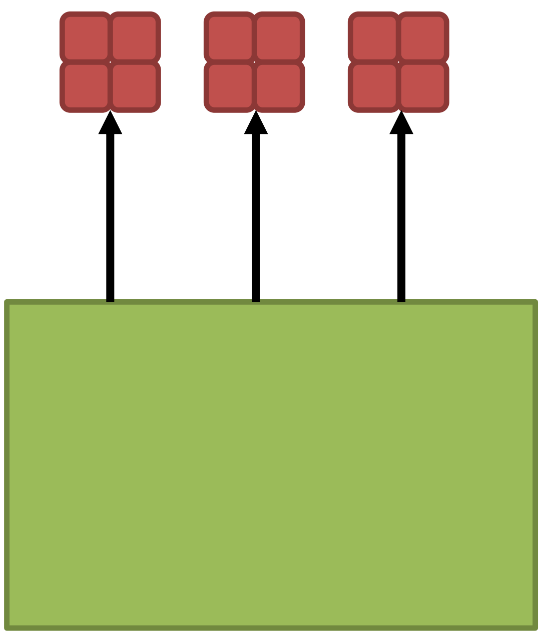
- deadlock: process sends a message that is not received or received in the wrong order

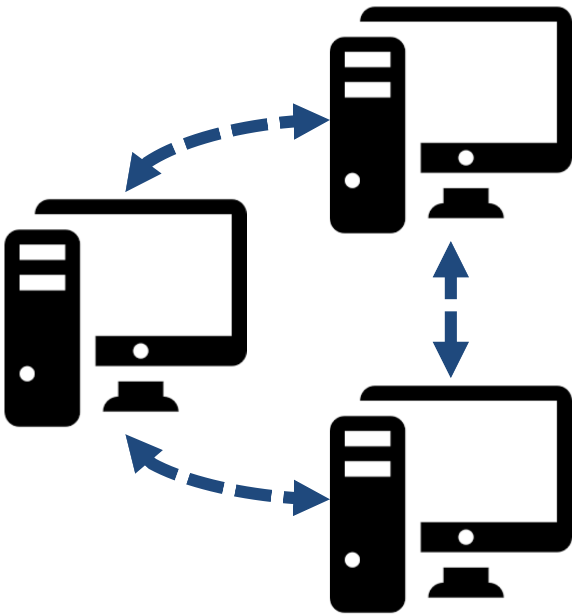
How can I debug/profile?

- Proprietary (licensed) software: Intel VTune, AMD CodeAnalyst, Apple Inc. Shark...
- No general open-source/free alternatives
- GCC/LLVM compiler suites contain some tools, like e.g. a thread-sanitizer
- Run serial debugger/profiler in parallel

Conclusion

- Modern computers are always parallel in some way
- Supercomputers are definitely highly parallel





Conclusion

- Shared memory parallelization allows you to use multiple computing units (cores) on the same memory
- Distributed memory parallelization allows you to use multiple computing units that have separate memories and are connected through some type of network (nodes)

Conclusion

- Which type of parallelism you want to use depends on
  - the system (single node/multinode?)
  - the algorithm
  - the problem size (does it fit in single node memory?)
  - how much time you have to implement it

Also...

Every OS supports running multiple programs in parallel (very efficiently)

So if you can split your problem into many small problems, that is almost guaranteed to be the most efficient strategy

Workflow Management Systems (see previous talk) can help you run many jobs in parallel