# Optimizing Matrix Transposes Using a POWER7 Cache Model and Explicit Prefetching

Gabriel Mateescu
Ecole Polytechnique Fédérale de Lausanne
Blue Brain Project, 1015 Lausanne, Switzerland

Gregory H. Bauer and Rober A. Fiedler
National Center for Supercomputing Applications
1205 W. Clark St., Urbana, IL 61801, USA

## ABSTRACT

We develop a matrix transpose approach on the POWER7 architecture based on modeling the memory access latency and cache, and then designing the cache blocking, data alignment, and prefetching techniques that enhance performance.

## Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques

## General Terms

Performance model, Memory Bandwidth, Concurrency

## Keywords

Matrix Transpose, Cache, Prefetching, POWER7

## 1. INTRODUCTION

We consider the problem of transposing a matrix out-of-place, and design a tiling and software prefetching approach that matches the POWER7 processor microarchitecture. We model the concurrent processes of transposing and prefetching and derive a model of the transpose time.

## 2. PROBLEM FORMULATION

Let $A$ be a matrix of order $n$, where $A = \{a_{i,j} \mid 1 \leq i, j \leq n, a_{i,j} \in \mathbb{R}\}$. We consider the out-of-place matrix transpose problem: construct the matrix $A^T = \{a^T_{j,i} \mid 1 \leq i, j \leq n\}$ at memory locations disjoint from $A$ such that $a^T_{j,i} = a_{i,j}$ for $1 \leq i, j \leq n$. The machine's RAM is assumed to be large enough for both $A$ and $A^T$ to fit in memory. We employ blocked (tiled) transpose, whereby $A$ and $A^T$ are divided into square blocks of order $B$. Assume that $B$ divides $n$; the number of blocks in $A$ or $A^T$ is $(n_B)^2 = (n/B)^2$.

## 3. APPROACH AND MODEL

We model and implement three widely used techniques for improving the performance of the transpose operation: (1) cache prefetching: to avoid compulsory cache misses; (2) model-based blocking: to improve the cache hit rate by improving the temporal locality; and (3) data alignment to avoid cache conflict misses. We model the time to perform the transpose in terms of algorithmic and processor microarchitecture parameters.

Prefetching can hide the latency associated with loading (or storing) data from (or to) memory. Prefetching works by launching the loading of the data from memory into the level-one (or L1) cache at a point in the program execution that is ahead of the point where the data is used, so that, by the time the program execution reaches point where it needs the data, the data has been loaded into the L1 cache.

Algorithm 1 implements cache-blocking and prefetching to perform the transpose operation; it traverses the $B \times B$ blocks of $A$ and $A^T$, in row-major and column-major order, respectively. The *prefetch stride* is the distance between the matrix element being copied from the current block and the matrix element being prefetched from the next block. The prefetch strides for traversing the blocks in row-major and column-major order are $S_r$ and $S_c$, respectively.

**Algorithm 1** Tiled Matrix Transpose with Prefetch

```
1.  for (i_b = 0; i_b < n; i_b = i_b + B) do
2.    for (j_b = 0; j_b < n; j_b = j_b + B) do
3.      for (i = i_b; i < i_b + B; i = i + 1) do
4.        j_r = j_b + S_r;
5.        j_c = j_b + S_c + i;
6.        Prefetch(a[i][j_r : j_r + B])
7.        Prefetch(a^T[j_c][i : i + B])
8.        for (j = j_b; j < j_b + B; j = j + 1) do
9.            a^T[j][i] = a[i][j]
10.       end for
11.     end for
12.   end for
13. end for
```

The unit for prefetching is a cache line of length $L$, measured in bytes. The block size $B$ must be a multiple of $L$. In order to avoid capacity misses in the L1 cache, which must simultaneously hold four blocks (the source and target blocks being transposed and the source and target blocks being prefetched), we find that the block size is $B \in \{16, 32\}$. For reference, the POWER7 processor [4] has $L = 128$ bytes and a L1 cache of 32 KB [4, 1].

The data alignment and the prefetch strides are related by $R$, the storage reserved for a row of $A$, which includes the $n$ elements and additional padding elements to avoid conflict cache misses: $R = L/8 \times (n_S \times \lceil (8\,n)/(L \times n_s) \rceil + 1)$. The stride $S_r$ is: (1) $B$, if $j_b < J_b$, where $J_b = \max\{j_b\}$; (2) $B \times R - j_b$, otherwise, where $j_b$ is the column index of the first column in block $b$ of $A$. The stride $S_c$ is (1) $B \times R$,


*PMBS'11,* November 13th, 2011, Seattle, Washington, USA.
ACM 978-1-4503-1102-1/11/11.

if $j_b < J_b$ where $J_b = \max\{j_b\}$; (2) $B - j_b \times R$, otherwise, where $j_b$ is the row index of the first row in block $b$ of $A^T$.

The L1 cache in POWER7 is an eight-way set associative memory [1, 3]. We can think of the L1 cache as an array with $n_S = 32$ rows and $n_A = 8$ columns, where $n_A$ is the number of cache lines in a set, and $n_S$ is the number of sets.

Since the L1 cache is not fully associative, alignment of the rows of $A$ and $A^T$ with respect to the sets of the L1 cache is needed to avoid conflict misses. During the transpose operations four blocks are in cache: two blocks that are tansposed and two that are prefetched. Data alignment ensures that these blocks fit in cache without a conflict miss. If consecutive block-rows of a block of $A$ and $A^T$ are located in consecutive sets of the L1 cache, then there is room in cache for the four blocks; by reserving $8 \times R$ bytes for each row of $A$ and $A^T$, we ensure this.

To get the transpose time we compute the time taken by one iteration of the loop at lines 3-10 in Algorithm 1, henceforth called the $i$ loop, when there are no memory stalls. The time is $T_{work} = 2\,B\,T_{L1} + (8\,B/L)\,T_{PF}$, where: $T_{L1}$ is the number of clock cycles needed to load (store) a floating point register from (to) the L1 cache; $8\,B/L$ is the number of cache lines whose prefetching is initiated in one iteration; $T_{PF}$ is the time taken by the prefetech instructions. For the POWER7 processor, $T_{L1} = 2$ cycles and $T_{PF} \approx 36$ cycles.

We avoid cache misses in the $i$ loop by prefetching, so that the actual time taken by one iteration of the loop is $T_{work}$. We implement prefetech-for-load with the L1 *data cache-block touch* (DCBT) instruction and prefetech-for-store with the L1 *data cache-block set to zero* (DCBZ) instruction [3]. The *prefetch distance*, $D$, is the difference between the iteration in which a cache line is used and the iteration in which the prefetch of that line was launched. We have $D = B$ for DCBT, and $D \leq B$ for DCBZ, so the number of concurrent prefetch instructions, $C$, is $C \leq 8\,B/L \times D = 8\,B^2/L$.

Little's law gives $T_{prefetch} = \lambda/C = (\lambda \times L)/(8\,B^2)$, where $\lambda$ is the memory latency. Prefetching occurs concurrently with copying $a[i][j]$ to $a^T[j][i]$. If $T_{prefetch} \leq T_{work}$ there are no stalls and an iteration of the $i$ loop takes $T_{work}$. For POWER7 $\lambda = 336$ cycles, so $T_{work} = (25/4)\,B$, $T_{prefetch} = (336 \times 16)/B^2$ and for $B \in \{16, 32\}$, $T_{work} > T_{prefetch}$ so the time per iteration is $T_{ideal} = (25/4)\,B$ cycles. The iteration that transposes the first block-row of block $b$+1 must wait for all previously launched DCBZ instructions to take effect, i.e., for all the $B$ block-rows of block $b$+1 of $A^T$ to arrive in the L1 cache. This iteration takes $T^S_{iter} = \lambda + T_{L1} - T_{ideal} = (338 - 25/4\,B)$ cycles. Out of $B$ consecutive iterations of the $i$ loop, $B-1$ take $T_{ideal}$ cycles and one takes $T^S_{iter}$ cycles, so the time to transpose a block of $A$ is $T_{block} = T^S_{iter} + (B-1) \times T_{ideal}$ cycles. The time to transpose $A$ is $T_{tr} = n_B^2 \times T_{block}$. The throughput of the transpose operation is the size of $A$ plus $A^T$, $16n^2$, divided by $T_{tr}$, i.e., $BW = 16\,n^2/T_{tr}$ bytes/cycle. The clock frequency of POWER7 is about 4 GHz, so we get: (1) for $B = 16$, $BW = 9.4$ GB/sec; and (2) for $B = 32$, $BW = 10.3$ GB/sec.

## 4. NUMERICAL EXPERIMENTS

POWER7 supports multiple page sizes and hardware streams. A POWER7 system running the Linux kernel version 2.6.35 supports two page sizes: 64 KB (default) and 16 MB. We control the hardware streaming with the *data streaming control register* (DSCR): (1) DSCR=1, streams are disabled; (2) DSCR=0, streams are enabled for load operations only; and (3) DSCR=15, streams are enabled for load and store operations. Hardware streams can be used with or without software-based prefetching.

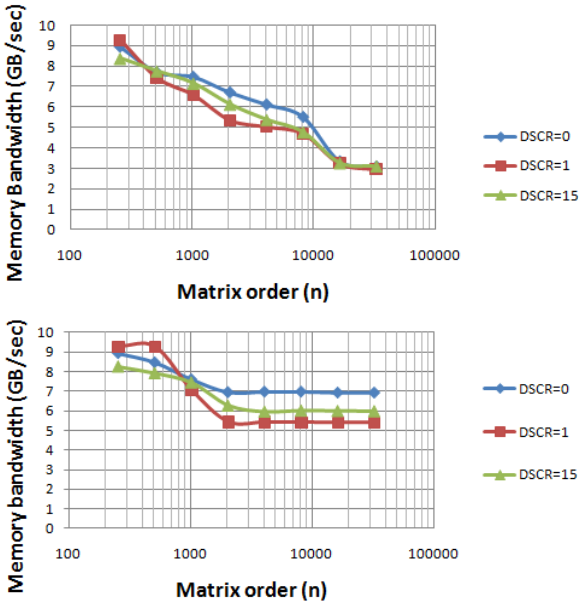


**Figure 1: Memory bandwidth for $B = 32$ and page sizes of 64 KB (top) and 16 MB (bottom)**

Figure 1 shows that the large page size greatly improves the bandwidth for large $n$ because it reduces the TLB misses, whose effect is significant 64 KB pages. We see that $DSCR = 0$ is better than $DSCR = 15$, so the hardware predicts better the loads than the stores.

The ratio of the observed and projected performance is 70%, which means that the model is a good predictor if we account for the machine overheads. The performance is up to five times higher than that of the out-of-place matrix transpose routine *dgetmo* of the IBM Engineering and Scientific Subroutine Library (ESSL).

## 5. CONCLUSIONS

We have modeled cache blocking and prefetching for matrix transpose in terms of the POWER7 cache organization, memory access latency and concurrency. Based on the model, we have designed a matrix transpose code whose memory bandwidth is higher than that of the *dgetmo* routine. The full paper can be found in [2].

## 6. REFERENCES


[1] Kalla, R. Power7: IBM's next-generation server processor. *Micro, IEEE 30*, 2 (march-april 2010), 7 –15.

[2] Mateescu, G., Bauer, G. H., and Fiedler, R. A. Optimizing matrix transposes using a POWER7 cache model and explicit prefetching. *SIGMETRICS Performance Evaluation Review 40*, 2 (2012).

[3] Power.org. Power ISA Version 2.06. http://www.power.org/resources/downloads.

[4] Sinharoy, B. IBM POWER7 multicore server processor. *IBM Journal of Research and Development 55*, 3 (2011), 1:1–1:29.