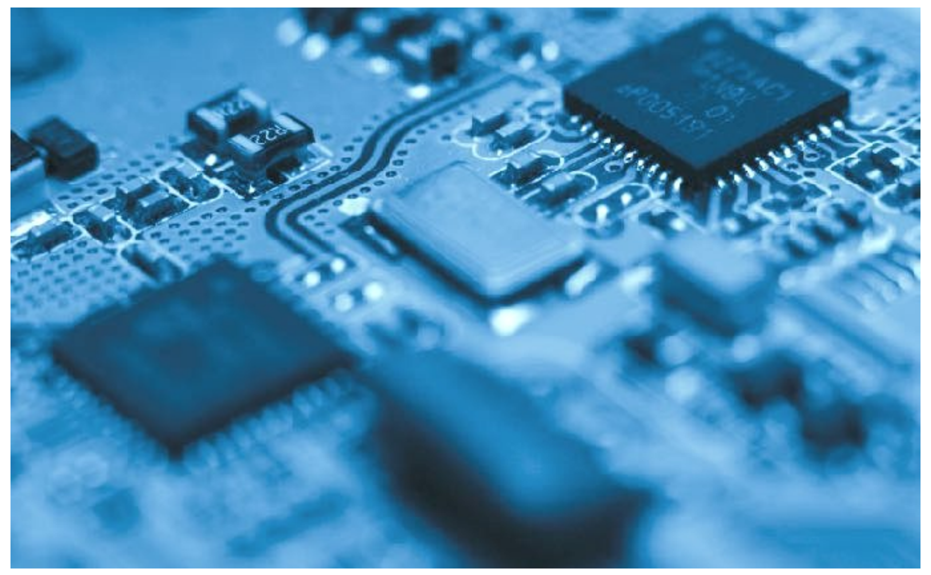
Funded by the European Union

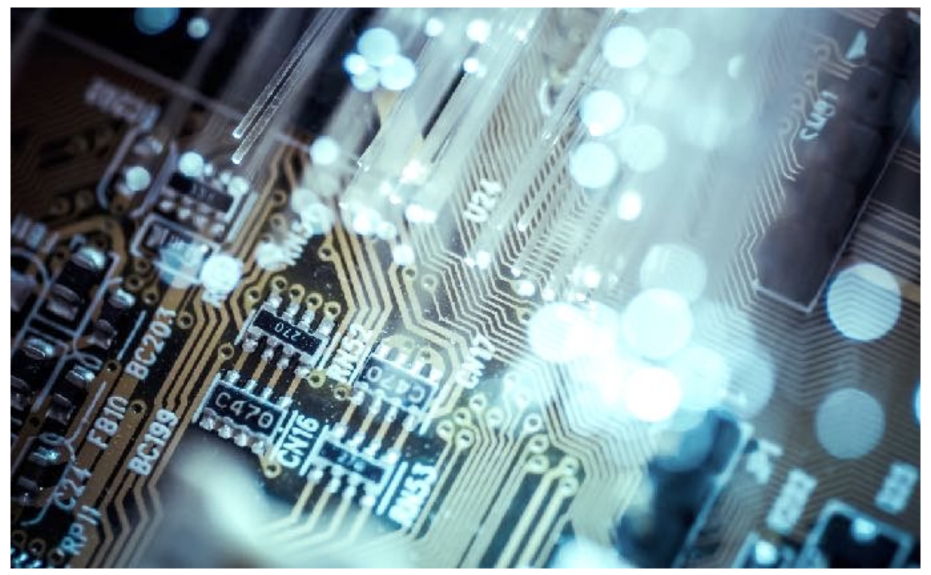
Co-ordinated by ECMWF

ESCAPE 2

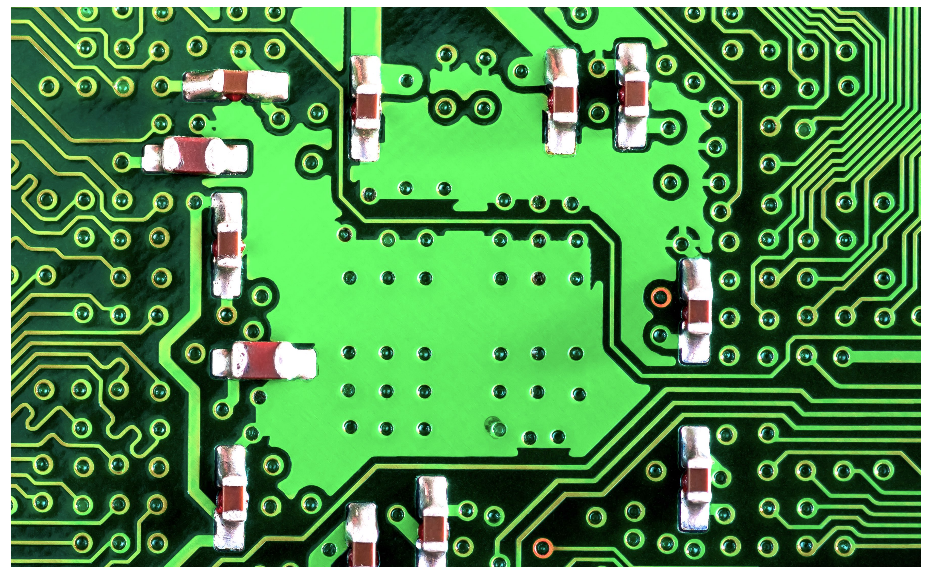
Funded by the European Union

Co-ordinated by ECMWF

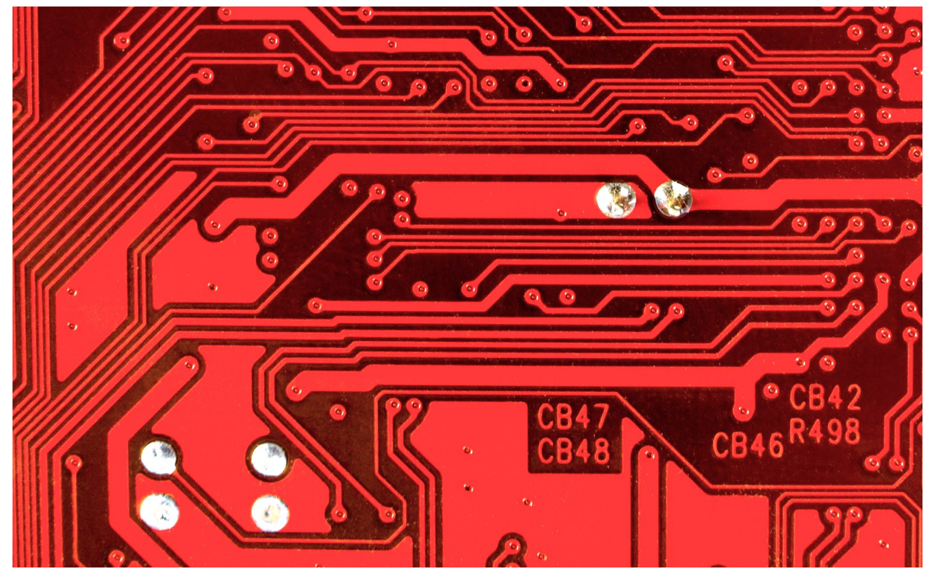




ESCAPE 2



Massively Parallel Computing for NWP and climate



Andreas Mueller

ECMWF DKRZ Deutsches Klimarechenzentrum Max-Planck-Institut für Meteorologie MeteoSwiss Barcelona Supercomputing Center Centro Nacional de Supercomputación cea Loughborough University RMI Politecnico Milano 1863 MOX DMI cmcc Centro Euro-Mediterraneo sui Cambiamenti Climatici Bull atos technologies

The ESCAPE-2 project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 800897

Funded by the European Union

Co-ordinated by ECMWF

Overview

- Why do scientists need to know so much about computer science?
- What do we need to be aware of to write efficient code?
- How good are we?

The ESCAPE-2 project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 800897

Funded by the European Union

Co-ordinated by ECMWF

Why do we as scientists need to know so much about computer science?

The ESCAPE-2 project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 800897

Why do we as scientists need to know so much about computer science?

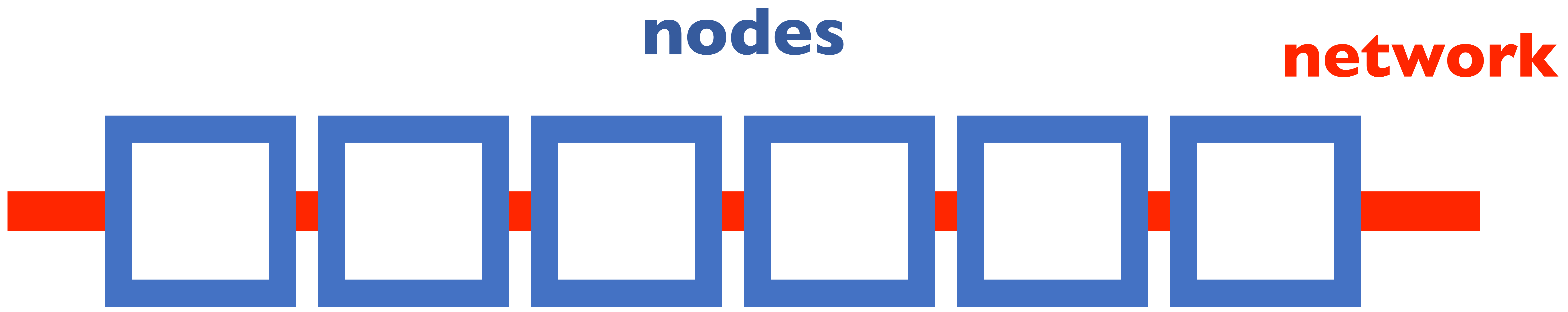
Funded by the European Union

- Excuse 1: let the computer scientists take care of it
- Response: computer scientists cannot do everything because they do not know about different numerical methods
- Excuse 2: just buy a faster computer if the code is not fast enough
- Response: we (and the environment) cannot afford wasting that much energy!

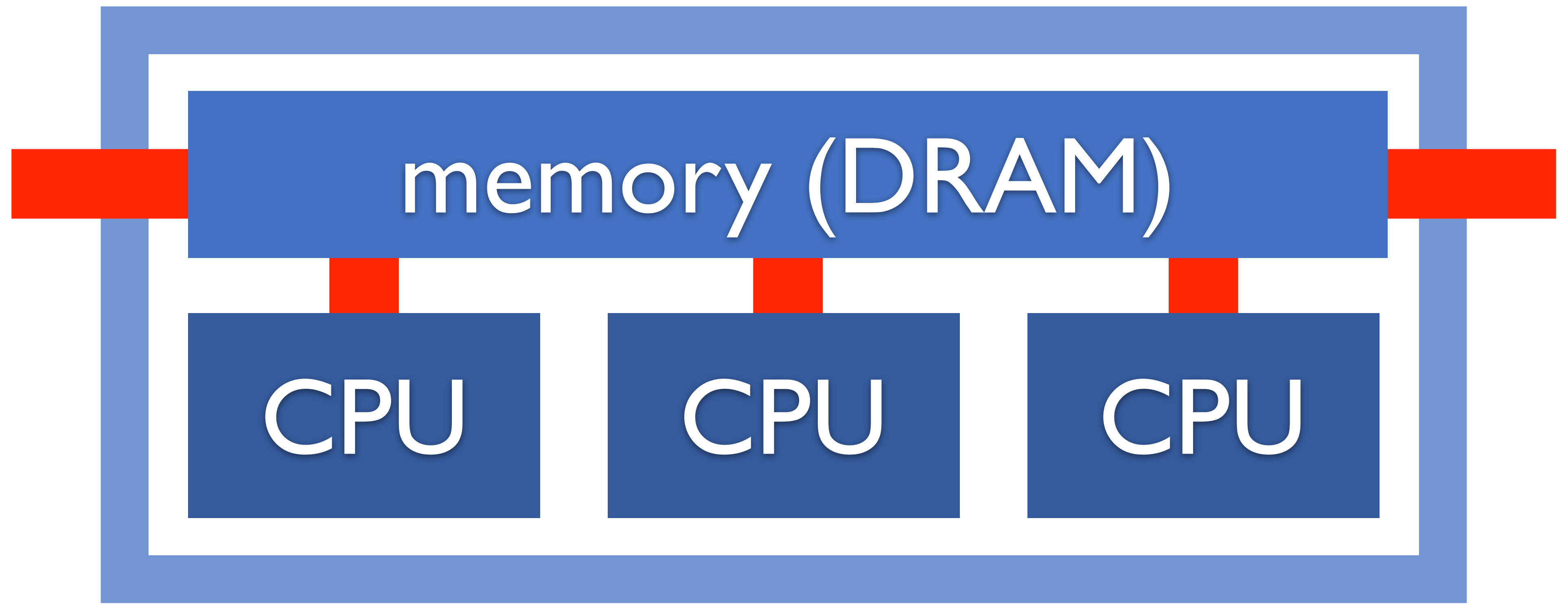
computer	electricity cost per year
ECMWF	~3 million £
fastest current supercomputer	~15 million $
next generation (exascale)	~20 million $

Funded by the European Union

Supercomputer/Cluster



Node



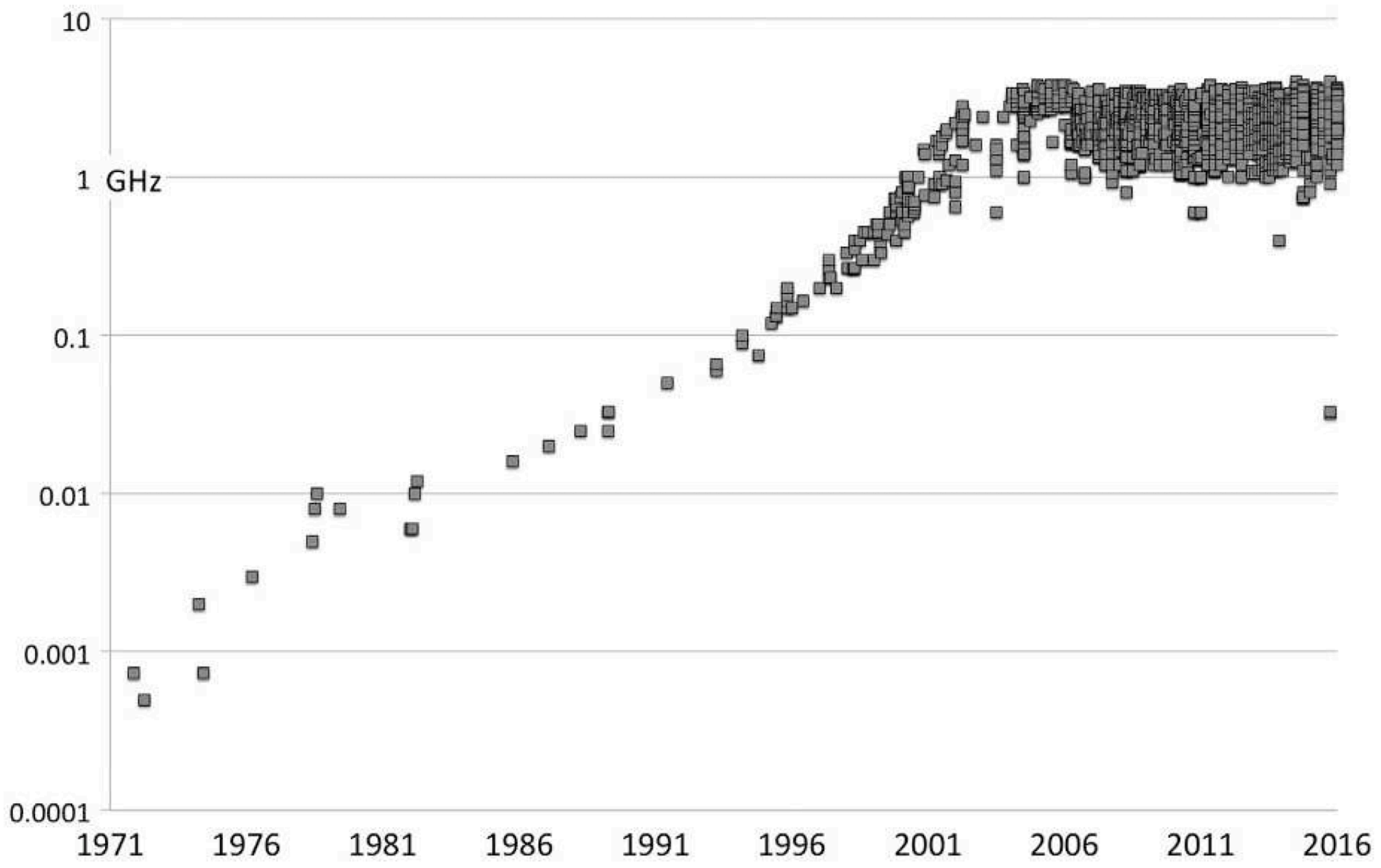
CPU

central processing unit; does one instruction like $c=a+b$ per clock cycle

ESCAPE 2

Funded by the European Union

CPU clock rate over time

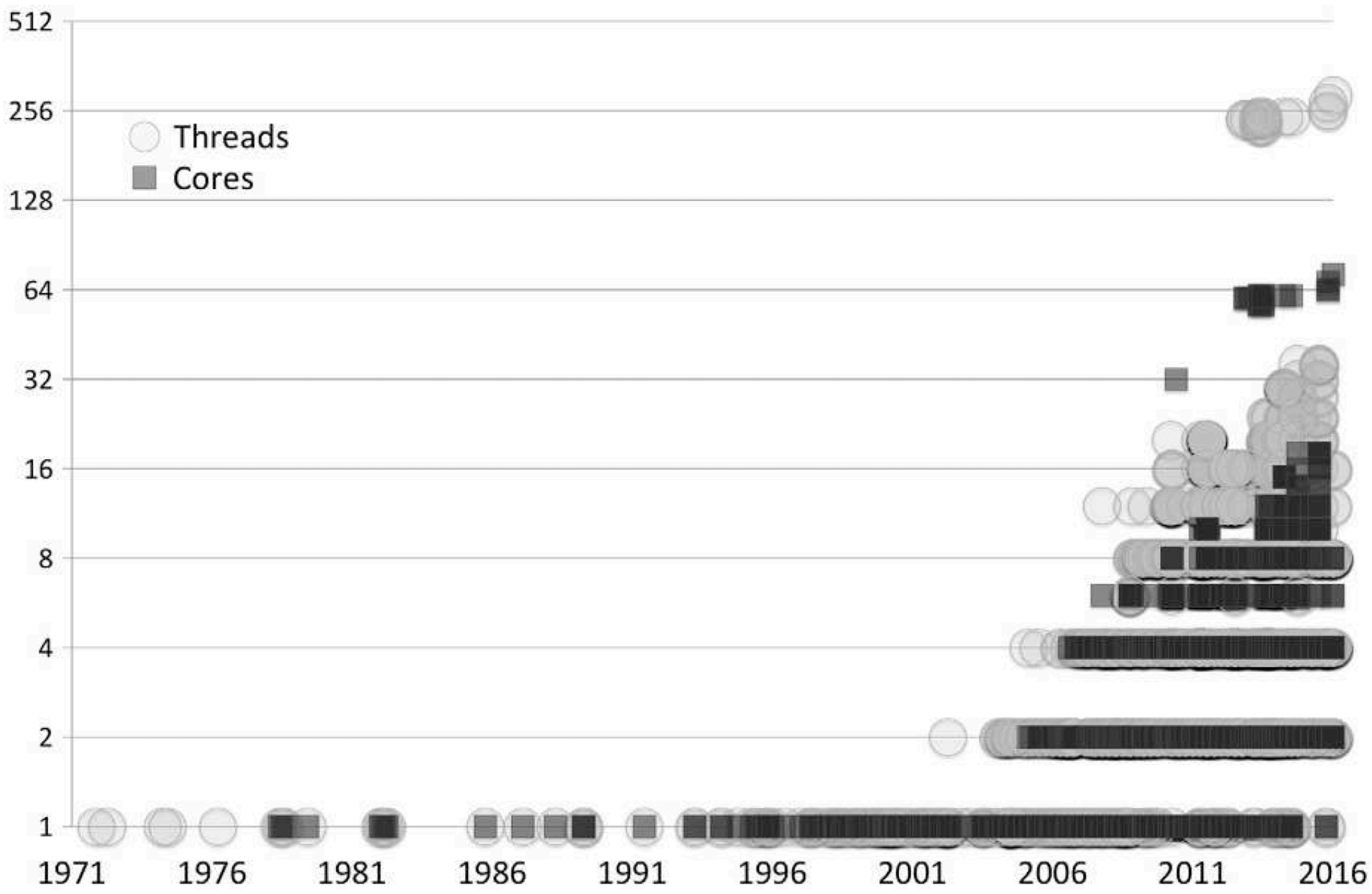


source: James Reinders, Intel Xeon Phi

ESCAPE 2

Funded by the European Union

Number of cores per chip over time

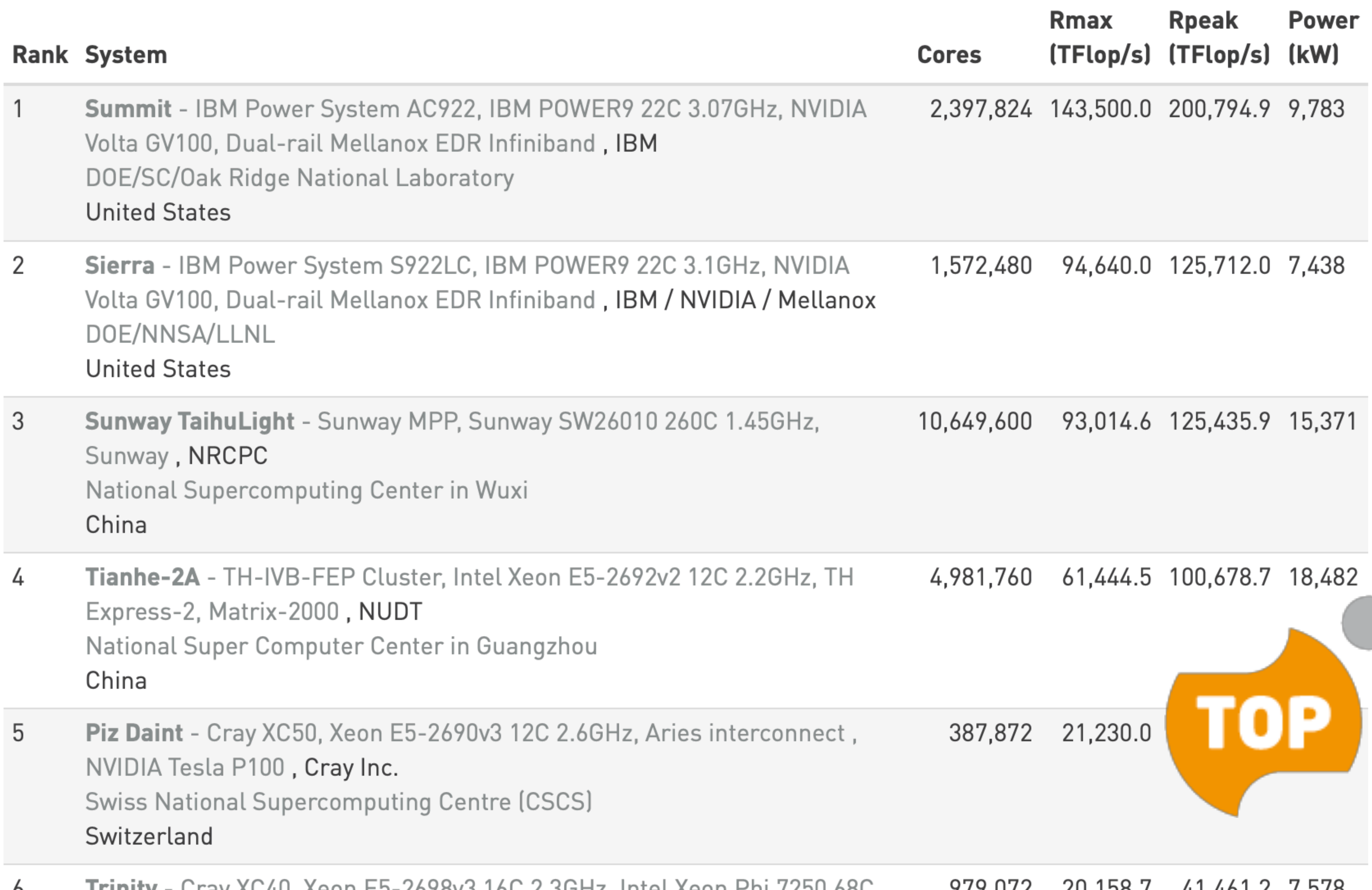


source: James Reinders, Intel Xeon Phi

ESCAPE 2

Funded by the European Union

http://top500.org

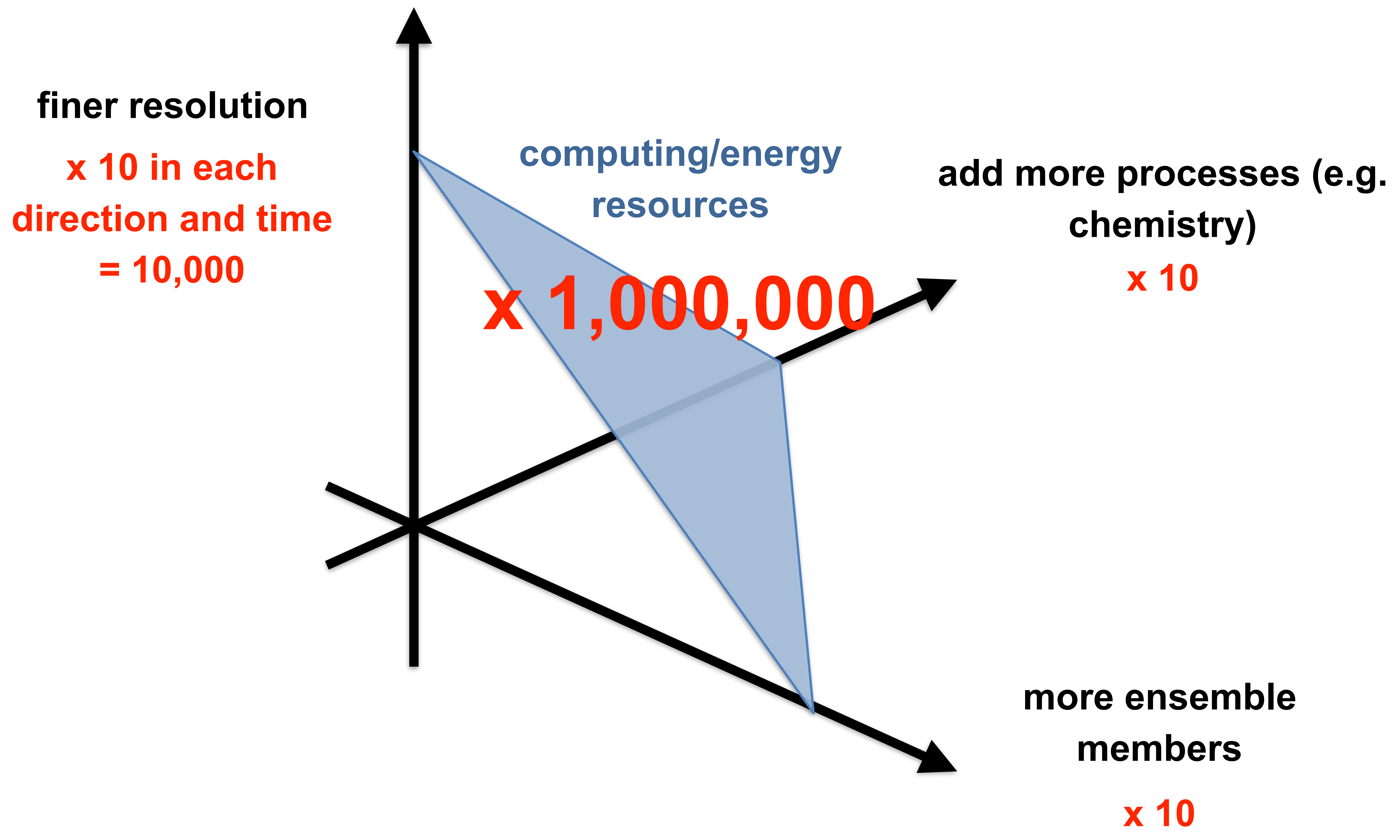


Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,397,824	143,500.0	200,794.9	9,783
2	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
3	**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
4	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
5	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	387,872	21,230.0		
6	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Intel Xeon Phi 7250 68C	979,072	20,158.7	41,461.2	7,578

TOP 500 The List.

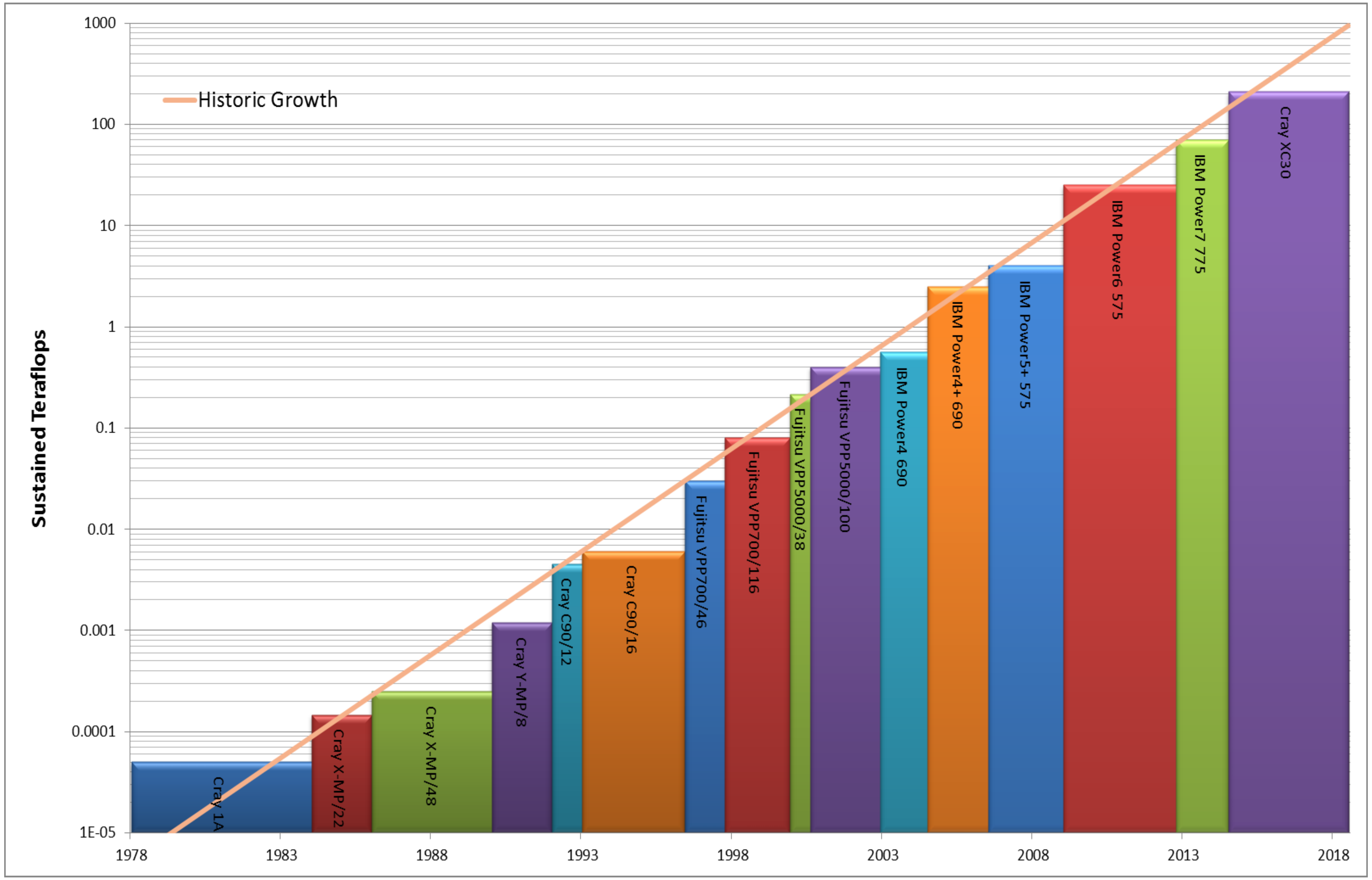
Funded by the European Union

What comes next?


finer resolution
x 10 in each direction and time = 10,000
computing/energy resources
x 1,000,000
add more processes (e.g. chemistry)
x 10
more ensemble members
x 10

Computing at ECMWF

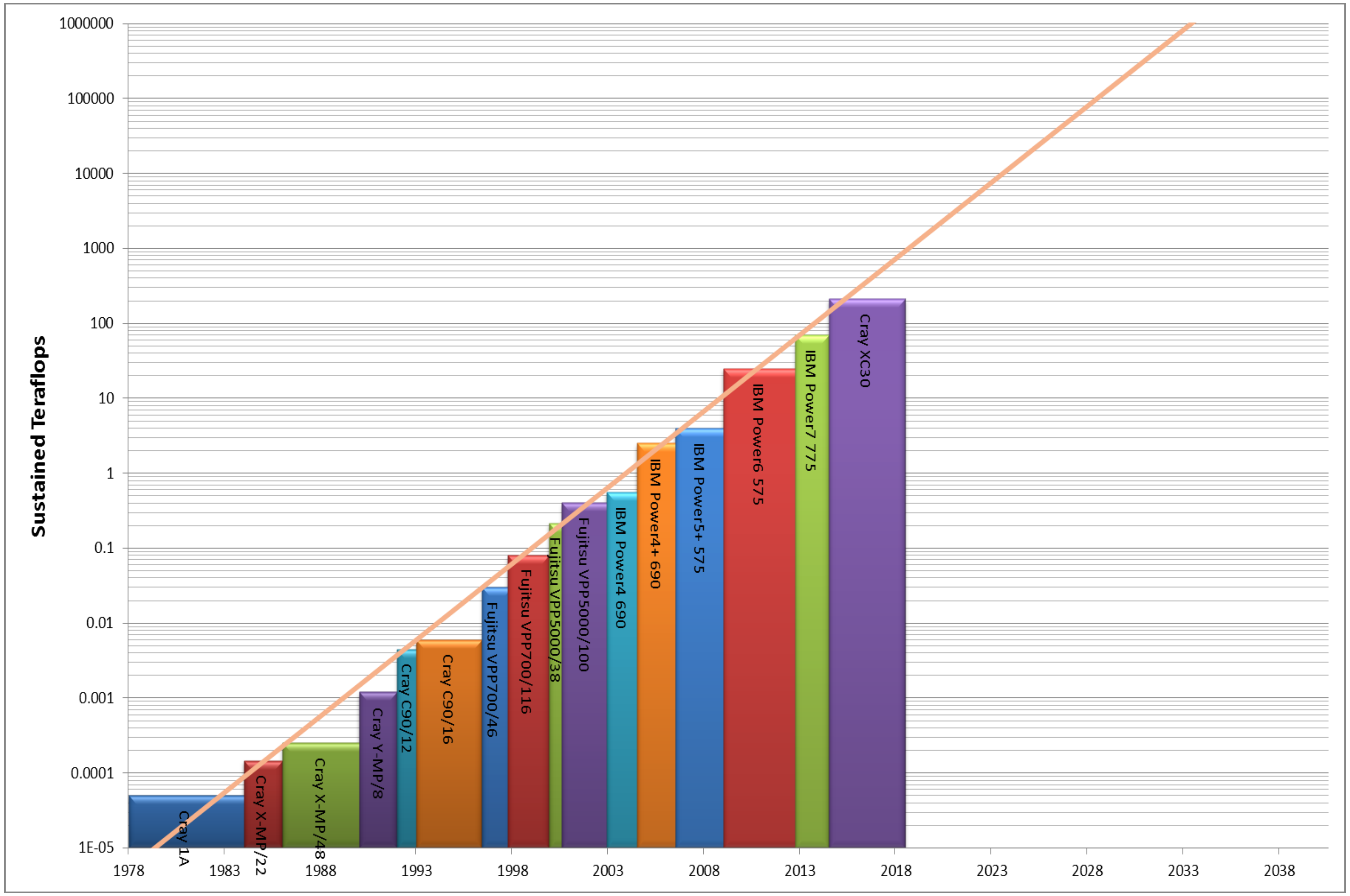
Funded by the European Union



ESCAPE 2

Funded by the European Union

Sustained Exaflop in 2033 ?



Funded by the
European Union

Co-ordinated by ECMWF

What do we need to be aware of to write efficient code?

The ESCAPE-2 project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 800897

Recommendations

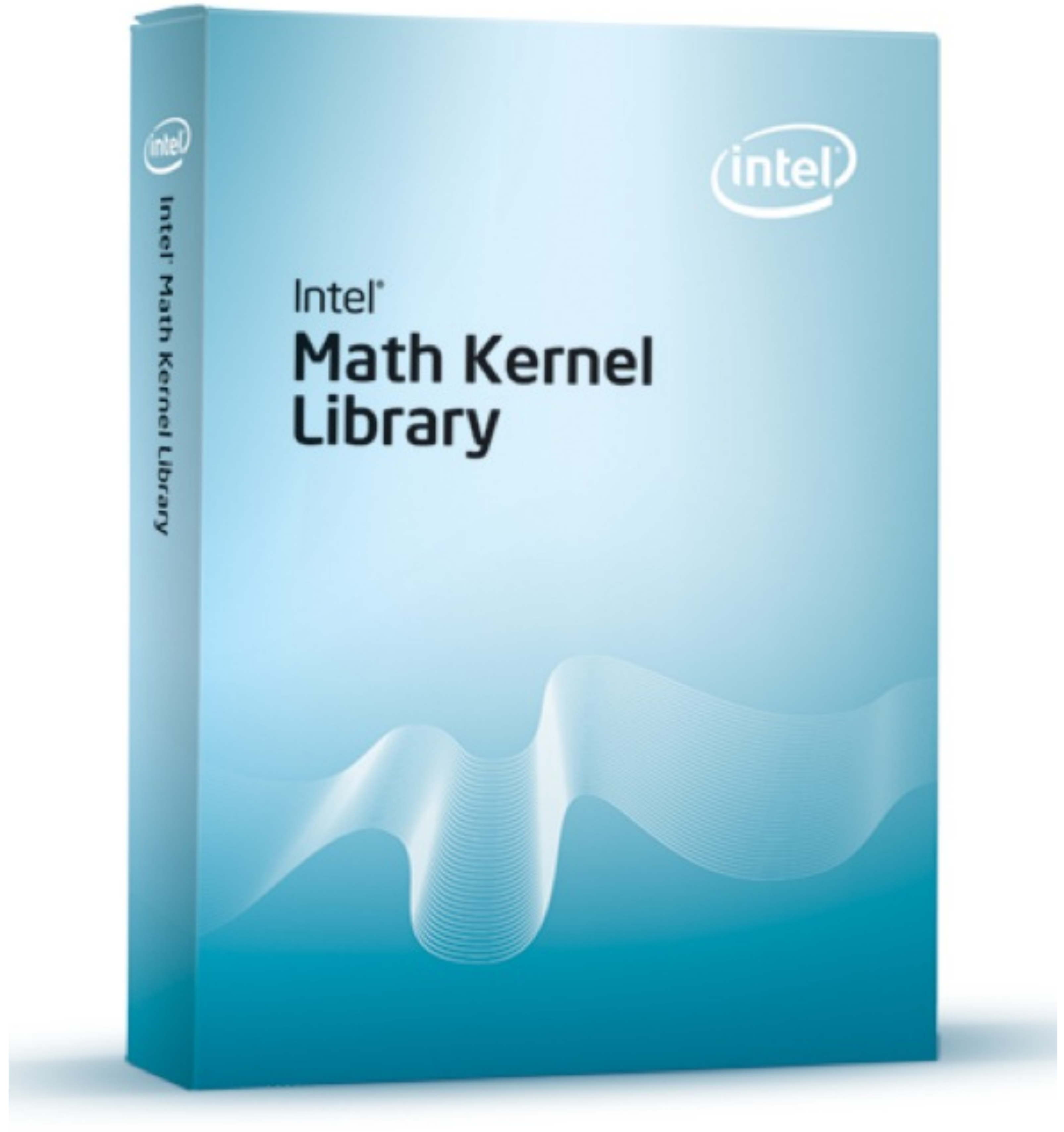
-
-
-

ESCAPE 2

Funded by the European Union

Libraries

- there are well optimised libraries for many tasks
- BLAS for vector-matrix product or matrix-matrix product (if matrices are large)
- Lapack for matrix factorisation (e.g. LU decomposition)
- some hardware vendors have special math libraries, e.g. MKL by Intel
- there are some cases in which libraries are fairly slow (e.g. BLAS with very small matrices)



Funded by the
European Union

Co-ordinated by ECMWF

Recommendations

- **try if using libraries is fast enough**

Compiler optimisation

- compilers have optimisation flag -On (O0: no optimisation, O3: strong compiler optimisation)
- O3 is usually much faster than O2, but it can also be slower than O2
- O3 can produce completely wrong results!
- you can use different compiler flags for different files
- different compiler versions can have very different performance
- check compiler messages (Intel: ifort -O2 -qopt-report=2 code.f90 -o program)
- make sure that your code runs correctly with different compilers

Funded by the
European Union

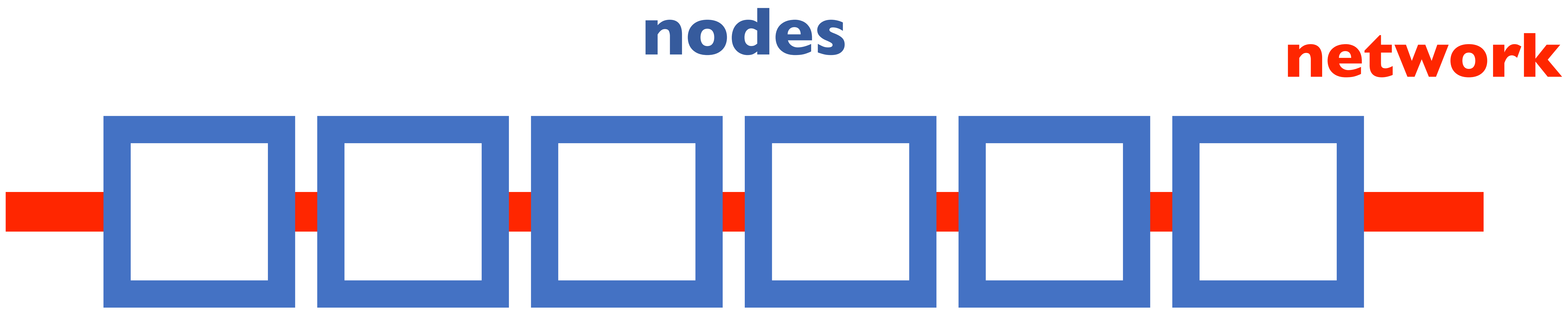
Co-ordinated by ECMWF

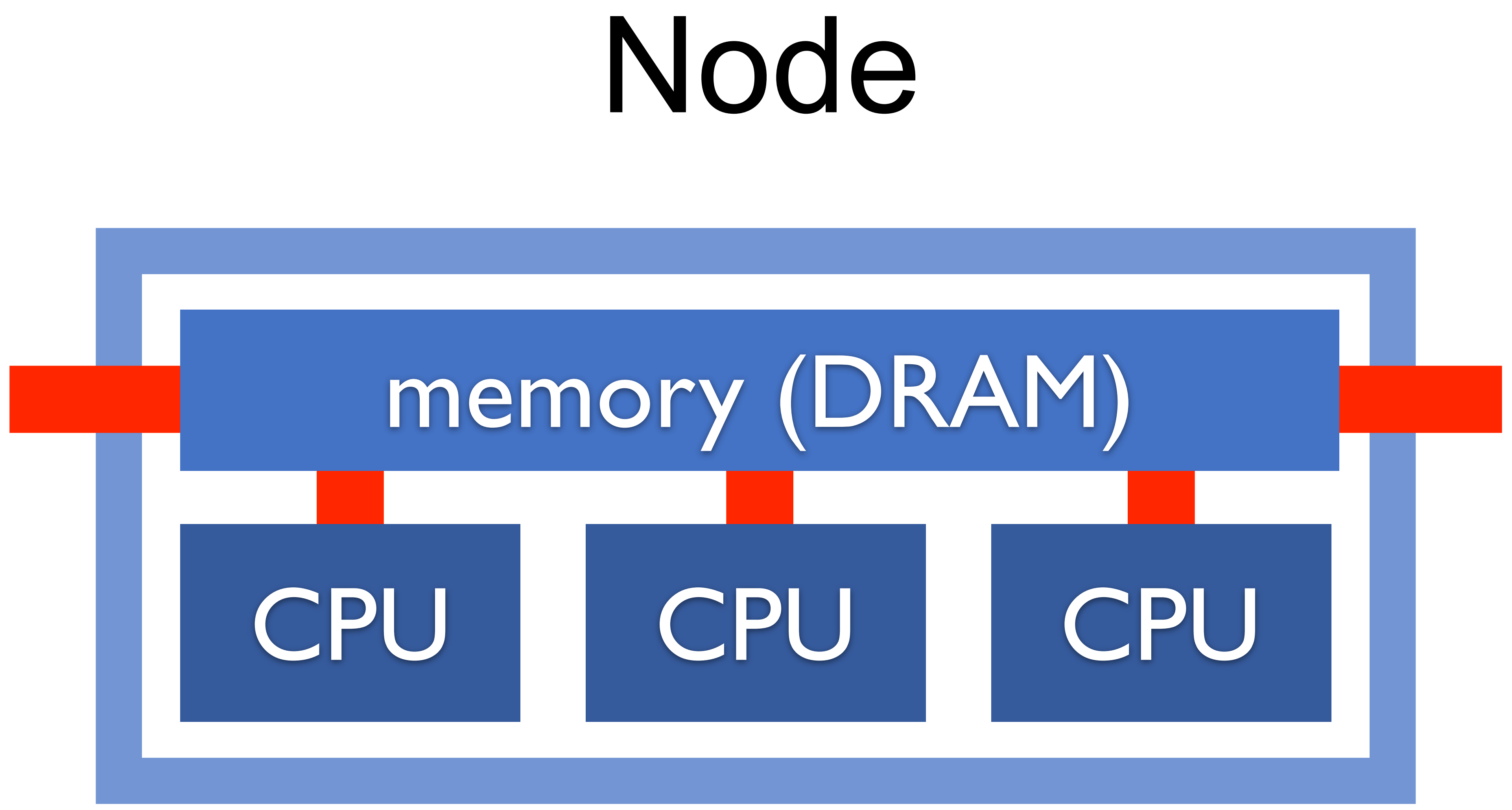
Recommendations

- try if using libraries is fast enough
- **try to use compiler optimisation (be careful!)**

Funded by the European Union

Supercomputer/Cluster





Bottlenecks

- network (connection between nodes)
- connection between DRAM and processor

Funded by the European Union

Co-ordinated by ECMWF

Recommendations

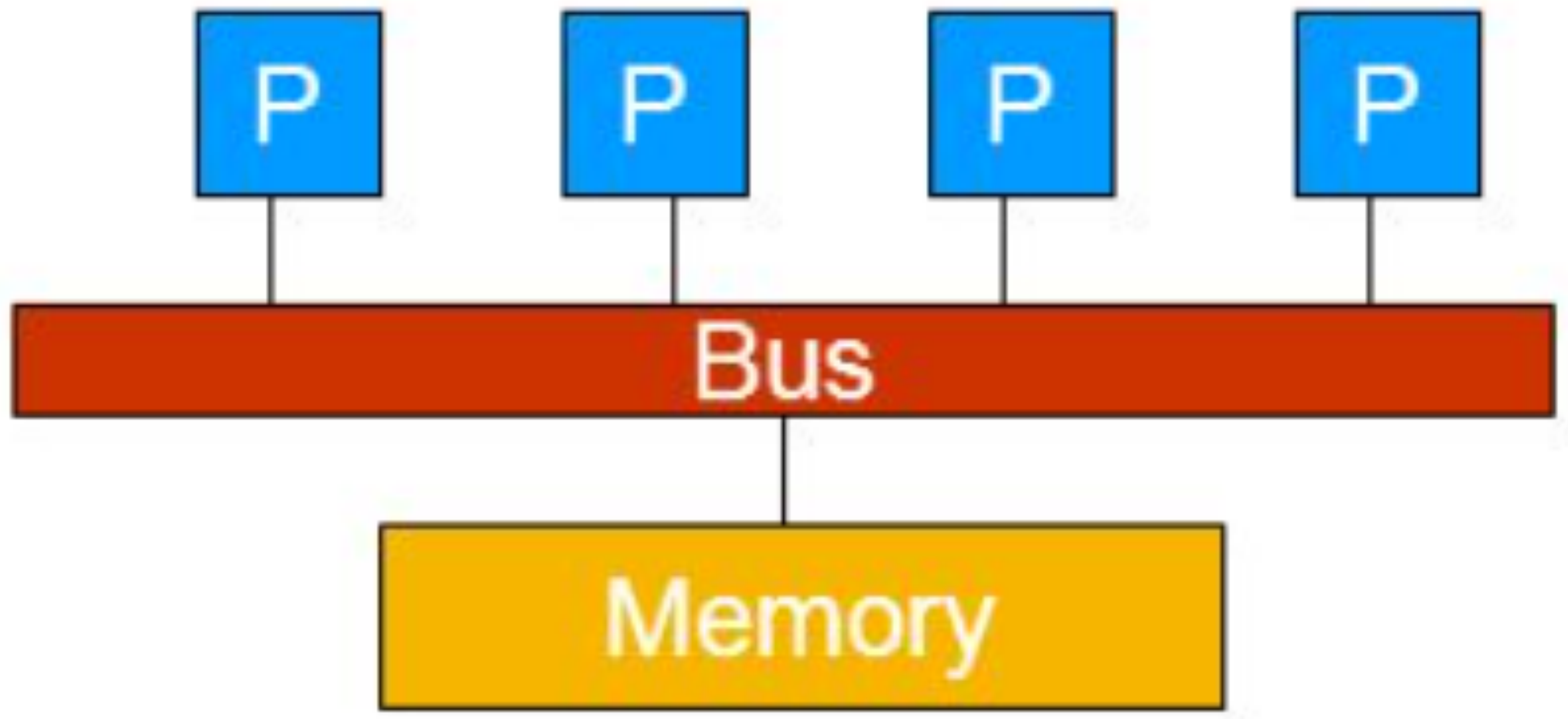
- try if using libraries is fast enough
- try to use compiler optimisation (be careful!)
- **avoid unnecessary computation and communication**

ESCAPE 2

Funded by the European Union

Shared memory: OpenMP

- many threads of a process run on a single node
- all threads can access the same data
- data may be physically distributed, but logically shared



without OpenMP:

```
real, dimension(N) :: a,b
integer :: i,N
do i=1,N
 a(i) = a(i) + b(i)
end do
```

with OpenMP:

```
real, dimension(N) :: a,b
integer :: i,N
!$omp parallel do private(i)
do i=1,N
 a(i) = a(i) + b(i)
end do
!$omp end parallel do
```

ESCAPE 2

Funded by the European Union

Shared memory: OpenMP

faster for bigger codes:

```
real, dimension(N) :: a,b
integer :: i, N, iStart, iEnd,
 myid, numthreads
!$omp parallel private(i,iStart,iEnd)
myid = omp_get_thread_num()
numthreads = omp_get_num_threads()
iStart = ...
iEnd = ...
do i=iStart,iEnd
 a(i) = a(i) + b(i)
end do
!$omp end parallel
```

without OpenMP:

```
real, dimension(N) :: a,b
integer :: i,N
do i=1,N
 a(i) = a(i) + b(i)
end do
```

with OpenMP:

```
real, dimension(N) :: a,b
integer :: i,N
!$omp parallel do private(i)
do i=1,N
 a(i) = a(i) + b(i)
end do
!$omp end parallel do
```

Funded by the European Union

Co-ordinated by ECMWF

Recommendations

- try if using libraries is fast enough
- try to use compiler optimisation (be careful!)
- avoid unnecessary computation and communication
- **give each thread as much work as possible**

ESCAPE 2

Funded by the European Union

Shared memory: OpenMP

Example 2: race conditions

without OpenMP:

```
real, dimension(N) :: a
real :: sum
integer :: i,N
do i=1,N
 sum = sum + a(i)
end do
```

with OpenMP (wrong!):

```
real, dimension(N) :: a
real :: sum
integer :: i,N
!$omp parallel do private(i)
do i=1,N
 sum = sum + a(i)
end do
!$omp end parallel do
```

working, but slow:

```
real, dimension(N) :: a
real :: sum
!$omp parallel do private(i)
do i=1,N
 !$omp atomic
 sum = sum + a(i)
end do
!$omp end parallel do
```

faster:

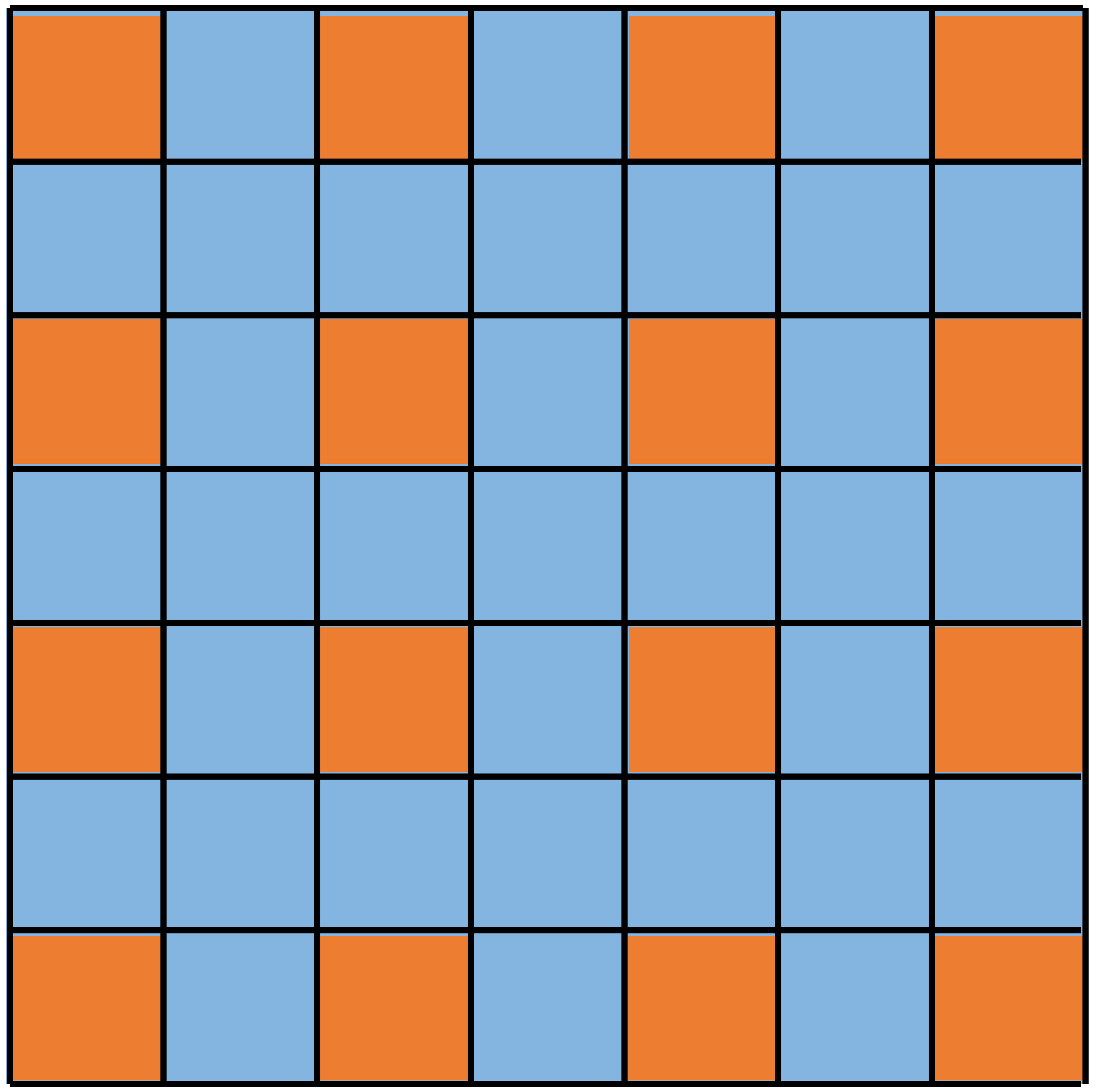
```
real, dimension(N) :: a
real :: sum
!$omp parallel do private(i
 reduction (+: sum )
do i=1,N
 sum = sum + a(i)
end do
!$omp end parallel do
```

Funded by the European Union

Shared memory: OpenMP

Example 2: race conditions

best: arrange work such that different threads work on different data



example: spectral element, start with orange (non-adjacent) elements

Funded by the
European Union

Co-ordinated by ECMWF

Recommendations

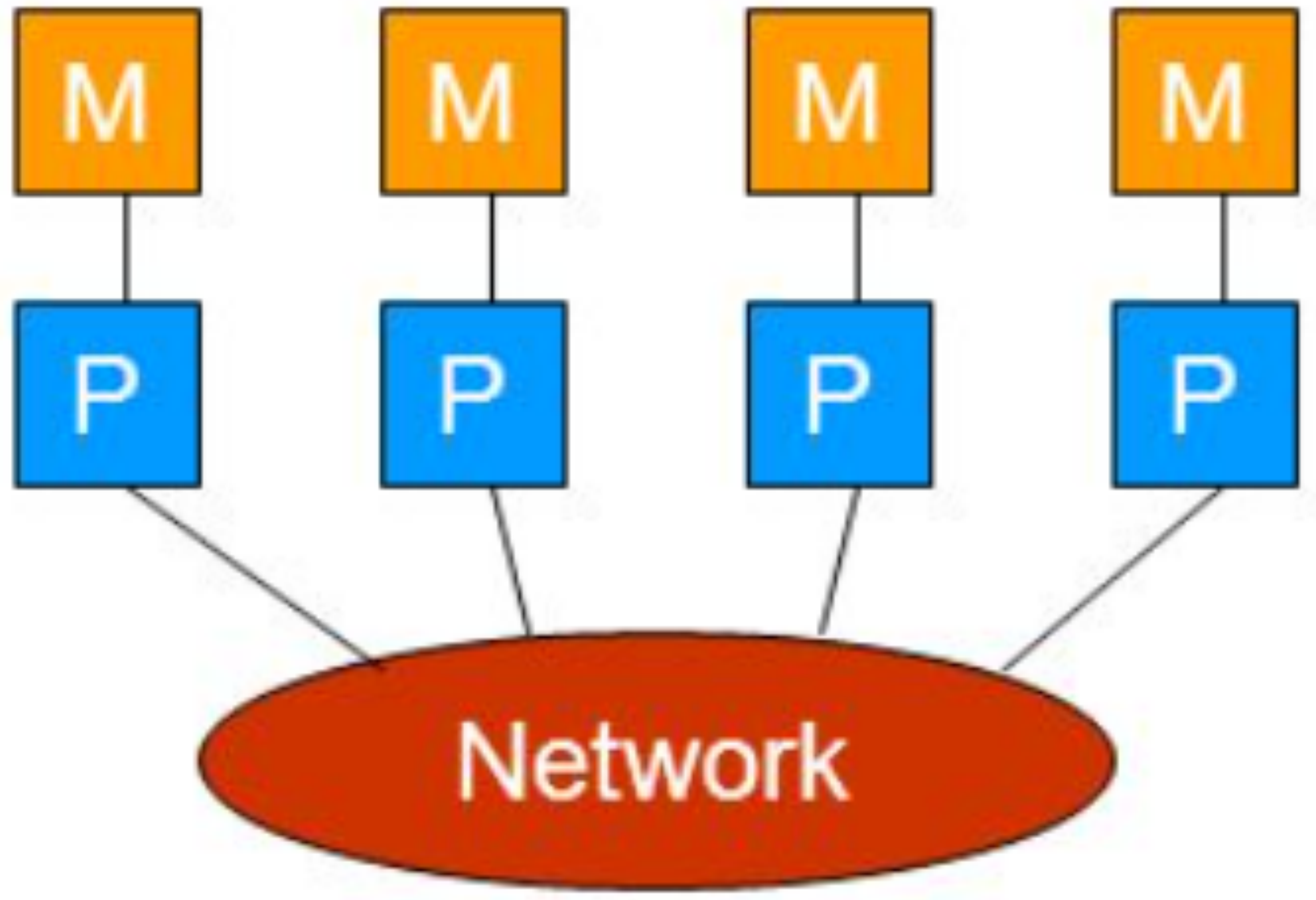
- try if using libraries is fast enough
- try to use compiler optimisation (be careful!)
- avoid unnecessary computation and communication
- give each thread as much work as possible
- **let the threads do work that does not affect others**

ESCAPE 2

Distributed memory: MPI

Funded by the European Union

- many processes run on multiple nodes
- process can access only data on the node it is running
- use communication library MPI (Message Passing Interface) to access data on other nodes

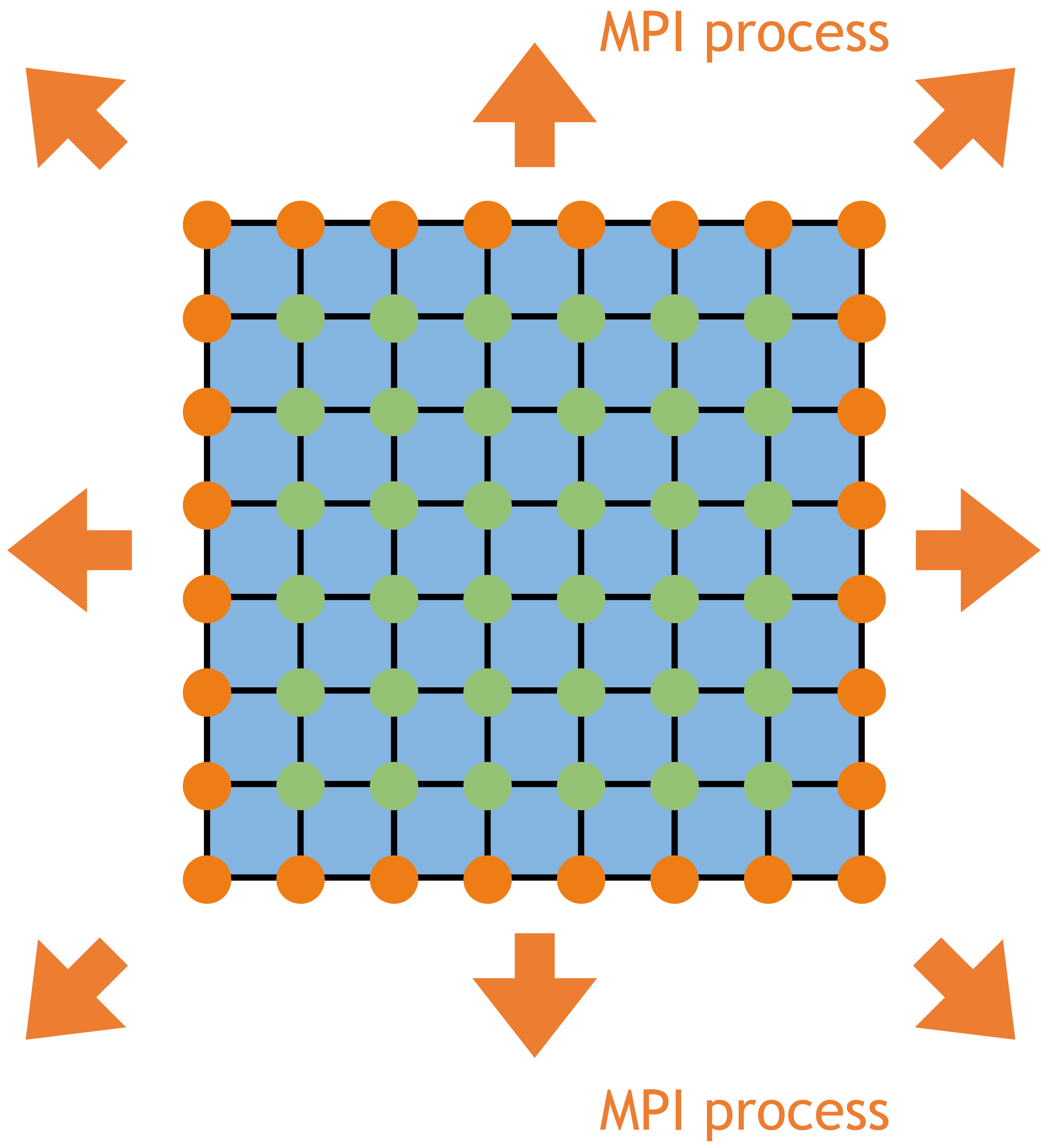


```
integer :: len, destination, tag, nreq
comm = mpi_comm_world
call mpi_init(ierr)
call mpi_comm_rank(comm, myid, ierr)
call mpi_comm_size(comm, numproc, ierr)
nreq = 0
...
do i=1,N ! loop over processors with which we
    want to communicate
 destination = ...
 nreq = nreq + 1
 call mpi_irecv(recvdata, len, mpi_real,
     destination, tag, comm, request(nreq), ierr)
 nreq = nreq + 1
 call mpi_isend(senddata, len, mpi_real,
     destination, tag, comm, request(nreq), ierr)
end do
... do some work ...
call mpi_waitall(nreq, request, status, ierr)
call mpi_finalize(ierr)
```

Funded by the European Union

Overlap communication and computation

- Example: grid point method with only next neighbour communication:
    - compute values along processor boundaries first (orange) and send result to neighbours
    - compute interior points while the data is on its way (green)
- try to reduce the physical distance that data needs to travel (difficult)



Funded by the European Union

Co-ordinated by ECMWF

Recommendations

- try if using libraries is fast enough
- try to use compiler optimisation (be careful!)
- avoid unnecessary computation and communication
- give each thread as much work as possible
- let the threads do work that does not affect others
- **overlap computation and communication**

Funded by the European Union

Use data once per time-step

bad example:

```
real, dimension(N) :: a,b
real :: sum
integer :: i,N
sum = 0.0
a = 0.0
b = 0.0
do i=1,N
 b(i) = i
end do
do i=1,N
 a(i) = a(i) + b(i)
end do
do i=1,N
 sum = sum + a(i)
end do
print*,sum
```

good:

```
real, dimension(N) :: a,b
real :: sum
integer :: i,N
sum = 0.0
do i=1,N
 a(i) = 0.0
 b(i) = i
 a(i) = a(i) + b(i)
 sum = sum + a(i)
end do
print*,sum
```

Funded by the European Union

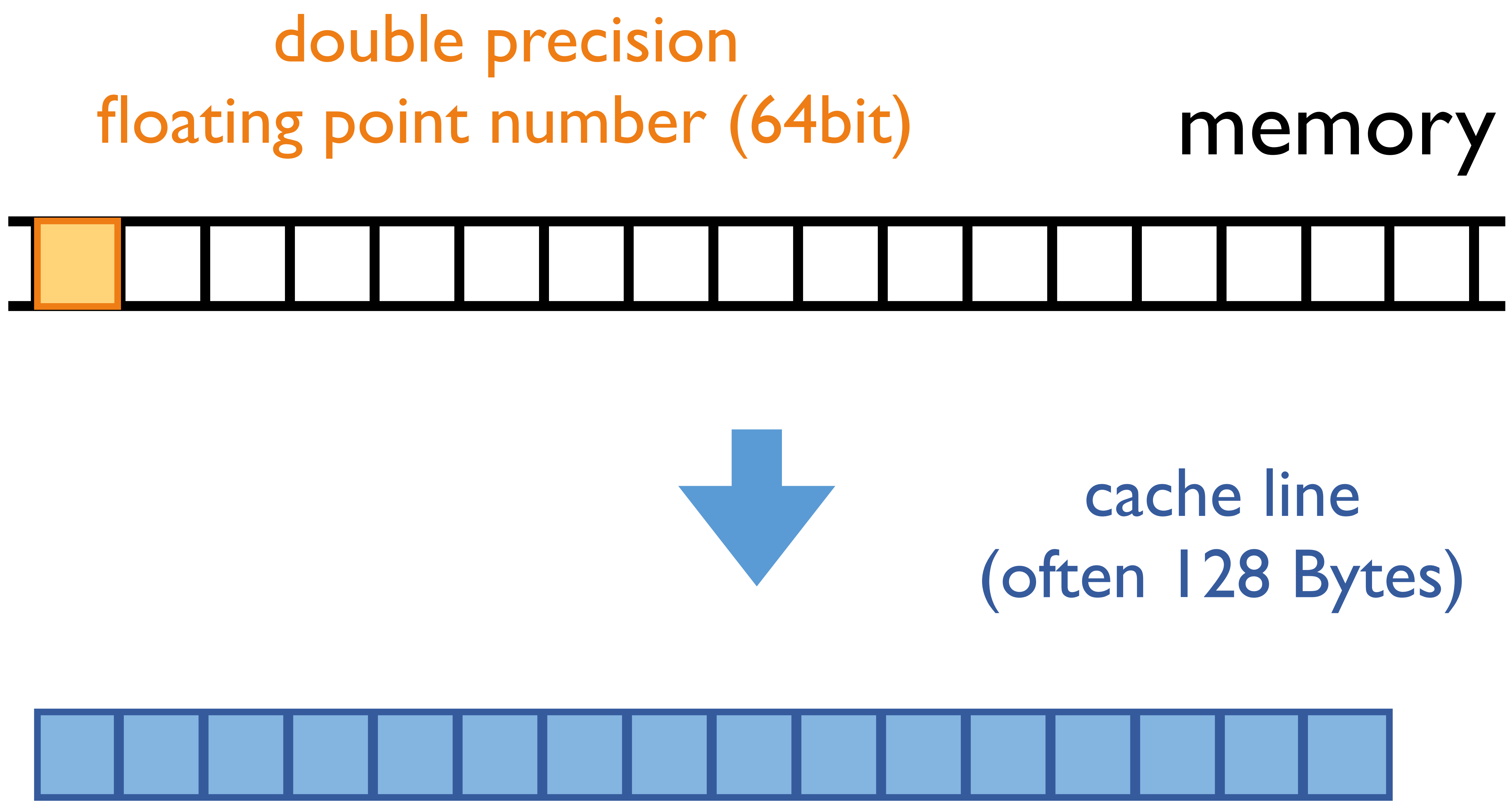
Co-ordinated by ECMWF

Recommendations

- try if using libraries is fast enough
- try to use compiler optimisation (be careful!)
- avoid unnecessary computation and communication
- give each thread as much work as possible
- let the threads do work that does not affect others
- overlap computation and communication
- **use data only once per time-step**

Funded by the European Union

Contiguous memory access



store data in the order in which you need it and use it in this order!

Fortran (column major order):

```
real, dimension(N,M) :: a,b
integer :: i,j,N,M
do j=1,M
 do i=1,N
  a(i,j) = a(i,j) + b(i,j)
  ! fast index should be i
 end do
end do
```

C (row major order):

```
int i,j,N,M;
for (i=0; i<N; i++) {
 for (j=0; j<M; j++) {
  a[i][j] = a[i][j] + b[i][j]
  // fast index should be j
 }
}
```

Funded by the European Union

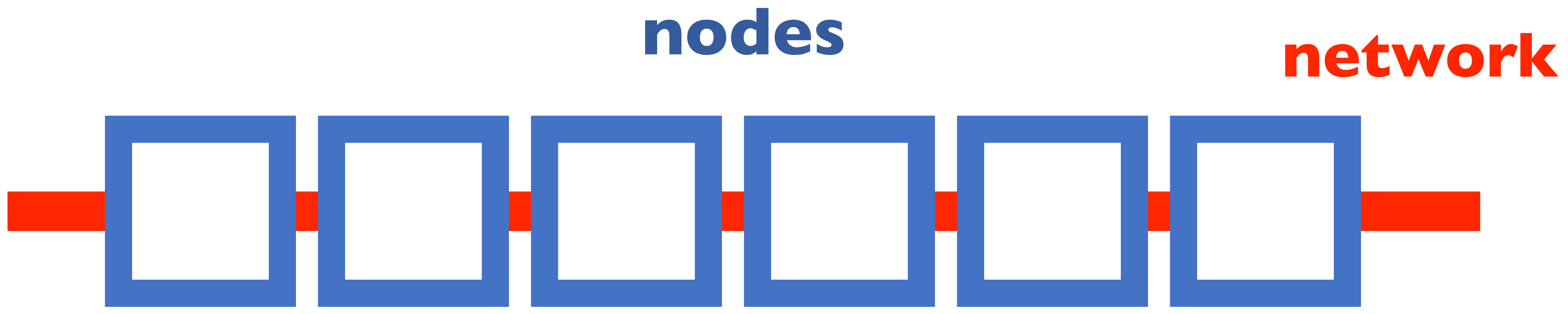
Co-ordinated by ECMWF

Recommendations

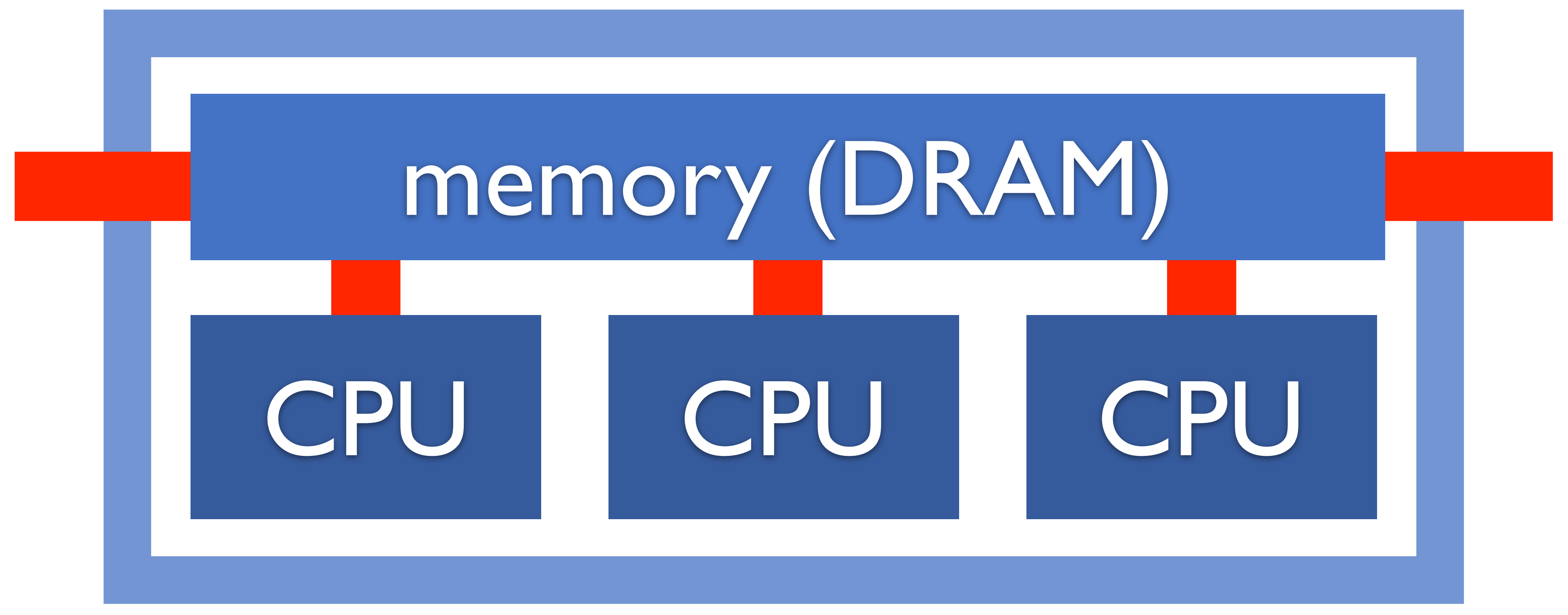
- try if using libraries is fast enough
- try to use compiler optimisation (be careful!)
- avoid unnecessary computation and communication
- give each thread as much work as possible
- let the threads do work that does not affect others
- overlap computation and communication
- use data only once per time-step
- **contiguous memory access**

Funded by the European Union

Supercomputer/Cluster



Node

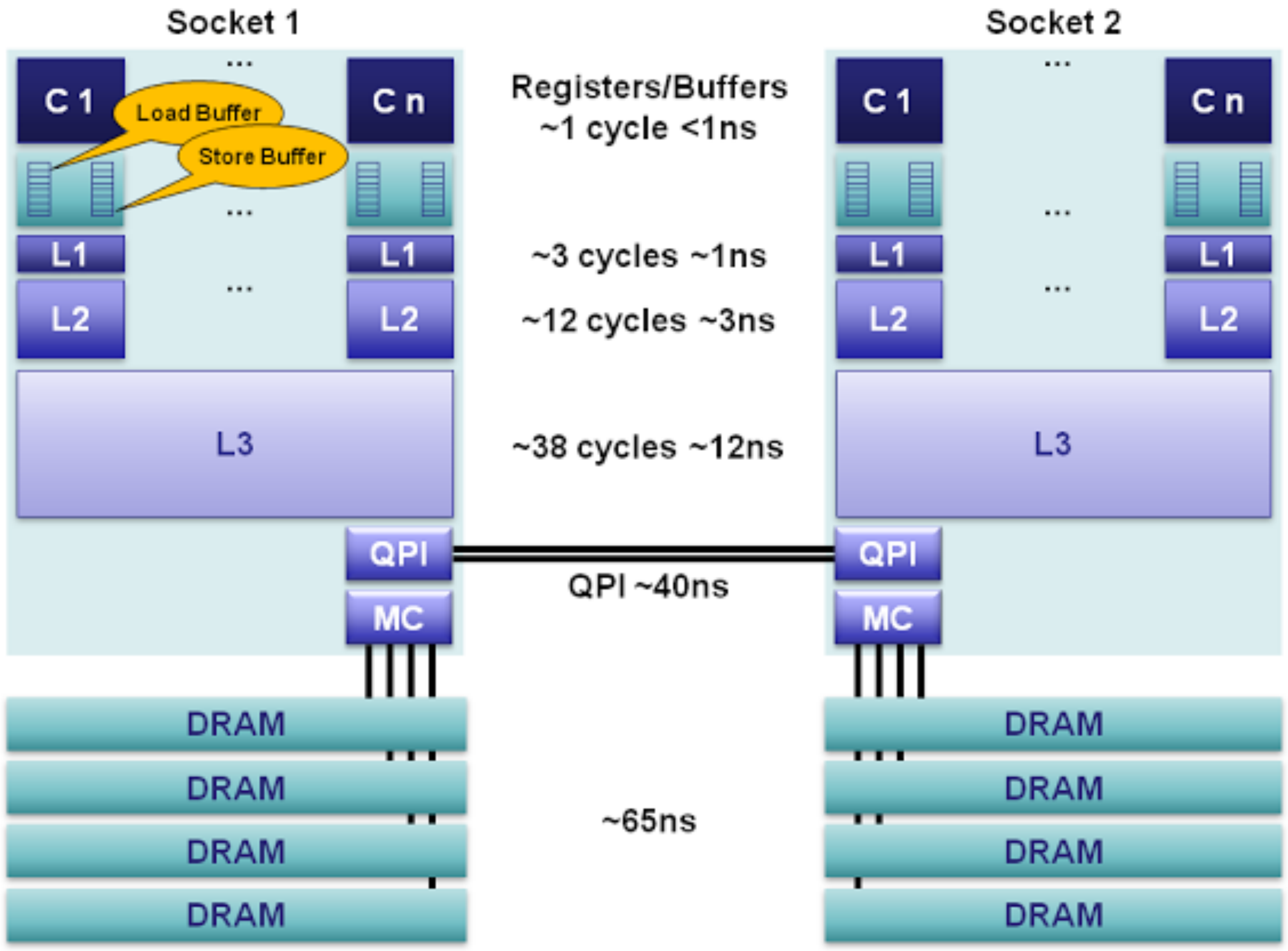


CPU

central processing unit; does one instruction like c=a+b per clock cycle

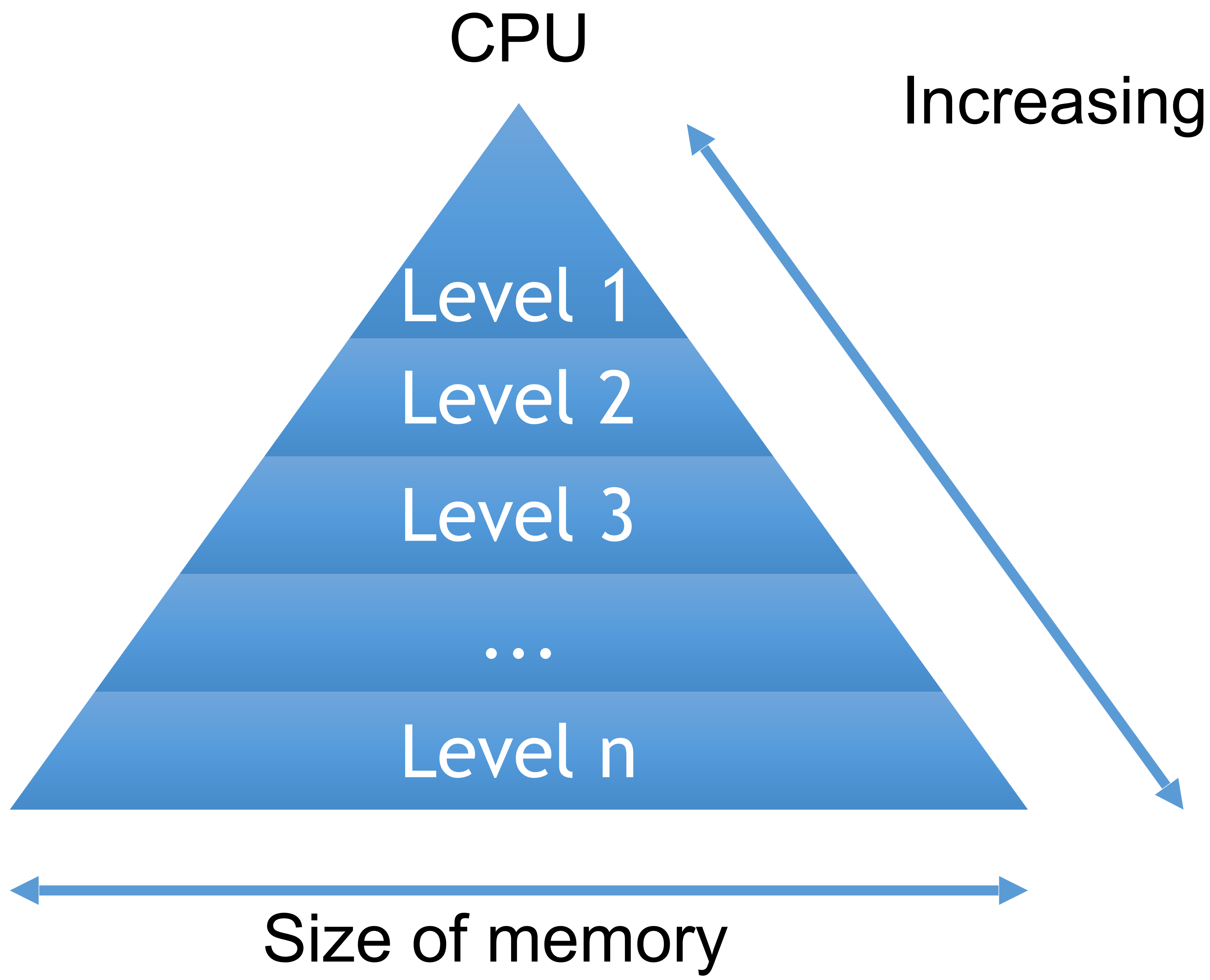
Funded by the European Union

Memory hierarchy inside one node



Funded by the European Union

Cache



Increasing distance from CPU = larger access time

Example:

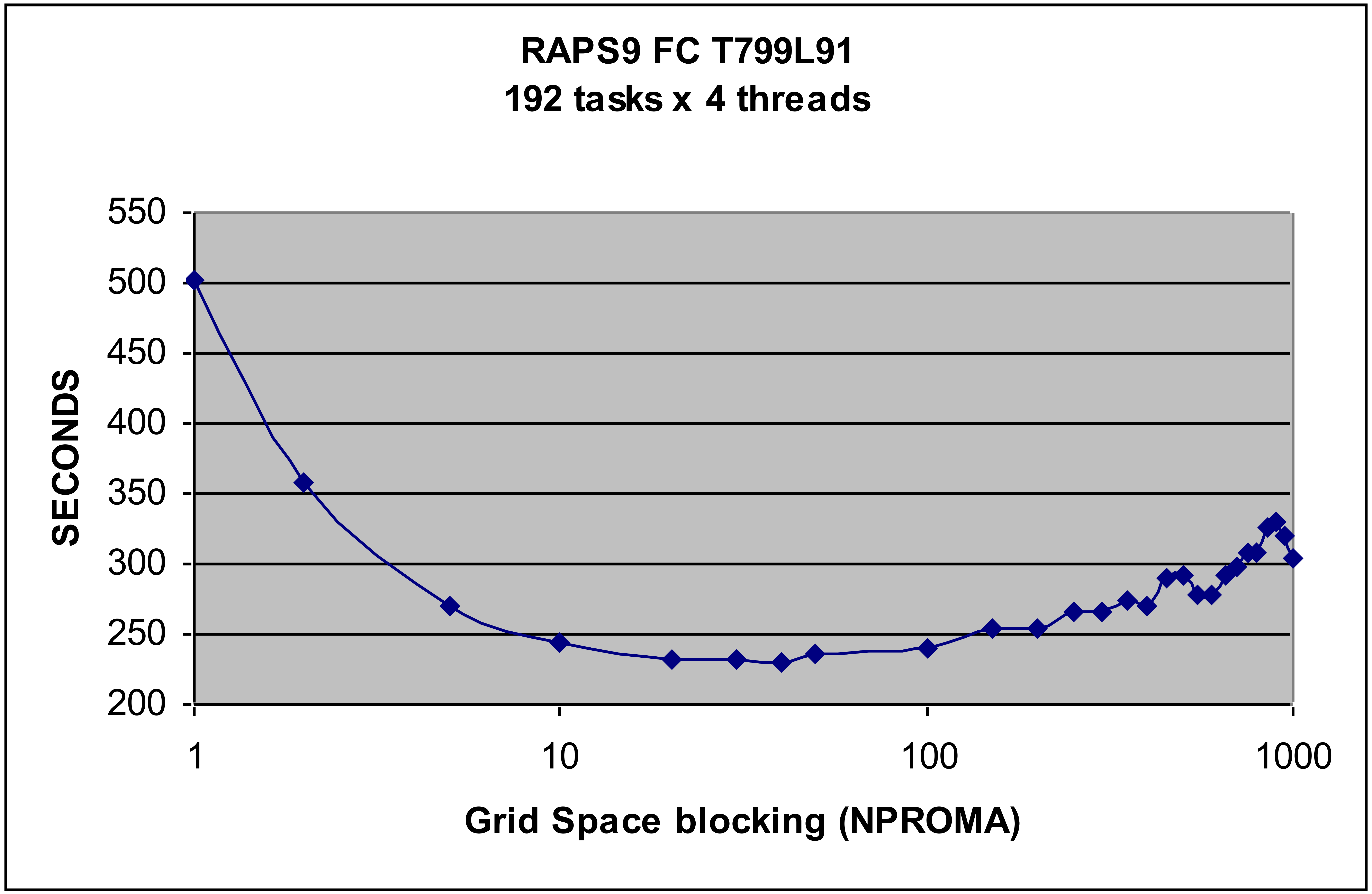
L1: 32 kB, latency 3 cycles
L2: 256 kB, latency 10 cycles
L3: 8MB, latency 40 cycles
DRAM: 16GB, latency 200 cycles
DISK: 1TB, latency 1.000.000 cycles

Cache hit – data found in cache
Cache miss – data not found in cache, thus must be copied from lower memory level
Capacity miss – cache runs out of space for new data
Conflict miss – more that one item is mapped to the same location in cache

ESCAPE 2

IFS: divide work into blocks with length NPROMA

Funded by the European Union



Funded by the European Union

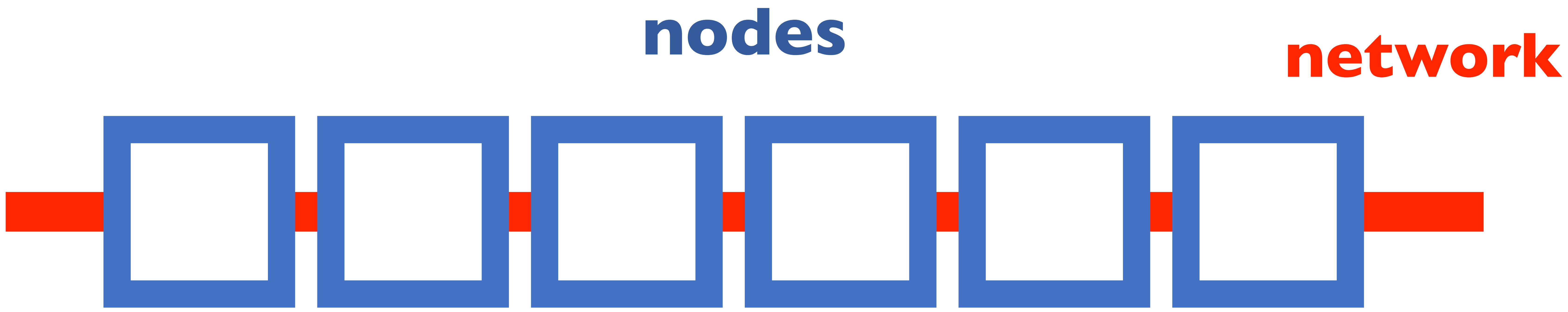
Co-ordinated by ECMWF

Recommendations

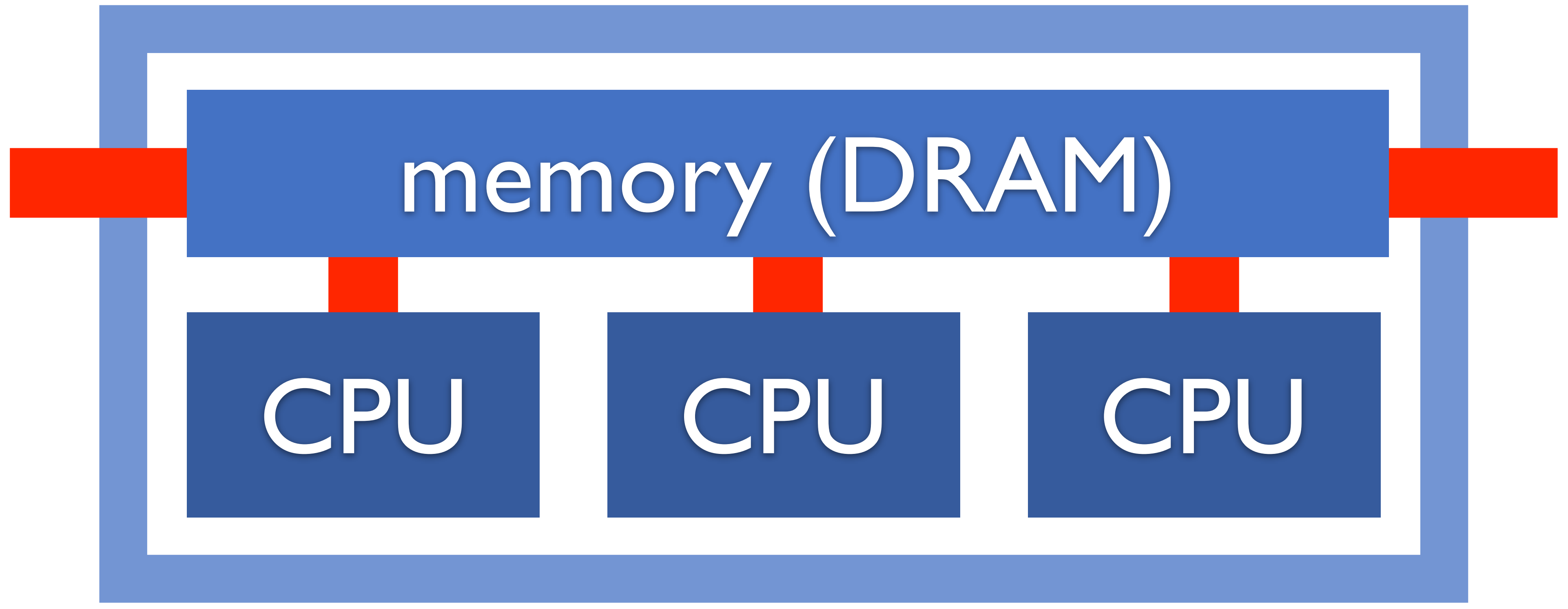
- try if using libraries is fast enough
- try to use compiler optimisation (be careful!)
- avoid unnecessary computation and communication
- give each thread as much work as possible
- let the threads do work that does not affect others
- overlap computation and communication
- use data only once per time-step
- contiguous memory access
- **try to fit data into cache**

Funded by the European Union

Supercomputer/Cluster



Node



Bottlenecks

- network (connection between nodes)
- connection between DRAM and processor

Fast and slow operations

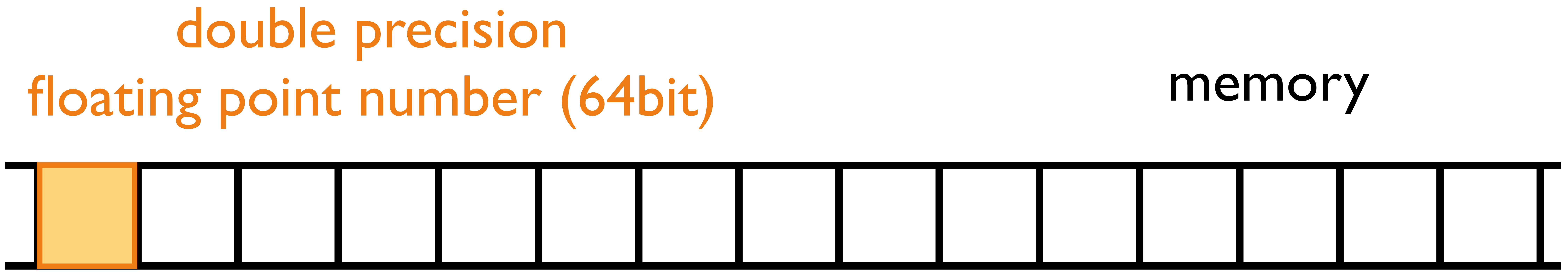
Funded by the European Union

- In terms of cost
- Fast and inexpensive: add, multiply, sub, fma (fused multiply add)
- Medium: divide, modulus, sqrt
- Slow: power, trigonometric functions

- try linear algebra (BLAS, LAPACK) and math libraries (Intel MKL)

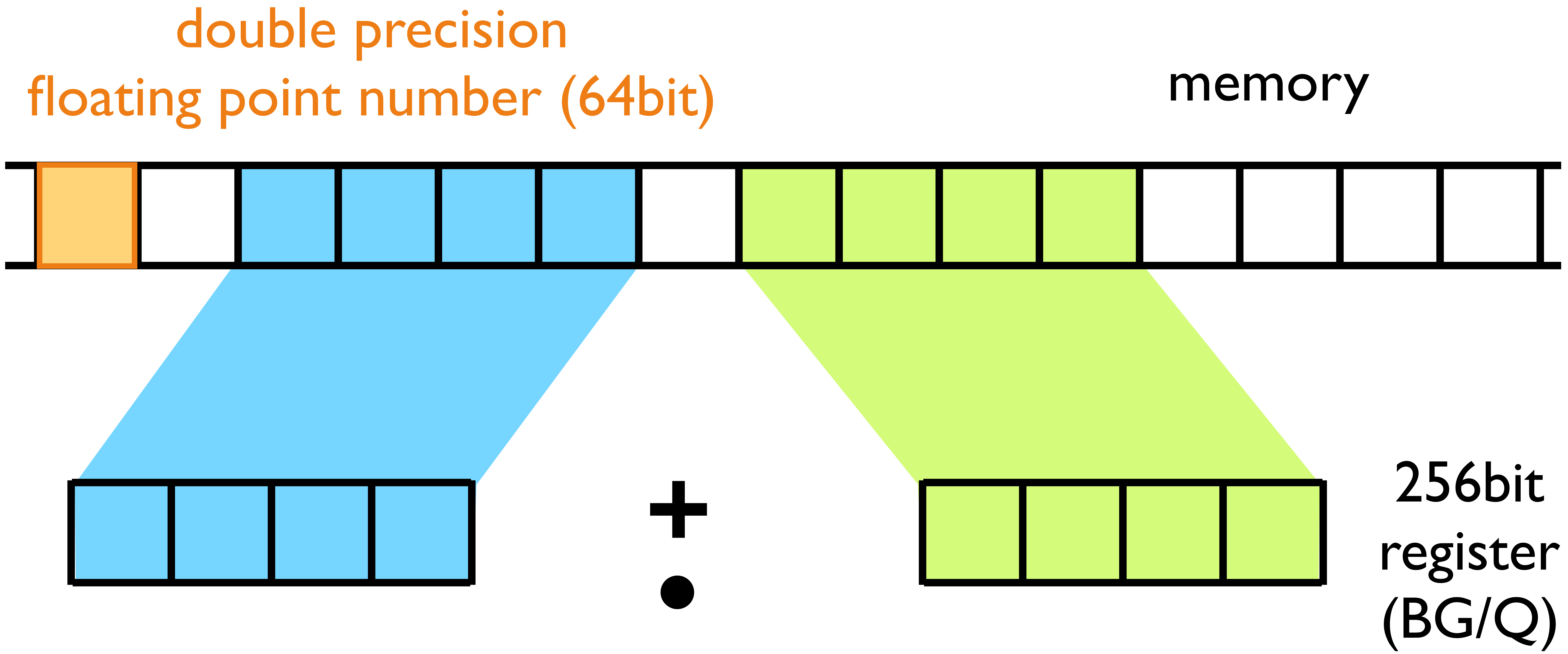
Funded by the European Union

Vectorisation



Funded by the European Union

Vectorisation



ESCAPE 2

Funded by the European Union

Vectorisation

initial version:

```
real :: rho, rho_x, rho_y, rho_z, u, v, w, rhs
do e=1,num_elem ! loop through all elements
   do i=1,num_points_e ! loop through all points of
     the element e
      ... ! compute derivatives rho_x, rho_y, rho_z
      rhs = u*rho_x + v*rho_y + w*rho_z + ...
   end do !i
end do !e
```

optimised for compiler vectorisation:

```
real, dimension(num_points_e) :: rho, rho_x, rho_y, &
   rho_z, u, v, w, rhs
do e=1,num_elem ! loop through all elements
   ... ! compute derivatives like rho_x, rho_y, rho_z
   rhs = u*rho_x + v*rho_y + w*rho_z + ...
end do !e
```

ESCAPE 2

Funded by the European Union

Vectorisation

initial version:

```
real :: rho, rho_x, rho_y, rho_z, u, v, w, rhs
do e=1,num_elem ! loop through all elements
   do i=1,num_points_e ! loop through all points of
     the element e
      ... ! compute derivatives rho_x, rho_y, rho_z
      rhs = u*rho_x + v*rho_y + w*rho_z + ...
   end do !i
end do !e
```

9.4s
14.4% vector
operations

optimised for compiler vectorisation:

```
real, dimension(num_points_e) :: rho, rho_x, rho_y, &
   rho_z, u, v, w, rhs
do e=1,num_elem ! loop through all elements
   ... ! compute derivatives like rho_x, rho_y, rho_z
   rhs = u*rho_x + v*rho_y + w*rho_z + ...
end do !e
```

2.1s
73.9% vector
operations

measurements: spectral element model NUMA, NPS

ESCAPE 2

Funded by the European Union

vector intrinsics (here for BG/Q)

```
real, dimension(4,4,4) :: rho, rho_x, rho_y, &
   rho_z, u, v, w, u_x, v_y, w_z, rhs
!IBM* align(32, rho, rho_x, rho_y, rho_z, u, v, w,
     u_x, v_y, w_z, rhs)
! declare variables representing registers: (each
     contains four double precision floating point
     numbers)
vector(real(8)) vct_rho, vct_rhox, vct_rhoy, vct_rhoz
vector(real(8)) vct_u, vct_v, vct_w, vct_rhs
if (iand(loc(rho), z'1F') .ne. 0) stop 'rho is not
     aligned'
... ! check alignment of other variables
do e=1,num_elem ! loop through all elements
   do k=1,4 ! loop over points in z-direction
      do j=1,4 ! loop over points in y-direction
         ... ! compute derivatives rho_x, ...
         ! load always four floating point numbers:
         vct_u = vec_ld(0, u(1,j,k))
         vct_v = vec_ld(0, v(1,j,k))
         vct_w = vec_ld(0, w(1,j,k))
         vct_rhox = vec_ld(0, rho_x(1,j,k))
         vct_rhoy = vec_ld(0, rho_y(1,j,k))
         vct_rhoz = vec_ld(0, rho_z(1,j,k))
         ! rhs = u*rho_x
```

ESCAPE 2

Funded by the European Union

vector intrinsics (here for BG/Q)

```
         ... ! compute derivatives rho_x, ...
         ! load always four floating point numbers:
         vct_u = vec_ld(0, u(1,j,k))
         vct_v = vec_ld(0, v(1,j,k))
         vct_w = vec_ld(0, w(1,j,k))
         vct_rhox = vec_ld(0, rho_x(1,j,k))
         vct_rhoy = vec_ld(0, rho_y(1,j,k))
         vct_rhoz = vec_ld(0, rho_z(1,j,k))
         ! rhs = u*rho_x
         vct_rhs = vec_mul(vct_u,vct_rhox)
         ! rhs = rhs + v*rho_y
         vct_rhs = vec_madd(vct_v,vct_rhoy,vct_rhs)
         ! rhs = rhs + w*rho_z
         vct_rhs = vec_madd(vct_w,vct_rhoz,vct_rhs)
         ! write result from register into cache:
         call vec_st(vct_rhs, 0, rhs(1,j,k))
         ...
      end do !j
   end do !k
end do !e
```

1.0s
98.6% vector operations

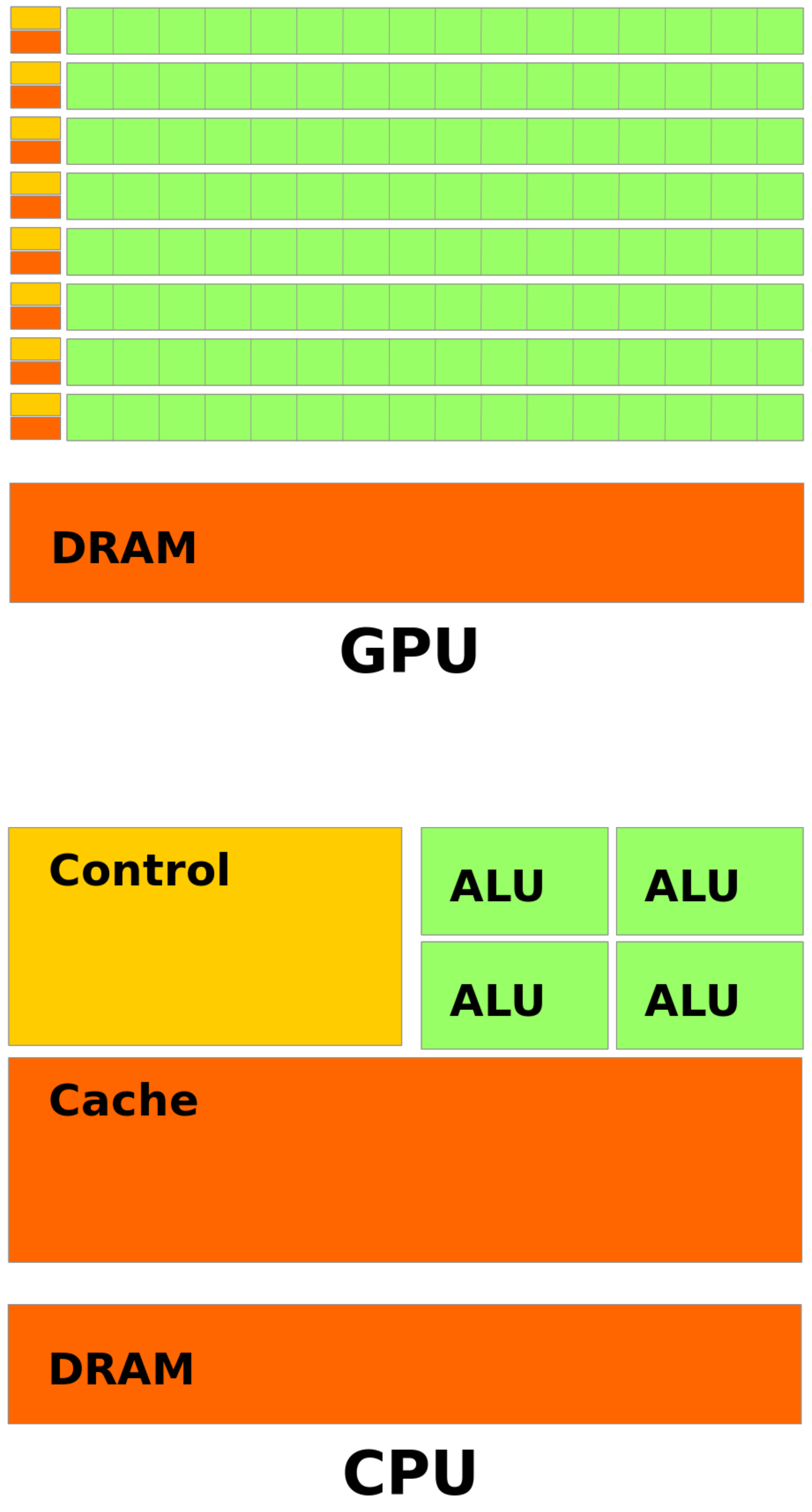
measurements: spectral element model NUMA, NPS

ESCAPE 2

Funded by the European Union

GPU (Graphics Processing Unit)

- small number of instructions => requires host CPU
- GPU/CPU interface (PCIe up to 16GB/sec, NVLINK up to 300GB/sec between GPUs in same node)
- more energy efficient than CPUs
- high performance GPUs today mainly supplied by NVIDIA
- lots of cores share one control unit
- very little memory inside the GPU



Recommendations

- try if using libraries is fast enough
- try to use compiler optimisation (be careful!)
- avoid unnecessary computation and communication
- give each thread as much work as possible
- let the threads do work that does not affect others
- overlap computation and communication
- use data only once per time-step
- contiguous memory access
- try to fit data into cache
- **make good use of vectorisation**

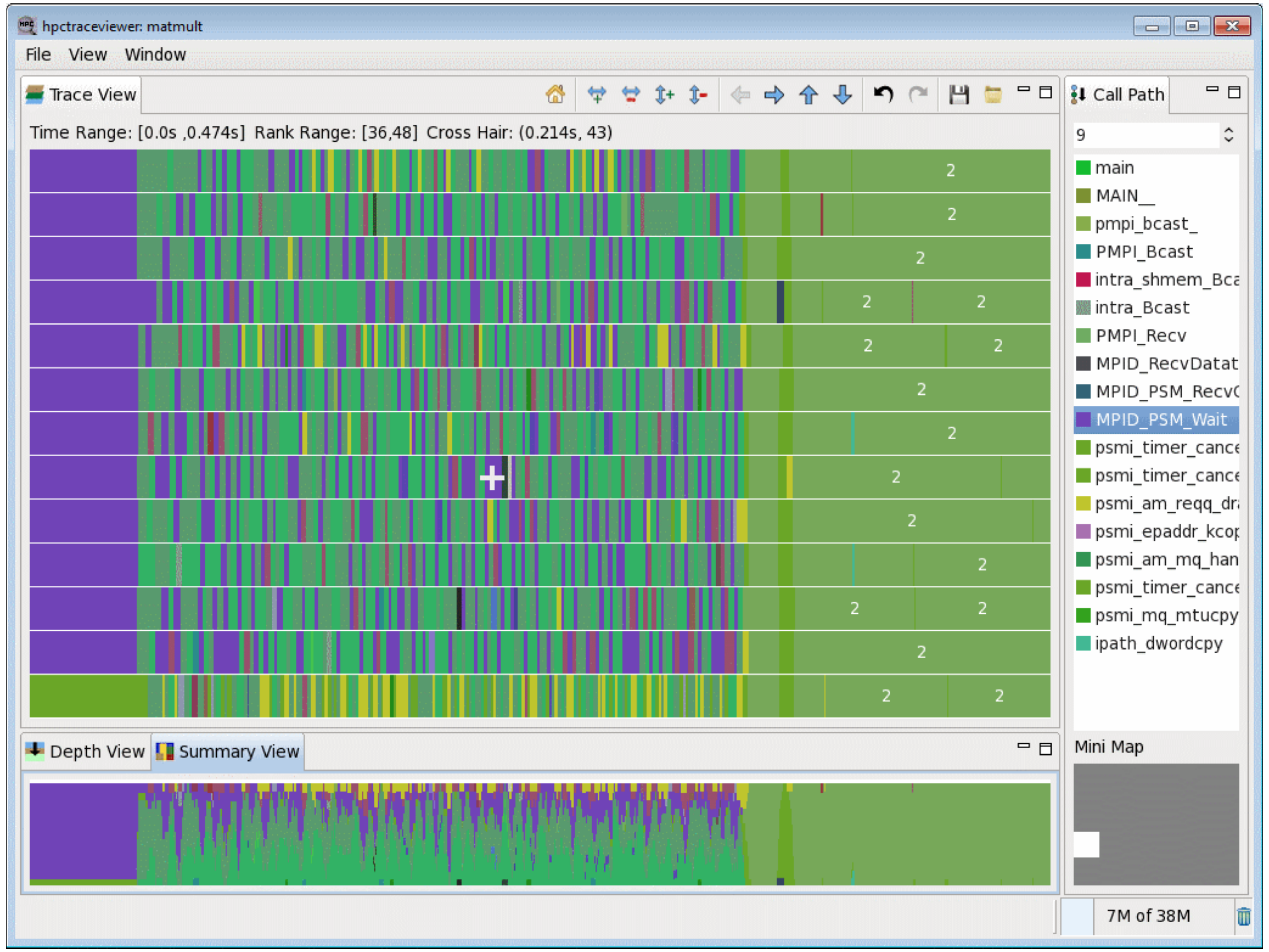
How good are we?

The ESCAPE-2 project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 800897

Funded by the European Union

Hardware performance counters

- set of special-purpose hardware registers to store counts of hardware-related activities
- can help in spotting the application bottlenecks
- allow for low-level performance analysis and tuning, though implementation may be somehow difficult
- tools: PAPI, VTUNE, HPCToolkit, ...



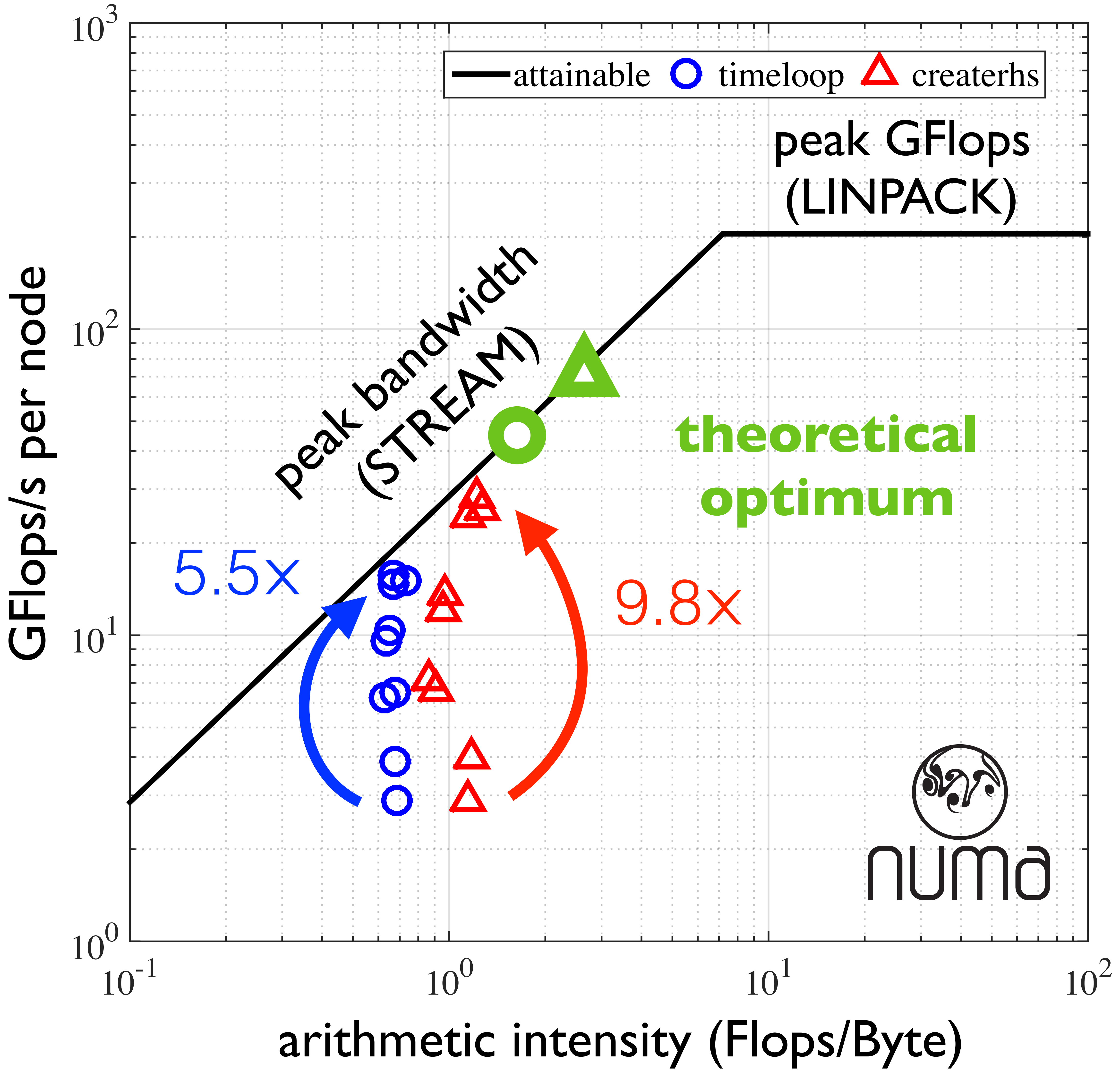
Time Range: [0.0s ,0.474s] Rank Range: [36,48] Cross Hair: (0.214s, 43)

main
MAIN__
pmpi_bcast_
PMPI_Bcast
intra_shmem_Bca
intra_Bcast
PMPI_Recv
MPID_RecvDatat
MPID_PSM_Recv(
MPID_PSM_Wait
psmi_timer_cance
psmi_timer_cance
psmi_am_reqq_dr
psmi_epaddr_kcop
psmi_am_mq_han
psmi_timer_cance
psmi_mq_mtucpy
ipath_dwordcpy

ESCAPE 2

Funded by the European Union

Roofline plot



blue: entire timeloop

red: main computational kernel

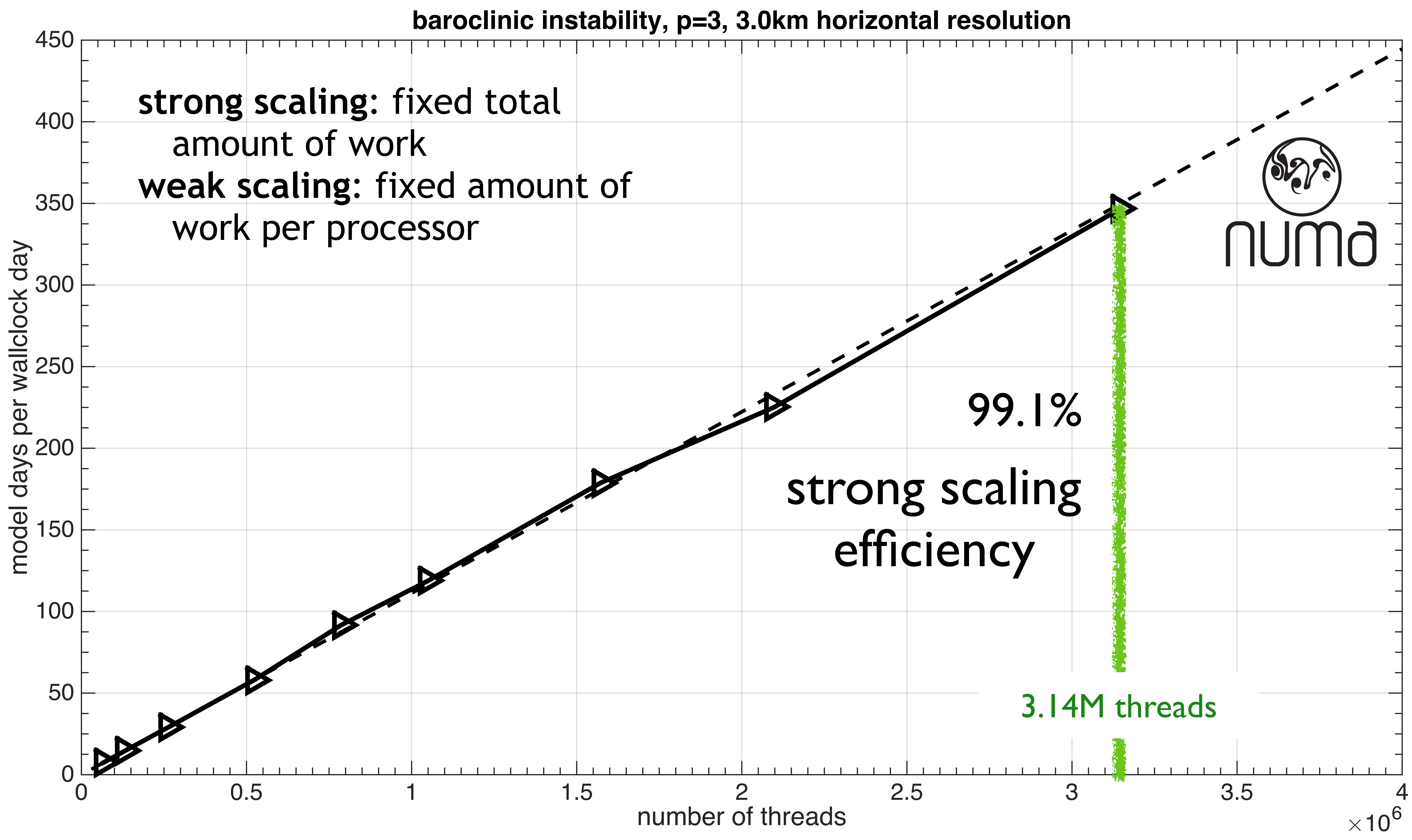
data points: different optimization stages

measurements: spectral element model NUMA, NPS

ESCAPE 2

Funded by the European Union

Strong scaling efficiency



measurements: spectral element model NUMA, NPS

ESCAPE 2

Funded by the European Union

Create performance model

example code:

```
real, dimension(N,M) :: a,b,c
integer :: i,j,N,M
do timestep=1,nstep
 do j=1,M
  do i=1,N
   a(i,j) = a(i,j) + b(i,j) * c(i,j)
  end do
 end do
end do
```

parameters:

parameter	value
N	1E+04
M	1E+05
nstep	100
GB/s	20
GFlops/s	200

floating point operations:

function	operations per step	
main	2*N*M	2E+11
total GFlops for all steps		20000
runtime		100.0

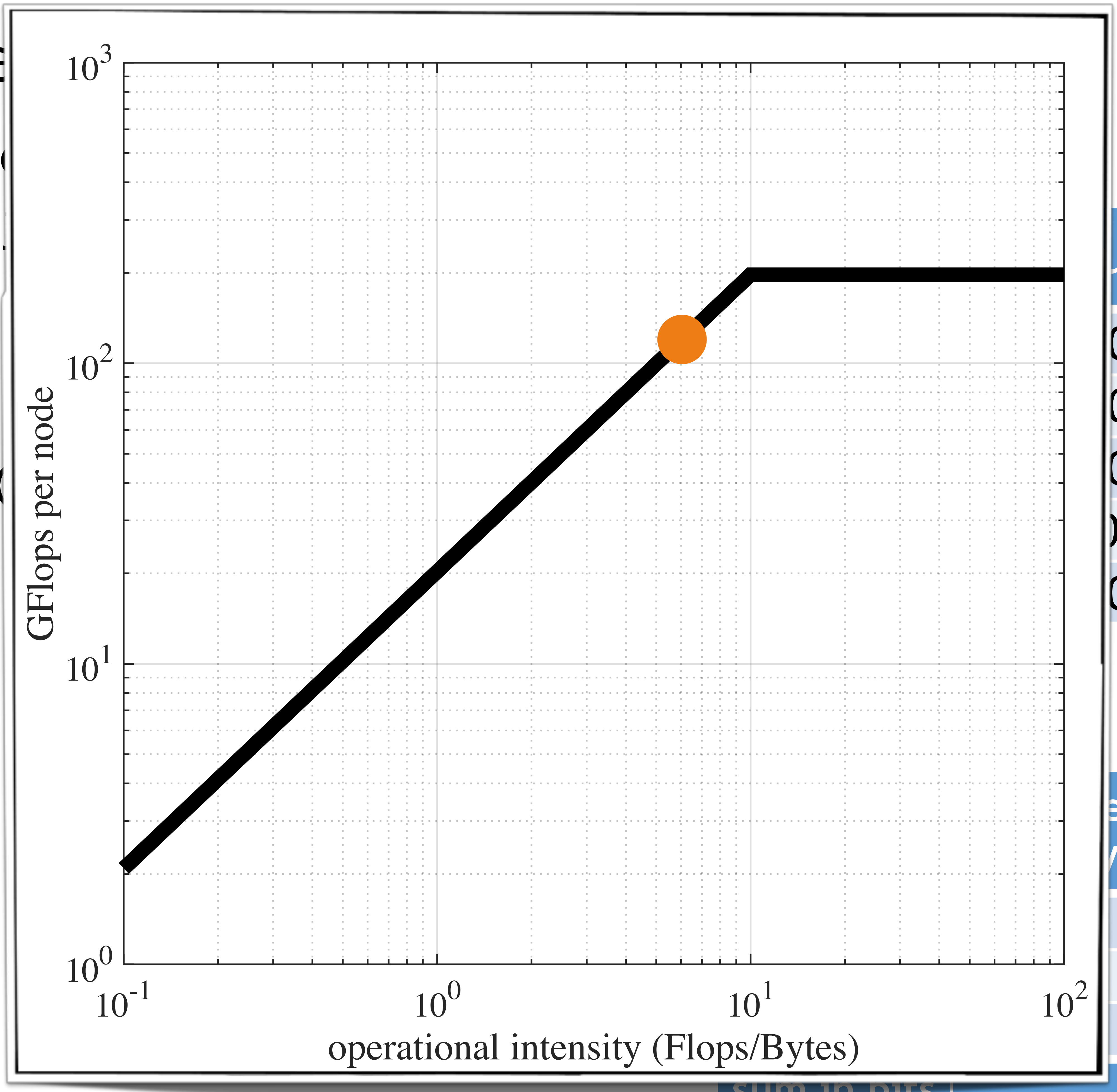
memory:

variable	bits per entry	size	#read per step	#write per step	total bits read	total bits written
a	64	N*M	1	1	6.4E+12	6.4E+12
b	64	N*M	1	0	6.4E+12	0E+00
c	64	N*M	1	0	6.4E+12	0E+00
sum in bits					1.92E+13	6.4E+12
sum in GB					2400	800
intensity	6.25			runtime in seconds		160.0

Create performance model

example code

```
real, dimensi
integer :: i,
do timestep=1
 do j=1,M
  do i=1,N
   a(i,j) = a(
  end do
 end do
end do
```



ue
04
05
0
)
0

floating point operations:

function	operations per step	
main	2*N*M	2E+11
total GFlops for all steps		20000
runtime		100.0

	er /	size	#read per step	#write per step	total bits read	total bits written
		N*M	1	1	6.4E+12	6.4E+12
		N*M	1	0	6.4E+12	0E+00
		N*M	1	0	6.4E+12	0E+00
sum in bits					1.92E+13	6.4E+12
sum in GB					2400	800
intensity	6.25			runtime in seconds		160.0

Funded by the European Union

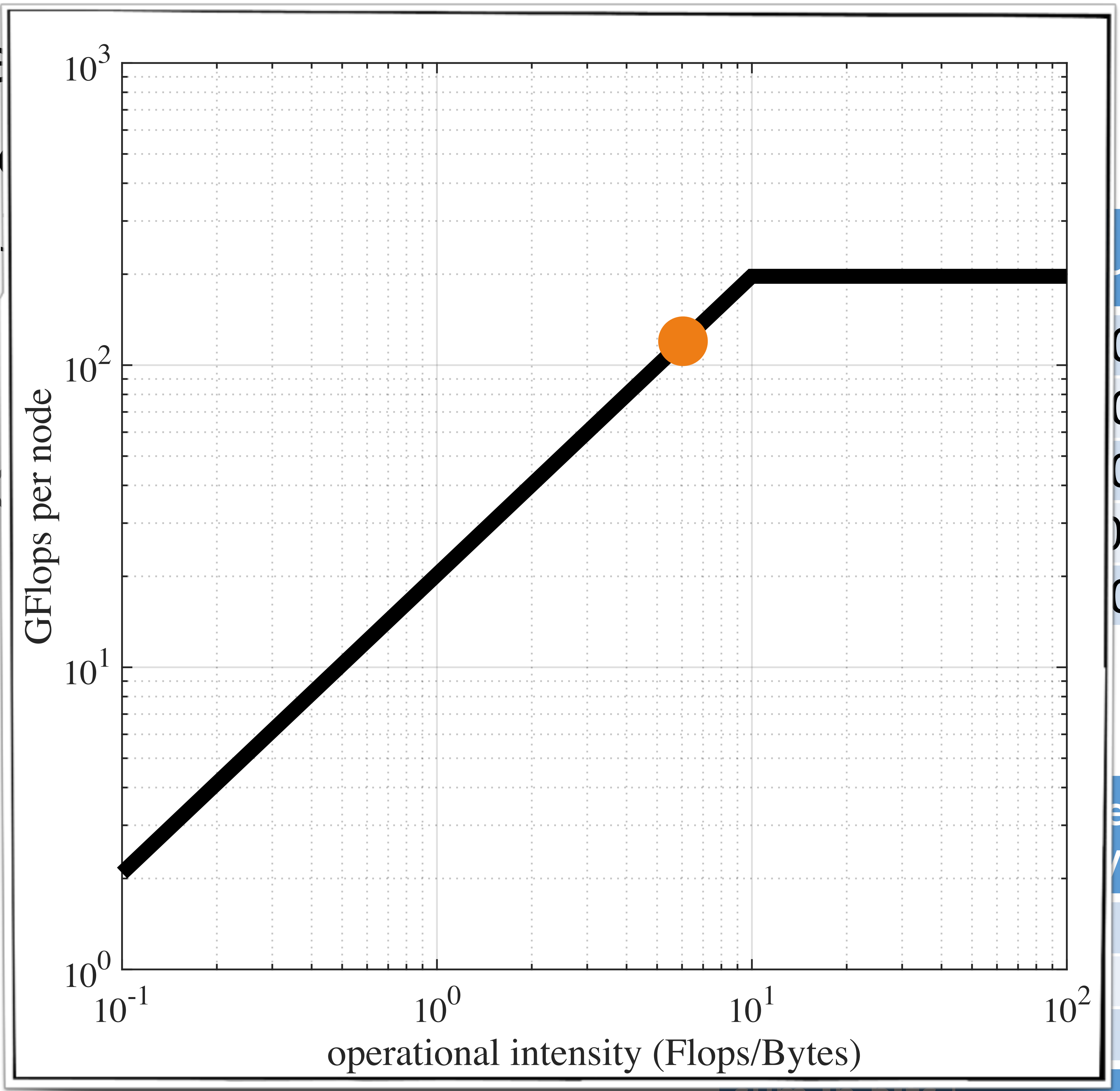
ESCAPE 2

Funded by the European Union

Create performance model

example code

```
real, dimensi
integer :: i,
do timestep=1
 do j=1,M
  do i=1,N
   a(i,j) = a(
  end do
 end do
end do
```



ue
04
05
0
)
0

floating point operations:

function	operations per step	
main	2*N*M	2E+11
total GFlops for all steps		20000
runtime		100.0

	er /	size	#read per step	#write per step	total bits read	total bits written
		N*M	1	1	6.4E+12	6.4E+12
		N*M	1	0	6.4E+12	0E+00
		N*M	1	0	6.4E+12	0E+00
sum in bits					1.92E+13	6.4E+12
sum in GB					2400	800
intensity	6.25			runtime in seconds		160.0

next step: distinguish between worst case (no data already in cache) and best case (previously used data is still in cache)

ESCAPE 2

Funded by the
European Union

Create performance model

example code:

```
real, dimension(N,M) :: a,b,c
integer :: i,j,N,M
do timestep=1,nstep
 do j=1,M
  do i=1,N
   a(i,j) = a(i,j) + b(i,j) * c(i,j)
  end do
 end do
end do
```

parameters:

parameter	value
N	1E+04
M	1E+05
nstep	100
GB/s	20
GFlops/s	200

floating point operations:

function	operations per step	
main	2*N*M	2E+11
total GFlops for all steps		20000
runtime		100.0

memory:

variable	bits per entry	size	#read per step	#write per step	total bits read	total bits written
a	64	N*M	1	1	6.4E+12	6.4E+12
b	64	N*M	0	0	0E+00	0E+00
c	64	N*M	0	0	0E+00	0E+00
sum in bits					6.4E+12	6.4E+12
sum in GB					800	800
intensity	12.5			runtime in seconds		80.0

next step: distinguish between worst case (no data already in cache) and best case (previously used data is still in cache)

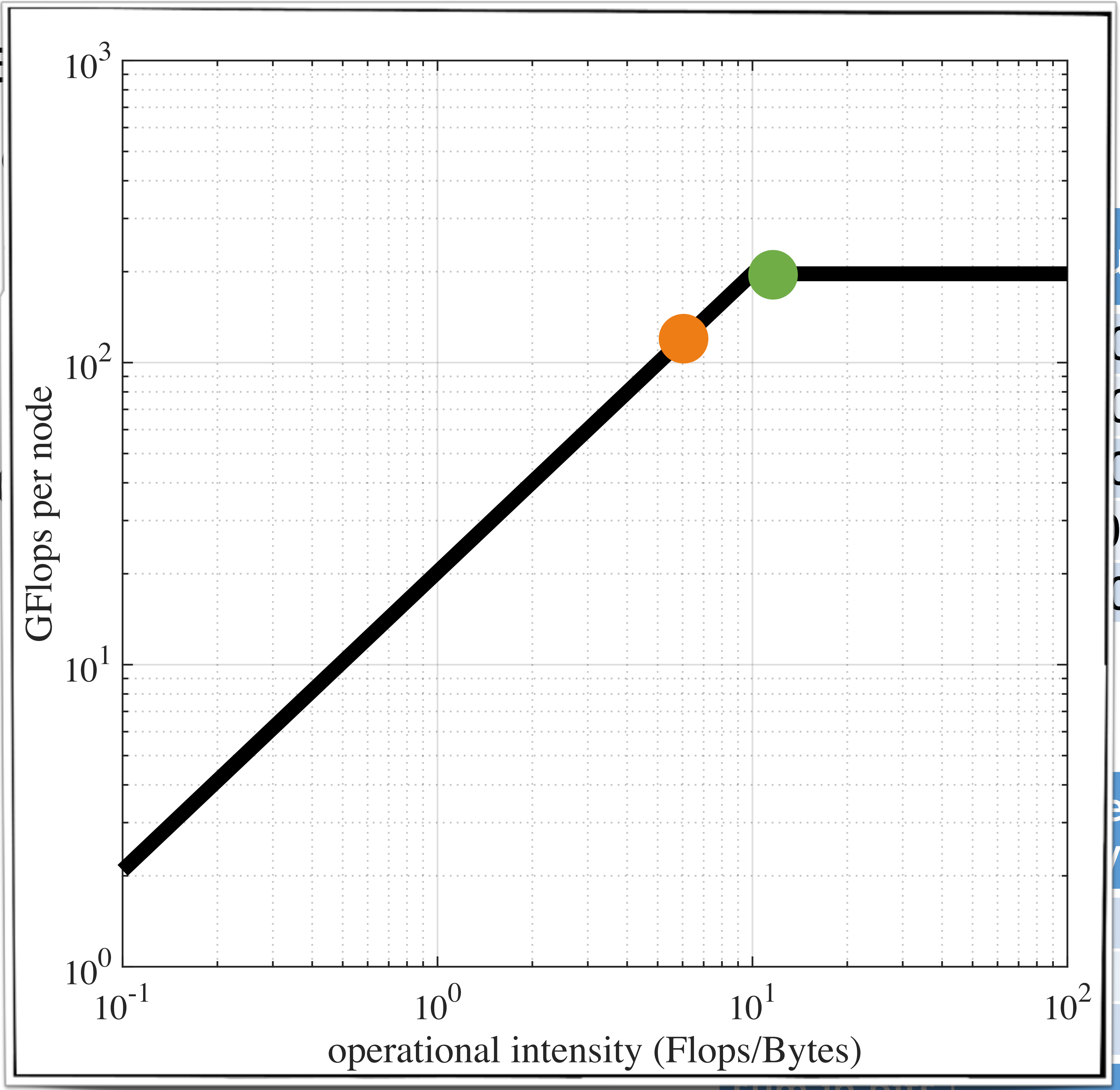
ESCAPE 2

Funded by the European Union

Create performance model

example code

```
real, dimensi
integer :: i,
do timestep=1
 do j=1,M
  do i=1,N
   a(i,j) = a(
  end do
 end do
end do
```



ue
04
05
0
)
0

floating point operations:

function	operations per step	
main	2*N*M	2E+11
total GFlops for all steps		20000
runtime		100.0

er /	size	#read per step	#write per step	total bits read	total bits written
	N*M	1	1	6.4E+12	6.4E+12
	N*M	0	0	0E+00	0E+00
	N*M	0	0	0E+00	0E+00
sum in bits				6.4E+12	6.4E+12
sum in GB				800	800
intensity	12.5		runtime in seconds		80.0

next step: distinguish between worst case (no data already in cache) and best case (previously used data is still in cache)

Funded by the
European Union

Co-ordinated by ECMWF

Recommendations

- try if using libraries is fast enough
- try to use compiler optimisation (be careful!)
- avoid unnecessary computation and communication
- give each thread as much work as possible
- let the threads do work that does not affect others
- overlap computation and communication
- use data only once per time-step
- contiguous memory access
- try to fit data into cache
- make good use of vectorisation
- **compare performance with expectations**

Funded by the European Union

Co-ordinated by ECMWF

Recommendations

- try if using libraries is fast enough
- try to use compiler optimisation (be careful!)
- avoid unnecessary computation and communication
- give each thread as much work as possible
- let the threads do work that does not affect others
- overlap computation and communication
- use data only once per time-step
- contiguous memory access
- try to fit data into cache
- make good use of vectorisation
- compare performance with expectations

open question

How to find right compromise between performance and readability, portability, maintainability?

Funded by the European Union

- Images on slide 2 used under license from shutterstock.com