Optimizing Three-Dimensional Stencil-Operations on Heterogeneous Computing Environments

Nina Herrmann[1*†], Justus Dieckmann[1†] and Herbert Kuchen[1]

[1*]Practical Computer Science, University of Münster, Leonardo-Campus 3, Münster, 48149, Germany.

*Corresponding author(s). E-mail(s): nina.herrmann@uni-muenster.de;
Contributing authors: justus.dieckmann@uni-muenster.de;
kuchen@uni-muenster.de;
†These authors contributed equally to this work.

Abstract

Complex algorithms and enormous data sets require parallel execution of programs to attain results in a reasonable amount of time. Both aspects are combined in the domain of three-dimensional stencil operations, for example, computational fluid dynamics. This work contributes to the research on high-level parallel programming by discussing the generalizable implementation of a three-dimensional stencil skeleton that works in heterogeneous computing environments. Two exemplary programs, a gas simulation with the Lattice Boltzmann method, and a mean blur, are executed in a multi-node multi-graphics processing units (GPUs) environment, proving the runtime improvements in heterogeneous computing environments.

Keywords: Skeleton Programming, Three-Dimensional Stencil Operations

1 Introduction

The field of High Performance Computing (HPC) is growing as algorithms become more complex and more data is available. Evaluating massive datasets, therefore, requires writing efficient parallel programs. Most HPC environments have multiple nodes equipped with multiple central processing units (CPUs) and GPUs. Creating programs which combine multiple types of hardware requires knowledge of low-level frameworks such as Message Passing Interface (MPI) [1], OpenMP [2], and CUDA [3].

Writing a parallel program is error prone and tedious as e.g. out of memory errors, and invalid memory accesses are troublesome to identify even for skilled programmers. Moreover, choosing memory spaces, distributing data, and assigning task to threads are design decision which have a high impact on performance but require experience. Writing a program which combines multiple frameworks overstrains scientist.

Since experts in this field are hard to find, high-level frameworks are often used. Those frameworks commonly abstract from the distribution of data, provide portable code for different hardware architecture, are adjustable to distinct accelerators, and require less maintainance for the end-user. In 1989 Cole introduced algorithmic skeletons, enclosing reoccurring parallel and distributed computing patterns, such as Map and Reduce as one of the most common approaches to abstract from low-level details [4]. Multiple libraries [5, 6], general frameworks [7, 8], and and domain-specific languages (DSLs)[9] utilize the concept.

The paper contributes to the ongoing work by focusing on a particularly arduous operation, namely three-dimensional stencil operations. Stencil operations calculate elements depending on other values inside the data structure and therefore require communication between the computational units used. Those operations are irreplaceable, for e.g., simulation of gas or computational fluid dynamics. Efficiently updating data in a generalizable way and dealing with 3D data structures are obstacles not solved in current high-level approaches.

This paper firstly elaborates on the related work, focusing on high-level approaches abstracting from problem specific details (Section 2). Section 3 outlines the library used (muesli), while Section 4 explains the additional implementation of the three-dimensional skeleton. The work is evaluated in Section 5 discusing our runtime experiments on multiple hardware set-ups. Lastly, Section 7 summarizes out work.

2 Related Wok

Ongoing work discussing three-dimensional stencil operations is twofold. On the one hand, there exist generic high-level frameworks which have the advantage of parallelizing pre- and postprocessing steps, as many of them offer a variety of operations/skeletons. On the other hand, specialized frameworks already contain algorithms' implementations but are often inefficient. Most related regarding high-level skeleton programming, SkePU3 targets multi-node and multi-GPU environments for most skeletons in combination with StarPU. However, for stencil operations (MapOverlap), the exchange of data between the programs is missing for multi-node programs [7]. FastFlow added GPU support but focuses on communication skeletons and misses a comparable stencil operation [10][8].

Specialized libraries such as Palabos for the Lattice Boltzmann methods (LBMs) [11], or publications discussing a single method, e.g., the Helmholtz equation [12] focus rather on the algorithm and do not include accelerators as GPUs, which provide a significant speed-up.

This work extends the mentioned work as the presented stencil skeleton is generalizable for multiple methods and abstracts from the multiple layers of parallelism.

This is profen by showing the implementation of a LBM and a three-dimensional mean blur. Both programs run on multiple nodes and with multiple GPUs.

3 The *Mue*nster *S*keleton *Li*brary *Muesli*

Skeleton Programming is mostly present in functional languages since it originates from functional languages [4]. Most of the frameworks are implemented in C/C++ [6–8, 13–15], as it offers interoperability with multiple parallel frameworks such as OpenMP, MPI, CUDA, and OpenCL and is exceptionally performant. Noteworthy, Python recently gained attention for natural science applications as it provides a easy interface to write packages in C/C++ which can provide roughly the same performance as C/C++ programs. However, those packages are very specialized as custom user can only be passed with a overhead decreasing the runtime. Therefore, especially in the HPC environment C/C++ is the first choice.

The used library is called Muenster Skeleton Library (Muesli). Muesli provides an object-oriented approach offering one, two and three-dimensional datastructures (DA, DM, DC) which call skeletons as member functions. The supported skeletons are for example multiple version of Map and Zip (index and inplace variants), Fold, Gather, and as discussed in this work MapStencil. Internally, , MPI, OpenMP, and CUDA are used, which enables simultaneously parallelism on multiple nodes, CPUs, annd GPUs. The library can be included with a simple include statement `#include<muesli.h>`. For writing a parallel program, muesli provides abstract methods to state the number of processes and GPUs used. Apart from that Muesli abstracts from parallel programming details by internally distributing the datastructures on the available computational units, choosing the number of threads started on the corresponding low-level framework and copying data to the correct memory spaces. This abstraction has the additional effect to reduce errors which are commonly made by inexperienced programmers such as race conditions and inefficient data distribution.

Listing 1 shows a simple program calculating the Scalar product of the distributed arrays a and b. In line 6 a distributed array of size three with a default value of 2 is created. In the skeleton calls in line 7-9 can be seen that skeletons have a user function as an argument which can either be a C++ function or a C++ functor. For the `index` variant in line 7 Muesli complements the missing arguments internally. For the `zip` skeleton the second required data structure is also passed as an argument. Lastly, l.7+9 show das function can be used in diffrent context, firstly for calculating the sum of the index and the value and secondly as a reduction operator.

Listing 1: Scalar product in Muesli

```
1 class Sum : public Functor2<int, int, int>{
2   public: MSL_USERFUNC int operator() (int x, int y)
3   const {return x+y;}};
4 Sum sum;
5 auto product = [] (int i, int j) {return i*j;};
6 DA<int> a(3,2);                    // delivers: {2,2,2}
7 DA<int> b = a.mapIndex(sum);       // delivers: {2,3,4}
```

```
a.zipInPlace(b,product);             // delivers: {4,6,8}
int scalarproduct = a.fold(sum);     // delivers: 18
```

4 Three-Dimensional Stencil Operations

Stencil operations are map operation which additionally require to read the surrounding elements of the datastructure. Figure 1 displays a two-dimensional stencil with the size one. The peculiarity regarding stencil operations on multiple nodes and accelerators is that each execution of the stencil operation requires to update elements which are shared between computational units. As communication of updated elements requires synchronization between the computational nodes it decreases the opportunity for executing task in parallel within the program. Muesli abstracts from all communication between the computational nodes with a MapStencil Skeleton. The usage of the skeleton for the end-user is shown in Listing 2.

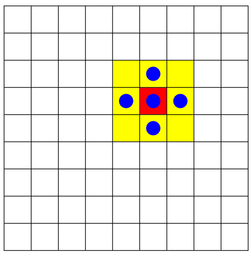
Fig. 1: Stencil Operation

4.1 Using the MapStencil Skeleton

Firstly, a function which is executed on each element is defined (l. 1-12). Merely functors of type `DCMapStencilFunctor` are permitted to be used with the `mapStencil` skeleton. Therefore, the first argument of the functor has to of type `PLCube` (*Padded-LocalCube*), and the following arguments have to be integers for indexing the data structure. The class `PLCube` most importantly offers a getter taking three index arguments, relieving the end-user from index calculations (l.9). The presented functor calculates the sum of all elements with a radius of two and divides the sum with the number of total elements, therefore calculating a mean blur. This functor can be applied to a distributed cube by calling the mapStencil skeleton as a member function (l.19). The skeleton takes the functor as a template argument, and requires a distributed cube of the same dimension with the current data [1], the radius of the stencil[2] and the neutral value for border elements.

Listing 2: Exemplary Functor for Stencil Skeleton

```
1  MSL_USERFUNC float update(const PLCube<float> &plCube,
2                     int x, int y, int z) {
3    float res = 0;
4    const int radius = 2;
5    const int elements = 48;
6    for (int mx = x - radius; mx <= x + radius; mx++) {
7      for (int my = y - radius; my <= y + radius; my++) {
```

[1]A variant of the skeleton which immediately overwrites old values by new values is however possible and could be applied, for instance, for implementing the Gauß-Seidel method for solving systems of linear equations.

[2]Other stencil shapes such as rectangular or irregular stencils can be handled by using the smallest surrounding square, although this may introduce some overhead.

```
      for (int mz = z - radius; mz <= z + radius; mz++) {
        res += plCube(mx, my, mz);
      }
    }
  }
  return res/(elements);
}
...
main () {
  ...
  int stencilradius = 2;
  dcp1->mapStencil<update>(*dcp2, stencilradius, 0);
  ...
}
```

4.2 Implementation of the MapStencil Skeleton

Adding the MapStencil skeleton to the existing distributed cube (DC) class requires to add two additional parameters: a vector of `PLCubes` and a supported stencil size. As previously mentioned the `PLCubes` class serves for the end-user to abstract from indexing of the datastructure. To make the access to the different memory spaces efficient each computational unit has a separate `PLCubes` storing merely the elements needed for the calculations of the assigned elements. This design choice makes the class flexible to be used for CPUs as well as for GPUs. It contains the following attributes to provide a light, minimal design:

- `int width, height, depth` the three dimensions of the datastructure,
- `int stencilSize` size of the Stencil required to calculate the overlapping elements,
- `int neutralValue` used when the index is outside the data structure,
- `T * data, T* topPadding, T* bottomPadding` CPU or GPU pointer for current data,
- four integers to save global indexes for start and end of datastructure and stencil size.

Most importantly, the index operator is implemented, taking three integers as arguments returning suitable value. This is either the neutral value or the corresponding element from the CPU or GPU memory.

Assuming GPUs are used a s accelerators, the skeleton updates the current datastructure in case the data is not up to date (Listing 3 l.6). Afterwards, it synchronizes the `PLCubes` inside one node, and the data between multiple nodes (l.7, l.9). Foreach GPU used the mapStencilKernelDC is called executing the functor on the appropiate part of the overall datastructure. In any other case (multiple nodes and cpu) it is only necessary to synchronize the nodes (l. 21), and thereafter call the functor with the corresponding arguments.

Listing 3: Implementation of MapStencil Skeleton

```
template<typename T>
template<msl::DCMapStencilFunctor<T> f>
void msl::DC<T>::mapStencil(msl::DC<T> &result, size_t stencilSize,
              T neutralValue) {
  #ifdef __CUDACC__
  this->updateDevice();
  syncPLCubes(stencilSize, neutralValue);
  msl::syncStreams();
  syncPLCubesMPI(stencilSize);
  for (int i = 0; i < this->ng; i++) {
    cudaSetDevice(i);
    dim3 dimBlock(Muesli::threads_per_block);
    dim3 dimGrid((this->plans[i].size + dimBlock.x - 1) / dimBlock.x);
    detail::mapStencilKernelDC<T, f><<<dimGrid, dimBlock, 0,
      Muesli::streams[i]>>>(result.plans[i].d_Data, this->plCubes[i],
      result.plans[i].size);
  }
  msl::syncStreams();
  result.setCpuMemoryInSync(false);
  #else
  syncPLCubesMPI(stencilSize);
  #ifdef _OPENMP
  #pragma omp parallel for
  #endif
  for (int k = 0; k < this->nLocal; k++) {
    int l = (k + this->firstIndex) / (ncol*nrow);
    int j = ((k + this->firstIndex) - l*(ncol*nrow)) / ncol;
    int i = (k + this->firstIndex) % ncol;
    result.localPartition[k] = f(this->plCubes[0], i, j, l);
  }
  #endif
}
```

4.3 Example Application for Three-Dimensionl Stencil Operations

For the evaluation of our implementation two examples where used: a LBM implementation and a mean blur. The exeamplary user function of the mean blur was already shown in Listing 2. However, the implementation of a LBM underlines the applicability for real application contexts.

LBMs are used for fluid simulations. It distinguishes between the collision and the streaming step which alternate in continuous simulations [16, p.61ff]. In the streaming step particles move from one cell to another. In the collision step the fluid flow caused by the colliding particles is calculated. The distribution function $f_i(x,t)$ calculates for a cell x and a timestamp t how many particles move in the next step to the neighbour

i. Zero is the cell itself.

$$f_i(x + c_i\Delta t, t + \Delta t) := f_i^*(x, t) \tag{1}$$

For the collision steps the Bhatnagar-Gross-Krook-operator is used. f_i^* defines the distribution after the collission of the particles, Δt the time period to be simulated and τ a constant defining the convergence of the simulation. Thus τ influences the viscosity of the gases.

$$f_i^*(x, t) := f_i(x, t) - \frac{\Delta t}{\tau}\left(f_i(x, t) - f_i^{\mathrm{eq}}(x, t)\right). \tag{2}$$

The equilibrium state is calculated by

$$f_i^{\mathrm{eq}}(x, t) := w_i\rho\left(1 + \frac{u \cdot c_i}{c_s^2} + \frac{u \cdot c_i}{2c_s^4} + \frac{u \cdot u}{2c_s^2}\right), \tag{3}$$

where w_i are the weights of the choosen grid and c_i is the position of the neighbour cells relative to the main cell. The constant number c_s is the sound velocity of the model. The mass density ρ and the puls density u are defined by

$$\rho(x, t) = \sum_i f_i(x, t), \qquad \rho u(x, t) := \sum_i c_i f_i(x, t). \tag{4}$$

For the implementation of the LBM a D3Q19-Grid was used, D being the number of dimensions and Q the number of neighbours. Both steps (collision and streaming) are combined in one `mapStencil` call. Noteworthy, the implementation has to take into consideration that single cells can be marked as blocked, simulating objects which are barriers for the flow of air or as distributing constant velocity. Therefore, special cells are marked with `Not a Number` values (Listing 5 l. 5-7). To simulate this behaviour without requiring additional storage, the handling of the floating numbers is extended. According to the IEEE-754 Standard each float has a maximal exponent with a mantisse which is not equal to zero. To identify special cells the most significant bit of the mantissa of f0 is set, so that the number is definitely understood as a `NaN`. The remaining bits of the mantissa can then be used freely to store other data. In the code, bit masks and a struct with bit- fields are defined in the code to access this information as easily as possible (Listing 4).

Listing 4: Handling of Barriers and Streaming Cells

```
const int FLAG_OBSTACLE = 1 << 0;
const int FLAG_KEEP_VELOCITY = 1 << 1;
typedef struct {
  unsigned int mantissa : 23;
  unsigned int exponent : 8;
  unsigned int sign : 1;
} floatparts ;
```

The data stored for each cell is an `array<float, Q>`. Q is a constant number for the neighbour cells an the cell itself (19). This type is appreviated in the following listing with `cell_t`. Moreover, it is abstracted from the three-dimensional vector operations (l. 28,29,31,34). The user function starts by transforming the current value of the cell into the single float parts (l.4). In case we have a cell which distributes gas (`FLAG_KEEP_VELOCITY`), the cell remains without changes (l.5-7). For all neighbour cells the current amount of paricles is read (l.10-12). In the collision step all cells which are obstacles reverse the flow of air (l.16-23). All other cells calculate the particles streaming from the next cells.

Listing 5: Implementation of a Exemplary LBM User Function

```
 1 MSL_USERFUNC cell_t update(const PLCube<cell_t> &plCube, int x,
 2                            int y, int z) {
 3   cell_t cell = plCube(x, y, z);
 4   auto* parts = (floatparts*) &cell[0];
 5   if (parts->exponent == 255 && parts->mantissa
 6       & FLAG_KEEP_VELOCITY) {
 7     return cell;
 8   }
 9   // Streaming.
10   for (int i = 1; i < Q; i++) {
11     cell[i] = plCube(x + (int) offsets[i].x,
12         y + (int) offsets[i].y, z + (int) offsets[i].z)[i];
13   }
14
15   // Collision.
16   if (parts->exponent == 255 && parts->mantissa & FLAG_OBSTACLE) {
17     if (parts->mantissa & FLAG_OBSTACLE) {
18       cell_t cell2 = cell;
19       for (size_t i = 1; i < Q; i++) {
20         cell[i] = cell2[opposite[i]];
21       }
22     }
23     return cell;
24   }
25   float p = 0;
26   vec3f vp {0, 0, 0};
27   for (size_t i = 0; i < Q; i++) {
28     p += cell[i];
29     vp += offsets[i] * cellwidth * cell[i];
30   }
31   vec3f v = p == 0 ? vp : vp * (1 / p);
32
33   for (size_t i = 0; i < Q; i++) {
34     cell[i] = cell[i] + deltaT / tau * (feq(i, p, v) - cell[i]);
```

```
  }
  return cell;
}
```

5 Evaluation

Our approach requires to measure the speedup achieved. For this purpose the presented exemplary programs, a mean blur and a LBM implementation were executed on the HPC machine Palma II[3]. Table 1 list the hardware specification of the partitions used. Those include two GPU-partitions and one CPU-partition.

The gpu2080 partition is equiped with 5 nodes each with 8 GeForce RTX 2080 Ti GPUs. Complementary we used the gpuhgx partition equipped with 2 Nodes each with 8 A100 SXM GPUs. One major advantage of the Nvidia A100 SXM is that one GPU has significantly more memory (80GB in contrast to 11GB) and more cores (6912 compared to 612). Therefore, experimenting with two partitions expands the research to proove the generalizability on varying GPUs. For testing CPU-parallelization the zen2 partition equiped with 12 nodes with each one Zen2 (EPYC 7742) CPUs with 64 cores. To provide generalizable results, the mean runtime of 20 execution was used. For running the sequential version on the HPC, a single Broadwell (E5-2683 v4) CPU was used. Two classification numbers are particularly important for the evaluation of the programs depending on the hardware, the maximum storage and the number of cores. As can be seen in the last two columns of Table 1 the GPU partitions vary. Therefore, experiments with the gpuhgx partition could be executed with bigger data structures, and provide a better speed-up.

5.1 LBM Experiment

The LBM is used to simulate the flow of fluid in the three dimensional space. Consequently, it is reasonable to run a experiment which does not only execute the mapSkeleton one time, but has multiple iterations simulating multiple dispersion steps. For every experiment 200 iteration were chosen to make runtimes comparable between different data sizes.

Datasizes were chosen by exceeding the available storage. For the LBM each cell requires 76 bytes, as each cell stores 19 32-bit floating point numbers. For the calculation a data structure to read and one to write is necessary. The required space for

[3]https://confluence.uni-muenster.de/pages/viewpage.action?pageId=27755336

Identifier	Nodes	GPU/CPU-type	Per Node		per computational unit	
			GPUs	CPUs	storage	cores
gpu2080	5	GeForce RTX 2080 Ti	8	1	11GB	612
gpuhgx	2	Nvidia A100 SXM	8	1	80GB	6912
zen2	12	Zen2 (EPYC 7742)	-	128	496GB	64

Table 1: Overview of used Hardware

the program can be simply calculated by:

$$d(\text{gb}) := \sqrt[3]{\text{gb} \cdot \frac{2^{30}}{2 \cdot 76}}$$

This result for the RTX 2080 Ti GPU in a maximum side length of 426, for the A100 SXM in a side length of 826. Although the CPU partition would support bigger data structures the datasizes was not increased to the maximum, as the speed-up converged, and the run-time of the sequential program became unreasonable high (10 hours for calculating the LBM simulation for a datasize of 960^3). Table 5 shows that for the CPU-zen2 partition a speed-up of 116 can be reached. As the partition has 64 cores, this effect is reached by multithreading, in total 128 threads were started on the 64 cores available. In this scenario it should be considered that the calculations are easy to execute in parallel as all data resides on the memory of the single CPU or GPU, acccessible for all threads. In contrast, the GeForce RTX 2080 Ti has 612 cores allowing to execute threads in parallel.

reasoning what is the maximum of threads which can be executed? what slows the application down?

To assure that a low-level program is not significantly faster, a native implementation was programmed to be compared againt the Muesli program. Figure 2. As can be seen, the implementations are close to each other. In contrast to the native implementation Muesli has a slight overhead, however, as it is very small this difference is not important. Runtimes for bigger data structures are not included for one GPU and two GPUs to increase the readbility of the graph. For two GPUs a speed-up of 1.7 compared to the single GPU version can be achieved and for four GPUs a speed-up of 2.75 is achieved. To assure that the overhead is caused by the communication the time for the update function was measured separatly. Without the communication a speed-up of 1.94 and 3.88 was achieved which can be subscribed to the synchronization of streams. However, using multiple GPUs also has the advantage of processing bigger data structures. Taking multiple levels of parallelism into consideration the program is also capable to run on multiple nodes equiped with mutliple GPUs. Runtimes are depicted in Figure 3.

Datasize ^{3}	seq	cpu	Speed-up	gpuhgx	speed-up	gpu2080	speed-up
40	1.69	0.48	3.5	0.01	141.12	0.02	75.86
80	15.75	0.58	27.09	0.06	275.29	0.14	115.86
120	56	1.3	42.96	0.18	317.67	0.44	128.28
160	135.61	2.02	67.19	0.41	332.04	1.14	119.4
...							
440	2994.47	30.35	98.67	8.69	344.4	21.65	**138.29**
...							
800	18572.9	173.88	106.81	52.52	**353.6**	-	-
...							
960	34880.8	300.44	**116.1**	-	-	-	-

Table 2: Speed-up for the Parallel Implementation of the LBM Gas Simulation for Single GPU or CPU Programs

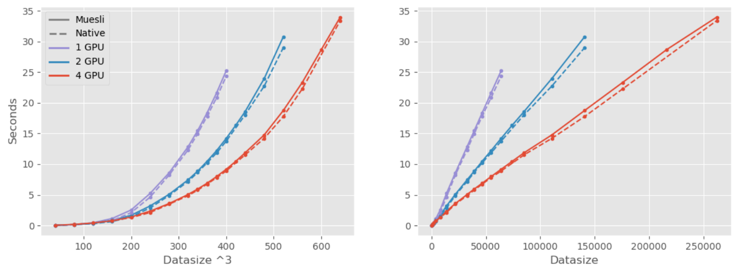

Fig. 2: Runtime comparison of a multi-GPU Muesli program and Native Implementation of the LBM on GeForce RTX 2080 Ti GPUs

Datasize ^{3}	GPUs	1 N	1 N-Com	2 Ns	2 Ns -Com	Speed-up	Speed-up
40	1	0.02	0.02	0.03	0.03	0.69	0.7
40	2	0.03	0.02	0.1	0.02	0.34	1.24
40	4	0.04	0.01	0.07	0.01	0.54	1
280	1	8.59	8.59	4.52	4.52	1.9	1.9
280	2	5.09	4.44	3.13	2.28	1.62	1.95
280	4	3.58	2.23	2.73	1.16	1.31	1.93
400	1	25.23	25.23	13.07	13.06	1.93	1.93
400	2	14.2	12.98	8.17	6.59	1.74	1.97
400	4	9.16	6.5	6.34	3.31	1.44	1.97

Table 3: Speed-up for the Parallel Implementation of the LBM Gas Simulation for Single GPU or CPU Programs

6 Blur Example

7 Conclusion

Equations in LaTeX can either be inline or on-a-line by itself ("display equations"). For inline equations use the `$...$` commands. E.g.: The equation $H\psi = E\psi$ is written via the command `$H \psi = E \psi$`.

For display equations (with auto generated equation numbers) one can use the equation or align environments:

$$\|\tilde{X}(k)\|^2 \leq \frac{\sum_{i=1}^{p} \left\|\tilde{Y}_i(k)\right\|^2 + \sum_{j=1}^{q} \left\|\tilde{Z}_j(k)\right\|^2}{p+q}. \tag{5}$$

where,

$$D_\mu = \partial_\mu - ig\frac{\lambda^a}{2}A^a_\mu$$

Datasize ^{3}	GPUs	1 N	1 N-Com	4 Ns	4 Ns -Com	Speed-up	Speed-up
40	1	0.02	0.02	0.04	0.01	0.56	2.7
40	2	0.03	0.02	0.08	0.02	0.44	1.29
40	4	0.04	0.01	0.07	0.01	0.55	0.99
280	1	8.59	8.59	2.74	2.16	3.14	3.97
280	2	5.09	4.44	2.63	1.19	1.93	3.74
280	4	3.58	2.23	2.06	0.39	1.73	5.74
400	1	25.23	25.23	7.21	6.32	3.5	3.99
400	2	14.2	12.98	6	3.35	2.37	3.87
400	4	9.16	6.5	5.14	1.7	1.78	3.82

Table 4: Speed-up for the Parallel Implementation of the LBM Gas Simulation for single GPU Programs

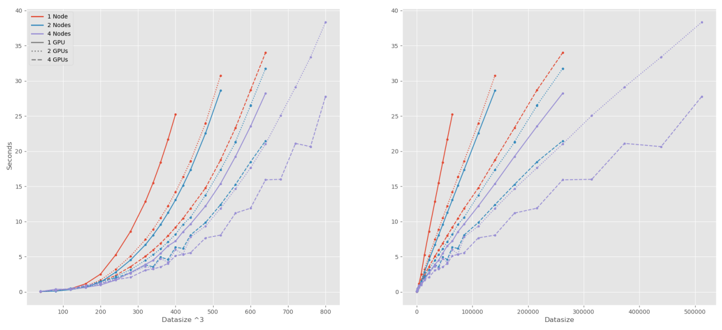

Fig. 3: Runtimes of the LBM Muesli program on multiple nodes and multiple GPUs

$$F_{\mu\nu}^a = \partial_\mu A_\nu^a - \partial_\nu A_\mu^a + g f^{abc} A_\mu^b A_\nu^a \tag{6}$$

Notice the use of `\nonumber` in the align environment at the end of each line, except the last, so as not to produce equation numbers on lines where no equation numbers are required. The `\label{}` command should only be used at the last line of an align environment where `\nonumber` is not used.

$$Y_\infty = \left(\frac{m}{\text{GeV}}\right)^{-3} \left[1 + \frac{3\ln(m/\text{GeV})}{15} + \frac{\ln(c_2/5)}{15}\right] \tag{7}$$

The class file also supports the use of `\mathbb{}`, `\mathscr{}` and `\mathcal{}` commands. As such `\mathbb{R}`, `\mathscr{R}` and `\mathcal{R}` produces $\mathbb{R}$, $\mathscr{R}$ and $\mathcal{R}$ respectively (refer Subsubsection **??**).

Tables can be inserted via the normal table and tabular environment. To put footnotes inside tables you should use `\footnotetext[]{...}` tag. The footnote appears

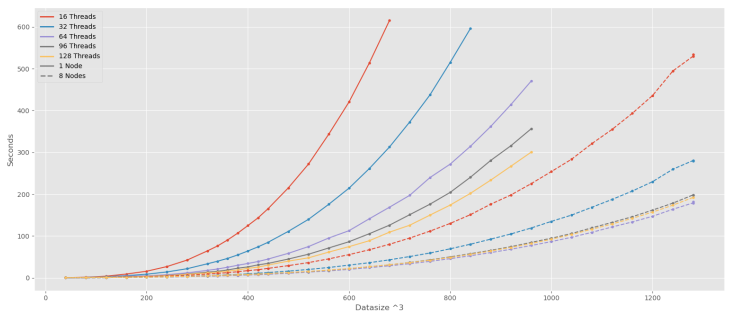

Fig. 4: CPU Runtimes of the LBM Muesli program

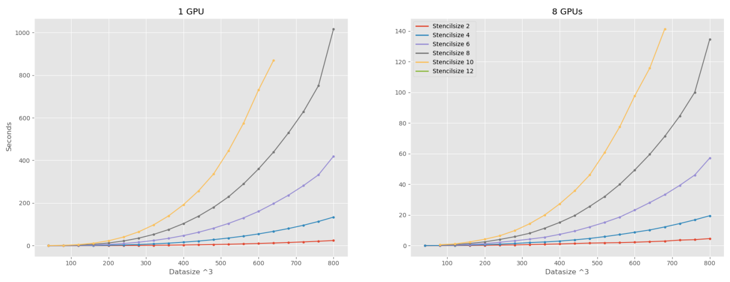

Fig. 5: Runtimes of the Blur Muesli program on A100 GPUs

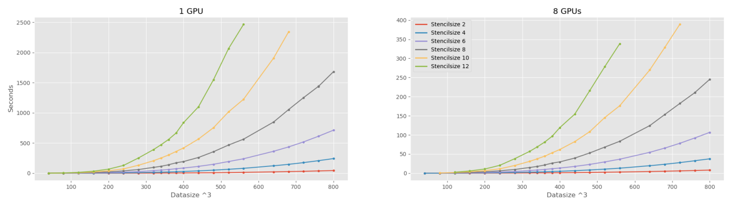

Fig. 6: Runtimes of the Blur Muesli program on a GeForce RTX 2080 Ti GPUs

just below the table itself (refer Tables 6 and 7). For the corresponding footnotemark use `\footnotemark[...]`
The input format for the above table is as follows:

```
\begin{table}[<placement-specifier>]
\caption{<table-caption>}\label{<table-label>}%
\begin{tabular}{@{}llll@{}}
\toprule
Column 1 & Column 2 & Column 3 & Column 4\\
```

blur	Datasize ^{3}	1 N	1 N-Com	4 Ns	4 Ns -Com	Speed-up	Speed-up	8 Ns	8 Ns -Com	Speed-up	Speed-up
2	120	0.16	0.16	0.07	0.05	2.1	3.41	0.09	0.03	1.76	5.56
	280	1.5	1.5	0.58	0.42	2.6	3.55	0.55	0.23	2.74	6.55
	400	4.46	4.46	1.44	1.15	3.09	3.89	1.24	0.61	3.61	7.33
	560	13.56	13.56	3.99	3.4	3.4	3.99	2.95	1.76	4.59	7.71
8	120	4.54	4.54	1.47	1.35	3.09	3.35	1.04	0.79	4.37	5.74
	280	60.58	60.58	16.44	15.85	3.68	3.82	9.71	8.47	6.24	7.15
	400	192.61	192.61	51.59	50.34	3.73	3.83	29.52	26.96	6.52	7.14
	560	563.2	563.2	148.68	146.41	3.79	3.85	83.38	78.58	6.75	7.17
12	120	13.59	13.59	4.2	4.03	3.23	3.37	2.53	2.17	5.38	6.27
	280	252.54	252.54	69.24	68.32	3.65	3.7	37.92	35.97	6.66	7.02
	400	836.59	836.58	222.12	220.38	3.77	3.8	118.89	115.15	7.04	7.27
	560	2464.76	2464.76	643.33	640.16	3.83	3.85	338.46	331.65	7.28	7.43

Table 5: Speed-up for the Parallel Implementation of the LBM Gas Simulation for single GPU Programs

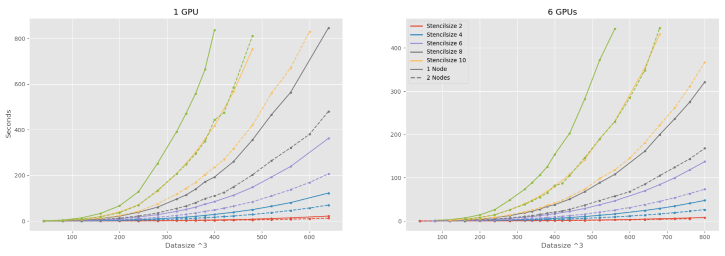

Fig. 7: Runtimes of the Blur Muesli program on multiple GeForce RTX 2080 Ti GPUs

Table 6: Caption text

Column 1	Column 2	Column 3	Column 4
row 1	data 1	data 2	data 3
row 2	data 4	data 5[1]	data 6
row 3	data 7	data 8	data 9[2]

Source: This is an example of table footnote. This is an example of table footnote.

[1]Example for a first table footnote. This is an example of table footnote.

[2]Example for a second table footnote. This is an example of table footnote.

```
\midrule
row 1 & data 1 & data 2  & data 3 \\
row 2 & data 4 & data 5\footnotemark[1] & data 6 \\
row 3 & data 7 & data 8  & data 9\footnotemark[2]\\
\botrule
\end{tabular}
\footnotetext{Source: This is an example of table footnote.
This is an example of table footnote.}
```

```
\footnotetext[1]{Example for a first table footnote.
This is an example of table footnote.}
\footnotetext[2]{Example for a second table footnote.
This is an example of table footnote.}
\end{table}
```

Table 7: Example of a lengthy table which is set to full textwidth

	Element 1[1]			Element 2[2]		
Project	Energy	σ_{calc}	σ_{expt}	Energy	σ_{calc}	σ_{expt}
Element 3	990 A	1168	1547 ± 12	780 A	1166	1239 ± 100
Element 4	500 A	961	922 ± 10	900 A	1268	1092 ± 40

Note: This is an example of table footnote. This is an example of table footnote this is an example of table footnote this is an example of table footnote this is an example of table footnote.

[1]Example for a first table footnote.

[2]Example for a second table footnote.

In case of double column layout, tables which do not fit in single column width should be set to full text width. For this, you need to use `\begin{table*} ... \end{table*}` instead of `\begin{table} ... \end{table}` environment. Lengthy tables which do not fit in textwidth should be set as rotated table. For this, you need to use `\begin{sidewaystable} ... \end{sidewaystable}` instead of `\begin{table*} ... \end{table*}` environment. This environment puts tables rotated to single column width. For tables rotated to double column width, use `\begin{sidewaystable*} ... \end{sidewaystable*}`.

As per the LaTeX standards you need to use eps images for LaTeX compilation and `pdf/jpg/png` images for `PDFLaTeX` compilation. This is one of the major difference between LaTeX and `PDFLaTeX`. Each image should be from a single input .eps/vector image file. Avoid using subfigures. The command for inserting images for LaTeX and `PDFLaTeX` can be generalized. The package used to insert images in `LaTeX/PDFLaTeX` is the graphicx package. Figures can be inserted via the normal figure environment as shown in the below example:

```
\begin{figure}[<placement-specifier>]
\centering
\includegraphics{<eps-file>}
\caption{<figure-caption>}\label{<figure-label>}
\end{figure}
```

In case of double column layout, the above format puts figure captions/images to single column width. To get spanned images, we need to provide `\begin{figure*} ... \end{figure*}`.

For sample purpose, we have included the width of images in the optional argument of `\includegraphics` tag. Please ignore this.

8 Algorithms, Program codes and Listings

Packages `algorithm`, `algorithmicx` and `algpseudocode` are used for setting algorithms in LaTeX using the format:

```
\begin{algorithm}
\caption{<alg-caption>}\label{<alg-label>}
\begin{algorithmic}[1]
. . .
\end{algorithmic}
\end{algorithm}
```

You may refer above listed package documentations for more details before setting `algorithm` environment. For program codes, the "verbatim" package is required and the command to be used is `\begin{verbatim} ... \end{verbatim}`.

Similarly, for `listings`, use the `listings` package. `\begin{lstlisting} ... \end{lstlisting}` is used to set environments similar to `verbatim` environment. Refer to the `lstlisting` package documentation for more details.

A fast exponentiation procedure:

Table 8: Tables which are too long to fit, should be written using the “sidewaystable” environment as shown here

	Element 1[1]			Element[2]		
Projectile	Energy	σ_{calc}	σ_{expt}	Energy	σ_{calc}	σ_{expt}
Element 3	990 A	1168	1547 ± 12	780 A	1166	1239 ± 100
Element 4	500 A	961	922 ± 10	900 A	1268	1092 ± 40
Element 5	990 A	1168	1547 ± 12	780 A	1166	1239 ± 100
Element 6	500 A	961	922 ± 10	900 A	1268	1092 ± 40

Note: This is an example of table footnote this is an example of table footnote this is an example of table footnote this is an example of table footnote this is an example of table footnote.

[1]This is an example of table footnote.

```
begin
  for $i := 1$ to 10 step 1 do
      expt(2, $i$);
      newline() od                                Comments will be set flush to the right margin
where
proc expt($x$, $n$) $\equiv$
  $z := 1$;
  do if $n = 0$ then exit fi;
     do if odd($n$) then exit fi;
        comment: This is a comment statement;
        $n := n/2$; $x := x * x$ od;
     { $n > 0$ };
     $n := n - 1$; $z := z * x$ od;
  print($z$).
end
```

Algorithm 1 Calculate $y = x^n$

Require: $n \geq 0 \vee x \neq 0$
Ensure: $y = x^n$

```
1: $y \Leftarrow 1$
2: if $n < 0$ then
3:     $X \Leftarrow 1/x$
4:     $N \Leftarrow -n$
5: else
6:     $X \Leftarrow x$
7:     $N \Leftarrow n$
8: end if
9: while $N \neq 0$ do
10:     if $N$ is even then
11:         $X \Leftarrow X \times X$
12:         $N \Leftarrow N/2$
13:     else[$N$ is odd]
14:         $y \Leftarrow y \times X$
15:         $N \Leftarrow N - 1$
16:     end if
17: end while
```

```
for i:=maxint to 0 do
begin
{ do nothing }
end;
Write('Case insensitive ');
Write('Pascal keywords.');
```

9 Cross referencing

Environments such as figure, table, equation and align can have a label declared via the `\label{#label}` command. For figures and table environments use the `\label{}` command inside or just below the `\caption{}` command. You can then use the `\ref{#label}` command to cross-reference them. As an example, consider the label declared for Figure 7 which is `\label{fig1}`. To cross-reference it, use the command `Figure \ref{fig1}`, for which it comes up as “Figure 7”.

To reference line numbers in an algorithm, consider the label declared for the line number 2 of Algorithm 1 is `\label{algln2}`. To cross-reference it, use the command `\ref{algln2}` for which it comes up as line 2 of Algorithm 1.

9.1 Details on reference citations

Standard LaTeX permits only numerical citations. To support both numerical and author-year citations this template uses `natbib` LaTeX package. For style guidance please refer to the template user manual.

Here is an example for `\cite{...}`: [1]. Another example for `\citep{...}`: [1]. For author-year citation mode, `\cite{...}` prints Jones et al. (1990) and `\citep{...}` prints (Jones et al., 1990).

All cited bib entries are printed at the end of this article: [1], [1], [1].

10 Examples for theorem like environments

For theorem like environments, we require `amsthm` package. There are three types of predefined theorem styles exists—`thmstyleone`, `thmstyletwo` and `thmstylethree`

`thmstyleone`	Numbered, theorem head in bold font and theorem text in italic style
`thmstyletwo`	Numbered, theorem head in roman font and theorem text in italic style
`thmstylethree`	Numbered, theorem head in bold font and theorem text in roman style

For mathematics journals, theorem styles can be included as shown in the following examples:

Theorem 1 (Theorem subhead)**.** *Example theorem text. Example theorem text. Example theorem text. Example theorem text. Example theorem text. Example theorem text. Example theorem text. Example theorem text. Example theorem text. Example theorem text. Example theorem text.*

Sample body text. Sample body text. Sample body text. Sample body text. Sample body text. Sample body text. Sample body text. Sample body text.

Proposition 2. *Example proposition text. Example proposition text. Example proposition text. Example proposition text. Example proposition text. Example proposition text. Example proposition text. Example proposition text. Example proposition text. Example proposition text.*

Sample body text. Sample body text. Sample body text. Sample body text. Sample body text. Sample body text. Sample body text. Sample body text.

Example 1. *Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem.*

Sample body text. Sample body text. Sample body text. Sample body text. Sample body text. Sample body text. Sample body text. Sample body text.

Remark 1. *Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem.*

Sample body text. Sample body text. Sample body text. Sample body text. Sample body text. Sample body text. Sample body text. Sample body text.

Definition 1 (Definition sub head)**.** *Example definition text. Example definition text. Example definition text. Example definition text. Example definition text. Example definition text. Example definition text. Example definition text.*

Additionally a predefined "proof" environment is available: `\begin{proof} ... \end{proof}`. This prints a "Proof" head in italic font style and the "body text" in roman font style with an open square at the end of each proof environment.

Proof. Example for proof text. Example for proof text. Example for proof text. Example for proof text. Example for proof text. Example for proof text. Example for proof text. Example for proof text. Example for proof text. Example for proof text. □

Sample body text. Sample body text. Sample body text. Sample body text. Sample body text. Sample body text. Sample body text. Sample body text.

Proof of Theorem 1. Example for proof text. Example for proof text. Example for proof text. Example for proof text. Example for proof text. Example for proof text. Example for proof text. Example for proof text. Example for proof text. Example for proof text. □

For a quote environment, use `\begin{quote}...\end{quote}`

> Quoted text example. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque.

Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Sample body text. Sample body text. Sample body text. Sample body text. Sample body text (refer Figure 7). Sample body text. Sample body text. Sample body text (refer Table 8).

11 Methods

Topical subheadings are allowed. Authors must ensure that their Methods section includes adequate experimental and characterization data necessary for others in the field to reproduce their work. Authors are encouraged to include RIIDs where appropriate.

Ethical approval declarations (only required where applicable) Any article reporting experiment/s carried out on (i) live vertebrate (or higher invertebrates), (ii) humans or (iii) human samples must include an unambiguous statement within the methods section that meets the following requirements:

1. Approval: a statement which confirms that all experimental protocols were approved by a named institutional and/or licensing committee. Please identify the approving body in the methods section
2. Accordance: a statement explicitly saying that the methods were carried out in accordance with the relevant guidelines and regulations
3. Informed consent (for experiments involving humans or human tissue samples): include a statement confirming that informed consent was obtained from all participants and/or their legal guardian/s

If your manuscript includes potentially identifying patient/participant information, or if it describes human transplantation research, or if it reports results of a clinical trial then additional information will be required. Please visit (https://www.nature.com/nature-research/editorial-policies) for Nature Portfolio journals, (https://www.springer.com/gp/authors-editors/journal-author/journal-author-helpdesk/publishing-ethics/14214) for Springer Nature journals, or (https://www.biomedcentral.com/getpublished/editorial-policies#ethics+and+consent) for BMC.

12 Discussion

Discussions should be brief and focused. In some disciplines use of Discussion or 'Conclusion' is interchangeable. It is not mandatory to use both. Some journals prefer a section 'Results and Discussion' followed by a section 'Conclusion'. Please refer to Journal-level guidance for any specific requirements.

13 Conclusion

Conclusions may be used to restate your hypothesis or research question, restate your major findings, explain the relevance and the added value of your work, highlight any limitations of your study, describe future directions for research and recommendations.

In some disciplines use of Discussion or 'Conclusion' is interchangeable. It is not mandatory to use both. Please refer to Journal-level guidance for any specific requirements.

Supplementary information. If your article has accompanying supplementary file/s please state so here.

Authors reporting data from electrophoretic gels and blots should supply the full unprocessed scans for key as part of their Supplementary information. This may be requested by the editorial team/s if it is missing.

Please refer to Journal-level guidance for any specific requirements.

Acknowledgments. Acknowledgments are not compulsory. Where included they should be brief. Grant or contribution numbers may be acknowledged.

Please refer to Journal-level guidance for any specific requirements.

Declarations

Some journals require declarations to be submitted in a standardised format. Please check the Instructions for Authors of the journal to which you are submitting to see if you need to complete this section. If yes, your manuscript must contain the following sections under the heading 'Declarations':

- Funding
- Conflict of interest/Competing interests (check journal-specific guidelines for which heading to use)
- Ethics approval
- Consent to participate
- Consent for publication
- Availability of data and materials
- Code availability
- Authors' contributions

If any of the sections are not relevant to your manuscript, please include the heading and write 'Not applicable' for that section.

Editorial Policies for:

Springer journals and proceedings: https://www.springer.com/gp/editorial-policies

Nature Portfolio journals: https://www.nature.com/nature-research/editorial-policies

Scientific Reports: https://www.nature.com/srep/journal-policies/editorial-policies

BMC journals: https://www.biomedcentral.com/getpublished/editorial-policies

Appendix A Section title of first appendix

An appendix contains supplementary information that is not an essential part of the text itself but which may be helpful in providing a more comprehensive understanding of the research problem or it is information that is too cumbersome to be included in the body of the paper.

References

[1] Forum, M.: MPI Standard. https://www.mpi-forum.org/docs/. Accessed: 10.05.2021 (2021)

[2] OpenMP: OpenMP The OpenMP API specification for parallel programming. https://www.openmp.org/. Accessed: 10.05.2021 (2021)

[3] Corporation, N.: CUDA. https://developer.nvidia.com/cuda-zone. Accessed: 10.05.2021 (2021)

[4] Cole, M.I.: Algorithmic Skeletons: Structured Management of Parallel Computation. Pitman London, ??? (1989)

[5] Ernsting, S., Kuchen, H.: Data parallel algorithmic skeletons with accelerator support. International Journal of Parallel Programming **45**(2), 283–299 (2017)

[6] Benoit, A., Cole, M., Gilmore, S., Hillston, J.: Flexible skeletal programming with eskel. In: European Conference on Parallel Processing, pp. 761–770 (2005). Springer

[7] Ernstsson, A., Ahlqvist, J., Zouzoula, S., Kessler, C.: Skepu 3: Portable high-level programming of heterogeneous systems and hpc clusters. International Journal of Parallel Programming **49**(6), 846–866 (2021)

[8] Aldinucci, M., Danelutto, M., Kilpatrick, P., Torquati, M.: Fastflow: high-level and efficient streaming on multi-core. Programming multi-core and many-core computing systems, parallel and distributed computing (2017)

[9] Wrede, F., Rieger, C., Kuchen, H.: Generation of high-performance code based on a domain-specific language for algorithmic skeletons. The Journal of Supercomputing **76**(7), 5098–5116 (2020)

[10] Goli, M., González-Vélez, H.: Heterogeneous algorithmic skeletons for fast flow with seamless coordination over hybrid architectures. In: 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, pp. 148–156 (2013). https://doi.org/10.1109/PDP.2013.29

[11] Latt, J., Malaspinas, O., Kontaxakis, D., Parmigiani, A., Lagrava, D., Brogi, F.,

Belgacem, M.B., Thorimbert, Y., Leclaire, S., Li, S., Marson, F., Lemus, J., Kotsalos, C., Conradin, R., Coreixas, C., Petkantchin, R., Raynaud, F., Beny, J., Chopard, B.: Palabos: Parallel lattice boltzmann solver. Computers and Mathematics with Applications **81**, 334–350 (2021) https://doi.org/10.1016/j.camwa.2020.03.022 . Development and Application of Open-source Software for Problems with Numerical PDEs

[12] Gonzales, R., Gryazin, Y., Lee, Y.T.: Parallel fft algorithms for high-order approximations on three-dimensional compact stencils. Parallel Computing **103**, 102757 (2021) https://doi.org/10.1016/j.parco.2021.102757

[13] Marques, R., Paulino, H., Alexandre, F., Medeiros, P.D.: Algorithmic skeleton framework for the orchestration of gpu computations. In: Wolf, F., Mohr, B., Mey, D. (eds.) Euro-Par 2013 Parallel Processing, pp. 874–885. Springer, Berlin, Heidelberg (2013)

[14] Alba, E., Luque, G., Garcia-Nieto, J., Ordonez, G., , G.L.: Mallba: a software library to design efficient optimisation algorithms. International Journal of Innovative Computing and Applications **1**(1), 74–85 (2007) https://doi.org/10.1504/IJICA.2007.013403

[15] Karasawa, Y., Iwasaki, H.: A parallel skeleton library for multi-core clusters. In: 2009 International Conference on Parallel Processing (2009)

[16] Krüger, T., Kusumaatmaja, H., Kuzmin, A., Shardt, O., Silva, G., Viggen, E.M.: The Lattice Boltzmann Method: Principles and Practice. Graduate Texts in Physics. Springer, Cham (2016)