12th International Conference on Hydroscience & Engineering
Hydro-Science & Engineering for Environmental Resilience
November 6-10, 2016, Tainan, Taiwan.

Optimizing Two-Dimensional Flood Model with SSE and Concurrent Processing

Satoshi Yamaguchi[1] , *Shigehito Yamaho*[2]
1. Center for Technology Innovation – System Engineering, R&D Group, Hitachi Ltd.
Kokubunji, Tokyo, Japan
2. ICT Solution Dept., Hitachi Power Solutions Co., Ltd.
Chiyoda, Tokyo, Japan

ABSTRACT

We optimized two-dimensional flood simulation model by using Streaming SIMD Extension (SSE/SSE2) and Parallel Patterns Library which enables concurrent processing, and made our program 2 to 4 times faster than normal implementation on a Personal Computer (PC) with a multi-core Central Processing Unit (CPU). Because SSE and concurrent processing is supported by almost all PCs, proposed method is applicable to wide variety of PCs (laptop, desktop, workstation, etc.).

KEY WORDS: 2D flood model; shallow water equations; Streaming SIMD Extensions (SSE); concurrent processing; dynamic domain defining method (Dynamic DDM), urban flood resilience; DioVISTA Flood.

INTRODUCTION

Two-dimensional (2D) flood model is an important tool in flood risk management. The model outputs time series of flood depth distribution, which is essential information for flood hazard mapping, flood warning, emergency response, flood insurance programs, and so on. A variety of software packages with the 2D flood modelling solvers are available on the market (Néelz and Pender, 2013).

The accuracy of the flood modeling is drastically improved by using high-resolution digital elevation model (DEM). High-resolution DEM is already available in many areas. For example, Japanese major cities have been covered by Light Detection and Ranging (LiDAR)-based DEM with 5-m spatial resolution.

In order to use the high-resolution DEM, it is necessary to use a finer grid in the flood modeling. A finer gird makes computational burden heavier. For example, if we reduce the grid size to half, the number of the grid is quadrupled. The model solver might require reducing its time step length (dt) to half, which makes the number of calculation steps double. As a result, computational burden becomes 8 times in this case. A real situation is sometimes more serious. We sometimes want to reduce the grid size from 50 m (typical conventional DEM resolution) to 5 m (typical LiDAR-base DEM resolution), then computational burden might become 1000 times.

Demand for rapid calculation methods is strong. Recent promising approach is general-purpose computing on graphic processing units (GPGPU). For example, Kalyanapu et al. (2011) implemented their flood model in CUDA, and calculation time was drastically reduced (1/88 to 1/80, compared to normal implementation in their test cases). Several software packages also provide GPGPU version of flood model solvers (Néelz and Pender, 2013).

Although utilizing GPGPU technology is a reasonable approach, conventional personal computers (PCs), which are not suitable for GPGPU, are still widely used in many organizations such as engineering consulting firms, governmental agencies, and universities. In this paper, we propose optimization methods without GPGPU. Our objective is to develop a rapid flood model solver, which works on conventional PCs.

PROPOSED METHODS

Governing Equations

2D shallow water equations are used. The equations are as follows:

$$\frac{\partial h}{\partial t}+\frac{\partial uh}{\partial x}+\frac{\partial vh}{\partial y}=0 \tag{1}$$

$$\frac{\partial uh}{\partial t}+\frac{\partial u^2h}{\partial x}+\frac{\partial uvh}{\partial y}=-gh\frac{\partial H}{\partial x}-\frac{gn^2u\sqrt{u^2+v^2}}{h^{1/3}} \tag{2}$$

$$\frac{\partial vh}{\partial t}+\frac{\partial uvh}{\partial x}+\frac{\partial v^2h}{\partial y}=-gh\frac{\partial H}{\partial y}-\frac{gn^2v\sqrt{u^2+v^2}}{h^{1/3}} \tag{3}$$

where, h is the water depth, H is the water surface elevation, u is the velocity in the x-direction, v is the velocity in the y-direction, t is the time, g is the acceleration due to gravity, and n is the Manning's roughness coefficient.

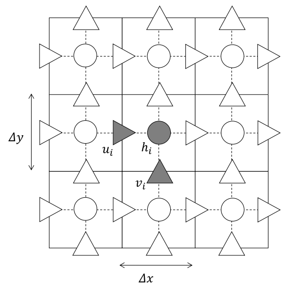


Fig. 1 Variable position in a staggered cell (h: depth, u: velocity in the x-direction, v: velocity in the y-direction, variable positions of i-th cell are shaded)

Rectangle cell is used for spatial discretization. Variables are allocated as a staggered grid (Fig. 1). A first-order upwind finite difference numerical scheme is used. Both the continuity and momentum equations are solved explicitly. The time step length (dt) is automatically determined to meet the Courant-Friedrich-Lewy (CFL) stability condition.

Dynamic DDM

We use dynamic domain defining method (Dynamic DDM, Yamaguchi et al., 2007). The Dynamic DDM automatically expands or shrinks the calculation area during the simulation.

In conventional 2D flood model solver, we need to define calculation domain in advance to the calculation (Fig. 2a). The calculation domain must include the whole flood area (wet cells) and should exclude non-flood area (dry cells). Since flood area and non-flood area are not known until the calculation is finished, the user defines the domain by trial and error.

In the Dynamic DDM, the calculation domain is automatically defined to include all wet cells and to exclude dry cells during the simulation. The modeling engine divides the entire space into subdomains. In Fig. 2b, subdomain consists of 4x4 cells. The modeling engine detects wet cells (Fig. 2b1), and loads the subdomains in which those cells are included (Fig. 2b2). Data in the subdomain is fetched from our spatial database. The equations are solved only at the loaded subdomains. Neighboring subdomains are automatically loaded when the water reaches the current domain boundary (Fig. 2b3). A subdomain will be unloaded when all the cells in the subdomain become dry. Thus, the Dynamic DDM keeps nearly optimal calculation domain during the simulation.

The efficiency of the Dynamic DDM depends on the number of cells in a subdomain. Usage of smaller subdomain results in a better optimization of the calculation domain but more frequent access to GIS is required. Yamaguchi & Iwamura (2007) compared the calculation time of flood simulation with 50 x 50 m cell for three size of subdomain (16 x 16, 32 x 32, and 64 x 64 cells). They concluded that when flood area is 1 – 100 km^2 (400-40,000 cells), 32 x 32-cell subdomain is most efficient. Therefore, we use the Dynamic DDM with 32 x 32-cell subdomain (Fig. 3). The ghost cells (copy of cells in neighboring subdomain) are also shown as shaded cells.

Implementation Using SSE and PPL

We implemented the model using C++ in Microsoft Visual Studio 2015, especially its intrinsic functions of Streaming SIMD Extensions (SSE and SSE2) and Parallel Patterns Library (PPL).

The SSE and SSE2 were introduced in Intel Pentium III in 1999, and in Intel Pentium 4 in 2001, respectively. Now almost all Windows PCs support those instruction sets. Thus, we can take advantage of SSE and SSE2 even on conventional PCs. SSE introduces a 128-bit width data type `__m128` which can contain four 32-bit floating point (`float`) values. Though a normal arithmetic instruction processes single `float` values at a time, a SSE arithmetic instruction processes single `__m128` value at a time. Normal and SSE sample code are shown in Table 1 and Table 2, respectively. We made SSE wrapper class (`float4`) by used C++ operator overloading, and the SSE code is similar to normal code. This facilitates code implementation and maintenance. Note that some heavy arithmetic instructions such as square root (`_mm_sqrt_ps`) are not used. Those instructions are replaced with as normal functions (such as `sqrt`) and conducted only if the cell is wet. Thus the rate of wet/dry cells in a subdomain affects the calculation time.

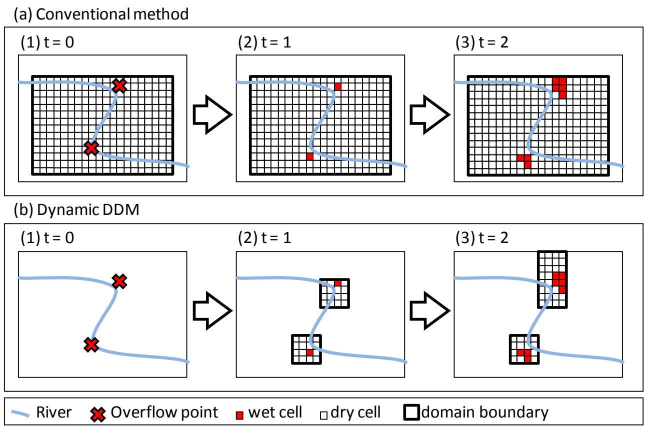


Fig. 2 Workflows of inundation models with (a) conventional method and (b) our Dynamic Domain Defining Method (Dynamic DDM)

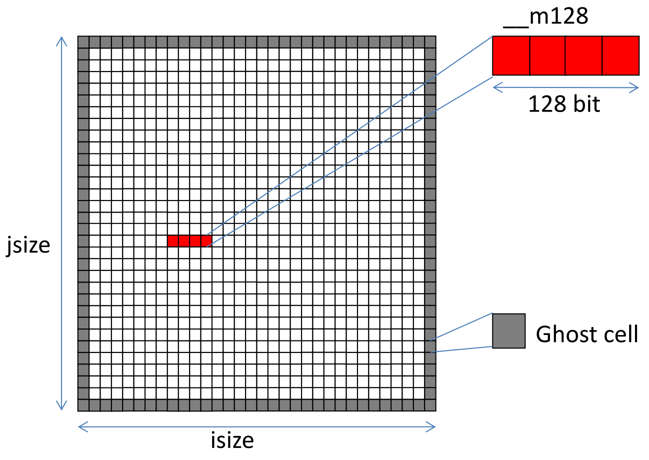


Fig. 3 Subdomain containing 32 x 32*cell

Table 1 Sample C++ code (normal version)

```
float var0[4] = { 0, 1, 2, 3 };
float var1[4] = { 4, 5, 6, 7 };
float var2[4];
for (int k = 0; k < 4; k++) {
        var2[k] = var0[k] + var1[k];
}
```

Table 2 Sample C++ code (SSE version)

```
#include <emmintrin.h>
__declspec(align(16)) float var0[4] = { 0, 1, 2, 3 };
__declspec(align(16)) float var1[4] = { 4, 5, 6, 7 };
__declspec(align(16)) float var2[4];
__m128 xmm0 = _mm_load_ps(var0);
__m128 xmm1 = _mm_load_ps(var1);
__m128 xmm2 = _mm_add_ps(xmm0, xmm1);
_mm_store_ps(var2, xmm2);
```

Because each subdomain is independent to other subdomains, we can concurrently process all subdomains. Unlike OpenMP, the PPL provides a dynamic scheduler that adapts to available resources and adjusts the degree of parallelism as workloads change. The PPL creates threads, and a work task (solving the equations in a subdomain) is assigned to a thread. The PPL dynamically moves work tasks which have not yet started to other threads which run out of work tasks. Thus all work tasks finish in the least overall time.

CASE STUDY

System Integration

We integrated the model into our flood modeling software (DioVISTA Flood), which is composed of (a) four-dimensional geographic information system (GIS), (b) spatial database containing worldwide maps, satellite image, digital elevation data, land-use data, and online maps available through the internet, and (c) modeling engine which provides a distributed runoff model, a one-dimensional river model, an empirical levee failure model, and the two-dimensional flood model. Yamaguchi et al. (2012) illustrates the system in detail.

Experiment Setting

Calculation time of the model was measured by three kinds of PCs (laptop, desktop, and workstation). Each PC has different number of CPU cores (2, 4, and 6). Because of Hyper-Threading each core has two virtual cores. The laptop PC is lightweight (1.03 kg) and thin (13.2 to17.9 mm), and highly portable. As shown in Table 3, it has no graphic card suitable for GPGPU. The desktop PC with mini tower computer case is newest one in the three PCs (its manufacturing year is 2016, see Table 4). The workstation is in middle tower computer case (Table 4).

We used the flood event in Fukui flood disaster (2004, Japan). In this event, a 54-m length section of levee was breached on July 18, and 2.3 km^2 area was flooded. We already have validated our model (Yamaguchi & Iwamura, 2007). The comparison between the simulation result and the site investigation conducted by Yamamoto (2007) is shown in Fig. 4.

RESULT AND DISCUSSIONS

We simulated the flood event for 10 hours in 10-m grid. The number of cells and the time step length were automatically adjusted. The results were saved as a binary formatted file. We saved the result in every 10 minutes, so 600 time-slices were saved. The file size was 45.3 MB.

Speedup by using SSE is shown in Fig. 5. The speedup effect is x1.45 to x1.63, and it is almost independent on the PC specs. Speedup by PPL is shown in Fig. 6. The more CPU cores we use, the higher the speedup effect becomes. Multiplying speedup by SSE and speedup by PPL, and you get a close value shown in Fig. 7. This suggests that the effect of SSE is almost independent on the effect of PPL. The calculation time (Fig. 8) indicates that the flood event of 2.3 km^2 flood extent in 10-m grid in 10 hours were simulated in 385 /179 seconds on the laptop, 262 /65 seconds on the desktop, and 406 /93 seconds on the workstation by normal /optimized versions of the solver, respectively.

Our method is similar to Castro et al (2008). They divided the calculation domain into multiple subdomains using domain decomposition method (DDM). Arithmetic operations in a subdomain were implemented by SSE. Solving multiple subdomains is parallelized by using Message Passing Interface (MPI). Major differences between our method and theirs are dividing method of the calculation domain and parallelization with/without load balancing. They used DDM, and we used Dynamic DDM. Dynamic DDM dynamically changes the number of subdomains. In addition, as we mentioned above, the rate of wet/dry cells in a subdomain also affects the calculation time of subdomain in our method. So the load balancing mechanism provided by PPL is important for our implementation.

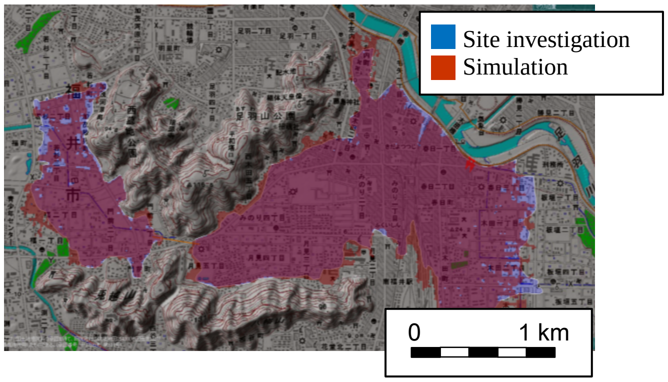


Fig. 4 Comparison between the simulation result with 10-m grid size and the site investigation conducted by Yamamoto (2007).

Table 3 Spec of laptop computer (PC1)

CPU Core	2
CPU	Intel® Core™ i7-5500U CPU @ 2.40GHz
Memory	8.0 GB
GPU	Intel HD Graphics 5500
OS	Windows 8.1 Pro Update, 64-bit
Storage	SSD 256 GB x 1, PCI Express x 4
Computer Model	VAIO Pro 13 mk2
Manufacturing year	2015

Table 4 Spec of desktop computer (PC2)

CPU Core	4
CPU	Intel® Core™ i7-6700 CPU @ 3.4GHz
Memory	16.0 GB
GPU	NVIDIA GeForce GTX 960
OS	Windows 7 Professional, 64-bit (Service Pack 1)
Storage	Seagate Desktop HDD ST500DM0
Computer Model	EPSON Endeavor MR7400
Manufacturing year	2016

Table 5 Spec of workstation (PC3)

CPU Core	6
CPU	Intel® Core™ i7 CPU X 990 @ 3.47GHz
Memory	12.0 GB
GPU	NVIDIA GeForce GTX 570
OS	Windows 10 Pro, 64-bit
Storage	Hitachi HDS721010CLA332
Computer Model	Mouse Computer MDV-AGG9230X
Manufacturing year	2011

CONCLUSIONS

1. We proposed an optimization method of two dimensional flood model using Streaming SIMD Extensions (SSE/SSE2) and concurrent processing. We implemented the model using programing language Microsoft Visual C++, especially its intrinsic functions of SSE and Parallel Patterns Library (PPL). The normal code and optimized code are almost identical by using of C++ operator overloading. This facilitates code implementation and maintenance.
2. We evaluated the optimized effect on three personal computers; laptop PC, desktop PC, and workstation. The normal /optimized versions simulated the 10-hours event in 385 /179 seconds on the laptop, 262 /65 seconds on the desktop, and 406 /93 seconds on the workstation. Speedup effects are x2.16 on the laptop, x4.04 on the desktop, and x4.39 on the workstation. We conclude that optimization with SSE and concurrent processing is effective in speeding up the solver on variety of PCs.

ACKNOWLEDGEMENTS

The application shown in this paper is DioVISTA Flood Version 3.0 by Hitachi Power Solutions Co., Ltd. We used GSI Maps provided by The Geospatial Information Authority of Japan (GSI).

REFERENCES

Castro, M. J., García-Rodríguez, J. A., González-Vida, J. M., & Parés, C. (2008). Solving shallow-water systems in 2D domains using Finite Volume methods and multimedia SSE instructions. Journal of Computational and Applied Mathematics, 221(1), 16-32.

Kalyanapu, A. J., Shankar, S., Pardyjak, E. R., Judi, D. R., & Burian, S. J. (2011). Assessment of GPU computational enhancement to a 2D flood model. Environmental Modelling & Software, 26(8), 1009-1016.

Néelz, S. & Pender, G. (2013). Benchmarking the latest generation of 2D hydraulic flood modelling packages, Environment Agency.

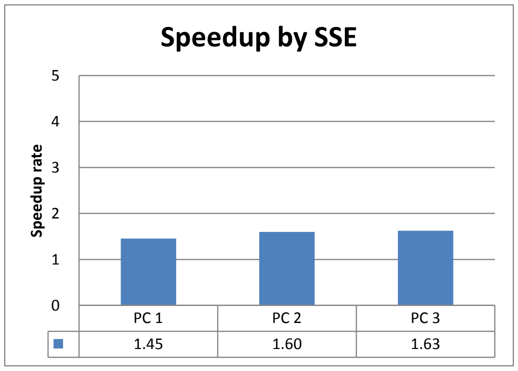
Yamaguchi, S. & Iwamura, K. (2007). Fast Flood Simulation Method Using Dynamic DDM. IPSJ Transaction on Mathematical Modeling and Its Application, 48 (SIG 6 TOM 17), 92-103. (in Japanese)

Yamaguchi, S., Ikeda, T., Iwamura, K., Naono, K., Ninomiya, A., Tanaka, K., & Takahashi, H. (2007). Development of GIS-based flood-simulation software and application to flood-risk assessment. In 2nd IMA International Conference on Flood Risk Assessment.

Yamaguchi, S., and Ikeda, T. (2010). Automatic linking and fast calculation methods of 1D/2D coupled model, In Proceedings of 9th International Conference on HydroScience and Engineering (ICHE), Chennai, India.

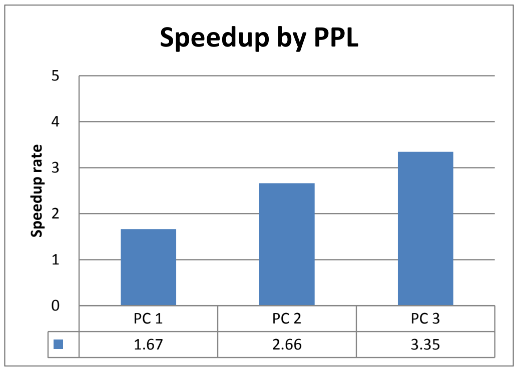
Yamaguchi, S., and Ikeda, T., & Yamaho, S. (2012). Flood risk assessment system for major metropolitan areas in Japan. In Proceedings of 10th International Conference on Hydroinformatics (HIC), Hamburg, Germany.

Yamamoto, H. (2007). Characteristics of Flood Disaster by the Fukui Heavy Rainfall on July 18, 2004 at the Fukui city area on the left bank of the Asuwa River, Fukui Prefecture. Journal of Japan Society for Natural Disaster Science, 26(1), 41-53. (in Japanese)



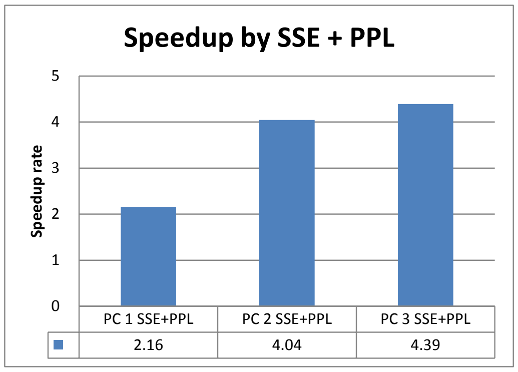
	PC 1	PC 2	PC 3
■	1.45	1.60	1.63

Fig. 5 Optimized effect by using Streaming SIMD Extensions (SSE).



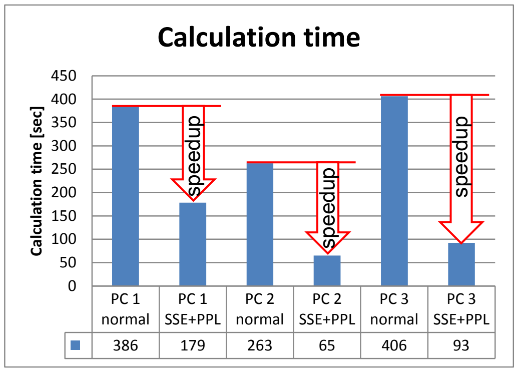
	PC 1	PC 2	PC 3
■	1.67	2.66	3.35

Fig. 6 Optimized effect by using Parallel Patterns Library (PPL).



	PC 1 SSE+PPL	PC 2 SSE+PPL	PC 3 SSE+PPL
■	2.16	4.04	4.39

Fig. 7 Optimized effect by using SSE and PPL.



	PC 1 normal	PC 1 SSE+PPL	PC 2 normal	PC 2 SSE+PPL	PC 3 normal	PC 3 SSE+PPL
■	386	179	263	65	406	93

Fig. 8 Calculation time comparison of normal implementation with optimized implementation by using SSE and PPL.