International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:9, No:5, 2015

A Parallel Approach for 3D-Variational Data Assimilation on GPUs in Ocean Circulation Models

Rossella Arcucci, Luisa D'Amore, Simone Celestino, Giuseppe Scotti, Giuliano Laccetti

Abstract—This work is the first dowel in a rather wide research activity in collaboration with Euro Mediterranean Center for Climate Changes, aimed at introducing scalable approaches in Ocean Circulation Models. We discuss designing and implementation of a parallel algorithm for solving the Variational Data Assimilation (DA) problem on Graphics Processing Units (GPUs). The algorithm is based on the fully scalable 3DVar DA model, previously proposed by the authors, which uses a Domain Decomposition approach (we refer to this model as the DD-DA model). We proceed with an incremental porting process consisting of 3 distinct stages: requirements and source code analysis, incremental development of CUDA kernels, testing and optimization. Experiments confirm the theoretic performance analysis based on the so-called *scale up factor* demonstrating that the DD-DA model can be suitably mapped on GPU architectures.

Keywords—Data Assimilation, Parallel Algorithm, GPU architectures, Ocean Models.

I. Introduction

DATA Assimilation (DA) is an Uncertainty Quantification technique widely used in simulation science to incorporate observational data into a prediction model [15]. Due to the scale of the forecasting area and the number of state variables used to describe ocean or atmosphere for climate or weather predictions, DA are large scale problems that should be solved in near real-time. During the last 20 years, parallel algorithms for data assimilations reached a widespread interests at many federal research institutes as well as at many universities [NCAR (National Center for Atmospheric Research), NCEP (National Centers for Environmental Prediction), DWD (Deutscher Wetterdienst), UK Met Office, JMA (Japan Meteorological Agency), CMC (Canadian Association of Management Consultants) and the CMCC (Euro- Mediterranean Center for Climate Changes)].
The CMCC makes use of a 3D Variational (3DVar) DA software, called OceanVar, for assimilating data in Mediterranean Forecasting System (MFS) context [10], [3]. MFS is a daily 10-day forecast system in operational use since 1998, and its ocean general circulation model (OGCM) is based on the Ocean Parallelise (OPA) code, which has subsequently been set up for the Mediterranean Sea (NEMO framework) [17].
Together with University of Naples Federico II, CMCC has developed a fully scalable 3DVar DA model which is based on a Domain Decomposition approach (called DD-DA model) [5], [4]. The resulted parallel algorithm consists of several copies of a slightly modified 3D-Var algorithm, each one requiring approximately the same amount of computations on each sub domain and an exchange of boundary conditions between adjacent sub domains. Data flow across the surfaces and a so-called surface-to-volume effect is produced [1].
Over the last few years, the rapid evolution of Graphics Processing Units (GPUs) into powerful, cost-effcient, programmable computing architectures for general purpose computations has provided application potential beyond the primary purpose of graphics processing. As the number of supercomputers equip GPUs is massively increasing [19], large scale problems are embracing GPUs for massive thread level parallelism. GPUs have enjoyed rapid adoption within the high-performance computing (HPC) community because GPUs enable high levels of fine-grain data parallelism. The latest GPU programming interfaces such as NVIDIAs Compute Unified Device Architecture (CUDA) [14], and more recently Open Computing Language (OpenCL) [16] provide the programmer a flexible model while exposing enough of the hardware for optimization. GPU clusters, where fast network connected compute-nodes are augmented with latest GPUs, [18] are now being used to solve challenging problems from various domains. These new systems are designed for high performance as well as high power efficiency, which is a crucial factor in future exascale computing.
However, GPU architecture is unlike that of any other, and designing algorithms to fully harness the capabilities of a GPU is not a straightforward task, especially when one considers the advantages and disadvantages of the various resources that a GPU has available to it. To best utilize the computing capabilities provided by the graphic processors, it is highly desired to study how to map algorithms and programs on them. Briefly, the goal is to reduce the total data transfer time as much as possible, meaning reducing the amount of data that is transferred back and forth between host (the CPU) and device (the GPU).
In this article we describe how DD-DA model is well-suited for efficiently using GPU architecture. The paper is organized as follows. In Section II our parallel approach is presented. The mathematical model we implement is reviewed in Section II-A. In Section II-B a brief overview of the GPU architecture and some programming basics required to understand our methods are provided. Section II-C shows the implementative strategy we used for efficiently using architecture of GPU to develope DD-DA model. In Section III

R.Arcucci is with the Imperial College London, London (UK) e-mail: r.arcucci@imperial.ac.uk
S.Celestino, G.Scotti, L.D'Amore, G.Laccetti are with University of Naples Federico II, Naples (IT), e-mail:
(simone.celestino,giuseppe.scotti,luisa.damore,giuliano.laccetti)@unina.it
L.D'Amore is with Euro Mediterranean Center for Climate Changes, Lecce (IT)

we present selected numerical results to demonstrate the effectiveness of the GPU-based parallel DD-DA algorithm. Section IV concludes the paper and outlines possible future work.

II. Domain Decomposition DA Model for GPUs

The DD-DA model is based on a Domain Decomposition approach for solving Variational Data Assimilation problem [5]. The model uses a partition of the global domain into sub domains. On these sub domains we define local 3D-Var functionals and we prove that the minimum of the global 3D-Var functional can be obtained by collecting the minimum of each local functional. The (global) problem is decomposed into (local) sub problems in such a way. The resulted algorithm consists of several copies of a slightly modified 3D-Var algorithm, each one requiring approximately the same amount of computations on each sub domain and an exchange of boundary conditions between adjacent sub domains.

Fig. 1 shows a simple example of how the DD-DA model works on a decomposition of the global domain in six subdomains. Red points represent the observed data which are distributed geographically as the physical subdomains. Green lines represent the overlapping regions between different physical subdomains. With this decomposition, local DA problem are solved concurrently, each subdomain is processed on a processor node of the supercomputer, which make this method fully scalable and highly parallel. It makes DA applications feasible for big forecasted data and observations.

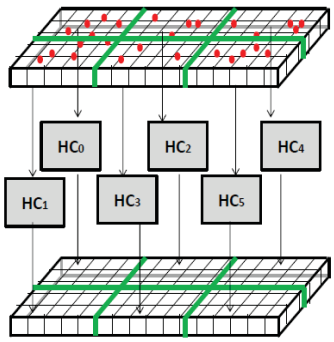

Fig. 1: Example of how the DD-DA method works on a decomposition of the domain in six subdomains processed by six Hardware Components (HC). The red points represent the observed data. The green lines represent the overlapping regions between different physical subdomains.

A. DD-DA computational model

Let t_k, $k = 0, 1, \ldots, n$ be a sequence of observation times and, for each k, let $x_{M_k} \equiv x_M(t_k) \in \Re^N$ be the vector denoting the state of the Mediterranean sea system at time t_k as defined in (1) where T is the three-dimensional temperature field, S the three-dimensional salinity field, η the two-dimensional free surface elevation, and u,v are the total horizontal velocity components and where with $\star$ we denote the vector transposed.

$$x_{M_k} = [T, S, \eta, u, v]^\star \tag{1}$$

At each time step t_k, let y_k be the observations vector as defined in (2) where $\mathcal{H}_k : \Re^N \mapsto \Re^p$ is a non-linear operator collecting the observations at time t_k [3].

$$y_k = \mathcal{H}_k(x_k) \tag{2}$$

Let (3) be an overlapping decomposition of the physical domain Ω such that $\Omega_i \cap \Omega_j = \Omega_{ij} \neq 0$ if Ω_i and Ω_j are adjacent and Ω_{ij} is called *o*verlapping region.

$$\Omega = \bigcup_{i=1}^{N_{sub}} \Omega_i \tag{3}$$

According to this decomposition the DD-DA computational model is a system of N_{sub} non-linear least square problems [10], [5] described in (4)-(6) where J_i in (6) is a cost-function.

$$x_{DA}(t_k) = \sum_{i=1}^{N_{sub}} \tilde{x}_{DA_i}(t_k) \tag{4}$$

$$\tilde{x}_{DA_i} = \begin{cases} argmin_{x_k} J_i(x_k) & on \quad \Omega_i \\ 0 & on \quad \Omega - \Omega_i \end{cases} \tag{5}$$

$$J_i(x_k) = \|\mathcal{H}_{\mathbf{k_i}}(\mathbf{x}_{k_i}) - \mathbf{y}_{k_i}\|^2_{\mathbf{R_i}} + \|\mathbf{x}_{k_i} - \mathbf{x}_{\mathbf{M}k_i}\|^2_{\mathbf{B_i}} + \\ + \|\mathbf{x}_{k_i}/\Omega_{ij} - \mathbf{x}_{k_j}/\Omega_{ij}\|^2_{\mathbf{B_{ij}}} \tag{6}$$

x_{DA} in (4) is the *analysis* (i.e. the estimation of the vector x_{k_i} at time t_k). The variables $\mathbf{x}_{\mathbf{M}k_i}$ and $\mathbf{y}_{k_i}$ in (6) are the same vectors $\mathbf{x}_{\mathbf{M}k}$ and $\mathbf{y}_k$ in (1) and (2) defined on the subdomain Ω_i, $\mathbf{R_i}$ and $\mathbf{B_i}$ are the covariance matrices defined in (7) and (8), whose elements provide the estimate of the errors on y_{k_i} and on $x_{M_{k_i}}$, respectively. Also the variables $\mathbf{x}_i/\Omega_{ij}$, $\mathbf{x}_j/\Omega_{ij}$, $\mathbf{B_{ij}}$ are the restriction on Ω_{ij} of these quantities.

$$R_i = \sigma_o^2 I_p \tag{7}$$

where σ_o^2 is the observational error variance.

$$B_i = \sigma_b^2 C \tag{8}$$

where

$$c_{ij} = \rho^{|i-j|^2}, \quad \rho = exp\left(\frac{\Delta x^2}{2L^2}\right), \quad |i-j| < N/2$$

N is the size of domain, C denotes the Gaussian correlation structure of the background errors while σ_b^2 is the background error variance. As a consequence:

$$\mu(B) = \mu(C)$$

where $\mu(\cdot)$ denote the condition number.

The ill conditioning of the DA inverse problem [11] (i.e. the sensitivity of the analysis to small perturbations in the data),

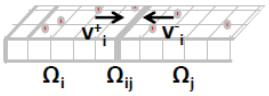

Fig. 2: Model variables on overlapping regions.

depends on the conditioning of the Hessian of each J_i in (6). Small errors in the Hessian lead to large errors in its inverse, so the computed solution to the DA problem may be very inaccurate. In designing of the DA schemes, it is important to ensure that the conditioning of the Hessian is as small as possible, or it is essential to use preconditioning techniques to improve the conditioning.
In our model, matrix B_i is decomposed as in (9) to have a preconditioner.

$$B_i = U_i D_i U_i^T = U_i D_i^{\frac{1}{2}} D_i^{\frac{1}{2}} U_i^T = \left(U_i D_i^{\frac{1}{2}}\right)\left(U_i D_i^{\frac{1}{2}}\right)^T \quad (9)$$

The matrix $V_i = U_i D_i^{\frac{1}{2}}$ such that (10) is a preconditioner.

$$B_i = V_i V_i^T. \quad (10)$$

Let $d = [y_k - H(x_k)]$ be the *misfit*, by using the following linearization of $\mathcal{H}$:

$$\mathcal{H}(x) = \mathcal{H}(x + \delta x) + H\ \delta x$$

where H is the matrix obtained by the first order approximation of the Jacobian of $\mathcal{H}$ and, by setting $v_i = V_i^T \delta x_i$, the *preconditioned* (see [2]) cost function is:

$$J_i(v_i) = \frac{1}{2} v_i^T v_i + \frac{1}{2}(H_i V_i v_i - d_i)^T R_i^{-1}(H_i V_i v_i - d_i) +$$
$$+\frac{1}{2}(V_{ij} v_i^+ - V_{ij} v_i^-)^T (V_{ij} v_i^+ - V_{ij} v_i^-) \quad (11)$$

where v_i^+ and v_i^- are shown in Fig. 2.

On each subdomain of Ω, the function J_i ($\forall i = 1, \ldots, N_{sub}$), is minimized using the L-BFGS method [20], [8].

B. GPU architecture

This section is a short overview of the basic key properties of a GPU device architecture and CUDA API necessary to comprehend our implementation of the DD-DA algorithm for GPU which is discussed on the next section. In this section we just summarize some key properties, in [13] can be found more detailed descriptions of the architecture and the programming model. Fig. 3 shows graphically the layout of a GPU. A GPU can be viewed as a set of independent streaming multiprocessors. One such multiprocessor contains, amid other components, several scalar processors which can execute floating-point arithmetic (ALU). The Global Memory can be accessed by all processors while the Shared Memory can be accessed by all scalar processors of a multiprocessor.

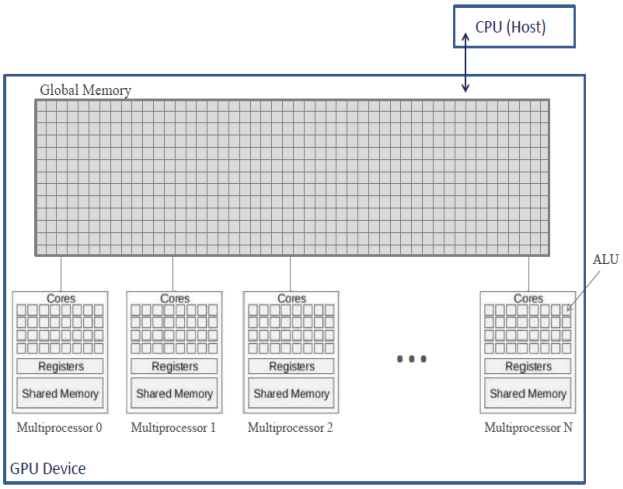

Fig. 3: GPU device architecture

Before describing the algorithm, it is worth noting that the GPU architecture is much more optimized for performing calculations than for memory accesses. Therefore, considering the multiple types of memory that the GPU architecture typically includes, it is important to keep this in mind when accessing these types of memory, particularly the slower, off-chip ones such as the GPUs global and the hosts main memory. The most costly memory access is by far the host-to-device (CPU to GPU) data transfer, and reducing that transfer can have a tremendous impact on the overall performance of any algorithm that is implemented in part or fully on a GPU.

C. Mapping of DD-DA model on GPU architecture

Let $n_x \times n_y \times n_z$ be the size of a computational grid which discretize Ω and let $N_{sub} = p \times q$ be the number of sub domains partition of Ω. We group gridpoints into blocks, we partition the computational grid into $p \times q$ three-dimensional (3-D) blocks of size $n_{x_{loc}} \times n_{y_{loc}} \times n_z$, each of which can be viewed as consisting of q two-dimensional (2-D) blocks (see Fig. 4) with $n_{x_{loc}}$ and $n_{y_{loc}}$ defined in (12). These dimensions include overlapping ($2o_x \times 2o_y$).

$$n_{x_{loc}} = \frac{n_x}{p} + 2o_x, \quad n_{y_{loc}} = \frac{n_y}{q} + 2o_y. \quad (12)$$

In Fig. 4, orange points represent overlapping regions. From the viewpoint of the blocks, the overlapping region are not completely part of blocks, but come from adjacent blocks: the North, South, East, West overlapping are from neighboring blocks in those respectively directions.
Let us now describe the mapping of the DD-DA model on the GPU architecture. In any GPU implementation, the CPU (the host) runs the program and unloads some kernel functions (generally the more computationally demanding code parts) to the GPU (the device). In our algorithm, the CPU acquires the input data (data from forecasting model and observations) of

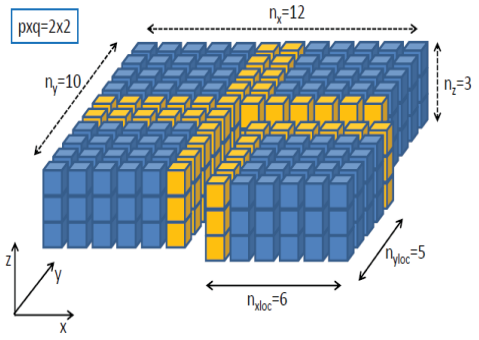

Fig. 4: Application of our partitioning approach on an example computational grid of size $n_x \times n_y \times n_z = 12 \times 10 \times 3$, with $n_{xloc} = 6$ and $n_{yloc} = 5$. In this example, the computational domain is partitioned into 3-D blocks of size $n_{xloc} \times n_{yloc} \times n_z = 6 \times 5 \times 3$.

the geographical region which is been assigned to it. It also computes N_{sub} covariances matrices and the misfits.

In any GPU implementation when a C program, using CUDA extensions and running on the CPU, invokes a kernel, many copies of this kernel (which are referred to as threads) are distributed to the available multiprocessors, where they are executed. Threads are grouped into thread-blocks, which are in turn arranged on a grid. Threads in a thread-block are executed by processors within a single multiprocessor. All threads in a thread-block can read from and write on any shared memory location assigned to that thread-block. Threads within the same thread-block are able to communicate with each other very efficiently via the shared memory and are able to synchronize their executions. Thread-blocks can execute in any order relative to each other, which allows transparent scalability in the parallelism of CUDA kernels.

In our algorithm, we decided to assign each 3-D block of physical grid-points to a thread-block which carries out all the computations/work associated with all grid-points. Hence, thread-blocks are been solved on the GPU concurrently. Each one computes the minimum of the function J_i defined in (11) on its part of the computational grid, by using a CUDA version of the L-BFGS routine [8]. The boundary conditions between adjacent subdomains are efficiently communicated via the shared memory by introducing a synchronization of thread-blocks, as described in Algorithm 1. The global solution x_{DA} defined in (4) is computed on the host finally.

III. Numerical Results

In this section, we present selected numerical results to demonstrate the effectiveness of the GPU-based parallel DD-DA algorithm. We used the CUDA 3.2 driver and toolkit, and all the experiments with the GPU code were conducted on a NVIDIA Tesla K20, which allows double-precision computations, and is connected to a quad-core Intel i7 CPU running at 3.07GHz, 12 GB of RAM. Out test case is based on shallow water equations which are a a simplified version of NEMO forecasting model. Occording with NEMO, we have a variable time t, and space coordinates (x and y) as independent

Algorithm 1: GPU implementation of DD-DA scheme

CPU
1: **Acquire** the observations $\mathbf{y}_k$ and the model vector $\mathbf{x}_{Mk}$
2: **for** i=1 **to** N_{sub} **{**
3: **Define** the operators $\mathbf{H}_{ki}$
4: **Compute** $di \leftarrow \mathbf{y}_{ki} - \mathbf{H}_{ki}\mathbf{x}_{Mki}$
5: **Estimate** the operators $\mathbf{R}_{ki}$ and $\mathbf{B}_{ki}$
6: **Compute** the matrix $\mathbf{V}_{ki}$ from $\mathbf{B}_{ki}$ **}**

GPU
7: **Define** the initial value of $\mathbf{x}^{DA}{}_{ki}$
8: **Compute** $\mathbf{v}_{ki} \leftarrow \mathbf{V}^{*}_{ki}\mathbf{x}^{DA}{}_{ki}$
9: **repeat**
10: **Synchronize** (until boundary conditions from the adjacent domains are available)
11: **Compute** $J_{ki} \leftarrow J_i(\mathbf{v}_{ki})$
12: **Compute** $gradJ_{ki} \leftarrow \nabla J_i(\mathbf{v}_{ki})$
13: **Compute** new values for $\mathbf{v}_{ki}$ by the L-BFGS steps from J_{ki} and $gradJ_{ki}$
14: **until** (Convergence on the $\mathbf{v}_{ki}$ values are obtained)
15: **Compute** $\mathbf{x}^{DA}{}_{ki} \leftarrow \mathbf{x}_{Mki} + \mathbf{V}_{ki}\mathbf{v}_{ki}$

CPU
16: **Compute** $\mathbf{x}^{DA}{}_{k} \leftarrow \text{sum}_i\, \mathbf{x}^{DA}{}_{ki}$

variables. The dependent variables are the fluid height or depth h and the two-dimensional fluid velocity field u and v. The state variable is given in (13).

$$x_{M_k} = [h, u, v]^{\star}. \tag{13}$$

We assume $n_x = n_y = n$ and $n_z = 3$ which implies a problem size $N = n^2 \times 3$. The time step used for the temporal discretization of model is $d_t = 0.01$.

For executing Algorithm 1 on shallow water test case, we need to acquire x_{M_k} in (13) by running shallow water forecasting code. Observations vector y_k is obtained by randomly choosing and randomly perturbing values of x_{M_k} [5]. We assume H_{k_i} in (2) is a piecewise linear interpolation function and operators B_{k_i} and R_{k_i} defined in (14) and (15).

$$B_{k_i} = \sigma_b^2 C, \quad \sigma_b^2 = 0.5, \quad L = 1 \tag{14}$$

$$R_{k_i} = \sigma_o^2 I, \quad \sigma_o^2 = 0.5. \tag{15}$$

For fixed values of N_{sub}, the size of each subdomain used is $\frac{N}{N_{sub}}$ as explained in Section II-C.

In [5] the authors provided a formal mathematical proof of the reliability of DD-DA model and accuracy of its solution. Also, our implementation on GPU does not affect accuracy of numerical results as the arithmetic system we are using is double precision.

Let $T^{N_{sub}}(N)$ be the execution time of Algorithm 1 for a problem size N defined in (16) where $T_{\mathbf{H}}^{N_{sub}}(N)$ is the execution time of algorithm running on the CPU, $T_{com(\mathbf{H}\leftrightarrow\mathbf{D})}^{N_{sub}}(N)$ is the communication time between host and device and with $T_{\mathbf{D}}^{N_{sub}}(N)$ execution time of algorithm running on the GPU. Then

$$T^{N_{sub}}(N) \underbrace{=}_{def} T_{\mathbf{H}}^{N_{sub}}(N) + T_{com(\mathbf{H}\leftrightarrow\mathbf{D})}^{N_{sub}}(N) + T_{\mathbf{D}}^{N_{sub}}(N) \quad . \tag{16}$$

TABLE I: Values of Execution Time of Algorithm Running on GPU for N=128 which Give a Problem Size $O(10^7)$.

N	p	N_{sub}	$T_{\mathbf{D}}^{N_{sub}}(N)$
$O(10^7)$	1	2	0.144
	2	4	0.044
	4	8	0.025
	8	16	0.024

TABLE II: Values of $T_{flop(\mathbf{D})}^{N_{sub}}$ and Measured Scale-Up Factor Compared with Theoretical One.

N	p	$T_{flop(\mathbf{D})}^{N_{sub}}(N)$	Measured S_{2p}^{DD-DA}	S_{2p}^{DD-DA}
$O(10^7)$	1	0.127	-	-
	2	0.027	4.7	4
	4	0.008	15.9	8
	8	0.007	18.1	16

As explained at the end of Section II-C, in Algorithm 1, the host acquires input data, it computes operators H_{k_i}, d_{k_i}, R_{k_i} and V_{k_i} and sends all these data to device which solves concurrently N_{sub} thread-blocks for computing the minimum of function J in (11). In practice, the host computes operators which are input for computations on device and the device solves the DD-DA model (4)-(6). Hence, $T_{\mathbf{H}}^{N_{sub}}(N)$ is execution time that CPU needs for building data. These data are transferred just once as well as output data $x_{k_i}^{DA}$ so we have that $T_{com(\mathbf{H}\leftrightarrow\mathbf{D})}^{N_{sub}}(N)$ is reduced to the time of I/O transfer. For this reasons we evaluate the performance of DD-DA implementation on GPU by analysing $T_{\mathbf{D}}^{N_{sub}}(N)$. Table I shows execution time of algorithm running on GPU for N=128 which give a problem size $O(10^7)$.

Execution time $T_{\mathbf{D}}^{N_{sub}}(N)$ is given by summing time for computing and time for global and local memories transfers (see for instance [6]):

$$T_{\mathbf{D}}^{N_{sub}}(N) \underbrace{=}_{def} T_{flop(\mathbf{D})}^{N_{sub}}(N) + T_{mem(\mathbf{D})}^{N_{sub}}(N), \qquad (17)$$

where $T_{mem(\mathbf{D})}^{N_{sub}}(N)$ is the time for global and local memories transfers into the device. $T_{flop(\mathbf{D})}^{N_{sub}}(N)$ is the computing time required for execution of floating point operations.

$T_{mem(\mathbf{D})}^{N_{sub}}(N)$ can be estimates as in (18) by using size of processed data D_N which is the problem size espressed in GB and bandwidth value B_W (see [12]) which is the rate of data transfer espressed in GB/seconds.

$$T_{mem(\mathbf{D})}^{N_{sub}}(N) \underbrace{\backsimeq}_{def} \frac{D_N}{B_W}\ secs\,. \qquad (18)$$

Theoretical bandwidth B_W can be calculated using hardware specifications available in the product literature as in (19) where M_{CR} is the memory clock rate and M_{SW} is the wide memory interface. In (19) we convert the memory clock rate to Hz, multiply it by the interface width (divided by 8, to convert bits to bytes) and multiply by 2 due to the double data rate. Finally, we divide by 10^9 to convert the result to GB/secs.

$$B_W = M_{CR}\cdot 10^6 \cdot \frac{M_{SW}}{8}\cdot\frac{2}{10^9}\ GB/secs\,. \qquad (19)$$

In our case, the NVIDIA Tesla K20 GPU uses DDR (double data rate) RAM with a memory clock rate of 2,6 MHz and a 320-bit wide memory interface. Using these data items, the peak theoretical memory bandwidth of the NVIDIA Tesla K20 is 208 GB/secs, as computed in the following

$$B_W = 2600\cdot 10^6\cdot\frac{320}{8}\cdot\frac{2}{10^9}\ GB/s = 208\ GB/secs\,.$$

For a domain of dimension $O(10^7)$ we have $D_N = 3.7\ GB$ which gives $T_{com(\mathbf{D})} \backsimeq 3.7/208\ secs \backsimeq 0.017\ secs$.

The Scale-Up factor (see [5]) of algorithm running on the GPU is function of $T_{flop(\mathbf{D})}^{N_{sub}}(N)$ as given in (20)

$$S_{N_{sub}}^{DD-DA} \underbrace{=}_{def} \frac{T_{flop(\mathbf{D})}^{N_{sub}}(N)}{N_{sub}\ T_{flop(\mathbf{D})}^{N_{sub}}\left(\frac{N}{N_{sub}}\right)}\,. \qquad (20)$$

We implement on GPU a decomposition into subdomains/thread-blocks as a multiple of 2, that is $N_{sub} = 2p$ where p is called "decomposition step". For example, for $p = 1$ the global domain is divided into $N_{sub} = 2$ subdomains, for $p = 2$ it is divided into $N_{sub} = 4$ subdomains, etc. Also we assume (21) be an eximation for $T_{flop(\mathbf{D})}^{N_{sub}}(N)$ in terms of time complexity, where $T(N)$ denotes the time complexity of Algorithm and t_{flop} is the time required for one floating point operation.

$$T_{flop(\mathbf{D})}^{N_{sub}}(N) = T(N)\times t_{flop}\,. \qquad (21)$$

With this assumption, the scale-up factor of algorithm is given in (22) as we have in our case $T(N) = O(N^2)$.

$$S_{2p}^{DD-DA} = \frac{T(N)}{2p\ T\left(\frac{N}{2p}\right)} = O\left(\frac{N^2}{2p\left(\frac{N^2}{2^2p^2}\right)}\right) = O\left(2p\right). \qquad (22)$$

Table 2 shows values of $T_{flop(\mathbf{D})}^{N_{sub}}$ and values of measured Scale-up factor compared with theoretical once. Finally, we observe that measured values of Scale-up factor are defined as in (23) with $\alpha_1 < 1$ and $\alpha_{N_{sub}} < 1$ as the parallel implementation we have by using CUDA.

$$measured\ S_{N_{sub}}^{DD-DA} \underbrace{=}_{def} \frac{T_{flop(\mathbf{D})}^{N_{sub}}(N)\ \alpha_1}{N_{sub}\ T_{flop(\mathbf{D})}^{N_{sub}}\left(\frac{N}{N_{sub}}\right)\ \alpha_{N_{sub}}}, \qquad (23)$$

since $\frac{\alpha_1}{\alpha_{N_{sub}}} = \beta \geq 1$, we have

$$measured\ S_{N_{sub}}^{DD-DA} \simeq \frac{T_{flop(\mathbf{D})}^{N_{sub}}(N)}{N_{sub}\ T_{flop(\mathbf{D})}^{N_{sub}}\left(\frac{N}{N_{sub}}\right)}\beta$$

$$\geq \frac{T_{flop(\mathbf{D})}^{N_{sub}}(N)}{N_{sub}\ T_{flop(\mathbf{D})}^{N_{sub}}\left(\frac{N}{N_{sub}}\right)} = S_{N_{sub}}^{DD-DA}.$$

IV. Conclusions and future work

Moving forward to exascale will put heavier demands on algorithms in at least two areas: the need for increasing amounts of data locality in order to perform computations efficiently, and the need to obtain much higher factors of fine-grained parallelism as high-end systems support increasing numbers of compute threads. As a consequence, parallel algorithms must adapt to this environment, and new algorithms and implementations must be developed to extract the computational capabilities of the new hardware.

We presented a parallel algorithm on GPU which is based on a domain decomposition approach. The standard approach for reducing the execution time of an algorithm on GPU is to place concurrency inside the most time-consuming computational kernels, i.e. to introduce a parallelism at the level of fine-grained computations. Furthermore, in order to reduce the data movement between host and device, thus increasing the computation/communication ratio, the parallel algorithm relies on a domain decomposition approach that introduces a coarse-grained data decomposition which have more favorable performance characteristics. Finally, coarse and fine grained computations are suitably mapped on the processing elements of our target architecture, that is the multiprocessors and the ALUs respectively [9].

We are currently working on the deployment of this algorithm in a concrete scenario. Mainly, we are working on the variational DA systems used with the NEMO ocean model, on emerging exascale computing architectures [7].

References

[1] L. Carracciuolo, L. D'Amore, A. Murli, Towards a parallel component for imaging in PETSc programming environment: A case study in 3-D echocardiography, Parallel Computing, Vol. 32, (1), 2006, pp. 67-83.

[2] L. D'Amore, R. Arcucci, L. Marcellino and A. Murli, *HPC computation issues of the incremental 3D variational data assimilation scheme in OceanVar software* - Journal of Numerical Analysis, Industrial and Applied Mathematics, vol. 7, no. 3-4, 2012, pp. 91-105.

[3] L. D'Amore, R. Arcucci, L. Marcellino, A. Murli - *A Parallel Three-dimensional Variational Data Assimilation Scheme* - Numerical Analysis and Applied Mathematics, AIP Conference Proccedings, Vol. 1389, 2011, pp. 1829-1831.

[4] L. D'Amore, R. Arcucci, L. Carracciuolo, A. Murli - *DD-OceanVar: a Domain Decomposition fully parallel Data Assimilation software in Mediterranean Sea* - Procedia Computer Science 18, 2013, pp. 1235-1244.

[5] L. D'Amore, R. Arcucci, L. Carracciuolo, A. Murli - *A Scalable Approach for Variational Data Assimilation* - Journal of Scientific Computing, Vol. 61, 2014, pp. 239-257.

[6] L. D'Amore, D. Casaburi, A. Galletti, L. Marcellino, A. Murli - *Integration of emerging computer technologies for an efficient image sequences analysis*, Vol. 18, (4), 2011, pp. 365-378.

[7] L. D'Amore, A. Murli, V. Boccia, L. Carracciuolo - *Insertion of PETSc in the NEMO stack software Driving NEMO towards Exascale Computing*, High Performance Computing and Simulation (HPCS), July 2014, pp. 724 - 731, DOI:10.1109/HPCSim.2014.6903761.

[8] L. D'Amore, G. Laccetti, D. Romano, G. Scotti, A. Murli - *Towards a parallel component in a GPU-CUDA environment: a case study with the L-BFGS Harwell routine* - International Journal of Computer Mathematics, DOI: 10.1080/00207160.2014.899589, 2015, Vol 92 (1), pp. 59-76.

[9] L. D'Amore , D. Casaburi, A. Galletti, L. Marcellino, A. Murli - *Integration of emerging computer technologies for an efficient image sequences analysis* - Integrated Computer-Aided Engineering, Vol. 18, (4), 2011, pp. 365-378.

[10] S. Dobricic, N. Pinardi, *An oceanographic three-dimensional variational data assimilation scheme* - Ocean Modelling 22, 2008, pp. 89-105.

[11] S.A. Haben, A.S. Lawless,N.K. Nichols: *Conditioning of the 3DVAR Data Assimilation Problem,* Mathematics Report 3/2009. Department of Mathematics, University of Reading (2009)

[12] M. Harris - *How to Implement Performance Metrics in CUDA C/C++* - November 7 2012, NVIDIA Web Site.

[13] NVIDIA, NVIDIA Compute Unified Device Architecture programming guide version 2.3, NVIDIA Developer Web Site, (2009). Available at http://developer.download.nvidia.com.

[14] NVIDIA, NVIDIA CUDA Programming Guide 3.1.1, 2010.

[15] E. Kalnay - *Atmospheric Modeling, Data Assimilation and Predictability*. - Cambridge University Press, Cambridge, MA (2003)

[16] Khronos OpenCL Working Group, The OpenCL Specification: Version 797 1.1, 2010.

[17] *The NEMO System Home Page* - http://www.nemo-ocean.eu

[18] M. Showerman, J. Enos, A. Pant, V. Kindratenko, C. Steffen, R. Pennington, W.M. Hwu - QP: *A heterogeneous multi-accelerator cluster* - Proceedings of the 10th LCD International Conference on High-Performance Clustered Computing, Boulder, Colorado, 2009.

[19] TOP500 Supercomputer Site. 2014. TOP500 Supercomputer Novermeber 2014 List. http://www.top500.org/lists/2014/11

[20] C. Zhu, R.H. Byrd, P. Lu, and J. Nocedal, *Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization,* ACM Trans. Math. Softw. 23, 1997, pp. 550-560.

Rossella Arcucci Her area of expertise is in Numerical Analysis, Scientific Computing and development of methods, algorithms and software for scientific applications on high performance architectures including parallel and distributed computing. During her master degree in Mathematics she started to work on mathematical models to study real phenomena and in her master degree thesis in 2008, she used discretization methods to solve partial differential equations (PDE). She gave particular attention to discretization methods for elliptic, parabolic and hyperbolic problems which are the basis of all the models that describe real phenomena. During and after her PhD (obtained in 2012), she worked on Data Assimilation problem. Her main contribution was been the development of a fully scalable mathematical model for Variational Data Assimilation based on Domain Decomposition method. Another interest of her is on high performance computing (HPC). In this field she worked on a project aimed at the development of fully scalable software for Data Assimilation able to effectively take advantage of the available HPC resources and handle big data. By working on numerical models which describe real phenomena, she realised the importance of producing solutions as accurate as possible. She became aware of the importance for a scientist of knowing how errors generated in hardware propagate in numerical solutions. She has worked on models for error detection in circuits in the context of reliable systems with electrical and electronic engineers. At the moment, she is combining her competences in numerical analysis and hardware architectures by working on numerical and parallel techniques for both improving accuracy of solution and reducing execution time for solving Data Assimilation models on supercomputers. She is a researcher of University of Naples Federico II and she is collaborating with a group of physics at Imperial College London on Data Assimilation problems for oceanographic data.

Luisa D'Amore. Degree in Mathematics, in 1988. PhD in Applied Mathematics and Computer Science, in 1995. Researcher in Numerical Analysis, in 1996. Associate professor of Numerical Analysis, , at University of Naples Federico II, (IT), since 2001. Associate staff researcher of the ASC (Advanced Scientific Computing) Division of CMCC (Euro Mediterranean Center on Climate Changes), since 2011. Member of the Academic Board of the PhD in Mathematics and Informatics, of University of Naples Federico II. Teacher of courses in Numerical Analysis, Scientific Computing and Parallel Computing. Research activity focuses on Scientific Computing and it is addressed to the numerical solution of ill-posed inverse problems with applications in image analysis, medical imaging, astronomy, digital restoration of films and data assimilation. The need of computing the numerical solution on a suitable time, induced by the applications, often requires the use of advanced computing architectures. This involves designing and development of algorithms and software capable of exploiting the high performance of emerging computing infrastructures. Research produces a total of about 100 publications in refereed journals and conference proceedings.

Simone Celestino He is currently a PhD student in Mathematics and Computer Science at University of Naples "Federico II". His area of interest is the High Performance Computing with particular attention to parallel implementations on General Purpose Graphic Processing Unit (GPGPU). He had a Master Degree in Computer Science obtained at the same University where he is a PhD student, the subject of him thesis was on the development of a parallel software implementing a rendering algorithm on GPUs.

Giuseppe Scotti. Degree in Computer Science at University of Naples "Parthenope". His thesis was on an implementation of a biological model on Graphics Processing Units (GPU). He is currently a master degree student in Computer Science at University of Naples Federico II. During his degree and master degree he also worked as software developer in High Performance Computing.

Giuliano Laccetti is presently Full professor of computer science at the University of Naples Federico II, in Naples, Italy. He received his Laurea degree (cum laude) in Physics from the University of Naples; his main research interests are Mathematical Software, Scientific Computing, High Performance Architecture for Scientific Computing, Distributed Computing, Grid Computing, Cloud Computing, Algorithms on emerging hybrid architectures (CPU+GPU, etc.). He is author (or co-author) of about 90 papers published on refereed international Journals, chapters of books, Conference Proceedings and Technical Reports. He has been involved in several EU funded Projects (EGEE, EGEE II, EGEE III; in this last case he served as University of Naples scientific coordinator). He has been involved also in National (EU funded) Projects as SCOPE, and, presently, RECAS; in this case, he is member of the Scientific and Management Board and he is also coordinator of the curriculum Master Degree of the University of Naples about Technologies for The High Performance Scientific Computing, funded by the RECAS Project itself. In the recent past, he organised and chaired international workshops (joined to the PPAM Conference, from 2007 to 2013) in Gdansk, Torun, Warsaw, as well as he is planning to organise another one next year, in Krakow, about methodologies, algorithms and software for hybrid computer architectures; most recently, he (co)organised and (co)chaired, in Naples, the International Conference Advances in Pure and Applied Mathematics. Giuliano Laccetti is head of the High Performance Scientific Computing Lab of the Department of Mathematics of the University of Naples; he is also member of the editorial board of the journal Advances in Computer Science and Engineering. Presently, he teaches Computer Programming I and Computer Programming II to Computer Science undergraduate students, and Parallel and Distributed Computing to Computer Science graduate students, as well as to Mathematics graduate students. He is also member of the Teaching and Steering Committee of the Ph.D. school on Mathematics and Computer Science of the University of Naples. Giuliano Laccetti is (or has been) member of ACM, IEEE-Computer Society, SIAM, SIMAI, AICA.