Distributed Systems seminar
Analyzing PyFR

Andre Tättar

1. Abstract

I will be taking part of the ISC'15 Student Cluster competition in july. Our task is to build a cluster and run scientific programs on the cluster. One of the scientific software we have to run is PyFR. I am currently the only one working on PyFR in my team and I wanted to write a paper about what PyFR is and what it does. Also give quick summaries of my and other work done with PyFR. It is hard to understand PyFR without hard scientific background and I hope this paper will give some information about what PyFR is, how to install and how to run.

2. Brief background in physics for PyFR

Here are some definitions and examples:
Numerical method – Study of algorithms that use numerical approximation for the problems of mathematical analysis.
Fluid Dynamics – In physics, Fluid Dynamics is study of fluid flow – the natural science of fluids (liquids and gases) in motion. Aerodynamics (the study of air and other gases in motion) and hydrodynamics (the study of liquids in motion) are subfields of Fluid Dynamics. Fluid dynamics has a wide range of applications, for example forces and moments on aircraft, determining the mass flow rate of liquid through pipelines, predicting weather patterns, understanding nebulae in interstellar space and modeling nuclear weapon detonation.
Advection - Diffusion type problems – Problems that are combined of diffusion and convection(advection) equations. They describe physical phenomena where particles, energy, or other physical quantities are transferred inside a physical system due to processes of diffusion and convection.
Diffusion – movement of substance from a region of high concentration to region of low concentration. It is a very common thing in nature. Examples are: Smoke diffuses into air. Heat (energy) is diffused during heat conduction, such as a mug getting hot when a hot liquid is placed in it.
Advection – is a transport mechanism of a substance or conserver property by a fluid due to fluid's bulk motion. Also a very common thing in a nature. Examples are: If you throw a bag of chips in a river, then the bag will start moving the same way as river.
Navier-Stokes equations – Describe the motion of viscous fluid substance. It is used to model the weather, ocean currents, water flow in a pipe and air flow around a wing.
Flux – In Fluid Dynamics, flux is defined as the rate of flow of a property per unit area. For example, the magnitude of a river's current (amount of water that flows through a cross-section of the river each second).
Runge-Kutta methods – are an important family of implicit and explicit iterative methods in numerical analysis, which are used in temporal discretization for the approximation of solutions of ordinary differential equations.
Unstructured (Irregular) grid – is a tessellation of a part of the Euclidean space by simple shapes, such as triangles, tetrahedra, quadrilaterals or hexahedrals, in an irregular pattern. Grids of this type may be used in finite element analysis when the input to be analyzed has an irregular shape. Unline structured grids, unstructured grids require a list of the connectivity which specifies the way a given set of vertices make up individual elements (graph).
Euler Equations – It is based on many assumptions, but Euler's equation for steady flow of an ideal fluid along a streamline is a relation between the velocity, pressure and density of a moving fluid. It is based on Newton's Second Law of Motion. The integration of the equation gives Bernoulli's equation in the form of energy per unit weight of the following fluid.
Strong scalability – problem size is fixed, but you increase processor count and you look how much time it takes. Our goal is to minimize the time-to-solution. Excellent parallel speedup for 8 processors would be if it took 8 times less time for specific problem, this would give us a speedup of factor 8.
Weak scalability – you increase the problem size and processor size by a factor and you look up how much time it takes. Our goal is to achieve constant time-to-solution for larger problems. For example, if you had 1 processor and a problem size a and it would take time b seconds, then after getting 8 more processors, ideally you could get 8*b seconds of wall clock time for the problem size 8*a.

3. What is PyFr?

PyFr is being developed in the Vincent Lab, Department of Aeronautics, Imperial College, London. They are supported by Engineering and Physical Science Research Council and Airbus. Major hardware supporters Nvidia, Intel and AMD. PyFr is an open-source Python based framework for solving advection-diffusion type problems on streaming architectures using the Flux Reconstruction approach of Huynh [1]. The framework is designed to solve a range

of governing systems on mixed unstructured grids containing various element types. It has the potential, to solve any advection-diffusion problem. Supports various hardware platforms thanks to Mako templating engine. The current stable release (PyFr 0.3.0) has the follow capabilities:

- Governing equations – Euler, Navier Stokes
- Dimensionality – 2D, 3D
- Element types – Triangles, Quadrilaterals, Hexahedra, Prisms, Tetrahedra, Pyramids
- Platforms – CPU clusters, Nvidia GPU clusters, AMD GPU clusters
- Spatial discretisation – High-order flux reconstruction
- Temporal discretisation – explicit Runge-Kutta
- Precision – Single, Double
- Mesh files read – Gmsh (.msh)
- Solution files produced – Unstructured VTK (.vtu)

One note for platforms – PyFR approach exhibits a significant degree of element locality, and is thus able to run efficiently on modern streaming architectures such as GPUs.

4. Summary of PyFR paper, where authors used Nvidia GPUs

Without going into much detail I will firstly speak about what they did. They used upto 104 Nvidia M2090 GPUs. They discussed inner kernel, performance per node and scalability.

4.1. Inner Kernels

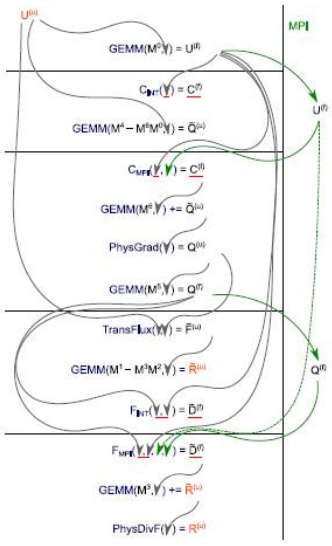

Figure 1. Inner kernels of PyFR. Source: [3]

Clearly from this picture we can see, that most of the time is spent doing GEMM, which is $O(n^3)$ complexity and takes the most time. Others don't take so much time. We can see that there is quite a lot of bandwidth also. Thanks to data locality, GEMM can be very efficient for all architectures. They measured performance of PyFR on an Nvidia M2090 GPU. It has theoretical double precision peak of 665 GFLOP/s and theoretical memory bandwidth of 177 GB/s (ECC disabled). For reference cuBlas obtained 407 GFLOP/s when multiplying square matrices of size 4096. The maximum device bandwidth they obtained by the bandwidth test application included with the CUDA SDK was 138,9 GiB/s (ECC disabled). So for this performance test they evaluated a fully periodic cuboidal domain with 50176 hexahedral elements. The double precision Navier-Stokes solver of PyFR was then run with different constants. In conducting the analysis kernels were grouped into on of three categories: DGEMM(matrix multiplications), point-wise kernels with direct memory access patterns(PD) and point-wise kernels with some level of indirect memory access (PI).

4.2. Single node performance

		Order		
		$p = 2$	$p = 3$	$p = 4$
Wall time/%				
	DGEMM	55,7	66,2	81,4
	PD	24,9	21,5	12,8
	PI	19,4	12,3	5,8
Bandwidth/GiB/s				
	PD	125,5	125,0	124,8
	PI	124,8	124,3	124,2
Arithmetic/GFLOP/s				
	DGEMM	205,3	368,1	305,4
	PD	0,7	0,7	0,7
	PI	0,9	0,8	0,9

Figure 2. Single node performance of PyFR, grouped by Inner kernels. Source: [3]

From this table, we can see, that most of the time spent is in DGEMM kernel. Also almost all of the aritmetic work is done in DGEMM kernel. So, that is a bottleneck and for fast PyFR code you will need fast DGEMM code/library. The amount of bandwith is divided into PI and PD quite evenly in terms of GiB/S, but for wall time, PD takes more % of time in general. If we look at the results, we can see that PyFR approaches quite high performance peaks compared to the reference results mentioned above. With $\gamma = 4$ we can see that DGEMM is 81,4 % of wall time and achieving more than 75% of peak and $\gamma = 2$ we can see that DGEMM is achieving more than 50% of peak, in conclusion the wall time is heavily dependant on libraries and with CUDA it achieves 50 to 90 % of reference peak and that is just by using libraries. For bandwidth we can see that PyFR achieves 90 % of efficiency for both kernels PI and PD.

4.3. Scalability

The scalability of PyFR has been evaluated on the Emerald GPU cluster, located at the STFC Rutherford Appleton

# M2090s	1	2	4	8	16	32	64	104
Runtime	1,00	1,00	1,01	1,01	1,01	1,01	1,01	1,01

Figure 3. Weak scaling of PyFR. Source: [3]

# M2090s	1	2	4	8	16	32
Speedup	1,00	2,03	3,96	7,48	14,07	26,18

Figure 4. Strong scaling of PyFR. Source: [3]

Laboratory. It has 60 HP SL390 nodes with 3 Nvidia M2090 GPUs and 24 HP SL390 nodes with eight Nvidia M2090 GPUs. Nodes are connected by QDR InfiniBand. This test was made using only those with 8 GPU nodes. Problem size was: 114688 structured hexahedral elements. It loaded 90 % of M2090 GPU memory. Like in tables 6 and 7, they measured weak and strong scaling. With weak scaling the working set had a size of 485 GiB. As we can see in table 6. PyFR has perfect weak scaling, the factor is 1,01, that means that this is very highly parallel code. Strong scaling is not so perfect, with 32 GPUs the speedup is 26 times, but that is to be expected, because in that case each GPU is loaded to less than 3% of maximum and so this result is to be expected. It is slower, because there is too much unnecessary communication between nodes. Communication between nodes is much slower than aritmetics. But we can still see, that it scales up fairly well, for example for 4 nodes the speedup is 3,96. One just has to make sure he does not do too much unnecessary communication.

4.4. Their Conclusions

They reported and showed that the kernels doing DGEMM are able to obtain between 50% and 90% reference peak FLOP's whereas the bandwidth bound point-wise kernels are able to obtain 90% of reference peak bandwith. Also PyFR code has good scaling, where they showed strong scaling up to 32 GPUs and weak scaling up to 104 GPUs.

5. Summary

I read through the paper about using PyFR with Nvidia GPU cluster. I installed and got PyFR to run on my pc and on one node of cluster. I read through PyFR own theory and guides provided on their webpage. I planned to get PyFR to run on an APU, but did not manage to do it yet. Didn't get ACML (AMD core math library) to work and couldn't test APU integrated graphics part. I did manage to get installation script to work and tested it. Now instead of having to spend 10 hours to install and looking at all the dependencies (I had a lot of errors while installing),with script I got it installed in less than 30 minutes. The script will be very useful in the competition, because we can get to code testing and performance measuring much faster. Right now it installs all python dependencies and pyfr itself, but does not provide OpenMP/CUDA/OpenCL backend, that is hardware architecture specific and users need to download that manually. I also looked over the code and concluded that python code has been well written and should not be changed. One just needs to pick correct and fast backend libraries, have fast interconnect and PyFR will run well in your cluster. I still need to test which libraries are fastest and test multiple nodes. That project will be coming out shortly. Also I hope I can test PyFR with OpenCL on one node and also look at scaling.

6. PyFR 0.3.0 installation script for linux Ubuntu

The dependancies are: Python 3.3 +, mako, mpi4py $\geq$ 1.3 , mpmath $\geq$ 0.18, numpy $\geq$ 1.8, pytools $\geq$ 2014.3. These dependencies also have some dependencies. So in addition you have to install Pip, setuptools, python3-dev, six, py, cython, hdf5 and mpich. To run PyFR 0.3.0 in parallel it is also necessary to have one of the following installed: Metis 5.0+ or scotch 6.0+ Cuda Backend targets Nvidia GPUs with a compute capability of 2.0 or greater. The backend requires: CUDA 4.2+ and pycuda 2011.2+ OpenCL Backend targets a range of accelerators including GPUs from AMD and NVIDIA. The backend requires: OpenCL and pyopencl 2013.2+ and clBLAS. OpenMP Backend targets multi-code CPUs. The backend requires GCC 4.7+ and A BLAS library compiled as a shared library (e.g. OpenBLAS) So my Installation script will install dependencies under the 1. bulletpoint and user has to download backend target software himself. I will write a backend script after I know which backend is most suitable for me.

Acknowledgments

I'd like to thank PyFR makers and the writers of paper I read, I have enjoyed looking at PyFR and the paper. Also thanks to Benson Muite, he was my supervisor. The biggest thanks goes to wonderful Annika Laumets, who helped with LaTeX.

7. Resources:

1 PyFr, http://www.pyfr.org/index.php [last accessed 8 June 2015]

2 Euler Equations, http://www.codecogs.com/library/engineering/fluid_mechanics/fundamentals/eulers-equation.php [last accessed 8 June 2015]

3 F. D. Witherden, A. M. Farrington, P. E. Vincent "PyFR: An open source framework for solving advection-diffusion type problems on streaming architectures using the flux reconstruction approach" Computer Physics Communications 185(11), pp. 3028-3040 (2014) http://dx.doi.org/10.1016/j.cpc.2014.07.011

4 Fluid Dynamics, http://en.wikipedia.org/wiki/Fluid_dynamics [last accessed 8 June 2015]

5 Navier-Stokes Equations, http://en.wikipedia.org/wiki/Navier%E2%80%93Stokes_equations [last accessed 8 June 2015]

6 Numerical Analysis, http://en.wikipedia.org/wiki/Numerical_analysis [last accessed 8 June 2015]
7 Diffusion equation, http://en.wikipedia.org/wiki/Convection%E2%80%93diffusion_equation [last accessed 8 June 2015]
8 Unstructured Grid, http://en.wikipedia.org/wiki/Unstructured_grid [last accessed 8 June 2015]
9 Diffusion, http://en.wikipedia.org/wiki/Diffusion [last accessed 8 June 2015]
10 Advection, http://en.wikipedia.org/wiki/Advection [last accessed 8 June 2015]
11 Flux, http://en.wikipedia.org/wiki/Flux [last accessed 8 June 2015]
12 Runge-Kutta methods, http://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods [last accessed 8 June 2015]
13 Weak and Strong scaling, http://www.hpcwire.com/2008/04/18/intel_ibm_speed_through_economic_slowdown-1/ [last accessed 8 June 2015]