International Journal of Hybrid Information Technology
Vol.8, No.7 (2015), pp.375-388
http://dx.doi.org/10.14257/ijhit.2015.8.7.34

GEMM Optimization for a Decoupled Access/Execute Architecture Processor

Zeng Zhao[1], Naijie Gu[1] and Yangzhao Yang[1]

[1] *School of Computer Science and Technology,*
University of Science and Technology of China, Hefei, 230026, China
tubac@sohu.com

Abstract

General dense matrix multiplication operation (GEMM) has a great impact on the performance of supercomputers. The typical blocking algorithm of GEMM divides the operation in to several levels. Because of the limitation of data transmission, previous works only focuses on optimizing the lowest level of the GEMM based on general-purpose processors. However, the Decoupled Access/Execute architecture (DAE) processor has enhanced the ability of data fetching. This paper will introduce several methods for optimizing GEMM based on a DAE processor. The Execute Processors (EP) of the platform is made up by Direct Register Access (DRA) and Direct Cache Access (DCA), which can be used to manage the data transport between registers, cache and memory. This paper pays more attention on optimizing the high levels of the blocking GEMM. The GEMM kernel based on the DAE processor was divided into 4 levels, and several levels of the new algorithm are capable to self-adjust. And the performance of our algorithm was effectively improved.

Keywords: *DAE Architecture, Optimizing, GEMM, Auto-Tuning, BLAS*

1. Introduction

The performance of numerical linear algebra libraries is an important indicator of high-performance computers. The most famous library is the High Performance LINPACK (HPL) [2], which is used to rate the systems on the Top500 list of the fastest computers in the world [3]. However, LINPACK has to invoke Basic Linear Algebra Subprograms (BLAS) libraries for performing basic vector and matrix operations [4]. The calculation overhead of HPL concentrated on a few functions. Previous studies show that the hottest function invoked by HPL is the general dense matrix multiplication operation (GEMM), which is a level-3 member of Basic Linear Algebra Subprograms library (BLAS) [6]. And the NO.2 hottest function is the general matrix-vector multiplication operation (GEMV). It is also a member of BLAS. Zhang Wenli et al. discovered that unoptimized GEMM occupies more than 93% of the computational overhead of HPL [5] on the Godson-3A platform. Similarly, we discovered that unoptimized GEMM takes up more than 90% of the calculation overhead of HPL Testing based on another processor - Godson-3B. Thus, optimizing the GEMM will effectively improve the performance of HPL Testing.

Godson-3B was developed at the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS) in China [1]. As a general-purpose MIPS64 CPU, Godson-3B remains all the feature of Godson-3A, but also makes improvement through the assistance of two extended vector units, 128-entry 256-bit register file and more than 300 SIMD instructions [2]. What is more, it is a typical Decoupled Access/Execute Computer architecture (DAE) processor. In the past few years, this DAE processor was tried to be used to build up the teraflop computer KD90. The

ISSN: 1738-9968 IJHIT

performance of HPL Testing based on the Godson-3B has been improved in advance.

For the same reason, many hardware manufacturers have done a lot of research works for GEMM optimization based on their own hardware platforms, such as Intel [7], AMD [8] and NVIDIA [9]. All of them have released their optimized BLAS libraries. The main methods for optimization include loop blocking, loop unrolling, loop permutation, fusion and distribution, prefetching, and software pipelining. Gunnels and Henry proposed the idea of block algorithms for GEMM in 2001 [10], their work reduces the overhead of copying matrices between different storage structures. Goto et al. introduced six GEMM block algorithms in their work, and analyzed the performance of each algorithm [11]. For Godson-3 platforms, Songsong-He et al. [13] and Xianyi-Zhang et al. [14] carried out their optimization GEMM based on Godson-3A respectively. On this basis, they released their optimized BLAS libraries for Godson-3A, these are named as UBLAS (another name of HPBLAS) and OpenBlas. Based on their works, Zhu Hai-tao designed an optimization GEMM kernel combining with the features of Godson-3B [15]. He implemented the optimized algorithm in a simulation platform for Godson-3B, proving that the overhead of memory access of the optimized kernel was completely concealed by computing. However, the simulation process takes a lot of time. Besides, there are many differences between the simulation platform and real platform. Furthermore, Zhu's algorithm uses a lot of hardware features. It increases the difficulty of coding and transplantation for the algorithm on the real platform with operating system.

On the other hand, model-driven optimization methods have already been used for accelerating the optimization process of GEMM. Utilizing the idea of model-driven optimization, GEMM of Automatically Tuned Linear Algebra Software (ATLAS) can find out the best parameters by running testing procedures [16]. Jakub Kurzak et al. have designed auto-tuning GEMM kernels for the Fermi GPU [17]. Xianyi-Zhang et al. also used model-driven methods to optimize GEMM based on Godson-3A [14].

In this paper, we design a model-driven optimization algorithm of GEMM for Godson-3B. First of all, we design the 4-levels kernels for our algorithm. Then, we will introduce a performance evaluation system based on Direct Register Access (DRA) and Direct Cache Access (DCA) of Godson-3B. With the help of this system, we collect the information about the relationship between fetch and calculation of these kernels. Several levels of the new algorithm are capable to self-adjust. Apart from this, we use the Godson-3B's hardware features to improve the performance for the incomplete Blocks of GEMM. The key contribution of this work is in the method for generating the search space and selecting the best parameter. In addition, we will give out the methods of making full use of the features of Godson-3B to improve the performance of memory access.

2. Basics for Godson-3B and GEMM

2.1 Godson-3B Architecture

Decoupled Access/Execute Computer architecture (DAE) was proposed by James E. Smith in 1982 [18]. The main feature of this architecture is preserving a high degree of decoupling between operand access and execution. Godson-3B is a chip of the DAE Architecture, which is improved based on Godson-3A. Besides, the processor of Godson-3B can use an instruction called VBMULADD to process multiply, add, and shuffle with a latency of 1 cycle. In this case, Godson-3B is equipped with Direct Register Access (DRA) and Direct Cache Access (DCA) to fix up its high performance of calculation. In other words, DRA and DCA make up the Access Processors (AP) of Godson-3B.

DRA can be used for transporting data between register, memory and L2-Cache, while DCA can be employed to manage the data transfer between memory and L2-Cache [20]. There are 3 reading channels and 1 writing channel in every DRA. In addition, it has 8 signal registers, which can be utilized for checking the working status of the channels. Apart from that, Godson-3B can always maintain data of a specified address space in L2-cache, named Cache-lock Window. The Architecture of Godson-3B is shown in Figure 1.

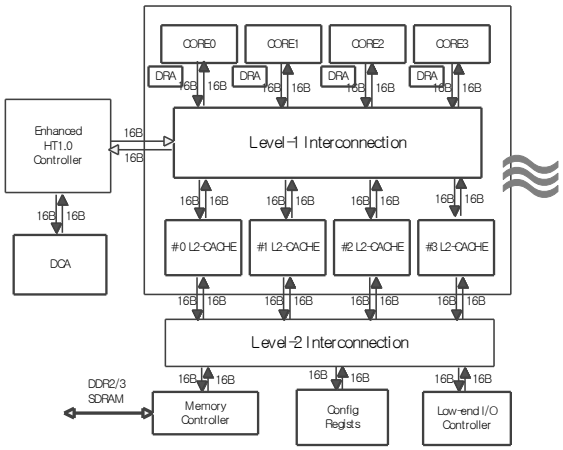


Figure 1: The Architecture of Godson-3B

2.2 GEMM Basics

It can be defined as $C = \alpha A \times B + \beta C$, where A, B and C are matrices of sizes $M \times K$, $K \times N$ and $M \times N$, respectively, while α and β are scalars. For simplicity, Matrix A and C are assumed to be stored in column major order. Meanwhile Matrix B is assumed to be stored in row major order. In canonical form, GEMM can be represented by three nested loops.

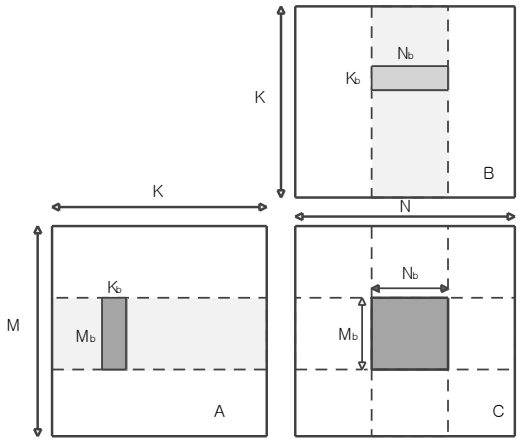


Figure 2. Blocking Algorithm of GEMM

The idea of blocking algorithm for GEMM was proposed by Henry. He divided Matrixes A, B, C into the unit blocks marked as A_b, B_b and C_b. The sizes of these blocks are $M_b \times K_b$, $K_b \times N_b$ and $M_b \times N_b$, respectively. Blocking algorithm can

enhance the performance of the memory access effectively. Go-to et al. designed a widespread blocking algorithm of GEMM [21]. They proved that storing unit blocks of Matrixes A in L2-cache can also meet the calculated demand for memory access. This algorithm raises the ratio of computation to memory operations for GEMM. A typical process of blocking GEMM is shown in Figure 2.

The ratio of computation to memory operations for the blocking algorithm $\tilde{\rho}$ is shown by equation (1):

$$\tilde{\rho} = \delta \frac{2M_b \times K_b \times N_b}{M_b \times K_b + K_b \times N_b + 2M_b \times N_b} \tag{1}$$

In equation (1), M_b, K_b and N_b are the sizes of the blocking algorithm. And δ is a scaling parameter of equation (1). We define the calculation amount as A_{calc} ,the time for calculation and accessing as T_{cacul} and T_{access} . So, the performance of GEMM can be calculated as:

$$Perf_{GEMM} = \frac{A_{cacul}}{T_{cacul} + T_{access}} \tag{2}$$

Based on the above definitions, equation (2) has an expanded mode as:

$$Perf_{GEMM} = \frac{A_{cacul}}{T_{cacul} + T_{L1} + T_{L2} + T_{mem}}, T_{L1} = \frac{A_{cacul}}{\tilde{\rho} \times \omega_{L1}}, T_{L2} = \frac{A_{cacul} P_{L1miss}}{\tilde{\rho} \times \omega_{L2}}$$
$$T_{mem} = \frac{A_{cacul} P_{L1miss} P_{L2miss}}{\tilde{\rho} \times \omega_{mem}} \tag{3}$$

In this equation, T_{cacul} means the time of calculation. T_{L1}, T_{L2} and T_{mem} express the time for accessing data of calculation from L1-cache, L2-cache and memory, respectively. And P_{L1miss} and P_{L2miss} represent the probability of cache-missing from L1-cache and L2-cache, while ω_{L1}, ω_{L2} and ω_{mem} stand for the bandwidth of L1-cache, L2-cache and memory. In the remaining chapters, we will introduce the methods of changing the value of these parameters for improving the performance of GEMM.

3. Optimization Algorithm for Godson-3B

As the transport model of Godson-3B is shown by Fig 3, the method of optimizing memory accessing for GEMM based on Godson-3B was first mentioned in the thesis of Haitao et al. They have designed a blocking algorithm based on DRA, DCA, SIMD Instructions and cache-locked window. DRA and DCA are utilized to hide the overhead of memory accessing in computing. As mentioned above, DCA can be used to prefetch data from memory to L2-cache directly. Meanwhile DRA can transport data from memory and L2-cache to registers without the control of Execute processor. Consequently execute processor has the ability to access data by L1-cache. That means when the execute processor calculates the result of multiplication operations, we can utilize DCA and DRA to prefetch data at the same time. In Zhu's algorithm, DCA is used to prefetch data of next loops of A_b s to the memory address space,

which is locked by the cache-locked window. Therefore, the value of P_{L2miss} is reduced to be 0. In the other hand, DRA is used to prefetch data of A_b and B_b from the space of cache-locked window to registers. Furthermore, DRA can be utilized to prefetch and write back the blocks of Matrix C. Accordingly, the overheads of accessing matrices are hidden by calculation.

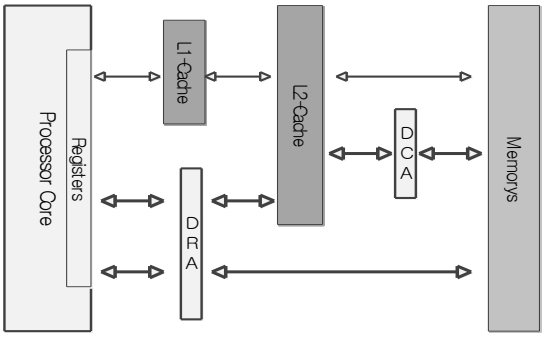


Figure 3. The Transport Model of Godson-3B

3.1 The 4-Levels GEMM for Godson-3B

In this paper, we divide that optimization algorithm of GEMM into 4 levels. These levels are named as macro-parallel, secondary kernel, inner kernel and computational kernel. The meaning and function of these levels are described as follows:

- **Macro-parallel level** provides sharing computation among the different cores of Godson-3B.
- **Secondary kernel** deals with the matrix multiplication operation of outer blocks of A, B and C. These outer blocks are defined as A_b , B_b and C_b . The sizes of them are $M_b \times K_b$, $K_b \times N_b$ and $M_b \times N_b$, respectively. The characteristic of this level is using DCA to prefetch A_b of next loop to L2-cache. As a result, the overhead of prefetching will be hidden by the accessing B_b and the calculation of secondary kernel.
- **Inner kernel** calculates the matrix multiplication as $C_i += A_i \times B_b$, where A_i and C_i is a block of A_b and C_b. For simplicity, A_i and C_i are named as inner blocks of Matrix A and C. The sizes of A_i and C_i are $M_i \times K_b$ and $M_i \times N_b$. And M_b is ε times of M_i , that can be shown as $M_b = \varepsilon \times M_i$. The characteristic of this level is using DRA to access C_i . That can be hidden by the calculation of inner kernels.
- **Computational kernel** is a micro-kernel. This level deals with the calculation of matrix multiplication operation as $C_m += A_m \times B_m$. The matrixes are divided into unit blocks. The size of these unit blocks are $m_b \times 1$, $1 \times n_b$ and $m_b \times n_b$. Computational kernel prefetches A_m and B_m by DRA. The data is prefetched from L2-cache to registers directly. The overhead of prefetching A_m and B_m is hidden by the calculation of this level.

The processes of the 4-levels algorithm are in Figure 4. Every level of the algorithm is called by its upper level successively.

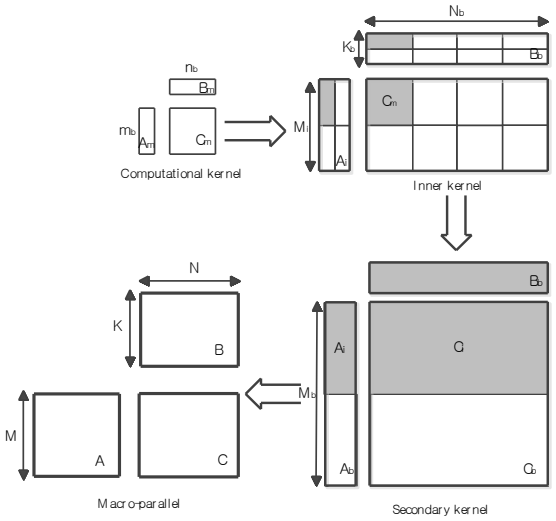


Figure 4. The Processes of the 4-levels Algorithm

3.2 Performance Analysis

In the 4-levels Algorithm, the T_{access} of GEMM can be calculated as

$$T_{access} = \max\{T_{pref-A_b}, T_{access-B_b} + \max\{\max\{T_{cacul}, \sum T_{pref-A_m}, \sum T_{pref-B_m}\}, \sum T_{access-C_i}\}\} \quad (4)$$

In equation (4), T_{pref-A_b}, T_{pref-A_m} and T_{pref-B_m} mean the time of prefetching A_b, A_m and B_m. And $T_{access-B_b}$ and $T_{access-C_i}$ express the time for accessing data of B_b and C_i. $M/M_b \times K/K_b \times N/N_b$ times of secondary kernel will be called by one macro-parallel. Every secondary kernel needs to fetch an outer block of A and B from memory to L2-cache. So the times of secondary kernel accessing data from A is $M \times K \times N / N_b$, and the times from B is $M \times K \times N / M_b$. As mentioned above, secondary kernel utilizes the DCA to access the data of A. In addition, every inner kernel needs to read an inner block of C, and then write it back when the calculation is finished. As $M_b / m_b \times K_b / k_b \times N_b / n_b$ times of inner kernel will be called by one secondary kernel, the macro-parallel will call inner kernel for $M / m_b \times K / k_b \times N / n_b$ times. The times of inner kernel accessing data from C is $M \times K \times N / k_b$. Apart from that, m_b members of A and n_b members of B will be fetched from L2-cache to registers for each computational kernel. If the following three conditions can be met, the algorithm will achieve the desired performance.

- The overhead of accessing of computational kernel can be hidden by calculation.
- Accessing C for inner kernel can be covered by calling computational kernels.
- The overhead of prefetching A_b can be hidden by calling inner kernels and accessing B_b of each secondary kernel.

These three conditions are independent of each other. Since every level uses different storage structure, the degree of coupling between different levels is relatively low. Besides, as the upper levels' performance relies on that of the lower level, we

design the auto-tuning kernels for every level of GEMM to satisfy these conditions one by one.

3.3 Evaluation Mechanism Based on Godson-3B

As the kernels of GEMM have been divided into 4 levels, the algorithm can find out the appropriate blocking parameters to help these levels get a higher performance. Based on the features of DAE architecture, we designed a fetch performance evaluation system -- DAEFS to collect the information about the relationship between fetching and calculation on the real platform.

In a conventional processor architectures, accessing and computing are both controlled by the execute processor. The processor is unable to assess the relationship between the overheads of accessing and computing. However, under the DAE computing architecture, a high degree of decoupling between operand access and execution can be preserved, consequently, the processor can evaluate the access efficiency of coprocessor more accurately.

As mentioned above, DRA and DCA of Godson-3B both have their own status registers. When the accessing is finished, these registers will be set up. Execute processor can obtain the status of the access coprocessor by checking these registers. And it will keep checking until the DRA and DCA finish their work. In addition, we design a struct named as Dstruct. The struct contains 5 members: componentName, hitTime, numTmp, hitThreshold and evenTags. The componentName is the name of access component. It may be DRA_X or DCA_X. X means the detailed module, just like DRA_readA or DRA_writeD. Whenever the processor query the status registers, the value of numTmp will increase by 1. While the numTmp equals the value of hitThreshold, hitTime will be added with 1. And then, numTmp will be set to be 0. What is more, the evenTags are used to mark the events which we are interested in, so we can get the value of hitTime for specific events.

Dstruct can be used to collect more information about the relationship between the accessing and computing. While the execute processor waits for the access processor to finish accessing, the value of hitTime will increase. Since the competition of memory access will lead to the performance fluctuations, we use hitThreshold to reduce volatility of statistical results. In this condition, if the value of hitTime is larger than 0, it means the overhead of accessing cannot be hidden by computing.

3.4 Auto-Tuning Kernels for Level 2 And 3

We add the code for operating Dstructs at the beginning of every prefetching. The block parameters can be changed according to hitTime and evenTags of Dstruct. We can get all the information about accessing for every level of our algorithm. For example, the componentName of Dstruct is DRA_readC and the evenTags is level 3. If the value of hitTime growing fast, it means that the overhead of prefetching C_i cannot be hidden. So, the auto-tuning kernel will increase the value of M_i or N_b. Accordingly, the value of $\tilde{\rho}$ for inner kernel will be, and more calculation can be used to hide the overhead of access C_i. Based on this principle, we design auto-tuning kernels for level 2 and 3. And the new GEMM based on these two levels auto-tuning kernels is shown by algorithm 1.

Algorithm 1: Auto-Tuning Kernels Based on DAEFS

Input: A, B, C, α, β
Output: $C := \alpha A \times B + \beta C$

```
1  init M_b, N_b, K_b, M_i ;
2  init Dstruct1, Dstruct2 ;
3  Dstruct1.componentName = DCA_NoTran ;
4  Dstruct1.evenTags = PRE_A_b ;
5  Dstruct2.componentName = DRA_readC||DRA_writeD ;
6  Dstruct2.evenTags = ACCESS_C;
7  for k from 0 to K step K_b do
8      for n from 0 to N step N_b do
9          for m from 0 to M step M_b do
10             calculate the beginning adress of A_b, B_b and C_b for this loop ;
11             check the status of DCA and collect the event of waiting by Dstruct1 ;
12             start DCA to prefetch next A_b ;
13             accessing B_b ;
14             for m_1 from 0 to M_b step M_i do
15                 calculate the beginning address of A_i and C_i for this loop ;
16                 check the status of DRA and collect the event of waiting by
                   Dstruct2 ;
17                 prefetch next C_b ;
18                 invoke Computational kernels;
19                 write back of the resault of C_i ;
20                 if Dstruct2.hitTime ≥ Threshold_c then
21                     increase the sizeof C_i;//M_i+ = x_m or N_b+ = x_n
22                 end
23             end
24             if Dstruct1.hitTime ≥ Threshold_a then
25                 increase the sizeof A_b;//M_b+ = x_m2 or K_b+ = x_k
26             end
27         end
28     end
29 end
```

The search space of our algorithm contains M_i, K_b N_b and M_b. It is a 4D parameter space. And constraints come from a few different sources. Main constraints deprive from the quantitative ceiling of data that can be calculated by one SIMD instruction, the performance of computational kernel, the size of the cache-locked window and the multiple relationships between these parameters. First of all, the performance of computational kernel is influenced by the value of m_b and n_b. As every SIMD instruction of Godson-3b can process 4 double-data at a time, we delimit the size of computational kernel as 12×12. Thus the search space of computational kernel is constrained as follow:

- There are 8 signal registers in DRA, correspondingly, the amount of A_m or B_m in the registers must be less than 8, and then the value of K_b should be a multiple of 8.
- As C_i s are all stored in registers, M_i and K_b have to satisfy the following equation:

$$M_i \times K_b \times n < sizeof(registers) n \geq 2 \quad (5)$$

- M_i must be a multiple of m_b, meanwhile M_b should be a multiple of M_i. That can be expressed as

$$M_i = \mu m_b, \mu \geq 1; M_b = \mu_b M_{i,} \mu_b \geq 1 \quad (6)$$

- In addition, the data of level-2 are all stored in cache-locked window, so the value of M_i, K_b and N_b are all limited by the size of cache-locked window. That can be expressed as

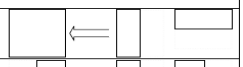
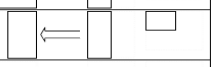
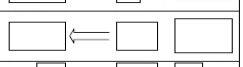
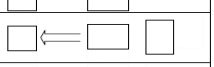
$$\begin{aligned} &\mu_b M_i \times K_b + K_b \times N_b \le sizeof(CLwindow), \mu_b \ge 2 \\ &sizeof(CLwindow) < sizeof(L2-cache) \end{aligned} \quad (7)$$

4. Optimizing for Incomplete Blocks

The auto-tuning kernels can find the appropriate parameters for the blocking algorithm. The matrices are divided into blocks to be calculated. The parameters of our algorithm are limited by constraint 1 - 4, when m and n are multiple of m_b and n_b, k is the multiple of 8. These corners can be treated by the main process of the 4-level algorithm. Apart from that, there are 7 kinds of incomplete blocks for our algorithm to be treated. These 7 kinds of Incomplete Blocks (IBs) are shown in table 1.

We designed a general algorithm which is capable to cope with these incomplete blocks. Just as mentioned above, Godson-3B remains all features of Godson-3A. We use 128-bit memory access instruction to substitute the DRA in our 4-level algorithm. In this condition, the constraints of the blocking parameters are no longer valid. So we can calculate all sizes of the corners by the general algorithm.

Table 1. Stimuli Category Explanations

Label	M	N	K	Illustration
IB-1	=	=	<	
IB-2	=	<	<	
IB-3	<	=	<	
IB-4	<	=	=	
IB-5	=	<	=	
IB-6	<	<	=	
IB-7	<	<	<	

However, the general algorithm is not efficient. As the features of fetching have not been fully utilized, access instructions occupy a lot of overhead of the Execute Processor. Although we can use DCA to prefetch block A or block B to the cache-locked window, the overhead for accessing data form L2-cache will take a lot of time.

4.1 Improved Algorithm for Incomplete Blocks

In addition to the general algorithm, we also design several improved algorithms. Depending on the size of the corners, our algorithm utilizes DRA and SIMD instructions to speed up the accessing and calculating. We introduce the improved algorithm under different circumstances.

IB-1: Just as table 1 shows, only K of this condition is smaller than the blocking parameter K_b. That means we still can use the SIMD instructions to calculate every unit block for the incomplete blocks. IB-4 can be divided in to $IB-4_8$ and $IB-4_1$,

which is shown by Figure 5. K_8 Is a multiple of 8, while the K_1 is smaller than 8. We can use Inner Kernel to calculate the result for $B_{a1} \times B_{b1}$, and then use general algorithm to deal with $B_{a2} \times B_{b2}$.

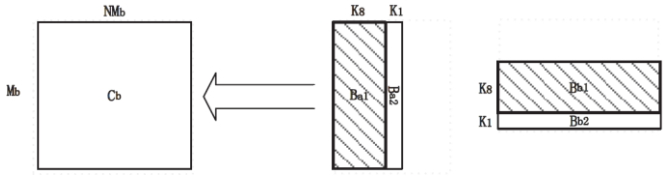


Figure 5. Improved Algorithm for IB-1

IB-4 & IB-5: We define IB-4 and IB-5 as block-panel (BP) operation, that means one block is small than another one, and the bigger block is a inner block of level-2. Taking IB-4 as an example, it also can be divided into 2 parts. Similar with IB-1, M can be divided as M_4 and M_1, the M_4 is multiple of 4. Under this condition, the block of A will be divided as B_{a4} and B_{a1}. We can use the Computational kernel to deal with $B_{a1} \times B_b$. However, when the size of B_{a1} is smaller than B_b, the accessing loads of read-channel A and B are not balanced. At the same time, if EP detects that DRA does not finish prefetching the data, it will be idle until the prefetching completes. Therefore, when M_4 is smaller than half of N_i, we use both read-channel A and B can be used to pre-fetch B_b, and 128-bit memory access will be utilized to fetch the data of B_{a1}. Under this circumstance, the accessing loads of read-channel A, B and execute processor will be much more balanced.

5. Experimental Results

We used a new hardware platform of Godson-3B for the experiment. The new platform fixed some bugs of the chip, and the ability of EP was weakened on the new platform. However, the new platform is more stable, and the clock frequency of DDR is increased from 300MHz to 400MHz. And the CPU frequency is 800MHz. We implement the computational kernel and Inner kernel by MIPS assembly language on the new platform. And the other 2 levels are implemented in C language. The GNU Compiler Collection for Gordosn-3 was used for the experiment, which is patched the tool-chain to support the new instructions of Godson-3B.

5.1 The Block Parameters of Auto-Tuning Kernels

With the help of DAEFS, we find the new block parameters of 4-level kernels for our algorithm. Due to the different constraints of single-core GEMM and parallel GEMM, the values of the block parameters are not exactly the same. And also, our experimental result shows that the block parameters of runtime system are different from that of simulation system. All the values of the blocking parameters are shown in table 2.

Table 2. Block Parameters of GEMMs

	M_b	K_b	N_b	M_i
Simulation	*240*	*480*	-	*12*
Single-Core	*48*	*256*	*192*	*12*
4-Cores	*24*	*384*	*144*	*12*

It is obvious that the values of the blocking parameters for 4-level kernels are smaller than these of the simulation algorithm, which means our 4-level kernels needs less cache size and registers number.

5.2 The Improvement for Debugging and Tuning

As we design the auto-tuning kernels to accelerate the optimization process of GEMM based on Godson-3B. Figure 6 shows the time for running GEMM process based on the DAEFS platform and simulation platform -- EVE. Once we run the GEMM process based on the simulation platform, we should launch a cropped operating system on the platform at first, which will take up about half an hour for simulation process. Besides, the performance of GEMM based on DAEFS is hundreds of times of that based on the simulation platform. The experimental results shows that using DAEFS platform can effectively improve the efficiency for debugging and tuning.

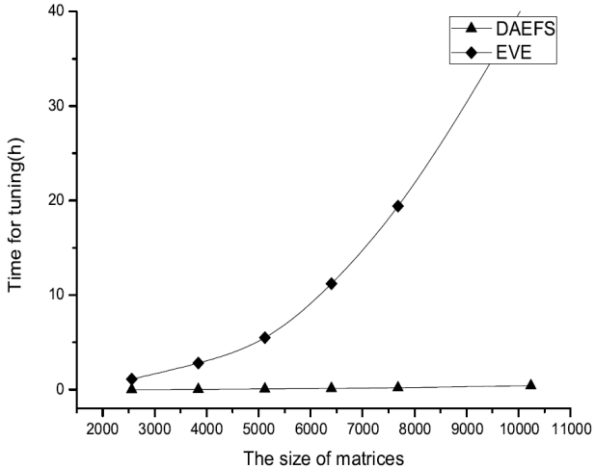


Figure 6. Time for Tuning of GEMM

5.3 The Performance of Optimized GEMM

We compared the performance of our single-core GEMM to UBLAS's. As shown in Fig 7(a), the best performance of Optimized GEMM is about 10.7 Gfplops. When the sizes of matrices are large enough, the performance of optimized GEMM is about 6% lower than that of GEMM kernels. As UBLAS optimized GEMM based on the characteristics of Godson-3A, the best performance of our GEMM is much higher than that of UBLAS's. The experimental result shows that when the size of M, N and K are close to 6000 the optimized GEMM will reach the best performance. And Figure 10 is the results of the HPL Testing based on the 4-cores Godson-3B platform. The memory size for testing is limited, and the biggest testing size is 10000. Due to the limitation of memory size, the result can not reflect the best performance of the 4-cores Godson-3B platform. Just as Figure 7(b) shows, the best speed of HPL invoke optimized GEMM is about 3 times of that invokes UBLAS's. The optimized GEMM effectively enhance the performance of HPL Testing.

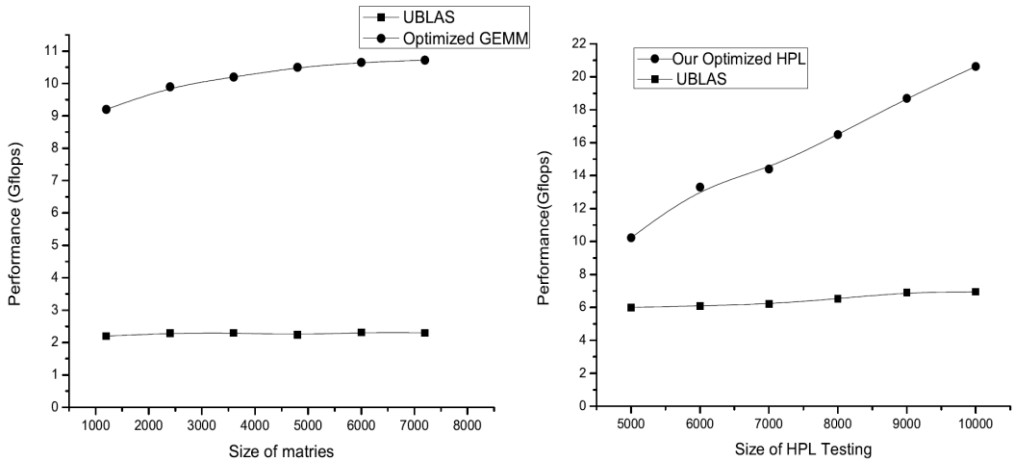


Figure 7(a) *Figure 7(b)*

Figure 7. The Performance Results of Algorithms and HPL Testing

6. Conclusions and Future Work

In conclusion, it has been shown that the new GEMM through the model-driven optimization algorithm designed by us, can find out the blocking parameters for 4-level kernels and needs less cache size and registers number than these of the simulation algorithm. Besides, we use the Godson-3B's hardware features to improve the performance for the incomplete blocks of GEMM. We use the features of Godson-3A and Godson-3B to balance the accessing loads of read-channel A, B and execute processor. The experiments show that, our algorithm combine with the features of DAE architecture. Using DAEFS platform can effectively improve the efficiency for debugging and tuning. Experimental results also show that the speed of HPL invokes optimized GEMM is 3 times higher than that invoke UBLAS's.

In the other hand, the results of performances of our GEMM kernels are lower than that of simulations by 6% to 8%. And also the peak of HPL Testing is lower than the best performance of the optimized GEMM. Performance impairments come from several sources. The overheads of interrupts and task scheduling of operating system synchronize and DAEFS all do not exist in simulation platform. In our future work, we plan to investigate the proportions of these overheads. Another significant part of future work is to design the optimization algorithms for other hot functions which are invoked by HPL Testing, for example, GEMV. WE will try to use the features of Godson-3B to get a higher performance on the HPL Testing.

Acknowledgment

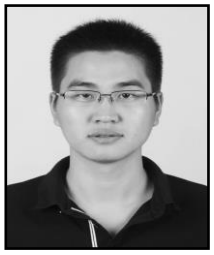
This paper is a revised and expanded version of a paper entitled"Auto-Tuning GEMM Kernels for a DAE Architecture Processor" presented at Candar2013, Japan, December 4-6. And it was supported in part by the National Science and Technology Major Project of China (No. 2009ZX01028-002-003-005).

References

[1] Godson homepage, http://www.loongson.cn
[2] W. Hu, R. Wang, Y. Chen, "Godson 3B A 1GHz 40W 8-core 128GFLOPS processor in 65nm CMOS", ISSCC'11, (**2011).**
[3] J. J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK Benchmark: Past, Present and Future," Concurrency Computation: Practice and Experience, vol. 15, no. 9, (**2003**), pp. 803-820.
[4] The Linpack Benchmark ,http://www.top500.org/project/linpack/

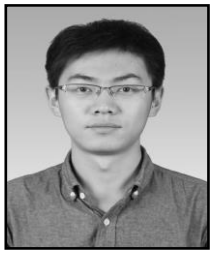
[5] W. Zhang, M. Chen and S. Feng, "Analysis and Optimization Discussion on Parallel Linpack [c]", Eighth Symposium Graduate of Institute of Computing Technology, Chinese Academy of Computer Science and Technology, (**2004**), p. 7.
[6] BLAS homepage, http://www.netlib.org/blas/
[7] Intel MKL homepage. http://software.intel.com/en-us/articles/intel-mkl/
[8] AMD ACML homepage. http://developer.amd.com/libraries/acml/pages/default.aspx
[9] NVIDIA CUBLAS homepage. http://cudazone.nvidia.cn/cublas/
[10] J. A. Gunnels, G. M. Henry and R. A. Vande Geijn, "A family of high-performance matrix multiplication algorithms [C]", Proceedings of the International Conference on Computational Science, (**2001**), pp. 51-60.
[11] K. Goto and R. A. Van de Geijn, "Anatomy of high-performance matrix multiplication", (**2008**), p. 3.
[12] H. Heidari and A. Chalechale, "Scheduling in Multiprocessor System Using Genetic Algorithm", International Journal of Advanced Science and Technology, vol. 43, (**2012**), pp. 81-94.
[13] S.-s. He, "Research on Key Issues of Program Optimization on Multi-core Loongson Architecture", [PhD thesis], University of Science and Technology of China, (**2012).**
[14] X.-y. Zhang, Q. Wang and Y.-q. Zhang, "OpenBLAS: A High Performance BLAS Library on Loongson 3A CPU", HPC China, (**2011).**
[15] H.-t. Zhu, Y.-j. Chen and C. Qian, "Optimization of matrix multiplication based on a multi-core architecture extended with vector units", Journal of University of Science and Technology of China, vol. 4, no. 2, (**2011**).
[16] ATLAS homepage. http://math-atlas.sourceforge.net/
[17] J. Kurzak, "Autotuning GEMM Kernels for the Fermi GPU", IEEE transactions on parallel and distributed systems, vol. 23, no. 11, (**2012)** November.
[18] J. E. Smith, "Decoupled access/execute computer architectures", Computer Systems, ACM Transactions on; vol. 2, no. 4, (**1984**) November, pp. 289-308.
[19] H.-t. Zhu, "High-density multi-core processor architecture research", [PhD thesis], University of Science and Technology of China, (**2011).**
[20] K. Goto and R. van de Geijn, "High-performance implementation of the level-3 BLAS", ACM Trans. Math. Softw., vol. 35, no. 1, (**2008**), pp. 1-14.

Authors



Zeng Zhao received the B.S. degree in computer science from the University of Science and Technology of China (USTC) in 2009. He is currently working toward the Ph.D. degree at the Department of Computer Science of USTC. His research interests include parallel algorithms, Cloud Computing and High Performance Computing.

Naijie GU received his B.S. in Mathematics from University of Science and Technology of China (USTC) in 1983, and M.E. in Computer Science from USTC in 1989. Now he is a Professor in the School of Computer Science and Technology at USTC. His research interests Distributed Systems, High Performance Computing and Information Security.\



Yangzhao Yang received the B.S. degree in computer science from the University of Science and Technology of China (USTC) in 2009. He is currently working toward the Ph.D. degree at the Department of Computer Science of USTC. His research interests include Compiler optimization and High Performance Computing.