A Beowulf-class architecture proposal for real-time embedded vision.

P.A. Revenga †, J. Sérot ‡, J. L. Lázaro †, J.P. Derutin ‡
† Dpto de Electrónica
Universidad de Alcalá de Henares
Madrid, 28806, Spain
revenga,lazaro@depeca.uah.es,
‡ LASMEA, UMR 6602 CNRS
Université Blaise Pascal.
Campus des Cézeaux, Aubière, France
jserot,derutin@lasmea.univ-bpclermont.fr

Abstract

In this paper a new type of parallel computer architecture dedicated to real-time vision is proposed. The proposed architecture is based upon the Beowulf *concept – i.e. COTS computing nodes interconnected by a high-speed network and running standard, free software tools – but also takes into account the constraints of* embedded *vision systems, such as "on the fly" processing of video streams, small volume, and low power consumption. Its main originality lies in the presence of two separate communication media: a dedicated bus for fast video i/o and a standard, switched inter-connection network for inter-process communication. Another distinctive point is the use of a high-level parallel programming model, based upon* algorithmic skeletons. *In order to assess the validity of the proposal, a first prototype has been built and is described in the paper. It uses G4 motherboards coming from Apple Cube mass market computers, a Fast Ethernet communication network and a FireWire (IEEE-1394) video bus. Results of preliminary benchmarks are presented.*

1 Introduction

Compared to classical HPC (High Performance Computing) applications, embedded vision applications, raise two specific issues: first, they implement *reactive systems*, operating "on the fly" on digital *streams* of images. This means that they must be able to absorb input data and output results at a minimum frequency and produce responses within a maximal latency. Second, they must meet stringent operational constraints in terms of volume and power consumption. These applications may be found for instance in autonomous navigation vehicles, like space rovers, satellites, robots and cars equipped with assisted-driving systems. In most of the cases, the need to provide high performance while meeting the afore-mentioned constraints effectively rules out implementations based upon stock-hardware.

In the context of High-Performance Computing, *Beowulf*-class parallel machines [3, 18] are enjoying an ever-increasing success. A *Beowulf* is a multi-computer, scalable architecture consisting of mass market common off-the-shelf components running a freely available operating system and software packages, and inter-connected *via* a high-bandwidth network like Fast Ethernet, Giganet, SCI, Myrinet or ATM. Typical *Beowulfs* are made of industry-standard PC CPU and interface cards and run commodity software like Linux OS and the PVM or MPI message passing libraries. Compared to other parallel architectures (vector computers, MPPs, SMPs), the main advantages of *Beowulfs* are their incomparable performance/cost ratio, their scalability – a *Beowulf* can be viewed as a *cluster* of pluggable CPU+memory *compute nodes* – and their ability to take advantage of advances in processor and networking technology. But most of *Beowulf* machines built today are dedicated to number-crunching, off-line computations (simulation, data-mining, ...). For these applications, the so-called "pile-of-PC" approach to clustering is effective: machines are built by "stacking" standard PC enclosures, interconnecting them via a dedicated switch and making i/o from/to disk on one or several dedicated server nodes. For embedded real-time vision systems (such as those embarked in autonomous vehicles) this approach is clearly not possible, due to the above-mentioned constraints (volume, power consumption).

The work described in this paper therefore aims at exploring the conceptual and technological issues associated to the design of a *Beowulf* architecture dedicated to the real-time (on-the-fly) processing of digital video streams for embedded vision systems. The paper will be organized as follows. Section 2 will recall the requirements for such systems. It will then be showed that none of the existing approaches can fulfill all these requirements and an extension of the *Beowulf* concept will be proposed as a possible answer. The resulting architecture will be defined in Section 3 and a prototype instantiation of this architecture presented in Section 4 with results of preliminary benchmarks. Section 5 will include a short survey of related work and Section 6 will concludes this paper.

2 Architectures for real-time embedded vision

Systems for embedded real-time vision must meet several constraints:

High performance. This comes from the necessity to process "on the fly" digital *streams* of images. This means that these systems must be able to absorb input data and output results at a minimum frequency and produce responses within a maximal latency. For assisted-driving applications, for instance, the typical frequencies and latencies are in the range of 10-100 frame/s and 4-40 ms respectively. For reasonably complex applications (involving 10-100 elementary operations per pixel) and medium-sized images (512×512), this requires a computing power in the range of 0.1 to 5 Gflops.

Scalability. This refers to the ability to increase the system performances without changing its architecture, by simply adding computing nodes, to match the application requirements.

Resistance to obsolescence. Dedicated architectures can be made obsolete very quickly by the rapid evolution of micro-processor technology. It is therefore crucial for these architectures to provide an upgrade path, by which technology advances can be included without major changes in the system architecture (by simply replacing processors for instance).

Volume and power consumption. This very pragmatical concern is critical for systems that must be embarked on vehicles or on satellites. For instance, Pentium processors are known to have a high power consumption and therefore require large power supplies and big heat dissipation systems. The critical point here is the ratio of computing power to power consumption (Flops/Watt) or of computing power to volume (Flops/dm^3).

Easy to use and efficient mechanism for video data acquisition. This refers to the possibility to use mass market, cheap video sub-systems (cameras, displays, recording devices, etc.) on the one hand, and to the ability to automatically and efficiently *broadcast* video data coming from an input device to all computing nodes on the other hand. The second point was proved to be of crucial importance in multi-processor architectures: in this case, the cost of explicitly broadcasting an image from one processor (the one controlling the frame grabber for instance) to all the others can easily destroy any potential gain obtained by parallelizing processing on this image.

Cost. This issue is more likely to be solved by resorting to M^{2}COTS[1] components both for the i/o sub-systems and the computing devices (processors, networking, etc.).

Programmability. This refers to the availability of user-friendly programming models environments and tools, including C (as opposed to assembler) compilers, debugging and profiling tools, application-specific libraries, etc. For multi-processors, a high-level parallel programming model is required, since low-level parallel programming models – relying on explicit message-passing like in MPI [11] or on shared-memory thread coordination like in OpenMP [13] – are known to place too much burden on the application programmer and to be too error-prone. This point is of critical importance if the machine is intended to be used by people who will not be parallel programmers in the first place.

None of the various attempts that have been made to build embedded systems for real-time vision have met all of these criteria.

Solutions based upon stock-hardware, high-end PCs may provide, in certain circumstances, the required computing power but raise significant problems as regards volume and power consumption. These problems can be solved by resorting to laptop PCs but at the expense of a significant increase in the cost/performance ratio. Scalability remains, anyway, problematic.

Architectures built from specialized processors (DSP, FPGA, ASIC) generally offers the best performance/volume or performance/watt ratio but are difficult to program (often requiring skills in assembler or VHDL programming). Interfacing to COTS components for video i/o may also be far from trivial.

Dedicated multi-processor machines, such as the TRANSVISION platform [9] or the SYMPHONIE machine [6] generally offer good scalability and better programmability but have proved to suffer from quick obsolescence.

On the other hand, the Beowulf approach to high-performance computing seems to match almost all of the above-mentioned criteria. The MIMD general architecture offers good scalability. The use of standard COTS components, both for CPU nodes and network devices provides an easy and cost-effective way for upgrading and hence resistance to obsolescence. The use of standard software components for OS (Linux) and programming tools makes them easy to operate and maintain. Moreover, high-level parallel programming tools can easily be ported to these architectures, thanks to OS-level standardization. However, two issues must be addressed for a Beowulf-like architecture to be used for embedded real-time vision. The first

[1]Mass Market Commodity Off The Self

one is the definition of an efficient hardware and software mechanism for broadcasting video data. The second concerns volume and power consumption. It is clear, indeed, that the so-called "pile-of-PC" approach used for existing Beowulf clusters – in which machines are built by simply "stacking" standard PC enclosures, interconnecting them via a dedicated switch and making i/o from/to disk on one or several dedicated server nodes – does not meet the volume and power consumption constraints of embedded vision systems. Answering these questions led us to propose a variation on the *Beowulf* concept dedicated to real-time embedded vision. This proposal is described in the next section.

3 Architecture proposal

The specific issues raised in the previous section (efficient broadcasting of images and low volume and power consumption) can be addressed by first relying on a dedicated bus for broadcasting images from video source(s) to computing nodes (and for sending video results to displaying devices) and, second, by choosing the computing nodes offering the best Gflops/watt and Gflops/dm^3 ratios in the available technology. The first point actually amounts to building a dual-network architecture, as illustrated in Fig. 1: one network (bus) dedicated to the communication of video data (preferably in a timely, synchronous manner), and another for process inter-communication. The latter can be any of the solutions used for "standard" Beowulfs (Fast Ethernet, Gigabit Ethernet, Myrinet, etc.). For the former, the best solution, in the current state of the art, seems to be the IEEE-1394 (Firewire) bus. IEEE1394 is an international, non-proprietary and inexpensive standard hardware-software digital interface for data transporting up to 400Mbps. It has a flexible topology, is hot pluggable and configurable. Digital video devices can send digital video data, can be controlled and powered by the bus. There are two types of IEEE 1394 data transfer: asynchronous and isochronous. Asynchronous transport is the traditional computer memory-mapped, load and store interface. In addition IEEE 1394 features a unique isochronous data channel interface – providing guaranteed data transport at a predetermined rate – and the possibility of controlling CMOS cameras with sub-sampling and windowing capabilities.

More latitude is given for the choice of the computing nodes. We have been experimenting with Power-PC G4 mother-boards coming from Apple Cube machines [7]. Several features of the G4 processor – and of its Cube incarnation – make it highly attractive in our context: First, it can deliver impressive performance, even at moderate clock frequencies, thanks to its built-in *Altivec* vector processing unit [1, 14]. The *Altivec* extension is specially useful for low-level iconic processing, such as found in the first stages of most of vision applications (in [8], speedups in the range of 10-15 are reported for 1D and 2D signal processing applications). Moreover, and contrary the Pentium MMX/SSE similar extensions, using the Altivec does not require assembly-level programming, thanks to the availability of an Altivec-aware version of the `gcc` compiler [2]. Second, the G4 processor has a very good Gflops/Watt ratio, with a peak power of 1 Gflops (with Altivec) and a power consumption of less than 4 Watts. Third, the Cube motherboard is very small (16x19 cm) (see Fig. 2). After removal of unnecessary devices (hard-disk, CD-drives, display controller, etc.), it should be possible to pack four G4 motherboards in the original Cube enclosure (20x20x20 cm, see Fig. 3).

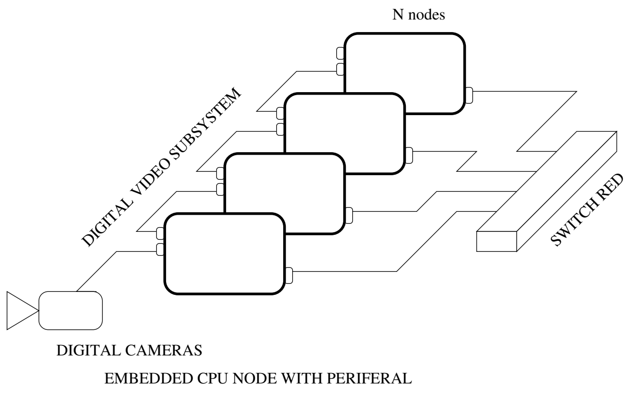

Figure 1. Ossian architecture proposal

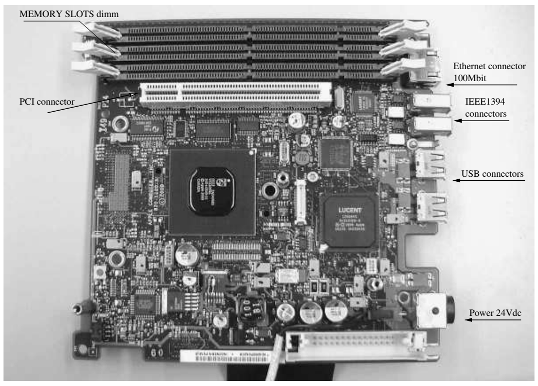

Figure 2. The G4-Cube motherboard

The software architecture of the proposed platform is a three-layered one:

- The operating system is Linux, following the *Beowulf* tradition. This is for cost and portability. The Linux distribution must of course support the target processor and chip-set (G4 in our case) and provide drivers

Figure 3. The Cube enclosure

for the two communication media (Firewire on one hand, Fast or Gigabit Ethernet, Myrinet, etc. on the other hand).

- The low-level parallel programming model is provided by a portable communication library, such as MPI [11]. Even if application programmers are not intended to write applications at this level (see below), this increases the portability of the overall software architecture.

- The high-level parallel programming model is based upon *algorithmic skeletons*. Skeletons [4, 5] are higher-order program constructs encapsulating common and recurrent forms of parallelism to make them readily available to application programmers. Experiments with the SKIPPER parallel programming environment – developed at LASMEA between 1996 and 2002 [17, 16] – have shown that skeleton-based parallel programming methodologies offer an effective way for conciliating fast prototyping and efficiency, by providing user guidance and a mostly automatic procedure for designing and implementing parallel applications, in particular in a specific application domain, such as image processing. The overall software architecture of our parallel programming environment appears in Fig 4. The application programmer provides a skeletal, structured description of the parallel program, the set of application-specific sequential functions used to instantiate the skeletons and a description of the target architecture. The SKIPPER suite of tools turns these descriptions into executable parallel code. The main software components are: a library of skeletons, a compile-time system (CTS) for generating the parallel C code and a run-time system (RTS) providing support for executing this parallel code on the target platform. The CTS can be further decomposed into a front-end, whose goal is to generate a target-independent intermediate representation of the parallel program, and a back-end system, in charge of mapping this intermediate representation onto the target architecture. The SKIPPER library of skeletons was built "bottom-up", from a careful analysis of a large corpus of existing low-to-mid level vision applications hand-coded in parallel C. It includes skeletons for handling fixed data-parallelism (geometric partitioning of images), variable data-parallelism (farming) and general task-level parallelism. More details on the SKIPPER project can be found in [17] and [16].

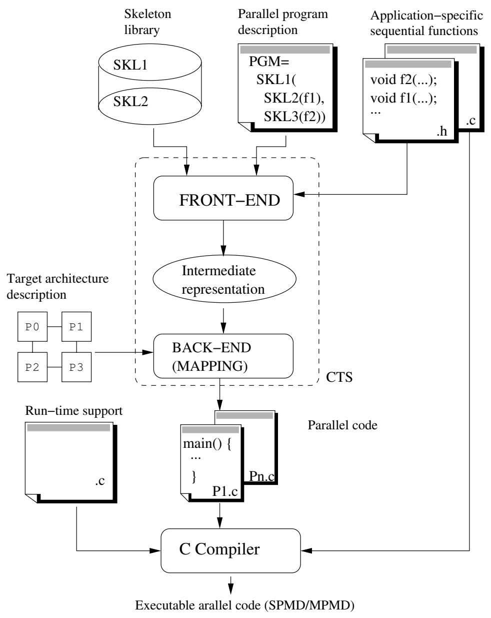

Figure 4. The SKIPPER parallel programming environment

4 Proof of concept

In order to show the validity of the architecture proposal, a prototype machine has been built. This machine – called OSSIAN [2] – is built from four Apple G4 Cubes (Power-PC 7400 processor, 64 MB, 450 MHz), a Fast Ethernet (100 Mb/s) switched inter-connection network for inter-process communication, a FireWire (IEEE-1394a) bus[3] for video broadcasting, and several digital cameras and DV-recorders for providing/recording video streams. Broadcasting of images on the Firewire is done in isochronous transmission mode, thus offering the possibility to run strict (hard) real-time applications. The software

[2]Ossian, like Beowulf, is the name of a famous knight is the Irish Celtic tradition.

[3]As shown in Fig. 1 the bus is physically made by wiring nodes one to another in a daisy-chain manner.

part consists of a Yellow Dog 2.0 distribution with kernel version 2.4.12 and a patched Linux1394 driver subsystem [10], the MPICH-1.2.2.3 [12] communication library and the SKIPPER-D [15] skeleton-based suite of tools for high level parallel programming. This version uses an intermediate representation of parallel programs as data-flow graphs (DFGs). These DFGs are automatically obtained from a high-level skeletal description which uses sequential functions written in C as arguments to skeletons taken from the SKIPPER library. For instance, the program using the `scm` (Split, Compute and Merge) skeleton for computing in parallel a Sobel filter on an image using a decomposition of this image into blocks of rows can be written as:

```
let out_img = SCM split sobel merge in_img
```

where `split`, `sobel` and `merge` are the application-specific sequential functions, with prototypes as follows:

```
void split(/*in*/ img *im, /*out*/ imgList *ims);
void sobel(/*in*/ img *im, /*out*/ img *im);
void merge(/*in*/ imgList *ims, /*out*/ img *im);
```

The run-time system (RTS) of SKIPPER-D is based upon a multi-threaded data-flow interpreter, running SPMD on all nodes of the OSSIAN machine.

4.1 Experimental results

The performances of the OSSIAN prototype have been assessed by running several benchmark applications, both in "batch" mode (with input images and results read from and written to disk) and "on the fly" (with digital video streams coming from a Firewire camera and results displayed on screen).

Table 1 summarizes results obtained with a very simple application, consisting in in applying a convolution mask (gaussian filter) on an image (read on disk). Two image sizes were tested (256 $\times$ 256, and 512 $\times$ 512) and several mask sizes were used, to vary the algorithmic complexity (i.e. the number of basic operations per pixel). In tests 1-6, the mask is applied once to each pixel. In tests 7-9, it is applied ten times, in order to (artificially) increase the algorithm complexity. The reported numbers do not include the time spent in disk i/o (reading and writing the input and result images). In each case, four timings are reported:

- the first one (t_{seq}) is the sequential reference time, measured on a single node (G4,450 MHz),
- the second one (t_{par}) is the parallel execution time, obtained on a four node architecture using a simple fixed data partitionning skeleton(SCM: each node computes the convolution on one quarter of the input image and the results are merged on one node for displaying),
- the third one (t_{vec}) is the execution time obtained on a single node with a vectorized version of the convolution algorithm, making use of the Altivec SIMD processing capabilities of the G4 processor,
- the last one (t_{pv}) is the execution time obtained when parallelizing, using the SCM skeleton, the vectorized code. This version therefore exhibits two levels of parallelism: SIMD at each processor level and MIMD (SPMD) between processors.

Test	Image	Mask	t_{seq}	t_{par}	t_{vec}	t_{vp}
1	256x256	1x5	17ms	8ms	1.3ms	2ms
2		3x3	27ms	11ms	2.5ms	2ms
3		5x5	62ms	17ms	6ms	2ms
4	512x512	1x3	50ms	18ms	4ms	4ms
5		3x3	108ms	27ms	10ms	5.5ms
6		5x5	250ms	68ms	23ms	11ms
7	512x512	1x3	500ms	131ms	40ms	16ms
8		3x3	1080ms	276ms	100ms	31ms
9		5x5	2500ms	632ms	230ms	64ms

Table 1. Absolute timings for several convolution algorithms.

These values show that, for simple, regular algorithms (like convolution) and moderately sized image (up to 512x512) the performance requirements announced in Section 2 can be met either by using SPMD-based parallelism on the four nodes or by vectorizing the code on a single node, but also that the same range of performances can be reached for more complex algorithms – or larger images – by using a combination of the two approaches. The most impressive results are observed in tests 8 and 9 (iterated convolution on a 512x512 image), where exploiting both SIMD parallelism at the processor level and SPMD parallelism between processors can reduce the total execution by a factor of 35-40.

Figures 5 and 6 show the corresponding relative speedups.

The P/S (t_{seq}/t_{par}) and PV/V (t_{vec}/t_{pv}) figures show the effect of SPMD parallelisation at the node level. With a purely scalar (non-vectorized) code on each node, the associated speedups (P/S) range from two to four with the higher values being associated with the more "complex" algorithms (i.e. the ones involving the largest number of operations per pixel). This is not surprizing, since increasing this number has a favourable effect on the compute *vs.* communication (CC) ratio of the application. When the code is vectorized at each node using the G4 Altivec

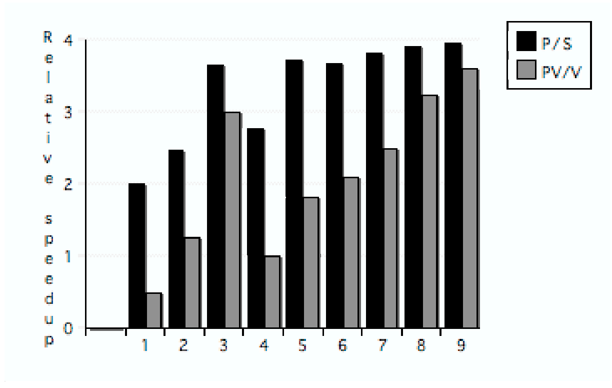

Figure 5. Relative speedups for convolution algorithms (part 1)

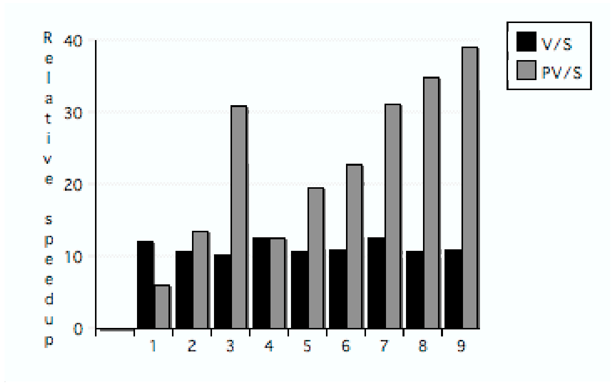

Figure 6. Relative speedups for convolution algorithms (part 2)

capabilities, the observed speedups (PV/V) are smaller - starting even at 0.5 before reaching 3.5. This is clearly explained by the fact that vectorisation at the processor level, by decreasing the computation time on each node, has an unfavourable effect on the CC ratio.

The effect of vectorization can observed separately on the V/S (t_{seq}/t_{vec}) figure. The corresponding speedup range between 10 and 12, not so far the theoretical maximum of 16[4].

Finally, the PV/S (t_{seq}/t_{pv}) figure demonstrates the combined effects of SIMD and SPMD parallelisation. Here speedups range from 6 to 40, with a variation roughly following the one observed in the P/S figure.

Performances of the OSSIAN prototype in a more realistic context have been assessed by implementing a algorithm performing image stabilisation and operating on the fly on a Firewire digital video stream. This algorithm involves two main stages: first, the detection of a set of points of interests (POIs) in the input image and, second, the tracking of these points within neighbooring windows using a correlation filter. The amount of computation for the first stage increases with the size of the input image and the number of tracked points. For the second stage, this amount depends on the size of the tracking window and the of mask used for the correlation. Table 2 gives the time to process one frame of the input stream for three values of the number of tracked points (np=21, 42 and 63) on 640x480 images[5]. The size of the tracking windows and of the correlation mask are 41x41 and 8x8 respectively. Results are first given for the two most significant configurations of the previous benchmark:

- sequential, non-vectorized code on a single node (G4, 450 MHz, 64M),
- parallel, vectorized code on four nodes with Altivec support.

For comparison, we also give timings measured on two other architectures[6]:

- a single 1.5GHz Pentium IV machine,
- a single 1Ghz G4 with Altivec support.

Although the speedups measured here are smaller than in the convolution case - due to a much less regular, and hence vectorizable, algorithm - the reported times show that a four-nodes OSSIAN architecture makes it possible to process more than 20 frames per second, even with 63 points, whereas a single Pentium machine can only cope with 21 points in this context.

5 Related work

Surprisingly enough, very few projects have attempted to apply the *Beowulf* concept to real-time vision.

In [19], Yoshimoto describes a Firewire-based PC cluster dedicated to real-time image processing. It consists of a cluster of PCs (Pentium III with Linux 2.2) interconnected by a IEEE-1394a digital 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), we

[4]The application described here operates on 8 bits pixels. The G4 Altivec unit operates on 128 bits vectors. So the maximum parallelism degree is 16.

[5]Acquisition time can be neglected here since input image are copied into processor memory by the DMA, in parallel with computations.

[6]The application was actually first developed for a Pentium IV machine and parallelized/vectorized for the OSSIAN platform afterwards.

Configuration	np	time(ms)
Single G4	21	78
Ossian SPMD+MIMD		41
Pentium IV		59
G4 1Ghz + Altivec		21
Single G4	42	104
Ossian SPMD+MIMD		40
Pentium IV		105
G4 1Ghz + Altivec		21.5
Single G4	63	140
Ossian SPMD+MIMD		51
Pentium IV		135
G4 1Ghz + Altivec		27

Table 2. Time to process one frame for the stabilisation algorithm.

believe that sharing the same bus for video broadcasting and inter-process communication can *in fine* compromise scalability. Moreover, the proposed architecture could by no way be termed as embedded since it does not take into account constraints on volume and power consumption. NASA also seems to have investigated the issues of embedded clusters for real-time image processing in the REE project. The Remote Exploration and Experimentation (REE) project focuses on the development of low-power, scalable, fault-tolerant, high-performance computing for use in space. The first test-bed hardware cluster was an VME rack and nine PowerPC 604 node boards made by Motorola. Each node has a dual CPU PowerG4 processor (7400 at 400Mhz) with 128Mb EDAC memory and connected by Myrinet. The operating system is Lynx. This machine makes 40 Million Operations per second per watt of power consumed, the communication between processors is at 132 MB/s using Myrinet. It allows automatic reconfiguration around failed components and fault injection capability for every software accessible component. There are similarities and differences between our work and the NASA REE project. The similarities are in the field of COTS embedded *Beowulf* for real time signal processing (including imaging) but the principal differences are related with the special hardware needed for spatial applications, the special video acquisition hardware, and the use of very high speed and expensive Myrinet data network. Our project relies upon M^2COTS components, commercial network interfaces, and standard video acquisition devices IEEE1394a.

6 Conclusion

From an hardware, architectural, point of view, the originality of the work described here lies in the adaptation of the well-known *Beowulf* concept to the constraints of real-time embedded vision. The main difference with "standard" *Beowulfs* is the presence of two, separate communication media: a dedicated, isochronous bus for broadcasting video data and a standard inter-connection network for parallelizing purposes. From a software point of view, the originality comes from the deliberate use of high-level programming model, based upon algorithmic skeletons, to allow image processing – as opposed to parallel programming – specialists to take advantage of high performance computing in the field of real-time vision.

We plan to build several OSSIAN-like architectures in order to demonstrate the generality of the concept and its ability to take advantage of advances in the processor and/or network technology. For instance, it seems possible, while keeping all other hardware and software components unchanged, to build machines using with SPARC motherboards or PowerPC 970 processors (recently announced).

References

[1] Altivec simd extension. http://www.altivec.org.

[2] Gnu gcc and binutils for altivec simd extensions. http://www.altivec.org/tools/Original_GNU_Tools.

[3] D. Becker, T. Stearling, D. Savarese, J. Dorband, U. Ranawake, and C. Packer. Beowulf: A parallel workstation for scientific computation. In *Proceedings of the 1995 International Conference on Parallel Processing (ICPP)*, pages 11–14, 1995.

[4] M. Cole. *Algorithmic skeletons: structured management of parallel computations.* Pitman/MIT Press, 1989.

[5] M. Cole. Algorithmic skeletons. In G. J. Michaelson and K. Hammond, editors, *Research Directions in Parallel Functional Programming*. Springer Verlag, 1999.

[6] T. Collette, C. Gamrat, D. Juvin, J. Larue, L. Letellier, M. Pethieux, R. Schmidt, and M. Viala. Symphonie, calculateur massivement parallèle : modélisation et réalisation. *Traitement du Signal*, 14(6), 1997.

[7] The apple g4 cube computer. http://www.info.apple.com/usen/g4cube.

[8] C. A. Hunter. An evaluation of PowerMac G4 systems for fortran-based scientific computing with application to computational fluid dynamics simulation. Technical report, NASA Langley Research Center, Hampton,Virginia, July 2000.

[9] P. Legrand, R. Canals, and J. Dérutin. Edge and region segmentation processes on the parallel vision machine Transvision. *Computer Architecture for Machine Perception*, pages 410–420, Dec 1993.

[10] The Linux-1394 project. http://linux1394.sourceforge.net.

[11] The Message Passing Interface (MPI) standard. http://www-unix.mcs.anl.gov/mpi.

[12] The MPICH Message Passing Interface library implementation. http://www-unix.mcs.anl.gov/mpi/mpich/.

[13] Openmp specification for parallelism compiler directives. http://www.openmp.org.

[14] M. Schmookler, M. Putrino, A. Mather J. Tyler, H. Nguyen, C. Roth, M. Sharma, M. Pham, and J. Lent. A low-power, high-speed implementation of a PowerPC(tm) microprocessor vector extension. In *Proceedings of the 14th IEEE Symposium on Computer Arithmetic*. IEEE, 1998.

[15] J. Sérot. Tagged-token data-flow for skeletons. *Parallel Processing Letters*, 11(4):377–392, Dec 2001.

[16] J. Sérot and D. Ginhac. Skeletons for parallel image processing : an overview of the SKiPPER project. *Parallel Computing*, 28(12):1785–1808, Dec 2002.

[17] J. Sérot, D. Ginhac, R. Chapuis, and J. Dérutin. Fast prototyping of parallel vision applications using functional skeletons. *Journal of Machine Vision and Applications*, 12(6):271–290, June 2001.

[18] T. Sterling, J. Salmon, D. Becker, and D. Savarese. *How to Build a Beowulf.* Massachusetts Institute of Technology, 1999.

[19] H. Yoshimoto, D. Arita, and R. Taniguchi. Real-time image processing on IEEE1394-based pc cluster. In *15th International Parallel and distributed processing symposium.* IEEE, 2001.