A Parallel Implementation of a 3D Reconstruction Algorithm for Real-Time Vision

J. Falcou, J. Sérot, T. Chateau, F. Jurie

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 663-670, 2006.

http://www.fz-juelich.de/nic-series/volume33

A parallel implementation of a 3d reconstruction algorithm for real-time vision.

J. Falcou[a],J. Sérot[a], T. Chateau[a], F. Jurie[b]

[a]LASMEA, UMR6602 UBP/CNRS, Campus des Cézeaux, 63177 Aubière, France.

[b]GRAVIR, INRIA/CNRS, 55 avenue de l'Europe, Montbonnot 38330, FRANCE

1. Introduction

Artificial vision requires of large amount of computing power, especially when operating on the fly on digital video streams. For these applications, real-time processing is needed to allow the system to interact with its environment, like in robotic applications or man/machine interfaces. Two braod classes of solutions have been used to solve the problem of balancing application needs and the constraints of the real-time processing: degrading algorithms or using dedicated hardware architecture like FPGA or GPU. These strategies were effective because of the specific properties of the images and the structure of the associated algorithms. However, the constant and fast progression of general purpose computers performance makes these specific solutions less and less interesting. Development time and cost now plead in favor of architectures based on standard components. During the last ten years, the use of these solutions increased with the generalization of *clusters* made up of off-the-shelf personal computers. But this type of solution has been rarely used in the context of complex vision applications operating on the fly. This paper evaluates this opportunity by proposing a cluster architecture dedicated to real-time vision applications. We describe the hardware architecture of such a solution – by justifying the technological choices carried out on the application requirements and the current state of the art – then the associated software architecture. The validity of the approach is shown with the description and performance evaluation of a real-time 3D reconstruction application.

2. Architecture Design

2.1. Hardware architecture

The hardware architecture of the cluster is shown on fig.1. The current configuration includes 14 compute nodes. Each node is a dual-processor G5[1] running at 2 GHz and with 1Gb of memory. Nodes are interconnected with Gigabit Ethernet and provided with digital video streams, coming from a pair of digital cameras, by a *Firewire IEEE1394a* bus [2]. This approach allows simultaneous and synchronized broadcasting of input images to all nodes, thus removing the classical bottleneck which occurs when images are acquired on a dedicated node and then explicitly broadcasted to all other nodes. The Ethernet network bandwidth is then fully available for application-level message-passing.

[1]Apple XServe Cluster Node

[2]In isochronous mode, at 400Mb/s

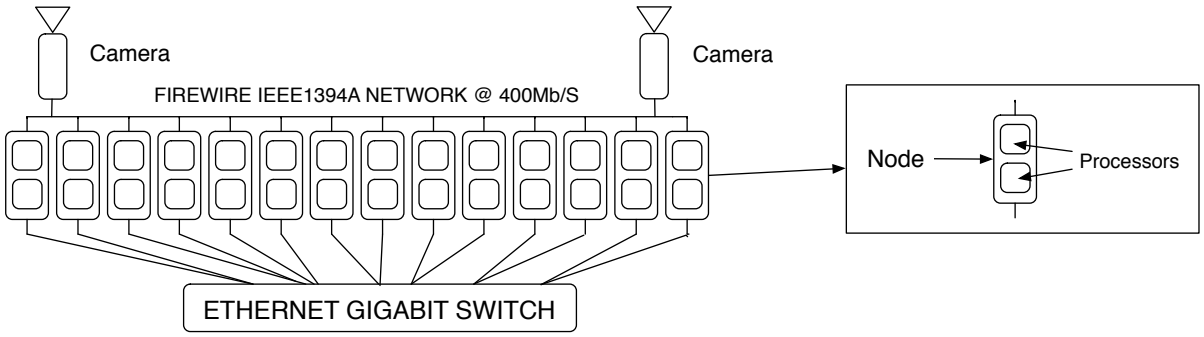

Figure 1. Cluster architecture

The following motivated the choice of the G5 processor :

- **The presence of the Altivec extension [7].** This extension can provide speedups in the range of 2-10 for a large class of low to mid-level image processing operations [5]. Without it, the size of the cluster needed for obtaining the required speed-ups (up to 50) would have been prohibitive (in terms of power consumption and cost).
- **The native support of Gigabit Ethernet network**. Gigabit Ethernet offers one of the best cost/performance ratio. More expensive solutions like MyriNet or InfiniBand have been considered, but their high cost will have reduced the actual number of node available for the cluster.

The suggested architecture is sufficiently open to enable it to benefit directly from technological progress, while supporting mainstream programming models and tools.

2.2. Software architecture

The software architecture is a layered one :

1. At the lowest level are the ***Firewire* drivers** handling the transfer of images from the cameras to the processor memory (using the DMA). For the programmer, the calls to these drivers are encapsulated within a dedicated library (CFOX[6]) which provides in particular functions for configuring cameras and obtaining a pointer to the most recently acquired frame in memory. *CFOX* also provides a easy way to synchronize video streams (sec. 3.3).
2. The middle layer deals with **parallelism issues**. We use a hybrid three-level parallel programming model, involving a fine-grain SIMD-type parallelism within each processor (Altivec), a coarse grain shared-memory multi-processing model between the two processors of a node and a coarse grain message passing based multi-processing between two processors of distinct nodes. The first level can be exploited using the Altivec native C API [7] or, more easily, using EVE [5], a high-level vectorization library specifically tuned for this extension. The second level is exploited by describing the process running on each node as two concurrent threads using the PTHREAD library. At the third level, the application is decomposed into a set of processes running in SPMD mode and communicating using the MPI library. The use of

PTHREAD to explicitly schedule the execution of threads on the two processors of each node is due to the fact that the lowest level of the software architecture doesn't map correctly onto the dual processor architecture. Each node is viewed as a single *Firewire* node by the driver layer and as a two-processor node by the parallel layer. This duality leads to resources sharing conflicts that can't be handled correctly by MPI. We need to keep a one-for-one mapping between the *Firewire* level and the MPI level and use PTHREAD for this purpose.

3. The upper layer acts as a **high-level programming framework for implementing applications**. This framework, implemented in C++, provides ready-to-use abstractions (skeletons, in the sense of [8]) for parallel decomposition, built-in functions for video i/o and mechanisms allowing run-time modification and parameterization of application-level software modules. Developing specific applications boils down to write independent algorithmic modules and to design an application workflow using those modules. One can develop various workflows using simple text description and dynamically choose which one is used by the application.

Here is a example of how we can write applications using this framework.

```
struct Data : public MPIData<Data>
{
  Frame input;
  Frame output;
};

struct Work : public Task<Work>
{
  bool operator()( Data& d )
  {
    d.output = where(d.input > 127, 255, 0);
    return true;
  }
};

int main(int argc, const char** argv )
{
  Data    d;
  Camera  camera( 30, res640x480, 8 );
  Cluster cluster(argc,argv);

  task_list(RowSplit<Frame>,Work,RowMerge<Frame>) act;
  cluster.task() = (SCM<act>(cluster.root(),cluster.world()));

  camera >> d.input;
  cluster.run(d);

  return 0;
}
```

Figure 2. A simple binary thresholding application using our software architecture

The code shown in figure 2 is a simple example that retrieves video frames from a single camera and relies on a classical scatter-compute-merge skeleton to perform a binary thresholding on them [3]. The first part of this code is the declaration of a class that defines the data we will work on and the operations that will be applied to this data. The `Work` class implements a `()` operator that performs the actual thresholding.This is done by using the EVE function `where` that implements a vectorized `if/then/else` construction. The application itself starts by building a `Data` object, the `Camera` object and the main `Cluster` object. Then tasks that will be run onto this `Cluster` are explicitly scheduled by using one of the built-in parallel skeleton class provided by the application framework. `SCM` is a simple scatter-compute-merge skeleton that splits the input image into slices, dispatches them onto a static number of workers, applies a given operation and merges the result. Here the built-in split and merge classes (`RowSplit` and `RowMerge`) and the `Work` class defined earlier are used. We also explicitely choose the root process of the `SCM` skeleton and the subgroup of nodes that will run the atual tasks. Once the activity is scheduled, video frames are grabbed and dispatched to the `Cluster` object.

3. Realistic application

The proposed platform has been assessed in the context of real-time vision applications in order to show that its dual bus architecture (Ethernet for communication and *Firewire* for video broadcasting) and its three-level parallel programming model (SIMD,SMP,MP) suit the needs of this class of applications in terms of performance and programmability.

3.1. Algorithm description

We implemented the first stages of a 3D reconstruction and tracking chain using stereoscopic vision. The application described here generates set of 3D points; it will obviously be followed by an interpretation stage that we don't cover in this paper. Basically, the algorithm consists in four different steps:

- **Image rectification (RECTIF)** : This stage warps camera frames by matching epipolar lines onto image pixels rows[9]. This simplifies further point matching by limiting search to a single pixel lines.
- **Key-point detection (DETECT)** : point of interest to be used as seed for matching stage are extracted from the rectified stereo pairs by a Harris and Stephen corner detector [4].
- **Key-point matching (MATCH)** : Each 2D key-point from the left image (resp. right image) is matched with a 2D key-point from the right image (resp. left image) by using a maximum likelihood search using a zeros normalized cross correlation (ZNCC) measure.
- **3D reconstruction (BUILD)** : Correctly matched key-point pairs are then triangulated [10] to outputs the 3D coordinates of the corresponding point.

This algorithm was chosen for three reasons: the huge number of operations per frame, the diversity of these operations and the fact it is at the root of numerous applications such as mobile robot navigation, motion capture or human-computer interface. A sample of the results obtained by this algorithm is shown on figure 3.

[3]For clarity purpose, we left out include directives and namespace declaration.

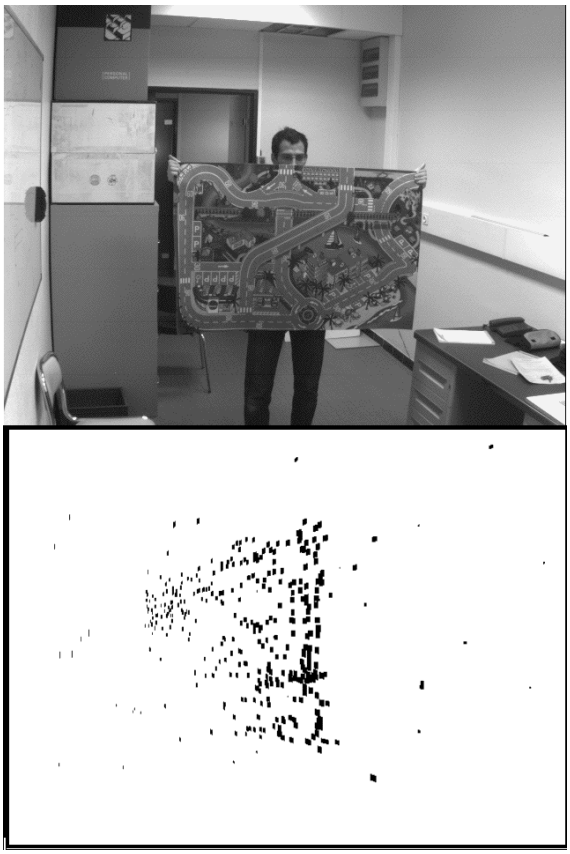

Figure 3. Visual output of the 3D reconstruction algorithm

3.2. Parallelization Strategy

The detection and matching steps both involve regular, iconic processing and have been vectorized using the EVE library [5] in order to take advantage of the SIMD level parallelism offered by the G5 Altivec extension. Parallelization of the rectification, matching and reconstruction steps is done by partitioning the source image into a fixed number of slices and having each processor operating within a slice[4]. A first approach was to use a `Farm` skeleton providing simple workload balancing. In this skeleton, a master node dynamically dispatches images slices to workers that apply each step of the reconstruction algorithm to a slice and outputs a set of 3D points. Finally, sets computed on each workers are merged and displayed. It turned out that the optimal slices size was one slice per processor. We therefore decided to use the `SCM` skeleton shown earlier. Globally, this parallelization scheme only needs two synchronization step : one before grabbing the frame from the camera and one at the end of the merging process.

3.3. Camera Synchronization

To be as accurate as possible, the algorithm must be run onto a stereo pair picturing the same space-time event. If left and right image of the pair are incoherent, the matching and reconstruction will lead to badly reconstructed scene. In practice, this means that acquisition of left and right frames on a single node must be synchronized as well as the broadcasting of images pairs to all node of the

[4]This is possible because the rectification step ensures that matching points will be on the same horizontal line of the rectified image.

cluster. This is done by a two-step synchronization mechanism :

- **A node synchronization step** : The CFOX driver implements a time-stamp verification and is able to provide synchronized frames from any number of cameras. We ensure that when the video frames are acquired, they picture the same space-time event. This synchronization step being a feature of the CFOX driver, users don't have to care about explicitly synchronizing frame acquisition on a single node.
- **A cluster synchronization step** : By using a message passing based synchronization (using `MPI_Barrier`) to force all nodes to wait each other before acquiring the next frame, we force the synchronization of frame acquisition on all nodes. This synchronization has to be explicitely triggered by the user.

4. Results

Preliminary results, obtained on 640×480×8bits video streams, appear in table 1. Computation times for each step of the algorithm and total execution times are given for several values of the processor number[5]. The first column gives the corresponding numbers for a sequential reference implementation measured on a single G5 processor at 2GHz.

Step	Seq	np=2	np=4	np=8	np=16	np=24	np = 28
RECTIF	$246ms$	$139.1ms$	$70.5ms$	$36.1ms$	$19.5ms$	$13.1ms$	$12.6ms$
DETECT	$262ms$	$80.1ms$	$40.5ms$	$20.6ms$	$11.2ms$	$7.1ms$	$6.4ms$
MATCH	$304.2ms$	$180ms$	$91.5ms$	$47.4ms$	$22.4ms$	$13.8ms$	$9.7ms$
BUILD	$180ms$	$100ms$	$53ms$	$27.5ms$	$18.2ms$	$12.0ms$	$9.5ms$
TOTAL	$992.2ms$	$479.2ms$	$244.6ms$	$122.6ms$	$68.2ms$	$42.9ms$	$38.2ms$
FPS	1.02	2.08	4.08	8.15	14.66	23.31	26.17

Figure 4. Preliminary results for the application

Minimum latency is 38.2 ms, obtained with 28 processors. This leads to a maximum throughput of 25.97 images processed per second and perfectly matches real-time vision constraints (cameras are programmed to deliver 30 FPS). The corresponding relative speedup (compared to the sequential reference implementation) is 25.97 (96% efficiency[6]). Two factors can explain these very encouraging results . First, thanks to the automatic broadcasting of images provided by the *Firewire* bus, the application exhibits a high compute *vs* communication ratio (between 0.90 and 0.92 typically). Second, the SIMD parallelization level offered by the Altivec can be efficiently exploited in at least two steps of the application : DETECT and MATCH. Without Altivec, the maximum throughput is only 17 im/s, showing that the presence of the SIMD level provides a noticeable speed-up increase. These preliminary[7] results were obtained with a rather naive implementation. They could be further improved by reformulating certain parts of the involved algorithms to make them more amenable to vectorization / parallelization and by fine tuning several algorithmic and parallelization parameters such has the dimension of the filters used in the MATCH step). For the DETECT and MATCH steps,

[5]The processor handling the display of results is not counted here.
[6]92% if we take into account the processor acting as display.
[7]Obtained with the first full working version of the platform in December 2004.

the speed-up due to AltiVec only is respectively 2,82 and 2,29. This acceleration could be easily increased by changing the data types used for computations, minimizing memory copy and swap or by rewriting some algorithm with a more SIMD-oriented approach than now.

5. Related Work

Running real-time 3D reconstruction applications on clusters has been applied to some variation of our original problem. A proposed implementation of a real-time 3D shape reconstruction algorithm uses N camera connected to N nodes in order to build a visual hull of the scene by a silhouette volume intersection method. Camera synchronization is needed to ensure that each PC/Camera pair is acquiring images at same time and solutions to this synchronization problem are described. The parallelization strategy is based on a simple pipeline in which each algorithm step is allocated to a computer node. Real-time performances are achieved with a small number of nodes and camera. Wu and Matsuyama[1] or Franco and al.[2] expose such results. Those approaches differ from the one presented in this paper in term of parallelization scheme and camera usage. Yoshimoto and Arita[3] describe a Firewire-based PC cluster dedicated to real-time image processing. It consists of a cluster of PCs interconnected by a IEEE-1394a Firewire bus. This bus is used both for transmitting video data from the camera to the processors and for communicating between processors. Although the possibility to obtain real-time performances is demonstrated (on a stereo-based 3D image restoration), sharing the Firewire bus for both video broadcasting and inter-process communication may lead to performance loss or scalability issues.

6. Conclusion

This paper shows that a cluster made of off-the-shelf computers with an efficient video streaming capability and with several parallelism levels meets the constraints of complex real-time vision applications. This was demonstrated by developing a 3D reconstruction application from a stereo video stream. As far as we know, this is the first use of such an architecture to solve real time vision application with this level of complexity. Those results are however preliminary as the cluster has been operational for a few months. Shortly, we plan to finalize the reconstruction/tracking chain to provide a fully functional real time 3D tracking application.

References

[1] Wu X., Matsuyama T. Real-time active 3d shape reconstruction for 3d video. In Proceedings of 3rd International Symposium on Image and Signal Processing and Analysis, pages. 186-191, 2003.

[2] Franco J.-S., Menier C. and Boyer E. A Distributed Approach for Real-Time 3D Modeling.CVPR Workshop on Real-Time 3D Sensors and their Applications, 2004

[3] Yoshimoto H. and al. Real-Time Image Processing on IEEE1394-based PC Cluster. In 15th International Parallel and Distributed Processing Symposium, 2001

[4] Harris C. and Stephens M. A combined corner and edge detector. In 4th Alvey Vision Conference, 1988, pp. 189-192.

[5] Falcou, J., Serot, J - E.V.E., An Object Oriented SIMD Library. In Practical Aspects of High-level Parallel Programming,ICCS 2004(3), pp. 323-330

[6] Falcou, J. - CFOX - A Firewire Camera Driver for OS X `http://wwwlasmea.univ-bpclermont.fr/Personel/Joel.Falcou/software/cfox`

[7] Ollman I. AltiVec Velocity Engine Tutorial. `http://www.simdtech.org/altivec`. March 2001.

[8] Cole M. - Algorithmic Skeletons. In Research Directions in Parallel Functional Programming, Springer-Verlag, 1999.
[9] Fusiello A.,Trucco E., Verri A. - A compact algorithm for rectification of stereo pairs. In Machine Vision and Application, vol. 12, no. 1, pp. 16–22, 2000
[10] Hartley R. I. and Zisserman A. - Multiple View Geometry in Computer Vision. Cambridge University Press - ISBN 0521623049, 2000.