Some “User Remarks”
NUMEXAS PROJECT
– Kratos Subproject -

Dr. Riccardo Rossi

Dr. Pooyan Dadvand

Dr. Cecilia Soriano

Engineering Problems on EXASCALE architectures

Very complex infrastructure needed for

- Pre/Post Processing
- Doing HPC calcuations
- “**steering the simulation**”

Actual calculations might even not be the most time intensive part!!

- Huge code base long development times must be futureproof but also very standard and not vendor-locked, example OpenMP, MPI, OpenCL, OpenACC …
- Portability is a MUST, **even to windows**!!
- Need to leverage **OO techniques** to manage software complexityC++ Industrial grade compiler needed. Currently GCC, Clang, Intel, MSVC
- **Use a scripting language as “main”** to allow flexibility

Application bottlenecks

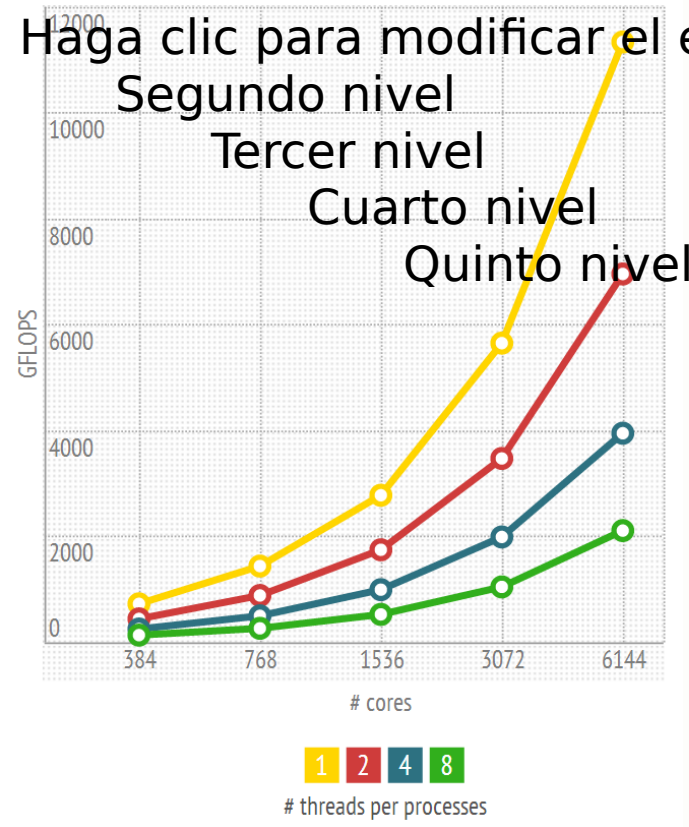
Applications tend to be severely **Bandwidth Limited**

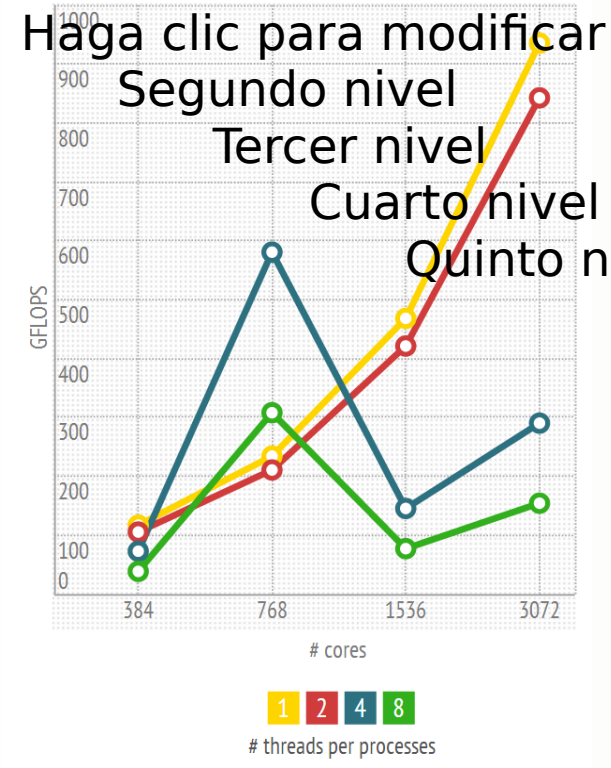
- **Linpack** performance does not faithfully predict application performance
- **Stream** or **HPCG** benchmarks much more relevant Please consider optimizing for this instead for Linpack, that would be much more significant for engineering applications (end hence industry) than raw FLOPs
- **Lots of parallelism available, although quite irregular** it is possible to run on GPUs, however the type of calculations is non-optimal (so far **my personal experience was quite deceiving**)

Epetra (right) and HPCG (left) benchmarks

- *sparse matrix times 8-column multivector*
  - *matrix size= (500,500)*
- *results averaged between different grids of processors*

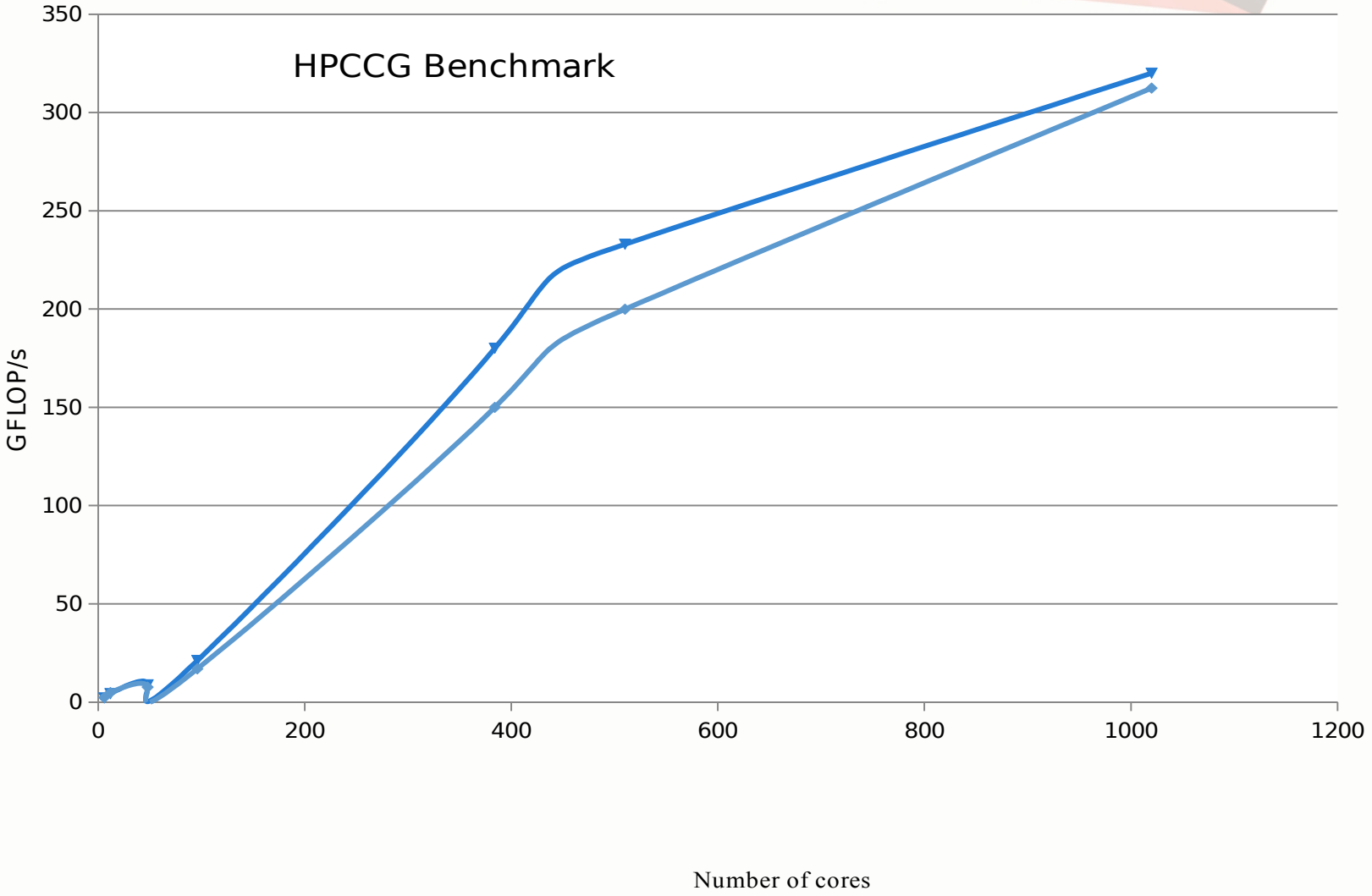
- *HPCG*



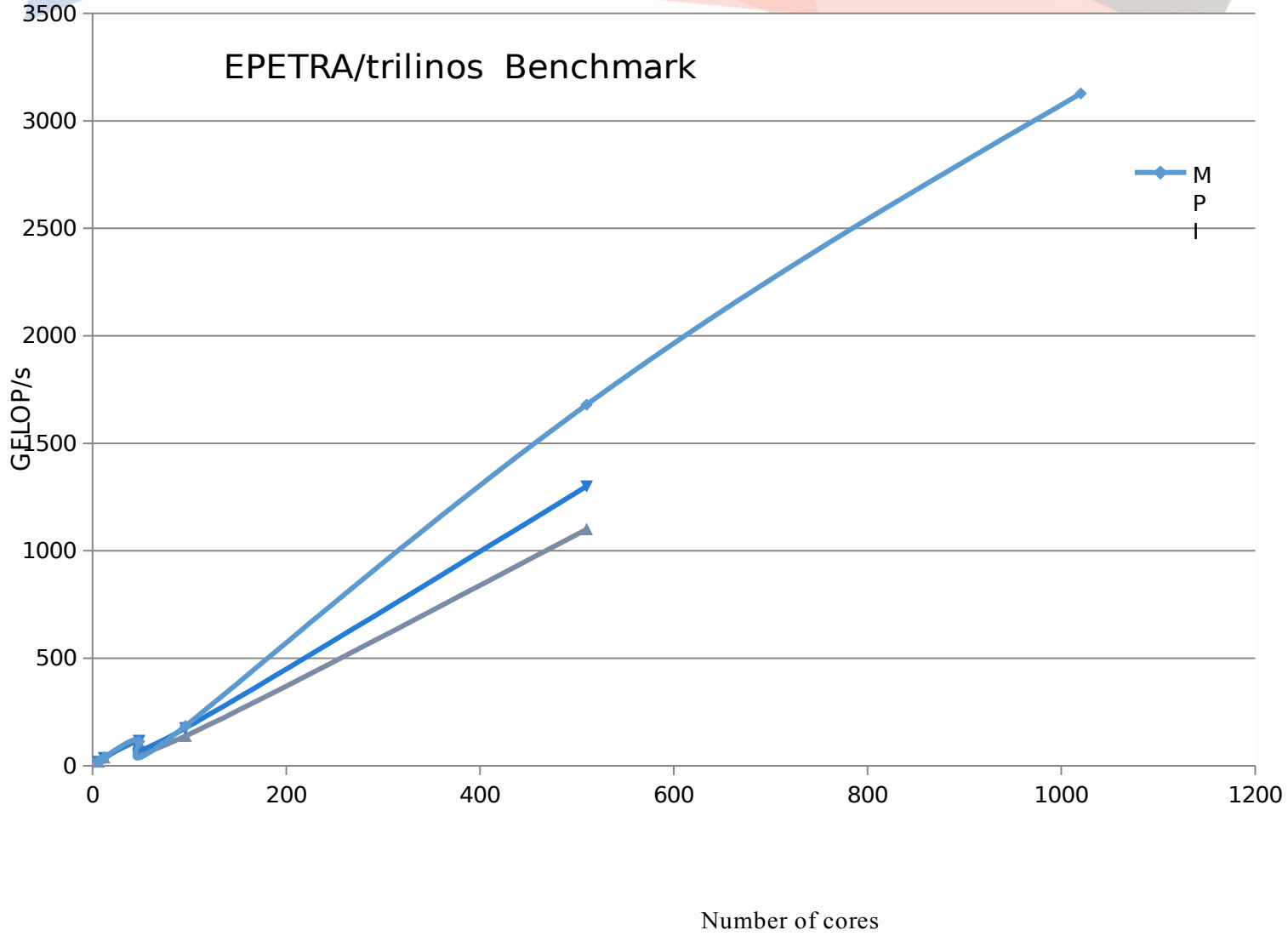


CIMNE

CSUC UV 1000 CLuster: Benchmarks



CSUC UV 1000 CLuster: Benchmarks



Profiling & Debugging

We still miss a good profiler to assess the performance of our algorithms.

Very problematic since the profiler needs:

- Support for multiple shared libraries loaded from python
- Support both OpenMP and MPI
- Shall not need annotations in the code. (Not feasible to annotate large code bases!!)

Debugging is also quite painful, due to the mix of python and C++

Programming paradigms to address new machines?

We need to choose a programming paradigm to address modern/future machines

Requirements:

- Compiler shall have industrial grade C++ support. In particular must be able to compile **Boost** and **Trilinos** and have good support for “template magic”
- Shall be portable to any OS (including Windows!!!!!)
- Shall NOT be vendor-locked
- Shall allow addressing accelerators
- Ideally shall allow OO programming (if this is not the case application will be limited inside our code)

As of now, the programming paradigms that comply with this requirements are

- OpenMP 4.0/OpenACC
- TBB (Threading Building Blocks)
- OpenCL (very very very verbose and tedious to program)

Open Questions:

Question 1:
Apparently a new technology can be found on the shelves. AMD HSA/hUMA shall, according to *marketing info*, allow to address GPUs in a more transparent way. **Will this maintain the promise of OO programming on GPU**? If so it would be a real revolution… could anyone comment on this

Question 2:
Does OpenMP 4.0 allow (in principle) OO programming on GPUs? No implementation out to test…