Introduction to parallel computations

EURO^2

Introductory HPC training for industrial users

12.10.2023, Sofia, Bulgaria

Todor Gurov, NCC Bulgaria, IICT-BAS (Тодор Гюров, НКЦ България, ИИКТ-БАН)

HPC/Parallel Systems - Examples

HOME | LISTS | STATISTICS | RESOURCES | ABOUT | MEDIA KIT

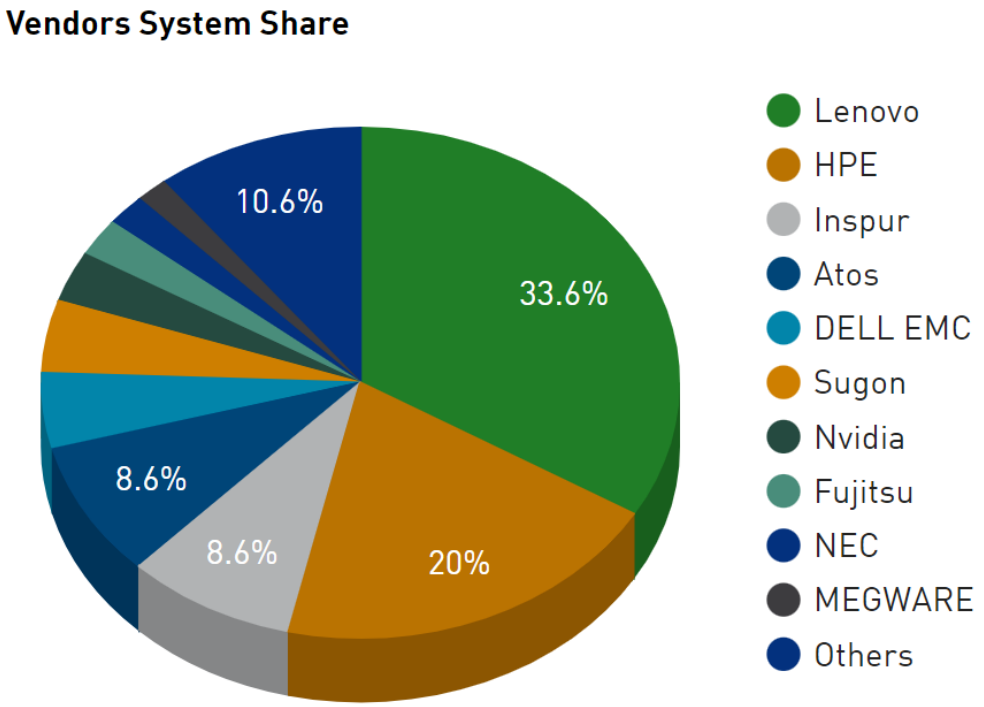
Frontier Remains As Sole Exaflop Machine And Retains Top Spot, Improving Upon Its Previous HPL Score

- May 22, 2023
- The 61st edition of the TOP500 reveals that the Frontier system out of Oak Ridge National Laboratory (ORNL) remains the only true exascale machine on the list.
- Increasing its HPL score from 1.02 Eflop/s in November 2022 to an impressive 1.194 Eflop/s on this list, Frontier was able to improve upon its score after a stagnation between June 2022 and November 2022.

Rank	Place/Country	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	DOE/SC/Oak Ridge National Laboratory **United States**	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE	8,699,904	1,194.00	1,679.82	22,703
2	Fujitsu RIKEN Center for Computational Science **Japan**	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D	7,630,848	442.01	537.21	29,899
3	EuroHPC/CSC **Finland**	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE	2,220,288	309.10	428.70	6,016
4	EuroHPC/ CINECA **Italy**	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 InfiniBand, Atos	1,824,768	238.70	304.47	7,404
5	DOE/SC/Oak Ridge National Laboratory **United States**	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR InfiniBand, IBM	2,414,592	148.60	200.79	10,096
6	DOE/NNSA/LLNL **United States**	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR InfiniBand, IBM / NVIDIA / Mellanox	1,572,480	94.64	125.71	7,438
7	National Super-computing Center in Wuxi **China**	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC	10,649,600	93.01	125.44	15,371
8	DOE/SC/LBNL/NERSC **United States**	**Perlmutter** - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE	761,856	70.87	93.75	2,589
9	NVIDIA Corporation **United States**	**Selene** - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR InfiniBand, NVidia	555,520	63.46	79.22	2,646
10	National Super-Computer Center in Guangzhou **China**	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT	4,981,760	61.44	100.68	18,482

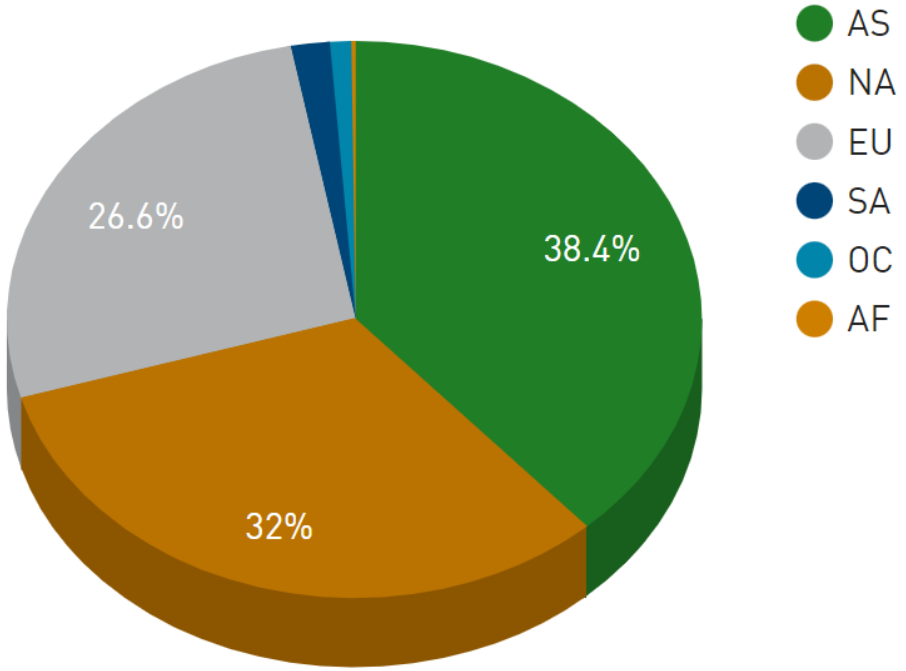
Key players in HPC Systems

	Vendors	Count	System Share (%)	Rmax (GFlops)	Rpeak (GFlops)	Cores
1	Lenovo	168	33.6	532,046,848	1,005,232,220	13,874,272
2	HPE	100	20	2,279,237,514	3,199,996,338	27,601,624
3	Inspur	43	8.6	94,774,410	239,125,103	2,184,960
4	Atos	43	8.6	516,852,490	737,156,702	7,946,016
5	DELL EMC	24	4.8	133,574,600	221,283,871	2,822,460
6	Sugon	23	4.6	49,443,000	138,317,626	2,088,760
7	Nvidia	16	3.2	198,794,000	259,415,712	1,645,600
8	Fujitsu	12	2.4	556,419,520	707,270,456	9,642,976
9	NEC	10	2	50,439,480	73,806,510	577,760
10	MEGWARE	8	1.6	22,044,630	32,286,904	353,320
11	Microsoft Azure	6	1.2	150,370,000	224,746,822	1,328,640
12	IBM	6	1.2	201,915,000	273,679,127	3,292,832
13	Penguin Computing, Inc.	5	1	15,511,780	21,531,420	377,688
14	NUDT	3	0.6	66,081,890	108,454,198	5,342,848
15	ACTION	3	0.6	7,993,980	63,814,099	178,368
16	Huawei Technologies Co., Ltd.	2	0.4	5,872,400	9,389,880	101,184
17	Liqid	2	0.4	5,393,230	7,517,800	102,144
18	IBM / NVIDIA / Mellanox	2	0.4	112,840,000	148,759,200	1,860,768
19	Quanta Computer / Taiwan Fixed Network / ASUS Cloud	2	0.4	11,297,560	19,562,790	220,752
20	YANDEX, NVIDIA	2	0.4	37,550,000	50,051,270	328,352

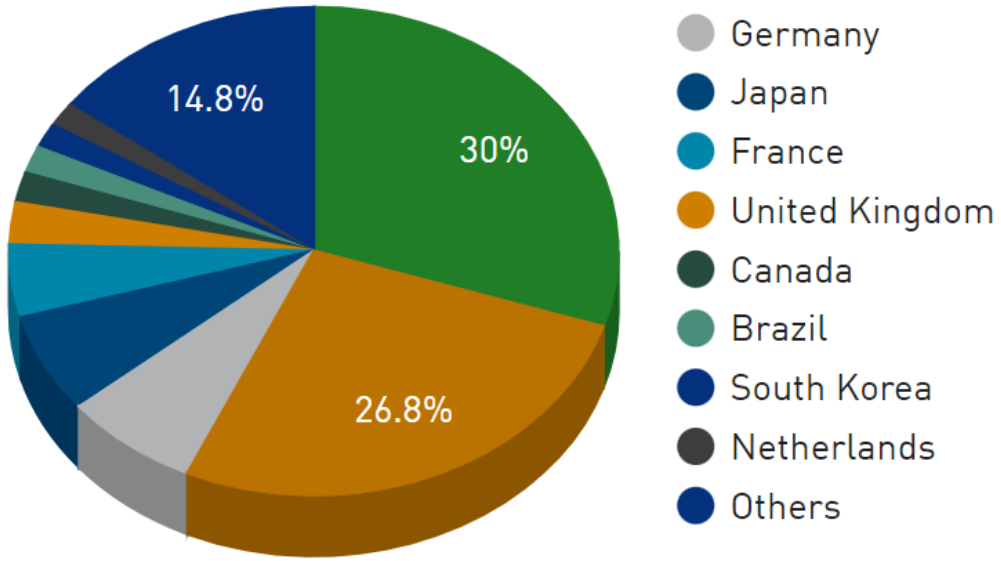


Geo distibution in HPC Systems

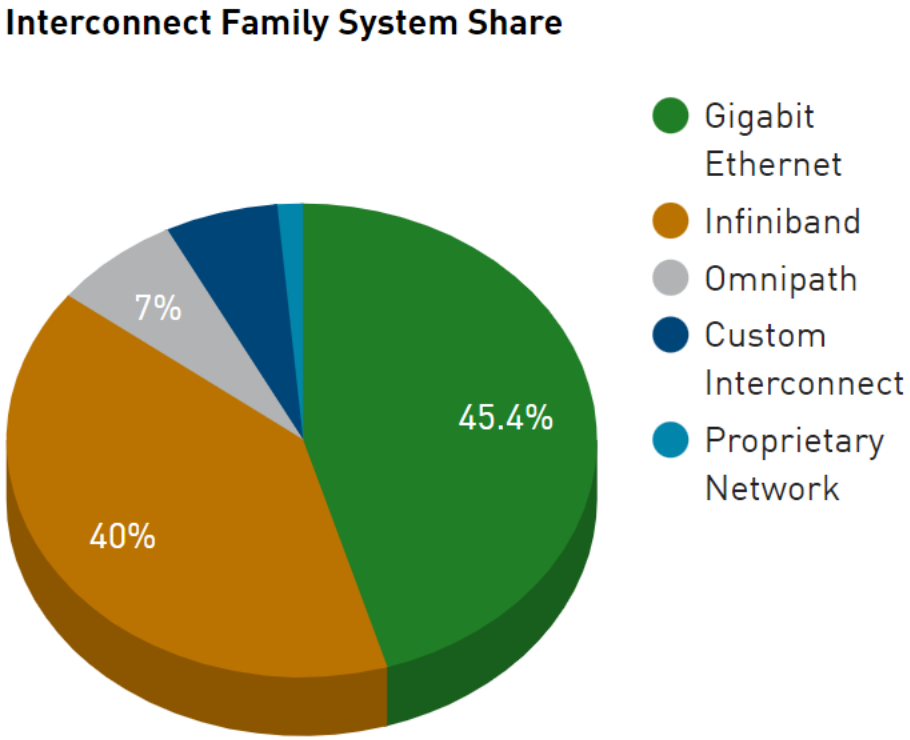
Continents System Share

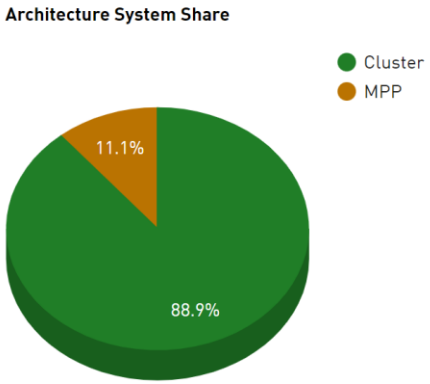


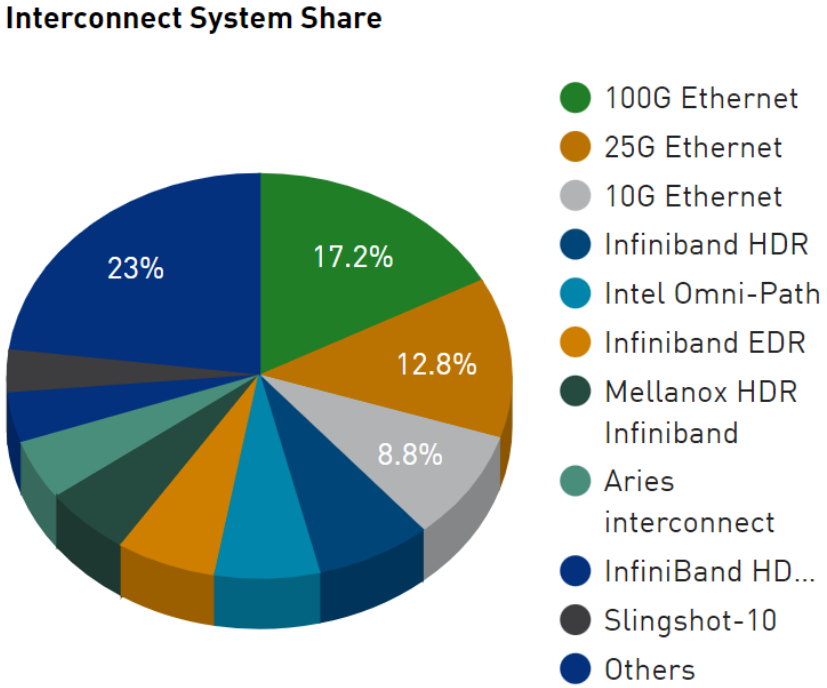
Countries System Share



Interconnect

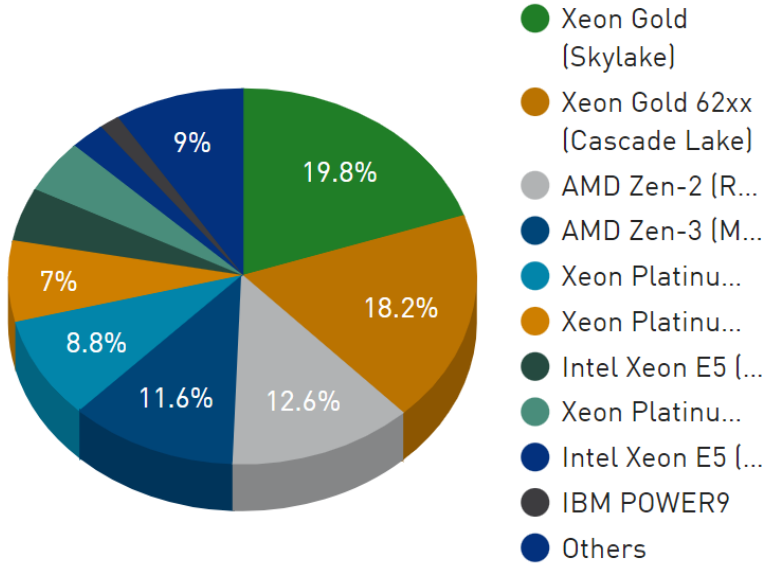




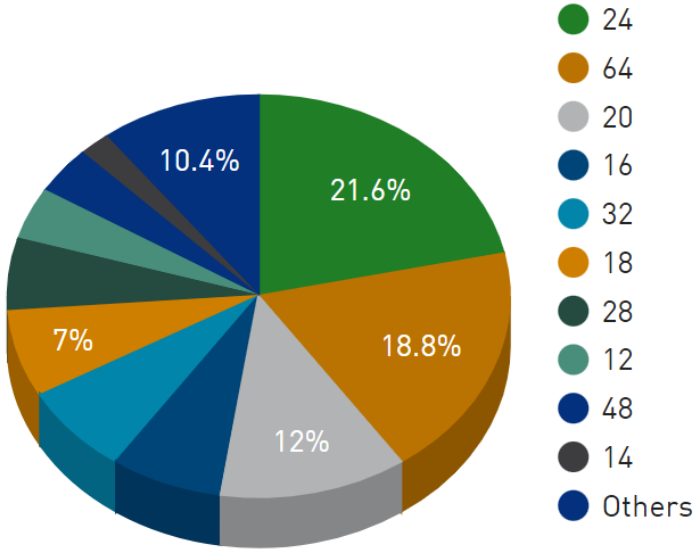


Processors and Operation Systems

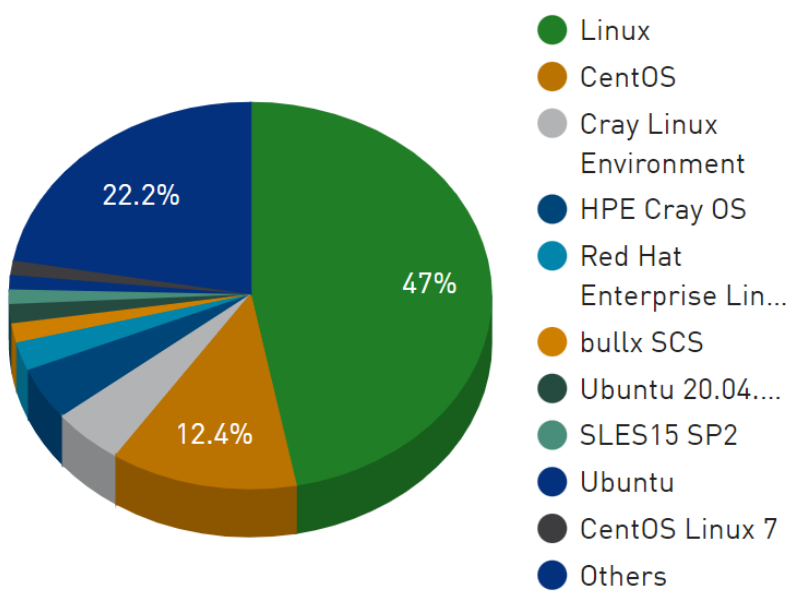
Processor Generation System Share



Cores per Socket System Share

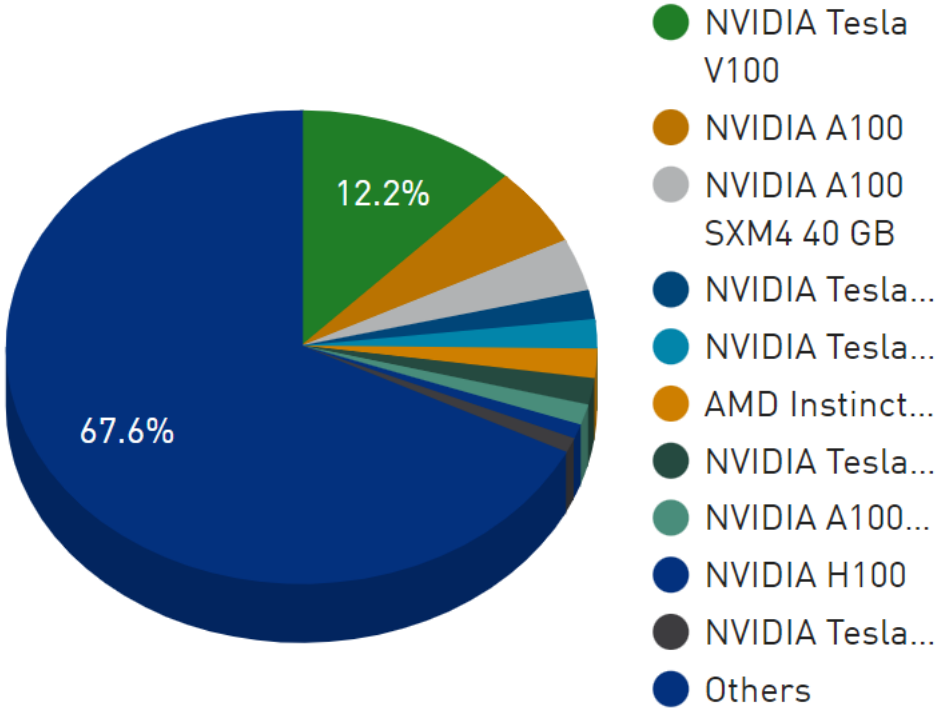


Operating System System Share

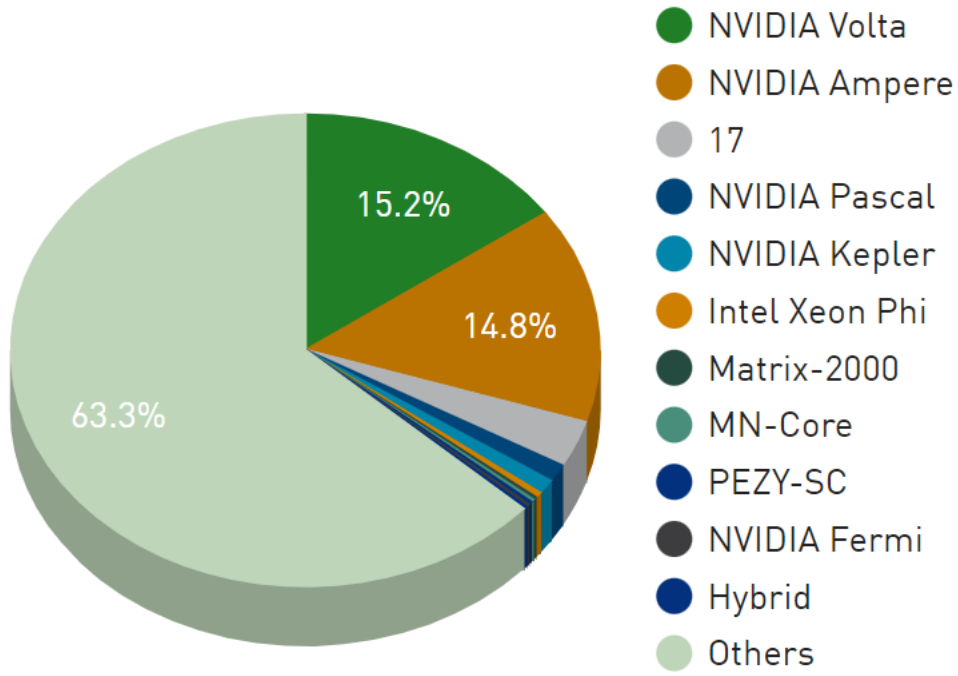


Accelerator/Co-Processor Systems

Accelerator/Co-Processor System Share



Accelerator/CP Family System Share



Where we are?

AVITOHOL - CLUSTER PLATFORM SL230S GEN8, INTEL XEON E5-2650V2 8C 2.6GHZ, INFINIBAND FDR, INTEL XEON PHI 7120P, **412 TFLOP/S =0.41 PFLOP/S, 20700 CORES, HPE**

Hosted at the Institute of Information and Communication Technologies, the Bulgarian Academy of Sciences

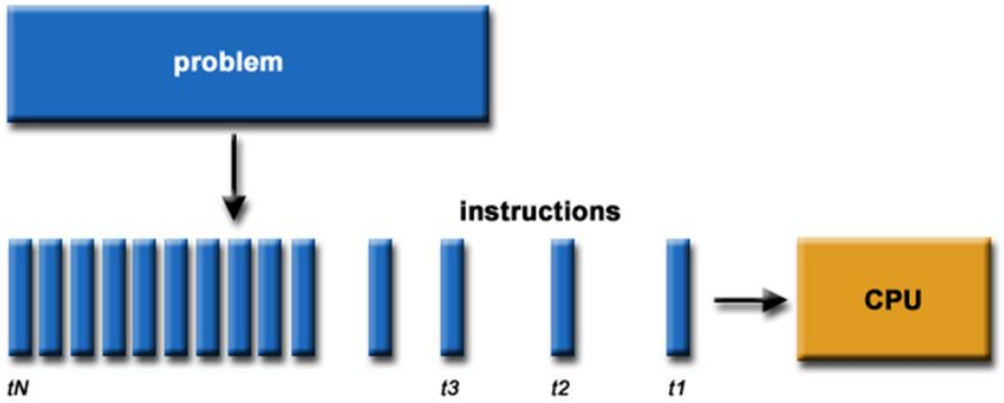
Operate since June 2015, ranking in top500 : 332 (June 2015 and 389 Nov 2015)

91(2021/134(2023), Discoverer - EuroHPC petascale supercomputer hosted at Sofia Tech Park, BullSequana XH2000, AMD EPYC 7H12 64C 2.6GHz, InfiniBand HDR, Consortium Petascale Supercomputer Bulgaria, 144,384 Cores, 4.52 **PFlop/s, operate October 2021.**

Какво е последователно/паралелно изчисление?

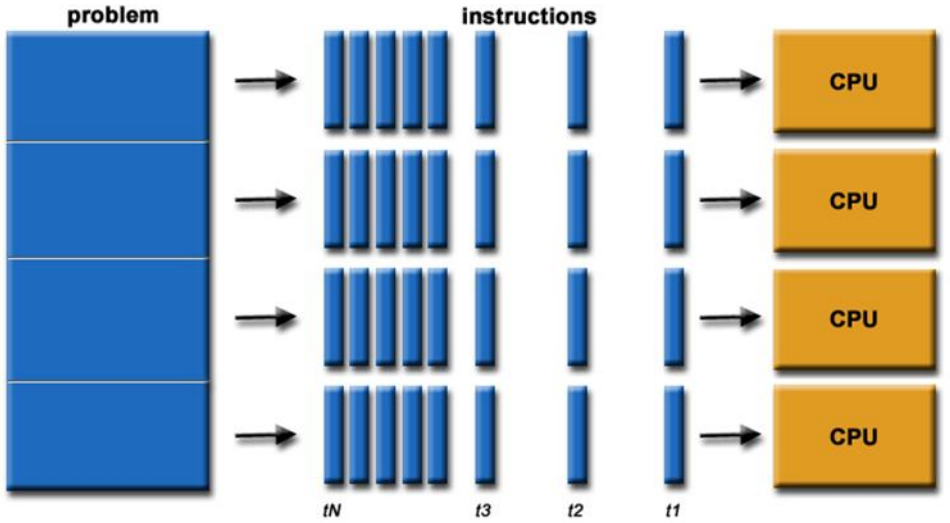
Последователно изчисление - традиционен програмен подход.

- Задачата се изпълнява на един компютър с един централен процесор (CPU).
- Проблемът се разделя на отделна поредица от инструкции.
- Инструкциите се изпълняват една след друга.
- Във всеки един момент може да бъде изпълнена само една инструкция



Паралелното изчисление е едновременното използване на множество изчислителни ресурси за решаване на изчислителна задача:

- Да се изпълнява с помощта на множество процесори (CPUs)
- Задачата е разделен на отделни части, които могат да бъдат решени едновременно
- Всяка част е допълнително разбита на поредица от инструкции
- Инструкциите от всяка част се изпълняват едновременно на различни процесори



Класическата таксономия на Флин

Има различни начини за класифициране на паралелни компютри.

- Една от по-широко използваните класификации, която се използва от 1966 г., се нарича таксономия на Флин.
- Таксономията на Флин разграничава многопроцесорните компютърни архитектури според това как те могат да бъдат класифицирани по двете независими измерения: Инструкция и Данни.
- Всяко от тези измерения може да има само едно от две възможни състояния: Единично или Множествено.
- Матрицата по-долу дефинира 4-те възможни класификации според Флин:

Класификация според нивото на паралелизъм на хардуера

- Многоядрени пресмятания (Multicore computing)
- Симетрични мултипроцесорни изчисления (Symmetric multiprocessing)
- Разпределени изчисления (Distributed computing)
  - Клъстерни изчисления (Cluster computing)
  - Масивно паралелни изчисления (Massive parallel processing)
  - Грид изчисления (Grid computing)
- Специализирани паралелни компютри
  - Векторни процесори (Vector processor)
  - Изчисления върху GPU (General-purpose computing on Graphics Processing Units)
  - Изчисления върху MIC (Many-integrated core)

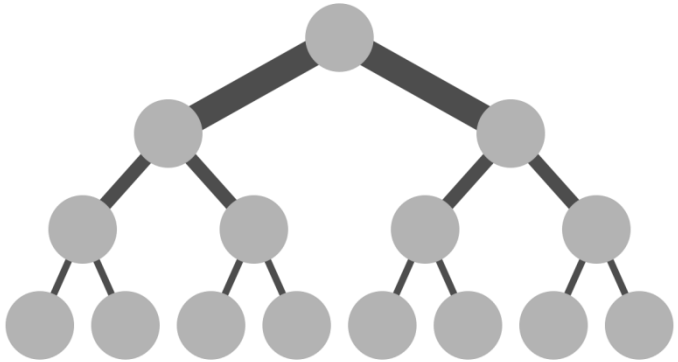
Терминология (1)

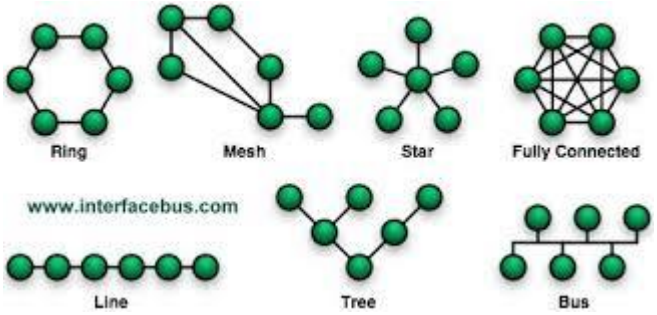
- Раздробеност (Granularity) – качествена мярка за отношението на изчисления към комуникации
  - Едро/грубо (coarse) – извършват се относително големи количества изчисления между комуникационни събития
  - Фино/дребно (fine) – извършват се относително малки количества изчисления между комуникационни събития
  - Независими задания (embarassingly parallel) – едновременно решаване на множество подобни, но независими задачи; малка до никаква необходимост от координиране между заданията.
- Паралелни ”забавяния” (Parallel overhead):

  времето необходимо за координиране на паралелни задания като противоположност на полезната работа

Терминология (2)

- **Скалируемост** – отнася се до възможността на паралелната система (софтуер и/или хардуер) да демонстрира пропорционално увеличаване на паралелното ускорение с добавяне на повече процесори. Фактори, които влияят на скалируемостта:
  - Хардуер – в частност ширината на лентата за комуникации между памет и процесор и тези по мрежа;
  - Прилагания алгоритъм
  - Паралелните забавяния
  - Характеристиките на вашите специфични приложение и програма
- **Топология на свързване** – физическата връзка между процесорите: звезда, пръстен, мрежа, хиперкуб, тор ...





Оценка на паралелната ефективността

- **Ускорение:** $S_P = \frac{T_1}{T_P}$, $S_P \leq P$; **Ефективност:** $E_P = \frac{S_P}{P} = \frac{T_1}{P.T_P}$, $E_P \leq 1$

- Наблюдавани ускорение и ефективност – T_1 и T_P са измерените

(wall-clock) времена съответно за последователно и паралелно изпълнение на програмата.

- Теоретична оценка на T_P при MIMD с разпределена памет и P процесора

$$T_P = T_a + T_{com}\ ,\ \ T_a = M * t_a,\ \ \ T_{com} = c_1 * t_s + c_2 * t_w$$

  - t_a е усреднено време за извършване на 1 ар. оп. от 1 процесор
  - t_s е времето за стартиране на комуникация
  - t_w е времето за изпращане на едно число до съседен процесор
  - c_1 и c_2 са константи или функции на размерността на задачата, броя на процесорите, разстоянието между тях

Amdahl's law (1)

- Нека ***S*** е частта в задачата, която пресмятаме и представлява последователно извършената работа
- Тогава ***1-S = P*** е часта, извършена паралелно
- Каква е максималната скорост за ***N*** процесори?

$$speedup = \frac{1}{(1-P) + \frac{P}{N}} \Rightarrow \lim_{N \to \infty} \frac{1}{1-P}$$

- Дори ако паралелната част е скалируема идеално, може да сме ограничени от последователната част на кода!

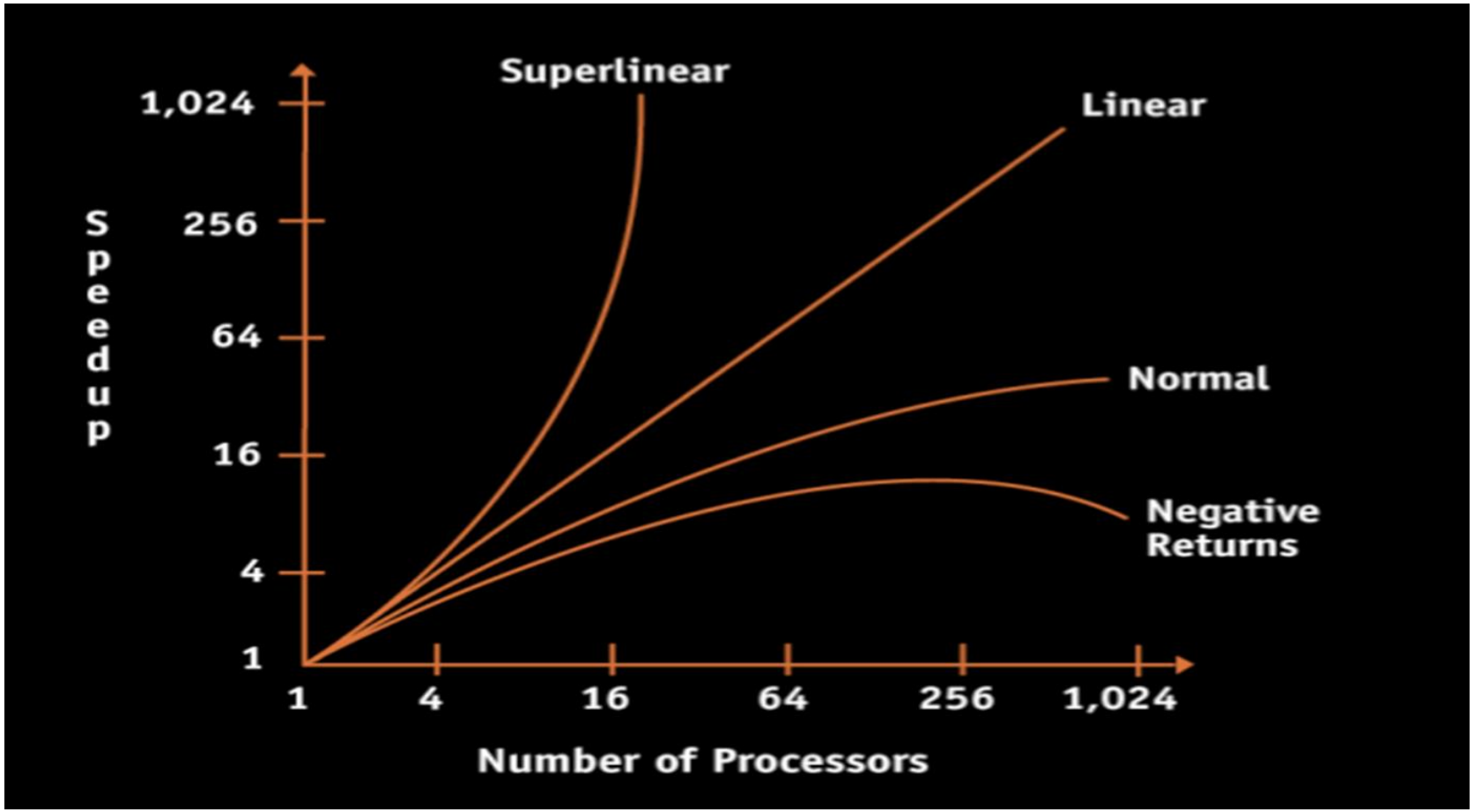
Amdahl’s law (2)

- Наличието на последователна част от кода е доста ограничаващо на практика:

>	2	4	8	32	64	256	512	1024
5%	1.91	3.48	5.93	12.55	15.42	18.62	19.28	19.63
2%	1.94	3.67	6.61	16.58	22.15	29.60	31.35	32.31
1%	1.99	3.88	7.48	24.43	39.29	72.11	83.80	91.18

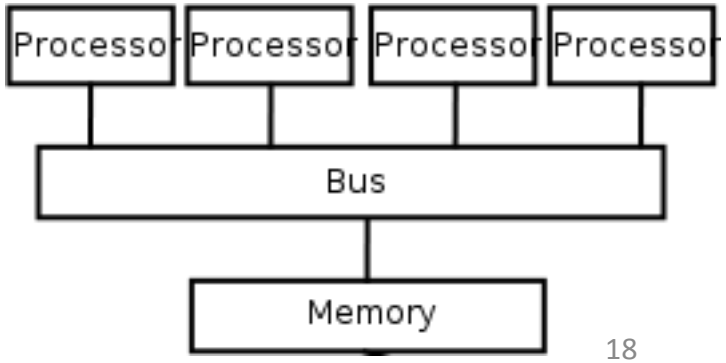
- Законът на Амдал е приложим само ако последователна част е независима от размера на проблема
- За щастие делът на изчисленията, които са последователни (непаралелни), обикновено намалява с увеличаване на размера на проблема (известен още като закон на Густафсон)

Speedup


Superlinear
Linear
Normal
Negative Returns
Speedup
1,024
256
64
16
4
1
1
4
16
64
256
1,024
Number of Processors

Обща памет - характеристики

- Всички процесори имат достъп до цялата памет като глобално адресно пространство.
- Няколко процесора могат да работят независимо един от друг, като споделят едни и същи ресурси памет.
- Промяна на стойност в паметта, направена от един процесор, е видима за всички други процесори.
- Два основни класа според времената за достъп до паметта на различните процесори – UMA (Uniform Memory Access) и NUMA (Non-Uniform Memory Access).
  - UMA – SMP, идентични процесори, равноправен достъп и време за достъп до паметта;
  - NUMA – няколко физически свързани SMP, не всички процесори имат равно време за достъп до всяка от паметите.



Обща памет – предимства и недостатъци

Предимства

- Глобалното адресно пространство предоставя удобни за потребителя средства за работа с паметта в програмни реализации.
- Времето за споделяне на данни между отделни паралелни задания е бързо и равномерно, благодарение на физическата близост на паметта и процесорите.

Недостатъци

- Основен недостатък е липсата на скалируемост за отношението на обем памет и брой използвани процесори. Добавянето на повече процесори може да доведе до забавяне, породено от геометрично увеличение на трафика по пътя между паметта и процесорите, а при системи със съгласуваност на кеша, увеличение на трафика асоцииран с управлението на връзките между кеша и паметта
- Програмистът има отговорността така да синхронизира изпълнението на заданията, че да гарантира ”коректни” резултати при заявки към паметта.
- Цена: все по-трудно и скъпо е да се проектират и произвеждат машини с обща памет с все по-голям брой процесори.

Разпределена памет - характеристики

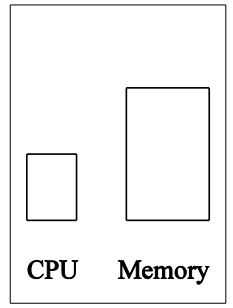
- За осъществяване на връзка между процесорите и паметта е необходима комуникационна мрежа.
- Процесорите имат собствена локална памет. Адресните пространства за различни процесори не съвпадат и не съществува понятие за общо адресно пространсвто за всички процесори.
- Тъй като всички процесори имат собствена памет, те работят независимо. Промени в локалната памет на един процесор не се отразяват в паметта на другите. Поради тази причина понятието за съгласуваност на кеша не е приложимо.
- При необходимост от достъп на един процесор до данните в паметта на друг, програмистът явно трябва да дефинира как и кога да се разменят тези данни, както и да синхронизира работата на съответните паралелни задания.
- Средствата за осигуряване на трансфера на данни са много разнообразни, като могат да се използват дори обикновени етернет мрежи.

Разпределена памет – предимства и недостатъци

Предимства

- Паметта е скалируема по отношение на броя процесори. При увеличаване на този брой, обема на паметта се увеличава пропорционално.
- Всеки процесор осъществява бърз и непрекъснат достъп до локалната си памет, без забавяне поради необходимост от съгласуване на кеша.

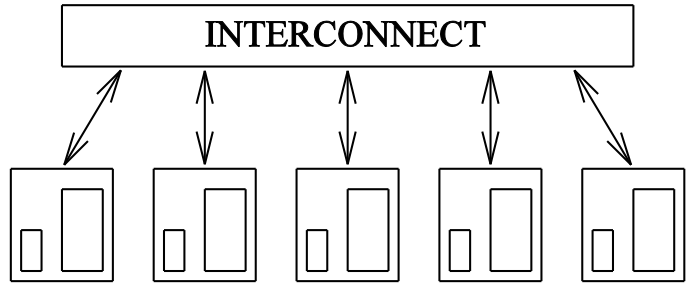
Ценова ефективност: могат да се използват широко достъпни процесори и мрежи.



Фон Нойман

Недостатъци

- Програмистът е отговорен за много от детайлите при осъществяване на комуникацията между процесорите.
- Може да се окаже трудно директното изобразяване на съществуващи структури от данни, проектирани за глобална памет, в програми използващи разпределена памет.
- Неравномерни времена за достъп до паметта.



Разпределена памет

Хибридни архитектури с обща и разпределена памет

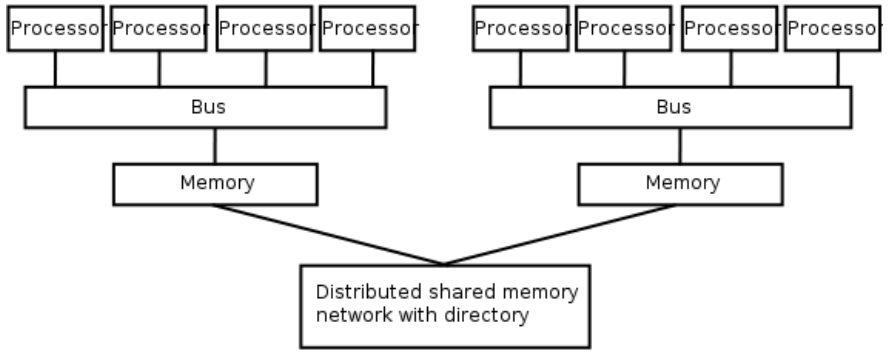
Общи характеристики

- Най-големите и бързи компютри в света в днешно време съчетават архитектури с обща и разпределена памет.
- Компонентът с обща памет най-често е SMP машина със съгласуван кеш. Процесорите в една SMP третират паметта на машината като глобална.
- Компонентът с разпределена памет се реализира чрез свързване в мрежа на няколко SMP машини. Отделните SMP виждат само своята памет, не и паметта на другите машини в мрежата. За прехвърляне на данни между тях се използват комуникации през мрежата.
- Настоящите тенденции предполагат, че в близко бъдеще хибридните архитектури ще продължават да преобладават и да увеличават списъка на най-бързите и мощни компютърни системи.

Предимства и недостатъци

- общите за архитектури с

обща и разпределена памет



Модели за паралелно програмиране

- Съществуват като абстракция над хардуера и организацията на паметта;
- НЕ са специфични за определен тип машина или архитектура, а теоретично всяка от тях може да се приложи върху произволен хардуер (виртуална обща памет, предаване на съобщения за система с обща памет)
- Разпространени са няколко модела:
  - Обща памет – заданията споделят общо адресно пространство; асинхронно четене и писане; контрол на достъпа с механизми като заключване и семафори;
  - + няма собственост върху данните и нужда от определяне на комуникации;
  - - става трудно да се разбере и управлява локалността на данните;
  - Нишки – POSIX Threads, Open MP
  - Предаване на съобщения – MPI
  - Паралелни данни – извършване на операции върху набор от данни (масив или куб); всяко задание работи върху различна част от тях; F90, F95, HPF
  - Хибрид – комбинация от вече споменатите.

Нишки (Threads)

Принципи

- ОС стартира главната програма **a.out** и зарежда необходимите ресурси
- a.out изпълнява последователна работа, след което генерира няколко задания (нишки), които могат да се стартират от ОС едновременно
- Всяка нишка има своя локална памет, но споделя и общите ресурси на a.out и печели от достъпа до общата памет, осигурен от a.out.
- Произволна нишка може да изпълни произволна подпрограма по едно и също време с останалите нишки.
- Нишките комуникират помежду си чрез глобалната памет. Изисква синхронизация, за да се гарантира, че не повече от една нишка променя даден адрес в глобалната памет в произволен момент.
- Нишките могат да идват и да си отиват, но a.out остава, за да сигури необходимите споделени ресурси до завършване на приложението.

Реализация:

- Обикновено включват библиотеки от подпрограми извиквани в рамките на паралелния код и/или набор от директиви на компилатора вградени в последователния или в паралелния код. Програмистът определя паралелизма.
- Стандартизация – **POSIX Threads (Pthreads) и Open MP.**

Предаване на съобщения (Message Passing)

Принципи

- Множество от няколко задания, които имат тяхна локална памет.
- Заданията могат да се намират върху една и съща физическа машина или върху произволен брой машини.
- Ако една променлива е декларирана във всяко задание на програма с p задания, тогава има p различни променливи с едно и също име, но с евентуално различни стойности.
- Ако изпълнението на функция от дадено задание изисква данни от друго задание, данните се изпращат чрез оператор за предаване на съобщения и в двете (изпращащо и получаващо) задания.
- Върху почти всички архитектури, предаването на съобщения е много по-бавно от аритметиката с плаваща запетая.

Реализация

- Включват библиотеки от подпрограми извиквани в рамките на паралелния код. Програмистът определя паралелизма.
- Стандартизация – **Message Passing Interface (MPI).**
- За архитектури с обща памет, реализациите на MPI обикновено не използват мрежа за комуникациите между заданията. Вместо това се използва общата памет за по-добра производителност.

Компоненти на OpenMP

- Включване на библитеката **#include <omp.h>**
- Директиви на компилатора

```
#pragma omp parallel
{       //code segment
}
```

```
#pragma omp for
for()
{ … }
```

- контрол на данни и среда чрез **private, firstprivate, lastprivate, num_threads**

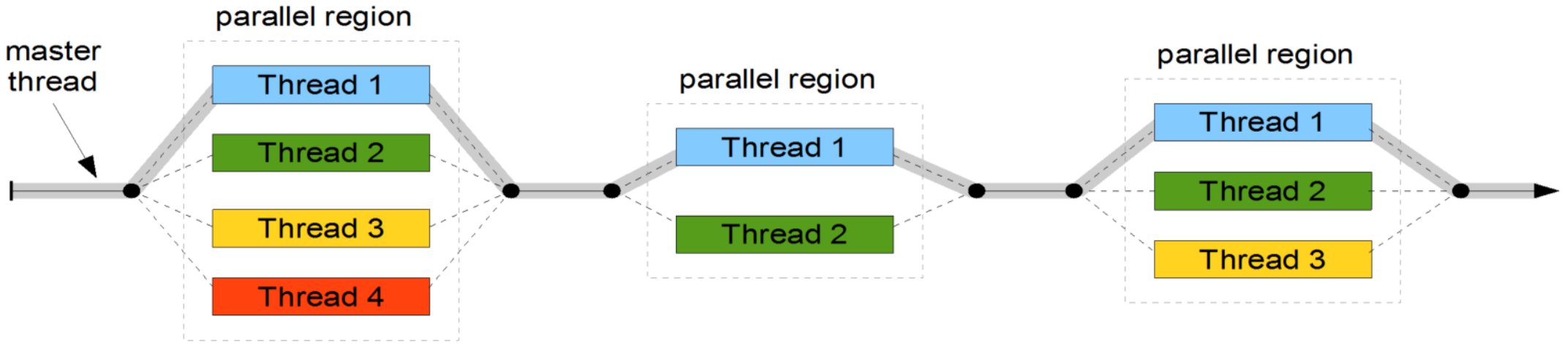
Библиотечни функции

- int omp_get_num_threads(void) - текущ брой използвани нишки
- int omp_set_num_threads(int NumThreads) - преди влизането в паралелна част указва с колко нишки да се изпълни
- int omp_get_thread_num(void) - връща номера на текущата нишка

  int omp_get_num_procs(void) - връща броя на достъпните ядра
- **Променливи на средата**
  - OMP_SCHEDULE - онтролира разпределянето на изпълнението на цикъл между нишките
  - OMP_NUM_THREADS - дефинира броя нишки в паралелния участък

Fork–Join model

- OpenMP използва **fork–join** модела
  - *Master* thread изпълнява последователен код
  - Fork – Master thread създава/събужда допълнителни нишки за паралелно изпълнение
  - Join – At the end of parallel code created threads die or are suspended



Hello OpenMP World!

```
#include <omp.h>
#include <stdio.h>

int main (int argc, char **argv) {
    #pragma omp parallel
    {
        int tid = omp_get_thread_num();
        int nth = omp_get_num_threads();
        printf("Hello World from thread = %d of %d\n",tid,nth);
    }

    return 0;
}
```

Compiling OpenMP programs

- Requires compiler support for OpenMP
  - Most modern compilers support OpenMP
    - GNU (`gcc`), LLVM (`clang`), Intel (`icc`), Portland (`pgcc`), IBM (`xlc`), Oracle (`suncc`), Microsoft (`cl.exe`) and many more
- GCC:

```
gcc -fopenmp hello_omp.c –o hello_omp
```

- Intel:

```
icc -qopenmp hello_omp.c –o hello_omp
```

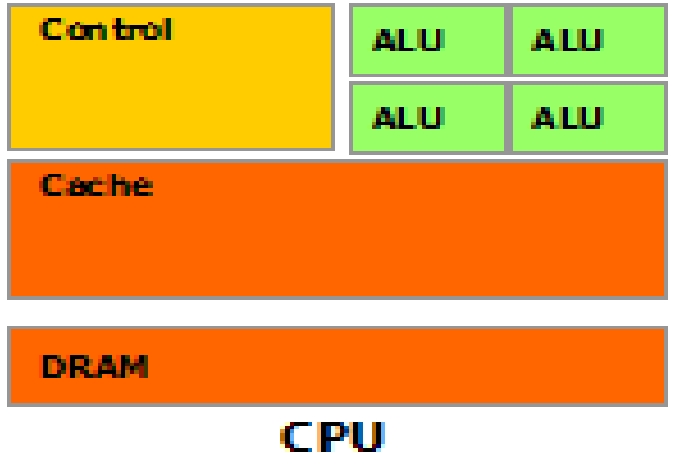
- Intel (older versions):

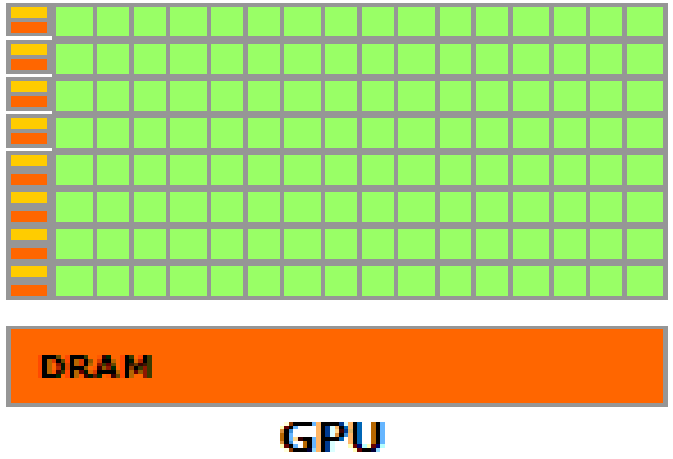
```
icc -openmp hello_omp.c –o hello_omp
```

CPU vs GPU architecture

- CPU (latency oriented design):
  - Large caches (големи кешове)
  - Sophisticated control (сложен)
  - Powerful ALU (Мощен)

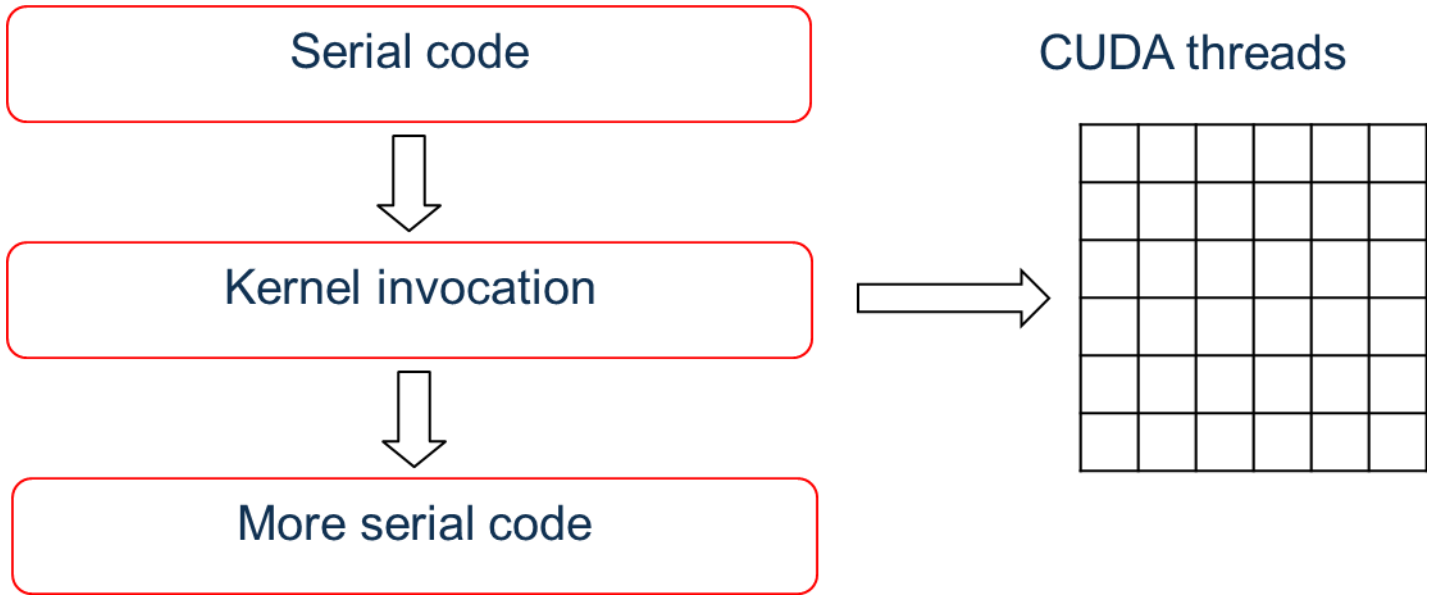
- GPU (throughput oriented design):
  - Small caches (малки кешове)
  - Simple control (прост контрол)
  - Energy efficient ALUs (ефективност от към енергия)
  - Latencies compensated by a large number of threads (Латенции са компенсирани от голям брой нишки)





Heterogeneous execution model

- **Host** — a CPU which executes the main program in serial.
- **Device** — a GPU which executes parallel portions of the code.
- Memory spaces are separate*
  - Allocation and data movement is the responsibility of the programmer.



Code for GPUs

- CUDA C program is written as follows:
  - Serial parts in host C code
  - Parallel parts in device SIMD kernel C code
- Source code is compiled separately
  - Standard C/C++ code for the CPU
  - Device code in PTX – compiled just-in-time for the exact device
- Use the nvcc for compilation
  - PTX is an assembly format
  - Specific binary code for the GPU devices

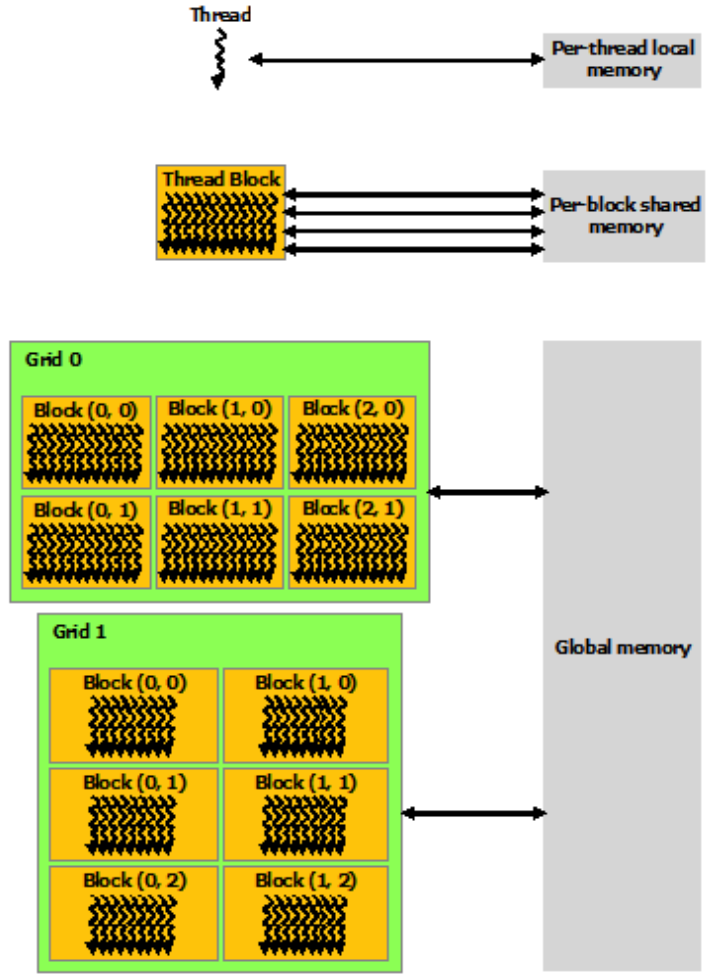
CUDA kernel example

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

GPU memory organization

- Registers (local memory) are per-thread
  - **very low latency, very high throughput**
  - limited resource, used for automatic variables
- Shared memory (and L1 cache) is per-block
  - **low latency, high throughput**
  - can yield significant performance boost, depends on algorithm
  - programmer is responsible for its usage
  - shared/cache split can be controlled using the API
- Global memory is visible to all threads
  - **high latency, moderate throughput**
  - memory allocated with cudaMalloc is global
  - has the highest capacity



Matrix multiplication example

Simple version:

```
__global__
void matrixMulKernel(float* A, float* B, float* C,
int width) {
      int i;
      int row = blockIdx.y*blockDim.y+threadIdx.y;
      int col = blockIdx.x*blockDim.x+threadIdx.x;

      if ((row<width) && (col<width)) {
            float tmp = 0;
            for (i = 0; i < width; ++i)
                  tmp += A[row*width+i]*B[i*width+col];
            C[row*width+col] = tmp;
      }
}
```

Matrix multiplication with shared memory

```
#define TILE_WIDTH 32

__global__
void matrixMulKernel(float* A, float* B, float* C, int width) {
      __shared__ float sA[TILE_WIDTH][TILE_WIDTH];
      __shared__ float sB[TILE_WIDTH][TILE_WIDTH];

      int bx=blockIdx.x, by=blockIdx.y;
      int tx=threadIdx.x, ty=threadIdx.y;
      int row = by*TILE_WIDTH+ty;
      int col = bx*TILE_WIDTH+tx;
      float tmp = 0;

      for (int i = 0; i < width/TILE_WIDTH; ++i) {
            sA[ty][tx] = A[row*width+i*TILE_WIDTH+tx];
            sB[ty][tx] = B[(i*TILE_WIDTH+ty)*width+col];
            __syncthreads();
            for (int j = 0; j < TILE_WIDTH; ++j) {
                  tmp += sA[ty][j]*sB[j][tx];
            }
            __syncthreads();
      }
      C[row*width+col] = tmp;
}
```

Насоки за постигане на оптимална производителност

- Има 3 начина за подобряване на производителността:
  - Работете по-усърдно
  - Работете по-умно
  - Извикай помощ
- Аналогия в компютърните науки
  - Използвайте по-бърз хардуер
  - Оптимизиране на алгоритми и техники, използвани за решаване на изчислителни задачи
    - Увеличете максимално паралелно изпълнение, за да постигнете максимално използване;
    - Оптимизирайте използването на паметта
    - Оптимизирайте използването на инструкции
  - Използвайте няколко компютъра за решаване на определена задача
- И трите стратегии могат да се използват едновременно!

Thanks!

Funded by
the European Union

EuroHPC
Joint Undertaking

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101101903. The JU receives support from the Digital Europe Programme and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia