

Dynamic Reconfiguration of a PMMLA for High-Throughput Applications

Gautam Ghare and Soo-Young Lee

Department of Electrical Engineering, Auburn University, Auburn, AL 36849.
sylee@eng.auburn.edu

Abstract. High throughput and dynamic reconfigurability are required in many tasks, especially real-time applications. A logical structure of parallel computing system for such applications is a pipeline of multiprocessor modules in form of a linear array (PMMLA). This paper proposes a scheme to dynamically reconfigure the system by varying the number of processors in each stage of the PMMLA to maintain load balancing among stages.

1 Introduction

In many applications, including signal and image processing, computer graphics, virtual reality, particle simulation, system control, etc. [1][2] the computational task can be linearly partitioned into subtasks that are pipelined. These tasks have the characteristic that a large volume of data is involved. In addition, they exhibit high degree of data parallelism in each stage of computing. Exploitation of data parallelism in each stage requires local results to be integrated from PEs and distributed over PEs in the next stage. The parallel computing system (model) PMMLA (*Pipelined Multiprocessor Modules based on Linear Array*) has been shown to achieve comparable or even better performance than any other interconnection topologies for such applications [1].

Most of these applications are *dynamic* in the sense that the computation time required at each stage changes with the input data. This change in computation requirements results in load imbalance between stages of a pipeline, resulting in performance degradation. We are interested in developing a reconfigurable multi-processor system that can provide both high throughput and dynamic reconfigurability for a class of applications. This paper describes an algorithm to dynamically reconfigure the PMMLA to provide load balancing between stages.

2 Reconfigurable PMMLA

2.1 Reconfigurable PMMLA Architecture

The PMMLA architecture was described in [1]. It is a pipeline of multiprocessor modules based on linear array (PMMLA). A PMMLA achieves high throughput with minimal hardware, i.e., only 3 links per PE, and therefore allows a compact high performance multiprocessor system. Also, it is scalable, i.e., the hardware complexity is linearly proportional to the total number of PEs in a PMMLA. It should be emphasized that it is easy to realize a PMMLA using modularized units (commercial or custom-designed), i.e., PEs with 3 links, for a varying problem size.

All PEs in a reconfigurable PMMLA have three links, two unidirectional links for the linear array connection and a bidirectional (but not full duplex) link to

be connected to the multiple buses, as illustrated in Figure 1. The number of multiple buses (B) is equal to the maximum number of pipeline stages, that need to be accommodated. The third link of each PE can be connected to any bus via a bidirectional ($1 \times B$) switch (or simply a mux/demux pair) and the connection can be set dynamically. Given a number of stages (rings), the linear array is partitioned into P subarrays and two end PEs in each subarray establish their connection to the corresponding bus to form a ring as illustrated in Figure 1 (where there are 3 PEs, 2 PE and 3 PEs in the first, second, and third stages, respectively).

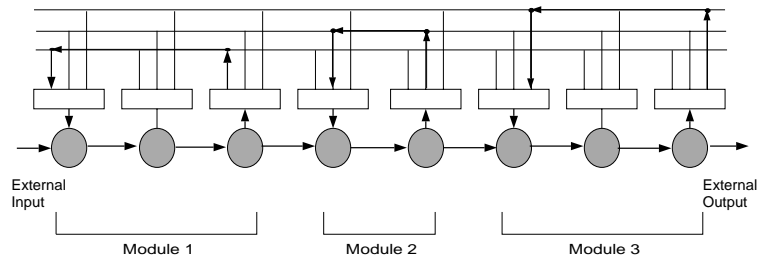


Fig. 1. A reconfigurable PMMLA

It should be noted that a bus in a reconfigurable PMMLA is to be distinguished from the conventional “bus” in that no (bus) contention occurs in the former. It is simply a set of switches. Once the switches are set, there is only one sender on each bus at any time. Therefore, no complicated hardware/software is required for bus control, which also contributes to a compact system.

P stages of a PMMLA process their respective data independently. PEs in each stage produce local results to be integrated and transferred to the next stage. Integration or collection of local results in a stage is carried out through the corresponding ring. Transfer of the integrated or collected results to the next stage (and distribution in next stage) is done along the linear subarrays of the two adjacent stages. Local integrated results in PEs of a stage are transferred through PEs in the two successive stages such that each local result in a stage goes through all PEs in the following stage. During this transfer, every PE in the following stage takes whatever portion of the integrated result it needs. All PEs in each stage are synchronized in the beginning of the communication phase (integration) and before the transfer takes place.

3 Reconfiguration

The following conventions are adopted to describe the reconfiguration of the PMMLA :

- P : The number of stages in a PMMLA
- S_i : The i th stage of PMMLA for $i = 0, 1, 2, \dots, P - 1$
- N : The total number of PEs in a PMMLA
- n_i : The number of PEs in S_i , where $N = \sum_{i=0}^{P-1} n_i$
- m_i : Size of data to be shared by PEs in S_i
- n'_i : Number of PEs in Stage i after reconfiguration.

$PE_{i,j}$: The j^{th} PE in Stage i for $j = 0, 1, \dots, n_i - 1$, before reconfiguration.
 $PE_{i',j'}$: The j'^{th} PE in Stage i' for $j' = 0, 1, \dots, n'_i - 1$ after reconfiguration.
 T_{P_i} : The time required for computation at stage i (parallel execution time).
 T_{I_i} : The time required for integration of the local results at stage i .
 T_{c_i} : The time required for sequential computation at stage i .

The term *configuration* refers to the set $(n_0, n_1, \dots, n_{P-1})$ and the term *new configuration* to the set $(n'_0, n'_1, \dots, n'_{P-1})$. In this paper, it is assumed that the number of stages, P , does not change.

In a pipelined system such as a PMMLA, the throughput depends on the maximum processing time taken by any stage [1]. In a dynamic problem, the time taken by each stage varies with input data (which varies with time). High throughput can be maintained by dynamically balancing the load over all stages of a PMMLA. The goal of dynamic reconfiguration is to perform this modification of the number of PEs in each stage with minimum overhead (in terms of performance, hardware cost, etc.). The overhead of reconfiguration should not offset the performance gain by reconfiguration. The task of reconfiguration generally consists of two steps : (i) To determine when to reconfigure and the new configuration (ii) to perform the reconfiguration. Step (i) is carried out for each frame and step (ii) is performed only when reconfiguration is to be done.

3.1 Determining reconfiguration

In order to determine the new configuration, a *reconfiguration message* consisting of *Change Status* bit, T_{P_i} and T_{I_i} (measured by each PE) is passed along the pipeline ultimately to the last PE of the PMMLA ($PE_{P-1, n_{P-1}-1}$). The last PE ($PE_{P-1, n_{P-1}-1}$) computes the sequential execution time T_{c_i} at each stage as $T_{c_i} = n_i \times T_{P_i} + T_{I_i}$. Then, the number of PEs desirable in each stage $(n'_0, n'_1, \dots, n'_{P-1})$, is determined by the ratio $(T_{c_0} : T_{c_1} : \dots : T_{c_{P-1}})$ of the sequential execution times of stages.

If no change is desired (i.e., $(T_{c_0} : T_{c_1} : \dots : T_{c_{P-1}}) = (n_0 : n_1 : \dots : n_{P-1})$), a message with the *Change Status* reset is passed to the first PE ($PE_{0,0}$) which is circulated through all PEs. Otherwise (if the desired configuration differs from the present configuration), it is checked if the performance gain by reconfiguration is worth the overhead to be incurred, i.e., whether $(\max_i \{ \frac{T_{c_i}}{n_i} \} - \max_i \{ \frac{T_{c_i}}{n'_i} \}) > Cost\ Threshold$. Reconfiguration is performed only if the gain exceeds the cost threshold. This is necessary to avoid reconfiguration for each of very gradual changes where the reconfiguration overhead incurred will exceed the performance gain.

To perform reconfiguration, a message containing the new configuration with the *Change Status* set is passed to the first PE ($PE_{0,0}$). The reconfiguration message is transferred along the pipeline, to inform all PEs. PEs in stage S_i reconfigure after processing $(P - i - 1)$ frames of data after receiving the message with the *Change Status* bit set. This ensures that all stages and hence all PEs reconfigure approximately at the same time. All the messages required for reconfiguration are piggy-backed on the data being transferred to minimize the communication overhead.

3.2 Performing reconfiguration

Performing reconfiguration requires changing the number of PEs in each stage. The architecture of the PMMLA allows us to easily modify the configuration dynamically. This basically involves establishing a new feedback connection between two end (first and last) PEs in each stage to form a ring. Also, the data presently in the pipeline has to be handled effectively.

Fast Reconfiguration The fast reconfiguration algorithm is designed to avoid the overhead of flushing/stalling of the pipeline.

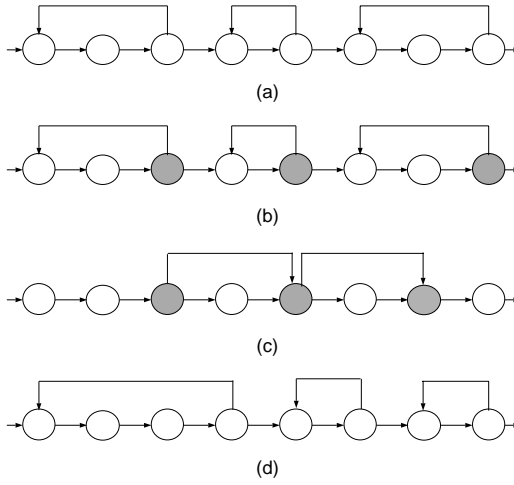


Fig. 2. An illustration of the fast reconfiguration algorithm for $N = 8$, $S = 3$. (a) The configuration before performing reconfiguration. (b) Current local data are collected to the last PE of the stage (PE_{i,n_i-1}). (c) The collected data is transferred to the first PE of the next stage after reconfiguration ($PE_{i',0}$). (d) The new configuration is set and the normal operation resumes.

The fast reconfiguration algorithm is illustrated in Figure 2. After completion of the integration phase [1], the data processed by the PEs in each stage (S_i) are collected to PE_{i,n_i-1} , the last PE of that stage as shaded in Figure 2(b). This is done in $n_i - 1$ transfers of data size $\frac{m_i}{n_i}$. The reconfiguration of the system is done after collecting the data at the last PE in each stage. PE_{i,n_i-1} , the PE which collected data of stage S_i , now transfers the collected data to $PE_{i+1,0}$, the first PE of the next stage after reconfiguration as shown in the Figure 2(c). This is done using the bus connection and requires only one transfer. The first PE ($PE_{i',0}$) of each stage in the new configuration ($S_{i'}$) then broadcasts it to all PEs in that stage in $n'_{i'+1} - 1$ steps.

Performance Analysis When reconfiguration is performed, $(n_i - 1)$ transfers of data of size $\frac{m_i}{n_i}$ are required to collect data at the last PE in the stage S_i .

Transferring the collected data from the last PE of the previous stage (S_i) to the first PE of the next stage S_{i+1} requires one transfer of data of size m_i . Finally, $n'_{i+1} - 1$ transfers of data of size m_i are required to broadcast the data in the stage S_{i+1} .

Hence, the total number of transfers required is $n_i + n'_{i+1} - 1$. This is equal to the number of transfers required in the normal operation. However, it should be noted that unlike during the normal data transfer phase, where every transfer involves data of size $\frac{m_i}{n_i}$, the reconfiguration phase requires n'_{i+1} transfers of data of size m_i . Hence, the reconfiguration overhead at stage S_i amounts to an additional transfer of data of size $(m_i - \frac{m_i}{n_i}) \cdot n'_{i+1}$. Since the reconfiguration is performed in all stages in parallel, the effective overhead for reconfiguration is the time taken to transfer data of size $\max_i \{ \frac{m_i \cdot (n_i - 1) \cdot n'_{i+1}}{n_i} \}$.

4 Experimental Results and Discussion

The performance of reconfiguration algorithm has been analyzed for dynamic visualization via emulation on nCUBE/2 with 32 PEs. In our implementation [1], the task was functionally decomposed into three stages: Transformation Stage, Clip Stage and Fill Stage. Object decomposition was used to distribute the load among the PEs within a stage [2]. The scene used for emulation shows an intersection of two streets, with a row of buildings on both sides. The results reported are from the five views shown in Figure 3.

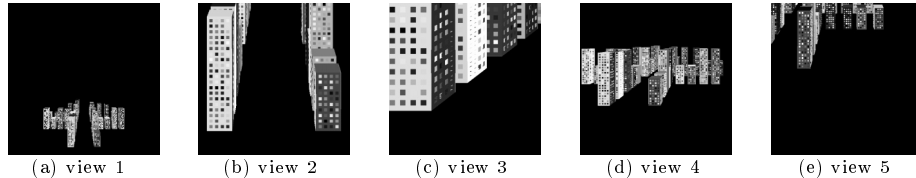


Fig. 3. The views used to measure performance

The number of polygons in each view ranged from 2211 to 8856. The output was a pix map image of size 480×480 , with each pixel having up to 256 gray levels. In this set of views, the computational load distribution over stages varies with the view.

A sequence of frames of each type (view) were input to the reconfigurable PMMLA, in the order of view 1 through view 5. The initial configuration was $(N_{trans}, N_{clip}, N_{fill}) = (2, 5, 9)$. N_{trans} , N_{clip} , N_{fill} are the number of PEs in the transformation, clip and fill stages, respectively. The results are summarized in Table 1. Column 1 shows the average time per frame in steady state, after reconfiguration. Column 2 shows the average time per frame during the transient period (i.e, during reconfiguration) including the reconfiguration overhead. The average time per frame with the configuration always equal to $(2, 5, 9)$ (i.e., without reconfiguration) is shown in column 3. The performance gain shown in column 4 indicates the percentage improvement in average time per output in steady state after reconfiguration w.r.t. that without reconfiguration. Column 5 shows the new configuration attained after reconfiguration.

First of all, it can be seen in Table 1 that throughput (time per frame or output) is significantly improved by dynamically reconfiguring a PMMLA according to the computation time distribution over stages. It is also observed that even the transient throughput which includes the reconfiguration overhead is higher than that without reconfiguration. This improved performance is mainly due to the better load balance throughput stages, which is achieved by our fast and effective reconfiguration scheme.

View No	Time per frame (<i>sec</i>)			Gain (%)	New Configuration ($N_{trans} : N_{clip} : N_{fill}$)
	With Reconfiguration		Without Reconfiguration		
	Steady	Transient			
1	4.94	-	-	-	2:5:9
2	4.95	5.71	6.09	18.7	1:2:13
3	3.95	4.15	4.97	20.5	2:3:11
4	5.72	5.93	6.50	12.0	3:5:8
5	3.77	3.98	4.73	20.3	4:5:7

Table 1. Comparison of the throughput with and without reconfiguration in terms of average execution time per frame.

Another fact to be noted is that the communication overhead in an actual PMMLA is expected to be substantially lower. In this experiment, a PMMLA has been emulated on the nCUBE/2 where a message is in general to be “routed”. This routing requires overheads such as address checking, etc., even for a message to be sent from a PE to one of its adjacent PEs. Such an overhead is not incurred in the actual PMMLA and therefore a better performance (higher throughput) is expected.

5 Conclusions

In this paper, a dynamic reconfiguration algorithm for a multiprocessor system, PMMLA, has been proposed, and its performance has been demonstrated for dynamic visualization via emulation on the nCUBE/2. The following may be concluded.

- A PMMLA has a good potential to provide high throughput required in dynamic applications.
- Dynamic reconfiguration of a PMMLA (i.e., changing the number of PEs in each stage) is simple and fast.
- Dynamic reconfiguration achieves a significant performance improvement, even after accounting for the reconfiguration overhead, compared to the case where no reconfiguration is done.

References

1. S-Y Lee and Gautam Ghare, “Compact and Flexible Linear-Array-Based Implementations of A Pipeline of Multiprocessor Modules (PMMLA) for High Throughput Applications”, Proceedings of *IEEE International Conference on High Performance Computing (HiPC'97)*, pp. 296-301, 1997.
2. Stuart Green, “Parallel Processing for Computer Graphics”, *MIT Press*, 1991.

This article was processed using the \LaTeX macro package with LLNCS style