Computers and Chemical Engineering 55 (2013) 61–70

Contents lists available at SciVerse ScienceDirect

Computers and Chemical Engineering

journal homepage: www.elsevier.com/locate/compchemeng

Solution of the population balance equation using parallel adaptive cubature on GPUs☆

Fabio P. Santos[a], Inanc Senocak[b], Jovani L. Favero[a], Paulo L.C. Lage[a,*]

[a] Programa de Engenharia Química, COPPE, Universidade Federal do Rio de Janeiro, PO Box 68502, Rio de Janeiro, RJ 21941-972, Brazil[1]
[b] Department of Mechanical and Biomedical Engineering, Boise State University, Boise, ID 83725, USA

ARTICLE INFO

Article history:
Received 28 December 2012
Received in revised form 29 March 2013
Accepted 4 April 2013
Available online 17 April 2013

Keywords:
Population balance modeling
Quadrature Method of Generalized Moments
Particulate systems
Adaptive cubature
GPU

ABSTRACT

The Dual Quadrature Method of Generalized Moments (DuQMoGeM) is an accurate moment method for solving the population balance equation (PBE). The drawback of DuQMoGeM is the high computational cost associated with numerical integrations of the PBE integral terms in which each integrand can be integrated independently and, therefore, amenable to parallelization on GPUs. In this work, two parallel adaptive cubature algorithms were implemented on a hybrid architecture (CPU–GPU) to accelerate the DuQMoGeM. The speedup and scalability of these parallel algorithms were studied with different types of Genz's test functions. Then, we applied these parallel numerical integration algorithms in the DuQMoGeM solution of the PBE for three bivariate cases, obtaining speedups between 11 and 15.

1. Introduction

The modeling and simulation of polydispersed multiphase flow is critically important, because they are present in numerous natural and industrial processes. The precipitation reactor is a good example of an industrial application in which a solid phase with specific features is crystallized from a liquid phase. The final market value of the crystallized product depends on its particle size distribution. Another example is the bubble column chemical reactor whose efficiency depends on the interfacial area density which is determined from the bubble size distribution (Yeoh & Tu, 2010).

A mesoscale framework called population balance equation (PBE) combined with Eulerian multifluid flow formulation (Silva & Lage, 2011) is a convenient way to model the particle dynamics in the above mentioned chemical engineering applications. But development of robust and accurate methods for solving the PBE (Attarakih, Drumm, & Bart, 2009; Bove, Solberg, & Hjertager, 2005; Fox, Laurent, & Massot, 2008; Kumar & Ramkrishna, 1996; Lage, 2011; Massot, Laurent, Kah, & Chaisemartin, 2010; Strumendo & Arastoopour, 2008; Yuan, Laurent, & Fox, 2012) has been proved to be challenging and remains to be an active research area.

Among the existing methods, the quadrature-based moment methods (QBMM) and their variants are suitable to be coupled to CFD models (Silva & Lage, 2011). This characteristic originates from the fact that these methods provide a discretization in the internal variable that yields n particles phases in an Eulerian multifluid approach for simulating polydispersed multiphase flows. The first QBMM was the Quadrature Method of Moments (QMoM) (McGraw, 1997). QMoM solves for the first $2n$ moments of the particle number density function (NDF) using the n-point Gauss–Christoffel quadrature formula, obtained by the product difference (PD) algorithm (Gordon, 1968), to overcome the so-called closure problem. Later, Marchisio and Fox (2005) proposed the Direct Quadrature Method of Moments (DQMoM) as an alternative to the QMoM. Instead of calculating the NDF moments, Marchisio and Fox (2005) evaluated directly the weights and abscissas of the n-point Gauss–Christoffel quadrature. Fundamentally, DQMoM does not need to compute the Gauss–Christoffel quadrature except for the discretization of the NDF at the initial or boundary conditions.

However, DQMoM and QMoM have limitations. The n-point Gauss–Christoffel calculation is an ill-conditioned problem that limits the number of quadrature points that can be used (Gautschi, 2004). Since a small number of quadrature points may not be enough to accurately calculate the integral source terms of the PBE moments, these methods accumulate errors due to the quadrature approximation of these terms that may eventually degenerate the solution.

In order to overcome the error accumulation problem, Lage (2011) introduced the Dual Quadrature Method of Generalized

☆ This article is dedicated to Professor Alberto Luiz Coimbra, in the 50th anniversary of COPPE (1963–2013), the Graduate School of Engineering of the Federal University of Rio de Janeiro.

* Corresponding author. Tel.: +55 21 2562 8346; fax: +55 21 2562 8300.
E-mail address: paulo@peq.coppe.ufrj.br (P.L.C. Lage).

[1] http://www.peq.coppe.ufrj.br/pesquisa/tfd.

0098-1354/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.compchemeng.2013.04.012

Nomenclature

a	aggregation frequency
A	aggregation matrix
B	number of blocks per grid
b	breakage frequency
c	coefficient in polynomial approximation of f
f	number density distribution function
H	Heaviside step function
L	breakage matrix
M	number of cubature points
N	number of integrands
n	moment order in PBE solution
P	probability density function of particle daughters
T	number of threads per blocks
t	time
x	first particle property
y	second particle property

Greek letters

δ	Dirac delta function
ε	tolerance
$d\lambda$	measure with a positive definite inner product
μ_k	moment of k order
ϑ	number of particle formed in breakage
ω	weight function
ϕ_k	k order polynomial
Π	moment of k order of P

Subscripts

a	aggregation
b	breakage
max	related to maximum value

Superscripts

ϕ	generalized moment
a	exact solution of standard moment
abs	related to absolute tolerance
rel	related to relative tolerance

Moments (DuQMoGeM). In this method, a quadrature rule of high accuracy can be used to compute the source terms of the PBE, while the NDF is still discretized by the Gauss–Christoffel quadrature formula, having also an approximate polynomial expansion. Lage (2011) showed that, if the PBE kernels do not belong to any polynomial space, a fixed point quadrature cannot be enough to guarantee the accuracy of the method.

Afterwards, Yuan et al. (2012) developed the Extended QMoM (EQMoM), which is also a dual quadrature method. The advantage of this method is that it approximates the NDF using non-negative functions, the kernel density functions (KDFs), which guarantees the positivity of the NDF. The EQMoM also uses the Gauss–Christoffel quadrature based on the first $2n$ moments for the KDF definition, which is completed by minimizing the difference between the values of the $2n$-order moment calculated from the PBE moment and from the Gauss–Christoffel quadrature formula.

Favero and Lage (2012) used an adaptive cubature (Johnson, 2008) to compute the integral terms of the DuQMoGeM, which was able to control the quadrature error within a specified tolerance. They also extended the DuQMoGeM to bivariate population balance problems also using the adaptive cubature to solve homogeneous bivariate problems with error control. However, the computational time using the adaptive cubature can be quite large, make it extremely expensive for CFD applications. But recent developments in programmable GPUs promise to broader the adoption of this technique.

Graphics processing units (GPU) were initially developed to accelerate specialized graphics applications. GPUs have emerged as a massively parallel co-processor to CPU for high-performance scientific computing. GPUs have become a new paradigm in the supercomputing field due to its computational power, low cost and high performance per watt (Kirk & Hwu, 2010).

In order to exploit GPU power, we implemented an adaptive cubature method in two different parallel algorithms in GPUs in order to accelerate the DuQMoGeM solution. These algorithms differ in the level of the task and data parallelism. One is parallelized in the integrand level and other in the integral formula level (Schurer, 2001). The parallel adaptive cubature algorithms were extensively tested with different types of Genz's test functions (Genz & Malik, 1983) and these algorithms were applied for solving three bivariate PBE problems.

2. Population balance modeling, PBM

The PBE is the conservation equation for the number of particles, represented by the mean number density distribution function $f(x, y, t)$, which is a function of particle properties, space and time (Ramkrishna, 2000). Here we consider the bivariate PBE for a homogeneous problem written in terms of additive particle properties, (x, y):

$$\frac{\partial f(x,y,t)}{\partial t} + \mathcal{L}_a f(x,y,t) + \mathcal{L}_b f(x,y,t) = S(x,y,t), \tag{1}$$

where $S(x, y, t)$ is a generic source term and the aggregation and breakage operators are defined by:

$$\begin{aligned}\mathcal{L}_a f(x,y,t) = &\int_0^{x_{max}}\int_0^{y_{max}} a(x,x',y,y',t) f(x,y,t) f(x',y',t)\,dx'\,dy' \\ &- \frac{1}{2}\int_0^{x}\int_0^{y} a(x-x',x',y-y',y',t) f(x-x',y-y',t) \\ &\times f(x',y',t)\,dx'\,dy',\end{aligned} \tag{2}$$

$$\begin{aligned}\mathcal{L}_b f(x,y,t) = b(x,y,t) f(x,y,t) - &\int_x^{x_{max}}\int_y^{y_{max}} \vartheta(x',y',t) b(x',y',t) \\ &\times P(x,y \mid x',y',t) f(x',y',t)\,dx'\,dy',\end{aligned} \tag{3}$$

where $b(x, y)$ is the breakage frequency, $\vartheta(x', y', t)$ is the number of particles originated by the breakage of a particle with property x' and y', $P(x, x|x', y')$ is their probability distribution function and $a(x, x', y, y')$ is the aggregation frequency (Ramkrishna, 2000).

The daughter probability distribution function has the following properties:

$$\begin{aligned}&\int_0^{x'}\int_0^{y'} P(x,y \mid x',y',t)\,dx = 1; \\ &P(x,y \mid x',y',t) = 0, \quad \forall x > x' \text{or} \forall y > y'; \\ &\int_0^{x'} xP(x,y \mid x',y',t)\,dx = \frac{x'}{\vartheta(x',y',t)}; \\ &\int_0^{x'} yP(x,y \mid x',y',t)\,dy = \frac{y'}{\vartheta(x',y',t)}.\end{aligned} \tag{4}$$

2.1. Dual Quadrature Method of Generalized Moments, DuQMoGeM

In DuQMoGeM, the NDF may be represented by an expansion using a convenient orthogonal polynomial basis (Favero & Lage, 2012):

$$f(x,y,t) = \omega(x)\tilde{\omega}(y)\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} c_{ij}\phi_i(x)\tilde{\phi}_j(y), \tag{5}$$

where c_{ij} are the coefficients of the expansion, $\phi_i(x)$ and $\tilde{\phi}_j(y)$ are the orthogonal polynomial bases in both variables and $\omega(x)$ and $\tilde{\omega}(y)$ are the corresponding weight functions of the inner products that define the orthogonality property of these polynomials:

$$\langle\phi_j,\phi_i\rangle_{d\bar{\lambda}(x)} = \int_0^{x_{max}} \phi_j(x)\phi_i(x)\omega(x)\,dx = \delta_{ij}\|\phi_i\|^2_{d\bar{\lambda}(x)}, \tag{6}$$

$$\langle\tilde{\phi}_j,\tilde{\phi}_i\rangle_{d\tilde{\lambda}(y)} = \int_0^{y_{max}} \tilde{\phi}_j(y)\tilde{\phi}_i(y)\tilde{\omega}(y)\,dy = \delta_{ij}\|\tilde{\phi}_i\|^2_{d\tilde{\lambda}(y)}, \tag{7}$$

and the corresponding measures $d\bar{\lambda}(x) = w(x)dx$ and $d\tilde{\lambda}(y) = \tilde{w}(y)dy$.

The coefficients c_{ij} can be obtained by the orthogonality properties as

$$c_{ij} = \frac{\mu_{ij}^{(\phi,\tilde{\phi})}}{\|\phi_i\|^2_{d\bar{\lambda}(x)}\|\tilde{\phi}_j\|^2_{d\tilde{\lambda}(y)}}, \tag{8}$$

where the generalized mixed moment is given by

$$\mu_{ij}^{(\phi,\tilde{\phi})} = \int_0^{x_{max}}\int_0^{y_{max}} \phi_i(x)\tilde{\phi}_j(y)f(x,y,t)\,dx\,dy. \tag{9}$$

The substitution of the approximation given by Eq. (5) into Eq. (1) gives (Favero & Lage, 2012):

$$\|\phi_k\|^2_{d\bar{\lambda}(x)}\|\tilde{\phi}_l\|^2_{d\tilde{\lambda}(y)}\frac{dc_{kl}}{dt} + \sum_{i,j,p,q=0}^{n-1} A_{klijpq}c_{ij}c_{pq} + \sum_{i,j=0}^{n-1} L_{klij}c_{ij} = S_{kl}, \tag{10}$$

where

$$\begin{aligned} A_{klijpq} = &\int_0^{x_{max}}\int_0^{x_{max}}\int_0^{y_{max}}\int_0^{y_{max}} \\ &\times \left[\phi_k(x)\tilde{\phi}_l(y) - \frac{1}{2}\phi_k(x+x')\tilde{\phi}_l(y+y')\right] \\ &\times a(x,x',y,y')\omega(x)\tilde{\omega}(y)\phi_i(x)\tilde{\phi}_j(y)\omega(x)\tilde{\omega}(y) \\ &\times \phi_p(x)\tilde{\phi}_q(y)\,dx\,dx'\,dy\,dy', \end{aligned} \tag{11}$$

$$\begin{aligned} L_{klij} = &\int_0^{x_{max}}\int_0^{y_{max}} b(x,y)[\phi_k(x)\tilde{\phi}_l(y) - \vartheta(x,y,t)\Pi_{kl}^{\phi}(x,y)] \\ &\times \omega(x)\tilde{\omega}(y)\phi_i(x)\tilde{\phi}_j(y)\,dx\,dy, \end{aligned} \tag{12}$$

$$\Pi_{kl}^{\phi}(x,y) = \int_0^{x}\int_0^{y} \phi_k(x')\tilde{\phi}_l(y')P(x,y\mid x',y',t)\,dx'\,dy', \tag{13}$$

$$S_{kl} = \int_0^{x_{max}}\int_0^{y_{max}} \phi_k(x)\tilde{\phi}_l(y)S(x,y)\,dx\,dy. \tag{14}$$

The accuracy of this method depends on the terms A_{klijpq} and L_{klij}, $k, l, i, j, p, q = 0\ldots n-1$, which can be calculated by an adaptive cubature. It is important to point out that the calculation of these terms is the bottleneck of the DuQMoGeM. Therefore, these terms were computed using the parallel adaptive cubature algorithms implemented on the CPU–GPU heterogeneous architecture in the present study. These integral terms are good candidates for acceleration on GPUs because they are highly parallelizable.

3. Basics of GPU and CUDA

The GPU is a dedicated and specialized device which was originally designed to accelerate graphics rendering. Presently, the GPU is not only being used for graphics applications, but also to accelerate scientific computation (Sanders & Kandrot, 2010).

The CPUs are optimized for sequential code performance and require a sophisticated logic control, while GPUs are highly parallel computing devices composed of hundreds of cores which dedicate more of their resources to computation. This difference of architecture provides a significant disparity between the computational power of the GPUs and CPUs. The GPUs are co-processors that have many scalar processors (SP) distributed over multiple streaming multiprocessors (SM). GPUs programming model are based on the single-instruction multiple threads (SIMT), in which each computational thread executes the same instruction on numerous data elements simultaneously (NVIDIA Corporation, 2010).

The CUDA (compute unified device architecture) is the application programming interface (API) provided by NVIDIA that allows the development of parallel applications on its GPUs. The CUDA is implemented as an extension of C/C++or Fortran. In CUDA programming, there exist three main abstractions: the hierarchical thread, which is divided in grids, blocks and threads, the memory management and the synchronization of threads (NVIDIA Corporation, 2010). A CUDA kernel is a parallel portion of an application that is executed on the GPU, one at a time. Each CUDA kernel executes on one grid that is composed of several blocks, each of them formed by a set of threads. This grid-block-thread hierarchy provides the fine-grained data-level and thread-level parallelism carried out by the threads of each block combined with coarse-grained data-level and the task-level parallelism carried out by blocks. For instance, Frezzotti, Ghiroldi, and Gibelli (2011) presented an algorithm for solving a kinetic equation model onto GPUs, where each GPU block was responsible for one group of particles with a given velocity which was decomposed in a grid of threads, with each thread being associated with one cell of a physical space.

The threads within a thread-block can communicate through the shared memory which can be synchronized using barriers. It means that shared memory is only accessible by threads within a thread-block. Likewise, threads of different blocks cannot communicate directly through the shared memory. In addition to shared memory, there are other types of memory: global, texture and constant. These memory spaces are accessible for all threads, but cannot be used for communication among them (NVIDIA Corporation, 2010). In this way, the correct usage of these three main abstractions can ensure the performance of CUDA-enabled GPU applications.

4. Adaptive cubature algorithm

Multidimensional integrals are present in various branches of science. However, in many cases there is no analytical closed solution (Schurer, 2001) and, therefore, a numerical solution is required.

Let us assume the multidimensional integral to be written as

$$\mathbf{F} = \int_{C_s} \mathbf{h}(\mathbf{x})\,d\mathbf{x}, \tag{15}$$

where $\mathbf{h} : C_s \longrightarrow \Re^N$, being C_s a hypercube of dimension s. There are four important numerical integration methods. Among them

Adaptive cubature algorithm.

```
Put the initial regions into a collection of regions.
for all regions in the collection do
  Apply the cubature rules.
  Select the subdivision direction.
  Compute the error estimate.
end for
Compute the global error estimate.
while the tolerance is not achieved and the maximum number of iterations is
  not reached do
  Choose a region with the largest absolute error.
  Split this region into new regions.
  for all new regions do
    Apply the cubature rules.
    Select the subdivision direction.
    Compute the error estimate.
  end for
  Store this new regions in the collection of regions.
  Compute the global error estimate.
end while
```

are the Monte Carlo and Quasi-Monte Carlo, that are typically used for large N, the interpolatory cubature and the product of Gauss quadrature rules, being both more suitable for small N (Rudolf & Schrer, 2003). Therefore, the latter two methods are suitable for population balance models due to the low dimensionality of the existing integrals.

Mazzia and Pini (2010) compared the interpolatory cubature and the product of Gauss quadrature rules in the integration over a circle. They concluded that the interpolatory cubature can obtain the same accuracy of the product of Gaussian quadrature rules with only a few quadrature points for two dimensional integrals.

Fundamentally, an interpolatory cubature rule is defined in the following form (Stroud, 1971):

$$\mathbf{F}_M \approx \sum_{i=1}^{M} w_i \mathbf{h}(\mathbf{x}_i), \tag{16}$$

where M is the number of cubature points, $\mathbf{x}_i \in C_s$ are the M abscissas and w_i are the corresponding weights. As Eq. (16) gives only an approximation to the integral (Rudolf & Schrer, 2003), an adaptive algorithm can be used to achieve the desired accuracy.

The adaptive algorithm consists of applying the cubature rule to sub-regions of the integration domain, which are generated by subdivision accordingly to their integral error estimates. The integration error over a sub-region can be estimated by using two cubature rules with different orders. Note that a split direction should be selected every time a region is divided whose choice affects the convergence rate. Usually, the split direction is chosen to be the direction with the largest integrand variation in the region. The adaptive cubature algorithm, given below, stops when the global error, which is computed from the local errors, satisfies a desired tolerance or the maximum number of function evaluation is reached (Rudolf & Schrer, 2003).

In order to accelerate the automatic cubature, researchers have proposed several parallel adaptive cubature algorithms. According to Schurer (2001), there are six forms of parallelizing an adaptive cubature algorithm. These forms differ in terms of their levels of parallelism (Genz, 1989). Among these, the most common are the parallelism at the integral formula, subdivision and integrand levels. In the integral formula level, the basic formula is parallelized, and each processor or thread computes a subset of the cubature points. The parallelization at the integrand level means that one or more integrands are calculated by one processor or thread. In the subdivision level, each processor calculates a subdomain of the entire domain (Schurer, 2001).

The majority of the implemented algorithms are parallelized at the subdivision level (Bull & Freeman, 1995; D'Apuzzo, Lapegna, & Murli, 1997; Schurer, 2001), because this strategy is well-suited for multiple-instruction multiple-data architectures (MIMD). Therefore, the parallelization of the adaptive cubature on the GPU programming model, which is based on single-instruction multiple threads, is a novel and challenging task.

5. GPU implementation

There are several packages that implemented an adaptive cubature algorithm (Hahn, 2005; Johnson, 2008; Schurer, 2001). However, none of those takes advantage of CUDA-enabled GPU devices. Generally, the parallel adaptive cubature implementations in MIMD architectures are restricted to the parallelism at the subdivision level. However, in the present study we implemented this algorithm on GPUs using parallelism also at the integrand and integral formula levels. These parallelism levels were chosen because they are better suited for the single-instruction multiple threads programming model, because each integrand can be computed independently (Schurer, 2001).

In each region, the integral and its error were computed using the embedded cubature rules of fifth and seventh-order developed by Genz and Malik (1983). Likewise, the direction with the largest value of the fourth divided difference operator (FDDO) was selected for the region subdivision (Genz & Malik, 1983). It is important to note that this operator does not contribute appreciably to the computational cost because it uses the integrand values at the cubature points of the Genz and Malik (1983) rules.

The error control strategy was based on a measure of the integration error over the whole domain, which was given by the sum of the magnitudes of the estimated local errors in all subregions. Clearly, this is a conservative measure of the global error, because it does not allow the possible cancellation of positive and negative errors of different subregions. Given values for the relative, ε_{rel}, and absolute, ε_{abs}, tolerances were used to define a mixed tolerance given by:

$$\boldsymbol{\varepsilon} = \max\left(\varepsilon_{abs}, \varepsilon_{rel}\left|\mathbf{F}_M\right|\right), \tag{17}$$

where $\mathbf{F}_M$ is the integral over the entire domain. The adaptive algorithm stops when the maximum number of iterations is achieved or the measure of the global error is less than ε.

5.1. Parallelism at the integral formula level (PIFL)

At the integral formula level, we adopt both task- and data-level parallelism. A CUDA kernel is responsible for computing the integral of a vector of integrands over a sub-region. The blocks perform the task-level parallelism and the threads the data-level parallelism. It means that each block of a grid calculates the integral of one integrand of the vector of integrands, while each thread of this block computes one cubature point to obtain this integral. Fig. 1 shows the PIFL applied to a problem with N integrands and M cubature points in which there are at least M threads per block and N blocks. Furthermore, as FDDO uses the cubature points, each block also calculates the value of this operator in all directions for its integrand.

In order to overlap the computation of different sub-regions, the cubature points and the integral interval are copied from CPU to the GPU using asynchronous copy functions. This strategy is also used to copy the vector of integrals, the sub-region absolute errors and the FDDO values from GPU to GPU. This feature is particularly related to the Fermi architecture where a GPU can execute multiple kernels based on the resource availability.

For each integrand, the cubature points calculated by the threads must be added for obtaining the integral in each block. This is performed by the so-called parallel reduction algorithm that uses the shared memory for thread collaboration inside a block (Sanders &

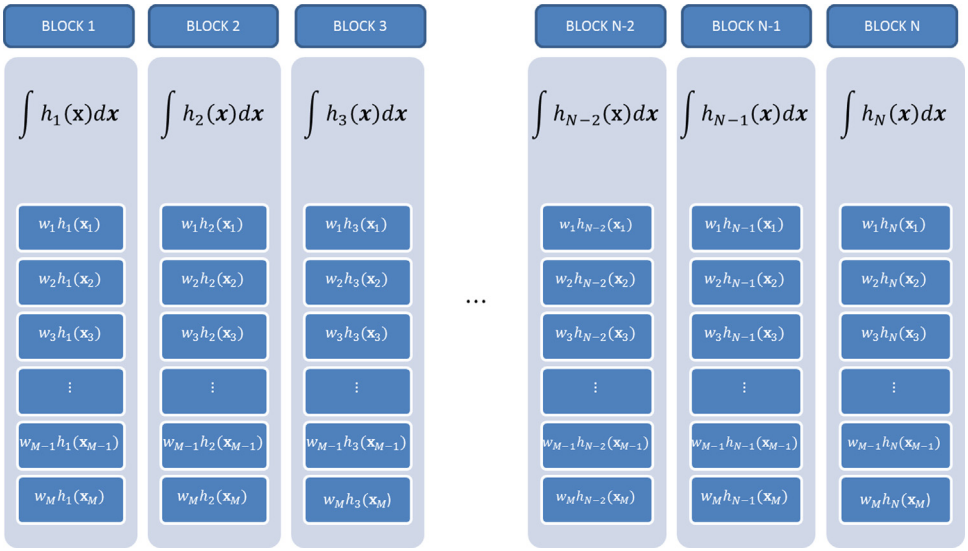

Fig. 1. Parallelism at the integral formula level (PIFL) with M cubature points and N integrands.

Kandrot, 2010). The main concept of this algorithm is that each thread add two values of the vector of function evaluations at the cubature points, $w_j h_i(\mathbf{x}_j)$, which are available in the shared memory, being the result saved back in the shared memory. This procedure is performed for $\log_2(M)$ steps, where M is the number of cubature points. After all steps, the final sum of the cubature points is stored at a variable with the corresponding block index.

For instance, Fig. 2 shows the parallel reduction algorithm for $M = 8$, where the cubature point calculated by *thread one* is added to the cubature point calculated by the *thread eight*, and the resulting value is stored in the shared memory with the index corresponding to thread one. This is carried out for all pair of threads with indexes j and $M - j + 1$ in this step. After 3 steps, the $\sum_{j=1}^{8} w_j h_1(\mathbf{x}_j)$ is stored in the shared memory, which is then copied to the global memory using the index of the corresponding block and integrand. The reader can find more details of the reduction algorithm in Sanders and Kandrot (2010).

5.2. Parallelism at the integrand level (PIL)

In this parallelism level, each CUDA thread calculates one integral of the vector of integrands. Each block has T integrands and, therefore, calculates the integrals of T integrands. Fig. 3 describes this algorithm. The FDDO is also calculated at the thread level. Again, the asynchronous copy is used to overlap the data copy and the computations. The cubature points and the intervals of integration are copied from the CPU to GPU and the resulting vectors of integrals, absolute errors and FDDO values are copied from GPU to GPU. In order to be efficient, this parallel algorithm requires a large number of integrands, which is typical of the solution of DuQMoGeM, especially when it is coupled to CFD simulations. For instance, a simultaneous aggregation and breakage bivariate PBE using $n = 3$ produces a total of 810 integrands that correspond to the n^6 integrals that define A_{klijpq}, Eq. (11), and the n^4 integrals in L_{klij} definition, Eq. (12). It should be noted that, if the same polynomial basis is used for both variables ($\phi = \tilde{\phi}$), there are several symmetries that can be used to reduce the total number of integrands. However, this is just a particular case and we preferred not to use such symmetries. When the PBE solution is coupled to a CFD simulation, the number of integrands is equal to 810 times the number of CFD control volumes. Equally important is the low dimensionality of the PBE integrals, which makes their cubature calculation at the thread level not too expensive.

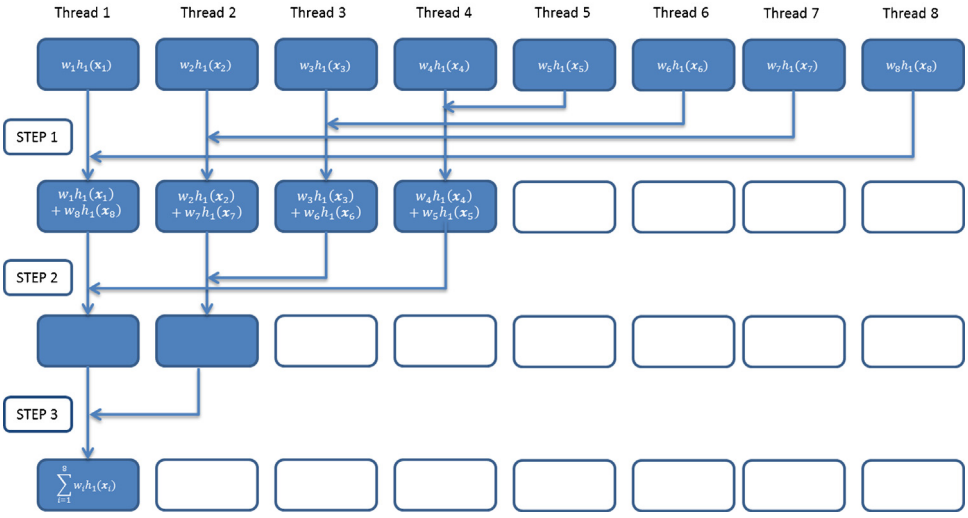

Fig. 2. Reduction algorithm for the sum of the cubature points in each block.

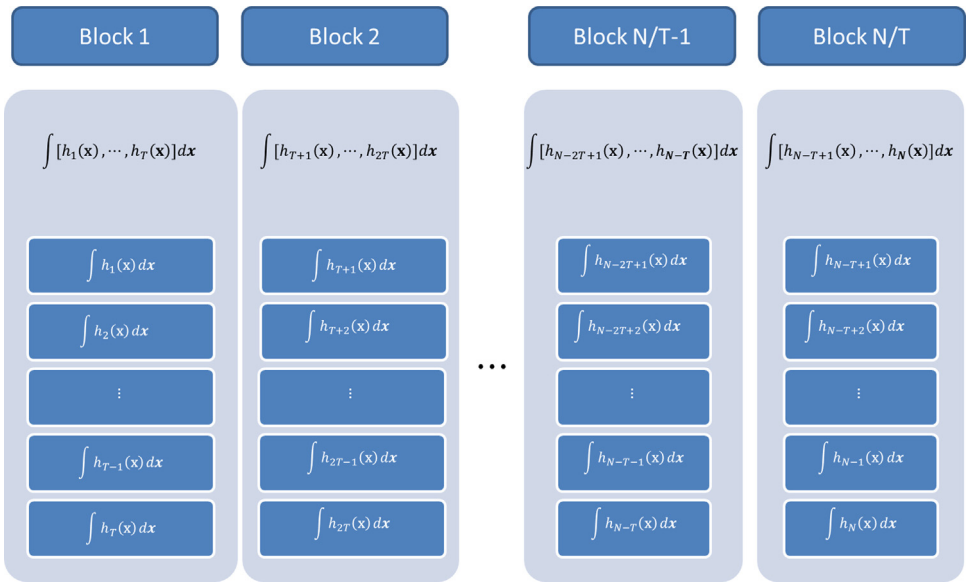

Fig. 3. Parallelism at the integrand level (PIL) with T integrands and threads per block and a total of N integrands.

6. Results and discussion

In this section, parallel algorithms are assessed regarding their computational speedup. This metric represents the ratio between the execution time of the CPU serial version of the code and the GPU version, as follows,

$$S = \frac{T_s}{T_{gpu}}, \tag{18}$$

The source codes were compiled with *g++* compiler, version 4.1.2, using the -03 high optimization flag and with *nvcc* compiler using CUDA compilation Toolkit 4.1 with double precision variables. The CPU and GPU codes were executed on an Intel(R) Xeon(R) CPU X5570 2.93GHz and NVIDIA GTX480 graphic card, respectively.

First, we carried out a speedup and scalability analysis using four multi-dimensional integrals with known analytical solutions. Then, the implemented parallel automatic cubature algorithms were applied to three bivariate population balance problems: two pure aggregation problems and one with simultaneous aggregation and breakage.

6.1. Performance analysis

Genz and Malik (1983) identified some groups of functions which are useful to evaluate multidimensional integrations routines, defining integrand families as the oscillatory, product peak, corner peak and Gaussian families. Four different integrands of these families with four dimensions were used in this work for the speedup and scalability tests of the implemented PIFL and PIL algorithms. The test integrands were:

1. Oscillatory integrand

$$f_1(\mathbf{x}) = \int_0^1 \int_0^1 \int_0^1 \int_0^1 cos(2\pi + \sum_{i=1}^{4} x_i)\, dx_1\, dx_2\, dx_3\, dx_4 \tag{19}$$

2. Product peak integrand

$$f_2(\mathbf{x}) = \int_0^1 \int_0^1 \int_0^1 \int_0^1 \frac{1}{\prod_{i=1}^{4}(1 + [x_i - 1]^2)}\, dx_1\, dx_2\, dx_3\, dx_4. \tag{20}$$

3. Corner peak integrand

$$f_3(\mathbf{x}) = \int_0^1 \int_0^1 \int_0^1 \int_0^1 \frac{1}{\left(1 + \sum_{i=1}^{4} x_i\right)^6}\, dx_1\, dx_2\, dx_3\, dx_4. \tag{21}$$

4. Gaussian integrand

$$f_4(\mathbf{x}) = \int_0^1 \int_0^1 \int_0^1 \int_0^1 e^{\sum_{i=1}^{4}(x_i - 0.5)^2}\, dx_1\, dx_2\, dx_3\, dx_4. \tag{22}$$

For the speedup and scalability analysis using the model integrands given above, two strategies were used to mimic problems with large computational costs. First, in order to analyzed the simultaneous integration of a large number of integrands, each of the above integrand was replicated to form an N dimension integrand vector. Second, as the Genz (1984) multi-dimensional integrand families were designed to be simple, the arithmetic load of each integrand was increased by calculating each quadrature point ten times.

Fig. 4 shows the speedup of the two CUDA implementations for several values of the number of integrands, N, for the four chosen integrands with absolute and relative tolerances of 10^{-8}. Fig. 5 shows similar results that were obtained with all integrands evaluated 10 times at each quadrature point, simulating a tenfold increase in the computational complexity of the integrand. From both figures, it can be seen that the speedup can be less than one for very low N values because the workload is too low to justify the GPU usage. However, when N increases, the speedup increases in a behavior that depends on the integrand and algorithm. For $N \leq 512$, it can be seen that the PIL and PIFL algorithms are equivalent but, as N further increases, it is clear that the PIL algorithm becomes superior due to its larger independence among threads.

These results also demonstrate that the speedup is strongly dependent on the complexity of the integrand functions. Comparing the maximum speedup values in Figs. 4 and 5 for the PIL algorithm, it can be verified that the tenfold increase in the computational cost of the integrand led to 2–6 times larger speedup values. Moreover, the speedup became less dependent on the integrand, being in the 50–70 range for all test integrands.

In order to verify the influence of the number of sub-regions generated by the domain division algorithm divisions on the speedup,

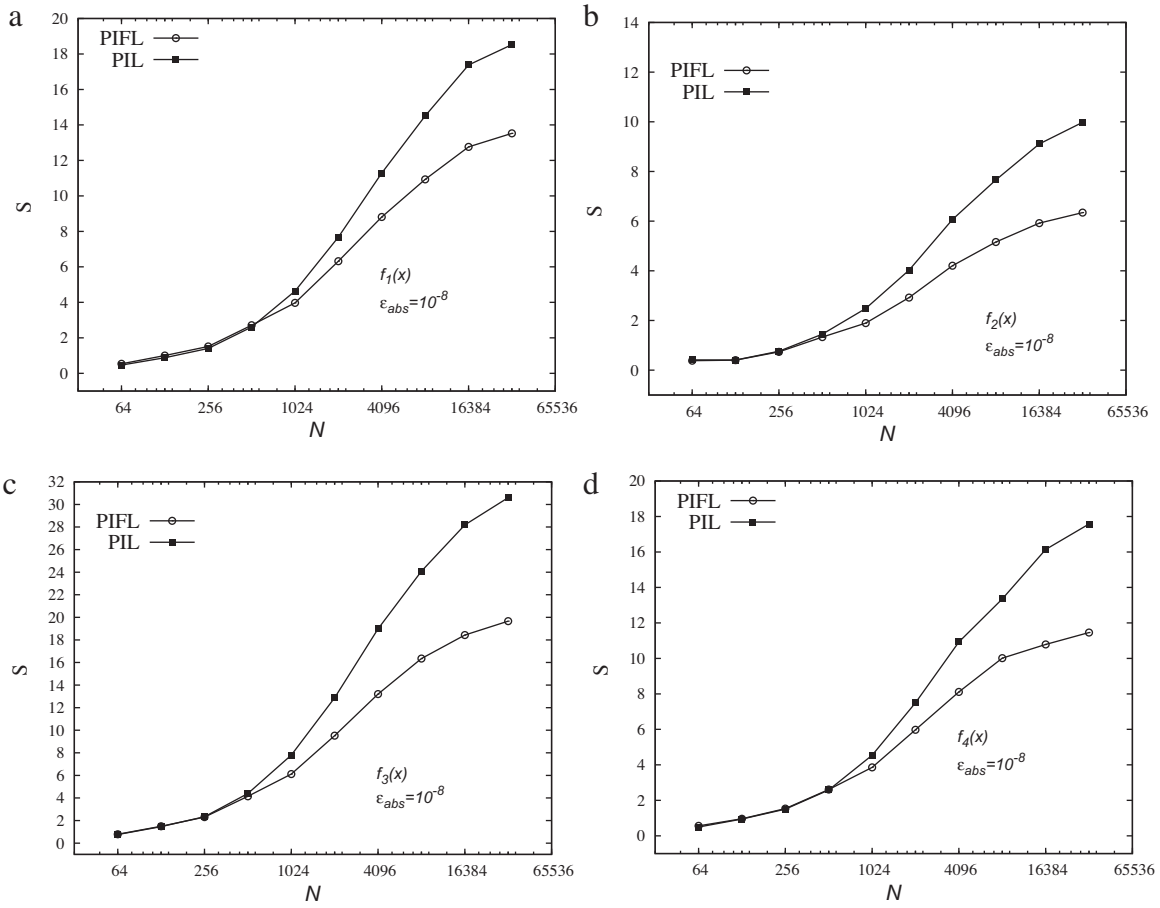

Fig. 4. Speedup analysis for N integrands with $\varepsilon_{abs} = \varepsilon_{rel} = 10^{-8}$: (a) $f_1(x)$, (b) $f_2(x)$, (c) $f_3(x)$ and (d) $f_4(x)$.

we decided to perform a similar speedup analysis to that shown in Fig. 5 but with the less strict value of 10^{-5} for both relative and absolute tolerances. The corresponding results are shown in Fig. 6. Comparing Figs. 5 and 6, it is clear that the speedup curves for both algorithms were very little affected by the value chosen for the tolerances. Therefore, the speedup of each algorithm is basically related to their parallel CUDA implementation.

It should be pointed out that the speedup achieved by both algorithms with a large number of integrands was considerable, which is important for PB-CFD simulations using DuQMoGeM. However, the PIL algorithm is more attractive for PB-CFD simulations for two reasons. First, it is faster than the PIFL algorithm for a large number of integrands. Second, it can compute more integrands simultaneously than the PIFL algorithm due to the Fermi architecture limitation of 65635 blocks (NVIDIA Corporation, 2010). In other words, as the limit of the total number of threads is larger than the total number of blocks, the PIL algorithm can support a larger number of integrands.

6.2. Application to DuQMoGeM solutions

The PIFL and PIL algorithms were used in the DuQMoGeM solution for some homogeneous bivariate population balance problems. The simulation program to solve the PBE was written in CUDA/C++. The time integration was carried out by DASSLC (Secchi, 2007) with required relative and absolute tolerances equal to 10^{-10}. The DuQMoGeM was applied to the PBE in a semi-finite domain using the Laguerre polynomial basis. The adaptive cubature domain was defined as $x \in [0, 100]$ and $y \in [0, 100]$, which is large enough not to affect the integral values. The speedup of the PBE solution includes the computational costs for both the adaptive cubature and the time integration. However, the latter is negligible and the speedup can be considered to be the speedup of the former.

Additionally, in order to evaluate the computational complexity of the PBE kernel integrands, we defined the ratio

$$E = \frac{T_c}{T_{ave}}, \tag{23}$$

where T_c is the total computational time of evaluating all PBE kernel integrands and T_{ave} is the average computational time of all the Genz's integrands used in the previous analysis but with the same number of integrands. This ratio was about 4.7 for all the cases described below, which means that the speedup results for the Genz's integrands with 10 evaluations and same N value should be comparable to the speedup obtained in the PBE solutions.

6.2.1. Pure aggregation problem (Case I)

The pure bivariate aggregation problem proposed by Gelbard and Seinfeld (1978) was simulated with the parallel algorithms in order to verify their speedup. For this case, the aggregation frequency is constant, $a(x, x', y, y') = 1.0$. Although Gelbard and Seinfeld (1978) have studied many analytical solutions for this type of problem with different initial conditions, the initial condition

$$f(x, y, 0) = e^{-(x+y)}, \tag{24}$$

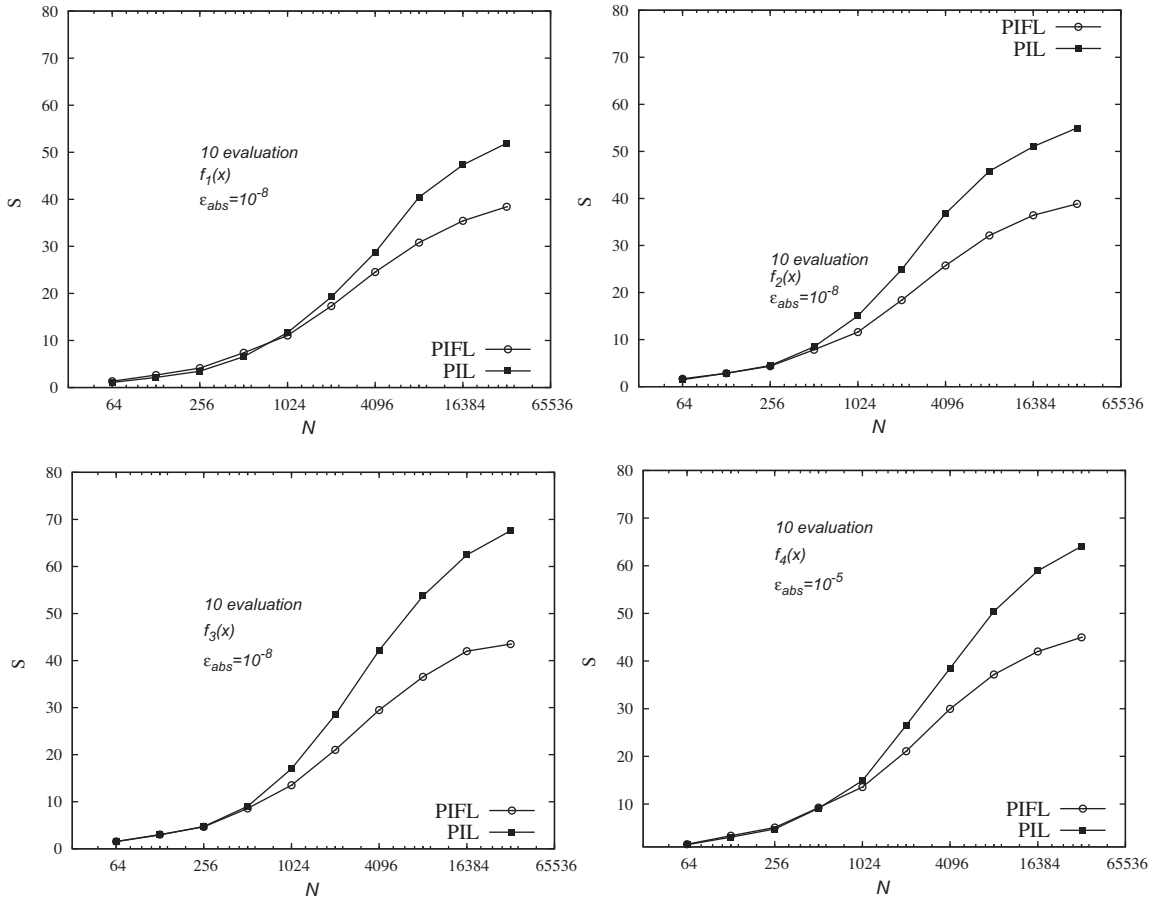

Fig. 5. Speedup analysis for N integrands, each one evaluated 10 times at each cubature point, with $\varepsilon_{abs} = \varepsilon_{rel} = 10^{-8}$: (a) $f_1(x)$, (b) $f_2(x)$, (c) $f_3(x)$ and (d) $f_4(x)$.

was used here, which leads to the analytical solution:

$$f(x,y,t) = \frac{4}{(t+2)^2} e^{-(x+y)} I_0\left(2\sqrt{\frac{t}{t+2}xy}\right), \tag{25}$$

where I_0 is the zeroth order Bessel function of the first kind. This simulation was carried out with $n = 3$, which yields a total of 729 integrands, using absolute and relative tolerances of 5×10^{-5}.

Fig. 7 shows the time evolution of moments and their relative errors. It can be seen from Fig. 7 that the results are in agreement with the analytical solution. Both the serial and the parallel versions of the code produced the same numerical results. The parallel version is about 12 times faster for the PIL algorithm and 11 times for PIFL algorithm.

6.2.2. Simultaneous aggregation and breakage problem (Case II)

This subsection presents the results for a simultaneous aggregation and breakage population balance model described in Favero and Lage (2012). Eq. (1) was considered with two additive internal variables which can represent the total particle mass, x, and the mass of a component, y, in a two-component particle. Binary breakage with linear breakage frequency and constant aggregation frequency were assumed:

$$\vartheta(x',y',t) = 2, \quad b(x,y) = x, \quad a(x,x',y,y') = 0.1. \tag{26}$$

The probability distribution function of daughter particles was considered to be uniform for the particle mass and a Dirac delta distribution for the daughter particle concentration, y/x, which have to be equal to the mother particle concentration, y'/x'. Therefore, it is written as:

$$P(x,x|x',y') = \frac{1}{x'} H(x'-x)\delta\left(y - \frac{y'x}{x'}\right). \tag{27}$$

If the analytical solution given by

$$f(x,y,t) = (2-e^{-t})xye^{-(x+y)}, \tag{28}$$

is imposed, the additional source term necessary to make it valid is (Favero & Lage, 2012):

$$S(x,y,t) = xye^{-(x+y)}\left(e^{-t} - (2-e^{-t})\left[\frac{1}{10}(2-e^{-t})\left(\frac{x^2y^2}{72} - 1\right) + 2x\left(\frac{x^2+y^2+2xy+2x+2y+2}{(x+y)^3} - \frac{1}{2}\right)\right]\right). \tag{29}$$

For this case, the absolute and relative tolerances were specified as 10^{-3} for the serial and parallel adaptive cubature codes. The simulation was performed with $n = 3$, which produces a total of 810 integrands. The results were in agreement with the analytical solution and the parallel version is about 15 times faster than the serial CPU code for both algorithms.

6.2.3. Pure aggregation problem with an additive frequency (Case III)

This aggregation problem is similar to that given above, being somewhat more complex due to the additive aggregation frequency

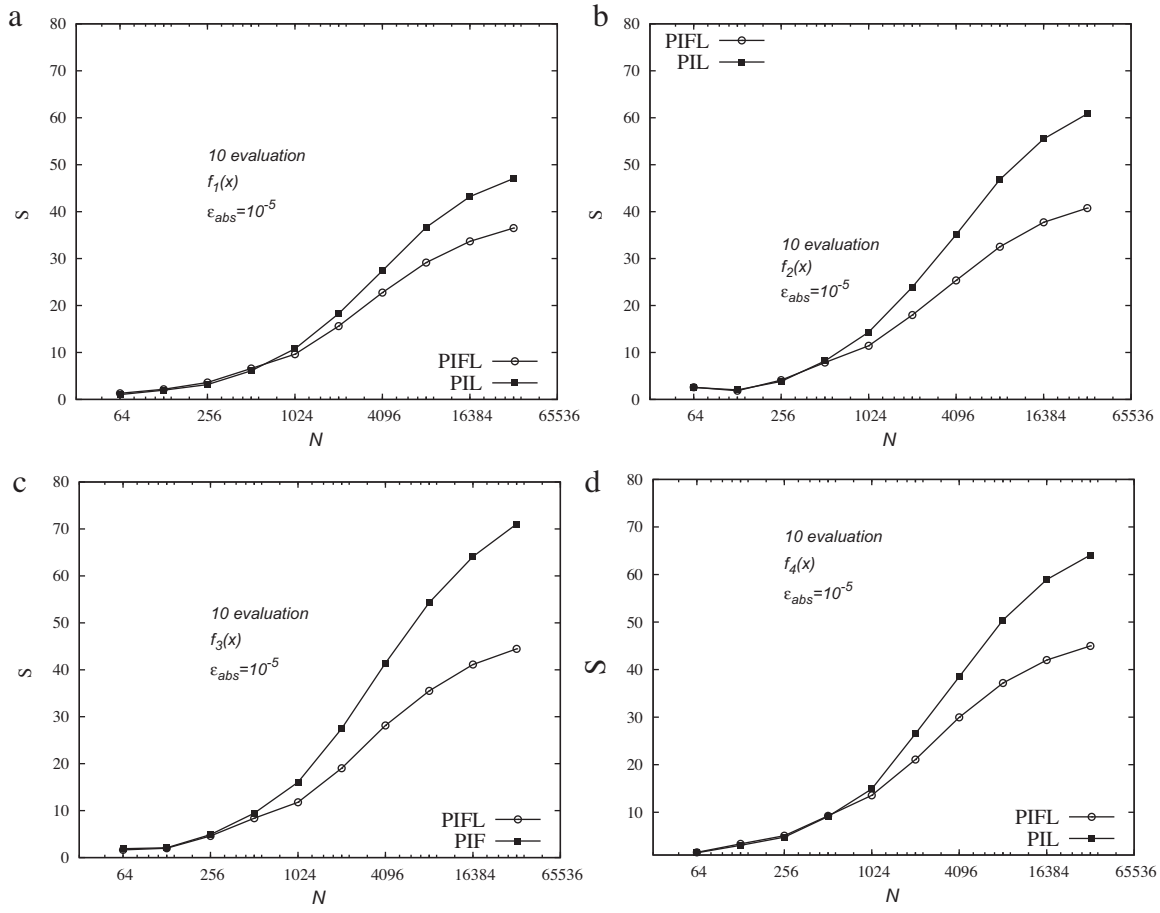

Fig. 6. Speedup analysis for N integrands, each one evaluated 10 times at each cubature point, with $\varepsilon_{abs} = \varepsilon_{rel} = 10^{-5}$: (a) $f_1(x)$, (b) $f_2(x)$, (c) $f_3(x)$ and (d) $f_4(x)$.

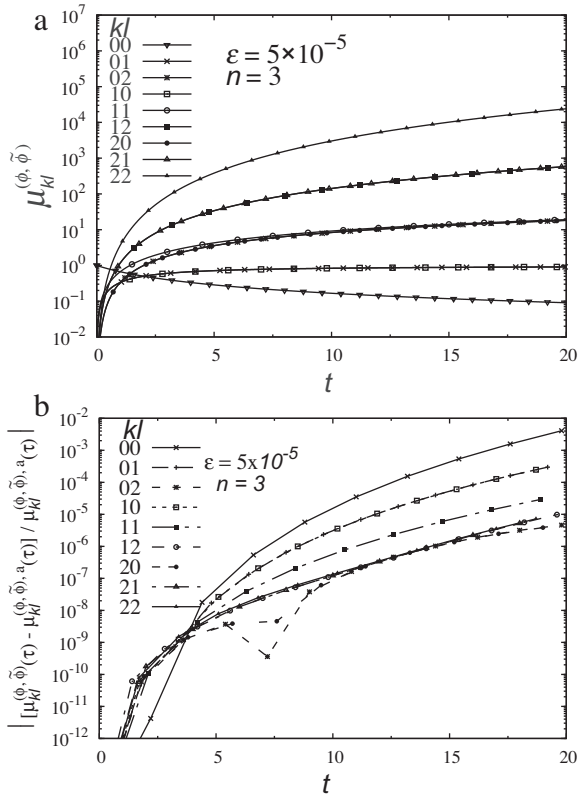

Fig. 7. Numerical solution of Case I with the parallel adaptive cubature algorithm: time evolution of (a) the moments and (b) their relative errors.

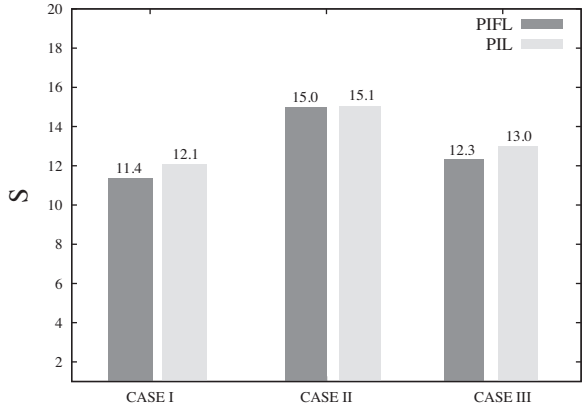

Fig. 8. Speedup values for the population balance simulations with DuQMoGeM using the PIFL and PIL algorithms.

given by $a(x, x', y, y') = (x + y) + (x' + y')$. Fernández-Díaz and Gómez-García (2010) developed an analytical solution given by

$$f(x, y, t) = exp\left(-x - y - \theta t - \frac{(x+y)\chi}{2}\right) \times \sum_{k=0}^{\infty} \frac{[xy(x+y)\chi]^k}{(k+1)!(k!)^2}, \quad (30)$$

where $\chi = 1 - e^{-\theta t}$ and $\theta = \mu_{01} + \mu_{10}$, with the initial condition given by

$$f(x, y, t) = (2 - e^{-t})xye^{-(x+y)}. \quad (31)$$

The parameters for this case were $n = 3$ and $\varepsilon_{abs} = \varepsilon_{rel} = 10^{-3}$. For this case, the speedup was 13 for the PIL algorithm and 12.3 for PIFL algorithm.

Fig. 8 summarizes the speedup for the population balance simulations presented here. As expected, these speedups of 12–15 are all comparable to those shown in Fig. 6 for $N \sim 700$–800. At first, these results might indicate that both parallel cubature algorithm can be utilized in PB-CFD simulations. However, in a PB-CFD simulation, N would be larger than 64k, which is its upper limit shown in Fig. 6. Therefore, the PIL algorithm is more adequate for PB-CFD simulations because it would obtain speedups of 50–70, even for small meshes.

7. Conclusions

Two parallel adaptive cubature codes were implemented on GPU in order to accelerate a dual quadrature method for solving the PBE. One was parallelized at the integrand level and other at the integral formula level.

From the performance results, it can be concluded that the speedups of both algorithms are little affected by the required tolerance in the integral values. On the other hand, their speedups increase with the computational cost of the integrand evaluation. Both methods have significant speedups, but the parallel algorithm at the integrand level is superior. In the test cases studied, the maximum speedup achieved was about 70 for compute-intensive integrands and between 6 and 30 for computationally cheap integrands. Furthermore, as the parallelism at the integrand level enables the integration of a larger number of integrand at one instance, it is considered more adequate for CFD simulations.

The parallel adaptive numerical integration algorithms were applied for solving three population balance problems with DuQMoGeM. All results were in excellent agreement with their respective analytical solutions and the speedups factors were between 11 and 15, being both algorithms equivalent in terms of speedup. Theses results were similar to the speedup values obtained with the compute-intensive test integrands for the same number of integrands. Therefore, it can be inferred that a PB-CFD simulation with DuQMoGeM in a 100-volume mesh should achieve a speedup around 70.

Finally, it should be pointed out that the parallel adaptive cubature codes developed in this work can be used to accelerate the computation of multidimensional integrals of a vector of integrands in other applications.

Acknowledgements

Paulo L.C. Lage and Fabio P. Santos acknowledge the financial support from CNPq, grant nos. 302963/2011-1, 476268/2009-5 and 140794/2010-7, CAPES, grant no. 0090/12-3, and FAPERJ, grant no. E-26/111.361/2010.

References

Attarakih, M. M., Drumm, C., & Bart, H.-J. (2009). Solution of the population balance equation using the sectional quadrature method of moments (SQMOM). *Chemical Engineering Science*, *64*, 742–752.

Bove, S., Solberg, T., & Hjertager, B. H. (2005). A novel algorithm for solving population balance equations: The parallel parent and daughter classes. Derivation, analysis and testing. *Chemical Engineering Science*, *60*, 1449–1464.

Bull, J., & Freeman, T. (1995). Parallel globally adaptive algorithms for multi-dimensional integration. *Applied Numerical Mathematics*, *19*, 3–16.

D'Apuzzo, M., Lapegna, M., & Murli, A. (1997). Scalability and load balancing in adaptive algorithms for multidimensional integration. *Parallel Computing*, *23*, 1199–1210.

Favero, J., & Lage, P. (2012). The dual-quadrature method of generalized moments using automatic integration packages. *Computers and Chemical Engineering*, *38*, 1–10.

Fernández-Díaz, J. M., & Gómez-García, G. J. (2010). Exact solution of a coagulation equation with a product kernel in the multicomponent case. *Physica D: Nonlinear Phenomena*, *239*, 279–290.

Fox, R., Laurent, F., & Massot, M. (2008). Numerical simulation of spray coalescence in an Eulerian framework: Direct quadrature method of moments and multi-fluid method. *Journal of Computational Physics*, *227*, 3058–3088.

Frezzotti, A., Ghiroldi, G., & Gibelli, L. (2011). Solving model kinetic equations on gpus. *Computers & Fluids*, *50*, 136–146.

Gautschi, W. (2004). *Orthogonal polynomials – Computation and approximation*. Oxford: Oxford University Press.

Gelbard, F., & Seinfeld, J. (1978). Coagulation and growth of a multicomponent aerosol. *Journal of Colloid and Interface Science*, *63*, 472–479.

Genz, A. C. (1984). Testing multidimensional integration routines. In *Proc. of international conference on Tools, methods and languages for scientific and engineering computation* Paris, France, (pp. 81–94). New York, NY, USA: Elsevier North-Holland, Inc. ISBN: 0-444-87570-0

Genz, A. C. (1989). Parallel adaptive algorithms for multiple integrals. In *Mathematics for large scale computing. Lecture notes in pure and applied mathematics* (pp. 35–47).

Genz, A. C., & Malik, A. A. (1983). An imbedded family of fully symmetric numerical integration rules. *SIAM Journal on Numerical Analysis*, *20*, 580–588.

Gordon, R. (1968). Error bounds in equilibrium statistical mechanics. *Journal of Mathematical Physics*, *9*, 655–663.

Hahn, T. (2005). Cuba – A library for multidimensional numerical integration. *Computer Physics Communications*, *168*, 78–95.

Johnson, S. G. (2008). *Cubature (multi-dimensional integration)*. http://ab-initio.mit.edu/wiki/index.php/Cubature

Kirk, D. B., & Hwu, W. W. (2010). *Programming massively parallel processors: A hands-on approach*. Burlington, MA: Morgan Kaufmann Elsevier.

Kumar, S., & Ramkrishna, D. (1996). On the solution of population balance equations by discretization-I. A fixed pivot technique. *Chemical Engineering Science*, *51*, 1311–1332.

Lage, P. L. C. (2011). On the representation of QMOM as a weighted-residual method the dual-quadrature method of generalized moments. *Computers and Chemical Engineering*, *35*, 2186–2203.

Marchisio, D. L., & Fox, R. O. (2005). Solution of population balance equations using the direct quadrature method of moments. *Journal of Aerosol Science*, *36*, 43–73.

Massot, M., Laurent, F., Kah, D., & Chaisemartin, S. D. (2010). A robust moment method for evaluation of the disappearance of evaporating sprays. *SIAM Journal on Applied Mathematics*, *70*, 3203–3234.

Mazzia, A., & Pini, G. (2010). Product Gauss quadrature rules vs. cubature rules in the meshless local Petrov–Galerkin method. *Journal of Complexity*, *26*, 82–101.

McGraw, R. (1997). Description of aerosol dynamics by the quadrature method of moments. *Aerosol Science and Technology*, *27*, 255–265.

NVIDIA Corporation (2010). *NVIDIA CUDA C programming guide. Version 3.2*. Santa Clara, CA: NVIDIA. http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

Ramkrishna, D. (2000). *Population balance – Theory and applications to particulate systems in engineering*. San Diego: Academic Press.

Rudolf, & Schrer. (2003). A comparison between (quasi-)Monte Carlo and cubature rule based methods for solving high-dimensional integration problems. *Mathematics and Computers in Simulation*, *62*, 509–517.

Sanders, J., & Kandrot, E. (2010). *CUDA by example: An introduction to general-purpose GPU programming*. Upper Saddle River, NJ: Addison-Wesley.

Schurer, R. (2001). *High-dimensional numerical integration on parallel computers*. Master's thesis. Austria: University of Salzburg.

Secchi, A. (2007). *DASSLC: User's manual, a differential-algebraic system solver*. Technical report UFRGS, Porto Alegre, RS/Brazil, http://www.enq.ufrgs.br/enqlib/numeric/DASSLC

Silva, L., & Lage, P. (2011). Development and implementation of a polydispersed multiphase flow model in OpenFOAM. *Computers & Chemical Engineering*, *35*, 2653–2666.

Stroud, A. H. (1971). *Approximate calculation of multiple integrals*. Englewood Chiffs, NJ: Prentice-Hall.

Strumendo, M., & Arastoopour, H. (2008). Solution of PBE by MOM in finite size domains. *Chemical Engineering Science*, *63*, 2624–2640.

Yeoh, G. H., & Tu, J. (2010). *Computational techniques for multiphase flows*. Oxford: Butterworth-Heinemann.

Yuan, C., Laurent, F., & Fox, R. (2012). An extended quadrature method of moments for population balance equations. *Journal of Aerosol Science*, *51*, 1–23.