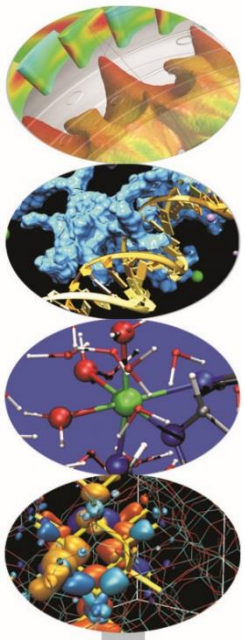
CINECA SCAI
SuperComputing Applications and Innovation

I/O: State of the art and Future developments



Giorgio Amati
SCAI Dept.

Rome, 14/15 May 2015

Cineca
COURSES
High Performance
Computing 2015

CINECA

Some questions

- Just to know each other:
  - ✓ Why are you here?
  - ✓ Which is the typical I/O size you work with?
    - GB?
    - TB?
  - ✓ Is your code parallelized?
  - ✓ How many cores are you intended to use?
  - ✓ Are you working in a small group or you need to exchange data with other researchers?
  - ✓ Which language do you use?

"Golden" rules about I/O

- Reduce I/O as much as possible: only relevant data must be stored on disks
- Save data in binary/unformatted form:
  - ✓ asks for less space comparing with ASCI/formatted ones
  - ✓ It is faster (less OS interaction)
- Save only what is necessary to save for restart or checkpointing, everything else, unless for debugging reason or quality check, should be computed on the fly.
- Dump all the quantities you need once, instead of using multiple I/O calls: if necessary use a buffer array to store all the quantities and the save the buffer using only a few I/O calls.
- Why?

What is I/O?

1. DATA
2. **`fwritef, fscanf, fopen, fclose, WRITE, READ, OPEN, CLOSE`**
3. Call to an external library: MPI I/O, HDF5, NetCDF, ecc…
4. Scalar/parallel/network Filesystems
    1. I/O nodes and Filesystem cache
    2. I/O network (IB, SCSI, Fibre, ecc..)
    3. I/O RAID controllers and Appliance (Lustre, GPFS)
    4. Disk cache
    5. FLASH/Disk (one or more Tier)
5. Tape

Latencies

- I/O operations involves
  - ✓ OS & libraries
  - ✓ IO devices (e.g. RAID controllers)
  - ✓ Disks
- I/O latencies of disks are of the order of microseconds
- RAM latencies of the order of 100-1000 nanoseconds
- FP unit latencies are of the order of 1-10 nanoseconds
- → I/O very slow compared to RAM of FP unit

Architectural trends

Peak Performance		Moore law
FPU Performance		Dennard law
Number of FPUs		Moore + Dennard
App. Parallelism		Amdahl's law

Architectural trends

		2020 estimates
Number of cores	↑	10^9
Memory x core	↓	100Mbyte or less
Memory BW/core	↓	500GByte/sec
Memory hierachy	↑	Reg, L1, L2, L3, …

Architectural trends

		2020 estimates
Wire BW/core		1GByte/sec
Network links/node		100
Disk perf		100Mbyte/sec
Number of disks		100K

What is parallel I/O?

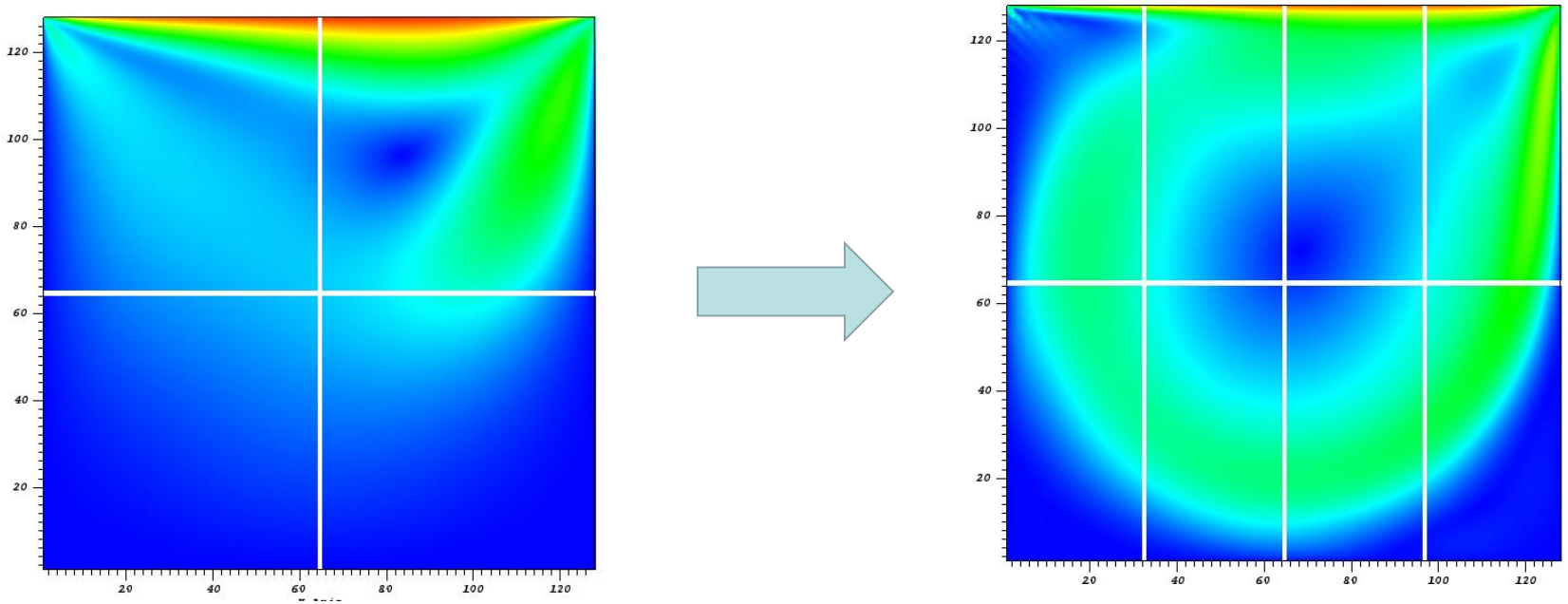
- A more correct definition in tomorrow
- Serial I/O
  - ✓ 1 task writes all the data
- Parallel I/O
  - ✓ All task write its own data in a different file
  - ✓ All task write its own data in a single file

- MPI/IO, HDF5, NetCDF, CGNS,……

Why parallel I/O?

- New Architectures: many many core (up to 10^9)
- As the number of task/threads increases I/O overhead start to affect performance
- I/O (serial) will be a serious bottleneck
- Parallel I/O is mandatory else no gain in using many many core
- Other issues:
  - ✓ Domain decomposition
  - ✓ ASCII vs binary
  - ✓ Endianess
  - ✓ blocksize
  - ✓ Data management

I/O: Domain Decomposition



- I want to restart a simulation using a different number of tasks: three solutions
  - ✓ pre/postprocessing (merging & new decomposition)
  - ✓ Serial dump/restore
  - ✓ Parallel I/O

I/O: ascii vs. binary/1

- ASCII is more demanding respect binary in term of disk occupation
- Numbers are stored in bit (single precision floating point number → 32 bit)

- 1 single precision on disk (binary) → 32 bit
- 1 single precision on disk (ASCII) → 80 bit
  - 10 or more **`char`** (**`1.23456e78`**)
  - Each char asks for 8 bit

✓ Not including spaces, signs, return, …
✓ Moreover there are rounding errors, …

I/O: ascii vs. binary/2

- Some figures from a real world application
- openFOAM
- Test case: 3D Lid Cavity, 200^3, 10 dump

- Formatted output (ascii)
  - ✓ Total occupation: 11 GB
- Unformatted output (binary)
  - ✓ Total occupation: 6.1 GB

- A factor 2 in disk occupation!!!!

I/O: endianess

- IEEE standard set rules for floating point operations
- But set no rule for data storage
- Single precision FP: 4 bytes (**B0**,B1,B2,B3)
  - ✓ Big endian (IBM): **B0** B1 B2 B3
  - ✓ Little endian (INTEL): B3 B2 B1 **B0**
- Solutions:
  - ✓ Hand made conversion
  - ✓ Compiler flags (intel, pgi)
  - ✓ I/O libraries (HDF5)

I/O: blocksize

- The blocksize is the basic (atomic) storage size
- One file of 100 bit will occupy 1 blocksize, that could be > 4MB

```
ls -lh TEST_10K/test_1
-rw-r--r-- 1 gamati01  10K 28 gen 11.22 TEST_10K/test_1
…
du -sh TEST_10K/test_1
512K        TEST_10K/test_1
…
du -sh TEST_10K/
501M        TEST_10K/
…
```

- Using **`tar`** commando to save space

```
ls -lh test.tar
-rw-r--r-- 1 gamati01 11M  5 mag 13.36 test.tar
```

I/O: managing data

- TB of different data sets
- Hundreds of different test cases
- Metadata
- Share data among different researchers
  - ✓ different tools (e.g. visualization tools)
  - ✓ different OS (or dialect)
  - ✓ different analysis/post processing

- You need a common "language"
  - ✓ Use I/O libraries
  - ✓ Invent your own data format

Some figures

Simple CFD program, just to give you an idea of performance loss due to I/O.

- 2D Driven Cavity simulation
- 2048*2048, Double precision (about 280 MB), 1000 timestep
- Serial I/O = 1.5”
  - ✓ 1% of total serial time
  - ✓ 16% of total time using 32 Tasks (2 nodes) → 1 dump = 160 timestep
- Parallel I/O = 0.3” (using MPI I/O)
  - ✓ 3% of total time using 32 Tasks (2 Nodes) → 1 dump = 30 timestep
- An what using 256 tasks?

A Strategy for 2020

- ✓ Understand architectural trends (at all level)
- ✓ Evaluate impact on application I/O design
- ✓ Plan application refactoring, new I/O algorithms
- ✓ Field test on current available machine (anticipating some arch trends), proof of concept.
- ✓ Bring stuff into the main trunk for production.

some strategies

I/O is the bottleneck → avoid I/O when possible
I/O subsystem work with locks → simplify application I/O
I/O has its own parallelism → use MPI-I/O
Raw data are not portable → use library
I/O is slow → compress reduce output data
I/O C/Fortran APIs are synchronous → use dedicated I/O tasks

Application DATA are too large → analyze it "on the fly", re-compute vs. write

At the end: moving data

- Now I have hundreds of TB. What I can do?
  - Storage using Tier-0 Machine is limited in time (e.g. Prace Project data can be stored for 3 Month)
  - Data analysis can be time consuming (eyen years)
  - I don’t want to delete data
  - I have enough data storage somewhere else?

How can I move data?

moving data: some figures

- Moving outside CINECA
  - ✓ **`scp`** → 10 MB/s
  - ✓ **`rsync`** → 10 MB/s
- I must move 50TB of data:
  - ✓ Using **`scp`** or **`rsync`** → 60 days
- No way!!!!!

- Bandwidth depends on network you use. Could be better, but in general is even worse!!!

moving data: some figure

- Moving outside CINECA
  - **`gridftp`** → 100 MB/s
  - **`globusonline`** → 100 MB/s
- I must move 50TB of data:
  - Using **`gridftp/globusonline`** → 6 days
- Could be a solution…
- We get these figures between CINECA and a remote cluster using a 1Gb Network

moving data: some hints

- Size matters: moving many little files cost more then moving few big files, even if the total storage is the same!
- Moving file from Fermi to a remote cluster via Globusonline

Size	Num. Of files	Mb/s
10 GB	10	227
100 MB	1000	216
1 MB	100000	61

✓ You can loose a factor 4, now you need 25 days instead of 6 to move 50TB!!!!!

moving data: some hints

- ✓ Plan your data-production carefully
- ✓ Plan your data-production carefully (again!)
- ✓ Plan your data-production carefully (again!)
- ✓ Clean your dataset from all unnecessary stuff
- ✓ Compress all your ASCII files
- ✓ Use **`tar`** to pack as much data as possible
- ✓ Organize your directory structure carefully
- ✓ Syncronize with **`rsync`** in a systematic way
- ✓ One example:
  - We had a user who wants to move 20TB distributed over more then 2'000'000 files…
  - **`rsync`** asks many hours (about 6) only to build the file list, without any synchronization at all