EURO^2

Dr. Hristo Iliev, UoS & NCC-BG

Parallel Programming with MPI Course

University of Sofia, 22.05.2024

Going Parallel

A Step by Step Guide

EURO²

Assumptions

- You have your own sequential C/C++/Fortran/Python code, or
- You understand someone else's sequential code, but
- It's too slow for your needs, or
- You run out of memory

Why Go Parallel?

EURO²

- Access more **compute power** (CPU / GPU cores)
  - Finish calculations sooner
- Access more **memory**
  - Work on (much) bigger problems
- Access more **I/O bandwidth**
  - Work with huge stored datasets

Why Go Parallel? More Compute Power

EURO²

- Physics forbids making infinitely fast CPUs, instead units get **replicated**:
  - Core ⇒ Multicore ⇒ Multi-socket ⇒ Cluster
- Distributed compute power comes with tradeoffs
  - Different programming paradigms with increased complexity
  - Increased operational complexity

Why Go Parallel?
More Memory

EURO²

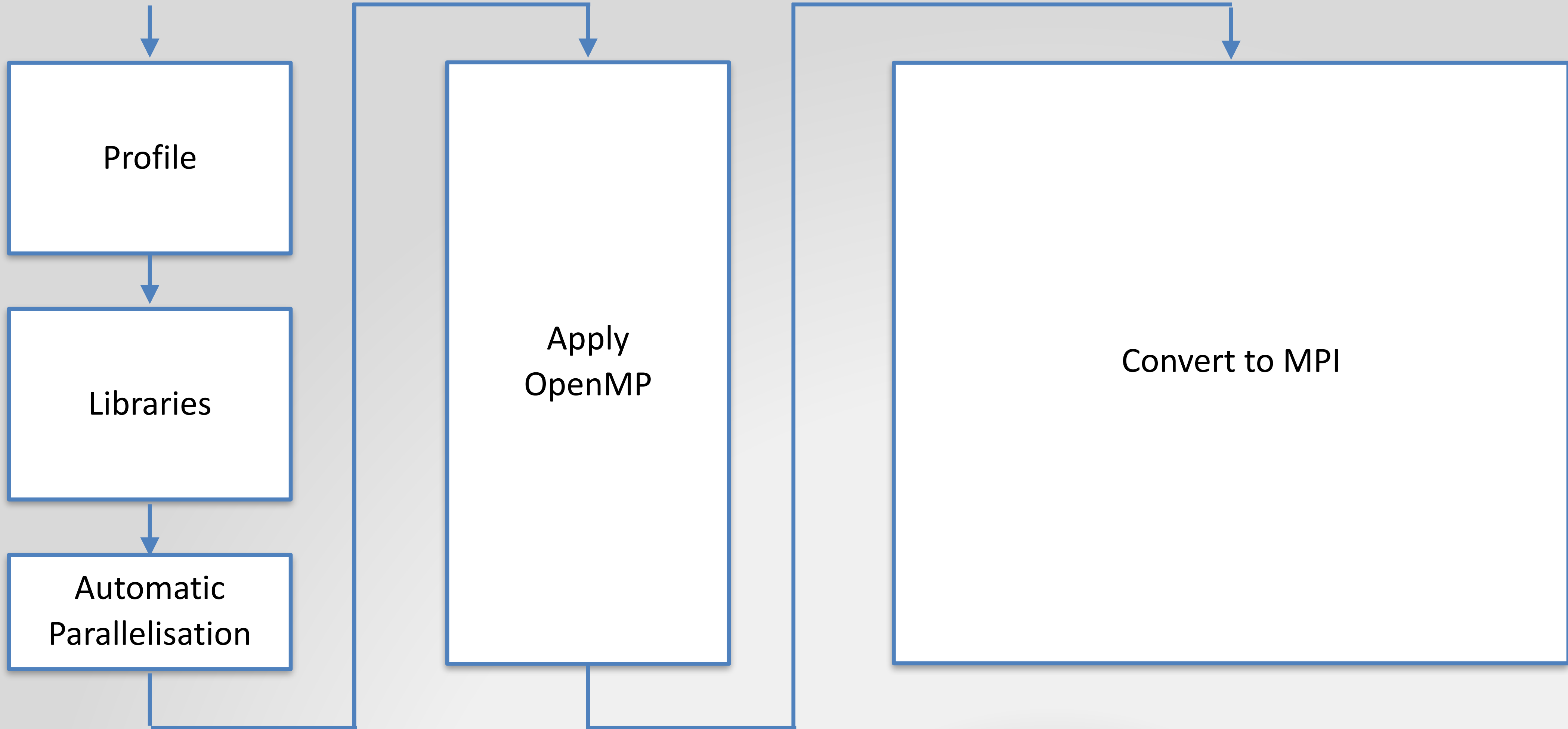
- Memory technology lags the advancements in computation speeds
- Engineering limits on how much memory a CPU can control
- Pool memory from different nodes ⇒ distributed memory computing
- More memory allows for larger and more detailed simulations

Why Go Parallel?
More I/O Bandwidth

EURO²

- Disk storage is the second slowest I/O device after Internet
- A node has limited I/O bandwidth
- Pool bandwidth from different nodes ⇒ parallel I/O
- Not necessarily related to the need for more CPU or memory
- For example: simple sequential processing of large amounts of data

Going Parallel
Step by Step



Get Your Bearings
Step 1: Profile

- Profile the code either with timing calls or with a dedicated profiler:
  - Easy: Intel VTune
  - Advanced: Score-P + CUBE
  - Hardcore: gprof, oprof
- Identify I/O and compute regions that take a lot of time

Get Your Bearings
Step 1: Profile

EURO²

<t> - Intel VTune Amplifier

Welcome | r001runsa

runsa Hardware Events viewpoint (change)

INTEL VTUNE AMPLIFIER 2018

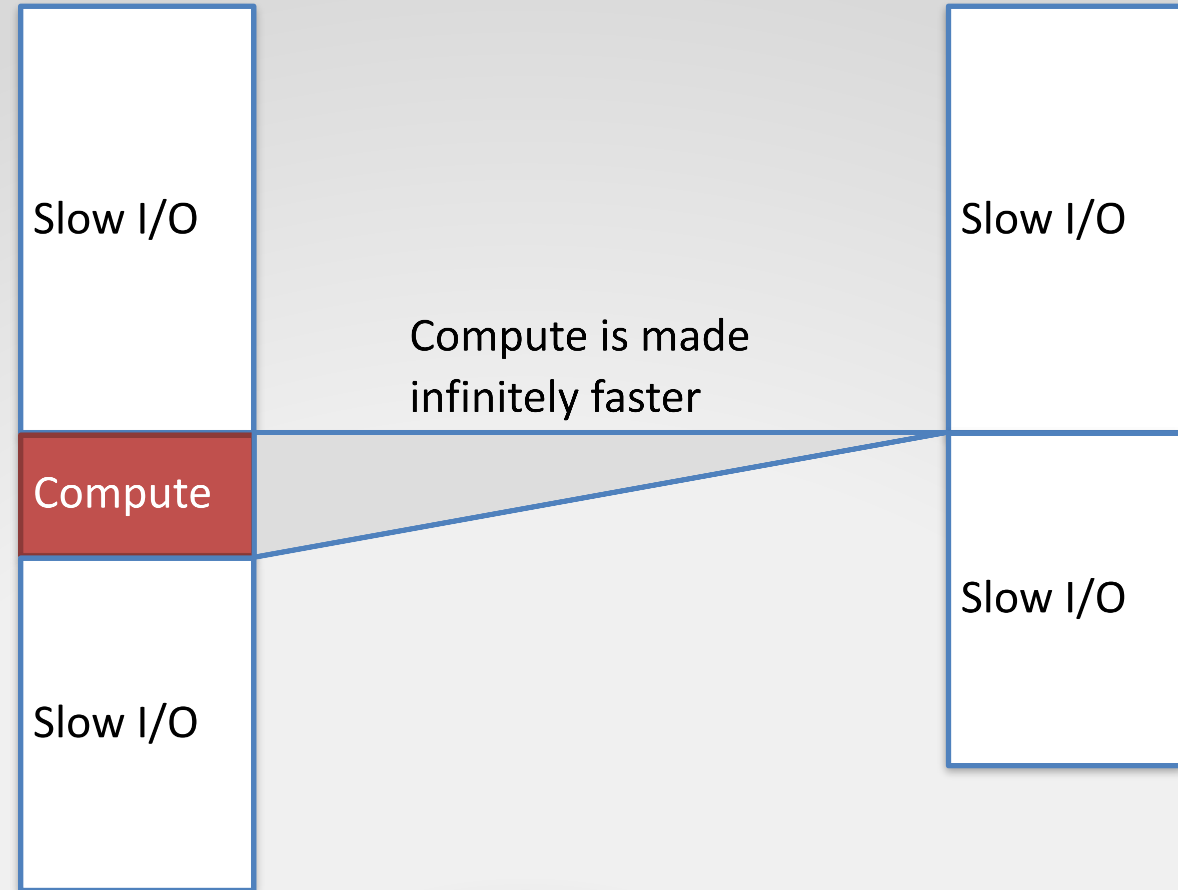
Collection Log | Analysis Target | Analysis Type | Summary | Event Count | Sample Count | Caller/Callee | Top-down Tree | Platform

Grouping: Call Stack

Function Stack	CPU_CLK_UNHALTED.THREAD ▼	INST_RETIRED.ANY	CPU_CLK_UNHALTED.REF	CPU_CLK_UNHALTED.REF_P
▼ Total	183,758,000,000	177,486,000,000	166,458,000,000	7,877,856,000
▼ main	91,004,000,000	48,804,000,000	82,240,000,000	3,899,028,000
▼ pagerank::compute_push	90,966,000,000	48,762,000,000	82,206,000,000	3,890,496,000
▶ [Loop at line 97 in pagerank::compute	88,336,000,000	47,612,000,000	79,818,000,000	3,778,264,000
▶ pagerank::reset_next_round	2,594,000,000	1,102,000,000	2,354,000,000	120,232,000
▶ pagerank::is_converged	36,000,000	48,000,000	34,000,000	0
▼ [Loop at line 148 in main]	38,000,000	42,000,000	34,000,000	532,000
▼ pagerank::compute_push	38,000,000	42,000,000	34,000,000	532,000
▶ pagerank::initialize	38,000,000	42,000,000	34,000,000	532,000
▶ std::vector<std::__cxx11::basic_string<cha	23,766,000,000	36,150,000,000	21,548,000,000	982,604,000
▶ std::__introsort_loop<__gnu_cxx::__norma	18,050,000,000	13,660,000,000	16,308,000,000	771,932,000
▶ std::__introsort_loop<__gnu_cxx::__norma	10,994,000,000	15,954,000,000	9,928,000,000	467,628,000
▶ boost::detail::function::function_obj_invoke	9,236,000,000	10,672,000,000	8,508,000,000	419,748,000
▶ boost::algorithm::iter_split<std::vector<std	6,744,000,000	11,816,000,000	6,110,000,000	292,600,000
▶ graph::construct_from_dimacs	5,815,000,000	10,816,000,000	5,302,000,000	252,168,000
▶ std::vector<std::__cxx11::basic_string<cha	4,890,000,000	7,692,000,000	4,180,000,000	220,780,000
▶ __intel_ssse3_rep_memcpy	4,382,000,000	9,260,000,000	4,178,000,000	175,028,000
▶ gen_transposed_csr	3,822,000,000	3,834,000,000	3,462,000,000	167,048,000

Get Your Bearings
Step 1: Profile

- Speeding up insignificant parts has little overall effect!
- Make sure you target the right part of the algorithm



Be Lazy
Step 2: Look for Alternatives

EURO²

- Are you doing linear algebra by hand-coding loops?
- Can you do it faster than BLAS and LAPACK?
- Some libraries already make use of parallelism
  - Intel MKL (part of oneAPI)
  - OpenBLAS, ATLAS
  - cuBLAS

Embrace Multicore
Step 3: Try Automatic Parallelisation

EURO²

- Some compilers provide automatic loop parallelisation
    - GCC
    - Intel oneAPI
    - MSVC
- Check your compiler’s manual for details
- Parallel code can only run on single node (threads)

Embrace Multicore
Step 3: Try Automatic Parallelisation

- GCC example – parallelise all loops with no loop-carried dependencies and distribute them over N threads

```
-floop-parallelize-all -ftree-parallelize-loops=N
```

Embrace Multicore
Step 3: Try Automatic Parallelisation

EURO²

- Very conservative – only “easy” loops are transformed
- Compiler-specific hints for guided parallelisation
- Cheap but often subpar
- Better use OpenMP

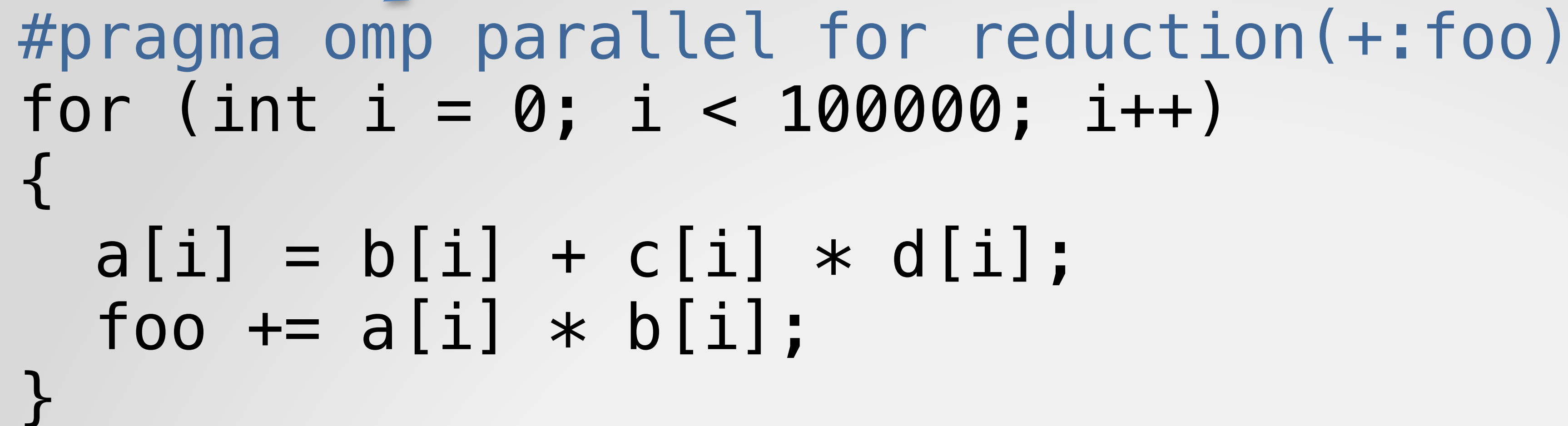
Embrace Multicore / Accelerators
Step 4: Use OpenMP

EURO²

- Pragma- (C/C++) or comments- (Fortran) based parallelisation standard
- Allows for gradual parallelisation
    - Same source code ⇒ both sequential and parallel executables
    - Focus on demanding loops
    - Tasks
- Support for offloading to accelerators (GPUs, etc.)

Embrace Multicore / Accelerators
Step 4: Use OpenMP

EURO²




```
#pragma omp parallel for reduction(+:foo)
for (int i = 0; i < 100000; i++)
{
   a[i] = b[i] + c[i] * d[i];
   foo += a[i] * b[i];
}
```

Embrace Multicore / Accelerators
Alternatives to OpenMP

- Language-specific constructs
  - C++17 parallel – worse tooling than OpenMP
  - CUDA – vendor lock-in with NVIDIA
  - SYCL, DPC++ – open standard (SYCL) but largely supported by Intel only (DPC++)
- Libraries
  - Intel TBB (C++)
  - Hardcore: POSIX Threads, Java threads

Embrace Multicore / Accelerators
Alternatives to OpenMP

- Python
  - `multiprocessing` module
  - CuPy – NVIDIA's GPU-accelerated NumPy and SciPy
  - Numba – Python JIT compiler targeting various CPUs and CUDA

Embrace Multicore / Accelerators
Caveats

- Threading is both easy and hard
- Easy – no need for proper domain decomposition with shared memory
- Hard – sharing issues, race conditions, false sharing
- Intel VTune is your best friend
- Cannot use resources from more than one computer node
- I/O bandwidth is still limited

Go Big
Step 5: MPI

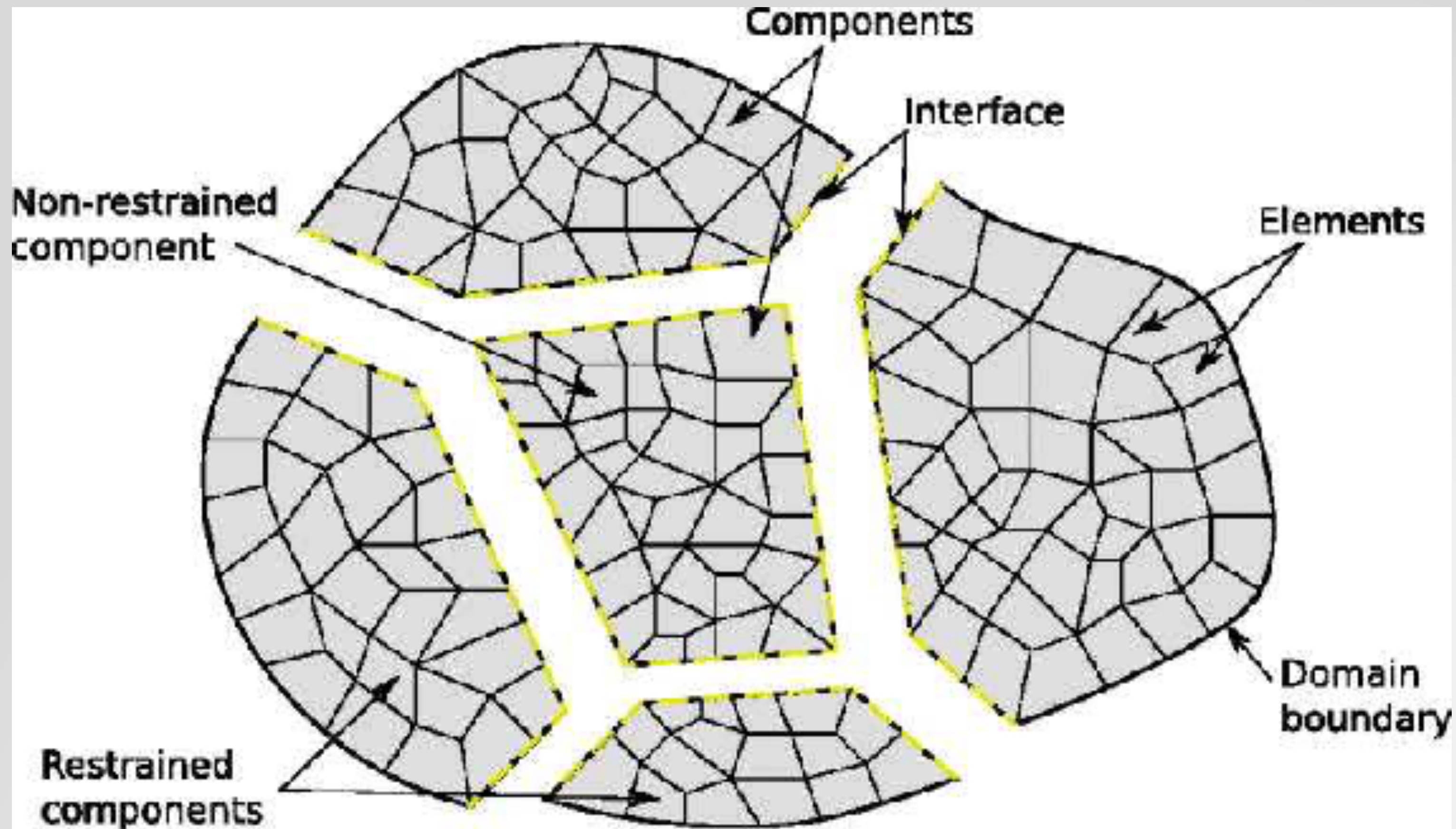
- Message Passing Interface – de factor standard for programming distributed memory computers
- Abstract communication library for C and Fortran
  - Non-standard bindings for C++ (Boost), Python (mpi4py), etc.
- Many implementations
  - Open-source: MPICH, MVAPICH, Open MPI
  - Vendor-specific: Intel MPI, Microsoft MPI, Cray MPI, etc.

Go Big
Step 5: MPI

EURO²

- MPI provides source-level compatibility between implementations
- Same code (theoretically) runs on a laptop and on a supercomputer
- Abstracts away the underlying communication mechanisms
- Single Program Multiple Data (SPMD)
    - Can be hard to grasp initially, but it gets easier with time

Go Big
Domain Decomposition



Go Big
Domain Decomposition

EURO²

- Proper **domain decomposition** is crucial for any parallel algorithm
  - Dictates data handling and communication
  - Distributed vs replicated data
  - Halos (ghost cells)
  - Keep global operations at minimum

Go Big
Alternatives to MPI

EURO²

- Libraries
  - CHARM++ (C++, used in NAMD)
- HPC languages
  - Chapel (Cray)
  - Unified Parallel C (Berkeley)

Go Big Data
Step 1: Use Apache Spark

EURO²

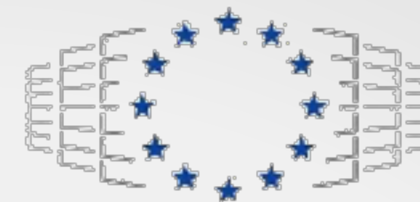
- For tasks involving huge amounts of stored or streaming data
  - Preference for structured (tabular) data
  - Unstructured data is also supported, but you get no help from the optimiser
- Scala, Java, Python, and SQL interfaces
- Robust in-memory computations

Go Big Data
Alternatives to Apache Spark

EURO²

- Dask (Python)
  - Pandas on steroids
  - More pythonistic than PySpark
  - Dask-ML
  - Integration with Hadoop and various HPC schedulers

Thanks! БЛАГОДАРЯ!



This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, United Kingdom, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, Republic of North Macedonia, Iceland, Montenegro