# Porting HemeLB for human-scale blood flow simulation on GPUs and high performance computers

**Zacharoudiou, I.[1], McCullough, J.W.S.[1], Lo, S.C.Y.[1], Coveney, P.V.[1,2]**

[1] Centre for Computational Science, Department of Chemistry, University College London, UK

[2] Institute of Informatics, University of Amsterdam, The Netherlands

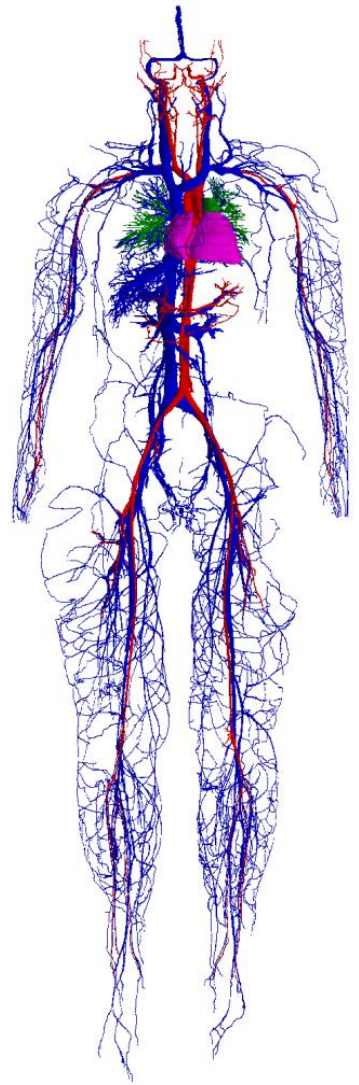
# Porting HemeLB for human-scale blood flow simulation on GPUs

1. Introduction
   - Motivation
   - What is HemeLB
   - Why porting to GPUs

2. Lattice Boltzmann Method
   - Basic concepts – Algorithmic steps

3. HemeLB – Algorithmic implementation

4. GPU code development
   - CUDA (NVIDIA)
   - Porting the CUDA code to HIP (AMD)
   - Porting the CUDA code to oneAPI (INTEL)



# Porting HemeLB for human-scale blood flow simulation on GPUs

## 1. Introduction - Motivation

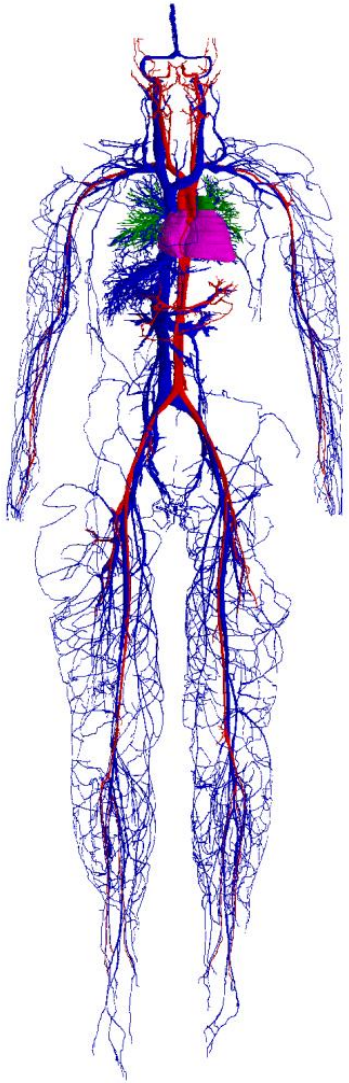
- Development of the **virtual human**
- Assist clinicians' ability to understand how a course of treatment will impact a given individual
- Simulating the patient using a personalised digital replica

  → **Patient specific modelling**
- Conducting simulations for the virtual human

  development of codes that can

  **execute** and **scale efficiently**

  on large-scale computing infrastructure



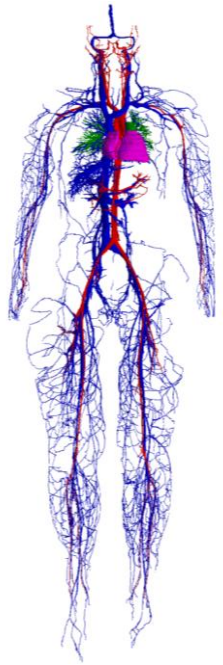
- **Develop a GPU version of HemeLB***

  Blood flow simulations on 3D vascular geometries.

* M.D. Mazzeo, P.V. Coveney (2008). Computer Physics Communications.

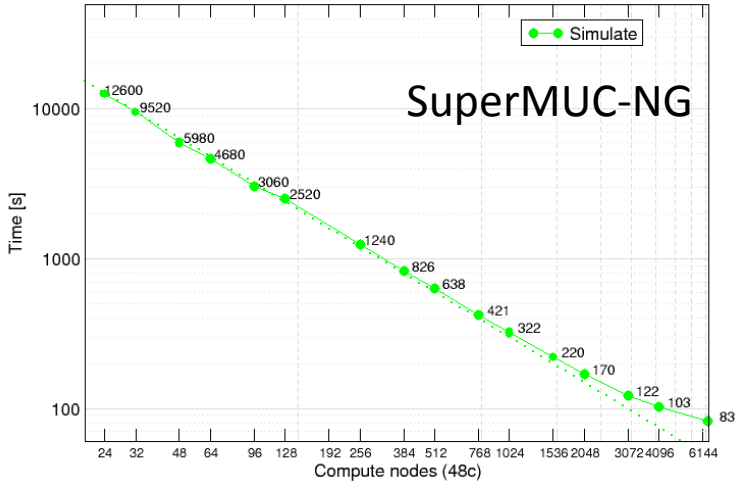
# Porting HemeLB for human-scale blood flow simulation on GPUs

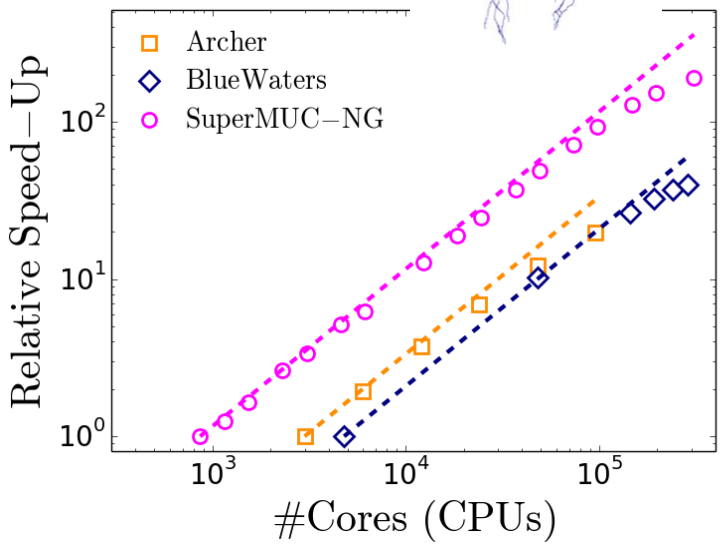
## 1. Introduction - HemeLB



- **Development of HemeLB* (heme from Hemodynamics)**
  - A high-performance, parallel **lattice Boltzmann (LB)** based fluid flow solver for simulating blood flow on patient specific images obtained from medical scans.
  - **C++** code parallelized using standard **MPI** communications
  - Optimised for sparse geometries (vascular trees)

- **MPI - Scalability of the code**
  - excellent strong scaling performance up to hundreds of thousands of CPU cores

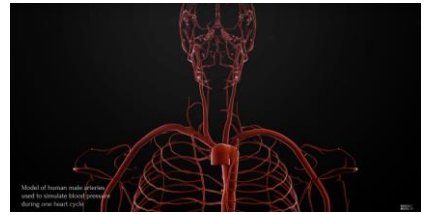



* M.D. Mazzeo, P.V. Coveney (2008). Computer Physics Communications.

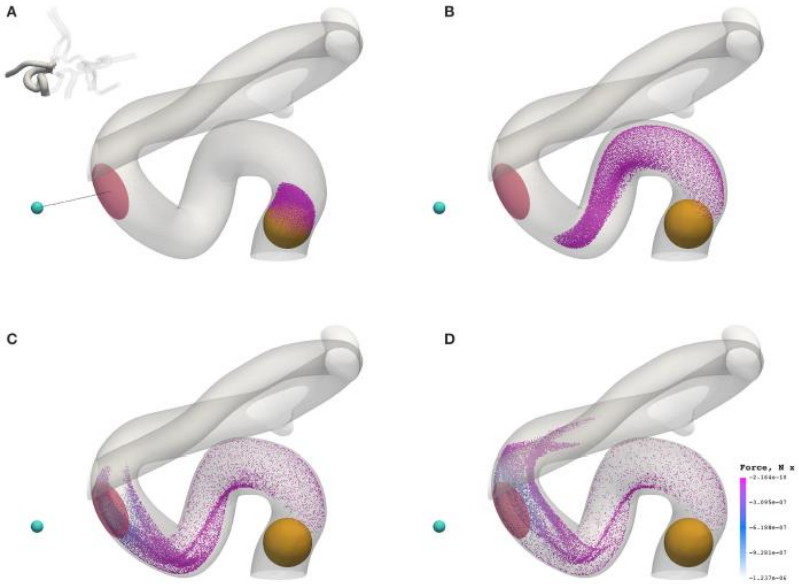
# Porting HemeLB for human-scale blood flow simulation on GPUs

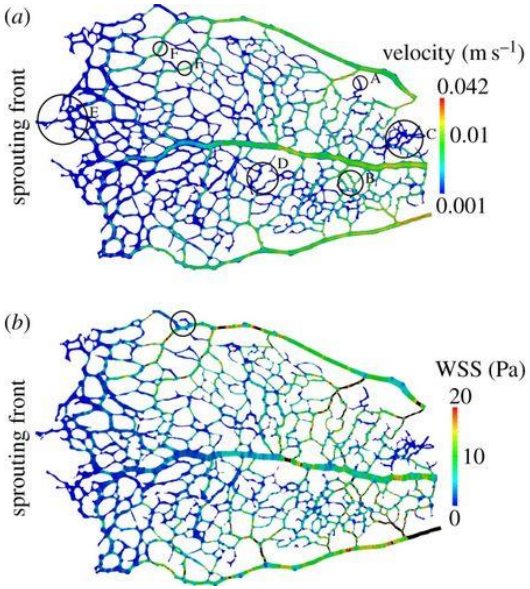
## 1. Introduction - HemeLB

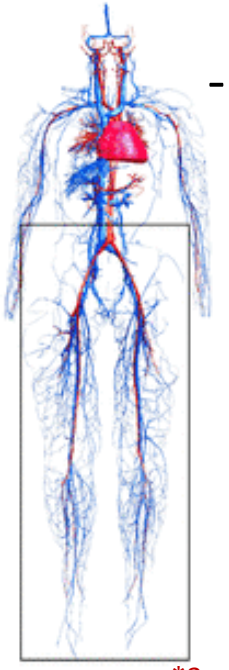


### Applications of HemeLB

- Cerebral aneurysms and blood flow
- Magnetic drug targeting[*1]
- Stent design
- Retinal vascular flow[*2]
- Coupling with organs, e.g the heart



- Self-coupling of HemeLB[*3] (simultaneous simulation of arterial and venous vascular trees)

[*1] A. Patronis et al. (2018). *Frontiers in physiology* 9, 331.

[*2] M.O. Bernabeu et al. (2014). *Journal of the Royal Society Interface*, *11*(99), 20140543.

[*3] J.W. McCullough et al. (2021). *Interface focus*, *11*(1), 20190119.

# Porting HemeLB for human-scale blood flow simulation on GPUs

## 1. Why porting HemeLB to GPUs?

### HPC machines accelerated by GPUs

**NVIDIA GPUs** | **AMD GPUs**

TOP500 The List.

PRESENTED BY: ICL Innovative Computing Laboratory; Berkeley Lab Lawrence Berkeley National Laboratory; ISC Group Moving Forward.

FIND OUT MORE AT: top500.org

| JUNE 2022 | SYSTEM | SPECS | SITE | COUNTRY | CORES | RMAX PFLOP/S | POWER MW |
|---|---|---|---|---|---|---|---|
| 1 | Frontier | HPE Cray EX235a, AMD Opt 3rd Gen EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-10 | DOE/SC/ORNL | USA | 8,730,112 | 1,102.0 | 21.3 |
| 2 | Fugaku | Fujitsu A64FX (48C, 2.2GHz), Tofu Interconnect D | RIKEN R-CCS | Japan | 7,630,848 | 442.0 | 29.9 |
| 3 | LUMI | HPE Cray EX235a, AMD Opt 3rd Gen EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-10 | EuroHPC/CSC | Finland | 1,268,736 | 151.9 | 2.94 |
| 4 | Summit | IBM POWER9 (22C, 3.07GHz), NVIDIA Volta GV100 (80C), Dual-Rail Mellanox EDR Infiniband | DOE/SC/ORNL | USA | 2,414,592 | 148.6 | 10.1 |
| 5 | Sierra | IBM POWER9 (22C, 3.1GHz), NVIDIA Tesla V100 (80C), Dual-Rail Mellanox EDR Infiniband | DOE/NNSA/LLNL | USA | 1,572,480 | 94.6 | 7.44 |

- **FRONTIER the first official exascale machine !!!**

# Porting HemeLB for human-scale blood flow simulation on GPUs

## 1. Why porting HemeLB to GPUs?

- **Graphics Processing Units (GPUs)** – become commonplace on HPC machines

- Develop a **GPU version of HemeLB (HemeLB_GPU)**

  - **CUDA (Compute Unified Device Architecture)**

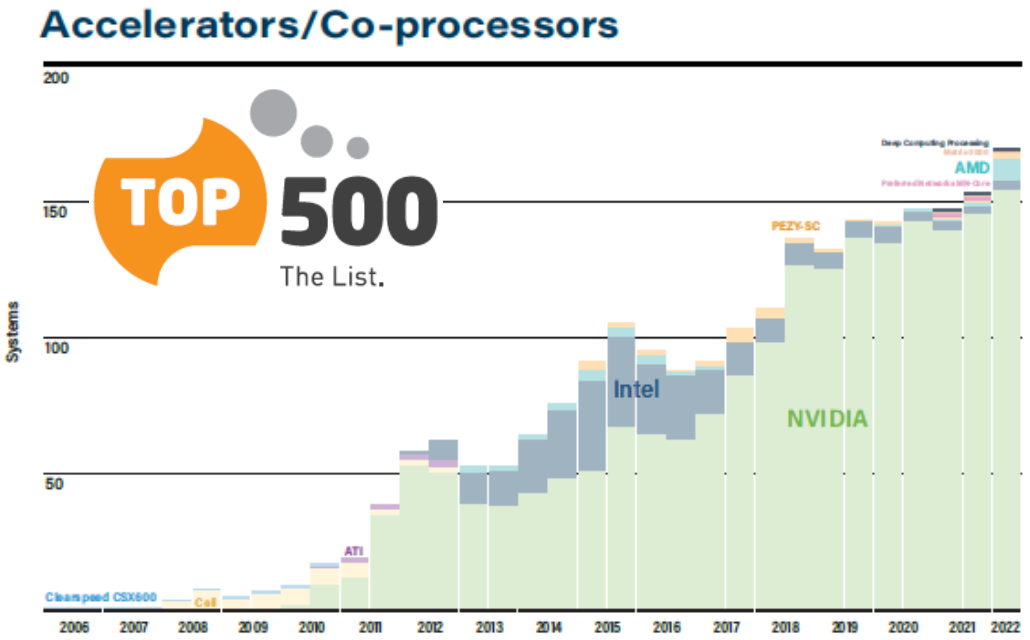
    **NVIDIA GPUs**

  - **HIP (Heterogeneous-Compute Interface for Portability)**

    **AMD GPUs** **NVIDIA GPUs**

  - **Intel oneAPI**

    Intel's hardware + NVIDIA + AMD




Make HemeLB_GPU **platform agnostic**

## 2. The lattice Boltzmann method (LBM)

- **Equations of motion**

Continuity eq. $\partial_t \rho + \partial_\alpha(\rho u_\alpha) = 0$

NS eq. $\partial_t(\rho u_\alpha) + \partial_\beta(\rho u_\alpha u_\beta) = -\partial_\alpha p + \partial_\beta\left[\eta\left(\partial_\beta u_\alpha + \partial_\alpha u_\beta\right)\right]$

$p(\boldsymbol{r},t) = c_s^2 \rho(\boldsymbol{r},t)$

$\eta = \rho(\boldsymbol{r},t) c_s^2 \left(\tau - \frac{\Delta t}{2}\right)$

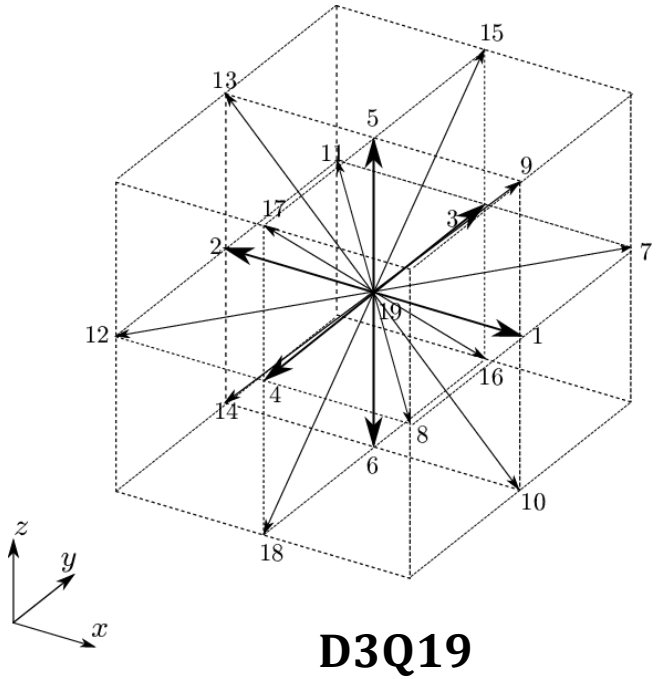
- **The lattice Boltzmann method**
  - Evolution equation of the particle distribution functions $f_i(\boldsymbol{r},t)$

    I. Collision step: $f_i'(\boldsymbol{r},t) = f_i(\boldsymbol{r},t) - \frac{1}{\tau}\left[f_i(\boldsymbol{r},t) - f_i^{eq}(\boldsymbol{r},t)\right]$

    II. Streaming step: $f_i(\boldsymbol{r} + \boldsymbol{e}_i \Delta t, t + \Delta t) = f_i'(\boldsymbol{r},t)$

  - Conservation of mass and momentum

$$\sum_{i=0}^{18} f_i^{eq} = \sum_{i=0}^{18} f_i = \rho \qquad \sum_{i=0}^{18} f_i^{eq} e_{i\alpha} = \sum_{i=0}^{18} f_i e_{i\alpha} = \rho u_\alpha$$




D3Q19

## 2. The lattice Boltzmann method (LBM)

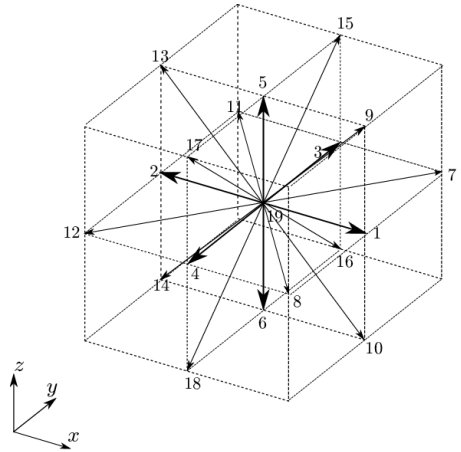
### Collide & Stream (propagate to the next lattice site)

- **Collision step**

$$f_i'(\boldsymbol{r},t) = f_i(\boldsymbol{r},t) - \frac{1}{\tau}\left[f_i(\boldsymbol{r},t) - f_i^{eq}(\boldsymbol{r},t)\right]$$

- **Streaming step**

$$f_i(\boldsymbol{r} + \boldsymbol{e}_i\Delta t, t + \Delta t) = f_i'(\boldsymbol{r},t)$$




**D3Q19**

### Apply Boundary Conditions

Missing incoming $f_i'(r,t)$

Inlets / outlets / solid surfaces

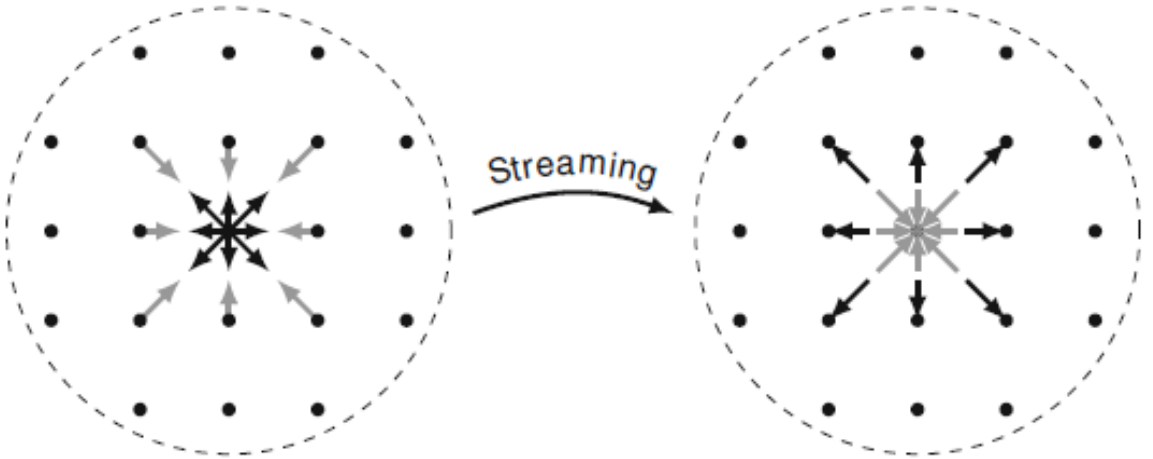
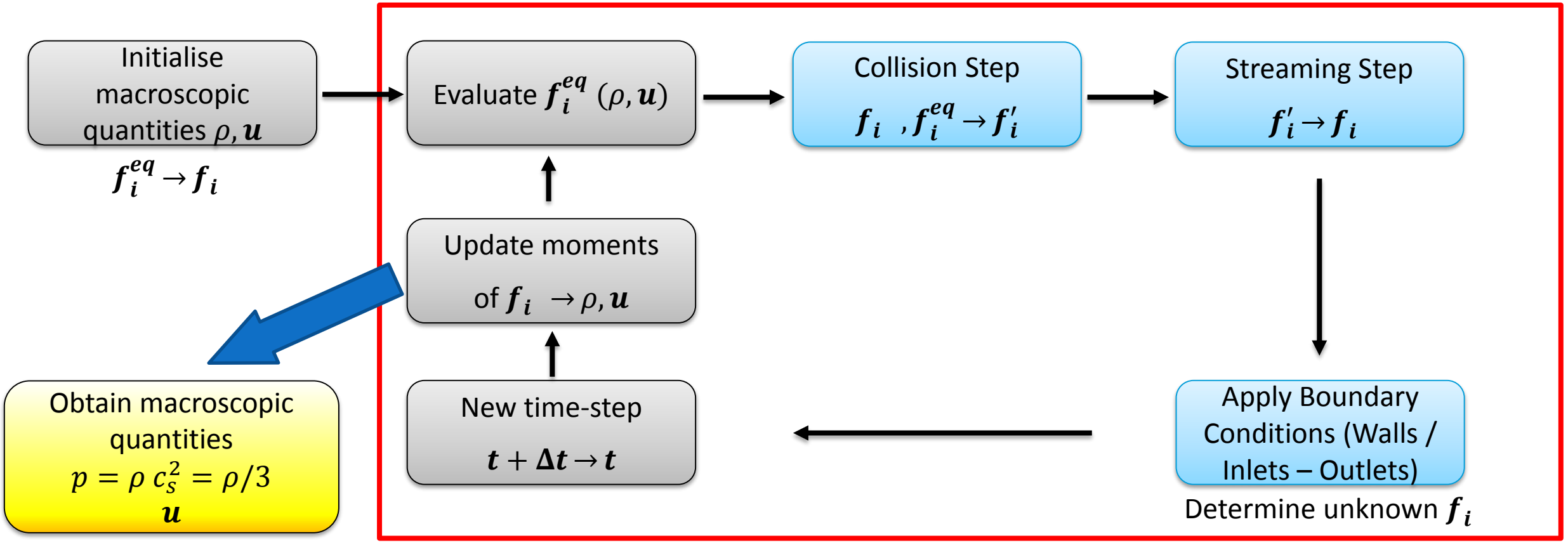



Image: Krüger, Timm, et al. "The lattice Boltzmann method." *Springer International Publishing* 10.978-3 (2017): 4-15.

# 3. HemeLB – Algorithmic implementation

## General background – The lattice Boltzmann Algorithm



# Porting HemeLB for human-scale blood flow simulation on GPUs

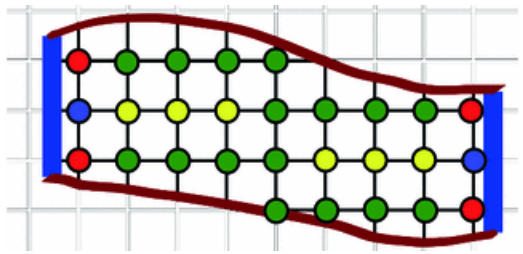
## 4. GPU CUDA version of HemeLB – Code Development

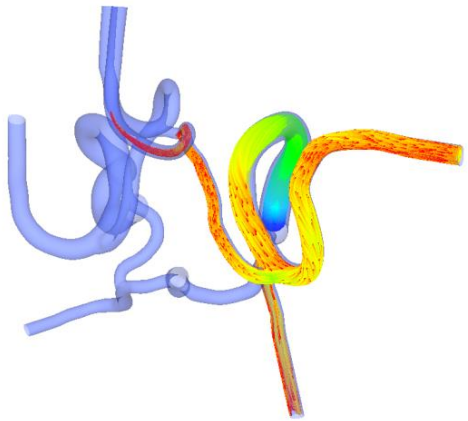
### General background

Porting HemeLB to GPUs → export the compute intensive parts of HemeLB onto the GPU

HemeLB distinguishes 6 types of **collision - streaming**:

1. **Inner domain**: only fluid sites without any links to any type of boundaries (walls or inlets/outlets),
2. **Walls**: fluid sites with a link to a solid surface,
3. **Inlet**,
4. **Outlet**,
5. **Inlet with Walls** and
6. **Outlet with Walls**.





**Collision – Streaming kernels**

Collision Step

$$f_i \; , f_i^{eq} \rightarrow f_i'$$

Streaming Step

$$f_i' \rightarrow f_i$$

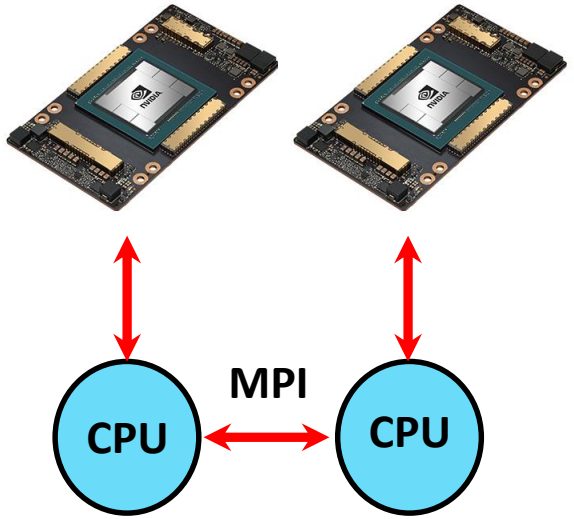
Apply Boundary Conditions (Walls / Inlets – Outlets)

**GPU Collision – Streaming kernels**

# Porting HemeLB for human-scale blood flow simulation on GPUs

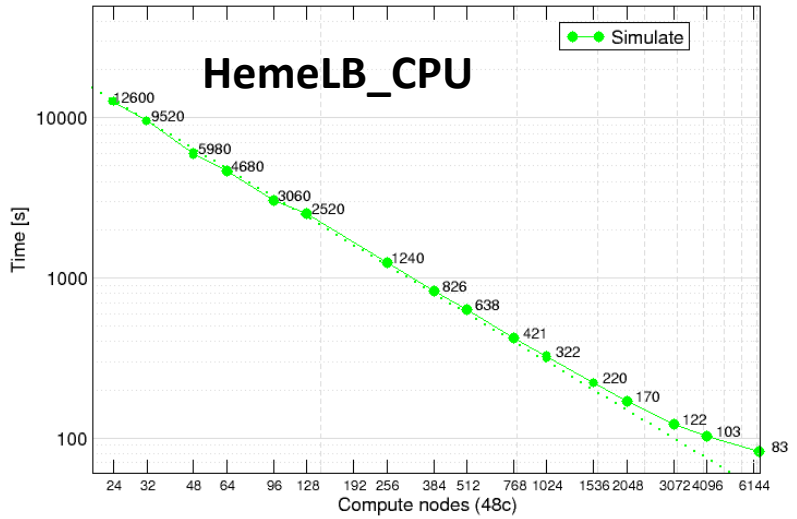
## 4. GPU CUDA version of HemeLB – Code Development

### Organising computations and MPI operations


- Collision - Streaming at **domain edges**
- **MPI exchange (send populations to neighbouring ranks)**
- Collision - Streaming at **mid-domain**

- Overlap **computations** and **MPI data exchange**

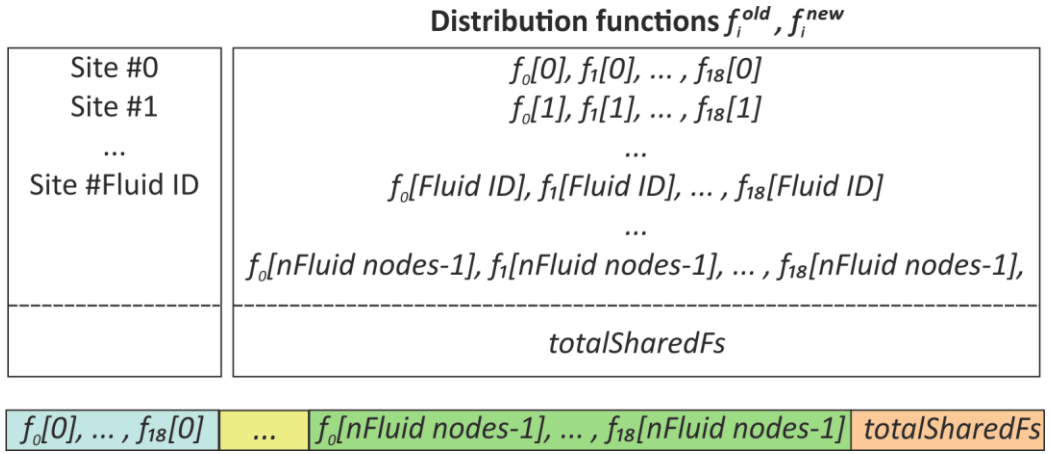



HemeLB_CPU: Strong scaling - SuperMUC-NG

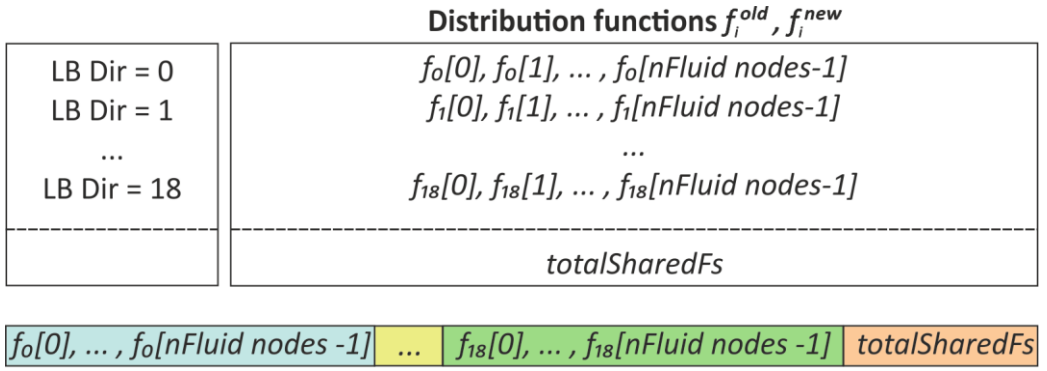
# 4. GPU CUDA version of HemeLB – Code Development

## Optimisation strategies

- Change of **data organisation – Take advantage of how GPUs read from GPU global memory**




**Array of Structures scheme (HemeLB_CPU)**




**Structure of Arrays scheme (HemeLB_GPU)**

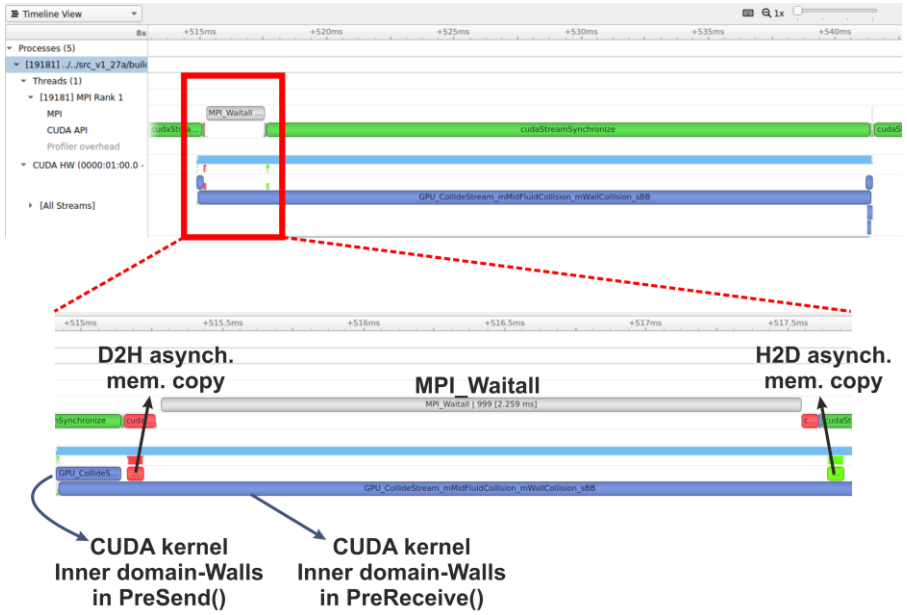
- Further optimisations (future work): **collected Structure of Arrays** scheme

# Porting HemeLB for human-scale blood flow simulation on GPUs

## 4. GPU version of HemeLB – Code Development

### Optimisation strategies

- Use of **different CUDA streams** for all GPU operations




CUDA kernels in different streams


**Overlap kernels' execution** and **memory copies**

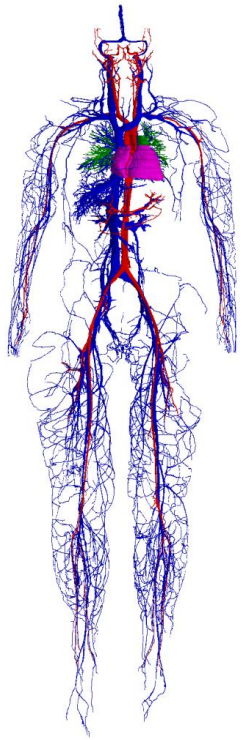
- Change the **sequence of steps**
  - Launch all kernels and then issue the MPI exchange

# Porting HemeLB for human-scale blood flow simulation on GPUs

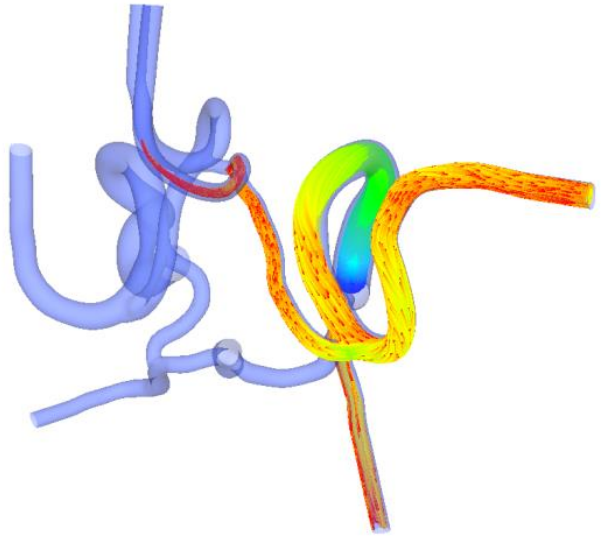
## Strong scaling performance

### Vascular domains

**Arteries of the legs**
$60 \times 10^6$ sites



**Full human venous tree**
$1.5 \times 10^9$ sites

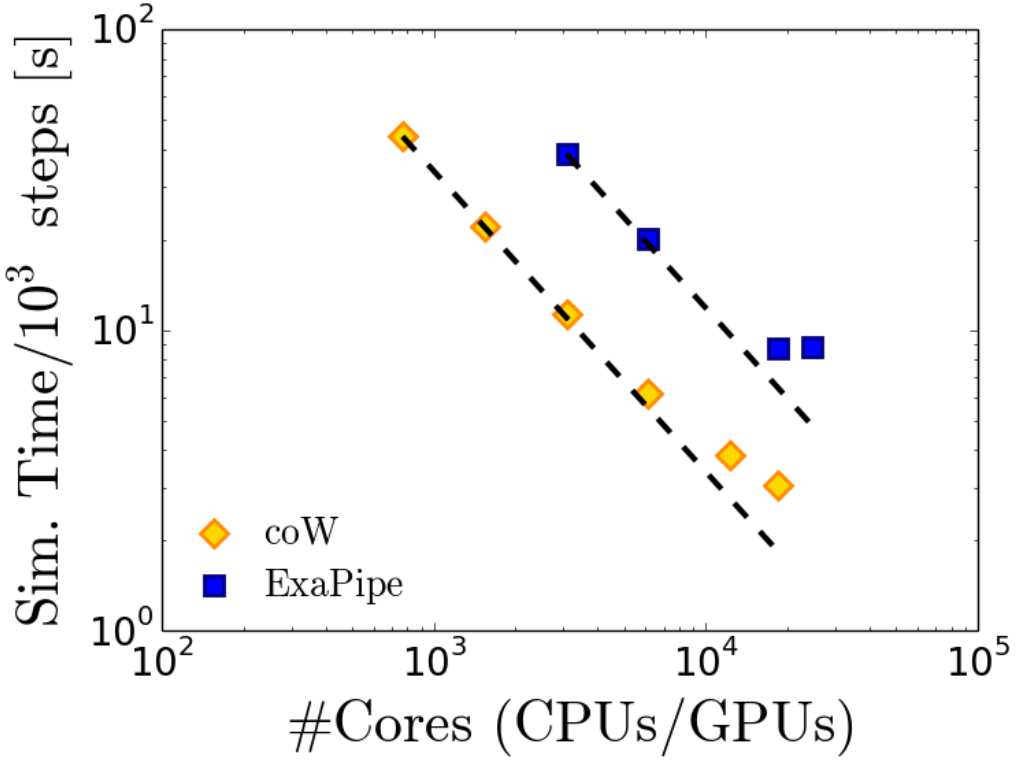


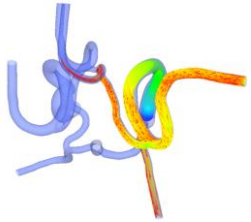
**Circle of Willis** geometry
system of arteries
at the base of the brain
$1.0 \times 10^{10}$ sites

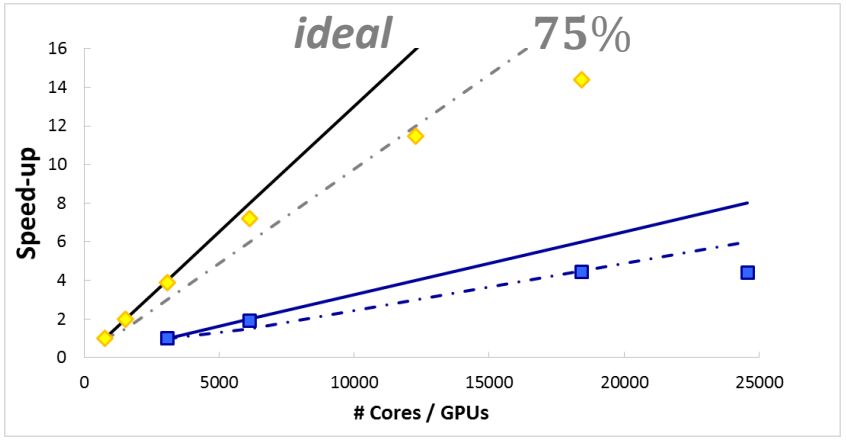
# Porting HemeLB for human-scale blood flow simulation on GPUs

## Strong scaling performance

**Summit** (ORNL) – 4th in Top500 list (2022)






**Circle of Willis (coW)** - $1.0 \times 10^{10}$ sites

- 90% perfect scaling performance on 6 144 V100 GPUs and continues strong scaling to 18 432 GPUs (approximately 2/3 of Summit's capacity).
- **Strong scaling efficiency** drops to
  - 72% at 12 288 GPUs and
  - 60% at 18 432 GPUs.

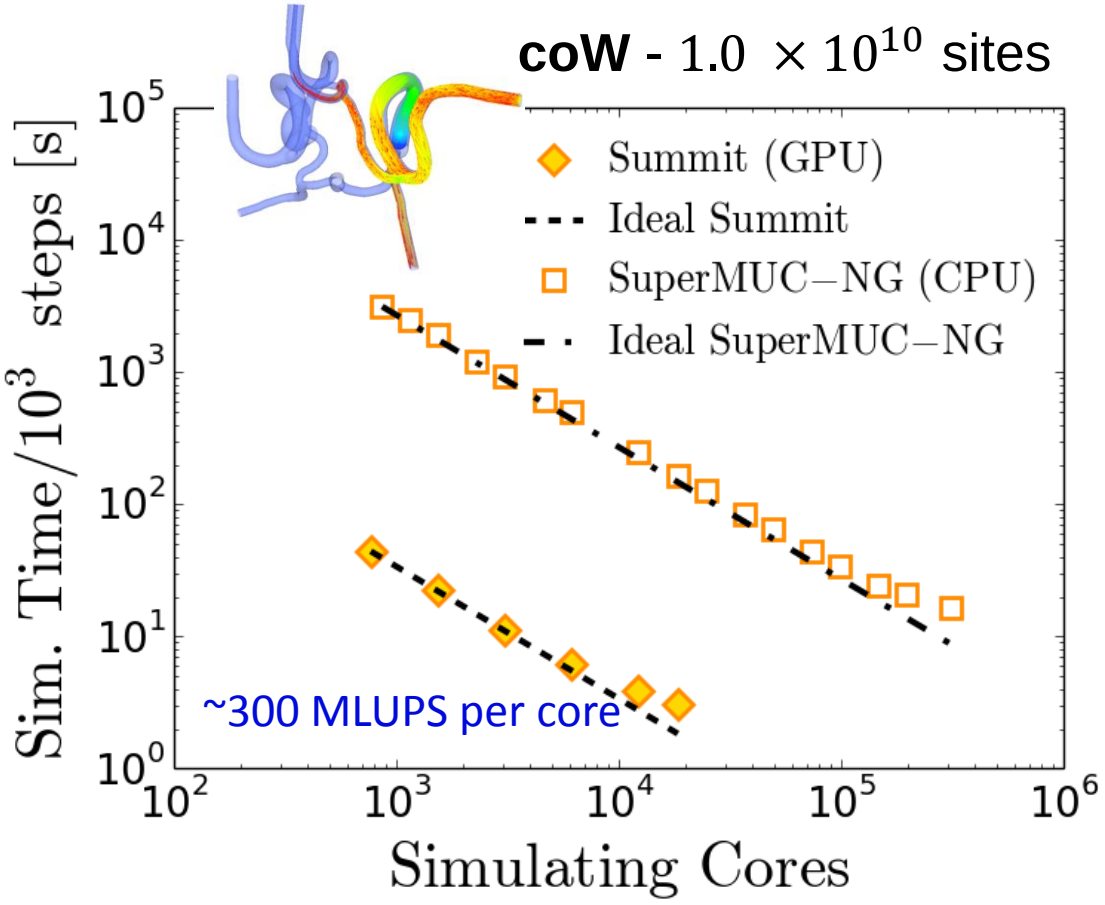
**exaPipe** - $3.7 \times 10^{10}$ sites

- improved strong scaling efficiency: 74% at 18 432 GPUs
- increased computation to communication ratio

# Porting HemeLB for human-scale blood flow simulation on GPUs

## 4. GPU CUDA version of HemeLB – Code Development

### Large scale performance comparison – CPU and GPU (CUDA) versions of HemeLB




- **Summit (GPU)**: 42 CPU cores & 6 V100 GPUs per node (1 V100 GPU - 5120 CUDA Cores)
- **SuperMUC-NG (CPU)**: 48 CPU cores per node
- Almost 2 orders of magnitude **speed-up** ($\times$ 85)
- Newer version of HemeLB_GPU available
  Additional $\times$ 1.9 speed-up

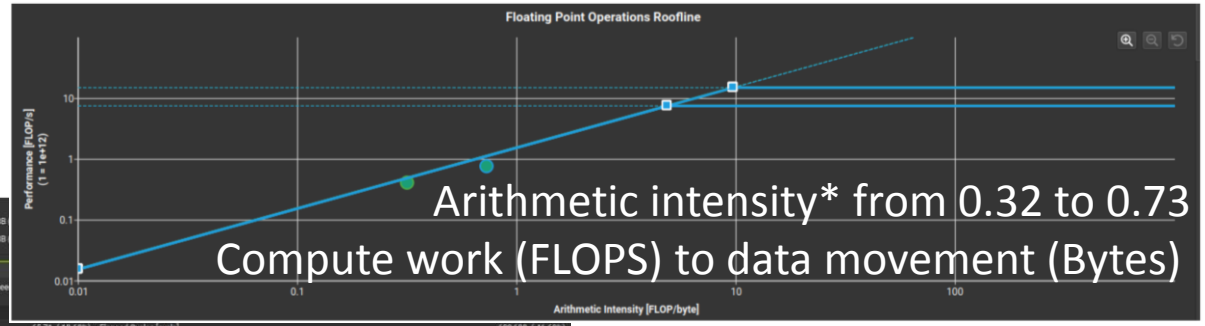
# Porting HemeLB for human-scale blood flow simulation on GPUs

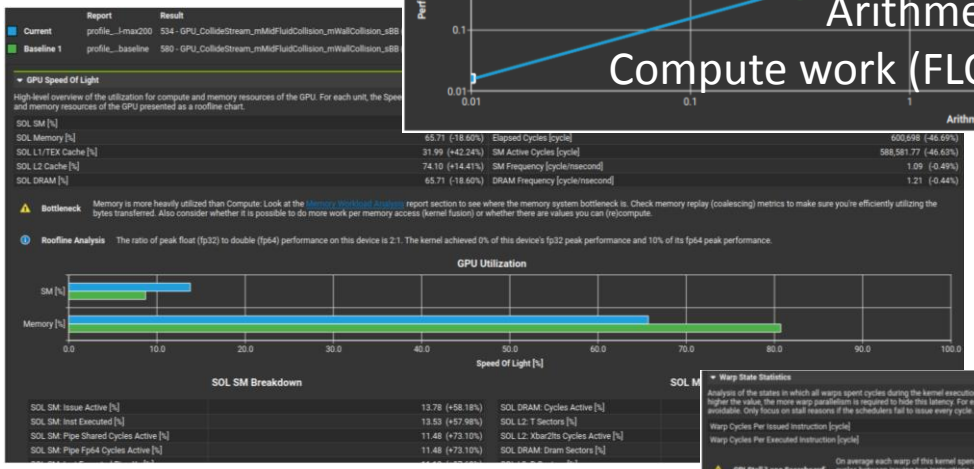
## 4. GPU CUDA version of HemeLB – Code Development

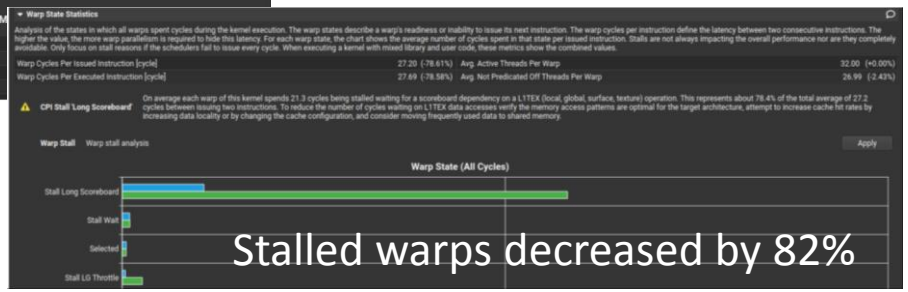
### Recent optimisations to the GPU (CUDA) version of HemeLB

- **Loop unrolling** (‘#pragma unroll 19’)
- **Loop merging**
- **Increase the number of registers per thread**
  Compile with flag **-maxrregcount 200**
  - Reduced memory traffic to/from the GPU global memory

Profiling with Nsight Compute




- **speed-up factor** ($\sim\times 1.9$)

Tested on Summit (**V100 GPUs**) with the **legs’ arteries geom. (many outlets)**
**From 285 to 550 MLUPs per CPU/GPU**

# Porting HemeLB for human-scale blood flow simulation on GPUs

## 4. Porting the CUDA version of HemeLB_GPU to HIP

- **Atos*** ported the C++ CUDA based HemeLB_GPU code to **HIP.**
- HemeLB_GPU can now also run on AMD's server GPUs **("hipified")**

***Paul Karlshöfer & Ludovic Hablot**
**Loris Lucido**

**What is HIP?**

- AMD's dedicated GPU programming environment -> high performance kernels on GPU hardware.
- Portability solution for GPU ready applications written in CUDA or applications aiming to target different GPU architectures.
- Provides a source-to-source translator to convert CUDA code to HIP.

**Two approaches to hipify a CUDA code:**

- **Hipify-perl:** perl script using regular expressions to substitute CUDA calls and headers to HIP
  - Easy to use, no CUDA dependencies
  - Might encounter difficulty with complex C++ construct
- **Hipify-clang:** clang-based conversion tool
  - Support complex construct (template, macro expansion, etc.) - Dependency with CUDA

# Porting HemeLB for human-scale blood flow simulation on GPUs

## 4. Porting the CUDA version of HemeLB_GPU to HIP

### Hipification
#### Hipify-clang

```
2447c2447
<       cudaError_t cudaStatus;
---
>       hipError_t cudaStatus;
2451c2451
<       if (myPiD!=0) cudaStreamSynchronize(stream_ReceivedDistr);
---
>       if (myPiD!=0) hipStreamSynchronize(stream_ReceivedDistr);
2480,2487c2480,2487
<         hemelb::GPU_CollideStream_mMidFluidCollision_mWallCollision_sBB
<           <<<nBlocks_Collide, nThreads_Collide, 0, Collide_Stream_PreSend_1>>> (...)
---
>         hipLaunchKernelGGL(hemelb::GPU_CollideStream_mMidFluidCollision_mWallCollision_sBB,
>           dim3(nBlocks_Collide), dim3(nThreads_Collide), 0, Collide_Stream_PreSend_1,
>           ...)
2637,2638c2637,2638
<             cudaStatus = cudaMemcpyAsync(d_ghostDensity, h_ghostDensity, n_Inlets * sizeof(distribn_t),
cudaMemcpyHostToDevice, stream_ghost_dens_inlet);
---
>             cudaStatus = hipMemcpyAsync(d_ghostDensity, h_ghostDensity, n_Inlets * sizeof(distribn_t),
hipMemcpyHostToDevice, stream_ghost_dens_inlet);
```

- CUDA Kernels untouched

9

Atos

**Similar syntax**

Errors' handling

Streams' synchronisation

GPU Kernels' launch

Memory copies

*** Loris Lucido (ATOS) – AMD GPU Workshop 2022 (SURF.NL)**

# Porting HemeLB for human-scale blood flow simulation on GPUs

## 4. Porting the CUDA version of HemeLB_GPU to HIP

- Target either AMD's or NVIDIA's GPU ( __HIP_PLATFORM_**HCC**__ OR __HIP_PLATFORM_**NVCC**__ )

```
# Could be fetched from hipconfig to handle CUDA target (__HIP_PLATFORM_NVCC__)
add_definitions(-D__HIP_PLATFORM_HCC__)
```

| | Node name | MI50 | V100 | MI100 |
|---|---|---|---|---|
| Processor | Processor SKU | Intel® Xeon® 8260 | Intel® Xeon® 6248 | AMD EPYC™ 7742 |
| | Processor class | Cascade Lake | Cascade Lake | Rome |
| | TDP | 165 Watts | 150 Watts | 225 Watts |
| | Core Frequency (nominal) | 2.4 GHz | 2.5 GHz | 2.25 GHz |
| | Cores per socket | 24 | 20 | 64 |
| | Processors per node | 2 | 2 | 2 |
| | Max. instruction set supported | AVX-512 | AVX-512 | AVX-2 |
| GPU | GPUs name | AMD Instinct™ MI50 | Nvidia Tesla™ V100 | AMD Instinct™ MI100 |
| | GPUs per node | 4 | 4 | 8 |

### Initial results

- Three different clusters
- Comparison not straightforward
  - Different CPU, Network
  - Different software stack & Linux distr.

**Bifurcation geom.** ($\sim 2 \times 10^6$ sites)

| MPI ranks | GPUs | V100 (time[s]/speedup) | MI50 (time[s]/speedup) |
|---|---|---|---|
| 5 | 1 | 421,4 ; 1 | 442,2 ; 1 |
| 5 | 2 | 249,3 ; 1,96 | 238,6 ; 1,85 |
| 5 | 4 | 161,6 ; 2,61 | 153,7 ; 2,87 |

**Legs' Arteries geom.** ($\sim 66 \times 10^6$ sites)

| Nodes | MPI ranks | GPUs | MI100 | V100 |
|---|---|---|---|---|
| 1 | 5 | 4 | 404s (x1) | 756s (x1) |
| 2 | 9 | 8 | 288s (x1,6) | 706s (x1,07) |
| 4 | 17 | 16 | 207s (x2,9) | 583s (x1,3) |

- **Test the "hipified" code on SUMMIT Vs CUDA code**
  target the same hardware (NVIDIA V100 GPUs)

→ **Similar performance of the codes**

# 4. Porting the CUDA version of HemeLB_GPU to Intel's oneAPI

## What is oneAPI

"The Intel® oneAPI Base Toolkit (Base Kit) is a core set of tools and libraries for developing high-performance, data-centric applications across diverse architectures. It features an industry-leading **C++ compiler that implements SYCL**, an evolution of C++ for heterogeneous computing."

## Porting Process: CUDA to DPC++ code

- DPC++ Compatibility tool (dpct)


- Produces human readable DPC++ code
  - with inline comments
  - 80-90% of the code transformed

```
/*
DPCT1003:40: Migrated API does not return error code. (*, 0) is inserted. You
may need to rewrite this code.
*/
cudaStatus = (q_ct1
                  .memcpy(GPUDataAddr_dbl_fOld_b, Data_dbl_fOld_b,
                          nArray_Distr * sizeof(double))
                  .wait(),
              0);
```

- Compile the produced DPC++ code

  clang++ -fsycl **-fsycl-targets=nvptx64-nvidia-cuda** … specify target hardware

# Porting HemeLB for human-scale blood flow simulation on GPUs

## 4. Porting the CUDA version of HemeLB_GPU to Intel's oneAPI

### Porting HemeLB_GPU

- Single HemeLB collision-streaming CUDA kernel to DPC++

- Manually fix errors

```
dpct::constant_memory<double, 0> dev_minusInvTau;
dpct::constant_memory<int, 1> _InvDirections_19(19);
// __constant__ double EQMWEIGHTS_19[19];
dpct::constant_memory<double, 1> _EQMWEIGHTS_19(19);
```

**Ported kernel**

```
cgh.parallel_for(
    sycl::nd_range<3>(sycl::range<3>(1, 1, nBlocks_Collide) *
                          nThreads_Collide,
                      nThreads_Collide),
    [=](sycl::nd_item<3> item_ct1) {
      GPU_CollideStream_mMidFluidCollision_mWallCollision_sBB(
          GPUDataAddr_dbl_fOld_b_ct0, GPUDataAddr_dbl_fNew_b_ct1,
          GPUDataAddr_dbl_MacroVars_ct2, GPUDataAddr_int64_Neigh_d_ct3,
          GPUDataAddr_uint32_Wall_ct4, nFluid_nodes_ct5, first_Index,
          (first_Index + site_Count_MidFluid),
          (first_Index + site_Count_MidFluid),
          (first_Index + site_Count), totalSharedFs_ct10,
          Write_GlobalMem, item_ct1, stream_ct1, *_NUMVECTORS_ptr_ct1,
          *dev_minusInvTau_ptr_ct1, _EQMWEIGHTS_19_ptr_ct1,
          _CX_19_ptr_ct1, _CY_19_ptr_ct1, _CZ_19_ptr_ct1);
    });
}); // (int64_t*)GPUDataAddr_int64_Neigh_b
```

**CUDA kernel**

```
GPU_CollideStream_mMidFluidCollision_mWallCollision_sBB <<<nBlocks_Collide, nThreads_Collide>>>
                                                        (double*)GPUDataAddr_dbl_fNew_b,
                                                        (double*)GPUDataAddr_dbl_MacroVars,
                                                        (int64_t*)GPUDataAddr_int64_Neigh_d,
                                                        (uint32_t*)GPUDataAddr_uint32_Wall,
                                                        nFluid_nodes,
                                                        first_Index, (first_Index + site_Count_MidFluid),
                                                        (first_Index + site_Count_MidFluid), (first_Index + site
```

- Testing the CUDA and ported codes on CSD3@Cambridge & Intel P630 GPU on Intel Dev-Cloud

| Hardware | Code | Average Runtime [s] |
|---|---|---|
| NVIDIA A100 | CUDA | 0.398578 +/- 0.00020542 |
| NVIDIA A100 | DPC++ | 0.436205 +/- 0.00793029 |
| Intel P630 | DPC++ | 15.4501 +/- 0.0939887 |

$10^5$ sites domain - $5 \times 10^3$ iterations

# 4. Porting the CUDA version of HemeLB_GPU to Intel's oneAPI

**Porting HemeLB_GPU**

- **Full CUDA HemeLB_GPU code (ongoing work)**
  a. Generate a .json file
     For projects using Make or CMake commands this will contain the build options of the input project's files, i.e. include path and macros definitions ("intercept-build make")

  b. Run compatibility tool (dpct) with the .json file

```
dpct -p build/compile_commands.json --in-root=. --out-root=../src_dpct_output --keep-original-code --process-all
```

  c. Manually fix any errors during conversion

  d. Compile (recover the compilation commands from the Makefile (e.g. make V=1)

# Porting HemeLB for human-scale blood flow simulation on GPUs

## Summary

- **HemeLB** is a numerical code, based on the **lattice Boltzmann Method (LBM)** for simulating blood flow within human-scale vasculature domains.
- **HemeLB_GPU: GPU** accelerated version using **CUDA** (NVIDIA GPUs).
- **Highly scalable**: CPU and GPU versions demonstrate **excellent strong scaling performance** to hundreds of thousands of CPU cores and tens of thousands of NVIDIA GPUs.
- At the arrival of exascale machines we will continue to develop HemeLB.
- Making **HemeLB_GPU platform agnostic -** Porting **HemeLB_GPU to HIP and oneAPI.**
- Aim for best performance on the widest range of machines (GPU accelerated HPC platforms) .

- More information from the **HemeLB website www.hemelb.org**
- **Repository link:**
  - **CPU code**: https://github.com/UCL-CCS/HemePure
  - **GPU code**: https://github.com/UCL-CCS/HemePure-GPU

# Acknowledgements

**We acknowledge:**

1. **Funding support from:**
   (a) the **European Commission CompBioMed Centre of Excellence**
   (b) The **UK Engineering and Physical Sciences Research Council**
   (c) the **Medical Research Council (MRC)**
   (d) special funding from the **UCL Provost**.
2. **The Gauss Centre for Supercomputing e.V.** (www.gauss-centre.eu) for providing computing time on the **GCS Supercomputer SuperMUC-NG** at Leibniz Supercomputing Centre (www.lrz.de)
3. **PRACE** for awarding us access to **JUWELS at GCS@FZJ**, Germany and **Piz Daint at CSCS**, Switzerland.
4. The **Oak Ridge Leadership Computing Facility** at the **Oak Ridge National Laboratory,** USA, for access to **SUMMIT**.