OPTIMIZATION AND PARALLELIZATION OF DIFFUSION SOLVERS USING ALTERNATING DIRECTION IMPLICIT AND RANDOM WALK METHODS

Samuel Maloney

Department of Mathematics
ETH Zürich
Zürich, Switzerland

ABSTRACT

An optimized code for diffusion is presented.

1. BACKGROUND

Motivation. Diffusion is the process by which a quantity of interest spreads from regions of high density to regions of low density and are integral in the study of fluids. Diffusion terms arise in all numerical models of computational fluid dynamics, which are used for such problems as aerodynamic design, turbine flows, chemical reaction mixing, and many others. Efficient and accurate simulation codes for diffusion are thus of great importance in many fields.

In this study, two different algorithms to simulate the diffusion process, Alternating Direction Implicit (ADI) and a Random Walk (RW), were implemented, optimized, and parallelized. The mathematical formulations for diffusion in general and for each of these two methods specifically are presented next.

Mathematical Formulation. The process of diffusion is driven by it's conentration gradient according to the diffusion equation:

$$\frac{\partial \rho(\mathbf{r},t)}{\partial t} = D\Delta\rho(\mathbf{r},t) \tag{1}$$

where $\rho(\mathbf{r},t)$, with position $\mathbf{r}$ at time t, is the quantity of interest and D is the diffusion constant and is a given value for the system. For this study, homogeneous Dirichlet boundary conditions are used at all boundaries. The initial density distribution used for this study is:

$$\rho(x,y,0) = \sin(\pi x)\sin(\pi y) \tag{2}$$

and it's corresponding analytic solution, which was used for verification, is:

$$\rho(x,y,t) = \sin(\pi x)\sin(\pi y)e^{-2D\pi^2 t} \tag{3}$$

The studied simulation domain is a unit square with x and y each ranging from 0 to 1, which is discretized using a uniform grid such that $\delta h = \delta x = \delta y$. A superscript n is used to denote the timestep $t_n = n\delta t$ and subscripts i and j to denote the indices of the nodes in the mesh $x_i = i\delta x$ and $y_j = j\delta y$ such that $\rho^n_{i,j} = \rho(x_i, y_j, t_n)$.

Alternating Direction Implicit. Lorem Ipsum dolor sit amet.

Random Walk. Lorem Ipsum dolor sit amet.

2. BASELINE IMPLEMENTATION

In this section, we provide an overview of the algorithm

ADI. Lorem Ipsum dolor sit amet.

RW. Lorem Ipsum dolor sit amet.

Cost Analysis. We define the cost of the code as the number of floating point operations (FLOPs) including additions, multiplications, and, for the ADI baseline implementation only, divisions. Moreover, the data movement in bytes to and from various levels of cache was measured in order to quantify the operational intensity.

The ADI complexity is roughly $O(N^2)$ as it does a constant number of operations for calculating the updated value at each point in the grid.

Scaling for the RW method depends on which version is being considered. The baseline implementation scales as $O(M)$, where M is the number of particles in the simulation. This is because a random number must be generated for each particle in the simulation to determine it's movement for a given timestep, independent of how many grid points there are in the simulation. The later versions that use a binomial random variable (see 3.2) to simulate particle movement scale as $O(N^2)$ because exactly four such random variables must be evaluated for each grid point, independent of the number of particles in the simulation.

3. OPTIMIZATION METHOD

In this section we explain the main optimizations that were undertaken. We discuss the optimizations of the ADI and RW methods in separate subsections.

3.1. ADI

We divide the optimizations in 4 sets called *revisions*.

Scalar. This revision consists mainly of

AVX. In this revision we used

OpenMP. This revision was targeted at

3.2. RW

TheRW optimizations can be similarly broken down into revisions, simlarily to those for the assembly.

Scalar. This is a direct implentation

AVX. Vectorization of the code was done by leveraging Intel fused-multiply-add (FMA) and advanced vector extension (AVX) intrinsics.

OpenMP. stuff...

4. OPTIMIZATION RESULTS

In this section the results of numerical benchmarks of the code at various stages are presented. Runtime, performance, and roofline analyses were performed where applicable.

Experimental setup. All data was collected using a single XC50 node of Piz Daint, which has a 12 core Intel Xeon E5-2690 v3 running at 2.6 GHz [1]. Each physical processor core has a 64 KB L1 cache split equally into two 32 KB data and instruction caches, and a 256 KB unified L2 cache [2]. The 30 MB L3 cache is shared between all cores [3], and there is 24.9 GB/s bandwidth to main memory [4]. The Intel C++ Compiler v17.0.1 was used with "-O3 -std=c++11 -DNDEBUG -march=core-avx2 -fno-alias -qopenmp -mkl" flags. The Intel time stamp counter (TSC) was used for runtime measurements and CrayPat was used for cache miss measurements.

The inputs are the number of grid points in domain discretization, the time stepbetween iterations, and the number of iterations performed. We used grids of various sizes, up to $N \times N = 7680 \times 7680$. The diffusion constant $D = 1$ is used for all simulations. All experiments were performed with warm cache.

4.1. ADI

Roofline, runtime, and scaling experiments were conducted on the ADI code.

Results. The runtime plots of various revisions of the ADI code (see 3.1) are shown in Fig. 1. Each consecutive revision of the code resulted in a speedup.

4.2. RW

Benchmarking experiments were conducted on the RW code

Results. Results of the roofline, performance, and runtime analyses of the RW code will now be presented.

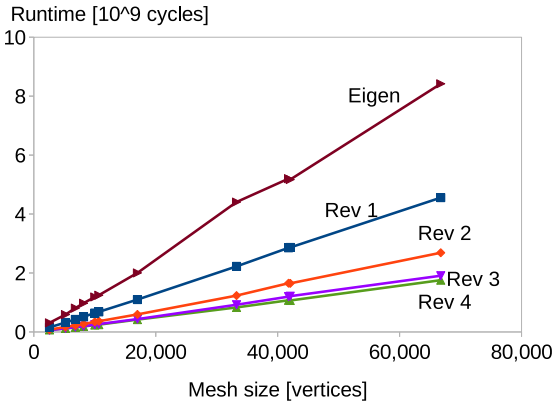

Fig. 1. Runtime analysis of the matrix assembly code.

5. CONCLUSION

Conclude things

6. REFERENCES

[1] CSCS. (Accessed 2017, Aug). *Piz Daint* [Online]. Available: http://www.cscs.ch/computers/piz_daint/

[2] CPU-World. (Accessed 2017, Aug). *Intel Xeon E5-2690 v3 specifications* [Online]. Available: http://www.cpu-world.com/CPUs/Xeon/Intel-Xeon%20E5-2690%20v3.html

[3] Intel Corporation. (Accessed 2017, Aug). *Intel Xeon Processor E5-2690 v3* [Online]. Available: http://ark.intel.com/products/81713/Intel-Xeon-Processor-E5-2690-v3-30M-Cache-2_60-GHzl

[4] 7-CPU. (Accessed 2017, Aug). *Intel Haswell* [Online]. Available: http://www.7-cpu.com/cpu/Haswell.html

[5] user2927848. (2016, Mar 23). *m256d TRANSPOSE4 Equivalent* [Online]. Available: https://stackoverflow.com/questions/36167517/m256d-transpose4-equivalent

[6] T. van den Berg. (Accessed 2017, Aug). *High Quality C++ Parallel Random Number Generator* [Online]. Available: https://www.sitmo.com/?p=1206

[7] Intel Corporation. (2016, Jun). *Intel 64 and IA-32 Architectures Optimization Reference Manual* [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html