

MIMD–SIMD hybrid system—towards a new low cost parallel system

Leo Chin Sim ^{a,*}, Heiko Schroder ^b, Graham Leedham ^a

^a Centre for High Performance Embedded Systems, School of Computer Engineering,
Nanyang Technological University (NTU), Blk N4, 2A-32 Nanyang Avenue, Singapore

^b School of Computer and Information Technology, Royal Melbourne Institute of Technology (RMIT),
GPO Box 2476V, Melbourne, 3001 Victoria, Australia

Received 20 August 2001; received in revised form 19 July 2002; accepted 9 September 2002

Abstract

This paper describes a new parallel architectural system which we have called an *MIMD–SIMD hybrid system*. As the name implies, *MIMD–SIMD hybrid system* (also denoted as hybrid system in this paper) is a combination of both SIMD and MIMD systems working concurrently to produce an optimal architecture. This new parallel architecture has the capability of achieving speedup rates more than its corresponding MIMD architecture can achieve alone. We introduce our new SIMD concept and also show the contribution of the SIMD on this hybrid system. We have also developed a general formula for determining the speedup of the hybrid system so that accurate predictions can be made on the performance of the hybrid system. A MIMD–SIMD hybrid system was constructed and was used to implement on a visualization algorithm.

© 2002 Elsevier Science B.V. All rights reserved.

Keywords: Visualization; Parallelization; SIMD; MIMD; Hybrid system; Speedup

1. Introduction

Today, computer problems are not only getting bigger but they are also getting more complicated. The range and complexity of computer applications grow because we strive to achieve a more correct representation of reality. At the current rate of progress, it is likely that the complexity of computer applications may outgrow

* Corresponding author.

E-mail address: ps1567958a@ntu.edu.sg (L.C. Sim).

the rate of increase in the processor technology. Hence there is a need for higher performance systems. Higher performance systems can be achieved by concurrent execution of programs either on a single processor or multiple processors. Parallel computing has in fact emerged as an indispensable tool for problem solving in many scientific domains. Some of these exciting problem areas include medical imaging, financial modeling applications and computer visualization, just to name a few. With the availability of increasingly faster and cheaper processors, parallel processing has become even more feasible and realizable. The majority of the parallel computers developed to date are based on processors built using the Von Neumann principle. Flynn produced a taxonomy of parallel machines that is widely used and classifies computers according to how they process the program instruction and data streams. We thus embarked on the task of integrating the taxonomy concept and developed a very new architecture suitable for most parallel applications. This new architecture is called a MIMD–SIMD hybrid system and we have developed an application suitable for this architecture. We have also formulated the speedup equation which can predict the effects of parameter change in this hybrid system. We have also verified the correctness of the equation. Our focused application is on a computer visualization application, namely volume rendering using a Ray-Casting algorithm. Not only can the algorithm be parallel at the pixel level, its very highly data-parallel nature makes it advantageous for use in massively parallel systems. Ray-Casting is also known to produce accurate rendering of images as it generates images directly from the data values without creating intermediate geometric representations. The basic idea of this direct volume rendering is the simulation of light interaction and hence it is very view dependent i.e. requiring re-calculation each time the view changes. This in turn makes it a good research prospect for parallelization with the general goal of speeding up the computation without sacrificing image quality [1].

1.1. Related work

Visualization of scientific or medical data is an area of computer graphics of increasing interest. Fast visualization such as volume rendering is one of the common uses of parallel processing. Volume rendering is a very computationally intensive task involving large volumes of data [6]. In the past, many researchers have investigated several methods to improve the computational task of volume rendering, including the use of large and powerful supercomputers. However, it has drawbacks e.g. making the processes becoming computationally expensive. Some researchers have tried to use parallel architectures to enhance the computational performance. Most parallel algorithms for volume rendering are categorized into direct Ray-Casting of volume data for MIMD (e.g. for IBM SP2 [12], nCube 6410 [7]) or SIMD system [1,5]. A parallel algorithm for a message passing MIMD machine was also implemented on a Pixel 5 multi-computer [3]. It employed a heterogeneous multi-processor architecture that comprised graphics processors, Renderers, RISC based distributed memory machines connected in a token ring network topology. Though it was able to achieve fast processing, the system had the disadvantage that it lacked modularity due to its heterogeneous nature. Tasks are also system specific with each

MIMD system being responsible for a specific type of purpose. The high computational requirements of visualization also led to the development of special-purpose systems. Pfister and Kaufman [11] developed a special purpose visualization machine called Cube-4 which is capable of fast visualization. The architecture uses distributed interleaved memory and several parallel processing pipelines. Though the system was able to achieve fast performance (4D visualization i.e. 3D data over time), Cube-4 however, is a dedicated special purpose machine for volume rendering and its hardware cost is high. Using special purpose hardware also mean that it loses its flexibility for other applications.

2. A new parallel architecture—MIMD–SIMD hybrid system (Hybrid 4PE)

Essentially, the MIMD–SIMD hybrid system is a new form of parallel architecture in that it is a combination of both single instruction multiple data (SIMD) and multiple instruction multiple data (MIMD) systems.

The concept of this hybrid architecture is to make use of ‘Off-The-Shelf’ components and standard components and thereby making the hardware easier to maintain and also lower in cost. We have constructed and investigated a prototype hybrid system and have named this new cluster *Hybrid 4PE*. As the name implies, the configuration of our hybrid system involves using 4 PCs networked together and in each PC, a Systola 1024 card is installed for fine granularity (see Fig. 1). The Systola 1024 boards make use of the Instruction Systolic Array concept [15]. This is discussed in greater detail in Section 3 of this paper. As the Systola 1024 is a new parallel computer in its prototype stage, we also present its architecture in Section 4. The PCs in the cluster were 333 MHz homogenous systems running Windows NT operating system and the network connection between them was via fast ethernet. Inter-PC communication was accomplished using *message passing interface* (MPI) routines. In our case, we used a new Window NT compatible MPI software, PaTENT MPI, which is compliant to the MPI-2 message passing standard. The message passing programs were written as sequential code and were basically a communication library that facilitates the parallelization of codes to run on multiple CPUs

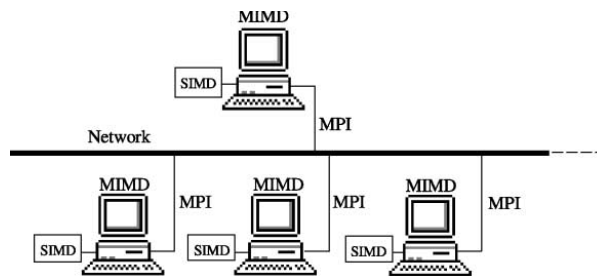


Fig. 1. MIMD–SIMD hybrid system architecture.

across network. Due to the modularity feature of such a system, additional hybrid nodes can always be easily inserted into the cluster as and when need arises.

3. Concept of instruction systolic array

In the hybrid system, there is fine granularity and coarse granularity working concurrently. Fine grain performance is provided by the SIMD and in our case, it is the Systola 1024 also known as the instruction systolic array (ISA) processor. The Systola 1024 is a quadratic array of identical processors, each connected to its four direct neighbours. The array is synchronised by a global clock that runs at a 50 MHz clock rate. The processors are controlled by instructions, row-selectors, and column-selectors. Instructions enter from the upper left corner of the processor array, and from there they move step by step in horizontal and vertical directions through the array. This guarantees that within each diagonal of the array the same instruction is active during each clock cycle. The selectors also move systolically through the array (the row-selectors horizontally from left to right, the column-selectors vertically from top to bottom). The selectors mask the execution of the instructions within the processors; i.e. an instruction is executed if and only if both selector bits, currently in that processor, are set to one. Otherwise, a no-operation is executed. This construction leads to a flexible structure which creates the possibility of very efficient solutions for a large variety of applications, e.g. image processing [2,13,14].

The upper image of Fig. 2 shows two consecutive instructions and all row and column selectors are active. The lower images depict the execution of these instructions in the first four cycles. Each processor has read and write access to its own memory. Besides that, it has a designated communication register (C-register) that can also be read by its four neighbour processors (C[NORTH], C[WEST], C[SOUTH], C[EAST]).

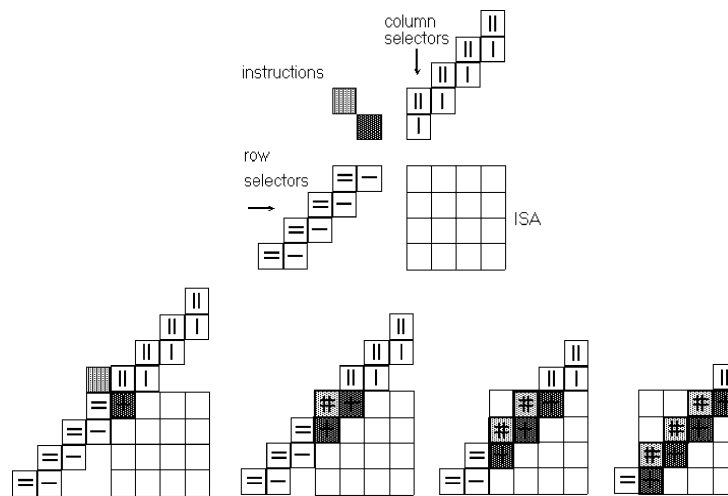


Fig. 2. Control flow in an ISA.

which denotes the C-register of a processor's neighbour). Within each clock phase reading access is always performed before writing access. Thus, two adjacent processors can exchange data within a single clock cycle in which both processors overwrite the contents of their own C-register with the contents of the communication register of its neighbour. This convention avoids read/write conflicts and also creates the possibility of performing *aggregate functions* within one instruction (or a constant number of instructions). Aggregate functions on a processor array are associative and commutative functions to which every processor provides an argument value. As they are commutative and associative, aggregate functions can be evaluated in many different ways (orders). The ISA supports top-down column operations and left-right row operations, due to the systolic flow of the instructions. Thus, an aggregate function can be implemented on the ISA by executing it firstly on all columns, placing the corresponding results within the last processor within each column, and secondly applying the aggregate function to these results in the last row, executing it within the last row (left-to-right). Simple examples of aggregate functions are the sum of all and the maximum of all. Other important operations that can be executed particularly well on the ISA are *row broadcast* (left-to-right), *column broadcast* (top-down) and *ring-shift operations*.

Row broadcast ($C := C[WEST]$): Each processor reads the value from its left neighbour. Since the execution of this operation is pipelined along the row, the same value is propagated from one communication register to the next, until it finally arrives at the rightmost processor. Note that the row broadcast requires only a single instruction.

Row ring-shift ($C := C[WEST]$; $C := C[EAST]$): The contents of the C-registers can be ring-shifted along the processor rows by two instructions. Every two horizontally adjacent processors exchange data (using one read left and one read right operation). Because of the instruction flow from west to east, this implements a ring-shift. Of course, a column ring-shift can be executed in the same way.

4. Architecture of the Systola 1024

The parallel computer Systola 1024 is a low cost add-on board for standard PCs. The ISA on the board is integrated on a 4×4 array of processor chips. Each chip contains 64 processors, arranged as an 8×8 square. This provides 1024 processors on the board. In order to exploit the computation capabilities of this unit, it is necessary to provide data and control information at an extremely high speed. Therefore, a cascaded memory concept is implemented on board that forms a fast input and output environment for the parallel processing unit. For the fast data exchange with the processor array there are rows of intelligent memory units at the northern and western borders of the array. These units are called *interface processors* (IPs). Eight IPs are integrated on one chip. Each IP is connected to its adjacent array processor by a single wire for data transfer in each direction. The IPs have access to an on-board memory by means of special fast data channels, those at the northern interface chips with the northern board RAM, and those of the western chips with the

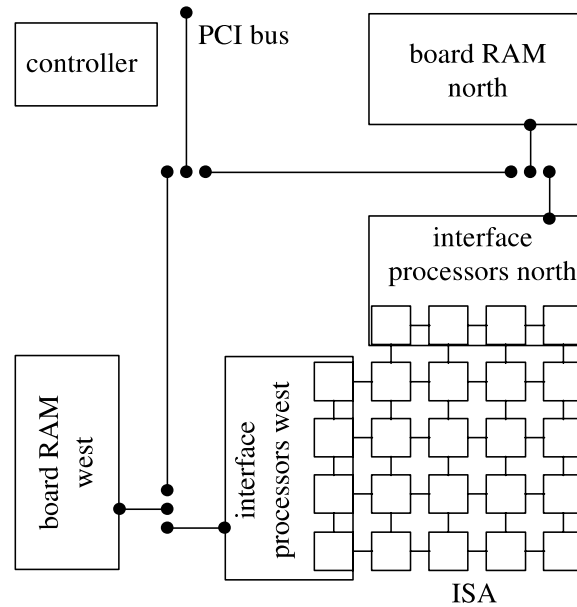


Fig. 3. Data paths in the parallel computer.

western board RAM. The northern and the western board RAM can communicate bi-directionally with the PC memory over the PCI bus. The data transfer between every two memory units within this hierarchy is controlled by an on-board controller chip (see Fig. 3). Due to the limited chip area each processor has to be very compact. In particular, it implies a bit serial data organization. The size of the local memory is limited to 32 internal registers plus a C-register. The word-length of the data items is 16 bits. The instruction set and the internal data formats were defined such that arbitrary formats for integer and floating point arithmetic could be handled.

5. Hybrid-parallelization mapping for volume data

Basically, parallelization mapping refers to the task of distributing volume data into the required parallel architecture system's resources [8–10]. Parallelization mapping for hybrid system differs slightly from the traditional parallel model such as the MIMD or SIMD in that it is a more layered mapping. There are two layers of mapping required for hybrid system and we termed these as *coarse parallelization mapping* (CPM) (Fig. 4) and *fine parallelization mapping* (FPM). The purpose of coarse and fine parallelization is essentially for each of these mapping tasks to apportion the data segment into suitable structures for loading into the MIMD and then to SIMD, respectively. In coarse parallel algorithm, there are three possible structures available, namely slabs, shafts and blocks.

Selection of CPM will be determined by the nature of the application algorithm. Each slab, shaft or block is in fact a series of segmented data slices. The distance be-

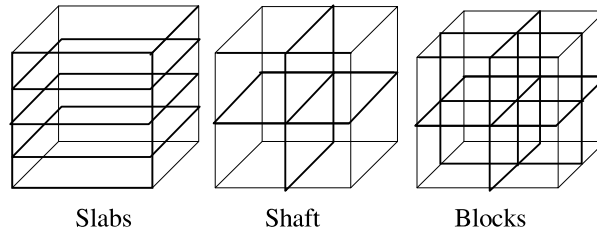


Fig. 4. Coarse parallelization mapping.

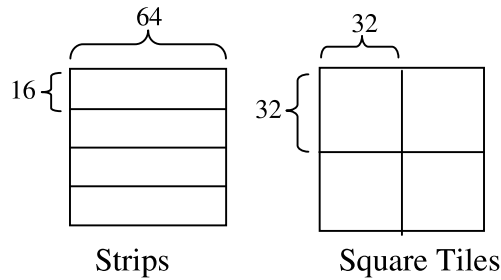


Fig. 5. Fine parallelization mapping.

tween two adjacent slices is a voxel distance apart. In the finer aspects of parallelization mapping (FPM), two possible structures exist, namely strips and square tiles. Each strip or square tile is a representation of a segment of an image slice from CPM. For example, in a 64^3 volume data, the data for each slice of image is 64×64 pixels and it can be loaded into the Systola 1024 using either of the two ways shown in Fig. 5. Conceptually, there are 3×2 or a total of six different possible combinations of hybrid-parallelization mapping available for this new form of architecture.

In our visualization application i.e. volume rendering application using the Ray-Casting approach, we need to composite all of the data slices for the final computation of the pixel's intensity. Also in the rendering pipeline, there is a need for acquiring six adjacent neighbouring voxel values for computation and a symmetrical loading structure would be preferred. Hence the slab for coarse parallelization into the MIMD and square tiling has been chosen as the fine parallelization mapping into the SIMD.

6. SIMD contribution to hybrid computing

Like all parallel architectures, it would be useful to establish the connection between the contribution of the SIMD to the hybrid system. Determining the hybrid speedup formula would help us to predict the expected performance of the cluster even before investment is poured in.

6.1. Establishing the hybrid speedup

The speedup of parallel processing is often determined by using the ‘Before and After’ approach. In other words, speedup is a measure of the processing time before implementing parallel processing over the processing time after parallel processing. In MIMD architecture and applying Gustafson’s law [4], speedup can be expressed as:

$$\text{Speedup} = \frac{T_s + N(T_p)}{T_s + T_p} \quad (1)$$

where: T_s is serial processing time; T_p is parallel processing time; N is number of MIMD processors.

In the context of the hybrid system, the processing time T_p can be again further subdivided into its serial and parallel processing components for its incorporation of the SIMD systems. Assuming that each SIMD system is homogenous to the MIMD machine, i.e. has the same processing speed and completes the required SIMD task in one pass, Eq. (1) can be re-written as:

$$\text{Speedup}_{\text{Hybrid}} = \frac{T_s + N(T_1 + PT_2)}{T_s + T_1 + T_2} = \frac{T_s + NT_1 + NPT_2}{T_s + T_1 + T_2} \quad (2)$$

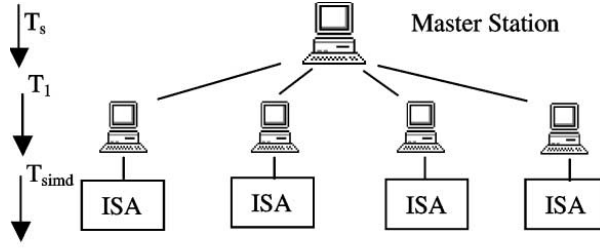
where: T_1 is serial processing time for SIMD; T_2 is parallel processing time for SIMD at MIMD speed; P is number of SIMD processors.

This, we do by substituting T_p process in the numerator by $T_1 + PT_2$. The denominator, which is the total processing time of hybrid system, is the summation of the serial of the SIMD and MIMD and the parallel times in the SIMD (i.e. $T_s + T_1 + T_2$).

The speedup equation (2), however, has assumed that both SIMD and MIMD have similar processing performance. This is an unlikely scenario. In our *Hybrid 4PE* model for instance, the SIMD is the Systola 1024 and the MIMD is the PC running Windows NT and both have different level of performance efficiency. The SIMD performance efficiency is also largely dependent on the SIMD processing speed as well as the code portability. This is because in SIMD systems, code re-generation is required as most SIMD architectures require programming of codes in low-level language. The *conversion ratio*, X , is the term which caters for the difference in the SIMD and MIMD processing speed as well as for code transformation from MIMD to SIMD. We can measure the conversion ratio, by taking the time for the MIMD system to run the same code that the SIMD takes to run. On careful analysis, we can ascertain that the time taken for processing T_2 is in fact equal to the product of the total SIMD processing time and the conversion ratio. We can equate the following:

$$T_2 = XT_{\text{simd}} \quad (3)$$

where: conversion ratio, X is ratio of processing time for MIMD to run code segment over processing time for SIMD to run same code segment; T_{simd} is total processing time in SIMD.



$$\text{Total processing Time} = T_s + T_1 + T_{\text{simd}}$$

Fig. 6. Hybrid system processing time.

The total time that the program takes to run using the hybrid system is equal to the serial time of both the master station and the MIMD system and the parallel time in the SIMD (refer to Fig. 6). From Eq. (2), we can now re-write the speedup equation for any hybrid systems as:

$$\text{Speedup}_{\text{Hybrid}} = \frac{T_s + NT_1 + NP(XT_{\text{simd}})}{T_s + T_1 + T_{\text{simd}}} \quad (4)$$

Conversion factor ratio, X and T_{simd} will balance each other so that the numerator always remains a constant. This is rightly so as the numerator represents the processing time prior to any parallelization and this value is independent of any SIMD processing speed. Any change in the SIMD processing speed should only affect the denominator. Effectively, by simple algebra, we can re-write Eq. (4) as follows:

$$= N + \frac{(1 - N)T_s}{T_1 + T_{\text{simd}} + T_s} + \frac{(PX - 1)(NT_{\text{simd}})}{T_1 + T_{\text{simd}} + T_s} \quad (5)$$

As a guiding principle, we can check the correctness of Eq. (4) by reverting it back as an MIMD system. In this, we make T_{simd} equals zero (i.e. no SIMD) and T_1 approaches T_p . The result of the hybrid speedup then becomes:

$$\lim_{T_1 \rightarrow T_p} \left(N + \frac{(1 - N)T_s}{T_1 + T_{\text{simd}} + T_s} + \frac{(PX - 1)(NT_{\text{simd}})}{T_1 + T_{\text{simd}} + T_s} \right) = N + \frac{(1 - N)T_s}{T_s + T_p} \quad (6)$$

As expected, this is consistent with Gustafson's law.

6.2. Effect of SIMD processing speed on hybrid speedup

Having established the hybrid speedup equation, we would like to determine the upper and lower bounds of the above equation. For a fixed problem size of a particular application, T_s and T_1 should remain constant and the only parameter that is likely to change is the T_{simd} i.e. varying the SIMD speed. Hence, the lower limit of the equation should be when T_{simd} is very large i.e. very slow SIMD system. The equation becomes:

$$\lim_{T_{\text{simd}} \rightarrow \infty} \left(N + \frac{(1-N)T_s}{T_1 + T_{\text{simd}} + T_s} + \frac{(PX-1)(NT_{\text{simd}})}{T_1 + T_{\text{simd}} + T_s} \right) = N + 0 + (PX-1)N = PXN \quad (7)$$

Conversely, the maximum limit of any hybrid speedup is achieved when the T_{simd} is very small i.e. very fast SIMD processing speed such that processing time approaches zero while the values of T_s and T_1 remains unchanged. This will therefore result in the speedup as:

$$\lim_{T_{\text{simd}} \rightarrow 0} \left(N + \frac{(1-N)T_s}{T_1 + T_{\text{simd}} + T_s} + \frac{(PX-1)(NT_{\text{simd}})}{T_1 + T_{\text{simd}} + T_s} \right) = N + \frac{(1-N)T_s}{T_s + T_1} \quad (8)$$

If we were to sketch the graph of speedup versus SIMD processing represented by T_{simd} , it can be ascertained that the sketch of the graph for a given SIMD is an ‘S’ curve (see Fig. 7). The SIMD speed and T_{simd} are inverse to one another in that as the SIMD processing speed gets faster, T_{simd} gets smaller. The asymptote indicates the maximum limit of any hybrid system’s speedup.

Data loading and unloading time from the MIMD to SIMD and vice versa is a time to be taken into account in T_{simd} i.e. the processing time in SIMD. Similarly the data loading from the Master Station to the MIMD system is a time to be incorporated in the serial part of MIMD, T_1 . From the graph sketch and Eqs. (7) and (8) formulated above, we can derive at the following two rules:

Rule 1: For a fixed problem size, increasing the SIMD processing speed increases the speedup of the hybrid system.

Rule 2: For a fixed problem size, the range of hybrid system speedup is from PXN to

$$N + \frac{(1-N)T_s}{T_s + T_1}$$

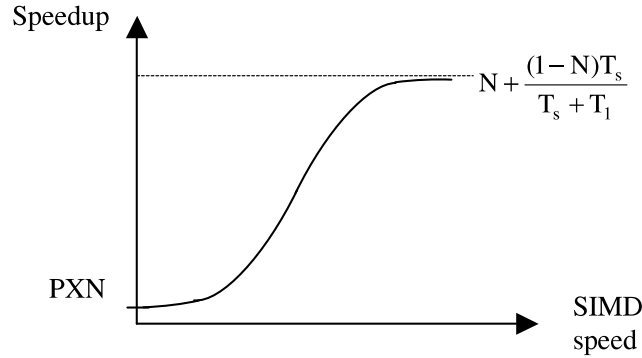


Fig. 7. Sketch of speedup vs SIMD.

6.3. Effects of increasing the problem size

In many practical applications, the problem size grows. When the increase in problem size can be formulated into a data parallel or function parallel problem, such an increase can be distributed across the various nodes for fine-grain processing in the SIMD or processed in the MIMD nodes respectively.

In the case of data parallel problems, speedup for these applications can be expected to increase if these additional data can be distributed evenly and each processing element performs the same amount of computation on them i.e. we need to divide the problem size equally and make the PE busy all the time. In such a case, the proportion of parallel task for the SIMD increases. The drawback of this increased problem size is that it leads to an unfortunate increase in data loading time and hence increases the communication time. However, with applications having favorable ratio of the parallel computation task to the data communication task, better speedup is expected. In the same manner, if the problem size growth can be formulated to function in parallel, this growth can then be parallelized to improve its speedup. In volume visualization applications, the problem size can grow in terms of the volume data, which is a data parallel problem. It is therefore anticipated that the speedup of the task will increase when a larger problem size is being processed.

6.4. Increasing of array processors

An important assessment consideration for any SIMD system is its data loading/unloading time. In fact in many applications, there is a requirement for each SIMD to run many times before completing its required task. Each time the SIMD is called to process, data loading/unloading is required. Effectively, T_{simd} is made up of two components, data loading/unloading time (T_{rw}) and execution time (T_{exe}) i.e.

$$T_{\text{simd}} = T_{\text{rw}} + T_{\text{exe}} \quad (9)$$

Expressing T_{rw} and T_{exe} as a factor (R) of the SIMD processing time.

$$T_{\text{rw}} = RT_{\text{simd}}$$

$$T_{\text{exe}} = (1 - R)T_{\text{simd}}$$

Increasing the number of array processors basically has two implications to the SIMD. One, it is essentially to reduce the number of calls to SIMD since more processors means more data can be computed at any one time and secondly, it also reduces the total SIMD processing time. Taking the increase of the array processor by a factor of 4 as an example, (i.e. the $n \times n$ array of processors of the Systola is now increased to $2n \times 2n$), the increase in either the number of rows or columns is only by 2 times. Hence, it can be said that though the T_{simd} processing time decreases by a factor of 4 (i.e. 2×2), the data loading increases by only a factor 2. From Eq. (9), we can express this new SIMD processing time, T'_{simd} in terms of the original SIMD processing time as:

$$T'_{\text{simd}} = 2RT_{\text{simd}} + \frac{(1-R)T_{\text{simd}}}{4} = \frac{T_{\text{simd}} + 7RT_{\text{simd}}}{4} \quad (10)$$

For any improvement in SIMD processing performance, this means that:

$$T_{\text{simd}} > T'_{\text{simd}} \quad (11)$$

$$T_{\text{simd}} > 2RT_{\text{simd}} + \frac{(1-R)T_{\text{simd}}}{4} \Rightarrow R < \frac{3}{7} \quad (12)$$

This means that for any data loading/unloading time that is less than 3/7 of the T_{simd} time, a 2 times increase in the number of rows and column will result in the SIMD processing time to be reduced. From Rule 1 above, this also translates to mean that the speedup will also increase. Reformulating Eq. (10) for the general case, we get:

$$T_{\text{simd}} > nRT_{\text{simd}} + \frac{(1-R)T_{\text{simd}}}{n^2} \quad (13)$$

$$R < \frac{n^2 - 1}{n^3 - 1} \quad (14)$$

where n : factor increase in both rows and columns of array processor.

7. Performance results

We have implemented and evaluated the hybrid system architecture as described and have made use of it to process a volume rendering application. The result of the performance has been encouraging. Using the volume rendering routines, we have managed to measure the conversion ratio X , of the Systola board and the T_{simd} for our visualization. As the value of X can be very small, we find the value of P times of X (i.e. PX) instead by running the code for P number of times. This, we found to be 4.0. We have used C++ routines to accurately time our models for a 256^3 data of a human spine data and a 64^3 of a symbol structure model. Table 1 depicts the rendering time for various configurations using different number of PCs in the hybrid cluster for two different models. The measured speedup can be easily obtained by dividing the processing time of the application that does not have any parallelization

Table 1
Performance results for volume rendering using MIMD-SIMD hybrid system

Model	Cluster size with SIMD				PC w/o SIMD
	1 PC	2 PC	3 PC	4 PC	
<i>Processing time (s)</i>					
Spine	18.61	12.67	7.36	5.65	42.0
Symbol	0.34	0.23	0.22	0.15	0.78
Speedup (spine)	2.25	3.31	5.71	7.43	1.00
Speedup (symbol)	2.29	3.39	3.55	5.20	1.00

with the various clusters having different number of PCs. In each of the PCs inside the MIMD nodes, the Systola 1024 has been added to function as the SIMD.

Table 2 shows its corresponding speedup using the usual MIMD system (i.e. each MIMD PC does not have a SIMD card). Evidently, the speedup performance of the hybrid system surpasses that of the MIMD system alone. This is due to the contributing factor of the SIMD in the MIMD. It is pertinent to note the conversion ratio will differ from application to application. The reason being that the performance efficiency of most SIMD systems improves when application uses integer computation as opposed to floating-point computation. Also, one of the strengths of our SIMD (Systola 1024) is in performing aggregate functions. Such functions when widely implemented in the algorithm will yield high performance results as compared to those which do not.

Table 2
Performance results for volume rendering without SIMD in the MIMD system

Model	Cluster size without SIMD			
	1 PC	2 PC	3 PC	4 PC
<i>Processing time (s)</i>				
Spine	42.0	28.1	16.47	12.73
Symbol	0.78	0.53	0.47	0.30
Speedup (spine)	1.00	1.50	2.55	3.30
Speedup (symbol)	1.00	1.47	1.66	2.60

Table 3
Comparison between the measured and predicted speedup (spine) (Panel A); (symbol) (Panel B)

Parameters	Cluster size			
	4 PC	3 PC	2 PC	1 PC
<i>Panel A</i>				
<i>Computed hybrid speedup (spine)</i>				
T_s	0.76	0.78	3.23	0.01
T_l	3.41	4.37	6.31	12.22
T_{simd}	1.55	2.04	3.10	6.21
Predicted speedup	6.86	5.34	3.22	2.01
Measured speedup	7.43	5.71	3.31	2.25
<i>Panel B</i>				
<i>Computed hybrid speedup (symbol)</i>				
T_s	0.01	0.01	0.01	0.00
T_l	0.10	0.18	0.13	0.24
T_{simd}	0.024	0.024	0.049	0.098
Predicted speedup	5.92	3.91	3.50	1.87
Measured speedup	5.20	3.55	3.39	2.29

See Fig. 8(a) and (b).

Special attempts have been made to ensure that there is even load balance achieved in each of the nodes. The load balance technique used is the *Slice Orientation* algorithm [16]. The algorithm basically orientates the image slices to be either vertical or horizontal depending on the viewing angle. If the viewing angle is less than 45° , the slices are orientated horizontally, otherwise, they are orientated vertically. In so doing, we are able to achieve a balanced number of voxels per slice for loading into each of the SIMD. In the volume rendering algorithm, larger volume data tends to exhibit higher speedups. The increased in volume data are being distributed over the nodes into SIMD for fine-grain processing and as expected, the speedup increases.

Table 3 (see Fig. 8(a) and (b)) compares the actual measured speedup of two models (spine and symbol) with its predicted speedup. The predicted speedup was obtained using the hybrid speedup equation that was derived earlier. We found the two sets of values to be relatively close. The discrepancies are likely to be introduced due to the measuring of processing time as well as from the truncation errors. These discrepancies tend to be slightly larger if the measured time is small.

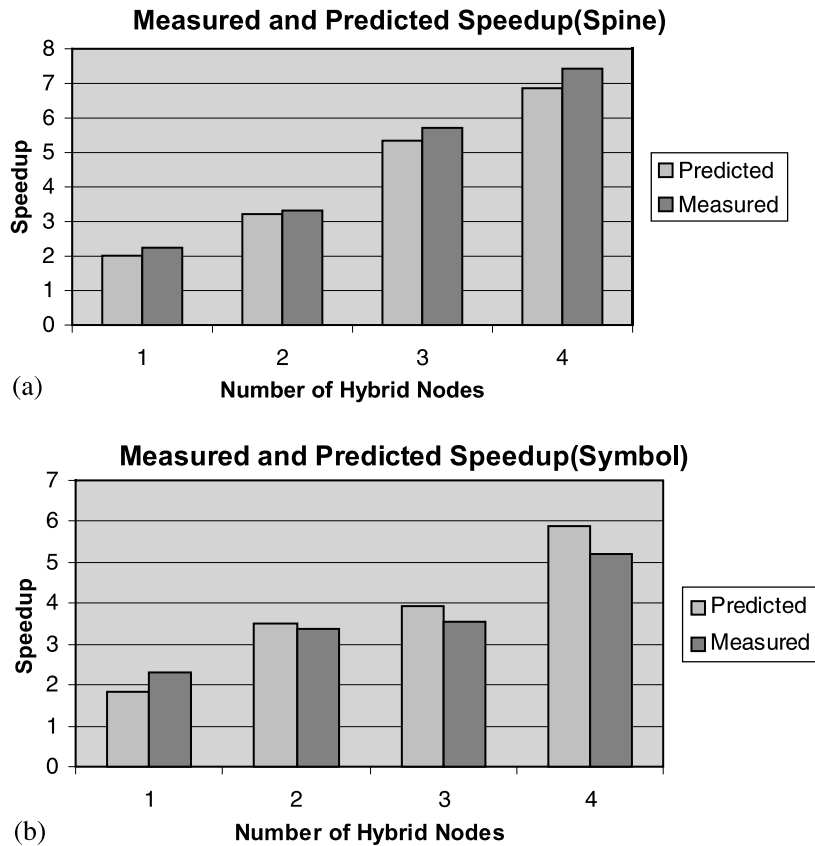


Fig. 8. Comparison between the measured and predicted speedup (spine) (a); (symbol) (b).

8. Conclusion

We have introduced a new type of parallel architecture which we called MIMD–SIMD hybrid system consisting of both SIMD and MIMD parallel systems functioning at the same architecture. The hybrid system, in general, is able to achieve speedup higher than what its corresponding MIMD system can achieve. Not only is the Hybrid 4PE scalable, it is also a low-cost system using ‘Off-The-Shelf’ components and without any proprietary or dedicated hardware. In the Hybrid 4PE, communications among stations within the cluster are accomplished using a MPI. The SIMD system that we are using is the Systola 1024 which uses an interesting instruction systolic array concept. We have introduced the Systola 1024 architecture and its instruction systolic array concept. Our current Systola is in a prototype stage and there are plans to enhance it further. With the increased performance of this SIMD or any other SIMD system, it can be expected that the performance speedup of the hybrid system can increase by a significantly larger magnitude. This we can predict by using our derived formula for the contribution of the SIMD on the hybrid system. Using the speedup formula and comparing it with the visualization application that we have developed on the hybrid system, we have found the prediction to be relatively accurate. With known parameters, this formula is able to compute the speedup for any hybrid system.

References

- [1] C.G. Cameron, P.E. Underill, Rendering volumetric medical image data on a SIMD architecture computer, in: *SPIE, Visualization in Biomedical Computing*, vol. 1808, 1992, pp. 137–142.
- [2] A. Dittrich, H. Schmeck, Given’s rotation on an instruction systolic array, in: *Proceedings Parcella’ 88*, LNCS 342, Springer Verlag, 1988, pp. 340–346.
- [3] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, L. Israel, A heterogeneous multiprocessor graphics system using processor enhanced memories, in: *Proceedings of SIGGRAPH 1989*, Computer Graphics, August 1989, pp. 65–74.
- [4] J.L. Gustafson, Reevaluating Amdahl’s Law, *CACM* 31 (5) (1988) 532–533.
- [5] D. Hillis, *The Connection Machine*, MIT Press, 1985.
- [6] B. Lichtendbelt, R. Crane, S. Naqvi, *Introduction to Volume Rendering*, Prentice Hall, 1998.
- [7] L. Misericordia, C. Montani, R. Perego, R. Scopigno, Parallel volume visualization on a hypercube architecture, in: *Proceedings of 1992 Workshop on Volume Visualization*, October 1992, pp. 9–16.
- [8] U. Neumann, Interactive volume rendering on a multi-computer, in: *Proceedings of the ACM 1992 Symposium on Interactive 3D Graphics*, March 1992, pp. 87–93.
- [9] U. Neumann, Communication cost for parallel volume rendering algorithms, *IEEE Computer Graphics and Applications* 14 (4) (1994) 49–58.
- [10] U. Neumann, Parallel-volume rendering algorithm performance on mesh-connected multicomputers, in: *Parallel Rendering Symposium Proceedings*, ACM Press, 1993, pp. 97–104.
- [11] H. Pfister, A. Kaufman, Cube-4—A scalable architecture for real-time volume rendering, in: *Proceedings of the 1996 Symposium on Volume Rendering*, 1996, pp. 47–54.
- [12] K. Sano, H. Kitajima, H. Kobayashi, T. Nakamura, Parallel processing of the shear warp factorization with the binary-warp method on a distributed-memory multiprocessor system, in: *Proceedings of the IEEE Symposium on Parallel Rendering*, 1997, pp. 87ff.
- [13] M. Schimmmler, H.W. Lang, The instruction systolic array in image processing applications, in: *Proceedings Europto 96*, SPIE 2748, 1996, pp. 136–144.

- [14] B. Schmidh, M. Schimmler, H. Schroder, Morphological Hough transform on the instruction systolic array, in: *Proceedings EuroPar 97*, LNCS 1300, Springer Verlag, 1997, pp. 798–806.
- [15] H. Schroder, Instruction Systolic Array—Tradeoff between flexibility and speed, *Computer Systems Science and Engineering* (1988) 83–91.
- [16] C.S. Leo, B. Schmidt, H. Schroder, Parallel volume rendering using slice orientation, in: *Proceedings of International Conference on Information, Communications and Signal Processing (ICICS)*, Singapore, 2001.