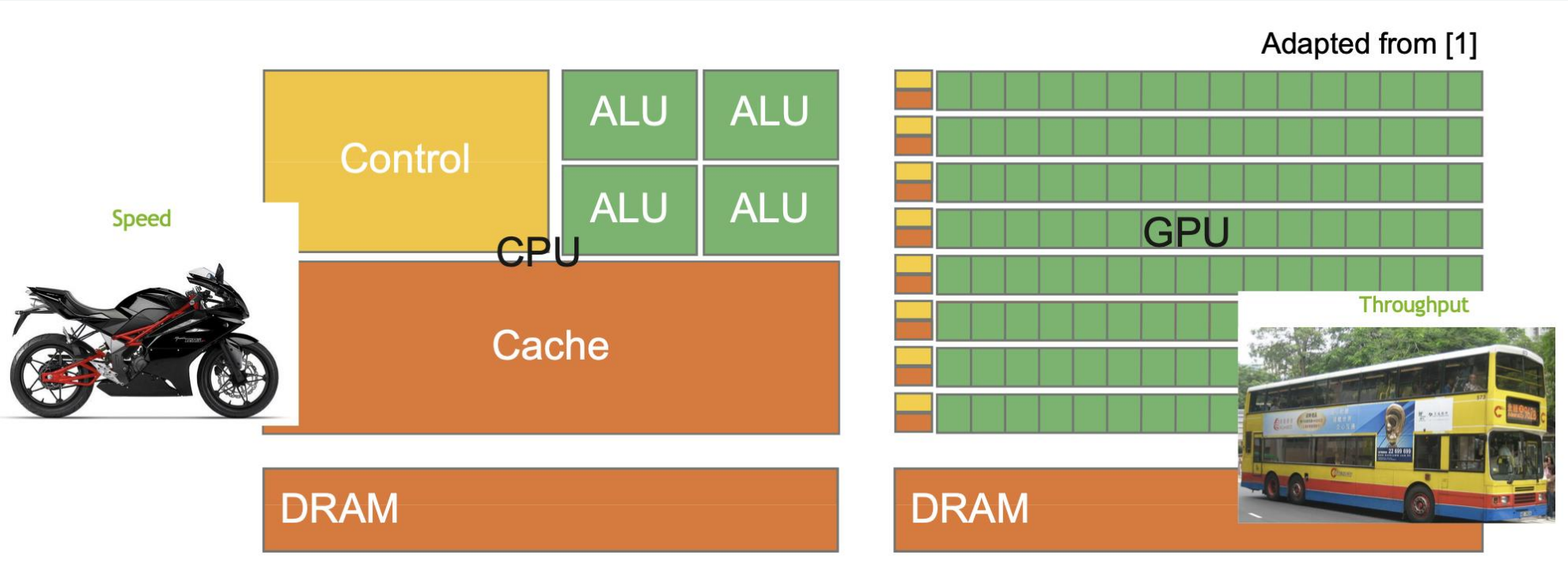
INTRODUCTION TO PC-A

"Pencil Code with Astaroth inside"

Work by M. Väisilä, J. Pekkilä, O. Lappi, T. Puro, M. Rheinhardt, led by M. Korpi-Lagg @ CS, Aalto University, Espoo, Finland

BASIC GPU ARCHITECTURE



optimising

sequential execution

parallel throughput

10x higher peak throughput
10x higher DRAM access speed

GPU HARDWARE ESSENTIALS

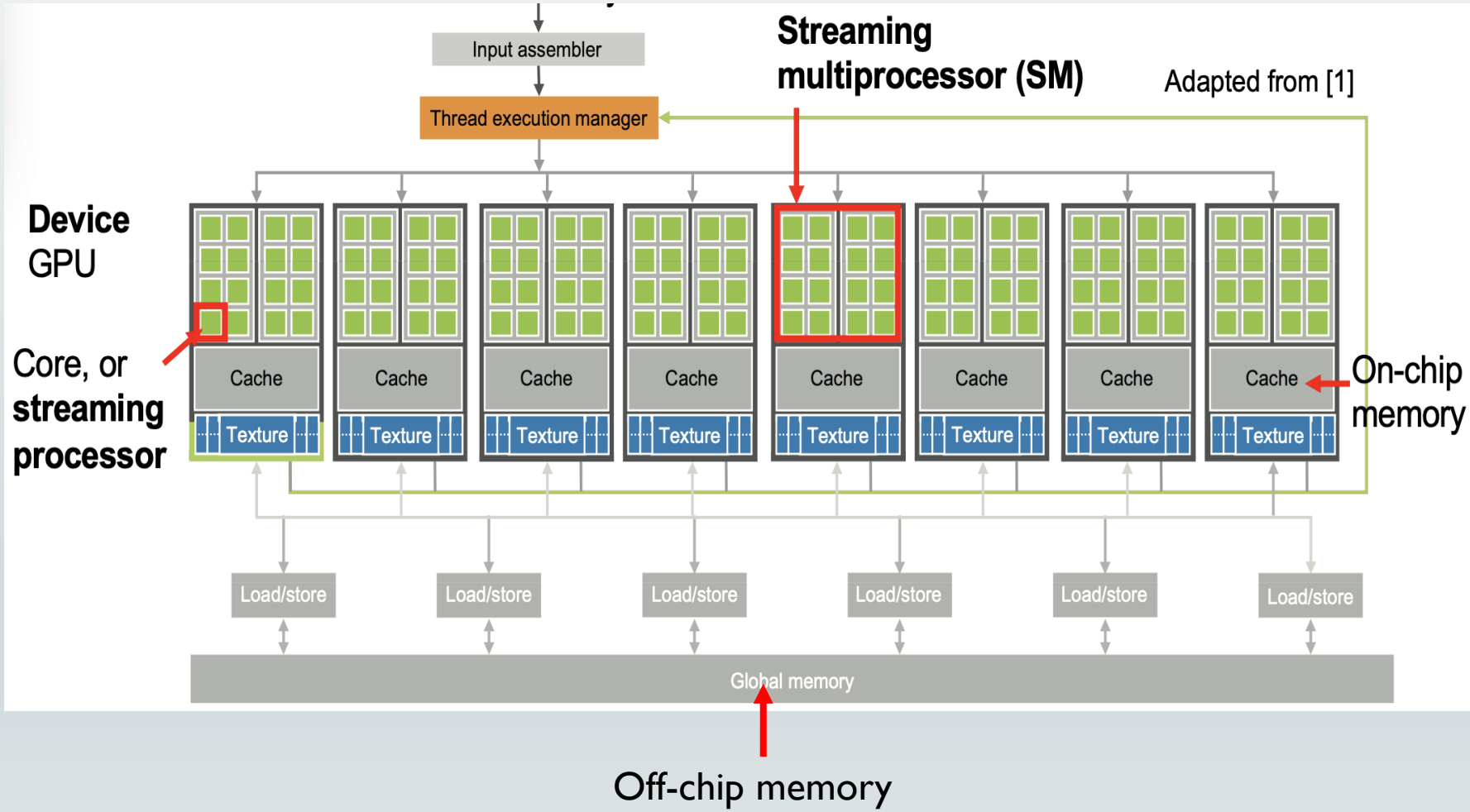
Correspondences

physical	logical
SP	thread
warp	-
SMP	thread block
GPU	block grid

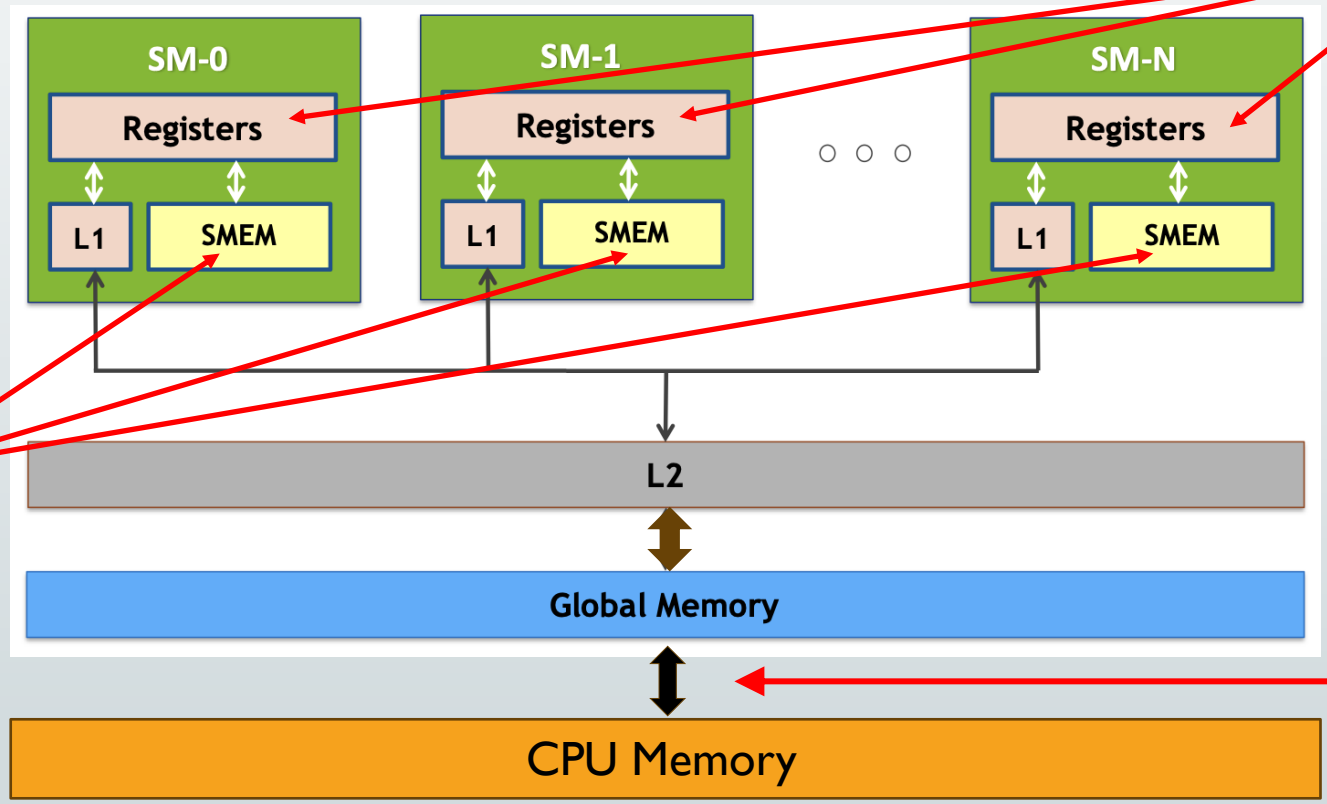
SIMT paradigm
(Single Instruction Mulitple Thread)

SIMD SIMT MIMD

autonomy



GPU MEMORY HIERARCHY

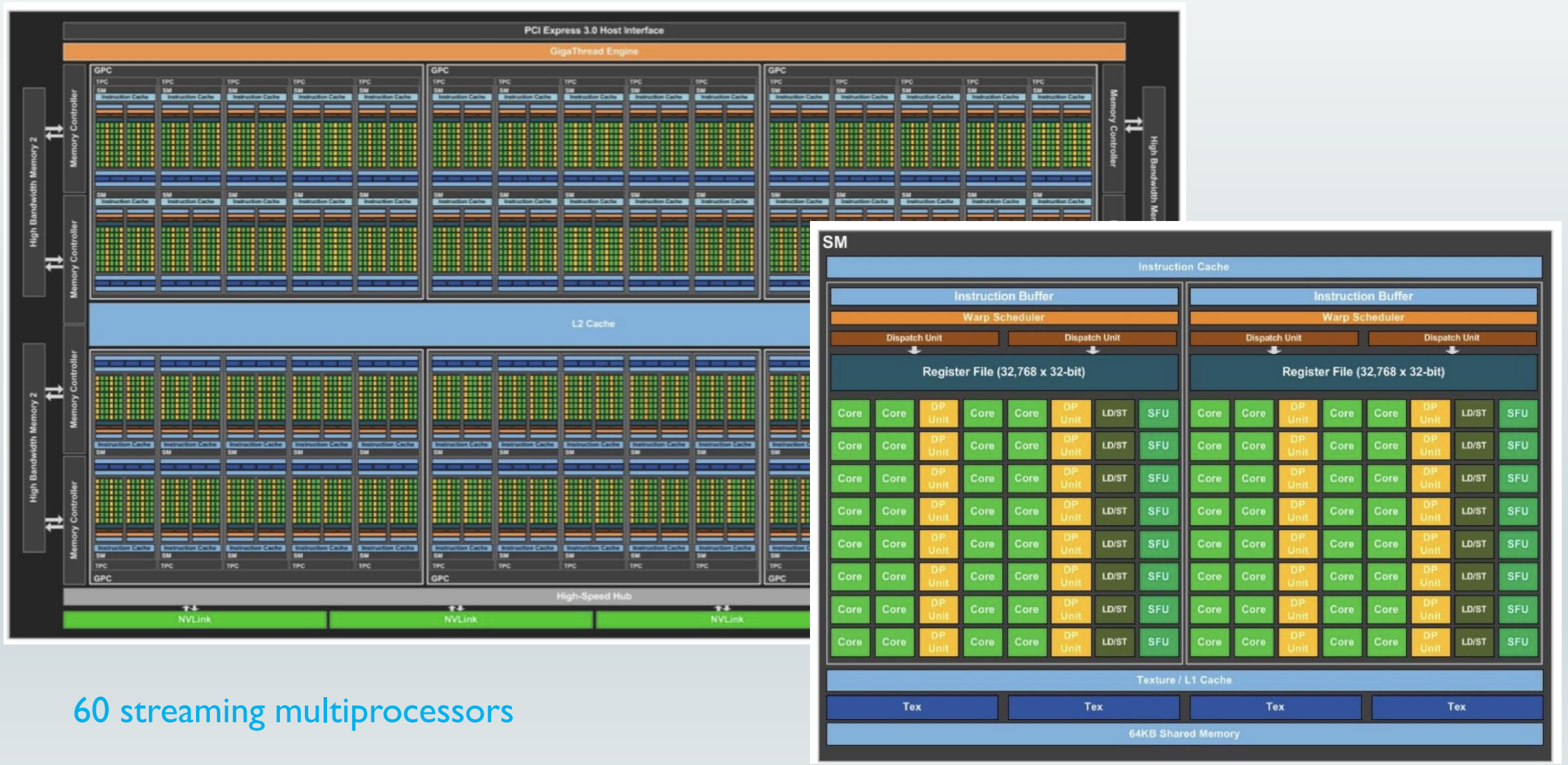


seen by single threads

for fast communication
between threads in a block

highest latency

GPU EXAMPLE: PASCAL GP100



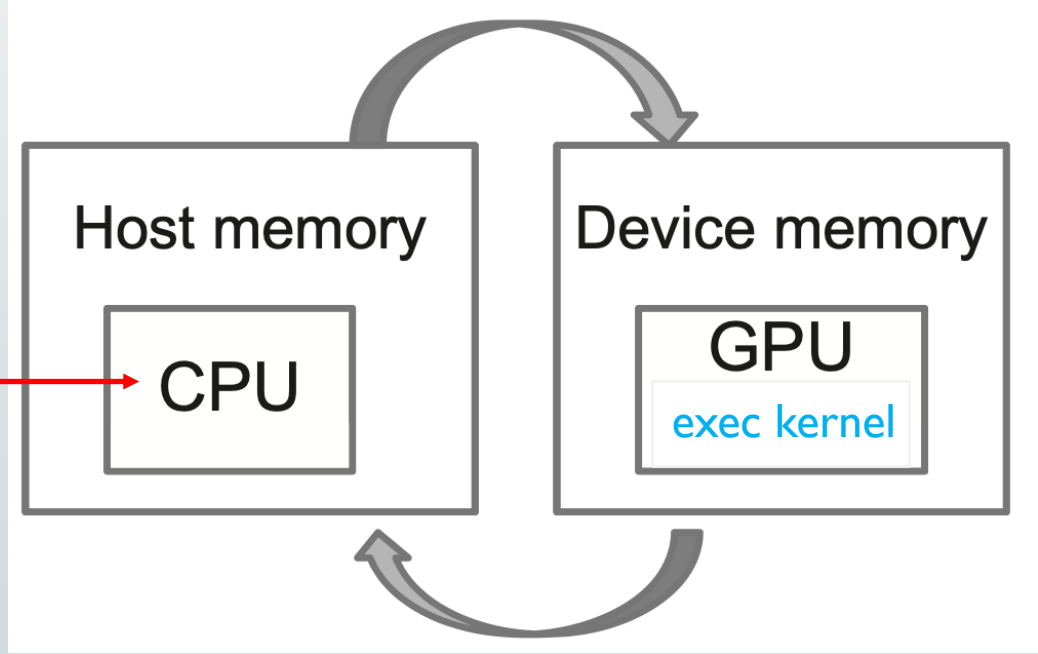
60 streaming multiprocessors

CPU-GPU WORKCYCLE

in every substep:

load program (kernel) and data (for PC: a few)

wait or act concurrently
during kernel execution



store results (for PC: a few)

PC-A: MINDSET

- two different codes in two different repositories

  (although sharing numerical methods)

- two different computers, although GPU dependent on CPU

  → everything needed for advancing the PDE variables

  - must be in GPU memory (parameters, initial conditions)
  - in kernel code (recipes for right hand sides, boundary conditions, etc.)

- f-array is updated on CPU only when needed, df-array does not exist

PC-A tries to use all resources on a node with maximum concurrency!

HOW TO GET THE CODES

- both codes together with a fresh pull:

```
git clone --recurse-submodules https://<username>@github.com/pencil-code/pencil-code.git.
(read-only)
```

or

```
git clone --recurse-submodules https://<username>@pencil-code.org/git/ pencil-code
source sourceme.sh
cd $PENCIL_HOME/src/astaroth/submodule
git checkout develop
```

- or add Astaroth and Fortran_parser to an existing PC installation:

```
cd $PENCIL_HOME
git checkout master (if necessary)
git submodule update --init  --recursive
cd src/astaroth/submodule
git checkout develop
```

BRANCHES MATTER!

HOW TO BUILD PC-A

- on clusters, load appropriate CUDA (NVIDIA) or HIP (AMD) and MPI modules, perhaps `cmake`
- set in `Makefile.local`

```
GPU                                    = gpu_astaroth
     MULTITHREADING              = openmp
     RUNTIME_COMPILATION = on
     TRANSPILATION                  = on
    GENERATE_DSL_CODE      = on
```

- if you haven't used PC-A before in work directory, execute `pc_setupsrc`

-> new symbolic links and directories in

`src/astaroth` interface routines and PC specific DSL code

`src/astaroth/submodule` Astaroth library and DSL compilation framework

- build Pencil Code as usual:
  - `make`: set `MODULE_[PRE|IN|SUF]FIX` environment variables as given in config file
  - `pc_build`: use proper config file (flag `-s|--serial` for troubleshooting)
    - `gfortran`: set FFLAGS+=–mcmodel=[medium|large] for large grids
    - CRAY: set FFLAGS+=-h pic -Wl,--no-relax ~

HOW TO BUILD PC-A

- build process creates
  - DSL and interface code according to the setup
  - Astaroth libraries and interface library

  - for libraries: separate `Makefile` in `src/astaroth`

    -> libraries can be built separately (perhaps after "`make clean`", answer "y")

- most importantly: kernels and their grouping in `src/astaroth/DSL/local`:

`mhdsolver.ac`

- which includes

`steps_two.h`

`boundconds.h`: boundary conditions (mandatory "step")

HOW TO RUN PC-A

Example SLURM batch script:

```
#SBATCH --nodes=2            # Total number of nodes
#SBATCH --ntasks-per-node=8
#SBATCH --cpus-per-task=7    # multithreading
CRAY (LUMI, Dardel, Frontier):
#SBATCH --gpus-per-node=8    # Allocate one gpu per MPI rank
CSC machines:
#SBATCH --gres=gpu:v100:4                ~

source src/.moduleinfo
export LD_LIBRARY_PATH=${CRAY_LD_LIBRARY_PATH}:$LD_LIBRARY_PATH
export OMP_NUM_THREADS= =${SLURM_CPUS_PER_TASK}
export OMP_MAX_ACTIVE_LEVELS=2
export OMP_PROC_BIND=close,spread
export OMP_WAIT_POLICY=PASSIVE
./start.csh
export MPICH_GPU_SUPPORT_ENABLED=1   # MPICH only
./run.csh
```

ALTERNATIVES: OFFLOADING

- OpenACC:

```
#pragma acc data copyin(a[0:n], b[0:n]) copyout(c[0:n])
// defines the data that needs to be moved to the device
{
    #pragma acc parallel loop.                                    //
offloads the loop for parallel execution
    for (int i = 0; i < n; ++i) {c[i] = a[i] + b[i];}
} // End of data region. The data is copied back from the device.
```

- OpenMP:

```
#pragma omp target "map(tofrom:y[0:n])"  //other variables implicitly
copied to device
#pragma omp teams distribute parallel for
for (int i = 0; i < n; i++) { y[i] = a * x[i] + y[i]; }
```

- Others: Kokkos (library calls).

WHAT IS ASTAROTH?

- library for creating efficient GPU kernels for stencil operations
  - analyzing task dependencies

    -> launch kernels concurrently using CUDA streams
  - holding intermediate data in caches for reuse
  - autotuning for optimal thread block sizes

- Domain-Specific Language (DSL) and compiler
  - facilitates writing of kernels
  - detects communication needs
  - establishes task graph

WHAT IS DOMAIN-SPECIFIC LANGUAGE?

- C-like, with some Python-like features (partly declaration-free)

- Example: rhs of continuity equation

```
Field LNRHO
        #define RHO LNRHO
        Field3 UU
       …
       dlnrho_dt(){
            glnrho = gradient(LNRHO)          // grad(rho) or
grad(lnrho)
            rhs =  - dot(UU, glnrho)
            if (ldensity_nolog){return  rhs - RHO*divergence(UU)}
        else                                  {return  rhs -
divergence(UU)}
```

WHAT IS DOMAIN-SPECIFIC LANGUAGE?

- operations to be understood pointwise on grid (one CUDA thread per grid point)
- essential types:
  - `Field, Field3` — array on grid subjectable to stencil operations
  - `Field chemistry[n_species]` — many-species array
  - `Kernel()` — transforms into CUDA Kernel
    – only way to call DSL code
  - `Stencil()` — compact stencil definition
- qualifiers:
  - `dconst` — on device constant memory (fast)
  - `run_const` — constant during runtime (optimized away)
  - `gmem` — on device global memory (for array-type parameters)

PARAMETER TRANSFER TO GPU

- parameters of physics modules are pushed to GPU

by `subroutine pushpars2c(p_par)` near end of each physics module:

`call copy_addr(<parameter>,p_par(<running index>))`

for a new parameter: add a `call copy_addr(…),` increase `n_pars` if needed

integer	parameters:	add	! int
logical	~	add	! bool
real array	~	add	! (<dim>) [(<dim>) …] at line end

- note: parameter manipulations, doable in module initialization,

  not to be coded in DSL

  -> push derived parameters (typically logicals)

- parameters from `src/cparam.h, src/cdata.h:` all available

CONTROL FLOW DESIGN

- in general: all operations modifying f-array should happen on GPU

  -> branches `if (lgpu) then …` at all relevant places in PC sources, e.g.:

```
call before_boundary_gpu
call rhs_gpu
```

- called routine finally calls acGridExecuteTaskGraph(<name of compute step>,…

- task graph: organizes execution of kernels in maximally concurrent way,

  inferred from task dependencies, derived from “compute step”

- communication not explicitly specified:

  can be inferred from the modification of the “output buffer”

- specifically, concurrency of communication and computation is maximized

COMPUTE STEPS

- compute step: collection of kernels, to be executed without interference of CPU, presently:
  - `AC_rhs`
  - `AC_calculate_timestep`
  - `AC_before_boundary_steps`
  - `AC_before_boundary_steps_including_halos`
  - `AC_after_timestep`
  - `AC_gravitational_waves_solve_and_stress`

  - `AC_calc_selfgravity_rhs`
  - `AC_sor_step`

  } for self-gravity

  - `get_source_function_and_opacity`
  - `Qintrinsic_steps`
  - `Qextrinsic_steps`

  } for radiation

- need to be extended by hand

TRANSPILATION

- handwritten DSL code: error prone, needs human interference for keeping in sync with Fortran

- alternative – transpilation:

  transforms all code employed in `rhs_cpu` into single DSL kernel

  - all used module variables become global, in const or global GPU memory
  - Pencil Case dismantled into scalar variables,

    all “penciled” variables and operations -> pointwise
  - built-in functions of Astaroth (stdlib) used instead `sub.f90`, etc.
  - f-array -> individual Field objects, df-array: local variables

- can be part of PC-A build process

TRANSPILATION

- other transpilable cases, not part of PC-A build process:
  - boundary conditions
  - `before_boundary, after_boundary` stuff

- workflow: costumize parse.py ($PENCIL_HOME/fortran_parser)

  transpile by hand (python transpile.py)

  set preprocessor guards (e.g., `#if  LHYDRO`)

  add reduction calls by hand if needed

  new fields: handwritten addenda to

  fieldecs.h, df_declares.h, handwritten_end.h

  commit transpiled code to src/astaroth/DSL

OPTIMIZATION: RUNTIME-COMPILATION

- for optimization, Astaroth needs to know

  which stencil operations and reductions are needed

- PDE rhss contain many conditionals, depending on runtime data

  -> optimization inhibited

- solution: turning run-time constants into compile-time constants

  -> re-compilation of DSL code at run-time after reading of input

RUNTIME-COMPILATION

- elimination of all conditionals

  -> totality of all stencil calculations can exactly be inferred

- for boolean/integer variables changing within the time loop:

  kernel variants are produced

- unneeded Fields and arrays not allocated, unneeded compute steps eliminated

- uncommunicated fields detected -> only one buffer in GPU global memory

LIMITATIONS

- uncovered modules: see exclusion list in `gpu_astaroth.f90, initialize_GPU`
- incomplete before/after boundary stuff
- missing diagnostics (when not derived from pencils)
- only one-processor FFT on GPU
- limited testing (see https://norlx51.nordita.org/tests/GPU/)
- build process & runtime-compilation are divas

-> always test against CPU version !!!

but promise: speedup = 10 ... 30 !

TROUBLESHOOTING

- enforce re-creation of interface code by

  `rm src/astaroth/PC_moduleflags.h` (tb improved)

  `pc_build` …
- and libraries

  ```
  cd src/Astaroth
  make clean
  make
  ```
- obey DSL syntax meticulously - consult:

  https://bitbucket.org/jpekkila/astaroth/src/develop/acc-runtime/README.md
- compare with samples in `GPU autotest`
- consult touko.puro@aalto.fi or matthias.rheinhardt@aalto.fi

OUTLOOK

- in preparation:
  - run-time compilation -> all variables constant during time-loop,

    esp. logical variables, are replaced by their values from `start.in/run.in`

    performance!

full transpilation of the rhss to DSL -> manual DSL coding no longer needed!

WHAT THE BUILD YIELDS

- virgin build -> void rhs functions in `equations.h` -> user intervention needed:

inspect directories in `src/astaroth/DSL/`

```
density
        entropy
        forcing
        hydro
        magnetic
        shock
        supernova
```

for useful code snippets

- indicate, which physics modules are supported presently
- differential operators etc. are in `src/astaroth/DSL/stdlib`

WHAT THE BUILD YIELDS

- Kernel and boundary conditions in `src/astaroth/DSL/local:`

  `mhdsolver.ac, boundconds.h`

```
density
        entropy
        forcing
        hydro
        magnetic
        shock
        supernova
```

for useful code snippets

- indicate, which physics modules are supported presently
- differential operators etc. are in `src/astaroth/DSL/stdlib`

CUSTOMIZE RHS

- to specify a rhs, modify e.g., function `dlnrho_dt` of `src/astaroth/DSL/local/equations.h`

```
dlnrho_dt(int step_num){ return 0. }
```

to

```
dlnrho_dt(int step_num){
    #include "../density/continuity.h"
}
```

or

```
duu_dt(int step_num){ return real3(0.,0.,0.) }
```

to

```
duu_dt(int step_num){
    #include "../hydro/momentum.h"
}
```

CUSTOMIZE RHS

- "physics branches" in DSL code can be selected by preprocessor statements like

```
#if LMAGNETIC
    ...
#endif
```

for each enabled physics module, a flag is predefined in `src/astaroth/PC_moduleflags.h`

- or conditionals in DSL syntax, like

```
if (ltemperature) {
...
}
```

- all switches from `src/cparam.inc` available

- changes to `equations.h` are permanent = not overwritten by future builds

(additional physics -> new empty rhs functions appear,
no longer needed functions do not disappear but are idle)

- `equations.h` is considered by `pc_newrun` and `cvsci_run`

CUSTOMIZE RHS

- Caveat: it is advisable to check predefined DSL code in the beginning,

  at least more complex functions like `denergy_dt`

- Note: to enable an additional physics module, the block in

  `src/gpu_astaroth.f90` has to be released;

  can require some non-standard code development

- Limitations:
  - particles/pointmasses/radiation/solid cells/self-gravity/testfields

  presently not supported
  - modifications of f-array in `*before/*after_boundary` routines completely implemented

  if needed every timestep
  - diagnostics, which are not only from `pencil_case/f-array` not calculated
  - not all boundary conditions "transpiled" yet (coming soon)