Formation MCIA

Olivier Lagrasse – Eric Michel

Lenovo 2023 Lenovo Internal. All rights reserved.

Agenda

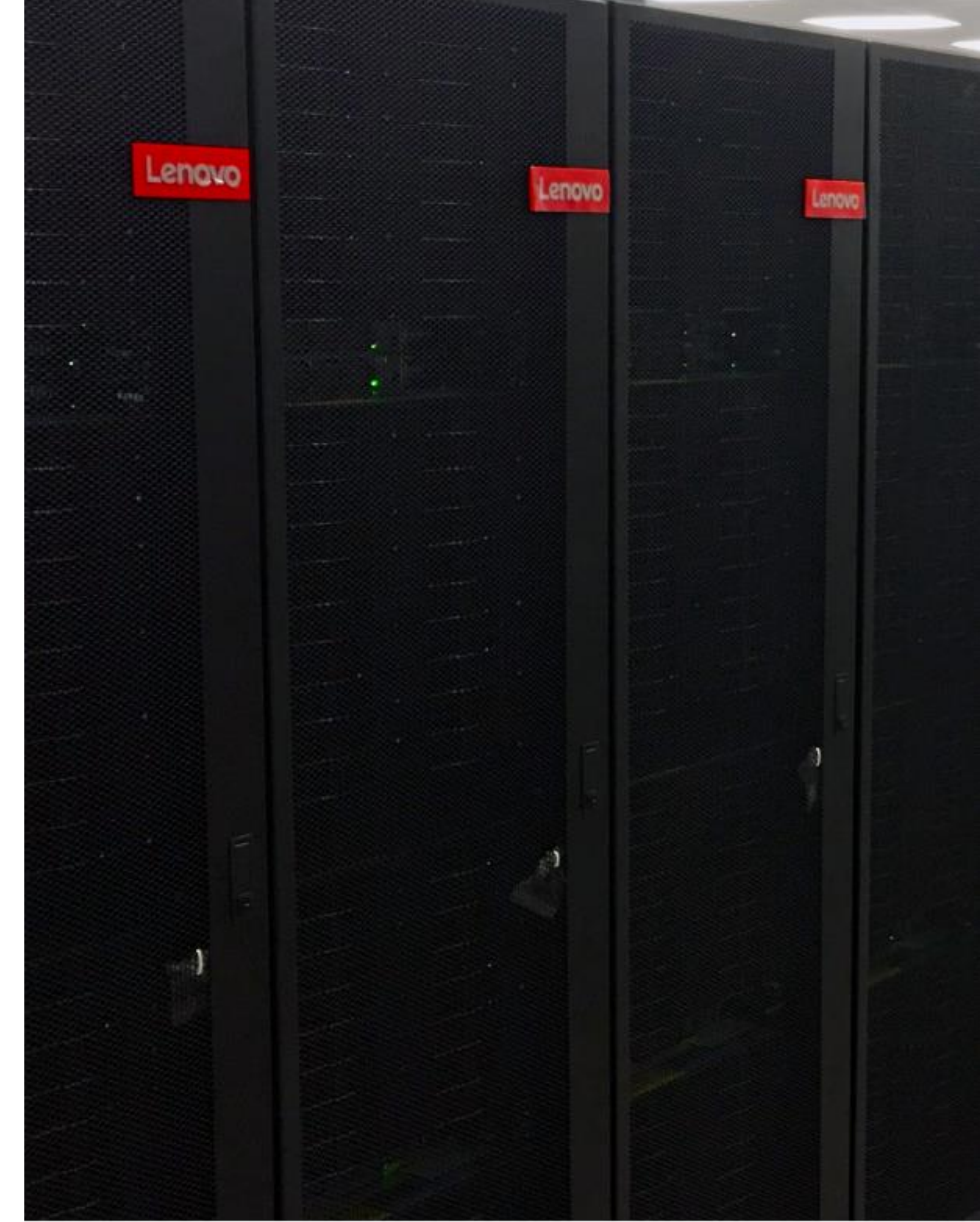
- Présentation de la machine Lenovo “Curta” du MCIA
- Concepts généraux pour une bonne performance des applications scientifiques
  - Processeur (fréquence, unités vectorielles, Hyper-Threading…)
  - Mémoire, réseau, stockage, parallélisme, consommation électrique…
  - Facteurs limitants la performance
- Présentation des outils de construction de code
  - Compilateurs : gnu, Intel (legacy et llvm)
  - Librairies mathématiques : open source, Intel MKL
  - Librairies MPI : IntelMPI, OpenMPI, mvapich2…
- Outils de diagnostic, profiling, analyse
  - Système (htop, perf, numactl, gprof …)
  - Suite Intel (APS, Vtune,…)
- Optimisation de l'exécution
  - Recherche des hotspots CPU, MPI, I/O…
  - Options de compilations avancées (vectorisation, auto parallélisation, inlining…)
  - Binding des tâches et des threads, mapping de la mémoire
  - Utilisation des librairies scientifiques optimisées (MKL…)
  - Optimisation des communications MPI (variables d’environnement…)
  - Optimisation des I/O parallèles

2023 Lenovo Internal. All rights reserved.

Présentation de la machine Lenovo “Curta” du MCIA

- 336 nœuds « compute » SD530 : n[001-336]
  - 2 * Intel Xeon Gold SKL-6130 16 cœurs @ 2.1GHz
  - 92 Go de mémoire
- 4 nœuds « bigmem » SR950 : bigmem[01-04]
  - 4 * Intel Xeon Gold SKL-6130 16 cœurs @ 2.1GHz
  - 3 To de mémoire
- 4 nœuds « gpu » SR650 : gpu[01-04]
  - 2 * Intel Xeon Gold SKL-6130 16 cœurs @ 2.1GHz
  - 192 Go de mémoire
  - 2 GPUs nVidia P100
- Réseau d’interconnexion : Intel OmniPath 100Go/s par nœud
- Système d’exploitation : Linux CentOS 7.4
- Gestionnaire de travaux : Slurm

- [From: https://redmine.mcia.fr/projects/cluster-curta/wiki](https://redmine.mcia.fr/projects/cluster-curta/wiki)



Lenovo 2023 Lenovo Internal. All rights reserved.

Présentation de la machine Lenovo “Curta” du MCIA

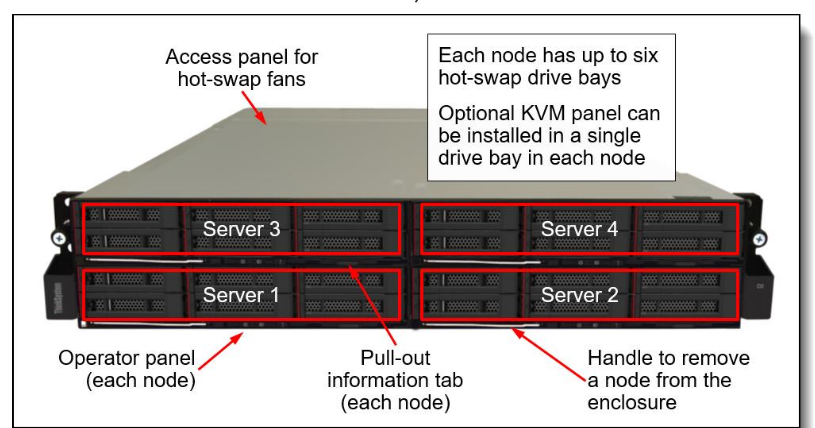
- Lenovo SD530 server overview:
  - https://lenovopress.lenovo.com/lp0635-thinksystem-sd530-server-xeon-sp-gen-1

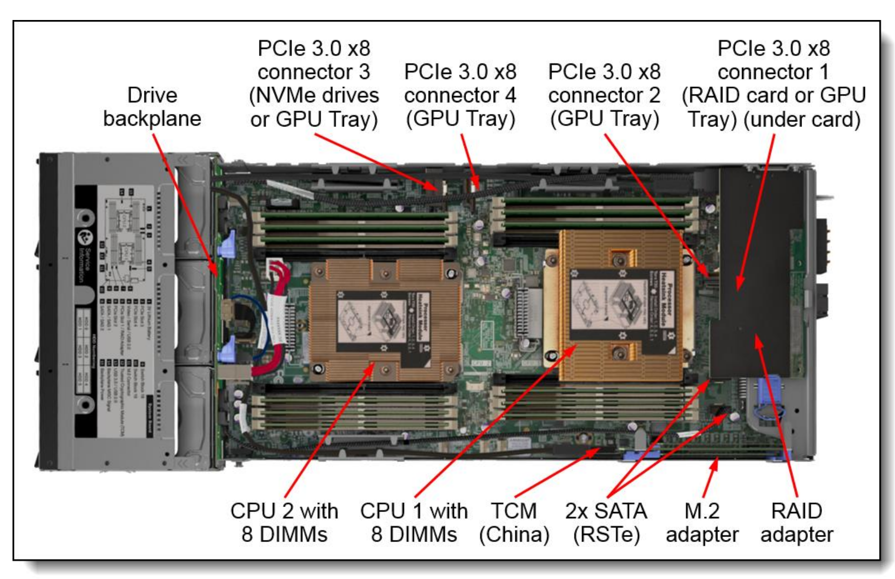
- 4 nodes per 2U enclosure
- Up to 84 nodes per 42U rack (21 enclosures)
- Up to 84*16*2 = 2688 cores per rack with Xeon Gold SKL-6130

- 4 racks = 336 nodes = 10752 cores
- Peak performance = 722 TF/s
- HPL performance = ~560 TF/s
- Cumulated STREAM performance = 62.2 TB/s

Four ThinkSystem SD530 servers installed in a D2 Enclosure

Front view of the ThinkSystem D2 Enclosure





Internal view of the SD530 compute node

2023 Lenovo Internal. All rights reserved.

Présentation de la machine Lenovo “Curta” du MCIA

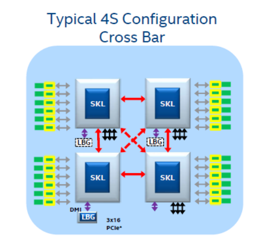
- Lenovo SR950 server overview:
  - https://lenovopress.lenovo.com/lp0647-thinksystem-sr950-server-xeon-sp-gen-1

- 4 Xeon Gold SKL-6130 processors per 4U (up to 8 maximum)
- Mesh interconnect between processors
- 3TB per node

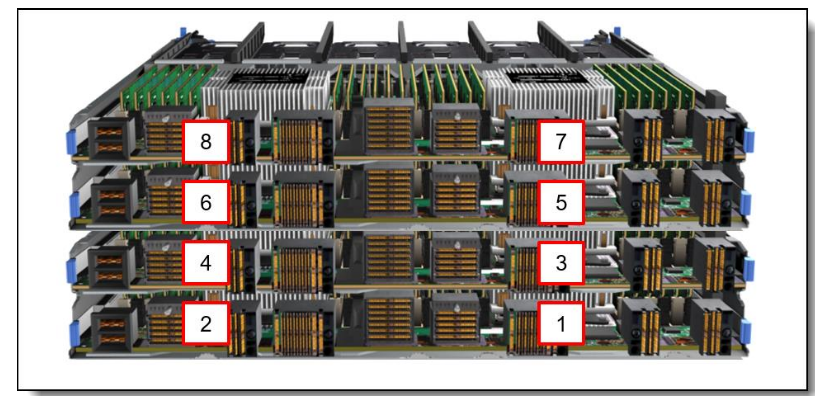
- 4 nodes = 128 cores with 12 TB of total memory

Lenovo ThinkSystem SR950

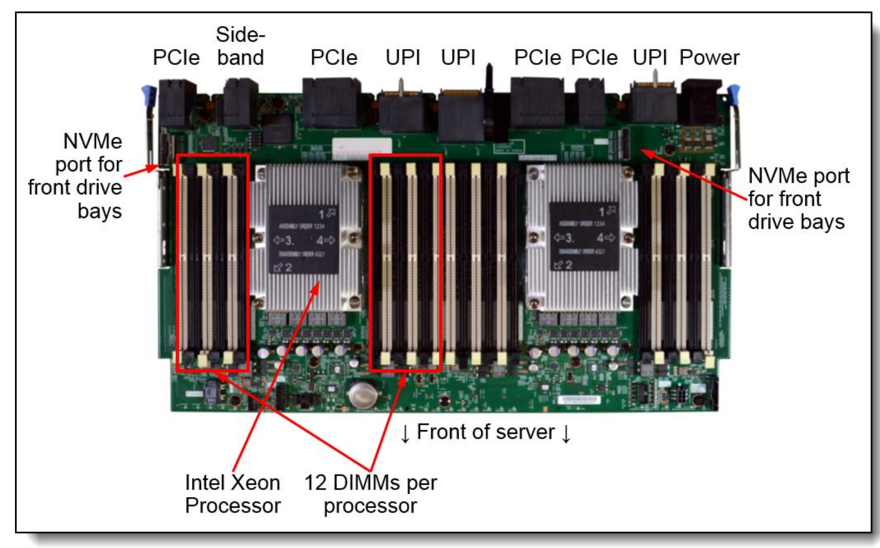
Typical 4S Configuration
Cross Bar



Processor numbering (viewed from the rear of the compute trays)



Compute system board



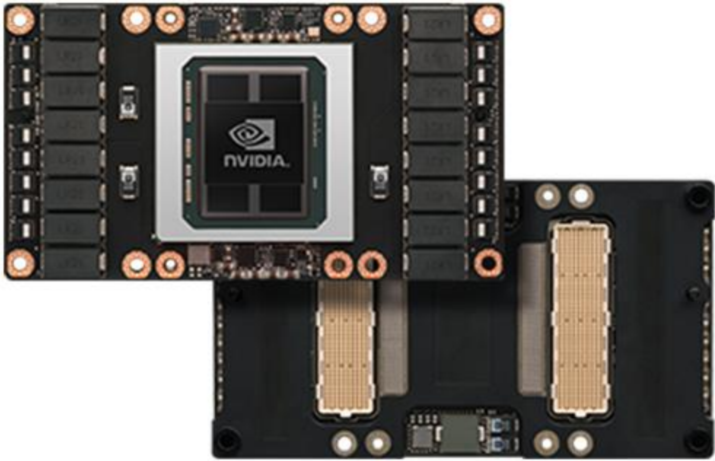
Lenovo 2023 Lenovo Internal. All rights reserved.

Présentation de la machine Lenovo “Curta” du MCIA

- Lenovo SR650 server overview:
  – https://lenovopress.lenovo.com/lp1050-thinksystem-sr650-server

- 2 Xeon Gold SKL-6130 processors and 2 GPUs per 2U
- nVidia P100 GPUs (12GB)
- 192GB per node

- 4 nodes = 128 cores , 8 GPUs ,1.5 TB of total memory
- HPL GPU performance = 28 TF/s
- Cumulated STREAM GPU performance = 5.6 TB/s



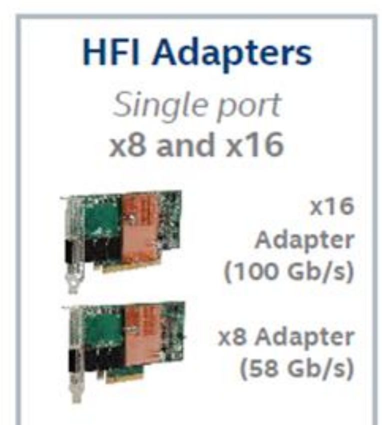
2023 Lenovo Internal. All rights reserved.

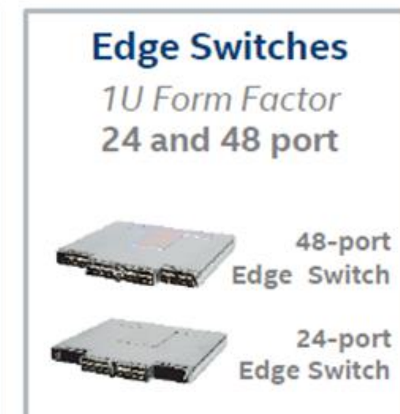
Présentation de la machine Lenovo “Curta” du MCIA

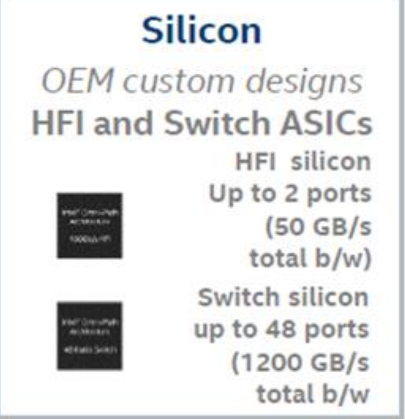
- Interconnect using **Intel Omni-Path (OPA)** architecture

- Up to 100Gb/s network bandwidth per adapter
- Low latency

- Re-use of existing OpenFabrics Alliance software
- Fully supported by IntelMPI software



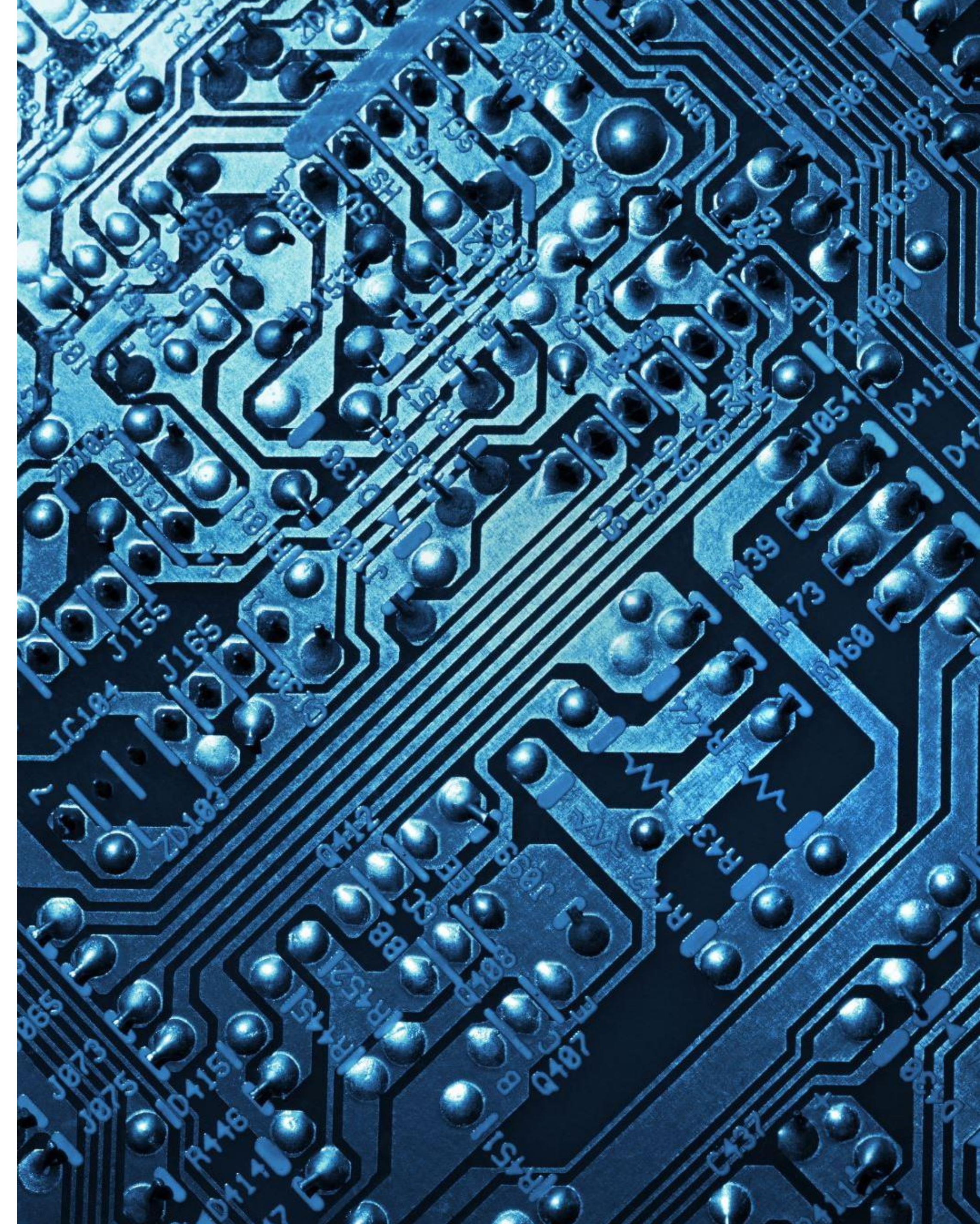




2023 Lenovo Internal. All rights reserved.

Concepts généraux pour une bonne performance des applications scientifiques

- Processor
  - Vector units
  - Frequencies
  - Hyper-Threading
- Memory
  - Affinity
- Network
- Storage
- Parallelism
- Power consumption
- Limiting factors
- Hardware evolution



Lenovo 2023 Lenovo Internal. All rights reserved.

General concepts

- The processor is the heart of the performance of HPC applications

- Processor **Intel Xeon Gold "Skylake" 6130**:
  - **16 cores** @ **2.1GHz** (nominal frequency) and **3.7GHz** (Turbo frequency)
    - Marketing numbers, see next slide for reality
  - **HyperThreading** allows using 2 threads per physical core
  - 22MB cache L3
  - **TDP** 125W
  - 2 * **AVX512** units, FMA…

- Single node performance:
  - Peak = 2 * 16 * 32 * 2.1 = 2150.4 GF/s
  - HPL = 1705 GF/s (79% of Peak)
  - STREAM Triad = 185 GB/s (5.9 GB/s per core)
  - HPCG = 35 GF/s

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                32
On-line CPU(s) list:   0-31
Thread(s) per core:    2
Core(s) per socket:    16
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 85
Model name:            Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz
Stepping:              4
CPU MHz:               1000.000
CPU max MHz:           2101.0000
CPU min MHz:           1000.0000
BogoMIPS:              4200.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              22528K
NUMA node0 CPU(s):     0-31
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse
2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid ape
rfmperf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2
x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb cat_l3 cdp_l3 invpcid
_single pti retpoline intel_ppin intel_pt mba rsb_ctxsw tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2
smep bmi2 erms invpcid rtm cqm mpx rdt_a avx512f avx512dq rdseed adx smap clflushopt clwb avx512cd avx512bw avx512vl xsaveopt xs
avec xgetbv1 xsaves cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local dtherm ida arat pln pts pku ospke
```

Lenovo 2023 Lenovo Internal. All rights reserved.

General concepts

- **Processor : Vector units**
- SIMD - Single Instruction Multiple Data
  - The same operation is applied simultaneously multiple inputs

- Compilers can generate vector instructions with appropriate options
- Intel MKL library transparently provides access to optimal vector units of hardware used
- Example of codes using AVX512:
  - HPL, Dgemm (Large matrix multiplications)
  - Gromacs, Lammps, Namd, Quantum-Espresso
  - AI codes (small matrix multiplications)
  - Cryptography…

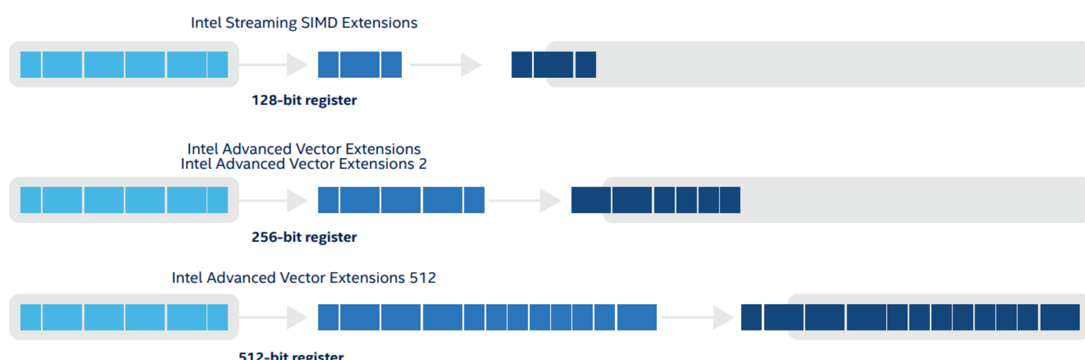
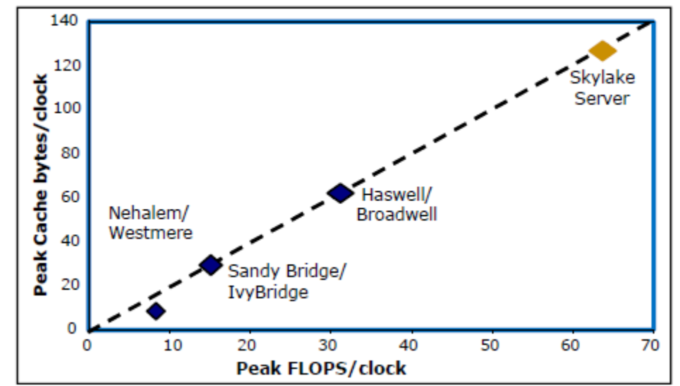


Figure 1. Illustration of the differences in register size and compute efficiency between Intel Streaming SIMD Extensions (Intel SSE), Intel AVX2, and Intel AVX-512

Skylake Server Peak FLOPS



uArch	Instruction Set	SP FLOPs per cycle	DP FLOPs per cycle
Nehalem	SSE (128-bits)	8	4
Sandy Bridge	AVX (256-bits)	16	8
Haswell / Broadwell	AVX2 & FMA	32	16
Skylake Server	AVX512 & FMA	64	32

Skylake Server Peak Compute Throughput

- AVX-512 with 2 FMAs per core provide 2x peak FLOPs/cycle of Haswell/Broadwell
- 2x cache bandwidth to feed wider vector units
  - 128-byte/64-byte load/store from L1
  - 2x L2 bandwidth

https://www.intel.com/content/www/us/en/architecture-and-technology/avx-512-animation.html

2023 Lenovo Internal. All rights reserved.

General concepts

- **Processor : Vector units**

- SIMD - Single Instruction Multiple Data
  – The same operation is applied simultaneously multiple inputs
- SSE - Streaming SIMD Extensions
  – Further iterations produced SSE2, SSE3, SSE4.1, SSE4.2
  – 128 bit XMM registers (4 floats, 2 doubles))
- AVX - Advanced Vector Extensions
  – 256 bit wide registers (YMM), supports 3 operand instructions
  – AVX-512 has 512-bit registers (ZMM)

Intel® AVX -512

Built-in acceleration and outstanding performance

3rd Gen Intel Xeon Scalable processors are the only data center CPU with 512-bit instruction processing. This wider vectorization speeds computation processes per clock, increasing frequency over the prior generation.

AVX 512 takes advantage of the processors' increased memory bandwidth new core architecture, improved frequency management .Additionally, 2 [1]FMA throughput is now available across the Platinum, Gold and Silver skus

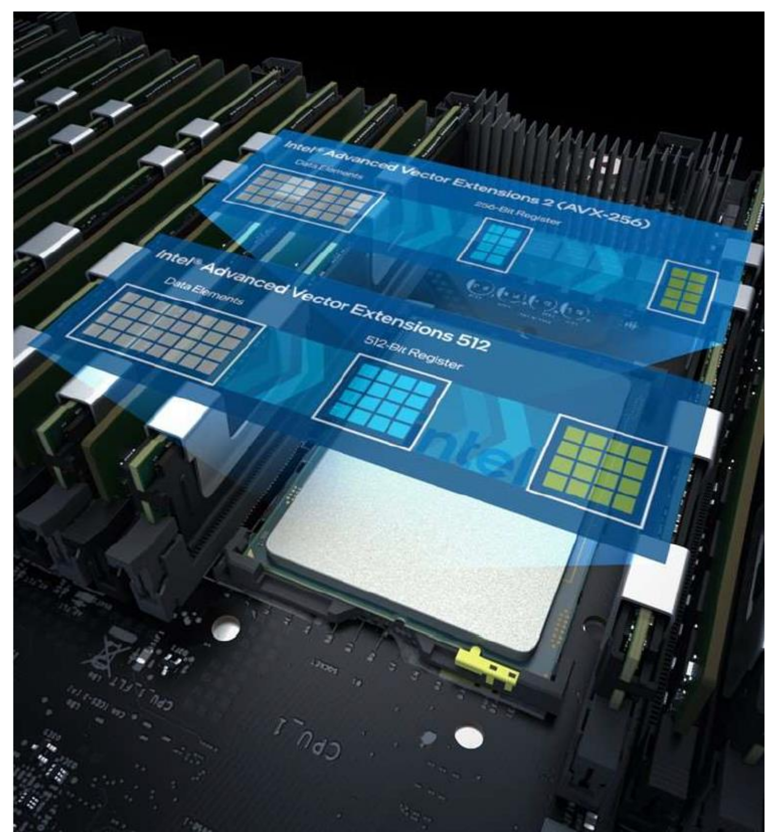
The latest version of Intel AVX-512 is easy to enable by using Intel tools, Intel compilers, Intel Math Kernel Library (Intel MKL), and Intel oneAPI Data Analytics Library. All of these have Intel AVX-512 built right in.

Vectorization and can dramatically increase performance for neural network processing, and for other operations that can be parallelized in this way.

2x More Registers vs. AVX-256

Learn more at intel.com/avx512

[1] Intel® AVX 2.0 delivers 16 double precision and 32 single precision floating point operations per second per clock cycle within the 256-bit vectors, with up to two 256-bit fused-multiply add (FMA) units. For workloads and configurations visit www.Intel.com/PerformanceIndex. Results may vary. Check full sku configurations for feature availability.



2023 Lenovo Internal. All rights reserved.

General concepts

- **Processor : Vector units**

- Vectorization can be enabled by:
  - Using compiler vectorization options (all compilers have vectorization options)
  - Using an already vectorized library (MKL, Libint, fftw…)

- How to check vectorization is occurring?
  - Use compiler options for reporting
    - "-qopt-report=5"
  - Check produced assembly code
    - Registers : zmm = AVX512 , ymm = AVX2 , xmm = SSE

- A portion of code can be almost fully vectorized, but you may not see the impact of vectorization:
  - Memory bandwidth bound code like STREAM
  - Data structure or size may limit improvement : array so small that time in reminder loop is similar that vector loop
  - Badly vectorized code : lot of "shift" instructions compared to computing instructions
  - Vectorized portion is very small compared to total application (profile your code execution!)

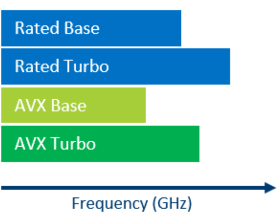
Lenovo 2023 Lenovo Internal. All rights reserved.

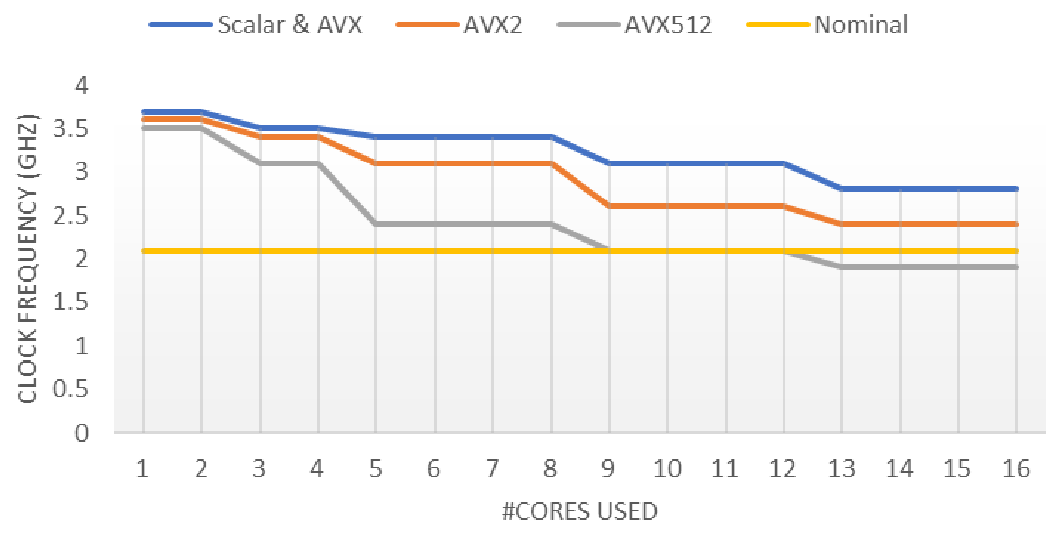
General concepts

- **Processor : Frequencies**
  - CPU speed is determined by how many calculations the processor can perform per cycle.
  - Clock speed is expressed in gigahertz — billions of cycles per second.
    - Higher clock speeds generate more heat and consume more power
  - Intel Turbo Boost technology enables processors to safely and efficiently increase clock speed beyond their usual operating limits.
  - Peak performance (GF/s) = #cores * #DP-Flops-per-cycle * frequency(GHz)
    - For Icelake 6130 CPU: 16*32*2.1 = 1075.2 GF/s

Intel® AVX* and Intel® Turbo Boost Technology 2.0

- AVX-optimized workloads still see significant performance benefit (when compared to non-AVX workloads)
- Some processors with turbo may not achieve any or maximum turbo frequency when running Intel® AVX* instructions





Lenovo 2023 Lenovo Internal. All rights reserved.

General concepts

- **Processor : Frequencies**

Figure 4. Intel® Xeon® Processor Scalable Family Non-Intel AVX Turbo Frequencies

81xx and 61xx processors

					# of active cores / maximum core frequency in turbo mode (GHz)																											
SKU	Cores	LLC (MB)	TDP (W)	Base non-AVX Core Frequency (GHz)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
8176	28	38.50	165	2.1	3.8	3.8	3.6	3.6	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.4	3.4	3.4	3.4	3.1	3.1	3.1	3.1	2.9	2.9	2.9	2.9	2.8	2.8	2.8	2.8
8170	26	35.75	165	2.1	3.7	3.7	3.5	3.5	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.3	3.3	3.3	3.3	3.0	3.0	3.0	3.0	2.8	2.8	2.8	2.8	2.8	2.8		
8164	26	35.75	150	2.0	3.7	3.7	3.5	3.5	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.2	3.2	3.2	3.2	2.9	2.9	2.9	2.9	2.7	2.7	2.7	2.7	2.7	2.7		
8160	24	33.00	150	2.1	3.7	3.7	3.5	3.5	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.2	3.2	3.2	3.2	3.0	3.0	3.0	3.0	2.8	2.8	2.8	2.8				
6152	22	30.25	140	2.1	3.7	3.7	3.5	3.5	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.1	3.1	3.1	3.1	2.9	2.9	2.9	2.9	2.8	2.8						
6138	20	27.50	125	2.0	3.7	3.7	3.5	3.5	3.4	3.4	3.4	3.4	3.2	3.2	3.2	3.2	2.9	2.9	2.9	2.9	2.7	2.7	2.7	2.7								
6140	18	24.75	140	2.3	3.7	3.7	3.5	3.5	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.1	3.1	3.1	3.1	3.0	3.0										
8153	16	22.00	125	2.0	2.8	2.8	2.6	2.6	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.3	2.3	2.3	2.3												
6130	16	22.00	125	2.1	3.7	3.7	3.5	3.5	3.4	3.4	3.4	3.4	3.1	3.1	3.1	3.1	2.8	2.8	2.8	2.8												

- 8176, 8170, 8160, and 6140 have 1.5TB/socket memory capacity versions (8180M, 8170M, 8160M, and 6140M – not listed above) with identical frequencies

Figure 5. Intel® Xeon® Processor Scalable Family Intel AVX 2.0 Turbo Frequencies

81xx and 61xx, processors.

					# of active cores / maximum core frequency in turbo mode (GHz)																											
SKU	Cores	LLC (MB)	TDP (W)	Base AVX 2.0 Core Frequency (GHz)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
8176	28	38.50	165	1.7	3.6	3.6	3.4	3.4	3.3	3.3	3.3	3.3	3.3	3.3	3.3	3.3	2.9	2.9	2.9	2.9	2.7	2.7	2.7	2.7	2.5	2.5	2.5	2.5	2.4	2.4	2.4	2.4
8170	26	35.75	165	1.7	3.6	3.6	3.4	3.4	3.3	3.3	3.3	3.3	3.2	3.2	3.2	3.2	2.8	2.8	2.8	2.8	2.6	2.6	2.6	2.6	2.4	2.4	2.4	2.4	2.4	2.4		
8164	26	35.75	150	1.6	3.6	3.6	3.4	3.4	3.3	3.3	3.3	3.3	3.0	3.0	3.0	3.0	2.7	2.7	2.7	2.7	2.5	2.5	2.5	2.5	2.3	2.3	2.3	2.3	2.3	2.3		
8160	24	33.00	150	1.8	3.6	3.6	3.4	3.4	3.3	3.3	3.3	3.3	3.2	3.2	3.2	3.2	2.9	2.9	2.9	2.9	2.6	2.6	2.6	2.6	2.5	2.5	2.5	2.5				
6152	22	30.25	140	1.7	3.6	3.6	3.4	3.4	3.3	3.3	3.3	3.3	3.0	3.0	3.0	3.0	2.7	2.7	2.7	2.7	2.4	2.4	2.4	2.4	2.4	2.4						
6138	20	27.50	125	1.6	3.6	3.6	3.4	3.4	3.2	3.2	3.2	3.2	2.7	2.7	2.7	2.7	2.5	2.5	2.5	2.5	2.3	2.3	2.3	2.3								
6140	18	24.75	140	1.9	3.6	3.6	3.4	3.4	3.3	3.3	3.3	3.3	3.0	3.0	3.0	3.0	2.7	2.7	2.7	2.7	2.6	2.6										
8153	16	22.00	125	1.6	2.7	2.7	2.5	2.5	2.4	2.4	2.4	2.4	2.2	2.2	2.2	2.2	2.0	2.0	2.0	2.0												
6130	16	22.00	125	1.7	3.6	3.6	3.4	3.4	3.1	3.1	3.1	3.1	2.6	2.6	2.6	2.6	2.4	2.4	2.4	2.4												

- 8176, 8170, 8160, and 6140 have 1.5TB/socket memory capacity versions (8180M, 8170M, 8160M, and 6140M – not listed above) with identical frequencies

Figure 6. Intel® Xeon® Processor Scalable Family Intel AVX-512 Turbo Frequencies

81xx and 61xx processors.

					# of active cores / maximum core frequency in turbo mode (GHz)																											
SKU	Cores	LLC (MB)	TDP (W)	Base AVX-512 Core Frequency (GHz)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
8176	28	38.50	165	1.3	3.5	3.5	3.3	3.3	3.0	3.0	3.0	3.0	2.6	2.6	2.6	2.6	2.3	2.3	2.3	2.3	2.1	2.1	2.1	2.1	2.0	2.0	2.0	2.0	1.9	1.9	1.9	1.9
8170	26	35.75	165	1.3	3.5	3.5	3.3	3.3	2.9	2.9	2.9	2.9	2.5	2.5	2.5	2.5	2.2	2.2	2.2	2.2	2.1	2.1	2.1	2.1	1.9	1.9	1.9	1.9	1.9	1.9		
8164	26	35.75	150	1.2	3.5	3.5	3.3	3.3	2.8	2.8	2.8	2.8	2.4	2.4	2.4	2.4	2.1	2.1	2.1	2.1	1.9	1.9	1.9	1.9	1.8	1.8	1.8	1.8	1.8	1.8		
8160	24	33.00	150	1.4	3.5	3.5	3.3	3.3	3.0	3.0	3.0	3.0	2.6	2.6	2.6	2.6	2.3	2.3	2.3	2.3	2.1	2.1	2.1	2.1	2.0	2.0	2.0	2.0				
6152	22	30.25	140	1.4	3.5	3.5	3.3	3.3	2.9	2.9	2.9	2.9	2.5	2.5	2.5	2.5	2.2	2.2	2.2	2.2	2.0	2.0	2.0	2.0	2.0	2.0						
6138	20	27.50	125	1.3	3.5	3.5	3.3	3.3	2.7	2.7	2.7	2.7	2.3	2.3	2.3	2.3	2.0	2.0	2.0	2.0	1.9	1.9	1.9	1.9								
6140	18	24.75	140	1.5	3.5	3.5	3.3	3.3	2.8	2.8	2.8	2.8	2.4	2.4	2.4	2.4	2.1	2.1	2.1	2.1	2.1	2.1										
8153	16	22.00	125	1.2	2.6	2.6	2.4	2.4	2.0	2.0	2.0	2.0	1.7	1.7	1.7	1.7	1.6	1.6	1.6	1.6												
6130	16	22.00	125	1.3	3.5	3.5	3.1	3.1	2.4	2.4	2.4	2.4	2.1	2.1	2.1	2.1	1.9	1.9	1.9	1.9												

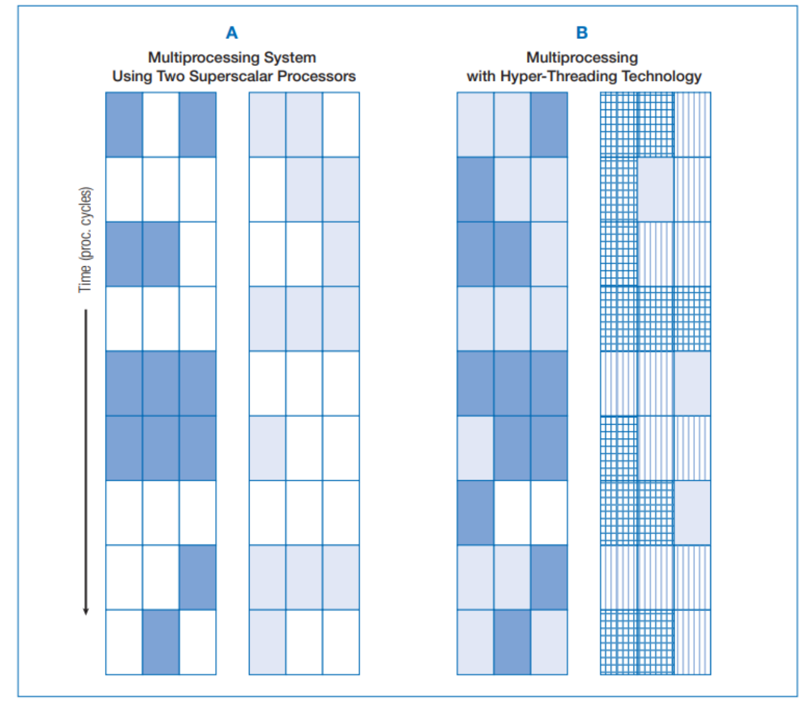
- 8176, 8170, 8160, and 6140 have 1.5TB/socket memory capacity versions (8180M, 8170M, 8160M, and 6140M – not listed above) with identical frequencies

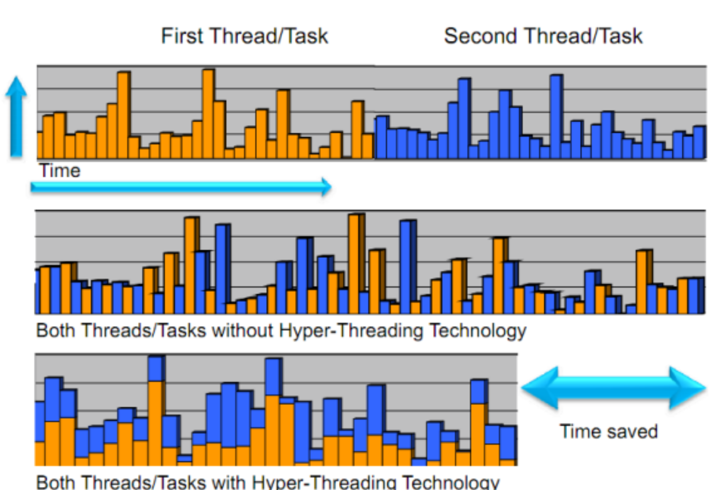
Specifications from Intel

Lenovo 2023 Lenovo Internal. All rights reserved.

General concepts

- **Processor : Hyper-Threading**
- This is Intel's implementation of "Simultaneous Multi Threading" SMT
- It allows 2 instructions flows (processes, threads) to execute on the same core and same hardware units
- Goal is to fill the "holes" in the instructions pipeline of the core and then increase average usage of the units:
  - Load, store, branches, integer and floating point units,…
- This can lead to significant performance improvement:
  - By doubling the number of threads or processes from one application on same number of processor cores
    - The 2 instructions flows are similar which can limit the global improvement
    - The same units are required by the 2 flows
  - By running 2 different applications on the same processor core
    - Potentially leads to the biggest improvements
    - The 2 instructions flows are different and may not need the same units to be used at same time
- From user perspective, it is transparent:
  - Operating systems (Linux…) shows 2 "virtual" cores for a "physical" core and they can be use as any core.
  - Default mode is set in the UEFI settings
    - Processors.HyperThreading=Enabled
  - It can be enabled and disabled dynamically thru Linux commands
    - "echo -n 1 > /sys/devices/system/cpu/cpu0/online"
    - But we recommend rebooting to avoid numbering mismatch or other side effects





Lenovo 2023 Lenovo Internal. All rights reserved.

General concepts

- **Processor : Hyper-Threading**
- If an application is not fully stressing a resource
  - Vector units, memory bandwidth, network bandwidth…
- Then it is a good candidate for testing impact of HT
  - Needs to have a level of parallelism:
    - OpenMP threads, MPI tasks, throughput execution

- HT impact depends also on
  - The input cases
  - The processor generation
  - The compiler version and optimizations
  - Pinning of threads/processes

- Example of "HT friendly" codes:
  - Gromacs : up to 20%
  - CP2K (with threads) : up to 15%
  - GRAPH500 (depends on implementation) : up to 20%
  - AVBP : up to 15%
  - NAMD, LAMMPS
  - Openssl or cryptographic codes in throughput mode

- **HT can provide performance improvement for free**

Hyper-Threading OFF

```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                  64
On-line CPU(s) list: 0-63
Thread(s) per core:   1
Core(s) per socket:    32
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 106
Model name:            Intel(R) Xeon(R)
Platinum 8352Y CPU @ 2.20GHz
Stepping:              6
CPU MHz:               2200.000
CPU max MHz:           2201.0000
CPU min MHz:           800.0000
BogoMIPS:              4400.00
Virtualization:        VT-x
L1d cache:             48K
L1i cache:             32K
L2 cache:              1280K
L3 cache:              49152K
NUMA node0 CPU(s):     0-31
NUMA node1 CPU(s):     32-63
…
```

Hyper-Threading ON

```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                  128
On-line CPU(s) list: 0-127
Thread(s) per core:   2
Core(s) per socket:    32
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 106
Model name:            Intel(R) Xeon(R)
Platinum 8352Y CPU @ 2.20GHz
Stepping:              6
CPU MHz:               2200.000
CPU max MHz:           2201.0000
CPU min MHz:           800.0000
BogoMIPS:              4400.00
Virtualization:        VT-x
L1d cache:             48K
L1i cache:             32K
L2 cache:              1280K
L3 cache:              49152K
NUMA node0 CPU(s):     0-31,64-95
NUMA node1 CPU(s):     32-63,96-127
…
```

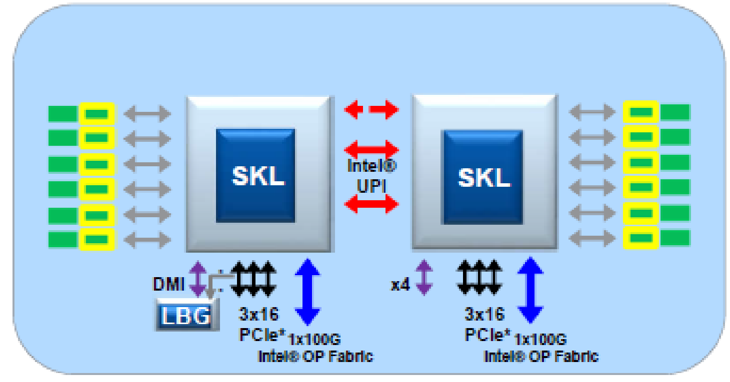
2023 Lenovo Internal. All rights reserved.

General concepts

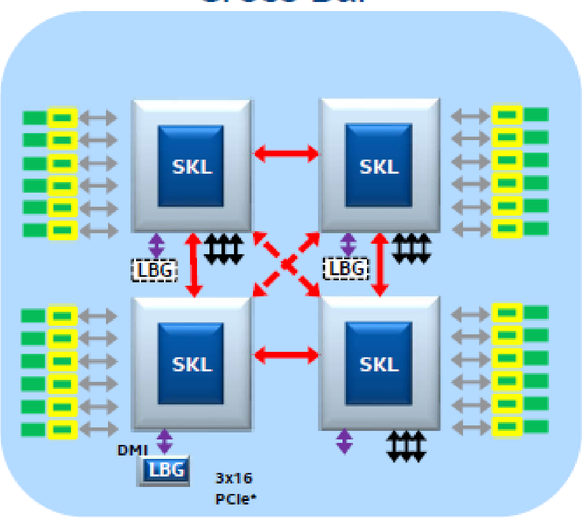
- **Memory**
- Each socket has its local main memory.
- Each socket can access main memory of other sockets.
- Local memory access is faster: **“Memory Affinity”**
  – Best performance when cores access memory local to their sockets.

- Memory is only allocated by OS when written to, not when allocated via malloc
  – Known as "first-touch" policy. Memory is allocated on memory local to the first core to write ("touch") to it.

Typical 2S Configuration



Typical 4S Configuration
Cross Bar

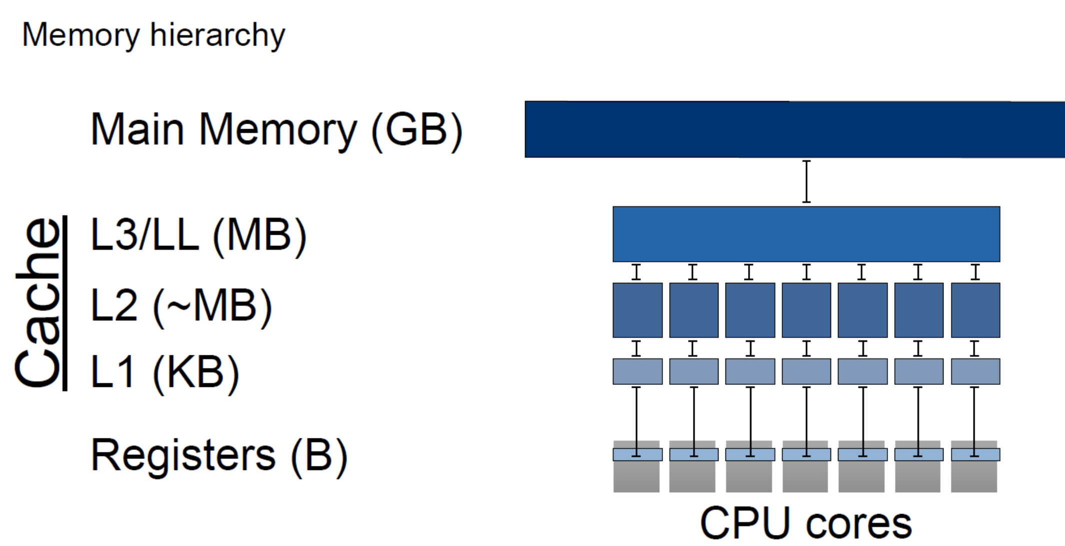


Lenovo 2023 Lenovo Internal. All rights reserved.

General concepts

- **Memory**

- Memory is organized by the hardware into pages. Typically a page size is 4096 bytes.
- Memory is not allocated all at once by first-touch per page.
- If cores on different sockets touch different pages that are part of the same array, the array will be physically allocated across different sockets' main memory.
- Logical access will be unaffected but some cores may require longer to access some elements than others.
- Best is when cores touch the pages they will need and do not access any other pages.

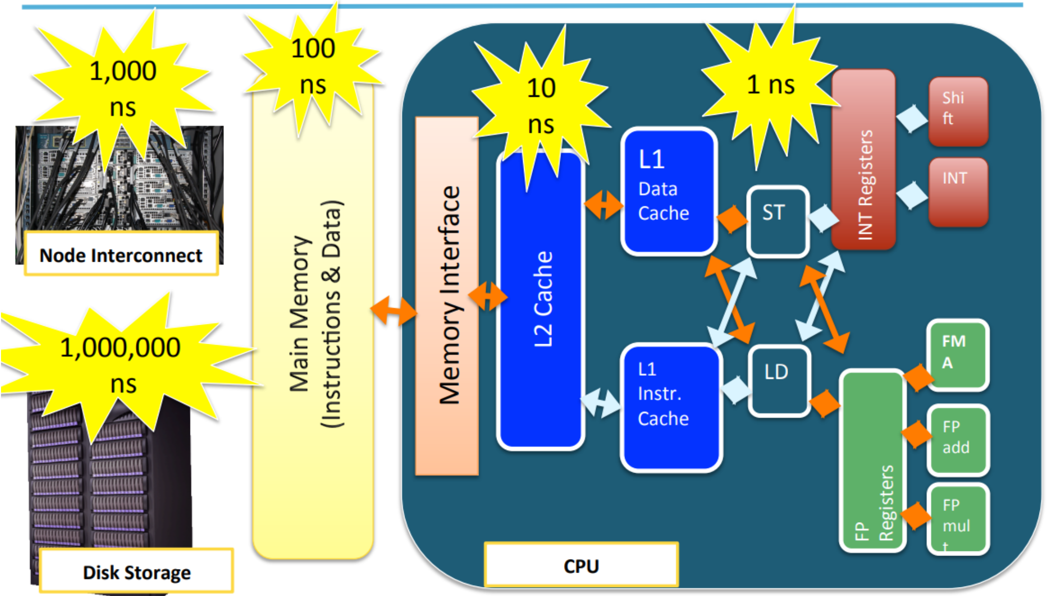


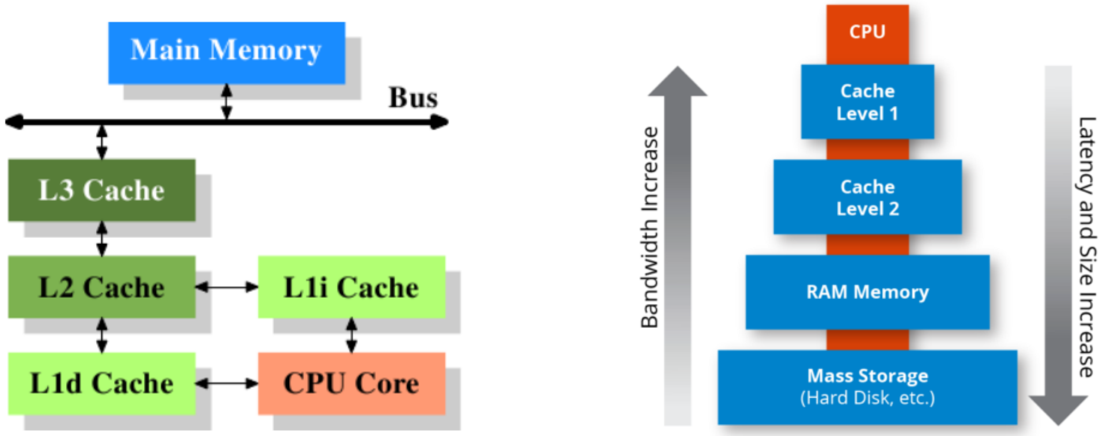
Lenovo 2023 Lenovo Internal. All rights reserved.

General concepts

- **Memory**

- Cache
  - Main memory is far from the CPU
  - Access is slow compared to data processing speeds
  - Cache is a smaller, but faster copy of data from main memory
  - CPU manages movement of data to and from main memory and cache





Lenovo 2023 Lenovo Internal. All rights reserved.

General concepts

- **Memory**

Sub-NUMA Clusters (SNC)

Prior generation supported Clusters-On-Die (COD)

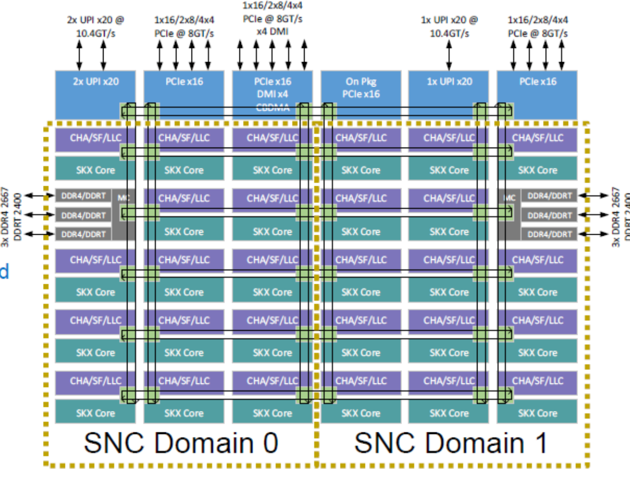
SNC provides similar localization benefits as COD, without some of its downsides

- Only one UPI caching agent required even in 2-SNC mode
- Latency for memory accesses in remote cluster is smaller, no UPI flow
- LLC capacity is utilized more efficiently in 2-cluster mode, no duplication of lines in LLC

Downside with SNC

- Addresses from remote cluster never get cached in local cluster LLC, resulting in larger latency compared to COD in some cases

	Clusters per socket	XPT/UPI Prefetch	
SNC off	1	No	No prefetch in UMA
1-SNC	1	Yes	Like SNC off, but can do prefetch
2-SNC	2	Yes	



Sub-NUMA Clusters – 2 SNC Example

Local SNC Access

Remote SNC Access

SNC 1

```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                64
On-line CPU(s) list:   0-63
Thread(s) per core:    1
Core(s) per socket:    32
Socket(s):             2
NUMA node(s):            2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 106
Model name:            Intel(R) Xeon(R)
Platinum 8352Y CPU @ 2.20GHz
Stepping:              6
CPU MHz:               2200.000
CPU max MHz:           2201.0000
CPU min MHz:           800.0000
BogoMIPS:              4400.00
Virtualization:        VT-x
L1d cache:             48K
L1i cache:             32K
L2 cache:              1280K
L3 cache:              49152K
NUMA node0 CPU(s):     0-31
NUMA node1 CPU(s):     32-63
…
```

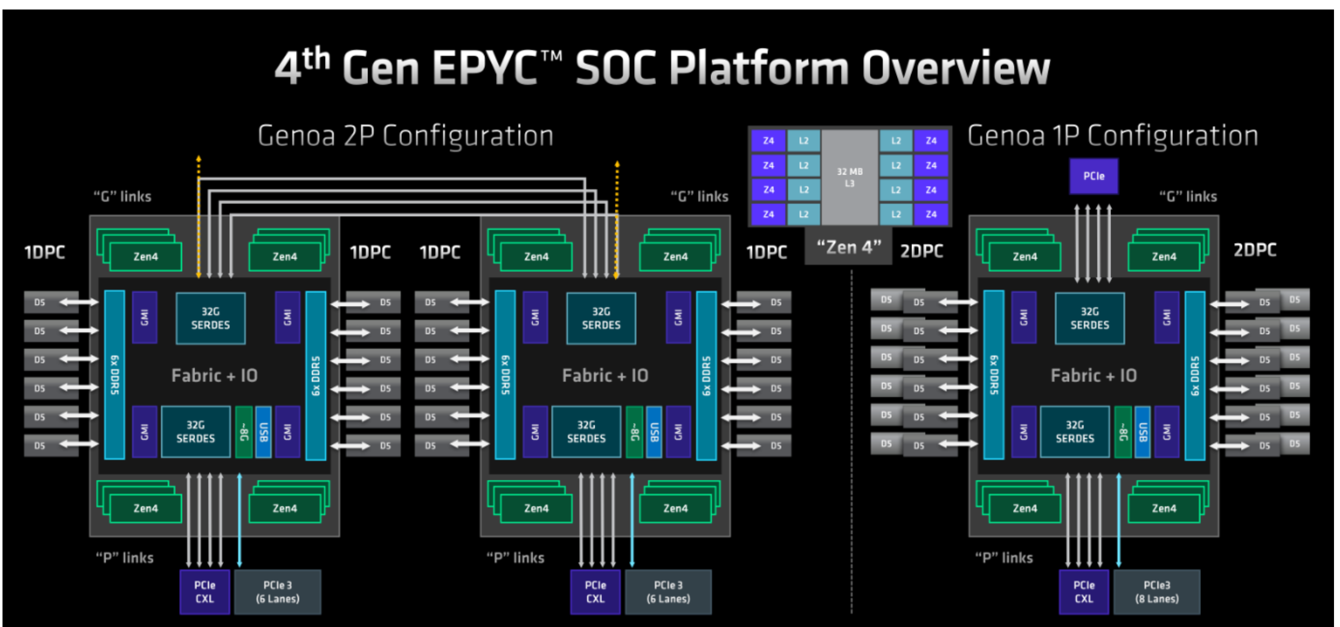
SNC 2

```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                64
On-line CPU(s) list:   0-63
Thread(s) per core:    1
Core(s) per socket:    32
Socket(s):             2
NUMA node(s):            4
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 106
Model name:            Intel(R) Xeon(R)
Platinum 8352Y CPU @ 2.20GHz
Stepping:              6
CPU MHz:               2200.000
CPU max MHz:           2201.0000
CPU min MHz:           800.0000
BogoMIPS:              4400.00
Virtualization:        VT-x
L1d cache:             48K
L1i cache:             32K
L2 cache:              1280K
L3 cache:              49152K
NUMA node0 CPU(s):     0-15
NUMA node1 CPU(s):     16-31
NUMA node2 CPU(s):     32-47
NUMA node3 CPU(s):     48-63
…
```

Lenovo 2023 Lenovo Internal. All rights reserved.

General concepts

- **Memory Affinity example**
- Naïve implementation of Laplacian (finite differences) without parallel initialization of the arrays used during computations
  - Initialization in serial (no OpenMP)
  - Computations in parallel (using OpenMP)



- Benchmark system with 2 * AMD 9654 processors : 2 * 96 cores , 8 numa nodes per node (NPS=4)
  - By default "Numa balancing" is disabled
    - (# cat /proc/sys/kernel/numa_balancing shows 0)
  - Threads are pinned to physical cores

- Execution
  - Numa Balancing OFF + Single thread : **27 min**
  - Numa Balancing OFF + 192 threads : **29 min**
    - So using all cores we are slower than with 1 core
    - All memory is allocated on numa node #0, and most threads are accessing remote memory : huge contention
  - Numa Balancing ON + 192 threads : **2.5 min**
    - The memory pages allocated to numa node #0 are moved to local memory by Linux, which improves significantly performance
  - Numa Balancing ON + 192 threads + numactl –interleave=all : **1 min 35 sec**
  - Modified source code (initialization in parallel with OpenMP) + 192 threads : **1 min 41 sec**

- **Memory affinity is critical for good performance**
  - Best is to implement smartly in source code (assuming Linux "first touch")
  - Can be somehow mitigated thru Linux tools : automatic "numa balancing" or manual "numactl" command.

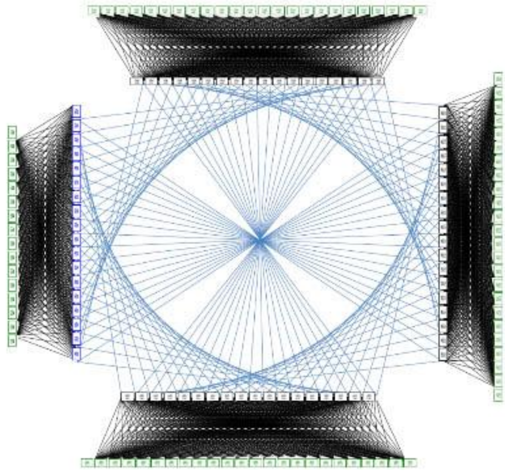
2023 Lenovo Internal. All rights reserved.

General concepts

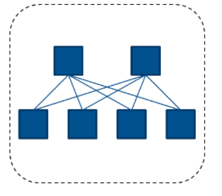
- **Interconnection Network**

- Many technologies and vendors exist:
  - nVidia Infiniband network (former Mellanox)
  - Intel Omni-Path (almost dead)
  - Proprietary
  - Gigabit Ethernet
  - …

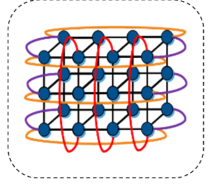
- Many topologies exist:
  - Flat Tree (with or without blocking factor)
  - Torus, DragonFly…



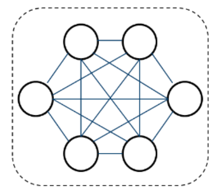
The Niagara Supercomputer and the Dragonfly+ Topology



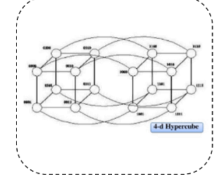
Fat Tree



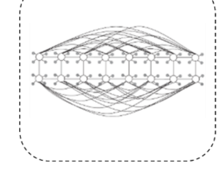
Torus



Dragonfly



Hypercube

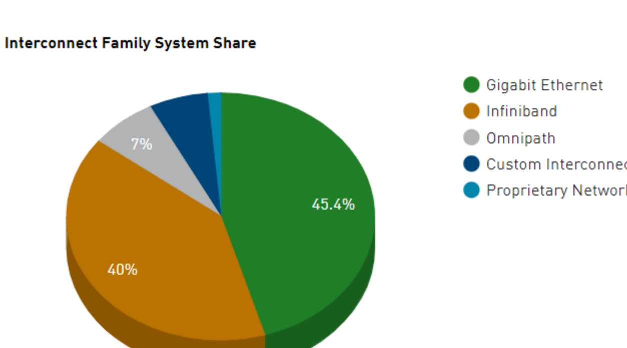


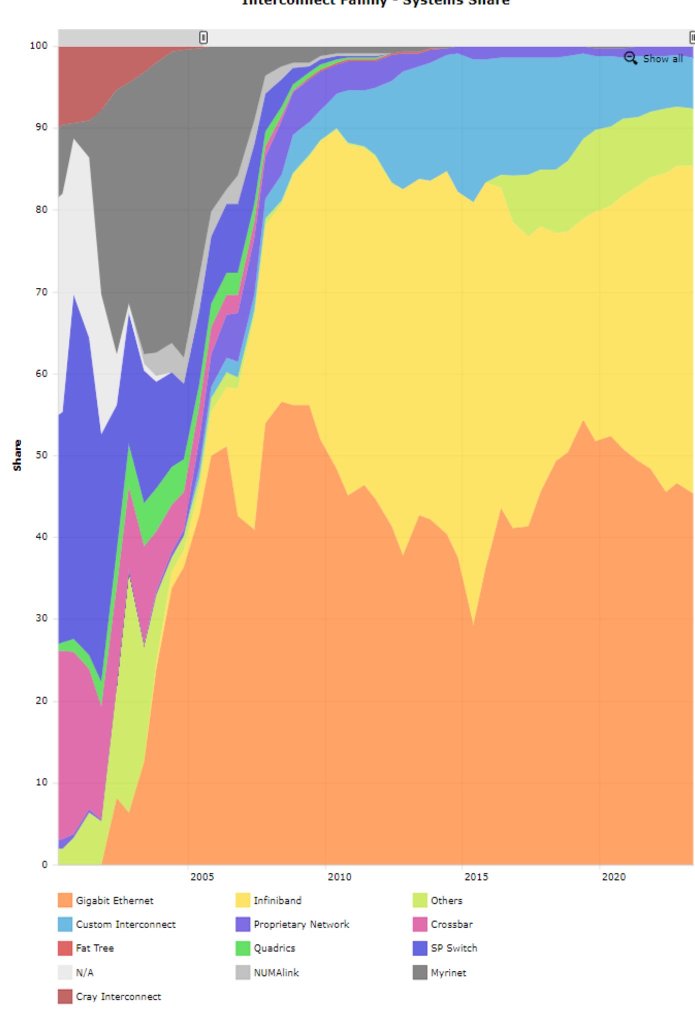
HyperX

Lenovo 2023 Lenovo Internal. All rights reserved.

General concepts

- **Interconnection Network :** TOP 500





	Interconnect Family	Count	System Share (%)	Rmax (GFlops)	Rpeak (GFlops)	Cores
1	Gigabit Ethernet	227	45.4	2,381,982,330	3,725,861,698	31,724,156
2	Infiniband	200	40	1,850,777,070	2,733,798,497	29,204,124
3	Omnipath	35	7	156,167,548	241,794,826	3,576,876
4	Custom Interconnect	31	6.2	333,456,918	498,207,618	21,940,716
5	Proprietary Network	7	1.4	516,640,800	626,257,837	9,015,296

Lenovo 2023 Lenovo Internal. All rights reserved.

https://www.top500.org

General concepts

- **Interconnection Network : Performance**
- Unidirectional bandwidth:
  - Infiniband :
    - FDR = 56 Gbit/s = ~6 GB/s
    - EDR = 100 Gbit/s = 12 GB/s
    - HDR = 200 Gbit/s = 24 GB/s
    - NDR = 400 Gbit/s = 48 GB/s
  - OmniPath: 100 Gbit/s = 12 GB/s
- Latency among all interconnects is very similar between close nodes : around 1µs
- Topology drives latency between far nodes
  - Number of links, switches,…

Table 1. Comparison of ICNs of the Top10 HPCs and two additional, representative ICNs with their main properties.

Rank	Computer Name	Manufacture	Interconnect Family	ICN	Bidirectional Bandwidth	Switch Radix	Latency	Topology
1	Fugaku [2,10]	Fujitsu	Proprietary Network	Tofu D	108.8 Gbps	20	≤0.54 µs	6D-Torus
2	Summit [2,17]	IBM	InfiniBand	EDR InfiniBand	200 Gbps	36	0.6 µs	Fat-tree
3	Sierra [2,17]	IBM/ NVIDIA/Mellanox	InfiniBand	EDR InfiniBand	200 Gbps	36	0.6 µs	Fat-tree
4	Sunway Taihu Light [2,11]	NRCPC	Custom Interconnect	Sunway	200 Gbps	36	1 µs	Fat-tree
5	Perlmutter [2,6]	HPE	Gigabit Ethernet	Slingshot-10	400 Gbps	64	N/A	Dragonfly
6	Selene [2,18,19,20]	NVIDIA	InfiniBand	HDR InfiniBand	400 Gbps	40	N/A	Fat-tree
7	Tianhe-2A [2,12,13,14]	NUDT	Custom Interconnect	TH Express-2	224 Gbps	24	0.6 µs	Fat-tree
8	JUWELS Booster Module [2,18,19,20]	Atos	InfiniBand	HDR InfiniBand	400 Gbps	40	N/A	Fat-tree
9	HPC5 [2,18,19,20]	Dell EMC	InfiniBand	HDR InfiniBand	400 Gbps	40	N/A	Fat-tree
10	Voyager-EUS2 [2,18,19,20]	Microsoft Azure	InfiniBand	HDR InfiniBand	400 Gbps	40	N/A	Fat-tree
42	Tera1000–2 [2,15]	Atos	Custom Interconnect	Bull BXI1.2	200 Gbps	48	<1 µs	N/A
N/A	N/A	NVIDIA	InfiniBand	NDR InfiniBand [18,19,20]	800 Gbps	64	≤1 µs	N/A

TOP 500 (June 2021)

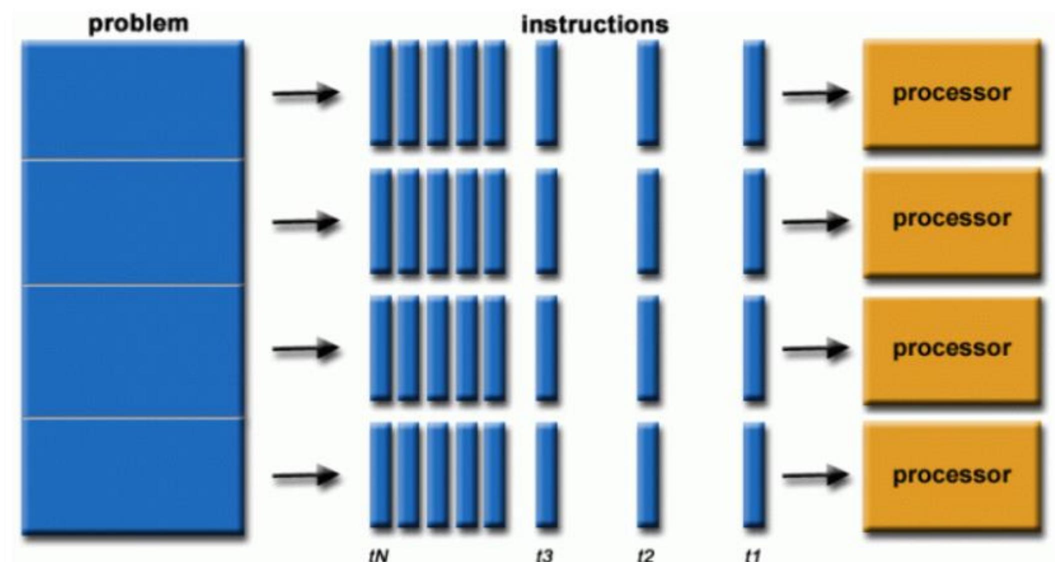
https://www.mdpi.com/2079-9292/11/9/1369

Lenovo 2023 Lenovo Internal. All rights reserved.

General concepts

- **Parallelism**
- Parallelism is the process of processing several sets of instructions simultaneously.

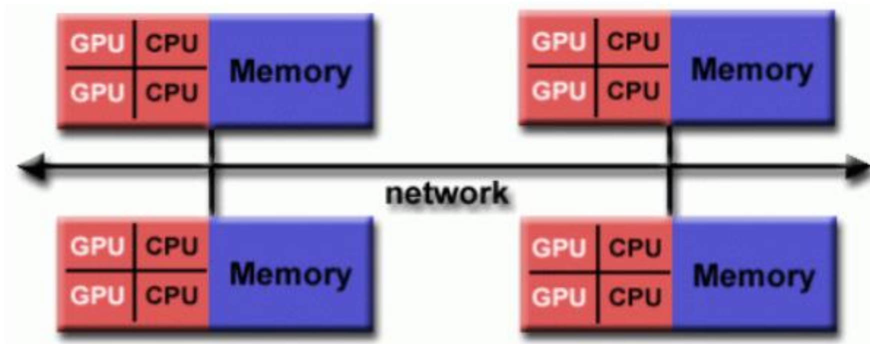
- Main Reasons for Using Parallel Programming
  - Save time and/or money
    - Using 16 cores instead of 1 core will ideally reduce execution time by 16x
  - Solve larger / more complex problems
    - Using 4 nodes allows application to use 4x more memory
  - Make better use of hardware
    - Hpc hardware is full of parallelism that must be used for good efficiency
  - Prepare the future
    - Trends in HPC are showing that
      - Processors and accelerators are getting more and more “cores”
      - Systems are continuing increasing in size

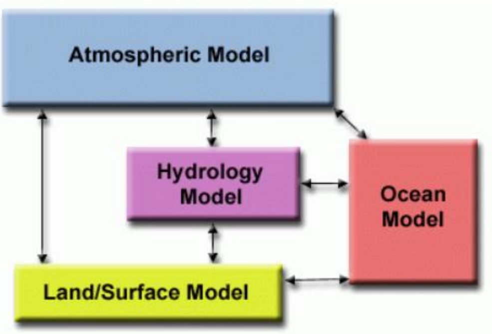


Lenovo 2023 Lenovo Internal. All rights reserved.

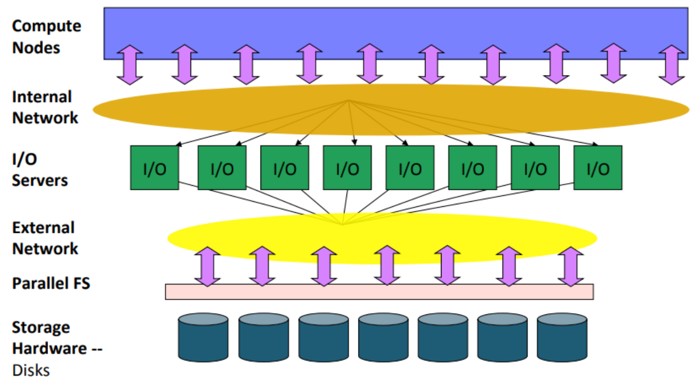
General concepts

- **Parallelism**
- Multiple levels of parallelism:
  - Processor core units : load, store, branch, scalar compute, vector units…
  - Hyper threading, SMT
  - Multi cores, multi processors
  - Accelerators
  - Multi nodes with each multiple processors and accelerators
  - Multiple applications coupled all together
    - Weather forecast models: Atmospheric + ocean + hydrology + ice + …
    - Each of the codes is using a multi nodes cluster
    - All codes are communicating to others, while internal communications occurs inside each code.





- HPC storage is also full of parallelism
  - Parallel filesystem
    - Spectrum Scale (GPFS), Lustre, WEKA…
  - Parallel I/O is critical for complex HPC simulation efficiency
    - PNETCDF, PHDF5, MPI I/O…



2023 Lenovo Internal. All rights reserved.

General concepts

- **Parallelism**

- Which tools to use for each level of parallelism ?
  - This list is not exhaustive, many alternatives exist

- Usually, complexity and time to learn and master is increasing from top to bottom of the list

Processor core units : load, store, branch, scalar compute, vector units…

- Compilers

Hyper threading & SMT, Multi cores, multi processors in a node

- OpenMP, MPI, throughput, SyCL…

Multi nodes with each multiple processors and accelerators

- MPI, PGAS, Charm++…

Accelerators

- Specific libraries : CUDA, ROCm, SyCL, OpenCL, OpenACC…

Multiple applications coupled all together

- Coupling library (OASIS, OpenPALM, LAM …)
- Own implementation

Lenovo 2023 Lenovo Internal. All rights reserved.

General concepts

- **Power Consumption**

- Current supercomputers are consuming huge amount of electricity for their operation:
  - #1 “Frontier” : 23 MW , #2 “Fugaku” : ~29 MW…
  - And this is not expected to decrease in near future: processors and GPUs are increasing their TDP :
    - Intel Sapphire Rapids processor is up to 400W
    - AMD Genoa/Bergamo is up to 400W
    - nVidia H100 is at 700W

- Electricity cost is increasing significantly, and this will continue
  - Global warming evidence and its consequences on reducing/stopping oil-based electricity production, war…

- Huge focus is now on **power efficiency for hpc centers**
  - Customer RFPs often ask for TCO analysis over multiple years with cost of operation included in the overall budget
  - Before it was mostly based on pure performance, without any consideration for operations costs

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	1,679.82	22,703
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	428.70	6,016
4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,824,768	238.70	304.47	7,404

https://www.top500.org/statistics/efficiency-power-cores/

https://tradingeconomics.com/france/electricity-price

Lenovo 2023 Lenovo Internal. All rights reserved.

General concepts

- **Power Consumption**

- Some solutions exist:
  - Liquid cooling
    - Hardware solution like Lenovo "Neptune" technology
    - https://www.lenovo.com/us/en/servers-storage/neptune/
    - Hot water can be reused for other usage (urban heating…)
  - EAR: Energy Management Framework
    - Software solution, almost transparent to users
    - https://www.bsc.es/sites/default/files/public/bscw2/content/software-app/technical-documentation/ear_sc19_november.pdf
    - https://lenovopress.lenovo.com/lp1646.pdf

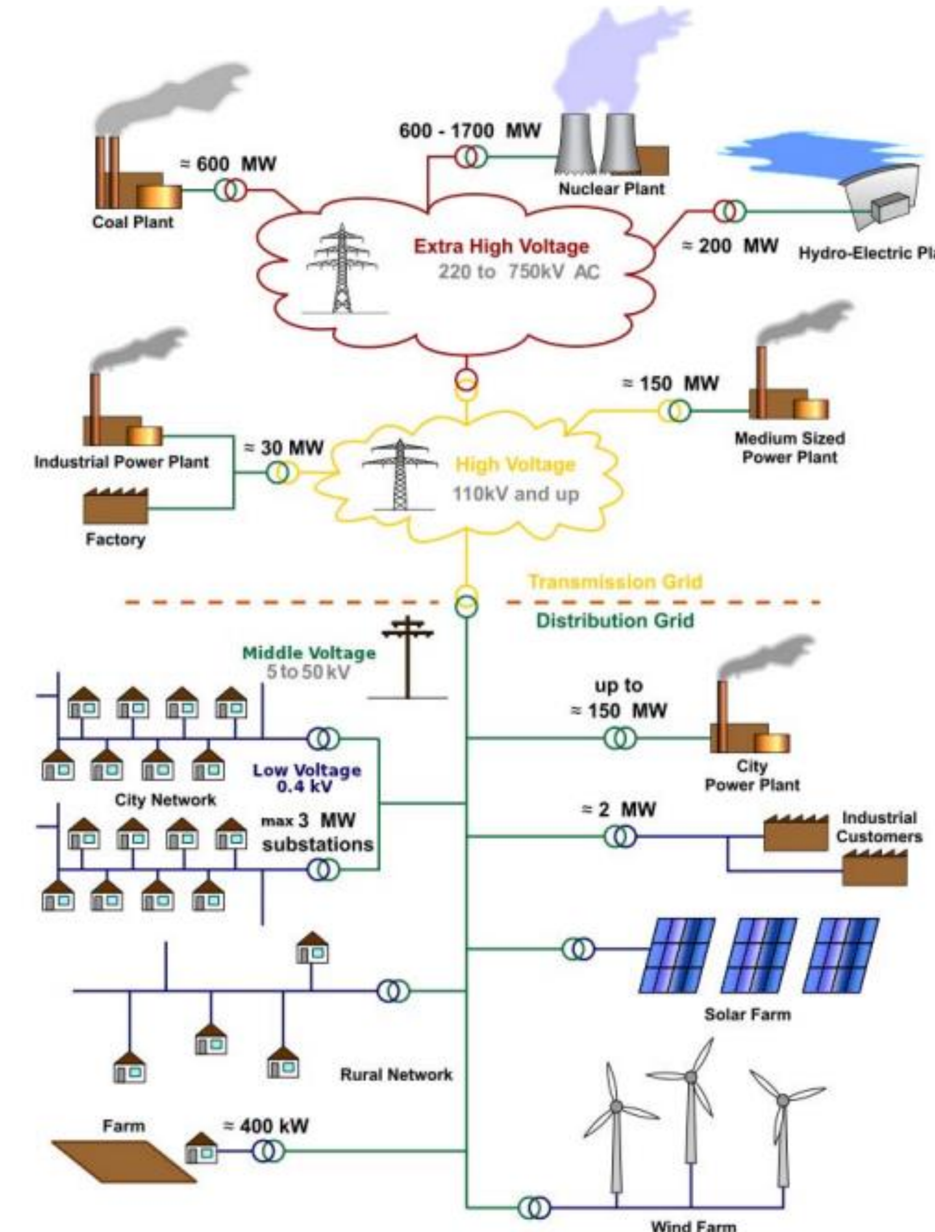


Figure 1: Electric power production, transmission and distribution

2023 Lenovo Internal. All rights reserved.

General concepts

- **Power Consumption**

- Power and energy data can be measured and retrieved from multiple sources:
  - Power DC from components with “sampling”:
    - `sudo ipmi-sensors --sensor-types=Current|grep -e "Sys Power" -e "CPU Power" -e "Sys Fan Pwr" |awk -F"|" '{print $4}'|xargs`
    - Provides min/average/median/max power
    - Allows computing energy value thanks to exécution time.
  - Energy AC & DC:
    - `sudo ipmitool raw 0x2e 0x81 0x66 0x4a 0 0x20 1 0x82 1 0x08`
    - `sudo ipmitool raw 0x2e 0x81 0x66 0x4a 0 0x20 1 0x82 0 0x08`
    - To be called before and after exécution (serial, mpi…), make the difference per node and sum accross all nodes to get total energy consumed.

- Higher level tools can show power and energy information
  - Job scheduler
  - System monitoring tools (ganglia…)
  - System management tools (Xclarity…)

Lenovo 2023 Lenovo Internal. All rights reserved.

General concepts

- **Hardware evolution** : Comparison of installed processor with current processor generation

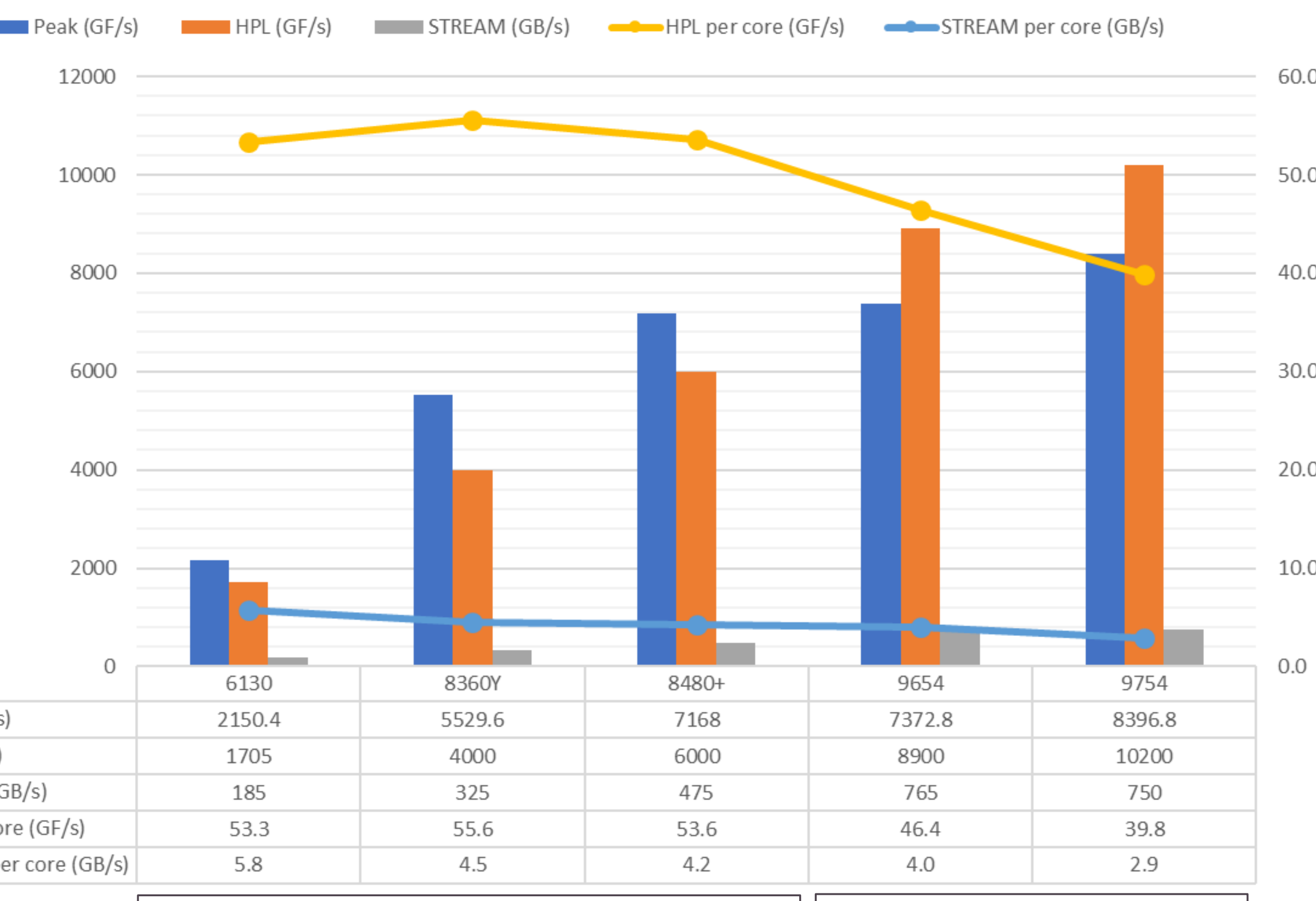
- Number of cores per socket is increasing a lot
  - From 16 up to 128 for AMD Bergamo CPU
- TDP is also increasing a lot
  - From 125W up to 400W
- Clock frequencies remain almost stable
  - Turbo frequencies on AMD processors are significantly higher than on Intel
- L3 cache size increases a lot
  - but not per core
- Memory bandwidth increases a lot
  - More memory channels, new DDR5 technology, increased dimm frequency

Manufacturer	Intel	Intel	Intel	AMD	AMD
Codename	**Skylake**	**Icelake**	**Saphirre Rapids**	**Genoa**	**Bergamo**
Model	**6130**	**8360Y**	**8480+**	**9654**	**9754**
Lithography (nm)	**14**	10	7	5	5
TDP (W)	**125**	250	350	400	400
#cores	**16**	36	56	96	128
Nominal frequency (GHz)	**2.1**	2.4	2	2.4	2.05
Max Turbo Frequency (GHz)	**3.7**	3.5	3.8	3.7	3.2
Max All cores Turbo Frequency (GHz)	**2.8**	3.1	3	3.55	3.2
Max All cores AVX512 Turbo Frequency (GHz)	**1.9**	2.6	2.5	3.55	3.2
L3 Cache (MB)	**22**	54	105	384	256
L3 Cache per core (MB)	**1.375**	1.500	1.875	4.000	2.000
Memory type	**DDR4**	DDR4	DDR5	DDR5	DDR5
Memory Max speed (MHz)	**2666**	3200	4800	4800	4800
#memory channels	**6**	8	8	12	12
Public price ($)	**1900**	5383	10710	11805	

Lenovo 2023 Lenovo Internal. All rights reserved.

General concepts

- **Performance evolution** on 2 sockets node
- Peak and HPL performance is growing a lot
  - HPL efficiency on AMD CPUs is higher than 100% because HPL benefits from Turbo unlike Intel
- Memory bandwidth (STREAM) is increasing significantly
  - From 185GB/s up to 765 GB/s
- But metrics per core are not progressing much
  - It depends on the processor model selected : 16 cores CPUs still exist in SPR and Genoa lines.



	6130	8360Y	8480+	9654	9754
Peak (GF/s)	2150.4	5529.6	7168	7372.8	8396.8
HPL (GF/s)	1705	4000	6000	8900	10200
STREAM (GB/s)	185	325	475	765	750
HPL per core (GF/s)	53.3	55.6	53.6	46.4	39.8
STREAM per core (GB/s)	5.8	4.5	4.2	4.0	2.9

Intel processors | AMD processors

Lenovo 2023 Lenovo Internal. All rights reserved.

General concepts

- **Optimize, but optimize what ?**

- There are multiple aspects of HPC application that can be optimized for:

  – **Execution time** (all codes) & performance metrics (ns/day for Gromacs, TF/s for HPL…)
    - Allow single execution to go as fast as possible

  – **Power consumption** and energy
    - Reduce energy consumed by the application

  – **Throughput** execution
    - Find best settings to run maximal number of parallel executions in shortest time

  – **TCO** optimization
    - Global center view : cooling, supercomputer, electricity cost…

Lenovo 2023 Lenovo Internal. All rights reserved.

General concepts

- **Limiting factors for optimal performance & Ways of optimizing**

- **Non optimal CPU performance**
- Compiler optimization
  - Play with compilers options to generate more optimized binary using hardware more efficiently
  - More details soon

- Source code optimization
  - Transform the source code to be more efficient:
    - Remove unnecessary operations
    - Improve data movement to reduce depend on memory subsystem or improve cache usage
    - Help compilers to make a better job…

- Algorithmic optimization
  - Make global changes in the source code to fully modify execution behavior and remove hard performance limit
    - Change algorithm used: iterative solver method…
    - Change data structure: AoS vs SoA, use sparse matrix formats vs dense, …
    - Usually takes long time to design and implement, but could provide highest improvement

- **Poor parallelization**
  - Add shared memory (OpenMP), distributed (MPI) and/or accelerated (CUDA/ROCm/SyCL…) parallelisms
  - Optimize synchronization, communications, efficiency…
  - Reduce load imbalance
  - Improve scalability

- **System not optimized**
  - Tune UEFI settings
  - Apply Linux tuning
    - Control frequencies, Turbo mode, memory balancing, thp…
  - Job scheduler customization
    - Allow multiple jobs per node
    - Shutdown nodes not used after some time
  - Storage tuning

- **Not efficient environment**
  - Ensure process and memory affinities
  - Tools update (compilers, libraries,…)
  - Libraries runtime
    - MPI environment variables…
    - I/O libraries

Lenovo 2023 Lenovo Internal. All rights reserved.

General concepts

- **Benchmarking Workflow for HPC Application Optimization**
- Standard iterative process →
- To be repeated until performance is estimated good enough
  - Criteria to be defined before starting
- Time consuming for people and systems
- No guarantee of huge improvement

Compilation
- Various opt options
- Inlining
- Vectorization
- OpenMP

Execution
- Various processors
- Various core counts
- Various frequencies
- Mix OpenMP * MPI

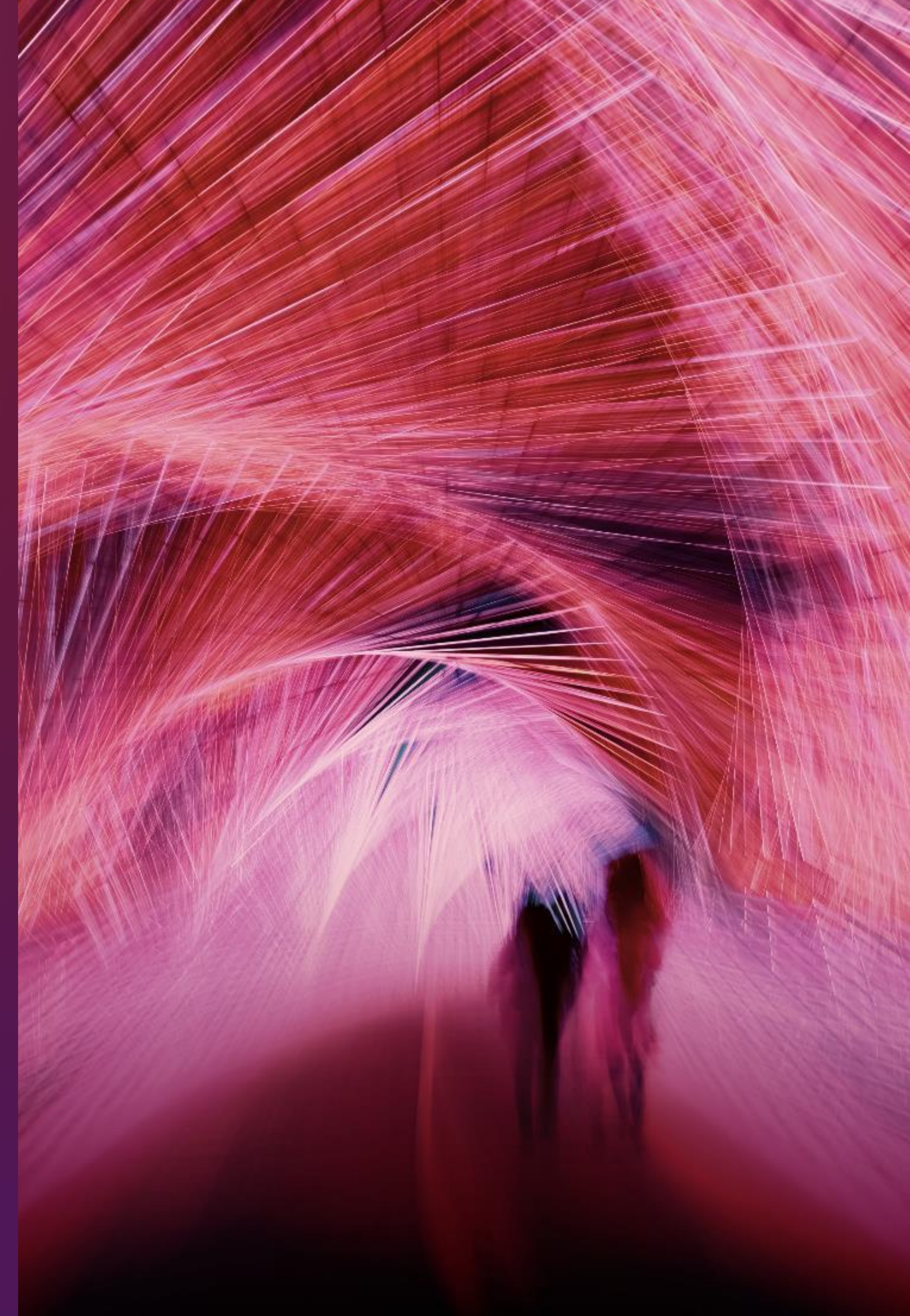
Profiling
- Flat profile
- OpenMP/MPI profile
- Hardware counters
- Power consumption

Analysis
- Characterization
- Roofline model
- Vectorization
- Load imbalance
- Perf/price/power analysis

Optimization
- Process/thread/memory Affinities
- Compiler options
- Vectorization
- OpenMP
- MPI
- System tuning
- HyperThreading
- Source code modifications

2023 Lenovo Internal. All rights reserved.

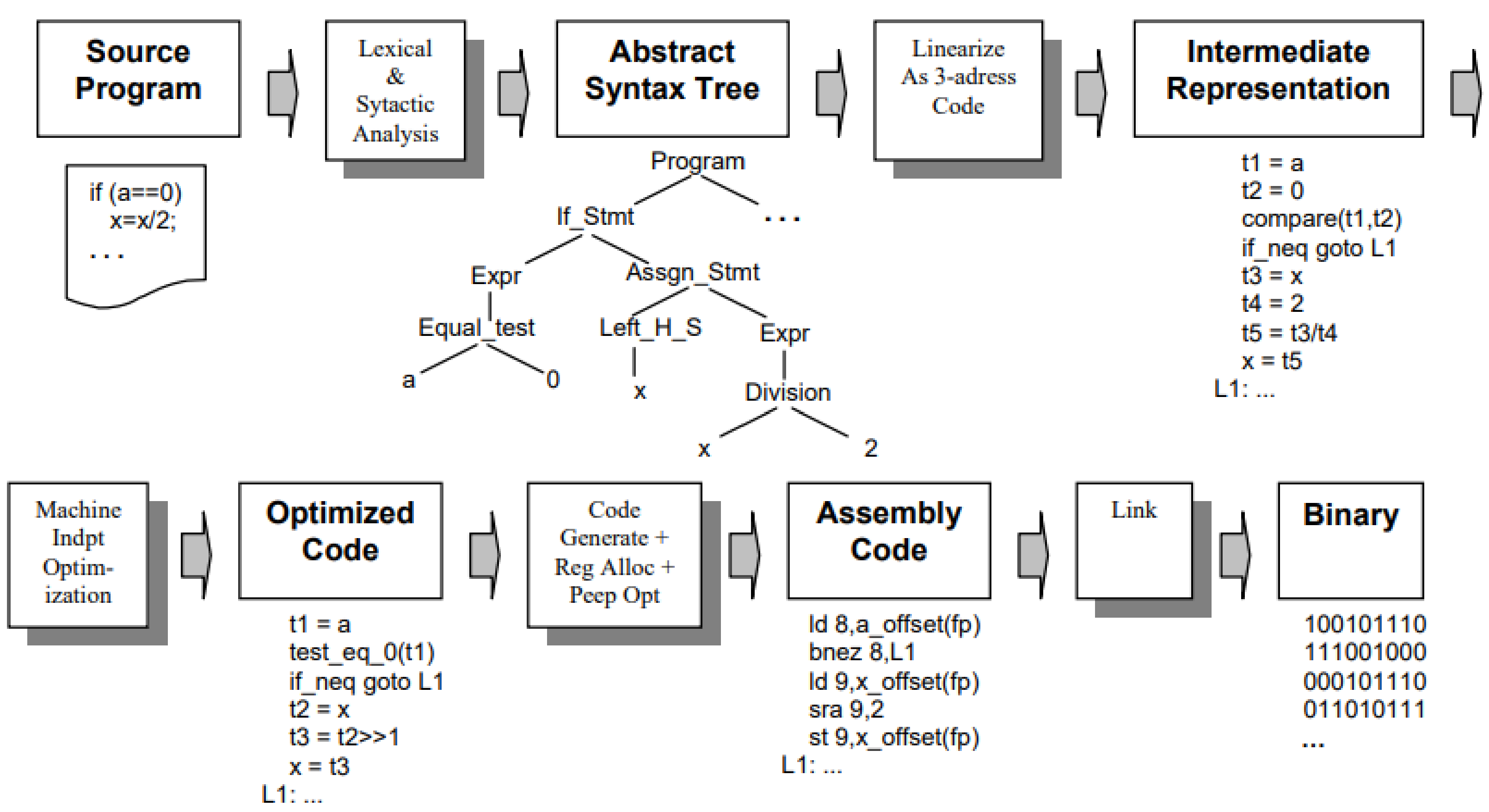
Présentation des outils de construction de code



Lenovo 2023 Lenovo Internal. All rights reserved.

Concepts généraux

- Generic compilation process



Lenovo 2023 Lenovo Internal. All rights reserved.

COMPILERS

2023 Lenovo Internal. All rights reserved.

Compiler: INTEL CLASSIC COMPILER

Compiler	Language
icc	C
ifort	Fortran (77 / 90 / 95 / 2003)
icpc	C++

Lenovo 2023 Lenovo Internal. All rights reserved.

Compiler: INTEL ONEAPI COMPILER

Compiler	Language
icx	C
ifx	Fortran (77 / 90 / 95 / 2003)
icpx	C++

Lenovo 2023 Lenovo Internal. All rights reserved.

Intel Compiler Common Flags

Disable optimization	-O0
Optimize for speed (no code size increase)	-O1
Optimize for speed (default)	-O2
High-level loop optimization	-O3
Create symbols for debugging	-g
Multi-file inter-procedural optimization	-ipo
Profile guided optimization (multi-step build)	-prof-gen -prof-use
Optimize for speed across the entire program ("prototype switch") *fast* options definitions changes over time!	-fast same as: -ipo –O3 -no-prec-div –static –fp-model fast=2 -xHost
OpenMP support	-qopenmp
Automatic parallelization	-parallel

Lenovo 2023 Lenovo Internal. All rights reserved.

Compiler: GNU COMPILER COLLECTION (GCC)

Compiler	Language
gcc	C (can also build FORTRAN programs)
gfortran	Fortran (77 / 90 / 95 / 2003)
g++	C++

Lenovo 2023 Lenovo Internal. All rights reserved.

CLANG: C Compiler from LLVM Framework

- C/C++ from LLVM project.
- FORTRAN version in development.
- Most flags from GCC can be used with CLANG.

Lenovo 2023 Lenovo Internal. All rights reserved.

GNU C and Intel Compiler Flag Comparison

GNU Compiler Flags	Intel Classic Compiler Flags	Intel OneAPI Compiler Flags	Description
-O2	-O2	-O2	Optimize for speed (default)
-O3	-O3	-O3	High-level optimizer
-fast	-fp-model fast	-fp-model fast	“fast” Boost computational if not sensible of inaccuracy
-g	-g	-g	Create symbols for debugging
-S	-S	-S	Generate assembly files
-fopenmp	-qopenmp	-qopenmp	OpenMP support
-march=icelake-server	-xCORE-AVX512	-xCORE-AVX512	Generate AVX512 code
-floop-parallelize-all	-parallel	-parallel	Automatic parallelization for OpenMP threading
-pg -fprofile-generate	-prof_gen	-prof_gen	Generate PGO files
-fprofile-use	-prof_use	-prof_use	Use PGO files

Lenovo 2023 Lenovo Internal. All rights reserved.

MATHEMATICAL LIBRARIES

2023 Lenovo Internal. All rights reserved.

Netlib Math Library

- Open Source
- BLAS (vector and matrix operations)
  - Level 1: vector-vector
  - Level2: matrix-vector
  - Level3: matrix-matrix
  - Single, double, complex and double complex precision
  - Written in FORTRAN
  - Ex:
    - DAXPY: compute double precision y=a*x+y (a is scalar, y and x vectors)
    - DGEMM: double precision matrix-matrix multiply
- CBLAS: C version of BLAS
- LAPACK: Solve linear equation systems.
  - Written in FORTRAN
  - Ex:
    - DTRSM: Solve A*X = alpha*B, alpha is scalar, X and B are matrices, A is triangular matrix. X is unknown.
- ScaLAPACK: distributed version of Lapack using BLACS (Communication BLAS) and PBLAS (Parallel BLAS).

Lenovo 2023 Lenovo Internal. All rights reserved.

Math Architecture Tuned Libraries

- Intel MKL
  - Intel tuned Math library
  - Use “-mkl” keyword with Intel compiler
  - BLAS, LAPACK and ScaLAPACK support. For FFT, need to build FFT wrapper library.
- AOCL (AMD Optimized CPU Libraries)
  - AMD tuned Math Library
  - BLIS, Sparse, ScalaPack, FFTW, etc.
- nVidia GPU Libraries:
  - cuBLAS, cuSPARSE, cuFFT, etc.

Lenovo 2023 Lenovo Internal. All rights reserved.

Intel MKL

- Mathematical library for Intel processor
- Features:
  - BLAS
  - LAPACK
  - BLACS
  - ScaLAPACK
  - FFT
    - Support complex-to-complex, real-to-complex, complex-to-real transform for one and two dimensional
    - For three to seven dimensional, support complex-to-complex
  - FFTW interface for FFT
- Some routines support OpenMP
  - LAPACK (GETRF, POTRF and GBTRF routines)
  - BLAS
  - DFT
  - FFT
  - Control with environment variable
    - MKL_NUM_THREADS

Lenovo 2023 Lenovo Internal. All rights reserved.

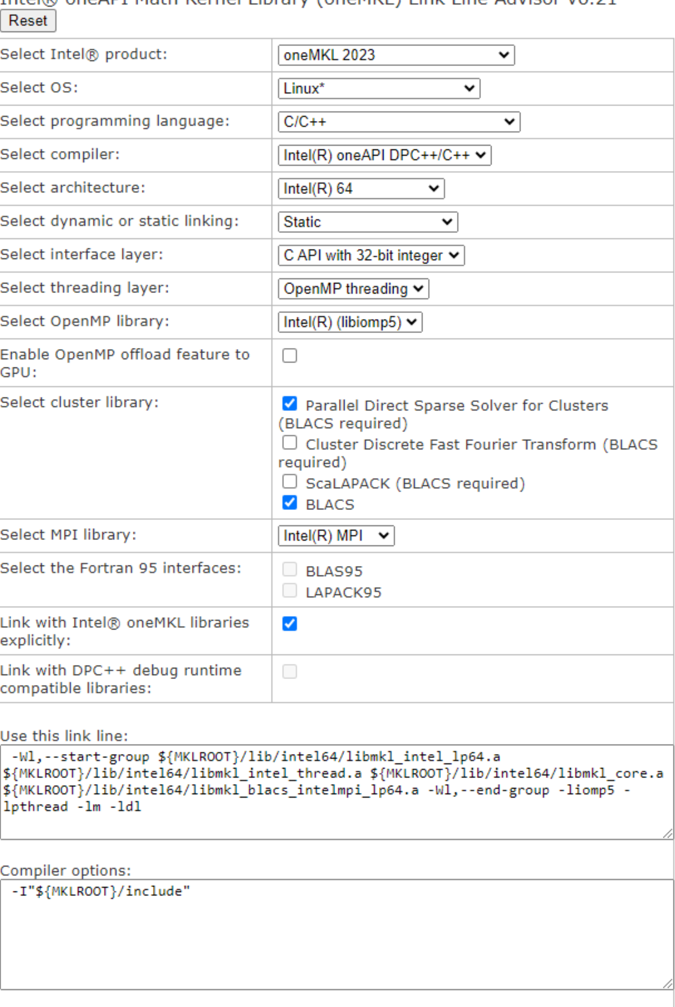
Static VS Dynamic, Serial VS Multi-Threads

- Static VS Dynamic
  - Intel MKL library can be linked statically
    - -static-intel
  - Advantages
    - Normally faster
    - Could benefit from IPO
  - Disadvantages
    - Bigger binary
    - Requires relinking to benefit from newer library version after upgrade
- Serial VS Multi-Threads
  - Code can be linked to multi-threaded version of the library
  - Advantages
    - Effort-free introduction of multi-threading into user application
  - Disadvantages
    - Can create conflicts with natively multi-threads code
      - OMP_NUM_THREADS VS MKL_NUM_THREADS
  - Note
    - Multi-threaded version is normally as fast as the serial one in single-thread mode

Lenovo 2023 Lenovo Internal. All rights reserved.

Intel MKL Command Line Advisor

https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl-link-line-advisor.html



Lenovo 2023 Lenovo Internal. All rights reserved.

MPI LIBRARIES

2023 Lenovo Internal. All rights reserved.

MPI Libraries

- Intel MPI
- OpenMPI
- MVAPICH2
- MPICH2

- Wrappers are available to avoid specifying header and library paths during compilation and link:
  - C: mpicc (using GCC), mpiicc (using Intel C Compiler)
  - C++ : mpicxx, mpiicpc
  - FORTRAN: mpif90, mpifort, mpiifort

- For Intel MPI, C compiler binary can be override using "-cc=" flags or setting I_MPI_CC/ environment variables (same for C++ using "–cxx=" or I_MPI_CXX and for FORTRAN using I_MPI_F90)
- For OpenMPI, OMPI_MPICC, OMPI_MPICXX, OMPI_MPIF90

Lenovo 2023 Lenovo Internal. All rights reserved.

Intel MPI
Compile Source Files

Wrapper	Compiler	Language
mpiicc	icc	C
mpiifort	ifort	Fortran
mpiicpc	icpc	C++

- Intel MPI environment Loading
  - Using Intel-provided scripts

    -source <Intel MPI>/bin64/mpivars.sh
  - Using Modules

    -module load intel_mpi/<Version>
- How-To: Check Wrapper Command Details
  - <Wrapper> -show
- Wrappers arguments come at the end of the compiler command
  - mpiicc <Compiler Flags> <Source File>

    =
  - icc <Compiler Flags> <Source File> <Wrapper Arguments>

Lenovo 2023 Lenovo Internal. All rights reserved.

Open MPI
Compile Source Files

Wrapper	Compiler	Language
mpicc	gcc	C
mpif77	gfortran	Fortran 77
mpif90	gfortran	Fortran 90
mpic++ mpiCC mpicxx	g++	C++

- Open MPI environment Loading
  - Using Modules

    -module load open_mpi/<Version>
  - Manually

    -export PATH

    -export LD_LIBRARY_PATH

Lenovo 2023 Lenovo Internal. All rights reserved.

Open MPI
Usual Mistakes

- bash: orted: command not found
  - Requires ORTED command to be in default user path
  - export PATH=/usr/mpi/intel/openmpi-1.4.2-qlc/bin
    - In File: .bashrc
- /usr/mpi/intel/openmpi-1.4.2-qlc/bin/orted: error while loading shared libraries: libimf.so: cannot open shared object file: No such file or directory
  - Requires Intel Compilers libraries to be located
  - export LD_LIBRARY_PATH=/logiciels/intel/Compiler/11.1/072/lib/intel64
    - In File: .bashrc
- <User Binary>: error while loading shared libraries: libmpi.so.0: cannot open shared object file: No such file or directory
  - Requires current user environment to be exported to MPI tasks
  - -x LD_LIBRARY_PATH
    - In MPIRUN command

Lenovo 2023 Lenovo Internal. All rights reserved.

MPICH2 / MVAPICH2

Similar syntax for execution than Intel MPI.

Wrapper	Compiler	Language
mpicc	gcc or icc (depends on compilation)	C
mpif77 / mpif90 / mpifort	gfortran or ifort (depends on compilation)	Fortran 77 / Fortran 90
mpicxx	g++ or icpc (depends on compilation)	C++

Lenovo 2023 Lenovo Internal. All rights reserved.

Makefile Overview

Makefile structure	Example
<rule>:<rule dependencies> [tab] <command 1> [tab] <command 2> […]	matrixmul.o:matrixmul.c matrix.h icc –c matrixmul.c
Recursive Makefile	**Example**
<rule>: cd <subdir> && $(MAKE)	matrixmul.o: cd multiply && $(MAKE)

- Execute the first rule of file ‘makefile’ or ‘Makefile’ presents in the current directory:
  - make
- Specify wich rule to use:
  - make clean

  -Execute the rule ‘clean’ of the makefile
- Concurrent execution
  - make -j 3

  -Launch 3 complementary executions of the Makefile commands.

Lenovo 2023 Lenovo Internal. All rights reserved.

Implicit Rules of Makefile

Implicit rules use implicit environment variables	Meaning
CC	C compiler
FC	FORTRAN compiler
CFLAGS	Flags for C compiler
FFLAGS	Flags for FORTRAN compiler
FDFLAGS	Flags for the linker
More: http://www.gnu.org/software/make/manual/make.html#index-flags-for-compilers-861	

Implicit rules use Implicit variables	Meaning
$@	the rule name
$^	the rule dependencies
$?	rule dependencies newer than the target
More: http://www.gnu.org/software/make/manual/make.html#index-automatic-variables-933	

Lenovo 2023 Lenovo Internal. All rights reserved.

Advanced Use of Makefile

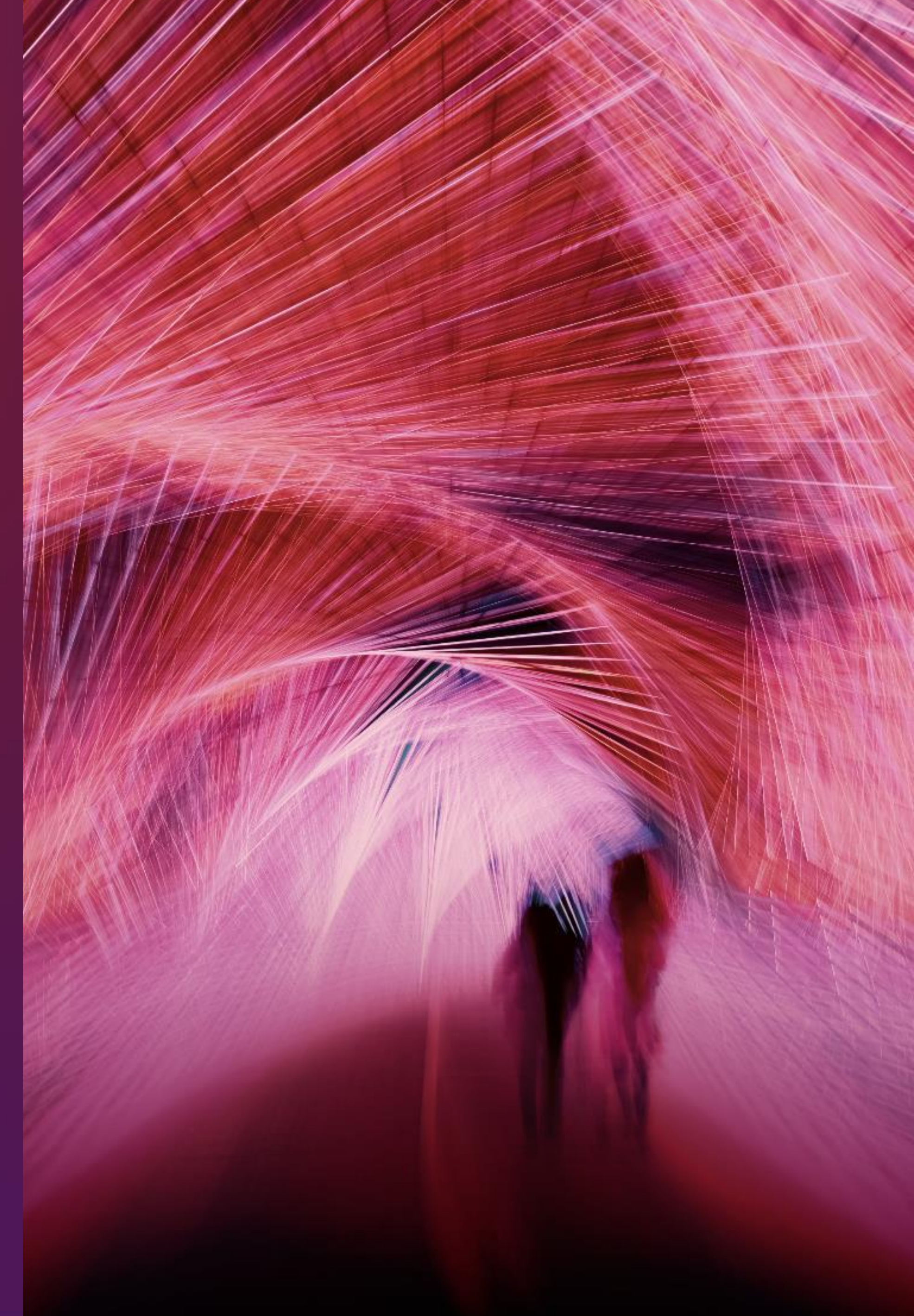
- Wildcard characters:
  - Use the Bourne shell: ‘*’, ‘?’ and ‘[…]’

Wildcard character	Normal form
clean: rm -f *.o	clean: rm –f matrix.o vector.o mulmv.o
sources := $(wildcard *.c)	sources := matrix.c vector.c mulmv.c
objects := $(patsubst %.c,%.o,$(wildcard *.c)	objects := matrix.o vector.o mulmv.o
Conditionnal value	**Example**

ifeq (<variable>, <value>)
<command 1>
<command 2>
[…]
else
<command 1’>
<command 2’>
[…]
endif
	ifeq ($(PROFILING), yes)
	FLAGS += -pg
	endif
	
	ifeq ($(INSTRUMENT_MPI), yes)
	$(CC) –c instr_mpi.c –o instr_mpi.o
	$(CC) mpi_mul.c –o mpi_mul instr_mpi.o
	else
	$(CC) mpi_mul.c –o mpi_mul
	endif

Lenovo 2023 Lenovo Internal. All rights reserved.

Outils de diagnostic, profiling, analyse



Lenovo 2023 Lenovo Internal. All rights reserved.

LSCPU

- The command **lscpu** prints CPU architecture information from sysfs and /proc/cpuinfo as shown below:

- Important information for HPC folks:
  - Name of processor
  - Number of cores
  - Clock frequencies
  - Hyperthreading mode
  - Number of Numa nodes
  - Vector units supported by the processor

```
[emichel@ice2740:~/ASSETS/Easy_STREAM]$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                144
On-line CPU(s) list:   0-143
Thread(s) per core:    2
Core(s) per socket:    36
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 106
Model name:            Intel(R) Xeon(R) Platinum 8360Y CPU @ 2.40GHz
Stepping:              6
CPU MHz:               900.000
CPU max MHz:           2401.0000
CPU min MHz:           800.0000
BogoMIPS:              4800.00
Virtualization:        VT-x
L1d cache:             48K
L1i cache:             32K
L2 cache:              1280K
L3 cache:              55296K
NUMA node0 CPU(s):     0-35,72-107
NUMA node1 CPU(s):     36-71,108-143
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm cons
tant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf
 pni pclmulqdq dtes64 ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid dca sse
4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm
 3dnowprefetch cpuid_fault epb cat_l3 invpcid_single intel_ppin ssbd mba ibrs ibpb stib
p ibrs_enhanced tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 a
vx2 smep bmi2 erms invpcid cqm rdt_a avx512f avx512dq rdseed adx avx512ifma clflushopt
clwb intel_pt avx512cd sha_ni avx512bw avx512vl xsaveopt xsavec xgetbv1 xsaves cqm_llc
cqm_occup_llc cqm_mbm_total cqm_mbm_local split_lock_detect wbnoinvd dtherm ida arat pl
n pts avx512vbmi umip pku ospke avx512_vbmi2 gfni vaes vpclmulqdq avx512_vnni avx512_bi
talg tme avx512_vpopcntdq la57 rdpid fsrm md_clear pconfig flush_l1d arch_capabilities
```


2023 Lenovo Internal. All rights reserved.

NUMACTL

- The command **numactl** allows lot of actions regarding NUMA information: scheduling, memory, binding…

- Reports
  - Number of numa nodes
  - Numbering of cores in numa nodes
  - Size of numa nodes

- Binds
  - Processes to cores & numa nodes
  - Memory to numa nodes
  - Interleaving…

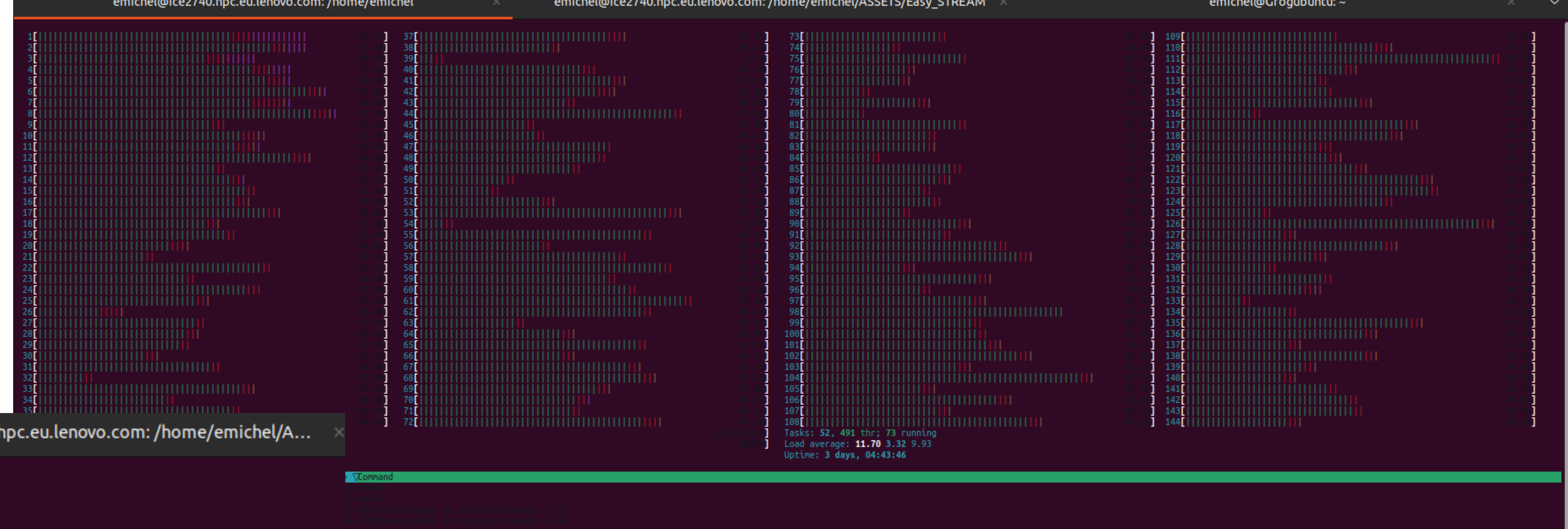
```
[emichel@ice2740:~]$ numactl -h
numactl: invalid option -- 'h'
usage: numactl [--all | -a] [--interleave= | -i <nodes>] [--preferred= | -p <node>]
               [--physcpubind= | -C <cpus>] [--cpunodebind= | -N <nodes>]
               [--membind= | -m <nodes>] [--localalloc | -l] command args ...
       numactl [--show | -s]
       numactl [--hardware | -H]
       numactl [--length | -l <length>] [--offset | -o <offset>] [--shmmode | -M <shmmo
de>]
               [--strict | -t]
               [--shmid | -I <id>] --shm | -S <shmkeyfile>
               [--shmid | -I <id>] --file | -f <tmpfsfile>
               [--huge | -u] [--touch | -T]
               memory policy | --dump | -d | --dump-nodes | -D

memory policy is --interleave | -i, --preferred | -p, --membind | -m, --localalloc | -l
<nodes> is a comma delimited list of node numbers or A-B ranges or all.
Instead of a number a node can also be:
  netdev:DEV the node connected to network device DEV
  file:PATH  the node the block device of path is connected to
  ip:HOST    the node of the network device host routes through
  block:PATH the node of block device path
  pci:[seg:]bus:dev[:func] The node of a PCI device
<cpus> is a comma delimited list of cpu numbers or A-B ranges or all
all ranges can be inverted with !
all numbers and ranges can be made cpuset-relative with +
the old --cpubind argument is deprecated.
use --cpunodebind or --physcpubind instead
<length> can have g (GB), m (MB) or k (KB) suffixes
```

```
[emichel@ice2740:~]$ numactl -H
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92
93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
node 0 size: 128597 MB
node 0 free: 120227 MB
node 1 cpus: 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
 61 62 63 64 65 66 67 68 69 70 71 108 109 110 111 112 113 114 115 116 117 118 119 120 1
21 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142
143
node 1 size: 128954 MB
node 1 free: 125129 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
```

Lenovo 2023 Lenovo Internal. All rights reserved.

TOP / HTOP



emichel@ice2740.hpc.eu.lenovo.com: /home/emichel   emichel@ice2740.hpc.eu.lenovo.com: /home/emichel/A...

```
top - 16:16:34 up 3 days,  4:47,  2 users,  load average: 11.76, 7.05, 9.95
Tasks: 1382 total,  23 running, 1358 sleeping,   0 stopped,   1 zombie
%Cpu(s): 47.7 us,  1.4 sy,  0.0 ni, 50.3 id,  0.0 wa,  0.2 hi,  0.3 si,  0.0 st
MiB Mem : 257552.6 total, 232631.7 free,  20270.9 used,   4650.0 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used. 231614.4 avail Mem
  scroll coordinates: y = 1/1382 (tasks), x = 1/12 (fields)
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 848392 emichel   20   0   17.2g  12.4g   5144 R  3743   4.9   1:55.66 bin/stream_f.icelake.556000000.10
   9016 root       0 -20   26.9g   4.6g 198804 S   3.2   1.8  79:02.43 /usr/lpp/mmfs/bin/mmfsd
 847595 emichel   20   0  276944   6472   3976 R   3.2   0.0   0:01.49 top
  24174 zabbix    20   0  132584   5196   3544 S   2.9   0.0   3:11.71 /usr/sbin/zabbix_agentd: active checks #1 [idle 1 sec]
   3764 root      20   0  787732 110192  80600 S   1.0   0.0  13:29.06 /usr/libexec/platform-python -Es /usr/sbin/tuned -l -P
     14 root      20   0       0      0      0 I   0.3   0.0   3:18.52 [rcu_sched]
     20 root      rt   0       0      0      0 S   0.3   0.0   0:00.24 [migration/1]
     38 root      rt   0       0      0      0 S   0.3   0.0   0:00.20 [migration/4]
     68 root      rt   0       0      0      0 S   0.3   0.0   0:00.17 [migration/9]
    224 root      rt   0       0      0      0 S   0.3   0.0   0:00.15 [migration/35]
    441 root      rt   0       0      0      0 S   0.3   0.0   0:00.10 [migration/71]
    681 root      rt   0       0      0      0 S   0.3   0.0   0:00.21 [migration/111]
    753 root      rt   0       0      0      0 S   0.3   0.0   0:00.11 [migration/123]
    807 root      rt   0       0      0      0 S   0.3   0.0   0:00.10 [migration/132]
    825 root      rt   0       0      0      0 S   0.3   0.0   0:00.10 [migration/135]
    849 root      rt   0       0      0      0 S   0.3   0.0   0:00.10 [migration/139]
    855 root      rt   0       0      0      0 S   0.3   0.0   0:00.10 [migration/140]
    861 root      rt   0       0      0      0 S   0.3   0.0   0:00.10 [migration/141]
   6788 root      20   0  175720  10764   9376 S   0.3   0.0   0:01.03 /usr/libexec/sssd/sssd_ssh --uid 0 --gid 0 --logger=files
  24172 zabbix    20   0  132108  13136  12004 S   0.3   0.0   4:05.38 /usr/sbin/zabbix_agentd: collector [idle 1 sec]
 314504 root      20   0       0      0      0 I   0.3   0.0   0:00.40 [kworker/116:3-events]
 701240 root      20   0       0      0      0 R   0.3   0.0   0:00.11 [kworker/125:3-events]
 749181 root      20   0       0      0      0 I   0.3   0.0   0:00.06 [kworker/115:2-events]
 752026 root      20   0       0      0      0 I   0.3   0.0   0:00.05 [kworker/57:2-events]
 781283 root      20   0       0      0      0 I   0.3   0.0   0:00.05 [kworker/42:2-events]
 790458 root      20   0       0      0      0 I   0.3   0.0   0:00.03 [kworker/126:0-events]
 790474 root      20   0       0      0      0 R   0.3   0.0   0:00.04 [kworker/55:2-events]
 818152 root      20   0       0      0      0 I   0.3   0.0   0:00.06 [kworker/105:2-events]
 822930 root      20   0       0      0      0 I   0.3   0.0   0:00.02 [kworker/73:1-events]
 833562 root      20   0       0      0      0 I   0.3   0.0   0:00.03 [kworker/39:1-events]
 837032 root      20   0       0      0      0 I   0.3   0.0   0:00.02 [kworker/140:0-events]
 843060 root      20   0       0      0      0 I   0.3   0.0   0:00.93 [kworker/0:1-events]
 843191 root      20   0       0      0      0 R   0.3   0.0   0:00.04 [kworker/4:1-events]
 843263 root      20   0       0      0      0 I   0.3   0.0   0:00.02 [kworker/95:1-events]
 843512 root      20   0       0      0      0 R   0.3   0.0   0:00.02 [kworker/143:3-events]
 846552 root      20   0       0      0      0 I   0.3   0.0   0:00.01 [kworker/13:0-events]
 847446 root      20   0       0      0      0 R   0.3   0.0   0:00.01 [kworker/8:0-events]
      1 root      20   0  240632  12412   8376 S   0.0   0.0   5:54.55 /usr/lib/systemd/systemd --switched-root --system --deserialize 16
      2 root      20   0       0      0      0 S   0.0   0.0   0:01.97 [kthreadd]
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 [rcu_gp]
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 [rcu_par_gp]
```

- System monitoring
  - Core usage
  - Memory usage
  - Process information
  - Running status
  - Owner
- Monitor the whole node
  - Limited by operating system
  - Can be difficult with large core counts

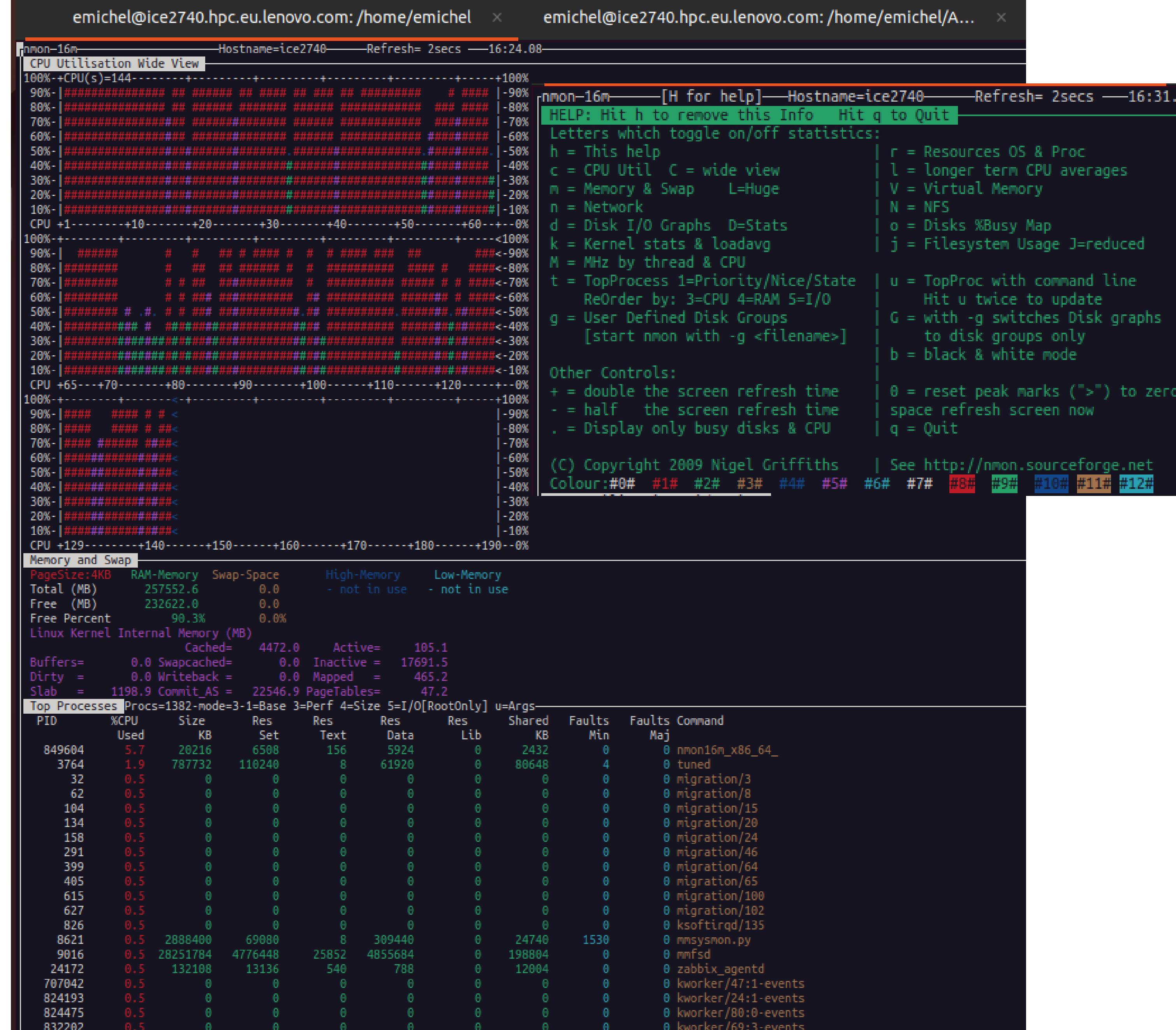
https://htop.dev/

Lenovo 2023 Lenovo Internal. All rights reserved.

NMON

- Core utilization and binding
- Process information
  - PID
  - CPU used
  - Virtual and physical memory
- Network
  - Data transfers on each interface
- Memory
  - Memory used, free
- I/O
  - Disk and partition usage

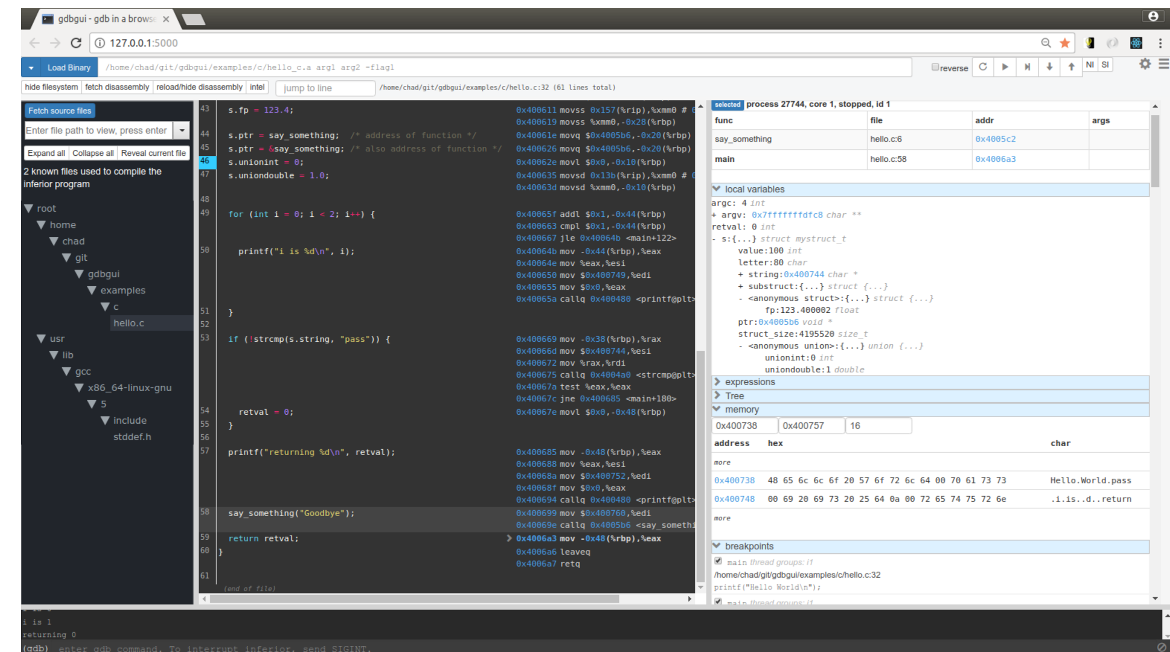
- And much more !
- https://nmon.sourceforge.net/pmwiki.php



Lenovo 2023 Lenovo Internal. All rights reserved.

The GNU Debugger (GDB)

- GNU Debugger
  - Interactive command line debugger
  - A graphic interface also exists: the Data Display Debugger
- Pre-requisite
  - Have to integrate symbols in your code
    - Compile with flag ‘-g’
- Invocation Command
  - Serial
    - gdb <Binary>
  - Distributed
    - <MPIRUN Command> xterm -e gdb <Binary>
      - Open as many Xterm windows as MPI jobs
      - Reserved for small number of MPI tasks



- Some GUI exist for GDB
  - https://sourceware.org/gdb/wiki/GDB%20Front%20Ends

 2023 Lenovo Internal. All rights reserved.

The GNU Debugger (GDB): GDB Internal Commands

Command	Argument	Explanation
b	<Function>	Set a breakpoint to the specified function
b	<File>:<Line>	Set a breakpoint on the specified source file / line
run	[<Binary Args>]	Launch program execution
p	<Variable Name>	Display the value or the specified variable
n	-	Execute the next instruction
c	-	Continue the program execution
quit		Quit the debugger and release the program

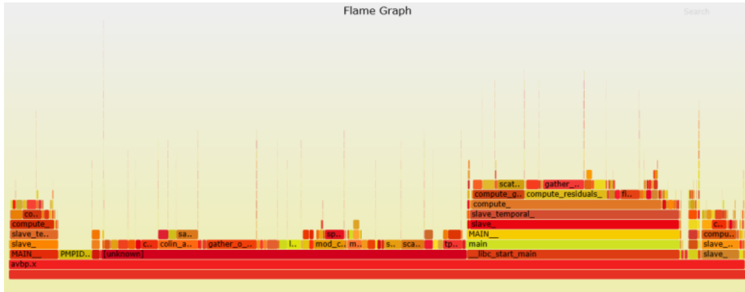
Lenovo 2023 Lenovo Internal. All rights reserved.

PERF

- **Perf** is a complex linux tool
- Included in the Linux kernel, under tools/perf, and is frequently updated and enhanced.

- It can instrument CPU performance counters, tracepoints, kprobes, and uprobes (dynamic tracing).

- It is capable of lightweight profiling.
  - perf record <command>
  - perf report
  - “post mortem” profiling (after execution ends)
  - https://www.brendangregg.com/perf.html




```
[emichel@ice2740:~]$ perf

 usage: perf [--version] [--help] [OPTIONS] COMMAND [ARGS]

 The most commonly used perf commands are:
   annotate        Read perf.data (created by perf record) and display annotated code
   archive         Create archive with object files with build-ids found in perf.data f
ile
   bench           General framework for benchmark suites
   buildid-cache   Manage build-id cache.
   buildid-list    List the buildids in a perf.data file
   c2c             Shared Data C2C/HITM Analyzer.
   config          Get and set variables in a configuration file.
   daemon          Run record sessions on background
   data            Data file related processing
   diff            Read perf.data files and display the differential profile
   evlist          List the event names in a perf.data file
   ftrace          simple wrapper for kernel's ftrace functionality
   inject          Filter to augment the events stream with additional information
   iostat          Show I/O performance metrics
   kallsyms        Searches running kernel for symbols
   kmem            Tool to trace/measure kernel memory properties
   kvm             Tool to trace/measure kvm guest os
   list            List all symbolic event types
   lock            Analyze lock events
   mem             Profile memory accesses
   record          Run a command and record its profile into perf.data
   report          Read perf.data (created by perf record) and display the profile
   sched           Tool to trace/measure scheduler properties (latencies)
   script          Read perf.data (created by perf record) and display trace output
   stat            Run a command and gather performance counter statistics
   test            Runs sanity tests.
   timechart       Tool to visualize total system behavior during a workload
   top             System profiling tool.
   version         display the version of perf binary
   probe           Define new dynamic tracepoints
   trace           strace inspired tool

 See 'perf help COMMAND' for more information on a specific command.
```

Lenovo 2023 Lenovo Internal. All rights reserved.

PERF

- **Perf** top provides real time execution profiling
  - Application
  - System functions too

- Access to source and assembly codes
  - With time spent on each “line” , instruction block in fact

- Required special Linux right (root or sudo)

```
                       xor        %edx,%edx
                       neg        %r11
                       xor        %eax,%eax
                       add        %r10,%r11
                       lea        (%rsi,%r9,8),%rsi
                       cmp        $0x1,%rcx
                     ↓ jae        1ccd
 20.58   17c8:         vmovups    -0x8(%rsi,%rcx,8),%zmm0
  0.20                 vmovntpd   %zmm0,-0x8(%r8,%rcx,8)
  0.00                 add        $0x8,%rcx
                       cmp        %r11,%rcx
  0.26               ↑ jb         17c8
         17e7:         lea        0x1(%r11),%rdx
                       cmp        %r10,%rdx
                     ↓ jbe        1ca2
         17f4:         mov        %r15,%rdi
                       mov        %r12d,%esi
                       vzeroupper
                     → callq      __kmpc_for_static_fini@plt
                       xor        %eax,%eax
  0.00                 mov        -0x40(%rbp),%r15
                       mov        -0x38(%rbp),%r14
```

```
Samples: 3M of event 'cycles', 4000 Hz, Event count (approx.): 1587027944218 lost: 0/0 drop: 0/0
Overhead  Shared Object                   Symbol
  86.82%  stream_f.icelake.556000000.10   [.] MAIN__
   3.59%  [kernel]                        [k] clear_page_erms
   2.41%  stream_f.icelake.556000000.10   [.] checksums_
   2.20%  libiomp5.so                     [.] _INTERNALc8ed1ec4::__kmp_wait_template<kmp_flag_64<false, true>, true, false,
   0.45%  [kernel]                        [k] update_sd_lb_stats.constprop.121
   0.29%  perf                            [.] 0x0000000000376c55
   0.28%  perf                            [.] 0x0000000000376bf0
   0.25%  libiomp5.so                     [.] _INTERNALc8ed1ec4::__kmp_hyper_barrier_gather
   0.19%  libc-2.28.so                    [.] _int_free
   0.14%  libc-2.28.so                    [.] _int_malloc
   0.13%  perf                            [.] 0x00000000002970c2
   0.12%  libiomp5.so                     [.] kmp_flag_native<unsigned long long, (flag_type)1, true>::notdone_check
   0.11%  perf                            [.] 0x0000000000376c48
   0.10%  perf                            [.] evsel__parse_sample
   0.09%  [kernel]                        [k] load_balance
   0.08%  perf                            [.] 0x0000000000376be0
   0.07%  perf                            [.] perf_mmap__read_event
   0.06%  perf                            [.] dso__find_symbol
   0.05%  perf                            [.] machine__findnew_thread
   0.05%  [kernel]                        [k] get_page_from_freelist
   0.05%  [kernel]                        [k] __x86_return_thunk
   0.05%  libc-2.28.so                    [.] cfree@GLIBC_2.2.5
   0.05%  [kernel]                        [k] update_rq_clock
   0.05%  [kernel]                        [k] update_blocked_averages
   0.05%  libc-2.28.so                    [.] __strcmp_avx2
   0.05%  [kernel]                        [k] __sched_text_start
   0.04%  [kernel]                        [k] _raw_spin_lock
   0.04%  perf                            [.] perf_hpp__is_dynamic_entry
   0.04%  [kernel]                        [k] dbs_update_util_handler
   0.03%  [kernel]                        [k] native_irq_return_iret
   0.03%  mmfsd                           [.] SharedHashTabIterator::nextObjP
   0.03%  [kernel]                        [k] __lock_text_start
   0.03%  [kernel]                        [k] native_queued_spin_lock_slowpath
   0.03%  libc-2.28.so                    [.] malloc
   0.02%  [kernel]                        [k] native_write_msr
   0.02%  [kernel]                        [k] clear_subpage
   0.02%  perf                            [.] 0x0000000000399840
   0.02%  [kernel]                        [k] trigger_load_balance
   0.02%  perf                            [.] comm__str
   0.02%  perf                            [.] 0x0000000000376bec
   0.02%  [kernel]                        [k] __free_pages_ok
   0.02%  perf                            [.] evsel__parse_sample_timestamp
   0.02%  [kernel]                        [k] __update_load_avg_cfs_rq
   0.02%  perf                            [.] 0x0000000000376d1a
   0.02%  [kernel]                        [k] ktime_get
   0.02%  [kernel]                        [k] __switch_to
   0.02%  [kernel]                        [k] update_irq_load_avg
   0.02%  [kernel]                        [k] update_curr
   0.02%  [kernel]                        [k] find_next_and_bit
   0.02%  perf                            [.] 0x00000000002970e8
For a higher level overview, try: perf top --sort comm,dso
```

2023 Lenovo Internal. All rights reserved.

GNU Profiler (GPROF)

- Compile the program with options: -g -pg
  - Will create symbols required for debugging / profiling
- Execute the program
  - Standard way
- Execution generates profiling files in execution directory
  - gmon.out.<MPI Rank>
    - Binary files, not human readable
  - Nb files depends on environment variable
    - 1 Profiling File / Process
    - 3 Profiling Files only
      - One file for the slowest / fastest / median process
- Allows profiling report generation
  - From profiling output files
  - Standard Usage
    - gprof <Binary> gmon.out.<MPI Rank> > gprof.out.<MPI Rank>

Nice example:
https://moodle.nhr.fau.de/pluginfile.php/2393/mod_resource/content/3/gprof_2023.pdf

Lenovo 2023 Lenovo Internal. All rights reserved.

GNU Profiler (GPROF)

- Gprof example file
  - 1: flat profile
    - -Summarize time spent in function and its children
  - 2: call graph
    - -Order functions according to time

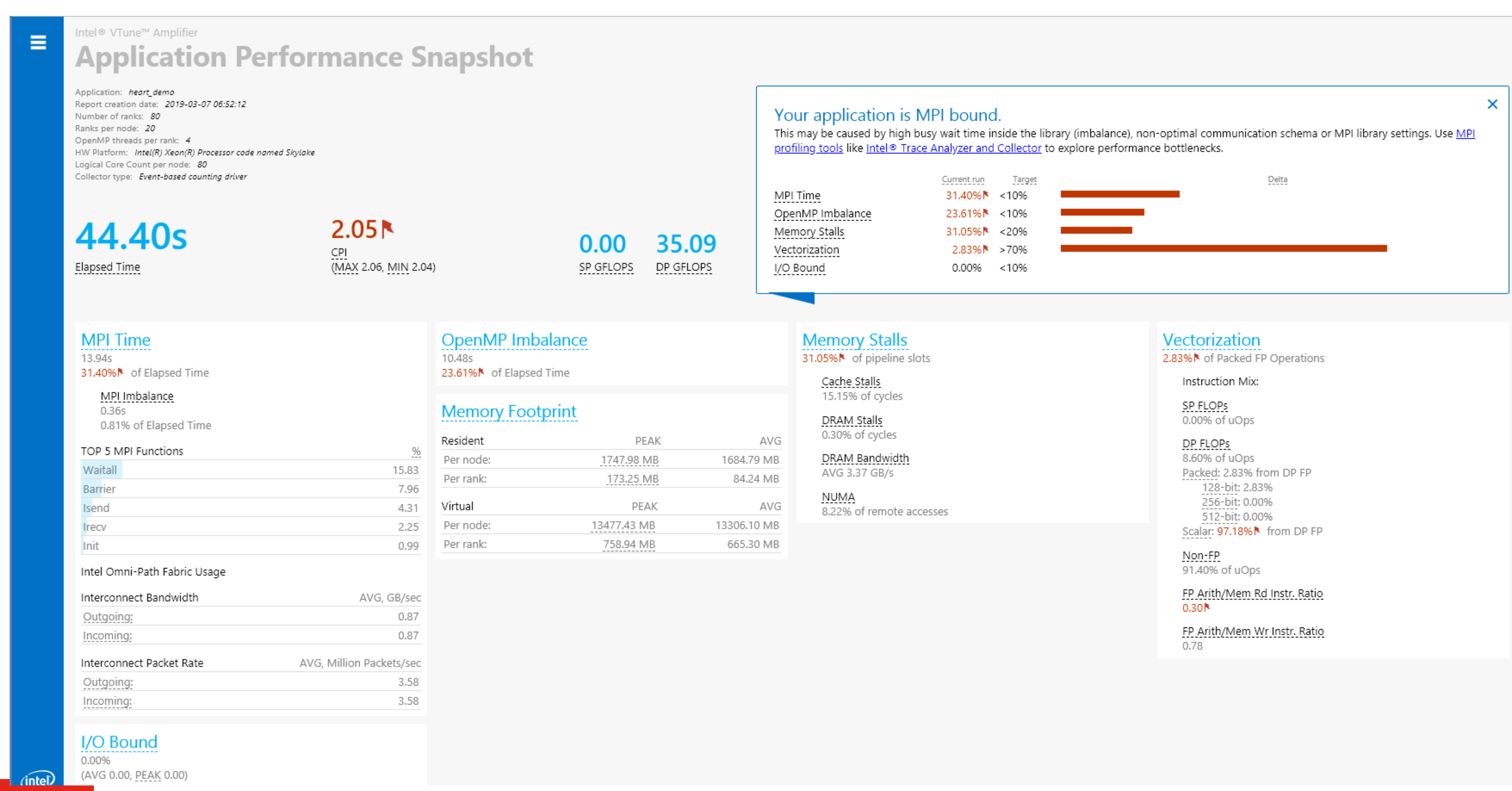
1

2

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  s/call   s/call  name
 75.81     92.88    92.88     7000    0.01     0.01  calcul_
 24.16    122.48    29.60                             __intel_new_memcpy
  0.02    122.50     0.02                             __intel_new_memset
[…]
index % time    self  children    called     name
                0.00   92.89       1/1           main [2]
[1]     75.8    0.00   92.89       1         MAIN__ [1]
               92.88    0.00    7000/7000        calcul_ [3]
                0.01    0.00       1/1           initialisation_ [6]
                0.00    0.00    7000/7000        communication_ [8]
                0.00    0.00       2/2           mpi_times_mp_mpi_time_ [9]
                0.00    0.00       1/1           voisinage_ [10]
-----------------------------------------------
                                                 <spontaneous>
[2]     75.8    0.00   92.89                 main [2]
                0.00   92.89       1/1           MAIN__ [1]
-----------------------------------------------
               92.88    0.00    7000/7000        MAIN__ [1]
[3]     75.8   92.88    0.00    7000         calcul_ [3]
-----------------------------------------------
[…]
```

Lenovo 2023 Lenovo Internal. All rights reserved.

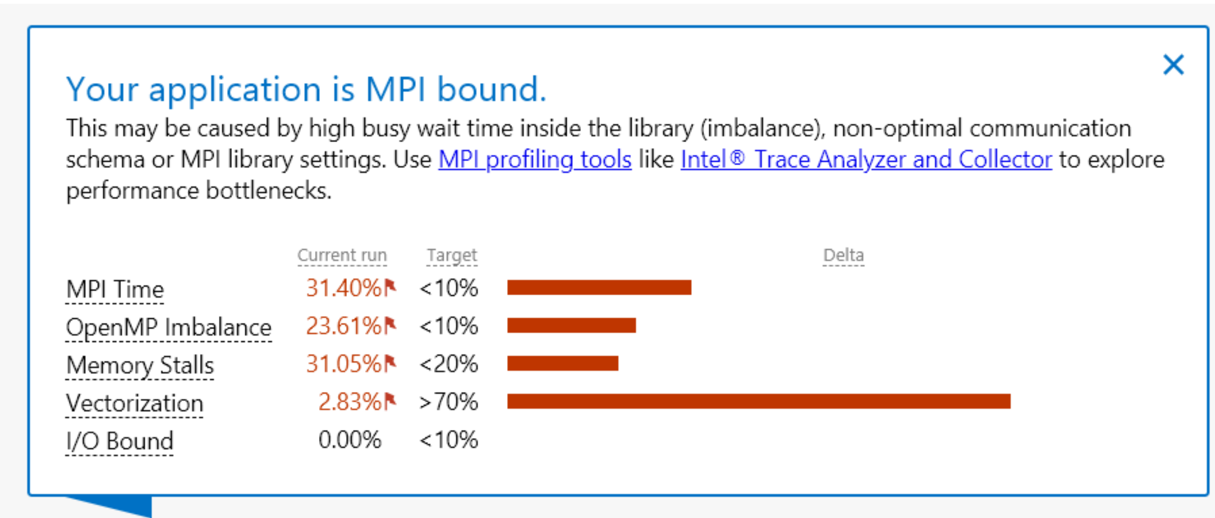
Intel APS - Introduction



Lenovo 2023 Lenovo Internal. All rights reserved.

Intel APS - Overview

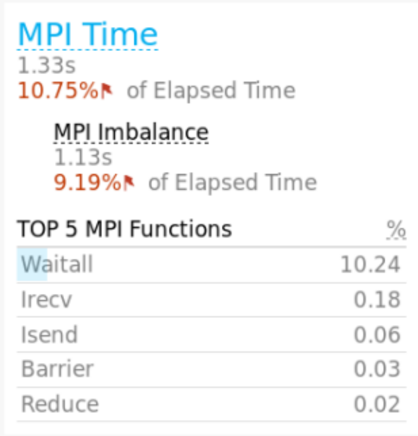
- Overview shows all areas and relative impact on code performance
- Provides recommendation for next step in performance analysis
- “X” collapses the summary, removing the flags (objective numbers only)



2023 Lenovo Internal. All rights reserved.

Intel APS – parallel runtimes

- MPI Time
  - How much time was spent in MPI calls
  - Averaged by ranks with % of Elapsed time
  - Available for MPICH-based MPI and OpenMPI
- MPI Imbalance
  - Unproductive time spent in MPI library waiting for data
    - Switched off by default
    - Available for Intel MPI with APS_IMBALANCE_TYPE=1
    - Over supported MPISs with APS_IMBALANCE_TYPE=2
- OpenMP Imbalance
  - Time spent at OpenMP Synchronization Barriers normalized by number of threads
  - Available for Intel OpenMP
- Serial time
  - Time spend outside OpenMP regions
  - Available for Intel OpenMP, shared memory applications only



OpenMP Imbalance
3.44s
42.25% of Elapsed Time

Serial Time
4.45s
31.11% of Elapsed Time

Lenovo 2023 Lenovo Internal. All rights reserved.

Intel APS – memory access and vectorization

- Memory stalls measurement with
- breakdown by cache and DRAM
  - Average DRAM Bandwidth
  - NUMA ratio

- Vectorization efficiency based on HW-event statistics with
  - Breakdown by vector/scalar instructions
  - Floating point vs memory instruction ratio

Vectorization
41.40% of Packed FP Operations

Instruction Mix:

SP FLOPs
0.00% of uOps

DP FLOPs
17.40% of uOps
Packed: 41.40% from DP FP
128-bit: 41.40%
256-bit: 0.00%
Scalar: 58.60% from DP FP

Non-FP
82.60% of uOps

FP Arith/Mem Rd Instr. Ratio
0.50

FP Arith/Mem Wr Instr. Ratio
4.14

Lenovo 2023 Lenovo Internal. All rights reserved.

Intel APS – MPI Statistics

- MPI Time per rank

  >aps-report –mpi-time-per-rank <result>

MPI Time per Rank

Rank	LifeTime(sec)	MPI Time(sec)	MPI Time(%)	Imbalance(sec)	Imbalance(%)
0007	72.52	14.31	19.74	4.84	6.67
0004	72.53	11.57	15.96	3.26	4.50
0005	72.52	11.40	15.72	3.20	4.42
0006	72.51	11.11	15.32	3.17	4.37
0000	72.49	11.08	15.29	4.33	5.97
0001	72.52	10.95	15.10	3.01	4.15
0002	72.49	10.79	14.88	2.57	3.55
0003	72.50	10.64	14.68	2.50	3.45
TOTAL	580.07	91.86	15.84	26.88	4.63
AVG	72.51	11.48	15.84	3.36	4.63

- Message Size Summary by all ranks
- *>aps-report –message-sizes <result>*
- *Requires setting MPS_STAT_LEVEL=2 before collection launch*

Message Sizes summary for all ranks

Message size(B)	Volume(MB)	Volume(%)	Transfers	Time(sec)	Time(%)
8	1.49	0.09	195206	27.79	37.93
176	0.41	0.02	2420	27.67	37.78
4	0.00	0.00	1150	15.55	21.22
100264	115.89	6.94	1212	0.27	0.37
98400	113.74	6.81	1212	0.19	0.26
66256	38.29	2.29	606	0.17	0.23
[filtered out 57 lines]					
TOTAL	1670.60	100.00	265160	73.25	100.00

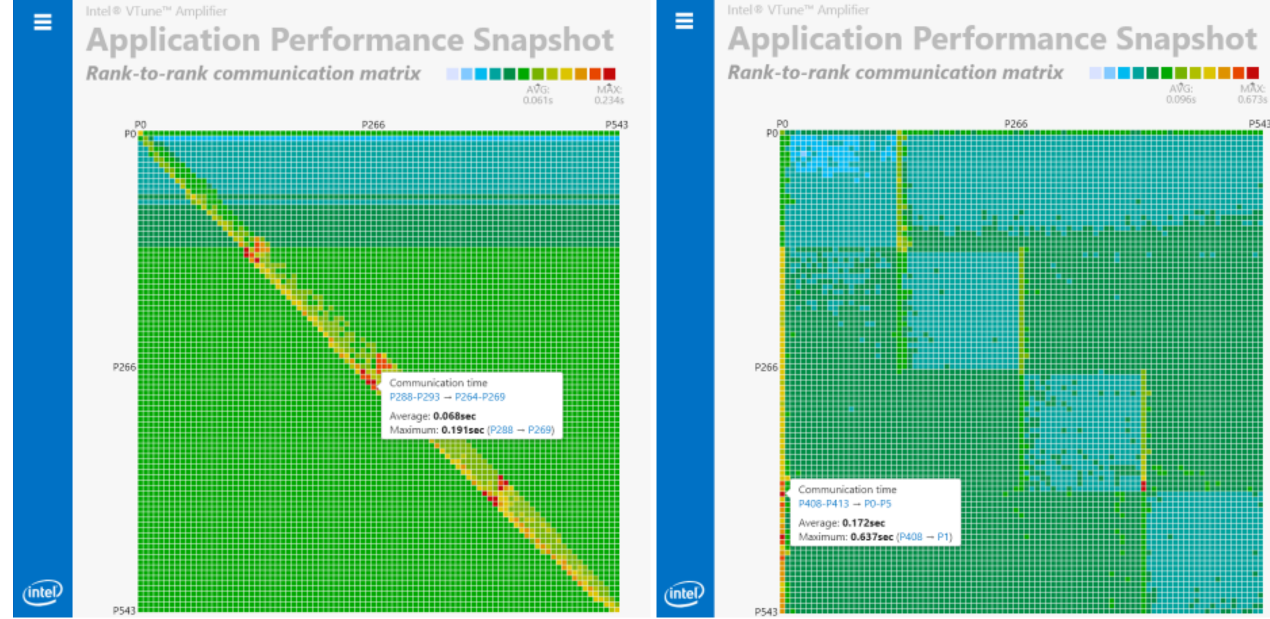
Lenovo 2023 Lenovo Internal. All rights reserved.

Intel APS – MPI Rank-to-Rank Communication

- Data Transfers for Rank-to-Rank Communication
  - *>aps-report –transfers-per-communication <result>*
  - *Requires setting MPS_STAT_LEVEL=4 before collection launch*

Rank --> Rank	Volume(MB)	Volume(%)	Transfers
0023 --> 0024	84.35	1.56	13477
0025 --> 0026	84.35	1.56	13477
0024 --> 0025	84.15	1.56	13477
0021 --> 0022	83.84	1.55	13477
0022 --> 0023	83.43	1.54	13477
[filtered out 16 lines]			
0012 --> 0011	69.60	1.29	13477
0020 --> 0019	69.29	1.28	13477
0026 --> 0025	68.78	1.27	13477
0025 --> 0024	68.38	1.27	13477
0022 --> 0021	68.38	1.27	13477
[filtered out 17 lines]			
0016 --> 0015	58.81	1.09	13477
0028 --> 0027	57.69	1.07	13477
0007 --> 0008	56.98	1.05	13477
0030 --> 0031	54.74	1.01	13477
0006 --> 0007	54.44	1.01	13477
[filtered out 1108 lines]			
TOTAL	5403.22	100.00	1415619
AVG	4.67	0.09	1224

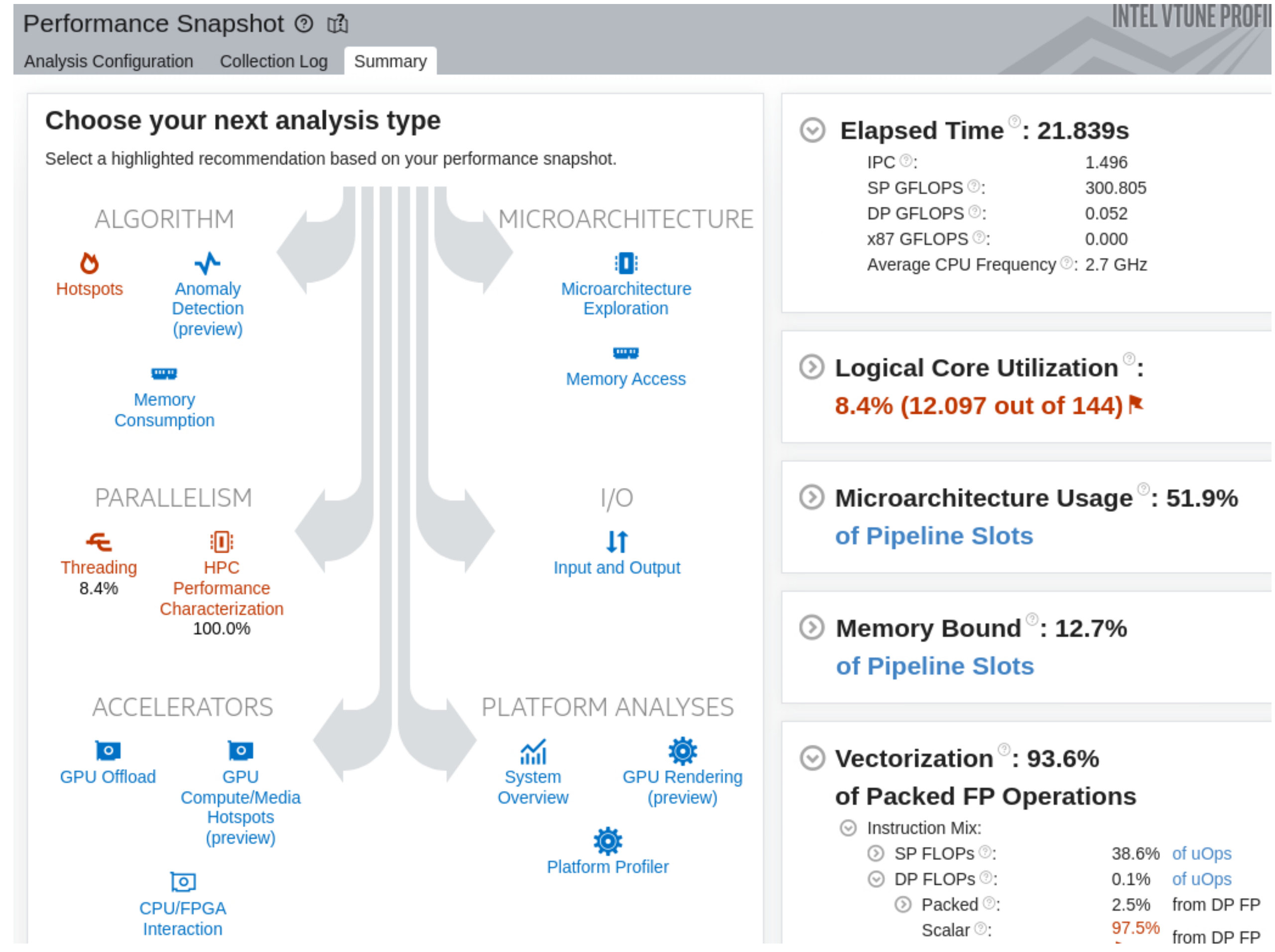
- Data Transfers for Rank-to-Rank Communication – UI representation
  - *>aps-report –transfers-per-communication --format=html <result>*
  - use "-v" to generate the chart by volume



2023 Lenovo Internal. All rights reserved.

Intel VTUNE

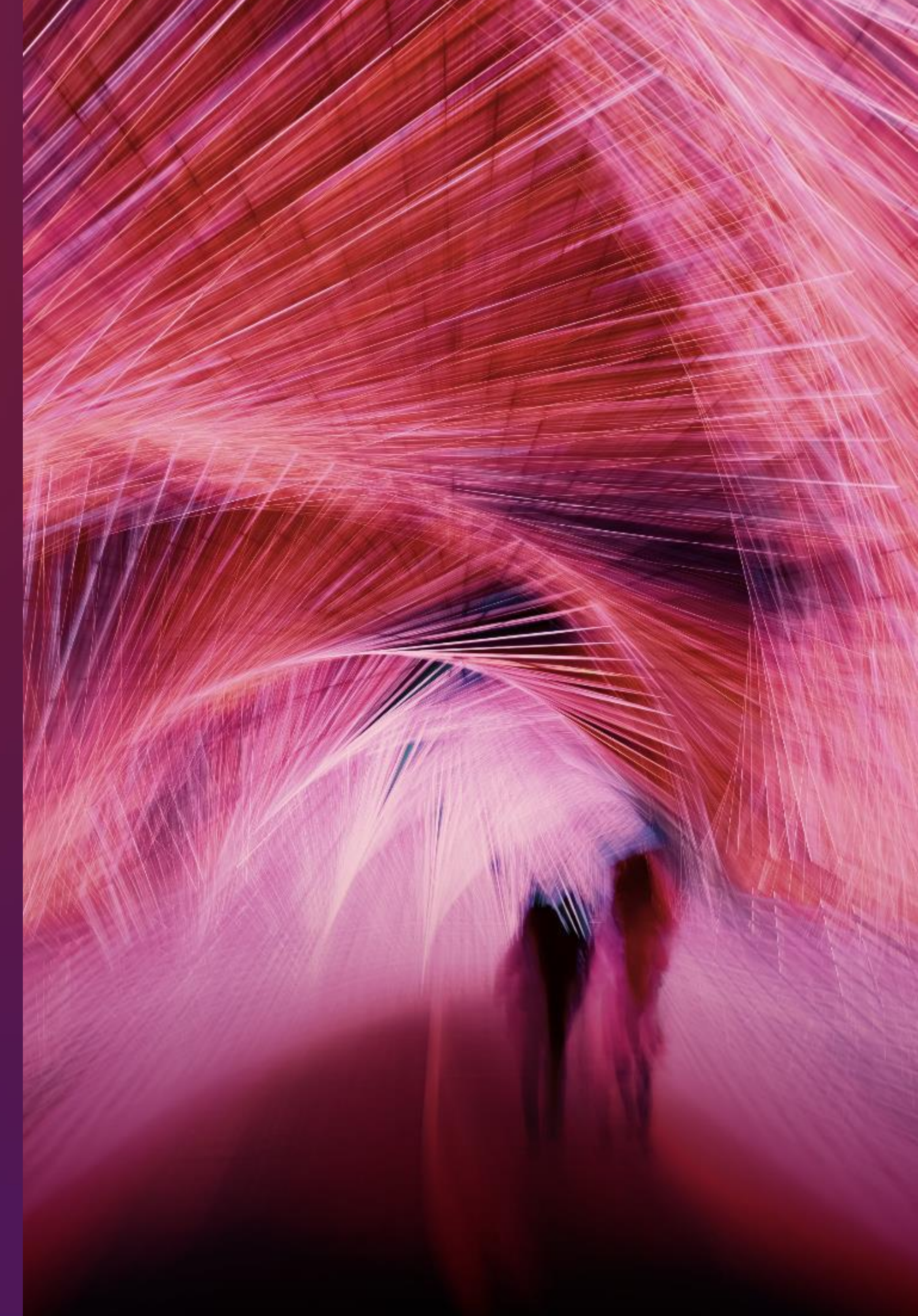
- mpirun –np XX vtune -collect performance-snapshot -r result_dir -- <bin>
- vtune-gui <folder name>



https://www.alcf.anl.gov/files/velesko_vtune_may.pdf

2023 Lenovo Internal. All rights reserved.

Optimisation de l'exécution



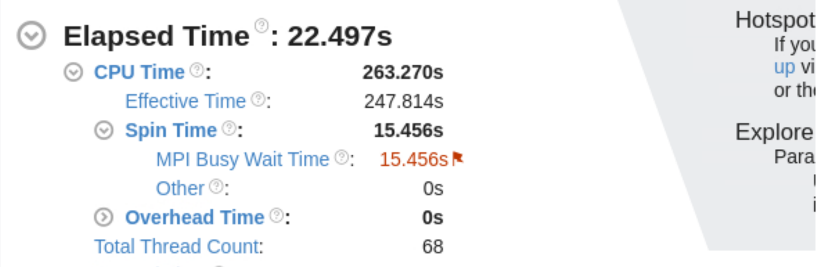
Lenovo 2023 Lenovo Internal. All rights reserved.

HOTSPOTS

2023 Lenovo Internal. All rights reserved.

VTUNE & Hotspot

- mpirun –np XX vtune -collect hotspot -r result_dir -- <bin>
- vtune-gui <folder name>



Top Hotspots

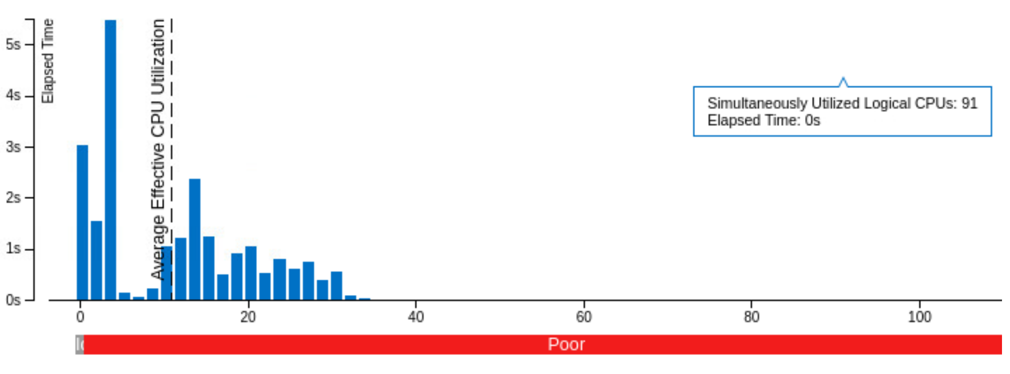
This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time	% of CPU Time
nbnxm_kernel_ElecEw_VdwLJFSw_F_2xmm	libgromacs_mpi.so.6	158.798s	60.3%
PMPI_Waitall	libmpi.so.12	9.198s	3.5%
spread_on_grid._omp_fn.1	libgromacs_mpi.so.6	7.900s	3.0%
nbnxn_make_pairlist_part<NbnxnPairlistCpu>	libgromacs_mpi.so.6	6.050s	2.3%
gather_f_bsplines	libgromacs_mpi.so.6	5.740s	2.2%
[Others]	N/A*	75.584s	28.7%

*N/A is applied to non-summable metrics.

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and utilization value.



Simultaneously Utilized Logical CPUs

Lenovo 2023 Lenovo Internal. All rights reserved.

GPROF

- Compile with -pg flag
- Execute. It generates gmon.out file.
- Call:
  – gprof <binary> gmon.out

```
                 Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 0.13% of 7.51 seconds

index % time    self  children    called     name
                4.45    0.00       1/1           main [2]
[1]     59.3    4.45    0.00       1         checkSTREAMresults [1]
-----------------------------------------------
                                                 <spontaneous>
[2]     59.3    0.00    4.45                 main [2]
                4.45    0.00       1/1           checkSTREAMresults [1]
-----------------------------------------------
                                                 <spontaneous>
[3]     21.0    1.58    0.00                 __libirc_nontemporal_store_512 [3]
-----------------------------------------------
                                                 <spontaneous>
[4]     18.1    1.36    0.00                 __intel_avx_rep_memcpy [4]
-----------------------------------------------
                                                 <spontaneous>
[5]      1.2    0.09    0.00                 __intel_avx_rep_memset [5]
-----------------------------------------------
                                                 <spontaneous>
[6]      0.4    0.03    0.00                 __libirc_nontemporal_store [6]
-----------------------------------------------
```

Lenovo 2023 Lenovo Internal. All rights reserved.

ADVANCED COMPILATION

2023 Lenovo Internal. All rights reserved.

Intel Procedural Optimizations (IPO)

icc -ipo

Analysis and optimization across function and/or source file boundaries, e.g.

- Function inlining; constant propagation; dependency analysis; data & code layout; etc.

2-step process:

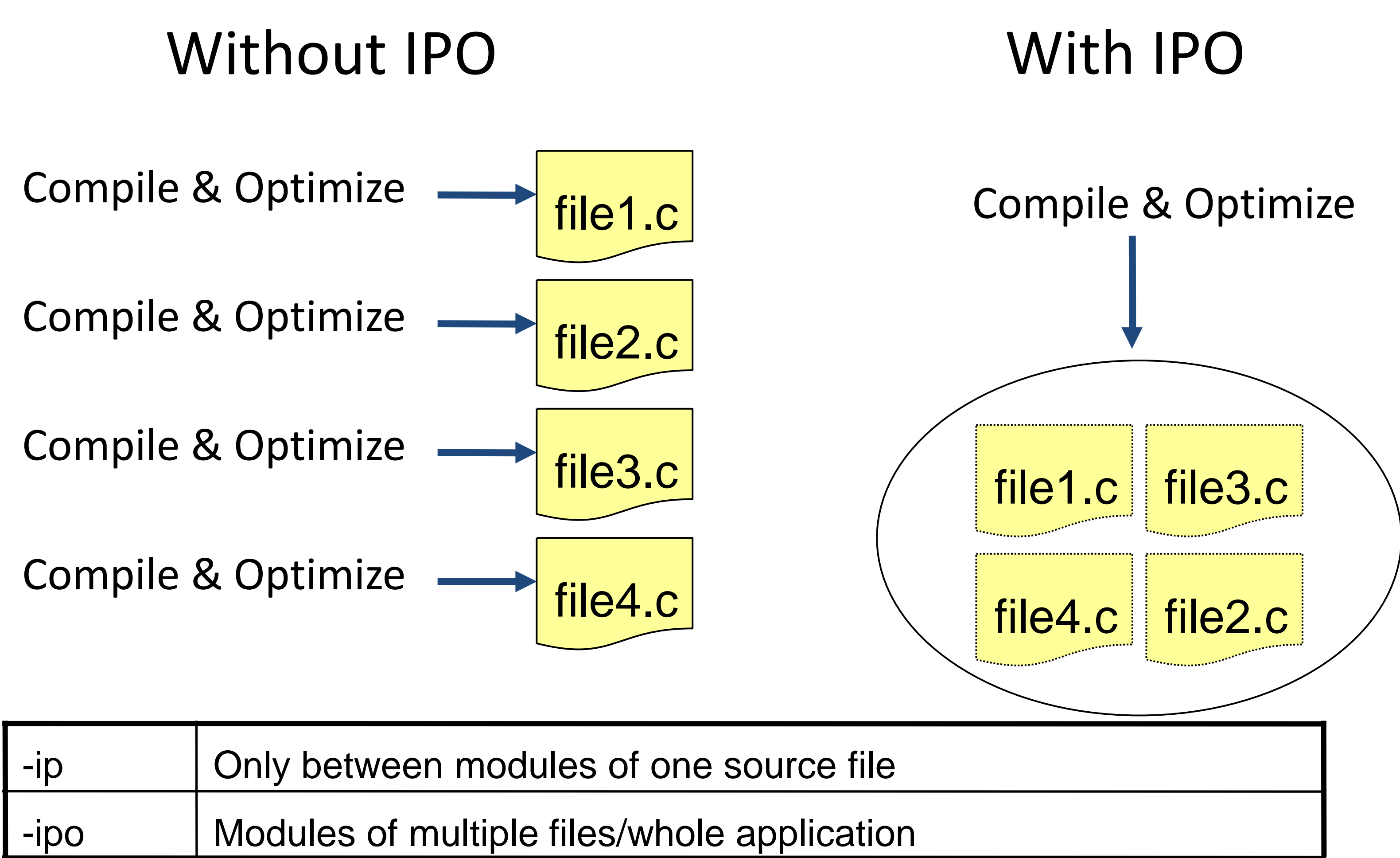
- Compile phase – objects contain intermediate representation
- "Link" phase – compile and optimize over all such objects
- Seamless: linker automatically detects objects built with -ipo and their compile options
- May increase build-time and binary size
- But build can be parallelized with -ipo=n
- Entire program need not be built with IPO, just hot modules

Particularly effective for applications with many smaller functions

Get report on inlined functions with -qopt-report-phase=ipo

2023 Lenovo Internal. All rights reserved.

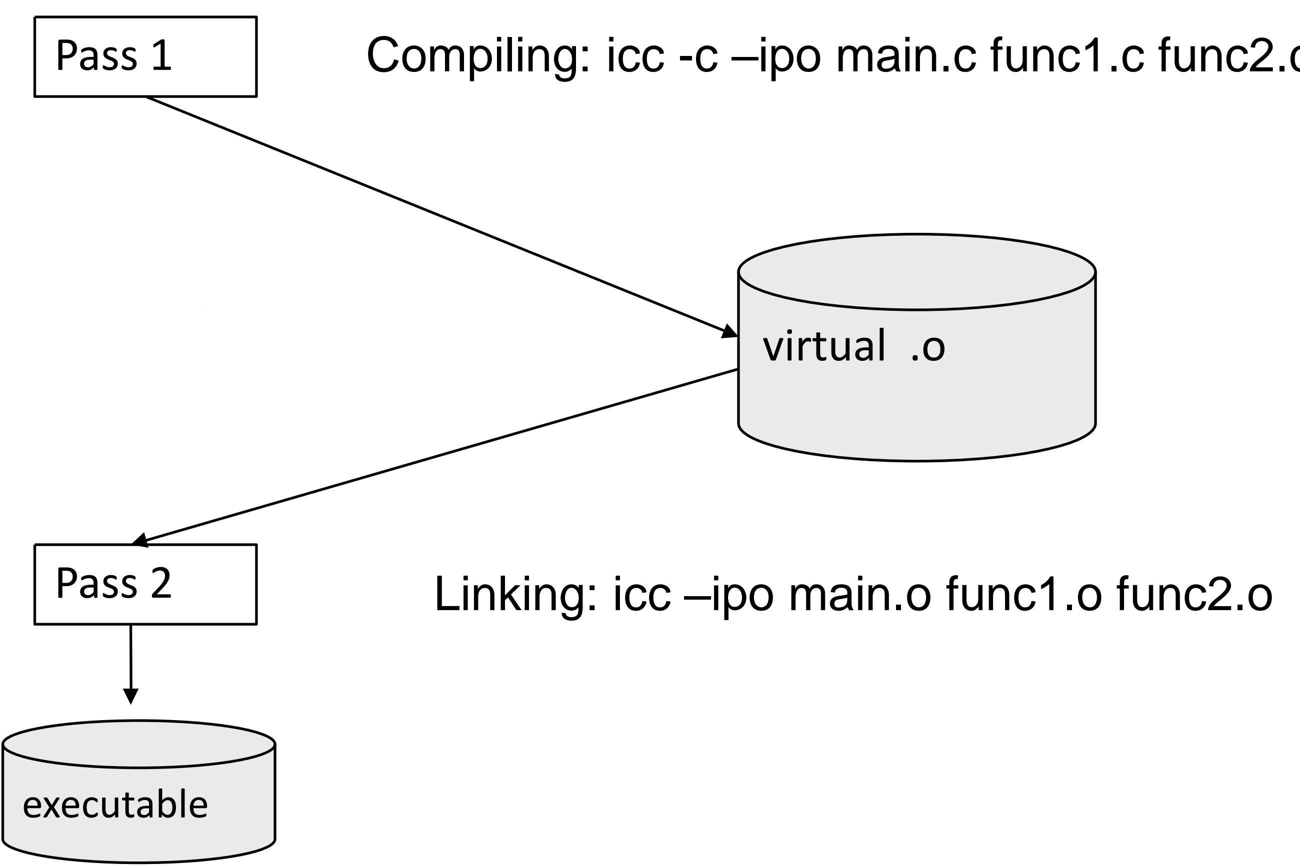
Extends optimizations across file boundaries



-ip	Only between modules of one source file
-ipo	Modules of multiple files/whole application

Lenovo 2023 Lenovo Internal. All rights reserved.

IPO: A Multi-Pass Optimization



Lenovo 2023 Lenovo Internal. All rights reserved.

What you should know about IPO

- Inlining of functions is the most important feature of IPO but there is much more
  - Inter-procedural constant propagation
  - MOD/REF analysis (for dependence analysis)
  - Routine attribute propagation
  - Dead code elimination
  - Induction variable recognition
- Inlining automatically enabled with O2 and O3
- IPO extends compilation time and memory usage

2023 Lenovo Internal. All rights reserved.

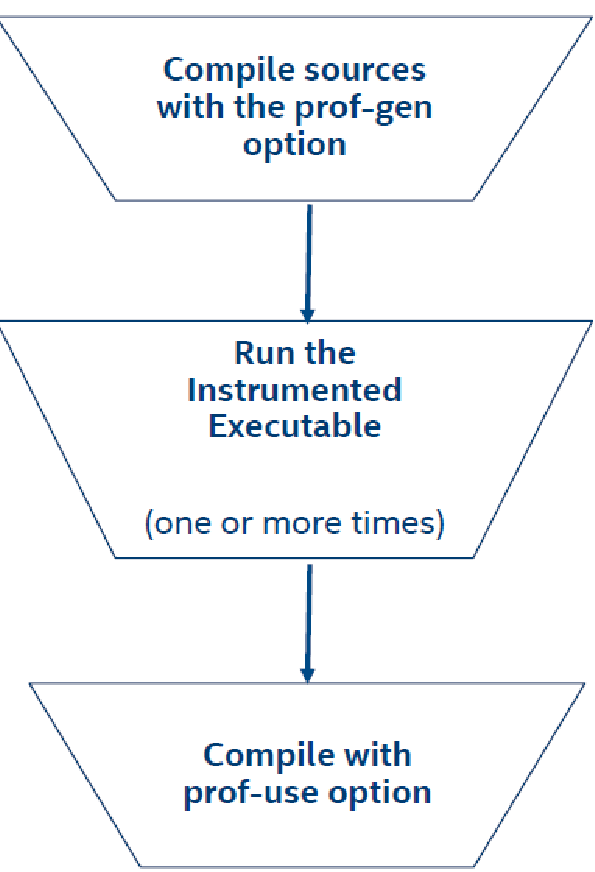
Profile-Guided Optimizations (PGO)

- Static analysis leaves many questions open for the optimizer like:
  - How often is x > y
  - What is the size of count
  - Which code is touched how often

```
if (x > y)
      do_this();
 else
      do that();
```

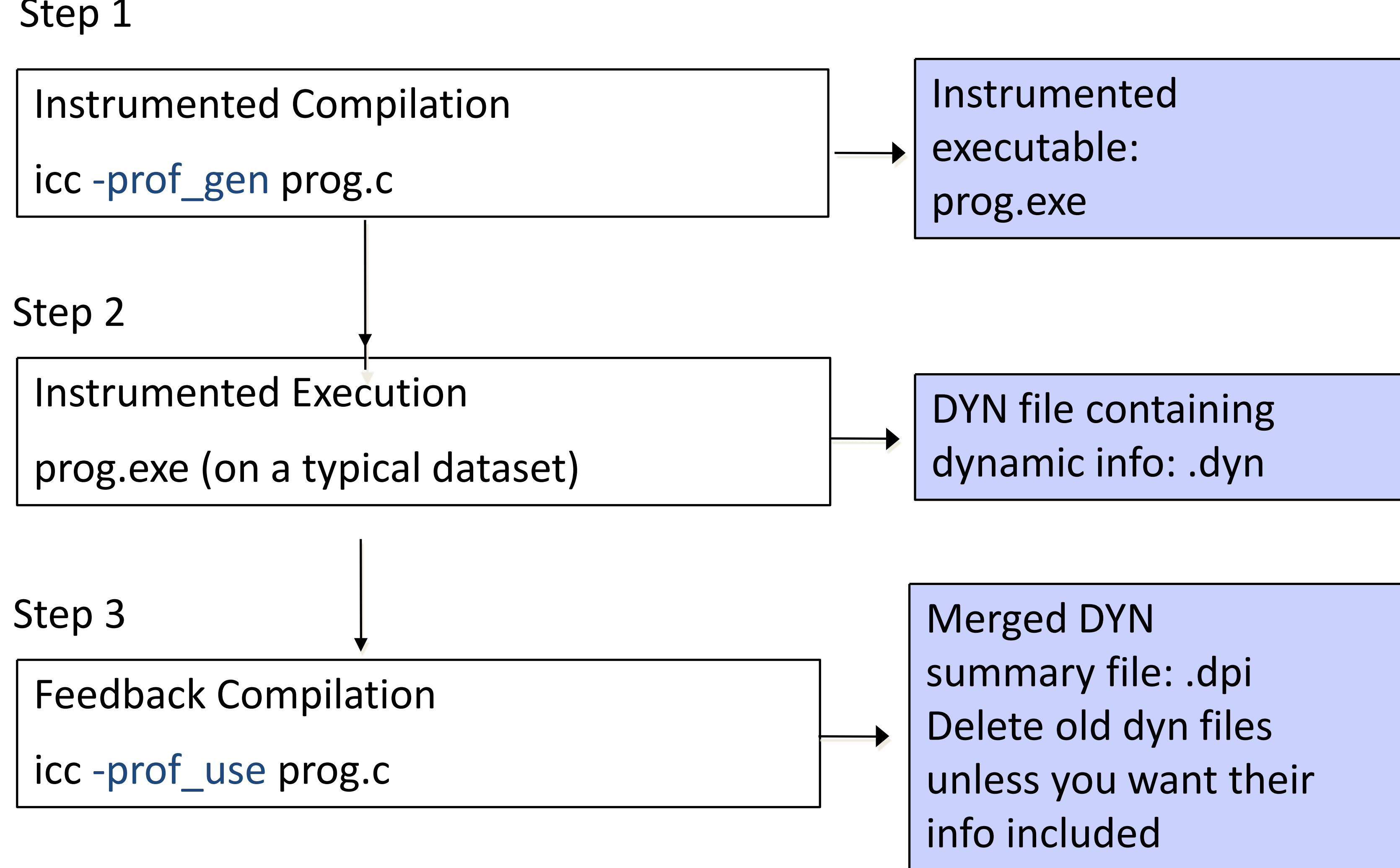
```
for(i=0; i<count; ++i)
do_work();
```

- Use execution-time feedback to guide (final) optimization
- Enhancements with PGO:
  - More accurate branch prediction
  - Basic block movement to improve instruction cache behavior
  - Better decision of functions to inline (help IPO)
  - Can optimize function ordering
  - Switch-statement optimization
  - Better vectorization decisions



2023 Lenovo Internal. All rights reserved.

PGO Usage: Three-Step Process



Lenovo 2023 Lenovo Internal. All rights reserved.

Auto-Parallelization Invocation

- Intel Compiler can automatically parallelize the code
- Only applicable to very regular loops
- Pointers make it hard for the compiler to deduce memory layout

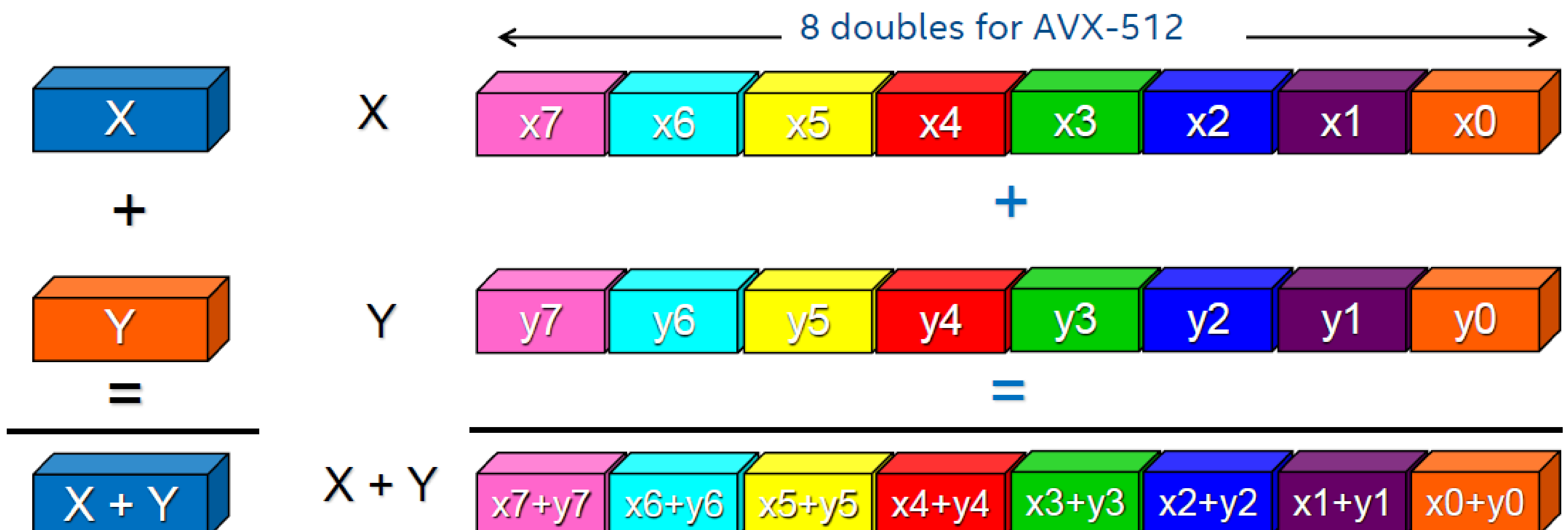
Options	Value	Purpose
-parallel		Enable automatic parallelization
-par-threshold		Parallelization threshold
	0	Parallelize always
	100	Parallelize only if performance gain is 100%
	50	Parallelize if probability of performance gain is 50%

Lenovo 2023 Lenovo Internal. All rights reserved.

SIMD: Single Instruction, Multiple Data

```
for (i=0; i<n; i++) z[i] = x[i] + y[i];
```

- Scalar mode
  - one instruction produces one result
  - E.g. vaddss, (vaddsd)
- Vector (SIMD) mode
  - one instruction can produce multiple results
  - E.g. vaddps, (vaddpd)



2023 Lenovo Internal. All rights reserved.

Evolution of SIMD for Intel Processors

128b SIMD

256b SIMD

512b SIMD

Willamette 2000	Prescott 2004	Merom 2006	Penryn 2007	Nehalem 2007	Sandy Bridge 2011	Haswell 2013	Knights Landing 2015	Skylake server 2015
							AVX-512 ER/PR	AVX-512 VL/BW/DQ
							AVX-512 F/CD	AVX-512 F/CD
						AVX2	AVX2	AVX2
					AVX	AVX	AVX	AVX
				SSE4.2	SSE4.2	SSE4.2	SSE4.2	SSE4.2
			SSE4.1	SSE4.1	SSE4.1	SSE4.1	SSE4.1	SSE4.1
		SSSE3	SSSE3	SSSE3	SSSE3	SSSE3	SSSE3	SSSE3
	SSE3	SSE3	SSE3	SSE3	SSE3	SSE3	SSE3	SSE3
SSE2	SSE2	SSE2	SSE2	SSE2	SSE2	SSE2	SSE2	SSE2
SSE	SSE	SSE	SSE	SSE	SSE	SSE	SSE	SSE
MMX	MMX	MMX	MMX	MMX	MMX	MMX	MMX	MMX

2023 Lenovo Internal. All rights reserved.

Many ways to vectorize

Compiler:
Auto-vectorization (no change of code)

Compiler:
Auto-vectorization hints (#pragma vector, ...)

Compiler:
OpenMP* SIMD directives

SIMD intrinsic class
(e.g.: F32vec, F64vec, ...)

Vector intrinsic
(e.g.: _mm_fmadd_pd(...), _mm_add_ps(...), ...)

Assembler code
(e.g.: [v]addps, [v]addss, ...)

 2023 Lenovo Internal. All rights reserved.

Basic Intel Flags for Vectorization (1/2)

-x<code>

- Might enable Intel processor specific optimizations
- Processor-check added to "main" routine:
  Application errors in case SIMD feature missing or non-Intel processor with appropriate/informative message

<code> indicates a feature set that compiler may target (including instruction sets and optimizations)

- Microarchitecture code names: BROADWELL, HASWELL, IVYBRIDGE, KNL, KNM, SANDYBRIDGE, SILVERMONT, SKYLAKE, SKYLAKE-AVX512
- SIMD extensions: CORE-AVX512, CORE-AVX2, CORE-AVX-I, AVX, SSE4.2, etc.
- Example: icc -xCORE-AVX2 test.c
  ifort -xSKYLAKE test.f90

Lenovo 2023 Lenovo Internal. All rights reserved.

Basic Intel Flags for Vectorization (2/2)

-ax<code>

- Multiple code paths: baseline and optimized/processor-specific
- Optimized code paths for Intel processors defined by <code>
- Multiple SIMD features/paths possible, e.g.: -axSSE2,AVX
- Baseline code path defaults to -msse2 (/arch:sse2)
- The baseline code path can be modified by –m<code> or –x<code>
- Example: icc -axCORE-AVX512 -xAVX test.c
  icc -axCORE-AVX2,CORE-AVX512 test.c

-m<code>

- No check and no specific optimizations for Intel processors:
  Application optimized for both Intel and non-Intel processors for selected SIMD feature
- Missing check can cause application to fail in case extension not available

- -xHost

2023 Lenovo Internal. All rights reserved.

Intel AVX512 Code Generation

Compile with processor-specific option -xCORE-AVX512

By default it will not optimize for more restrained ZMM register usage which works best for certain applications

A new compiler option -qopt-zmm-usage=low|high is added to enable a smooth transition from AVX2 to AVX-512

```
void foo(double *a, double *b, int size) {
  #pragma omp simd
  for(int i=0; i<size; i++) {
     b[i]=exp(a[i]);
  }
}
```

```
icpc -c -xCORE-AVX512 -qopenmp -qopt-report:5 foo.cpp
```

```
remark #15305: vectorization support: vector length 4
  …
remark #15321: Compiler has chosen to target XMM/YMM
vector. Try using -qopt-zmm-usage=high to override
  …
remark #15478: estimated potential speedup: 5.260
```

Lenovo 2023 Lenovo Internal. All rights reserved.

Compiler Optimization Reports

- -qopt-report[=n]: tells the compiler to generate an optimization report

  n: (Optional) Indicates the level of detail in the report. You can specify values 0 through 5. If you specify zero, no report is generated. For levels n=1 through n=5, each level includes all the information of the previous level, as well as potentially some additional information. Level 5 produces the greatest level of detail. If you do not specify n, the default is level 2, which produces a medium level of detail.

- -qopt-report-phase[=list] specifies one or more optimizer phases for which optimization reports are generated.

  loop: the phase for loop nest optimization
  vec: the phase for vectorization
  par: the phase for auto-parallelization
  all: all optimizer phases

- -qopt-report-filter=string: specified the indicated parts of your application, and generate optimization reports for those parts of your application.

Lenovo 2023 Lenovo Internal. All rights reserved.

Optimization Report Example (1/3)

$ icc -c -xcommon-avx512 **-qopt-report=3** -qopt-report-phase=loop,vec foo.c

Creates foo.optrpt summarizing which optimizations the compiler performed or tried to perform.
Level of detail from 0 (no report) to 5 (maximum).
-qopt-report-phase=loop,vec asks for a report on vectorization and loop optimizations only
Extracts:

LOOP BEGIN at foo.c(4,3)
Multiversioned v1
  remark #25228: Loop multiversioned for Data Dependence…
  remark #15300: LOOP WAS VECTORIZED
  remark #15450: unmasked unaligned unit stride loads: 1
  remark #15451: unmasked unaligned unit stride stores: 1
  …. (loop cost summary) ….
LOOP END

```
#include <math.h>
void foo (float * theta, float * sth)  {
  int i;
  for (i = 0; i < 512; i++)
       sth[i] = sin(theta[i]+3.1415927);
}
```

LOOP BEGIN at foo.c(4,3)
<**Multiversioned v2**>
  remark #15304: loop was not vectorized: non-vectorizable loop instance from multiversioning
LOOP END

Lenovo 2023 Lenovo Internal. All rights reserved.

Optimization Report Example (2/3)

$ icc -c -xcommon-avx512 -qopt-report=4 -qopt-report-phase=loop,vec -qopt-report-file=stderr **-fargument-noalias** foo.c

report to stderr instead of foo.optrpt

```
...
LOOP BEGIN at foo.c(4,3)
...
   remark #15417: vectorization support: number of FP up converts: single precision to double precision 1
   remark #15418: vectorization support: number of FP down converts: double precision to single precision 1
   remark #15300: LOOP WAS VECTORIZED
   remark #15450: unmasked unaligned unit stride loads: 1
   remark #15451: unmasked unaligned unit stride stores: 1
   remark #15475: --- begin vector cost summary ---
   remark #15476: scalar cost: 111
   remark #15477: vector cost: 10.310
   remark #15478: estimated potential speedup: 10.740
   remark #15482: vectorized math library calls: 1
   remark #15487: type converts: 2
   remark #15488: --- end vector cost summary ---
   remark #25015: Estimate of max trip count of loop=32
LOOP END
```

```c
#include <math.h>
void foo (float * theta, float * sth)  {
  int i;
  for (i = 0; i < 512; i++)
       sth[i] = sin(theta[i]+3.1415927);
}
```


2023 Lenovo Internal. All rights reserved.

Optimization Report Example (3/3)

```
$ icc -S -xcommon-avx512 -qopt-report=4 -qopt-report-phase=loop,vec -qopt-report-file=stderr -
fargument-noalias foo.c
LOOP BEGIN at foo2.c(4,3)
...
remark #15305: vectorization support: vector length 32
remark #15300: LOOP WAS VECTORIZED
  remark #15450: unmasked unaligned unit stride loads: 1
  remark #15451: unmasked unaligned unit stride stores: 1
  remark #15475: --- begin vector cost summary ---
  remark #15476: scalar cost: 109
  remark #15477: vector cost: 5.250
  remark #15478: estimated potential speedup: 20.700
  remark #15482: vectorized math library calls: 1
  remark #15488: --- end vector cost summary ---
  remark #25015: Estimate of max trip count of loop=32
LOOP END

$ grep sin foo.s
      call      __svml_sinf16_b3
```

```
#include <math.h>
void foo (float * theta, float * sth)  {
  int i;
  for (i = 0; i < 512; i++)
       sth[i] = sinf(theta[i]+3.1415927f);
}
```


2023 Lenovo Internal. All rights reserved.

Intel Floating Point Flags

-fp-model

- fast [=1] allows value-unsafe optimizations (default)
- fast=2 allows a few additional approximations
- precise value-safe optimizations only
- source | double | extended imply "precise" unless overridde
- except enable floating-point exception semantics
- strict precise + except + disable fma + don't assume default floating-point environment
- consistent most reproducible results between different processor types and optimization options

-fp-model precise -fp-model source

- recommended for best reproducibility
- also for ANSI/ IEEE standards compliance, C++ & Fortran
- "source" is default with "precise" on Intel 64

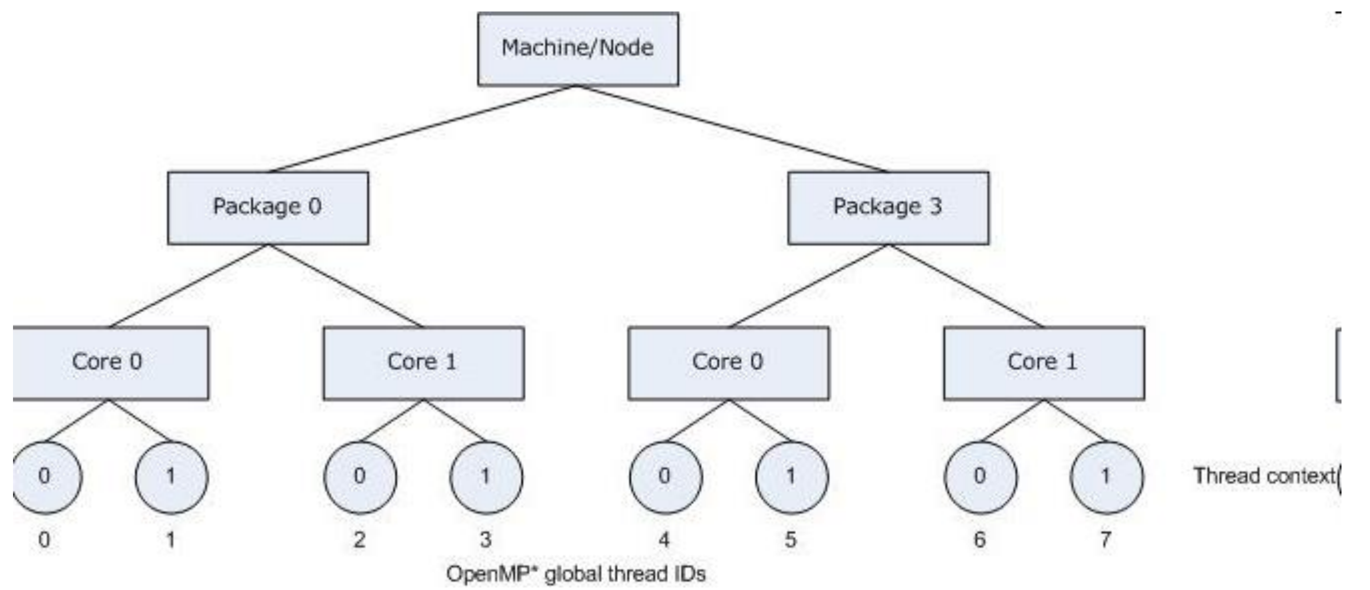
Lenovo 2023 Lenovo Internal. All rights reserved.

BINDING & MAPPING

2023 Lenovo Internal. All rights reserved.

Processor Affinity Management: Threads Affinity

- OMP_NUM_THREADS / KMP_NUM_THREADS / GOMP_CPU_AFFINITY



2023 Lenovo Internal. All rights reserved.

Intel MPI
Task Placement

- Default process layout is “group round-robin”
  - Puts consecutive MPI processes on same node (this may be changed soon)
- Default process layout can be overriden through
  - MPIEXEC Command Options
  - I_MPI_PERHOST Environment Variable

Setting	Purpose
-perhost <Nb Tasks> -ppn <Nb Tasks> -grr <Nb Tasks>	Processes per host = group round-robin distribution
-rr I_MPI_PERHOST=1	Round-robin distribution; same as -perhost 1
I_MPI_PERHOST=all	Map processes to all logical CPUs on a node
I_MPI_PERHOST=allcores	Map processes to all physical CPUs on a node

Lenovo 2023 Lenovo Internal. All rights reserved.

Intel MPI
Processor Affinity Management: MPI Jobs

- Processor affinity is managed through environment variable
  - I_MPI_PIN_PROCESSOR_LIST
- Possible values
  - List of processors
    - I_MPI_PIN_PROCESSOR_LIST=0,6
  - Range of processors
    - I_MPI_PIN_PROCESSOR_LIST=0-11
  - Combination
    - I_MPI_PIN_PROCESSOR_LIST=0-4,6-9

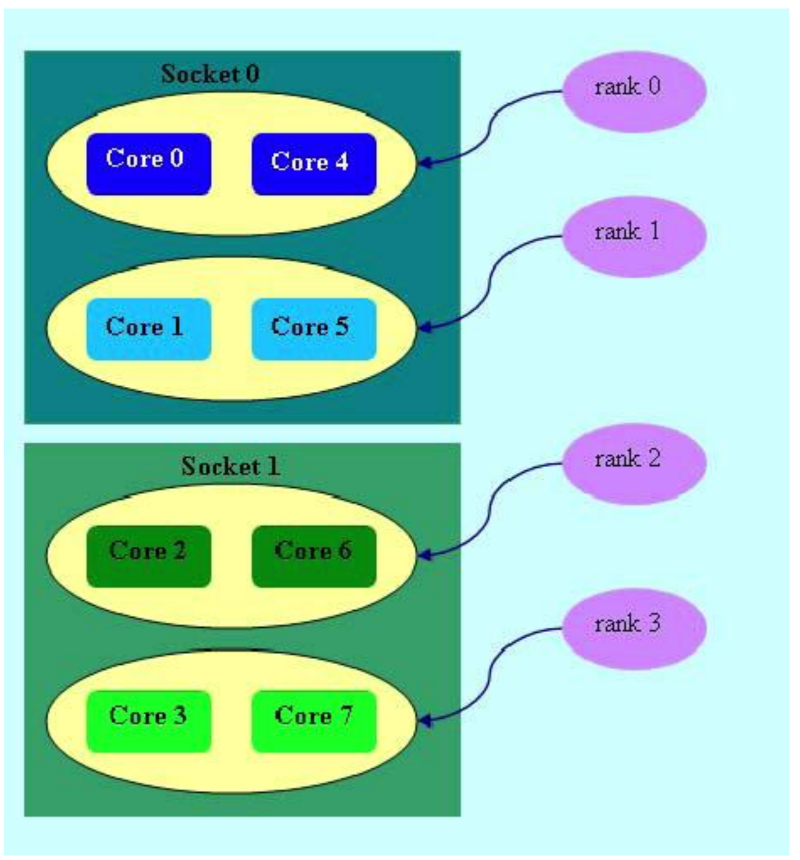
Lenovo 2023 Lenovo Internal. All rights reserved.

Intel MPI
Processor Affinity Management: MPI/OpenMP Jobs

- Processor affinity is managed through environment variable
  - I_MPI_PIN_DOMAIN
- An Intel MPI domain contains
  - One single MPI process
  - All its attached threads
- Syntax Forms
  - Domain description through multi-core terms
  - Domain description through domain size and domain member layout
  - Explicit domain description through bit mask
- Tips'n Tricks
  - Use explicit domain mask to be sure of desired processor affinity
  - Make sure of the outcome through monitoring

Lenovo 2023 Lenovo Internal. All rights reserved.

Intel MPI
Processor Affinity Management: MPI/OpenMP Jobs



Lenovo 2023 Lenovo Internal. All rights reserved.

Intel MPI
Processor Affinity Management: MPI/OpenMP Jobs

Option	Variable Value	Purpose
Multi-Core	core | socket | node	Create one domain per physical core / socket / node
Explicit Shape	<Size>[:<Layout>] omp auto <Size> :platform :compact :scatter	Create domains shaped according to following rules Number of threads Automatic (Nb Logical Processors / Nb MPI Processes) Specified size Following BIOS numbering of logical processors On closest logical processors (default) On farthest logical processors
Bit Mask	[<Mask1>,<Mask2>]	Create explicit domains from hexadecimal masks (cf. next slide)

Lenovo 2023 Lenovo Internal. All rights reserved.

Intel MPI
Processor Affinity Management: MPI/OpenMP Jobs

- Bit Mask
  - Bit mask is expressed through hexadecimal format
  - Calculation Method
    - Write bit mask in binary format
    - Translate value into hexadecimal
  - Example
    - Execution configuration: 2 processes per node x 6 threads per process
    - Bit Masks
      - Domain #1: 000000111111 ⇨ 3F
      - Domain #2: 111111000000 ⇨ FC0
- Usage Notes
  - Process is not always bound to first logical processor of its domain
    - Behaviour seems to have changed in latest Intel MPI version
      - To be confirmed
  - Process affinity does not imply threads affinity
    - Other environment variables required
      - cf. Threads Affinity

2023 Lenovo Internal. All rights reserved.

Open MPI
Processor Affinity Management: MPI Jobs

- Processor affinity is managed through MCA parameter
  - mpi_paffinity_alone = 1
- MCA parameter set through (highest priority first):
  - Command Line
    - -mca mpi_paffinity_alone 1
  - Environment Variable
    - OMPI_MCA_mpi_paffinity_alone=1
  - Configuration File
- Requires exclusive use of the computation nodes
- Processor affinity automatically activates memory affinity

Lenovo 2023 Lenovo Internal. All rights reserved.

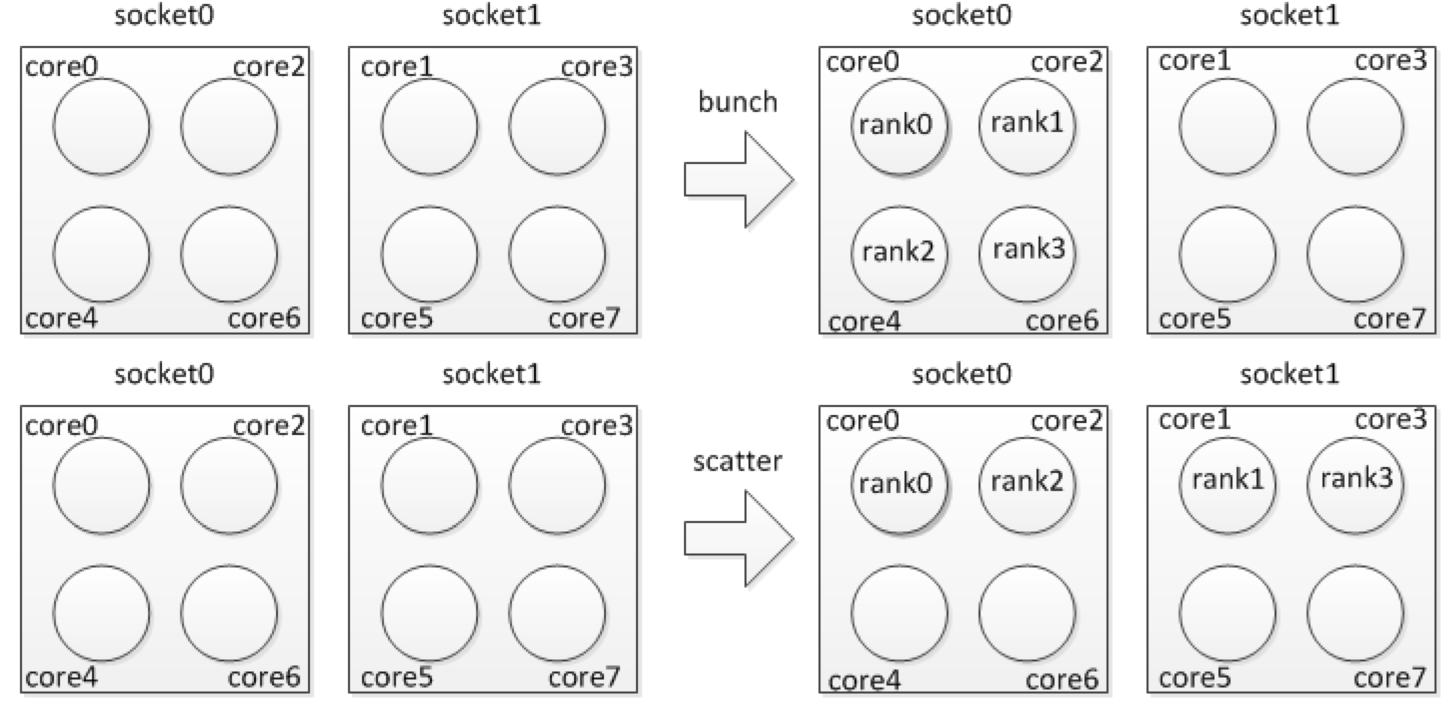
Open MPI
Processor Affinity Management: MPI/OpenMP Jobs

- Processor affinity is managed through rankfile
  - Rankfile allows task placement as well
  - Rankfile provided to MPIRUN command through argument -rf
- Rankfile Syntax
  - rank <Rank #>=<Hostname> slot=<Processor ID>
    - <Processor ID> is OS core ID
      - cf. CPUINFO listing
- Examples
  - 6 MPI Processes / Node, 2 Threads / Process
    - rank 0=ku-auh-ccn01-ib slot=0,1
      rank 1=ku-auh-ccn01-ib slot=2,3
      rank 2=ku-auh-ccn01-ib slot=4,5
      …
  - 2 MPI Processes / Node, 6 Threads / Process
    - rank 0=ku-auh-ccn01-ib slot=0-5
      rank 1=ku-auh-ccn01-ib slot=6-11
      …

Lenovo 2023 Lenovo Internal. All rights reserved.

MVAPICH2
Processor Affinity

- Two policies: bunch and scatter
- Syntax: mpirun_rsh -np 4 -hostfile hosts MV2_CPU_BINDING_POLICY=bunch ./a.out



- By hands: MV2_CPU_MAPPING=0:1:4:5
- For Threads: MV2_CPU_MAPPING=0,2,3,4:1:5:6 (task #0 binding to cores 0,2-4)

Lenovo 2023 Lenovo Internal. All rights reserved.

MPI OPTIMIZATIONS

2023 Lenovo Internal. All rights reserved.

Intel MPI Environment Variables

Variable Name	Value	Purpose
I_MPI_DEBUG	Level 0 / 1 / 2 / 3 / 4 / 5	Level of debugging for execution
I_MPI_FABRICS	<Intra-Node>: <Inter-Node>	Fabrics to be used for intra-node and inter-node communications
I_MPI_OUTPUT_CHUNK_SIZE	<Size (KB)>	Size of the stdout / stderr buffer
I_MPI_PIN_DOMAIN	<Domain Mask>	Processor affinity management in mixed MPI/OpenMP mode
I_MPI_PIN_PROCESSOR_LIST	<Processor List>	Processor affinity management in pure-MPI mode
I_MPI_ADJUST_<COLLECTIVE>	<ALGO NUMBER>	Set the algorithm <ALGO NUMBER> when running MPI collective <COLLECTIVE>

Lenovo 2023 Lenovo Internal. All rights reserved.

Intel MPI
Typical Sanity Checks

- Check selected fabric initialization
  - Through environment variable I_MPI_DEBUG=2
- Check task placement
  - Through environment variable I_MPI_DEBUG=3
- Check process pinning
  - Through environment variable I_MPI_DEBUG=4

Lenovo 2023 Lenovo Internal. All rights reserved.

Optimizations of Intel MPI: Binding

- I_MPI_PIN=on Pinning Enabled
- I_MPI_PIN_MODE=pm Use Hydra for Pinning
- I_MPI_PIN_RESPECT_CPUSET=on Respect process affinity mask
- I_MPI_PIN_RESPECT_HCA=on Pin according to HCA socket
- I_MPI_PIN_CELL=unit Pin on all logical cores
- I_MPI_PIN_DOMAIN=auto:compact Pin size #lcores/#ranks : compact
- I_MPI_PIN_ORDER=compact Order domains adjacent
- I_MPI_PIN_PROCESSOR_LIST

- I_MPI_JOB_RESPECT_PROCESS_PLACEMENT=0: conflict with job scheduler

Lenovo 2023 Lenovo Internal. All rights reserved.

Optimization : I/O

- AVBP application is using MPIIO routines to write output files (thru parallel HDF5 library)
- Using default execution environment and large test case on 512 MPI tasks, we got the following profile for I/O routines: (mpi task 0 - IMPI - no specific tuning for I/O)

MPI Routine	#calls	avg. bytes	time(sec)
MPI_File_close	1	0.0	0.007
MPI_File_open	1	0.0	0.491
MPI_File_set_view	84	0.0	89.461
MPI_File_write_at	6	21.3	0.410
MPI_File_write_at_all	42	33753.8	90.456

- These calls to MPI I/O routines will use embedded versions from MPI library used to perform the parallel I/O operations, and specially the ROMIO implementation.

Lenovo 2023 Lenovo Internal. All rights reserved.

Optimization I/O

ROMIO in IntelMPI

- The latest version of ROMIO implementation in [MPICH 3.2] is introducing several new features like asynchronous collectives
- Latest IntelMPI library includes previous version and therefore does not contain latest features, but still provides optimized implementation:
  - IntelMPI allows the user selecting the filesystem matching the execution environment, in our case GPFS filesystem:

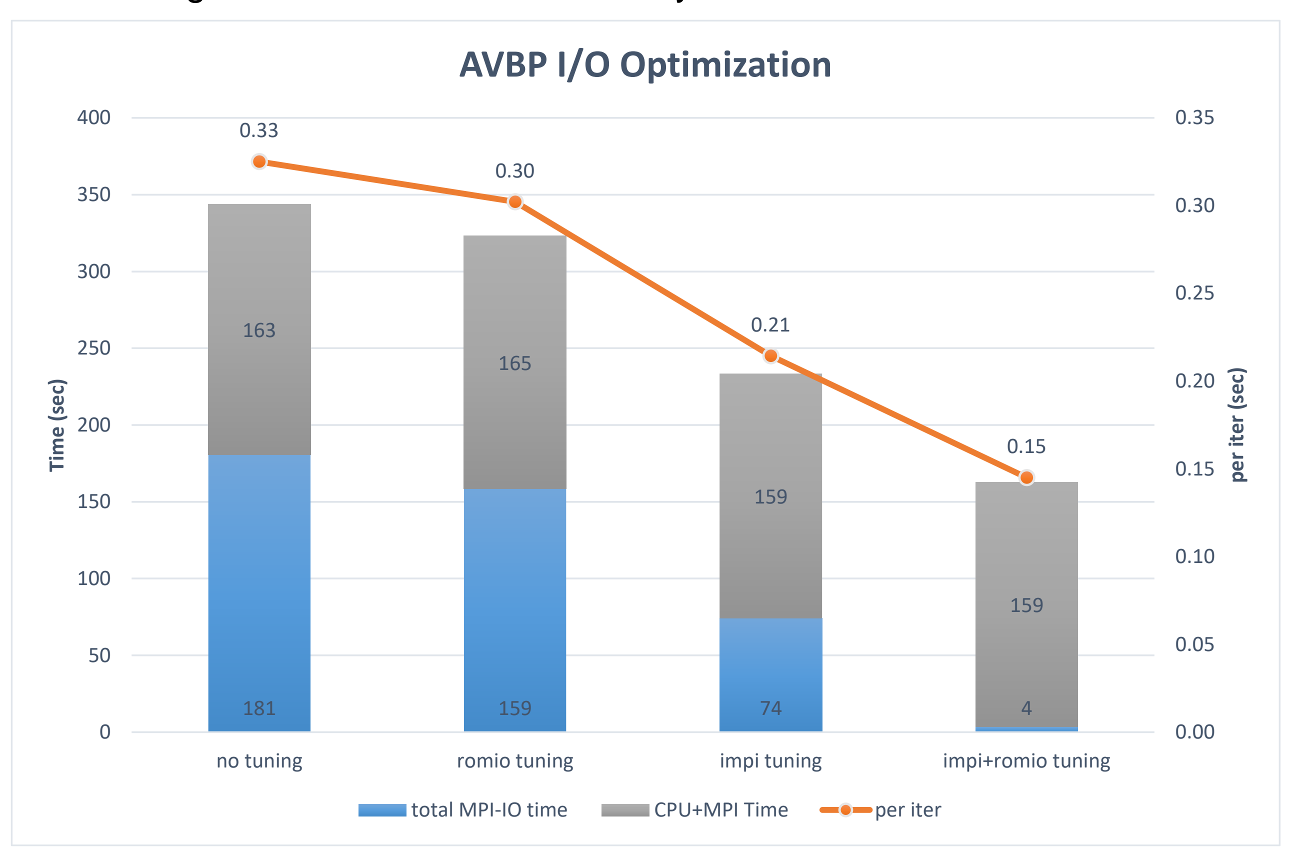
```
export I_MPI_EXTRA_FILESYSTEM=on
Outdated : export I_MPI_EXTRA_FILESYSTEM_LIST=gpfs
Replaced by:  I_MPI_EXTRA_FILESYSTEM_FORCE=gpfs
```

  - It provides **great performance improvement** over default options

- **Additional optimization** can be made on ROMIO performance by using :

```
export ROMIO_PRINT_HINTS=yes
export ROMIO_HINTS=./romio_hints.txt
cat > $ROMIO_HINTS  << EOOOF
romio_cb_read disable
romio_cb_write enable
romio_ds_read disable
romio_ds_write disable
EOOOF
```

- Global performance improvement is huge: **> 40x for the I/O time**



simulation_end_iteration = 1000
save_solution.iteration = 1000
save_temporal.iteration = 100
TwoPhasesCombustion test case, Lenox, 512 MPI, 512 cores, IFORT 16.0.1, IMPI 5.1.2 Dapludmixed

Lenovo 2023 Lenovo Internal. All rights reserved.

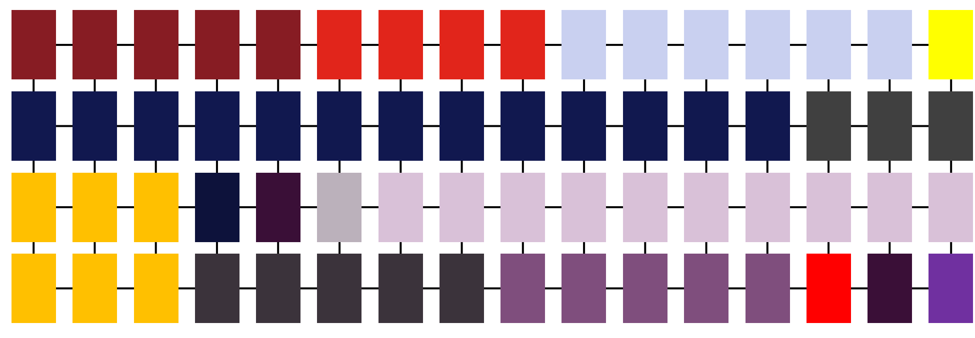
Smarter Throughput for Lenovo HPC Clusters

Current Status

- On HPC cluster in production at customer site, scientific simulations are submitted to execution to a job scheduler, which is the brain managing job queues, executions, steps, priorities, hardware assignments…
- Usual strategy for a job submission is to book a range of compute nodes and launch the job on these nodes.
  - One application using all nodes exclusively
  - Fixed resources for single code: processor cycles, memory/network/filesystem bandwidths, hyperthreads…

- Pros:
  - Easy to manage and report accounting from job scheduler point of view
  - Reproducibility of code performance across runs

- Cons:
  - A single application is usually not able to saturate all available resources
    - Wasted CPU cycles or memory/network/filesystem bandwidths
  - Power consumption of the nodes may not be optimal



HPC cluster:
- Each rectangle is a node
- Each color represents a job
- **Each node is filled with one single code**

Lenovo 2023 Lenovo Internal. All rights reserved.

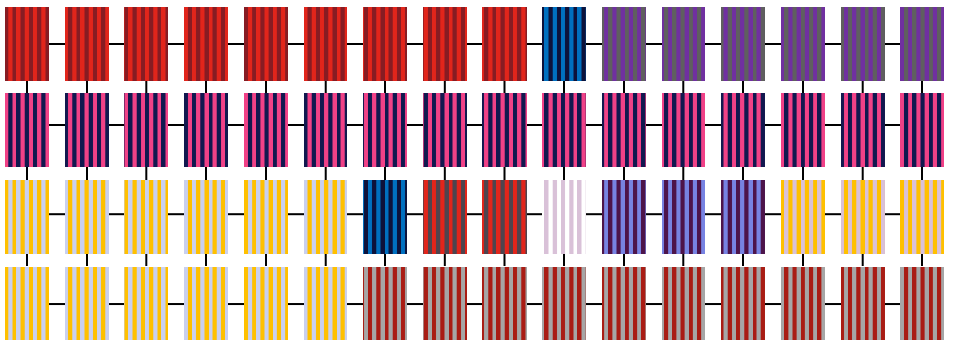
Smarter Throughput for Lenovo HPC Clusters

Concept

- We propose to optimize usage of cluster resources at job scheduling level by sharing computing nodes between applications stressing different resources:
  - For instance: One code stressing CPU cycles, another saturating memory bandwidth, on same nodes, using 2 times more nodes.

- Pros:
  - Usage of compute, network, I/O, memory… resources is optimized
  - Better throughput performance at cluster level
  - Better performance for some jobs (memory BW bound, network BW bound…)
  - Smaller power consumption for same workload computed (?? To be checked)

- Cons:
  - Performance reproducibility can't be guaranteed
  - Sometimes lower pure performance for some jobs
  - More complex to setup: Job scheduler…

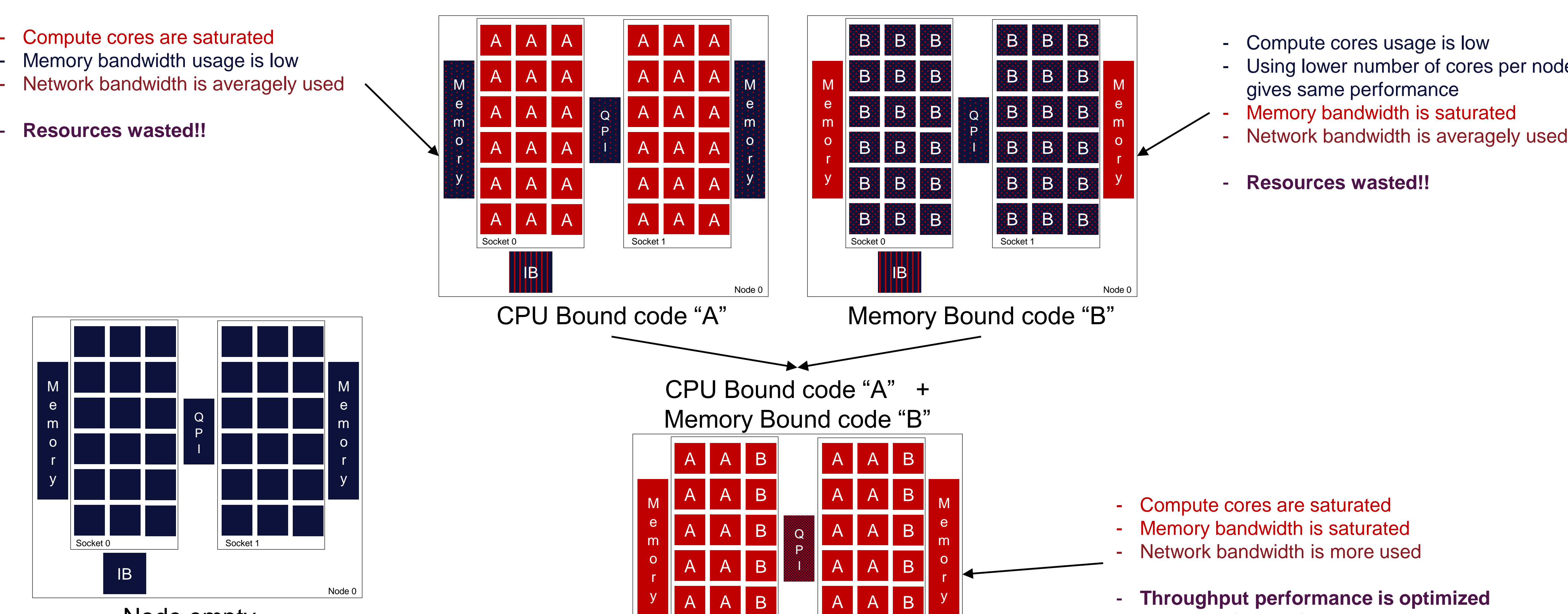


HPC cluster:
- Each rectangle is a node
- Each color represents a job
- **Each node is filled with one or more codes**

Lenovo 2023 Lenovo Internal. All rights reserved.

Smarter Throughput for Lenovo HPC Clusters

Illustration for MPI codes



Lenovo 2023 Lenovo Internal. All rights reserved.

Smarter Throughput for Lenovo HPC Clusters

Customer case : context

- Customer has 2 types of codes running on their computing center:
  - CPU Bound codes, represented by CodeA (CFD application)
  - Memory Bandwidth bound codes, represented by CodeB (meso-scale atmospheric application)
- Near future usage of HPC center is going to be half / half with these types of applications.
  - Benchmark requirements are standalone runs for both codes on 5/10/20 nodes and estimation of throughput for the whole system.
  - No output I/O during benchmark runs, only medium I/O reads for input files.

- Proposed system by Lenovo:
  - Intel Xeon SKL-6132 14 cores @ 2.6GHz, network OPA 100Gbs with blocking factor 2:1 and 8X adapters 58Gbs.

- Throughput simulation based on 100 jobs of each required benchmark run = 600 jobs.
  - CodeA on 5 / 10 / 20 nodes, CodeB on 5 / 10 / 20 nodes.
  - Trivial interpolation for global performance simulation
  - Memory bandwidth bound code CodeB launched with optimal number of processes per node to reach roof performance : ppn=12
  - CPU bound code CodeA is then using all remaining cores on the compute node: ppn=24
  - MPI processes of each code are "striped" across the 2 processors, i.e. both are using the 2 processors:
    - Almost no degradation impact for CodeA
    - CodeB benefits from the full memory bandwidth of the node (memory bandwidth per core is doubled)

2023 Lenovo Internal. All rights reserved.

Smarter Throughput for Lenovo HPC Clusters

Customer case: 2 codes study on Broadwell

- Running only 2 codes together
- Varying on Broadwell E5-2697v4:
  - Clock frequency: nominal and turbo
  - Memory speed: 2133MHz / 2400MHz
  - Network: EDR / OPA
  - Node counts
- Comparing Broadwell to Skylake
  - Broadwell E5-2697v4 18 cores @ 2.3GHz
  - SkyLake 8160 24 cores @ 2.1 GHz
  - On 5 and 10 nodes with OPA

CUSTOMER MIX : 1 CodeA + 1 CodeB running together on same nodes

2*5/10/20 nodes, both codes spread over the 2 proc

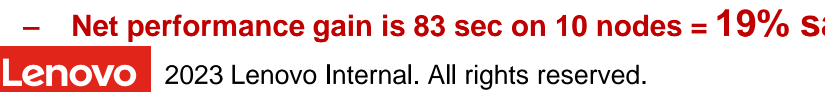
Best standalone perf on BDW@2.3GHz:
CodeA = 503 / 269 / 145 sec
CodeB = 386 / 198 / 106 sec

Lenox OPA nodes have 2133MHz memory dimms vs 2400MHz on EDR nodes

Configuration	Memory	Clock	Network	CPU	Total	CodeA	CodeB
5nodes*2: 240x120	2400	2300	edr	E5-2697v4	361	345	361
5nodes*2: 240x120	2400	turbo	edr	E5-2697v4	351	310	351
10nodes*2: 480x240	2400	2300	edr	E5-2697v4	194	176	194
10nodes*2: 480x240	2400	turbo	edr	E5-2697v4	186	157	185
20nodes*2: 960x480	2400	2300	edr	E5-2697v4	112	97	112
20nodes*2: 960x480	2400	turbo	edr	E5-2697v4	104	90	104
5nodes*2: 240x120	2133	2300	opa	E5-2697v4	375	342	375
5nodes*2: 240x120	2133	turbo	opa	E5-2697v4	371	309	370
10nodes*2: 480x240	2133	2300	opa	E5-2697v4	201	171	201
10nodes*2: 480x240	2133	turbo	opa	E5-2697v4	195	155	194

- Analysis:
  - OPA and EDR provides same perf
  - CodeA scales with clock frequency
  - CodeB scales with memory frequency
  - 2 jobs performance is limited by CodeB

- Conclusions on Broadwell:
  - Instead of 503 sec * 5 nodes + 386 sec * 5 nodes = **444 sec in average on 10 nodes in dedicated mode**
  - Mix approach leads to **361 sec on 10 nodes**
  - **Net performance gain is 83 sec on 10 nodes = 19% saved**



2023 Lenovo Internal. All rights reserved.

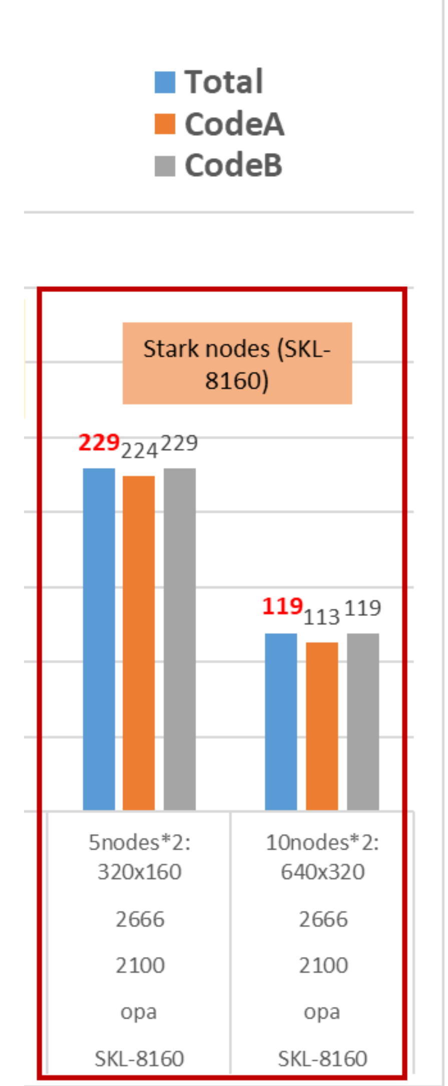
Smarter Throughput for Lenovo HPC Clusters

Customer case: 2 codes study on SkyLake

- Conclusion :
  - Similar behavior and benefits
    - ~1.2x for CodeA
    - ~1.35x for CodeB

- Instead of 311 sec * 5 nodes + 259 sec * 5 nodes = **285 sec in average on 10 nodes in dedicated mode**
- **Mix approach leads to 229 sec on 10 nodes**
- Net performance gain is 56 sec on 10 nodes = **17% saved**

RATIOS measured on SKL-8160		5	10	mix vs standalone	
CodeA	standalone	259	141		
CodeA	mix (32ppn)	224	113	1.16	1.25
CodeB	standalone	311	166		
CodeB	mix (16ppn)	229	119	1.36	1.39



Lenovo 2023 Lenovo Internal. All rights reserved.

Smarter Throughput for Lenovo HPC Clusters

Customer case: full system simulation

- In order to select best SKL processor for Customer cluster, we have
  - build performance projections for each case (code * #nodes * SKU)
  - simulated the full 600 jobs throughput on the several possible configurations based on 4 different SKUs, each SKU (and its price) impacting the global number of nodes possible in customer budget:
- Results:
  - **SKL 6132 14 cores @ 2.6GHz is the best processor to optimize throughput performance.**
  - Global throughput simulation time in standard mode (dedicated nodes to codes) is **22.1 hours**
  - Applying our throughput optimization method, global time is **18.04 hours** to do the same amount of work.

- This is ~19% computing time saved for the same workload.

				number of nodes used for each code						
				AVBP projections		MESONH projections				USING MIX
Total number of nodes in cluster	Processor type	#cores per socket	Freq GHz	5	10	5	10	20	Time for the whole workload (hours) in proposed cluster	Time for the whole workload (hours) in proposed cluster
102	SKL-6132	14	2.6	347	175	301	162	83	**22.10**	**18.04**
93	SKL-6140	18	2.3	308	156	303	163	84	**22.91**	**18.58**
88	SKL-6142	16	2.6	314	158	301	162	83	**24.28**	**19.71**
85	SKL-6148	20	2.4	270	137	302	162	84	**23.52**	**18.95**

Lenovo 2023 Lenovo Internal. All rights reserved.

References

- https://doku.lrz.de/education-and-training-10745708.html
  - Multiple good trainings from LRZ and its partners, PRACE, Intel, nVidia
- https://hpc.llnl.gov/documentation/tutorials
  - https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial
- https://www.nersc.gov/assets/pubs_presos/OLCF-Data-Intro-IO-Gerber-FINAL.pdf
- https://www.rc.fas.harvard.edu/wp-content/uploads/2016/03/Intro_Parallel_Computing.pdf
- https://prace-ri.eu/wp-content/uploads/hpc-centre-electricity-whitepaper-2.pdf
- https://lwn.net/Articles/250967/
- https://events.prace-ri.eu/event/1353/attachments/2100/4272/2022%20-%20PRACE%20EE%20kurz%20-%20theory.pdf

Lenovo 2023 Lenovo Internal. All rights reserved.

Smarter
technology
for all

Lenovo

thanks.