Eric: A Special-Purpose Processor for ERI Calculations in Quantum Chemistry Applications

Nakamura, Kenta
Kyushu University

Hatae, Hidenori
Kyushu University

Harada, Muneyuki
Kyushu University

Kuwayama, Yoji
Kyushu University

他

Eric: A Special-Purpose Processor for ERI Calculations in Quantum Chemistry Applications

Kenta NAKAMURA[1], Hidenori HATAE[1], Muneyuki HARADA[1], Yoji KUWAYAMA[1],
Masamitsu UEHARA[2], Hisao SATO[2], Shigeru OBARA[3], Hiroaki HONDA[5],
Umpei NAGASHIMA[4], Yuichi INADOMI[5], Kazuaki MURAKAMI[1]
1 Kyushu University, 2 Seiko Epson Corp., 3 Hokkaido University of Education,
4 National Institute of Advanced Industrial Science and Technology,
5 Fuji Research Institute Corp.
ehpc-lsi@star.fuji-ric.co.jp

ABSTRACT

Ab initio molecular orbital (MO) calculation is useful for solving many challenging problems regarding the development of new drugs, chemicals, polymers, materials, and so on. In the EHPC (Embedded High Performance Computing) project, we are now developing a special-purpose computer system for *ab initio* MO calculations in order to reduce the calculation time.

The sequential execution time of *ab initio* MO is $O(N^4)$ where N is the number of basis functions, the heaviest computation being the electron repulsion integrals (ERI's). In order to accelerate *ab initio* MO calculations, it is necessary to develop a special-purpose processor for ERI calculation.

Using the characteristics of ERI in the Obara algorithm makes it possible to reduce the calculation time. In this work, we investigate a chip-multiprocessor (CMP) architecture, called Eric, for an application-specific processor able to perform fast ERI computations.

Keywords

Ab initio molecular orbital calculation, electron repulsion integral, application-specific processor, chip-multiprocessor architecture

INTRODUCTION

The *ab initio* molecular orbital (MO) method presents important information about electronic state of molecules. It is therefore indispensable to analyze experimentally observed phenomena[1] for designing function materials, for developing new drugs and so on. It can also present information about inter-atomic interaction in molecular assemblies. This method is one of the basis of all molecular simulation approaches. The *ab initio* MO method spends most (+90%) of its execution time in the electron repulsion integral (ERI) calculation. Consequently, in order to improve the performance of MO, we have to speed up the ERI calculation. Having for goal to achieve this speed-up, we are currently developing an MO-specific processor, called Eric.

Our ERI calculation algorithm is a new version of the Obara algorithm [3], whose current version is widely used today. By means of the Obara algorithm, we can express an ERI calculation as a recurrence formula. The recurrence calculation consists of many floating-point multiply-and-add operations and has a large amount of instruction-level parallelism (ILP). Therefore we can perform fast ERI calculation by processing several multiply-and-add operations in parallel. The Obara algorithm is roughly divided into two segments whose behaviors are quite different from each other. One is the initial integral calculation and the other is the recurrence calculation. The recurrence calculation is a series of floating-point multiply-and-add operations and has a rich ILP; on the other hand, the initial integral calculation contains some complex floating-point operations such as division, inverse square root, exponential function and error function, and it lacks in ILP. The MO-specific processor Eric is optimized to the characteristics of the Obara algorithm, and can efficiently process both parts. Hence we have employed a chip-multiprocessor (CMP) architecture which consists of two engines: the initial integral calculation engine and the recurrence calculation engine. We also have investigated the architecture of each engine.

The rest of the paper is organized as follows: Section "ERI Calculation" explains the *ab initio* molecular orbital calculations and the characteristics of the Obara algorithm. Section "Eric: ERI-Calculation Processor" describes an outline of the Eric processor architecture which consists of two engines. Section "Discussion on RC Engine Architecture" details the recurrence calculation engine and estimates different organizations. Finally Section "Conclusions" concludes the paper.

ERI CALCULATION

This section outlines the *ab initio* MO calculations and describes the characteristics of the Obara algorithm for solving ERI calculation.

[1] Experiments are performed at the atomic level and based on quantum chemistry.

Table 1: Computation Time for Some Peptide Molecules

Types of Peptide Molecules	G	GA	GAQ	GAQM	GAQMY
No. of Atoms	10	20	37	58	75
No. of Shells	25	50	93	145	190
Basic Size <4-31G Basis>	55	110	207	316	427
Computation Time *<seconds>*					
Setup	0.1	0.1	0.1	0.1	0.3
Initial Orbital Set	0.1	0.6	4.4	18.9	57.3
ONE-ERI Calculation	0.1	0.3	1.5	5.0	10.1
TWO-ERI Calculation & Fock-Matrix Generation	22.9 <96.6%>	269.4 <98.7%>	1871.0 <98.6%>	8482.1 <98.8%>	23284.3 <98.6%>
Fock-Matrix Diagonalization	0.2	1.7	11.0	60.9	211.7
Property Calculation	0.1	0.3	2.2	9.15	27.5
Total CPU Time	23.7	272.9	1892.7	8584.9	23614.5
No. of Iterations	12	14	15	16	19

Note: 4-31G basis function set, using GAMESS, on Pentium-III with 512MB Main Memory

Overview of MO

We use the Hartree-Fock (HF) method for solving *ab initio* MO calculations. The HF method is the most standard approximation technique. In the HF method, the most time-consuming step is the Fock matrix generation including ERI calculations. The computational load of the ERI calculation is $O(N^4)$, where N is the number of basis functions. We measured the computation time for the HF method on a PC using GAMESS, a world-widely distributed free software for MO calculation. The PC consists of a Pentium III CPU (500MHz), with 512M bytes main memory. Table 1 shows the *ab initio* MO computation time (sec) for five peptide molecules G, GA, GAQ, GAQM and GAQMY, and summarizes the time distribution for each step in the HF method, where the basis function set is 4-31G. The results show that the TWO-ERI calculation and Fock-Matrix generation consumes more than 95% of the total computation time. It is important to reduce the computation time of ERI's, in order to accelerate the *ab initio* MO calculation.

The Obara Algorithm for ERI Calculation

In several algorithms for the ERI calculation, we adopt the Obara method [1,3]. The formulation in recently developed Obara method [1] is based on the recurrence formula over the contracted Cartesian Gaussian functions, it can be utilized in developing recurrence formulation of ERIs and one-electron integrals. Each Gaussian function is defined by informations of atoms. The angular momenta of each ERI can be specified by the 4 sets of orbital quantum numbers in which each set is written in three-dimensional vector form, we let to rewrite ERIs to $(\boldsymbol{abcd})$, where $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, $\boldsymbol{d}$ are orbital quantum vectors.

First, an initial integral (ss,ss) is computed from input data. And ERI $(\boldsymbol{abcd})$ computation is shown in the following recurrence formula.

$$(\boldsymbol{abcd}) = < \boldsymbol{abcd} \mid m : g_i, g_j, G_{ij}, g_k, g_l, G_{kl} >$$

$$< (\boldsymbol{a}+\boldsymbol{1_u})\boldsymbol{bcd} \mid m : g_i, g_j, G_{ij}, g_k, g_l, G_{kl} >$$

$$= (\boldsymbol{B}-\boldsymbol{A})_u < \boldsymbol{abcd} \mid m : g_i, (g_j+1), (G_{ij}+1), g_k, g_l, G_{kl} >$$

$$+(\boldsymbol{A}-\boldsymbol{B})_u < \boldsymbol{abcd} \mid m+1 : g_i, (g_j+1), (G_{ij}+2), g_k, g_l, G_{kl} >$$

$$+(\boldsymbol{D}-\boldsymbol{C})_u < \boldsymbol{abcd} \mid m+1 : g_i, g_j, (G_{ij}+1), g_k, (g_l+1), (G_{kl}+1) >$$

$$+(\boldsymbol{C}-\boldsymbol{A})_u < \boldsymbol{abcd} \mid m+1 : g_i, g_j, (G_{ij}+1), g_k, g_l, G_{kl} >$$

$$+\frac{(\boldsymbol{a})_u}{2} < (\boldsymbol{a}-\boldsymbol{1_u})\boldsymbol{bcd} \mid m : g_i, g_j, (G_{ij}+1), g_k, g_l, G_{kl} >$$

$$-\frac{(\boldsymbol{a})_u}{2} < (\boldsymbol{a}-\boldsymbol{1_u})\boldsymbol{bcd} \mid m+1 : g_i, g_j, (G_{ij}+2), g_k, g_l, G_{kl} >$$

$$+\frac{(\boldsymbol{c})_u}{2} < \boldsymbol{ab}(\boldsymbol{c}-\boldsymbol{1_u})\boldsymbol{d} \mid m+1 : g_i, g_j, (G_{ij}+1), g_k, g_l, (G_{kl}+1) >$$

$$-\frac{(\boldsymbol{b})_u}{2} < \boldsymbol{a}(\boldsymbol{b}-\boldsymbol{1_u})\boldsymbol{cd} \mid m : g_i, g_j, (G_{ij}+1), g_k, g_l, G_{kl} >$$

$$-\frac{(\boldsymbol{b})_u}{2} < \boldsymbol{a}(\boldsymbol{b}-\boldsymbol{1_u})\boldsymbol{cd} \mid m+1 : g_i, g_j, (G_{ij}+2), g_k, g_l, G_{kl} >$$

$$-\frac{(\boldsymbol{d})_u}{2} < \boldsymbol{abc}(\boldsymbol{d}-\boldsymbol{1_u}) \mid m+1 : g_i, g_j, (G_{ij}+1), g_k, g_l, (G_{kl}+1) >$$

Where i denotes the value of the three-dimensional orbital quantum vector, $\boldsymbol{a_i}$, $\boldsymbol{b_i}$, $\boldsymbol{c_i}$, $\boldsymbol{d_i}$ denote i

components of $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, $\boldsymbol{d}$. 1_i means the vector that if its component is i then it equals 1, else equals 0. And m is the sum of the orbital quantum number. The sum of ingredients of the orbital quantum vector is called the orbital quantum number, its values: 0, 1 and 2 are called the orbital s, p and d. Thus, ERIs can be symbolized (ss,ss), (ps,ss), (dd,dd) etc.

The calculation of initial integrals $< ss,ss \mid m : g_i, g_j, G_{ij}, g_k, g_l, G_{kl} >$ is one of the most time-consuming steps in the ERIs calculations. Initial integral of the ERI calculation reduces to

$$< ss,ss \mid m : g_i, g_j, G_{ij}, g_k, g_l, G_{kl} >=$$

$$\sum_{i=0}^{N_i}\sum_{j=0}^{N_j}\sum_{k=0}^{N_k}\sum_{l=0}^{N_l}(\zeta_i)^{g_i}(\zeta_j)^{g_j}\left(\frac{1}{\zeta_i+\zeta_j}\right)^{G_{ij}}(\zeta_k)^{g_k}(\zeta_l)^{g_l}\left(\frac{1}{\zeta_k+\zeta_l}\right)^{G_{kl}}\rho^m V_{ijklm},$$

where

$$\rho=\frac{(\zeta_i+\zeta_j)(\zeta_k+\zeta_l)}{\zeta_i+\zeta_j+\zeta_k+\zeta_l}$$

and

$$V_{ijklm}=\frac{2\pi^{\frac{5}{2}}u_i u_j u_k u_l F_m(T)}{\sqrt{\zeta_i+\zeta_j+\zeta_k+\zeta_l}}K(\zeta_i,\zeta_j,\boldsymbol{A},\boldsymbol{B})K(\zeta_k,\zeta_l,\boldsymbol{C},\boldsymbol{D}).$$

$$K(\zeta_i,\zeta_j,\boldsymbol{A},\boldsymbol{B})=\frac{1}{\zeta_i+\zeta_j}\exp\left[-\frac{\zeta_i\zeta_j}{\zeta_i+\zeta_j}(\boldsymbol{A}-\boldsymbol{B})^2\right],$$

where $F_m(T)$ is shown as follows, which is called the error function.

$$F_m(T)=\int_0^1 t^{2m}\exp(-Tt^2)dt$$

and

$$T=\rho\left\{\frac{\zeta_j(\boldsymbol{B}-\boldsymbol{A})}{\zeta_i+\zeta_j}+\frac{\zeta_l(\boldsymbol{C}-\boldsymbol{D})}{\zeta_k+\zeta_l}+(\boldsymbol{A}-\boldsymbol{C})\right\}^2.$$

The error functions $F_m(T)$ are evaluated using two formulas depending on the value T. For values from zero to 90, we employ the four-term Taylor expansion:

$$F_m(T)=$$

$$F_m(T_0)+(-\varepsilon)\left\{F_{m+1}(T_0)\frac{(-\varepsilon)}{2}\left\{F_{m+2}(T_0)+\frac{(-\varepsilon)}{3}F_{m+3}(T_0)\right\}\right\},$$

where

$$\varepsilon=T_0-T,$$

where $F_m(T_0)$ has been evaluated for T_0 at intervals and tabulated. Another formula, used for T greater than 90, is an asymptotic formula of $F_m(T)$, where the upper limit of the integration range is replaced by positive infinity.

$$F_m(T)\approx\int_0^\infty t^{2m}\exp(-Tt^2)dt=\frac{(2m-1)!!}{2(2T)^m}\sqrt{\frac{\pi}{T}}.$$

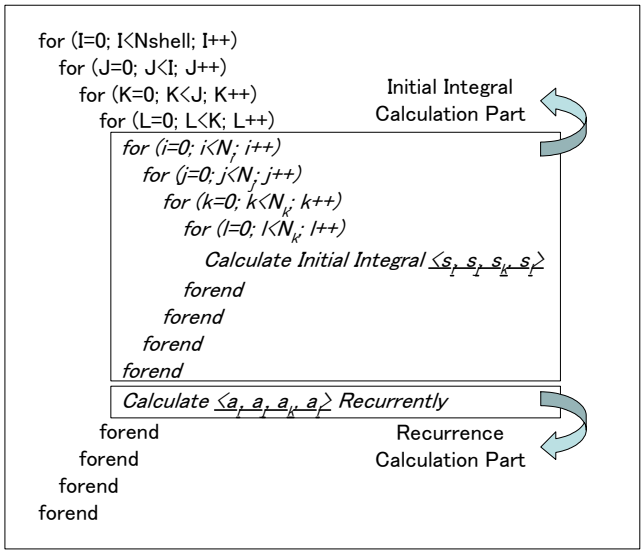

Figure 1: Loop Structure of the Obara Algorithm

Characteristics of the Obara Algorithm

The most characteristic point of the Obara algorithm is its process. We first calculate an initial integral (ss,ss) with input data. We then use this result to perform the recurrence calculation for a certain ERI (ab,cd). Therefore, the Obara algorithm can be roughly divided into two segments: initial integral calculation and recurrence calculation as shown in Figure 1.

Initial Integral Calculation

The initial integral calculation has a four-fold loop structure, as we can see in Figure 1. In the initial integral calculation, there is a small number of operations per iteration, each with a small amount of instruction-level parallelism (ILP). To make matters worse, the initial integral calculation contains some complex floating-point operations such as division, inverse square root, exponential function, and error function.

Recurrence Calculation

Recurrence calculation computes a certain ERI (ab,cd) after the processing of initial calculation is completed. The structure of the recurrence calculation part consists of two functions, called "recurrence function" and "horizontal function", respectively, as shown in Figure 2. Moreover, each function has a hierarchical structure which calls the ERI_reccal subfunctions and ERI_horcal subfunctions, respectively. A subfunction consists of a large number of floating-point multiply-and-add operations. Since there are lots of parallelism between these multiply-and-add operations, we can process many operations simultaneously, namely we can process a subfunction in parallel. And there are a lots of parallelism between subfunctions as well as between multiply-and-add operations, therefore we can process several subfunction in parallel too.

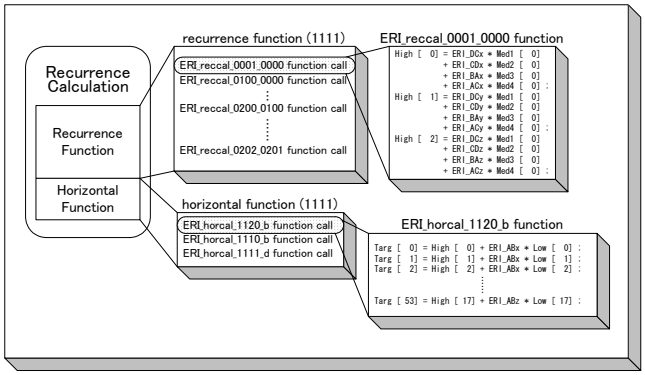

Figure 2: Hierarchical Structure of a Recurrence Calculation

ERIC: ERI-CALCULATION PROCESSOR

The Obara algorithm consists of two parts: the initial integral calculation and the recurrence calculation. Their characteristics differ greatly, and the MO-specific processor Eric must process both at high speed. This section describes the outline of the Eric processor architecture for fast ERI calculation.

Design Goals

In the previous chapter, we saw that the Obara algorithm have an extensive ILP. To exploit this ILP, the Eric processor has many floating-point multiply-and-accumulate units and performs ERI calculation in parallel. However, the design of a processor which has a lot of functional units brings the following problems:

- Register file which supplies functional units with operands needs to have a great number of ports. For the N arithmetic units, the area of the register file grows as N^3, and the delay as $N^{3/2}$ [4]. Hence the area and the delay of multi-port register file restrict the area and the performance of the processor.
- Since the instruction code becomes very long, it is difficult to supply arithmetic units with it.

In addition, there are also the following two problems:

- The initial integral calculation part has less ILP, and consequently, arithmetic units are not fully used during its processing.
- We suppose that the processor has internal program memory in order to ease the supply of instruction code, but there is a limit in the memory size which can be placed on a chip. Therefore, we have to examine techniques for reducing the program size.

We have to take the above-mentioned points into consideration when we develop the Eric processor architecture.

Design Philosophy

The Obara algorithm consists of two calculation parts which characteristic is quite different. The design philosophy for coping with the two calculation parts, IIC part and RC part is as follows:

- An IIC part including some complex floating-point operations accelerates processing by installing special functional units to a processor.
- An RC part having rich ILP accelerate by installing many functional units to a processor and parallel processing.

However, realizing the both functions by one processor is not efficient. Because, it is the reason that IIC part does not need many functional units since parallelism is low, and RC part dose not need special functional units since there is no complex floating-point operation. Therefore we decided to divide Eric processor into two engines: IIC engine and RC engine.

Overview of Eric Processor

As shown in Figure 3, the Eric processor architecture consists of the following five modules.

- Host Interface for communication with Host CPU,
- External Memory Interface for accessing External Memory,
- Internal Data Memory for storing the calculation results,
- IIC Engine for the initial integral calculation, and
- RC Engine for the recurrence calculation.

The data used in this processor is IEEE 754 double-precision floating-point numbers. Therefore, all the functional unit, memory and interfaces are 64-bit width, and so, single precision floating-point numbers are not supported.

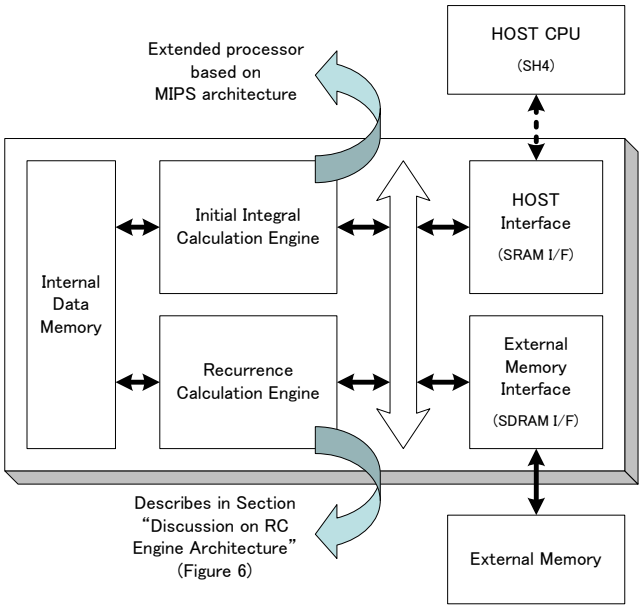

Figure 3: Overview of Eric Processor Architecture

Table 2: Design Alternatives for RC Engine

	Macro routine	Micro routine	Parallelism
FS	Subfunction call	Subfunction execution	Instruction-level
FP	Subfunction call	Subfunction execution	Inter-functions & Instruction-level
SP	Statement call	Statement execution	Inter-statements & Instruction-level

IIC Engine

The initial integral calculation consists of various operations such as arithmetic, floating-point addition, multiplication, division, inverse square root, exponential function, function call and conditional branch. Therefore, it is better to design the initial integral calculation (IIC) engine with the same organization as a general purpose processor. In this processor, the IIC engine is an extension to the MIPS architecture.

RC Engine

Recurrence calculation (RC) engine is a processor which is specialized to the RC part of the Obara algorithm. RC consists of only great many floating-point multiply-and-adds operations. Therefore, if many multiply-and-accumulate units are installed in an RC engine, it can be expected that a program can be executed at high speed. However, as mentioned above, there is a problem in installing many MAC units in RC engine. Then we consider the organization of RC engine with the following plans.

- In RC engine, plural clusters which consist of register file (RF), load-store unit (LSU), and multiply-and-accumulate (MAC) unit are created. We call it subengine. By dividing RC engine into plural subengine, the number of the functional units connected to a register file can be reduced. Thereby, some problems which multi-port register file causes are avoidable with it.
- RC engine has internal program memory, in order to ease supplying instruction code.
- In order to make the characteristics of the Obara algorithm optimize, RC engine adopts hierarchical instruction code set which consist of two: macro routine and micro routine. This is also for reducing program size.

Then we consider the composition of detailed RC engine after this.

DISCUSSION ON RC ENGINE ARCHITECTURE

In the previous chapter, we showed that the Recurrence Calculation in ERI has a hierarchical structure and extensive instruction-level parallelism. In this section, we examine the RC Engine architecture to be optimized with regards to those characteristics.

Design Alternatives for RC Engine

In the design of the RC engine, in order to reduce a size of program on chip, we adopt a program execution method which consists of two levels of routines: a macro routine and a micro routine. As shown in Table 2, we evaluated three kinds of processor models: Function Serial (FS), Function Parallel (FP) and Statement Parallel (SP). These three models are classified by the difference in the program portion assigned to a macro routine and a micro routine, and the difference in the amount of parallelism.

Function Serial (FS)

The FS is the processor model which assumed VLIW architecture as the RC engine, and it is the base model of FP and SP. In the FS model, subfunction calls are assigned to a macro routine which is placed in external memory. On the other hand subfunction execution is assigned to a micro routine which is placed in the internal program memory. When processing the recurrence calculation, the macro routine is sequentially read from external memory and the corresponding micro routine is processed using all functional units. In this model, we use the ILP only for calculation of the recurrence calculation.

Function Parallel (FP)

The FP is the processor model which based on the FS. In this model, assignment of the recurrence calculation to a macro routine and micro routine is the same, however there is a difference in the parallelism we can use. As described in Subsection "Characteristics of The Obara Algorithm", since there is parallelism between subfunctions in recurrence calculation, we can process many subfunctions in parallel. In this model, two or more sub engines which consist of a set of some functional units is prepared into the RC engine. When processing the recurrence calculation, several macro routines are read from the external memory in parallel, and the corresponding micro routines are processed in parallel too.

Statement Parallel (SP)

The SP model resembles FP with respect to the hardware organization. However, the parallelism to achieve is different as there is parallelism between statements as well as between subfunctions. Here, the statement is defined

as a series of floating-point multiply-and-add operations. Therefore, many statements can be simultaneously processed in parallel. When processing the recurrence calculation, this model reads several macro routines consisting of statement calls from external memory, and executes the body of statement in parallel with the two or more subengines as micro routines.

Performance Evaluation

We evaluated the performance of the above-mentioned three processor models.

Evaluation Methodology

In order to decide which processor model (FS, FP or SP) to adopt, the number of clock cycles which are reported by a code scheduler was used as a criterion. For this estimation, we built a code scheduler for each model of RC engine. The code scheduler generates the micro routine after having scheduled the input program (written in C). Then, from the generated micro routine, we estimated schedule length. As a static scheduling algorithm, we adopted the critical path method, which is a one of list scheduling algorithms, based on the "critical path, most immediate successors first" priority [2].

We made evaluations for 12 models shown in Table 3. The result latency and the issue latency of the functional units common to each model are shown in Table 4. In this paper, the result latency means the number of intervening cycles between an instruction that produces a result and an instruction that uses the result. The issue latency means the number of cycles that must elapse between issuing two same instructions.

In this model, the size of register file is assumed infinite. And both the total number of MAC units and LSUs is not more than six as resources restrictions.

Table 3: Evaluation Models

Model	No. of Subengine	No. of MAC per Subengine	No. of LSU per Subengine	Total Number	
				MAC	LSU
For evaluation focusing on the difference in parallelism					
$FS_{1,6,6}$	1	6	6		
$FP_{2,3,3}$	2	3	3		
$FP_{3,2,2}$	3	2	2		
$FP_{6,1,1}$	6	1	1	6	6
$SP_{2,3,3}$	2	3	3		
$SP_{3,2,2}$	3	2	2		
$SP_{6,1,1}$	6	1	1		
For evaluation focusing on the difference in organization					
$FP_{2,1,1}$	2	1	1	2	2
$FP_{2,2,1}$	2	2	1	4	2
$FP_{2,3,1}$	2	3	1	6	2
$FP_{3,1,1}$	3	1	1	3	3
$FP_{4,1,1}$	4	1	1	4	4

Table 4: Latency of Each Functional Unit

	MAC	LSU
Result Latency <*clock cycles*>	6	5
Issue Latency <*clock cycles*>	1	1

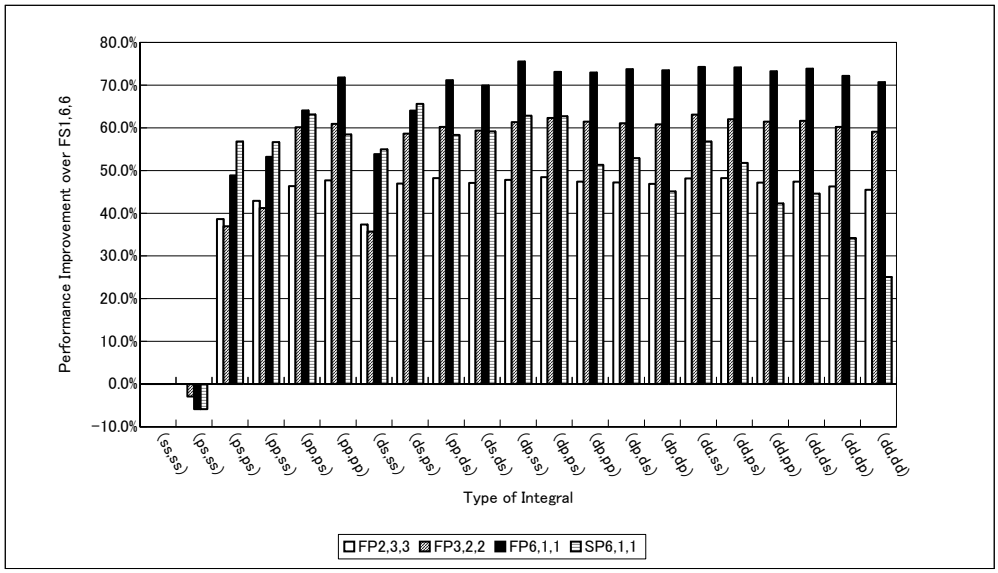

Figure 4: Performance Comparison among FS, FP, and SP

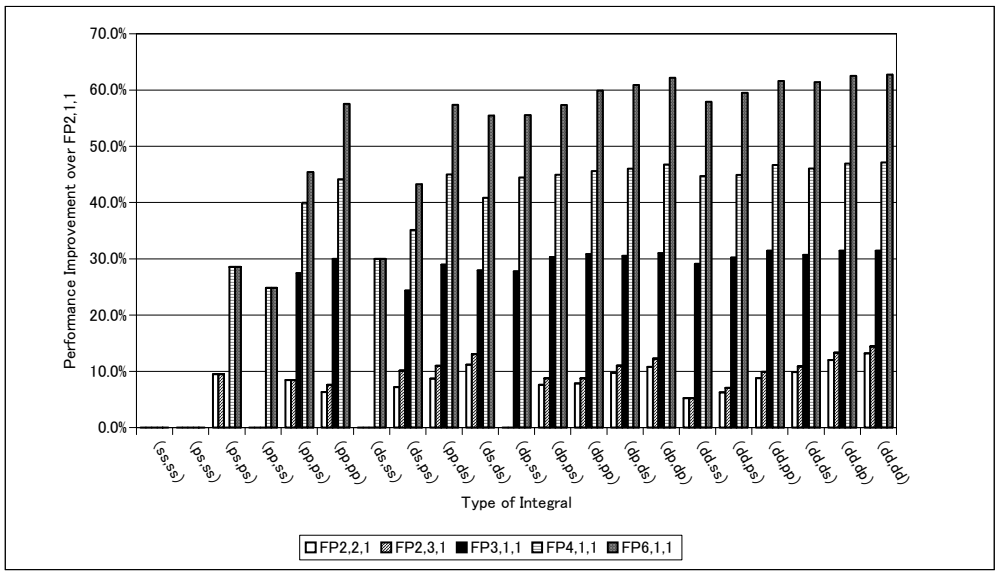

Figure 5: Performance Comparison among Different Organization of FP

Evaluation Results

We tested the three models (FS, FP and SP) and evaluated which technique is the optimal. In this experimentation, the number of LSU and MAC was six in each model. We evaluated about all possible organizations. Among these, only the results whose features stood out are shown in Figure 4. That is, since the performance of $SP_{2,3,3}$ and $SP_{3,2,2}$ were lower than that of $SP_{6,1,1}$, only the result of $SP_{6,1,1}$ is shown in the graph. The graph is normalized to $FS_{1,6,6}$, so we can see the speedup ratio of each model compared to $FS_{1,6,6}$ from the graph.

Meanwhile we limited to model FP and evaluated about the difference in the organization about subengines. When $FP_{2,1,1}$ was a base model, we evaluated the difference of performance about each case: added FU to the base model ($FP_{2,2,1}$ or $FP_{2,3,1}$) and increased the number of subengines ($FP_{3,1,1}$ or $FP_{4,1,1}$). Figure 5 showed a graph of the result. The graph is normalized to $FP_{2,1,1}$, so we can see the speedup ratio of each model compared to $FP_{2,1,1}$ from the graph. Moreover, the graph of $FP_{6,1,1}$ is appended to Figure 5 for reference of comparison with Figure 4.

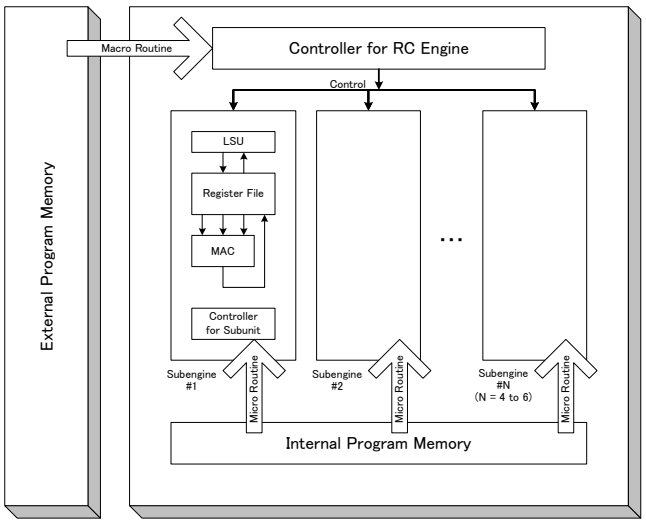

Figure 6: Organization of RC Engine

Discussion

As the Figure 4 which shows $FP_{6,1,1}$ is the fastest in almost all cases.

Moreover, as the Figure 5 which shows the evaluation result about the difference in the organization shows that, model $FP_{3,1,1}$ and $FP_{4,1,1}$ which the number of subengines made to increase show the high improvement in a performance in comparison with model $FP_{2,2,1}$ and $FP_{2,3,1}$ which added MAC units to base model $FP_{2,1,1}$.

That is, the following considerations are drawn from these results:

- As a result of comparing Models FS, FP, and SP, a model FP with many subengines is the fastest.
- From the evaluation result about the organization of FP, increasing the number of a subengine leads to improvement in a performance, in comparison with the case which increased the number of MAC per one subengine.

RC Engine Design

From above stated investigation, we determined RC engine's specification as follows:

- RC engine has four to six subengines which consist of a register file, a load-store unit and a multiply-and-accumulate unit.
- We adopt a program execution method consisting of two routines: a macro routine and a micro routine.
- Subfunction call and subfunction execution are assigned to a macro routine and a micro routine, respectively.
- RC engine has the internal memory which supplies a micro routine to subengines.

The final schematic of RC engine design is shown in Figure 6.

CONCLUSIONS

In this paper, we discussed the architecture of an ERI calculation specific processor architecture Eric. Our ERI calculation algorithm is a new version of the Obara algorithm which consists of two parts: initial integral calculation and recurrence calculation. Although their characteristics differ greatly, Eric processor needs to process both at high speed. Then, we divided Eric processor into two engines: IIC engine and RC engine. The IIC engine is an extension to the MIPS architecture and has special functional units for complex floating-point operation included in initial integral calculation. The RC engine has four to six subengines which consist of a register file, a load-store unit and a multiply-and-accumulate unit. And the RC engine adopts hierarchical instruction code set which consist of two routines: macro routine and micro routine.

Now we are creating specifications of Eric processor and performing logic design. The exact performance of Eric processor will be estimated in winter 2002.

ACKNOWLEDGMENTS

The research is granted by Japanese Ministry of Education, Culture, Sports, Science and Technology. We thank Dr. Lovic Gauthier and Ms. Natasha Devroye for their proofreading.

REFERENCES

1. H. Honda, T. Yamaki, and S. Obara. Molecular integrals evaluated over contracted Gaussian functions by using auxiliary contracted hyper-Gaussian functions. *Journal of Chemical Physics*, Vol. 117, No. 4, 1457-1469, 2002.
2. R. Allen and K. Kennedy. Optimizing Compilers for Modern Architectures: A Dependence-based Approach. Morgan Kaufmann Publishers, Oct. 2001.
3. S. Obara and A. Saika. Efficient recursive computation of molecular integrals over Cartesian Gaussian functions. *Journal of Chemistry Physics*, Vol. 84, No. 7, 3963-3974, Apr. 1986.
4. S. Rixner, W. Dally, B. Khailany, P. Mattson, U. Kapasi and J. Owens. Register Organization for Media Processing. *Proceedings of the 6th High-Performance Computer Architecture*, 375-386, Jan. 2000.