Foundations of High Performance Computing

Lecture 12: Benchmarking HPC systems

2021-2022 Stefano Cozzini

"Foundation of HPC" course

DATA SCIENCE & SCIENTIFIC COMPUTING

Aims of this lecture

- Give you the feeling how much is important to know how your system/ application/computational experiment is performing..
- Name a few standard benchmarks that can help you in making/taking a decision
- Discuss in some details some of them:
  - HPL/ HPCG/ STREAM/Graph500
- Show you some tricks and tips how to make your own benchmarking procedure

Benchmarking: a definition

a benchmark is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it

[from Wikipedia]

A few important notes:

- Benchmarking is essential
- No single number can reflect overall performance
- You needs several representative benchmarks
- the only benchmark that matters is the intended workload.
- The purpose of benchmarking is not to get the best results, but to get consistent repeatable accurate results that are also the best results.

Benchmarking is an art..

- Should be done by people who know the application, the hardware, and the operating system

- Be careful with artificial benchmarks or marketing myths

Do not be fooled…

- Measuring and reporting performance is the basis for scientific advancement in HPC.
- Not always scientific papers/reports guarantee reproducibility
- A lack of standards/rule is actually present in benchmarking arena.

Interpreting benchmark numbers

- EXTRAORDINARILY DIFFICULT
- Vendors tend to tune their products specifically for industry-standard benchmarks. Use extreme caution in interpreting their results.
- Many benchmarks focus entirely on the speed of computational performance, neglecting other important features of a computer system.
- Benchmarks seldom measure real world performance of mixed workloads

What we need to benchmark on HPC systems?

- Single core performance:
  - only a single processor (core) is performing computations.
- Single node performance:
  - each node in the entire system is performing computations but they do no communicate with each other explicitly.
- Global performance:
  - all processors/nodes in the system are performing computations and they explicitly communicate with each other.

Which kind of codes for benchmarking?

- Synthetic codes
  - Basic hardware and system performance tests
  - Meant to determine expected future performance and serve as surrogate for workload not represented by application codes
  - useful for performance modeling
- Application codes
  - Actual application codes as determined by requirements and usage
  - Meant to indicate current performance
  - Each application code should have more than one real test case

A good benchmark is…

- Relevant and meaningful to the target application domain
- Applicable (portable) to a broad spectrum of hardware architecture
- Adopted both by users and by vendors to enable comparative evaluation

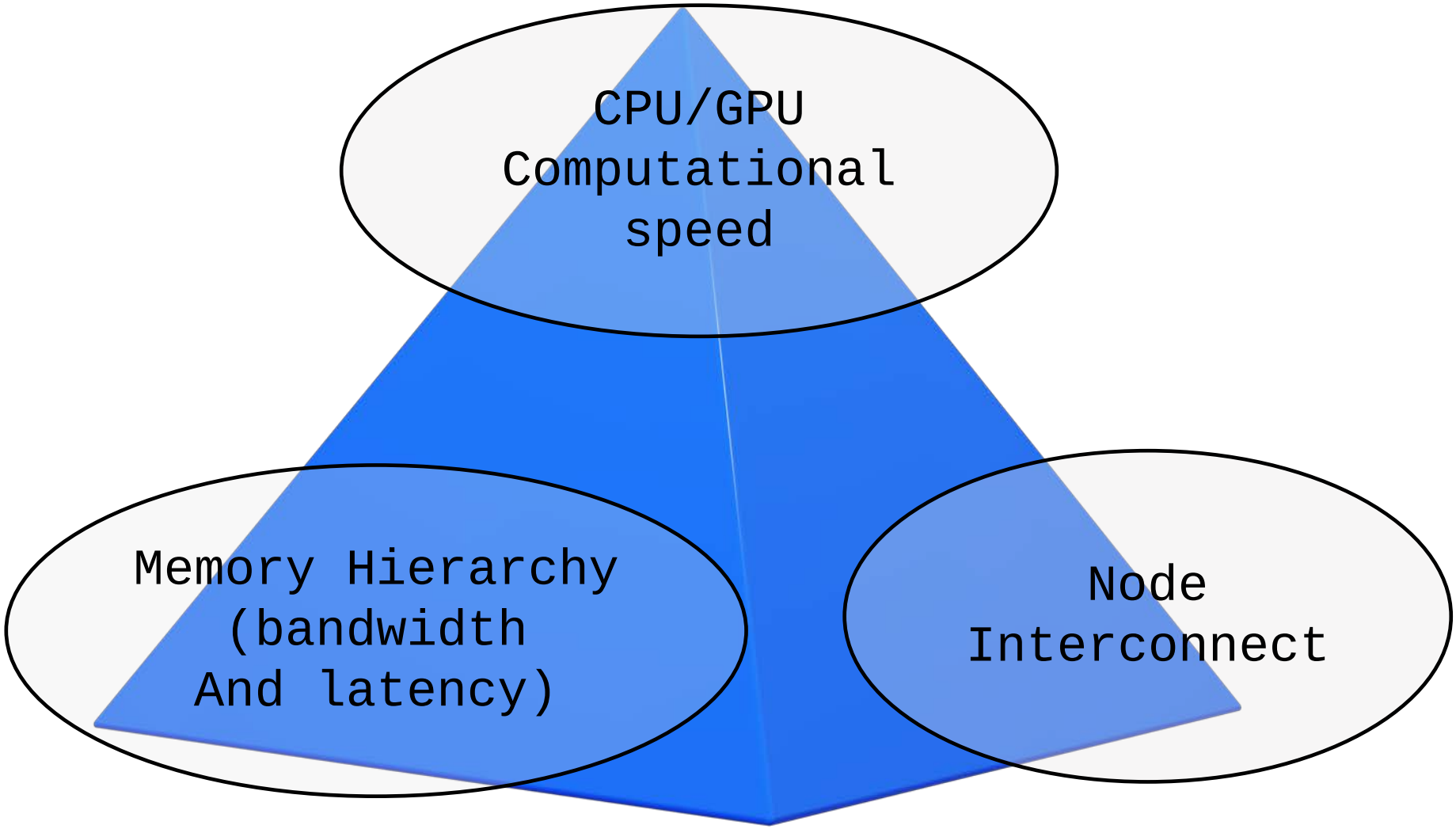
Some freely available benchmark (1)

- HPL Linpack (for Top500)
- HPC Challenge Benchmark:
  - a collection of basic benchmark beyond HPL
- NAS benchmark suite
  - math kernel implemented both in MPI and openMP
  - (http://www.nas.nasa.gov/publications/npb.html)
- HPCG
  - A recently introduced benchmark to “fix” the HPL one
- Stream
  - Memory benchmark
- Graph500
  - Data intensive application (used for Graph500 list)

Some freely available benchmark (2)

- Network benchmark:
  - INTEL MPI benchmark
    - https://software.intel.com/en-us/articles/intel-mpi-benchmarks
  - OSU benchmarks:
    - http://mvapich.cse.ohio-state.edu/benchmarks/
- I/O benchmarks:
  - Iozone
  - IOR

Resource to benchmark



Scientific application as benchmark..

- HPC benchmarks so far presented are not able to be too much representative of actual workload in specific scientific domain
- However running a complex application as standard benchmark could be difficult and sometime almost impossible.
- There are also the so called Mini-applications: smaller version of the real production scientific package
- They provide time to solution for several kernels and strong/weak scalability information

Some examples:

- CoMD:
  - A simple proxy for the computations in a typical molecular dynamics application. The reference implementation mimics that of SPaSM. In addition, we provide an OpenCL implementation which allows testing on multicore and GPU architectures, with both array-of-structures and structure-of-arrays data layouts.
  - https://github.com/exmatex/CoMD
- MiniFE
  - MiniFE is an proxy application for unstructured implicit finite element codes. MiniFE is intended to be the "best approximation to an unstructured implicit finite element or finite volume application, but in 8000 lines or fewer."
- MiniMD:
  - A simple proxy for the force computations in a typical molecular dynamics applications. The algorithms and implementation used closely mimics these same operations as performed in LAMMPS.
- MiniGhost:
  - A Finite Difference proxy application which implements a difference stencil across a homogenous three-dimensional domain.

Let us play with benchmarks !

- Graph500
- STREAM
- HPL
- HPCG

Graph 500

- from Graph500.org: *This is the first serious approach to augment the Top 500 with data-intensive applications.*
- Available from 2010
- It builds a graph and then it transverse it implementing a so-called Breadth-First Search (BFS) algorithms
- Another algorithm is also implemented: Single Source Shortest Path (SSSP)
- Separated lists are constructed for the two algorithms from 2017 on..
- Easy to compile and run: it requires just one parameter, i.e number of vertices
- Code available on github:
  - graph500/graph500: Graph500 reference implementations (github.com)

Graph500: class sizes

Table 4.6 Problem Size Classes, Number of Vertices, and Memory Requirements for Graph500 Search Benchmark

Level	Scale	Size	Vertices (Billions)	Terabytes
10	26	Toy	0.1	0.02
11	29	Mini	0.5	0.14
12	32	Small	4.3	1.1
13	36	Medium	68.7	17.6
14	39	Large	549.8	141
15	42	Huge	4398.0	1126

Graph in action..

- graph500 dir on the repo

```
…
Running BFS 63
Time for BFS 63 is 5.458120
TEPS for BFS 63 is 1.96721e+08
Validating BFS 63
Validate time for BFS 63 is 19.377711
SCALE:                           26
edgefactor:                      16
NBFS:                            64
graph_generation:                40.991
num_mpi_processes:               16
construction_time:               23.9508
bfs  min_time:                   5.40125
..
bfs  max_time:                   5.53835
bfs  mean_time:                  5.46142
bfs  stddev_time:                0.0277278
bfs  min_TEPS:                   1.93871e+08
bfs  median_TEPS:                1.96633e+08
..
```

Stream benchmark

- Created in 1991
- Intended to be an oversimplified representation of low-computing intensity and long vector operations
- Realistic baseline for memory bandwidth on HPC system
- Widely used, more 1100 results in the database
- Hosted at www.cs.virginia.edu/stream
- Compilation and execution trivial
- Warning: reported and actual BW numbers may differ..

Stream

- Four kernels, separately timed:

TYPE	OPERATION	BYTE/cycle	FLOPS/cycle
COPY	A(:) = C(:)	16	0
SCALE	A(:) = s * C(:)	16	1
ADD	A(:) = B(:) + C(:)	24	1
TRIAD	A(:) = B(:) + s * C(:)	24	2

- N chosen to make each array >> cache size
- Repeated several times, first iteration ignored
- Min/Max/Avg reported and the best time used to compute Bandwidth

Stream: which kind of performance can we expect ?

- Let us take a look how memory is organized on modern CPU

Memory layout on skylakeX processors

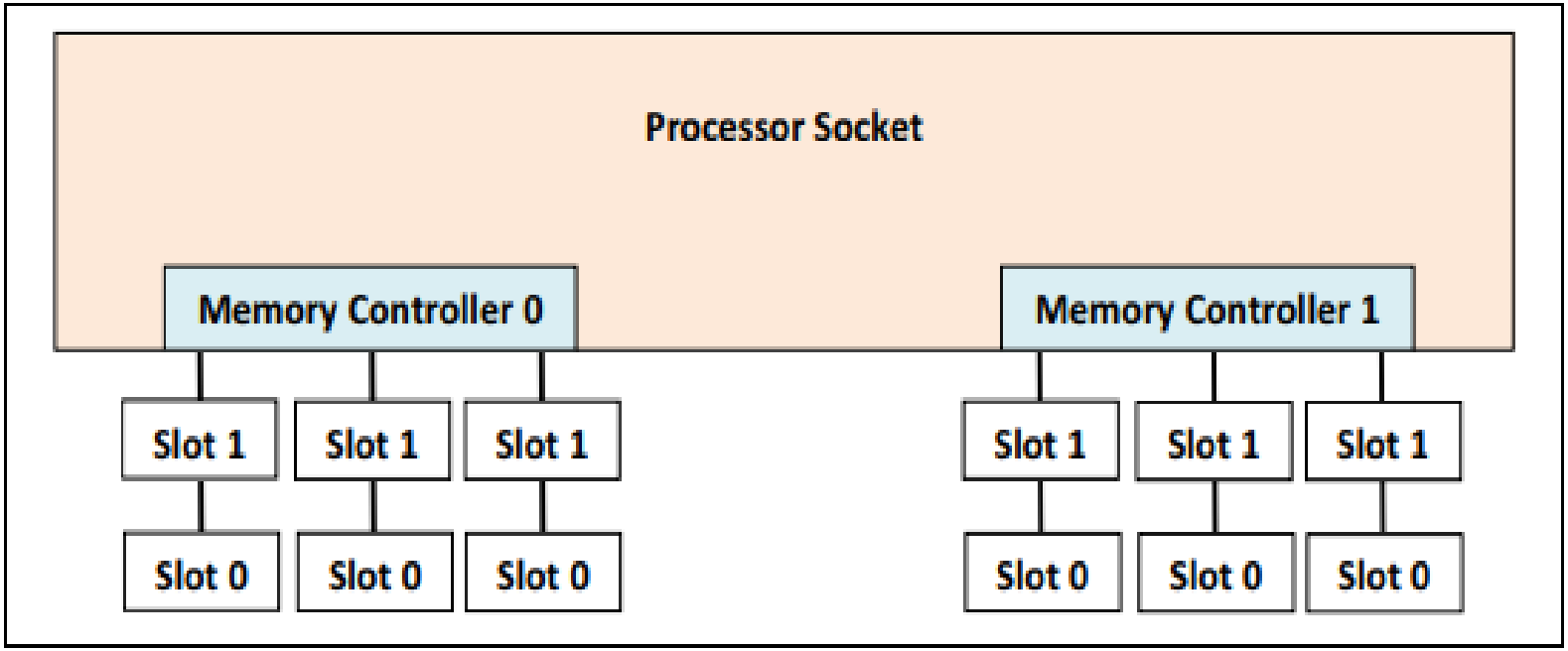
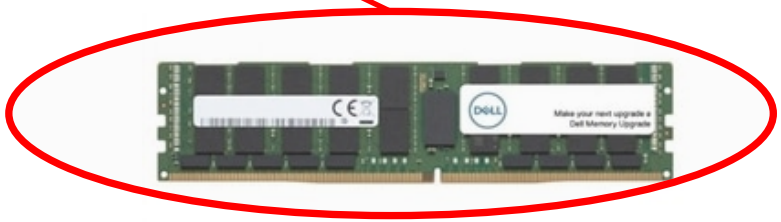


Figure 1 Scalable Family processor with two memory controllers, six memory channels and twelve memory DIMM slots

Picture from : Intel Xeon Scalable Family Balanced Memory Configurations (lenovopress.com)

Memory should be balanced !

- Three rules:
  - All populated memory channels should have the same total memory capacity
  - All memory controllers on a processor socket should have the same configuration of DIMMs
  - All processor sockets on the same physical server should have the same configuration of DIMMs



ORFEO: RAM

NODE TYPE	processor	# Channels	Max speed	Memory types
THIN	Xeon(R) Gold 6126	6	2666MT/sec	DDR4-2666
FAT	Xeon(R) Gold 6154	6	2666MT/sec	DDR4-2666
GPU	Xeon(R) Gold 6226	4	2933MT/sec	DDR4-2933

$$b_{peak} = \#\,Channels \times f_{MEM} \times 8B/cycle$$

see https://www.intel.com/content/www/us/en/products/processors/xeon/scalable/gold-processors.html

Memory layout on ORFEO thin node

```
RAM=768GB 12 LRDIMM of 32GB each
6 on each single processor
```

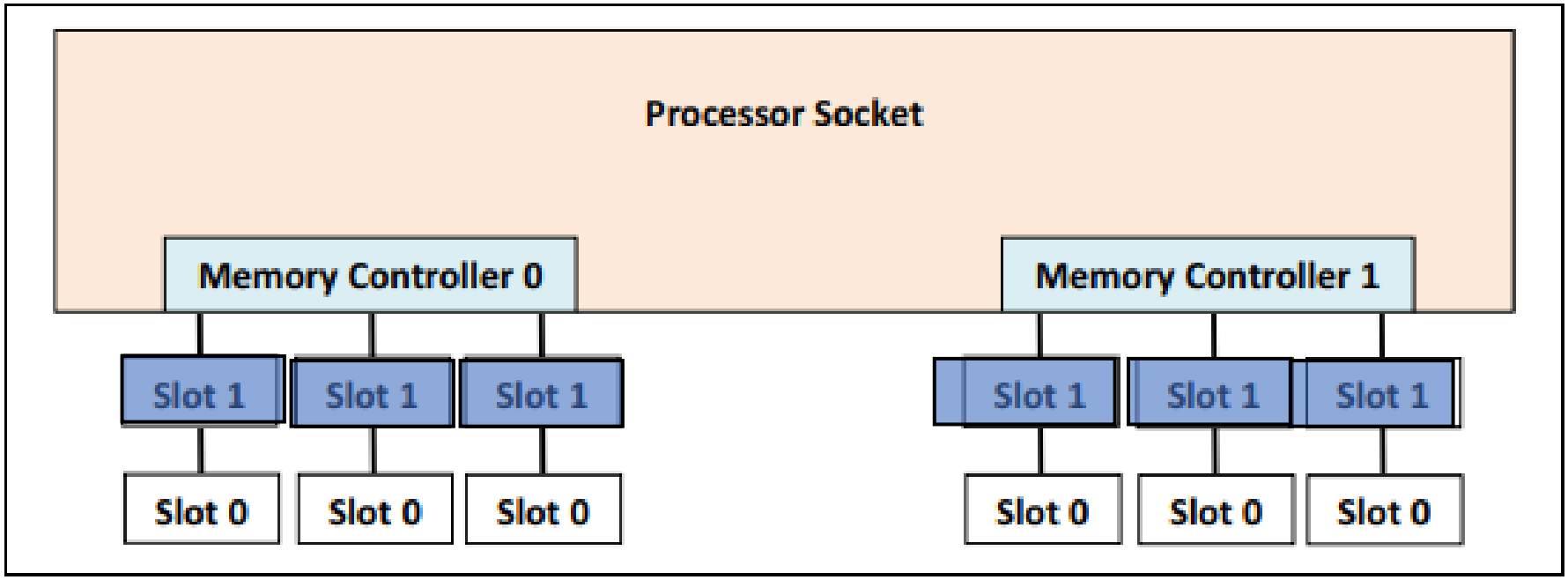


Figure 1 Scalable Family processor with two memory controllers, six memory channels and twelve memory DIMM slots

Memory layout on ORFEO fat node

```
RAM=768GB 12 LRDIMM of 64GB each
6 on each single processor
```

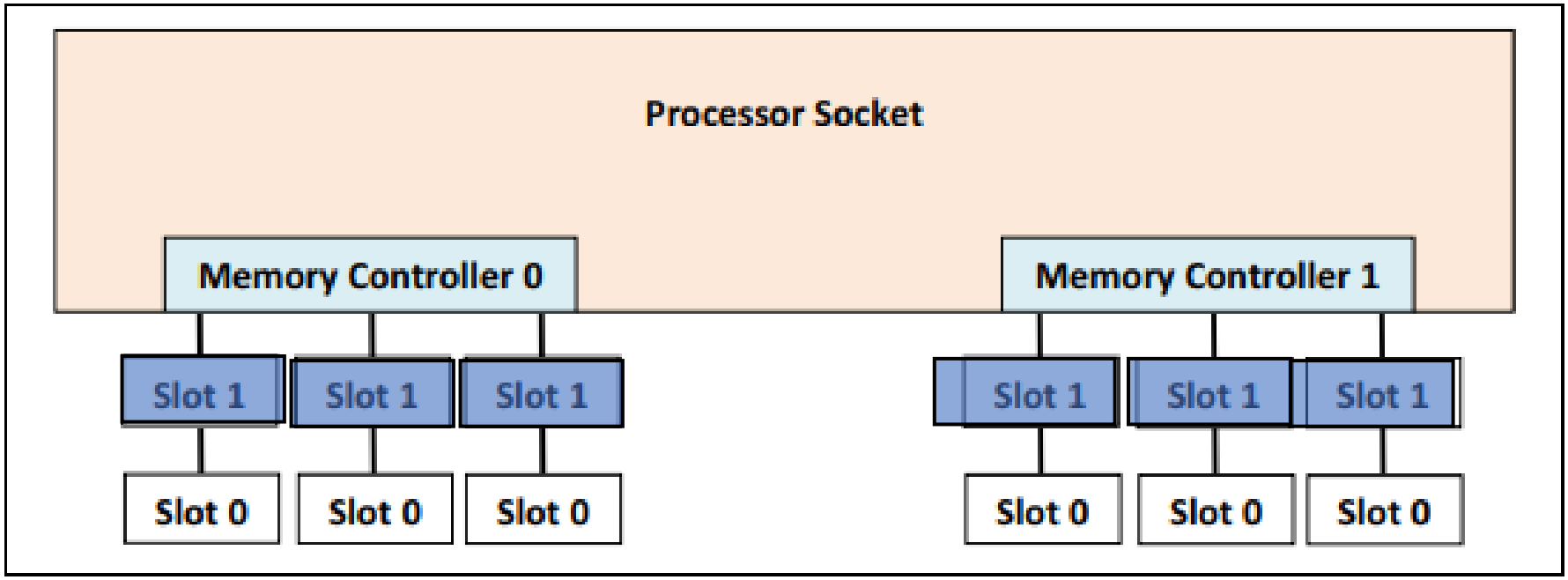


Figure 1 Scalable Family processor with two memory controllers, six memory channels and twelve memory DIMM slots

Memory layout on ORFEO gpu node

```
RAM=512 GB 8 LRDIMM of 32 GB each
6 on each single processor
```

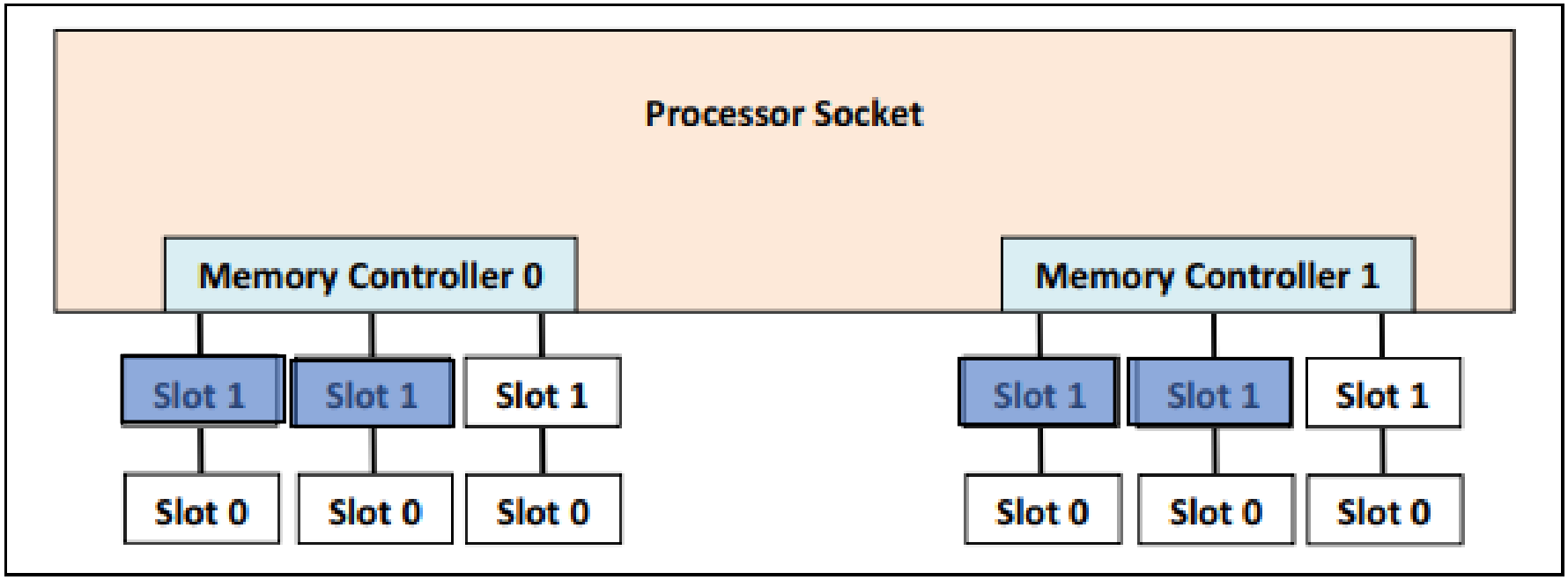


Figure 1 Scalable Family processor with two memory controllers, six memory channels and twelve memory DIMM slots

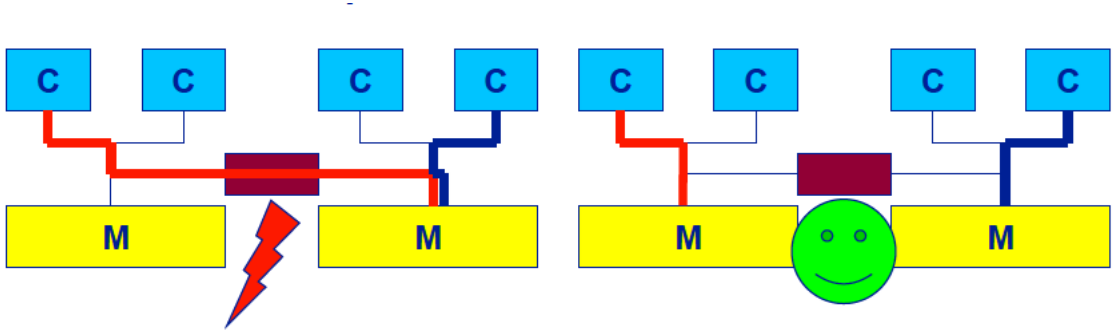
ORFEO: peak performance

NODE TYPE	processor	# Channels	# socket	Memory types
THIN	Xeon(R) Gold 6126	6	2	6x2666x8x2 =~ 256GB/sec
FAT	Xeon(R) Gold 6154	6	2	6x2666x8x2 =~ 256GB/sec
GPU	Xeon(R) Gold 6226	4	2	4x2933x8x2 =~ 188GB/sec

ccNUMA performance problem

- CcNUma:
  - Whole memory is transparently accessible
  - But physically distributed with different latency and bandwidth

    => potential contention
- How to we make sure that memory access is always as “local” and “distribute” as possible ?



Thread Affinity and Data Locality

- Affinity
  - Process Affinity: bind processes (MPI tasks, etc.) to CPUs
  - Thread Affinity: further binding threads to CPUs that are allocated to their parent process
- Data Locality
  - Memory Locality: allocate memory as close as possible to the core on which the task that requested the memory is running
  - Cache Locality: use data in cache as much as possible
- Correct process, thread and memory affinity is the basis for getting optimal performance.

Thread placement - numactl

- numactl is a command of the operating system providing much focused on the NUMA features of a system.
- numactl understands which processors form a NUMA node and how threads need to be grouped together
- Important options:

  --membind <n>: place pages on NUMA node <n>

  --cpunodebind <n>: pin threads to node <n>

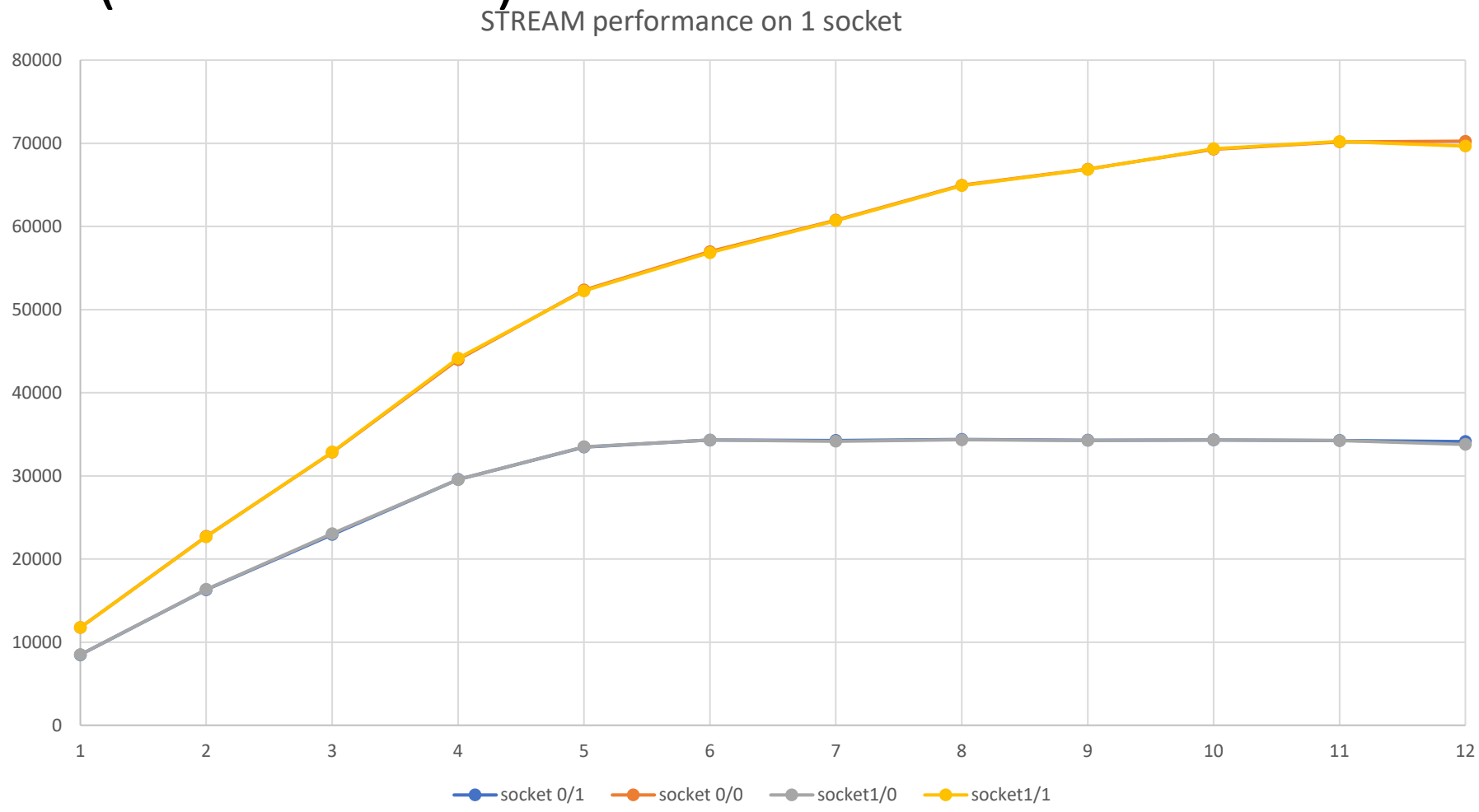
  --interleave <nodes>: put the pages round-robin on <nodes>

Example:

```
numactl --cpunodebind=0 --membind=0,1 ./a.out
```

This puts memory on nodes 0 an 1, but threads only on node 0.

STREAM performance on 1 socket (thin node)



Stream in action:

- Naïve way:

```
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           86972.1     0.017072     0.016189     0.020788
Scale:          87095.2     0.016653     0.016166     0.019954
Add:            99201.2     0.022018     0.021290     0.027231
Triad:          98645.5     0.022046     0.021410     0.026984
```

- Pinning processor: likwid-pin –c N:0-23 ./stream.x

```
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:          160364.4     0.008870     0.008780     0.009004
Scale:         159313.2     0.009750     0.008838     0.015147
Add:           166651.7     0.013455     0.012673     0.019113
Triad:         166639.1     0.012754     0.012674     0.012905
-------------------------------------------------------------
```

More on stream on the tutorial

In this tutorial we play with the STREAM benchmark to measure memory bandwidth on the HPC multicore nodes

THE BENCHMARK: run HPL on ORFEO nodes..

- Check the README.md on github account

A few notes:

- Standard input file should be present
- Beware of threads !

What about N ?

- N should be large enough to take ~75% of RAM..
- N = sqrt (0.75 * Number of Nodes * Minimum memory of any node / 8)
- You can try to compute it from here:

http://www.advancedclustering.com/act-kb/tune-hpl-dat-file/

Parameter for input file:

N	Problem size	**Pmap**	Process mapping
NB	Blocking factor	**threshold**	for matrix validity test
P	Rows in process grid	**Ndiv**	Panels in recursion
Q	Columns in process grid	**Nbmin**	Recursion stopping criteria
Depth	Lookahead depth	**Swap**	Swap algorithm
Bcasts	Panel broadcasting method	**L1, U**	to store triangle of panel
Pfacts	Panel factorization method	**Align**	Memory alignment
Rfacts	Recursive factorization method	**Equilibration**	

Tips to get performance:

- Figure out a good block size (NB) for the matrix multiply routine. The best method is to try a few out. If you happen to know the block size used by the matrix-matrix multiply routine, a small multiple of that block size will do fine. This particular topic is discussed in the FAQs section.
- The process mapping should not matter if the nodes of your platform are single processor computers. If these nodes are multi-processors, a row-major mapping is recommended.
- HPL likes "square" or slightly flat process grids. Unless you are using a very small process grid, stay away from the 1-by-Q and P-by-1 process grids.

What does this means this ?

- For a THIN node: 24 cores → 6x4 4x6
- For a GPU node: 48 cores → 8x6 6x8
- For a FAT node: 36 cores → 6x6

High Performance Conjugate Gradient (HPCG).

- High Performance Conjugate Gradient (HPCG).
  - Solves Ax=b, A large, sparse, b known, x computed.
  - An optimized implementation of PCG contains essential computational and communication patterns that are prevalent in a variety of methods for discretization and numerical solution of PDEs
- Patterns:
  - Dense and sparse computations.
  - Dense and sparse collective.
  - Data-driven parallelism (unstructured sparse triangular solves).

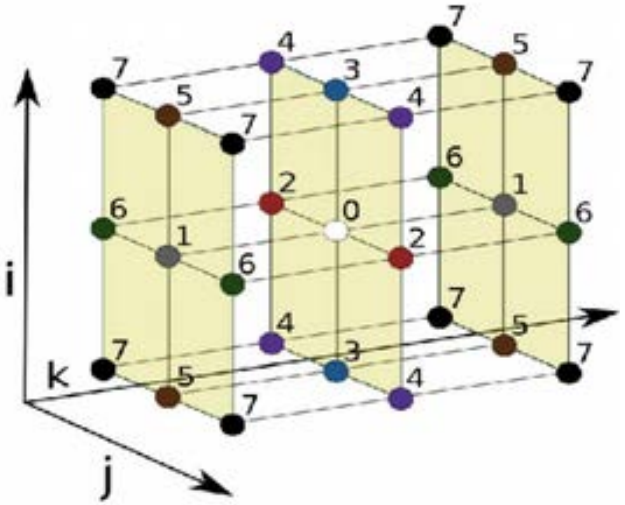
http://www.hpcg-benchmark.org/index.html

HPCG benchmark: why ?

- HPL has problems..
  - The gap between HPL predictions and real application performance will increase in the future.
  - A computer system with the potential to run HPL at an Exaflop is a design that may be very unattractive for real applications.
  - Future architectures targeted toward good HPL performance will not be a good match for most applications.
  - This leads us to a think about a different metric

Model problem description

- Synthetic discretized 3D PDE (FEM, FVM, FDM).
- Local domain: n_x n_y n_z (see hpcg.dat)
- Process layout: $p_x * p_y * p_z$ ($p_x * p_y * p_z$ =N of processor)
- Global domain: (p_x * n_x) X (p_y* n) X (n * n)
- Sparse matrix:
  - 27 nonzeros/row interior.
  - 8 – 18 on boundary.
  - Symmetric positive definite



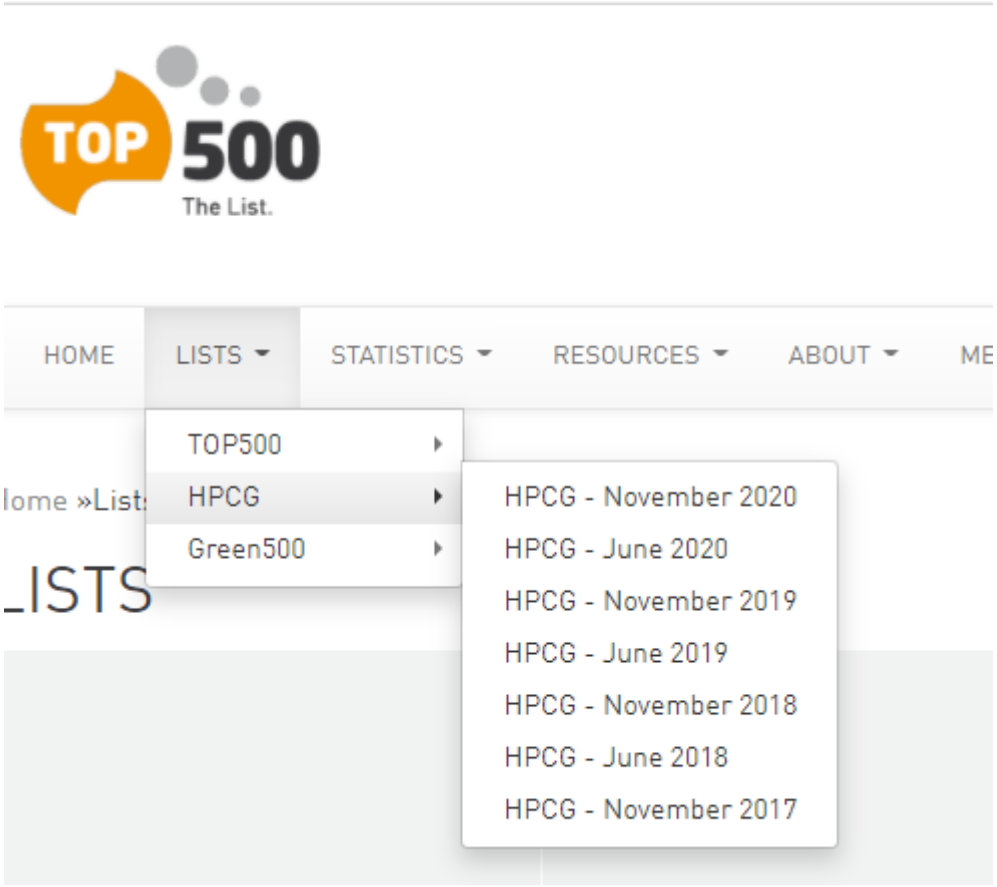
27-point stencil operator

HPCK benchmark: computational characteristic..

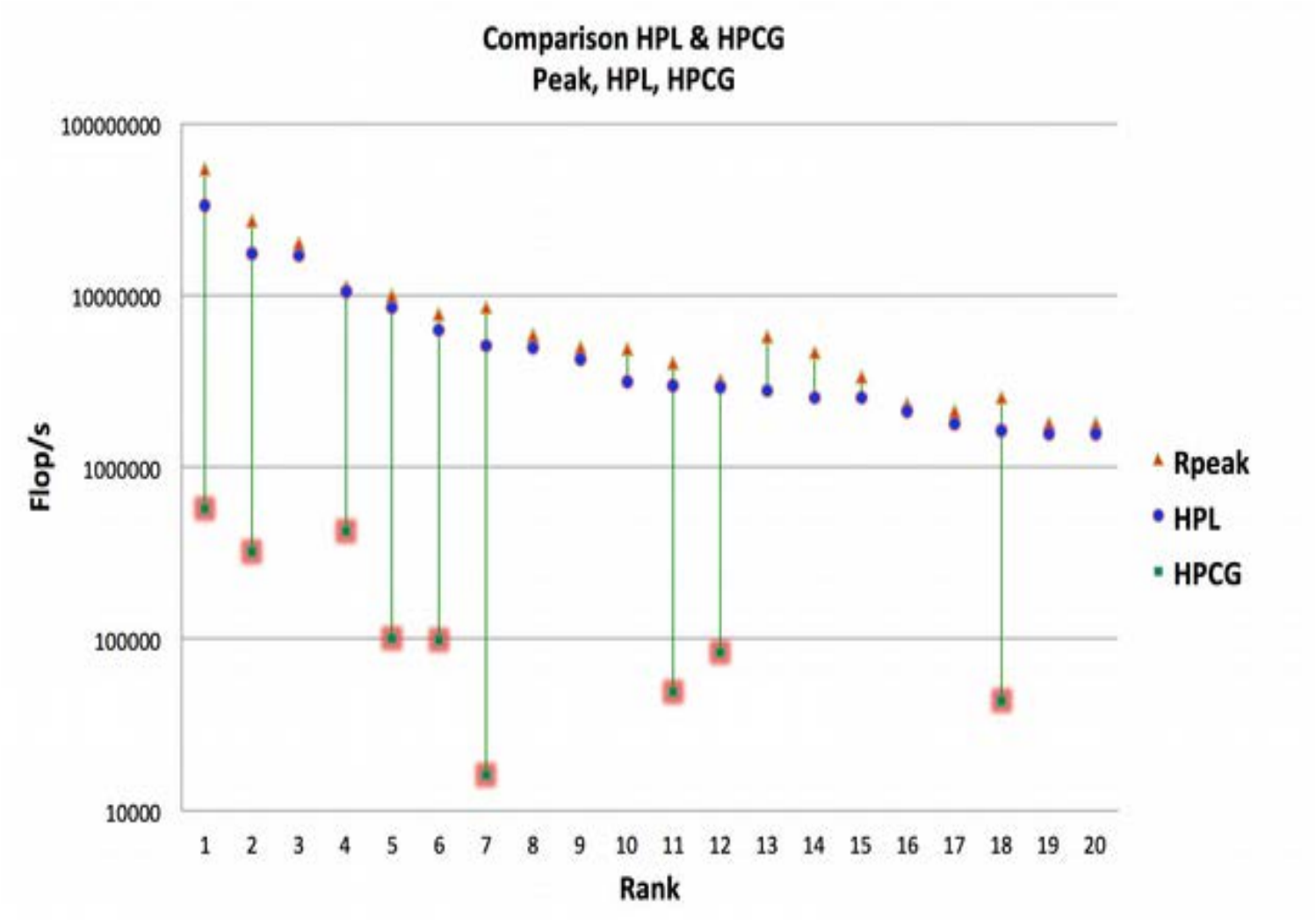
- Balanced BW and compute:
- HPCG is memory BW bound in modern processors
  - 6 Byte/FLOP
  - HPCG can utilize at most x/6 of peak FLOP
- Scalable collectives: HPCG uses all-reduce
- Efficient parallelization of Gauss-Seidel:
  - HPCG spends 2/3 of time in GS

HPCG benchmark and top 500

- From November 2017 all the Top500 machine are measured against HCPG as well…



HPCG benchmark and top 500

- 

Running HPCG…

- See README.MD file in HPCG directory

Tips for benchmarking..

- use /usr/bin/time and take note of all times wall time/ user time /sys time
- repeat the same run at least a few time to estimate the fluctuations of the numbers (this should be generally within a few percent)
- be sure to be alone on the system you are using and with no major perturbation on your cluster
- execution runs should be at least in the order of tens of minutes
- always check the correctness of your scientific output

Performance evaluation process

- Monitoring your System:
  - Use monitoring tools to better understand your machine's limits and usage
- Is the system limit well suited to run my application ?
  - Observe both overall system performance and single-program execution characteristics.
- Monitoring your own code
  - Is the system doing well ?
  - Is my program running in a pathological situation ?