Meas. Sci. Technol. **23** (2012) 025301 (14pp) doi:10.1088/0957-0233/23/2/025301

A parallel algorithm for 3D particle tracking and Lagrangian trajectory reconstruction

Douglas Barker[1], **Jonathan Lifflander**[2], **Anshu Arya**[2] **and Yuanhui Zhang**[1]

[1] Department of Agricultural and Biological Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA
[2] Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA

E-mail: dbarker2@illinois.edu, jliffl2@illinois.edu, arya3@illinois.edu and yzhang1@illinois.edu

Received 24 May 2011, in final form 16 November 2011
Published 16 December 2011
Online at stacks.iop.org/MST/23/025301

Abstract
Particle-tracking methods are widely used in fluid mechanics and multi-target tracking research because of their unique ability to reconstruct long trajectories with high spatial and temporal resolution. Researchers have recently demonstrated 3D tracking of several objects in real time, but as the number of objects is increased, real-time tracking becomes impossible due to data transfer and processing bottlenecks. This problem may be solved by using parallel processing. In this paper, a parallel-processing framework has been developed based on frame decomposition and is programmed using the asynchronous object-oriented Charm++ paradigm. This framework can be a key step in achieving a scalable Lagrangian measurement system for particle-tracking velocimetry and may lead to real-time measurement capabilities. The parallel tracking algorithm was evaluated with three data sets including the particle image velocimetry standard 3D images data set #352, a uniform data set for optimal parallel performance and a computational-fluid-dynamics-generated non-uniform data set to test trajectory reconstruction accuracy, consistency with the sequential version and scalability to more than 500 processors. The algorithm showed strong scaling up to 512 processors and no inherent limits of scalability were seen. Ultimately, up to a 200-fold speedup is observed compared to the serial algorithm when 256 processors were used. The parallel algorithm is adaptable and could be easily modified to use any sequential tracking algorithm, which inputs frames of 3D particle location data and outputs particle trajectories.

Keywords: particle-tracking velocimetry, high performance computing, experimental fluid mechanics

(Some figures in this article are in colour only in the electronic version)

Nomenclature

α	user-defined adjustment parameter for search radius
β	user-defined threshold for cost of linking a particle to a trajectory $\phi_{i,j}$
Δt	time step between image frames
Δ_o	mean spacing between particles in an image frame
γ_{correct}	correct ratio of tracked particles in a data set
γ_{coverage}	coverage ratio of tracked particles in a data set
$\hat{\mathbf{x}}_i^f$	predicted particle 3D position vector at time step (frame) f for trajectory i
$\mathbf{A}_i$	3×1 constant vector in second order term in equation (6) for trajectory i
$\mathbf{a}_i^f$	estimated particle 3D acceleration vector at time step (frame) f for trajectory i
$\mathbf{B}_i$	3×1 constant vector in first order term in equation (6) for trajectory i
$\mathbf{C}_i$	3×1 constant vector in equation (6) for trajectory i

$\mathbf{D}_k$	3×1 particle displacement vector between frames k and $k+1$ in a linked trajectory
$\mathbf{G}$	3×1 constant vector determined from regression of $\mathbf{D}_k$ in equation (9)
$\mathbf{H}$	3×1 constant vector determined from regression of $\mathbf{D}_k$ in equation (9)
$\mathbf{u}_i^f$	estimated particle 3D velocity vector at time step (frame) f for trajectory i
$\mathbf{x}_i^f$	particle 3D position at time step (frame) f for trajectory i
$\phi_{i,j}$	cost of adding the jth candidate particle to the ith trajectory
τ_k	mid-time between frame k and $k+1$
θ	angle of rotation in equations (15) through (17)
d	diameter of rotation in equations (15) through (17)
f	frame index
F_n	frame-set ID for the nth frame-set
i	local particle trajectory index
L_{correct}	number of correct particle links made in the tracking process
L_{total}	number of total particle links in the known trajectories
L_{tracked}	number of particle links made in the tracking process
M	total number of global trajectories
m	global trajectory index
N	total number of frame-sets
n	frame-set index
p	particle spacing–displacement ratio
S	frame-set size, a tunable parameter for parallel decomposition
t_f	time at frame f
$T_{n,i}$	local trajectory segment ID for trajectory segment i in frame-set n
u	total particle speed

1. Introduction

Vision-based particle-tracking techniques have been widely developed and implemented over the last two decades. A large base of the particle-tracking research has focused on hardware and algorithm development for 3D fluid velocity measurement. Particle tracking in this field, often referred to as particle-tracking velocimetry (PTV) or (3D-PTV), is emerging as a vital research tool for its benefits over statistic-based image-correlation approaches such as particle image velocimetry (PIV). These benefits include the ability to measure velocity fields with higher spatial resolution through sub-pixel accuracy in particle localization [1] and the ability to reconstruct long particle trajectories with high temporal resolution [2]. Higher order spatial and temporal derivatives can be directly evaluated from these long particle trajectories, which enables many new studies in fundamental fluid mechanics including experimental characterization of turbulent diffusion [3] and vorticity [4].

Often a goal of particle-tracking experiments is to build a statistical representation of chaotic object motion through the largest number of observations possible. Naturally, there is motivation to develop particle-tracking systems with higher resolution and accuracy, which are inherently limited by the hardware (cameras, computers, illumination, etc) employed. To increase spatial resolution, the number of resolved tracer particles per unit volume must be increased. This can be most readily achieved through three imaging hardware improvements: (1) increase image sensor resolution (>1 megapixel) [2, 5], (2) add more cameras to increase the chance of particles being observed in three or more view planes [3, 6], and (3) increase the camera frame rate to increase the particle spacing–displacement ratio, allowing higher particle densities to be resolved in time [7]. As the cost per performance for increased sensor resolution, frame rate and memory continues to drop, the particle-tracking algorithms must scale up to utilize many processors in order to handle the massive amount of data.

As the resolution of particle-tracking systems has increased with advancing camera technology, the ability to efficiently transfer, process and store the enormous amount of image data has persisted as a limit to measurement capabilities [8, 9]. One main bottleneck occurs in data transfer between camera and computer. With enormous data generation rates and limited transfer rates, the camera must hold images in buffer memory. For example, a PTV system with four 1 megapixel 8 bit monochrome cameras recording at 500 frames per second (fps) will generate 120 GB of image data in a single minute. Hoyer *et al* observed that the measurement duration with their 500 Hz cameras was limited to only 4 s due to camera memory, which led to convergence issues in their statistical analysis [5]. Processing on the computer has also been limited because the data generated do not fit into the faster random access memory (RAM) for a single processor, which is usually about 4 GB in current commodity machines. Such limitations inhibit the possibility of efficiently handling the ever growing data sets and eventually achieving real-time analysis.

There has been significant recent work with new camera technologies that can eliminate the camera to computer memory transfer bottleneck. Chan *et al* developed a data-compression system that reduced data transfer between the camera and the computer by a factor of 1000, increasing the continuous recording time from 6.5 s up to 1 week [10]. Demonstrating the use of new embedded smart cameras, Medeiros *et al* used cameras with single-instruction multiple-data processors to track several objects through image space in real time [11]. Kreizer *et al* have shown significant progress toward real-time 3D-PTV by addressing the data accumulation limitation through the use of smart cameras with embedded field programmable gate arrays (FPGA). The FPGA processes each pixel in parallel, offering a fast method to filter noise, remove background and locate particle centroids in real time prior to transferring data to the host computer [12]. Therefore, instead of transferring an image, these smart cameras only output the pixel coordinates of the particles' centroids; thus, the overall data transfer was reduced by a factor of up to 1000 [12]. In this case, the processing bottleneck can be eliminated in the camera and remaining tracking steps on the computer become the main bottleneck preventing real-time processing.

Rapidly increasing central processing unit (CPU) clock rates have begun to level off as limits on heat dissipation

have emerged. As a result, CPU manufacturers have recently shifted their focus to increasing the number of processor cores per unit. However, these multi-core processors can only be efficiently utilized through scalable parallel algorithms, which are facilitated by programming tools such as message passing interface (MPI) [13] and Charm++ [14]. Many benefits can be achieved through parallel processing in 3D-PTV including reduced run-times, scalable data storage, real-time multi-camera data streaming and the potential for complete real-time measurement. Researchers have predicted that parallel processing will speed up particle-tracking run-times [1, 6] and potentially enable real-time measurement capabilities. However, very few parallel particle-tracking algorithms have been published and data processing in the computer has become a bottleneck.

Parallel processing has been discussed since the early days of 3D particle tracking and related PIV research as a means of expediting processing times. Most have focused on parallelizing the expensive fast Fourier transform (FFT) used in PIV cross correlation and tomographic PTV. Meinhart *et al* noted that processing 1000 PIV-generated vectors required about 3 h. They parallelized the FFT operation and achieved a speedup of up to tenfold [15]. This approach was very machine specific since the standard parallel programming tools such as MPI had not yet been fully developed. Most recently, Satake *et al* developed a parallel algorithm for holographic PTV based on using MPI and achieved a 100-fold speedup for the FFT operation [16]. Satake continued this work and developed a Windows-based grid system and evaluated both spatial and temporal data-decomposition methods for parallelization of the FFT. They concluded that temporal decomposition of the image video provided an efficient method of parallelization and suggested that this approach could be useful for both PIV and PTV [17]. Thus far, no general parallel tracking algorithm has been published for multi-core CPUs and high performance clusters.

Our objective in this research is to develop and evaluate a general parallel algorithm for the particle-tracking step of PTV for Lagrangian measurements. This new parallel algorithm is designed to be easily adapted to utilize many different variants of the multi-frame tracking algorithm and scale from a single multi-core desktop to a large CPU cluster. This paper is organized as follows. Section 2 covers the current approaches to particle tracking and details the particle-tracking algorithm implemented in this research. Section 3 introduces the parallel algorithm including data decomposition, trajectory merging, parallel computation objects and communication patterns. Section 4 discusses the evaluation methods and results for three data sets used to determine the algorithm's trajectory reconstruction accuracy, consistency and scalability beyond 500 processors.

2. Particle-tracking algorithm

The underlying particle-tracking technique in 3D-PTV can be described in general form as follows [18, 19]. First, images of a particle-laden flow from three or more cameras are processed to identify particle image pixel coordinates. Next, the stereo correspondence problem between multiple camera planes is solved and the matched particle locations are used to reconstruct their 3D object locations based on camera calibration. Finally, a tracking algorithm is used to identify the temporal correspondence of particles through object space over time.

As noted by Malik [7], a key criterion to predict tracking difficulty for a given particle-tracking system and flow field is the particle spacing–displacement ratio p:

$$p = \frac{\Delta_o}{u\Delta t}. \tag{1}$$

This ratio is defined by the mean spacing between particles (Δ_o) and the mean particle displacement from frame to frame ($u\Delta t$). If p is much greater than 1, then tracking can be completed with high accuracy and relative ease [7].

This final tracking step was selected as the focus of this research on parallel algorithm development. The reasoning for this is that as camera frame rates increase over time, data parallelism can most immediately be exploited in the temporal domain. Most PTV systems today use only three or four cameras, and therefore, coarse grain data parallelism is less abundant in the spatial correspondence and 3D reconstruction algorithms. In addition, the tracking algorithm presents a greater challenge to parallelize due to the inherent dependence from one time step to another as will be discussed below.

Often the goal of the particle-tracking algorithm is to reconstruct long trajectories on the order of 100 frames and greater. Many variations of the tracking algorithm exist, but at the most general level, they share a common structure. Nearly all take the input of 3D particle coordinates at each instant in time (frame) and output the reconstructed trajectories. One modification to this approach is the use of 2D temporal information to enhance tracking in 3D space [20]. The multi-frame tracking algorithm [2], characterized by the use of a particle's location in up to five previous time steps to predict its future position, appears to be the most common for Lagrangian measurement in PTV. Several other tracking techniques have shown promise including the neural-network approach [1] and particle-filter-based algorithms such as the combined extended Kalman filter and nearest neighbor standard filter approach described by Straw *et al* [6]. In many cases, the multi-frame algorithm has been shown to be superior for long trajectory tracking and more robust against noise [2, 19, 21].

The multi-frame tracking algorithm initiates trajectories through a nearest neighbor search in an initial frame, extrapolates the particle trajectories to subsequent frames, and searches for and evaluates candidate particle quality for addition to each trajectory. Common trajectory extrapolation methods include prediction of particle position at the next time step using simple kinematic models as shown in the following equations [7, 19]:

$$\hat{\mathbf{x}}_i^{f+1} = \mathbf{x}_i^f + \mathbf{u}_i^{f-1/2}\Delta t \tag{2}$$

$$\hat{\mathbf{x}}_i^{f+1} = \mathbf{x}_i^f + \mathbf{u}_i^{f-1/2}\Delta t + \mathbf{a}_i^{f-1}\Delta t^2. \tag{3}$$

The velocity vector $\mathbf{u}_i^{f-1/2}$ is approximated by the first-order accurate finite-difference approximation and acceleration $\mathbf{a}_i^{f-1}$

can be estimated by the second-order accurate finite-difference approximation given as follows:

$$\mathbf{u}_i^{f-1/2} = \frac{\mathbf{x}_i^f - \mathbf{x}_i^{f-1}}{\Delta t} + O(\Delta t) \tag{4}$$

$$\mathbf{a}_i^{f-1} = \frac{\mathbf{x}_i^f - 2\mathbf{x}_i^{f-1} + \mathbf{x}_i^{f-2}}{\Delta t^2} + O(\Delta t^2). \tag{5}$$

An alternative to the finite-difference approach is polynomial regression, which has proven to be very robust against noise and uncertainty in the particle positions [22]. Multiple quality or 'cost' metrics have been proposed for selecting particles for addition to a trajectory including: nearest neighbor [7], minimum acceleration [7], minimum change in acceleration [7], and the ratio of regression residual to geometric mean displacement [22, 23].

2.1. Sequential particle-tracking algorithm

In this paper, the regression-based multi-frame tracking algorithm (RMT) developed by Li [22] has been augmented for 3D particle tracking and selected for parallelization. However, any tracking algorithm that inputs frames of particles and outputs trajectories could be easily substituted. The RMT algorithm was selected as a starting point due to its proven robustness against input noise and cross-gap tracking capability [22]. Also, its computational complexity warrants the speedup achieved through parallel processing.

The key aspects that were implemented from the RMT algorithm can be summarized as follows. The particle's extrapolated position in the next frame $\hat{\mathbf{x}}_i^{f+1}$ is determined through the second-order polynomial regression of up to five previous locations as follows:

$$\hat{\mathbf{x}}_i^{f+1} = \mathbf{A}_i t_{f+1}^2 + \mathbf{B}_i t_{f+1} + \mathbf{C}_i \tag{6}$$

where $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are 3×1 vectors determined by regression and t_{f+1} is the time at the $f+1$ frame. A search sphere with radius r is created around $\hat{\mathbf{x}}_i^{f+1}$ based on the estimated velocity from the previously connected points in the trajectory as shown in equation (7), where t is time corresponding to the frame index subscript and α is a user-defined constant. Then, all particles that fall inside this sphere are identified as candidates for the trajectory:

$$r = \alpha \frac{t_{f+1} - t_f}{t_f - t_{f-1}} \left| \mathbf{x}_i^f - \mathbf{x}_i^{f-1} \right|. \tag{7}$$

For each candidate, a cost function is evaluated based on the smoothness of the trajectory formed. The cost function used is defined in detail by Li *et al* [22] and given in equation (8). With this function, last four linked particles of trajectory i and a candidate particle j are evaluated to determine the associated cost denoted as $\phi_{i,j}$. If this cost is below a set threshold β, then the candidate particle is added to the trajectory:

$$\phi_{i,j} = \frac{\sqrt{\sum_{k=f-3}^{f} |\mathbf{D}_k - \mathbf{G}\tau_k - \mathbf{H}|^2}}{\sqrt{\sum_{k=f-3}^{f} |\mathbf{D}_k|^2}} \tag{8}$$

$$\hat{\mathbf{D}}_k = \mathbf{G}\tau_k + \mathbf{H}. \tag{9}$$

In this formulation, τ_k is the mid-time between frames k and $k+1$. f is the frame index of the last linked particle in the trajectory. $\mathbf{D}_k$ is the particle displacement between frames k and $k+1$. $\mathbf{G}$ and $\mathbf{H}$ are 3×1 matrices determined from the regression process of equation (9).

2.2. Sequential algorithm optimizations

Prior to parallelization, several steps were taken to minimize the computational cost of the sequential algorithm. Given that the Vandermonde matrix for the regressions in the tracking algorithm is guaranteed to be non-singular, a simple Householder rotation with backward substitution was implemented for the first- and second-order polynomial fitting. For low-noise data sets, two additional finite-difference approximations of particle velocity were derived to replace the regression in the particle search step for three- and four-point trajectories. The velocity at frame f along a trajectory i can be calculated from the second- and third-order accurate finite-difference approximations given in equations (10) and (11). These optimizations yielded up to a twofold speedup in the serial code, with negligible change in results for the data sets presented in section 4:

$$\mathbf{u}_i^f = \frac{\mathbf{x}_i^{f-2} - 4\mathbf{x}_i^{f-1} + 3\mathbf{x}_i^f}{2\Delta t} + O(\Delta t^2), \tag{10}$$

$$\mathbf{u}_i^f = -\frac{2\mathbf{x}_i^{f-3} - 9\mathbf{x}_i^{f-2} + 18\mathbf{x}_i^{f-1} - 11\mathbf{x}_i^f}{6\Delta t} + O(\Delta t^3). \tag{11}$$

3. Parallel algorithm development

To meet the research objective and create a general parallel tracking algorithm, the new parallel algorithm was designed to meet the following three requirements.

- Consistent: must provide results that are consistent with the serial version and not introduce tracking errors in the form of erroneous trajectories.
- Scalable: must scale up from one to hundreds of processors without significant reductions in speedup per processor added.
- Adaptable: must allow substitution of any sequential tracking algorithm.

3.1. Parallel implementation strategy

The parallel algorithm for particle tracking was implemented in C++ and Charm++, allowing the algorithm to be naturally expressed using object-oriented programming. Charm++ is an object-oriented parallel programming paradigm that acts as an extension to the C++ language. It allows programming objects (i.e. data structures, classes, etc) to be distributed across multiple processors and asynchronously communicate by sending and receiving messages. Thus, the resulting program can be run on shared memory systems such as multi-core workstations and on distributed memory systems including high performance clusters.

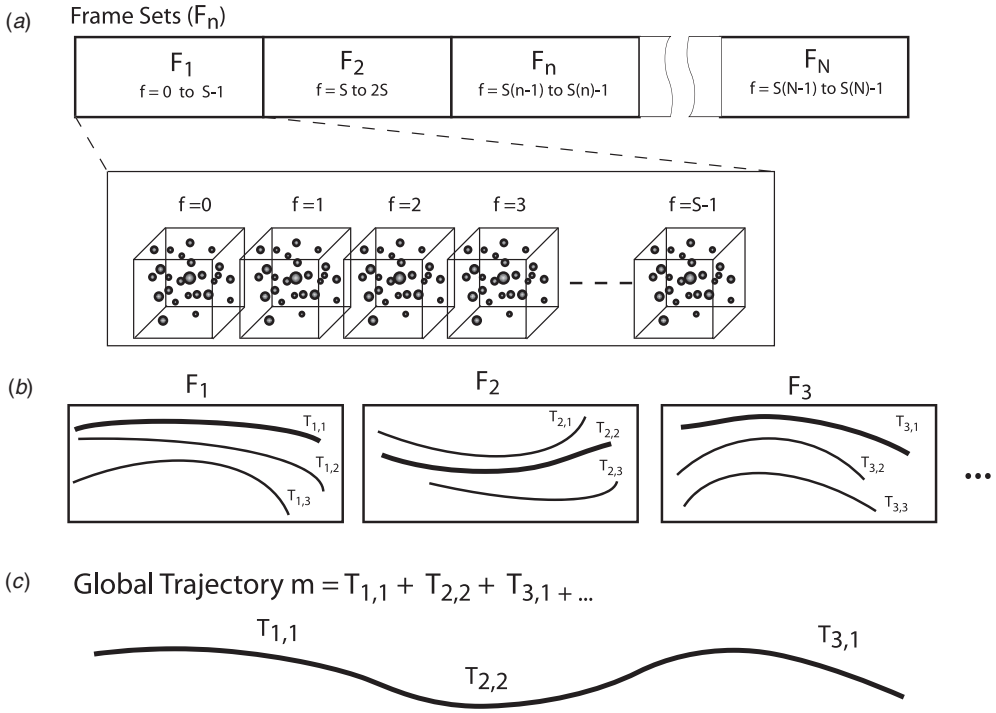

Figure 1. Parallel implementation strategy: (*a*) 3D particle location data are divided into frame sets, (*b*) trajectory segments are tracked within all frame sets simultaneously on parallel processors and (*c*) trajectory segments are merged to create global trajectories.

The general strategy used in this parallel particle-tracking algorithm is to first decompose the particle data, consisting of 3D particle positions at all time steps, into multiple sets of consecutive frames. These frame sets are distributed between a group of processors, where trajectory segments are built in parallel. The trajectory segments from each frame set are then compared with all segments in adjacent frame sets to be merged into longer global trajectories. This approach is shown in figure 1.

3.2. Data decomposition

The first step in parallelizing the tracking algorithm was to decompose the problem into multiple work units to be distributed across many processors. There are three possible data-decomposition strategies for the tracking algorithm: distributed particles, distributed frames and distributed object space. The distribution of particles or object space would require extensive communication between processors that do not share memory. Therefore, frame decomposition was chosen because the communication costs are low and it exposed sufficient parallelism for both shared and distributed memory systems. In this decomposition, the data are divided into frame sets of size S consecutive frames, which are distributed across processors as shown in figure 1. The number of frames in a set S is the parallel decomposition factor or frame-set size and is left as a tunable parameter with a minimum value of eight frames as required for the trajectory merge operation. These frame sets can be processed in parallel using any sequential tracking algorithm. The key challenge to this approach lies in merging the disjoint trajectory segments without significant processor communication overhead.

3.3. Trajectory merging

A merge operation is required to concatenate local trajectory segments spanning across a single frame set into global trajectories that span multiple frame sets. This operation falls between steps (*b*) and (*c*) in figure 1 and begins once all local trajectory segments from two adjacent frame sets have been constructed. Therefore, trajectory merging can happen asynchronously without waiting for all trajectory segments from all frame sets to be constructed. To maintain consistency with the serial version, the linear regression based cost function developed by Li *et al* [22] given in equation (8) was used to determine if two local trajectory segments from adjacent frame sets constitute a single trajectory. Only the tails of each trajectory, composed of the first four and last four linked particles, are sent to the merge function in order to minimize data transfer between processors. The merge function compares all trajectories constructed within the frame set n to those in the frame set $n + 1$, where n is the frame-set index from 1 to $N - 1$. If the first particle of the trajectory segment from the frame set $n + 1$ is within a certain proximity to the last particle from the frame set n, then the cost function is evaluated. As shown in figure 2, the cost evaluation requires four iterations per candidate trajectory to fully examine the quality of fit for each particle in the tails. If the cost associated with each of the four iterations is below a set threshold β, then the two local trajectory segments are paired for merger.

3.4. Parallel communication and data flow

Parallel particle tracking occurs in five steps: (1) data input and distribution, (2) tracking, (3) merge identification, (4)

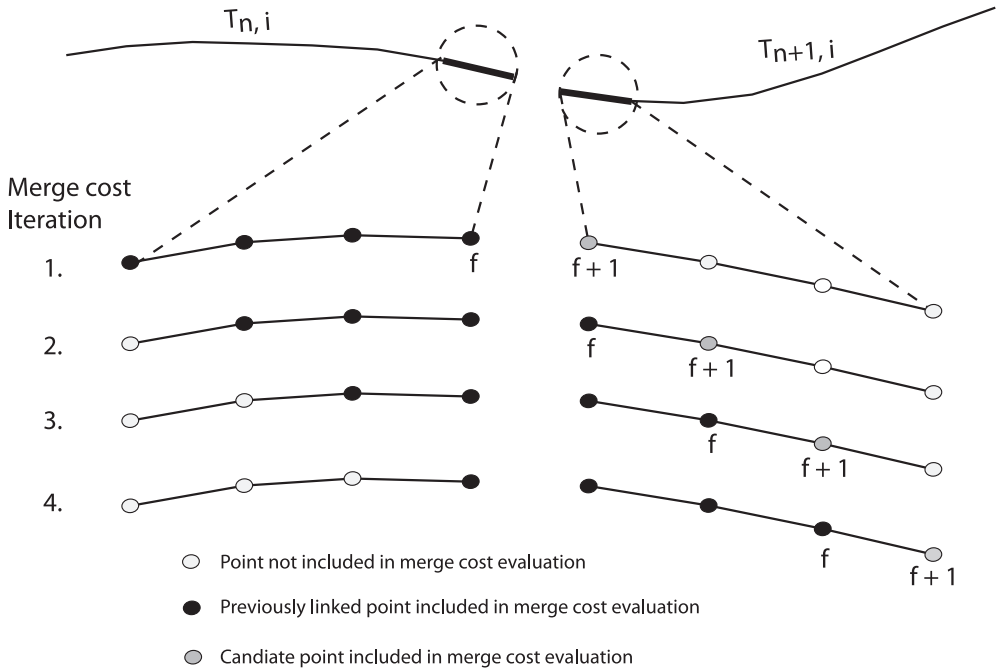

Figure 2. Trajectory merge operation: four cost-function iterations are required to evaluate the merge of the last four linked particles of trajectories in the frame set F_n with the first four linked particles of the trajectories in the frame set F_{n+1}.

global trajectory construction and (5) trajectory data output to file. Figure 3 shows a simplified example of the parallel communication and flow using only two processors and particle data divided into four frame sets, i.e. two per processor. In the first step, the 3D particle location data are read from memory and distributed in frame sets of S frames to the pool of processors. Next, each processor runs the sequential tracking algorithm on its frame sets to build local trajectory segments. Once the trajectory segments spanning two adjacent frame sets have been constructed, the merge operation is conducted as discussed in the previous section. The result of the merge operation is a local mapping of trajectory pairings between adjacent frame sets. The actual trajectory data remain fragmented across processors at this point and only the locally paired trajectory segment IDs are known by each processor. Once all local trajectory merges have been identified between each adjacent frame set, the global trajectory construction process can begin. The purpose of this phase is to consolidate the trajectory segments belonging to a single global trajectory on the same processor. First, a set of instructions is generated that defines the segments to be merged along with their respective frame-set IDs and host processor. A sample of these instructions is shown in table 1. Next, each processor selects an equal subset of global trajectories and begins communicating with the other processors to obtain the segments needed for their construction. Once a processor has received all of the contiguous trajectory segments and built the global trajectories, it outputs them to a single file and exits. The final results are a series of files (one per processor) containing full length trajectories.

Table 1. Global merge map: final instructions for global trajectory assembly from local trajectory segments.

	Frame set number				
Global trajectory ID	1	2	n	$\cdots$	N
1	$T_{1,1}$	$T_{2,2}$	$T_{n,1}$	$\cdots$	$T_{N,13}$
2	$T_{1,34}$	$T_{2,7}$	$T_{n,19}$	$\cdots$	$T_{N,15}$
m	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
M	$T_{1,3}$	$T_{2,1}$	$T_{n,24}$	$\cdots$	$T_{N,4}$

4. Algorithm evaluation and results

The parallel particle-tracking algorithm was evaluated for accuracy, consistency and scaling using three data sets. The first data set was used to evaluate consistency with the sequential version and was obtained from the PIV 3D standard images data set #352 [24]. The second set consisted of a large data set with uniform characteristics and was used to evaluate the optimal parallel performance of the algorithm on several large clusters. The third data set was generated using computational fluid dynamics (CFD) and used to test the parallel performance with non-uniform data and inherent work load imbalance across processors. A wide range of machines were used in the evaluation including a desktop workstation, a moderate-size cluster (Turing) and one very large cluster (BlueGene/P). The specifications of these machines can be found in table 2.

4.1. Performance metrics

The trajectory reconstruction accuracy of the algorithm is measured by two key metrics: the coverage ratio and the

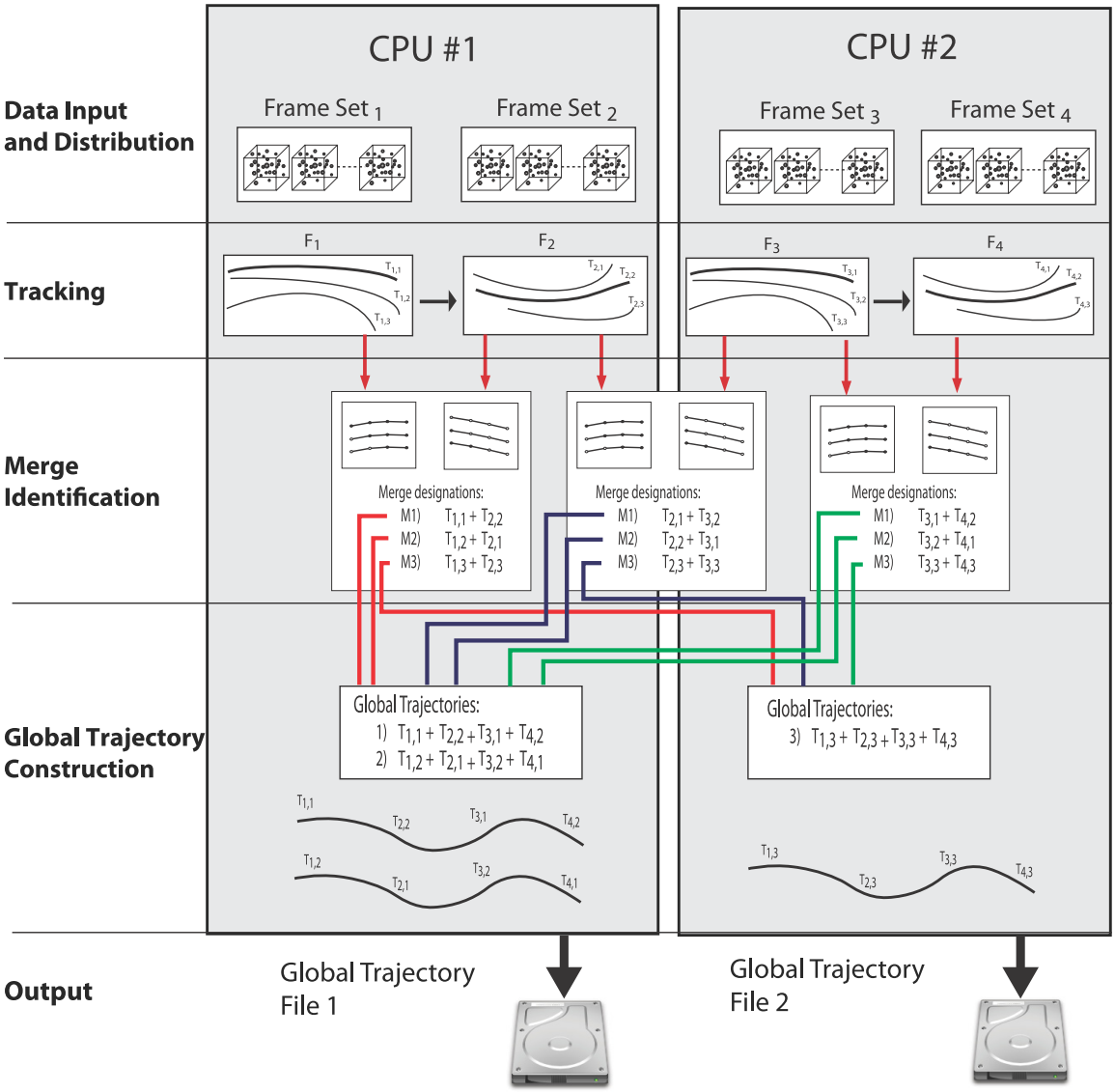

Figure 3. Parallel tracking algorithm flow diagram for a simplified case of three global trajectories spanning four frame sets on two processors.

Table 2. Computer systems used in parallel algorithm evaluation.

System name	Number of processors	CPU architecture	RAM
Multi-core workstation	2	Intel(R) Xeon(R) E5530 2.4 GHz quad-core CPUs	16 GB/CPU
Turing cluster	1536	Apple G5 2 GHz X-serve cluster	4 GB/node
BlueGene/P cluster	163 840	PowerPC 450 CPUs 850 MHz	2 GB/node

correct ratio as shown in the equations below. The coverage ratio (γ_{coverage}) is the ratio of correct two-frame particle links made during the tracking process (L_{correct}) to the total number of known input links (L_{total}) [22]. A coverage value of 1 indicates that all of the input particles were tracked correctly. The correct ratio (γ_{correct}) refers to the number of correct links made with respect to the total number of links established in the tracking process (L_{tracked}) [22]. A correct ratio of 1 indicates that all established particle links were accurately reconstructed:

$$\gamma_{\text{coverage}} = \frac{L_{\text{correct}}}{L_{\text{total}}} \tag{12}$$

$$\gamma_{\text{correct}} = \frac{L_{\text{correct}}}{L_{\text{tracked}}}. \tag{13}$$

Parallel performance can be measured in terms of speedup and scalability. The speedup of the parallel algorithm is the

Table 3. Trajectory reconstruction results using the standard PIV data set with no noise #352 [24].

Number of processors	Frames/set	Run-time (s)	γ_{correct}	γ_{coverage}	Average trajectory length	Trajectories tracked
2	8	0.619	0.984	0.923	32	1400
2	16	0.662	0.981	0.942	32	1444
2	32	0.692	0.980	0.954	32	1463
2	64	0.707	0.980	0.958	32	1468
1 (serial)	145	0.640	0.979	0.960	32	1471

Table 4. Trajectory reconstruction results using the standard PIV data set with heavy noise #352 [24]

Number of processors	Frames/set	Run-time (s)	γ_{correct}	γ_{coverage}	Average Trajectory length	Trajectories tracked
2	8	0.485	0.992	0.245	9	1375
2	16	0.529	0.990	0.288	9	1627
2	32	0.554	0.980	0.311	9	1762
2	64	0.570	0.989	0.319	9	1805
1 (serial)	145	0.510	0.989	0.324	9	1831

ratio of serial execution time to parallel execution time given the same work (equation (14)). The algorithm is timed from data input to data output excluding reading and writing of data from/to the hard drive. The measure of how well a parallel application scales is the ratio of speedup achieved to the number of processors. The optimal case is when $\frac{\text{speedup}}{\text{number of processors}}$ equals unity, in which case perfect scaling is observed. However, in real applications, adding processors creates overhead, and eventually, a loss in parallel efficiency is observed:

$$\text{speedup} = \frac{\text{sequential time}}{\text{parallel time}}. \tag{14}$$

4.2. PIV standard 3D images data set #352

The standard 3D images data set #352 from [24] was selected to test the parallel algorithm for trajectory reconstruction accuracy and consistency in comparison with the serial version. This simulated data set consists of an average of 300 particles per frame in three cameras over 145 frames [24] and is available at `www.piv.jp/image3d/image352`. The flow field is 2 cm × 2 cm × 2 cm and contains a jet impinging on a wall with the inlet speed of 15 cm s^{-1} and the Reynolds number of 3000. A subset of 3D trajectories from this set are shown in figure 4. Accuracy was measured by comparing the output trajectories with the true trajectories from the known input data. This data set is too small for a full evaluation of the parallel scaling and speedup, which are evaluated in the following sections.

Five tracking runs were completed on this data set as shown in table 3. The first run was conducted with the serial algorithm to build the performance benchmark followed by four runs of the parallel algorithm with different frame-set-decomposition sizes (8, 16, 32 and 64 frames) on a desktop workstation with two quad-core processors.

To evaluate the algorithm in the presence of noise, the data set was heavily modified and used for reevaluation. 10% of the known particles were randomly selected for removal to simulate occlusion, 10% more ghost particles were randomly added throughout the domain to simulate false detections and all particle positions were perturbed by an average of 0.003 cm in each dimension (equivalent to a 0.5 pixel error in particle centroid localization) to simulate common detection uncertainty.

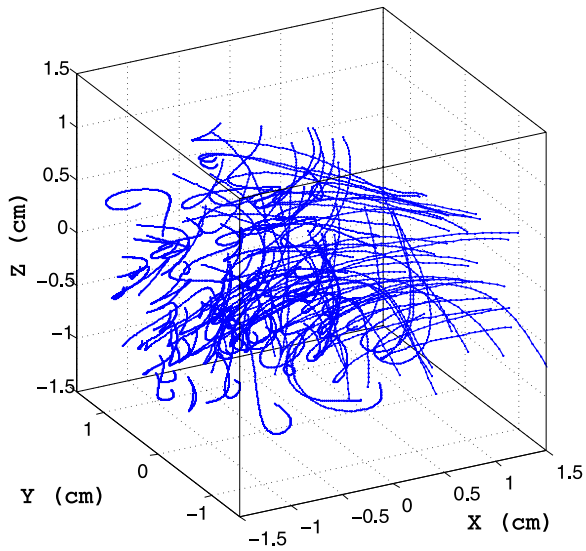

Figure 4. Trajectories from the standard PIV data set #352 spanning 75 frames or greater [24].

Results and discussion: The results show that tracking was consistent between the serial and the parallel versions, achieving average tracking correct ratios of 0.98 and average coverage ratios of 0.94. The average length of the trajectories remains at 32 frames and is consistent with the serial results and input data. This indicates that the merge operation is performing successfully. When the frame-set size S is reduced from 64 frames to 8, the correct tracking ratio increases slightly, while the coverage ratio decreases slightly. This is acceptable deviation since the accuracy (correctness) of the tracked particles remains constant and no tracking errors are introduced. Overall, the parallel algorithm was successful in preserving long and accurate trajectories when noise is low. In the presence of heavy noise and uncertainty, the performance diminishes significantly. The results of the parallel algorithm's performance in the presence of noise are shown in table 4.

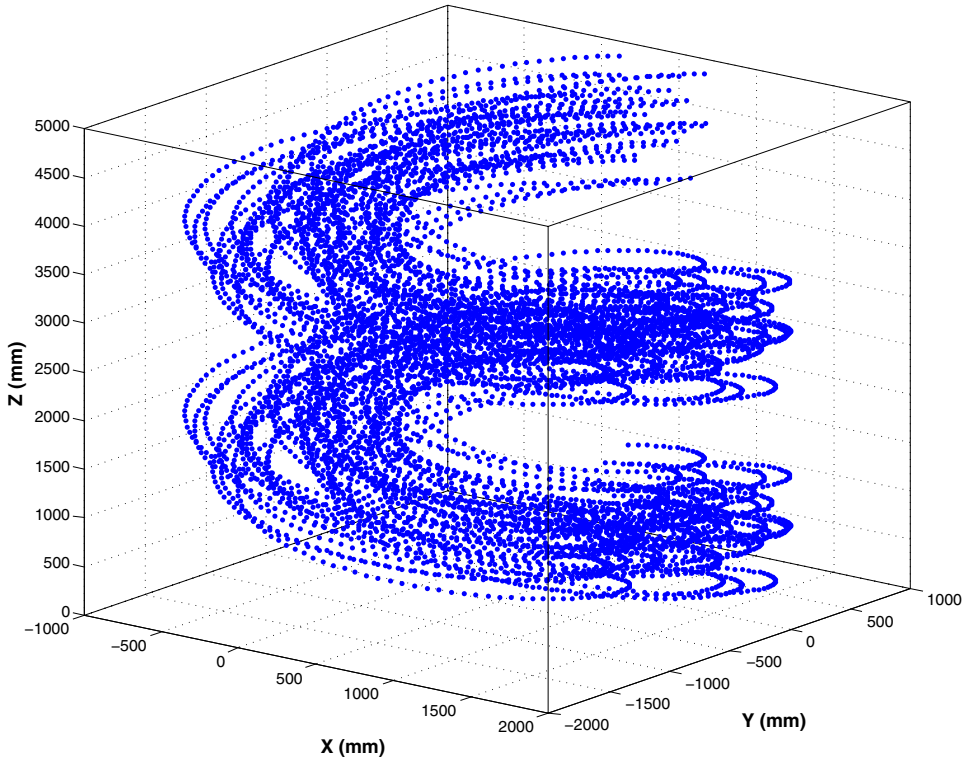

Figure 5. Simulated trajectories for scaling analysis.

Table 5. Impact of parallel decomposition factor, frame-set size (S), on parallel speedup (Turing cluster) on the data set of 1024 particles and 8192 frames.

Number of processors	Frame-set size S	Frame sets per processor	Time (s)	Speedup	Processed frames per second (fps)
1 (serial)	8192	1	3662.57	1.00	2.23
32	8	32	66.79	54.84	122.65
32	16	16	69.22	52.91	118.35
32	32	8	67.95	53.90	120.56
32	64	4	69.37	52.80	118.09
32	128	2	70.04	52.29	116.96
32	256	1	85.74	42.72	95.54

While the coverage ratio decreases with added noise, it is important to note that the correct ratio is still near 99%.

4.3. Simulated vortex for parallel performance evaluation

A large uniform data set was created to test the optimal parallel performance of the algorithm under near-perfect load balancing for up to 512 processors. This data set consists of 1024 particles moving with uniform acceleration in a downward spiral through a 2 m × 2 m × 6 m domain as shown in figure 5. The spacing–displacement ratio was greater than 10 in order to ensure 100% tracking coverage and accuracy. All trajectories are of equal length and span 8192 frames; therefore, each frame contains the same number of particles and parallel workload is balanced. This type of data set eliminates the possibility of tracking errors and permits isolated evaluation of the parallel performance in terms of scaling efficiency and speedup. To assist in the evaluation, the data set was parsed to create six total sets representing three variations in total particles (1024, 512 and 256 particles) and three trajectory lengths (8192, 4096 and 2048 frames). The particle trajectories are described in the following equations, where θ and d are the angle and the diameter of rotation, respectively, and $[x_o, y_o, z_o]$ is the particle's random location in the initial frame:

$$x = x_o + \frac{d}{2}\sin(\theta) \tag{15}$$

$$y = y_o + \frac{d}{2}\cos(\theta) \tag{16}$$

$$z = z_o + \frac{d}{2\pi}\theta. \tag{17}$$

The Turing cluster was used to evaluate the impact of varying the frame decomposition (frame-set size) from 8 to 256 frames on the parallel performance for a fixed number of processors. The BlueGene/P cluster was used to test the scalability and speedup when the number of frames and particles are varied.

Results and discussion: Table 5 shows how the parallel decomposition factor, the frame-set size S, impacts speedup. With 32 processors working on the data set of 1024

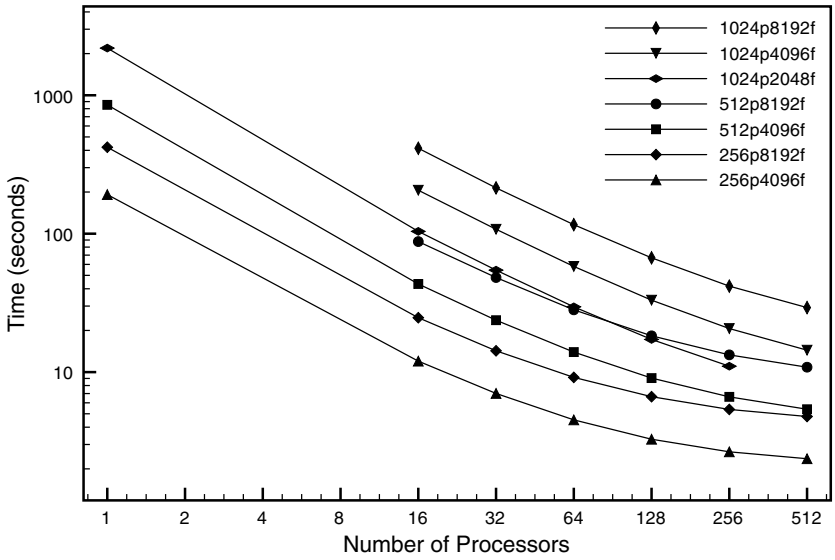

Figure 6. Scaling results from parallel execution on BlueGene/P (data sets are labeled as *A*p*B*f, where *A* is the number of particles and *B* is the number of frames).

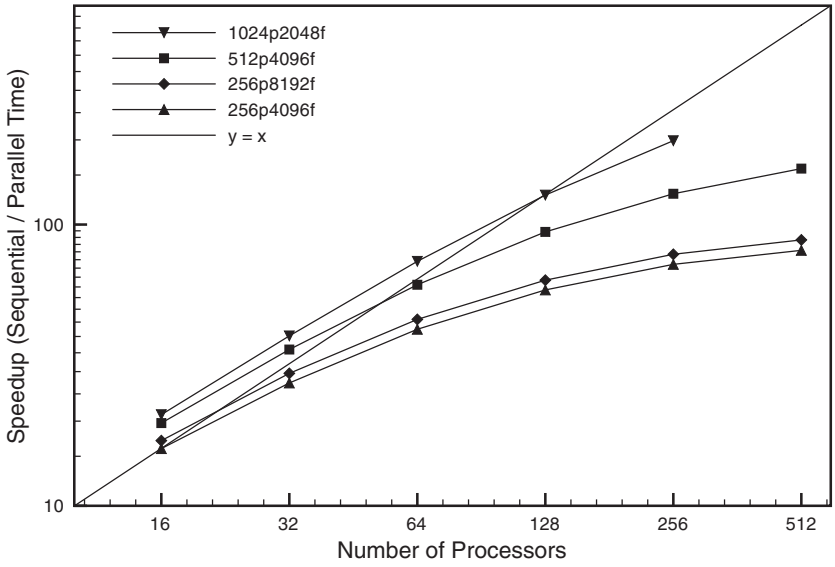

Figure 7. Speedup graph from parallel execution on BlueGene/P (data sets are labeled as *A*p*B*f, where *A* is the number of particles and *B* is the total number of frames).

particles and 8192 frames, the speedup remains nearly constant for all frame-set sizes until the number of frame sets per processor approaches 1. Once this happens, the processors are unable to hide communication latency by overlapping communication with computation. Thus, two or more frame sets should be assigned to each processor to minimize idle time. A frame-set size of $S = 8$ frames was selected for the following analysis to ensure that at least 512 processors could be used with the largest data set. The speedup of 54 achieved in this evaluation was greater than the number of processors used, indicating that the parallel algorithm has better memory characteristics than the sequential version due to slight differences in implementation.

Figure 6 shows strong scaling of the six data sets up to 512 processors on the BlueGene/P cluster. This graph demonstrates the impact of diminishing returns and loss of parallel efficiency as the number of processors increases. The run-time for the data set with 1024 particles remains at a near-constant slope with added processors, while the data set with only 256 particles begins to reduce in slope as inefficiencies arise. Clearly, the data set with more particles has more work and can be processed more efficiently with a larger number of processors. Thus, the program scales very well with an increasing number of particles tracked. The slopes of the performance curves for data sets of common particle numbers are nearly equal when the number of frames is 4096 or 8192, indicating that the number of frames processed has little impact on the scaling performance.

Figure 7 shows the speedup over the sequential algorithm. The straight line represents a linear speedup and perfect

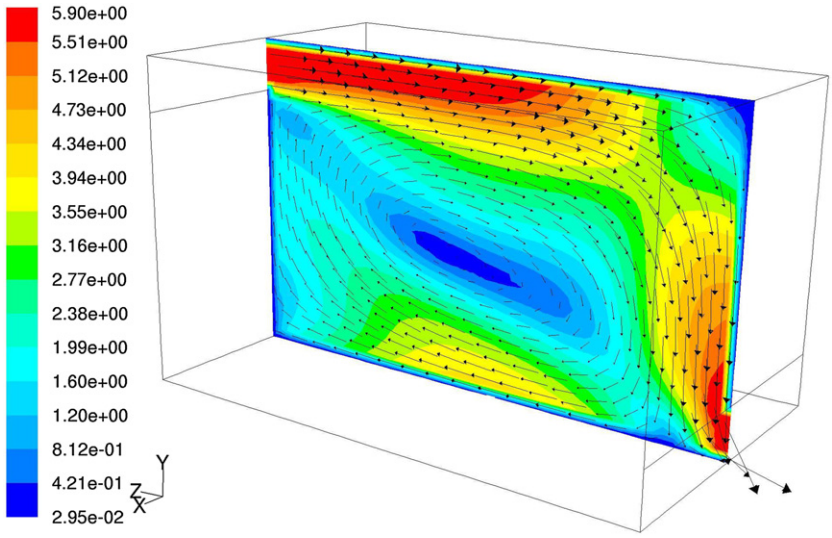

Figure 8. CFD simulated displacement indoor air ventilation velocity vector field and velocity magnitude contours (m s^{-1}).

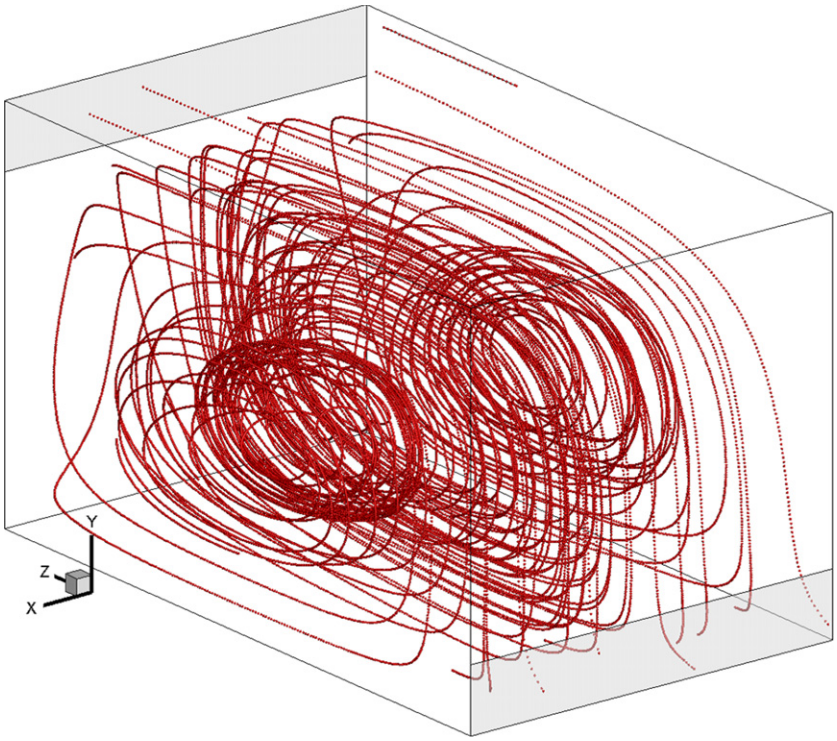

Figure 9. Subset of the CFD simulated particle trajectories in displacement indoor air ventilation.

scaling. For the first two points on the data set of 512 particles and 4096 frames and the three points on the data set of 1024 particles and 2048 frames, a super-linear speedup is observed. This phenomenon is normally due to differences between the sequential and the parallel algorithms or cache effects (the parallel version has better memory characteristics).

As the number of processors increases (the problem size remaining constant), the curve becomes sub-linear due to a decline in parallel efficiency. As the amount of work per processor decreases, the communication is more prevalent (because of less overlap with computation) and this decreases performance (less communication is being overlapped with computation). Again, this graph clearly demonstrates that the algorithm scales very well with an increasing number of particles, and the number of frames has little effect. A maximum speedup of roughly 200 is achieved with 256 processors for 1024 particles and 2048 frames. The speedup would continue to increase for this number of processors if larger data sets (particles) were used.

4.4. Simulated displacement ventilation flow

A CFD simulated indoor air flow field was used to test the trajectory reconstruction accuracy and parallel performance

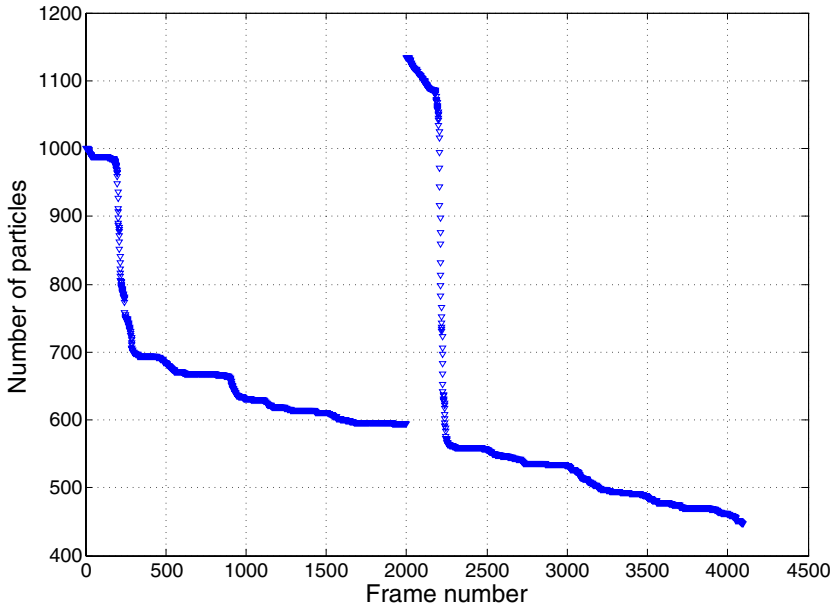

Figure 10. Fluctuation of the number of particles per frame for the CFD simulated data set.

of the new tracking algorithm in the presence of large velocity gradients and non-uniform particle seeding over time. This is done to determine how the algorithm performs when the computational load is unbalanced across processors. The data set includes 1540 particles tracked over 4096 frames to accurately evaluate parallel speedup. The flow domain was a large room (3 m × 3 m × 6 m) with a slot inlet spanning the width of the room and located on the front wall near the ceiling and a slot outlet located on the opposite wall near the floor (figure 8). The inlet boundary condition was a constant uniform velocity of 4 m s^{-1} and the outlet boundary was a standard pressure outlet set to atmospheric conditions. Turbulence was modeled using a Reynolds averaged Navier–Stokes approach. The resulting steady-state flow field solution is shown in figure 8.

Particle trajectories were simulated using a Lagrangian tracking model, assuming massless particles shown in figure 9. Particles were injected throughout the domain at two instances in time (frames 0 and 2000) to obtain a non-uniform number of particles per frame as shown in figure 10. The data were further unbalanced due to the presence of large velocity gradients, which caused a portion of the particles to leave the domain more quickly than others, leading to variation in trajectory lengths.

Results and discussion: The results from the trajectory reconstruction analysis are presented in table 6. The algorithm worked well and reconstructed the trajectories with nearly 100% coverage and correctness. However, the parallel algorithm constructed more trajectories and had a lower average trajectory length than the serial version, which indicates that some shorter trajectories did not completely merge. This is likely due to a locally small particle spacing–displacement ratio near the boundaries of several frame sets, which resulted in match ambiguity. This, however, does not introduce tracking errors as seen in no reduction of the correct tracking ratio values and therefore it may be an acceptable trade-off for increased processing speed and scalability of data storage. The frame-set size of 8 resulted in the highest percentage of correct trajectories and was used for the parallel performance analysis.

Table 6. Trajectory reconstruction accuracy results for CFD data set (workstation).

Frame set size S	γ_{correct}	γ_{coverage}	Average trajectory length	Trajectories tracked
8	0.999	0.995	1234	2057
64	0.998	0.998	1232	2070
256	0.998	0.999	1228	2079
512	0.998	0.999	1222	2088
4096 (serial)	0.998	0.999	1359	1879

The Turing cluster and multi-core workstation were used to evaluate the parallel performance with the non-uniform data set and the results are given in table 7. As expected, the speedups achieved were lower than those for the uniform data sets in the previous section due to the inherent load imbalance, which caused an increase in processor idle time. On the Turing cluster, the maximum speedup of 42 was achieved with 128 processors for a processed frame rate of 586 fps. The multi-core workstation achieved a maximum speedup of 7 with 8 processors and processed 402 fps. The workstation with 2.4 GHz processor cores and shared memory was four times faster than the Turing cluster with 2 GHz cores and distributed memory when processing the sequential code. These results show that real-time processing of the tracking algorithm for a camera frame rate of 100 fps could be possible with either machine.

Table 7. Parallel performance results for CFD data set, frame-set size (S) = 8 frames.

System	Number of processors	Run-time (s)	Speedup	Processed frames per second (fps)
Workstation	8	10.18	7.02	402.36
	4	19.54	3.66	209.62
	2	59.01	1.21	69.41
	1 (serial)	71.43	1.00	57.34
Turing cluster	128	6.98	42.02	586.82
	64	8.68	33.79	471.89
	32	12.41	23.63	330.06
	16	27.42	10.70	149.38
	8	37.88	7.74	108.13
	1 (serial)	293.31	1.00	13.96

5. Conclusion

Parallel processing of the particle-tracking algorithm is a key step in achieving a scalable 3D-PTV measurement system, where large data sets are seamlessly distributed and processed across many computers. Such a scalable system directly addresses the data management issues experienced in 3D-PTV experiments and could eventually lead to real-time measurement capabilities. A parallel-processing framework was developed and evaluation on three simulated data sets proved that it was consistent with the serial version and could efficiently scale to over 500 processors. The algorithm was based on frame decomposition and programmed using object-oriented C++ with the Charm++ extensions for asynchronous message passing between distributed objects. This approach makes adapting the new parallel algorithm for other serial particle-tracking methods relatively trivial. Any serial tracking algorithm that inputs frames of particle data and outputs trajectories can be encapsulated in an object (C++ class) and substituted with minimal modifications. One key aspect of the parallel algorithm was the asynchronous trajectory merge operation that minimizes processor idle time and data transfer between nodes.

Evaluation of the new algorithm with the PIV standard 3D images dataset #352 demonstrated that it was consistent with the optimized serial version in terms of trajectory reconstruction accuracy as quantified by the correct tracking ratio. This data set also validated the new algorithm's ability to handle merging of trajectories of non-uniform length distributed across many processors. In a few instances, several local trajectory segments did not merge due to short trajectories formed near the frame-set intersections. However, this may be an acceptable trade-off for runtime speedup and scalability since major tracking errors were not introduced into the results. Future work can be conducted to optimize the merge function.

The parallel performance evaluation showed that the new algorithm scaled well with an increasing number of particles tracked, while the number of frames processed had very little impact on the scaling performance. This implies that the parallel performance of the algorithm will remain nearly constant if only a few frames are processed at a time, as in real-time data streaming from 'smart cameras', or if thousands of frames are processed in a batch. If camera resolution is increased to grow the number of resolved tracer particles, then a proportional number of processors could be added to maintain parallel performance. The parallel decomposition factor (frame-set size S) did not influence the speedup significantly as long as each processor was assigned more than one frame set. A significant speedup of up to 200 was obtained with 256 processors for the optimal case of inherently load-balanced data (1024 particles and 2048 frames), while for a more realistic data set containing non-uniform trajectories it was observed that the speedups were still significant: 42 on 128 processors at 586 frames processed per second. While scaling is consistent from multi-core workstations to large clusters, the magnitude of speedup achieved is very dependent on the specific architecture of the system including cache size and processor clock rate.

Acknowledgments

We gratefully acknowledge the use of the Turing cluster maintained and operated by the Computational Science and Engineering Program at the University of Illinois.

References

[1] Pereira F, Stuer H, Graft E C and Gharib M 2006 Two-frame 3d particle tracking *Meas. Sci. Technol.* **17** 1680–92
[2] Ouellette N T, H Xu and Bodenschatz E 2006 A quantitative study of three-dimensional Lagrangian particle tracking algorithms *Exp. Fluids* **40** 301–13
[3] Virant M and Dracos T 1997 3D PTV and its application on Lagrangian motion *Meas. Sci. Technol.* **8** 1539–52
[4] Luthi B, Tsinober A and Kinzelbach W 2005 Lagrangian measurement of vorticity dynamics in turbulent flow *J. Fluid Mech.* **528** 87–118
[5] Hoyer K, Holzner M, Lüthi B, Guala M, Liberzon A and Kinzelbach W 2005 3D scanning particle tracking velocimetry *Exp. Fluids* **39** 923–34
[6] Straw A D, Branson K, Neumann T R and Dickinson M H 2011 Multi-camera real time 3d tracking of multiple flying animals *J. R. Soc. Interface* **8** 395–409
[7] Malik N A, Dracos T and Papantoniou D A 1993 Particle tracking velocimetry in three-dimensional flows: II. Particle tracking *Exp. Fluids* **15** 279–94
[8] Adrian R J 1991 Particle-imaging techniques for experimental fluid mechanics *Annu. Rev. Fluid Mech.* **23** 261–304
[9] Kreizer M and Liberzon A 2010 Three-dimensional particle tracking method using FPGA-based real-time image processing and four-view image splitter *Exp. Fluids* **50** 613–20
[10] Chan K Y, Stich D and Voth G A 2007 Real-time image compression for high-speed particle tracking *Rev. Sci. Instrum.* **78** 023704
[11] Medeiros H, Holguín G, Shin P J and Park J 2010 A parallel histogram-based particle filter for object tracking on SIMD-based smart cameras *Comput. Vis. Image Underst.* **114** 1264–72 (special issue on embedded vision)
[12] Kreizer M, Ratner D and Liberzon A 2009 Real-time image processing for particle tracking velocimetry *Exp. Fluids* **48** 105–10
[13] Gropp W, Lusk E, Doss N and Skjellum A 1996 A high-performance, portable implementation of the MPI message passing interface standard *Parallel Comput.* **22** 789–828
[14] Kale L V and Krishnan S 1993 CHARM++: A portable concurrent object oriented system based on C++ *Proc.*

Conf. on Object-Oriented Programming Systems, Languages and Applications pp 91–108
[15] Meinhart C D, Prasad A K and Adrian R J 1993 A parallel digital processor system for particle image velocimetry *Meas. Sci. Technol.* **4** 619–26
[16] Satake S I, Kanamori H, Kunugi T, Sato K, Ito T and Yamamoto K 2007 Parallel computing of a digital hologram and particle searching for microdigital-holographic particle-tracking velocimetry *Appl. Opt.* **46** 538–43
[17] Satake S I, Anraku T, Kanamori H, Kunugi T, Sato K and Ito T 2008 Study on high speed parallel algorithm using pc grid environment for visualization measurements by digital holographic particle tracking velocimetry *Comput. Phys. Commun.* **178** 1–7
[18] Maas H G, Gruen A and Papantoniou D 1993 Particle tracking velocimetry in three-dimensional flows: I. Photogrammetric determination of particle coordinates *Exp. Fluids* **15** 133–46
[19] Shindler L, Moroni M and Cenedese A 2010 Spatial–temporal improvements of a two-frame particle-tracking algorithm *Meas. Sci. Technol.* **21** 115401
[20] Willneff J and Gruen A 2002 A new spatio-temporal matching algorithm for 3d-particle tracking velocimetry *Proc. 9th Int. Symp. on Transport Phenomena and Dynamics of Rotating Machinery* vol 10 p 14
[21] Kitzhofer J and Bruecker C 2010 Tomographic particle tracking velocimetry using telecentric imaging *Exp. Fluids* **49** 1307–24
[22] D Li, Zhang Y, Sun Y and Yan W 2008 A multi-frame particle tracking algorithm robust against input noise *Meas. Sci. Technol.* **19** 105401
[23] Biwole P H, Yan W, Zhang Y and Roux J J 2009 A complete 3d particle tracking algorithm and its applications to the indoor airflow study *Meas. Sci. Technol.* **20** 115403
[24] Okamoto K, Nishio S, Kobayashi T, Saga T and Takehara K 2000 Evaluation of the 3d-PIV standard images (PIV-STD project) *J. Vis.* **3** 115–23