Low-hanging fruits of optimization

Björn Dick (HLRS)

Some remarks first …

- **There might be runtime variations**

  → if possible, use median of multiple runs in order to establish baseline and assess benefit of optimisations

- Please focus on time stepping loop, neglect setup and clean-up phases

- Please state improvements in terms of „overall speedup“, i.e.

  $S = \frac{t_{old}}{t_{new}}$, e.g. 1.2x

- Please mention whether you are talking about improvements of *absolute* runtimes or *fraction* of walltime!

- From time to time, you should validate your optimizations w.r.t. your production job (size)

- Start with measuring baseline

Placement

- Have in mind the structure of the processor (in particular CCX), cf.
  `https://kb.hlrs.de/platforms/upload/Processor.pdf`
- Try different numbers of active cores / placements of those:
  `mpirun … omplace -c <range>:bs=<bs>+st=<st> <binary>`
  - `<range>`: 0-127 w/o hyperthreads, 0-255 with hyperthreads
  - `<st>` : length of blocks (“stride”)
  - `<bs>` : number of active cores per block (“block size”)
- For more complex patterns, use the scripts provided at
  `https://kb.hlrs.de/platforms/index.php/Batch_System_PBSPro_(Hawk)#Shall_I_use_all_the_available_cores.3F`
- Figure out sweet spot by means of fine-grained sweeps,
  do *not* use powers of 2 only!
- Vary MPI vs. OpenMP size

Try different …

- … compilers:
  - GNU (`module load gcc`)
  - AMD (`module load aocc`)
  - Intel (`module load intel`)
- … MPI implementations:
  - MPT (`module load mpt`)
  - MPT (`module load openmpi`)
- … BLAS / LAPACK implementations (but also FFTW etc.):
  - MKL (`module load mkl && export MKL_DEBUG_CPU_TYPE=5 && export MKL_ENABLE_INSTRUCTIONS=AVX2`)
  - AOCL (`module load blis / libflame`)
  - PGI (no module available yet)

Compiler flags

- cf. AMD's "Field Application Engineering HPC Optimisation Guide – 'Rome'"
  - Section 7.1.3
  - provided in `/lustre/cray/ws9/2/ws/hpcbjdic-HPE1/`
- cf. AMD's "Compiler Options Quick Reference Guide"
  - `https://www.amd.com/system/files/documents/compiler-options-guide-amd-epyc-7xx1-series-processors.pdf`
- cf. PRACE's "Best Practice Guide - AMD EPYC "
  - `https://prace-ri.eu/wp-content/uploads/Best-Practice-Guide_AMD.pdf`
  - Section 3.1.1
- cf. AMD's "Compiler Usage Guidelines forAMD64 Platforms "
  - `https://www.amd.com/system/files/TechDocs/32035.pdf`
  - Chapter 3

Compiler flags (cont.)

- Inter-procedural optimization:
  - By default, compilers optimize within routines / files only
  - Allow them to do global optimizations by means of:
    - `-flto` (GNU, AOCC)
    - `-ipo` (Intel)

- Profile-guided optimization:
  - Compiler determines optimization potential by means of running generated code and optimizes again afterwards.
    - *GNU*: `https://developer.ibm.com/articles/gcc-profile-guided-optimization-to-accelerate-aix-applications/`
    - *AMD*: `https://clang.llvm.org/docs/UsersManual.html#profiling-with-instrumentation`
    - *Intel*: `https://software.intel.com/en-us/cpp-compiler-developer-guide-and-reference-profile-guided-optimization-pgo`

Tweak NUMA behavior

- System runs in NUMA mode NPS=4
- NPS=2 or NPS=1 might be better for your application
  (if so, it points to performance bugs in your application!)
- We provide a skript in `/lustre/cray/ws9/2/ws/hpcbjdic-HPE1/bin/` to "fake" NPS=2 or NPS=1
- Please ask Christian with respect to how to use it
- Disabling "transparent huge pages" might also help (coming soon)
- Enabling "NUMA balancing" might also help (coming soon)
- Use **`dlook`** to investigate NUMA behavior of your code
- Use **`numactl`** to further tune NUMA behavior of your code

Tweak MPI runtime parameters

- MPT
  - With respect to details cf. (slide 28 ff.)
    `https://fs.hlrs.de/projects/par/events/2020/HPE-1/04_HLRS_MPT-Summary_CS.pdf`
  - MPI_BUFFER_MAX
  - MPI_BUFS_PER_PROC
  - MPI_BUFS_PER_PROC
  - MPI_COLL_OPT
  - MPI_ADJUST_*
  - MPI_IB_TM
  - MPI_COLL_HCOLL
  - MPI_IB_CONGESTED
  - MPI_IB_SERVICE_LEVEL
- OpenMPI:
  - Coming soon

Tweak I/O

- On Lustre-based ws9, adapt
  - Stripe count (`lfs setstripe -c <stripe count>`)
  - Stripe size (`lfs setstripe -S <stripe size>`)
  - Multiplier (`MPI_Info_set("cray_cb_nodes_multiplier", "<multiplier>")`)
  - Lustre-Lock Ahead (`MPI_Info_set("cray_cb_write_lock_mode", "2")`)

- In order to find appropriate values, use `https://kb.hlrs.de/platforms/index.php/MPI-IO#Optimal_striping_on_ws9_.26_choice_of_multiplier` as a starting point for trial & error

Further ideas

- Use 32bit versions of libraries if sufficient
  - This will require less data to be transferred and hence reduce memory pressure of memory-bound codes
- Try static instead of dynamic linking
  - Might reduce overhead induced by loading libraries
  - However increases size of binary
- …