OpenACC performance for simulating 2D radial dambreak using FVM HLLE flux

P H Gunawan and M R Pahlevi
School of Computing, Telkom University, Jalan Telekomunikasi No. 1 Terusan Buah Batu, Bandung 40257, Indonesia.
E-mail: phgunawan@telkomuniversity.ac.id; rezhapzz@student.telkomuniversity.ac.id

Abstract. The aim of this paper is to investigate the performances of openACC platform for computing 2D radial dambreak. Here, the shallow water equation will be used to describe and simulate 2D radial dambreak with finite volume method (FVM) using HLLE flux. OpenACC is a parallel computing platform based on GPU cores. Indeed, from this research this platform is used to minimize computational time on the numerical scheme performance. The results show the using OpenACC, the computational time is reduced. For the dry and wet radial dambreak simulations using 2048 grids, the computational time of parallel is obtained 575.984 s and 584.830 s respectively for both simulations. These results show the successful of OpenACC when they are compared with the serial time of dry and wet radial dambreak simulations which are collected 28047.500 s and 29269.40 s respectively.

1. Introduction
Shallow water equations (SWE) are the set of hyperbolic equations which have two parts, called mass and momentum conservation. This set of equation is also known as Saint-Venant equations. Several applications for describing the fluid flows using shallow water model are available. For instance, the model is used to describe the water flow of rivers, ocean, channels, and etc [1]. In two-dimensional, the shallow water equations are written as

\[ U_t + F(U)_x + G(U)_y = 0 \]  

where

\[ U = (h, hu, hv)^T, \]  

\[ F(U) = \left( hu, hu^2 + \frac{1}{2}gh^2, huv \right)^T, \]  

\[ G(U) = \left( hv, huv, hv^2 + \frac{1}{2}gh^2 \right)^T. \]  

The notation \( h(x, y, t) \) describes the water depth, \( g \) the gravitational force, \( u(x, y, t) \) and \( v(x, y, t) \) the velocity in \( x \) and \( y \)-direction respectively. Moreover, the variables \( x \) and \( y \) are space, and \( t \) is time (see Figure 1 for more detail about the situation).
The robust scheme for approximating (1 - 4) is known as collocated scheme of Finite Volume Method (FVM) [2, 3, 4, 5] and [6]. This scheme has widely used to simulate the fluid flow and has been proved to satisfy the entropy condition.

The discretization of two-dimensional of SWE using FVM produces a large discrete points (grids). Thus, this will lead to the long computational time for solving all discrete variables in all grids. Several ideas to reduce the computation in time is available. One of the examples can be used is to the parallel computing techniques. for instance, in the references [7, 8, 9, 10, 11, 12, 13, 14, 15] and [16] a good performance of parallel computing in accelerate the numerical computation is shown satisfied. Therefore, the aim of this paper is to show the performance of parallel computing for simulating dry-wet radial dambreak cases in two-dimensional. In this case, the parallel platform OpenACC (Open Accelerator) will be used.

2. Finite Volume Method for Shallow Water Equations

Given the control volume $V_{i,j}$ over domain $\Omega := [0, L_x] \times [0, L_y]$, in the discrete space point $i,j$ as shown in Figure 2. Since this control volume is on 2D spatial domain, thus $V_{i,j}$ can be defined on $(x_{i-1/2}, x_{i+1/2}) \times (y_{j-1/2}, y_{j+1/2})$ at time $t^n$ to $t^{n+1}$. Using the uniform discrete space, the space is discretized by $N_x \times N_y \in \mathbb{N}^2$ such that $x_i = i \times \Delta x$, $y_j = j \times \Delta y$ with $i,j \in \mathcal{M} = \{0,1,\cdots,N_x\} \times \{0,1,\cdots,N_y\}$. Therefore the space step for $x$ and $y$-direction are given as $\Delta x = L_x/N_x$ and $\Delta y = L_y/N_y$ respectively.

In another hand, the time is discretized with time step $\Delta t$ which is defined by the CFL (Courant-Frederich-Levy) condition (11). Thus, the discrete space $t^n$ can be defined as $t^n = n \times \Delta t$ with $n \in \mathcal{T} = \{0,1,\cdots\}$.

In order to approximate the shallow water equations (1), the discrete notation of $U_{i,j}^n$ is defined as the average approximation over control volume as

$$U_{i,j}^n \approx \frac{\int_{V_{i,j}} U(x,y,t^n) \; d\Omega}{|V_{i,j}|}, \quad \forall i,j \in \mathcal{M}, \; \forall n \in \mathcal{T}. \quad (5)$$

Finally, the discretization of shallow water equations in finite volume method can be written
Figure 2. The control volume of discretization space and time. The space steps $\Delta x$ and $\Delta y$, and time step $\Delta t$ are set to be uniform.

as follows

$$U_{i,j}^{n+1} = U_{i,j}^n - \Delta t \left( F_{i+\frac{1}{2},j}^n - F_{i-\frac{1}{2},j}^n + G_{i,j+\frac{1}{2}}^n - G_{i,j-\frac{1}{2}}^n \right), \quad \forall i,j \in \mathcal{M}, n \in \mathcal{T} \quad (6)$$

Indeed in the finite volume method, the interesting part is to approximate the numerical fluxes $F_{i+\frac{1}{2},j}^n$ and $G_{i,j+\frac{1}{2}}^n$. Several numerical fluxes can be found in some references [6, 2]. In this paper, the HLLE (Harten, Lax, van Leer and Einfeld) numerical flux will be used for simplicity and robustness. The numerical fluxes are given as

$$F_{i+\frac{1}{2},j}^n = \mathcal{F}(U_{i,j}^n, U_{i+1,j}^n) = a_1 F(U_{i+1,j}^n) + a_2 F(U_{i,j}^n) - a_3 (U_{i+1,j} - U_{i,j}), \quad (7)$$

for the flux in $x$-direction, and

$$G_{i,j+\frac{1}{2}}^n = \mathcal{G}(U_{i,j}^n, U_{i,j+1}^n) = a_1 G(U_{i,j+1}^n) + a_2 G(U_{i,j}^n) - a_3 (U_{i,j+1} - U_{i,j}), \quad (8)$$

for the flux in $y$-direction.

The numerical fluxes $F(U_{i,j}^n)$ and $G(U_{i,j}^n)$ are the flux function as defined in (3) and (4) with the parameter $U$ at point $i,j$ and current time $n$. Another parameters are defined as

$$a_1 := \frac{\min(\lambda_2, 0) - \min(\lambda_1, 0)}{\lambda_2 - \lambda_1}, \quad a_2 := 1 - a_1, \quad a_3 := \frac{\lambda_2 |\lambda_1| - \lambda_1 |\lambda_2|}{2(\lambda_2 - \lambda_1)}. \quad (9)$$

Moreover, the coefficients $\lambda_1$ and $\lambda_2$ can be obtained in some references such as in [2, 17, 18] for more detail. Thus the final discretization of (6) with numerical fluxes HLLE can be given as follows

$$U_{i,j}^{n+1} = U_{i,j}^n - \frac{\Delta t}{\Delta x} \left( \mathcal{F}(U_{i,j}^n, U_{i+1,j}^n) - \mathcal{F}(U_{i-1,j}^n, U_{i,j}^n) \right) - \frac{\Delta t}{\Delta y} \left( \mathcal{G}(U_{i,j}^n, U_{i,j+1}^n) - \mathcal{G}(U_{i,j-1}^n, U_{i,j}^n) \right), \quad \forall (i,j) \in \mathcal{M}, n \in \mathcal{T}. \quad (10)$$
The numerical scheme (10) has the stability condition. It stability is controlled by the following CFL (Courant-Frederich-Levy) condition
\[ \Delta t \leq \frac{\Delta \Psi}{\max_i \max_j |u_{i,j}| \pm \sqrt{gh_{i,j}}, |v_{i,j}| \pm \sqrt{gh_{i,j}}} \] (11)
Here, grid cells is given in uniform for x and y-direction, hence \( \Delta \Psi = \Delta x = \Delta y \), with \( 0 < \nu \leq 1 \) which is called the Courant number.

3. Parallel Algorithm using OpenACC Platform
OpenACC is known as a programming platform for parallel computing. This programming platform is developed by several enterprises such as Cray, CAPS, Nvidia, and PGI [19, 20]. OpenACC is designed to porting the codes with wide variety of heterogeneous HPC hardware platform and architectures with less programming effort than required with a low-level model. Moreover, OpenACC supports C, C++, and Fortran programming languages and multiple hardware architectures. In order to accelerate the application with GPU or accelerator, OpenACC has porting cycle is used.

The preparation steps for using OpenACC is given as follows:

(i) Before start to use OpenACC in an application, the observation of routines and loops which are the most spending the bulk of time should be done. The advantage of this observation indeed to maximize the acceleration of computational time in the application.

(ii) After the important parts of the programming code have been identified, the OpenACC directives should be used to accelerate these regions on the accelerator. Parallel loops within the code should be decorated with \#pragma acc loop directives to provide OpenACC compiler information to parallelize the code (see Figure 3).

(iii) In the programmers side, the additional instruction to compiler should be done. For instance, how to manage the local memory to the accelerator and to keep the information moved between two memories if absolutely necessary.

(iv) Another important step is when the information coming from the CPU architecture to GPU architecture, restructuring the loops should be done in order to reduce the frequency of data movement.

Previous steps have been successful in many applications since its simplicity and applicable such that the programmer can quickly and productively achieve the acceleration of time. Therefore, in this paper OpenACC will be used to accelerate the simulation time.

In Figure 3, the parallel programming can be started in initialization value of \( U_{i,j}^{n=0} \). Then the initialized value will be sent to the accelerator only once with Peripheral Component Interconnect express (PCIe), it is because transfer value between CPU and GPU is very expensive. It has been explained in [21], the value from the host for each routines and loops will be mapped automatically inside gangs, where a gang consists of several workers. The first parallel part is given to compute \( \Delta t \) for CFL condition, then the second part is given to compute primitive variables at new time step \( h_{ij}^{n+1}, u_{ij}^{n+1}, \) and \( v_{ij}^{n+1} \). The third part of parallel is given for boundary condition. Once those parts are done, the results will be sent back to the CPU for the next iteration.

4. Numerical Results and Parallel Performance
All the following numerical results are compiled from HPC lab work server School of Computing, Telkom University with the specifications as shown in Table 1.
Figure 3. The flowchart of parallel computation.

Table 1. The computer specifications for running the numerical simulations.

| Name               | Type                        |
|--------------------|-----------------------------|
| Operating System   | Ubuntu 16.10                |
| Processors         | Intel(R) Xeon(R) 2 socket @16core |
| RAM                | 6GB                         |
| GPU                | Nvidia GTX670               |
| CUDA cores         | 1024                        |

4.1. Dry and wet radial dambreak simulations

In order to see the performance of parallel computing, two numerical simulations of radial dambreak problem are given. Here, two numerical simulations of radial dambreak which are dry and wet cases are given. Following are the initial configuration of two radial dambreak problems on domain $\Omega = [0, 1] \times [0, 1]$ and at a middle point $(x_0, y_0) = (0.5, 0.5)$. 
The initial configurations of dry radial dambreak (water height and fluxes) are given as

\[ h(x, y, 0) = \begin{cases} 
1, & \text{if } \sqrt{(x-x_0)^2 + (y-y_0)^2} \leq \frac{3}{10}, \\
0, & \text{otherwise},
\end{cases} \tag{12} \]

\[ q(x, y, 0) = p(x, y, 0) = 0, \tag{13} \]

where \( q(x, y, t) = h(x, y, t)u(x, y, t) \) and \( p(x, y, t) = h(x, y, t)v(x, y, t) \).

The results of water surface and velocity in \( x \) and \( y \)-direction profiles of dry radial dambreak simulation are shown in Figures 4 and 5 respectively. From Figure 4, the numerical method is shown able to handle the vacuum problem. The water surface smoothly flow over dry bed. In this simulation, the final time of simulation is \( t = 0.03 \) s.

![Figure 4](image1.png)

**Figure 4.** Simulation results of dry radial dambreak problem. (Left) The slice profile of dry radial dambreak at \( y = 0.5 \). (Right) the 3D view of water height profile dry bed.

![Figure 5](image2.png)

**Figure 5.** The velocity profile of dry radial dambreak simulation. (Left) velocity profile in \( x \)-direction \( (u) \). (Right) velocity profile in \( y \)-direction \( (v) \).

In other side, the initial configurations of wet radial dambreak are given as follow

\[ h(x, 0) = \begin{cases} 
1, & \text{if } \sqrt{(x-x_0)^2 + (y-y_0)^2} \leq \frac{3}{10}, \\
0.2, & \text{otherwise},
\end{cases} \tag{14} \]

\[ q(x, y, 0) = p(x, y, 0) = 0. \tag{15} \]
This initial conditions of wet radial dam break problem are for water depth and velocity in $x$ and $y$-direction profiles. The results of this simulation are shown in Figures 6 and 7.

**Figure 6.** Simulation results of wet radial dam break problem. (Left) The slice profile of wet radial dam break at $y = 0.5$. (Right) the 3D view of water height profile wet bed.

**Figure 7.** The velocity profile of wet radial dam break simulation. (Left) velocity profile in $x$-direction ($u$). (Right) velocity profile in $y$-direction ($v$).

Here, the water surface profile of wet radial dam break is shown in Figure 6. Moreover, the velocity in $x$ and $y$-direction profiles are shown in Figure 7(Left) and Figure 7(Right) respectively. It can be seen in Figure 6(Left), the wet radial dam break produces the shock waves in around $0.1$ and $x = 0.9$. Indeed this shock is also shown in one-dimensional problem (see [1] for more detail). Our focus in this paper is to reduce the computational time of this simulation by using large discrete points. Next section will present the result of parallel computing using OpenACC with GPU architecture model.

4.2 Parallel performance

The computational time in serial and parallel for dry and wet radial dam break simulation can be found in Table 2. Here, the results are obtained from uniform discrete points $N_x = N_y$ (follow the reference [22]), hence the simulations use several numbers of discrete points $(N_x \times N_y) = (512 \times 512), (1024 \times 1024), (2048 \times 2048),$ and $(4096 \times 2048)$ with final time $T = 1$ s. Moreover, in this simulation, the number of threads for parallel computation is used is 128 threads in GPU.
Table 2. Computational time in serial and parallel for dry and wet radial dambreak simulation.

| Grids     | Serial time for dry bed (s) | Parallel time for dry bed (s) | Serial time for wet bed (s) | Parallel time for wet bed (s) |
|-----------|-----------------------------|-------------------------------|-----------------------------|-------------------------------|
| 512 × 512 | 359.245                     | 8.495                         | 404.898                     | 8.956                         |
| 1024 × 1024 | 3272.480                   | 70.760                        | 3414.270                    | 71.460                        |
| 2048 × 2048 | 28047.500                   | 575.984                       | 29269.400                   | 584.830                       |
| 4096 × 4096 | Nan                         | 4823.369                      | Nan                         | 4881.460                      |

As shown in Table 2, unfortunately, the computer could not get the exact time in serial using grid number 4096 due to lack of computer resource to compute such large grid. Here, serial experiment using 4096 grids took more than 3 days. Thus, this computational time is denoted by Nan. However, by using OpenACC, the computational time for dry and wet simulation are obtained 4823.369 s and 4881.460 s respectively.

From the Table 2, the computational time of parallel is shown significantly increasing along the increasing of discrete points. However, compared with the serial, indeed the parallel programming by OpenACC is successfully reduce the computational cost. For instance using grid sizes 2048, the serial time for dry and wet radial dambreak simulation are obtained 28047.500 s and 29269.400 s respectively. Meanwhile, the parallel time for both simulations are collected 575.984 s and 584.830 s respectively. The discrepancy of CPU time of dry and wet dambreak is not to large. This discrepancy can be obtained due to the factor of floating point arithmetic.

Figure 8. The speedup (left) and efficiency (right) in parallel programming for both simulations.

Figure 8 shows the performances of parallel computing using speedup and efficiency measurements. In Figure 8 (left) the speedup is shown increasing for both simulations. Using 2046 number of grids, the speedup for dry and wet radial dambreak is obtained around 48.69 and 50.04 times respectively. Meanwhile, the efficiency for dry and wet radial dambreak is 38.04% and 39.10% respectively. The detail of speedup and efficiency measurement formula can be found in references [12, 23].

5. Conclusion
The application of OpenACC platform to reduce the computational time of 2D dry and wet radial dambreak simulations are given. The modified parallel algorithm of numerical scheme for
approximating shallow water equations are also given. The results show that the computational
time of dry and wet cases using the OpenACC are satisfactory. Here, OpenACC platform can
minimize the time execution for large discrete points. For the dry and wet radial dambreak
simulations, using 2048 grids, the computational time for both simulations are 575.984 s and
584.830 s respectively. Those results show that the OpenACC can reduce the computational time
where the serial time for both simulations are obtained 28047.500 s and 29269.40 s respectively
using 2048 grid sizes. Moreover, the maximum speedup 50.04 times and efficiency 39.10% are
observed at wet radial dambreak simulation by similar number of grids.

References
[1] Doyen D and Gunawan P H 2014 Finite Volumes for Complex Applications VII-Methods and Theoretical
Aspects (Springer) pp 227–235
[2] Delestre O, Cordier S, James F and Darboux F 2008 12th International Conference on Hyperbolic Problems
vol 67 (American Mathematical Society) pp 537–546
[3] Delestre O and Marche F 2011 Journal of Scientific Computing 48 41–51
[4] Audusse E, Bouchut F, Breitsteu M O, Klein R and Perthame B 2004 SIAM Journal on Scientific Computing
25 2050–2065
[5] Gunawan P H and Lhébrard X 2015 Computers & Fluids 121 44–50
[6] Bouchut F 2004 Nonlinear stability of finite Volume Methods for hyperbolic conservation laws: And Well-
Balanced schemes for sources (Frontiers in Mathematics. Birkhäuser Verlag, Basel)
[7] Gunawan P H 2016 Information and Communication Technology (ICoICT), 2016 4th International
Conference on (IEEE) pp 1–5
[8] de la Asunci´ on M, Castro M, Mantas J and Ortega S 2016 Advances in Engineering Software 99 59–72
[9] De La Asunci´ on M, Mantas J M and Castro M J 2011 The Journal of Supercomputing 58 206–214
[10] Brodtkorb A R, Sætra M L and Altinakar M 2012 Computers & Fluids 55 1–12
[11] Mulyani, Putri N D and Gunawan P H 2017 Information and Communication Technology (ICoICT), 2017
5th International Conference on (IEEE) pp 1–5
[12] Pahelevi M R and Gunawan P H 2017 Information and Communication Technology (ICoICT), 2017 5th
International Conference on (IEEE) pp 1–4
[13] Juliati S and Gunawan P H 2017 Information and Communication Technology (ICoICT), 2017 5th
International Conference on (IEEE) pp 1–5
[14] Alamsyah M N A, Simanjuntak C A, Bagustara B A R H, Pradana W A and Gunawan P H 2017 Information
and Communication Technology (ICoICT), 2017 5th International Conference on (IEEE) pp 1–6
[15] Iryanto and Gunawan P H 2017 Information and Communication Technology (ICoICT), 2017 5th
International Conference on (IEEE) pp 1–5
[16] Castillo D, Ferreiro A, García-Rodríguez J A and Vázquez C 2013 Applied Mathematics and Computation
219 11233–11257
[17] LeVeque R J 2002 Finite volume methods for hyperbolic problems vol 31 (Cambridge university press)
[18] Toro E F 2013 Riemann solvers and numerical methods for fluid dynamics: a practical introduction (Springer
Science & Business Media)
[19] Wienke S, Terboven C, Beyer J C and Müller M S 2014 European Conference on Parallel Processing (Springer)
pp 812–823
[20] Wienke S, Springer P, Terboven C and an Mey D 2012 Euro-Par 2012 Parallel Processing 859–870
[21] Kirk D B and Wen-Mei W H 2013 Programming massively parallel processors: a hands-on approach (Morgan
kaufmann)
[22] Cheng J, Grossman M and McKercher T 2014 Professional Cuda C Programming (John Wiley & Sons)
[23] Rauber T and Rünger G 2013 Parallel Programming: For multicores and cluster systems (Springer Science &
Business Media)