C2115
Praktický úvod do superpočítání

XIII. lekce

Petr Kulhánek

kulhanek@chemi.muni.cz

Národní centrum pro výzkum biomolekul, Přírodovědecká fakulta,
Masarykova univerzita, Kotlářská 2, CZ-61137 Brno

Obsah

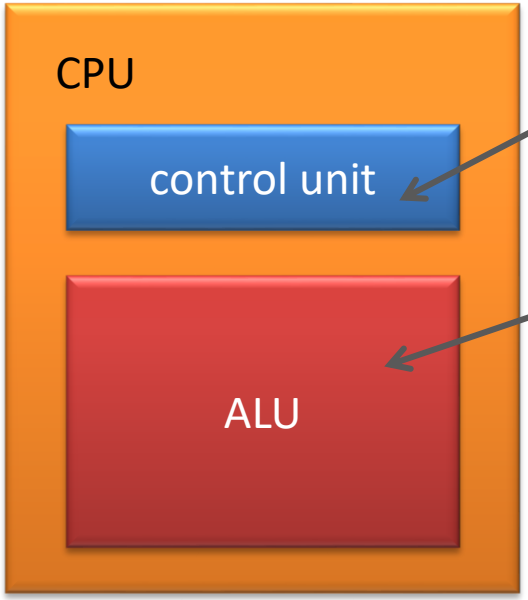
- **Zvyšování výkonu superpočítačů**
  SMP, multicore CPU, NUMA
- **Paralelizace programů**
  numerická integrace, OpenMP, MPI

Architektura počítače

CPU

Processor (CPU - **Central Processing Unit**) is an essential part of the computer; it is a very complex circuit that (sequentially) executes the machine code stored in the computer's memory. The machine code is composed of the instructions, which are processed by ALU.

www.wikipedia.org

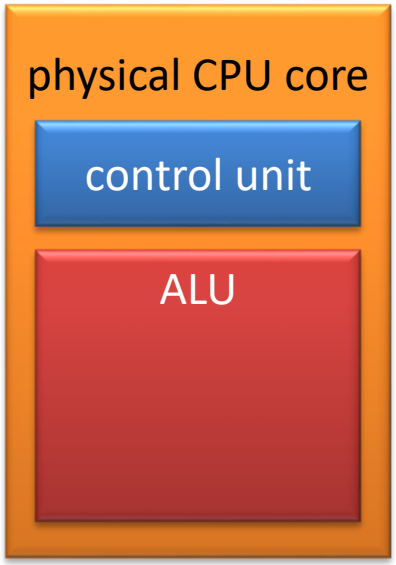


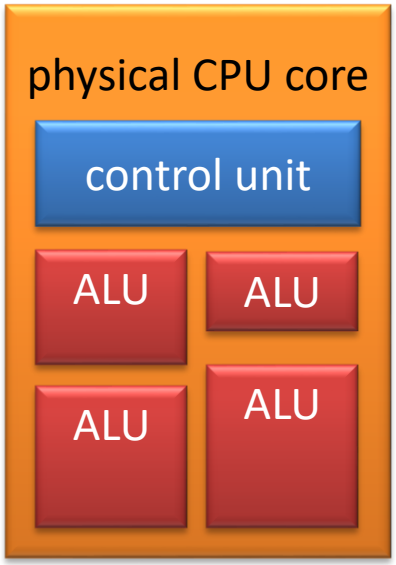
Control unit reads machine code (instructions) and data and prepares them for execution on ALU.

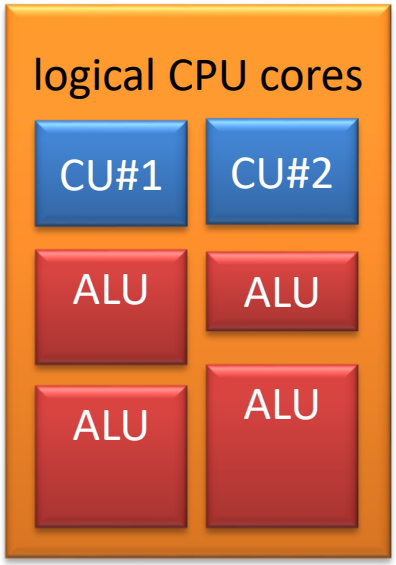
ALU (arithmetic and logic unit) executes arithmetic operations and evaluates logical conditions.

(Sequential) execution of machine code is controlled by internal clock.

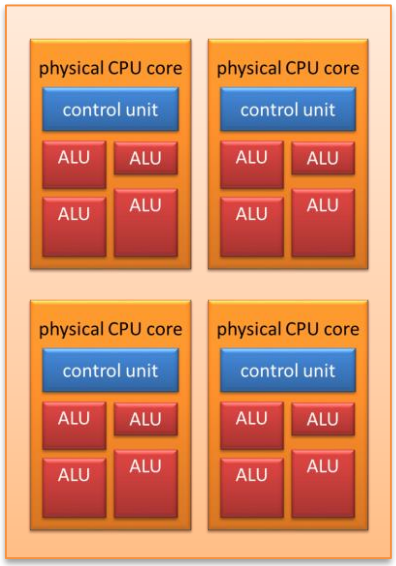
Increasing Computing Power







hyperthreading



multi-core processor

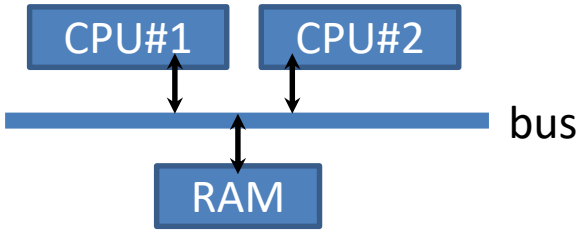
Strategies:

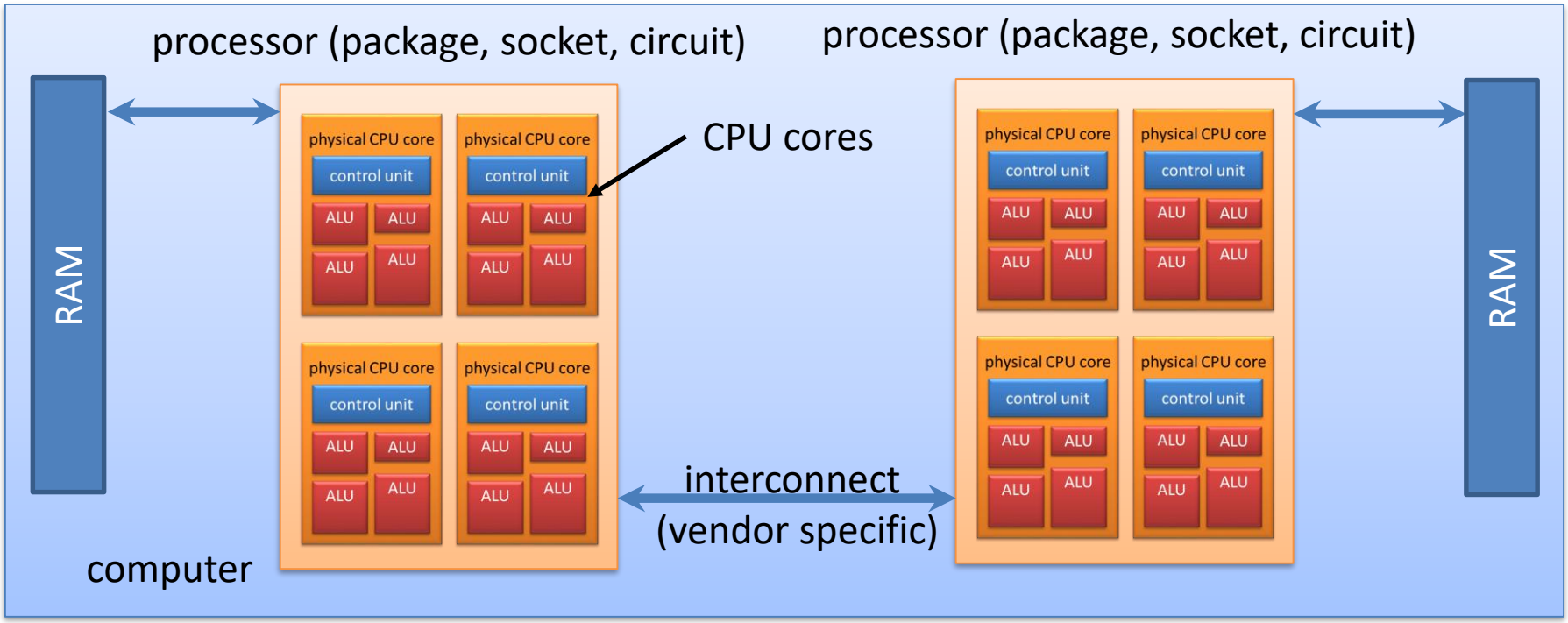
- **increasing clock frequency**
  - – physical limitations (miniaturization, lowering voltage)
- **increasing number of ALUs** and their specializations (out-of-order execution, speculative execution, vector instructions)
  - – efficiency limited by executed code
- **sharing ALUs among control units** (hyperthreading)
  - – efficiency limited by executed code
- **multi-core processor**
  - – efficiency limited by executed code

software optimization
or
new algorithms
are necessary to benefit from these features

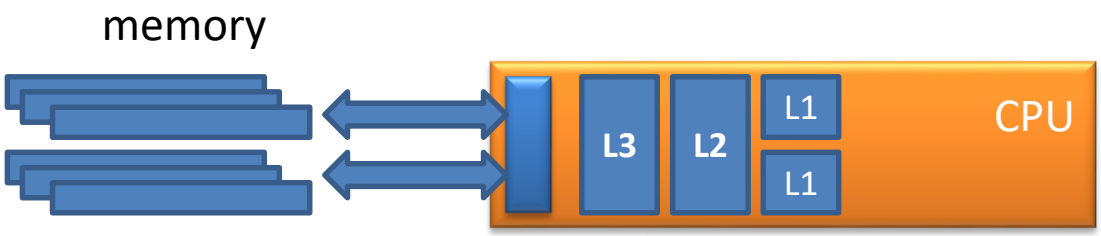
Symmetric Multiprocessing (SMP)

Symmetric multiprocessing represents a system containing identical processors accessing shared memory.
Utilizations of larger number of CPUs **increases computing power** of the system.

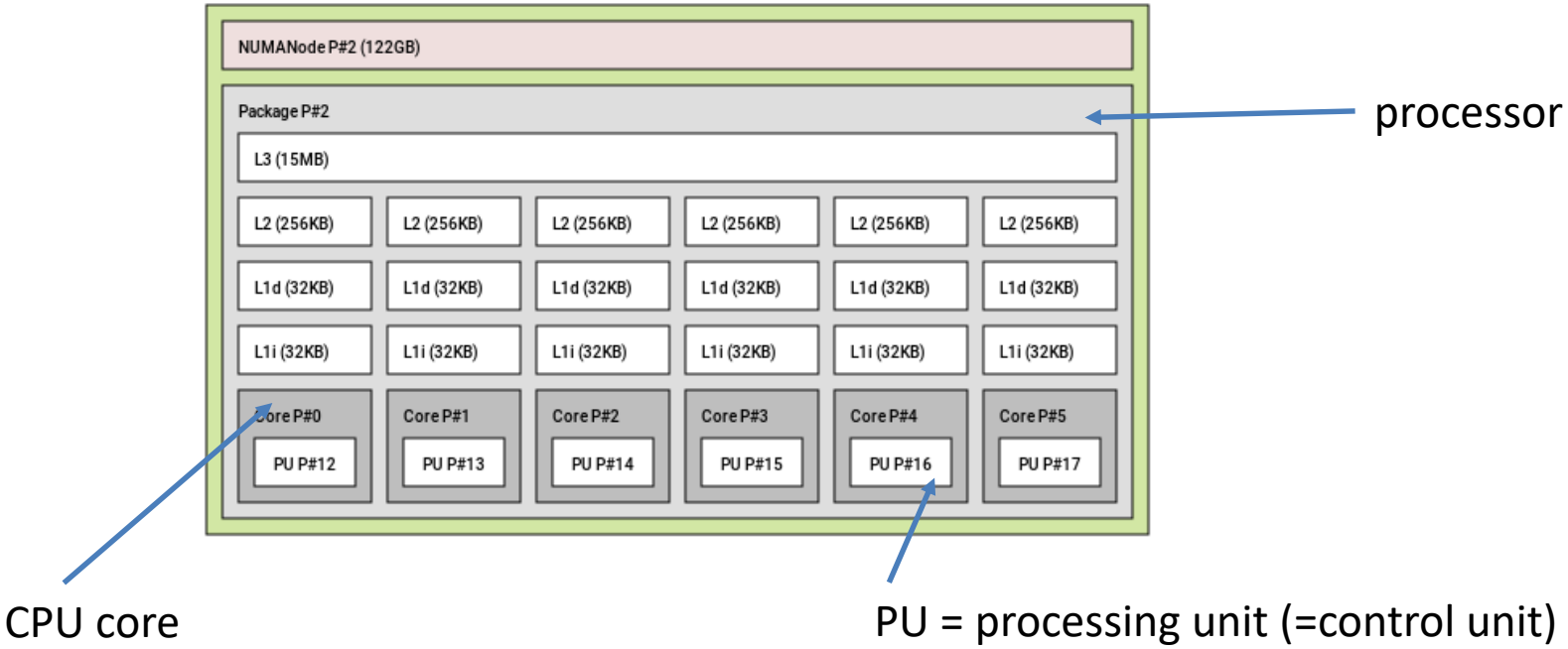




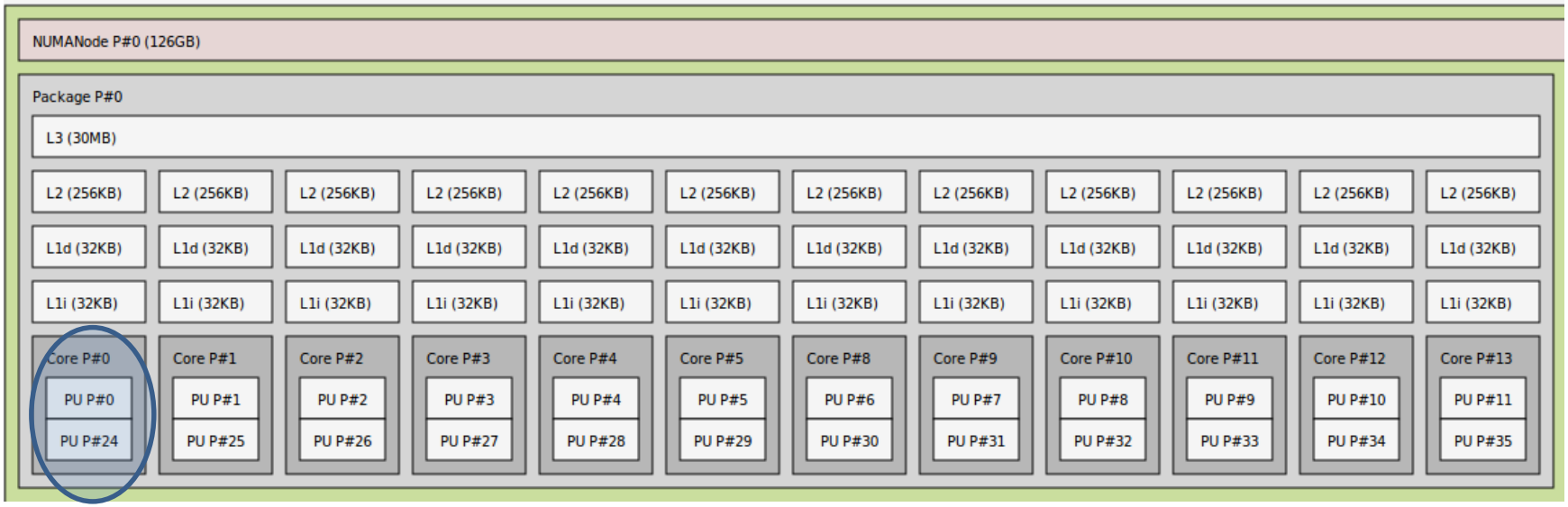
Processor Caches



Processor caches improves access into memory (latency and bandwidth).



Processor Caches

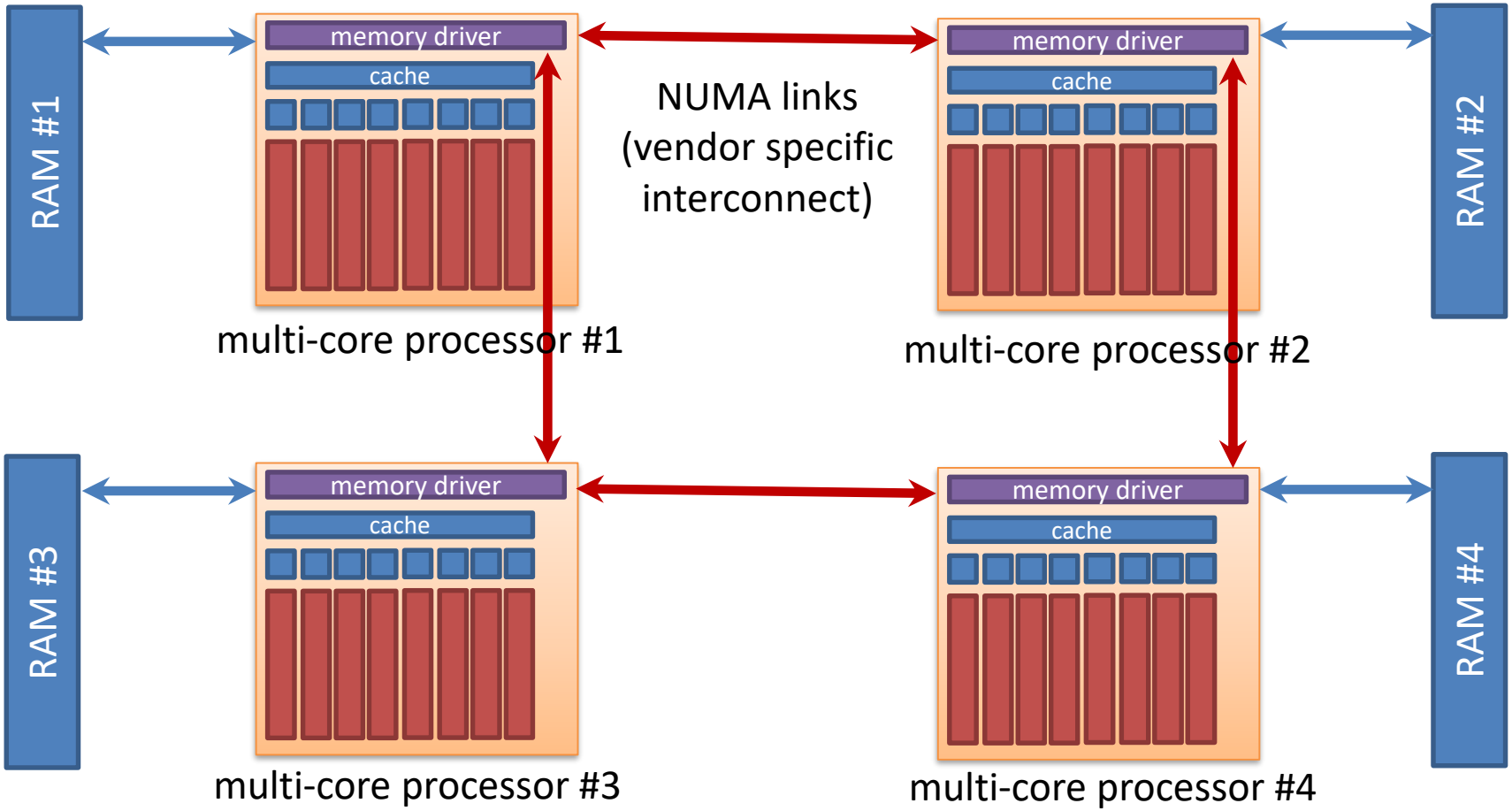


two processing units per core (hyperthreading)

L1i – instruction cache, L1d – data cache
L2, L3 – other caches

Speed:
L1d,L2d >> L2 > L3

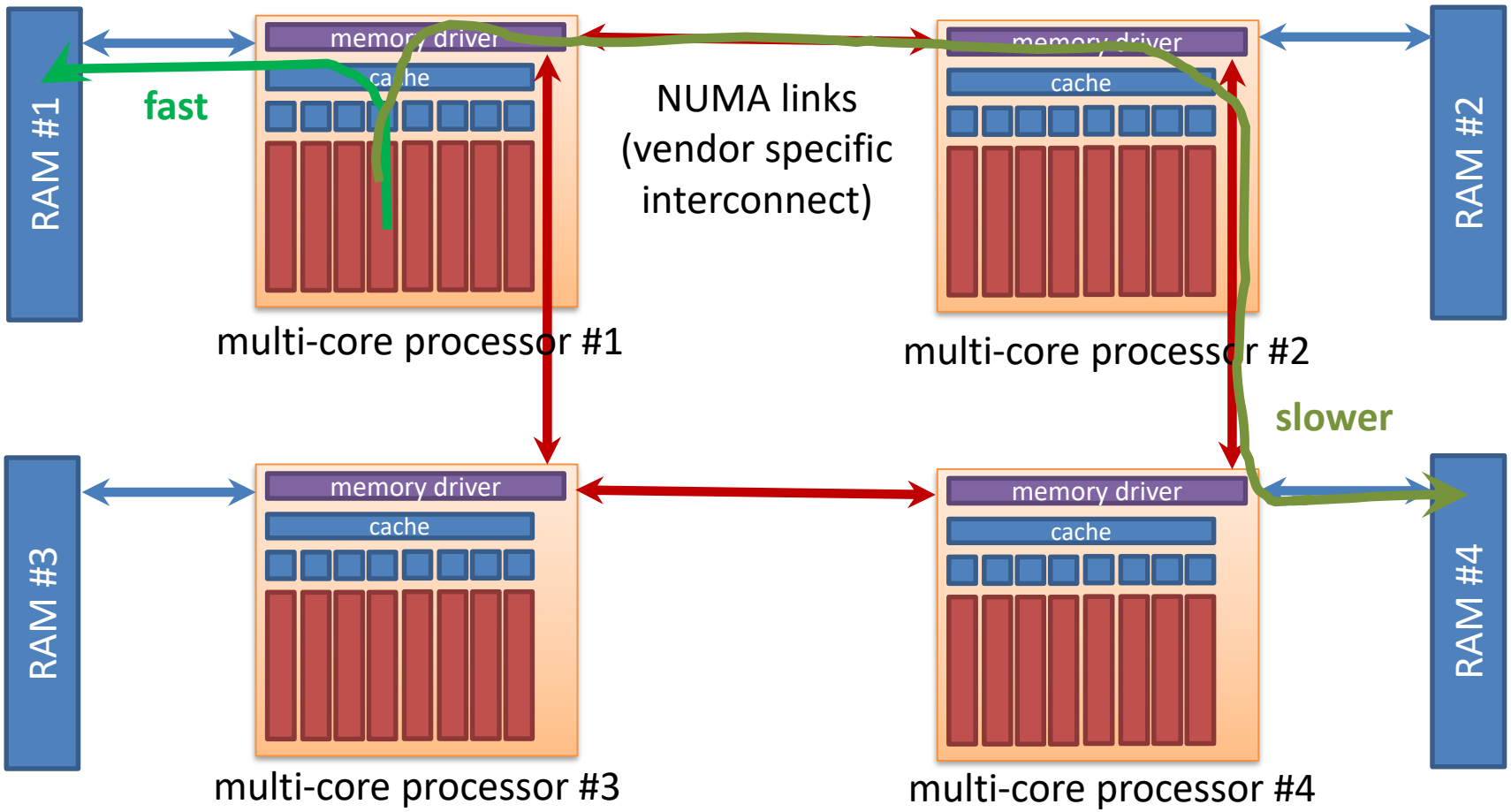
NUMA (Nonuniform Memory Access)



Compare communication between Processor#1 <> RAM#1 and Processor#1 <> RAM#4.

NUMA links can have various topologies to improve memory access and latency.

NUMA (Nonuniform Memory Access)



Compare communication between Processor#1 <> RAM#1 and Processor#1 <> RAM#4.

NUMA links can have various topologies to improve memory access and latency.

Exercise PI.E1

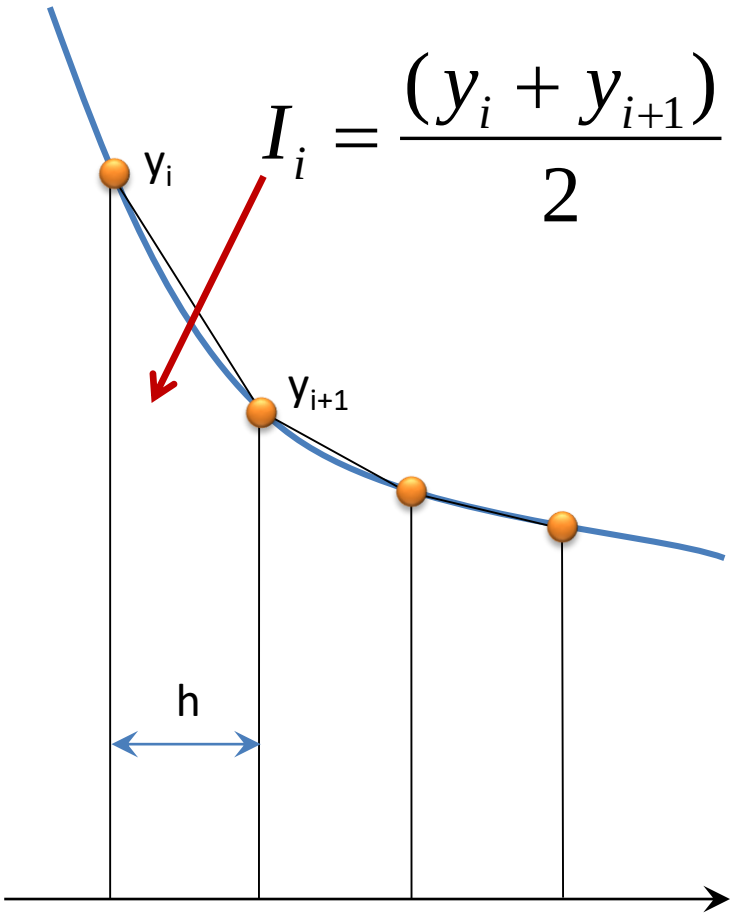
1. Examine type and parameters of processor on your workstation (command lscpu, file /proc/cpus).
2. Examine NUMA topology on your workstation (command lstopo, module hwloc).
3. Does your CPU support hyperthreading?
4. What is a process?
5. What is difference between CPU intensive and data intensive tasks?
6. A parallel task is data intensive. Each its process works with different data sets. What is better for speeding up the calculation?
    1. To double number of CPU cores.
    2. To double number of processors (sockets).

Užitečné příkazy:

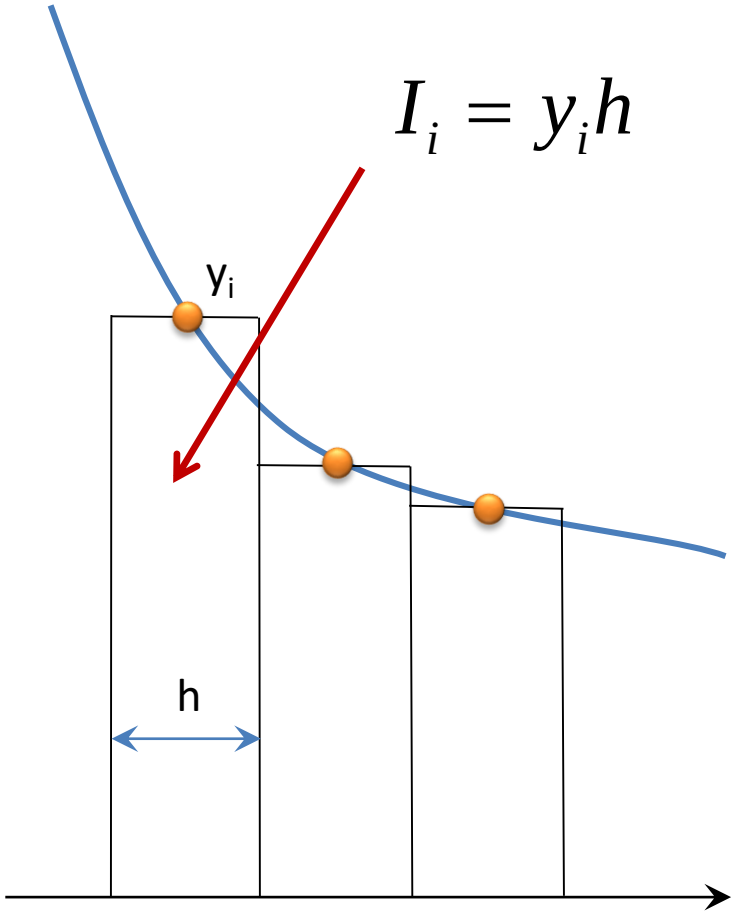
```
$ lscpu
$ lstopo              # module add hwloc
$ cat /proc/cpuinfo
$ ams-host            # Infinity
```

Paralelizace programů

Lichoběžníková vs obdélníková metoda



lichoběžníková metoda



obdélníková metoda

Která metoda je vhodnější pro paralelní výpočet?

Sekvenční implementace

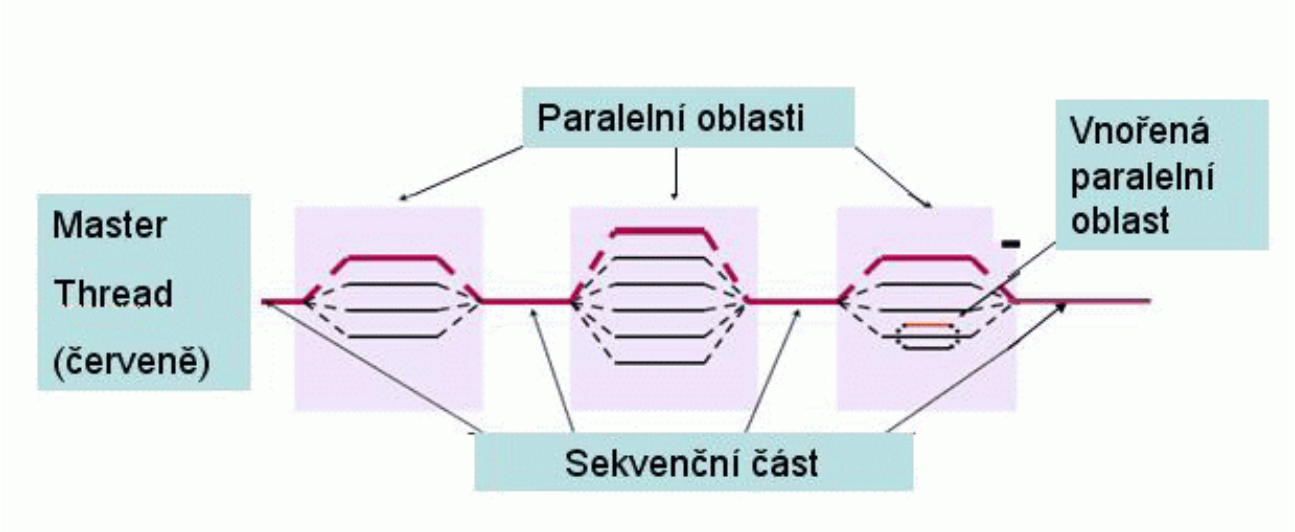
obdélníková metoda

```
program integral

  implicit none
  integer(8)                :: i
  integer(8)                :: n
  double precision          :: h,v,y,x
  !-------------------------------------------------
  n = 2000000000
  h = 1.0d0/n
  v = 0.0d0

  do i=1,n
    x = (i-0.5d0)*h
    y = 4.0d0/(1.0d0+x**2)
    v = v + y*h
  end do

  write(*,*) 'integral = ',v

end program integral
```

Paralelizace - OpenMP

OpenMP je soustava **direktiv** pro překladač a knihovních procedur pro paralelní programování. Jedná se o standard pro programování počítačů se sdílenou pamětí. OpenMP usnadňuje vytváření vícevláknových programů v programovacích jazycích Fortran, C a C++.

www.wikipedia.org



Specifikace: **www.openmp.org**

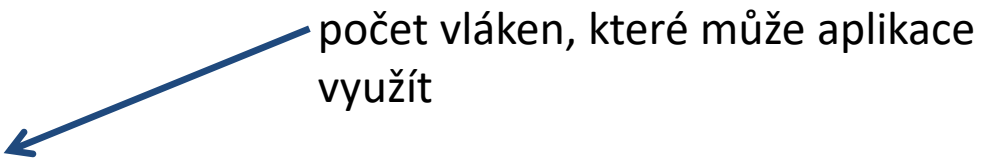
OpenMP implementace

```
ncpu = 1
!$ ncpu = omp_get_max_threads()
write(*,*) 'Number of threads = ',ncpu

!$omp parallel

!$omp do private(i,x,y),reduction(+:v)
do i=1,n
  x = (i-0.5d0)*h
  y = 4.0d0/(1.0d0+x**2)
  v = v + y*d
end do
!$omp end do

!$omp end parallel
write(*,*) 'integral = ',v
```

OpenMP kompilace

```
$ gfortran -O3  integral.f90 -o integral
$ ldd ./integral
        linux-vdso.so.1 =>
        libgfortran.so.3 => /usr/lib/x86_64-linux-gnu/libgfortran.so.3
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
        libquadmath.so.0 => /usr/lib/x86_64-linux-gnu/libquadmath.so.0
        libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6
        /lib64/ld-linux-x86-64.so.2

$ gfortran -O3 -fopenmp integral.f90 -o integral
$ ldd ./integral
        linux-vdso.so.1 =>  (0x00007fff593ff000)
        libgfortran.so.3 => /usr/lib/x86_64-linux-gnu/libgfortran.so.3
        libgomp.so.1 => /usr/lib/x86_64-linux-gnu/libgomp.so.1
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
        libquadmath.so.0 => /usr/lib/x86_64-linux-gnu/libquadmath.so.0
        libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6
        librt.so.1 => /lib/x86_64-linux-gnu/librt.so.1
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0
        /lib64/ld-linux-x86-64.so.2
```

OpenMP spuštění




```
$ export OMP_NUM_THREADS=4
$ ./integral
 Number of threads =                    4
 integral =    3.1415925965295672
```

Poznámka: pokud není proměnná OMP_NUM_THREADS nastavena, použije se maximální počet dostupných CPU jader
(na klastru WOLF je však výchozí hodnota proměnné OMP_NUM_THREADS explicitně nastavena na 1)

Cvičení 2

1. Zkompilujte program **integral.f90** z adresáře **/home/kulhanek/Data/C2115/Lesson12/integral/openmp** s optimalizací **-O3** bez podpory OpenMP.
2. Určete dobu běhu aplikace potřebnou pro integraci funkce. K měření doby použijte program **/usr/bin/time**.
3. Zkompilujte program **integral.f90** s optimalizací **-O3** a zapnutou podporou OpenMP.
4. Určete počet CPU jader na vašem počítači (lscpu).
5. Spouštějte program **integral** postupně pro 1, 2, 3, až N vláken, kde N je maximální dostupný počet CPU jader. Pro každé spuštění určete dobu běhu. Získaná data zapisujte do následující tabulky a vyhodnoťte.

N	T_{real} [s]	Speedup	CPU effectivity [%]
1	27.8	1.0	100.0
2	14.7	1.9	94.8
3	11.0	2.5	84.1
4	8.2	3.4	84.7

naměřený čas

$$Speedup = \frac{T_{real}(N=1)}{T_{real}}$$

$$CPUeffectivity = \frac{Speedup}{N} 100$$

Cvičení 3

1. Co je to Amhdalův zákon?

Paralelizace - MPI

Message Passing Interface (dále jen MPI) je knihovna implementující stejnojmennou specifikaci (protokol) pro podporu paralelního řešení výpočetních problémů v počítačových clusterech. Konkrétně se jedná o rozhraní pro vývoj aplikací (API) založené na zasílání zpráv mezi jednotlivými uzly. Jedná se jak o zprávy typu point-to-point, tak o globální operace. Knihovna podporuje jak architektury se sdílenou pamětí, tak s pamětí distribuovanou.

MPI implementace

MPI

výhody x nevýhody

MPI kompilace

```
$ module add openmpi:2.0.3-gcc-5.4.0
$ mpif90 -O3  integral.f90 -o integral
```

MPI spuštění

počet procesů, které aplikace využívá k výpočtu

```
$ mpirun -np 2 ./integral
```

```
$ mpirun -np 2 -machinefile nodes ./integral
```

soubor, který obsahuje seznam uzlů, na kterých se spouští procesy

```
wolf01 slots=2
wolf02 slots=3
```

počet CPU

název výpočetního uzlu

Předpoklady:

- ssh bez hesla
- aplikace musí být ve stejné cestě na všech uzlech, na kterých se spouští procesy

Cvičení 4

1. Zkompilujte program **integral.f90** z adresáře **/home/kulhanek/Data/C2115/Lesson12/integral/mpi** s optimalizací **-O3**.
2. Spouštějte program **integral** postupně pro 1, 2, 3, až N procesů, kde N je maximální dostupný počet CPU jader. Pro každé spuštění určete dobu běhu. Získaná data zapisujte do tabulky a vyhodnoťte jako ve cvičení 2
3. Spouštějte program **integral** postupně pro 1, 2, 4, 8 až N procesů (násobky 2), kde N je maximální dostupný počet uzlů na dvou uzlech klastru WOLF. Pro každé spuštění určete dobu běhu. Získaná data zapisujte do tabulky a vyhodnoťte jako ve cvičení 2. Spouštění je nutné koordinovat s uživatelem druhého výpočetního uzlu. V jiném terminálu monitorujte běžící procesy na obou výpočetních uzlech příkazem **top**.

Samostudium

Cvičení 5

1. Zkompilujte program **mult_mat_blas_dp.f90** z adresáře **/home/kulhanek/Data/C2115/Lesson12/matmult** s optimalizací **-O3** proti systémové knihovně blas (program obsahuje jiný způsob měření času oproti LIV.2).
2. Určete dobu běhu aplikace **mult_mat_blas_dp** programem **/usr/bin/time**.
3. Zkompilujte program **mult_mat_blas_dp.f90** s optimalizací **-O3** proti knihovně MKL.

```
gfortran -O3 mult_mat_blas_dp.f90 -o mult_mat_blas_dp \
         -lmkl_gf_lp64 -lmkl_gnu_thread -lmkl_core \
         -lgomp -lpthread \
-L/software/ncbr/softrepo/common/intelcore/2015.0.090/x86_64/para/lib
```

4. Určete dobu běhu aplikace **mult_mat_blas_dp** programem **/usr/bin/time**. Pro spuštění aplikace musíte aktivovat modul **intelcore:2015.0.090** Porovnejte dobu běhu mezi MKL a systémovou knihovnou blas. Která knihovna poskytuje větší výpočetní výkon?
5. Spouštějte program **mult_mat_blas_dp** postupně pro 1, 2, 3, až N vláken, kde N je maximální dostupný počet CPU jader. Počet vláken pro MKL knihovnu se nastavuje pomocí systémové proměnné **MKL_NUM_THREADS**. Pro každé spuštění určete dobu běhu. Získaná data zapisujte do tabulky a vyhodnoťte jako ve cvičení 2.