

## A Prediction based CMP Cache Migration Policy \*

Song Hao<sup>1</sup>, Zhihui Du<sup>1+</sup>, David Bader<sup>2</sup> and Man Wang<sup>1</sup>

<sup>1</sup> Tsinghua National Laboratory for Information Science and Technology

Department of Computer Science and Technology, Tsinghua University, 100084, Beijing, China

<sup>2</sup>College of Computing, Georgia Institute of Technology, Atlanta, GA, 30332, USA

<sup>+</sup>Corresponding Author's Email: [duzh@tsinghua.edu.cn](mailto:duzh@tsinghua.edu.cn)

### Abstract

*The large L2 cache's access latency, which is mainly caused by wire delay, is a critical problem to improve the performance of CMP (Chip Multi-Processor) in NUCA (Non-Uniform Cache Architecture). A CMP L2 cache accessing performance model is provided first to analyze and evaluate the L2 access efficiency in this paper. The total L2 cache access latency problem is formalized as an optimal problem and the lower bound of L2 cache access latency is given based on this model. A novel PBM (Prediction based L2 cache data Migration) algorithm, which employs the sequential prediction technology to identify the data to be accessed in the near future, is designed to migrate the data to be accessed toward their users in early and this method can enable the cores to perform their accesses to the L2 cache in close banks. The analysis results show that this active data migration algorithm can take advantage of the principle of locality to reduce the data access latency much more than the traditional lazy data migration policy. To evaluate the theoretic analysis results, the HMTT toolkit is used to capture the complete memory trace of the SPEC 2000 benchmark running on an SMP computer. The memory trace shows that our prediction technology can work well and at the same time, an L2 cache access simulator is developed to deal with the memory trace data. The simulation experiments show that both the shorter block transfer distance and the lower average access latency can be achieved in the PBM policy. The average block transfer distance can be reduced by up to 16.9%, and the average L2 access latency can be reduced by up to 8.4%.*

**Key words** Chip Multi-Processor, L2-cache, Non-Uniform Cache Architecture, Migration policy, sequential prediction

## 1. Introduction

### 1.1. Background

CMP (Chip multiprocessor) has become more and more important both in academic research and industrial applications. Small-scale CMPs, with two or four cores per

---

\* This paper is partly supported by National Natural Science Foundation of China (No. 60773148, No.60503039 and No.10778604), and China's National Fundamental Research 973 Program (No. 2004CB217903)

chip, are already commercially available [1, 2]. With the increase of transistor integration density, more and more cores will be integrated in a single chip [3, 4]. At the same time, on-chip memory hierarchies are summoning innovative designs.

Most of the CMPs available on the market use the private L1 cache structure. But the proposals for the organization of the on-chip L2 cache are different. A shared L2 cache architecture was widely used now, but with the growth of the number of cores and the size of L2 cache, the average L2 cache access latency is heavily influenced by the latency of accessing remote cache banks, which in turn is influenced by on-chip wire delays. Private L2 cache has the advantage that most L1 misses can be handled locally, which could reduce remote accesses, but this will result in relatively much more off-chip accesses than a shared L2 caches.

Wire delay plays a significant role in cache design. For example, in the 65-nm technology (2004), transmitting data 1 cm requires only 2-3 cycles, but in the 32-nm technology (which will be achieved around 2010 according to Moore's Law), this will necessitate over 12 cycles [5]. So the data in the shared cache should be placed close to the core(s) to minimize the access latency. This makes it difficult to provide uniform access latencies to all the shared cache banks.

A Non-Uniform Cache Architecture (NUCA) was provided in [6] by Kim et al, which allows nearer cache banks to have lower access latency than further banks. This idea could hide the wire delay effectively. And in [7], Kim et al. proposed the application of NUCA on CMP system. The main improvement is that, by allowing data block to move in both vertical and horizontal directions, each processor core can perform its accesses in nearer banks, which will reduce the access latency.

### 1.2. Migration policy of NUCA

The main design points of NUCA are as follows (as shown in figure 1):

- The L2 cache is divided into multiple banks which are organized as a 2-D matrix;
- An on-chip 2-D mesh network connects all the banks. Each processor core has an interface to the network, and it can access data in any bank;
- Data blocks can be moved among the banks in both vertical and horizontal directions.

Traditional data migration policy (called TMP below for short) of NUCA uses an improved LRU algorithm. That is, if a data block is accessed by a processor core, it will be moved into an adjacent bank which is nearer to the user. And to prevent migration conflicts, a two-bit saturated counter is embedded in each cache tag. When a block is accessed, the relevant counter increases, and if the counter is saturated, migration is started and the counter is reset.

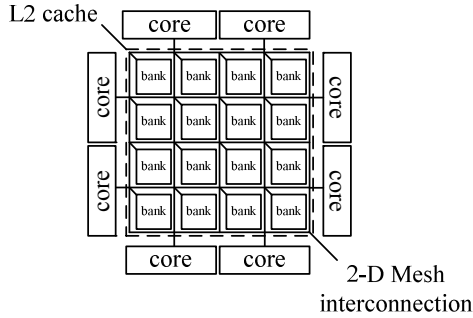


Figure 1. NUCA based CMP L2 cache

### 1.3. Prediction based migration

A lot of hardware prediction techniques have been developed for data prefetch [8, 9, 10, 16]. Generally they can be classified into sequential prefetch, related prefetch and stride prefetch. Next-line or one-block look-ahead prefetch is one kind of the sequential prefetch techniques. When cache miss occurs, it can prefetch not only the block which cache misses but also the next block. This method uses the spatial locality of the program. Related prefetch techniques use the history information of addresses to prefetch the data. But in this method a very large related prediction table must be set. Stride prefetch technique checks the memory access address to find the available stride. If the stride is available, the prefetch request is sent. This method can improve the precision of prediction.

Most prediction techniques mentioned above are used for data prefetch only when L2 miss occurs. That's because, in traditional cache architecture, data movement in cache only happens when L2 miss occurs and a block has to be replaced by another one. However, in NUCA architecture, data in L2 cache move not only when L2 miss occurs, but also when migration happens. So we propose that prediction techniques can also be used to design migration policy, which is called prediction based migration, or PBM for short. That is, when a block is accessed and migrated toward the processor core, the predicted block should also be migrated toward the same core. There are two advantages for the proposed policy:

(i) The PBM policy is very suitable for NUCA because it makes the assumption of NUCA become reality. NUCA can achieve high performance only when the processor core can perform its accesses to L2 cache in a near bank. The PBM policy can migrate the data toward its user before it is actually referenced.

(ii) The PBM policy does not need to add more hardware than traditional policy, because the pre-fetching circuit can also be used for migration prediction.

In section 3, we will give out a migration algorithm based on sequential prediction. The rest of the paper is organized as follows. In section 2 the CMP L2 cache access problem is formulated as an optimal problem. Section 3 gives the data migration algorithm based on the problem provided in section 2. Section 4 provides the experimental results. Related work is discussed in Section 5 and the conclusion is presented in section 6.

## 2. Description of the problem

### 2.1. Model of the problem

In order to analyze and evaluate different applications performance on NUCA based L2 cache, L2 Cache Accessing Performance ( $C_{L2AP}$ ) model is provided. It can be defined as a tuple of five elements:

$$\langle T_{Location}, T_{bank-to-network}, T_{wire}, T_{off-chip}, T_{contention} \rangle$$

in which:  $T_{Location}$  is the searching time to identify the location of a given memory address in L2 cache.  $T_{bank-to-network}$  is the time of accessing a given bank and put the referenced L2 cache block onto the network.  $T_{wire}$  is the time of transferring the given cache block on wire from source bank to target core.  $T_{off-chip}$  is the time to get missed data from off-chip memory to L2 cache.  $T_{contention}$  is the waiting time when the referenced bank is being accessed by another processor core.

Given an L2 Cache implementation, the value of  $T_{bank-to-network}$  and  $T_{off-chip}$  can't be changed. And the value of  $T_{Location}$  depends on the hardware implementation of the tag array. For centralized tag array  $T_{Location}$  can't be changed, and for distributed tag array  $T_{Location}$  depends on the locating algorithm [6, 17]. In our experiment, we simulated a centralized tag array, so  $T_{Location}$  is constant. The value of  $T_{contention}$  depends on when the previous access from another core to the same bank will finish. The value of  $T_{wire}$  is highly dependent on two factors: the distance from the processor core to the referenced cache bank, and the contention of the network-links. The first is the foremost factor to determine the  $T_{wire}$ , if the bank, which the cache block belongs to, is very close to the core, the transport path will be short, so that both the cache blocks' transport latency and the probability of network-links' contention caused by intersecting transport paths can be reduced. This is why it is critical to migrate data toward its user. The second factor is quite dependent on the router algorithm and in a NUCA-based L2 cache with the sharing degree of two (which has been proved most effective in [7]), the probability of network-links contention is quite small.

### 2.2. Formulation of the Problem

Based on the  $C_{1,2}AP$  model provided in section 2.1, it will take time  $T_{i,j}^{Single}$  to finish each L2 cache access for  $Core_i$  to get data from  $Bank_j$

$$T_{i,j}^{Single} = \begin{cases} T_{Location} + T_{bank-to-network} + T_{wire,i,j}, & \text{if hit and no contention} \\ T_{Location} + T_{contention} + T_{bank-to-network} + T_{wire,i,j}, & \text{if contention} \\ T_{Location} + T_{off-chip}, & \text{if miss} \end{cases}$$

For a given application or task, let  $R_i$  be the average L2 cache hit rate of  $Core_i$ . Let  $C_i$  be the average bank contention rate between  $Core_i$  and other processor cores sharing the same L2 cache. Let  $NA_{i,j}$  be the total number of L2 cache accesses to  $Block_j$  for  $Core_i$ . Let  $NA_i$  be the total number of L2 cache accesses for  $Core_i$ . Let NB be the total number of L2 cache blocks which has been accessed during the application's runtime, then  $NA_i = \sum_{j=1}^{NB} NA_{i,j}$ .

So for a given application or task running on  $Core_i$ , the total L2 cache access time  $T_i^{total}$  will be

$$\begin{aligned} T_i^{total} &= \sum_{j=1}^{NB} [(T_{Location} + T_{bank-to-network} + T_{wire,i,j}) \times NA_{i,j} \times R_i \\ &\quad + T_{contention} \times NA_{i,j} \times R_i \times C_i + NA_i \times (1 - R_i) \times (T_{Location} + T_{off-chip})] \\ &= NA_i \times T_{Location} + NA_i \times T_{bank-to-network} \times R_i \\ &\quad + \sum_{j=1}^{NB} NA_{i,j} \times (T_{wire,i,j} + T_{contention} \times C_i) \times R_i + NA_i \times (1 - R_i) \times T_{off-chip} \end{aligned}$$

To get the optimal solution of problem

$$Min: T_i^{total} \quad (1)$$

Two methods to reduce the value of  $T_i^{total}$ . The first is to make  $Core_i$  perform its accesses to  $Block_j$  in the nearest banks. The second is to increase the value of  $R_i$  as large as possible. Let  $T_{wire,i}$  be the least time for  $Core_i$  to access any bank, we can get the lower bound of the problem (1),

$$LB_i = NA_i \times (T_{Location} + T_{bank-to-network} + T_{wire,i})$$

For all the cores, we can get the optimal problem as

$$Min: \underset{i=1}{C} Max(T_i^{Total}) \quad (2)$$

And the lower bound will be

$$LB = \max(LB_i), 1 \leq i \leq C$$

in which: C is the total number of cores.

### 3. Migration algorithm design

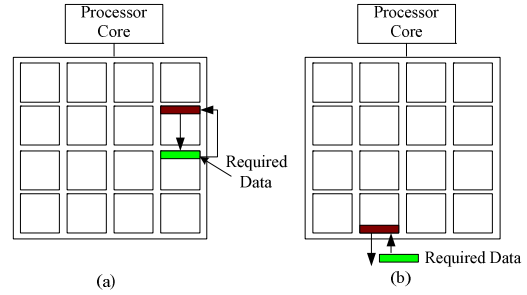
#### 3.1. Baseline policies

In this paper, our work mainly aims at reducing  $T_{wire}$ . That is, to reduce the time of transferring the cache block from the source bank to the target processor core by pre-migrating the predicted cache block toward its user. Before we introduce the PBM policy, some baseline policies of Dynamic-NUCA [7] are listed as follows:

- Location Policy, which will decide how to find a data block in the L2 cache. There are two approaches: centralized tag

array and distributed tag array. The advantages and shortcomings of each approach have been discussed in [7]. In our simulation experiments, we simulated a centralized tag array to get an uniform locating time, in order that the  $T_{Location}$  won't disturb our comparison and evaluation on  $T_{wire}$ ;

- Replacement policy. That is, when a cache block is migrated from one bank to a nearer bank to the user or from off-chip memory into L2 cache, which cache block will be replaced. For the first condition, an improved LRU algorithm will be used in the target cache bank to find a least recently used cache block, this block is swapped with the required block. For the second condition, the cache block fetched from off-chip memory is placed in a farthest bank to the user and the least used cache block is evicted (as shown in figure 2).



**Figure 2. Replacement policy: (a) on-chip migration; (b) data fetching from off-chip memory or other L2 caches.**

- Migration counter. In CMP system, the cache block may be accessed by multiple processor cores concurrently, so according to the migration policy the accessed block may thrash in different directions. The migration counter is used to prevent this. When a processor core accesses a block, the counter increases, and if the counter is saturated, migration is started and the counter is reset.

#### 3.2. PBM policy

In this part, we present a PBM policy based on sequential prediction.

**3.2.1 Sequential prediction.** As we mentioned in 1.3, in sequential pre-fetching techniques, when a cache miss occurs, it can prefetch not only the block which the cache misses, but also the next block. This method uses the spatial locality of the program, which follows the principle that the likelihood of referencing a storage location is greater if a storage location near it has been recently referenced. We checked the feature of spatial locality of applications in SPEC2000 benchmark, and collected the address distances between two sequential accesses to memory, which is called stride. Figure 6 shows that the sequential prediction, which is also called Next-line prediction, is very effective and for most applications a stride of one block accounts for the largest proportion. The detailed simulation experiment is in section 4.

**3.2.2 PBM algorithm.** We import the sequential prediction into traditional migration policy, and provide a prediction

based migration (PBM) policy. And in the PBM policy, when a block is migrated among the banks, the next block (if it is available on chip) is also migrated. So the procedure of an access to L2 cache can be described in step as follows, where the symbol  $A$  is the accessed block and  $B$  is the next block after  $A$  in off-chip memory.

#### Prediction based migration:

- Step 1: search  $A$  in L2 cache. If  $A$  is unavailable in L2 cache, issue an off-chip access, else access  $A$ ;
- Step 2: increase  $A$ 's migration counter by 1, if the migration counter of  $A$  is saturated and  $A$  hasn't reach the nearest bank column, migrate block  $A$  towards the user and reset the counter;
- Step 3: search  $B$  in L2 cache. If  $B$  is available in L2 cache and  $B$  hasn't reach the nearest bank column, increase  $B$ 's migration counter by 1, if the migration counter of  $B$  is saturated, migrate block  $B$  towards the user and reset the counter.

Here are some explanations to the algorithm:

1) in step 1, the searching mechanism we used in our simulation experiment is the centralized tag array policy. Because in the centralized tag array, the searching time  $T_{Location}$  is uniform and will not affect the comparison of  $T_{wire}$  in different migration policies;

2) in step 1, if an L2 miss occurs, an off-chip memory access will be launched through DMA operation. In our simulation experiments we will give a constant parameter to denote the off-chip overhead. One more radical policy is to launch an off-chip memory access even if the predicted block  $B$  is unavailable on chip, but this will make it unclear that whether the performance contribution comes from the migration policy or from the off-chip pre-fetching, and what's more, this may result in cache pollution.

3) In order to prevent the block from thrashing between two processor cores, we use a migration counter. When a block is accessed, the migration counter increases. And when the counter reaches a threshold [7], the block is migrated. And what's more, the prediction will stop when the block has been in the nearest bank column to the processor cores, which will also reduce the thrashing (according to [11], the processor cores sharing the same L2 cache are placed on the same side of L2 cache, and they share the same closest bank column).

### 3.3 Performance analysis

As we discussed in 2.2, there are two ways to reduce the L2 cache access latency. The first is to make L2 cache access occur in as nearer banks to the core as possible, which contributes to reduce  $T_{wire}$ ; the second is to make the L2 cache miss rate be as lower as possible. The policy provided in 3.2 mainly aims to reduce  $T_{wire}$  by migrate the predicted block toward the user without increasing other access time. In this section, we will analyze the performance improvement of the policy.

The main notations used throughout this paper are summarized in Table 1 for clarity.

**TABLE 1**  
**SUMMARY OF NOTATIONS**

| Symbol                          | Description  |
|---------------------------------|--|
| <b>Cache block parameters</b>   |  |
| $d_m$                           | the normalized distance between a data block and a processor core              |
| $Block$                         | cache block being accessed by a processor core                                 |
| <b>Applications' parameters</b> |  |
| $Core$                          | a processor core on which the application is running during its runtime        |
| $NB$                            | Total number of blocks which has been accessed during an application's runtime |
| $NA_i$                          | Total number of L2 accesses for $Core_i$                                       |
| $NA_{i,j}$                      | Total time of the application accessing $Block_j$ during the runtime.          |
| <b>Other symbols</b>            |  |
| $TD$                            | Latency increment of transferring a cache block for one further bank           |

As we discussed in 2.2, for a given application or task running on  $Core_i$ , the total L2 cache access time  $T_i^{total}$  will be:

$$T_i^{total} = NA_i \times T_{Location} + NA_i \times T_{bank-to-network} \times R_i + \sum_{j=1}^{NB} NA_{i,j} \times (T_{wire,i,j} + T_{contention} \times C_i) \times R_i + NA_i \times (1 - R_i) \times T_{off-chip} \quad (3)$$

So the average L2 latency can be formulated as:

$$L_{i2} = \frac{T_i^{total}}{NA_i} = T_{Location} + T_{bank-to-network} \times R_i + (1 - R_i) \times T_{off-chip} + \sum_{j=1}^{NB} (NA_{i,j} \times (T_{wire,i,j} + T_{contention} \times C_i) \times R_i) / NA_i$$

So

$$L_{i2} \propto \sum_{j=1}^{NB} NA_{i,j} \cdot T_{wire,i,j} \quad (4)$$

In formula (4),  $NA_{i,j} \cdot T_{wire,i,j}$  is the total transfer time when accessing  $Block_j$  and for a single access the transport time can be calculated as:

$$T_{wire,m} = d_m \times TD$$

in which:  $d_m$  is the normalized distance between  $Block_j$  and processor  $Core_i$ .  $TD$  is the latency increment of transporting a cache block for one further bank.

The object of our PBM policy is to reduce the average L2 access latency by reducing the average transfer distance between the block and the processor core.

Let the average transfer distance in TMP policy be  $d$ , and that in PBM policy be  $d'$ . According to our analysis, in any situation we can get  $d' \leq d$ . We didn't list the detail procedure because of the limit on paper's length. But we can find that the PBM policy can reduce the average transfer

distance and the proportion of reduction can be expressed as  $\frac{d-d'}{d} \times 100\%$ . In order to find out the upper bound of reduction, we define  $d_m$  as:

$$d_m = d_0 + d_n$$

in which:  $d_0$  is the distance between the nearest bank and the interface of  $Core_i$  to the on-chip network.  $d_n$  is the distance between  $Block_j$  to the nearest bank of  $Core_i$ . So  $d$  and  $d'$  can be expressed as:

$$d = d_0 + d_{n1}$$

$$d' = d_0 + d_{n2}$$

in which  $d_{n1}$  and  $d_{n2}$  are the distances between the average position of  $Block_j$  to the nearest bank of  $Core_i$ .

In the best situation, the PBM can pre-migrate  $Block_j$  into the nearest bank of  $Core_i$  before it is actually accessed, then  $d_{n2} = 0$ ,  $d' = d_0$ , so the upper bound of reduction is

$$UB_{reduction} = \frac{d_{n1}}{d} \times 100\% \quad (5)$$

We can find from (5) that the upper bound of reduction depends on  $d_{n1}$  and in TMP policy  $d_{n1}$  depends on the initial distance between  $Block_j$  and processor  $Core_i$ , so if the data block is initially quite far from the processor core, the reduction of average transfer distance can be considerable.

## 4. Simulation experiment and results

### 4.1 Experiment setup

In order to evaluate the proposed PBM policy, we set up a NUCA-based L2 cache model, as shown in figure 3, and the system parameters we list in Table 2 have been verified to be optimized in [7].

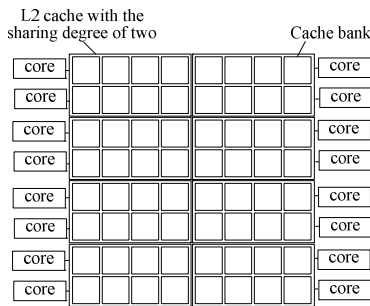


Figure 3. The simulation model for experiment

To simulate real applications' L2 cache behavior, we ran the SPEC2000 benchmark on an SMP computer (the parameters are shown in table 3), and we captured the memory trace with the HMTT toolkit [16]. The HMTT toolkit is designed especially for memory trace capture. It is composed of a Memory Trace Board (MTB) and a trace Packets Capture and Analysis Toolkits (PCAT). The MTB is

a hardware monitor board. When the SPEC2000 is running on the SMP machine, the MTB is plugged in an idle DIMM slot and it can snoop on memory command signals which are sent to DDR SDRAM from memory controller. We shut down the L2 cache of the processors, so the required memory addresses in the command signal is directly leaked from the L1 cache. Then MTB forwards the command to a Recorder machine. The Recorder machine extracts the memory addresses from the memory command signal and organizes them into trace items (see figure 4) which is a tuple  $\langle address, r/w, timestamp \rangle$ . After SPEC2000 finished running on the SMP machine, the Recorder machine will get the whole memory trace.



Figure 4. The format of memory trace items

TABLE 2  
SYSTEM PARAMETERS

| PARAMETER         | VALUE                                      |
|-------------------|--|
| Number of CPUs    | 16   |
| Processor Model   | In-order                                   |
| L2 cache          | 8 × 8 banks                                |
| L2 cache bank     | 64k, 4 cycle latency                       |
| Sharing degree    | 2  |
| Cache block size  | 64B  |
| Network           | 1 cycle latency between two adjacent banks |
| On-chip directory | Centralized partial tag arrays             |

TABLE 3  
PARAMETERS OF COMPUTER USED FOR CAPTURING MEMORY TRACE

| FEATURE   | PARAMETER   |
|-----------|---|
| CPU       | 2 × Intel P4 2.0GHz   |
| L1 Cache  | 12KB I, 64B/Line, pseudo-LRU<br>8KB D, 64B/Line, pseudo-LRU |
| L2 Cache  | Shut down   |
| Memory    | DDR 200, 512M   |
| OS        | Fedora Core 7 (Kernel 2.6.22)                               |
| Benchmark | SPEC 2000, 12 CINT+13CFP programs                           |

Then we decode the memory with the PCAT and feed them into a simulator we write in C language. The simulator is

composed of three modules: *Task synchronizer*, *Network assigner* and *Block container*.

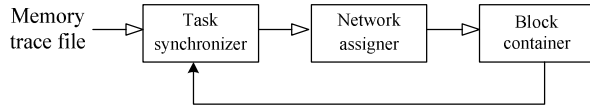


Figure 5. The simulator's working mechanism

The mechanism of the simulator is shown in figure 5. A simulated tag array is kept in the *Task synchronizer*. When the simulation starts, the *Task synchronizer* reads the memory trace items in batches, and extracts the block number and the timestamp, then searches the block number in the tag array sequentially. If cache hits, the *Task synchronizer* sends a request to the *Network assigner* according to the timestamp, or else start an off-chip access, then updates the tag array and the *Block container*. The request which is sent by the *Task synchronizer* to the *Network assigner* contains the information of source, target, and block number. The *Network assigner* contains a soft X-Y on-chip router. When it receives a request from the *Task synchronizer*, the network assigner marks all the links on the path from the source to the target as OCCUPIED, and return a SUCCESS signal to the *Task synchronizer*. Or else if there is no available path found, it will return a signal of FAILED. According to the signal from the *Network assigner*, the *Task synchronizer* can choose to send a new request or resend the old request a fixed period of time later. The *Block container* records the block numbers in each bank, and what's more it implements the mapping policy, migration policy and the replacement policy. For each operation to the blocks, the *Block container* will return a signal to the *Task synchronizer* to update the tag array.

## 4.2 Experimental results

In this part, we introduce the result of our simulation experiments which will show the effect of our PBM policy.

**4.2.1 Sequential Prediction.** Our PBM policy is based on the sequential prediction, which claims that the block after the presently accessed block will be referenced right away. We record the memory stride of the sequential memory trace. The result is show in figure 6.

We can see from figure 6 that for both integer and floating-point applications the memory stride of one block accounts for the largest proportion. So sequential prediction is accurate for most applications to foretell which block will be referenced before long. However, we can also see that for some applications (e.g. bzip2, vpr and apsi), the one-block stride only accounts for over 30% (but still the largest proportion). For such applications, the memory stride's distribution is quite out of order. So how to get more accurate prediction will be our next research.

**4.2.2 Average Transfer distance.** The target of our PBM policy is to reduce the transfer distance between the source bank to the target processor core, which will reduce the  $T_{wire}$  and consequently reduce the average L2 access latency.

Figure 7 shows the comparison of average transfer distance between TMP and PBM policy. We can see that for each application the average transfer distance is reduced in a certain proportion. In the best situation, the average transfer distance is reduced by 16.9%.

Figure 8 shows the improvement on average access latency owing to the reduction of average transfer distance. We can see the reduction of L2 access latency is achieved in all the applications. And the improvement can reach up to 8.4%.

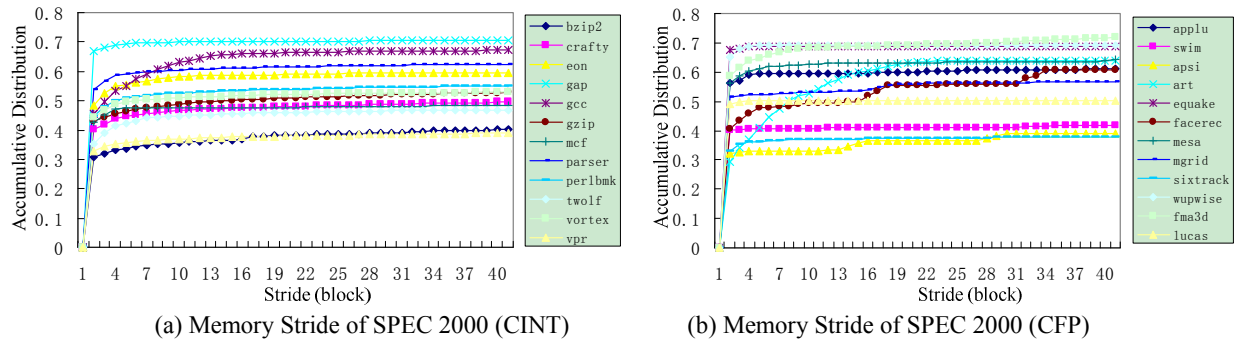
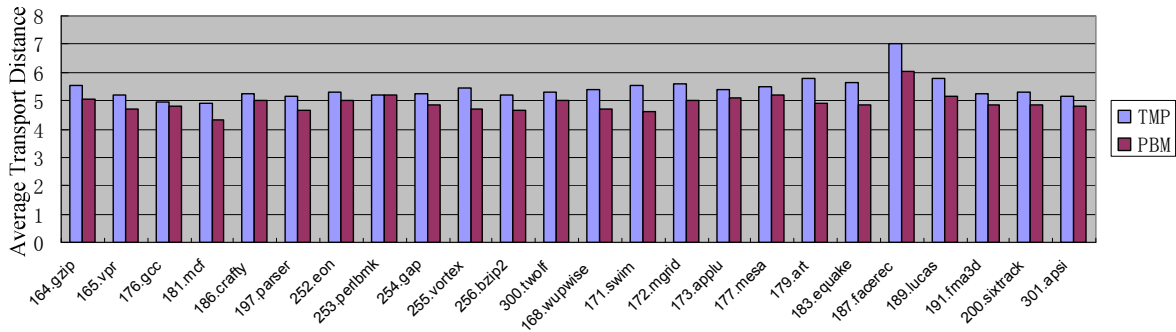
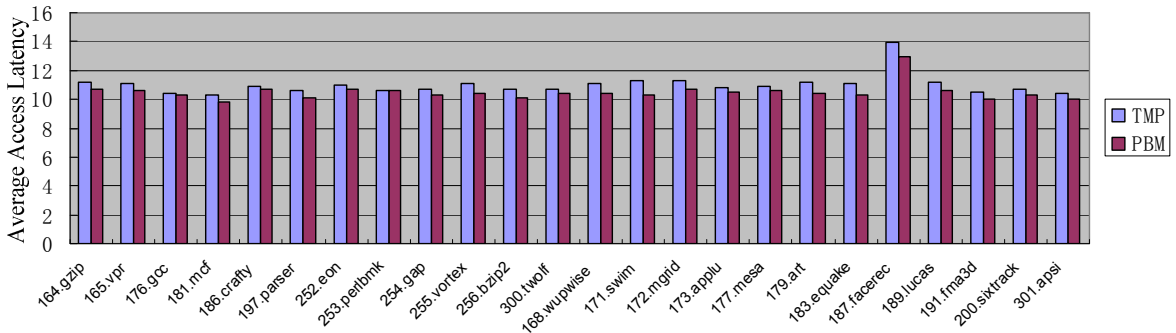


Figure 6. Spatial locality feature of SPEC 2000



**Figure 7. Comparison of the average transfer distance between TMP and PBM policy. The results are normalized by wire length between two adjacent routers**



**Figure 8. Comparison of L2 cache’s average access latency (cycles)**

## 5. Related work

The increasing wire delay makes the physical position of data in the cache very important to the access latency. If a core in the CMP wants to use the data far from it, the access latency will be significantly high compared to the data near the core.

The NUCA architecture [6, 11] was first presented to hide the wire delay. In the NUCA, the cache was divided into several banks, if a core wants to use data in a remote bank, the data will be provided to it, and at the same time the data will be moved to a bank near the core for the next access.

As a development, Huh et al. [7] design a CMP cache to support a spectrum of sharing degrees, denoting the number of processors sharing a pool of their local L2 banks. The average access latency can be decreased by partitioning the aggregate on-chip cache into disjoint pools, to fit the run application’s capacity requirement and sharing patterns.

Lastly, Chang and Sohi present the Cooperative Caching (CC) [13], which could achieve the benefits of both private and shared cache designs. But the shortage of CC is its lack of support to TLP which could improve the performance of CMP significantly. CC uses a private cache based architecture which can reduce the number of expensive cross-chip and off-chip accesses.

## 6. Conclusion

In this paper, a CMP L2 cache accessing performance model to analyze and evaluate the L2 access efficiency is designed. According to this model, a prediction based migration policy is proposed and an active data migration algorithm is designed. This proposed policy is based on the principle of locality and employs the sequential prediction technology to indicate the data to be accessed in the near future. According to the applications’ features of accessing L2 cache, the predicted block will be pre-migrated toward its user before it is actually accessed.

Two objects of the proposed PBM policy are to achieve shorter block transfer distance and lower average access latency. The analysis results show that this active data migration algorithm can take advantage of the features of L2 cache access to improve the performance of L2 cache much more than the traditional data migration policy.

According to the simulation experiment, the proposed policy exhibits significant improvements in L2 cache access latency compared to traditional migration policy. The experiment shows that the average block transfer distance can be reduced up to 16.9%, and the average access latency can be reduced by up to 8.4%.

### Acknowledgement

The authors would like to thank the Advanced System Laboratory of the Institute of Computing Technology, Chinese Academy of Science for providing their HMTT toolkit and especially thank Dr. Yungang Bao for spending several weeks with us to help to capture the memory trace of SPEC2000.

### References

- [1] R. Kalla, B. Sinharoy, and J.M. Tandler. "IBM Power5 Chip: A Dual-Core Multithreaded Processor". *Micro, IEEE* Volume 24, Issue 2, Mar-Apr 2004, Page(s): 40 – 47
- [2] C. McNairy, R. Bhatia. "Montecito: A Dual-Core, Dual-Thread Itanium Processor". *Micro, IEEE* Volume 25, Issue 2, March-April 2005, Page(s): 10 – 20
- [3] P. Congetira, K. Aingaran, and K. Olukotum. "Niagara: A 32-Way Multithreaded Sparc Processor". *Micro, IEEE* Volume 25, Issue 2, March-April 2005, Page(s): 21 - 29
- [4] D. Pham, S. Asaon, M. Bolliger and etc. "The Design and Implementation of a First-Generation CELL Processor". *Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International Volume 1*, 6-10 Feb. 2005, Page(s): 184 – 592
- [5] B. M. Bechmann, and D. A. Wood. "Managing Wire Delay in Large Chip-Multiprocessor Caches". *Microarchitecture, 2004. MICRO-37 2004. 37th International Symposium on 04-08 Dec. 2004*, Page(s): 319 - 330
- [6] C. Kim, D. Burger, S.W. Keckler. "An Adaptive Non-Uniform Cache Structure for Wire-Delay Dominated On-chip Caches". *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS-10)*, ACM Press, 2002, Page(s): 211-222.
- [7] J. Huh, C. Kim, H. Shafi, L.X. Zhang. "A NUCA Substrate for Flexible CMP Cache Sharing". In the 19th ICS, June 2005, Page(s): 31–40.
- [8] P.J. Denning, S.C. Schwartz. "Locality of reference". *Communications of the ACM*, Volume 15 , Issue 3 (March 1972), Page(s): 191-198
- [9] X. Li, L. Ji, B. Shen, W. Li, Q. Zhang. "VLSI implementation of a high-performance 32-bit RISC microprocessor". *Communications, Circuits and Systems and West Sino Expositions, IEEE 2002 International Conference on*, Volume 2, 29 June-1 July 2002, Page(s): 1458 – 1461
- [10] D. Prycker, M. "Representing the Effect of Instruction Prefetch in a Microprocessor Performance Model". *Transactions on Computers*, Volume C-32, Issue 9, Sept. 1983, Page(s): 868 – 872
- [11] C. Kim, D. Burger, S. W. Kechler. "Nonuniform Cache Architectures for Wire-Delay Dominated On-Chip Caches". *Micro, IEEE* Volume 23, Issue 6, Nov.-Dec. 2003 Page(s):99 – 107
- [12] M. Zhang, K. Asanovic. "Victim Migration: Dynamically Adapting Between Private and Shared CMP Caches". Technical Report of Computer Science and Artificial Intelligence Laboratory, MIT. Oct. 10, 2005.
- [13] J. Chang, and G. S. Sohi. "Cooperative Caching for Chip Multiprocessors". *Computer Architecture, 2006. ISCA '06. 33rd International Symposium on 2006* Page(s):264 – 276
- [14] M. Zhang, K. Asanovic. "Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessor". *Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on Jun. 4-8 2005* Page(s):336 – 345
- [15] K. Asanovic et al. "The Landscape of Parallel Computing Research: A View from Berkeley". Technical Report No. UCB/EECS-2006-183, <http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html>
- [16] Y. Bao, M. Chen, Y. Ruan, L. Liu, J. Fan, Q. Yuan, B. Song, J. Xu. HMTT: a platform independent full-system memory trace monitoring system. *SIGMETRICS 2008*, Annapolis, Maryland, USA, June 2-6,2008
- [17] A. Sayaka, L. Feihui, K. Mahmut, R. Padma, I.M. Jane. "Ring Prediction for Non-Uniform Cache Architecture". *Parallel Architecture and Compilation Techniques, 2007. PACT 2007. 16th International Conference on*, 15-19 Sept. 2007 Page(s):401 - 401