Environmental Modelling and Software 137 (2021) 104958

Contents lists available at ScienceDirect

Environmental Modelling and Software

journal homepage: http://www.elsevier.com/locate/envsoft

Utilizing dynamic parallelism in CUDA to accelerate a 3D red-black successive over relaxation wind-field solver

Behnam Bozorgmehr [a], Pete Willemsen [b], Jeremy A. Gibbs [a,c], Rob Stoll [a], Jae-Jin Kim [d], Eric R. Pardyjak [a,*]

[a] Department of Mechanical Engineering, University of Utah, USA
[b] Department of Computer Science, University of Minnesota Duluth, USA
[c] NOAA/OAR National Severe Storms Laboratory, Norman, OK, USA
[d] Department of Environmental Atmospheric Sciences, Pukyong National University, Republic of Korea

ARTICLE INFO

Keywords:
QES-Winds
Poisson equation
Fast-response
Wind modeling
Iterative method
Staggered grid

ABSTRACT

QES-Winds is a fast-response wind modeling platform for simulating high-resolution mean wind fields for optimization and prediction. The code uses a variational analysis technique to solve the Poisson equation for Lagrange multipliers to obtain a mean wind field and GPU parallelization to accelerate the numerical solution of the Poisson equation. QES-Winds benefits from CUDA dynamic parallelism (launching the kernel from the GPU) to speed up calculations by a factor of 128 compared to the serial solver for a domain with 145 million cells. The dynamic parallelism enables QES-Winds to calculate mean velocity fields for domains with sizes of 10km^2 and horizontal resolutions of $1-3$ m in under 1 min. As a result, QES-Winds is a numerical code suitable for computing high-resolution wind fields on large domains in real time, which can be used to model a wide range of real-world problems including wildfires and urban air quality.

1. Introduction

The urban population of the world has grown rapidly from 751 million in 1950 to 4.2 billion in 2018. In 2018, cities contained 55% of the world's population and by 2050, urban areas will account for 68% of the world's population (United Nations and Department of Economic and Social Affairs, 2019). This rapid urbanization creates multiple meteorological and climate related phenomena linked to negative human-health outcomes, including air pollution (Shukla and Parikh, 1992) and urban heat island effects (Akbari and Kolokotsa, 2016). Additionally, urban population growth expands the wildland-urban interface increasing the risk to life and property from wildfires (Radeloff et al., 2018; Calkin et al., 2014). The risk is compounded by the increased number of wildfires over the past three decades. Since 2000, at least 10 states in the United States have had their largest fires on record and currently fire seasons are 78 days longer than in the 1970s, while over 70,000 communities are at risk of wildfires (USDA, 2019).

Proper modeling of the physics of wildfires (Moody et al., 2019; Linn et al., 2020) and pollution dispersion in cities (Pardyjak and Brown, 2001; Williams et al., 2004; Singh et al., 2008) requires high-resolution representation of wind fields in natural and urban areas. For the purpose of prediction, where model run-times should be near or faster than real time or for design optimization problems where thousands of simulations must be performed in a short period of time, traditional computational fluid dynamics (CFD) models such as Reynolds-averaged Navier-Stokes (RANS) simulations and large-eddy simulations (LES) are not fast enough (Hayati et al., 2019). Another option is to use a semi-empirical fast-response approach.

The complexity of urban and natural land-surface geometries, along with the complicated resulting wind flow, requires sophisticated physical models. However, more detailed models require a significant increase in computational costs (time and computational power). As a result, fast-response wind flow simulators are needed that compromise some accuracy to shorten computation time.

The Quick Environmental Simulation (QES) tool is a microclimate simulation platform for computing the transport of three-dimensional environmental scalars in urban areas and over complex topography. QES is organized into separate components each designed to simulate a different aspect of environmental transport. QES-Winds is the fast-response 3D diagnostic wind modeling module written in C++, based

* Corresponding author.
E-mail address: pardyjak@eng.utah.edu (E.R. Pardyjak).

https://doi.org/10.1016/j.envsoft.2021.104958
Accepted 30 December 2020
Available online 13 January 2021

on the often-used FORTRAN code QUIC-URB (Quick Urban and Industrial Complex-Urban) (Brown et al., 2013; Pardyjak and Brown, 2003). QES-Winds solves a mass-conservation equation for the wind field rather than the slower and more physics-based solvers that include conservation of momentum. While QES-Winds uses reduced-order physics to simulate urban flows, the solutions are rapid and compare quite favorably with higher-order physics-based models in both idealized (Hayati et al., 2017, 2019) and realistic cities (Neophytou et al., 2011).

The QES-Winds model is based on the 3D diagnostic urban wind model proposed by Röckle (1990). First, an initial wind field is prescribed by combining an incident flow with localized flows that account for the effects of building geometries via empirical parameterizations (Singh et al., 2008). Conservation of mass is then enforced using a variational analysis (a type of data assimilation) technique (Sasaki, 1958; Sasaki, 1970; Sasaki, 1970) to minimize the differences between the initial guess field and the final mass-conserving wind field. This technique requires the solution of a Poisson equation for Lagrange multipliers and results in calculating a quasi-time-averaged velocity field. The resulting complex 3D wind field resembles time-averaged experimental data (Hayati et al., 2017, 2019).

The Poisson equation is discretized over the computational domain and rearranged into matrix form creating a system of linear equations. The matrix form of the Poisson equation can theoretically be solved using sparse direct solvers, which should be fast compared to iterative solvers. However, after applying the complex boundary conditions specific to our case, the coefficient matrix (A) becomes a sparse, non-diagonal, non-banded, non-triangular, non-symmetric, and not real positive diagonal matrix. As a result, the matrix must be solved using the *MA*57 algorithm (Duff, 2004) (sparse symmetric system: multifrontal method), which is numerically expensive and quite slow for our purposes. To overcome these shortcomings, the Poisson equation in QES-Winds is solved using the Successive Over-Relaxation (SOR) method, a variant the of Gauss-Seidel method with faster convergence (Young, 1954; Varga, 1962). The SOR method is a sequential iterative method (Adams, 1982), which means that it is not fast enough for the purpose of optimization and prediction for domains with a high number of cells when a large number of iterations is required for convergence. To reduce the execution time of QES-Winds, the SOR method must be parallelized.

Despite the sequential nature of SOR Poisson solvers, they can be executed in parallel if the discretized equations are ordered according to the classical red-black coloring scheme (Hayes, 1974; Lambiotte, 1975). As illustrated in Fig. 1, in the red-black scheme, cells are divided into two partitions such that all of the neighboring cells of a red cell are black, and vice versa. As a result, the discretized equation can be solved for two sub-iterations in parallel: once for all of the red cells and once for all of the black cells inside the domain (Adams, 1982; Evans, 1984).

Utilizing parallel computing on CPUs (Central Processing Units) (Zapata et al., 2018; Krupka and Šimecek, 2010) and GPUs (Graphics Processing Units) (Helfenstein and Koko, 2012; Li and Saad, 2013; Cotronis et al., 2014; Konstantinidis and Cotronis, 2011; Itu et al., 2011) to accelerate the SOR solver has been the subject of extensive research. Kruptka and Šimeček (2010) showed that the achievable speedup by parallelizing on the CPU depends on the number of computational nodes available and size of the computational domain. Their results indicated that the maximum speedup for red-black SOR is equal to the number of computational nodes available. On the GPU, finding the maximum speedup is not as easy since various factors are involved. All codes that run on the GPU need to be launched from the CPU, and all data required for calculations must be copied from the CPU's memory to the GPU's global memory. Memory access time on the GPU and the bidirectional data transfer between the CPU and GPU are the most important parameters that affect the potential speedup. All of the aforementioned research mainly focused on reducing memory access time on the GPU by using shared memory or padded global memory (Itu et al., 2011). Cotronis et al. (2014) reported ~11 times speedup using the global memory of an NVIDIA GTX480 GPU (with 480 computational cores) over a sequential solver on a CPU for a domain size of 2882×2882 cells. In this paper, we focus on reducing the copy overhead between the CPU and GPU to accelerate the SOR solver. CUDA (Compute Unified Device Architecture) dynamic parallelism has been used in the literature (Jones, 2012; Kirk and Wen-Mei, 2016; Ding and Tan, 2015) to reduce the bidirectional data transfer between the CPU and GPU. Another option is to rely on the GPU memory to hold the data during the whole iterative process.

There are other similar diagnostic wind-modeling software packages that have been described in the literature. Most notably, WindNinja (Forthofer et al., 2014) and Micro SWIFT (Moussafir et al., 2004; Tinarelli et al., 2007). WindNinja uses a conjugate-gradient method with Jacobi preconditioning to solve the Poisson equation in a terrain-following coordinate system (Forthofer et al., 2014). The conjugate-gradient method with Jacobi preconditioning is computationally expensive and time-consuming, but it is the best option using terrain-following coordinates. As a result, WindNinja is not the best option for simulating wind fields with fine grids (e.g., of the order of 1 m). In addition, the terrain-following coordinate system is not suitable for computing flows around buildings, which means that WindNinja is not applicable for modeling urban areas. Micro SWIFT utilizes the SOR method to solve Poisson's equation in a three-dimensional Cartesian coordinate system (Moussafir et al., 2004; Tinarelli et al., 2007). However, Micro SWIFT is not able to compute flows over complex terrain since it does not process terrain geometry. In addition, because Micro SWIFT only has a serial solver, solving for high-resolution wind fields over urban areas is computationally expensive. Because it has the same

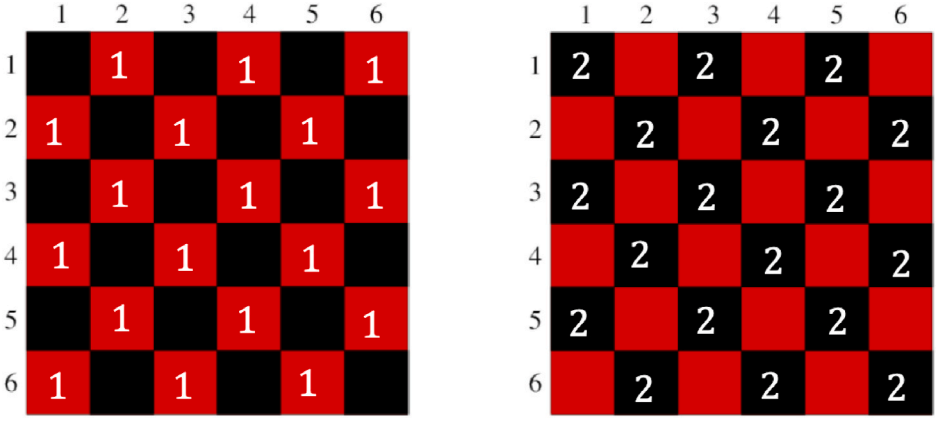

Fig. 1. Schematics of a 2D domain with the red-black coloring scheme for solving SOR in parallel: (left) sub-iteration to solve for all red cells, (right) sub-iteration to solve for all black cells. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

basic solver, Micro SWIFT could benefit from the parallelization method described in this paper.

There have been few attempts to accelerate diagnostic wind models using GPUs in the literature. The study most similar to our effort was conducted by Pinheiro et al. (2017). They utilized GPU parallelization to accelerate a mass-consistent wind model used in predicting atmospheric dispersion of radionuclides. They used the Winds Extrapolated from Stability and Terrain (WEST) model, which has an irrotational correction to the initial wind field that minimizes the divergence of the initial velocity field. The irrotational correction is based on the perturbation velocity potential and transmission coefficients, which are defined based on temperature profiles obtained from upper-air soundings. In this method, they calculate the divergence in each cell and the gradient of the perturbation velocity potential. They then update the velocity field and repeat the process until convergence, which is negligible divergence. Homicz (2002) reviewed different wind-modeling approaches and concluded that the WEST approach (called the NOABL model in the paper) is different than models based on variational calculus (e.g. QES-Winds). Also, in order to get a divergence-free final velocity field, there are restrictions on the values of transmission coefficients, which means that the model produces mass-consistent wind fields in special circumstances. Pinheiro et al. (2017) reported about 25 times speedup using an NVIDIA GTX-680 GPU for a domain with their finest grid (about 1.5 million cells).

The GPU parallelization technique (dynamic parallelism) discussed here can be incorporated into other types of wind solvers and dispersion models to significantly accelerate them. Singh et al. (2011) developed a new dispersion model called GPU Plume, that utilized the parallel computational capabilities available on the GPU to accelerate the calculations. Our group is developing an improved version of the GPU Plume that has the potential to decrease the execution time by exploiting benefits of the dynamic parallelism. Also, the new model in conjunction with the QES-Winds, could run air quality simulations of large urban areas in near real time. Other examples include *WindStation*, which is software developed by Lopes (2003) as a tool to simulate atmospheric flows over complex terrain. Two models were used to handle the complex topography: the first one is a mass-conservative wind solver much simpler than QES-Winds, while the second one solved for three-dimensional Naiver-Stokes equations. Incorporating the same GPU technique described here in these models can speed up calculations and allow for high-resolution simulations on massive domain including those with complex topography. Lastly, Linn et al. (2020) recently developed a fast-running tool to model complex behavior of wildland fire propagation. They used QUIC-URB (Brown et al., 2013; Pardyjak and Brown, 2003) as the wind solver to provide a high-resolution wind field for the fire propagation model. QES-Winds, which evolved from QUIC-URB, can provide a wind to fire propagation models to predict wildfire behavior over larger domains in near real time.

The overall goal of this research is to significantly decrease the execution time required for QES-Winds. NVIDIA's parallel GPU computing platform and CUDA application programming interface (API) (NVIDIA, 2019) are used to substantially accelerate QES-Winds. By utilizing GPU parallel computing capabilities, QES-Winds will be fast enough to use for prediction and optimization purposes.

2. Methods

QES-Winds uses a variational analysis technique (Sasaki, 1958; Sasaki, 1970; Sasaki, 1970) to obtain a quasi-time-averaged velocity field. This method requires the solution of a Poisson equation for the Lagrange multipliers, λ:

$$\frac{\partial^2\lambda}{\partial x^2}+\frac{\partial^2\lambda}{\partial y^2}+\left(\frac{\alpha_1}{\alpha_2}\right)^2\frac{\partial^2\lambda}{\partial z^2}=R \tag{1}$$

where x, y, and z are the spatial coordinates in the streamwise, spanwise, and ground-surface normal directions, respectively, and α_1 and α_2 are Gaussian precision moduli. To numerically implement Eq. (1), we discretize the computational domain using a staggered grid where λ and the divergence of the initial velocity field R are cell-centered variables and flow components u, v and w corresponding to the x, y, and z directions, respectively, are cell-faced values. The divergence of the initial wind field is defined as

$$R_{i,j,k}=-2\alpha_1^2\left[\frac{u^0_{i+\frac{1}{2},j,k}-u^0_{i-\frac{1}{2},j,k}}{\Delta x}+\frac{v^0_{i,j+\frac{1}{2},k}+v^0_{i,j-\frac{1}{2},k}}{\Delta y}+\frac{w^0_{i,j,k+\frac{1}{2}}-w^0_{i,j,k-\frac{1}{2}}}{\Delta z}\right] \tag{2}$$

where i, j, and k are cell indices in the x, y, and z directions, respectively, a half-index step indicates a cell face value, Δx, Δy, and Δz are the cell dimensions in the x, y, and z directions, respectively, and the superscript 0 denotes an initial estimated value.

Equation (1) is solved using the SOR method (Young, 1954; Varga, 1962) resulting in the following relationship for the Lagrange multipliers:

$$\lambda_{i,j,k}=\frac{\omega}{e_{i,j,k}+f_{i,j,k}+g_{i,j,k}+h_{i,j,k}+m_{i,j,k}+n_{i,j,k}}\left[\begin{array}{l}-(\Delta x)^2R_{i,j,k}+e_{i,j,k}\lambda_{i+1,j,k}+f_{i,j,k}\lambda_{i-1,j,k}\\+A\left(g_{i,j,k}\lambda_{i,j+1,k}+h_{i,j,k}\lambda_{i,j-1,k}\right)+B\left(m_{i,j,k}\lambda_{i,j,k+1}+n_{i,j,k}\lambda_{i,j,k-1}\right)\end{array}\right]+(1-\omega)\lambda_{i,j,k} \tag{3}$$

where $e_{i,j,k}$, $f_{i,j,k}$, $g_{i,j,k}$, $h_{i,j,k}$, $m_{i,j,k}$, and $n_{i,j,k}$ are boundary condition coefficients and $A=(\Delta x)^2/(\Delta y)^2$ and $B=\eta(\Delta x)^2/(\Delta y)^2$ (where $\eta=[\alpha_1/\alpha_2]^2$) are domain constants. Gaussian precision moduli α_1 and α_2 are set to unity in this study and the SOR over-relaxation factor $\omega=1.78$ is based on the recommendation by (Röckle, 1990). Neumann boundary conditions ($\partial\lambda/\partial n=0$) are applied to solid surfaces and Dirichlet boundary conditions ($\lambda=0$) are applied to inlet/outlet surfaces. To implement the solid surface boundary condition, the boundary condition coefficient related to the surface is set to zero. The boundary condition coefficients $e_{i,j,k}$, $f_{i,j,k}$, $g_{i,j,k}$, $h_{i,j,k}$, $m_{i,j,k}$, and $n_{i,j,k}$ are related to cell surfaces located at $(i+1/2)$, $(i-1/2)$, $(j+1/2)$, $(j-1/2)$, $(k+1/2)$ and $(k-1/2)$ of the cell (i,j,k), respectively.

The final velocity field is then updated through the Euler-Lagrange equations:

$$u_{i,j,k}=u^0_{i,j,k}+\frac{1}{(2\alpha_1^2)\Delta x}\left[\lambda_{i+1,j,k}-\lambda_{i,j,k}\right],\quad\text{and} \tag{4}$$

$$v_{i,j,k}=v^0_{i,j,k}+\frac{1}{(2\alpha_1^2)\Delta y}\left[\lambda_{i,j+1,k}-\lambda_{i,j,k}\right],\quad\text{and} \tag{5}$$

$$w_{i,j,k}=w^0_{i,j,k}+\frac{1}{(2\alpha_2^2)\Delta z}\left[\lambda_{i,j,k+1}-\lambda_{i,j,k}\right]. \tag{6}$$

To ensure convergence, the error for each iteration is calculated as:

$$Error=Max\left(\left|\lambda^t_{i,j,k}-\lambda^{t-1}_{i,j,k}\right|\right), \tag{7}$$

- Copy u_0, v_0, w_0 and R from the CPU's memory to the GPU's global memory
- Call the *"divergenceGlobal"* kernel to calculate R using Eq. 2
- Copy e, f, g, h, m, n and λ from the CPU's memory to the GPU's global memory

$tolerance = 10^{-9}$ `// Convergence criterion`
$iteration = 0$ `// Iteration counter`
$error = 1$
`// SOR loop starts here`
while *((iteration < maximum iterations) || (error > tolerance))* **do**
{
- Call the *"saveLambdaGlobal"* kernel to set $\lambda_{old} = \lambda$
- Call the *"SOR_RB_Global"* kernel to calculate λ values for the red cells using Eq. 3
- Call the *"SOR_RB_Global"* kernel to calculate λ values for the black cells using Eq. 3
- Call the *"applyNeumannBCGlobal"* kernel to apply Neumann boundary condition for the solid surface below
- Call the *"calculateErrorGlobal"* kernel to calculate maximum relative error between λ and λ_{old}

$iteration = iteration + 1$ `// Increase the iteration counter`
}
- Copy *cell flag* from the CPU's memory to the GPU's global memory
- Call the *"finalVelocityGlobal"* kernel to calculate the final wind field using Eqs. 4-6
- Copy u, v and w from the GPU's global memory to the CPU's memory

where t represents the current iteration and $t-1$ stands for the previous iteration. This guarantees that all calculated Lagrange multipliers in the computational domain converge to the same criteria set by the user.

2.1. Solver options

The combination of the divergence (Eq. (2)), the SOR loop (Eq. (3)), and Euler-Lagrange equations (Eqs. (4)–(6)) comprise the QES-Winds solver. QES-Winds has two options for the solver. The first option is to solve the equations on the GPU using CUDA kernels and the second option uses the CPU for computations. The GPU solver makes use of a red-black coloring scheme explained in the following section.

2.2. Red-black coloring scheme

In Eq. (3), the Lagrange multiplier for each cell, $\lambda_{i,j,k}$, depends on the Lagrange multiplier values for neighboring cells ($i-1, i+1, j-1, j+1, k-1$ and $k+1$). The SOR method is sequential (Adams, 1982), which means that the Lagrange multiplier values for $i-1, j-1$ and $k-1$ cells are calculated in the current iteration while the values for $i+1, j+1$ and $k+1$ cells are from the previous iteration. The dependency and sequence in calculating the Lagrange multipliers prevents us from solving for all the cells in parallel (Adams, 1982). Several papers including Hayes et al. (Hayes, 1974) and Lamblotte et al. (Lambiotte, 1975) suggested the red-black coloring scheme as a solution to eliminate this dependency and sequence.

In the red-black scheme, the computational domain is divided into two sets of cells colored red and black in which the cell being solved for and its neighboring cells are different colors. In QES-Winds, cells are colored red (black) if $(i+j+k)$ is odd (even). By applying the red-black ordering scheme, the SOR iterations turn into two separate sub-iterations, each done in parallel across all available GPU cores. Equation (3) is first solved for all red cells, then subsequently for all black cells. Fig. 1 shows a two dimensional domain with the red-black coloring order applied and indicates the two sub-iterations for red and black cells. Three different GPU-solver implementations are introduced in the following sections.

2.3. GPU solver using global memory

Three kernels: *divergenceGlobal*, *SOR_RB_Global*, and *finalVelocityGlobal* are written to access the global memory to compute the divergence (Eq. (2)), solve for the Lagrange multipliers using the red-black SOR method (Eq. (3)), and solve the Euler-Lagrange equations (Eqs. (4)–(6)). In addition, three other kernels: *saveLambdaGlobal*, *applyNeumannBCGlobal* and *calculateErrorGlobal* save a copy of the Lagrange multipliers from the previous iteration, apply the Neumann boundary conditions to the ground surface and calculate maximum relative error for the iteration. Algorithm 1 shows the details of incorporating the solver on the GPU using global memory kernels.

Data required for calculations are copied once from the CPU's memory to GPU's global memory and must remain there until the end of the process. NVIDIA GPUs with pre-Volta architectures (Pascal, Maxwell, Kepler, Fermi and Tesla) use the Multi-Process Service (MPS) that does not fully isolate the threads and memory required for an application run (NVIDIA, 2020). According to the CUDA MPS overview (NVIDIA, 2020), this means that other applications running concurrently on a shared GPU, can possibly allocate over and/or modify data allocated for another application, without triggering an error. Pietro et al. (2016) reported an illegal memory access (memory leakage) by two independent host processes while using the global memory of pre-Volta architecture GPUs. To guarantee security of QES-Winds data, all data required for each kernel must be copied to the GPU's global memory before launching the kernel. This solution means that the Lagrange multipliers and boundary-condition coefficients must be copied back and forth to the CPU's memory before and after each call to the red-black SOR kernel, which leads to a massive copying overhead and slow down in the solver. In the present study, we stick to the current version of the solver, without copying back and forth, since we have access to GPUs that are not shared with other applications.

Algorithm 1. GPU solver using global memory

2.4. GPU solver using shared memory

Global memory accesses on the GPU usually have an associated time delay. One way to reduce the delay is to load the required data for calculations from the global memory (accessible by all cores) to faster shared memory (accessible only by threads in a block). Because the *SOR_RB* kernel has the most memory accesses, we only applied the shared memory to the *SOR_RB* kernel. The rest of the kernels and the algorithm are the same as the ones for the global memory solver.

2.5. GPU solver using dynamic parallelism

Another way to address the aforementioned memory leakage issue is to use the CUDA dynamic parallelism. By utilizing dynamic parallelism,

the host does not have information about the number of threads and the amount of memory required for the calculation. As a result, the whole global memory and all the cores are reserved to run the dynamic parallel kernel and the MPS does not share the GPU resources (NVIDIA, 2020). This means that data on the global memory is secured even for pre-Volta GPU architectures.

Dynamic parallelism has been used to reduce the copying overhead (Jones, 2012; Kirk and Wen-Mei, 2016; Ding and Tan, 2015). In this method, data required for calculations are copied to the GPU global memory once, which makes the solver much faster compared to the most secure version of the global memory solver (with copies back and forth to the GPU). In this solver, the *dynamicParallel* kernel is called with one thread from the host. Next, all kernels are called from inside the *dynamicParallel* kernel (already on the GPU). Algorithm 2 details the dynamic parallel method. All kernels that are launched from the *dynamicParallel* kernel are the same as the ones in the global memory solver.

Algorithm 2. GPU solver using dynamic parallelism

- Copy $u_0, v_0, w_0, R, e, f, g, h, m, n, x, y, z, \lambda$ and λ_{old} from the CPU's memory to the GPU's global memory
$tolerance = 10^{-9}$ `// Convergence criterion`
- Call the *"dynamicParallel"* kernel with one thread to calculate the final wind field
`// The following lines will be executed from GPU ("dynamicParallel" kernel)`
$iteration = 0$ `// Iteration counter`
$error = 1$
- Call the *"divergence"* kernel to calculate R using Eq. 2
`// SOR loop starts here`
while *((iteration < maximum iterations) || (error > tolerance))* **do**
{
- Call the *"saveLambda"* kernel to set $\lambda_{old} = \lambda$
- Call the *"SOR_RB"* kernel to calculate λ values for the red cells using Eq. 3
- Call the *"SOR_RB"* kernel to calculate λ values for the black cells using Eq. 3
- Call the *"applyNeumannBC"* kernel to apply Neumann boundary condition for the solid surface below
- Call the *"calculateError"* kernel to calculate maximum relative error between λ and λ_{old}
$iteration = iteration + 1$ `// Increase the iteration counter`
}
- Call the *"finalVelocity"* kernel to calculate the final wind field using Eqs. 4-6
`// End of the dynamicParallel kernel`
- Copy u, v and w from the GPU's global memory to the CPU's memory

3. Results and discussion

3.1. Convergence criteria

A common convergence criterion for iterative methods is when the maximum error in the domain falls below 10^{-6} to 10^{-10} (Adams, 1982; Kelley, 1995). The complex boundary conditions and imposed building parameterizations in QES-Winds makes convergence difficult to achieve. In most cases, converging to 10^{-9} requires up to 200,000 iterations without a significant difference in the final wind field. As a result, QES-Winds imposes the maximum number of iterations as a secondary convergence criteria. In order to define the maximum number of iterations, a test case with $100 \times 100 \times 100$ cells and cell size of $2 \times 2 \times 1$ m was investigated. The test case was a three-dimensional flow around a single cubical building with a 20-m edge length located in the middle of domain. This is the simplest case in QES-Winds. All building flow parameterizations were applied to the building. Winds are specified for simulation initialization using a sensor at 10-m height with a measured wind speed of 5 ms^{-1} at 270° from the north. A logarithmic profile has been used to create the initial velocity field based on the sensor data.

Since the QES-Winds error does not reach 10^{-9} for this test case, the

Table 1
Convergence error (Eq. (7)), maximum difference for each velocity component, and the relative difference between the solution and the reference solution after the number of iterations for the isolated cubical-building test case with $100 \times 100 \times 100$ cells and cell size of $2 \times 2 \times 1$ m.

Number of iterations	Error	Maximum difference u (ms^{-1})	Maximum difference v (ms^{-1})	Maximum difference w (ms^{-1})	Relative difference u (ms^{-1})	Relative difference v (ms^{-1})	Relative difference w (ms^{-1})
100	8.57E-02	0.18	0.14	0.12	0.15	0.09	0.02
250	1.41E-02	0.04	0.03	0.03	0.02	0.02	4.39E-3
500	2.07E-03	4.64E-03	3.45E-03	3.61E-03	2.48E-03	3.02E-03	5.97E-04
1000	1.45E-04	2.46E-04	2.04E-04	1.54E-04	1.32E-04	8.25E-03	5.07E-03
2000	4.96E-05	1.72E-05	1.53E-05	2.67E-05	8.81E-06	7.64E-03	1.76E-05
3000	3.81E-05	1.43E-05	1.53E-05	1.91E-05	4.37E-06	1.98E-05	1.47E-05
4000	3.81E-05	1.34E-05	1.34E-05	2.29E-05	1.43E-05	4.52E-05	2.66E-04
5000	4.58E-05	1.34E-05	1.72E-05	2.10E-05	2.48E-05	2.23E-05	2.35E-04

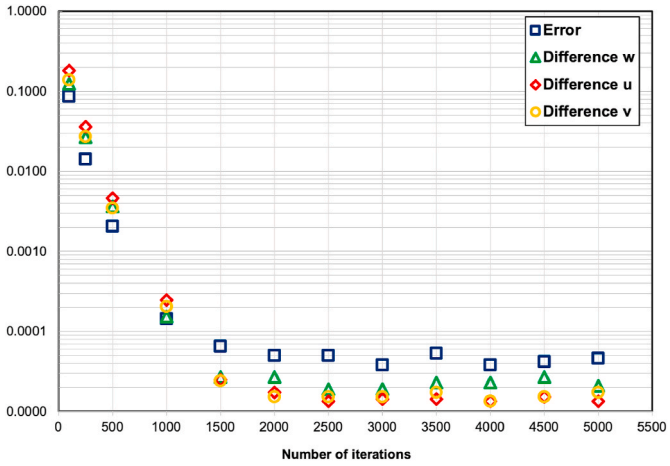

Fig. 2. Maximum error and maximum difference for velocity components compared to the reference solution as a function of iteration count.

solution after 100,000 iterations was chosen as a reference. Details from the test case are shown in Table 1.

Fig. 2 illustrates the maximum convergence error (Eq. 7) and maximum difference for velocity components compared to the reference solution as a function of iteration count. The results in Table 1 and Fig. 2 show that while the error and maximum differences decrease with number of iterations, they do not decrease beyond 1500 iterations. Thus, performing more than 1500 iterations does not improve the solution and wastes time and computational resources. The values of maximum and relative differences for each velocity component for the solution with 500 iterations are of the order of 10^{-3}. Velocity values less than 0.01 ms^{-1} cannot be measured experimentally, which means that 0.01 ms^{-1} is an acceptable threshold for our calculations. In addition, converging to the error criterion 10^{-9} is much harder for realistic cases such flow wind over cities. This is a result of multiple buildings and overlapping building parameterizations for cities or irregular geometry in complex terrain flows. As a result, the maximum number of iterations for the purposes of QES-Winds is set to 500 iterations.

3.2. Benchmarking for CPU and GPU solvers

The CPU solver is quite efficient, but slow in comparison to the GPU solvers, especially for large domains, since it lacks parallel capabilities. A suite of test cases were designed and analyzed to illustrate differences between solvers. Each case includes a cubical building with edge-lengths of 20 m situated in the middle of the domain with all the building parameterizations applied. Winds are specified for simulation initialization using a sensor at 10 m above ground with a wind speed of 5 ms^{-1} coming from 270° (relative to north). A logarithmic profile has been used to create the initial velocity field based on the sensor data. Details for each are provided in Table 2. The CPU-solver solution is considered a reference because it is identical to the well-validated QUIC-URB solver (Brown et al., 2013; Pardyjak and Brown, 2003). QUIC-URB has compared quite well with higher-order physics-based models and available experimental results (Shukla and Parikh, 1992; Hayati et al., 2017, 2019; Neophytou et al., 2011; Bagal et al., 2004; Booth and Pardyjak, 2012; Bowker et al., 2004; Balwinder et al., 2006). Hence, we consider QES-Winds CPU solver accurate enough for the purposes of diagnostic wind modeling.

The CPU solver was run for 500 iterations in each test case and then the GPU solvers were run until they reached the same error as the CPU solver after 500 iterations. Table 2 shows the number of iterations required for the Global Memory (GM), the Shared Memory (SM), and the dynamic-parallel (DP) solvers to converge in addition to the total time required. This benchmarking was performed on a machine with an Intel Xeon Gold 6130 CPU @ 2.10 GHz CPU with 12 GB RAM and an NVIDIA TITAN V GPU with CUDA 10.1 toolkit (NVIDIA, 2019) and 12 GB of global memory.

Since the CPU solver and each of the GPU solvers run for different numbers of iterations, the total time for each solver is divided by the number of iterations. Table 3 displays the time per iteration for the different solvers in addition to the speedup for each of the GPU solvers over the CPU solver. Fig. 3 shows the time per iteration as a function of the number of cells for all test cases and solvers. All GPU solvers are much faster than the CPU solver due to parallelization benefits. For very large domains (145 million cells), all GPU solvers are approximately 128 times faster than the CPU solver. Differences in time per iteration for all GPU solvers are negligible since they only have the overhead associated with one copy from the CPU's memory to GPU's global memory and back, and they have similar kernels with small differences. QES-Winds cannot solve for more than 145 million cells on the TITAN V GPU due to constraints on its global memory (12 GB). According to the description of CUDA Dynamic Parallelism in the CUDA C++ Programming Guide (NVIDIA, 2019), launching kernels from inside the dynamic-parallel kernel has the potential to add a large amount of overhead on the application. However, in the case of QES-Winds, results show that the dynamic-parallel solver is less than five percent slower than the global and shared memory solvers.

The shared memory solver is slightly faster than the global and dynamic-parallel solvers for smaller domains (number of cells $\leq 50M$). The only exception is for the case of a total number of cells of $0.1M$ because the number of accesses to the global memory is not large enough to realize the advantage of loading the data on memory with less latency (shared memory) can give. Generally, since there is no reuse of data loaded to the shared memory in the *SOR_RB_Shared*, the speedup for the shared-memory solver is less than five percent.

The solver accounts for most of the execution time in QES-Winds. Although the single-building case with building parameterization is the simplest test case, as long as the more realistic cases (e.g. flow over cities or complex terrain) have the same number of cells, the execution time is almost the same. The only difference is the set-up time, which

Table 2
Benchmarking of the QES-Winds solvers in terms of computational time in seconds (s) and number of iterations (NI). Cell size indicates the physical size of each computational cell in the *x*, *y*, and *z* directions, respectively. GM is global memory, SM is shared memory, and DP is dynamic-parallel. Each test case includes the isolated cubical building with the edge-length of 20 m in the middle of domain with all the building parameterizations applied.

nx*ny*nz	Cell size (m*m*m)	Number of cells	GM (s)	GM (NI)	SM (s)	SM (NI)	DP (s)	DP (NI)	CPU (s)
100*50*20	2*4*5	0.1 M	0.25	436	0.26	436	0.26	436	3.66
100*100*50	2*2*2	0.5 M	0.42	561	0.38	561	0.43	561	14.34
100*100*100	2*2*1	1 M	0.54	498	0.50	498	0.54	498	30.10
250*200*100	0.8*1*1	5 M	1.43	480	1.42	480	1.49	480	150.86
400*250*100	0.5*0.8*1	10 M	2.64	481	2.59	481	2.72	481	306.98
500*400*250	0.4*0.5*0.4	50 M	12.70	488	12.66	488	12.79	488	1590.92
600*500*250	0.34*0.4*0.4	75 M	19.12	490	19.18	490	19.28	490	2431.57
800*500*250	0.25*0.4*0.4	100 M	25.61	493	25.61	493	25.76	493	3220.47
1000*580*250	0.2*0.35*0.4	145 M	37.32	496	37.31	496	37.43	496	4824.57

Table 3
Time per iteration for different QES-Winds solvers along with the speedup each GPU solver has over the CPU solver. GM is global memory, SM is shared memory, and DP is dynamic-parallel. Each test case includes the isolated cubical-building with the edge-length of 20 m in the middle of domain with all the building parameterizations applied.

Number of cells	GM per iteration (s)	SM per iteration (s)	DP per iteration (s)	CPU per iteration (s)	Speedup GM	Speedup SM	Speedup DP
0.1 M	5.83E-04	5.95E-04	5.85E-04	7.33E-03	12.56	12.31	12.53
0.5 M	7.40E-04	6.81E-04	7.73E-04	2.87E-02	38.74	42.10	37.08
1 M	1.08E-03	1.00E-03	1.08E-03	6.02E-02	55.52	60.17	55.98
5 M	2.98E-03	2.96E-03	3.10E-03	3.02E-01	101.40	101.77	97.48
10 M	5.49E-03	5.39E-03	5.65E-03	6.14E-01	111.89	113.99	108.69
50 M	2.60E-02	2.59E-02	2.62E-02	3.18	122.27	122.27	121.36
75 M	3.90E-02	3.91E-02	3.93E-02	4.86	124.62	124.27	123.62
100 M	5.20E-02	5.19E-02	5.22E-02	6.44	123.97	124.00	123.28
145 M	7.52E-02	7.52E-02	7.55E-02	9.65	128.24	128.29	127.86

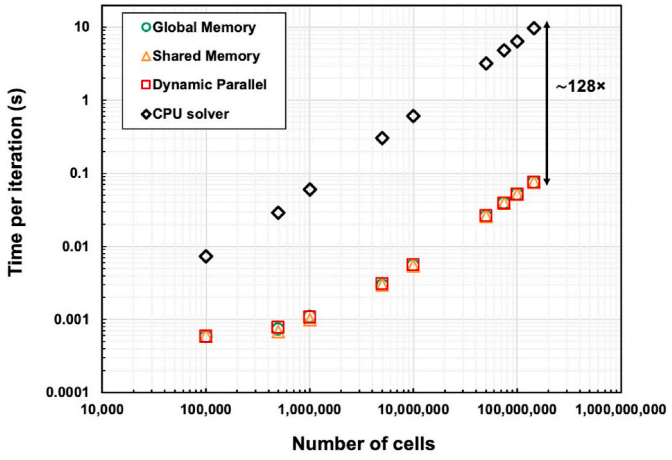

Fig. 3. Scaling plot showing the impact of the different GPU parallel-computing implementations. Time per iteration is shown for each QES-Winds solver as a function of cell count.

depends on the case type. Processing buildings and apply building parameterizations prior to running the SOR solver leads to different execution times depending on the specific geometry being simulated. Assuming 1-m horizontal and 3-m vertical resolution, QES-Winds can compute wind fields on a 1.18 km by 1.21 km by 210 m domain ($1180 \times 1210 \times 70$ cells, the 100 million-cell case), an area as big as the central business district in downtown Oklahoma City with all building parameterizations applied, in about 130 s. No other wind-modeling systems is capable of simulating such a large domain with such a fine resolution in real time (note that the CPU solver takes about 55 min).

Kernel execution metrics are required to explain why the GPU solvers behave this way. The NVIDIA visual profiler (NVIDIA, 2019) is used for this purpose. Since the NVIDIA visual profiler version 10.1 does not support CUDA dynamic-parallel profiling on GPUs with compute capability of 7.0 and higher (TITAN V has compute capability of 7.0), the profiling was performed on a different machine with an NVIDIA GeForce GTX TITAN X (Maxwell architecture) with the CUDA 10.1 toolkit (NVIDIA, 2019) and 12 GB of global memory. Table 4 contains results of profiling on the GPU solvers for three test cases with cell counts of 1, 10, and 50 million cells. The GPU activities of the dynamic-parallel solver that have equivalent parts in the global and shared memory solvers, are listed in Table 4 in order to provide a fair comparison between the

Table 4
Profiling details for three GPU solvers: the global and shared memory and dynamic-parallel solvers for three test cases with total cell counts of 1, 10, and 50 million cells. memcpy is copying memory, HtoD is host to device (CPU to GPU), and DtoH is device to host (GPU to CPU). Each test case includes the isolated cubical-building with the edge-length of 20 m in the middle of domain with all the building parameterizations applied.

GPU solver type	GPU activity name	Number of cells					
		1 M		10 M		50 M	
		Time		Time		Time	
DP	CUDA memcpy HtoD	0.19%	4.91 ms	0.21%	49.34 ms	0.21%	254.12 ms
	CUDA memcpy DtoH	0.04%	0.99 ms	0.04%	9.51 ms	0.04%	46.83 ms
	SOR_RB	6.77%	172.16 ms	6.72%	1.59 s	7.02%	8.47 s
	finalVelocity	0.08%	1.92 ms	0.08%	18.83 ms	0.08%	95.71 ms
	divergence	0.04%	1.08 ms	0.04%	10.46 ms	0.04%	47.37 ms
	calculateError	91.61%	2.33 s	91.97%	21.87 s	91.72%	110.75 s
	saveLambda	1.13%	28.68 ms	0.92%	219.44 ms	0.90%	1.09 s
	applyNeumannBC	0.14%	3.60 ms	0.02%	5.32 ms	0.01%	13.68 ms
GM	CUDA memcpy HtoD	0.2%	5.05 ms	0.23%	53.51 ms	0.21%	249.46 ms
	CUDA memcpy DtoH	0.08%	1.99 ms	0.04%	10.58 ms	0.04%	48.22 ms
	SOR_RB_Global	6.34%	156.69 ms	6.60%	1.56 s	6.93%	8.30 s
	finalVelocityGlobal	0.08%	1.91 ms	0.08%	18.91 ms	0.08%	95.24 ms
	divergenceGlobal	0.04%	0.89 ms	0.04%	8.74 ms	0.04%	45.39 ms
	calculateErrorGlobal	92.59%	2.29 s	92.35%	21.79 s	92.06%	110.31 s
	saveLambdaGlobal	0.62%	15.43 ms	0.65%	153.16 ms	0.64%	767.69 ms
	applyNeumannBCGlobal	0.04%	0.94 ms	0.01%	1.99 ms	0.01%	3.26 ms
SM	CUDA memcpy HtoD	0.20%	4.98 ms	0.22%	51.99 ms	0.21%	247.75 ms
	CUDA memcpy DtoH	0.08%	2.01 ms	0.04%	10.61 ms	0.04%	48.13 ms
	SOR_RB_Shared	6.32%	155.98 ms	6.57%	1.56 s	6.92%	8.29 s
	finalVelocityShared	0.08%	1.91 ms	0.08%	18.94 ms	0.04%	95.30 ms
	divergenceShared	0.04%	0.89 ms	0.04%	10.51 ms	0.04%	47.00 ms
	calculateErrorShared	92.63%	2.29 s	92.39%	21.88 s	92.07%	110.31 s
	saveLambdaShared	0.62%	15.3 ms	0.65%	153.18 ms	0.64%	767.95 ms
	applyNeumannBCShared	0.04%	0.94 ms	0.01%	2.01 ms	0.01%	3.26 ms

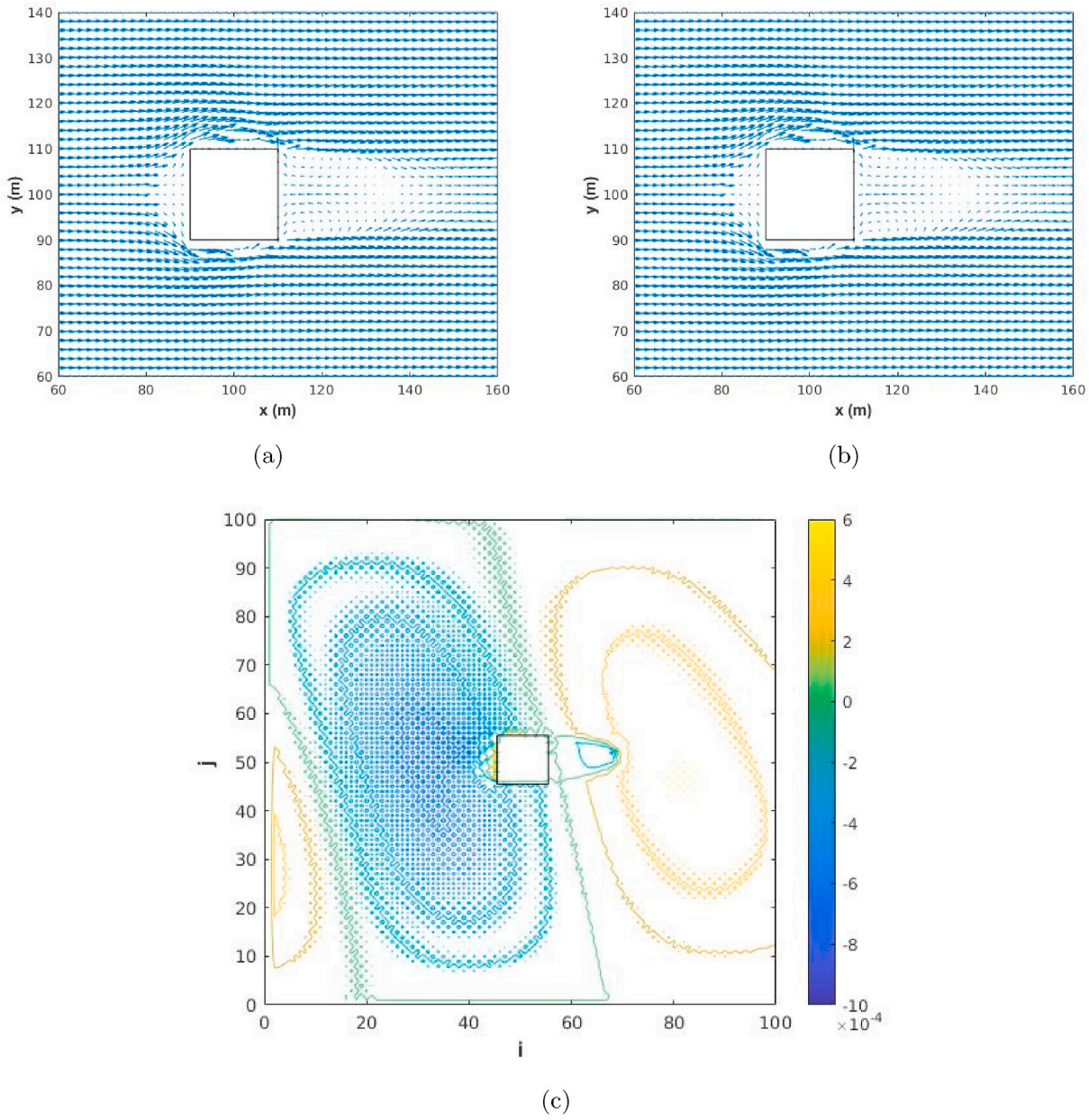

Fig. 4. Velocity vectors in a horizontal plane at $z = 10$ m for the flow around a 20-m cubical-building test case. (a) CPU solver and (b) GPU solver. (c) Difference between the velocity magnitude for the CPU and the GPU solvers in a horizontal plane at $z = 10$ m. The black solid line shows the boundaries of the building.

solvers.

It can be seen that the copying overhead to the GPU (CUDA memcpy HtoD) and from the GPU (CUDA memcpy DtoH) is slightly higher for the global and shared memory compared to the dynamic-parallel solver. The reason for this behavior is that the error value must be copied back and forth between the host and the device to check for convergence. Meanwhile, kernels launched from the *dynamicParallel* kernel have slightly longer execution times, which must be related to the execution overhead of the dynamic parallelism (NVIDIA, 2019) since all the kernels are identical for the global-memory solver and dynamic-parallel solver.

In the most secure versions of the global- and shared-memory solvers, which require copying back and forth during each iteration, the copying overhead increases the time per iteration (related to CUDA memcpy HtoD) by a factor of about four. In this case, the dynamic-parallel solver is much faster than the global- and shared-memory solvers.

3.3. Comparing red-black SOR to serial SOR

Surprisingly, to our knowledge, there has never been a discussion on how well the red-black SOR method (i.e., parallel version) compares to serial SOR in the literature. We conducted a test using the same test case described above of flow around an isolated 20 m cube with 1 million cells (the third row case in Table 2) using the red-black SOR (GPU) and serial-SOR (CPU) solvers. Winds are specified for simulation initialization using a sensor at 10 m height with a measured wind speed of 5 ms^{-1} at 270° from the north. A logarithmic profile has been used to create the initial velocity field based on the sensor data. The converged solutions from each case are compared in Fig. 4. Fig. 4(a) and (b) show velocity vectors for the CPU and the GPU solvers, while Fig. 4(c) show the differences in velocity magnitudes between the CPU and the GPU solvers. All data in the figure are presented for a horizontal plane at $z = 10$ m. The solid black line shows the boundaries of the building. There is a notable checkerboard pattern present in the difference field, which is

caused by the red-black SOR procedure in which the red cells use the Lagrange multiplier values from the previous iteration while the black cells use the Lagrange multiplier values of the red cells from the current iteration. The flow of data between neighboring cells in the red-black SOR solver is different from the serial SOR, Gauss-Seidel, and Jacobi methods and prevents smoothing and causes the observed checkerboard pattern.

QES-Winds outputs the velocity field as an averaged cell-centered field for visualization purposes, which slightly smooths the checkerboard pattern. The maximum difference between the velocity components from the two solutions are 0.0006, 0.0008, and 0.001 ms^{-1}, for u, v, and w respectively. This means that for the purpose of QES-Winds (modeling mean wind fields), the checkerboard pattern is not a significant issue. Because one the most important components of the QES-Winds is speed, the accuracy can be sacrificed for a faster running solver. However, for applications that need the gradient of the velocity field (e.g., computing the turbulence field), the checkerboard pattern can pose issues. More complex and expensive smoothing techniques may be required for such applications.

4. Conclusion

Optimizing and predicting wind fields for fast-response applications such as wildfires and urban air quality require modeling high-resolution three-dimensional mean wind fields in real-time for large domains. QES-Winds uses the parallel capabilities of the GPU to accelerate wind-field computations. Three different implementations of the GPU solver were examined. While all three solvers were much faster than the CPU (serial) solver, only one of them (dynamic-parallel) demonstrated the ability to guarantee the security of global memory data from potential illegal accesses. The dynamic-parallel solver reduces the execution time by a factor of 128 compared to the serial solver for a domain with 145 million cells. QES-Winds was able to solve for the wind field on a 10 km^2 domain with a horizontal grid spacing of $1-3$ m in less than 1 min. The application of QES-Winds as a fast-response wind-modeling code can be further enhanced by improving its physics modules.

NVIDIA's CUDA dynamic parallelism can greatly accelerate iterative methods and other codes that require a large amount of copying overhead between the CPU and the GPU while protecting data on all GPU architectures. There were several challenges related to the dynamic-parallel implementation. First, current GPU's global memory limits QES-Winds to solving domains of no more than 145 million cells. Second, the NVIDIA visual profiler is unable to run for a code using CUDA dynamic parallelism on newer GPUs with higher compute capabilities. These challenges will be alleviated with the advent of newer GPUs with higher global memory that are supported by the CUDA visual profiler.

Software availability

The Quick Environmental System (QES) fast-response wind solver (QES-Winds) has been developed as a collaboration between the University of Utah, University of Minnesota Duluth and Pukyong National University. The code is written mainly in C++ and NVIDIA's CUDA language. QES-Winds is publicly accessible, currently hosted on GitHub (https://github.com/UtahEFD/QES-Winds-Public).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was partly supported by a grant from the National Institute of Environment Research (NIER), funded by the Ministry of Environment (MOE) of the Republic of Korea (NIER-SP2019-312), the United States Department of Agriculture National Institute for Food and Agriculture Specialty Crop Research Initiative Award No. 2018–03375 and the United States Department of Agriculture Agricultural Research Service through Research Support Agreement 58-2072-0-036.

References

Adams, L.M., 1982. Iterative algorithms for large sparse linear systems on parallel computers. NASA, NASA-CR-166027.
Akbari, H., Kolokotsa, D., 2016. Three decades of urban heat islands and mitigation technologies research. Energy Build. 133, 834–842.
Bagal, N., Pardyjak, E., Brown, M., 2004. Improved upwind cavity parameterization for a fast response urban wind model. In: 84th Annual AMS Meeting. AMS, Seattle, WA.
Balwinder, S., Pardyjak, E., Brown, M.J., Williams, M.D., 2006. Testing of a far-wake parameterization for a fast response urban wind model. In: Sixth Symposium on the Urban Environment. AMS, Atlanta, GA.
Booth, T.M., Pardyjak, E., 2012. Validation of a data assimilation technique for an urban wind model. Department of Mechanical Engineering, University of Utah.
Bowker, G.E., Perry, S.G., Heist, D.K., 2004. A comparison of airflow patterns from the QUIC model and an atmospheric wind tunnel for a two-dimensional building array and a multi-city block region near the World Trade Center site, 13th Confernce on the Applications of Air Pollution Meteorolgy with the Air and Waste Management Assoc. AMS, Vancouver, BC.
Brown, M.J., Gowardhan, A.A., Nelson, M.A., Williams, M.D., Pardyjak, E.R., 2013. QUIC transport and dispersion modelling of two releases from the Joint Urban 2003 field experiment. Int. J. Environ. Pollut. 52, 263–287.
Calkin, D.E., Cohen, J.D., Finney, M.A., Thompson, M.P., 2014. How risk management can prevent future wildfire disasters in the wildland-urban interface. Proc. Natl. Acad. Sci. 111, 746–751.
Cotronis, Y., Konstantinidis, E., Louka, M.A., Missirlis, N.M., 2014. A comparison of CPU and GPU implementations for solving the convection diffusion equation using the local modified SOR method. Parallel Comput. 40, 173–185.
Ding, K., Tan, Y., 2015. Attract-repulse fireworks algorithm and its CUDA implementation using dynamic parallelism. Int. J. Swarm Intell. Res. (IJSIR) 6, 1–31.
Duff, I.S., 2004. MA57—a code for the solution of sparse symmetric definite and indefinite systems. ACM Trans. Math Software 30, 118–144.
Evans, D.J., 1984. Parallel SOR iterative methods. Parallel Comput. 1, 3–18.
Forthofer, J.M., Butler, B.W., Wagenbrenner, N.S., 2014. A comparison of three approaches for simulating fine-scale surface winds in support of wildland fire management. Part I. Model formulation and comparison against measurements. Int. J. Wildland Fire 23, 969–981.
Hayati, A.N., Stoll, R., Kim, J., Harman, T., Nelson, M.A., Brown, M.J., Pardyjak, E.R., 2017. Comprehensive evaluation of fast-response, Reynolds-averaged Navier–Stokes, and large-eddy simulation methods against high-spatial-resolution wind-tunnel data in step-down street canyons. Boundary-Layer Meteorol. 164, 217–247.
Hayati, A.N., Stoll, R., Pardyjak, E.R., Harman, T., Kim, J., 2019. Comparative metrics for computational approaches in non-uniform street-canyon flows. Build. Environ. 158, 16–27.
Hayes, L.J.H., 1974. Comparative Analysis of Iterative Techniques for Solving Laplace's Equation on the Unit Square on a Parallel Processor. Ph.D. thesis. University of Texas at Austin.
Helfenstein, R., Koko, J., 2012. Parallel preconditioned conjugate gradient algorithm on GPU. J. Comput. Appl. Math. 236, 3584–3590.
Homicz, G.F., 2002. Three-dimensional wind field modeling: a review. SAND Report 2597. Sandia National Laboratories.
Itu, L.M., Suciu, C., Moldoveanu, F., Postelnicu, A., 2011. GPU accelerated simulation of elliptic partial differential equations. In: Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems, 1. IEEE, pp. 238–242.
Jones, S., 2012. Introduction to dynamic parallelism. In: GPU Technology Conference Presentation S, vol. 338, p. 2012.
Kelley, C.T., 1995. Iterative Methods for Linear and Nonlinear Equations. SIAM.
Kirk, D.B., Wen-Mei, W.H., 2016. Programming Massively Parallel Processors: a Hands-On Approach. Morgan kaufmann.
Konstantinidis, E., Cotronis, Y., 2011. Accelerating the red/black SOR method using GPUs with CUDA. In: International Conference on Parallel Processing and Applied Mathematics. Springer, pp. 589–598.
Krupka, J., Šimecek, I., 2010. Parallel Solvers of Poisson's Equation. Department of Computer Systems, Faculty of Information Technology, Czech Technical University, Prague, MEMICS.
Lambiotte Jr., J.J., 1975. The Solution of Linear Systems of Equations on a Vector Computer. University of Virginia.
Li, R., Saad, Y., 2013. GPU-accelerated preconditioned iterative linear solvers. J. Supercomput. 63, 443–466.
Linn, R.R., Goodrick, S., Brambilla, S., Brown, M.J., Middleton, R.S., O'Brien, J.J., Hiers, J.K., 2020. QUIC-fire: a fast-running simulation tool for prescribed fire planning. Environ. Model. Software 125, 104616.
Lopes, A., 2003. WindStation - software for the simulation of atmospheric flows over complex topography. Environ. Model. Software 18, 81–96.
Moody, M., Stoll, R., Gibbs, J., Pardyjak, E., 2019. QES-fire: a microscale fast response wildfire model. In: AGU Fall Meeting 2019. AGU.

Moussafir, J., Oldrini, O., Tinarelli, G., Sontowski, J., Dougherty, C.M., 2004. 5.26 a New Operational Approach to Deal with Dispersion Around Obstacles: the Mss (Micro Swift Spray) Software Suite.
Neophytou, M., Gowardhan, A., Brown, M., 2011. An inter-comparison of three urban wind models using Oklahoma city joint urban 2003 wind field measurements. J. Wind Eng. Ind. Aerod. 99, 357–368.
NVIDIA, C., 2019. Cuda C++ Programming Guide. NVIDIA Corp.
NVIDIA, C., 2020. CUDA Multi Process Service Overview. NVIDIA Corp.
Pardyjak, E.R., Brown, M.J., 2001. Evaluation of a fast-Response urban wind model: comparison to single building wind-tunnel data. Proceedings of the 3rd International Symposium on Environmental Hydraulics. International Association for Hydro-Environment Engineering and Research, Tempe, AZ.
Pardyjak, E.R., Brown, M., 2003. QUIC-URB v. 1.1: Theory and User's Guide. LA-UR-07-3181. Los Alamos National Laboratory.
Pietro, R.D., Lombardi, F., Villani, A., 2016. CUDA leaks: a detailed hack for CUDA and a (partial) fix. ACM Trans. Embed. Comput. Syst. 15, 1–25.
Pinheiro, A., Desterro, F., Santos, M.C., Pereira, C.M., Schirru, R., 2017. GPU-based implementation of a diagnostic wind field model used in real-time prediction of atmospheric dispersion of radionuclides. Prog. Nucl. Energy 100, 146–163.
Radeloff, V.C., Helmers, D.P., Kramer, H.A., Mockrin, M.H., Alexandre, P.M., Bar-Massada, A., Butsic, V., Hawbaker, T.J., Martinuzzi, S., Syphard, A.D., et al., 2018. Rapid growth of the us wildland-urban interface raises wildfire risk. Proc. Natl. Acad. Sci. Unit. States Am. 115, 3314–3319.
Röckle, R., 1990. Bestimmung der Strömungsverhältnisse im Bereich komplexer Bebauungsstrukturen.
Sasaki, Y., 1958. An objective analysis based on the variational method. Journal of the Meteorological Society of Japan. Ser. II 36, 77–88.
Sasaki, Y., 1970. Some basic formalisms in numerical variational analysis. Mon. Weather Rev. 98, 875–883.
Sasaki, Y., 1970. Numerical variational analysis formulated under the constraints as determined by longwave equations and a low-pass filter. Mon. Weather Rev. 98, 884–898.
Shukla, V., Parikh, K., 1992. The environmental consequences of urban growth: cross-national perspectives on economic development, air pollution, and city size. Urban Geogr. 13, 422–449.
Singh, B., Hansen, B.S., Brown, M.J., Pardyjak, E.R., 2008. Evaluation of the QUIC-URB fast response urban wind model for a cubical building array and wide building street canyon. Environ. Fluid Mech. 8, 281–312.
Singh, B., Pardyjak, E.R., Norgren, A., Willemsen, P., 2011. Accelerating urban fast response Lagrangian dispersion simulations using inexpensive graphics processor parallelism. Environ. Model. Software 26, 739–750.
Tinarelli, G., Brusasca, G., Oldrini, O., Anfossi, D., Castelli, S.T., Moussafir, J., 2007. Micro-swift-spray (mss): a new modelling system for the simulation of dispersion at microscale. general description and validation. In: Air Pollution Modeling and its Application XVII. Springer, pp. 449–458.
Varga, R.S., 1962. Iterative Analysis. Springer.
United Nations, Department of Economic and Social Affairs, 2019. World Urbanization Prospects, the 2018 Revision. United Nations, New York.
USDA, 2019. Wildland fire — usda. https://www.usda.gov/topics/disaster/wildland-fire.
Williams, M.D., Brown, M.J., Boswell, D., Singh, B., Pardyjak, E.M., 2004. Testing of the QUIC-PLUME model with wind-tunnel measurements for a high-rise nuilding. Fifth Symposium on the Urban Environment. AMS, Vancouver, BC.
Young, D., 1954. Iterative methods for solving partial difference equations of elliptic type. Trans. Am. Math. Soc. 76, 92–111.
Zapata, M.U., Van Bang, D.P., Nguyen, K., 2018. Parallel simulations for a 2D x/z two-phase flow fluid-solid particle model. Comput. Fluid 173, 103–110.