Understanding and Optimizing Partial Differential Equations Solving, Namely Navier-Stokes Equations using GPU and Based on an Open-Source Nvidia® Project: OpenCurrent

To Model Pipeline Flow of Incompressible Oil Products for Leak Detection Purposes

Contents

1. <u>Problem Description</u>

1.1. Importance of Pipeline Flow Modeling

From now on, when we say pipeline modeling we refer to pipeline flow modeling. Similarly, a pipeline model is a pipeline flow model.

The Importance of pipeline modeling can summarized in the following points:

1- **Operation optimization:** Conducting studies to determine the most optimized pipeline operation in terms of pressures, operating sequence, power usage (electricity or diesel pumps) versus using the DRA. To minimize operation cost.
2- **Simulating Operational Transient's States:** This is necessary to evaluate risks of rapture and determine maximum operating pressure for safety operations for each leg the pipeline.
3- **Pipeline Controllers Training:** Training of pipeline controllers (what sequences to use start/stop pumps or valves among others) to achieve the optimum flow rates.
4- **Leak Detection:** The objective of leak detection system is to detect (over time or over batch changes) line imbalances. To assess what happens along the pipeline between the net injected flow and the net delivery flow out of the pipeline. This is crucial to maintain the pipeline integrity. A flow model leverages the calculation of an extremely precise line balances by compensating the change in line packing rate.
5- **Products Batch Tracking:** Batches of products and their properties including quality; pipeline location at any moment; chemical composition; cost; origin; destination; and proprietorship are traceable through a pipeline flow model. Two types of batch tracking should be supported: Online and offline. The reason for offline product tracking models is for schedule estimates based on the business aspect. Real-time models' objective is to estimate interface arrival times actual deliveries based on the operational status of the pipeline (Accommodate for sudden shutdowns among other cases). Finally, product tracking determines ownership allocations of and precise transmission costs for each batch. [1]
6- **Line-pack Distribution:** Pipeline models produce an approximation of product distribution in the pipeline. These also supervise and calculate product inventory stored in the pipeline at any given moment. [1]
7- **Real-time Pipeline Pressure Awareness:** Through continuous observation and monitoring**,** pipeline models make available an accurate estimation of maximum pressures (to avoid line raptures) and minimum pressures (to avoid column separation i.e., slack condition) at any points between pressure sensing sites. (The industry standard is to allocate a measurement site every 30 miles of pipeline). [1]
8- **Maximum Deliverability**: Pipeline models is capable of calculating the maximum attainable flow rates at any delivery point. Equipment Performance Evaluation. Degradation of equipment performance could be detected by assessing the model estimates of current performance versus the declared manufacturer's specifications. [1]

1.2. Importance of Pipeline Leak Detection Flow Modeling

1.2.1.Impact of pipeline leaks

Aside of deaths and personal injuries, contamination or damages due to pipeline leaks have huge environmental and economic impact in addition to sever public concerns. For that, reliable leak analysis of pipeline assets is vital.

Pipeline and Hazardous Materials Safety Administration (PHMSA) is a federal agency with the U.S Department of Transportation (DOT). PHMSA published an analysis of federal records demonstrating that since 1986 as a total sum, more than 8,700 significant incidents with U.S. pipelines. These involve deaths, injuries, economic and environmental severe harm. This averages to be more than 300 per year. A Pipeline incident occur in the country every 30 hours[2]. Tables 1 and 2 detail these leaks incidents.

The Center for Biological Diversity published a new estimate. This estimate presents that since July 2013 and until November 2014, pipeline companies encountered for 372 oil and gas leaks, spills, among other incidents. This has led to 20 deaths, 117 injuries, and over a $256 million in damages.[2] [3].

1.2.2.Reasons for pipeline leaks

Pipeline leaks presents a major problem around the globe. Leaks could happen due to defective or poor quality pipeline materials. Pipeline could rapture breaks because of low standards in the craftsmanship of welding. Poor quality assurance standards. Operational mistakes (e.g., excessive pressure, sudden closure or opening valves). Long-term corrosion and lack of preventive maintenance is one of the main cause of leaks. Accidental damage to the assets by third parties through digging has been reported abundantly in the past. In summary, pipeline rapture could be attributed to many factors [4]:

1- Low quality or faulty materials used in production of pipelines and junctures,
2- Design errors
3- Insufficient preventive maintenance
4- Exceeding the recommend operating pressure
5- Random pipeline digging accidents damage.

The punchline: Leaks are going to happen. The more accurate pipeline flow modeling the faster the model will detect a leak. Thus, reduce the impact of the leak. The worst of all, when a company keeps pumping into a leak because their flow model did not detect the leak.

1.3. Problem Statement

Leak detection systems suffers from low number of modeling points to fulfill the real-time crucial requirement. By design, the number of partial differential equations and modeling points is limited to what current hardware and CPUs can handle. We need to beat computing resources limitation to achieve more accurate flow modeling for leak detection. This can be achieved by increasing the modeling points as a benefit of performance speed-ups.

1.4. Challenges of this Research

Most of the numerical methods to solve Partial Differential Equations (PDEs) in the literature are not parallel by nature. There is abundance of attempts in the literature to optimize the parallelization of many numerical methods of these equations solvers. None of them take leak detection (need for real-time) aspects in consideration.

Table 1- PHMSA Pipeline Serious Incidents for all a states [2]:

Year	Number	Fatalities	Injuries
1994	76	22	120
1995	59	21	64
1996	63	53	127
1997	49	10	77
1998	70	21	81
1999	66	22	108
2000	62	38	81
2001	40	7	61
2002	36	12	49
2003	61	12	71
2004	44	23	56
2005	39	16	47
2006	32	19	34
2007	43	15	47
2008	37	8	55
2009	46	13	62
2010	34	19	104
2011	32	12	51
2012	28	10	54
2013	24	9	44
Total	**941**	**362**	**1,393**

Table 2- PHMSA Pipeline Significant Incidents for all a states [2]:

Year	Count	Fatalities	Injuries	Property Damage in Dollars
1994	326	22	120	$229,864,143
1995	259	21	64	$72,978,742
1996	301	53	127	$157,237,457
1997	267	10	77	$106,467,248
1998	295	21	81	$168,366,265
1999	275	22	108	$171,954,256
2000	290	38	81	$248,585,734
2001	233	7	61	$76,344,769
2002	258	12	49	$121,661,817
2003	296	12	71	$159,730,017
2004	310	23	56	$308,168,808
2005	334	16	47	$1,421,763,106
2006	257	19	34	$151,670,059
2007	268	15	47	$145,563,397
2008	279	8	55	$568,864,817
2009	275	13	62	$173,645,089
2010	264	19	104	$1,567,492,098
2011	288	12	51	$395,284,307
2012	248	10	54	$218,796,466
2013	298	9	44	$327,649,677
Total	**5,621**	**362**	**1,393**	**$6,792,088,273**

1.5. What are Navier-Stokes Equations?

Nervier-stokes equations are PDEs widely used to model flow of air or liquid. These equations are roadly used in many domains because their ability model the physics of movement and flow of objects and currents. There are many applications, e.g., weather wind currents, liquids flow in a pipeline, aviation airflow, vehicles airflow models among many others. Mathematical background of Navier-Stokes equations:

$$\frac{\partial u}{\partial t} = -\nabla u\,.\,u - P + \mu\nabla^2 u + f \qquad (1)$$

$$\nabla u = 0 \qquad (2)$$

Where:

u: The flow velocity

P: Pressure

μ: Viscosity

f: Other forces

∇: (This is called many names: e.g., Nabla, Gradient, Divergence) $\nabla = \frac{\partial}{\partial \mathrm{x}} + \frac{\partial}{\partial \mathrm{y}} + \frac{\partial}{\partial \mathrm{x}}$

∇^2: (Laplacian): $\nabla^2 = \frac{\partial^2}{\partial \mathrm{x}^2} + \frac{\partial^2}{\partial \mathrm{y}^2} + \frac{\partial^2}{\partial \mathrm{z}^2}$

2. Serial Solution

The literature is abundant with many serial solutions for the Navier-Stokes equation. However, OpenCurrent started from famous serial implementation. Called Time Marching Algorithm (TMA). The serial solution depends on the following steps:

$$u^{t+1} = u^t + \delta t \frac{\partial u}{\partial t} \tag{3}$$

Let us plug (1) into (3):

$$\mathrm{u}^{t+1} = \mathrm{u}^t + \delta t\,[-\nabla u.u - P + \mu\,\nabla^2 u + f] \Rightarrow \hat{\mathrm{u}}^{t+1} = \mathrm{u}^t + \delta t\,[-\nabla u\,u - \mathrm{P} + \mu\,\nabla^2 u + f]$$

$$\mathrm{u}^{t+1} = \hat{\mathrm{u}}^{t+1} - \delta t\,P \tag{4}$$

From (4) we plug the 2nd Navier-Stokes Equation for pressures limit. Equation (2): This means that

$$\nabla u = 0$$

Pressure divergence is zero. This implies that pressure is zero at all times.

$$\nabla\,\mathrm{u}^{(t+1)} = 0 \Rightarrow \nabla[\hat{\mathrm{u}}^{t+1} - \delta t\,P] = 0 \Rightarrow \nabla[\delta\mathrm{t}\ \mathrm{p}] = \nabla\,\hat{\mathrm{u}}^{t+1} \Rightarrow$$

$$\delta t\,\nabla^2 P = \nabla\hat{\mathrm{u}}^{t+1} \tag{5}$$

Pseudo code of the basic Time Marching Computational Fluid Dynamics (CFD) Algorithm:

Listing 1- Pseudo code of the serial iterative Pseudo code of the basic Time Marching CFD Algorithm:

```
1- Input is u at time t =0;
2- Calculate at t=1, 2…
    a. Time March to move to $\hat{u}$
       $u^{t+1} = u^t + \delta t\,[-\nabla u.u - P + \mu\nabla^2 u + f]$
    b. Solve for p from (6)
       $\delta t\,\nabla^2 P = \nabla\hat{u}^{t+1} \Rightarrow \nabla^2 P = \frac{\nabla\hat{u}^{t+1}}{\delta t} \Rightarrow$
       $\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} + \frac{\partial^2 P}{\partial z^2} = \frac{\nabla\hat{u}^{t+1}}{\delta t}$
       where,
       $\frac{\partial^2 P}{\partial x^2} \approx \frac{\frac{\partial(P+\Delta x)}{\Delta x} - \frac{\partial P}{\partial x}}{\delta x}$
    c. Projection: Subtract p to get to u From (5):
       $u^{t+1} = \hat{u}^{t+1} - \delta t\,P$
```

The following diagram helps understanding the solving for pressure:

P[0]	P[1]	P[2]	P[3]	P[4]	P[5]	P[6]	P[7]
			1	-2	1		

How to solve for P. Let us focus only on X component of the Laplacian. We know that:

$$\frac{\partial P}{\partial x} = \frac{P[4]-P[3]}{\Delta x} \;\&\; \frac{\partial(P+\Delta x)}{\Delta x} = \frac{P[5]-P[4]}{\Delta \mathrm{x}} \Rightarrow$$

$$\frac{\partial^2 P}{\partial \mathrm{x}^2} \approx \frac{\frac{P[5]-P[4]}{\Delta \mathrm{x}} - \frac{P[4]-P[3]}{\Delta X}}{\Delta x} = \frac{1}{(\Delta x)^2}\,(P[3]-2P[4]+P[5])$$

Pseudo code of the serial iterative Poisson Solver is the following listing:

Listing 2- Pseudo code of the serial iterative Poisson Solver

```
$M\ P = r$ Where and r and known. Error is easy to estimate $E = M\ P' - r$
$P'$ is the solution.
Start with a guess for $P$ and call it $P'$ and then apply this pseudo code:
Until $M\ P' - r < tolerance$ (Tolerance the accuracy of my solution)
    $P' \leftarrow Update(P', M, r)$
Return $P'$
```

A replacement of this is the Serial Gauss-Seidel Relaxation:

This relies on formulating pressure the equations as a sparse linear system, e.g.,

$$\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
-2 & 1 & & & & & & 1 \\ \hline
1 & -2 & 1 & & & & & \\ \hline
 & 1 & -2 & 1 & & & & \\ \hline
 & & 1 & -2 & 1 & & & \\ \hline
 & & & 1 & -2 & 1 & & \\ \hline
 & & & & 1 & -2 & 1 & \\ \hline
 & & & & & 1 & -2 & 1 \\ \hline
1 & & & & & & 1 & -2 \\ \hline
\end{array}
\begin{array}{|c|}
\hline
P[0] \\ \hline
P[1] \\ \hline
P[2] \\ \hline
P[3] \\ \hline
P[4] \\ \hline
P[5] \\ \hline
P[6] \\ \hline
P[7] \\ \hline
\end{array}
= \Delta X^2 \frac{\nabla \hat{u}^{t+1}}{\delta t}$$

At first, we start by rewriting the equations:

$$P[7] - 2\,P[0] + P[1] = h \times h \times \frac{\nabla \hat{u}^{t+1}}{\delta t} \xRightarrow{Rewrite} P[0] = \frac{1}{2}\left(P[7] + P[1] - h \times h \times \frac{\nabla \hat{u}^{t+1}}{\delta t} \right)$$

$$P[0] - 2\,P[1] + P[2] = h \times h \times \frac{\nabla \hat{u}^{t+1}}{\delta t} \xRightarrow{Rewrite} P[1] = \frac{1}{2}\left(P[0] + P[2] - h \times h \times \frac{\nabla \hat{u}^{t+1}}{\delta t} \right)$$

$$P[1] - 2\,P[2] + P[3] = h \times h \times \frac{\nabla \hat{u}^{t+1}}{\delta t} \xRightarrow{Rewrite} P[2] = \frac{1}{2}\left(P[1] + P[3] - h \times h \times \frac{\nabla \hat{u}^{t+1}}{\delta t} \right)$$

Then looping on all equations such on the

Listing 3- Pseudo code of the Serial iterative Gauss-Seidel Relaxation

```
Loop until convergence
For each equation j=1 to n
  Solve for P[j]
```

3. <u>Related Work</u>

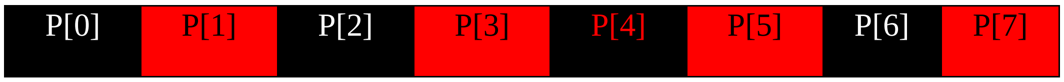
Parallel Computing GPUs is an immense fine-grained parallel computation platform. This platform becoming more and more appealing for researchers to model their CFD-based problems. Specially with the potential 10 to 100 times gain in performance speedup. OpenCurrent [5],[6] is a free and open source software dedicated to solve PDEs and specially Navier-Stocks. OpenCurrent's architecture is flexible (However, not easy to comprehend) and built around reusability. This open library is built in layers to allow for maximum reusability. It has been widely utilized by many research projects. This library is the one we selected to optimize their parallel solution. In despite the little wiggle room left in this regard.

Many other successful attempts have been published in recent years as in [7],[8],[9],[10],[11], and [12]. Early attempts of utilizing GPU for CFD has been troubled by the heterogeneous nature of GPU hardware and somehow a sophisticated programing scheme. Currently, GPU technology incurred substantial improvement. The appearance of an OpenMP-like directive-based programming scheme, i.e., OpenACC [13], further decreases the complications for an efficient GPU programming experience. This is achieved by enabling researches to employ a pool of compiler directives to highlight loops and regions in their codes to be loaded from a host CPU to GPU, this scheme strikes balance between development efforts and performance gain. As shown in [14],[15]. This solution was inspired by a CPU-based well-validated set of flow modeling problems [16],[17].

A solution for the 3D incompressible Navier-Stocks equations was introduced in [18]. Solution environment depended on a in a structured grid. PDEs were solved using the finite volume method (FVM).

4. <u>Parallelization Strategy</u>

OpenCurrent introduced a parallelized Gauss-Seidel Relaxation to solve for pressure. It was called Parallel (Red-Black) Gauss-Seidel Relaxation: (Chess).



Listing 4- Pseudo code of the parallelized Gauss-Seidel Relaxation (red-black)

```
Loop n times (until convergence)
    For each even equation j = 0 to n-1
        Solve for P[j]
    For each odd equation j = 1 to n
        Solve for P[j]
```

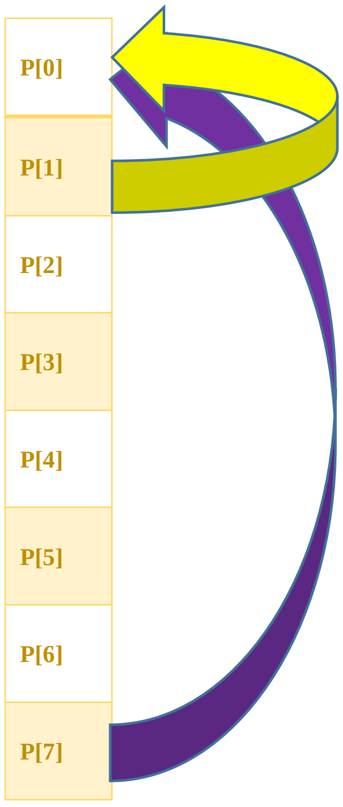
One pass of parallel Algorithm is presented in the following diagram:

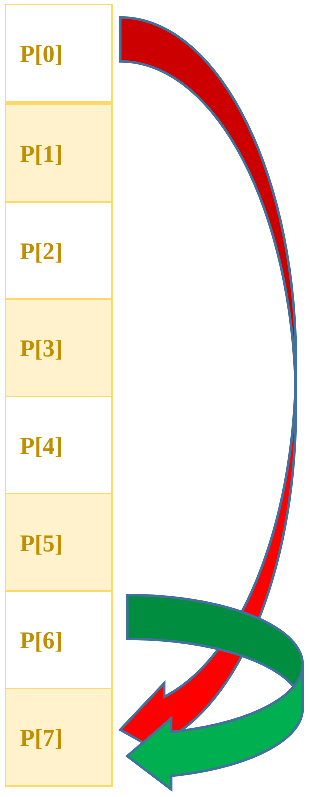
$$P[0] = \frac{\left(P[7] + P[1] - h \times h \times \frac{\nabla \hat{u}^{t+1}}{\delta t}\right)}{2} \qquad P[1] = \frac{\left(P[0] + P[2] - h \times h \times \frac{\nabla \hat{u}^{t+1}}{\delta t}\right)}{2}$$

$$P[2] = \frac{\left(P[1] + P[3] - h \times h \times \frac{\nabla \hat{u}^{t+1}}{\delta t}\right)}{2} \qquad P[3] = \frac{\left(P[2] + P[4] - h \times h \times \frac{\nabla \hat{u}^{t+1}}{\delta t}\right)}{2}$$

$$P[4] = \frac{\left(P[3] + P[5] - h \times h \times \frac{\nabla \hat{u}^{t+1}}{\delta t}\right)}{2} \qquad P[5] = \frac{\left(P[4] + P[6] - h \times h \times \frac{\nabla \hat{u}^{t+1}}{\delta t}\right)}{2}$$

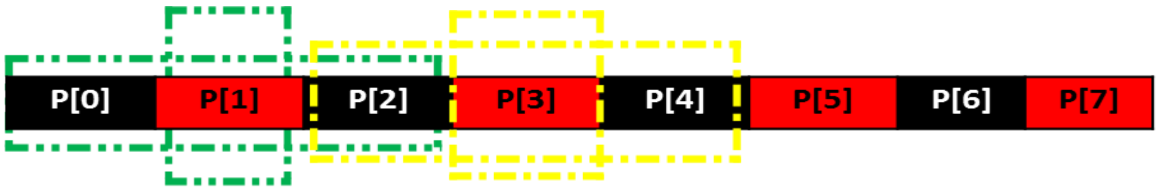
$$P[6] = \frac{\left(P[5] + P[7] - h \times h \times \frac{\nabla \hat{u}^{t+1}}{\delta t}\right)}{2} \qquad P[7] = \frac{\left(P[6] + P[0] - h \times h \times \frac{\nabla \hat{u}^{t+1}}{\delta t}\right)}{2}$$





Notes about this method:

- Red-Black scheme is extremely cache friendly P[2] is right for P[1] and left for P[3].
- Reusability is between adjacent threads (green and yellow) could be depicted in the following figure:



5. Parallel Optimization

5.1. Stencil Like Optimizations

From this equation: You can think of this as a stencil! All stencil optimizations could be applied here, i.e., shared memory tiling and optimized thread coarsening.

5.1.1.Attempted shared memory optimizations.

(I did try to do that. However, failed to launch kernels I still did not understand the layered architecture. Although I spent a lot of time on it)

Original: sol_Laplaciancent1Ddev.cu

```
__global__ void Sol_Laplacian1DCentered_apply_stencil(float inv_h2, float *deriv_densitydt,
float *density, int nx)
{
  int i = threadIdx.x + __umul24(blockIdx.x, blockDim.x);

  i--;

  // note that density & deriv_densitydt are both shifted so that they point to the "0"
element, even though the arrays start
  // at element -1.  hence by offsetting i as above, we will get better coalescing for the
cost of an added test if i>=0

  if (i>=0 && i < nx)
    deriv_densitydt[i] = inv_h2 * (density[i-1] - 2.0f * density[i] + density[i+1]);
}
```

Optimized: sol_Laplaciancent1Ddev.cu

```
#define TILE_SIZE 1024

__global__ void Sol_Laplacian1DCentered_apply_stencil(float inv_h2, float *deriv_densitydt,
float *density, int nx)
{
  int i = threadIdx.x + __umul24(blockIdx.x, blockDim.x);
   i--;
  //ASO
          __shared__ float ds_A[TILE_SIZE];
      float current = density[i];
            ds_A[i]= current;
            __syncthreads();
            float right = ds_A[i+1];
            float left  = ds_A[i-1];


  // note that density & deriv_densitydt are both shifted so that they point to the "0"
element, even though the arrays start
  // at element -1.  hence by offsetting i as above, we will get better coalescing for the
cost of an added test if i>=0

  if (i>=0 && i < nx)
  {
    //deriv_densitydt[i] = inv_h2 * (density[i-1] - 2.0f * density[i] + density[i+1]);
    //ASO
      deriv_densitydt[i] = inv_h2 * (left - 2.0f * current + right);
  }
}
```

5.1.2. Thread coarsening on top of the shared memory tiling.

David Cohen with Nvidia already implemented thread coarsening where applied in some of the modules. However, few other modules could take some optimization. This was so hard to achieve, as after I finished optimizing some of the kernels, was so hard to design a test to through the compounded library structure to test my optimizations. However, I did achieve this optimization on the 1D Laplacian.

Further Optimized: sol_Laplaciancent1Ddev.cu

```
#define TILE_SIZE 1024


//*********************************************************************************
// Kernels (currently must be outside of namespaces)
//*********************************************************************************


__global__ void Sol_Laplacian1DCentered_apply_stencil(float inv_h2, float *deriv_densitydt,
float *density, int nx)
{
   int ii = threadIdx.x + __umul24(blockIdx.x, blockDim.x);
   ii--;
   unsigned int CoarseningScale = nx/blockDim.x +1;
   shared__ float ds_A[TILE_SIZE];


  //ASO
  if (ii < nx/CoarseningScale) {
         unsigned int StartIdx = ii * CoarseningScale;
         unsigned int EndIdx= StartIdx + CoarseningScale;
         float current = density[ii];
         ds_A[ii]= current;
         __syncthreads();
         float right = ds_A[ii+1];
         float left  = ds_A[ii-1];
         for(unsigned int i = StartIdx; i < EndIdx; ++i)
         {
           left = current;
      current = right;
           right = density[ii];
           ds_A[ii]= right;
         __syncthreads();
           if (i>=0 && i < nx)
           {
                  deriv_densitydt[i] = inv_h2 * (left - 2.0f * current + right);
           }//End if
    }//End for
 }//End Big If
}//End Kernel
```

5.2. Idea: Parameters sizes compressing [NOT IMPLMENTED]

Based on real life parameter sizes. **This was more complex and compounded than I thought and needs more time.** However, an overview of my suggested parameters compressing scheme:

- For example real-life temperature (Celsius) cannot be more than -50 to 120C with one digit after the decimal point   -xx.x to xxx.x (11 bits)

- For example, real-life pressure (psi) cannot be more than 7000 to -50 with one digit after the decimal point. –xx.x to xxxx.x (13 bits)
- Same analysis for other parameters:
- Flow cannot be more than 40,000 and less than 0 also with one
- Viscosity
- Density (Among others)

The motive behind this (novel scheme in this domain, to best of my knowledge) i.e., using only one digit after the decimal point, among other compressing ideas is: **The instrumentation error ranges on incompressible liquid pipelines.**

Example: The Error in even the state of the art pressure sensors/transmitters cannot be lower than ± 0.25 % of full scale. That is estimated at ± 10 psi. Although, the range of needed liquid pipelines cannot exceed 4100 psi (pounds/square inches). Range of current state of art pressure sensors the does not exceed 4100 psi [19][20].

5.3. Structural Optimization:

Joining some of the kernel launches in a single kernel to reduce the overhead of kernel launching **(Optimization failed for inability of launch kernels due to the complexity of the layerd approach and using many classes, templates, constructors and destructors among others)**

6. Evaluation

An evaluation of the Naïve implementation, according to Cohen et al., shows a speed up to 8.5 times from the baseline of the serial FORTRAN code. This is shown in Table 3.

Table 3- Benchmark Evaluation of Navier-Stokes Equation vs Serial FORTRAN (Serial) Version [6]

Resolution	**CUDA time/step (ms)**	**Fortran time/step (ms)**	**Speedup of Naïve Version**
64 x 64 x 32	24	47	2.0 times
128 x 128 x 64	79	327	4.1 times
256 x 256 x 128	489	4070	8.2 times
384 x 384 x 192	1616	13670	8.5 times

Table 4 proves valuable the (5.3) point of our parallelization optimization strategy. We can expect that compressing variable sizes in memory is a promising potential way that could help increase the speed up ratio. This conclusion was made when OpenCurrent compared computational speedup between fp64 vs. fp32 they indeed achieved some speedup.

Table 4- Benchmark Evaluation Fermi: Single vs Double Precision [6]

Resolution	**fp64 time/step**	**fp32 time/step**	**Ratio**
64^3	0.012349	0.010778	1.15 x
128^3	0.041397	0.030348	1.36 x
256^3	0.311494	0.208250	1.49 x

7. Conclusion

7.1. Final Note and Lessons Learnt

In spite that this was my first CUDA course, I dared to choose a large project with a huge learning curve! I admit that was mistake. However, I really wanted to study those equations to learn how to solve them in both and serial parallel methods. However, I totally enjoyed working on this project. I did achieve some optimizations from the one I learned throughout the course. I totally needed another month to feel satisfied about this work.

Understanding how to install and operate the open source code. Then understanding how the code itself behaves (how the layers are interacting with other) before even optimizing any kernel was a big deal! Especially with the lack of documentation of the chosen open-source project. (Like many open-source projects. Most of which could really take some serious documentation).

Deeper understanding of the library could have expedited troubleshooting the many bugs incurred when attempting to launch my optimized version the current code.

7.2. Contributions

7.2.1. Course-inspired optimizations:

I did optimize a kernel (one of the so many kernels in this humongous library). I identified a stencil like computation. I went ahead and applied shared memory tiling and thread coarsening on that one.

7.2.2. A novel idea/application:

To the best of my knowledge, there is not in the literature a study about Navier-Stokes equations solvers for real-time PDEs for leak detection purposes. This could be a foundation for future publication down the road.

7.2.3.Domain specific tune-ups:

To the best of my knowledge, I introduced another novel solution. This suggests imposing an optimization on the general CUDA solution by focusing only on liquid pipeline modeling accuracy needs. Reducing the computation loads and reducing memory usage: Introducing the compressed and custom tailor-sized variable sizes e.g., pressures and temperatures, among others. We can expect that compressing variable sizes in memory is a promising potential way that could help increase the speed up ratio. This assumption proved correct as OpenCurrent compares their computational speedup between fp64 vs. fp32 they indeed achieved some speedup.

I also suggested the utilization of binning. However, I did not find any opportunity in code for that.

8. References

[1] J. L. Modisette and J. P. Modisette, "Transient and Succession-of-Steady-States Pipeline Flow Models", Pipeline Simulation Interest Group Technical Paper, August 2014

[2] "PHMSA Published Incidents" [Online] Available: http://www.phmsa.dot.gov/pipeline/library/datastatistics/pipelineincidenttrends, accessed 2014.

[3] "Center Biological Diversity" [Online] Available: http://www.biologicaldiversity.org/campaigns/americas_dangerous_pipelines/, accessed 2014.

[4] Lay-Ekuakille, A.; Vendramin, G.; Trotta, Amerigo, "Robust Spectral Leak Detection of Complex Pipelines Using Filter Diagonalization Method," Sensors Journal, IEEE , vol.9, no.11, pp.1605,1614, Nov.2009 doi: 10.1109/JSEN.2009.2027410

[5] "OpenCurrentProject" [Online]. Available: https://code.google.com/p/opencurrent/wiki/OpenCurrent , accessed 2014.

[6] Cohen, J., and Molemaker, M. "A fast double precision CFD code using CUDA" J. Phys. Soc. Japan 66 (2009), 2237–2341.

[7] J. C. Thibault and I. Senocak, "CUDA implementation of a navier-stokes solver on multi-gpu desktop platforms for incompressible flows," in Proceedings of the 47th AIAA Aerospace Sciences Meeting, 2009, pp. 2009–758.

[8] E. H. Phillips, Y. Zhang, R. L. Davis, and J. D. Owens, "Rapid aerodynamic performance prediction on a cluster of graphics processing units," in Proceedings of the 47th AIAA Aerospace Sciences Meeting, 2009, pp. 2009–565.

[9] Y. Xia, H. Luo, L. Luo, J. Edwards, J. Lou, and F. Mueller, "OpenACC-based GPU acceleration of a 3-D unstructured discontinuous galerkin method," in 52nd Aerospace
Sciences Meeting, 2014.

[10] D. Goddeke, S. H. Buijssen, H. Wobker, and S. Turek, "GPU Acceleration of an Unmodified Parallel Finite Element Navier-Stokes Solver," in International Conference on High Performance Computing & Simulation, 2009. HPCS09. IEEE, 2009, pp. 12–21.

[11] T. Brandvik and G. Pullan, "Acceleration of a 3d euler solver using commodity graphics hardware," in 46th AIAA aerospace sciences meeting and exhibit, 2008, p. 607.

[12] A. Corrigan, F. Camelli, R. Löhner, and F. Mut, "Semi-automatic porting of a large-scale fortran CFD code to GPUs," International Journal for Numerical Methods in Fluids, vol. 69, no. 2, pp. 314–331, 2012.

[13] The OpenACC Application Programming Interface, 2013.

[14] MVAPICH2: High performance MPI over InfiniBand, 10GigE/iWARP and RoCE.

[15] L. Luo, J. R. Edwards, H. Luo, and F. Mueller, "Performance assessment of a multiblock incompressible navier-stokes solver using directive-based gpu programming in a cluster environment," in 52nd Aerospace Sciences Meeting, 2013.

[16] J.-I. Choi, R. C. Oberoi, J. R. Edwards, and J. A. Rosati, "An immersed boundary method for complex incompressible flows," Journal of Computational Physics, vol. 224,no. 2, pp. 757–784, 2007.

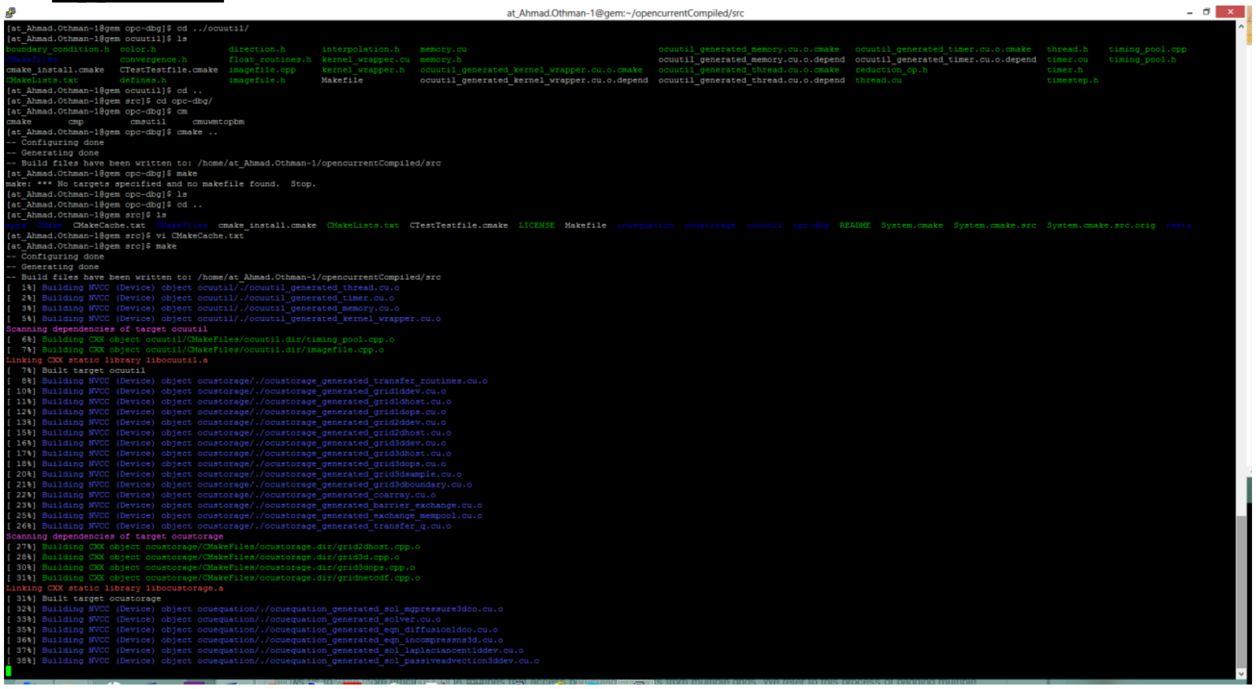
[17] J. R. Edwards and M.-S. Liou, "Low-diffusion flux-splitting methods for flows at all speeds," AIAA journal, vol. 36, no. 9, pp. 1610–1617, 1998.

[18] A. J. Chorin, "Numerical Solution of the Navier-Stokes equations," Mathematics of Computation, vol. 22, no. 104, pp. 745–762, 1968

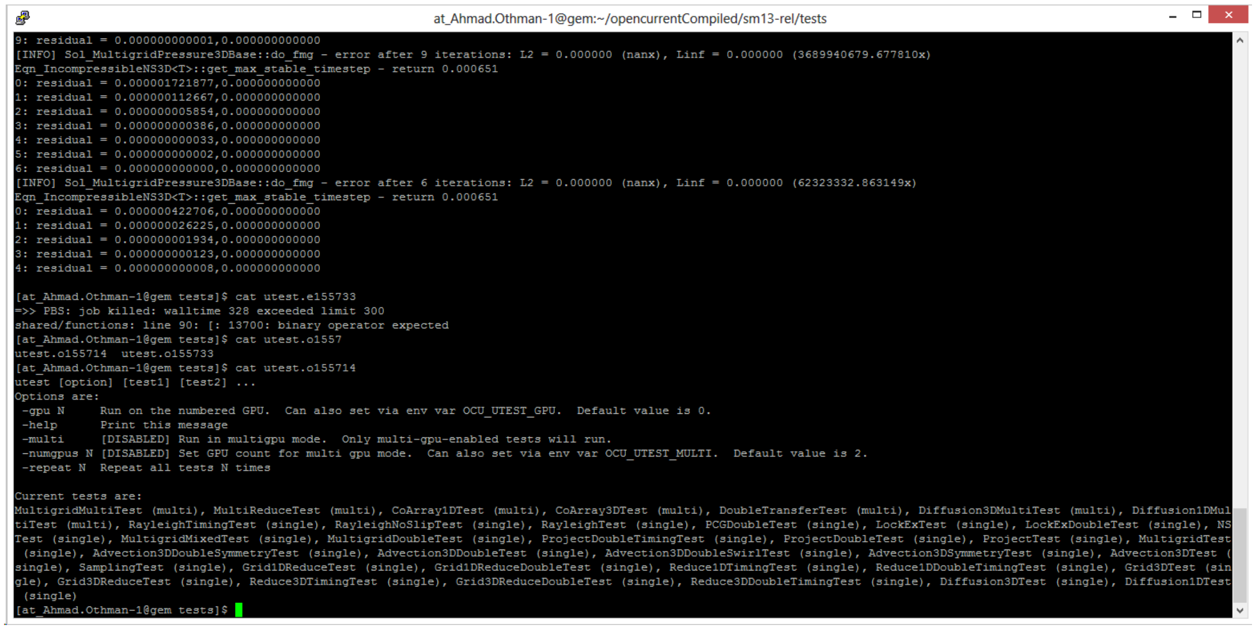
[19] "Siemens Pressure Sensors Data Sheet" [Online]. Available: http://www.buildingtechnologies.siemens.com/bt/global/en/buildingautomation-hvac/hvac-products/hvac-sensors/pressure-sensors/pages/pressure-sensors.aspx, accessed 2014.

[20] "General Eclectic (GE) Pressure Sensors Data Sheet" [Online]. http://www.ge-mcs.com/download/pressure-level/K332-issue1.pdf, accessed 2014.

9. Appendices


at_Ahmad.Othman-1@gem:~/opencurrentCompiled/src

```
[at_Ahmad.Othman-1@gem opc-dbg]$ cd ../ocuutil/
[at_Ahmad.Othman-1@gem ocuutil]$ ls
boundary_condition.h  color.h              direction.h       interpolation.h    memory.cu                                      ocuutil_generated_memory.cu.o.cmake   ocuutil_generated_timer.cu.o.cmake   thread.h    timing_pool.cpp
CMakeFiles            convergence.h        float_routines.h  kernel_wrapper.cu  memory.h                                       ocuutil_generated_memory.cu.o.depend  ocuutil_generated_timer.cu.o.depend  timer.cu    timing_pool.h
cmake_install.cmake   CTestTestfile.cmake  imagefile.cpp     kernel_wrapper.h   ocuutil_generated_kernel_wrapper.cu.o.cmake    ocuutil_generated_thread.cu.o.cmake   reduction_op.h                       timer.h
CMakeLists.txt        defines.h            imagefile.h       Makefile           ocuutil_generated_kernel_wrapper.cu.o.depend   ocuutil_generated_thread.cu.o.depend  thread.cu                            timestep.h
[at_Ahmad.Othman-1@gem ocuutil]$ cd ..
[at_Ahmad.Othman-1@gem src]$ cd opc-dbg/
[at_Ahmad.Othman-1@gem opc-dbg]$ cm
cmake       cmp         cmsutil     cmuwmtopbm
[at_Ahmad.Othman-1@gem opc-dbg]$ cmake ..
-- Configuring done
-- Generating done
-- Build files have been written to: /home/at_Ahmad.Othman-1/opencurrentCompiled/src
[at_Ahmad.Othman-1@gem opc-dbg]$ make
make: *** No targets specified and no makefile found.  Stop.
[at_Ahmad.Othman-1@gem opc-dbg]$ ls
[at_Ahmad.Othman-1@gem opc-dbg]$ cd ..
[at_Ahmad.Othman-1@gem src]$ ls
apps  CMake  CMakeCache.txt  CMakeFiles  cmake_install.cmake  CMakeLists.txt  CTestTestfile.cmake  LICENSE  Makefile  ocuequation  ocustorage  ocuutil  opc-dbg  README  System.cmake  System.cmake.src  System.cmake.src.orig  tests
[at_Ahmad.Othman-1@gem src]$ vi CMakeCache.txt
[at_Ahmad.Othman-1@gem src]$ make
-- Configuring done
-- Generating done
-- Build files have been written to: /home/at_Ahmad.Othman-1/opencurrentCompiled/src
[  1%] Building NVCC (Device) object ocuutil/./ocuutil_generated_thread.cu.o
[  2%] Building NVCC (Device) object ocuutil/./ocuutil_generated_timer.cu.o
[  3%] Building NVCC (Device) object ocuutil/./ocuutil_generated_memory.cu.o
[  5%] Building NVCC (Device) object ocuutil/./ocuutil_generated_kernel_wrapper.cu.o
Scanning dependencies of target ocuutil
[  6%] Building CXX object ocuutil/CMakeFiles/ocuutil.dir/timing_pool.cpp.o
[  7%] Building CXX object ocuutil/CMakeFiles/ocuutil.dir/imagefile.cpp.o
Linking CXX static library libocuutil.a
[  7%] Built target ocuutil
[  8%] Building NVCC (Device) object ocustorage/./ocustorage_generated_transfer_routines.cu.o
[ 10%] Building NVCC (Device) object ocustorage/./ocustorage_generated_grid1ddev.cu.o
[ 11%] Building NVCC (Device) object ocustorage/./ocustorage_generated_grid1dhost.cu.o
[ 12%] Building NVCC (Device) object ocustorage/./ocustorage_generated_grid1dops.cu.o
[ 13%] Building NVCC (Device) object ocustorage/./ocustorage_generated_grid2ddev.cu.o
[ 15%] Building NVCC (Device) object ocustorage/./ocustorage_generated_grid2dhost.cu.o
[ 16%] Building NVCC (Device) object ocustorage/./ocustorage_generated_grid3ddev.cu.o
[ 17%] Building NVCC (Device) object ocustorage/./ocustorage_generated_grid3dhost.cu.o
[ 18%] Building NVCC (Device) object ocustorage/./ocustorage_generated_grid3dops.cu.o
[ 20%] Building NVCC (Device) object ocustorage/./ocustorage_generated_grid3dsample.cu.o
[ 21%] Building NVCC (Device) object ocustorage/./ocustorage_generated_grid3dboundary.cu.o
[ 22%] Building NVCC (Device) object ocustorage/./ocustorage_generated_coarray.cu.o
[ 23%] Building NVCC (Device) object ocustorage/./ocustorage_generated_barrier_exchange.cu.o
[ 25%] Building NVCC (Device) object ocustorage/./ocustorage_generated_exchange_mempool.cu.o
[ 26%] Building NVCC (Device) object ocustorage/./ocustorage_generated_transfer_q.cu.o
Scanning dependencies of target ocustorage
[ 27%] Building CXX object ocustorage/CMakeFiles/ocustorage.dir/grid2dhost.cpp.o
[ 28%] Building CXX object ocustorage/CMakeFiles/ocustorage.dir/grid3d.cpp.o
[ 30%] Building CXX object ocustorage/CMakeFiles/ocustorage.dir/grid3dops.cpp.o
[ 31%] Building CXX object ocustorage/CMakeFiles/ocustorage.dir/gridnetcdf.cpp.o
Linking CXX static library libocustorage.a
[ 31%] Built target ocustorage
[ 32%] Building NVCC (Device) object ocuequation/./ocuequation_generated_sol_mgpressure3dco.cu.o
[ 33%] Building NVCC (Device) object ocuequation/./ocuequation_generated_solver.cu.o
[ 35%] Building NVCC (Device) object ocuequation/./ocuequation_generated_eqn_diffusion1dco.cu.o
[ 36%] Building NVCC (Device) object ocuequation/./ocuequation_generated_eqn_incompressns3d.cu.o
[ 37%] Building NVCC (Device) object ocuequation/./ocuequation_generated_sol_laplaciancent1ddev.cu.o
[ 38%] Building NVCC (Device) object ocuequation/./ocuequation_generated_sol_passiveadvection3ddev.cu.o
```


at_Ahmad.Othman-1@gem:~/opencurrentCompiled/sm13-rel/tests

```
9: residual = 0.000000000001,0.000000000000
[INFO] Sol_MultigridPressure3DBase::do_fmg - error after 9 iterations: L2 = 0.000000 (nanx), Linf = 0.000000 (3689940679.677810x)
Eqn_IncompressibleNS3D<T>::get_max_stable_timestep - return 0.000651
0: residual = 0.000001721877,0.000000000000
1: residual = 0.000000112667,0.000000000000
2: residual = 0.000000005854,0.000000000000
3: residual = 0.000000000386,0.000000000000
4: residual = 0.000000000033,0.000000000000
5: residual = 0.000000000002,0.000000000000
6: residual = 0.000000000000,0.000000000000
[INFO] Sol_MultigridPressure3DBase::do_fmg - error after 6 iterations: L2 = 0.000000 (nanx), Linf = 0.000000 (62323332.863149x)
Eqn_IncompressibleNS3D<T>::get_max_stable_timestep - return 0.000651
0: residual = 0.000000422706,0.000000000000
1: residual = 0.000000026225,0.000000000000
2: residual = 0.000000001934,0.000000000000
3: residual = 0.000000000123,0.000000000000
4: residual = 0.000000000008,0.000000000000

[at_Ahmad.Othman-1@gem tests]$ cat utest.e155733
=>> PBS: job killed: walltime 328 exceeded limit 300
shared/functions: line 90: [: 13700: binary operator expected
[at_Ahmad.Othman-1@gem tests]$ cat utest.o1557
utest.o155714  utest.o155733
[at_Ahmad.Othman-1@gem tests]$ cat utest.o155714
utest [option] [test1] [test2] ...
Options are:
 -gpu N      Run on the numbered GPU.  Can also set via env var OCU_UTEST_GPU.  Default value is 0.
 -help       Print this message
 -multi      [DISABLED] Run in multigpu mode.  Only multi-gpu-enabled tests will run.
 -numgpus N  [DISABLED] Set GPU count for multi gpu mode.  Can also set via env var OCU_UTEST_MULTI.  Default value is 2.
 -repeat N   Repeat all tests N times

Current tests are:
MultigridMultiTest (multi), MultiReduceTest (multi), CoArray1DTest (multi), CoArray3DTest (multi), DoubleTransferTest (multi), Diffusion3DMultiTest (multi), Diffusion1DMul
tiTest (multi), RayleighTimingTest (single), RayleighNoSlipTest (single), RayleighTest (single), PCGDoubleTest (single), LockExTest (single), LockExDoubleTest (single), NS
Test (single), MultigridMixedTest (single), MultigridDoubleTest (single), ProjectDoubleTimingTest (single), ProjectDoubleTest (single), ProjectTest (single), MultigridTest
 (single), Advection3DDoubleSymmetryTest (single), Advection3DDoubleTest (single), Advection3DDoubleSwirlTest (single), Advection3DSymmetryTest (single), Advection3DTest (
single), SamplingTest (single), Grid1DReduceTest (single), Grid1DReduceDoubleTest (single), Reduce1DTimingTest (single), Reduce1DDoubleTimingTest (single), Grid3DTest (sin
gle), Grid3DReduceTest (single), Reduce3DTimingTest (single), Grid3DReduceDoubleTest (single), Reduce3DDoubleTimingTest (single), Diffusion3DTest (single), Diffusion1DTest
 (single)
[at_Ahmad.Othman-1@gem tests]$
```

cat utest.o155714

```
utest [option] [test1] [test2] ...
Options are:
 -gpu N     Run on the numbered GPU.  Can also set via env var OCU_UTEST_GPU.  Default value is 0.
 -help      Print this message
 -multi     [DISABLED] Run in multigpu mode.  Only multi-gpu-enabled tests will run.
 -numgpus N [DISABLED] Set GPU count for multi gpu mode.  Can also set via env var OCU_UTEST_MULTI.  Default value is 2.
 -repeat N  Repeat all tests N times

Current tests are:
MultigridMultiTest (multi), MultiReduceTest (multi), CoArray1DTest (multi), CoArray3DTest (multi), DoubleTransferTest (multi),
Diffusion3DMultiTest (multi), Diffusion1DMultiTest (multi), RayleighTimingTest (single), RayleighNoSlipTest (single), RayleighTest
(single), PCGDoubleTest (single), LockExTest (single), LockExDoubleTest (single), NSTest (single), MultigridMixedTest (single),
MultigridDoubleTest (single), ProjectDoubleTimingTest (single), ProjectDoubleTest (single), ProjectTest (single), MultigridTest
(single), Advection3DDoubleSymmetryTest (single), Advection3DDoubleTest (single), Advection3DDoubleSwirlTest (single),
Advection3DSymmetryTest (single), Advection3DTest (single), SamplingTest (single), Grid1DReduceTest (single),
Grid1DReduceDoubleTest (single), Reduce1DTimingTest (single), Reduce1DDoubleTimingTest (single), Grid3DTest (single),
Grid3DReduceTest (single), Reduce3DTimingTest (single), Grid3DReduceDoubleTest (single), Reduce3DDoubleTimingTest (single),
Diffusion3DTest (single), Diffusion1DTest (single)
```

submitjob ./utest RayleighNoSlipTest

```
............ DONE ...............

ms/step = 3.896338
16  - 1674.296911 (error -33.463089)
32  - 1699.254119 (error -8.505881) - order(1.976038)
64  - 1705.590663 (error -2.169337) - order(1.971207)

estimated_ra_exact = 1707.638016, error = 0.1219839032

Advection3DD_apply_upwind_TEXCPU:     Total 612.380615, Count 18290, Avg 0.033482
Eqn_IncompressibleNS3D_add_thermal_forceCPU:  Total 292.294342, Count 18290, Avg 0.015981
Grid3DDevice_linear_combination2_3DCPU:  Total 1145.033325, Count 73160, Avg 0.015651
Sol_Divergence3DDevice_calculate_divergence::calculate_divergenceCPU:   Total 308.536987, Count 18290, Avg 0.016869
Sol_Gradient3DDevice_subtract_gradCPU:  Total 374.848633, Count 18290, Avg 0.020495
Sol_LaplacianCentered3DDevice_stencilCPU:     Total 1378.604980, Count 73160, Avg 0.018844
Sol_MultigridPressure3DDevice::relax_on_host(4)CPU:   Total 19268.234375, Count 29069, Avg 0.662845
Sol_MultigridPressure3DDeviceD_calculate_residual(16)CPU:     Total 516.795898, Count 34636, Avg 0.014921
Sol_MultigridPressure3DDeviceD_calculate_residual(32)CPU:     Total 780.409058, Count 39662, Avg 0.019676
Sol_MultigridPressure3DDeviceD_prolong(16)CPU:  Total 454.517365, Count 14642, Avg 0.031042
Sol_MultigridPressure3DDeviceD_prolong(8)CPU:  Total 573.556946, Count 29069, Avg 0.019731
Sol_MultigridPressure3DDeviceD_relax(16)CPU:  Total 2652.482910, Count 136020, Avg 0.019501
Sol_MultigridPressure3DDeviceD_relax(32)CPU:  Total 2006.352173, Count 58568, Avg 0.034257
Sol_MultigridPressure3DDeviceD_restrict(16)CPU:  Total 332.821960, Count 24348, Avg 0.013669
Sol_MultigridPressure3DDeviceD_restrict(8)CPU:  Total 378.675476, Count 31660, Avg 0.011961
Sol_PassiveAdvection3D_apply_interpCPU:  Total 441.600830, Count 18290, Avg 0.024144
cudaFreeCPU:  Total 633.393555, Count 281554, Avg 0.002250
cudaFreeHostCPU:  Total 5.810738, Count 24, Avg 0.242114
cudaHostAllocCPU:  Total 4.259109, Count 24, Avg 0.177463
cudaMallocCPU:  Total 810.290894, Count 281554, Avg 0.002878
cudaMemcpy(DeviceToHost)CPU:  Total 747.533997, Count 58138, Avg 0.012858
cudaMemcpy(HostToDevice)CPU:  Total 439.966614, Count 29085, Avg 0.015127
cudaMemcpyCPU:  Total 1194.088013, Count 93822, Avg 0.012727
cudaMemsetCPU:  Total 567.052795, Count 56050, Avg 0.010117
kernel_apply_3d_boundary_conditions_level1_nocorners(16)CPU:  Total 350.409241, Count 15314, Avg 0.022882
kernel_apply_3d_boundary_conditions_level1_nocorners(8)CPU:   Total 65.535484, Count 2976, Avg 0.022021
kernel_apply_3d_boundary_conditions_level1_x(16)CPU:  Total 1087.906982, Count 68948, Avg 0.015779
kernel_apply_3d_boundary_conditions_level1_x(4)CPU:   Total 871.024109, Count 60729, Avg 0.014343
kernel_apply_3d_boundary_conditions_level1_x(8)CPU:   Total 1857.472412, Count 126996, Avg 0.014626
kernel_apply_3d_boundary_conditions_level1_y(16)CPU:  Total 1045.376831, Count 68948, Avg 0.015162
kernel_apply_3d_boundary_conditions_level1_y(4)CPU:   Total 817.397339, Count 60729, Avg 0.013460
kernel_apply_3d_boundary_conditions_level1_y(8)CPU:   Total 1818.705933, Count 126996, Avg 0.014321
```

```
kernel_apply_3d_boundary_conditions_level1_z(16)CPU:  Total 1178.512573, Count 68948, Avg 0.017093
kernel_apply_3d_boundary_conditions_level1_z(4)CPU:   Total 842.578735, Count 60729, Avg 0.013874
kernel_apply_3d_boundary_conditions_level1_z(8)CPU:   Total 2086.350830, Count 126996, Avg 0.016428
kernel_apply_3d_mac_boundary_conditions_level1_x(16)CPU:      Total 666.994446, Count 30630, Avg 0.021776
kernel_apply_3d_mac_boundary_conditions_level1_x(8)CPU:       Total 116.068733, Count 5954, Avg 0.019494
kernel_apply_3d_mac_boundary_conditions_level1_y(16)CPU:      Total 639.755981, Count 30630, Avg 0.020887
kernel_apply_3d_mac_boundary_conditions_level1_y(8)CPU:       Total 114.884789, Count 5954, Avg 0.019295
kernel_apply_3d_mac_boundary_conditions_level1_z(16)CPU:      Total 742.733643, Count 30630, Avg 0.024249
kernel_apply_3d_mac_boundary_conditions_level1_z(8)CPU:       Total 136.168701, Count 5954, Avg 0.022870
reduce_kernelCPU:  Total 1786.051880, Count 93822, Avg 0.019037

FLOPS

Total 52212.320312ms
---------------------
 36.90% - Sol_MultigridPressure3DDevice::relax_on_host(4)CPU
 5.08% - Sol_MultigridPressure3DDeviceD_relax(16)CPU
 4.00% - kernel_apply_3d_boundary_conditions_level1_z(8)CPU
 3.84% - Sol_MultigridPressure3DDeviceD_relax(32)CPU
 3.56% - kernel_apply_3d_boundary_conditions_level1_x(8)CPU
 3.48% - kernel_apply_3d_boundary_conditions_level1_y(8)CPU
 3.42% - reduce_kernelCPU
 2.64% - Sol_LaplacianCentered3DDevice_stencilCPU
 2.29% - cudaMemcpyCPU
 2.26% - kernel_apply_3d_boundary_conditions_level1_z(16)CPU
 2.19% - Grid3DDevice_linear_combination2_3DCPU
 2.08% - kernel_apply_3d_boundary_conditions_level1_x(16)CPU
 2.00% - kernel_apply_3d_boundary_conditions_level1_y(16)CPU
 1.67% - kernel_apply_3d_boundary_conditions_level1_x(4)CPU
 1.61% - kernel_apply_3d_boundary_conditions_level1_z(4)CPU
 1.57% - kernel_apply_3d_boundary_conditions_level1_y(4)CPU
 1.55% - cudaMallocCPU
 1.49% - Sol_MultigridPressure3DDeviceD_calculate_residual(32)CPU
 1.43% - cudaMemcpy(DeviceToHost)CPU
 1.42% - kernel_apply_3d_mac_boundary_conditions_level1_z(16)CPU
 1.28% - kernel_apply_3d_mac_boundary_conditions_level1_x(16)CPU
 1.23% - kernel_apply_3d_mac_boundary_conditions_level1_y(16)CPU
 1.21% - cudaFreeCPU
 1.17% - Advection3DD_apply_upwind_TEXCPU
 1.10% - Sol_MultigridPressure3DDeviceD_prolong(8)CPU
 1.09% - cudaMemsetCPU
 0.99% - Sol_MultigridPressure3DDeviceD_calculate_residual(16)CPU
 0.87% - Sol_MultigridPressure3DDeviceD_prolong(16)CPU
 0.85% - Sol_PassiveAdvection3D_apply_interpCPU
 0.84% - cudaMemcpy(HostToDevice)CPU
 0.73% - Sol_MultigridPressure3DDeviceD_restrict(8)CPU
 0.72% - Sol_Gradient3DDevice_subtract_gradCPU
 0.67% - kernel_apply_3d_boundary_conditions_level1_nocorners(16)CPU
 0.64% - Sol_MultigridPressure3DDeviceD_restrict(16)CPU
 0.59% - Sol_Divergence3DDevice_calculate_divergence::calculate_divergenceCPU
 0.56% - Eqn_IncompressibleNS3D_add_thermal_forceCPU
 0.26% - kernel_apply_3d_mac_boundary_conditions_level1_z(8)CPU
 0.22% - kernel_apply_3d_mac_boundary_conditions_level1_x(8)CPU
 0.22% - kernel_apply_3d_mac_boundary_conditions_level1_y(8)CPU
 0.13% - kernel_apply_3d_boundary_conditions_level1_nocorners(8)CPU
 0.01% - cudaFreeHostCPU
 0.01% - cudaHostAllocCPU
 [PASSED]
```