Procedia Computer Science

Volume 51, 2015, Pages 170–179

ICCS 2015 International Conference On Computational Science

# Toward a multi-level parallel framework on GPU cluster with PetSC-CUDA for PDE-based Optical Flow computation

S. Cuomo[1], A. Galletti[2], G. Giunta[2], and L. Marcellino[2]

[1] University of Naples Federico II, Department of Mathematics and Applications, Naples, Italy
salvatore.cuomo@unina.it

[2] University of Naples Parthenope, Department of Science and Technology, Naples, Italy
ardelio.galletti, giulio.giunta, livia.marcellino@uniparthenope.it

**Abstract**

In this work we present a multi-level parallel framework for the Optical Flow computation on a GPUs cluster, equipped with a scientific computing middleware (the PetSc library). Starting from a flow-driven isotropic method, which models the optical flow problem through a parabolic partial differential equation (PDE), we have designed a parallel algorithm and its software implementation that is suitable for heterogeneous computing environments (multiprocessor, single GPU and cluster of GPUs). The proposed software has been tested on real SAR images sequences. Numerical experiments highlight the performance of the proposed software framework, which can reach a gain of about 95% with respect to the sequential implementation.

*Keywords:* optical flow, numerical solution of PDEs, scientific computing libraries, GP-GPU, mulitilevel parallel computing.

## 1 Introduction

A large number of processes in computer vision is based on the image motion measurement known as optical flow computation [1]. It has been widely used in many applications such as medical imaging, video coding, tracking problems, and so on. Generally, optical flow estimation techniques are classified into four main categories: differential, matching, energy-based, and phase-based methods. In this paper, we deal with a differential method, namely a version of Horn and Schunck method [12] proposed in [19]. Because of the well known high computational complexity required for solving the optical flow problem, several approaches for efficient parallel software implementations have been proposed. Among them, multicore-cluster approaches [9], parallel pipeline procedures [5], domain decomposition strategies [14]. Recently, on the track of current developments of Graphics Processing Units (GPUs) for High Performance Computing (HPC) [6, 7, 10], motion estimation has been accelerated, with good results, in [15, 17]. The

 Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2015


doi:10.1016/j.procs.2015.05.220

main contribution of this paper is the development of a parallel software for computing the optical flow specifically tailored for a cluster of multiprocessors composed of many-core CPUs and GPUs. In order to exploit the computational power of such architecture, our software is structured as a multi-level framework which implements a fine-to-coarse parallelization strategy, based on a standard fine-grained concurrency at level of the floating-point operations, and a coarse-grained concurrency through problem decomposition. We use a finite differences approach to discretize the differential problem and provide a suitable matrix representation of the resulting discrete problem. The linear systems of equations is solved by means of the preconditioned Krylov iterative method GMRES [16]. The latter compute intensive numerical task is carried out using the PetSc library (Portable, Extensible Toolkit for Scientific Computation) [21]. PetSc consists of routines for managing vectors, sparse and dense matrices, linear and nonlinear system solvers, and even special classes for operations on GPUs [22]. As case study, we considered the analysis of a SAR image sequence of moving clouds, available at [23]. Our experiments show an improvement in performance up to 95% with respect to the sequential software.
The paper is organized as follows. In section 2 we describe the parabolic PDE that models the problem. In section 3, we present the discretization of the space-time and scale variables, and the numerical formulation of the problem. In section 4, we briefly discuss the algorithm for the basic computational task of the numerical problem. In section 5, we outline the main features of our software and the underlying parallel strategies. In section 6, we show the results of some experiments on real sequences of images and highlight the performance gain that our software can achieve. Final conclusions are reported in section 7.

## 2 Mathematical model

An exhaustive overwiev of differential methods for the computation of the optical flow is in [18]. To mathematically define the approach, let us give the following **definition**:
*Let $J \subset \Re$ be a bounded interval. Given $t \in J$ , let $P(t) = (x(t), y(t)) \in \Omega$ , where $\Omega = \Omega_x \times \Omega_y \subset \Re^2$ is the image plane. The image sequence brightness is the piecewise differentiable function:*

$$I : \big(t, P(t)\big) \in J \times \Omega \mapsto I(t) \equiv I(t, P(t)) \in [0, L]$$

*where $L$ is the intensity level depth.*
According to the latter definition, the *motion trajectory* of each point $P(t) = (x(t), y(t)) \in \Omega$ is defined as a sequence of positions of $P(t)$, as $t$ varies in $J$.

*The optical flow $\omega(t) \equiv \omega(t, P(t))$ is the velocity vector whose components are*

$$u(t) = \frac{d}{dt}x(t), v(t) = \frac{d}{dt}y(t).$$

In other words, the optical flow $\omega(t) = (u(t), v(t))$ defines a motion field.

In a differential approach setting, for each time $t \in J$ and for each point $P(t)$, the optical flow $\omega(t)$ can be obtained as the asymptotic value ($\theta \mapsto \infty$) of the solution $\omega^\theta \equiv \omega(\theta; t, P(t)) = (u^\theta, v^\theta)$) of the system of two non linear parabolic (diffusion-reaction) PDEs, with zero initial and Dirichlet boundary conditions [19]

$$\begin{cases} \frac{\partial}{\partial\theta}u^\theta = \alpha\nabla\cdot\left[g\left(||\nabla u^\theta||^2 + ||\nabla v^\theta||^2\right)\nabla u^\theta\right] - 2\left[< \nabla_s I, \omega^\theta > + I_t\right] I_x \\ \frac{\partial}{\partial\theta}v^\theta = \alpha\nabla\cdot\left[g\left(||\nabla u^\theta||^2 + ||\nabla v^\theta||^2\right)\nabla v^\theta\right] - 2\left[< \nabla_s I, \omega^\theta > + I_t\right] I_y \end{cases} \quad (1)$$

where: $\alpha$ is the regularization parameter, $\nabla\cdot$ is the divergence operator, $<,>$ is the dot scalar product, $||\nabla u^\theta||^2$ and $||\nabla v^\theta||^2$ are the Euclidean norms of the spatial gradients of $u^\theta$ and $v^\theta$, the diffusivity function $g(s) = 1/\sqrt{1+s^2/\lambda^2}$ is the edge-indicator function (or smoothness constraint), $\nabla_s I$ is the spatial gradient of the brightness function $I$, $I_x$ and $I_y$ are the spatial derivatives with respect to $x$ and $y$, and $I_t$ is the time derivative of $I$. Finally, $\theta$ is the *scale* parameter, which represents an artificial evolution variable, generally introduced to regularize diffution-reaction PDEs [13].

We remark that $t$ is the time variable that specifies a frame in the image sequence, while $\theta$ is an evolution variable which describes an artificial diffusion-reaction process that has the value of the optical flow $\omega(t) = (u(t), v(t))$ as steady state, i.e. $\lim_{\theta\to\infty} \omega^\theta = \omega(t)$.

Denoting by $g^\theta$ the value of the diffusity-function at a scale value $\theta$, i.e. $g^\theta = g\left(||\nabla u^\theta||^2 + ||\nabla v^\theta||^2\right)$, then after expanding the terms $\nabla\cdot(g^\theta\nabla u^\theta)$, $\nabla\cdot(g^\theta\nabla v^\theta)$, system (1) can be written as

$$\begin{cases} \frac{\partial}{\partial\theta}u^\theta = \alpha g^\theta\nabla^2 u^\theta - 2\left[<\nabla_s I, \omega^\theta> + I_t\right] I_x \\ \\ \frac{\partial}{\partial\theta}v^\theta = \alpha g^\theta\nabla^2 v^\theta - 2\left[<\nabla_s I, \omega^\theta> + I_t\right] I_y \end{cases} \tag{2}$$

The parabolic PDEs in (2) define the problem that we will deal with in the next sections.

# 3 Numerical approach

According to [3], we use a first order forward finite differences scheme for the discretization of the first space-time derivatives, and a second order central finite differences scheme for the discretization of the second space derivative.
We have developed the following matrix formulation of the discretization of the PDE system in (2) on the whole $N \times M$ frame

$$\begin{cases} U_\theta^\theta = \alpha\left[G^\theta \cdot H_{(x,y)^2}\right] U^\theta - 2\left(I_x \cdot U^\theta + I_y \cdot V^\theta + I_t\right)\cdot I_x \\ \\ V_\theta^\theta = \alpha\left[G^\theta \cdot H_{(x,y)^2}\right] V^\theta - 2\left(I_x \cdot U^\theta + I_y \cdot V^\theta + I_t\right)\cdot I_y \end{cases} \tag{3}$$

where:

- $\cdot$ is the elementwise matrix multiplication;
- $U_\theta^\theta$ and $V_\theta^\theta$ are the column vectors of length $N \times M$ containing the first order partial derivatives of the optical flow components with respect to the evolution parameter $\theta$ (arranged in row-major order);
- $U^\theta$ and $V^\theta$ are the column vectors of length $N \times M$ containing the modulus of the horizontal and vertical optical flow components (arranged in row-major order) at the scale value $\theta$, respectively;
- $I_x, I_y, I_t$ are the column vectors of length $N \times M$ containing the discretized space-time partial derivatives of the image matrix $I$ (arranged in row-major order);
- $H_{(x,y)^2}$ is a penta-diagonal block matrix of size $(N\times M)\times(N\times M)$, with tridiagonal blocks along the main diagonal and diagonal blocks along the upper and lower sub-diagonals, arising from the discretization of the Laplacian.

- $G^\theta$ is the matrix of size $(N \times M) \times (N \times M)$ containing in each row the values $g^\theta$ of the diffusivity function, for every pixel of the image matrix $I$, at the scale value $\theta$.

System (3) is a semi-discretization of (2), i. e. the PDE problem is discretized only with respect to the spatial variables, while the scale variable is still continuous.
By using a forward finite differences scheme to discretize the scale derivative, three iterative schemes can be obtained: *explicit*, *semi-implicit* and *fully-implicit*. An analysis of the convergenge and accuracy order of those schemes in our setting is provided in [8]. Here, we use a *semi-implicit* scheme, since it is invariant to many transformations, such as grey level shift, translation, rotation, etc., and enjoys consistency, convergency and stability properties [4, 20].
Now, we consider a uniform grid of values of the scale variable $\theta$, with step-size $h_\theta$. In the sequel, we shall still denote by $\theta$ a generic discrete value of the scale grid. Therefore, at each discrete scale-step $\theta$, a new approximation of the optical flow components $(U^\theta, V^\theta)$ can be obtained by means of the previous approximation $(U^{\theta-h_\theta}, V^{\theta-h_\theta})$ and the following two linear systems of equations

$$\begin{cases} \left[I_d - h_\theta H^{\theta-h_\theta}\right] U^\theta = U^{\theta-h_\theta} - 2\left(I_x \cdot U^{\theta-h_\theta} + I_y \cdot V^{\theta-h_\theta} + I_t\right) \cdot I_x \\ \left[I_d - h_\theta H^{\theta-h_\theta}\right] V^\theta = V^{\theta-h_\theta} - 2\left(I_x \cdot U^{\theta-h_\theta} + I_y \cdot V^{\theta-h_\theta} + I_t\right) \cdot I_y \end{cases} \tag{4}$$

where $I_d$ is the identity matrix and $H^{\theta-h_\theta} = \alpha\left[G^{\theta-h_\theta} \cdot H_{(x,y)^2}\right]$.
In other words, at every scale-step two linear systems of order $N \times M$ must be solved: in the first one, the unknown is $U^\theta$, i.e. the vector of length $N \times M$ containing the horizontal components of optical flow (arranged in row-major order); in the second one, the unknown is $V^\theta$, i.e. the vector of length $N \times M$ containing the vertical components of optical flow (arranged in row-major order).
Notice that the two linear systems in (4) have the same coefficient matrix $A^{\theta-h_\theta} = I_d - h_\theta H^{\theta-h_\theta}$, whose structure is similar to the structure of the matrix $H^\theta$ (sparse penta-diagonal block matrix), while the right-hand terms are the vectors

$$\begin{aligned} b_x^{\theta-h_\theta} &= U^{\theta-h_\theta} - 2\left(I_x \cdot U^{\theta-h_\theta} + I_y \cdot V^{\theta-h_\theta} + I_t\right) \cdot I_x \\ b_y^{\theta-h_\theta} &= V^{\theta-h_\theta} - 2\left(I_x \cdot U^{\theta-h_\theta} + I_y \cdot V^{\theta-h_\theta} + I_t\right) \cdot I_y. \end{aligned} \tag{5}$$

# 4 Computational kernels

We briefly outline the numerical algorithm for solving the PDE system in (2). At each scale-step $\theta$ a new approximation of the optical flow is obtained by solving two linear systems with the same large, sparse, structured coefficient matrix. More precisely, at each scale-step, the matrix and the right-hand vectors are first updated, using the values computed at the previous step, and then a new approximation $(U^\theta, V^\theta)$ is computed, as shown in Algorithm 1.
Setting a threshold value $\tau_\theta$ for the required accuracy and a maximum number of iterations $n_\theta$, the iterative process ends up when the stopping criterion in stage **2** of Algorithm 1 is satisfied. This gives an approximation of the optical flow field at each pixel of the frame, at a certain time $t$. The linear systems in (4) are solved, at stages **6, 7** of Algorithm 1, by means of the GMRES iterative method. GMRES is a Krylov subspace method for non-symmetric linear systems, which is very effective when the matrix is sparse and large [16]. In order to accelerate the convergence, we use the Jacobi preconditioner [16].
Notice that the resulting algorithm for solving the optical flow problem is based on two iterative nested schemes. The outer iteration allows the evolution of the diffusion-reaction operator

**Algorithm 1** Iterative method for the solution of the flow-driver diffusion equation

1: initial condition: $(U^0, V^0) = (\underline{0}, \underline{0})$;
2: **while** $(D \geq \tau_\theta$ *.or.* $\theta \leq h_\theta n_\theta)$ **do**
3: computation of $G^{\theta - h_\theta}$ and $H^{\theta - h_\theta}$;
4: estimation of $A^{\theta - h_\theta} := [I_d - h_\theta H^{\theta - h_\theta}]$;
5: estimation of $b_x^{\theta - h_\theta}$ $b_y^{\theta - h_\theta}$ as in (5);
6: **solution of the first linear system:**
$A^{\theta - h_\theta} U^\theta = b_x^{\theta - h_\theta}$;
7: **solution of the second linear system:**
$A^{\theta - h_\theta} V^\theta = b_y^{\theta - h_\theta}$;
8: $D = ||(U^\theta, V^\theta) - (U^{\theta - n_\theta}, V^{\theta - h_\theta})||$;
9: $\theta := \theta + h_\theta$;
10: **end while**

associated to the method, with respect to the scale parameter $\theta$; at each outer iteration, a new pair of linear systems of equation is built and solved. The internal scheme computes the solution of such systems and consists of the iterations of the Jacobi-preconditioned GMRES method.

**Remark.** *Let $T$ be the number of frames of an images sequence and $N \times M$ be the size of a frame. The computational cost of the algorithm, using the CSR (Compressed Sparse Row) format to store the system sparse matrix $A^{\theta - h_\theta}$, is*

$$O((T-1) n_\theta \cdot n_K \cdot 5(N \times M)),$$

*where $n_\theta$ is the number of iterations of the outer scheme and $n_K$ is the number of iterations of the internal scheme.*

# 5 The parallel framework

The algorithm described in the previous section is very expensive when $T$, $N$ and $M$ are large. Therefore, a standard implementation often turns out to be unsatisfactory, especially for real-time computations. In order to obtain high performance, we need to exploit the hierarchical parallelism of novel architectures such as multi-core, cluster of multi-processors and GPUs.
Here, we consider a GPU cluster, i.e. a computer cluster in which each node is equipped with a GPU. While GPU computing is quite easy to approach, the use of a GPU cluster is still a challenge, since we lack an efficient, general programming model for them. However, GPU clusters can provide high peak performance at small cost, and so their importance and spreading are currently increasing among the scientific sotware community.
In our software, GPU programming relies on CUDA, the parallel computing platform and programming model of NVIDIA [22], while communications among the cluster nodes are based on the standard Message Passing Interface (MPI) [22].
The basic idea of our parallel software for implementing Algorithm 1 is to follow a domain decomposition approach for the outer iteration task of the algorithm, while the linear algebra of the internal scheme (stage **6, 7** of Algorithm 1) is carried out by the GPUs. The latter task has been accomplished by using the PetSc library. To be specific, we use the `KSP`, a package of solvers for linear systems, which includes direct/iterative methods and several preconditioners.

Recently, GPU support has been added to PetSc. Such GPU-enabled version of PetSc is still under development and its performance is promising. It relies on the open source libraries CUSP, a library for sparse linear algebra and graph computations on CUDA [2], and THRUST, a collection of data parallel primitives that provide high level abstractions to describe efficient computations on GPU [11].

Our parallel software is organized as a framework that allows three different options of execution. Each option implements a specific parallel strategy, namely the *intra-node* strategy, aimed at multiprocessor systems without GPU acceleration, the *inter-node* strategy, for the execution on a single node with a GPU, and the *multilevel-node* strategy, which is targeted towards general GPU clusters.
These parallelization strategies, which have been implemented as three different codes, are briefly described below.

- **INTRA-NODE strategy**. It relies on a coarse-grained parallelism, which exploits the intra-node concurrency of multiprocessors in order to parallelize stage **3** - **7** of Algorithm 1. As shown in Figure 1 (left), the intra-node code is executed on $N$ concurrent nodes. A suitable domain decomposition splits the frame into sub-frames that are distributed to the nodes. The data distribution is carried out by a direct *scattering-gathering*, with MPI functions. More precisely, at each scale-step, $G^{\theta-h_\theta}$ and $H^{\theta-h_\theta}$ are scattered to all nodes; therefore each process solves the assigned linear (sub)system on its own multicore CPU using the `KSPGMRES` solver with the `PCJACOBI` option of the standard version of PetSc. Finally, the results are gathered in $U^\theta$ and $V^\theta$.

- **INTER-NODE strategy**. It is based on a fine-grained parallelism aimed at exploiting concurrency at GPU level. This strategy uses the GPU-enabled version of PetSc for solving the linear systems on the GPU. As shown in Figure 1 (right), the inter-node code is executed on a single node with a GPU. In this way, we compute the solutions $U^\theta$ and $V^\theta$, at each scale-step, without distributing data via MPI, yet exploiting the massive parallelism of the GPU inside a single node.

- **MULTILEVEL-NODE strategy**. It is the main contribution of this work. It combines the previous two strategies, as shown in Figure 2. First, a preliminary domain decomposition, as in the INTRA-NODE strategy, is performed. Then, each node uses its own GPU to solve the assigned linear (sub)system by means of the `KSPGMRES` solver with the `PCJACOBI` option of the GPU-enabled version of PetSC, as in the INTER-NODE strategy.

We observe that in the multi-level parallel code the exchange of data among nodes occurs only at the initial and final steps of the outer loop, and corresponds to the distribution and gathering tasks of the frame decomposition. Conversely, all communications required by the linear solver occur as host-device memory data exchange in a single node.

# 6 Experimental results

Our framework, with its three different parallel codes, has been tested on real images arising from the tracking of moving clouds in SAR image sequences. Here, we briefly report the results of some tests in order to highlight the improvement provided by the three ways of execution. The experiments are carried out on a cluster of eight nodes, each equipped with a Quadro K5000 GPU and an Intel Core i7 CPU (2.8GH, 8GB Cache). The image sequence is composed of 10 frames, which correspond to a map of the weather status on a time interval of 15 minutes. The

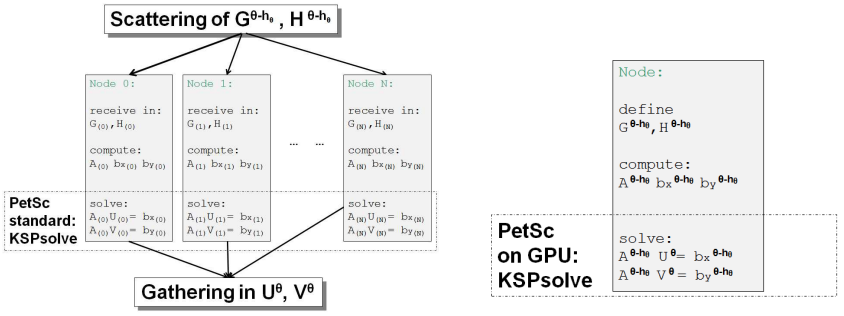


Figure 1: Left: the INTRA-NODE code. Right: the INTER-NODE code.

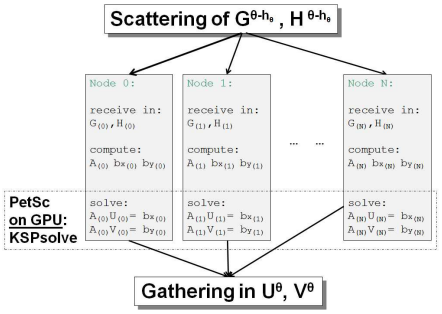


Figure 2: The MULTILEVEL-NODE code.

analysis of longer sequences becomes useless for practical purposes, because at longer times the clouds could change their shape so that the estimation of the optical flow would be altered. We considered short sequences, with little change in brightness, where each image has size $555 \times 845$. Some results are shown in Figure 3. Notice that the computed optical flow reliably follows the course of the clouds. The accuracy of the differential methods for the optical flow has been widely addressed in [19]. In order to assess the reliability of our software, we performed the same accuracy test as in [19] obtaining results of comparable accuracy.
We are specifically interested in studying the performance of our parallel software in terms of execution time. The execution times for the three strategies are shown below. First, we have compared the sequential version to the multicore version on one node of our cluster. A trivial execution of the sequential version gives an execution time equal to $1554s$, i.e. nearly 25 minutes. Since each node is multicore, a clever setting of the compiler options, namely the use of Pthreads (a set of C language programming types and functions for an efficient execution on a multi-core architecture) reduces the execution time to $259s$. The latter execution time will be taken as reference benchmark for the performance of the code on a single node.

In Table 1, we report the execution times of the INTRA-NODE code. It is apparent that, as the number of nodes increases, we do not achieve a significant gain. This is due to the fact that on a cluster of multiple nodes a large amount of data communication is required, especially when solving the system of linear equations concurrently among nodes. We observe that the best speedup is obtained when only two multicore nodes are used, with a gain of 49.4%.
In Table 2, we show the execution times of the INTER-NODE and the MULTILEVEL-NODE

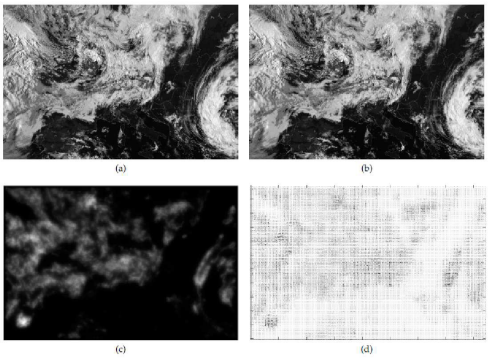

Figure 3: Optical Flow estimation for a couple of frames (subfigures (a) and (b)). Double visualization of results (subfigures (c) and (d)).

| hw conf | execution times | speedup |
|---|---|---|
| 1 multicore-CPU | 259s | 1 |
| 2 multicore-CPU | 131s | 1.98 |
| 4 multicore-CPU | 88s | 2.94 |
| 8 multicore-CPU | 60s | 4.31 |

Table 1: Execution times (in seconds) for the INTRA-NODE software: each node is an Intel Core i7 CPU

codes. Notice that the first row of Table 2 refers to the INTER-NODE software.

| hw conf | execution times | speedup |
|---|---|---|
| 1 multicore-CPU + GPU | 50s | 5.18 |
| 2 multicore-CPU + GPU | 29s | 8.9 |
| 4 multicore-CPU + GPU | 20s | 12.95 |
| 8 multicore-CPU + GPU | 13s | 19.93 |

Table 2: Execution times (in seconds) for the intra-node and multilevel-node codes: each node is an Intel Core i7 CPU equipped with a Quadro K5000 GPU

The execution time of the INTER-NODE code is $50s$, which entails a gain of 80.6% with respect to the execution time obtained on a single multicore node. As long as the MULTILEVEL-NODE code is concerned, we note that, even though a large amount of data must be exchanged for the frame decomposition among nodes, the most computationally expensive portion of the software is executed locally on the GPUs, so that a higher gain up to nearly 95% is obtained. Moreover, the use of an optimized library, such as the GPU-enabled PetSc, minimizes the exchange of data between host and device memories and fully exploits the intrinsic parallelism of the GPUs.
Finally, the behavior of the speedup, for an increasing number of nodes and GPUs, is shown in Figure 4. Notice that the speedup of the INTRA-NODE code grows slowly since the communication overhead raises as the number of nodes increases, especially for the linear solvers. Conversely, the speedup of the MULTILEVEL-NODE code grows more rapidly, due to the

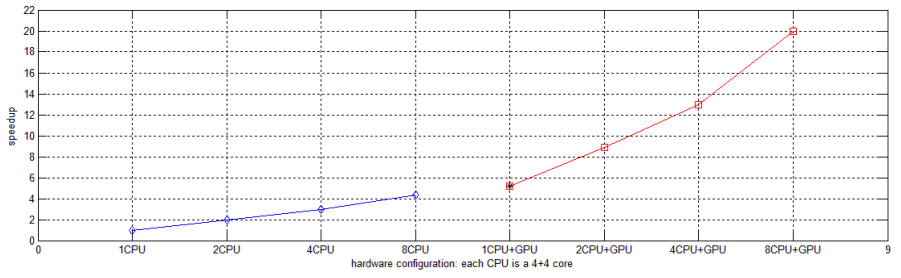


Figure 4: Speedup curves: blue curve with diamond points is the speedup of the intra-node code; red curve with squared points is the speedup of the multilevel-node code; the star point is the speedup of the inter-node code.

limited contribution of the intra-node communication for the frame decomposition, and to a more effective exploitation of the degree of parallelism at GPU level.

# 7 Conclusions

In order to benefit from the multilevel parallelism that characterizes the current architecture of the GPU-accelerated multiprocessor clusters, we have developed a parallel software framework which implements a differential method for computing the optical flow in image sequences. The computational kernel of the resulting numerical problem consists of two iterative nested schemes and has been solved by means of a parallelization strategy that combines a coarse parallelism for the outer scheme, which accounts for most of the data communication among nodes, and a fine parallelism for the inner scheme, which exploits the high computational power of the GPUs in performing basic linear algebra operations. Numerical experiments confirmed that our parallel software is able to leverage the GPU power while avoiding both space limitation issue and communication overhead among nodes, and allowed to estimate that the resulting computational gain is nearly equal to 95% with respect to the sequential version of the software.

**Acknowledgements** This work has been partially supported by the Italian PON project *SIRIO: Servizi per lInfrastruttura di Rete wIreless Oltre il 3G.*

# References

[1] J. Barron, D. J. Fleet, S. S. Bacuemin - *Performance of Optical Flow tecniques.* International Journal of Computer Vision. Volume 12(1), pp: 43-77, February 1994.

[2] N. Bell , M. Garland - *Cusp: Generic parallel algorithms for sparse matrix and graph computations.* Version 0.3.0, 2012.

[3] D. Casaburi, L. D'Amore, A. Galletti, L. Marcellino - *A numerical algorithm for image sequence inpainting that preserves fine textures.* International Journal of Computer Mathematics. Volume 88, Issue 11, pp: 2331-2347, 2011.

[4] D. Casaburi, L. D' Amore, L. Marcellino, A. Murli - *A Motion-aided Ultrasound Image Sequence Segmentation.* Proceeding of ENUMATH2009 - Numerical Mathematics and Advanced Applications. Part 2, pp: 217-225, 2010.

[5] D. Casaburi, L. D'Amore, L. Marcellino, A. Murli - *Real time ultrasound image sequence segmentation on multicores*. Proceeding of PARCO2009 - Parallel Computing: From Multicores and GPU's to Petascale, vol 19, pp: 185-192, 2010.

[6] S. Cuomo, A. Galletti, G. Giunta, A. Starace, A. - *Surface reconstruction from scattered point via RBF interpolation on GPU*. Federated Conference on Computer Science and Information Systems, FedCSIS 2013, pp. 433-440, 2013.

[7] S. Cuomo, P. De Michele, F. Piccialli - *3D data denoising via Nonlocal means filter by using parallel GPU strategies*. Computational and Mathematical Methods in Medicine, 2014 .

[8] L. D' Amore, D. Casaburi, L. Marcellino, A. Murli - *Numerical Solution of Diffusion Models in Biomedical Imaging on Multicore Processor*. International Journal of Biomedical Imaging: Special Issue on Parallel Computation in Medical Imaging Applications. Hindawi Publishing Corporation, 2011.

[9] A. Dopico, M. Correia, J. Santos, and L. Nunes - *Parallel Computation of Optical Flow*, ICIAR 2004, LNCS 3212, 2004.

[10] R. Farina, S. Cuomo, P. De Michele, F. Piccialli - *A smart GPU implementation of an elliptic kernel for an ocean global circulation model*. Applied Mathematical Sciences, 7 (61-64), pp. 3007-3021, 2013.

[11] J. Hoberock, N. Bell - *Thrust: A parallel template library*. Version 1.5.2, 2012.

[12] B. K. P. Horn, B. G. Schunck - *Determining optical Flow*. Artif. Intell., vol. 17, pp. 185203, 1981.

[13] P. Knabner, L. Angermann - *Numerical Methods for Elliptic and Parabolic Partial Differential Equations*. Texts in Applied Mathematics, Vol. 44. Springer, 2003.

[14] T. Kohlberger, C. Schnorr, A. Bruhn, and J. Weickert - *Parallel Variational Motion Estimation by Domain Decomposition and Cluster Computing*. ECCV2004, LNCS 3024, pp: 205-216, 2004.

[15] P. Montero, J. Taibo - *Fast GPU approximation of EPZS motion estimation using branching*. Multimedia Signal Processing (MMSP), pp.356,361, 2013.

[16] Y. Saad, M. Schultz - *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*. SIAM J. Sci. Statist. Comput. Vol 7, pp. 856-869, 1996.

[17] X. Wang, L. Song, M. Chen, J. Yang - *Paralleling variable block size motion estimation of HEVC on CPU plus GPU platform*. Multimedia and Expo Workshops (ICMEW) IEEE International Conference, 2013.

[18] J. Weickert , A. Bruhn , N. Papenberg, T. Brox - *A Survey on Variational Optic Flow Methods for Small Displacements*. Mathematical Models for Registration and Applications to Medical Imaging Mathematics in industry Volume 10, pp: 103-136, 2006.

[19] J. Weickert, C. Shnorr - *A Theoretical Framework for Convex Regularizeds in PDE-Based Computation of Image Motion* TR 13/2000, Computer Science Series, 2000

[20] J.Weickert - - *Recursive separable schemes for nonlinear diffusion filters*. Scale-Space Theory in Computer Vision, Lecture Notes in Computer Science, Springer, Berlin, Vol. 1252, , pp. 260-271, 1997.

[21] http://www.mcs.anl.gov/petsc/

[22] http://www.nvidia.com/

[23] http://www.sat24.com/

[24] http://www.mcs.anl.gov/research/projects/mpi/